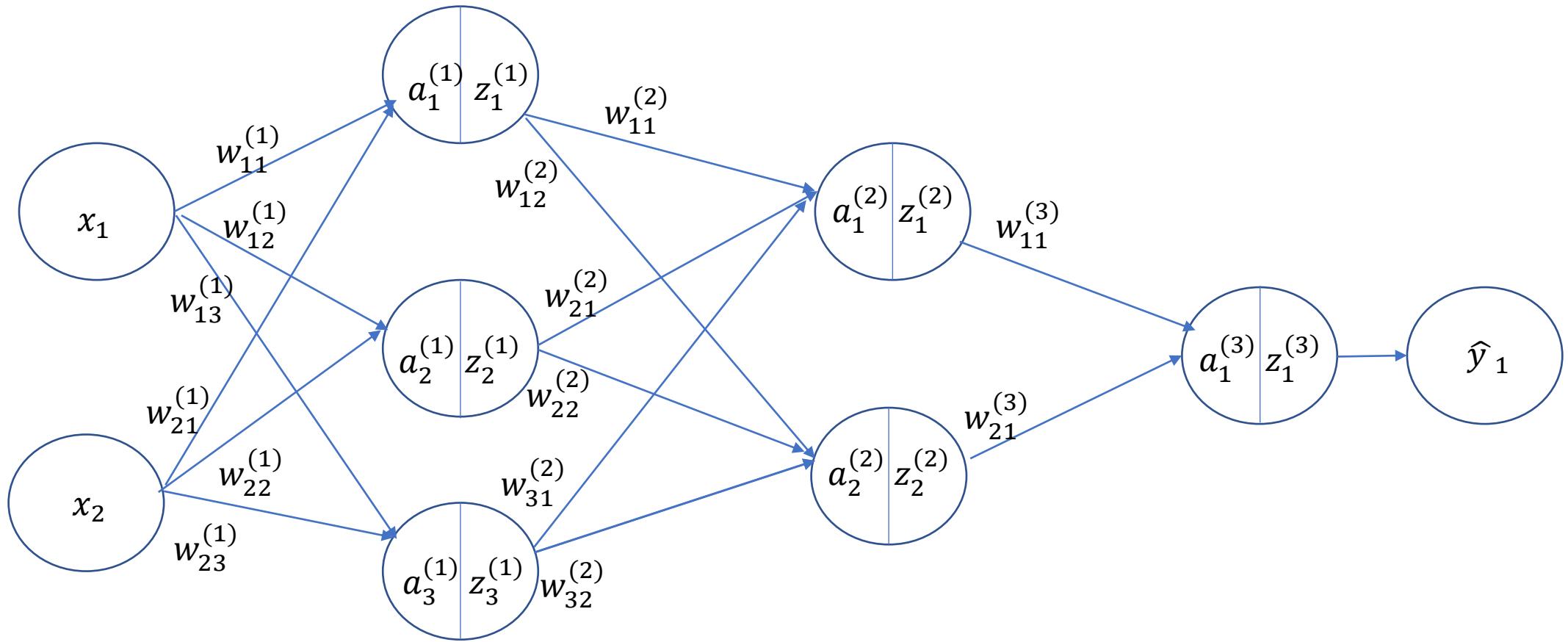


Backpropagation



$$d = 2, \quad L = 2 \quad s = 1$$

$$z_i^{(0)} = x_i, \quad i = 1, \dots, d$$

$$z_i^{(L+1)} = \hat{y}_i, \quad i = 1, \dots, s$$

$$a_i^{(\ell)} = \sum_{j=0}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)}$$

f : activation function: funzione **derivabile** non lineare.

Sia $\hat{y} = [\hat{y}_1, \dots, \hat{y}_s]$ l'output prodotto dalla rete in corrispondenza all'esempio di training $x = [x_1, x_2, \dots, x_d]$, e sia $y = [y_1, y_2, \dots, y_s]$ il vettore delle etichette corrispondente dell'esempio di training.

Scegliamo come loss function la somma dei quadrati degli errori tra l' output della rete e la corrispondente etichetta:

$$L(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^s (y_i - \hat{y}_i)^2 = \frac{1}{2} \|y - \hat{y}\|_2^2$$

La dipendenza dai pesi w è implicita in \hat{y} .

L'errore $C(w)$ sull'intero training set è la media di $L(w, x)$ su tutti gli esempi appartenenti n_T al training set.

$$C(w) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)})$$

dove $\hat{y}^{(j)} = [\hat{y}_1^{(j)}, \dots, \hat{y}_s^{(j)}]$ l'output prodotto dalla rete in corrispondenza al j -esimo esempio di training $x^{(j)} = [x_1^{(j)}, x_2^{(j)}, \dots, x_d^{(j)}]$, e sia $y^{(j)} = [y_1^{(j)}, y_2^{(j)}, \dots, y_s^{(j)}]$ il vettore delle etichette corrispondente dell'esempio di training.

Il problema dell'addestramento

La costruzione di una rete multistrato con d ingressi e s uscite consiste nello **scegliere l'architettura** (numero di strati e numero di neuroni di ogni strato), e **nell'addestrare la rete**, ossia nel **determinare il vettore $w \in \mathbb{R}^m$ le cui componenti corrispondono ai parametri incogniti (pesi e soglie dei neuroni nei vari strati).**

La scelta dei parametri, in addestramento di tipo **supervisionato**, per una architettura fissata, viene in genere effettuata definendo un opportuno sottoinsieme dei dati disponibili etichettati

$$T = \{(x^{(j)}, y^{(j)}), x^{(j)} \in \mathbb{R}^d, y^{(j)} \in \mathbb{R}^s, j = 1, \dots, n_T\},$$

che costituisce il **training set** e risolvendo successivamente un problema di ottimizzazione del tipo:

$$\arg \min_w C(w) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)})$$

in cui $L(y^{(j)}, \hat{y}^{(j)})$ è il termine di errore relativo al j -mo campione di training e misura la distanza tra l'uscita desiderata $y^{(j)}$ e l'uscita $\hat{y}^{(j)}$ fornita dalla rete. La misura più usata è l'errore quadratico ma è possibile considerare anche funzioni di errore di struttura diversa. Nel seguito ci limiteremo a supporre che E sia una funzione continuamente differenziabile.

Scopo dell'addestramento non è quello di interpolare i dati di training, quanto piuttosto quello di modellare il processo che ha generato i dati.

Ciò implica che:

- la scelta dell'architettura,
- la selezione dei dati da includere in T ,
- la definizione di L
- e la strategia di addestramento

devono tener conto dell'esigenza di assicurare buone capacità di generalizzazione.

Dal punto di vista teorico, uno dei problemi più importanti è quello di **definire opportunamente la complessità del modello, e quindi il numero di parametri liberi, in relazione ai dati disponibili**. Per le reti multistrato, a partire dai risultati sulla teoria statistica dell'apprendimento sono state stabilite delle stime del numero minimo dei campioni di training occorrenti per addestrare una rete in modo tale che si abbia una "buona" capacità di generalizzazione.

In pratica, tuttavia le stime teoriche possono essere inadeguate ed **occorre basarsi su opportune euristiche per la scelta della struttura e la definizione del training set**. In linea di massima, nel caso delle reti multistrato, vengono seguite due strategie fondamentali.

La **prima**, chiamata ***stabilizzazione strutturale*** consiste nello scegliere il numero di unità , attraverso **l'addestramento di una sequenza di reti in cui viene fatto crescere (o diminuire) il numero di neuroni**.

- Per ciascuna di tali reti i parametri vengono determinati minimizzando l'errore sul training set e le prestazioni delle varie reti sono confrontate attraverso una tecnica di *cross-validation*, valutando l'errore che ogni rete commette su un altro insieme di dati (*validation set*) non inclusi in T . **La rete selezionata è quella che fornisce l'errore minimo sul validation set.**

Le prestazioni di una rete addestrata vengono valutate utilizzando un terzo insieme di dati denominato *test set*, che non deve essere stato utilizzato né per la scelta del l'architettura, né per la determinazione dei parametri.

La **seconda strategia** si basa su una **tecnica di regolarizzazione** e consiste nell'aggiungere alla funzione di errore un termine di penalità sulla norma di w che ha l'effetto di restringere l'insieme entro cui vengono scelti i parametri. Ciò equivale, essenzialmente, ad imporre condizioni di regolarità sulla classe di funzioni realizzata dalla rete. L'addestramento della rete viene effettuato definendo la nuova funzione obiettivo

$$\arg \min_w C(w) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)}) + \gamma \|w\|_2^2$$

con $\gamma > 0$.

Il valore "ottimale" di γ può essere determinato utilizzando, anche in questo caso, una tecnica di cross-validation. In particolare, in corrispondenza a differenti valori di γ , si addestrano varie reti minimizzando la funzione d'errore C e viene successivamente prescelto il valore di γ a cui corrisponde il minimo errore sul validation set.

In alternativa alla tecnica di regolarizzazione, una strategia euristica talvolta utilizzata è quella **di interrompere prematuramente la minimizzazione (early stopping) della funzione d'errore**.

Questa tecnica si basa sull'idea di valutare periodicamente, durante il processo di minimizzazione, l'errore che la rete commette su un validation set ausiliario.

In generale, nelle prime iterazioni l'errore sul validation set diminuisce con la funzione obiettivo, mentre può aumentare se l'errore sul training set diviene “sufficientemente piccolo”.

Il processo di addestramento termina quindi quando l'errore sul validation set inizia ad aumentare, perché ciò potrebbe evidenziare l'inizio della fase di overfitting della rete, cioè della fase in cui la rete tende a interpolare i dati di training a scapito della capacità di generalizzazione.

Quale che sia la strategia di addestramento seguita, pur non essendo possibile ridurre banalmente i problemi di addestramento alla soluzione di un problema di ottimizzazione, è necessario, in pratica, **ripetere la minimizzazione in corrispondenza a varie architetture o a varie funzioni di errore**. La disponibilità di **algoritmi efficienti di ottimizzazione costituisce, quindi, uno strumento essenziale per la costruzione delle reti neurali**. La minimizzazione dell'errore di training C è, in generale, un difficile problema di ottimizzazione non lineare, in cui le difficoltà computazionali sono tipicamente dovute a

- **forti non linearità di E , che creano “valli ripide” e zone “piatte” nella superficie della funzione di errore;**
- **elevata dimensionalità di w ed elevato numero n_T di campioni;**
- **presenza di minimi locali non globali.**

Una ulteriore difficoltà è costituita dal fatto che gli insiemi di livello della funzione d'errore, ossia gli insiemi

$$L(\alpha) = \{w \in R^m : E(w) \leq \alpha\}$$

possono non essere compatti, per cui non è possibile assicurare la convergenza globale di molti algoritmi a partire da punti iniziali arbitrari. E da notare, tuttavia, che in presenza di un termine di regolarizzazione *tutti* gli insiemi di livello sono compatti.

Uno dei primi algoritmi proposti per il calcolo dei pesi in una rete neurale è il metodo iterativo noto come **metodo di backpropagation** che è interpretabile come una versione euristica del *metodo del gradiente*. La riscoperta di questo metodo verso la metà degli anni '80 da parte di **Rumelhart , Hinton e Williams** ha reso possibile definire algoritmi di addestramento per reti multistrato e quindi è stata alla base del successivo considerevole sviluppo degli studi sulle reti neurali negli ultimi due decenni.

Sono state introdotte, in particolare, due classi principali di metodi iterativi per il calcolo dei pesi:

- **metodi batch** in cui ad ogni passo i pesi vengono aggiornati utilizzando informazioni relative a tutti i campioni dell'insieme di addestramento T ;
- **metodi on-line** in cui ad ogni passo i pesi vengono aggiornati tenendo conto soltanto di un singolo campione di T .

I **metodi batch** possono essere utilizzati esclusivamente per l'addestramento *fuori linea*, supponendo di disporre di tutto l'insieme T prima di avviare il processo di minimizzazione.

I **metodi on-line** possono essere impiegati sia per l'addestramento fuori linea, sia per l'addestramento in tempo reale, cioè quando gli elementi di T vengono acquisiti durante il processo di addestramento.

I metodi batch sono ovviamente riconducibili a **metodi di ottimizzazione non vincolata per la minimizzazione di C** . Infatti, nella letteratura neurale più recente sono sempre più frequentemente utilizzati, in luogo dei primi metodi euristici, metodi efficienti già sviluppati da tempo nel campo dell'ottimizzazione, che garantiscono la convergenza a *punti stazionari* della funzione di errore e usualmente assicurano una buona riduzione dell'obiettivo rispetto alla stima iniziale.

Il metodo di *backpropagation* (BP) è tuttora uno dei metodi di addestramento più diffusi. Il termine “**backpropagation**” (retropropagazione) è legato essenzialmente **alla tecnica utilizzata per il calcolo delle derivate della funzione di errore, basata sulle regole di derivazione delle funzioni composte**

Il metodo di BP è stato utilizzato in due versioni, note rispettivamente come:

- **BP batch**, in cui i pesi vengono aggiornati dopo la presentazione di tutti i campioni del training set T ;
- **BP on-line**, in cui i pesi vengono aggiornati in corrispondenza a ciascun campione di T .

La **BP batch** è definita dall'iterazione

$$w^{k+1} = w^k - \eta \nabla C(w^k),$$

dove $\nabla C(w^k)$ è il *gradiente* di C nel vettore corrente w^k e lo scalare $\eta > 0$ (detto *learning rate*) definisce il passo lungo l'antigradiente

La BP on-line consiste invece nel selezionare ad ogni passo un campione $(x^{j(k)}, y^{j(k)})$ dell'insieme di addestramento e nell'aggiornare i pesi utilizzando soltanto il termine $\nabla L_{j(k)}$ del gradiente di C , ossia nel porre:

$$w^{k+1} = w^k - \eta \nabla L_{j(k)}(w^k),$$

Il metodo di **BP è in genere inefficiente e può non convergere se il passo η non è scelto in modo appropriato;** nella letteratura neurale sono state quindi considerate **varie tecniche euristiche per effettuare una scelta adattativa del passo e per modificare la direzione di ricerca.**

Adesso introduciamo la procedura di calcolo del gradiente mediante backpropagation, che si può ricondurre a una particolare tecnica di *differenziazione automatica* .

Richiamiamo prima i seguenti strumenti di calcolo differenziale utili per la comprensione della backpropagation

Derivata di una funzione composta

Chain rule

- Se $g: R \rightarrow R$ e $f: R \rightarrow R$ sono derivabili, allora $g \circ f : R \rightarrow R$ è differenziabile, e se poniamo $\mathbf{h}(x) = g(f(x))$
- $\frac{dh}{dx} = g'(f(x)) \cdot f'(x)$
- Se $q: R \rightarrow R$ e $g: R \rightarrow R$ e $f: R \rightarrow R$ sono derivabili, allora $q \circ g \circ f: R \rightarrow R$ è derivabile,
$$\mathbf{h}(x) = q(g(f(x)))$$
- $\frac{dh}{dx} = q'(g(f(x)) \cdot g'(f(x)) \cdot f'(x))$

Derivata composta di funzioni di più variabili reali:

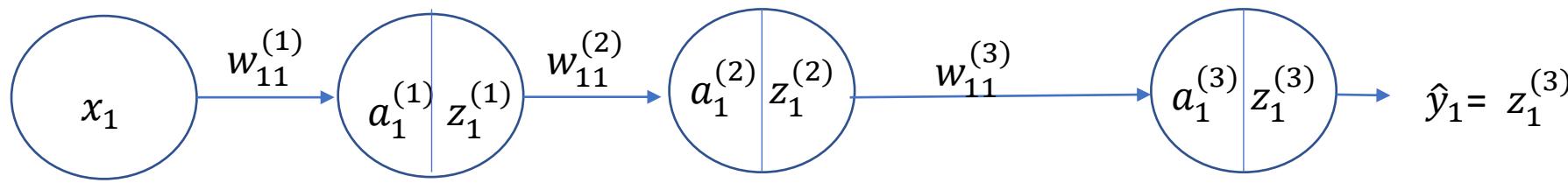
- Se $x(t) = (x_1(t), x_2(t), \dots, x_n(t))$ è un vettore di R^n le cui componenti sono funzioni derivabili e se f è una funzione differenziabile in $x(t)$, allora la funzione composta $F(t) = f(x(t))$ è differenziabile nella variabile t e si ha:

$$F'(t) = \sum_{i=1}^n \frac{\partial f(x(t))}{\partial x_i} \cdot x'_i(t) = \langle \nabla f(x(t)), x'(t) \rangle$$

- Esempio: data la funzione $f(h(t), g(t))$, la derivata di f rispetto a t si calcola come:

$$\frac{df}{dt} = \frac{df}{dh} \frac{dh}{dt} + \frac{df}{dg} \frac{dg}{dt}$$

Consideriamo il seguente esempio di rete MLP molto semplice, formata da un nodo di input, 2 layer nascosti ciascuno dei quali costituito da un solo neurone ed un nodo di output:



Sia y_1 l'etichetta di x_1 . La loss function che misura l'errore tra l'output prodotto dalla rete e il valore atteso è $L(y_1, \hat{y}_1)$.

Ricordiamo che il nostro compito è, in questo semplice esempio, aggiornare i pesi w in maniera tale da rendere minimo $L(y_1, \hat{y}_1)$, cioè individuare i pesi w tali che

$$\arg \min_w L(y_1, \hat{y}_1(w))$$

Per ottenere l'insieme di pesi $w = [w_{11}^{(1)}, w_{11}^{(2)}, w_{11}^{(3)}]$ ricorreremo al metodo di discesa del gradiente.

$$w^{k+1} = w^k - \eta \nabla L(w^k),$$

E' necessario calcolare:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{(1)}} \\ \frac{\partial L}{\partial w_{11}^{(2)}} \\ \frac{\partial L}{\partial w_{11}^{(3)}} \end{bmatrix}$$

$$\begin{aligned} L(\hat{y}_1) &= L(z_1^{(3)}) = L(z_1^{(3)}(a_1^{(3)})) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)})))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(z_1^{(1)})))) = \\ &= L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(z_1^{(1)}(a_1^{(1)})))))) = L(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(z_1^{(1)}(a_1^{(1)}(w_{11}^{(1)}))))))) \end{aligned}$$

Calcoliamo la derivata parziale di L rispetto a $w_{11}^{(1)}$

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} \frac{\partial z_1^{(1)}}{\partial a_1^{(1)}} \frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}}$$

$$\frac{\partial L}{\partial z_1^{(3)}} = \frac{\partial L}{\partial \hat{y}_1}$$

Si calcola facilmente derivando rispetto all'output della rete l'espressione analitica della loss-function

$$z_1^{(3)} = f(a_1^{(3)})$$

$$\frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} = f'(a_1^{(3)})$$

$$a_1^{(3)} = w_{11}^{(3)} z_1^{(2)}$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} = w_{11}^{(3)}$$

$$z_1^{(2)} = f(a_1^{(2)})$$

$$\frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} = f'(a_1^{(2)})$$

$$a_1^{(2)} = w_{11}^{(2)} z_1^{(1)}$$

$$\frac{\partial a_1^{(2)}}{\partial z_1^{(1)}} = w_{11}^{(2)}$$

$$z_1^{(1)} = f(a_1^{(1)})$$

$$\frac{\partial \textcolor{red}{z_1^{(1)}}}{\partial \textcolor{red}{a_1^{(1)}}} = \textcolor{red}{f'(a_1^{(1)})}$$

$$a_1^{(1)} = w_{11}^{(1)} z_1^{(0)} \qquad \qquad z_1^{(0)} = x_1$$

$$\frac{\partial \textcolor{red}{a_1^{(1)}}}{\partial \textcolor{red}{w_{11}^{(1)}}} = \textcolor{red}{z_1^{(0)}}$$

$$\frac{\partial L}{\partial \textcolor{teal}{w_{11}^{(1)}}} \!=\! \frac{\partial L}{\partial \hat{y}_1} \, \textcolor{blue}{f'(a_1^{(3)})} \textcolor{teal}{w_{11}^{(3)}} \, \textcolor{blue}{f'(a_1^{(2)})} \textcolor{teal}{w_{11}^{(2)}} \textcolor{blue}{f'(a_1^{(1)})} \textcolor{teal}{z_1^{(0)}}$$

Calcoliamo la derivata parziale di L rispetto a $w_{11}^{(2)}$

$$L(\hat{y}_1) = L\left(z_1^{(3)}\right) = L\left(z_1^{(3)}(a_1^{(3)})\right) = L\left(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}))\right) = L\left(z_1^{(3)}(a_1^{(3)}(z_1^{(2)}(a_1^{(2)}(w_{11}^{(2)})))\right)$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial w_{11}^{(2)}}$$

$$a_1^{(2)} = w_{11}^{(2)} z_1^{(1)}$$

$$\frac{\partial a_1^{(2)}}{\partial w_{11}^{(2)}} = z_1^{(1)}$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) z_1^{(1)}$$

Calcoliamo la derivata parziale di L rispetto a $w_{11}^{(3)}$

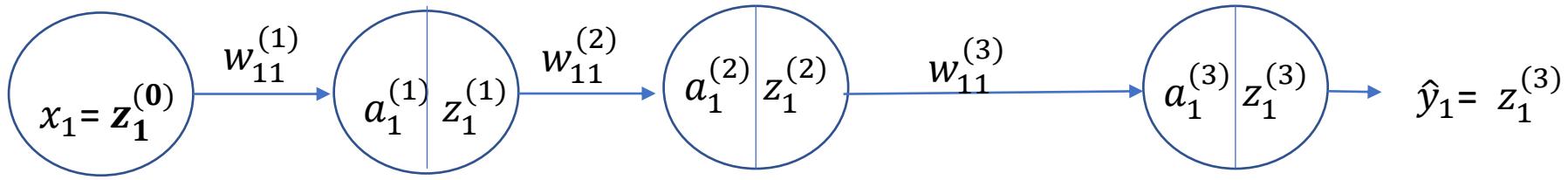
$$L(\hat{y}_1) = L\left(z_1^{(3)}\right) = L\left(z_1^{(3)}(a_1^{(3)})\right) = L\left(z_1^{(3)}(a_1^{(3)}(w_{11}^{(3)}))\right)$$

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \frac{\partial L}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial w_{11}^{(3)}}$$

$$a_1^{(3)} = w_{11}^{(3)} z_1^{(2)}$$

$$\frac{\partial a_1^{(3)}}{\partial w_{11}^{(3)}} = z_1^{(2)}$$

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) z_1^{(2)}$$



$$\frac{\partial L}{\partial w_{11}^{(1)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) w_{11}^{(2)} f'(a_1^{(1)}) z_1^{(0)} =$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) w_{11}^{(3)} f'(a_1^{(2)}) z_1^{(1)}$$

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)}) z_1^{(2)}$$

Poniamo:

$$\delta_1^{(3)} = \frac{\partial L}{\partial \hat{y}_1} f'(a_1^{(3)})$$

$$\delta_1^{(2)} = \delta_1^{(3)} w_{11}^{(3)} f'(a_1^{(2)})$$

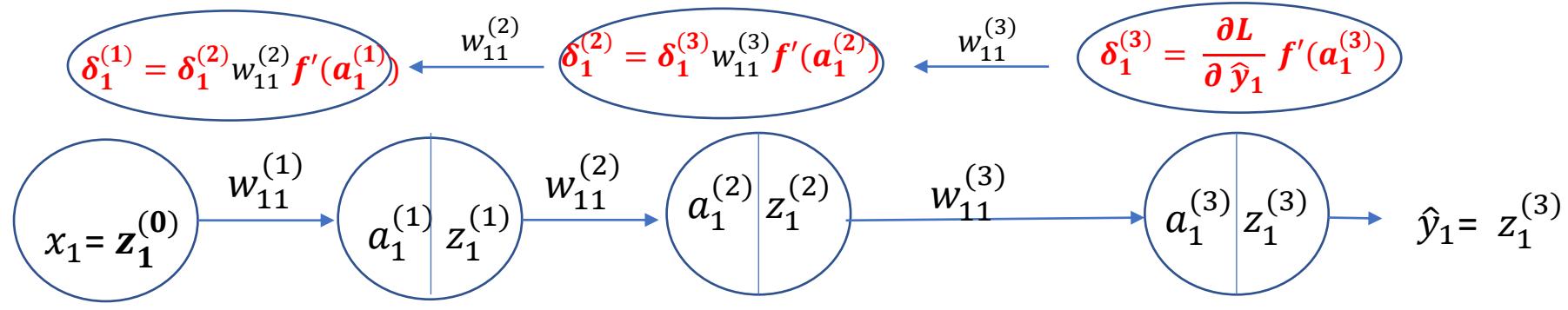
$$\delta_1^{(1)} = \delta_1^{(2)} w_{11}^{(2)} f'(a_1^{(1)})$$

Le formule del gradiente della loss function L rispetto a tutti i pesi
 $\{w_{11}^{(1)}, w_{11}^{(2)}, w_{11}^{(3)}\}$ si potranno quindi così esprimere:

$$\frac{\partial L}{\partial w_{11}^{(1)}} = \delta_1^{(1)} z_1^{(0)}$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \delta_1^{(2)} z_1^{(1)}$$

$$\frac{\partial L}{\partial w_{11}^{(3)}} = \delta_1^{(3)} z_1^{(2)}$$



Aggiornamento dei pesi per l'epoca successiva:

$$w^{k+1} = w^k - \eta \nabla L(w^k),$$

Omettendo per semplicità di scrittura l'indice k dell'epoca, l'ultima relazione è equivalente a:

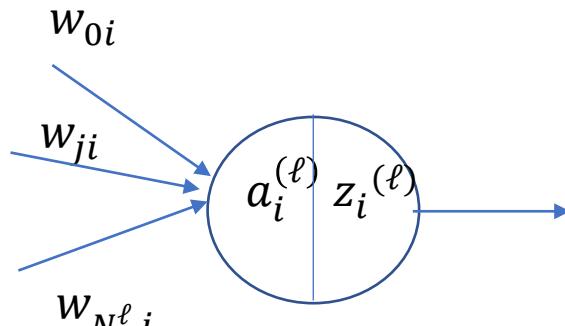
$$w_{11}^{(1)} = w_{11}^{(1)} - \eta \delta_1^{(1)} x_1$$

$$w_{11}^{(2)} = w_{11}^{(2)} - \eta \delta_1^{(2)} z_1^{(1)}$$

$$w_{11}^{(3)} = w_{11}^{(3)} - \eta \delta_1^{(3)} z_1^{(2)}$$

Back propagation (dimostrazione)

$$a_i^{(\ell)} = \sum_{j=0}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} \quad z_i^{(\ell)} = f(a_i^{(\ell)}) \quad z_i^{(0)} = x_i, \quad i = 0, \dots, d$$



$$\frac{\partial L}{\partial w_{ji}^{(\ell)}} = \frac{\partial L}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial w_{ji}^{(\ell)}}$$

$$a_i^{(\ell)} = \sum_{j=0}^{N^{(\ell)}} w_{ji}^{(\ell)} z_j^{(\ell-1)} = w_{0i}^{(\ell)} z_0^{(\ell-1)} + w_{1i}^{(\ell)} z_1^{(\ell-1)} + \dots + w_{ji}^{(\ell)} z_j^{(\ell-1)} + \dots + w_{N^\ell i}^{(\ell)} z_{N^\ell}^{(\ell-1)}$$

$$\frac{\partial a_i^{(\ell)}}{\partial w_{ji}^{(\ell)}} = z_j^{(\ell-1)}$$

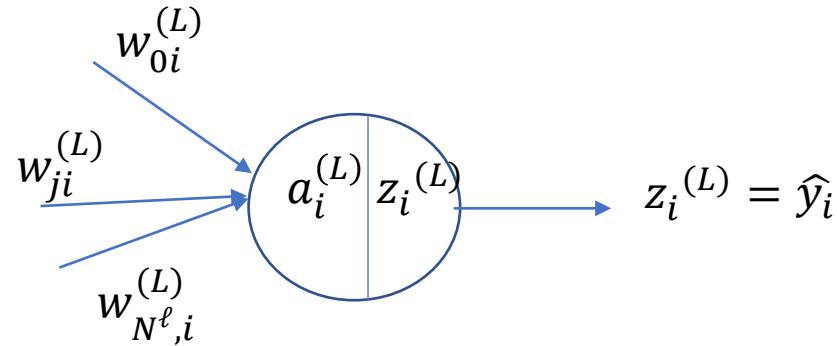
Definiamo $\delta_i^{(\ell)} := \frac{\partial L}{\partial a_i^{(\ell)}}$

$$\frac{\partial L}{\partial w_{ji}^{(\ell)}} = \delta_i^{(\ell)} z_j^{(\ell-1)}$$

Per il calcolo di $\delta^{(i)}$ consideriamo due casi:

Caso 1: Il neurone appartiene allo **strato di uscita L**

$$\delta_i^{(L)} = \frac{\partial L(\hat{y}_i)}{\partial a_i^{(L)}}$$



$$L(\hat{y}_i) = L\left(z_i^{(L)}\right) = L\left(z_i^{(L)}(a_i^{(L)})\right)$$

$$\frac{\partial L(\hat{y}_i)}{\partial a_i^{(L)}} = \frac{\partial L(\hat{y}_i)}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial a_i^{(L)}} = \frac{\partial L(\hat{y}_i)}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial a_i^{(L)}}$$

$\frac{\partial L(\hat{y}_i)}{\partial \hat{y}_i}$ viene calcolato analiticamente derivando la formula analitica della funzione loss.

Essendo

$$z_i^{(L)} = f(a_i^{(L)})$$

segue:

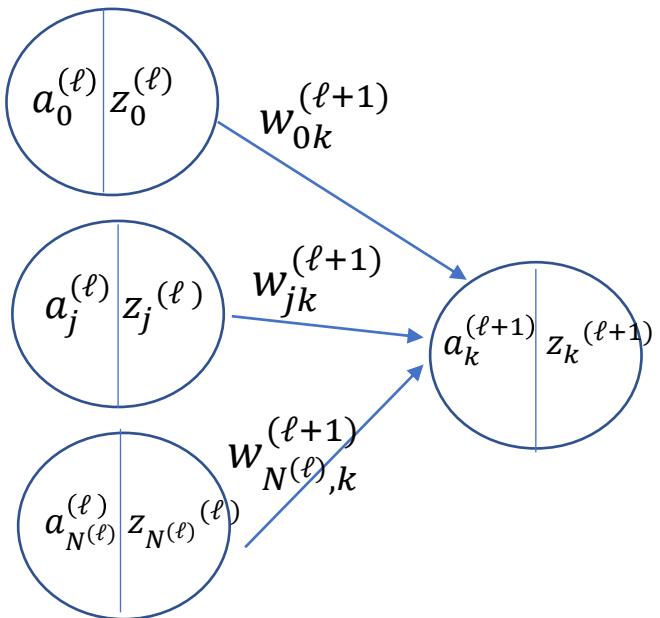
$$\frac{\partial z_i^{(L)}}{\partial a_i^{(L)}} = f'(a_i^{(L)})$$

Quindi, nel caso di **nodi appartenenti al layer di uscita** si ha:

$$\delta_i^{(L)} = \frac{\partial L(\hat{y}_i)}{\partial a_i^{(L)}} = f'(a_i^{(L)}) \frac{\partial L(\hat{y}_i)}{\partial \hat{y}_i}$$

Caso 2: Il neurone **non** appartiene allo **strato di uscita**

Indichiamo **con i un neurone nascosto**:



Dobbiamo calcolare $\frac{\partial L}{\partial a_i^{(\ell)}}$

L dipende da $a_i^{(\ell)}$ tramite $a_k^{(\ell+1)}$, $k = 0, \dots, N^{(\ell)}$

Essendo $z_i^{(\ell)} = f(a_i^{(\ell)})$

$$a_k^{(\ell+1)} = \sum_{j=0}^{N^{(\ell)}} w_{jk}^{(\ell+1)} z_j^{(\ell)} = \sum_{j=0}^{N^{(\ell)}} w_{jk}^{(\ell+1)} f(a_j^{(\ell)})$$

$$\delta_i^{(\ell)} := \frac{\partial L}{\partial a_i^{(\ell)}} = \sum_{k=0}^{N^{(\ell)}} \frac{\partial L}{\partial a_k^{(\ell+1)}} \frac{\partial a_k^{(\ell+1)}}{\partial a_i^{(\ell)}}$$

$$\delta_k^{(\ell+1)} := \frac{\partial L}{\partial a_k^{(\ell+1)}}$$

$$a_k^{(\ell+1)} = w_{0k}^{(\ell+1)} f(a_0^{(\ell)}) + w_{1k}^{(\ell+1)} f(a_1^{(\ell)}) + \dots + w_{ik}^{(\ell+1)} f(a_i^{(\ell)}) + \dots + w_{N^{(l)}k}^{(\ell+1)} f(a_{N^{(\ell)}}^{(\ell)})$$

Segue facilmente che

$$\frac{\partial a_k^{(\ell+1)}}{\partial a_i^{(\ell)}} = f'(a_i^{(\ell)}) w_{ik}^{(\ell+1)}$$

E quindi

$$\delta_i^{(\ell)} := \frac{\partial L}{\partial a_i^{(\ell)}} = \sum_{k=0}^{N^{(\ell)}} \delta_k^{(\ell+1)} f'(a_i^{(\ell)}) w_{ik}^{(\ell+1)}$$

che può essere riscritto come:

$$\delta_i^{(\ell)} = f'(a_i^{(\ell)}) \sum_{k=0}^{N^{(\ell)}} \delta_k^{(\ell+1)} w_{ik}^{(\ell+1)}$$

Quindi,

$$\frac{\partial L}{\partial w_{ji}^{(\ell)}} = \delta_i^{(\ell)} z_j^{(\ell-1)}$$

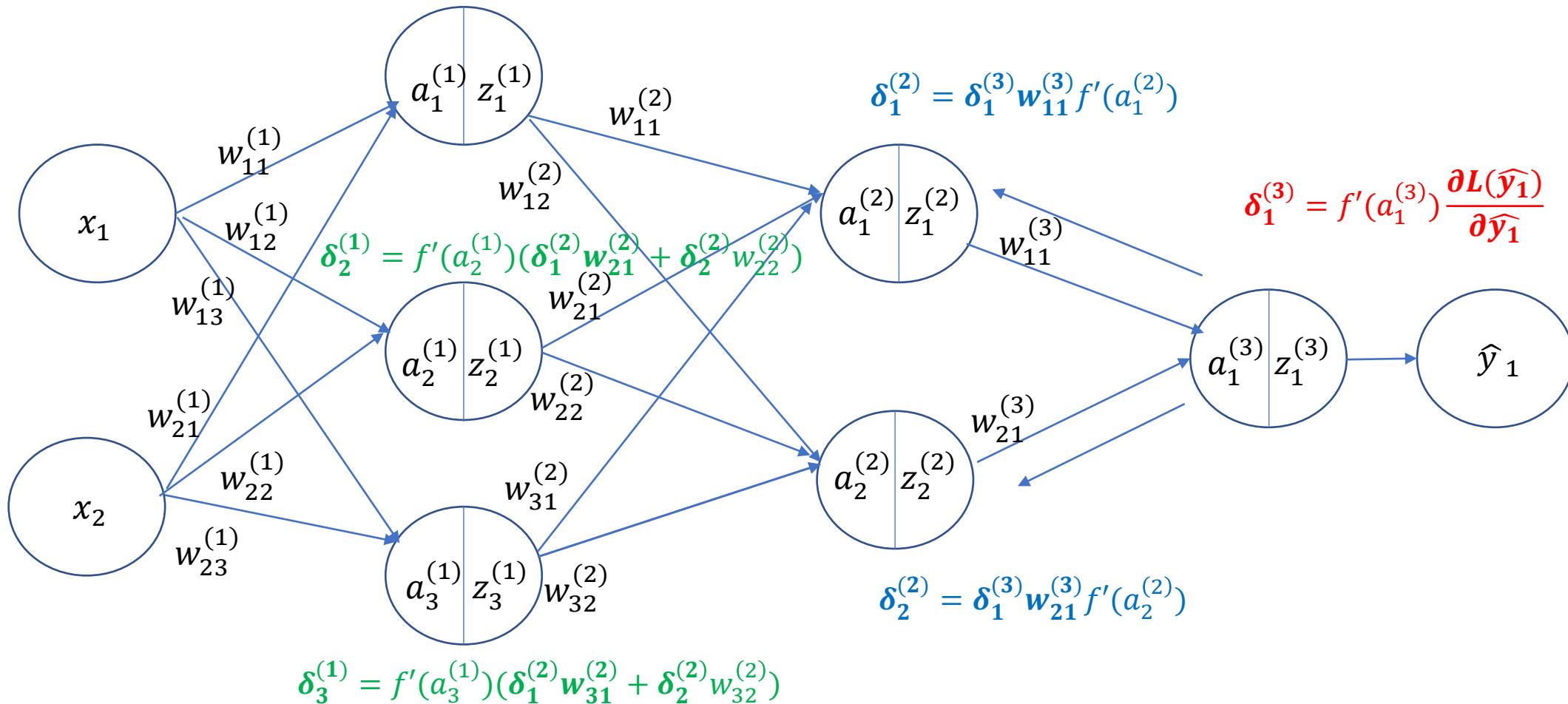
dove, se il neurone $i, i = 1, \dots, k$, appartiene al layer di uscita L ,

$$\delta_i^{(L)} = \frac{\partial L(\hat{y}_i)}{\partial a_i^{(L)}} = f'(a_i^{(L)}) \frac{\partial L(\hat{y}_i)}{\partial \hat{y}_i}$$

se il neurone $i, i = 1, \dots, k$, non appartiene al layer di uscita, ma al layer nascosto ℓ ,

$$\delta_i^{(\ell)} = f'(a_i^{(\ell)}) \sum_{k=0}^{N^{(\ell)}} \delta_k^{(\ell+1)} w_{ik}^{(\ell+1)}$$

$$\delta_1^{(1)} = f'(a_1^{(1)}) (\delta_1^{(2)} w_{11}^{(2)} + \delta_2^{(2)} w_{12}^{(2)})$$



Sia $\hat{y} = [\hat{y}_1, \dots, \hat{y}_s]$ l'output prodotto dalla rete in corrispondenza all'esempio di training $x = [x_1, x_2, \dots, x_d]$, e sia $y = [y_1, y_2, \dots, y_s]$ il vettore delle etichette corrispondente dell'esempio di training.

Se scegliamo come **loss function** la somma dei quadrati degli errori tra l'output della rete e la corrispondente etichetta:

$$L(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^s (y_i - \hat{y}_i)^2 = \frac{1}{2} \|y - \hat{y}\|_2^2$$

Ricordando che:

$$L(y, \hat{y}) = \frac{1}{2} ((y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2 + \dots + (y_i - \hat{y}_i)^2 + \dots + (y_s - \hat{y}_s)^2)$$

Segue facilmente che la derivata parziale di L rispetto all'output della rete $\hat{y}_i, i = 1, \dots, s$ sarà dato da :

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{1}{2} 2(y_i - \hat{y}_i)(-1) = -(y_i - \hat{y}_i) = \hat{y}_i - y_i$$

Quiz a scelta multipla tratti dalle prove scritte di Fisica a Ingegneria e Scienze Informatiche nell'AA 2020/21 - Prof. Guiducci

MECCANICA

Se mi trovo su una giostra che gira in modo uniforme e mi sposto radialmente dal centro verso il bordo della giostra, quale grandezza del mio moto varia, se misurata da un osservatore che si trovi sulla strada?

- a. l'angolo descritto nell'unità di tempo
- b. la frequenza
- c. nessuna delle altre risposte
- d. il periodo
- e. la velocità tangenziale

Sia $x(t) = 3t^3 - 5t^2 + 2$ la legge oraria del moto di un punto materiale. Il punto materiale si muove di:

- a. Moto uniforme
- b. È necessario conoscere la massa del corpo
- c. Nessuna delle altre risposte
- d. Moto con accelerazione variabile
- e. Moto uniformemente accelerato

Se un oggetto è *in equilibrio*, quale delle seguenti affermazioni **non** è corretta?

- a. La forza risultante sull'oggetto è zero
- b. La velocità dell'oggetto rimane costante
- c. L'oggetto deve essere a riposo
- d. L'accelerazione dell'oggetto è zero
- e. Se sull'oggetto agiscono una o più forze, queste devono essere almeno due

Una stella di neutroni ha un raggio di 20 km e ruota alla velocità di 1 giro al secondo. La massa minima della stella affinché un oggetto posto sulla superficie all'equatore non sfugga dalla superficie della stella è circa

- a. $3 \times 10^{18} \text{ kg}$
- b. nessuna delle altre risposte
- c. $1 \times 10^{20} \text{ kg}$
- d. $8 \times 10^{30} \text{ kg}$
- e. $5 \times 10^{24} \text{ kg}$

Il lavoro necessario per portare un punto materiale di massa pari a 10 kg inizialmente fermo a una velocità di 20 m/s è

- a. 200 J
- b. 2000 W
- c. 200 W
- d. 2000 J
- e. nessuna delle altre risposte

Un sasso viene scagliato verso il basso dall'alto di una torre. Se si trascura la resistenza dell'aria l'accelerazione del sasso durante la caduta:

- a. dipende dalla massa del sasso
 - b. dipende dalla spinta iniziale
 - c. è circa uguale a 9.8 m/s^2
 - d. nessuna delle altre risposte
 - e. è maggiore di 9.8 m/s^2
-
-

Quali delle seguenti affermazioni sul moto circolare uniforme sono corrette? (È possibile più di una risposta)

- a. È causato da una forza netta di modulo costante con verso uscente dal centro
 - b. Causa una forza centripeta
 - c. La velocità vettoriale del corpo è costante
 - d. È la composizione di due moti armonici su assi cartesiani con origine nel centro
 - e. È causato da una forza netta di modulo costante diretta verso il centro
-
-

Una cassa è posta su un piano inclinato, e resta ferma. Quali affermazioni riguardo al modulo della forza di attrito statico agente sulla cassa sono corrette?

- a. È maggiore del modulo della componente parallela al piano della forza di gravità agente sulla cassa
 - b. È uguale al modulo della componente parallela al piano della forza di gravità agente sulla cassa
 - c. È inferiore al modulo della componente parallela al piano della forza di gravità agente sulla cassa
 - d. È pari a $\mu_s N$ dove μ_s è il coefficiente di attrito statico e N è il modulo della forza normale
 - e. È maggiore del modulo del peso della cassa
-

Quale tra le seguenti grandezze non ha le dimensioni di un'energia. Si consideri che m è una massa, g è l'accelerazione di gravità, h e d sono lunghezze, F è una forza, v una velocità, a un'accelerazione, P è una potenza e t il tempo.

- a. Fd
 - b. mgh
 - c. mv^2
 - d. Pt
 - e. ma
-

Quali delle seguenti relazioni tra accelerazione a e spostamento x di una particella determinano un moto armonico semplice?

- a. $a = -3x^2$
 - b. Nessuna delle altre risposte è corretta
 - c. $a = 0.5x$
 - d. $a = 400x^2$
 - e. $a = -20x$
-

Se con un'astronave mi trasferisco dalla superficie della Terra sulla superficie della Luna, cosa cambia?

- a. nessuna delle altre risposte è corretta
 - b. né la mia massa, né il mio peso
 - c. sia la mia massa che il mio peso
 - d. la mia massa ma non il mio peso
 - e. il mio peso ma non la mia massa
-

Un camion può percorrere una curva di raggio 150 m alla velocità massima di 32.0 m/s. A quale velocità massima può percorrere una curva di raggio 75.0 m?

- a. 16 m/s
 - b. 23 m/s
 - c. 45 m/s
 - d. 64 m/s
 - e. 32 m/s
-
-

Un camion frena bruscamente (bloccando le ruote) e si ferma dopo avere percorso una distanza d . In un secondo caso in cui il camion si muove con la stessa velocità, il carico dell'autocarro è tale che ha complessivamente il doppio della massa rispetto al primo caso. Quale sarà ora lo spazio di frenata?

- a. $\sqrt{2}d$
- b. $d/2$
- c. d
- d. $4d$
- e. $2d$

Due punti materiali di massa $m_A = 10 \text{ kg}$ e $m_B = 20 \text{ kg}$ sono lanciati dalla stessa altezza h , con velocità iniziali verticali e uguali in modulo ($|v_0| = 12 \text{ m/s}$), ma il corpo A verso l'alto, il corpo B verso il basso. Durante il moto la resistenza dell'aria può essere trascurata. Una volta giunti al suolo, il rapporto dei moduli delle loro velocità finali $v_{f,A}$ e $v_{f,B}$ sarà:

- a. $v_{f,A}/v_{f,B} = 2$
- b. nessuna delle altre risposte
- c. $v_{f,A}/v_{f,B} = 1$
- d. non è possibile stabilirlo con i dati disponibili
- e. $v_{f,A}/v_{f,B} = 0.5$

Un corpo di $m = 1 \text{ kg}$ attaccato ad una molla oscilla con periodo $T = 3 \text{ s}$. Qual è il periodo di oscillazione T' di un corpo di massa $m' = 9 \text{ kg}$?

- a. $T' = 27 \text{ s}$
- b. nessuna delle altre risposte
- c. $T' = 9 \text{ s}$
- d. occorre conoscere la costante elastica
- e. $T' = 1 \text{ s}$

Una forza si dice conservativa se e solo se

- a. non produce una variazione di energia potenziale
- b. non compie lavoro per qualsiasi traiettoria
- c. il lavoro dipende solo dal punto di partenza e da quello di arrivo della traiettoria
- d. il lavoro è proporzionale alla lunghezza della traiettoria
- e. nessuna delle altre risposte

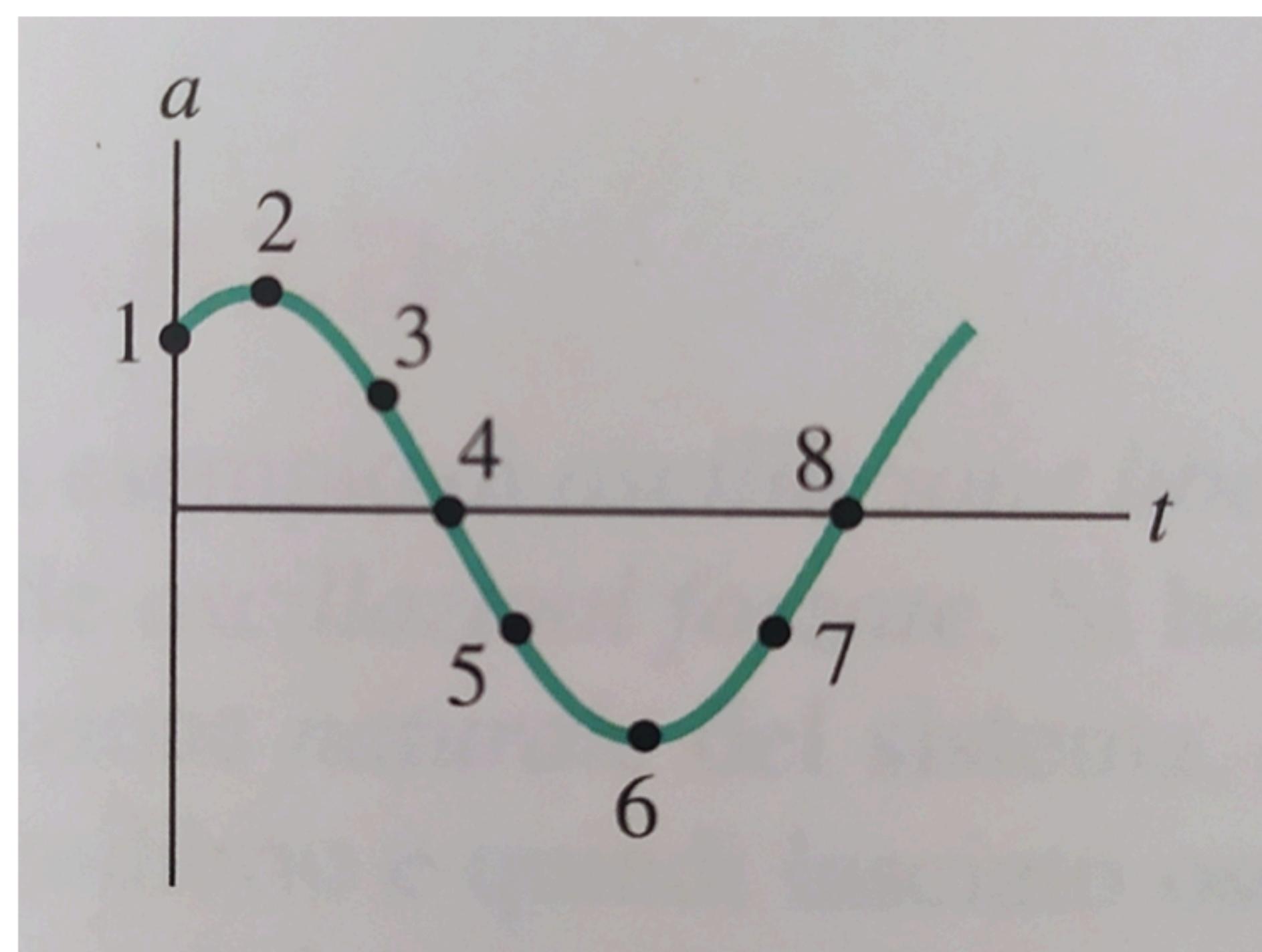
La seconda legge di Keplero si può spiegare secondo Newton perché la forza gravitazionale

- a. nessuna delle altre risposte
- b. è centrale e conserva il momento angolare
- c. dipende dalle masse di entrambi i corpi interagenti
- d. è proporzionale all'inverso del quadrato della distanza
- e. è responsabile della forza centrifuga

Una palla viene lanciata dritto verso l'alto. Quali sono velocità e accelerazione nel punto più alto del suo percorso?

- a. $v = 0, a = 0$
- b. $v = 0, a \approx 9.81 \text{ m/s}^2$ verso l'alto
- c. $v = 0, a \approx 9.81 \text{ m/s}^2$ verso il basso
- d. $v \approx 9.81 \text{ m/s}$ verso l'alto, $a = 0$
- e. $v \approx 9.81 \text{ m/s}$ verso il basso, $a = 0$

Nella figura è riportata la legge oraria dell'accelerazione $a(t) = \frac{d^2x}{dt^2}$ di una particelle che si muove di moto armonico tra i punti $-A$ e $+A$ di un asse x . Quale dei punti indica che la particella si trova nella posizione $x = -A$?



- a. 1
- b. 2
- c. 3
- d. 4
- e. 5
- f. 6
- g. 7
- h. 8

Un oggetto di massa m si muove con accelerazione a lungo un piano inclinato dotato di attrito. Quali delle seguenti forze dovrebbero comparire nel diagramma di corpo libero dell'oggetto? Scegli tutte le risposte corrette.

- a. La forza di gravità esercitata dalla terra
- b. ma nella direzione del moto
- c. La forza normale esercitata dal piano inclinato
- d. L'attrito esercitato dal piano inclinato
- e. La forza esercitata dall'oggetto sul piano inclinato

Un moto circolare uniforme di un corpo di massa m è definito dal raggio R della circonferenza e dalla velocità tangenziale v . Per ottenere una situazione in cui la forza centripeta necessaria raddoppi, agendo su uno solo dei parametri mentre gli altri due sono mantenuti invariati, posso:

- a. dimezzare la massa m
- b. raddoppiare il raggio R
- c. raddoppiare la velocità v
- d. dimezzare il raggio R
- e. nessuna delle altre risposte

Indicare la relazione corretta:

- a. $[MLT^{-1}] = W$
 - b. $[ML^2T^{-1}] = W$
 - c. $[ML^2T^2] = J$
 - d. $[MLT^3] = J$
 - e. nessuna delle altre risposte
-
-

L'energia potenziale gravitazionale di un sistema di due corpi

- a. è massima se sono a distanza infinita
- b. è minima se sono a distanza infinita
- c. è massima se sono a distanza nulla
- d. dipende solo dalla massa del corpo più grande
- e. nessuna delle altre risposte

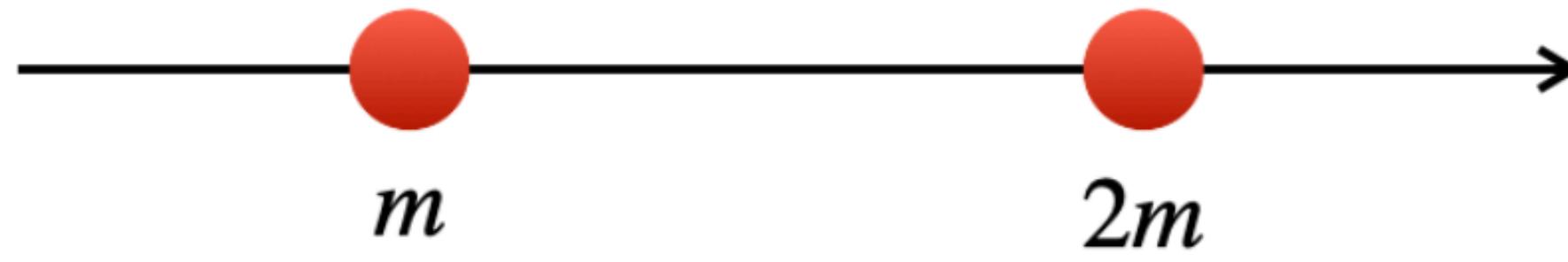
Due punti materiali di massa $m_A = 10 \text{ kg}$ e $m_B = 20 \text{ kg}$ sono lanciati dalla stessa altezza h , con velocità iniziali verticali e uguali in modulo ($|v_0| = 12 \text{ m/s}$), ma il corpo A verso l'alto, il corpo B verso il basso. Durante il moto la resistenza dell'aria può essere trascurata. Una volta giunti al suolo, il rapporto dei moduli delle loro velocità finali $v_{f,A}$ e $v_{f,B}$ sarà:

- a. non è possibile stabilirlo con i dati disponibili
 - b. $v_{f,A}/v_{f,B} = 2$
 - c. $v_{f,A}/v_{f,B} = 1$
 - d. $v_{f,A}/v_{f,B} = 0.5$
 - e. nessuna delle altre risposte
-
-

Una cassa di massa m è posta sul pianale di un camion senza essere assicurata. Quando il camion accelera in avanti con accelerazione a , la cassa resta ferma rispetto al camion. Quale forza provoca l'accelerazione della cassa?

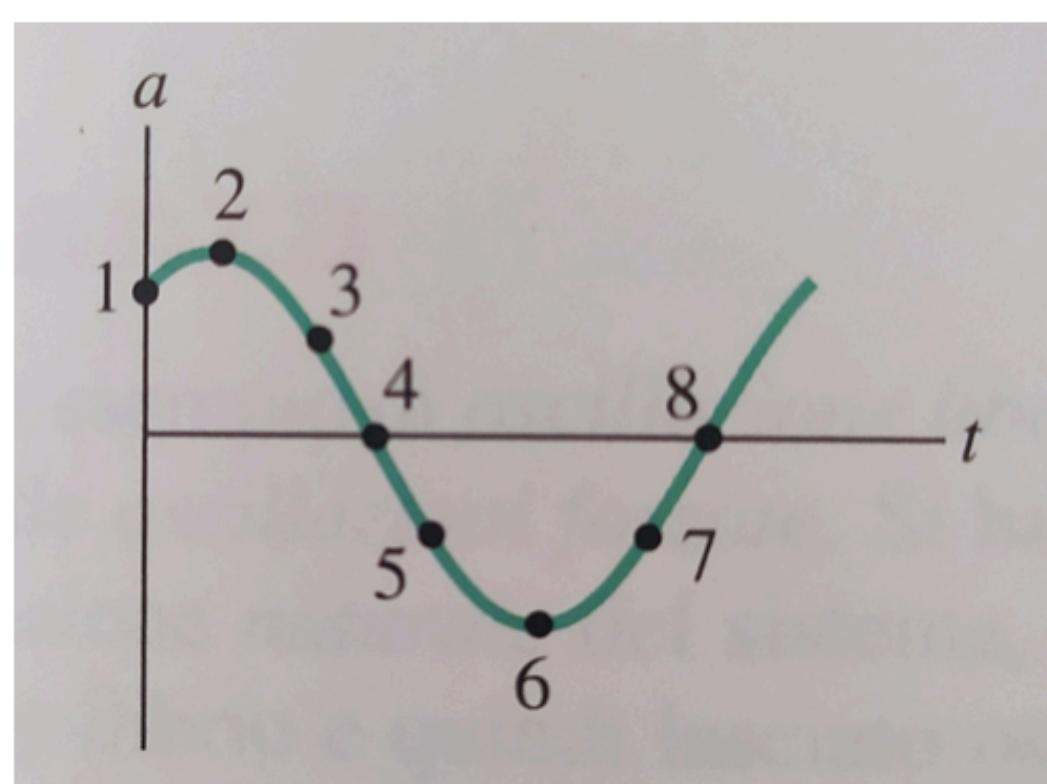
- a. Non è necessaria alcuna forza
- b. La forza ma esercitata dalla cassa
- c. La forza di attrito
- d. La forza normale
- e. La forza di gravità

Due particelle di massa m e $2m$ sono situate su un asse come in figura. Dove dovrebbe collocarsi sull'asse una terza particella di massa $3m$ per fare in modo che su di essa sia esercitata una forza gravitazionale pari a zero?



- a. A destra di entrambe
 - b. A metà tra le due
 - c. Tra le due ma più vicino a quella di massa minore
 - d. A sinistra di entrambe
 - e. Tra le due ma più vicino a quella di massa maggiore
-

Nella figura è riportata la legge oraria dell'accelerazione $a(t) = \frac{d^2x}{dt^2}$ di una particella che si muove di moto armonico tra i punti $-A$ e $+A$ di un asse x . Quale dei punti seguenti indica che la particella ha velocità massima?



- a. 3
- b. 2
- c. 1
- d. 5
- e. 4
- f. 7
- g. 6

Un astronauta sta girando in una centrifuga di raggio 5 m. Quanti giri al minuto compie l'astronauta se l'accelerazione che subisce ha modulo $7g$?

- a. Circa 180 giri al minuto
- b. Non è possibile determinarlo
- c. Circa 35 giri al minuto
- d. Nessuna delle altre risposte
- e. Circa 130 giri al minuto

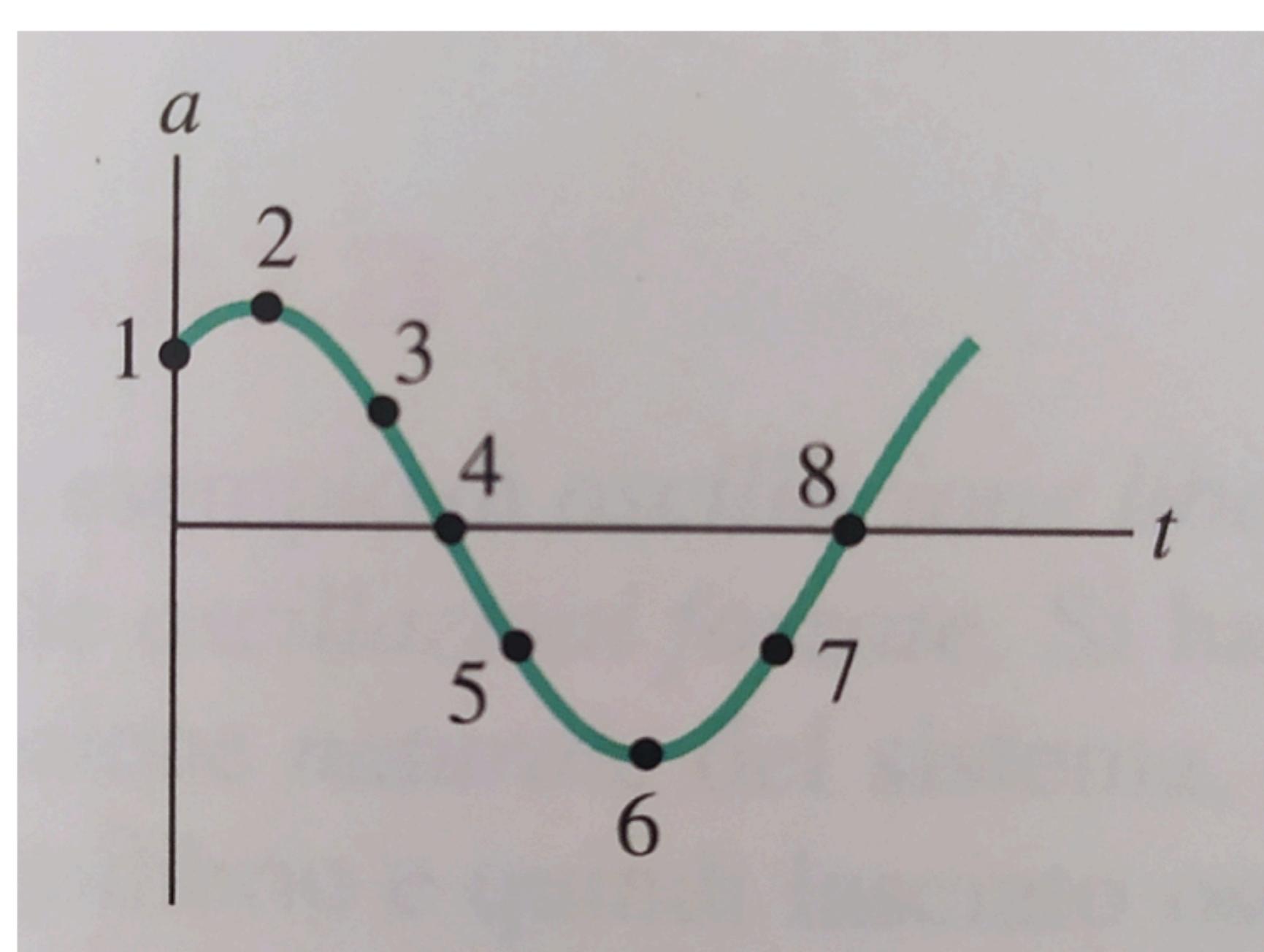
Un moto unidimensionale lungo la coordinata x , che rappresenta una lunghezza, avviene secondo la legge oraria della velocità: $v_x(t) = at + bx \cos(cxt)$, dove a, b e c sono costanti. Le dimensioni delle tre costanti a, b, c devono essere:

- a. $[a] = L^{-2}T^2, [b] = LT^{-1}, [c] = T^{-1}$
- b. $[a] = LT^{-2}, [b] = T^{-1}, [c] = L^{-1}T^{-1}$
- c. nessuna delle altre risposte
- d. $[a] = LT^{-1}, [b] = T^{-1}, [c] = LT$
- e. $[a] = L^2, [b] = T, [c] = L^{-1}T^{-1}$

Un camion frena bruscamente (bloccando le ruote) e si ferma dopo avere percorso una distanza d . In un secondo caso l'autocarro sta procedendo a velocità doppia, quando frena. Quale sarà ora lo spazio di frenata?

- a. $4d$
- b. $2d$
- c. d
- d. $\sqrt{2}d$
- e. $d/2$

Nella figura è riportata la legge oraria dell'accelerazione $a(t) = \frac{d^2x}{dt^2}$ di una particelle che si muove di moto armonico tra i punti $-A$ e $+A$ di un asse x . Nel punto 5 la particella si trova:



- a. in $x = +A$
- b. in $x = 0$
- c. compresa tra $x = 0$ e $x = +A$
- d. in $x = -A$
- e. compresa tra $x = -A$ e $x = 0$

Si vuole creare la sensazione di gravità in una enorme stazione spaziale a forma di ciambella. Il raggio R della circonferenza che corrisponde alla parte abitata della stazione è di 980 m. A quale velocità deve ruotare per simulare un'accelerazione di gravità circa uguale a quella presente sulla superficie della terra?

- a. circa 1 giro all'ora
- b. nessuna delle altre risposte
- c. circa 1 radiante al secondo
- d. circa 1 giro al secondo
- e. circa 1 giro al minuto

Supponiamo di attaccare un dinamometro ad una parete e di tirare con una forza di modulo F . Successivamente, faccio reggere ad un amico l'estremità del dinamometro precedentemente attaccata alla parete, e io tiro con la stessa forza di prima. Il dinamometro resta fermo. Come è la lettura sul dinamometro in questo secondo caso, rispetto al primo?

- a. Uguale
- b. Il doppio
- c. Dipende dalle caratteristiche del dinamometro
- d. Minore
- e. Maggiore

Due pianeti orbitano intorno ad una stella. Il primo compie un'orbita circolare, di raggio $R_1 = 10^8$ km, con un periodo di rivoluzione di un anno terrestre. A che distanza dalla stella si trova il secondo pianeta, se completa un'orbita circolare in 1000 anni terrestri?

- a. 10^{12} km
- b. nessuna delle altre risposte
- c. 10^{16} km
- d. 10^{14} km
- e. 10^8 km

Se un oggetto è *in equilibrio*, quale delle seguenti affermazioni **non** è corretta?

- a. La velocità dell'oggetto rimane costante
- b. L'oggetto deve essere a riposo
- c. L'accelerazione dell'oggetto è zero
- d. La forza risultante sull'oggetto è zero
- e. Se sull'oggetto agiscono una o più forze, queste devono essere almeno due

Tre proiettili, che indichiamo con R, S e T, vengono sparati con velocità iniziale uguale in modulo, ma con diverso angolo di elevazione rispetto all'orizzontale; gli angoli sono rispettivamente 30° , 45° , 60° . Ordina i tre proiettili in base alla gittata che essi avrebbero in assenza di attrito (dalla massima alla minima).

- a. S; R e T alla pari
- b. R; S; T
- c. T; S; R
- d. nessuna delle risposte precedenti
- e. S; T; R



Deep Learning



Con il termine DNN (Deep Neural Network) si denotano reti « profonde » composte da molti livelli (almeno 2 hidden) organizzati gerarchicamente

Le DNN oggi maggiormente utilizzate consistono di un numero di livelli compreso tra 7 e 50

Reti più profonde (100 livelli e oltre) hanno dimostrato di poter garantire prestazioni leggermente migliori, a discapito però dell'efficienza



Principali tipologie di DNN

Modelli feedforward « discriminativi » per la classificazione (o regressione) con training prevalentemente supervisionato

- **FC-DNN: Fully Connected DNN** (MLP con almeno due livelli hidden)
- **CNN: Convolutional Neural Network (o ConvNet)**

Modelli ricorrenti con memoria e attenzione (utilizzati per sequenze, speech recognition natural language processing)

- **RNN**
- **Recurrent Neural Network**
- **LSTM**
- **Long Short Term Memory**
- **Transformers**

Addestramento non supervisionato: modelli addestrati a ricostruire l'input originale prendendo come input una versione a più bassa dimensionalità (utilizzati per denoising, anomaly detection)

- **Autoencoders**

Modelli « generativi » per generare dataset sintetici (data augmentation) style transfer, art applications

- **GAN Generative Adversarial Networks**
- **VAE Variational Autoencoders**

Reinforcement learning (per apprendere comportamenti)

- **Deep Q Learning**



Ingredienti necessari

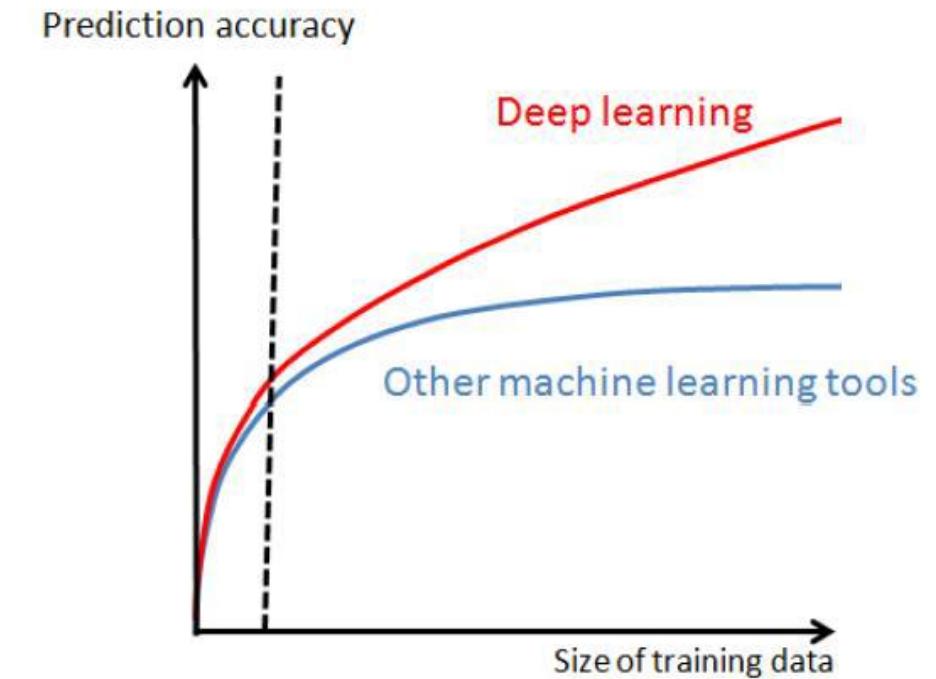
BigData

disponibilità di dataset etichettati di grandi dimensioni (es ImageNet milioni di immagini, decine di migliaia di classi)

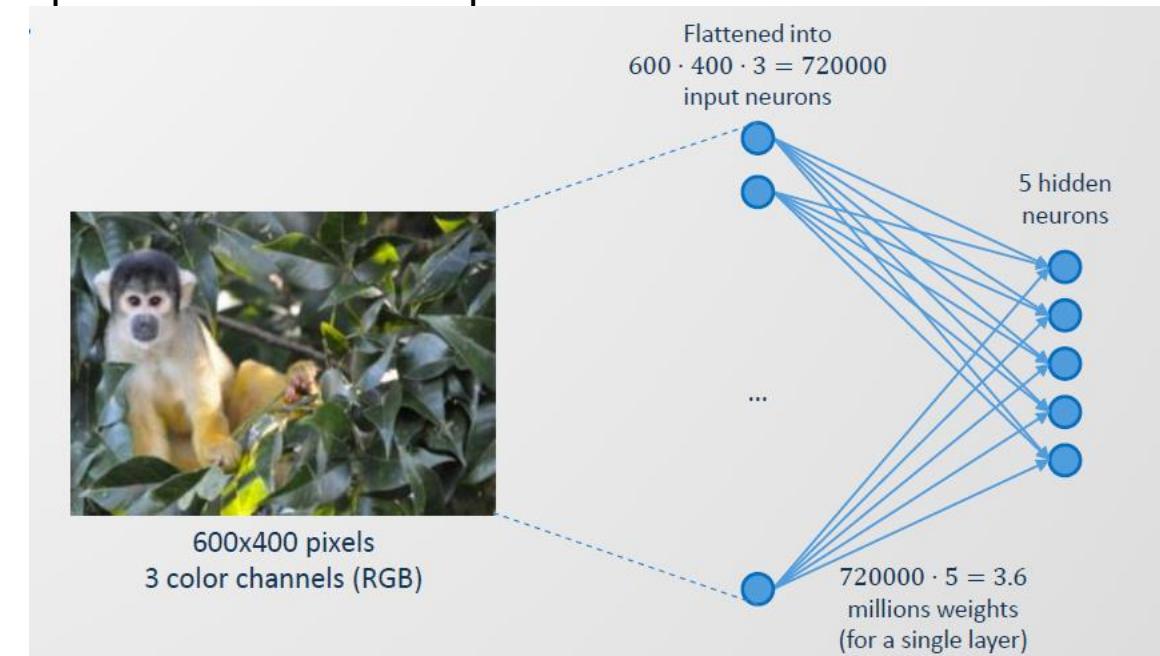
GPU computing :

il training di modelli complessi (profondi e con molti pesi e connessioni) richiede elevate potenze computazionali. La disponibilità di GPU con migliaia di core e GB di memoria interna ha consentito di ridurre drasticamente i tempi di training da mesi a giorni.

La superiorità delle tecniche di deep learning rispetto ad altri approcci si manifesta quando sono disponibili grandi quantità di dati di training.



- Le reti di tipo MLP sono computazionalmente troppo pesanti per essere impiegate **nell'elaborazione di immagini**.
- L'immagine viene appiattita e non si tiene conto della sua struttura 2D, **si prevede un neurone per ogni pixel che è connesso con tutti i neuroni del layer successivo**.
- Poiché ciascun neurone viene connesso a tutti i neuroni del layer successivo, il numero di parametri da stimare sarebbe eccessivo per i casi di interesse pratico..





Le reti MLP **non hanno alcuna invarianza per traslazione.**

Una MLP reagirà in **modo diverso ad un'immagine ed ad una sua versione traslata**, anche se le immagini sono visivamente simili. Ciò è dovuto al fatto che la posizione dei pixel nell'immagine è un fattore importante per determinare l'output della rete.

Le informazioni spaziali più importanti vengono perse quando l'immagine viene appiattita in un MLP. Conoscere le relazioni di vicinanza tra i pixel è importante perché aiuta a definire le caratteristiche di un'immagine.

Le **Convolutional Neural Network (CNN)** sono progettate per essere invarianti alla traslazione e ciò le rende più efficaci nell'elaborazione delle immagini. Le CNN **utilizzano strati convoluzionali** che applicano filtri all'immagine, consentendo alla rete di identificare le caratteristiche indipendentemente dalla loro posizione nell'immagine. Inoltre, vengono utilizzati strati di pooling per ridurre la dimensione dell'output degli strati convoluzionali, aumentando ulteriormente l'invarianza alla traslazione della rete.

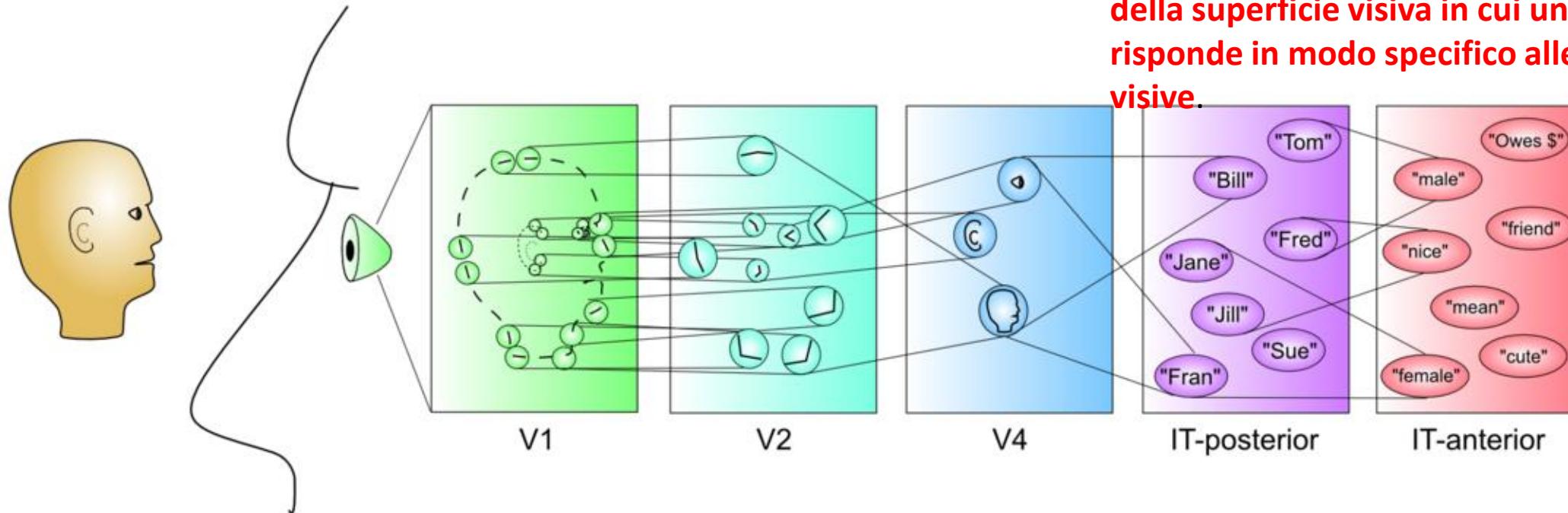


Le reti neurali convoluzionali (CNN) sono reti profonde (deep) ispirate alle ricerche biologiche di Hubel e Wiesel durante lo studio del cervello dei gatti.

Corteccia visiva

- Nel 1965 DH Hubel e TN Wiesel hanno dimostrato che i mammiferi percepiscono visivamente il mondo che li circonda utilizzando un'architettura a strati di neuroni nel cervello.
- La struttura della **corteccia visiva è a strati. Man mano che le informazioni passano dai nostri occhi al cervello, si formano rappresentazioni di ordine sempre più alto.**

VISIONE UMANA



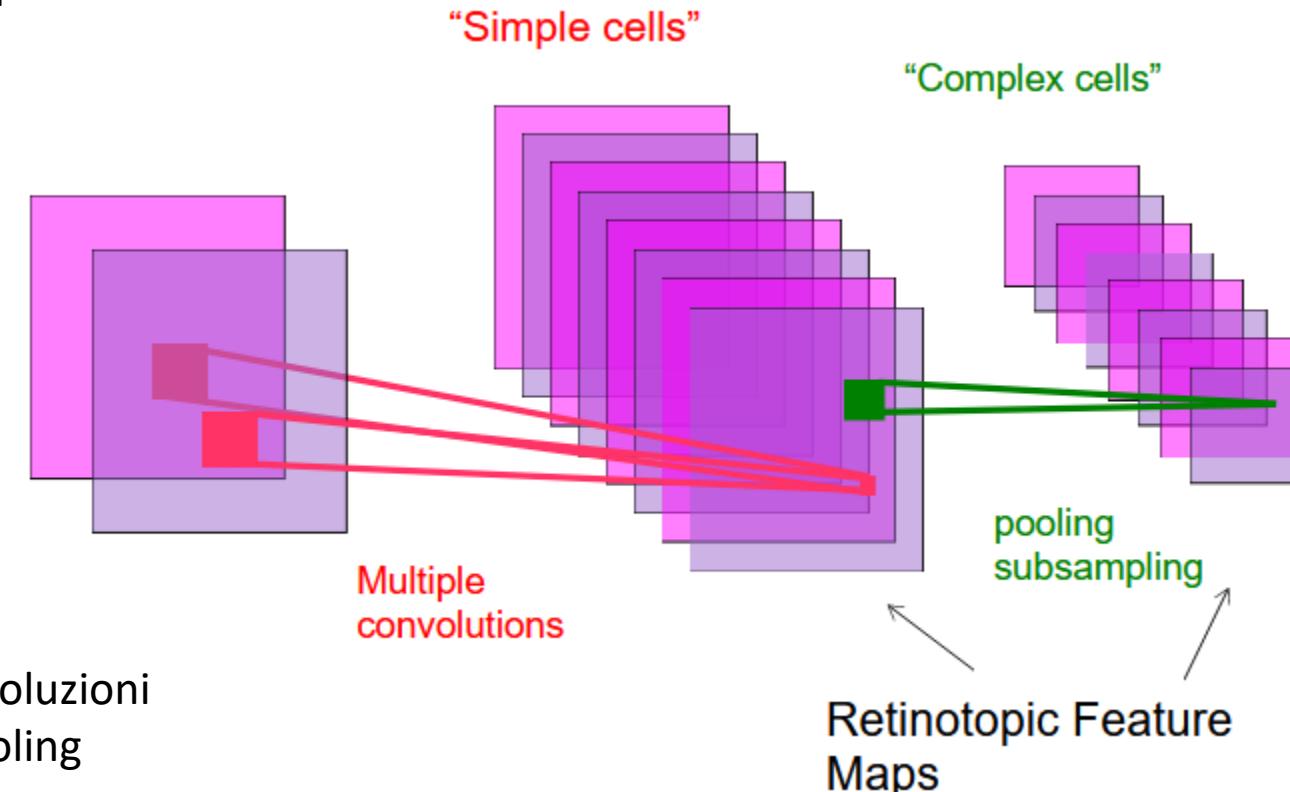
- Il sistema visivo umano elabora le caratteristiche dell'oggetto in un **approccio feed-forward e gerarchico**.
- **La gerarchia inizia dalla Corteccia Visiva Primaria (V1)**, caratterizzata da neuroni con campi recettivi piccoli e specializzati, elabora le informazioni visive di base, bordi e linee.
- Queste informazioni vengono poi elaborate dalle **Areae visive (V2 e V4)**, specializzate nella percezione di informazioni visive più complesse, come forme e oggetti (e sono caratterizzate da neuroni con campi recettivi più estesi).
- Infine, le informazioni visive vengono elaborate dalla **corteccia inferotemporale (IT)**, che è specializzata nella percezione di oggetti complessi, come il riconoscimento facciale (campi recettivi più ampi e completi)

Il **campo recettivo** di un neurone visivo è l'**area della superficie visiva in cui un dato neurone risponde in modo specifico alle informazioni visive**.

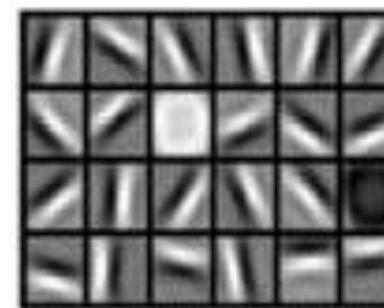
[Hubel & Wiesel 1962]:

Due tipologie di neuroni nella corteccia visiva del gatto

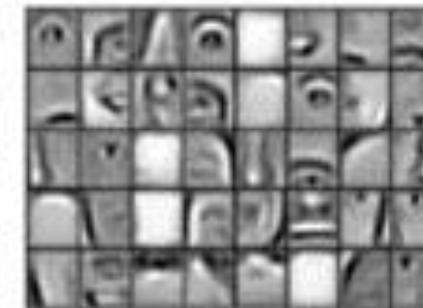
- **celle semplici** : specializzate nella rilevazione di caratteristiche locali dell'input visivo (feature extractor),
- **celle complesse** : specializzate nell'integrazione (pooling) delle informazioni provenienti da diverse posizioni retinotopiche per formare una rappresentazione globale dell'input visivo, preservando le caratteristiche invarianti per posizione.



Celle semplici → Convoluzioni
Celle complesse -> Pooling



Low-level features



Mid-level features



High-level features

- Come il sistema visivo umano, le CNN sono composte da una serie di strati o livelli di elaborazione, ognuno dei quali svolge una particolare funzione nell'elaborazione dell'immagine di input.
- I primi strati di una CNN sono specializzati nella rilevazione di bordi e linee, proprio come i neuroni visivi nella corteccia visiva primaria (V1).
- Strati successivi delle CNN sono invece specializzati nella percezione di caratteristiche visive più complesse, come forme e oggetti, in modo simile alle aree visive V2 e V4.
- Infine, gli strati più profondi delle CNN possono essere considerati analoghi alla corteccia inferotemporale (IT) nella gerarchia visiva, poiché sono in grado di elaborare informazioni visive complesse e riconoscere oggetti e classi di oggetti.

Architettura di una CNN

Convolutional Neural Networks CNN introdotte da LeCun et al a partire dal 1998

Le principali differenze rispetto a MLP usano gli **operatori convoluzionali** per estrarre le features

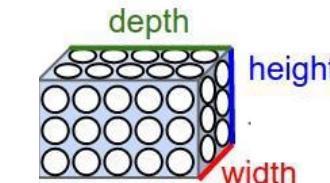
- **Processing locale**: i neuroni sono connessi solo localmente ai neuroni del livello precedente. Ogni neurone esegue quindi un'elaborazione locale. Forte riduzione numero di connessioni.
- **Pesi condivisi**: i pesi sono condivisi a gruppi. Neuroni diversi dello stesso livello eseguono lo stesso tipo di elaborazione su porzioni diverse dell'input. Forte riduzione numero di pesi
- **Alternanza livelli di feature extraction e pooling**

I neuroni in ogni livello sono organizzati secondo una piccola griglia 3d,

Width

Height

depth

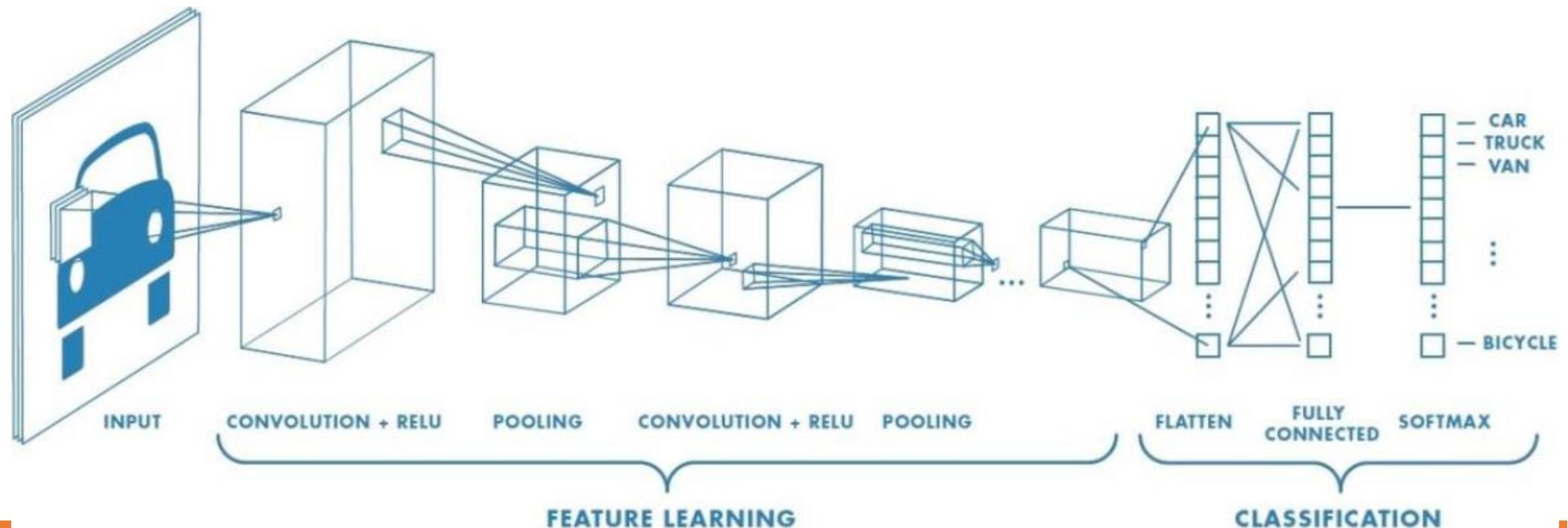


che ha dimensioni piccole se confrontate con l'immagine, in genere Larghezza ed altezza variano tra 3,5,7,11

Architettura di una CNN

Una CNN è una combinazione di due parti fondamentali

- La parte convoluzionale consiste di strati convoluzionali seguite da funzioni di attivazione non lineare tipo(RELU) e di pooling. Questa parte costituisce il componente essenziale dell'estrazione di feature
- La parte fully-connected consiste in un'architettura di rete neurale completamente connessa. Questa parte esegue il compito di classificazione in base all'input dalla parte convoluzionale





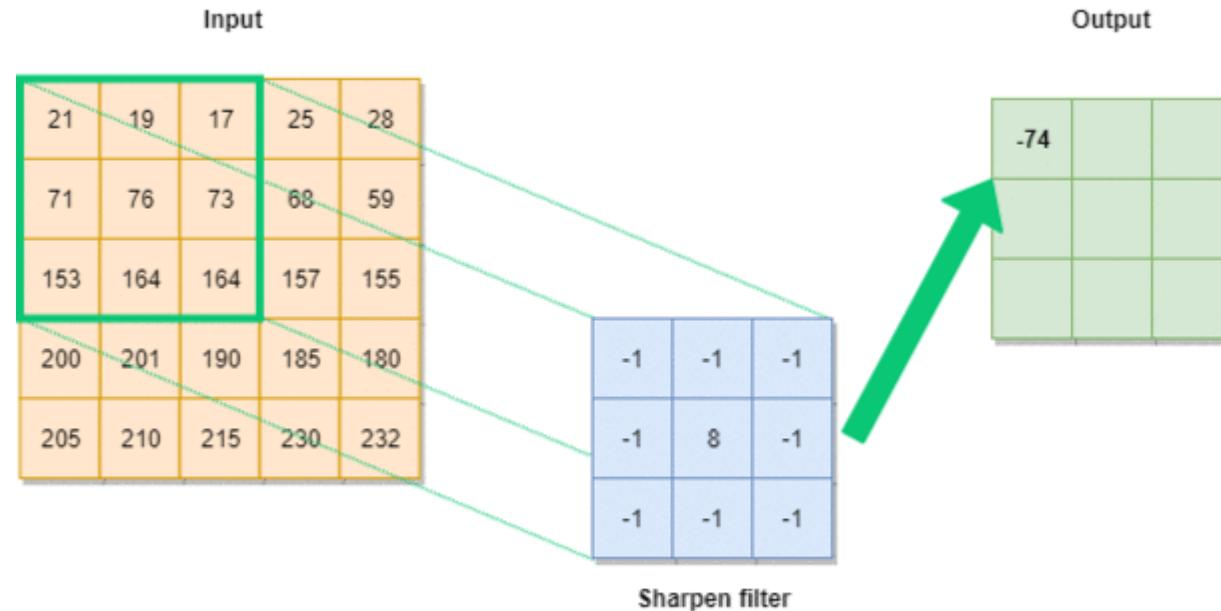
- Tre tipi principali di livelli vengono utilizzati per costruire architetture CNN
- **Strato convoluzionale:** contiene una serie di filtri apprendibili. La larghezza e l'altezza dei filtri sono più piccoli di quelli del volume di ingresso. Il filtro scorre attraverso l'ingresso e il prodotto scalare tra l'input e il filtro vengono calcolati in ogni posizione spaziale. E' seguito da una funzione di attivazione non lineare (tipo RELU)
- **Strato di pooling** riduce il numero di parametri e preserva l'invarianza per traslazione
- I neuroni dello **strato fully connected** in uno strato completamente connesso hanno connessioni complete con tutte le attivazioni nel livello precedente, come nelle ANNs tradizionali



Convoluzione

- La convoluzione è una delle più importanti operazioni di **image processing** attraverso la quale si applicano filtri digitali, per estrarre feature dalle immagini.
- Un filtro digitale h (una piccola maschera 2D di pesi, di dimensione $F \times F$) viene fatto scorrere su ogni pixel (x, y) di un'immagine di input, f , *di dimensione* $m \times n$) per ogni posizione viene generato un valore di output $g(x, y)$, eseguendo il prodotto scalare tra la maschera e la porzione dell'input coperta (entrambi trattati come vettori). L'output $g(x, y)$ prende il nome di *features map*.

$$g(x, y) = (f * h)(x, y) = \sum_{i=0}^{F-1} \sum_{j=0}^{F-1} f\left(x - \left\lceil \frac{F}{2} \right\rceil + i, y - \left\lceil \frac{F}{2} \right\rceil + j\right) h(i, j)$$





Gestione dei bordi dell'immagine durante la convoluzione.

Quando si applica la convoluzione su un'immagine, ci si trova spesso di fronte al problema dei bordi dell'immagine. Infatti, **se si applica la convoluzione direttamente sull'immagine, i pixel ai bordi dell'immagine vengono trattati in modo diverso rispetto ai pixel al centro dell'immagine, poiché il filtro non può essere centrato su di essi.**

Ci sono diverse strategie per **gestire i bordi dell'immagine durante la convoluzione**.

Una soluzione comune è quella di aggiungere dei bordi di padding all'immagine prima di applicare la convoluzione.

Il **padding** consiste nell'aggiungere dei pixel intorno ai bordi dell'immagine, in modo da creare una cornice di pixel che permette di applicare la convoluzione anche sui bordi dell'immagine.

Esistono diversi tipi di padding, ad esempio:

Padding con zeri (zero-padding): si aggiungono dei pixel con valore zero intorno ai bordi dell'immagine.

Padding a specchio (mirror-padding): si copiano i pixel lungo i bordi dell'immagine, in modo da creare una specie di riflesso rispetto ai bordi.



Zero padding

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114			



mirror-padding

input

7	6	5	5	6	7
6	4	3	3	4	6
5	3	2	2	3	5
5	3	2	2	3	5
6	4	3	3	4	6
7	6	5	5	6	7

output

0	-1	0
-1	5	-1
0	-1	0
0	-1	0



Kernel diversi individuano features differenti

Input image



Convolution
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



Esempio di kernel che evidenzia i contorni di una immagine.



*

$$\begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array}$$

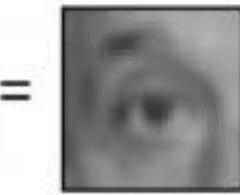


Esempio di kernel che produce un effetto embossed.



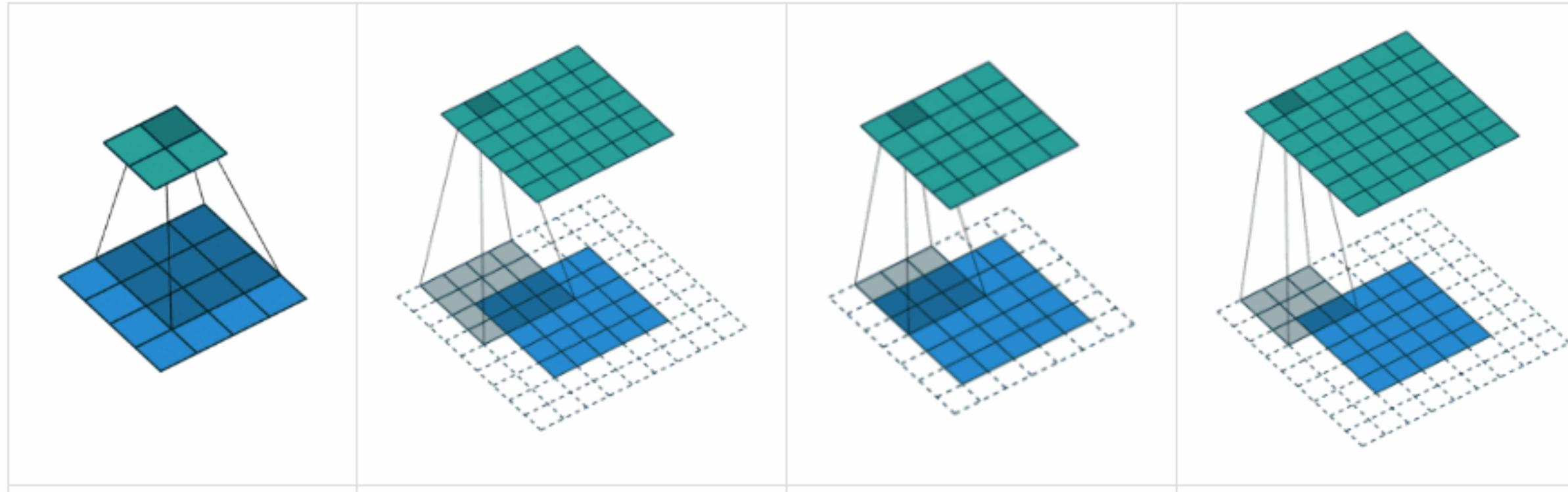
*

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

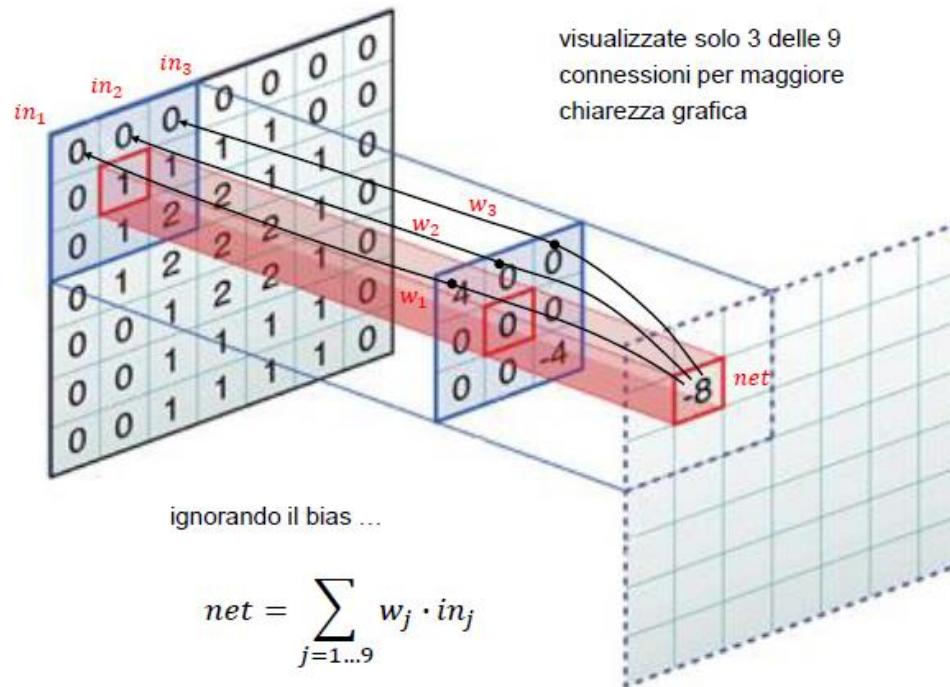


Original

Esempi di kernel che producono un effetto blur



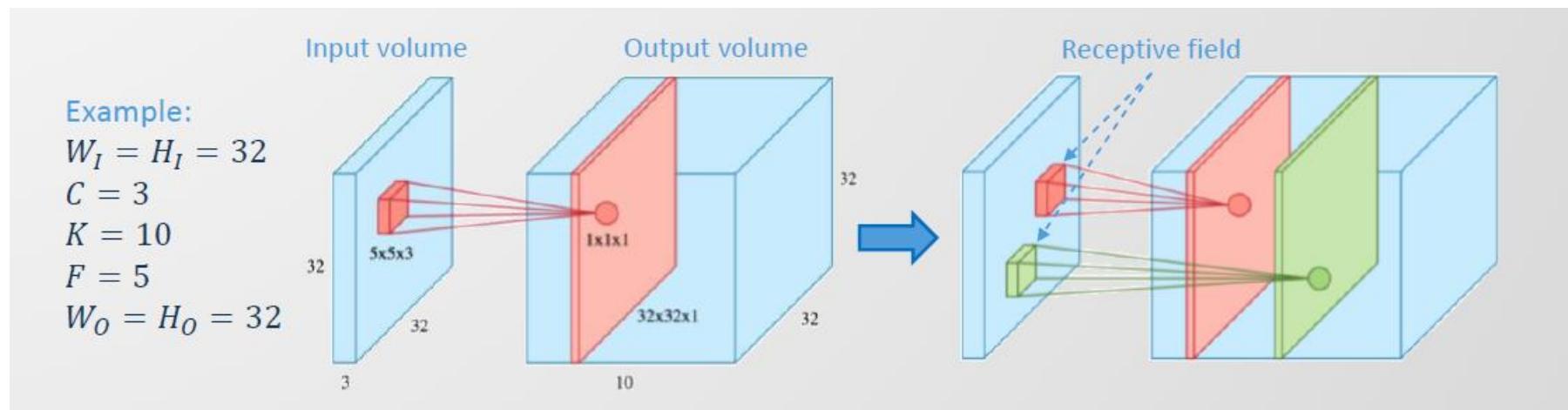
Le reti neurali convoluzionali hanno i layer e i neuroni ad essi appartenenti, organizzati secondo larghezza, altezza e profondità (volume 3D) e non completamente connessi e con valori sinaptici condivisi, così da permettere di ridurre i parametri (pesi e bias) in gioco.



In una CNN, l'obiettivo dell'operazione di convoluzione è quello di estrarre caratteristiche dall'immagine di input mantenendo la relazione spaziale tra i pixel.

Ogni layer convoluzionale contiene un insieme di filtri (o kernel convoluzionali).

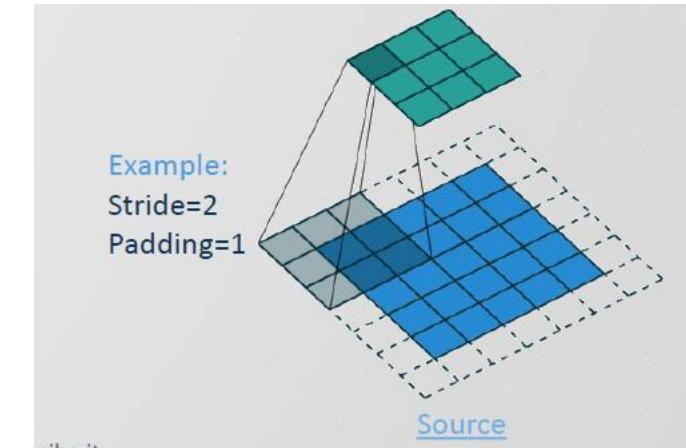
Data una volume di input tridimensionale (ad esempio l'immagine di input) di dimensioni $W_I \times H_I \times C$ e un insieme di K kernel di dimensioni $F \times F \times C$, l'output di un layer convoluzionale è un volume tridimensionale di dimensioni $W_O \times H_O \times K$ composto da K mappe di features di dimensioni $W_O \times H_O$

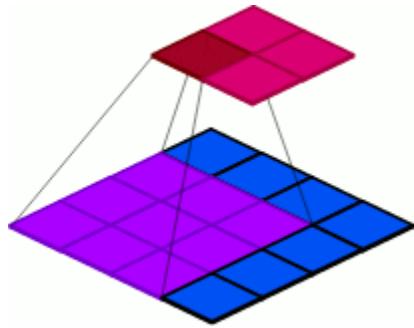


La dimensione del volume di output di una convoluzione dipende da:

- **Profondità:** corrisponde al numero di filtri K che desideriamo utilizzare, ognuno dei quali impara a cercare qualcosa di diverso (ad esempio bordi orientati o blocchi di colore) nell'input.
- **Stride:** è la quantità di movimento tra l'applicazione del filtro al volume di input. Riduce la dimensione spaziale del volume di output e di conseguenza il numero di connessioni.
- **Padding:** aggiunge un bordo al volume di input per ottenere una dimensione spaziale specifica. Ci consente di controllare la dimensione spaziale del volume di output.

$$W_o = \frac{(W_{in} - F + 2 \cdot Padding)}{Stride} + 1$$





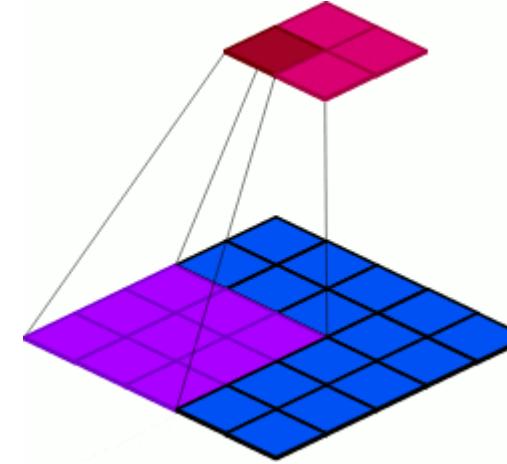
Dimensione del filtro (F) 3×3

Stride (S): 1

Padding: 0

Dimensione input: ($W_i \times H_i$): 4×4

Dimensione output ($W_0 \times H_0$): 2×2



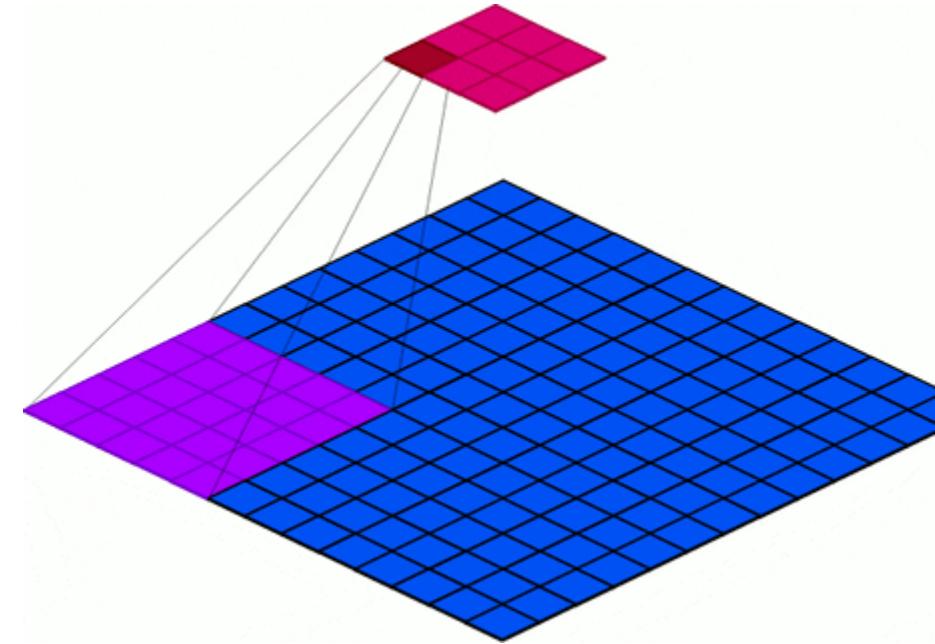
Dimensione del filtro (F) 3×3

Stride (S): 2

Padding (P): 0

Dimensione input ($W_i \times H_i$): 5×5

Dimensione output($W_0 \times H_0$): 2×2



Dimensione del filtro (F) 5×5

Stride: 4,

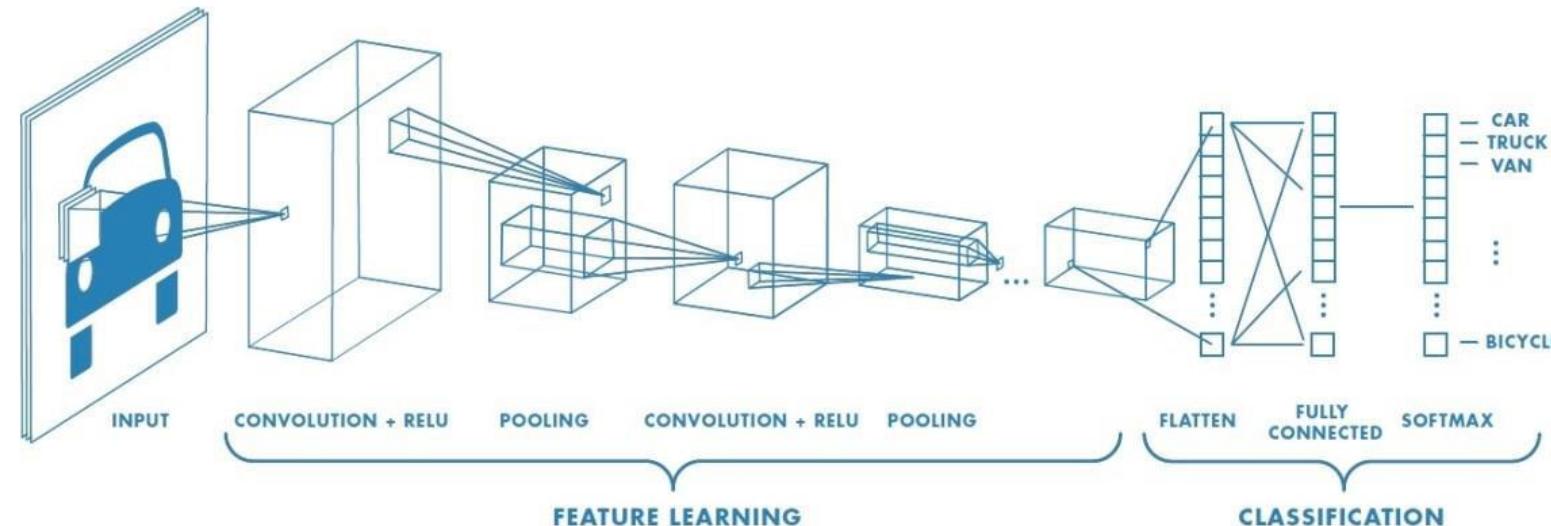
padding (P): 0,

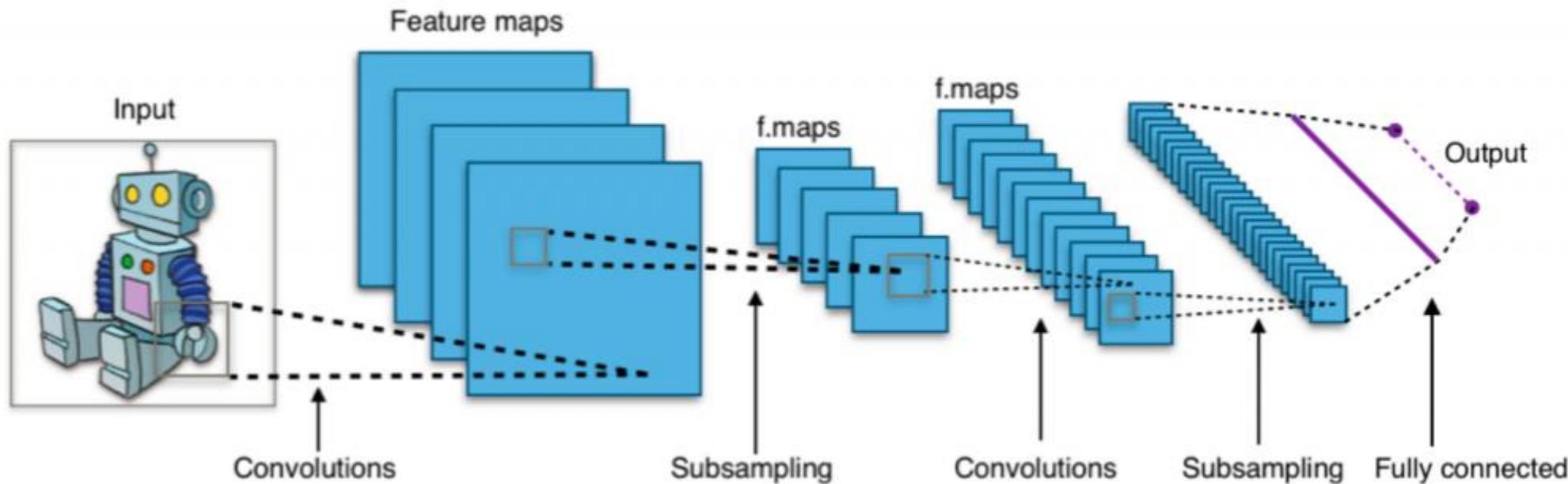
Dimensione input($W_i \times H_i$): 13×13 ,

Dimensione output($W_0 \times H_0$): 3×3

Architettura di una CNN

- L'architettura di una rete neurale convoluzionale differisce da una comune rete neurale per il suo strato nascosto che non è composto da soli layer fully connected (neuroni interconnessi ad ogni altro neurone dei livelli precedenti e successivi) ma anche da **livelli parzialmente connessi**.
- La struttura tipica di una rete ConvNet è quella che osserviamo nella figura







- **L'input layer** è lo strato in cui i dati vengono forniti alla rete. Questo livello è strutturato ad hoc sul dato in ingresso secondo le sue features specifiche, ad esempio in caso di un'immagine il livello di input è rappresentato come un insieme di pixel (e.g. h 224x w224x ch3).
- Il **Conv Layer** ha lo scopo di riconoscere pattern specifici (quadrati, angoli, texture, curvature) in modo efficiente ed accurato. Una CNN può essere formata da più livelli di questo genere e ognuno di questi è in grado di riconoscere un pattern specifico differente.
- Aumentando il numero di *Conv Layer* è possibile raggiungere il riconoscimento di caratteristiche sempre più complesse.
- Lo strato successivo ad ogni convolutional layer è **il Rectified Linear Units layer** (abbreviato ReLu) che, come la funzione di attivazione lineare da cui prende il nome, ha l'obiettivo di rendere trascurabili i valori negativi derivanti dai layer precedenti.
- Altro livello estremamente importante posto sempre dopo il conv layer e il ReLu layer è lo strato di **Pool**. Questo livello rende in grado di **individuare gli elementi principali e più caratteristici di un'immagine e di ridurne la dimensionalità eliminando ciò che è superfluo per la rete per svolgere una corretta elaborazione e classificazione** (successivamente verranno approfondite le modalità con cui questi livelli operano). **L'immagine in output da un livello di pool sarà notevolmente semplificata e più grezza rispetto all'originale.**



Posta alla fine della rete neurale convoluzionale, c'è una rete fully connected utilizzata per la classificazione delle immagini, quindi della generazione dell'output di una CNN.

Questo strato di livelli completamente connessi riceve in input l'immagine manipolata dai precedenti layer e produce un vettore di N dimensione dove N corrisponde al numero di classi in cui si vuole classificare il volume in input.



Pooling

- L'operatore di Pooling è costituito da una finestra di forma fissa che scorre su tutte le regioni nell'input e calcola un singolo output per ogni posizione
- A differenza dei livelli convoluzionali, il layer di pooling non contiene parametri addestrabili
- Non è un livello addestrabile ma deterministico che in genere calcola il valore massimo o medio degli elementi nella finestra di pooling

Max Pooling

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2
pool size

100	184
12	45

Average Pooling

31	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2
pool size

36	80
12	15



Convoluzione (Padding 0)

1	-1	-1
-1	1	-1
-1	-1	1

Filtro 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Prodotto scalare



Immagine 6×6



Convoluzione

1	-1	-1
-1	1	-1
-1	-1	1

Filtro 1

Se stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

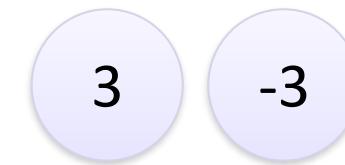


Immagine 6 x 6



Convoluzione

stride=1

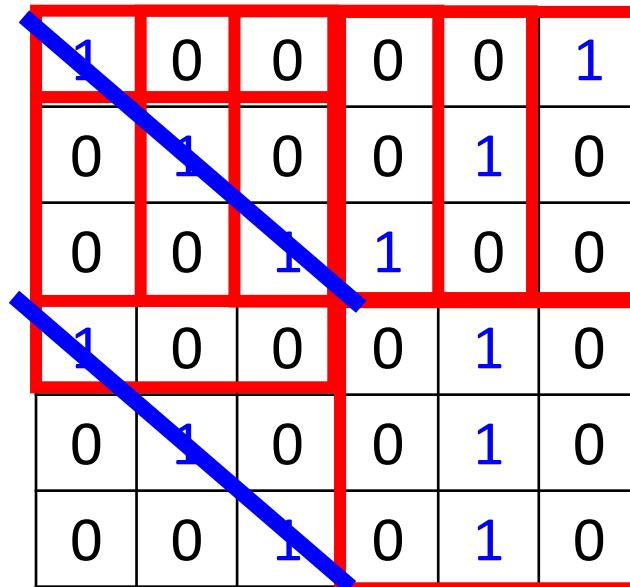
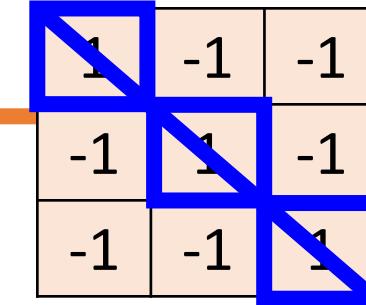
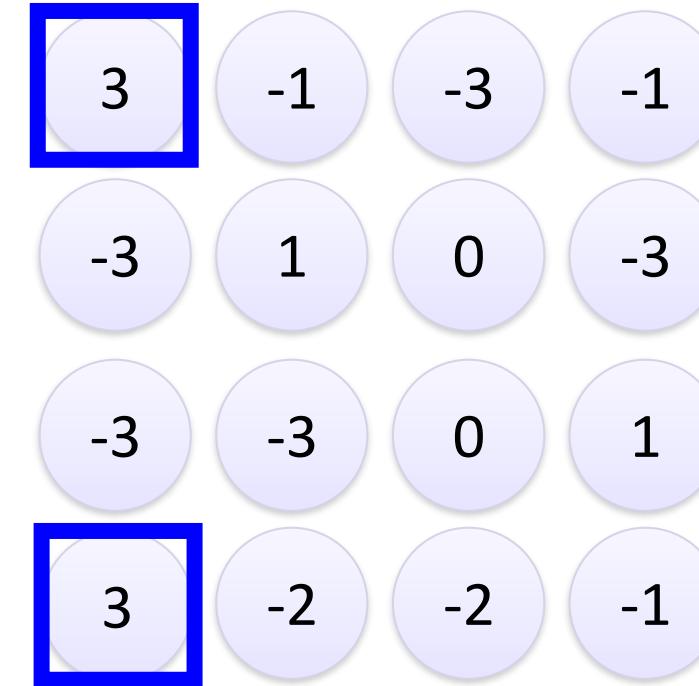


Immagine 6×6



Filtro 1





Convoluzione

stride=1

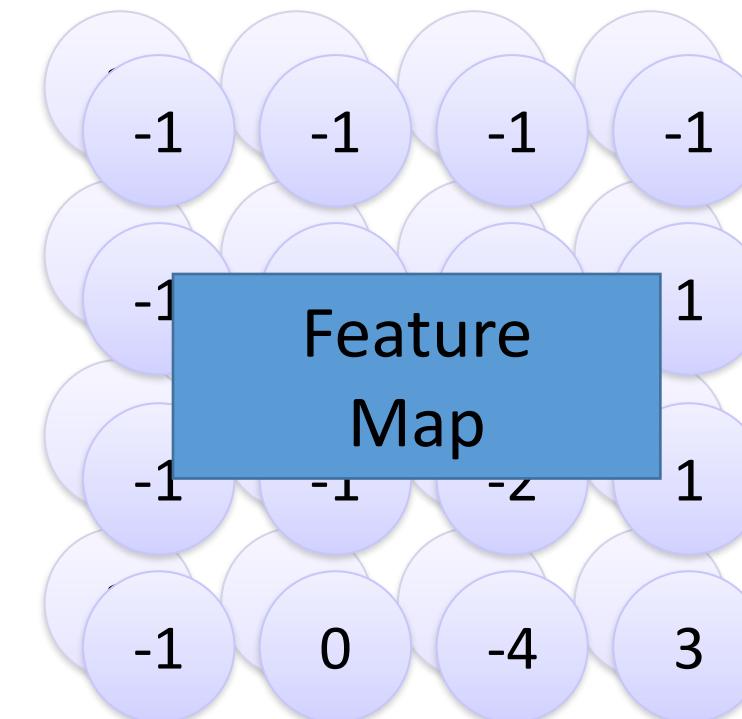
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

Filtro 2

Ripetere per ogni filtro



Due immagini 4 x 4

Formano una matrice 4 x 4 x 2

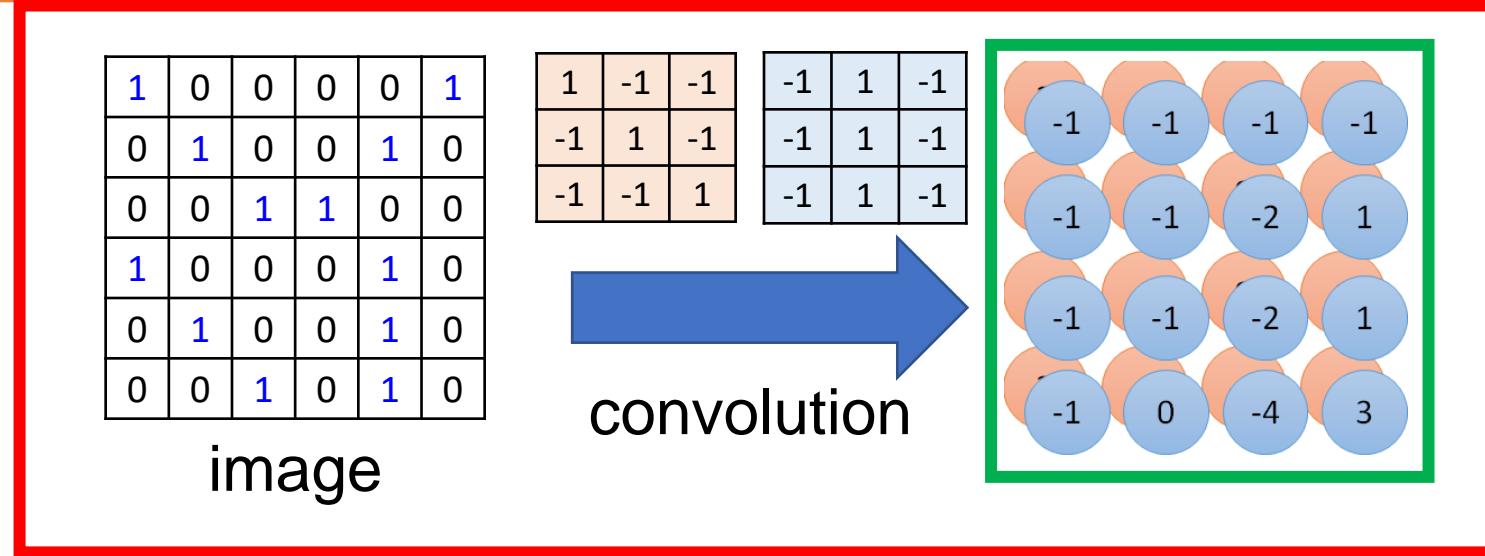




Nelle CNN i neuroni in uno strato nascosto sono collegati solo a una piccola regione dello strato precedente (chiamato campo recettivo locale)
La profondità di ogni feature map corrisponde al numero di filtri convoluzionali utilizzati in ogni layer

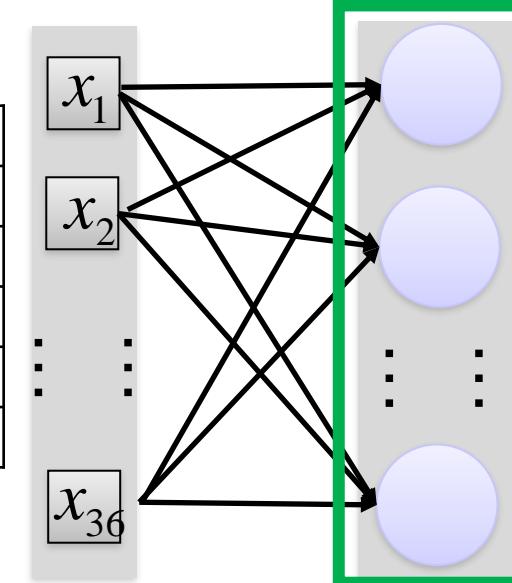


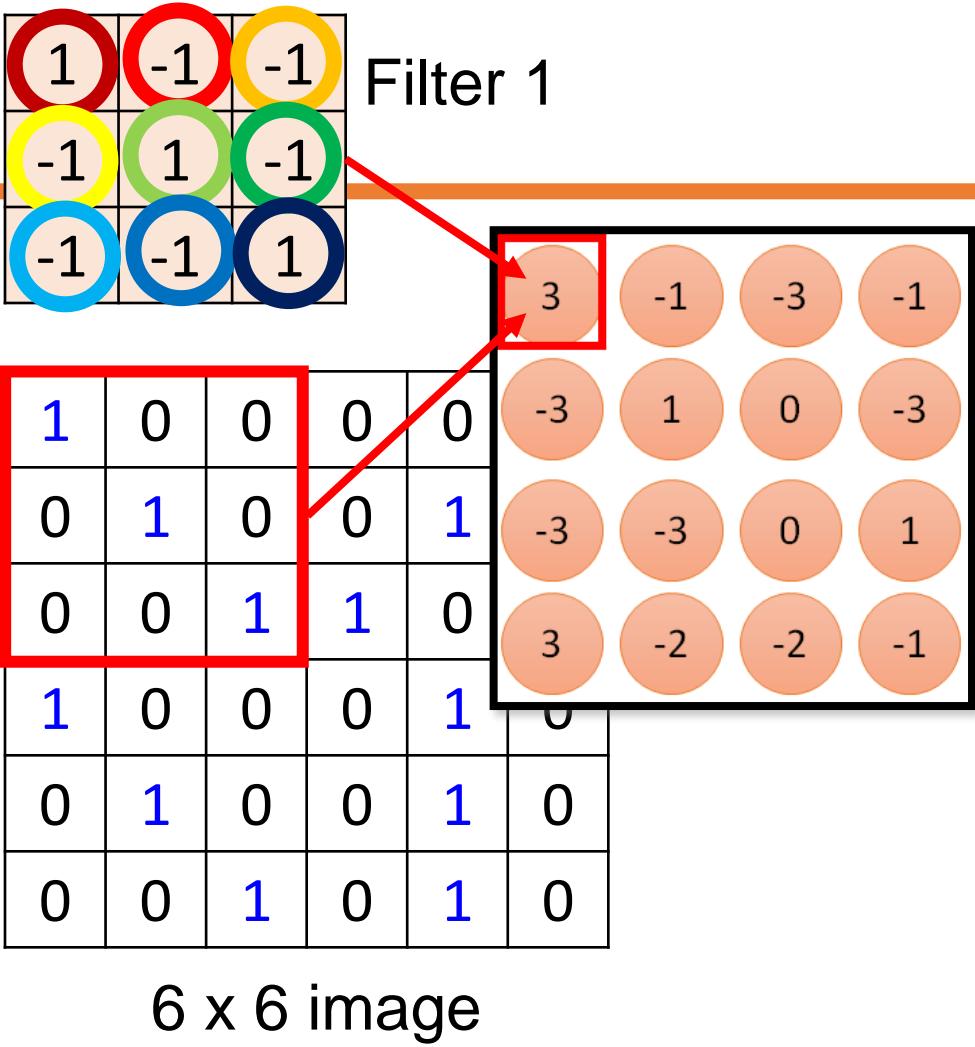
Convolution v.s. Fully Connected



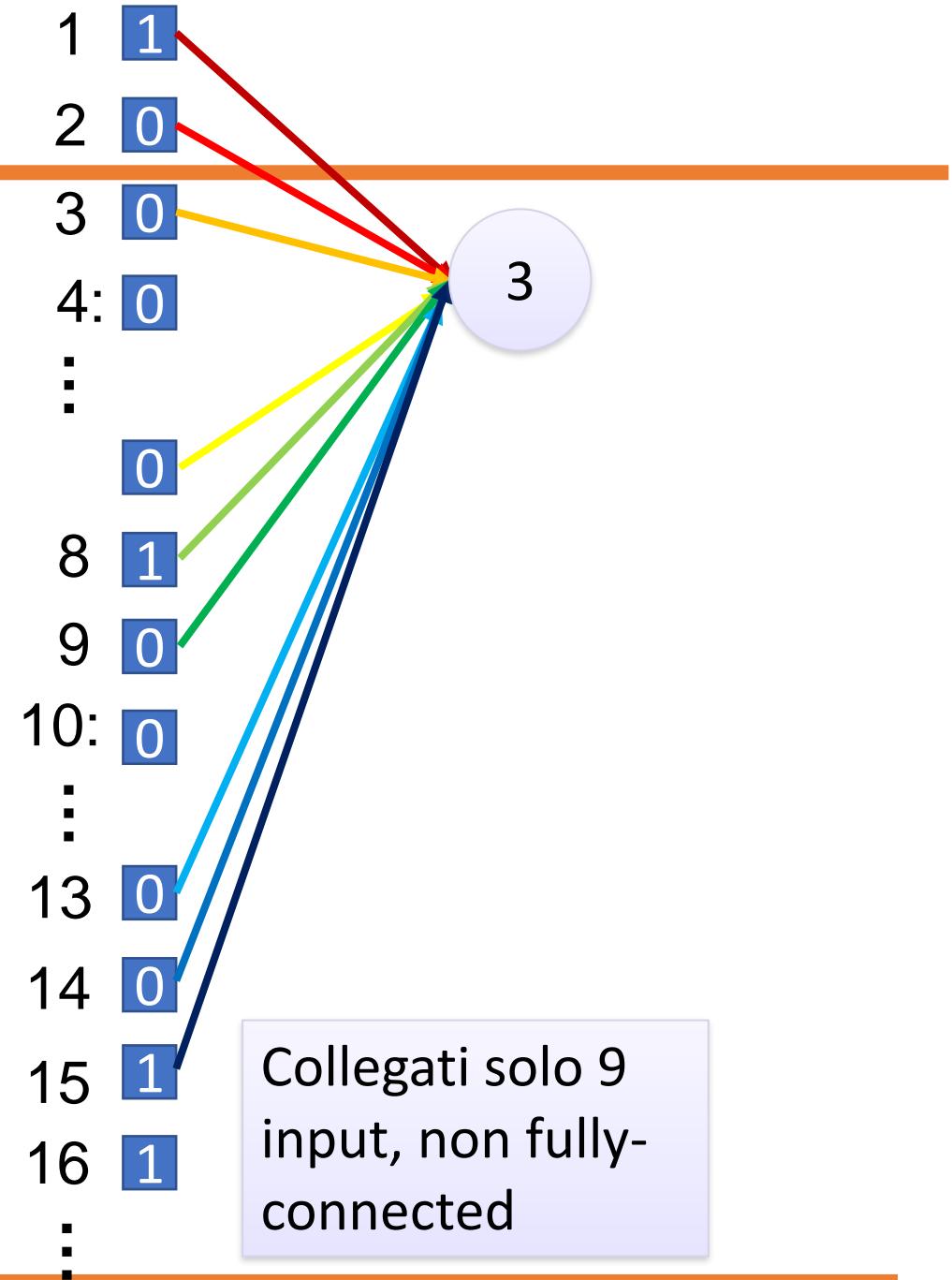
Fully-
connected

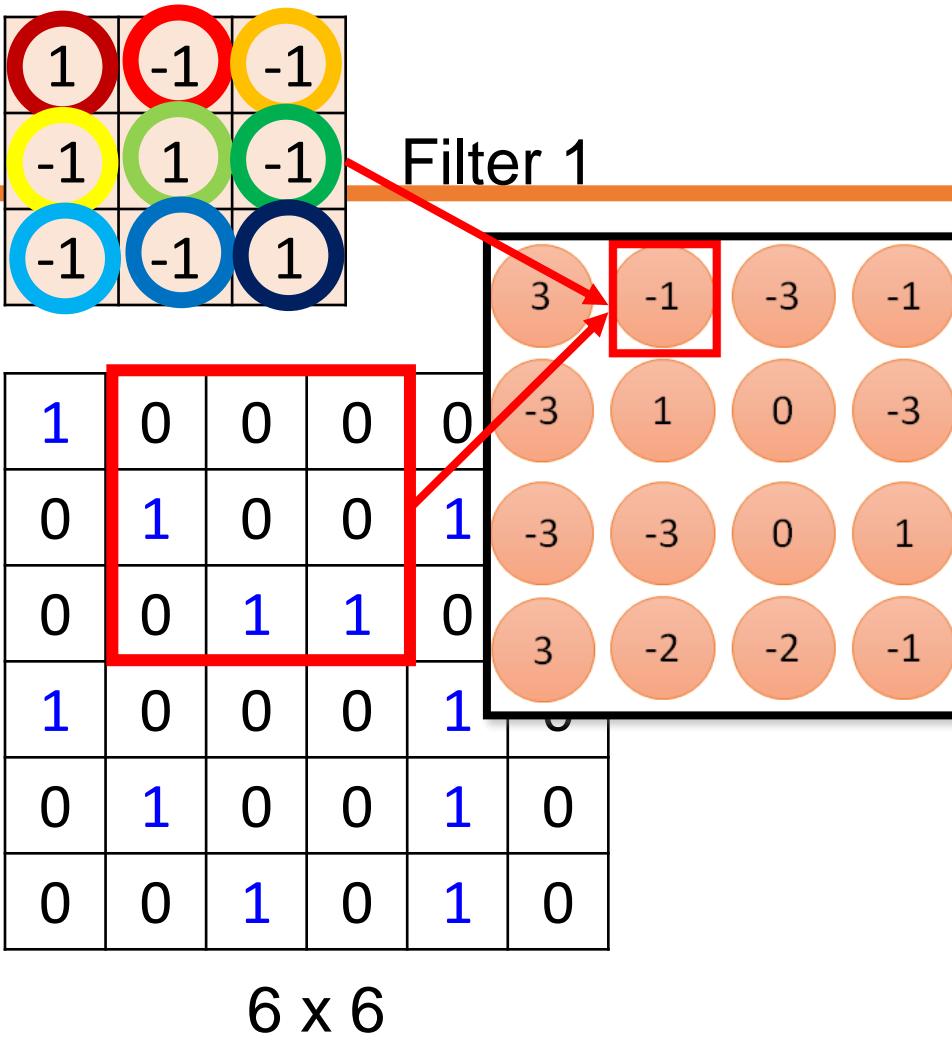
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0



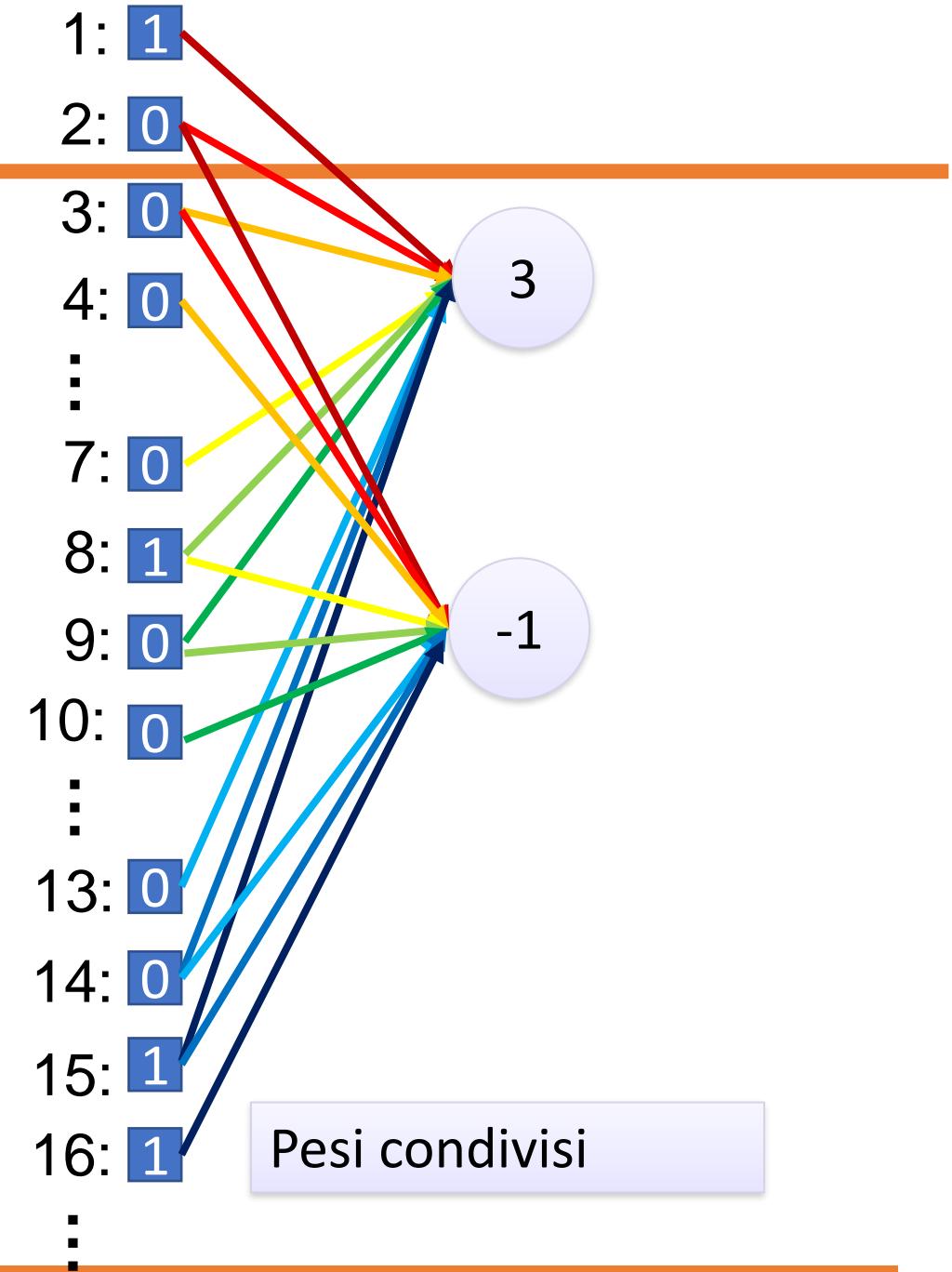


Meno parametri!



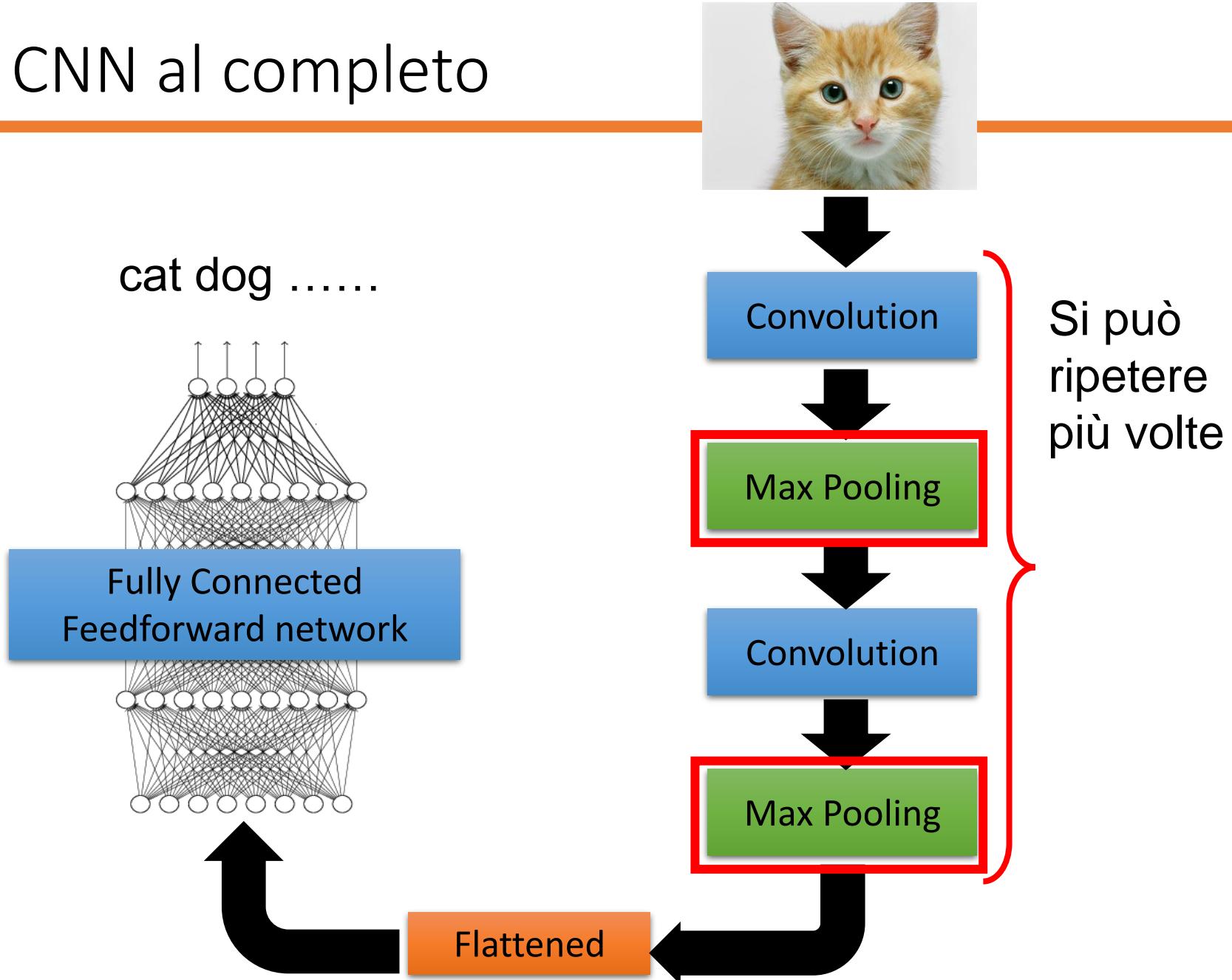


Numero di parametri
inferiore
Ancora meno parametri





La CNN al completo





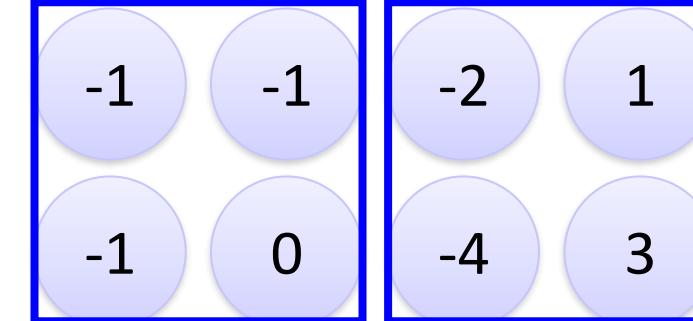
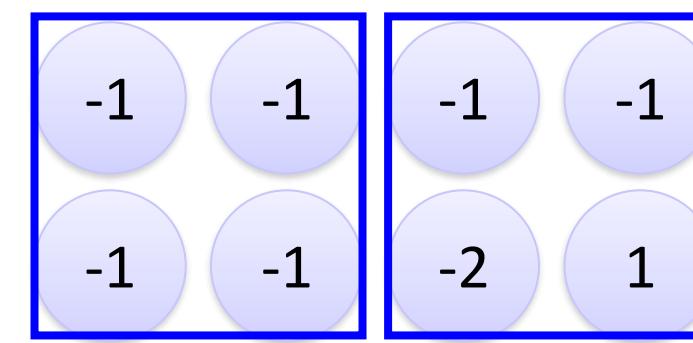
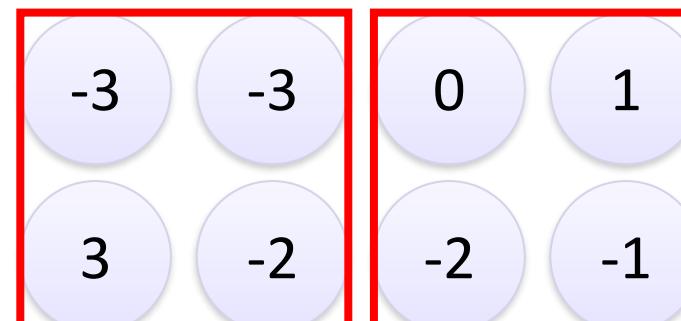
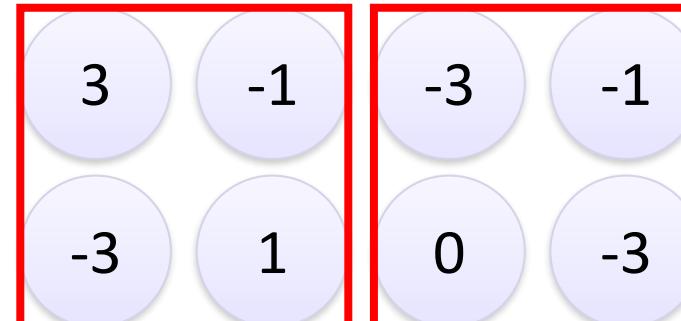
Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Una CNN comprime una rete completamente connessa in due modi:

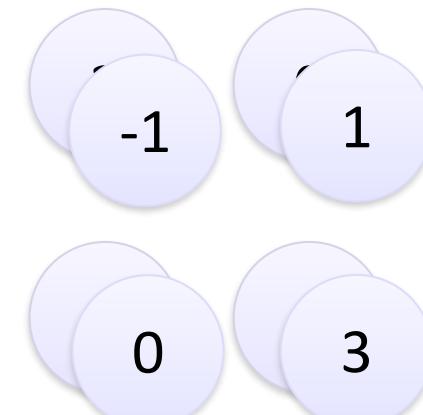
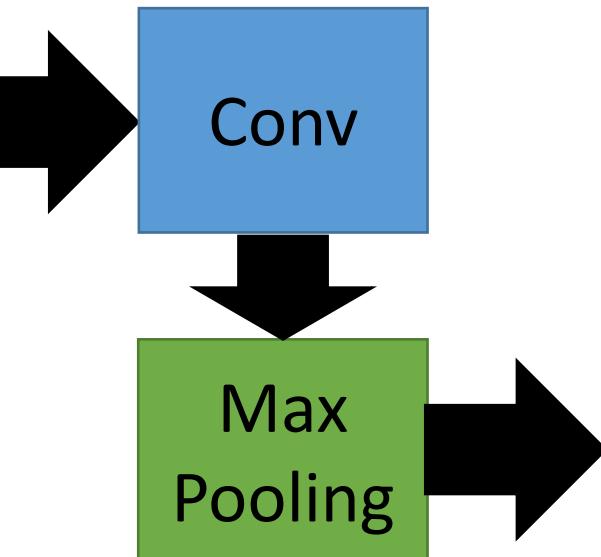
- Riducendo il numero di connessioni
 - Pesi condivisi sugli edge
 - Max pooling reduce ulteriormente la complessità.
-



Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

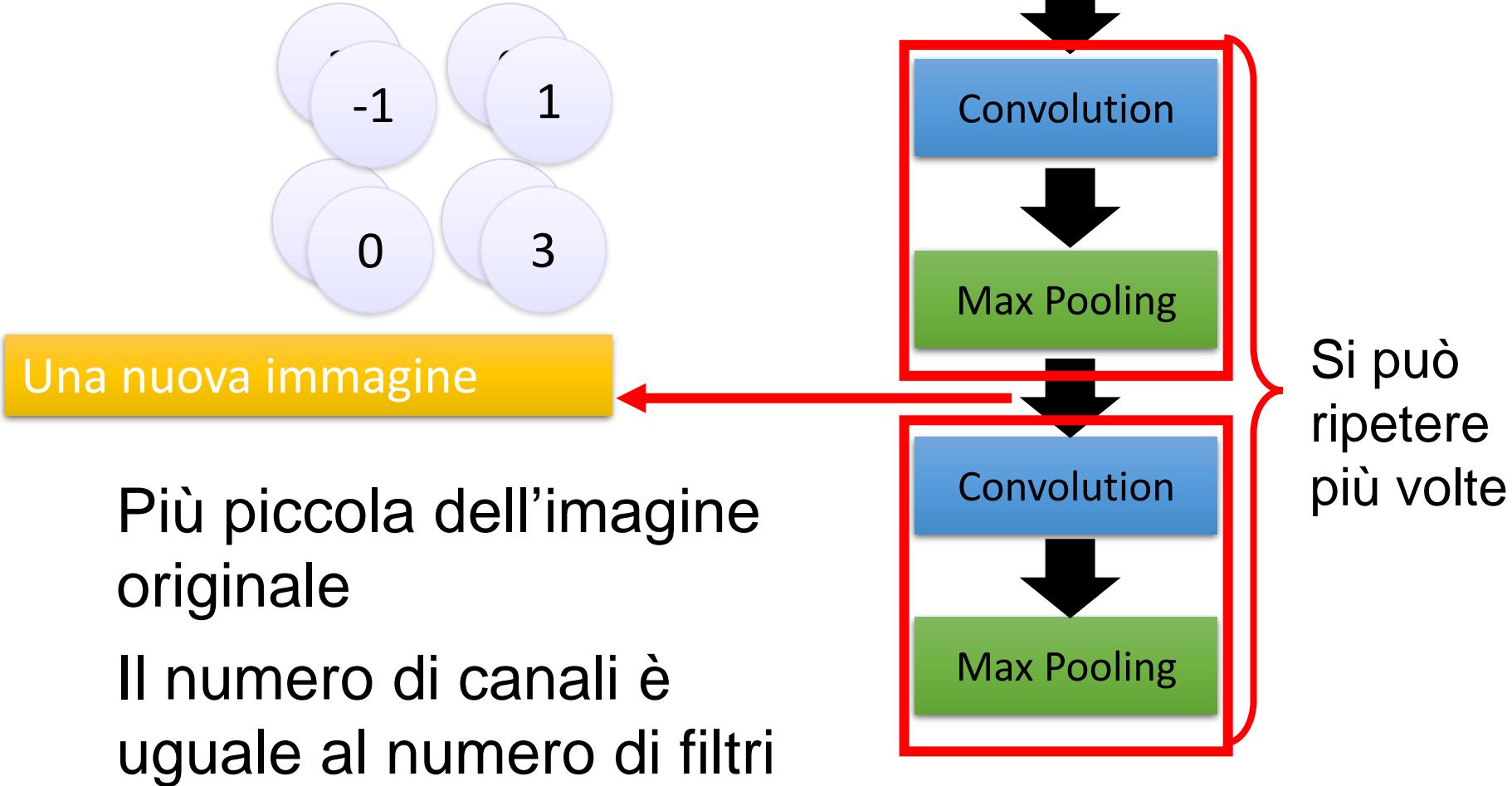
6×6



2×2



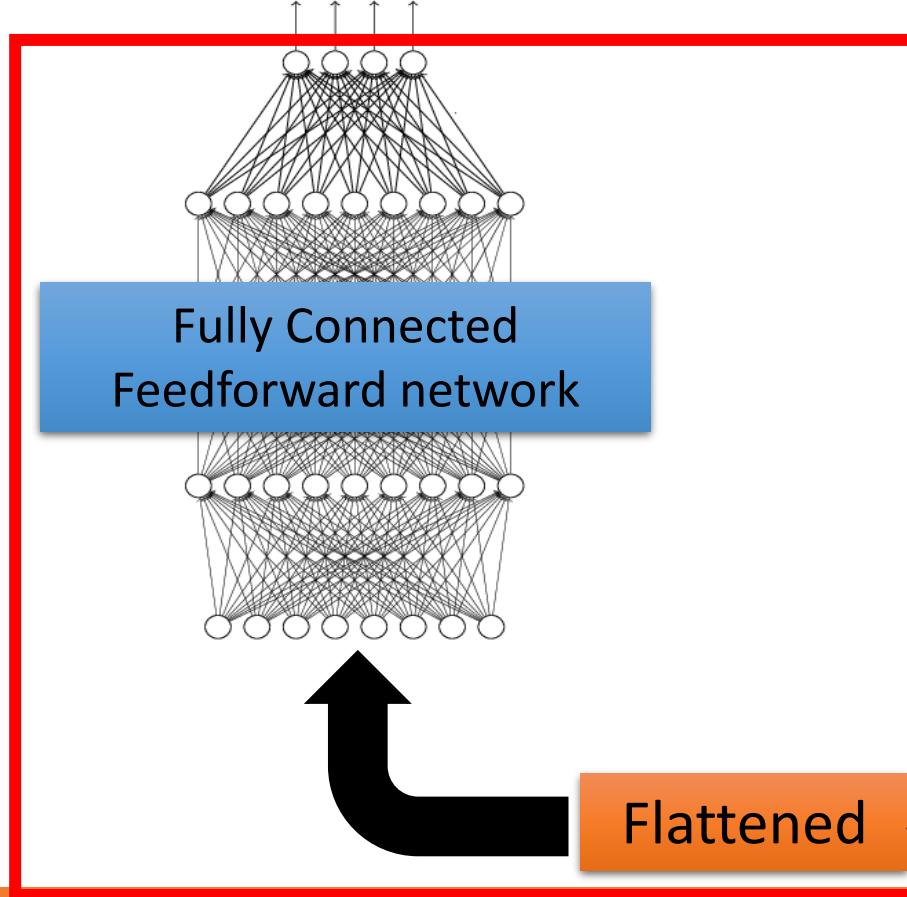
La CNN al completo





La CNN al completo

cat dog



Convolution



Max Pooling



Convolution



Max Pooling

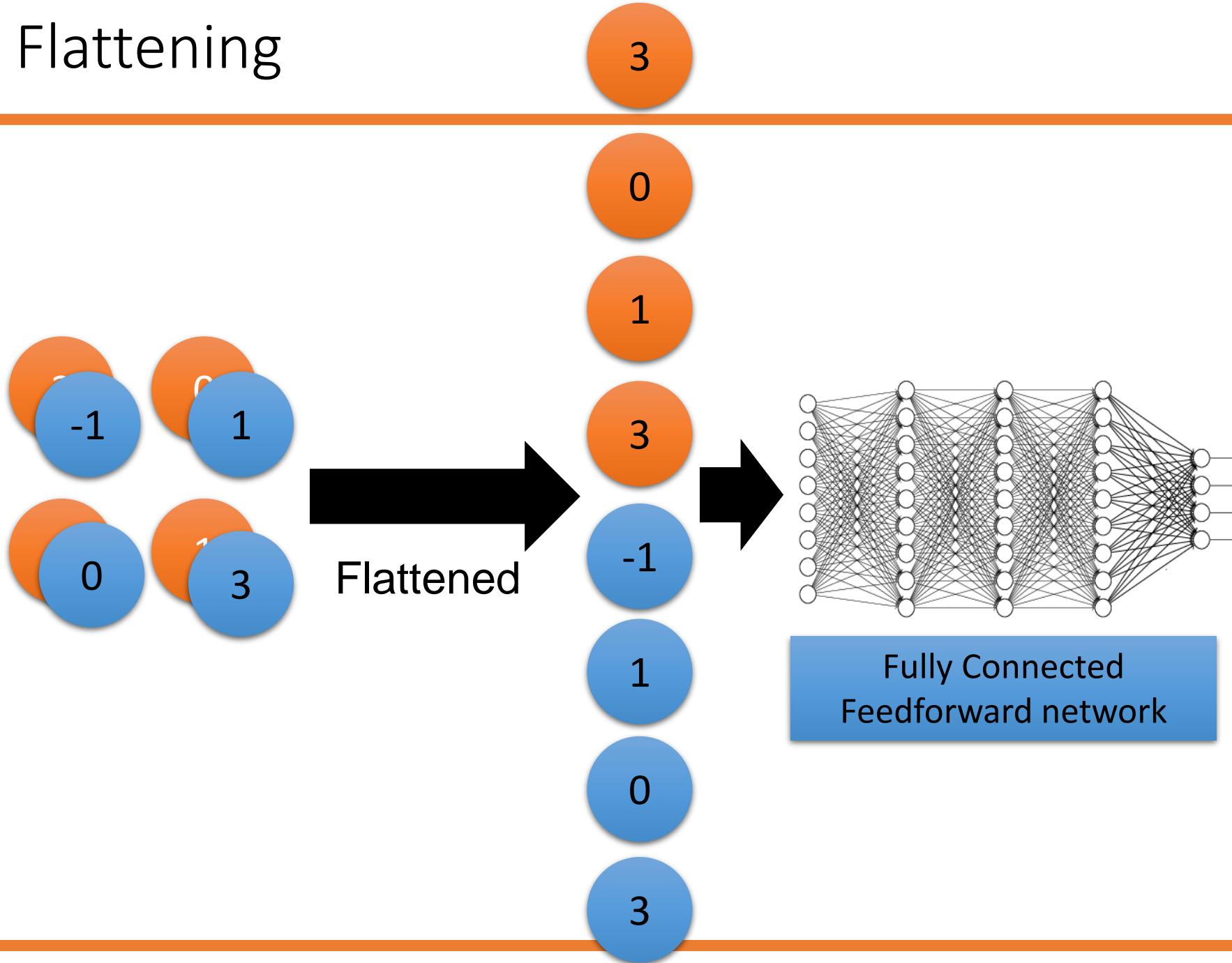
Una nuova immagine

Una nuova immagine

Flattened

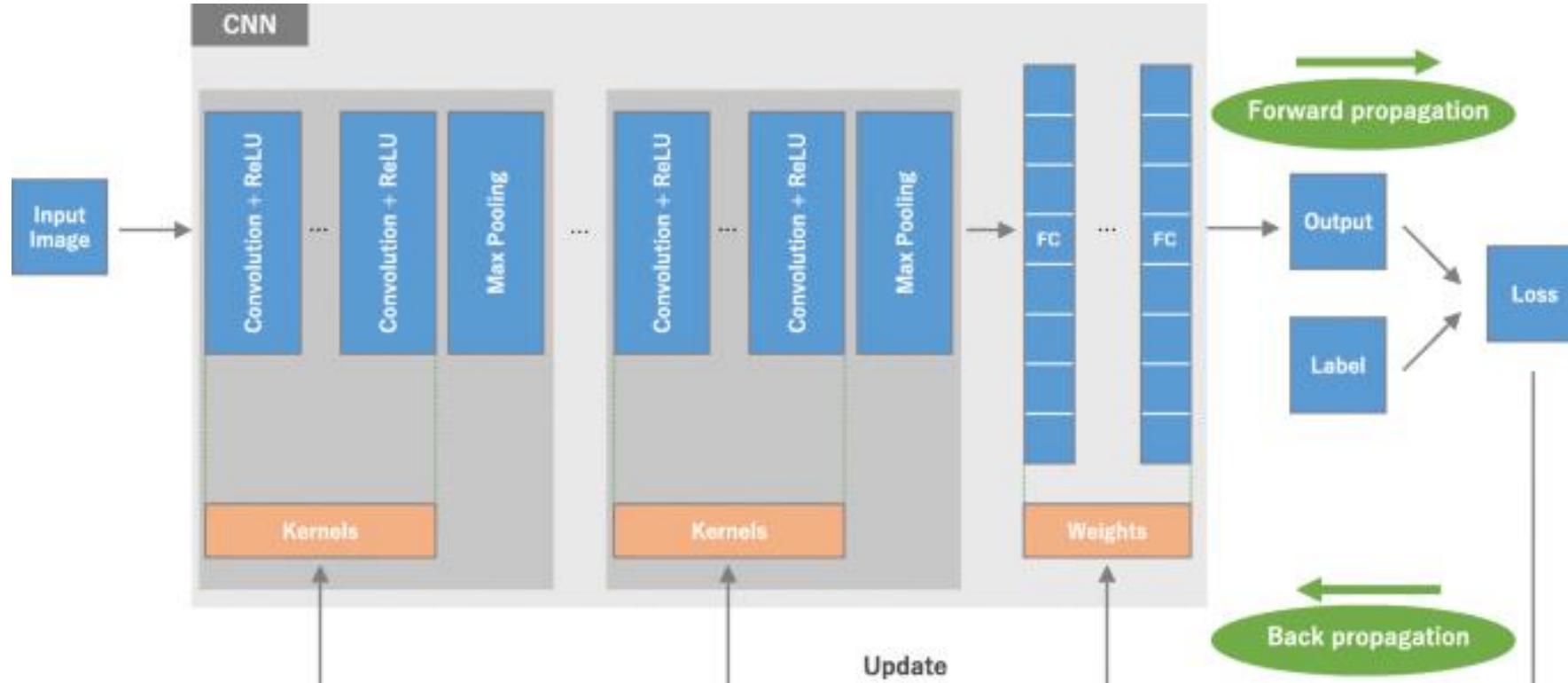


Flattening





Training di una Convolutional Neural Network





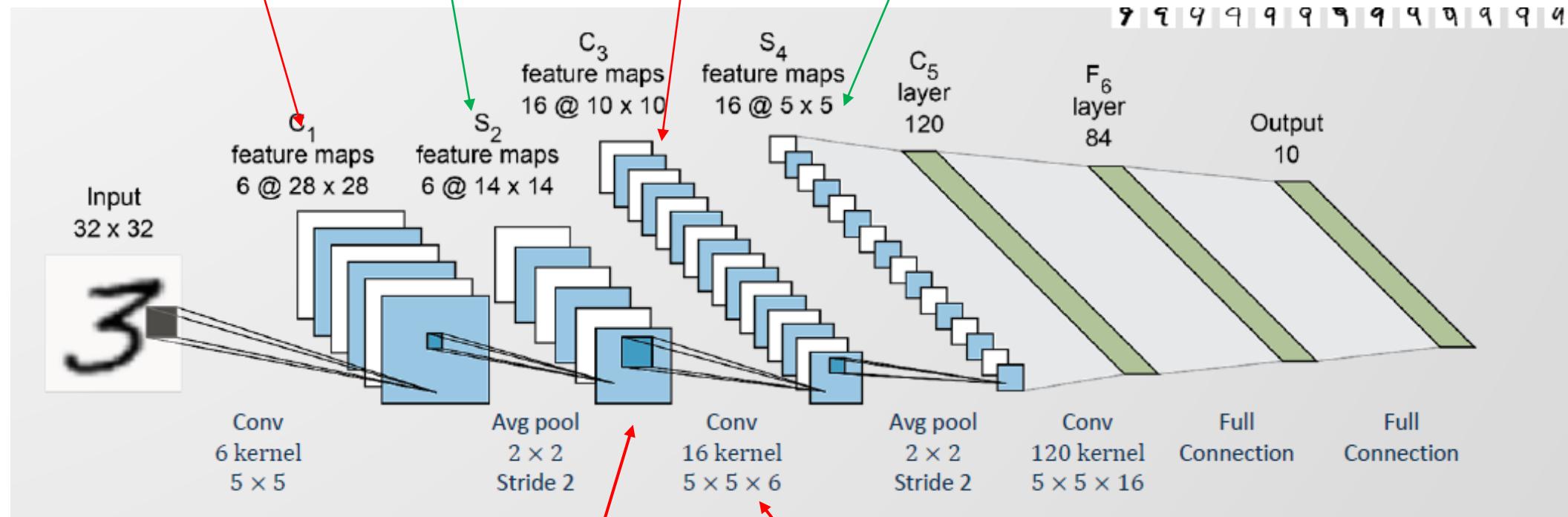
Simple cells

Complex cells

Simple cells

Complex cells

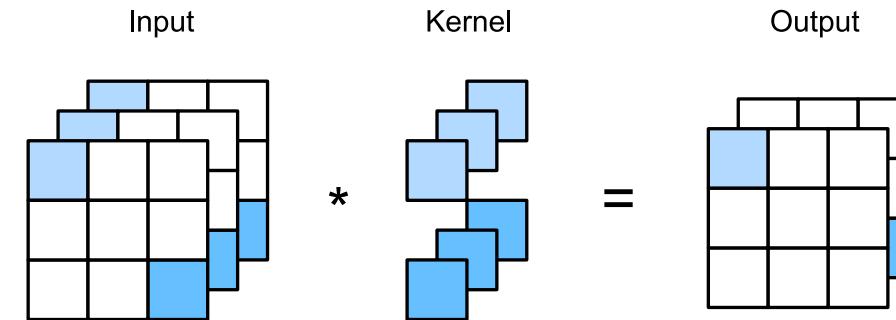
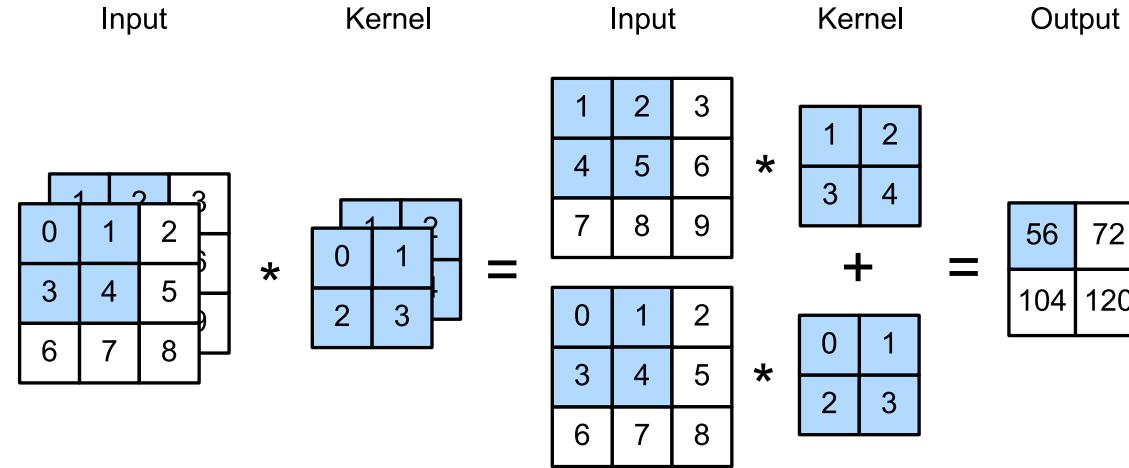
Architettura LENET-5 - Rete CNN per riconoscere cifre scritte a mano



Nota: Quando il volume di input in un'operazione di convoluzione contiene più canali (cioè è un tensore tridimensionale), solitamente si **utilizzano kernel di convoluzione con lo stesso numero di canali del volume di input**. In questo modo, per ogni kernel si esegue un'operazione di convoluzione tra ogni canale di input e il corrispondente canale del kernel. Alla fine, i risultati delle convoluzioni su ciascun canale vengono sommati insieme per produrre la mappa delle caratteristiche bidimensionale finale.



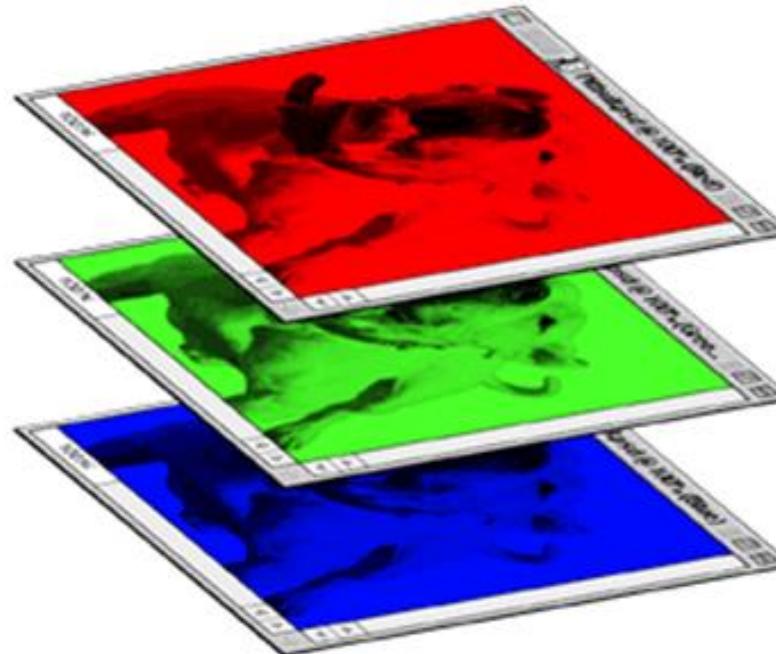
Caso in cui il volume di input in un'operazione di convoluzione contiene più canali





Input immagine a colori: RGB 3 canali

Immagine a colori



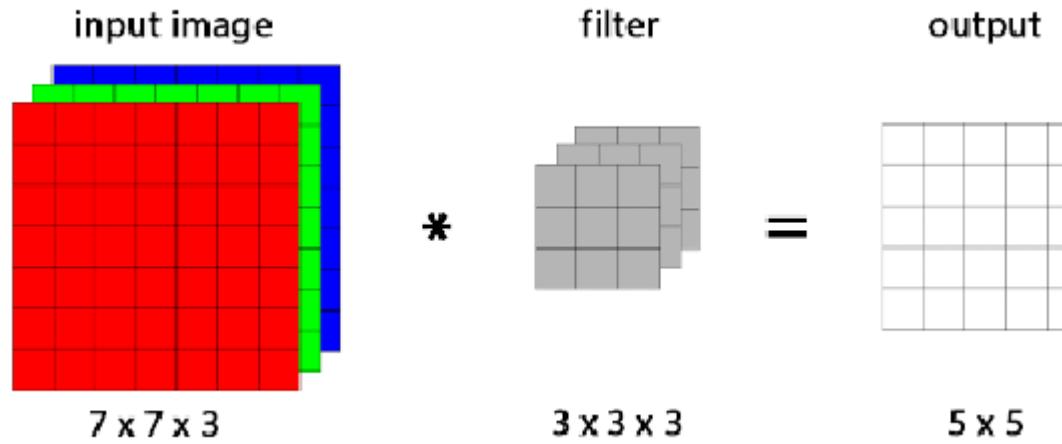
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0



Proprietà di convergenza ad un punto stazionario del gradient descent.

Il Gradient Descent aggiorna i parametri con

$$w_{k+1} = w_k - \eta \nabla C(w_k)$$

w è il vettore dei parametri e η è il learning rate.

La discesa del gradiente trova il minimo globale se la funzione è convessa.

Se sono soddisfatte le seguenti condizioni:

Condizione di Lipschitz per il gradiente: La norma del gradiente della funzione obiettivo deve essere limitata da una costante di Lipschitz. Cioè, esiste una costante $L > 0$ tale che per ogni punto x e per ogni punto y , si abbia $\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|$, dove $\nabla f(x)$ rappresenta il gradiente di f nel punto x . Questa condizione impone che il gradiente non cresca troppo velocemente, e che la funzione sia sufficientemente regolare.

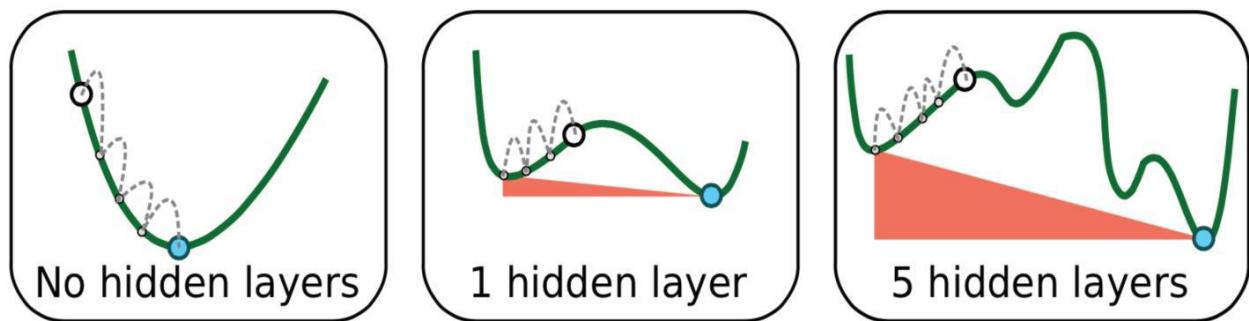
Step-size (o learning rate) : Il passo di aggiornamento α deve essere scelto in base alla costante di Lipschitz L . In particolare, per garantire la convergenza, il passo η deve essere minore o uguale a $1/L$. Questa condizione assicura che il passo non sia troppo grande rispetto alla crescita del gradiente, limitando così la possibilità di oscillazioni e garantendo la convergenza del metodo.

Sotto queste condizioni, il metodo di discesa del gradiente con passo fisso converge a un punto stazionario della funzione obiettivo, che può essere un minimo globale se la funzione è convessa.

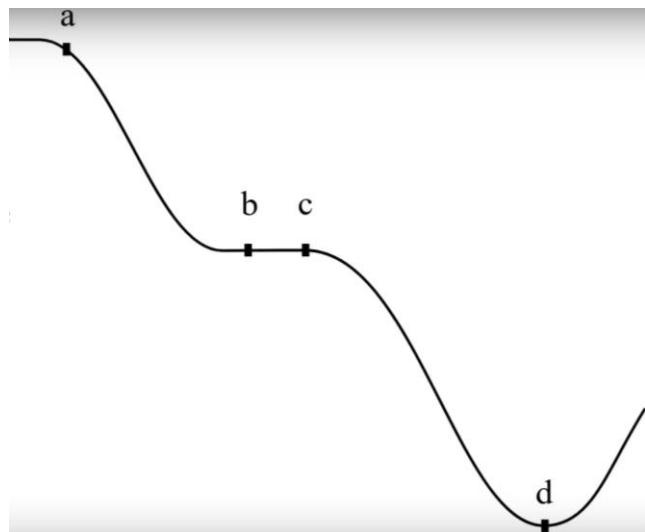
Tuttavia, è importante sottolineare che la scelta del passo fisso può essere limitante in termini di efficienza e velocità di convergenza in problemi più complessi. In pratica, spesso si utilizzano metodi di discesa del gradiente con passo adattivo, come ad esempio il metodo di discesa del gradiente con regola di Armijo o altre strategie avanzate, per ottenere una migliore convergenza.

Sfortunatamente, quasi tutti i problemi di ottimizzazione che sorgono in Deep Learning sono non convessi

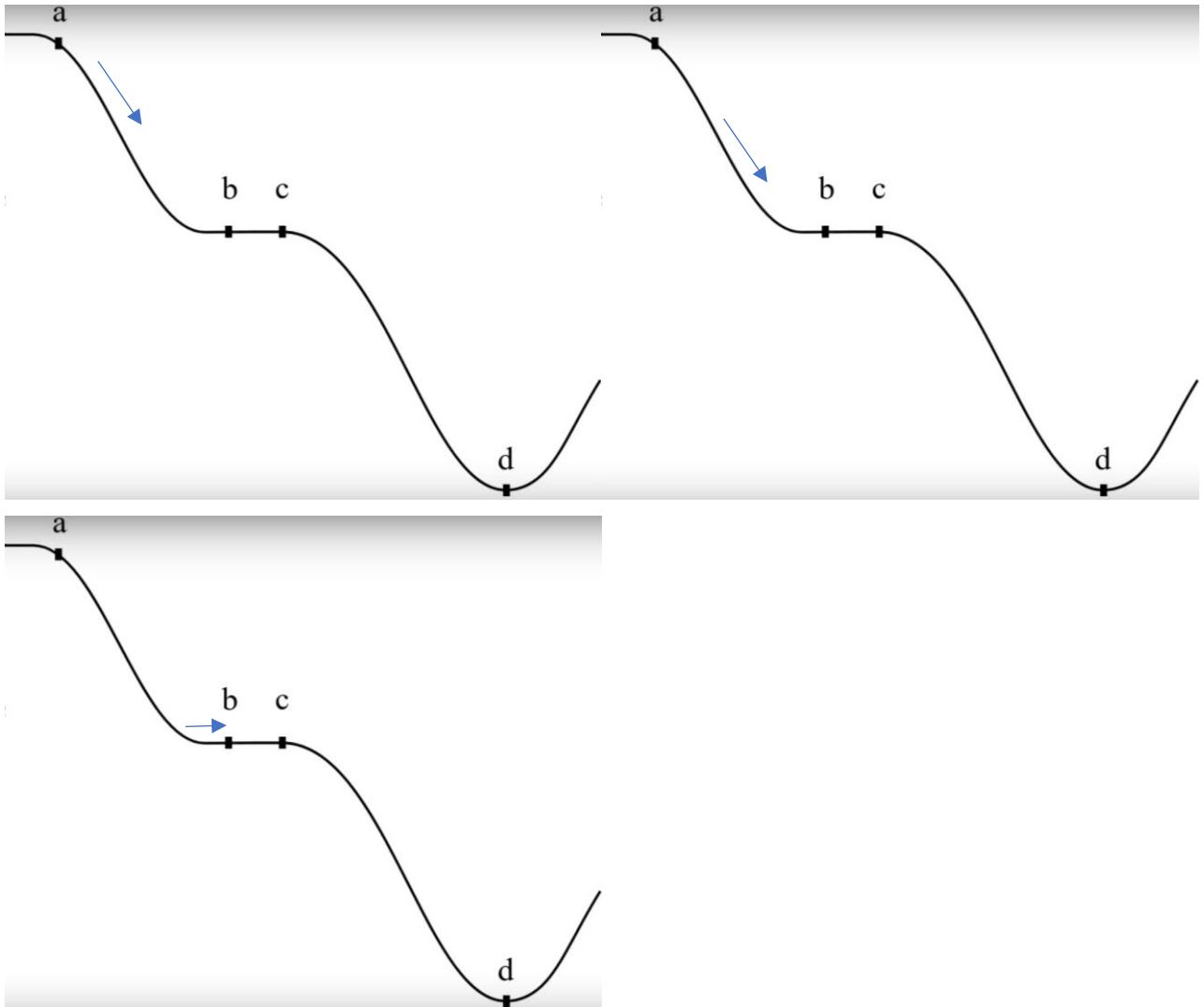
- introducendo la non linearità nella rete (aggiungendo strati nascosti), la funzione costo diventa non convessa e compaiono i minimi locali



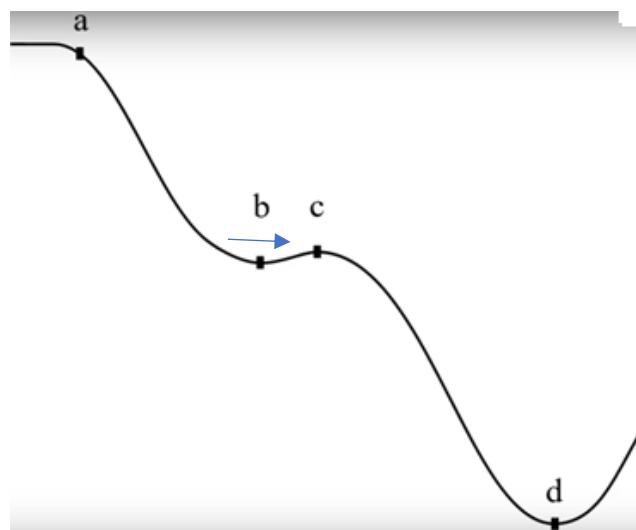
Consideriamo questa curva



Inizializziamo w ad un punto a ed applichiamo il metodo iterativo del gradiente.



w raggiunge il punto b e si ferma, poiché in b il gradiente ai annulla, la funzione è caratterizzata da una zona piatta, un *plateau* (*vanishing gradient*). Se invece di una regione piatta, in b c'è un minimo locale



w non può superare il massimo locale nel punto c e rimane bloccato in b .

In entrambi i casi non può raggiungere il minimo locale in d .

Simili situazioni si possono verificare nelle superfici multidimensionali. Specialmente nei punti sella dove il gradiente è zero, ma il punto non è un minimo oppure un massimo.

Importanza del Learning rate

Valori bassi di η

- possono far sì che sia necessario un numero elevato di passi prima che l'allenamento sia completato.

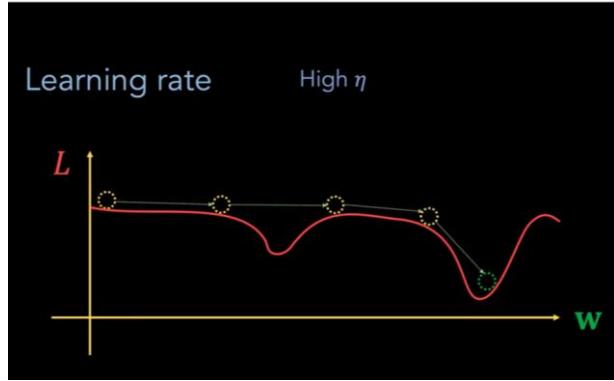


- possono far sì che i pesi rimangano bloccati in un minimo locale, che è sub-optimal se confrontato con il global minimum.

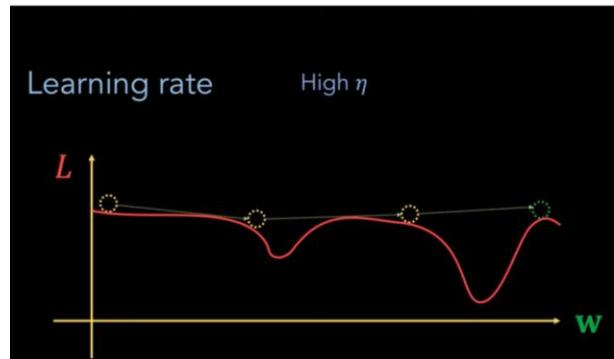


Valori alti di η

Valori alti di learning rate permettono di mitigare questi problemi passando oltre i minimi locali e raggiungendo il minimo in numero limitato di passi.



i pesi possono anche finire per superare il minimo target



Queste osservazioni fanno intuire che la scelta del learning rate è cruciale ed in pratica può richiedere molta sperimentazione per scoprire quale può essere il valore ottimale del learning rate.

Exact line search

La dimensione ottimale del learning rate può essere trovata risolvendo un problema di ottimizzazione unidimensionale

$$\eta^{(k)} = \arg \min_{\eta \geq 0} C(w^{(k)} + \eta p^{(k)})$$

Dove $p^{(k)}$ è la direzione di discesa. Gli algoritmi di ottimizzazione unidimensionale per trovare la dimensione del passo ottimale sono genericamente chiamati **exact line search**

Sono state proposte delle **inexact line search rule**. Tra le quali la prima è stata

Armijo rule

la regola di Armijo consente di trovare un passo appropriato che garantisca una riduzione significativa della funzione obiettivo, garantendo al contempo un'adeguata convergenza dell'algoritmo di ottimizzazione,

La regola di Armijo, chiamata anche regola di Armijo-Goldstein, è un criterio utilizzato nell'ottimizzazione numerica per determinare la dimensione del passo o lo step size da utilizzare durante la ricerca lineare in direzione di discesa all'interno di un algoritmo di ottimizzazione.

La regola di Armijo si basa sul concetto di "backtracking", che prevede di ridurre gradualmente la dimensione del passo fino a trovare un valore che soddisfi determinate condizioni. In particolare, la regola di Armijo si concentra sulla scelta di un passo che garantisca una riduzione sufficiente del valore della funzione obiettivo durante la ricerca lineare.

L'idea principale è che, partendo da un certo punto iniziale, si riduca progressivamente la dimensione del passo moltiplicandola per il parametro σ ($0 < \sigma < 1$) fino a quando non si ottiene una riduzione sufficiente nella funzione obiettivo. Questo processo viene ripetuto fino a quando la condizione di Armijo è soddisfatta .

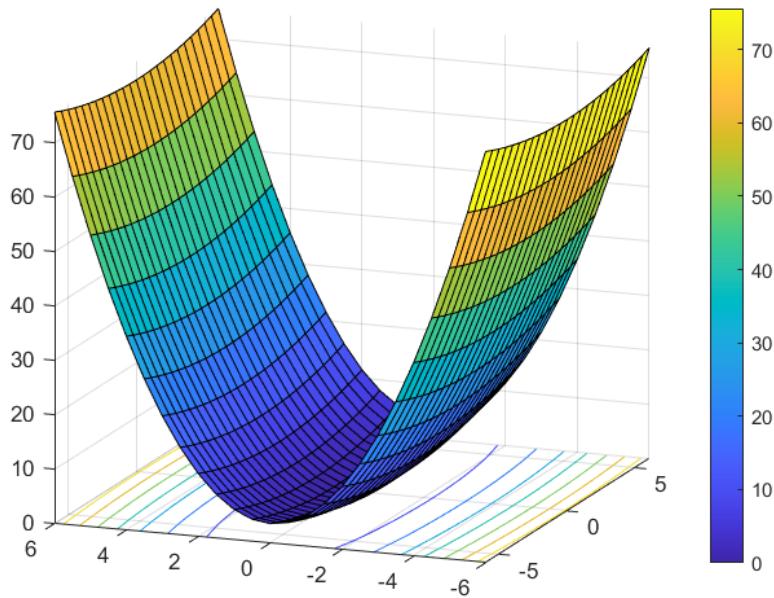
Se questa condizione viene soddisfatta

$$C(w^{(k)} + \eta p^{(k)}) \leq C(w^{(k)}) + \sigma \cdot \eta \nabla C(w^{(k)})^T p^{(k)}$$

allora il passo η viene accettato; altrimenti, il passo viene ridotto di un fattore σ e il processo viene ripetuto.

Armijo rule: parte con $\eta = \eta_0$ e lo fa decrescere moltiplicandolo per $\sigma \in (0,1)$, finché la funzione descresce sufficientemente.

Consideriamo adesso una lossy del tipo $f(w_1, w_2) = 0.1 w_1^2 + 2 w_2^2$

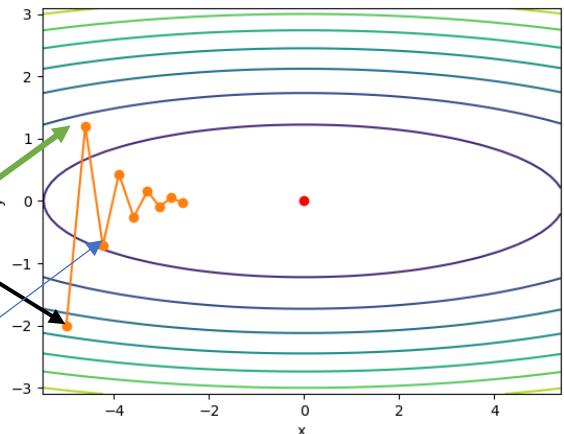
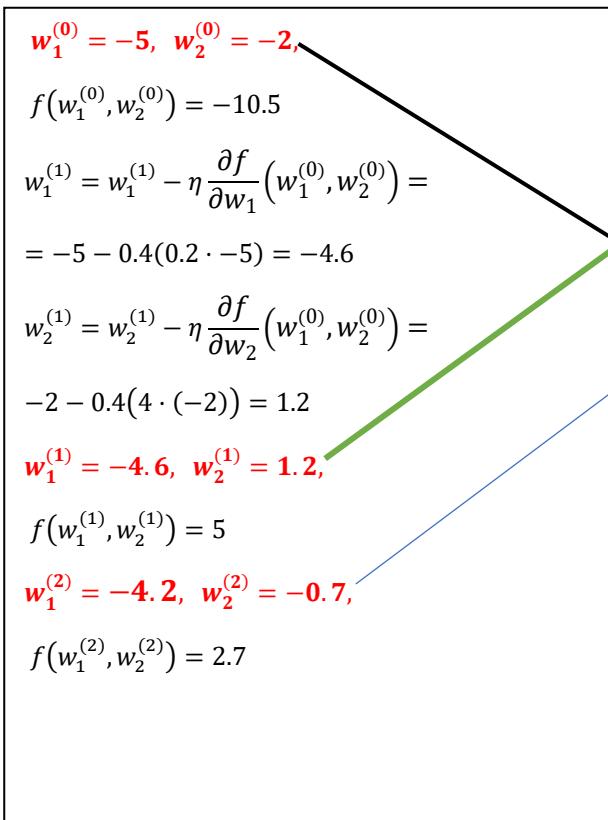


Ha il suo minimo in $(0,0)$. Questa funzione è *molto* piatta nella direzione di w_1 .

$$\nabla f(w_1, w_2) = \begin{bmatrix} \frac{\partial f(w_1, w_2)}{\partial w_1} \\ \frac{\partial f(w_1, w_2)}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 0.2 w_1 \\ 4w_2 \end{bmatrix}$$

Vediamo cosa succede quando eseguiamo la discesa del gradiente per calcolarne il minimo.

Scegliamo un learning rate di $\eta = 0.4$.



Per come è definita la funzione costo $f(w_1, w_2)$ il gradiente nella direzione w_2 è *molto* più alto e cambia molto più rapidamente rispetto alla direzione orizzontale. Quindi siamo bloccati tra due scelte: se scegliamo un learning rate basso che la soluzione non diverge nella direzione verticale w_2 , ma siamo gravati da una lenta convergenza nella direzione orizzontale w_1 . Viceversa, con un elevato tasso di apprendimento progrediamo rapidamente nella direzione orizzontale, ma c'è divergenza nella direzione verticale.

Dopo 20 epochi $w_1 : -0.943467, w_2 : -0.000073$

Per risolvere questi problemi si introduce il gradient descent con momento.

Il **Gradient Descent con momento** viene utilizzato per risolvere i problemi elencati.

Si definisce la **velocità** $v_k = \beta v_{k-1} + \nabla C(w_k)$

β prende il nome di momento e può variare tra 0 ed 1. v_0 viene inizializzato a zero.

La formula di aggiornamento dei pesi diventa:

$$w_{k+1} = w_k - \eta v_k$$

Se si pone $\beta = 0$, si ricade nel classico metodo del gradiente. Un valore appropriato è tra 0.8 e 0.9.

Questo metodo aggiunge al gradiente istantaneo la media su più gradienti *passati*.

Analizziamo il termine $v_k = \beta v_{k-1} + \nabla C(w_k)$

$$v_1 = \beta v_0 + \nabla C(w_1)$$

$$v_2 = \beta v_1 + \nabla C(w_2)$$

$$v_2 = \beta(\beta v_0 + \nabla C(w_1)) + \nabla C(w_2) = \beta^2 v_0 + \beta \nabla C(w_1) + \nabla C(w_2)$$

$$\begin{aligned} v_3 &= \beta v_2 + \nabla C(w_3) = \beta(\beta^2 v_0 + \beta \nabla C(w_1) + \nabla C(w_2)) + \nabla C(w_3) \\ &= \beta^3 v_0 + \beta^2 \nabla C(w_1) + \beta \nabla C(w_2) + \nabla C(w_3) \end{aligned}$$

$$v_k = \beta^k v_0 + \beta^{k-1} \nabla C(w_1) \dots + \beta \nabla C(w_{k-1}) + \beta^0 \nabla C(w_k)$$

Posto $v_0 = 0$

$$v_k = \sum_{j=1}^k \beta^{k-j} \nabla C(w_j)$$

Si basa sullo stesso concetto di **momento**, *quantità di moto in fisica*. Un classico esempio del concetto è una palla che rotola giù da una collina che raccoglie abbastanza slancio per superare una regione di altopiano e raggiungere un minimo globale invece di rimanere bloccata in un minimo locale.

Il metodo SGD con Momento aggiunge la “cronologia” agli aggiornamenti dei parametri dei problemi di discesa, il che accelera notevolmente il processo di ottimizzazione.

La nuova sostituzione del gradiente non punta più nella direzione della discesa più ripida in un caso particolare, ma piuttosto nella direzione di una media ponderata esponenziale dei gradienti passati.

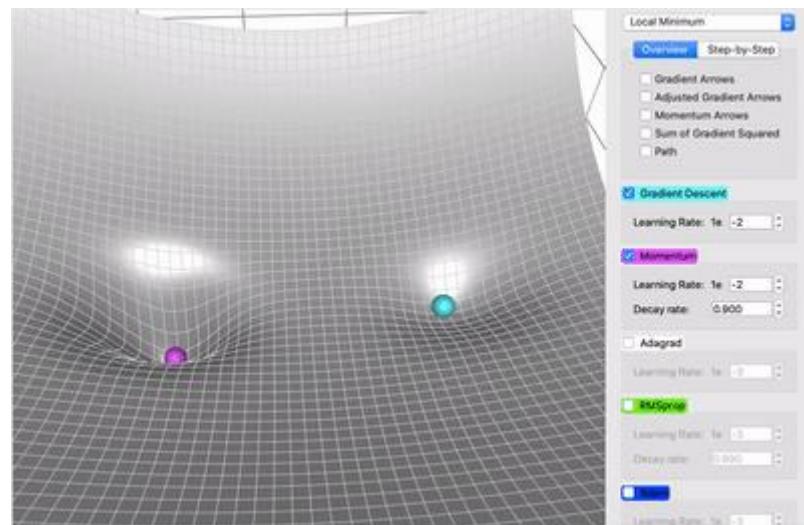
Il termine momento aumenta le dimensioni degli aggiornamenti quando i gradienti puntano nella stessa direzione mentre riduce le dimensioni degli aggiornamenti quando i gradienti cambiano direzione.

In questo modo si ottiene una più veloce convergenza e si riduce l'oscillazione.

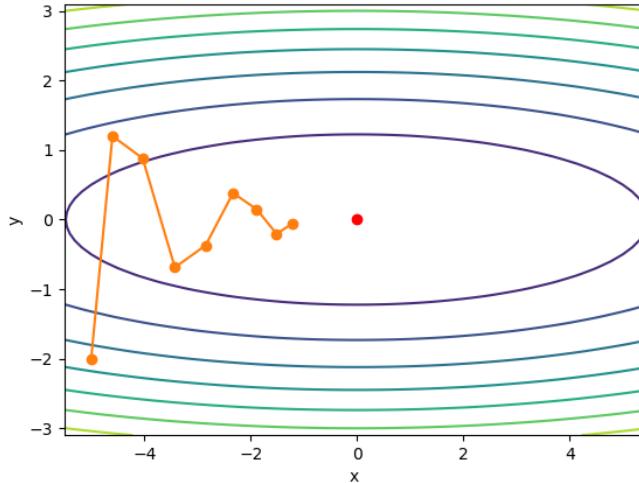
In questo modo la scelta del learning rate è meno cruciale poiché la procedura di training si adatta alla particolare loss function.



La quantità di cronologia inclusa nell'equazione di aggiornamento è determinata tramite l'iperparametro β . Questo iperparametro è un valore compreso tra 0 e 1, dove un valore di quantità di moto pari a 0 equivale alla discesa del gradiente senza quantità di moto. Un valore di momentum più alto significa che vengono considerati più gradienti del passato (storia).



Tornando al nostro esempio, il metodo del gradiente con momento produrrà le seguenti iterazioni:



nella direzione w_1 il metodo del gradiente con momento accumulerà gradienti ben allineati, aumentando così la distanza che percorriamo ad ogni passo. Al contrario, nella direzione w_2 in cui oscillano i gradienti, l'accumulo dei gradienti ridurrà la dimensione del passo a causa delle oscillazioni che si annullano a vicenda.

Tuttavia, una palla che rotola giù da una collina, seguendo ciecamente il pendio, è altamente insoddisfacente

Gradiente accelerato di Nesterov usa un termine di momento più intelligente, che ha un'idea di dove sta andando in modo che sappia rallentare prima che la funzione salga di nuovo.

- **Varianti del Gradient Descent**

Batch Gradient Descent

Per il calcolo della funzione costo $C(w)$ vengono usate tutte gli n_T campioni del training set .

$$T = \{(x^{(j)}, y^{(j)}), x^{(j)} \in \mathbb{R}^d, y^{(j)} \in \mathbb{R}^s, j = 1, \dots, n_T\},$$

$$\arg \min_w C(w) = \frac{1}{n_T} \sum_{j=1}^{n_T} L(y^{(j)}, \hat{y}^{(j)})$$

Quindi:

- si considera l'intero set di addestramento,
- si esegue la Forward Propagation e si calcola la funzione di costo.
- si aggiornano i parametri usando il tasso di variazione di questa funzione di costo rispetto ai parametri.

Epoca: l'intero set di addestramento viene passato attraverso il modello, vengono eseguite la propagazione in avanti e la propagazione all'indietro e i parametri vengono aggiornati.

Nel batch Gradient Descent poiché stiamo utilizzando l'intero set di addestramento, i parametri verranno aggiornati solo una volta per epoca.

Discesa del gradiente stocastico (SGD)

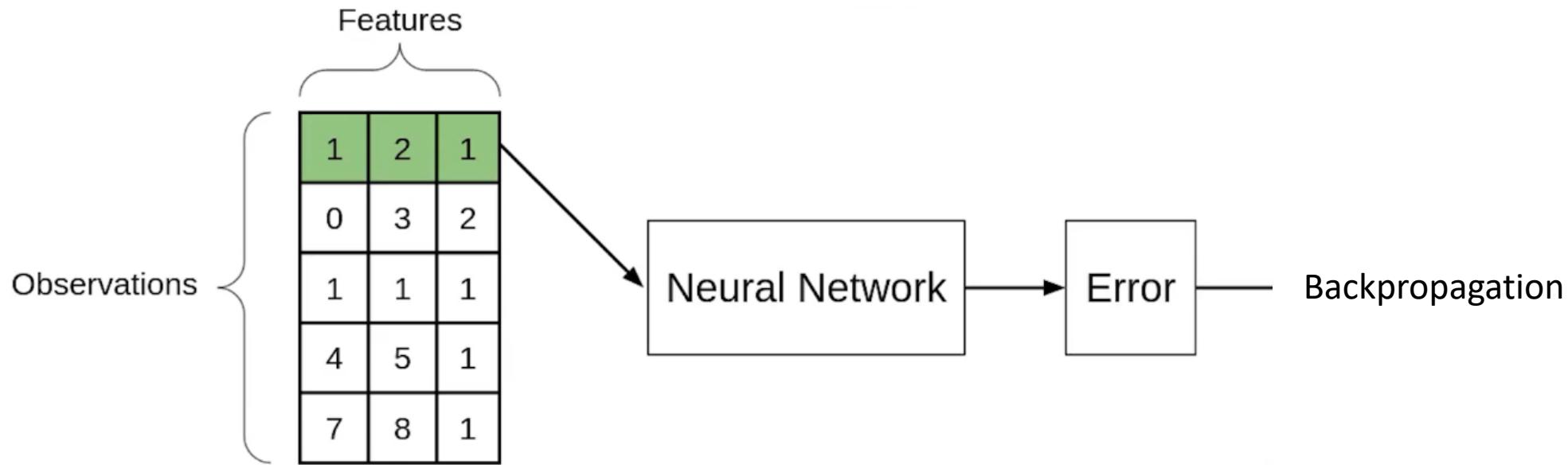
Se si utilizza una singola osservazione per calcolare la funzione di costo, si parla di **Stochastic Gradient Descent**, comunemente abbreviato **SGD**. Passiamo una sola osservazione alla volta, calcoliamo il costo e aggiorniamo i parametri.

Esempio:

Supponiamo di avere un dataset di training formato da 5 campioni, ognuno dei quali costituito da 3 input

Features			
Observations	1	2	1
	0	3	2
	1	1	1
	4	5	1
	7	8	1

Ora, se usiamo l'SGD, prenderemo la prima osservazione, poi la passeremo attraverso la rete neurale, calcoleremo l'errore e quindi aggiorneremo i parametri.



Quindi entrerà in input la seconda osservazione ed saranno eseguiti passaggi simili su di essa. Questo passaggio verrà ripetuto fino a quando tutte le osservazioni non saranno passate attraverso la rete e i parametri non saranno stati aggiornati.

Ogni aggiornamento dei parametri, viene chiamato **Iterazione**.

In questo esempio, poiché abbiamo 5 osservazioni, i parametri verranno aggiornati 5 volte (sono state effettuate 5 iterazioni).

Nel caso del Batch Gradient Descent tutte le osservazioni insieme sarebbero state elaborate dalla rete ed i parametri sarebbero stati aggiornati solo una volta.

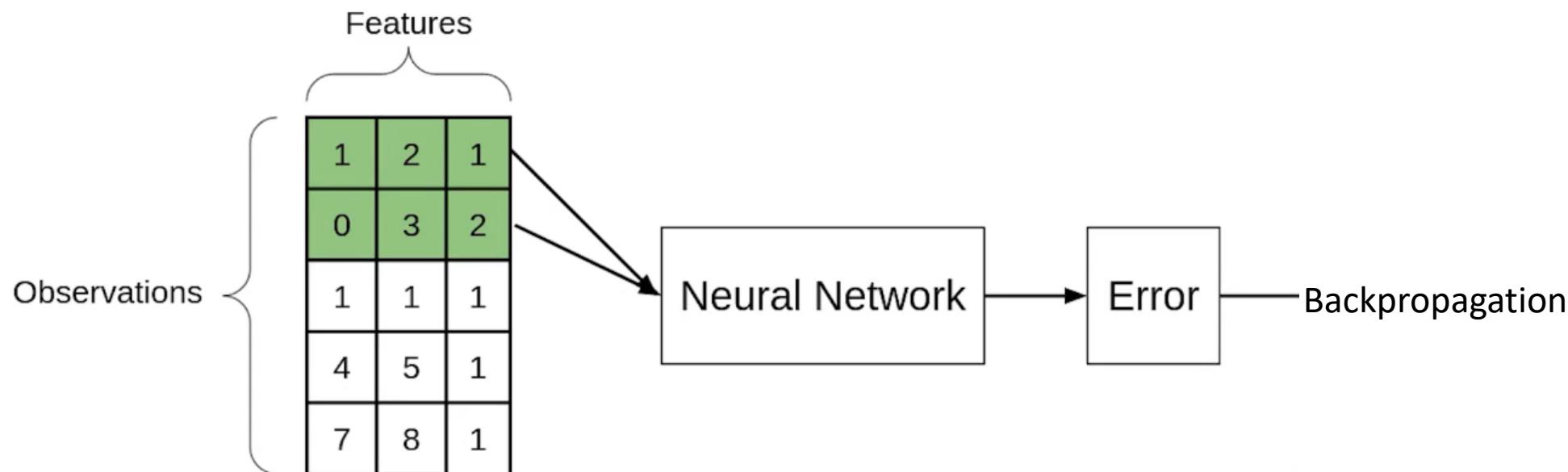
Nel caso di SGD, ci saranno n_T iterazioni per epoca, dove n_T è il numero di osservazioni nel dataset di training.

Se si utilizza l'intero set di dati per calcolare la funzione di costo si parla di **Batch Gradient Descent** e se si utilizza una singola osservazione per calcolare il costo si parla di **SGD**.

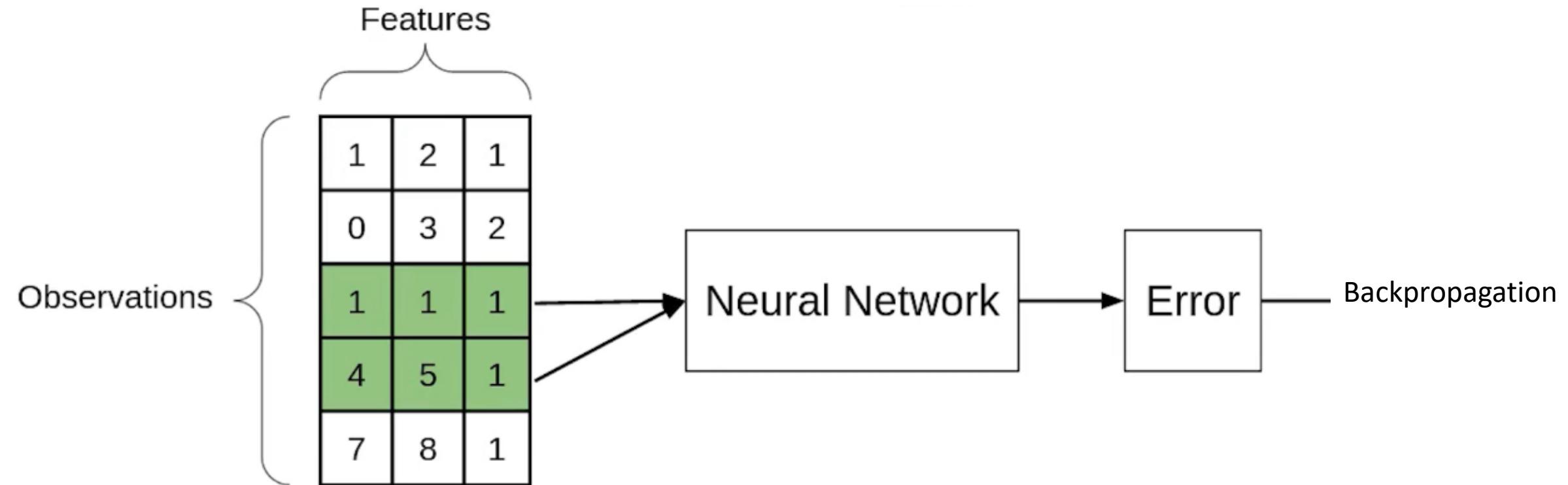
Mini-batch Stochastic Gradient Descent:

Per calcolare la funzione di costo si considera un sottoinsieme dell'intero set di dati. Quindi, se ci sono n_T osservazioni, il numero di osservazioni in ciascun sottoinsieme o mini-batch sarà maggiore di 1 e minore di n_T .

Ancora una volta prendiamo lo stesso esempio. Supponiamo che la dimensione del batch sia 2. Quindi prenderemo le prime due osservazioni, le passeremo attraverso la rete neurale, calcoleremo l'errore e quindi aggiorneremo i parametri.



Quindi prenderemo le due osservazioni successive ed eseguiremo passaggi simili, passeremo attraverso la rete, calcoleremo l'errore e aggiorneremo i parametri.



Poiché ci rimane la singola osservazione nell'iterazione finale, ci sarà solo una singola osservazione e i parametri saranno aggiornati utilizzando questa osservazione.

Confronto tra Batch GD, SGD e Mini-batch SGD:

Segue un confronto tra le diverse varianti di Gradient Descent

Confronto: numero di osservazioni utilizzate per l'aggiornamento.

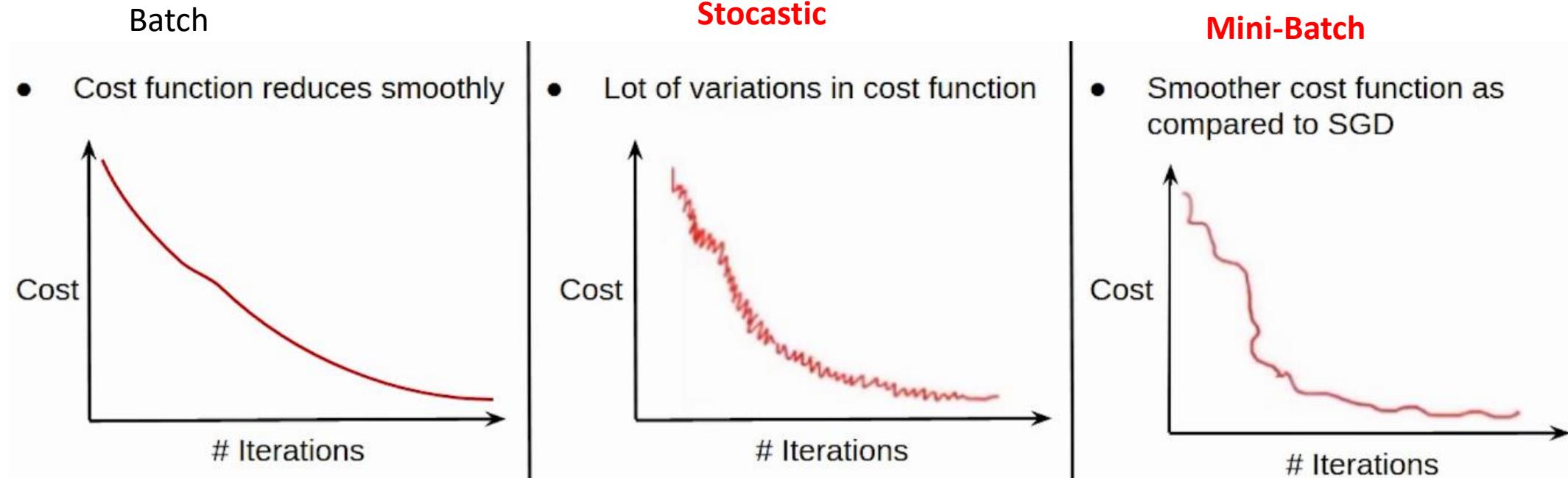
In batch Gradient Descent

- prendiamo l'intero set di dati
- calcoliamo la funzione di costo
- aggiorniamo i parametri

Nel caso di **Stochastic Gradient Descent**, aggiorniamo i parametri dopo ogni singola osservazione e sappiamo che ogni volta che i pesi vengono aggiornati si parla di iterazione.

Nel caso di **Mini-batch Stochastic Gradient Descent**, prendiamo un sottoinsieme di dati e aggiorniamo i parametri in base a ogni sottoinsieme.

Confronto Funzione costo



- Poiché nel caso del **Batch GD** aggiorniamo i parametri utilizzando l'intero set di dati la funzione costo, in questo caso, si riduce uniformemente.
- Questo aggiornamento nel caso di **SGD** non è così fluido. Poiché stiamo aggiornando i parametri sulla base di una singola osservazione, ci sono molte iterazioni. Potrebbe anche essere possibile che il modello inizi ad apprendere anche il rumore.
- L'aggiornamento della funzione di costo nel caso di **Mini-batch Gradient Descent** è più fluido rispetto a quello della funzione di costo in SGD. Dal momento che non aggiorniamo i parametri dopo ogni singola osservazione ma dopo ogni sottoinsieme dei dati

Veniamo ora al costo di calcolo e al tempo impiegato da queste varianti **di Gradient Descent**.

Poiché dobbiamo caricare l'intero set di dati alla volta, eseguire la propagazione in avanti su di esso e calcolare l'errore e quindi aggiornare i parametri, nel **caso Batch Gradient Descent** il costo di calcolo è molto elevato.

Il tempo di calcolo della funzione costo nel caso di **SGD** è inferiore rispetto a **Batch Gradient Descent** poiché dobbiamo caricare ogni singola osservazione alla volta, ma il tempo di calcolo qui aumenta poiché ci sarà un numero maggiore di aggiornamenti che si tradurrà in un numero maggiore di iterazioni .

Nel caso di **Mini-batch Stochastic Gradient Descent**, prendendo un sottoinsieme dei dati ci sono un numero minore di iterazioni o aggiornamenti e quindi il tempo di calcolo nel caso di mini-batch Gradient Descent è inferiore a SGD.

Inoltre, poiché non stiamo caricando l'intero set di dati alla volta mentre carichiamo un sottoinsieme dei dati, anche il tempo di calcolo della funzione costo è inferiore rispetto alla discesa del gradiente batch. Questo è il motivo per cui di solito si preferisce utilizzare il **Mini-batch Stochastic Gradient Descent**.

Iperparametri

Gli iperparametri sono parametri esterni al modello di machine learning che devono essere impostati prima dell'avvio del processo di addestramento. A differenza dei parametri del modello, che vengono appresi durante il processo di addestramento stesso, gli iperparametri influenzano il comportamento del processo di addestramento e la configurazione del modello.

Gli iperparametri determinano come avviene l'addestramento del modello e possono includere:

1. **Learning rate** : Determina quanto velocemente o lentamente il modello si adatta ai dati.
2. **Numero di epoch**: il numero di volte in cui l'intero set di dati di addestramento viene utilizzato per addestrare il modello. Un numero insufficiente di epoch può portare a un modello non addestrato adeguatamente, mentre un numero eccessivo di epoch può portare a un overfitting.
3. **Dimensione del mini-batch**: il numero di esempi di addestramento utilizzati in ciascuna iterazione dell'algoritmo di ottimizzazione (ad esempio, SGD o mini-batch GD). La dimensione del mini-batch può influenzare la velocità di apprendimento e la stabilità dell'addestramento.

1. Regolarizzazione: i parametri che controllano la regolarizzazione del modello, come il peso della regolarizzazione L1 o L2, che influenzano la complessità del modello e la tendenza all'overfitting.

2. Inizializzazione dei pesi: il metodo utilizzato per inizializzare i pesi del modello. Una buona inizializzazione può favorire una convergenza più rapida e una migliore performance.

3. Funzione di attivazione: la funzione utilizzata per calcolare l'output di un'unità nel modello. Le funzioni di attivazione comuni includono ReLU, sigmoid e tanh.

La scelta corretta degli iperparametri può influenzare significativamente le prestazioni del modello. Spesso, gli iperparametri vengono regolati attraverso tentativi ed errori, eseguendo iterazioni multiple di addestramento e valutando le prestazioni su un set di dati di validazione. L'esperienza, la conoscenza del dominio e le best practice possono guidare la scelta degli iperparametri ottimali per un determinato problema di machine learning.

1. Architettura del modello: la struttura del modello, inclusi il numero di strati, il numero di unità in ciascun strato e le connessioni tra gli strati. La scelta dell'architettura dipende dal problema e dai dati specifici.

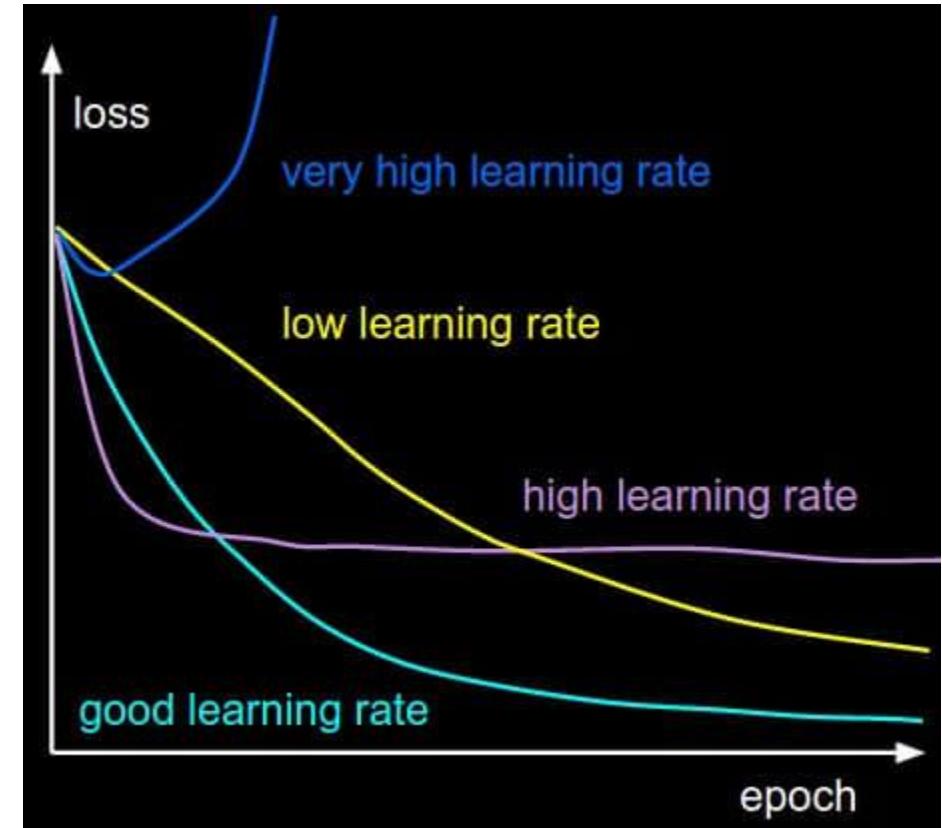
Nel seguente collegamento saranno analizzate le proprietà della discesa del gradiente per minimizzare un funzionale e sarà introdotto il metodo di discesa con momento

[GradientDescentMomentum.pdf](#)

Learning rate adattivo

Uno degli iperparametri più difficili da regolare nelle reti neurali è il **learning rate**.

- La funzione costo diminuisce ma richiede molto più tempo per converge (**learning rate basso**, marrone)
- La funzione costo raggiunge un valore migliore di quello iniziale,ma è ancora lontano da un valore ottimale (**Learning rate alto**: viola)
- La funzione costo inizialmente diminuisce poi inizia ad aumentare (**Learning Rate molto alto**, curva blu)
- La funzione costo diminuisce costantemente fino a raggiungere il valore minimo possibile (**Learning rate buono**, curva azzurra)



Effetto di differenti learning rate sulla minimizzazione della funzione costo.

Learning rate scheduling

La regolazione del learning rate durante l'allenamento è spesso importante tanto quanto la selezione dell'ottimizzatore

- un Learning rate alto è auspicabile all'inizio poiché i pesi sono lontani dai minimi
- un Learning rate basso è più appropriato nella fase finale dell'apprendimento perché i pesi sono già vicini ai minimi (aumentando la possibilità di raggiungere il minimo)

Per regolare il Learning rate , ci sono una serie di aspetti da considerare

- Grandezza: se il learning rate è troppo grande, l'ottimizzazione diverge, se è troppo piccolo ci vuole troppo tempo per l'allenamento o si finisce con un risultato non ottimale
- Tasso di decadimento se l'LR rimane grande potremmo rimbalzare intorno al minimo senza raggiungerlo
- Inizializzazione: come i parametri sono impostati inizialmente e come si evolvono? Utilizzare Learning rate iniziali grandi potrebbe non essere utile, poiché i parametri iniziali sono casuali le iniziali direzioni di aggiornamento potrebbero essere prive di significato

Le tecniche di aggiornamento del learning rate cercano di regolarlo durante l'allenamento riducendolo in accordo ad uno schema predefinito

Sapere quando far decadere il Learning rate può essere complicato

- se si fa decadere lentamente si sprega tempo di calcolo rimbalzando con pochi miglioramenti per molto tempo
- decadimento troppo aggressivo e il sistema si raffredderà troppo rapidamente incapace di raggiungere la posizione migliore che può.

Esistono tre tipi comuni di implementazione del decadimento del learning rate:

- **Step decay** : riduce il tasso di apprendimento iniziale η_0 di un fattore δ ogni numero predefinito di epoche s

$$\eta = \eta_0 \cdot \delta^{\left\lfloor \frac{n}{s} \right\rfloor}$$

dove η_0 e δ e s sono iperparametri e n è il numero di epoche eseguite

- Il **decadimento esponenziale** ha la forma matematica

$$\eta = \eta_0 \cdot e^{-kt}$$

dove η_0 e k sono iperparametri e t è il numero di iterazione corrente

- Il **decadimento basato sul tempo** divide il learning rate iniziale η_0 in funzione del numero di iterazioni eseguite (t)

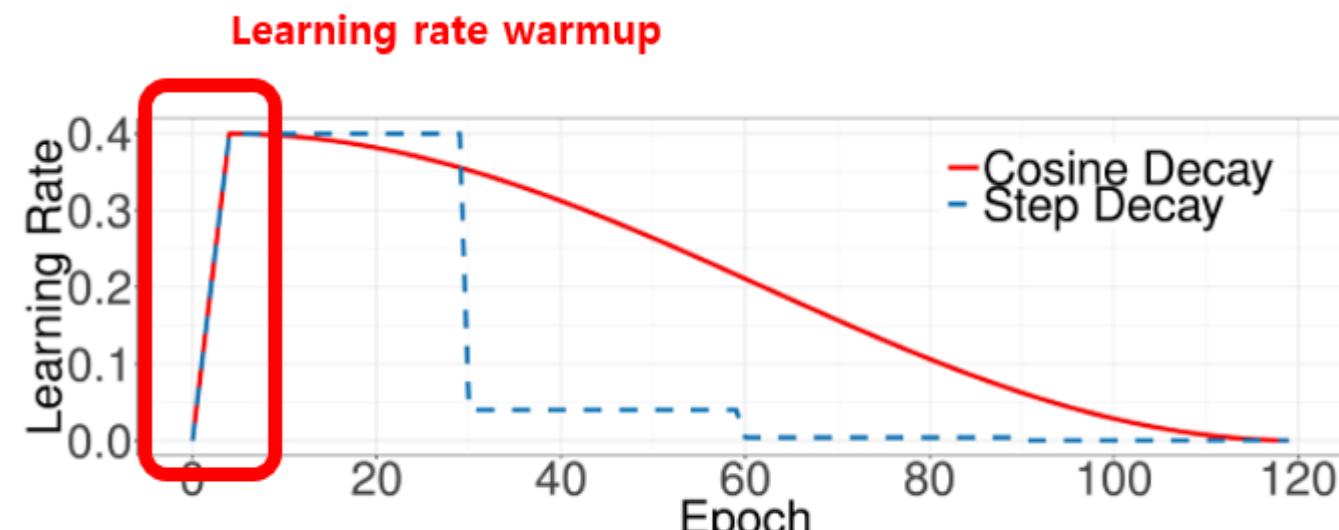
$$\eta = \frac{\eta_0}{1 + k \cdot t}$$

dove η_0 e k sono iperparametri

In alcuni casi, l'inizializzazione casuale dei parametri **non garantisce una buona soluzione** soprattutto se all'inizio viene utilizzato un **learning rate** grande che porta alla divergenza

Questo problema può essere affrontato scegliendo un **learning rate** sufficientemente piccolo per evitare divergenze all'inizio, purtroppo questo significa che il progresso è molto lento.

Una soluzione semplice consiste nell'utilizzare un periodo di **warm-up**: si inizia con un learning rate molto inferiore al learning rate "iniziale" e poi lo si aumenta in alcune iterazioni o epoche fino a raggiungere quel learning rate "iniziale" e poi lo si riduce fino alla fine del processo di ottimizzazione.



(a) Learning Rate Schedule

Learning rate adattivo

La sfida dell'utilizzo di un piano di aggiornamento dei **learning rate** è che gli iperparametri devono essere definiti in anticipo e dipendono fortemente dal tipo di modello e problema.

Un altro problema è che lo stesso **learning rate** viene applicato a tutti gli aggiornamenti dei pesi.

Se disponiamo di dati sparsi, potremmo invece voler aggiornare i pesi in misura diversa

Abbiamo bisogno di diminuire il valore del LR in modo diverso per ciascun peso man mano che l'allenamento procede

Ci sono quattro metodi rappresentativi che modificano in modo adattivo i LR

Adagrad

RMSProp

Adadelta

Adam

Adagrad

- Adagrad **adatta il Learning Rate ai parametri**, eseguendo aggiornamenti più grandi per i parametri poco frequenti e aggiornamenti più piccoli per quelli frequenti.
- Regola il tasso di apprendimento per i parametri in proporzione alla loro cronologia di aggiornamenti, più aggiornamenti = più decadimento

$$s_k = s_{k-1} + (\nabla C(w_k))^2$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{s_t + \epsilon}} \nabla C(w_k)$$

Minore è il gradiente accumulato, minore sarà il valore s_k , portando a un learning rate maggiore (

Uno dei principali vantaggi è che Adagrad elimina la necessità di regolare manualmente il learning rate
Il principale punto debole è l'accumulo dei gradienti al quadrato
durante l'addestramento la somma accumulata cresce, il learning rate diminuisce diventando infinitesimamente piccolo fino a quando la rete non è più in grado di acquisire ulteriore conoscenza

RMSProp

RMSProp è stato introdotto **per ridurre la diminuzione aggressiva** del learning rate di Adagrad

Modifica la parte di accumulo del gradiente di Adagrad con una media ponderata esponenziale dei gradienti al quadrato invece della somma dei gradienti al quadrato

$$s_k = \gamma s_{k-1} + (1 - \gamma) (\nabla C(w_k))^2$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{s_t + \epsilon}} \nabla C(w_k)$$

ADADELTA

Adadelta è un'altra variante di Adagrad proposta per superare il suo principale inconveniente

Come RMSProp calcola l'accumulo del gradiente come una media ponderata esponenziale dei gradienti al quadrato ma, a differenza di RMSProp, **non richiede di impostare un learning rate** in quanto utilizza la quantità di cambiamento stessa come calibrazione per il cambiamento futuro

$$s_k = \gamma s_{k-1} + (1 - \gamma) (\nabla C(w_k))^2$$

$$\tilde{\nabla} C = \frac{\sqrt{\Delta w_{k-1} + \epsilon}}{\sqrt{s_t + \epsilon}} \nabla C(w_k)$$

$$w_{k+1} = w_k - \tilde{\nabla} C$$

$$\Delta w_k = \gamma \Delta w_{k-1} + (1 - \gamma) (\tilde{\nabla} C)^2$$

dove Δw_k è la media pesata esponenziale dei quadrati dei gradienti pesati. ($\tilde{\nabla} C$)

ADAM

L'obiettivo principale di ADAM (Adaptive Moment Estimation) è quello di combinare i vantaggi di due altri algoritmi di ottimizzazione: RMSprop e Momento.

Utilizza la media pesata esponenziale dei gradienti ai passi precedenti, per ottenere una stima del momento del gradiente

$$v_k = \beta_1 v_{k-1} + (1 - \beta_1) \nabla C(w_k)$$

e del momento secondo del gradiente

$$s_k = \beta_2 s_{k-1} + (1 - \beta_2) (\nabla C(w_k))^2$$

Si noti che se si inizializzano $v_0 = 0$ ed $s_0 = 0$, i momenti iniziali "sbilanciati" all'inizio del processo di apprendimento, per compensare questo sbilanciamento si usano le seguenti normalizzazioni

$$\hat{v}_k = \frac{v_k}{1 - (\beta_1)^k} \quad \hat{s}_k = \frac{s_k}{1 - (\beta_2)^k}$$

Allora l'equazione di aggiornamento diventa

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{s}_k + \epsilon}} \hat{v}_k$$

il metodo ADAM adatta in modo dinamico il learning rate per ciascun parametro del modello utilizzando le stime dei momenti del primo e del secondo ordine. Questo approccio consente ad ADAM di convergere più velocemente e di essere più robusto rispetto ad altri metodi di ottimizzazione tradizionali.

Ci sono diversi metodi di ottimizzazione utilizzati nel machine learning, ognuno con le proprie caratteristiche e vantaggi. Di seguito ti fornisco un confronto tra alcuni dei metodi di ottimizzazione più comuni.

1. Gradient Descent (GD):

1. È il metodo di base per l'ottimizzazione dei modelli di machine learning.
2. Richiede il calcolo del gradiente dell'intero set di dati di addestramento ad ogni iterazione, rendendolo computazionalmente costoso per grandi set di dati.
3. Può soffrire di problemi come il plateau dei gradienti o la convergenza lenta in presenza di curve di costo complesse o di minime locali.

2. Stochastic Gradient Descent (SGD):

1. Utilizza un campione casuale dal set di dati di addestramento per calcolare il gradiente ad ogni iterazione.
2. Rispetto al GD, SGD è più efficiente computazionalmente, soprattutto per grandi set di dati.
3. Tuttavia, la variabilità introdotta dall'utilizzo di un singolo campione può rendere l'ottimizzazione più rumorosa e richiedere una scelta attenta del tasso di apprendimento.

3. Mini-batch Gradient Descent:

1. È una variante di SGD in cui il gradiente viene calcolato su un piccolo sottoinsieme di campioni (mini-batch) a ogni iterazione.
2. Combina vantaggi di GD e SGD: più efficiente di GD e meno rumoroso di SGD.
3. Richiede una scelta appropriata della dimensione del mini-batch, che può influenzare la convergenza e la velocità di apprendimento.

1. Momentum-based Methods (e.g., Momentum, Nesterov Accelerated Gradient):

1. Utilizzano un momento che tiene conto dei gradienti precedenti per accelerare la convergenza.
2. Sono particolarmente efficaci per ridurre l'effetto di oscillazioni o rumore nella direzione del gradiente.
3. Possono aiutare a superare zone di plateau e minimi locali piatti.

2. Adagrad:

1. Adatta il learning rate per ciascun parametro in base alla somma dei gradienti passati.
2. Funziona bene in presenza di sparsi gradienti o feature sparse.
3. Tuttavia, può diminuire troppo velocemente il tasso di apprendimento nel corso dell'addestramento, rendendolo inefficiente nelle fasi successive.

3. RMSprop:

1. Adatta il learning rate in base alla media mobile delle varianze dei gradienti passati.
2. Aiuta ad affrontare il problema di diminuzione eccessiva del tasso di apprendimento in Adagrad.
3. È particolarmente utile quando i parametri del modello hanno diverse scale di aggiornamento.

4. ADAM:

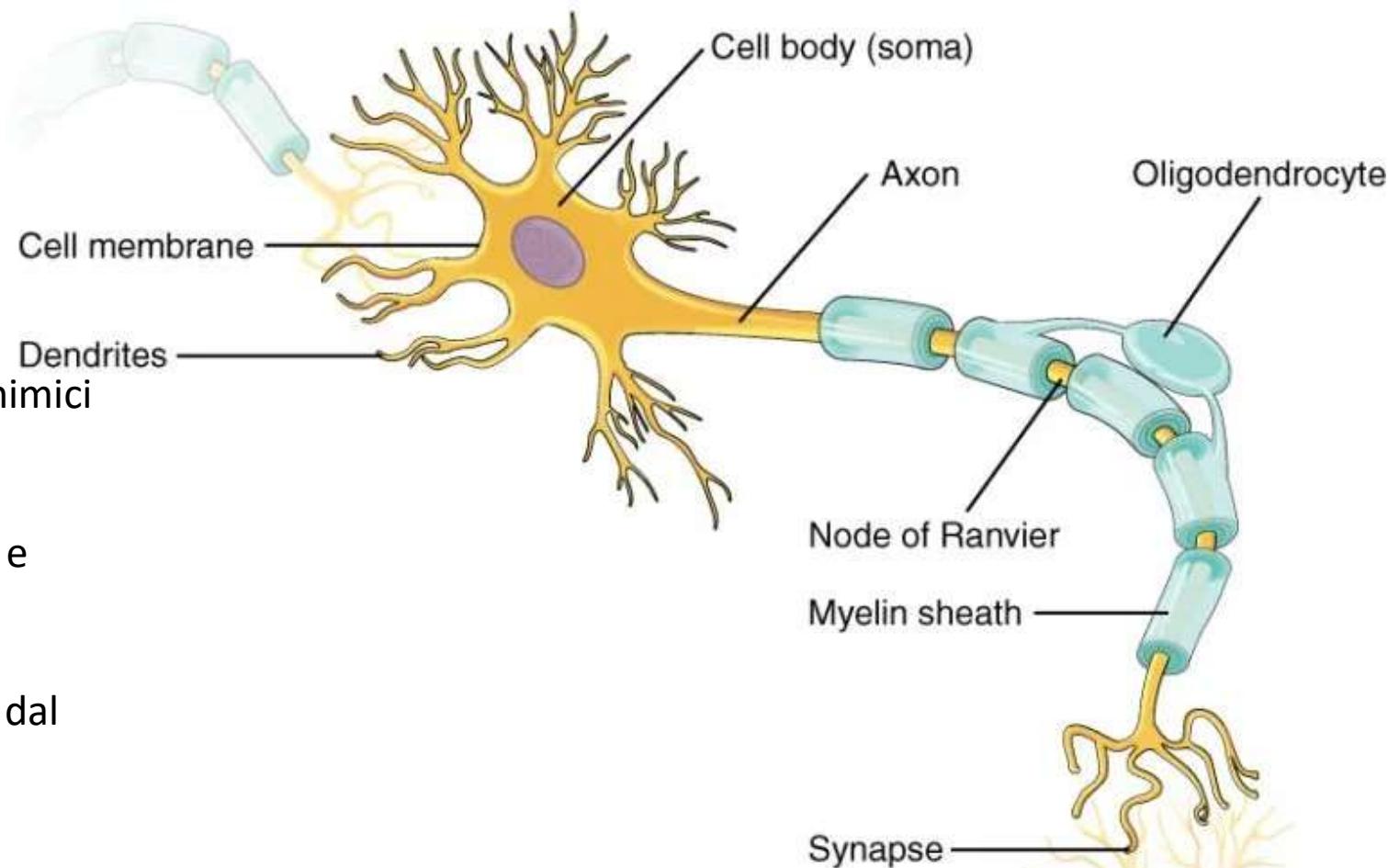
1. Combina le idee di RMSprop e momento per adattare il learning rate in modo individuale per ciascun parametro.
2. È efficace in diverse situazioni e spesso offre una convergenza più rapida rispetto ad altri metodi di ottimizzazione. Tuttavia, può richiedere una scelta accurata dei suoi iperparametri per ottenere prestazioni ottimali.

Training di una rete neurale

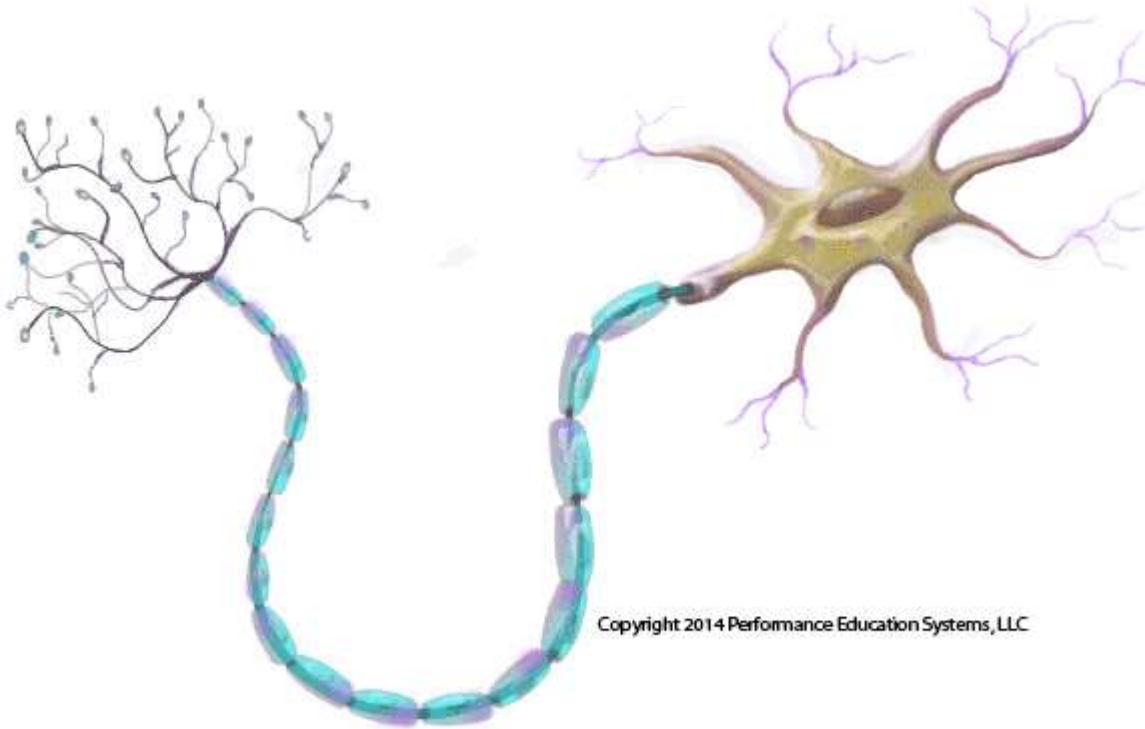
Il neurone Biologico

Il neurone biologico è composto da 4 parti principali:

- il **corpo cellulare o soma**,
- le **estensioni delle cellule, dendriti**
- un'ulteriore estensione, **assone**
- le **sinapsi**
- I **dendriti** ricevono (*input*) segnali elettrici e chimici dagli altri neuroni e li trasmettono al **soma**.
- il **soma** elabora i segnali in ingresso nel tempo e converte il valore elaborato in un **output**
- **L'assone**, invece, trasmette il segnale elettrico dal soma ad altri neuroni o a cellule muscolari o ghiandolari.
- All'estremità dell'assone ci sono le **sinapsi, che collegano il neurone ad** altri neuroni per trasmettere il segnale in uscita



Ogni singola **sinapsi può modificare la propria risposta e variare, in questo modo, l'efficienza di trasporto dell'informazione.**

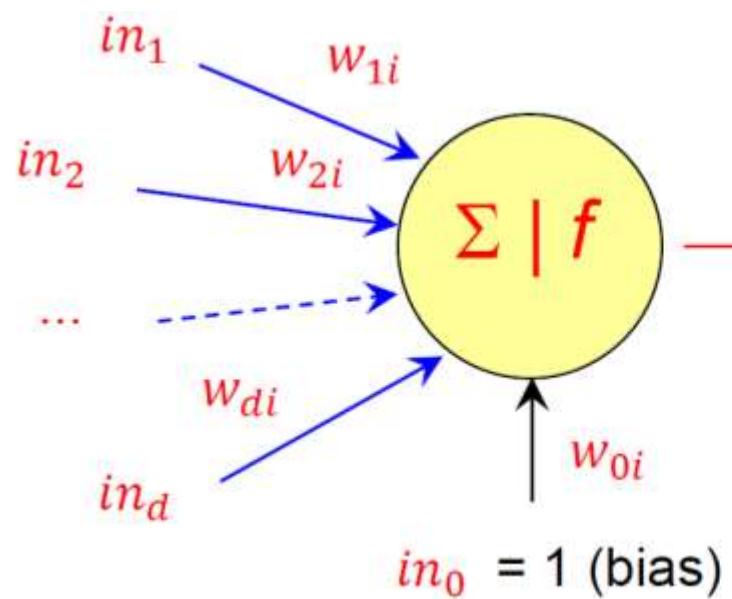


Quando il soma riceve i segnali in ingresso dai dendriti, esegue una “elaborazione”. **Se i segnali ricevuti superano una certa soglia**, viene prodotto un nuovo segnale di uscita sull’assone. Questo segnale si propagherà ad altri neuroni, anche molto distanti tra loro. **Il valore di questa soglia e l’efficienza di trasmissione elettrochimica delle sinapsi sono strettamente legate ai processi di apprendimento.**

Neurone Artificiale

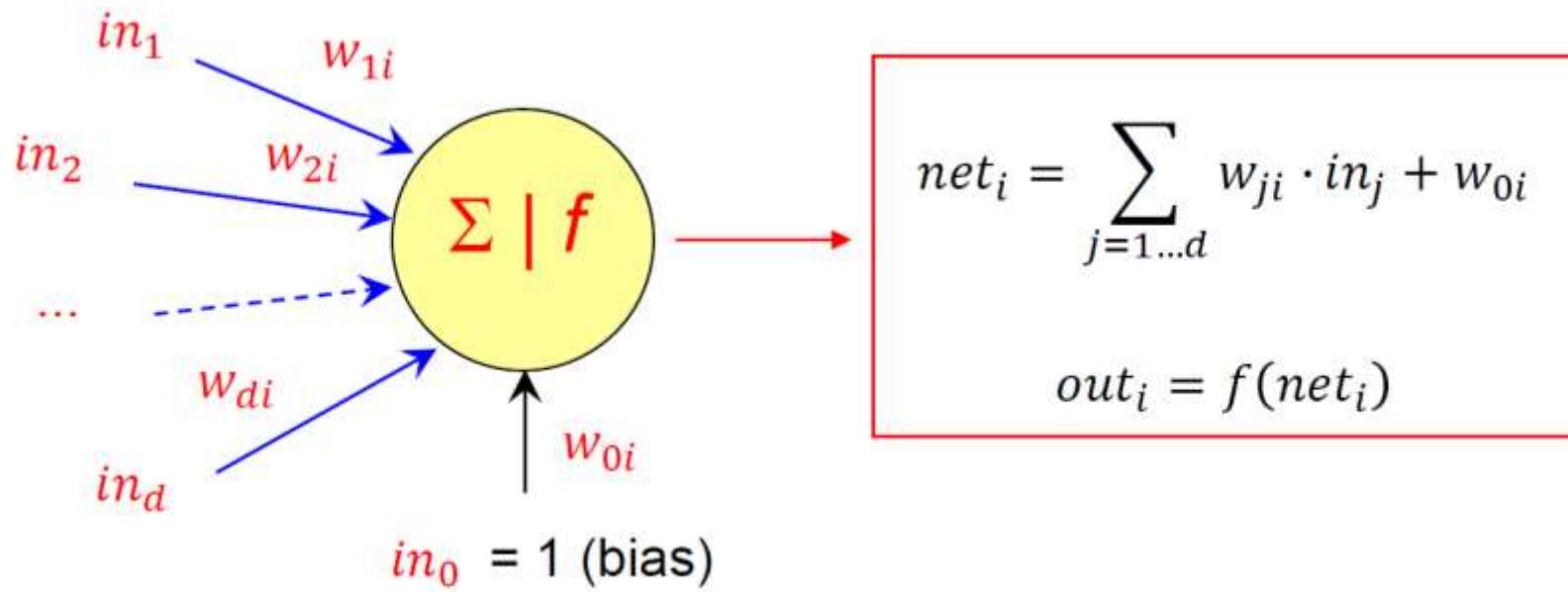
Primo modello del 1943 di McCulloch and Pitts. Con input e output binari era in grado di eseguire computazioni logiche.

Il neurone artificiale i-esimo

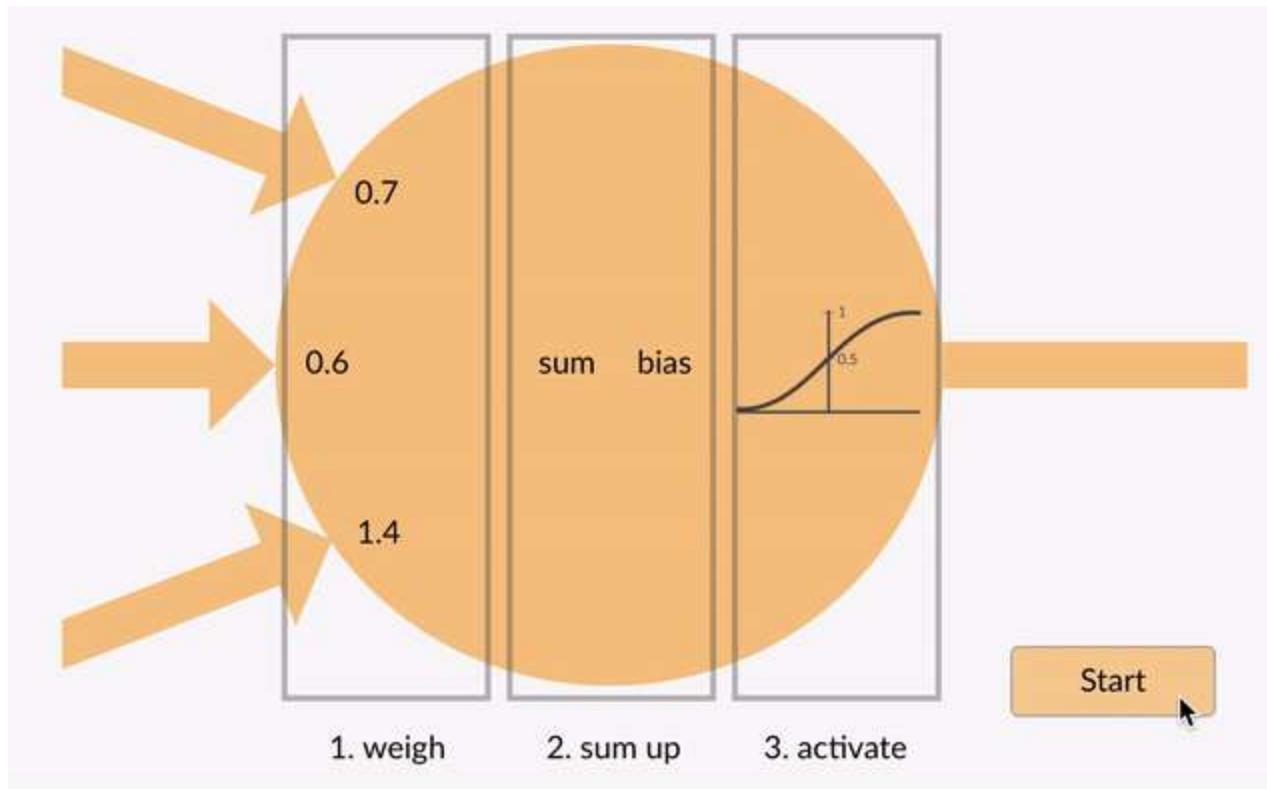


$$net_i = \sum_{j=1 \dots d} w_{ji} \cdot in_j + w_{0i}$$

$$out_i = f(net_i)$$



- in_1, in_2, \dots, in_d sono i d ingressi che il neurone i riceve da assoni di neuroni afferenti
- $w_{1i}, w_{2i}, \dots, w_{di}$ sono i pesi (weight) che determinano l'efficacia delle connessioni sinaptiche dei dendriti (agiremo su questi valori durante l'apprendimento), l'importanza dell'input i -esimo sull'output.
- w_{0i} (detto bias) è un ulteriore peso che si considera collegato a un input fittizio con valore sempre 1 questo peso è utile per «tarare» il punto di lavoro ottimale del neurone.
- net_i è il livello di eccitazione globale del neurone (potenziale interno).
- $f(\cdot)$ è la funzione di attivazione che determina il comportamento del neurone (ovvero il suo output out_i in funzione del suo livello di eccitazione net_i). La funzione di attivazione simula il comportamento del neurone biologico di attivarsi solo se i segnali in ingresso superano una certa soglia.

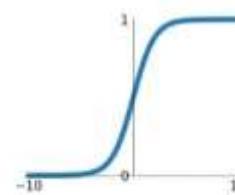


Funzioni di attivazione

Una funzione di attivazione determina se un neurone deve essere **attivato o meno**. Si tratta di alcune semplici operazioni matematiche per determinare se l'input del neurone alla rete è rilevante o meno nel processo di previsione. Le funzioni di attivazione possono essere di diversi tipi, ma in generale devono essere **non lineari** per consentire alla rete di apprendere relazioni complesse tra le sue variabili di input, e **derivabili**.

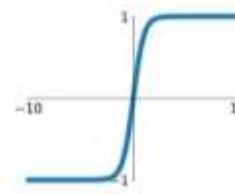
Sigmoid

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



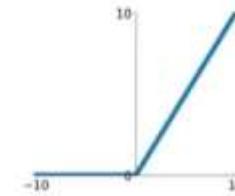
tanh

$$\tanh(x)$$



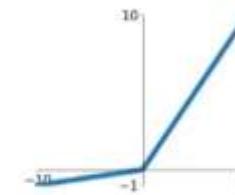
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

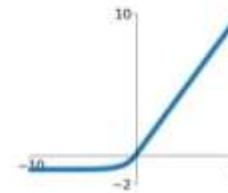


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Reti neurali artificiali (ANN)

- Similmente al cervello, una rete neurale artificiale ANN è costituita da neuroni artificiali collegati tra loro.
- Ogni **connessione** (chiamata **edge**), come le sinapsi in un cervello biologico, può trasmettere un segnale ad altri neuroni.
- Il **peso associato** a ciascuna connessione **aumenta o diminuisce la forza del segnale**.
- Tipicamente, i neuroni sono aggregati in livelli. Diversi livelli possono eseguire diverse trasformazioni sui loro input.
- I segnali viaggiano dal primo livello (il livello di input), all'ultimo livello (il livello di output), possibilmente dopo aver attraversato i livelli più volte.

Tipologie di reti neurali

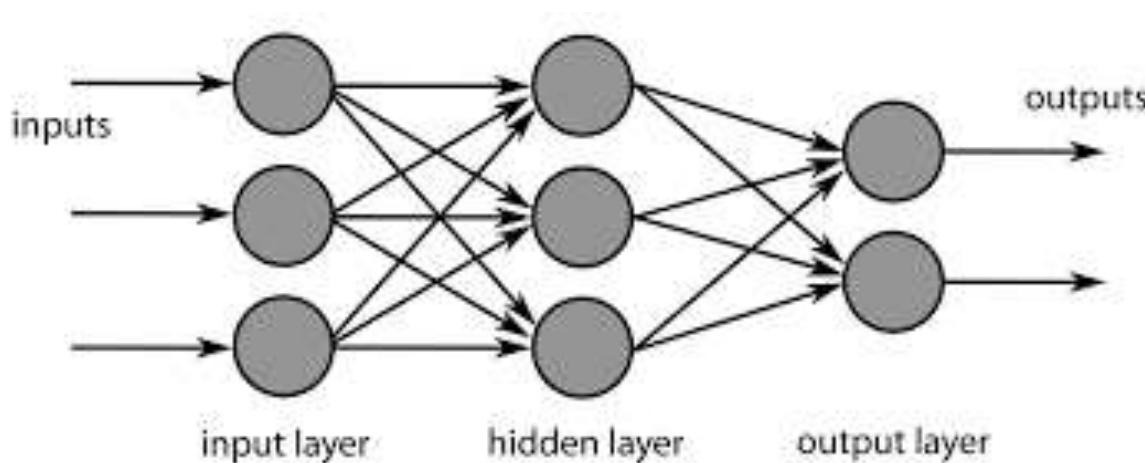
Le reti neurali sono composte da gruppi di neuroni artificiali organizzati in livelli.

Tipicamente sono presenti un livello di input, un livello di output, e uno o più livelli intermedi.

Ogni livello contiene uno o più neuroni. I **layer intermedi** sono chiamati **hidden layer** in quanto restano “invisibili” dall'esterno della rete, la quale si interfaccia all'ambiente solo tramite il layer di ingresso e quello di uscita.

Feedforward (FFNN)

nelle reti feedforward («alimentazione in avanti») le connessioni collegano i neuroni di un livello con i neuroni di un livello successivo. **Non** sono consentite connessioni all'indietro o connessioni verso lo stesso livello. È di gran lunga il tipo di rete più utilizzata



Multi Layer Perceptron (MLP)

Multi Layer Perceptron (MLP) è l'FFNN più comune costituito da tre o più layer (strati):

- un layer di input
- uno o più layer nascosti
- un layer di output

Le MLP sono fully-connected: ogni neurone in uno strato è connesso con ogni neurone nello strato successivo

Un teorema noto **come universal approximation theorem** asserisce che ogni funzione continua che mappa intervalli di numeri reali su un intervallo di numeri reali può essere approssimata da un MLP con un solo hidden layer. Questa è una delle motivazioni per cui per molti decenni (fino all'esplosione del deep learning) ci si è soffermati su reti neurali a 3 livelli.

Training di una rete neurale

Il processo di apprendimento è una caratteristica chiave delle ANN ed è strettamente correlato al modo in cui il cervello umano apprende:

- eseguiamo un'azione
- siamo approvati o corretti da un trainer o coach per migliorare in un determinato compito.

Iterativamente

- i dati di addestramento vengono presentati alla rete (**forward**),
- quindi i pesi vengono aggiustati (**backward**) sulla base di quanto sono simili i valori restituiti dalla rete rispetto a quelli desiderati (**loss**)
- Dopo che tutti i casi sono stati presentati, il processo spesso ricomincia da capo

Durante la fase di apprendimento, i pesi vengono regolati per migliorare la performance sui dati di allenamento

Forward propagation

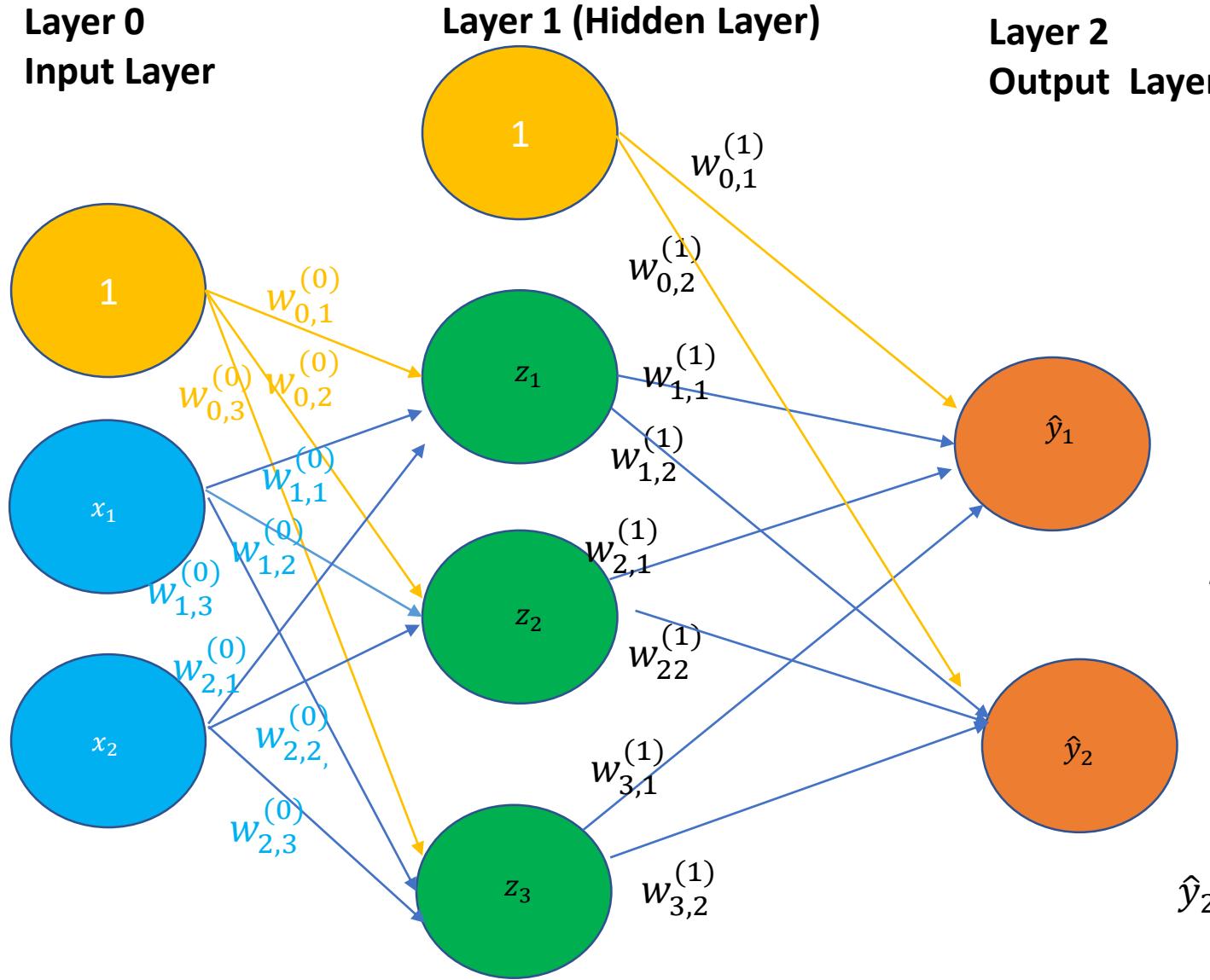
- Con forward propagation (o inference) si intende la propagazione delle informazioni in avanti dal livello di input a quello di output.
- Una volta addestrata, una rete neurale può semplicemente
- processare pattern attraverso forward propagation

Forward Propagation

nell'esempio una rete a 3 livelli ***d:nH:s***

input *d=2* neuroni, livello nascosto **hidden *nH=3*** neuroni, **output *s=1* neuroni**

Layer 0
Input Layer



$w_{j,i}^{(h-1)}$ = peso che l'input del *j*-esimo neurone del layer $h - 1$ ha sull'*i*-esimo neurone del layer h

$$\hat{y}_1 = f\left(\sum_{j=1}^{nH} w_{j,1}^{(1)} z_j + w_{0,1}^{(1)}\right)$$

$$z_j = \sum_{i=1}^d w_{i,j}^{(0)} x_i + w_{0,j}^{(0)}$$

$$\hat{y}_1 = f\left(\sum_{j=1}^{nH} w_{j,i}^{(1)} f\left(\sum_{i=1}^d w_{i,j}^{(0)} x_i + w_{0,j}^{(0)}\right) + w_{0,1}^{(1)}\right)$$

$$\hat{y}_2 = f\left(\sum_{j=1}^{nH} w_{j,2}^{(1)} z_j + w_{0,2}^{(1)}\right)$$

$$\hat{y}_2 = f\left(\sum_{j=1}^{nH} w_{j,2}^{(1)} f\left(\sum_{i=1}^d w_{i,j}^{(0)} x_i + w_{0,j}^{(0)}\right) + w_{0,2}^{(1)}\right)$$

In generale data una rete MLP con layer di input formato da d nodi, un layer nascosto formato da nH neuroni ed un layer di output formato da s nodi, l'espressione per l'output \hat{y}_k , $k = 0, \dots, s$ è data dalla seguente espressione:

$$\hat{y}_k = f\left(\sum_{j=1}^{nH} w_{j,k}^{(1)} z_j + w_{0,k}^{(1)}\right) = f\left(\sum_{j=1}^{nH} w_{j,k}^{(1)} \cdot f\left(\sum_{i=1}^d w_{i,j}^{(0)} x_i + w_{0,j}^{(0)}\right) + w_{0,k}^{(1)}\right)$$

MLP: Training

- Fissata la topologia (numero di livelli e neuroni), l'addestramento supervisionato di una rete neurale consiste nel determinare il valore dei pesi w che determinano il mapping desiderato tra input e output

Loss Function

- La loss function, o funzione di perdita, è **una misura dell'errore della previsione prodotta da un modello di machine learning rispetto ai dati di training**. Essa rappresenta la **discrepanza tra l'output previsto dal modello e l'output reale associato ai dati di training**.
- L'obiettivo del modello di machine learning è quello di minimizzare la loss function, ossia di trovare i valori dei parametri del modello che producono la previsione migliore possibile sui dati di training.
- In pratica, la scelta della loss function dipende dal tipo di problema di machine learning che si vuole risolvere. Ad esempio, se si sta risolvendo un **problema di classificazione binaria**, la loss function più comune è la funzione di entropia incrociata binaria (**binary cross-entropy**), mentre se si sta risolvendo un problema di **classificazione multiclasse**, la loss function più comune è la funzione di **entropia incrociata categorica** (categorical cross-entropy).
- In generale, la scelta della loss function può influenzare significativamente le prestazioni del modello di machine learning, e pertanto è una scelta importante da fare durante la progettazione del modello.

Una **loss function** è per un **singolo esempio di addestramento**

Una **cost function** è la **perdita media sull'intero set di dati di addestramento**. Le strategie di ottimizzazione mirano a **minimizzare la funzione di costo**.

Generalmente, **la funzione di costo** viene calcolata come

$$C = \frac{\sum_{i=1}^n L(y_i, \hat{y}_i)}{n}$$

dove

- n è il numero di esempi di training
- y_i è l'osservazione effettiva dell'esempio di training i-esimo
- \hat{y}_i la previsione dell'esempio di training i-esimo

L è la loss-function

- Nel caso di regressione, la **cost function** più comune è l'errore quadratico medio (**mean squared error o MSE**). L'errore quadratico medio è definito come la media dei quadrati delle differenze tra l'output previsto dal modello e l'output reale associato ai dati di training. In altre parole, l'MSE è calcolato come:

$$\bullet \quad C(W) = \frac{\sum_{i=1}^n (y_i - \hat{y}_i(W))^2}{n}$$

- dove n è il numero di esempi di training, y_i è l'**output reale associato a ciascun esempio di training** e $\hat{y}_i(W)$, che dipende dai parametri della rete che indichiamo con W , è l'output previsto dal modello per l'input corrispondente.
- Esistono anche altre cost function utilizzate in problemi di regressione, come ad esempio la cost function **di errore assoluto medio** (**mean absolute error o MAE**).
- $$\bullet \quad C(W) = \frac{\sum_{i=1}^n |y_i - \hat{y}_i(W)|}{n}$$

In generale, consideriamo un esempio di training costituito dai dati etichettati $x = [x_1, x_2, \dots, x_d]$ (dati etichettati) e $y = [y_1, y_2, \dots, y_d]$ (etichette), e sia $\hat{y} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_s]$ l'output prodotto dalla rete ([forward propagation](#)).

Scegliamo come loss function la somma dei quadrati degli errori per l'esempio (x, y) :

$$C(W, x) = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i(W))^2 = \frac{1}{2} \|y - \hat{y}\|_2^2$$

che quantifica quanto l'output prodotto per l'esempio (x, y) si discosta da quello desiderato. La dipendenza dai pesi w è implicita in \hat{y} . L'errore $C(w)$ sull'intero training set è la media di $C(w, x)$ su tutti i gli esempi del training set.

Caso MLP: rete a **3** livelli ***d:nH:s***

$$\hat{y}_k = f \left(\sum_{j=1}^{n_H} w_{j,k}^{(1)} z_j + w_{0,k}^{(1)} \right) = f \left(\sum_{j=1}^{n_H} w_{j,k}^{(1)} \cdot f \left(\sum_{i=1}^d w_{i,j}^{(0)} x_i + w_{0,j}^{(0)} \right) + w_{0,k}^{(1)} \right) \quad k = 1, \dots, s$$

Derivata di una funzione composta

Chain rule

- Se $g: R \rightarrow R$ e $f: R \rightarrow R$ sono derivabili, allora $g \circ f : R \rightarrow R$ è differenziabile, e se poniamo $\mathbf{h}(x) = g(f(x))$
 - $\frac{dh}{dx} = g'(f(x)) \cdot f'(x)$
-
- Se $q: R \rightarrow R$ e $g: R \rightarrow R$ e $f: R \rightarrow R$ sono derivabili, allora $q \circ g \circ f: R \rightarrow R$ è derivabile,
- $$\mathbf{h}(x) = q(g(f(x)))$$
- $\frac{dh}{dx} = q'(g(f(x)) \cdot g'(f(x)) \cdot f'(x))$

Derivata composta di funzioni di più variabili reali:

- Se $x(t) = (x_1(t), x_2(t), \dots, x_n(t))$ è un vettore di R^n le cui componenti sono funzioni derivabili e se f è una funzione differenziabile in $x(t)$, allora la funzione composta $F(t) = f(x(t))$ è differenziabile nella variabile t e si ha:

$$F'(t) = \sum_{i=1}^n \frac{\partial f(x(t))}{\partial x_i} \cdot x'_i(t) = \langle \nabla f(x(t)), x'(t) \rangle$$

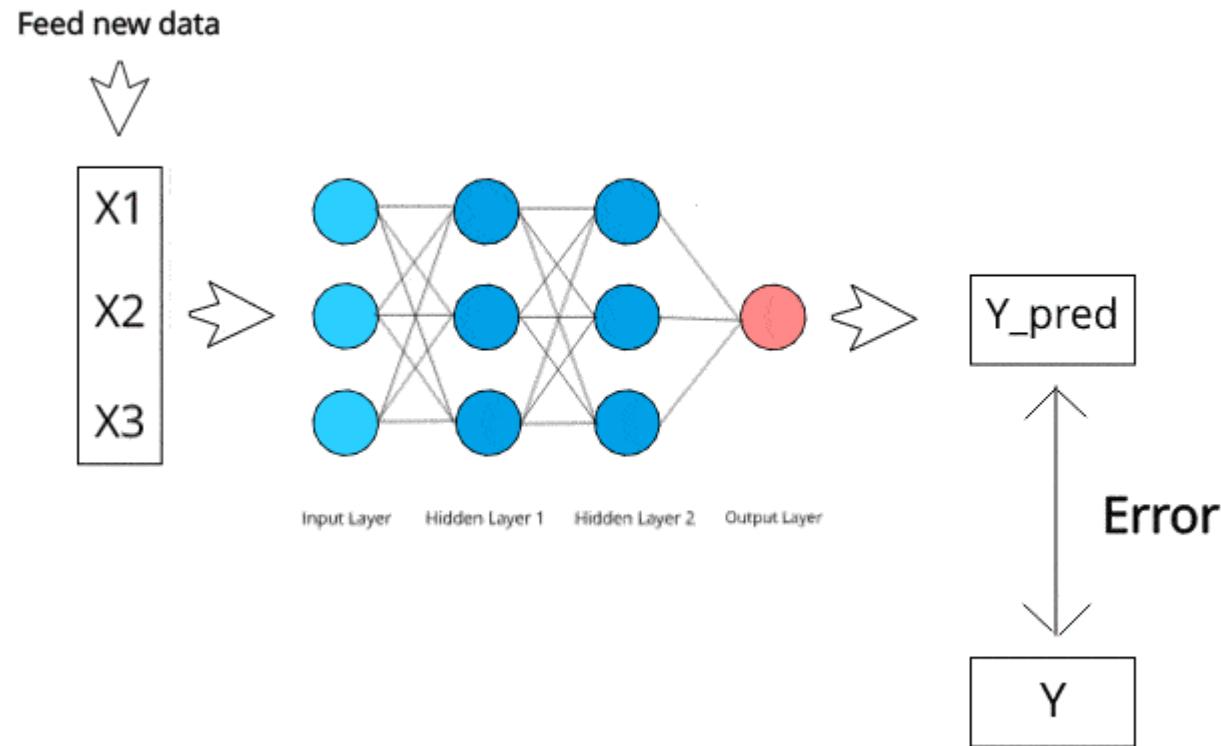
- Esempio: data la funzione $f(h(t), g(t))$, la derivata di f rispetto a t si calcola come:

$$\frac{df}{dt} = \frac{df}{dh} \frac{dh}{dt} + \frac{df}{dg} \frac{dg}{dt}$$

nell'esempio una rete a **4 livelli** $d:nH_1:nH_2:s$

input $d=3$ neuroni, **livello nascosto 1 $nH_1 =3$ neuroni**,
livello nascosto 2 $nH_2 =3$ neuroni, **output $s =1$ neurone**.

In questo esempio viene trascurato il bias.



Il numero totale di pesi (o parametri), in questo caso, senza tener conto del bias è dato da:
è $d \times nH_1 + nH_1 \times nH_2 + nH_2 \times s = 3 \times 3 + 3 \times 3 + 3 \times 1 = 21$