unimc
UNIVERSITÀ DI MACERATA
l'umanesimo che innova

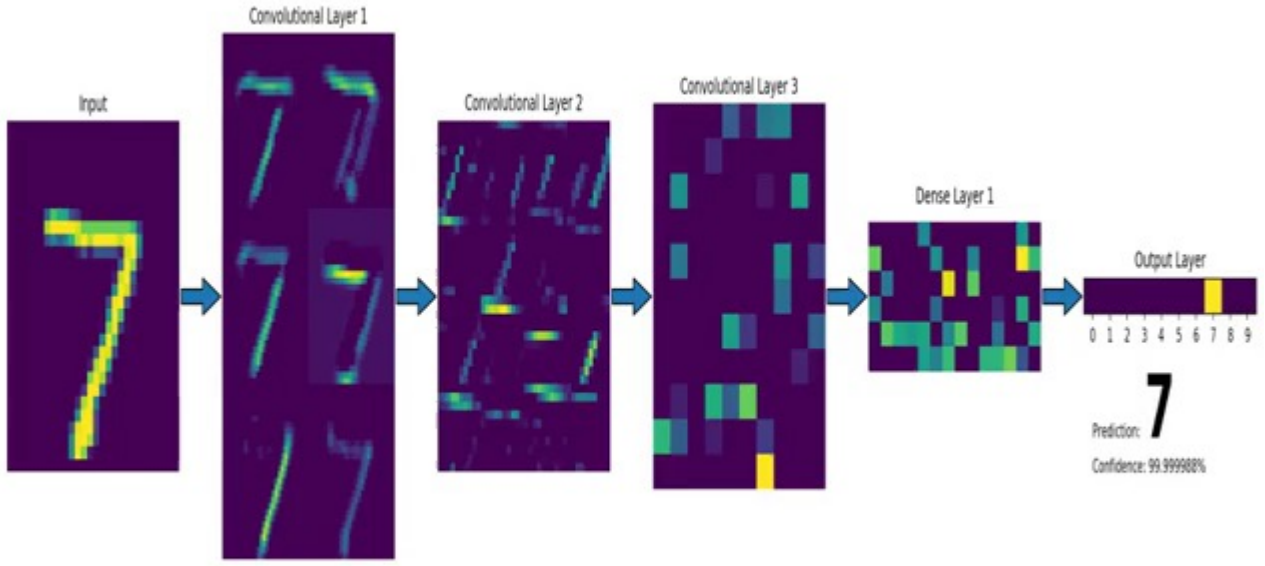
Introduzione al Deep Learning

Corso di LAUREA MAGISTRALE (Classe LM-19)
Comunicazione e Culture Digitali

DIPARTIMENTO DI
SCIENZE POLITICHE,
DELLA COMUNICAZIONE,
E DELLE RELAZIONI INTERNAZIONALI

Deep Learning e Deep Neural Networks

Il termine "deep" in Deep Learning e Deep Neural Network, si riferisce alla presenza di molteplici strati in cascata ottenendo rappresentazione dei dati di input a livelli sempre più astratti.



Title

Yann LeCun, Geoffrey Hinton And Yoshua Bengio Received The Turing Award, The Nobel Prize Of Computing

Reti Neurali

- Introduzione
- Neuroni Biologici
- Neuroni Artificiali
- Tipologie di Reti

- Multilayer Perceptron (MLP)
- Forward propagation
- Training: Error Backpropagation
- On-line Backpropagation
- Stochastic Gradient Descent (SGD)

- Approfondimenti
- Softmax e Cross-Entropy
- Regolarizzazione (Weight Decay) Momentum
- Learning Rate adattivo

Calcolatori vs Reti Neurali

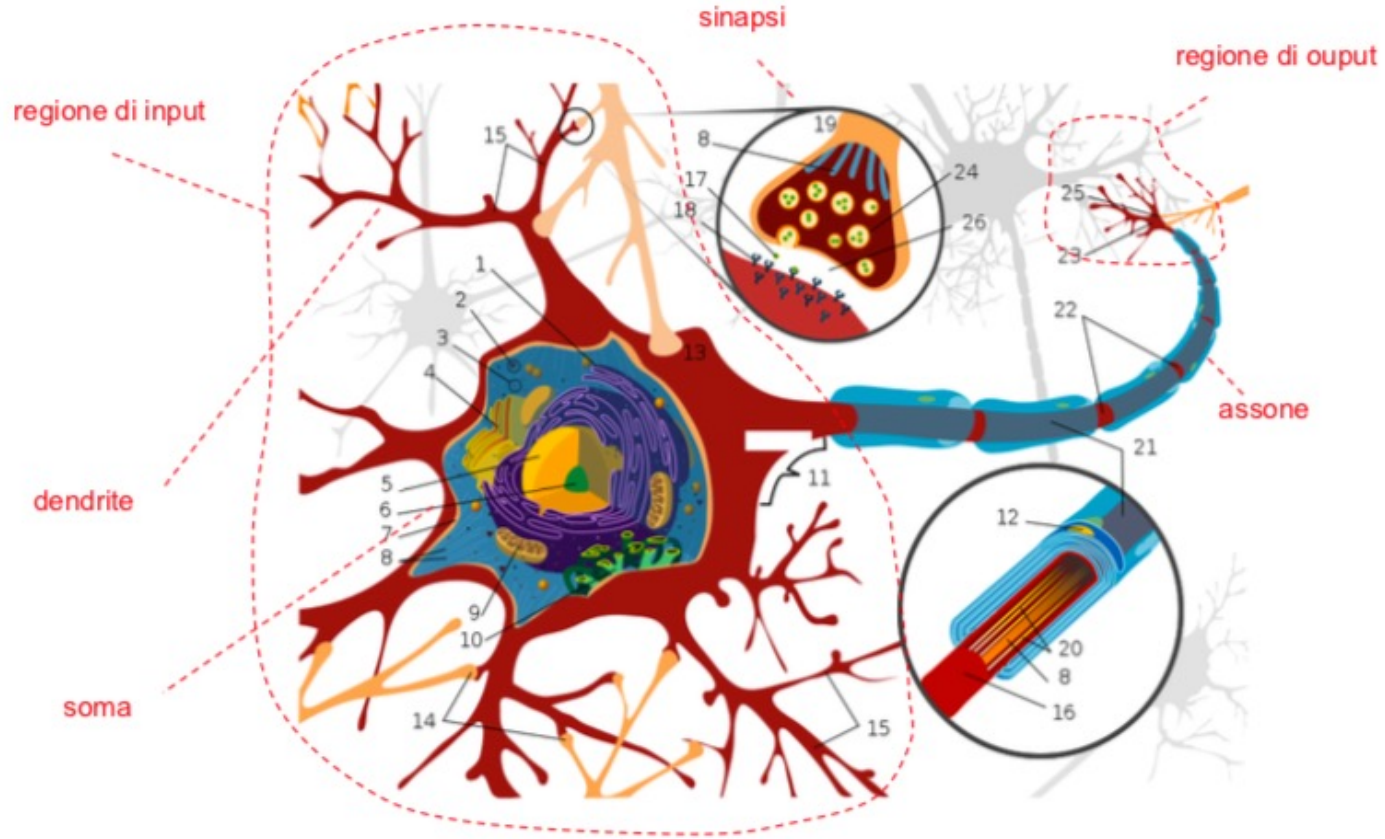
Circuiti logici tradizionali (Microprocessori)

- tempi di elaborazione (~ ns)
- esecuzione delle operazioni in maniera strettamente sequenziale (accesso sequenziale alla memoria)

Rete neurale biologica (Cervello Umano)

- elevato numero di cellule neurali (~ 10^{11})
- tempi di elaborazione (~ ms)
- elevata interconnessione (~ 10^{4})
- elaborazione parallela (memoria distribuita)

Neuroni Biologici



Neuroni Biologici

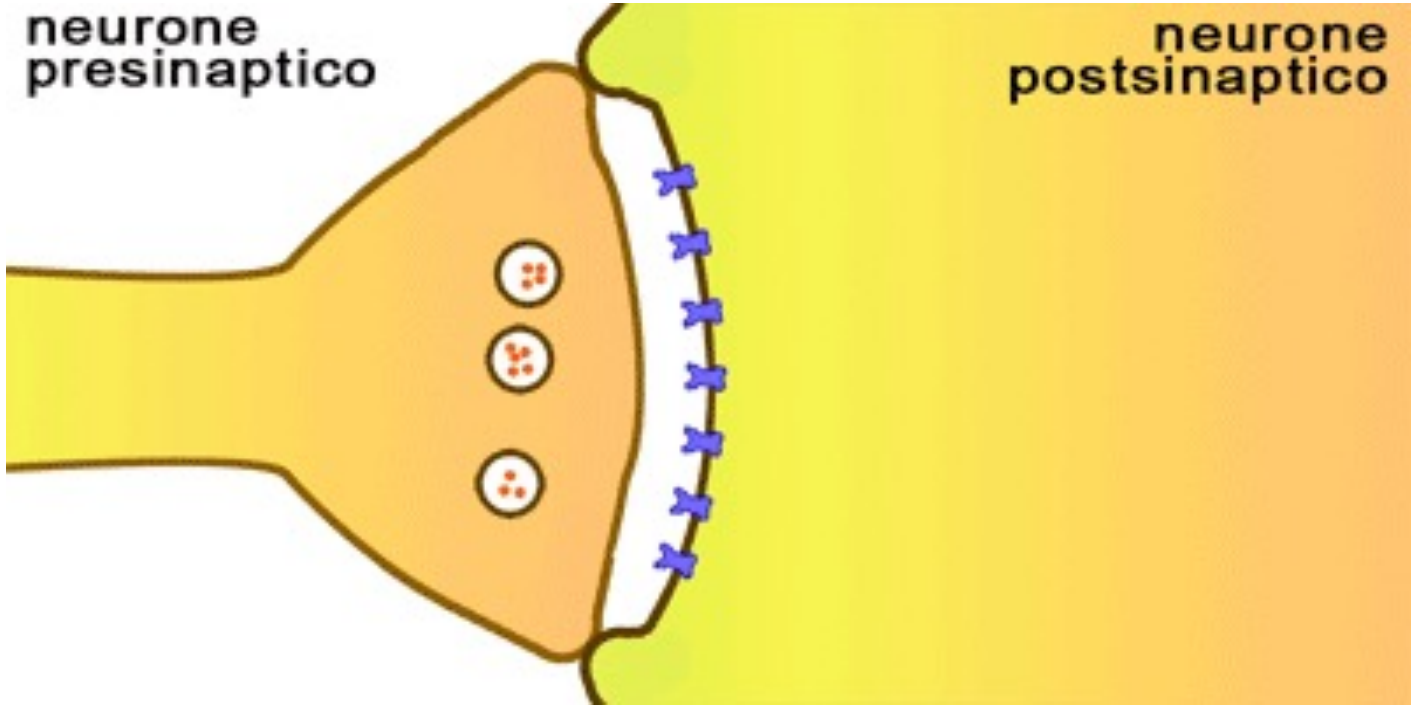
- Le connessioni sinaptiche o (sinapsi) agiscono come porte di collegamento per il passaggio dell'informazione tra neuroni.

- I dendriti sono fibre minori che si ramificano a partire dal corpo cellulare del neurone (detto soma). Attraverso le sinapsi i dendriti raccolgono input da neuroni afferenti e li propagano verso il soma.

- L'assone è la fibra principale che parte dal soma e si allontana da esso per portare ad altri neuroni (anche distanti) l'output.

Neuroni Biologici

- Il passaggio delle informazioni attraverso le sinapsi avviene con processi elettro-chimici.



Neuroni Biologici

- L'ingresso di ioni attraverso le sinapsi dei dendriti determina la formazione di una differenza di potenziale tra il corpo del neurone e l'esterno. Quando questo potenziale supera una certa soglia si produce uno spike (impulso): il neurone propaga un breve segnale elettrico detto potenziale d'azione lungo il proprio assone: questo potenziale determina il rilascio di ioni dalle sinapsi dell'assone.

- Il reweighting delle sinapsi (ovvero la modifica della loro efficacia di trasmissione) è direttamente collegato a processi di apprendimento e memoria in accordo con la regola di Hebb.

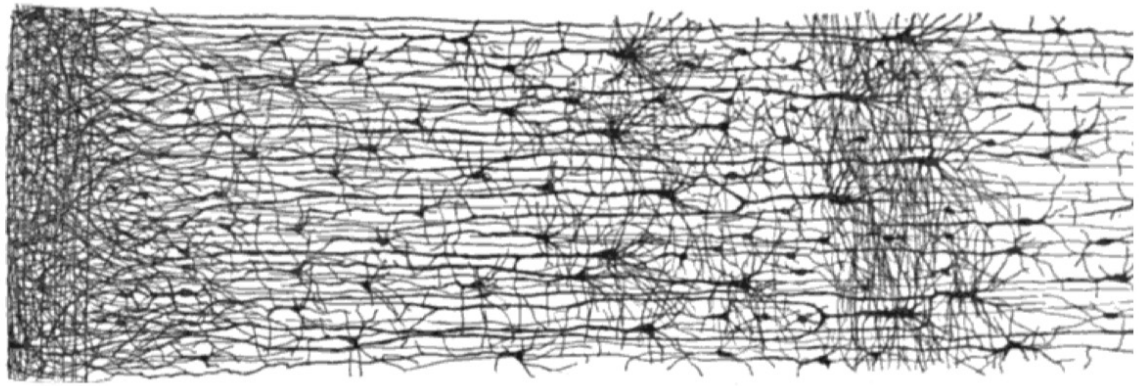
- Hebbian rule: se due neuroni, tra loro connessi da una o più sinapsi, sono ripetutamente attivati simultaneamente allora le sinapsi che li connettono sono rinforzate.

Reti Neurali Biologiche

- Il cervello umano contiene circa 100 miliardi di neuroni ciascuno dei quali connesso con circa altri 1000 neuroni (10^{14} sinapsi).

- La corteccia celebrale (sede delle funzioni nobili del cervello umano) è uno strato laminare continuo di 2-4 mm, una sorta di lenzuolo che avvolge il nostro cervello formando numerose circonvoluzioni per acquisire maggiore superficie. Sebbene i neuroni siano disposti in modo «abbastanza» ordinato in livelli consecutivi, l'intreccio di dendriti e assoni ricorda una foresta fitta e impenetrabile.

Reti Neurali Biologiche



NO ROAD, NO trail can penetrate this forest. The long and delicate branches of its trees lie everywhere, choking space with their exuberant growth. No sunbeam can fly a path tortuous enough to navigate the narrow spaces between these entangled branches. All the trees of this dark forest grew from 100 billion seeds planted together. And, all in one day, every tree is destined to die.

[Sebastian Seung, 2012:
Connectome -
How the Brain's Wiring
Make Us Who We Are]

Connectome

- Il connectome è la mappa delle connessioni dei neuroni nel cervello.

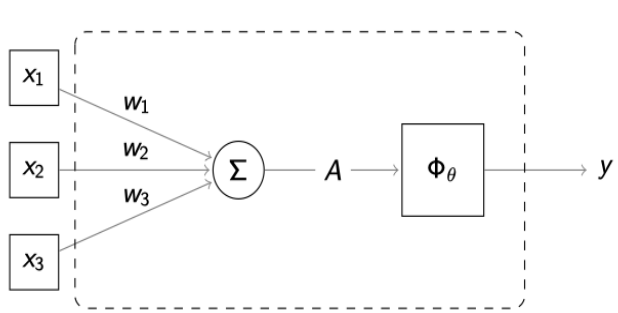
- Il suo sviluppo pre-natale è guidato dal DNA:
- I geni sono responsabili della formazione dei neuroni e del loro posizionamento (migrazione).
- Una volta giunti in posizione i neuroni iniziano a estendere le loro ramificazioni (dendriti e assone) e a formare sinapsi. Questi processi sono parzialmente guidati dai geni e in parte random.
- Alla nascita la creazione e la migrazione sono di fatto completate (solo in poche aree del cervello nuovi neuroni possono essere creati dopo la nascita), mentre la creazione di nuove ramificazioni e sinapsi prosegue ben oltre.

Connectome

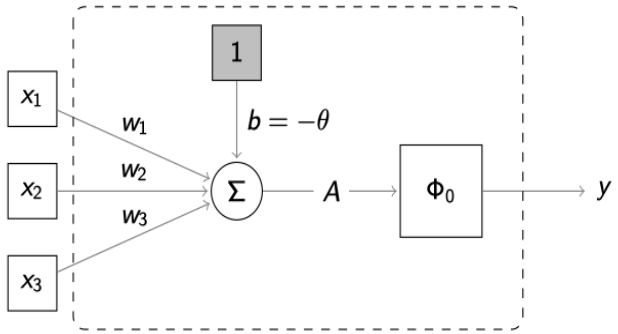
- Lo sviluppo post-natale (guidato dal DNA e dall'apprendimento).
- La rete non è totalmente connessa e le sinapsi non sono create «on-demand» durante i processi di apprendimento.
- I meccanismi di reconnection non sono completamente noti ma una delle teorie più accreditate è il Neural Darwinism: ipotizza che nuove sinapsi siano continuamente create random tra neuroni vicini. Quelle successivamente potenziate dal reweighting Hebbiano rimangono attive (contribuendo alla formazione di memorie a lungo termine), mentre quelle non usate sono via via eliminate. I dendriti che perdono gran parte di sinapsi vengono anch'essi eliminati.
- La velocità di formazione di nuove sinapsi è strabiliante nei neonati (oltre mezzo milione al secondo tra i 2 e 4 mesi di età), ma prosegue anche in età adulta.

Neurone Artificiale

- Primo modello del 1943 di McCulloch and Pitts. Con input e output binari era in grado di eseguire computazioni logiche.
- Nel seguito un neurone (moderno) di indice i: una rete neurale ne contiene molti, dobbiamo distinguerli ...



Modello con soglia



Modello con bias

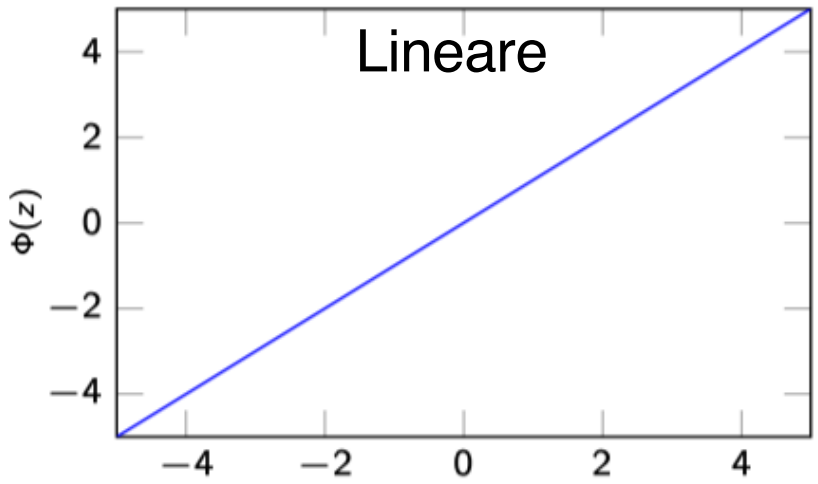
- somma ponderata degli ingressi
- funzione di attivazione a soglia
- possibilità di interpretare la soglia come il peso associato ad un ulteriore ingresso costante di valore 1

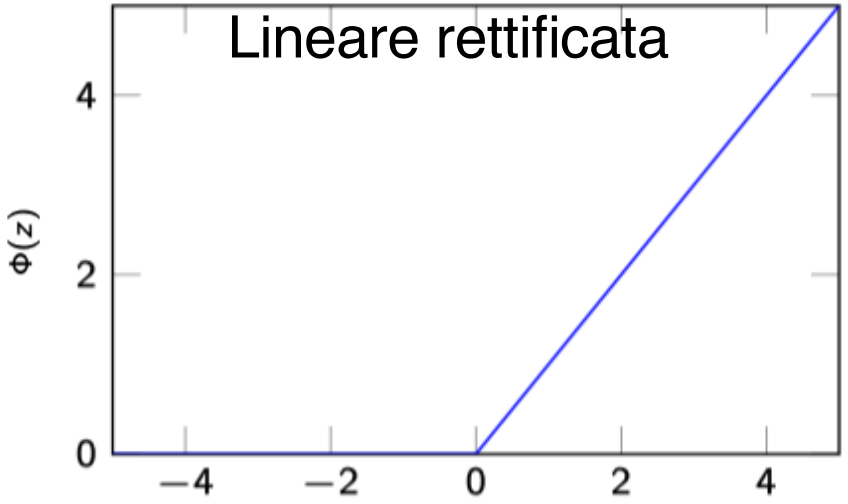
Neurone Artificiale

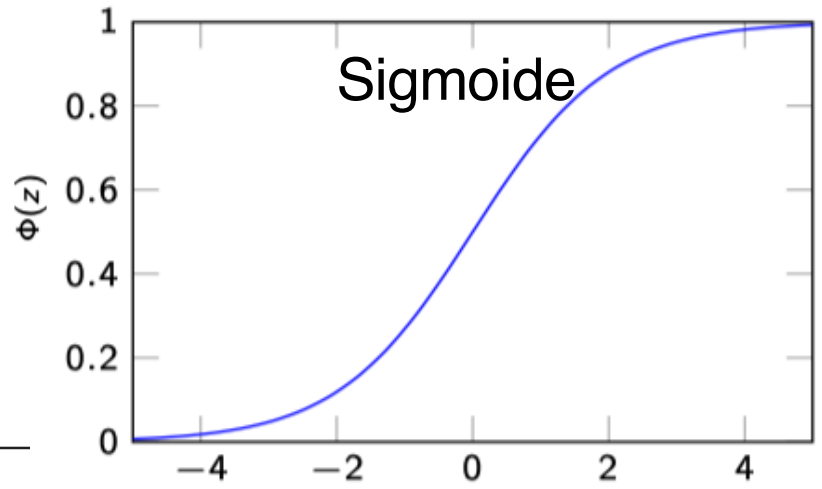
In figura è raffigurato un neurone di indice i: una rete neurale ne contiene molti, dobbiamo distinguerli …

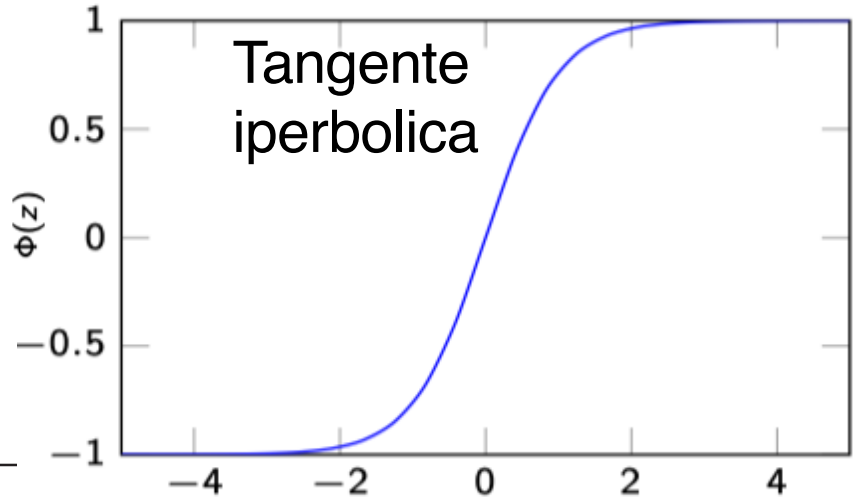
- x_1, x_2, … x_N sono gli N ingressi che il neurone riceve da assoni di neuroni afferenti.
- w_1 , w_2 , … w_N sono i pesi (weight) che determinano l'efficacia delle connessioni sinaptiche dei dendriti (agiremo su questi valori durante l'apprendimento).
- B (detto bias) è un ulteriore peso che si considera collegato a un input fittizio con valore sempre 1; questo peso è utile per «tarare» il punto di lavoro ottimale del neurone.
- A è il livello di eccitazione globale del neurone (potenziale interno);
- ϕ è la funzione di attivazione che determina il comportamento del neurone (ovvero il suo output y) in funzione del suo livello di eccitazione A.

Funzioni di attivazione









Neurone Artificiale

In base alla funzione di attivazione possiamo definire:

- **Percettrone lineare**: se la funzione di attivazione è lineare
- **Percettrone binario**: se la funzione di attivazione è limitata superiormente e inferiormente (sigmoide, tangente iperbolica)

Neurone: funzione di attivazione

- Nei neuroni biologici $f(\cdot)$ è una funzione tutto-niente temporizzata: quando net_i supera una certa soglia, il neurone «spara» uno spike (impulso) per poi tornare a riposo.
- Esistono reti artificiali (denonimate spiking neural network) che modellano questo comportamento e processano l'informazione attraverso treni di impulsi.
- È necessaria questa «complicazione»? Oppure codificare l'informazione con impulsi (invece che con livelli continui) è solo una questione di risparmio energetico messa a punto dall'evoluzione?

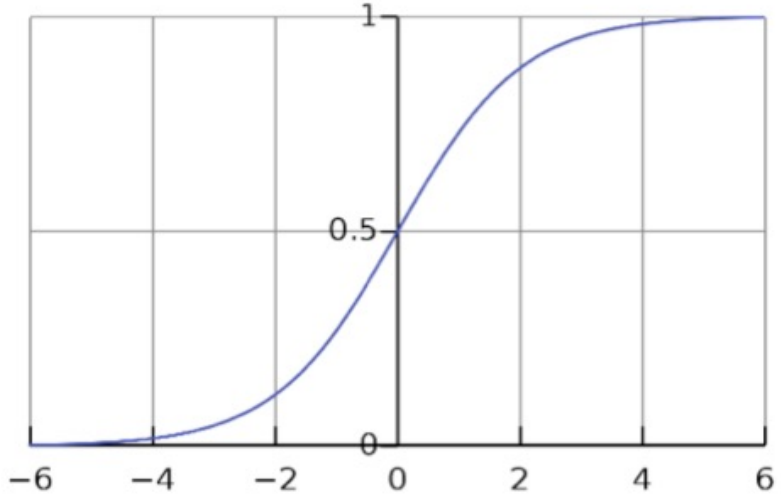
Neurone: funzione di attivazione

- Le reti neurali più comunemente utilizzate operano con livelli continui e $f(\cdot)$ è una funzione non-lineare ma continua e differenziabile (quasi ovunque).
- Perchè non-lineare? Se vogliamo che una rete sia in grado di eseguire un mapping (complesso) dell'informazione di input sono necessarie non-linearità.
- Perché continua e differenziabile? Necessario per la retro-propagazione dell'errore (come scopriremo presto).
- Una delle funzioni di attivazione più comunemente utilizzate è la sigmoide nelle varianti:
- standard logistic function (chiamata semplicemente sigmoid)
- tangente iperbolica (tanh)

Funzione di attivazione: sigmoide

- Standard logistic function (Sigmoid), (valori in [0...1]):

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



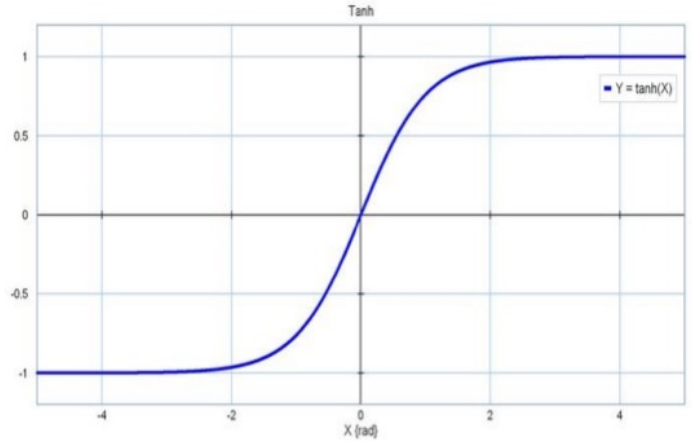
La derivata:

$$\sigma'(net) = \frac{\partial}{\partial net}\left(\frac{1}{1 + e^{-net}}\right) = \frac{e^{-net}}{(1 + e^{-net})^2} = \sigma(net)\big(1 - \sigma(net)\big)$$

Funzione di attivazione: sigmoide

- Tangente iperbolica (Tanh), (valori in [-1...1]): Può essere ottenuta dalla funzione precedente a seguito di trasformazione di scala (×2) e traslazione (-1).

$$f(net) = \tau(net) = 2\sigma(2 \cdot net) - 1$$



La derivata: $\tau'(net) = 1 - \tau(net)^2$

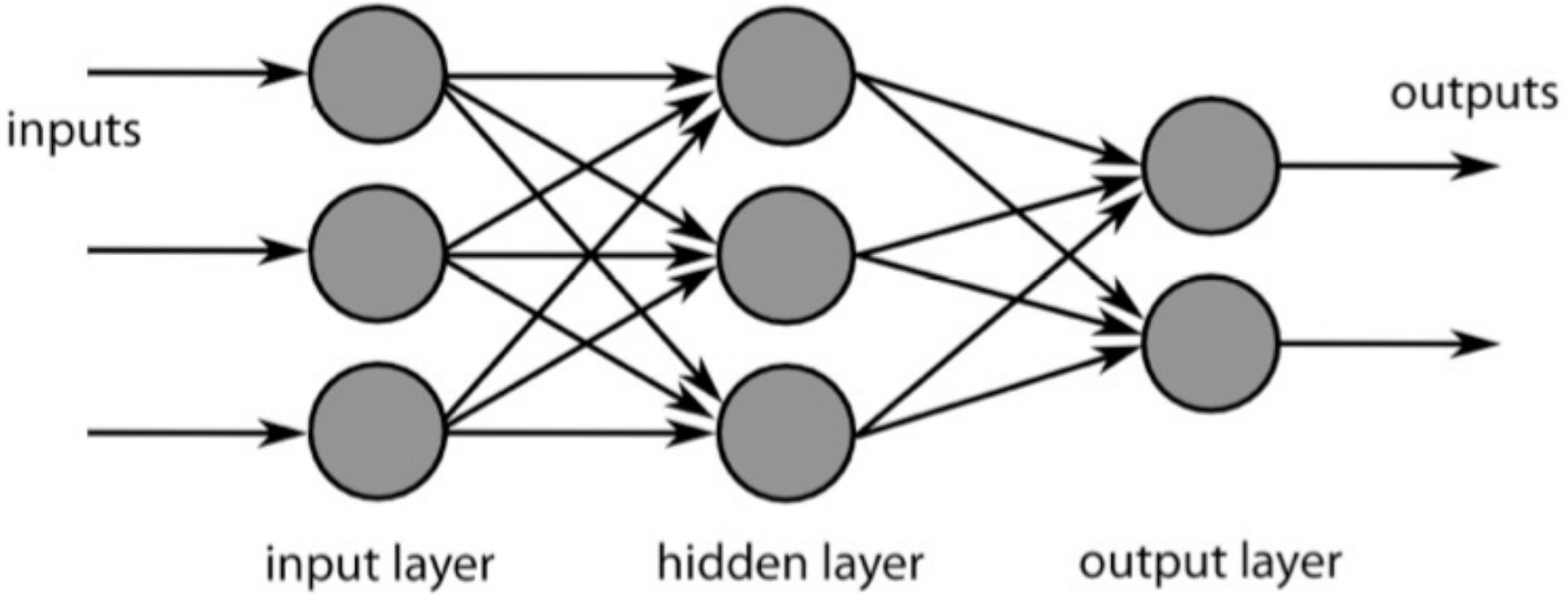
Tanh è leggermente più complessa di sigmoid ma preferibile (convergenza più rapida), in quanto simmetrica rispetto allo zero.

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 o nascosti (hidden). Ogni livello contiene uno o più neuroni.

- Feedforward: 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.

Tipologie di reti neurali



Tipologie di reti neurali

- Recurrent: nelle reti ricorrenti sono previste connessioni di feedback (in genere verso neuroni dello stesso livello, ma anche all'indietro). Questo complica notevolmente il flusso delle informazioni e l'addestramento, richiedendo di considerare il comportamento in più istanti temporali (unfolding in time).

D'altro canto queste reti sono più indicate per la gestione di sequenze (es. audio, video, frasi in linguaggio naturale), perché dotate di un effetto memoria (di breve termine) che al tempo t rende disponibile l'informazione processata a $t - 1$, $t - 2$, ecc.

Un particolare tipo di rete ricorrente è LSTM (Long Short Term Memory).

Reti neurali Artificiali

Sistemi computazionali ispirati a processi neurali biologici con le seguenti caratteristiche:

- tante semplici unità operative (neuroni)
- tante interconnessioni (sinapsi)
- non lineari (mapping complesso dei dati in ingresso)
- adattativi (non programmabili)
- elevato parallelismo (velocità, robustezza e tolleranza ai guasti)
- nessuna distinzione tra area di calcolo e memoria (memoria associativa nelle connessioni, memoria diffusa)

Apprendimento

In generale, la funzione di uscita di una rete neurale dipende da:

1. architettura della rete
2. funzione di attivazione di ciascun neurone
3. pesi dei collegamenti tra i neuroni e bias di ciascun neurone

Tipicamente i primi due sono vincolati a priori. Quindi l'apprendimento consiste nell'aggiustare i pesi in modo che la rete produca le risposte desiderate

Delta Rule

Addestramento del percettrone

Sia $x = [x_1, x_2, ..., x_i, ..., x_n]$ il segnale in ingresso al neurone, y è l'uscita predetta e t il ground truth.

L'errore è $\delta = y - t$

La Delta Rule stabilisce che la variazione del generico peso sia $\Delta w_i = \eta \delta x_i$

dove $\eta \in [0, 1]$ è il tasso di apprendimento (learning rate). Il learning rate determina la rapidità di apprendimento di un neurone.

La Delta Rule modifica i pesi in maniera proporzionale all'errore. $w_i = w_i - \Delta w_i$

Delta Rule

Convergenza

Si dimostra che l'apprendimento (Delta Rule) equivale alla discesa del gradiente lungo la superficie errore nello spazio dei pesi.

Il learning rate rappresenta la rapidità di discesa sulla superficie di E

- η basso → apprendimento troppo lento
- η alto → oscillazioni dell'errore attorno al valore minimo e problemi di convergenza

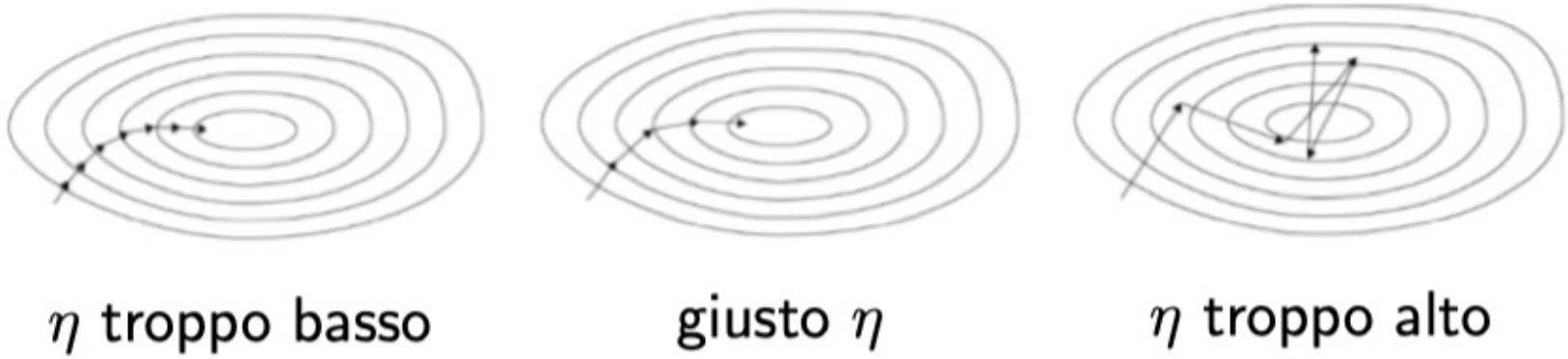
La discesa del gradiente (Gradient descend), se converge, converge

- al minimo globale se le fda sono tutte lineari (ottimo)
- ai minimi locali se ci sono fda non lineari (non ottimo)

Delta Rule

Convergenza

Rappresentazione della discesa del gradiente per spostarsi nello spazio dei pesi lungo la superficie dell'errore.



il passo Δw è proporzionale a η

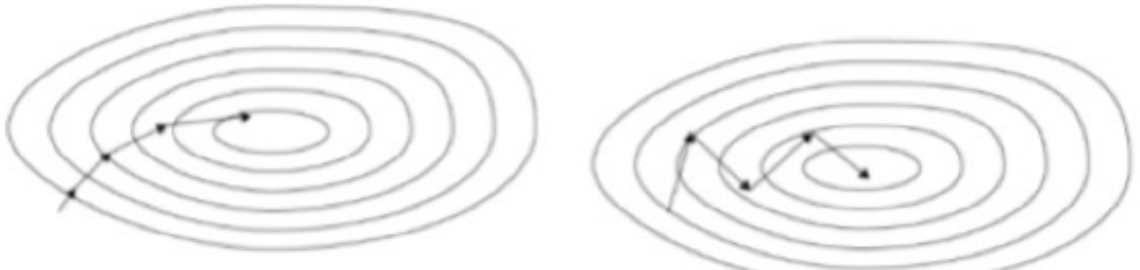
Miglioramento della convergenza

Possibili miglioramenti

introduzione del momento $\alpha \in [0, 1]$

$$\Delta w_i\,(k) \;=\; \eta \delta x i \;+\; \alpha \Delta w_i\,(k \;-\; 1)$$

abbassare η man mano che la rete si addestra



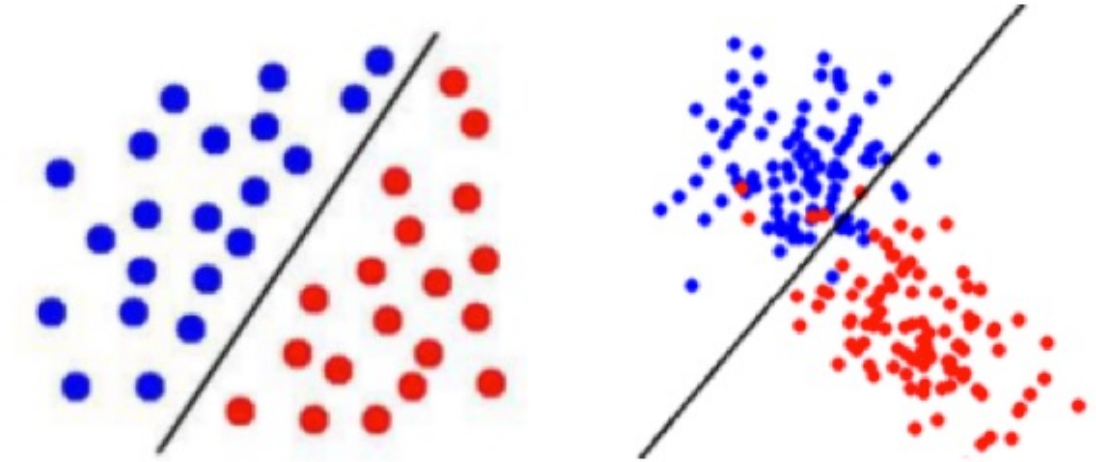
η basso con momento

η alto con momento

il passo Δw è modulato dal momento α per il passo all'iterazione precedente

Problemi lineari e non lineari

Un problema di classificazione nel quale si devono separare in due classi i punti appartenenti ad un certo insieme, si dice lineare se una retta (in uno spazio 2D) o un iperpiano possono separare completamente tutti i punti.



Linearmente separabili

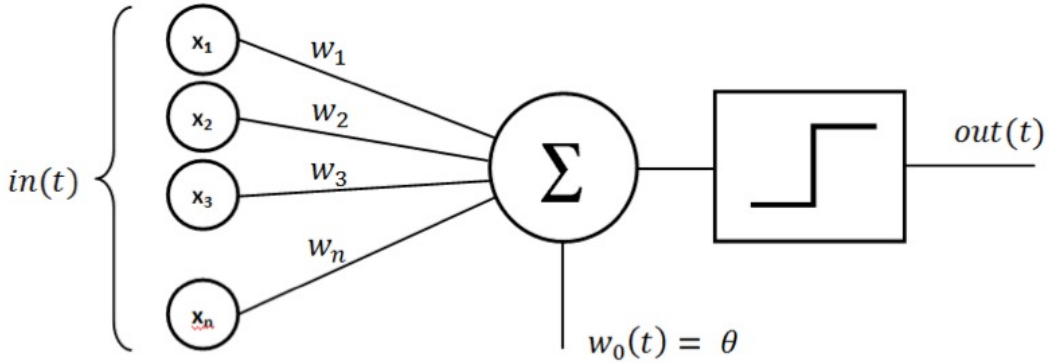
Non linearmente separabili

Problemi lineari e non lineari

- se le classi sono linearmente separabili l'errore di training può convergere a zero
- se le classi non sono linearmente separabili, l'errore può convergere alla massima distanza di separazione tra le classi

Multilayer Perceptron (MLP)

- Il termine perceptron (percettrone) deriva dal modello di neurone proposto da Rosenblatt nel 1956.
- Il percettrone utilizza una funzione di attivazione lineare a soglia (o scalino). Un singolo percettrone, o una rete di percettroni a due soli livelli (input e output), può essere addestrato con una semplice regola detta delta rule (ispirata alla regola di Hebb).



- Una rete a due livelli di percettroni lineari a soglia è in grado di apprendere solo mapping lineari e pertanto il numero di funzioni approssimabili è piuttosto limitato.

Tipo di reti

In base all'organizzazione delle connessioni:

- totalmente connesse
- parzialmente connesse
- a strati

In base al flusso delle informazioni:

- feed-forward
- ricorrenti

Multi Layer Perceptron

Limiti di un percettrone

- modella una retta, piano o iperpiano di dimensione N – 1 nello spazio degli ingressi di dimensione N
- classificatore binario
- applicabile a classi linearmente separabili
- problema dello XOR

Soluzioni

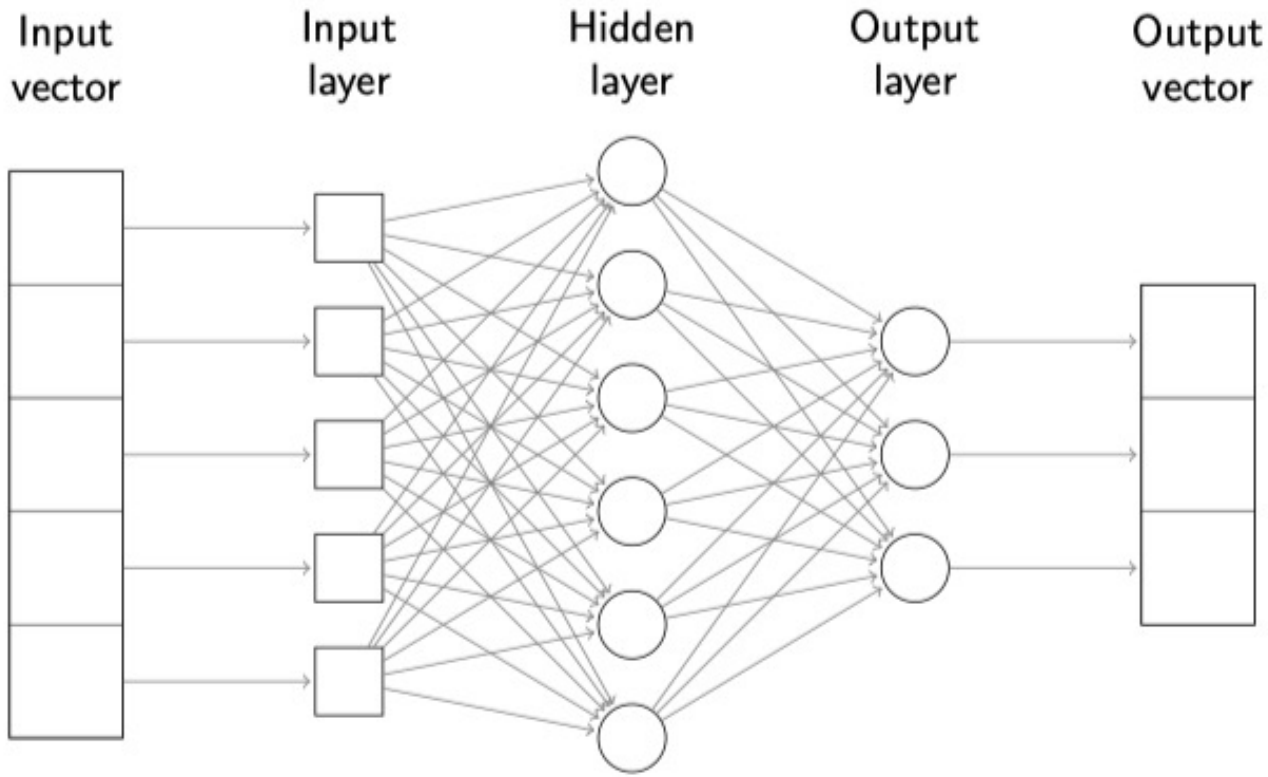
- combinazioni non lineari degli ingressi o fda non lineari
- neuroni disposti a strati (MLP): modellano più iperpiani per ottenere regioni decisionali complesse

Multi Layer Perceptron (MLP)

- Un Multilayer Perceptron (MLP) è una rete feedforward con almeno 3 livelli (almeno 1 hidden) e con funzioni di attivazione non lineari.
- 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. D'altro canto l'esistenza teorica di una soluzione non implica che esista un modo efficace per ottenerla...

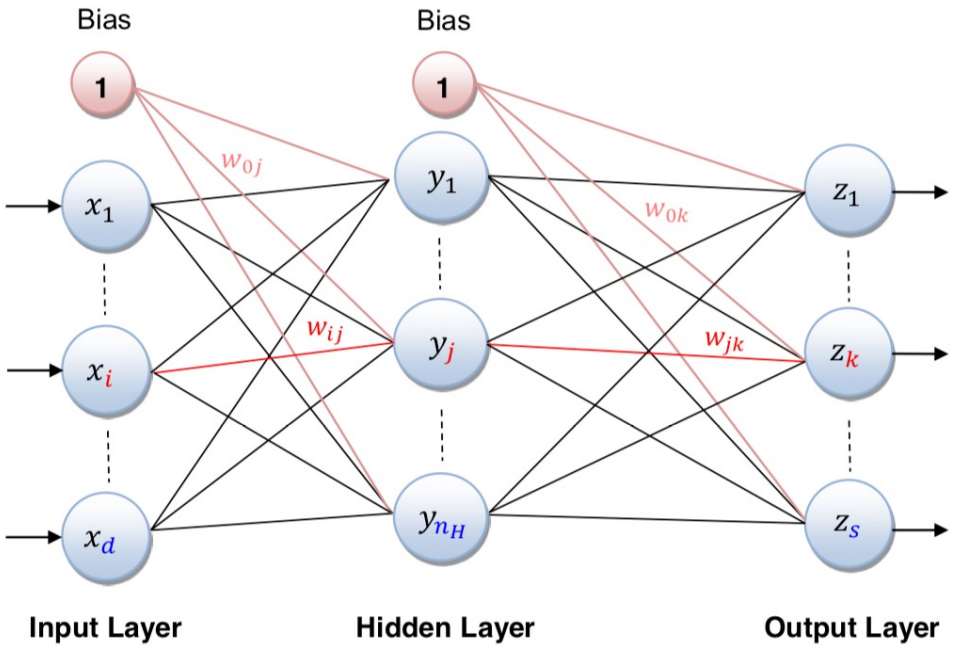
Multi Layer Perceptron

- 1 layer d'ingresso
- 1 layer d'uscita
- 1 o più strati nascosti fully connected layer



MLP: 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.



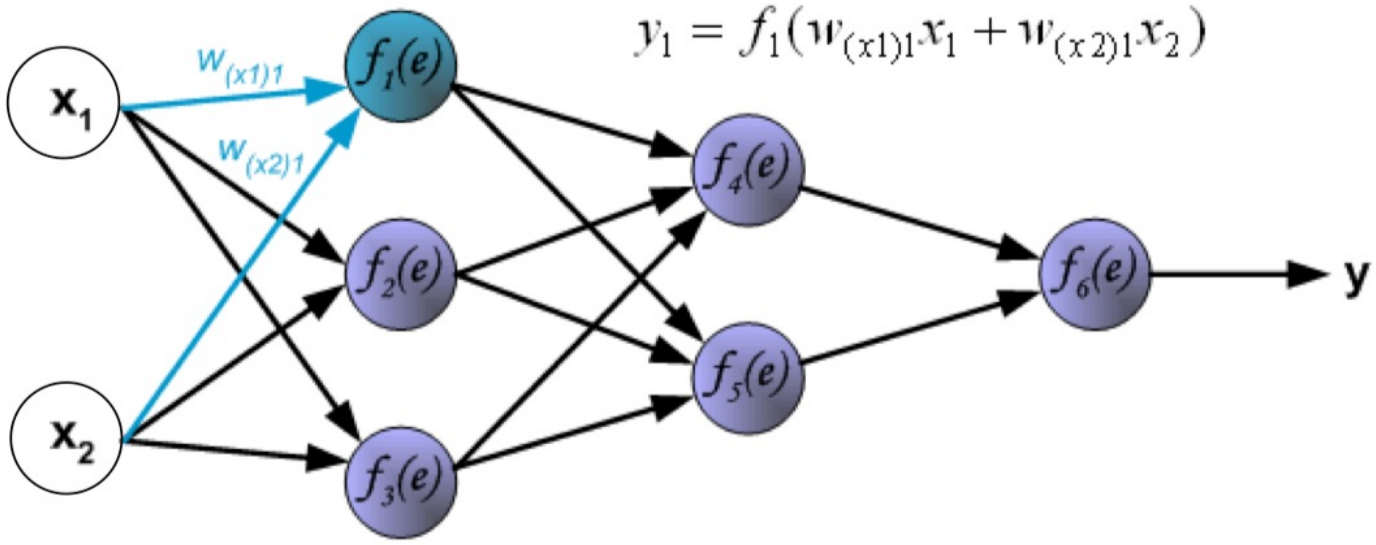
MLP: forward propagation

- nell'esempio una rete a 3 livelli (d:n_H:s): input d neuroni, livello nascosto (hidden) n_H neuroni, output s neuroni.
- Il numero totale di pesi (o parametri) è: $d \times n_H + n_H \times s + n_H + s$ dove gli ultimi due termini corrispondono ai pesi dei bias.
- Il k-esimo valore di output può essere calcolato come: (Eq. 1)

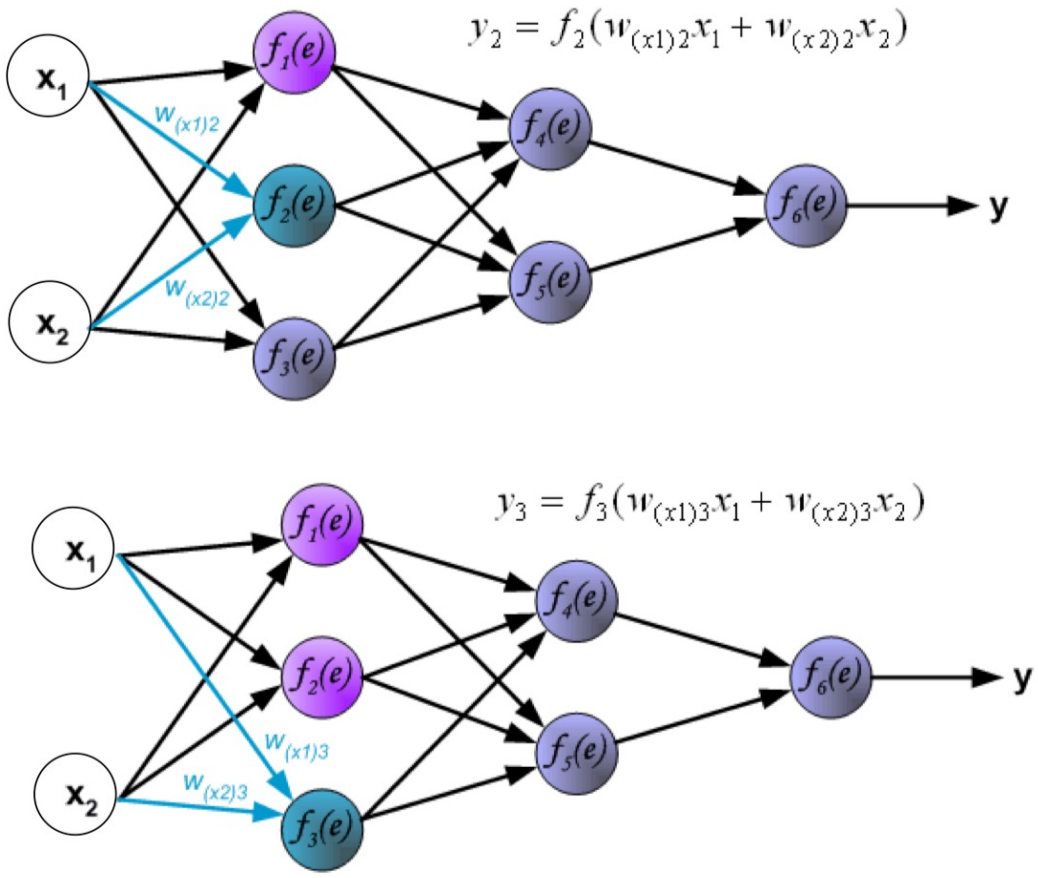
$$z_k = f\left(\sum_{j=1\ldots n_H} w_{jk} \cdot y_j + w_{0k}\right) = f\left(\sum_{j=1\ldots n_H} w_{jk} \cdot f\left(\sum_{i=1\ldots d} w_{ij} \cdot x_i + w_{0j}\right) + w_{0k}\right)$$

Forward propagation: esempio grafico

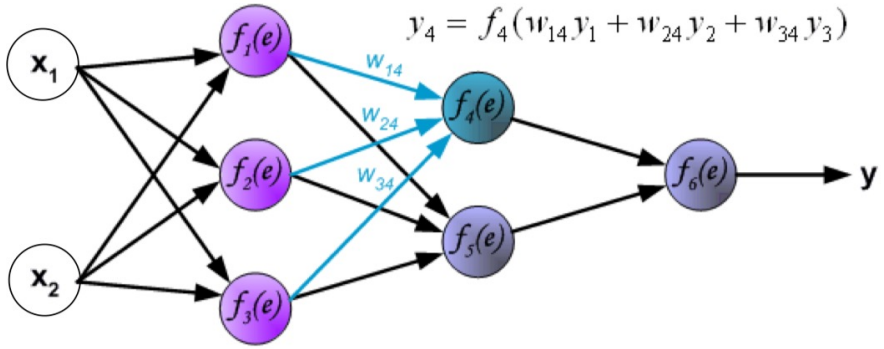
Nell'esempio una rete a 4 livelli 2:3:2:1 (2 livelli nascosti); non sono usati bias; nella grafica dell'esempio la notazione è un po' diversa dalla precedente ma facilmente comprensibile.

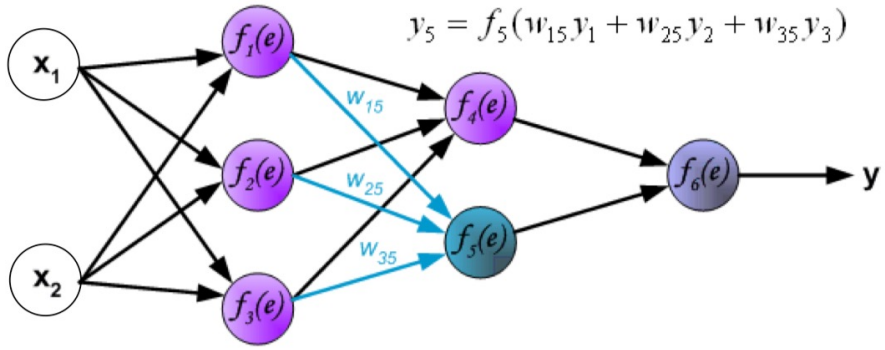


Forward propagation: esempio grafico

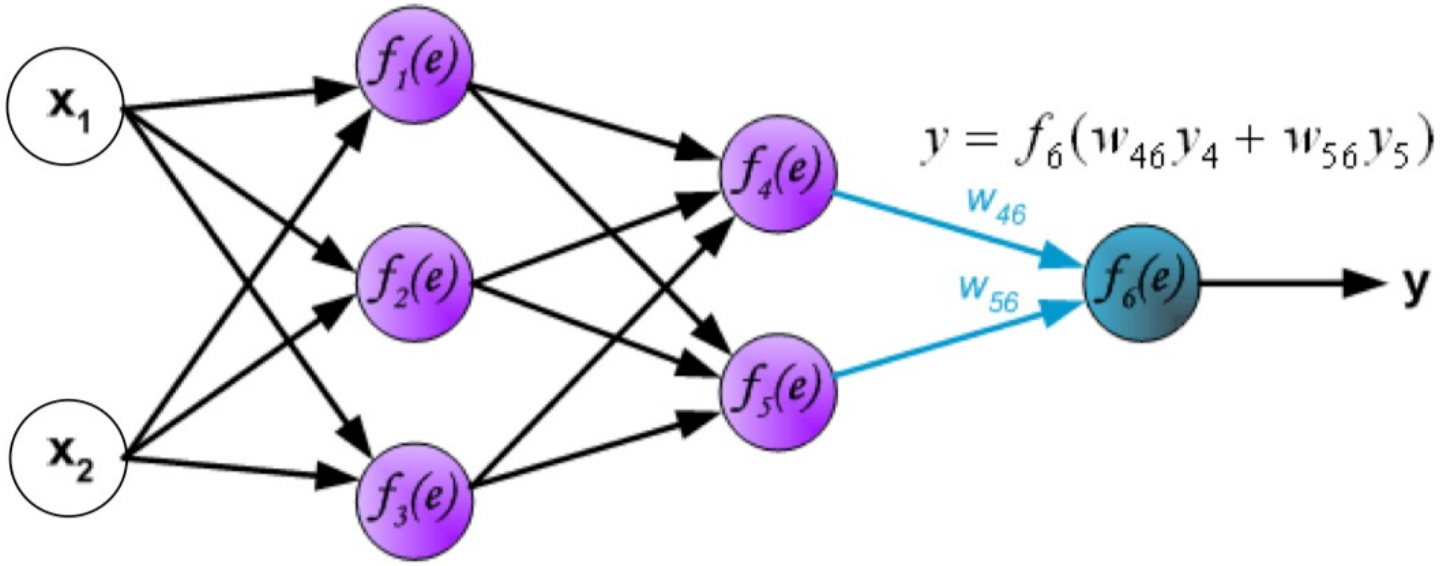


.....continua


$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$
x1
x2
f1(e)
f2(e)
f3(e)
w14
w24
w34
f4(e)
f5(e)
f6(e)
y


$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3)$
x1
x2
f1(e)
f2(e)
f3(e)
w15
w25
w35
f4(e)
f5(e)
f6(e)
y

.....continua



Multi Layer Perceptron

- Aumentando la dimensione dello strato di ingresso aumenta la dimensione dello spazio di classificazione. Un strato di ingresso di dimensione n corrisponde ad uno spazio di classificazione nD.
- Uno strato nascosto di p neuroni può separare fino ad un massimo di $(p^2 + p + 2)/2$ regioni dello spazio. Una regione si presenta come un poligono (2D) o un iperpoliedro con al massimo p lati o iperfacce.
- Utilizzando più neuroni nello strato di uscita è possibile aumentare il numero delle classi separabili. Per uno strato di uscita con q neuroni è possibile separare:
    - fino ad un massimo di 2^q (uscita binaria)
    - q classi (one-hot-encoding)

https://playground.tensorflow.org/

MLP: Training

- Fissata la topologia (numero di livelli e neuroni), l'addestramento di una rete neurale consiste nel determinare il valore dei pesi $\mathbf{w}$ che determinano il mapping desiderato tra input e output.
- Che cosa intendiamo per mapping desiderato? Dipende dal problema che vogliamo risolvere:
- se siamo interessati ad addestrare una rete neurale che operi come classificatore, l'output desiderato è l'etichetta corretta della classe del pattern in input.
- se la rete neurale deve risolvere un problema di regressione, l'output desiderato è il valore corretto della variabile dipendente, in corrispondenza del valore della variabile indipendente fornita in input.

MLP: Training

- Sebbene i primi neuroni artificiali risalgano agli anni 40', fino a metà degli anni 80' non erano disponibili algoritmi di training efficaci.
- Nel 1986 Rumelhart, Hinton & Williams hanno introdotto l'algoritmo di Error Backpropagation suscitando grande attenzione nella comunità scientifica.
- Si tratta in realtà dell'applicazione della regola di derivazione a catena, idea che però per decenni nessuno aveva applicato con successo in questo contesto.

Training Supervisionato di un Classificatore

Nel seguito focalizziamo l'attenzione su un problema di classificazione supervisionata.

- Per gestire un problema di classificazione con s classi e pattern d-dimensionali, si è soliti utilizzare una rete con $d : n_H : s$ neuroni nei tre livelli. Ovvero tanti neuroni di input quante sono le feature e tanti neuroni di output quante sono le classi. n_H è invece un iperparametro: un valore ragionevole è $n_H = {}^{1}/_{10}\, n$.
- La pre-normalizzazione dei pattern di input (attraverso min-max scaling, standardization o whitening) favorisce la convergenza durante l'addestramento.

Training Supervisionato di un Classificatore

- L'addestramento (supervisionato) avviene presentando alla rete pattern di cui è nota la classe e propagando (forward propagation) gli input verso gli output attraverso l'equazione (1).
- Dato un pattern di training di classe g, il vettore di output desiderato $\mathbf{t}$ (usando Tanh come funzione di attivazione) assume la forma:

$$\mathbf{t} = [-1, -1 \ldots \underset{\text{posizione } g}{1} \ldots -1]$$

- La differenza tra l'output prodotto della rete e quello desiderato è l'errore della rete. Obiettivo dell'algoritmo di apprendimento è di modificare i pesi della rete in modo da minimizzare l'errore medio sui pattern del training set.
- Prima dell'inizio dell'addestramento i pesi sono tipicamente inizializzati con valori random:
  - quelli input-hidden nel range $\pm 1/\sqrt{d}$
  - quelli hidden-output nel range $\pm 1/\sqrt{n_H}$

Backpropagation

Propagazione all'indietro dell'errore calcolato in uscita del layer finale alle uscite dei layer intermedi (Backpropagation 1986). L'algoritmo backpropagation si compone di due fasi, in cui il segnale analizzato viaggia in direzioni opposte:

- **Percorso diretto** il segnale di input si propaga dall'ingresso all'uscita. L'uscita predetta viene confrontata col ground truth per calcolare l'errore.
- **Percorso inverso** il segnale errore viene propagato all'indietro, e ad ogni passo vengono aggiustati i pesi del rispettivo layer usando la delta rule.

L'algoritmo backpropagation richiede che i neuroni abbiano tutti una funzione di attivazione derivabile

MLP: Error Backpropagation

- Sia $\mathbf{z} = [z_1, z_2 \dots z_s]$ l'output prodotto dalla rete (forward propagation) in corrispondenza del pattern $\mathbf{x} = [x_1, x_2 \dots x_d]$ di classe g fornito in input; come già detto l'output desiderato è $\mathbf{t} = [t_1, t_2 \dots t_s]$, dove $t_i = 1$ per $i = g$, $t_i = -1$ altrimenti.
- Scegliendo come loss function la somma dei quadrati degli errori, l'errore (per il pattern $\mathbf{x}$) è:

$$J(\mathbf{w}, \mathbf{x}) \equiv \frac{1}{2} \sum_{c=1 \dots s} (t_c - z_c)^2$$

che quantifica quanto l'output prodotto per il pattern $\mathbf{x}$ si discosta da quello desiderato. La dipendenza dai pesi $\mathbf{w}$ è implicita in $\mathbf{z}$. L'errore $J(\mathbf{w})$ sull'intero training set è la media di $J(\mathbf{w}, \mathbf{x})$ su tutti i pattern $\mathbf{x}$ appartenenti al training set.

MLP: Error Backpropagation

- $J(\mathbf{w})$ può essere ridotto modificando i pesi $\mathbf{w}$ in direzione opposta al gradiente di J. Infatti il gradiente indica la direzione di maggior crescita di una funzione (di più variabili) e muovendoci in direzione opposta riduciamo (al massimo) l'errore.
- Quando la minimizzazione dell'errore avviene attraverso passi in direzione opposta al gradiente l'algoritmo backpropagation è denominato anche gradient descent. Esistono anche tecniche di minimizzazione del secondo ordine che possono accelerare convergenza (applicabili in pratica solo a reti e training set di piccole/medie dimensioni).
- Nel seguito, consideriamo:
  - prima la modifica dei pesi w_{jk} hidden-output.
  - poi quella dei pesi w_{ij} input-hidden.

Modifica pesi hidden-output

$$\frac{\partial J}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}\left(\frac{1}{2}\sum_{c=1\ldots s}(t_c - z_c)^2\right) = (t_k - z_k)\frac{\partial(-z_k)}{\partial w_{jk}} =$$

solo z_k è influenzato da w_{jk}

$$= (t_k - z_k)\frac{\partial\left(-f(net_k)\right)}{\partial w_{jk}} = -(t_k - z_k)\cdot\frac{f(net_k)}{\partial net_k}\cdot\frac{\partial net_k}{\partial w_{jk}} =$$

$$= -(t_k - z_k)\cdot f'(net_k)\cdot\frac{\partial\sum_{s=1\ldots n_H} w_{sk}\cdot y_s}{\partial w_{jk}} = -(t_k - z_k)\cdot f'(net_k)\cdot y_j$$

posto $\delta_k = (t_k - z_k)\cdot f'(net_k)$ (Eq. 2)

allora $\frac{\partial J}{\partial w_{jk}} = -\delta_k \cdot y_j$ (Eq. 3)

pertanto il peso w_{jk} può essere aggiornato come:

$$w_{jk} = w_{jk} + \eta\cdot\delta_k\cdot y_j \quad \text{(Eq. 4)}$$

Modifica pesi hidden-output

dove η è il learning rate.

Se η troppo piccolo convergenza lenta, se troppo grande può oscillare e/o divergere. Partire con $\eta = 0.5$ e modificare il valore se non adeguato (monitorando l'andamento del loss durante le iterazioni).

N.B. se si usano i bias il peso w_{0k} si aggiorna ponendo $y_0 = 1$

Modifica pesi input-hidden

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\frac{1}{2}\sum_{c=1\dots s}(t_c - z_c)^2\right) = -\sum_{c=1\dots s}(t_c - z_c)\frac{\partial z_c}{\partial w_{ij}} =$$

$$= -\sum_{c=1\dots s}(t_c - z_c)\frac{\partial z_c}{\partial net_c}\cdot\frac{\partial net_c}{\partial w_{ij}} = -\sum_{c=1\dots s}\underbrace{(t_c - z_c)\cdot f'(net_c)}_{\delta_c \text{ vedi Eq. 2}}\cdot\frac{\partial net_c}{\partial w_{ij}}$$

$$\frac{\partial net_c}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\sum_{r=1\dots n_H} w_{rc}\cdot y_r = \frac{\partial}{\partial w_{ij}}\sum_{r=1\dots n_H} w_{rc}\cdot f(net_r) =$$

solo net_j è influenzato da w_{ij}

$$= \frac{\partial}{\partial w_{ij}}\left(w_{jc}\cdot f(net_j)\right) = w_{jc}\frac{\partial f(net_j)}{\partial net_j}\cdot\frac{\partial net_j}{\partial w_{ij}} =$$

$$= w_{jc}\cdot f'(net_j)\cdot\frac{\partial}{\partial w_{ij}}\sum_{q=1\dots d} w_{qj}\cdot x_q = w_{jc}\cdot f'(net_j)\cdot x_i$$

Modifica pesi input-hidden

$$\frac{\partial J}{\partial w_{ij}} = -\sum_{c=1\ldots s} \delta_c \cdot w_{jc} \cdot f'(net_j) \cdot x_i = -x_i \cdot f'(net_j) \sum_{c=1\ldots s} w_{jc} \cdot \delta_c$$

posto $\delta_j = f'(net_j) \cdot \sum_{c=1\ldots s} w_{jc} \cdot \delta_c$ (Eq. 5)

allora $\frac{\partial J}{\partial w_{ij}} = -\delta_j \cdot x_i$ (Eq. 6)

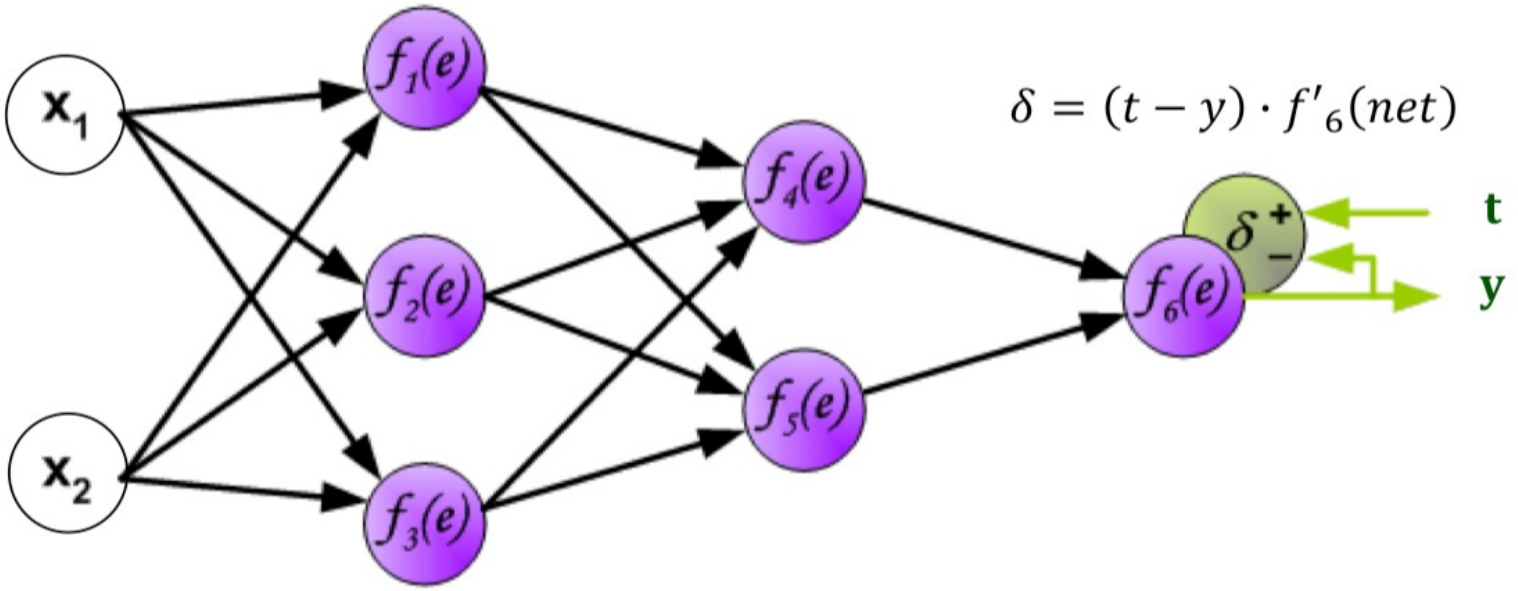
pertanto il peso w_{ij} può essere aggiornato come:

$$w_{ij} = w_{ij} + \eta \cdot \delta_j \cdot x_i \quad \text{(Eq. 7)}$$

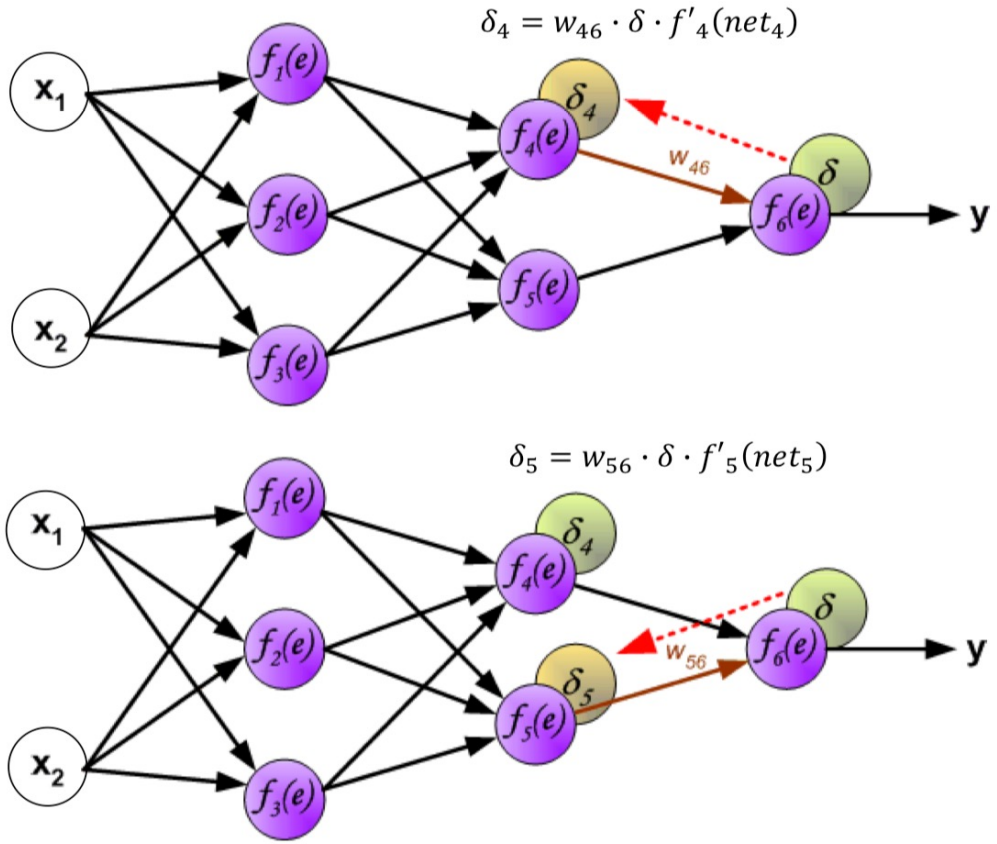
N.B. se si usano i bias il peso w_{0j} si aggiorna ponendo $x_0 = 1$

Backpropagation: esempio grafico

Stessa rete a 4 livelli su cui abbiamo eseguito forward propagation:

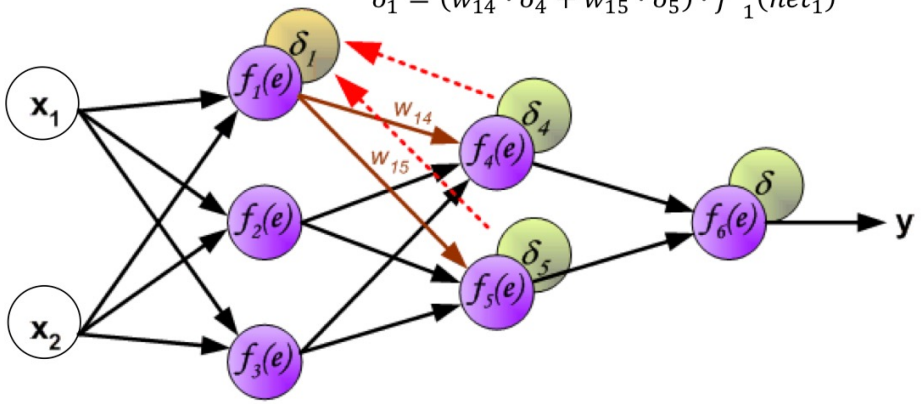


Backpropagation: esempio grafico

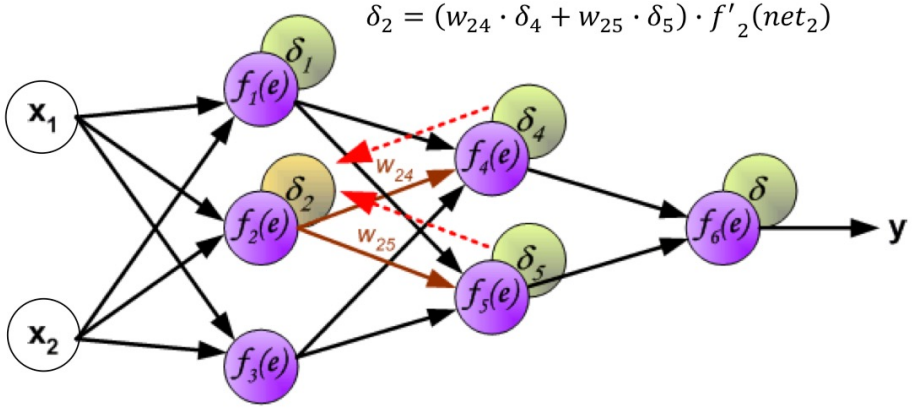


....continua

$$\delta_1 = (w_{14} \cdot \delta_4 + w_{15} \cdot \delta_5) \cdot f'_1(net_1)$$

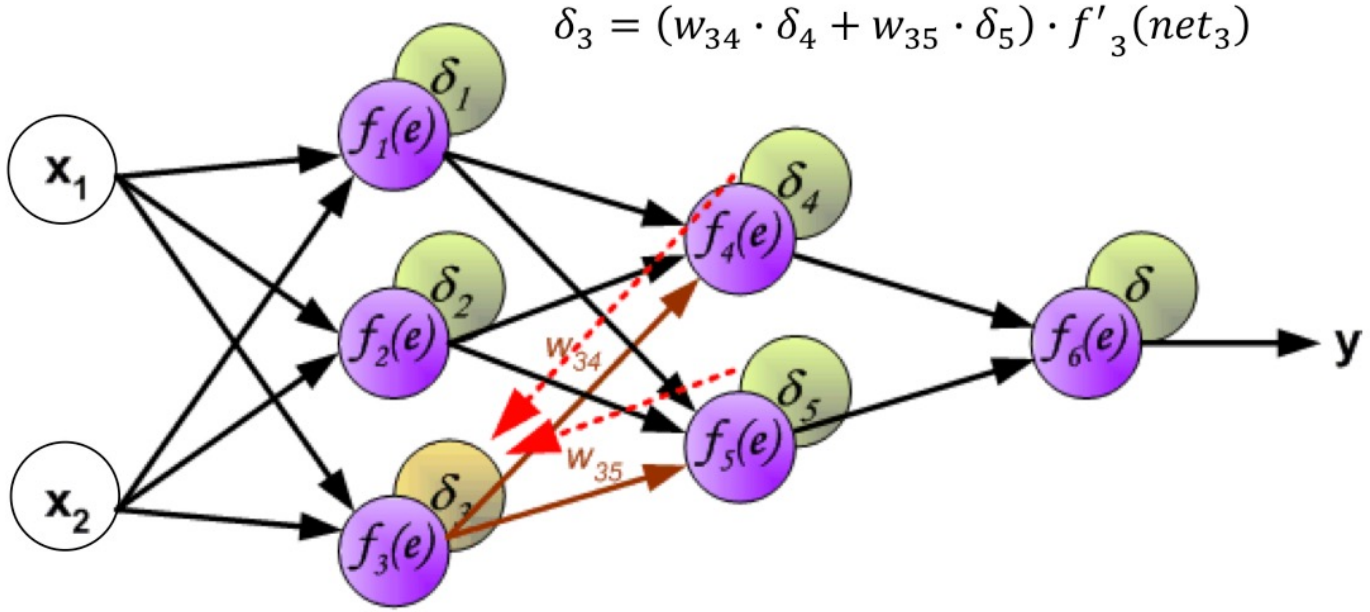


$$\delta_2 = (w_{24} \cdot \delta_4 + w_{25} \cdot \delta_5) \cdot f'_2(net_2)$$



....continua

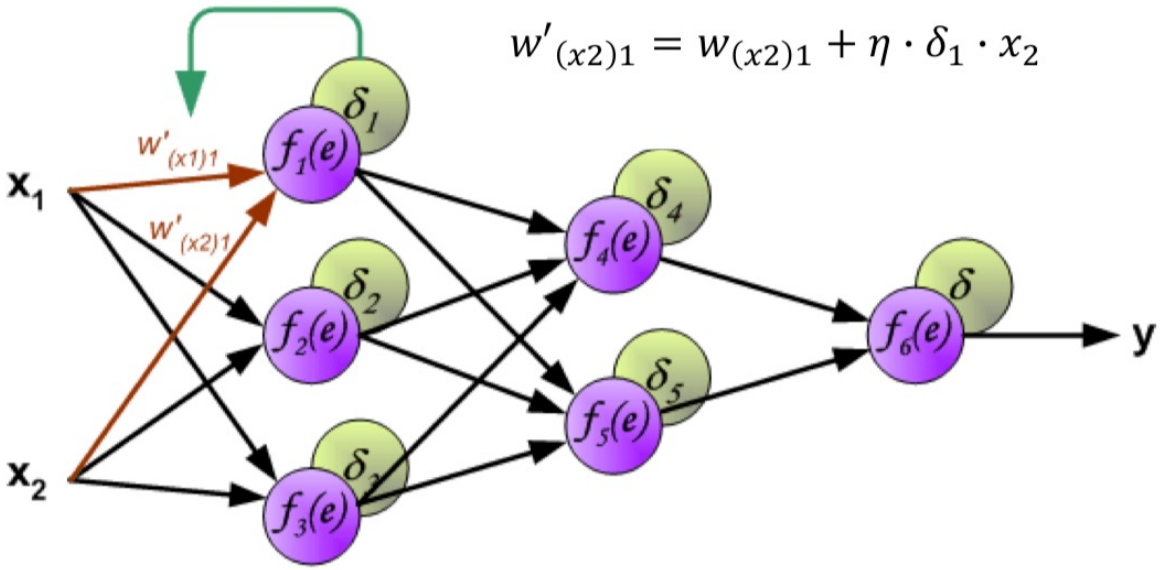
$$\delta_3 = (w_{34} \cdot \delta_4 + w_{35} \cdot \delta_5) \cdot f'_3(net_3)$$



e infine: aggiornamento pesi

$$w'_{(x1)1} = w_{(x1)1} + \eta \cdot \delta_1 \cdot x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \cdot \delta_1 \cdot x_2$$



…analogamente per tutti gli altri pesi

Algoritmo Backpropagation (Online)

Nella versione online i pattern del training set sono presentati sequenzialmente e i pesi sono aggiornati dopo la presentazione di ogni pattern:

Inizializza n_H, $\mathbf{w}$, η, num_{epoch}, $epoch \leftarrow 0$

do $epoch \leftarrow epoch + 1$

$totErr \leftarrow 0$ // *errore cumulato su tutti i pattern del TS*

for each $\mathbf{x}$ in Training Set

forward step: $\mathbf{x} \rightarrow z_k, \quad k = 1 \ldots s$ (eq. 1)

$totErr \leftarrow totErr + J(\mathbf{w}, \mathbf{x})$

backward step: $\delta_k, \ k = 1 \ldots s$ (eq. 2), $\delta_j, \ j = 1 \ldots n_H$ (eq. 5)

aggiorna pesi hidden-output $w_{jk} = w_{jk} + \eta \cdot \delta_k \cdot y_j$ (eq. 4)

aggiorna pesi input-hidden $w_{ij} = w_{ij} + \eta \cdot \delta_j \cdot x_i$ (eq. 7)

$loss \leftarrow totErr / n$ // *errore medio sul TS*

Calcola accuratezza su Train Set e Validation Set

while (*not convergence and* $epoch < num_{epoch}$)

Algoritmo Backpropagation (Online)

- La convergenza si valuta monitorando l'andamento del loss e l'accuratezza sul validation set.

- L'approccio on-line richiede l'aggiornamento dei pesi a seguito della presentazione di ogni pattern. Problemi di efficienza (molti update di pesi) e robustezza (in caso di outliers passi in direzioni sbagliate).

Modalità di training

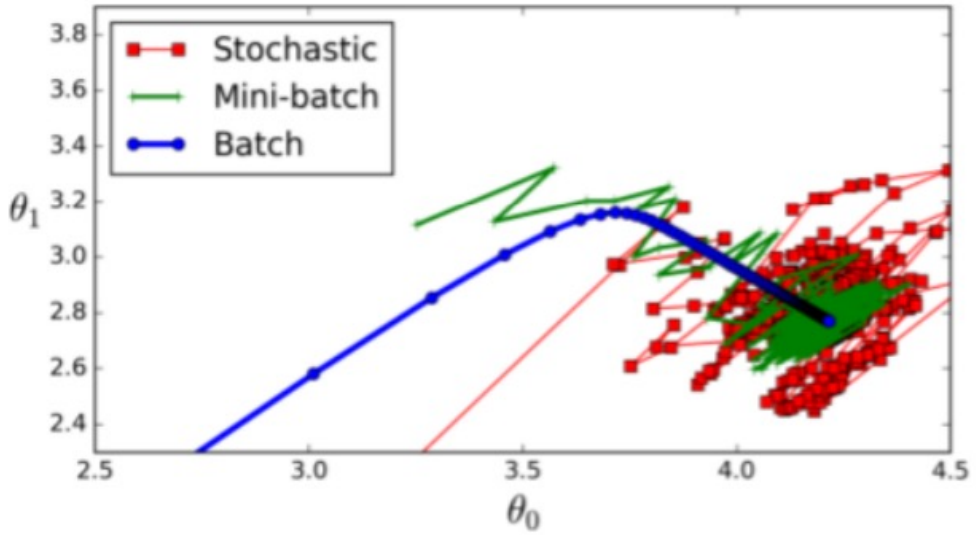
In base al calcolo dell'errore di training e conseguente aggiornamento dei pesi:

- **On-line** quando i pesi sono aggiornati ad ogni esempio di training in base all'errore ottenuto su quell'esempio.
- **Mini-batch** quando l'errore parziale è calcolato su un piccolo lotto di esempi (mini-batch) del training set.
- **Batch** quando l'errore complessivo è calcolato su tutti gli esempi del training set.

- Il **numero di epoche** definisce quante volte l'intero set di dati è stato mostrato alla rete durante il training.
- Il **numero di iterazioni** definisce quanti lotti sono necessari per completare un'epoca.

Modalità di Training

- **batch**: un unico aggiornamento ad epoca, giusta stima dei gradienti, può essere lento e richiedere molta memoria.
- **mini batch**: più aggiornamenti ad epoca con approssimata stima dei gradienti spostano globalmente i pesi nella giusta direzione
- **on-line**: molti aggiornamenti, peggiore stima dei gradienti, più veloce e richiede minore memoria



Suddivisione del dataset

- **Training Set** è l'insieme dei dati (etichettati o non) su cui addestrare la rete neurale, trovando il valore ottimo dei parametri (pesi e bias).
- **Validation Set** è l'insieme dei dati su cui tarare gli iperparametri H (ciclo esterno).
- **Test Set** è l'insieme dei dati su cui valutare le prestazioni finali del sistema.

Il validation set deve essere rappresentativo del test set

- Se non si vuole condurre l'**ottimizzazione degli iperparametri**, possiamo usare gli stessi dati per il validation set e test set.
- Da evitare l'ottimizzazione degli iperparametri su validation set e test set indistinti. In questo caso le prestazioni saranno sovrastimate.

Training

Sia data una rete multistrato con funzioni di attivazione derivabili e un training set costituito da m esempi (X k , T K)

1. Si inizializzano i pesi con valori casuali.
2. Il pattern X k viene presentato all'ingresso della rete che produce un uscita.
3. Dato il pattern T k , si calcolano gli errori δ j e di conseguenza si aggiornano pesi dell'ultimo layer.
4. Procedendo all'indietro (backpropagation), si calcolano gli errori per ogni layer nascosto.
5. a partire dagli errori calcolati, per ogni layer si calcolano le variazioni dei pesi, dopo di che vengono aggiornati
6. Ripetere il processo dal punto 2, per ogni esempio o lotto (scelto casualmente) del training set.
7. Ripetere il processo dal punto 2, per tante iterazioni.

Training

Iperparametri

Inizializzazione dei pesi:
- random (uniform, normal)
- ...

Loss function:
- distanza (δ)
- errore quadratico medio
- cross entropy
- ...

Algoritmo di ottimizzazione:
- SGD
- Adam
- ...

Validation e Test

Sia data una rete, presumibilmente addestrata e un validation set costituito da m esempi (X k , T K)

1. Il pattern X k viene presentato all'ingresso della rete che produce un uscita.
2. La metrica di validazione `e calcolata comparando l'uscita desiderata (ground truth) T K e l'uscita predetta.
3. Si ripete la procedura dal punto 1 per ogni esempio del validation set.
4. Viene calcolata la metrica di validazione complessiva sull'intero validation set.

Generalizzazione

Overfitting e underfitting

capacità di trasferire l'accuratezza raggiunta sul training set al validation set (rappresentativo del test set) obiettivo di massimizzare l'accuratezza sul test set

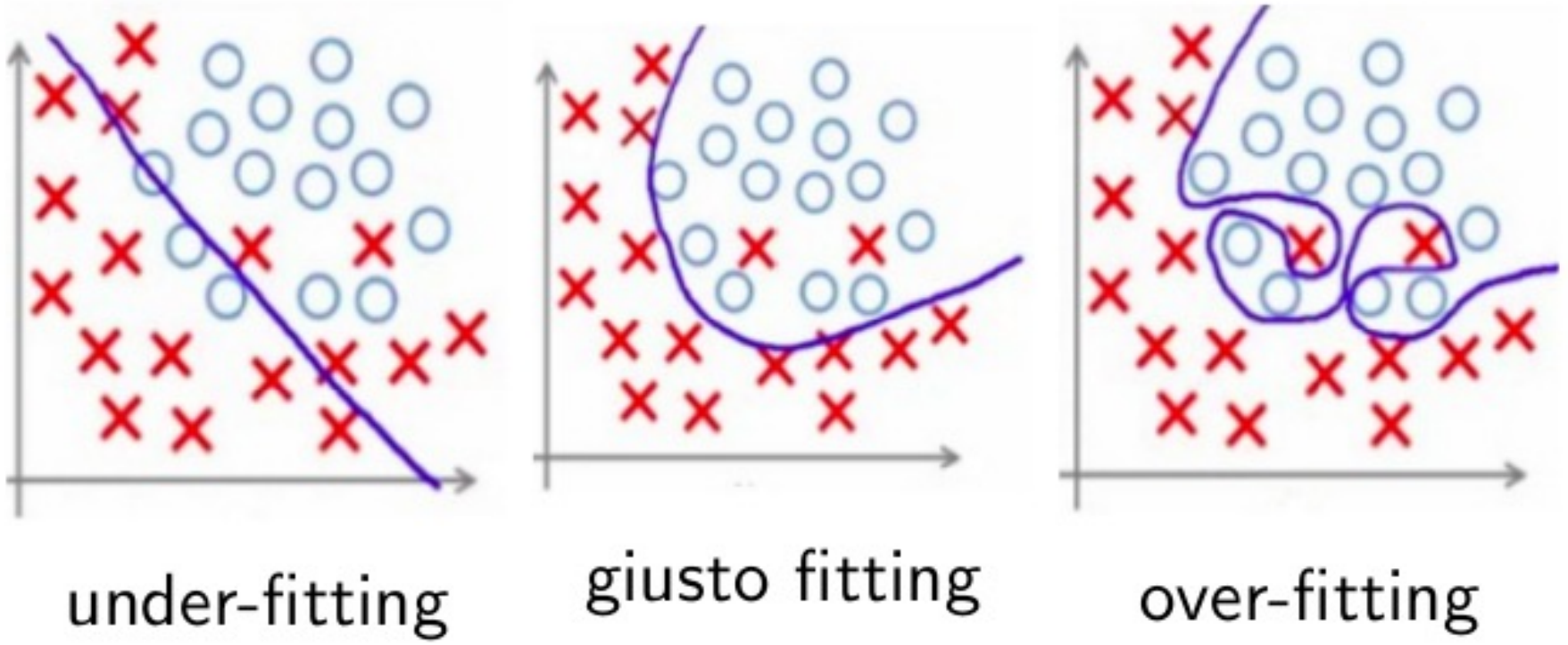
In generale si può avere:

- **Overfitting** quando si cerca di addestrare reti di grandi dimensioni rispetto al training set. (La rete impara a memoria gli esempi di training e non generalizza agli esempi non visti d urante il training)

- **Underfitting** quando si addestrano modelli piccoli con molti dati di training. (Rete troppo piccola per modellare una precisa superficie di separazione dei dati)

Generalizzazione

Overfitting e underfitting

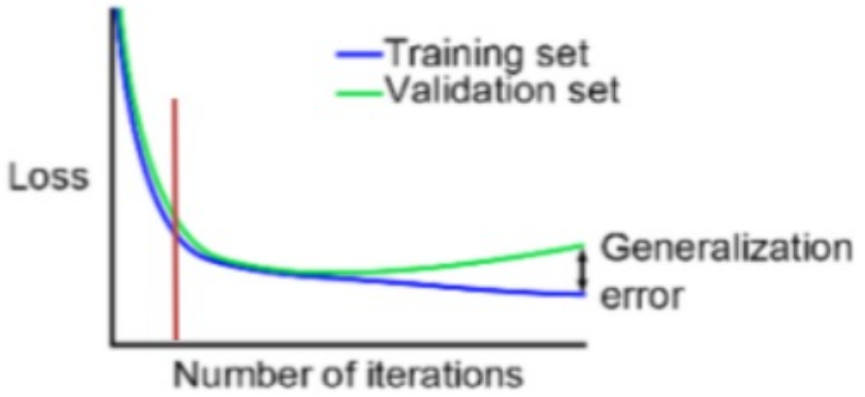


Generalizzazione

Overfitting e underfitting

All'aumentare della complessità (numero di parametri addestrabili), il modello tende a memorizzare "troppo bene" gli esempi di training perdendo la capacità di generalizzare su esempi non visti durante il training.

All'aumentare del numero di epoche di training, l'errore globale di training continua a diminuire, ma la rete perde la capacità di generalizzare.



Generalizzazione

Soluzioni al problema dell'overfitting

- Early stopping: criterio di stop al minimo errore di validazione.
- Data augmentation: tecniche di aumento della dimensione del dataset.
- Dropout: training casuale di n neuroni di un layer alla volta.
- Batch normalization: normalizza l'output di un layer ad avere media nulla e deviazione standard unitaria.
- Max-pooling: riduce i parametri addestrabili a valle.
- Ensemble: rete neurale molto profonde → reti neurali poco profonde in parallelo.

Stochastic Gradient Descent

- È l'approccio più utilizzato per implementare backpropagation:
  - a ogni epoca, gli n pattern del training set sono ordinati in modo casuale e poi suddivisi, considerandoli sequenzialmente, in gruppi (denominati mini-batch) di uguale dimensione $size_{mb}$.
    - se $size_{mb} = 1 \rightarrow$ «*stochastic*» *online*
    - se $size_{mb} = n \rightarrow$ «*full*» *batch*
  - i valori del gradiente sono (algebricamente) accumulati in variabili temporanee (una per ciascun peso); l'aggiornamento dei pesi avviene solo quando tutti i pattern di un mini-batch sono stati processati.

Stochastic Gradient Descent

Inizializza n_H, $\mathbf{w}$, η, num_{epoch}, $size_{mb}$, $epoch \leftarrow 0$

do $epoch \leftarrow epoch + 1$

  random sort the Training Set (n patterns)

  for each mini-batch B of $size_{mb}$

    reset gradiente

    for each $\mathbf{x}$ in B

      forward step

      backward step *// gradiente cumulato su tutti i pattern del mini-batch*

    aggiorna pesi

  Calcola Loss

  Calcola accuratezza su Training Set e Validation Set

while (*not convergence and* $epoch < num_{epoch}$)

Terminologia (SGD)

- Attenzione alla terminologia:
  - Per epoca (epoch) si intende la presentazione (1 volta) alla rete di tutti i pattern del training set.
  - Per iterazione (iteration) si intende la presentazione (1 volta) dei pattern costituenti un mini-batch e il conseguentemente aggiornamento dei pesi.
    - $n/size_{mb}$ è il numero di iterazioni per epoca. Se non è un valore intero all'ultima iterazione si processano meno pattern.

Terminologia (SGD)

Inizializza n_H, $\mathbf{w}$, η, num_{epoch}, $size_{mb}$, $epoch \leftarrow 0$

do $epoch \leftarrow epoch + 1$

- random sort the Training Set (n patterns)
- for each mini-batch B of $size_{mb}$
  - reset gradiente
  - for each $\mathbf{x}$ in B
    - forward step
    - backward step
  - aggiorna pesi
- Calcola Loss
- Calcola accuratezza su Training Set e Validation Set

epoca

iterazione

while (*not convergence and epoch* < num_{epoch})

Terminologia (SGD)

Invece di num_{epoch} e $size_{mb}$ alcuni tool (tra cui Caffe) richiedono in input il numero totale di iterazioni num_{iter} e $size_{mb}$; in questo caso il numero di epoche (non necessariamente intero) è:

$$num_{epoch} = \frac{num_{iter} \cdot size_{mb}}{n}$$

SoftMax: Output Probabilistico

- Se nel livello di output si utilizza come funzione di attivazione:
  - Tanh: i valori di uscita sono nell'intervallo [-1...1], e quindi non sono probabilità.
  - Sigmoid (standard logistic function): i valori di uscita sono compresi nell'intervallo [0…1], ma non abbiamo nessuna garanzia che la somma sui neuroni di uscita sia 1 (requisito fondamentale affinché si possano interpretare come distribuzione probabilistica).

$$\sum_{c=1\ldots s} z_c \neq 1$$

SoftMax: Output Probabilistico

- Quando una rete neurale è utilizzata come classificatore multi-classe, l'impiego della funzione di attivazione softmax consente di trasformare i valori di net prodotti dall'ultimo livello della rete in probabilità delle classi:
  - Il livello di attivazione net_k dei singoli neuroni dell'ultimo livello si calcola nel modo consueto, ma come funzione di attivazione per il neurone k -esimo si utilizza:

$$z_k = f(net_k) = \frac{e^{net_k}}{\sum_{c=1\ldots s} e^{net_c}}$$

  - i valori z_k prodotti possono essere interpretati come probabilità delle classi: appartengono a [0…1] e la loro somma è 1.
  - l'esponenziale (utile nella combinazione con Cross Entropy Loss, vedi slide seguenti) interpreta i valori net come unnormalized log probabilities delle classi.

Cross Entropy

- Nelle slide precedenti abbiamo utilizzato per un problema di classificazione multi-classe la funzione di attivazione Tanh e la loss function Sum of Squared Error.
  - questa scelta non è ottimale, in quanto i valori di output non rappresentano probabilità, e la non imposizione del vincolo di somma a 1, rende (in genere) meno efficace l'apprendimento.

- Per un problema di classificazione multi-classe si consiglia l'utilizzo di Cross-Entropy (detta anche Multinomial Logistic Loss) come loss function:
  - La cross-entropy tra due distribuzioni discrete p e q (che fissata p misura quanto q differisce da p) è definita da:

$$H(p,q) = -\sum_{v} p(v) \cdot log\big(q(v)\big)$$

Cross Entropy Loss

- Nell'impiego come loss function: p (fissato) è il vettore target, mentre q il vettore di output della rete.

- Il valore minimo di H (sempre ≥ 0) si ha quando il vettore target coincide con l'output. Attenzione nella formula il logaritmo non esiste in 0, ma gli output forniti da softmax e sigmoid non valgono mai 0 (anche se vi possono tendere asintoticamente).

- Per approfondimenti e significato in teoria dell'Informazione: https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/

Cross Entropy e One-hot targets

- Quando il target vector per $\mathbf{x}$ assume la forma one-hot (tutti 0 tranne un 1 per la classe corretta g):

posizione g

$$\mathbf{t} = [0, 0 \ldots 1 \ldots 0]$$

allora:

$$J(\mathbf{w}, \mathbf{x}) \equiv H(\mathbf{t}, \mathbf{z}) = -log(z_g)$$

Cross-Entropy e One-hot targets

Se z_g è ottenuta con funzione di attivazione softmax, allora:

$$J(\mathbf{w}, \mathbf{x}) = -log\left(\frac{e^{net^g}}{\sum_{c=1\ldots s} e^{net_c}}\right) = -net^g + log \sum_{c=1\ldots s} e^{net_c}$$

Per la derivazione del gradiente (backpropagation) si veda l'ottimo report "Notes on Backpropagation" di Peter Sadowski

Loss Function per Classificazione

- Classificazione multi-classe: utilizzare funzione di attivazione softmax nel livello finale e cross-entropy come loss function.
  - se i target vector **t** assumono la forma one-hot (default) si utilizza la semplificazione della slide precedente.
  - in caso di incertezza sulla classe corretta, è possibile utilizzare «soft target», con singoli valori tra 0 e 1 e somma a 1. In questo si usa formula completa di cross-entropy.

Loss Function per Classificazione

- Classificazione binaria: è possibile ricadere nel caso precedente e trattare il problema come multiclasse con 2 classi, ma risulta preferibile (spesso convergenza migliore):
  - utilizzare 1 neurone di uscita che codifica la probabilità della sola prima classe (la seconda probabilità è il complemento a 1). A tal fine si utilizza la funzione di attivazione sigmoid che produce valori in [0…1] per il primo neurone.
  - la normalizzazione a 1 della probabilità delle due classi è implicita nella semplificazione di cross-entropy adottata (~~multinomial~~ logistic regression). Si noti che con 1 neurone t, z sono valori scalari in [0...1]:

$$H(t,z) = -(t\ log(z) + (1-t)\ log(1-z))$$

Loss Function per Classificazione

- Un caso particolare è la cosiddetta classificazione multi-label dove un pattern può appartenere a più classi. Ad esempio un'immagine che contiene sia un cane che un gatto potrebbe essere classificata come appartenente a 2 classi.
  - in questo caso ogni uscita è in [0…1] ma deve essere rilassato il vincolo di somma a 1 per gli output. Si può ottenere come estensione della classificazione binaria (sopra) utilizzando attivazioni sigmoid e sommando $H(t, z)$ su più neuroni.

Loss Function per Regressione

- Consideriamo un problema di regressione dove sia la variabile indipendente (input) che quella dipendente (output) sono vettori.
  - Utilizzando la terminologia introdotta per la regressione stiamo affrontando un problema di multivariate multiple regression
  - La rete neurale è in grado di trovare un mapping non lineare tra input e output, pertanto il termine linear non si applica

- Se l'output assume valori reali non limitati si consiglia di non utilizzare funzione di attivazione nel livello finale e adottare sum of squared error come loss function. I target vector $\mathbf{t}$ non sono soggetti a vincoli:

$$\mathbf{t} = [t_1, t_2 \dots t_s]$$

Regolarizzazione

- Per ridurre il rischio di overfitting del training set da parte di una rete neurale con molti parametri (pesi), si possono utilizzare tecniche di regolarizzazione. La regolarizzazione è molto importante quando il training set non ha grandi dimensioni rispetto alla capacità del modello.
- reti neurali i cui pesi, o parte di essi, assumono valori piccoli (vicino allo zero) producono output più regolare e stabile, portando spesso a migliore generalizzazione.
- Per spingere la rete ad adottare pesi di valore piccolo si può aggiungere un termine di regolarizzazione alla loss. Ad esempio nel caso di Cross-Entropy Loss:

$$J_{Tot} = J_{Cross-Entropy} + J_{Reg}$$

Regolarizzazione

- Nella regolarizzazione L2 il termine aggiunto corrisponde alla somma dei quadrati di tutti i pesi della rete:

$$J_{Reg} = \frac{1}{2}\lambda \sum_i {w_i}^2$$

- Nella regolarizzazione L1 si utilizza il valore assoluto:

$$J_{Reg} = \lambda \sum_i |w_i|$$

- In entrambi i casi il parametro λ regola la forza della regolarizzazione.
- L1 può avere un effetto sparsificante (ovvero portare numerosi pesi a 0) maggiore di L2. Infatti quando i pesi assumono valori vicini allo zero il calcolo del quadrato in L2 ha l'effetto di ridurre eccessivamente i correttivi ai pesi rendendo difficile azzerarli.

Regolarizzazione -> Weight Decay

- Consideriamo la regolarizzazione L2.
  - il gradiente della J_{Tot} rispetto ad uno dei parametri della rete corrisponde al somma del gradiente di $J_{Cross-Entropy}$ e del gradiente di J_{Reg}. Quest'ultimo vale:

$$\frac{\partial J_{Reg}}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\frac{1}{2}\lambda \sum_i {w_i}^2\right) = \lambda \cdot w_k$$

  - pertanto l'aggiornamento dei pesi a seguito di backpropagation include un ulteriore termine denominato weight decay (decadimento del peso) che ha l'effetto di tirarlo verso lo 0.

Regolarizzazione -> Weight Decay

- Ad esempio considerando l'aggiornamento pesi di SGD (vedi precedenti equazioni 4 e 7) :

$$w_k = w_k - \eta \cdot \frac{\partial J_{Cross-Entropy}}{\partial w_k}$$

diventa:

$$w_k = w_k - \eta \left(\frac{\partial J_{Cross-Entropy}}{\partial w_i} + \lambda \cdot w_i \right)$$

Analogo ragionamento per la regolarizzazione L1. *Quanto vale la derivata del valore assoluto?*

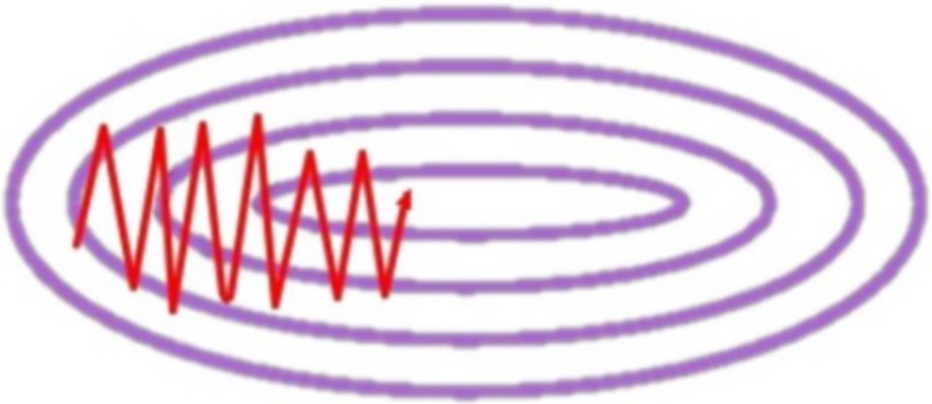
Momentum

- SGD con mini-batch può determinare una discesa del gradiente a zig-zag che rallenta la convergenza e la rende meno stabile.
  - la discesa del gradiente può essere vista come la traiettoria di una biglia che rotola su una superficie scoscesa. Attratta dalla forza di gravità la biglia cerca di portarsi sul punto più basso. Nella fisica il momento conferisce alla biglia un moto privo di oscillazioni e cambi di direzione repentini.
  - in SGD per evitare oscillazioni si può emulare il comportamento fisico: si tiene traccia dell'ultimo aggiornamento Δw_k di ogni parametro w_k, e il nuovo aggiornamento è calcolato come combinazione lineare del precedente aggiornamento (che conferisce stabilità) e del gradiente attuale $\frac{\partial J}{\partial w_i}$ (che corregge la direzione):

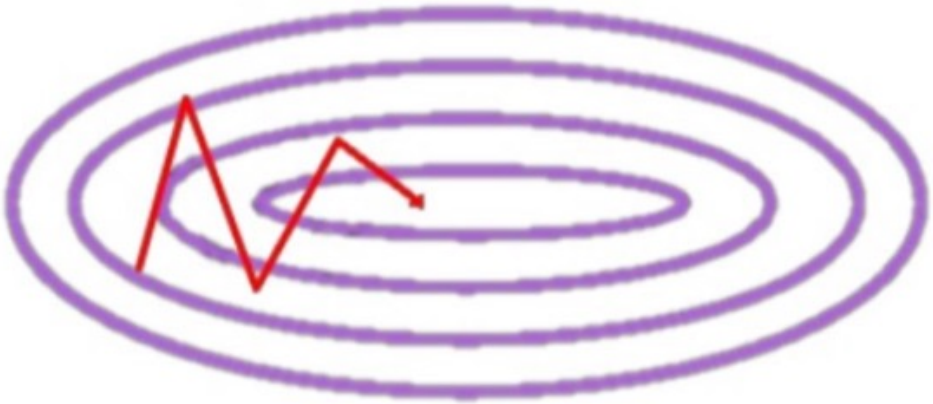
$$\Delta w_k = \mu \cdot \Delta w_k - \eta \frac{\partial J_{Tot}}{\partial w_k}$$

Momentum

$$w_k = w_k + \Delta w_k$$



Steps without Momentum



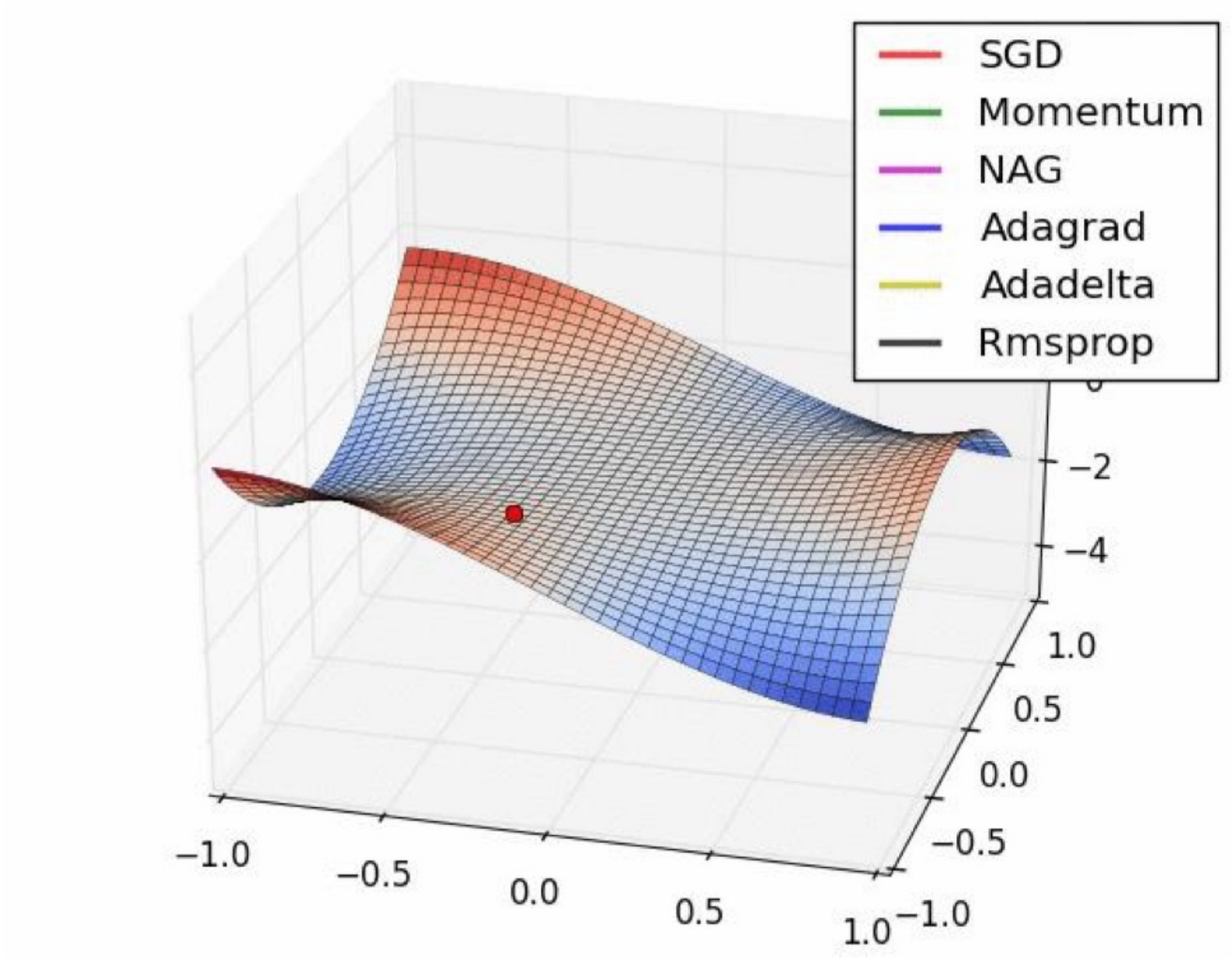
Steps with Momentum

Valore tipico del parametro $\mu = 0.9$

Learning Rate Adattivo

- Oltre a Momentum esistono altre regole per l'aggiornamento dei pesi che possono accelerare la convergenza nella discesa del gradiente (vedi http://cs231n.github.io/neural-networks-3/):
  - Nesterov Accelerate Gradient (NAG)
  - Adaptive Gradient (Adagrad)
  - Adadelta
  - RMSProp
  - Adam
- La maggior parte di queste adatta il learning rate globale η a ogni specifico peso. Adam è considerato lo stato dell'arte, ma non sempre fornisce risultati migliori del più classico SGD with Momentum.

Learning Rate Adattivo



Tricks of the Trade

- In molti sostengono che il training di una rete neurale (complessa) sia più un'arte che una scienza.
  - Sicuramente per gestire efficacemente le molteplici scelte implementative e iperparametri: architettura, funzione di attivazione, inizializzazione, learning rate, ecc. serve esperienza (e metodo).

- Un articolo molto utile (un po' datato ma ancora attuale), che discute dal punto di vista pratico l'implementazione efficiente di backpropagation è:
  - Efficient BackProp, by Yann LeCun et al.

    http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Tricks of the Trade

- Un articolo più recente, più specifico per reti deep, ma i cui suggerimenti sono applicabili anche a MLP è:
  - Practical Recommendations for Gradient-Based Training of Deep Architectures, by Yoshua Bengio

    https://arxiv.org/pdf/1206.5533v2.pdf

Molti concetti introdotti in queste slide possono essere sperimentati e «visivamente» compresi utilizzando il simulatore di NN disponibile al link: http://playground.tensorflow.org

Data Augumentation

Data augmentation significa aumentare il numero di punti dati. In termini di immagini, può significare che l'aumento del numero di immagini nell'insieme di dati. In termini di dati tradizionali in formato riga/colonna, significa aumentare il numero di righe o oggetti.

Perché?

Non abbiamo una quantità infinita di dati. Più dati ci sono, migliori saranno, in linea di principio, i nostri modelli ML. Ma ogni processo di raccolta dati è associato ad un costo. Questo costo può essere in termini di dollari, sforzo umano, risorse computazionali e tempo fuori rotta consumato nel processo. Pertanto, potremmo aver bisogno di aumentare i dati esistenti per aumentare la dimensione dei dati che forniamo ai nostri classificatori ML e per compensare il costo che comporta la raccolta di ulteriori dati.

Data Augumentation

Come?

Ci sono molti modi per aumentare i dati. Nelle immagini, è possibile ruotare l'immagine originale, cambiare le condizioni di illuminazione, ritagliarla in modo diverso, in modo da poter generare diversi sottocampioni per una sola immagine. In questo modo è possibile ridurre il sovradimensionamento del classificatore[1]. D'altra parte, se si generano dati artificiali utilizzando metodi di sovracampionamento come SMOTE[2], c'è una buona probabilità che si possa introdurre un sovracampionamento. Quindi bisogna fare attenzione alla scelta dell'aumento dei dati.

[1] - Shehroz Khan's answer to How exactly do data augmentation techniques like mirroring and cropping reduce overfitting?

[2] - http://www.jair.org/media/953/live-953-2037-jair.pdf

- rotazione e capovolgimento
- random cropping
- ingrandimenti e deformazioni
- inversione dei canali (robustezza ai cambiamenti di colore)
- aggiunta di rumore

unimc
UNIVERSITÀ DI MACERATA
l'umanesimo che innova

DIPARTIMENTO DI
SCIENZE POLITICHE,
DELLA COMUNICAZIONE,
E DELLE RELAZIONI INTERNAZIONALI

Deep Learning

unimc UNIVERSITÀ DI MACERATA
l'umanesimo che innova
DIPARTIMENTO DI SCIENZE POLITICHE, DELLA COMUNICAZIONE, E DELLE RELAZIONI INTERNAZIONALI

Deep Learning

- Introduzione
- Perché deep?
- Livelli e complessità
- Tipologie di DNN
- Ingredienti necessari
- Da MLP a CNN

- Convolutional Neural Networks (CNN)
- Architettura
- Volumi e Convoluzione 3D
- Relu, Pooling
- Esempi di reti
- Training e Transfer Learning

Deep Learning

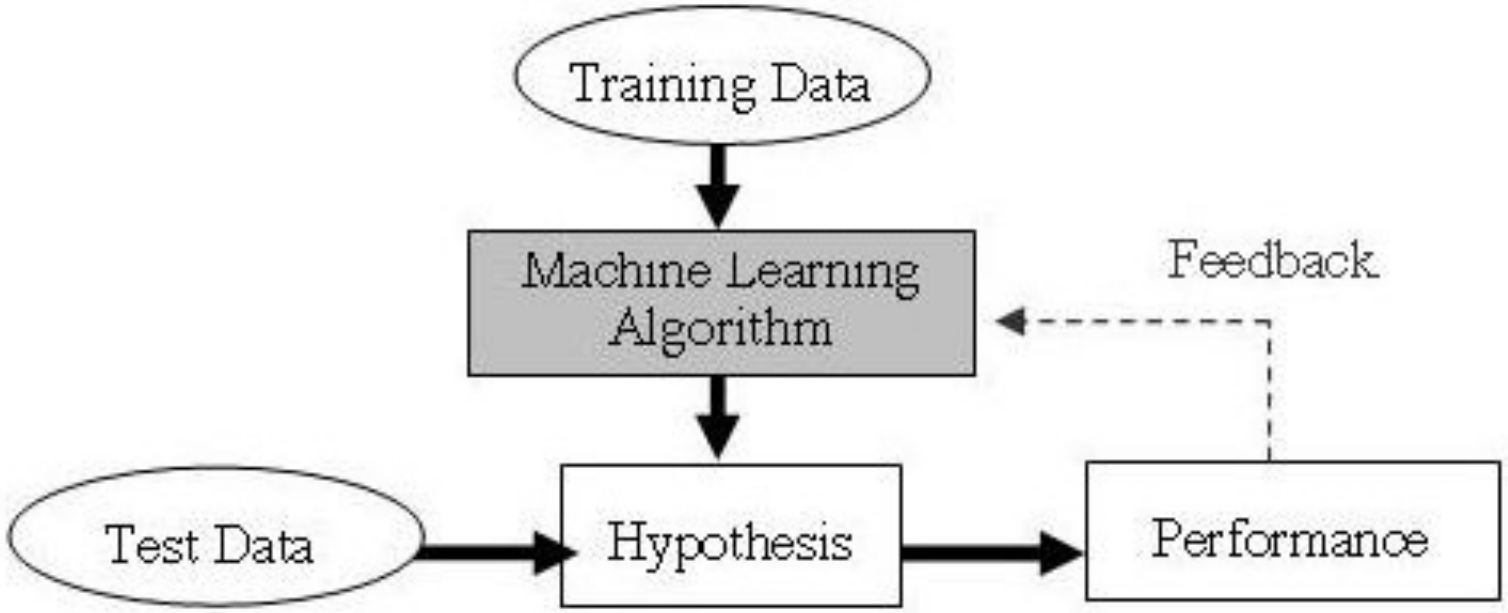
- Recurrent Neural Networks (RNN)
- Sequence to Sequence
- Unfolding in Time
- Basic Cells, LSTM, GRU
- Natural Language Processing

- Reinforcement Learning
- Q-Learning
- Deep Q-Learning

Deep Learning vs Machine Learning

Machine Learning (ML o Appredimento Automatico) è essenzialmente una forma di statistica applicata mirata ad utilizzare i computer per stimare statisticamente una funzione complessa. Mitchell nel 1997 fornì la seguente definizione di Machine Learning: "Un algoritmo apprende dall'esperienza E riguardanti una classe di problemi T con una misura pari a P, se la sua performance sui problemi T, misurata tramite P, aumenta con l'esperienza E". In sostanza, il ML è un insieme di tecniche che permettono alla macchine di "imparare" dai dati e in seguito prendere decisioni o fare una predizione su di essi. Un sistema di Machine Learning può essere applicato ad una base di "Conoscenza" proveniente da sorgenti multiple per risolvere diversi compiti: classificazione facciale, riconoscimento del parlato, riconoscimento di oggetti, ecc. A differenza degli algoritmi euristici, ossia quegli algoritmi che seguono un insieme specifico di istruzioni per risolvere un dato problema, il Machine Learning abilita un computer ad apprendere come riconoscere "configurazioni percettive" da solo e fare predizioni su di esse.

Deep Learning vs Machine Learning



Deep Learning vs Machine Learning

Il *Deep Learning* è una sotto-area del Machine Learning che fa uso delle *"Reti Neurali Profonde" (Deep Neural Network),* ossia dotate di molti strati e di nuovi algoritmi per il pre-processamento dei dati per la regolarizzazione del modello: *word embeddings, dropout, data-augmentation, ecc.* Il *Deep Learning* trae ispirazione dalle *Neuroscienze* dal momento che le Reti Neurali sono un modello dell'attività neuronale del cervello. A differenza del cervello biologico, dove qualsiasi neurone può connettersi a qualsiasi altro neurone sotto alcuni vincoli fisici, le Reti Neurali Artificiali (ANN) hanno un numero finito di strati e connessioni, e infine hanno una direzione prestabilita della propagazione dell'informazione. Finora, le Reti Neurali Artificiali sono state ignorate sia dalla comunità della ricerca che dall'industria I principale problema il loro costo computazionale.

Deep Learning vs Machine Learning

Tuttavia, tra il 2006 e il 2012, il gruppo di ricerca guidato da **Geoffrey Hinton** dell'Università di Toronto è stato in grado finalmente di parallelizzare gli algoritmi per le ANN su architetture parallele. Il principale risultato è stato un notevole incremento del numero di strati, neuroni e parametri del modello in generale (anche oltre i 10 milioni di parametri) permettendo alle macchine di computare una quantità massiccia di dati addestrandosi su di essi.

Deep Learning vs Machine Learning

Pertanto, il primo requisito per l'addestramento di un modello di *Deep learning (Apprendimento profondo)* è avere a disposizione train-set molto grandi. Questo rende il *Deep Learning* molto adatto ad affrontare l'era dei **Big Data**.

Deep learning ha Influenzato le applicazioni industriali come mai era successo prima al Machine Learning. Infatti esso è in grado di trattare un enorme quantità di dati – milioni di immagini, per esempio – e riconoscere alcune caratteristiche discriminative.

Deep Learning vs Machine Learning

Le ricerche basate su testo, l'individuazione di frodi o spam, il riconoscimento delle scritte, la ricerca delle immagini, il riconoscimento del parlato, i recommendation system, la Street View detection e la traduzione di lingue, sono solo alcuni dei compiti che il Deep Learning è in grado di affrontare. In Google, le reti deep hanno già rimpiazzato decine di "sistemi a regole".

Oggi il Deep Learning per la Computer Vision già mostra di avere capacità super-umane, e che variano dal riconoscimento di figure comuni come cani e gatti fino all'individuazione di noduli cancerosi in immagini tomografiche polmonari.

Deep Learning vs Machine Learning

Machine Learning. Netta separazione tra

- estrazione di features
- hand-crafted ("a mano" in base all'esperienza di chi implementa il codice)
- learned (apprese da una rete neurale convoluzionale)

- classificazione

Deep Learning. Estrazione delle features e classificazione nello stesso processo. Features estratte automaticamente dalla rete. (Black box)

Perché Deep?

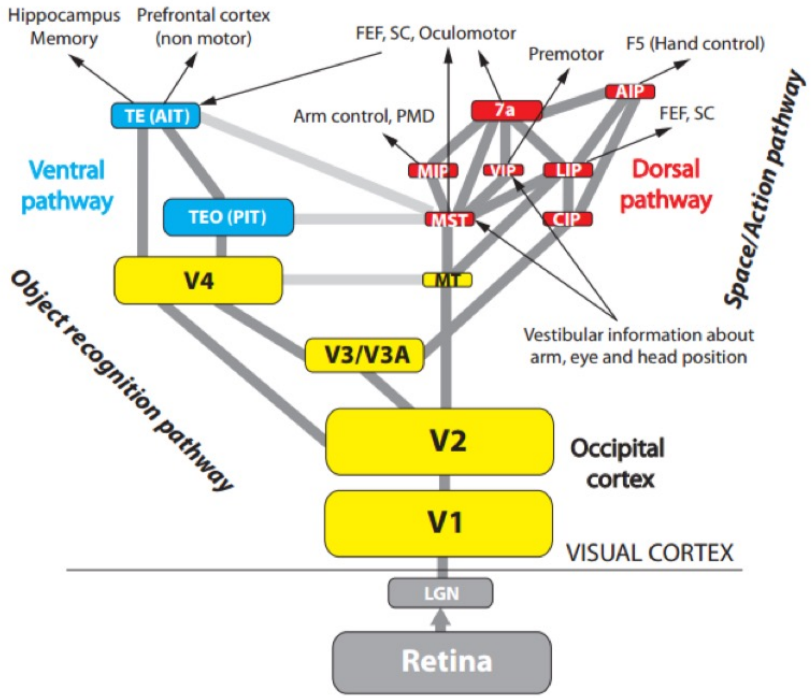
Con il termine DNN (Deep Neural Network) si denotano reti «profonde» composte da molti livelli (almeno 2 hidden) organizzati gerarchicamente.

- Le implicazioni di universal approximation theorem e la difficoltà di addestrare reti con molti livelli hanno portato per lungo tempo a focalizzarsi su reti con un solo livello hidden.
  - L'esistenza di soluzioni non implica efficienza: esistono funzioni computabili con complessità polinomiale operando su k livelli, che richiedono una complessità esponenziale se si opera su $k-1$ livelli (Hastad, 1986).

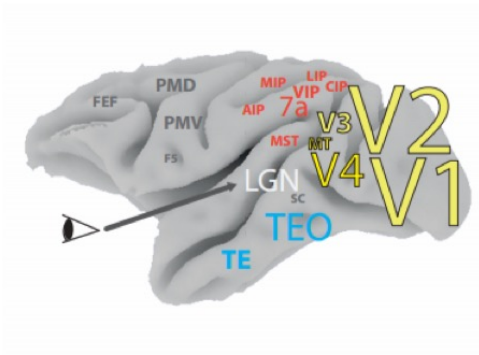
unimc
UNIVERSITÀ DI MACERATA
l'umanesimo che innova
DIPARTIMENTO DI
SCIENZE POLITICHE,
DELLA COMUNICAZIONE,
E DELLE RELAZIONI INTERNAZIONALI

Perché Deep?

- L'organizzazione gerarchica consente di condividere e riusare informazioni (un po' come la programmazione strutturata). Lungo la gerarchia è possibile selezionare feature specifiche e scartare dettagli inutili (al fine di massimizzare l'invarianza).
- Il nostro sistema visivo opera su una gerarchia di livelli (deep):



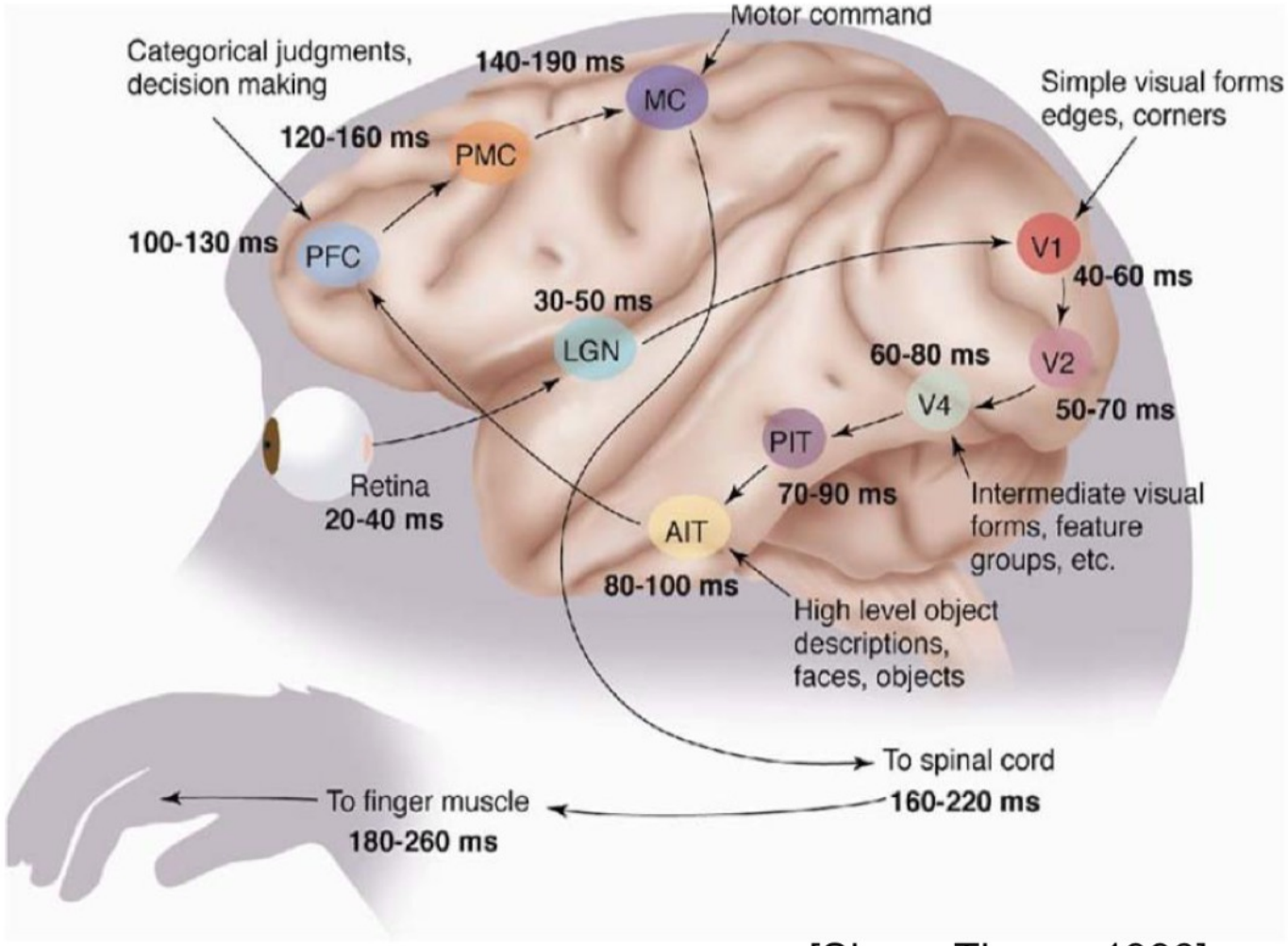
[Kruger et al. 2013]



Ma quanto Deep?

- 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.
  - «solo» una decina di livelli tra la retina e i muscoli attuatori (altrimenti saremmo troppo lenti a reagire agli stimoli).

Ma quanto Deep?

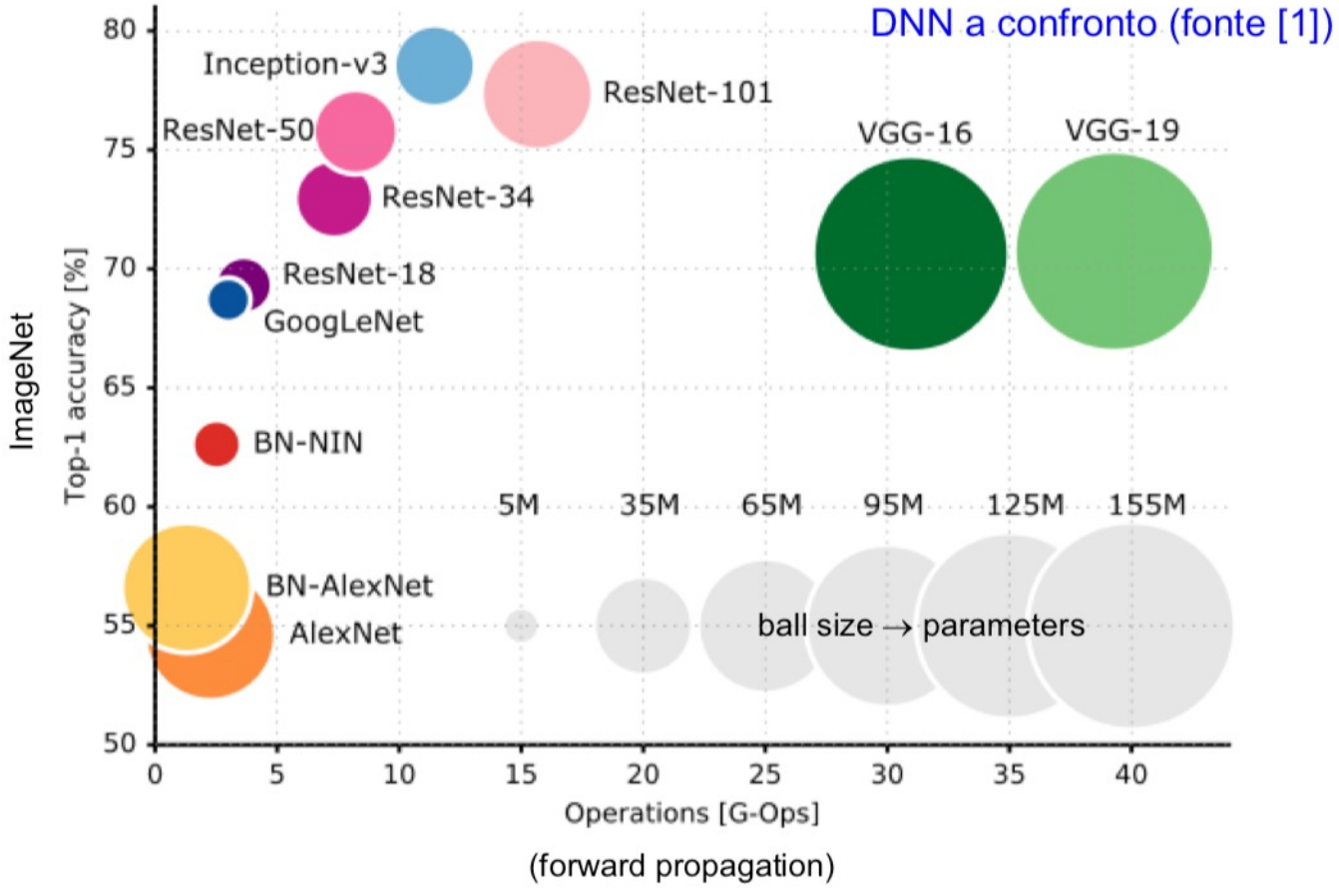


[Simon Thorpe 1996]

Livelli e Complessità

- La profondità (numero di livelli) è solo uno dei fattori di complessità. Numero di neuroni, di connessioni e di pesi caratterizzano altresì la complessità di una DNN.
- Maggiore è il numero di pesi (ovvero di parametri da apprendere) maggiore è la complessità del training. Al tempo stesso un elevato numero di neuroni (e connessioni) rende forward e back propagation più costosi, poiché aumenta il numero (G-Ops) di operazioni necessarie.
  - AlexNet: 8 livelli, 650K neuroni e 60M parametri
  - VGG-16: 16 livelli, 15M neuroni e 140M parametri
  - Corteccia umana: 10^{11} neuroni e 10^{14} sinapsi

Livelli e Complessità



[1] Canziani et al. 2016, *An Analysis of Deep Neural I Practical Applications*

Principali tipologie di DNN

- Modelli feedforward «discriminativi» per la classificazione (o regressione) con training prevalentemente supervisionato:
  - CNN - Convolutional Neural Network (o ConvNet)
  - FC DNN - Fully Connected DNN (MLP con almeno due livelli hidden)
  - HTM - Hierarchical Temporal Memory

Principali tipologie di DNN

- Training non superivisionato (modelli «generativi» addestrati a ricostruire l'input, utili per pre-training di altri modelli e per produrre feature salienti):
  - Stacked (de-noising) Auto-Encoders
  - RBM - Restricted Boltzmann Machine
  - DBN - Deep Belief Networks

Principali tipologie di DNN

- Modelli ricorrenti (utilizzati per sequenze, speech recognition, sentiment analysis, natural language processing,…):
  - RNN - Recurrent Neural Network
  - LSTM - Long Short-Term Memory
- Reinforcement learning (per apprendere comportamenti):
  - Deep Q-Learning

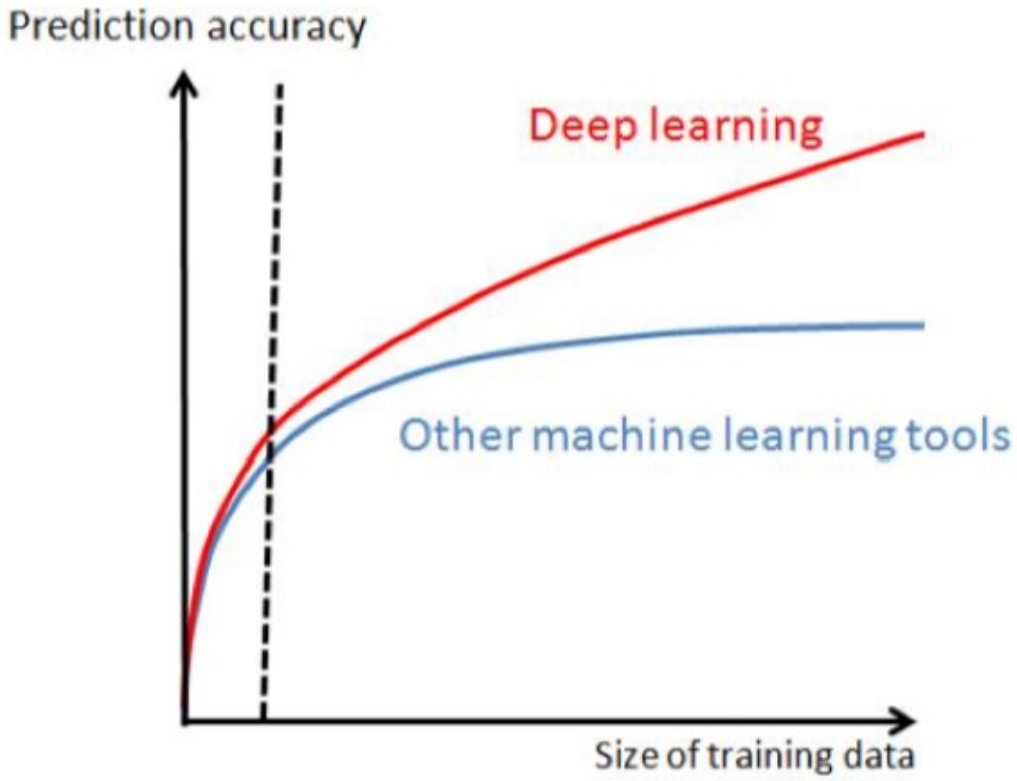
Ingredienti necessari

CNN ottengono già nel 1998 buone prestazioni in problemi di piccole dimensioni (es. riconoscimento caratteri, riconoscimento oggetti a bassa risoluzione), ma bisogna attendere il 2012 (AlexNet) per un radicale cambio di passo. AlexNet non introduce rilevanti innovazioni rispetto alle CNN di LeCun del 1998, ma alcune condizione al contorno sono nel frattempo cambiate:

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

Ingredienti necessari

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



Ingredienti necessari

- 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.

- Vanishing (or exploding) gradient: la retro propagazione del gradiente (fondamentale per backpropagation) è problematica su reti profonde se si utilizza la sigmoide come funzione di attivazione. Il problema può essere gestito con attivazione Relu (descritta in seguito) e migliore inizializzazione dei pesi (esempio: Xavier inizialization).

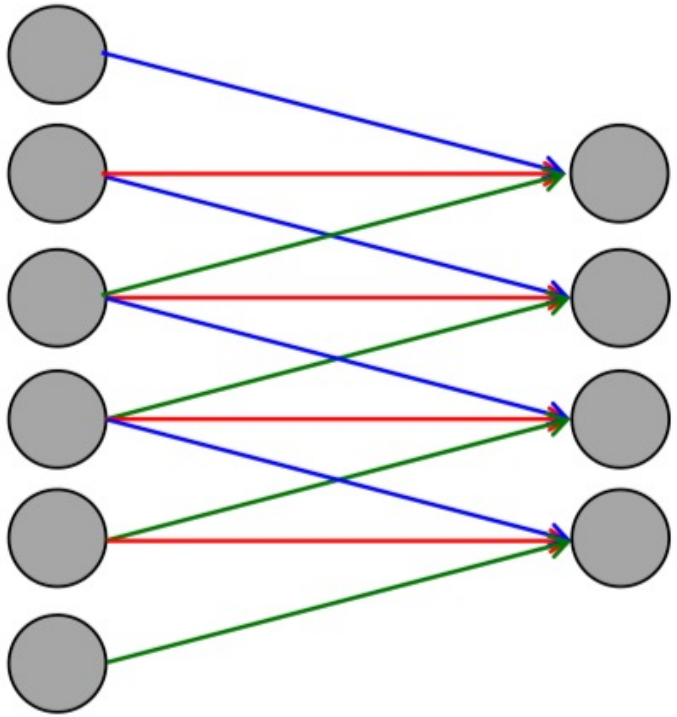
Da MLP a CNN

- Hubel & Wiesel (1962) scoprono la presenza, nella corteccia visiva del gatto, di due tipologie di neuroni:
  - Simple cells: agiscono come feature detector locali (fornendo selettività)
  - Complex cells: fondono (pooling) gli output di simple cell in un intorno (garantendo invarianza).
- Neocognitron (Fukushima, 1980) è una delle prime reti neurali che cerca di modellare questo comportamento.

Da MLP a CNN

- Convolutional Neural Networks (CNN) introdotte da LeCun et al., a partire dal 1998. Le principali differenze rispetto a MLP:
  - 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.

Da MLP a CNN



Esempio: ciascuno dei 4 neuroni a destra è connesso solo a 3 neuroni del livello precedente. I pesi sono condivisi (stesso colore stesso peso). In totale 12 connessioni e 3 pesi contro le 24 connessioni + 24 pesi di una equivalente porzione di MLP.

- alternanza livelli di feature extraction e pooling.

CNN: Architettura

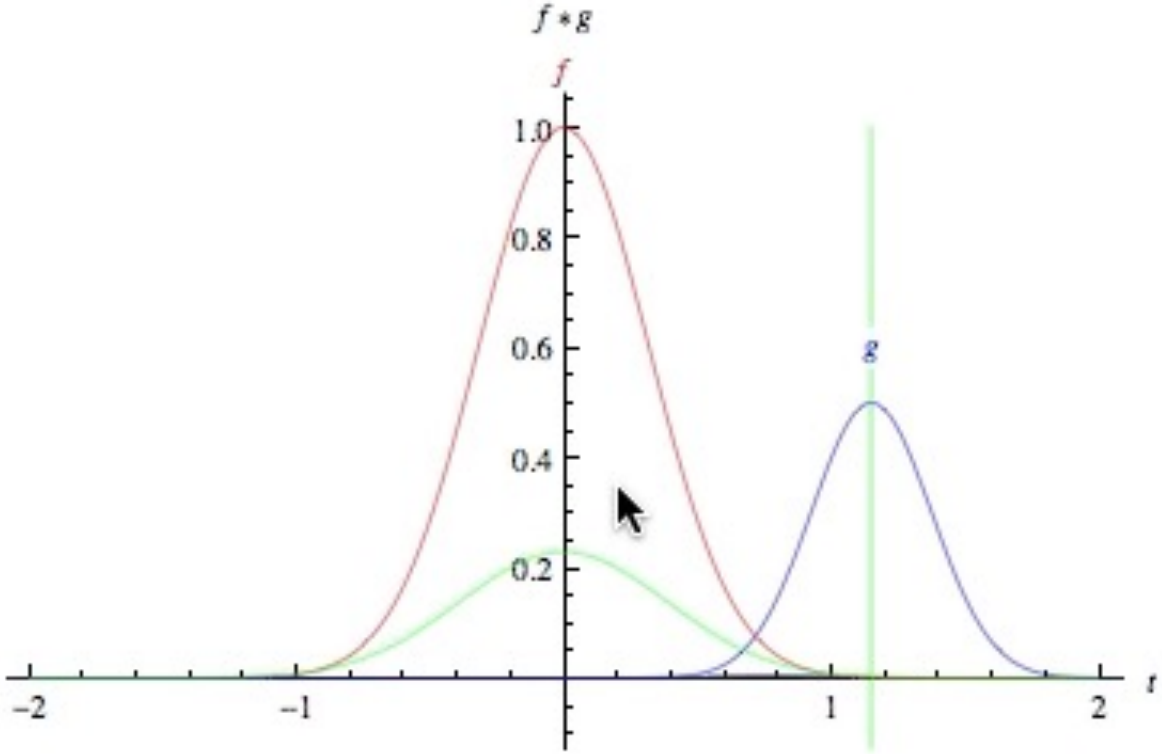
Esplicitamente progettate per processare immagini, per le quali elaborazione locale, pesi condivisi, e pooling non solo semplificano il modello, ma lo rendono più efficace rispetto a modelli fully connected. Possono essere utilizzate anche per altri tipi di pattern (es. speech).

- Architettura: una CNN è composta da una gerarchia di livelli. Il livello di input è direttamente collegato ai pixel dell'immagine, gli ultimi livelli sono generalmente fully-connected e operano come un classificatore MLP, mentre nei livelli intermedi si utilizzano connessioni locali e pesi condivisi.

Convolutional Neural Networks

Le **Convolutional Neural Networks** (**CNNs**) rappresentano un'architettura di reti neurali profonde di grande successo nelle applicazioni di visione artificiale e ampiamente utilizzate anche in applicazioni che processano media come audio e video.
L'applicazione più popolare di rete neurale convoluzionale resta comunque quella di *identificare,* da parte di un computer (e con una certa probabilità), *cosa un'immagine rappresenta*.

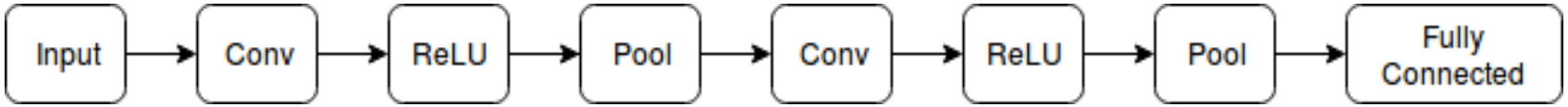
Convolutional Neural Networks



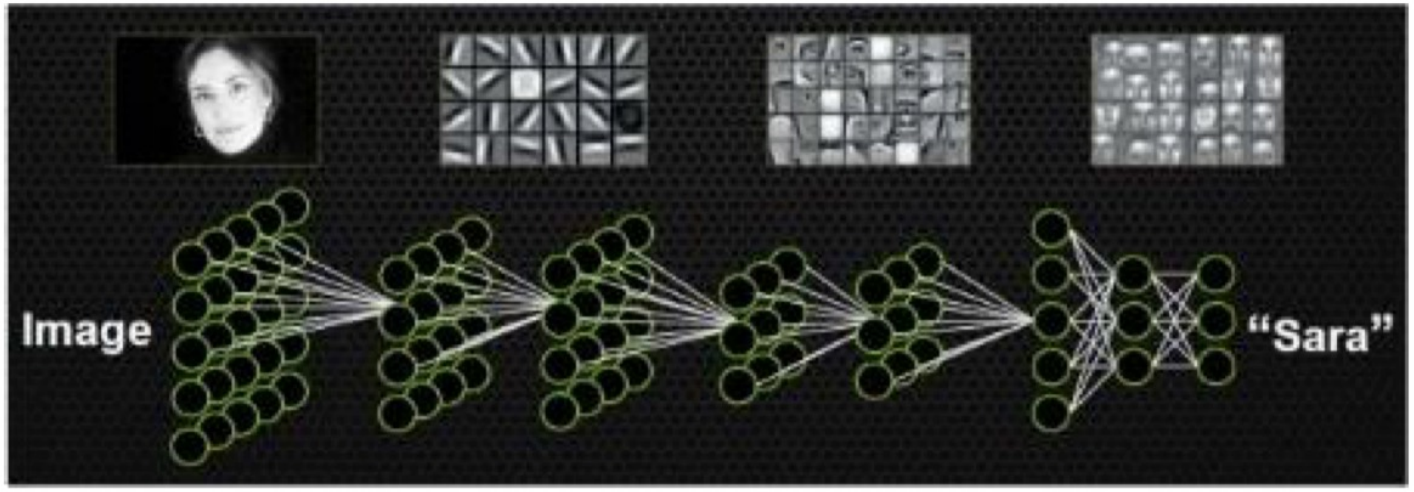
La curva verde mostra la convoluzione delle curve blu e rosse in funzione di t. La regione scura in funzione di t è precisamente la convoluzione.

Convolutional Neural Networks

L'architettura di una rete neurale convoluzionale può essere formata da:



CNN Architettura

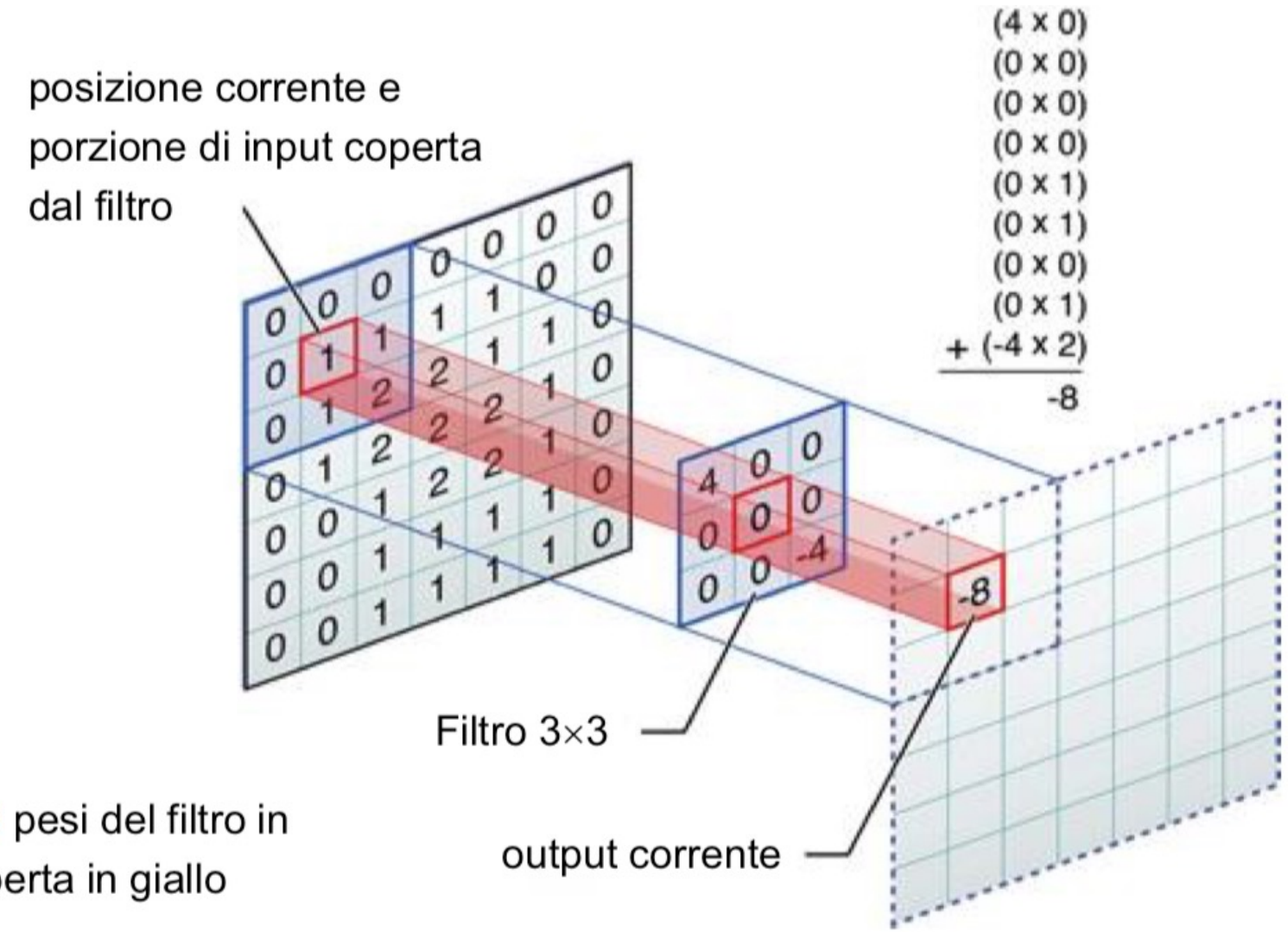


- Il campo visivo (receptive field) dei neuroni aumenta muovendosi verso l'alto nella gerarchia.
- Le connessioni locali e condivise fanno sì che i neuroni processino nello stesso modo porzioni diverse dell'immagine. Si tratta di un comportamento desiderato, in quando regioni diverse del campo visivo contengono lo stesso tipo di informazioni (bordi, spigoli, porzioni di oggetti, ecc.).

Convoluzione

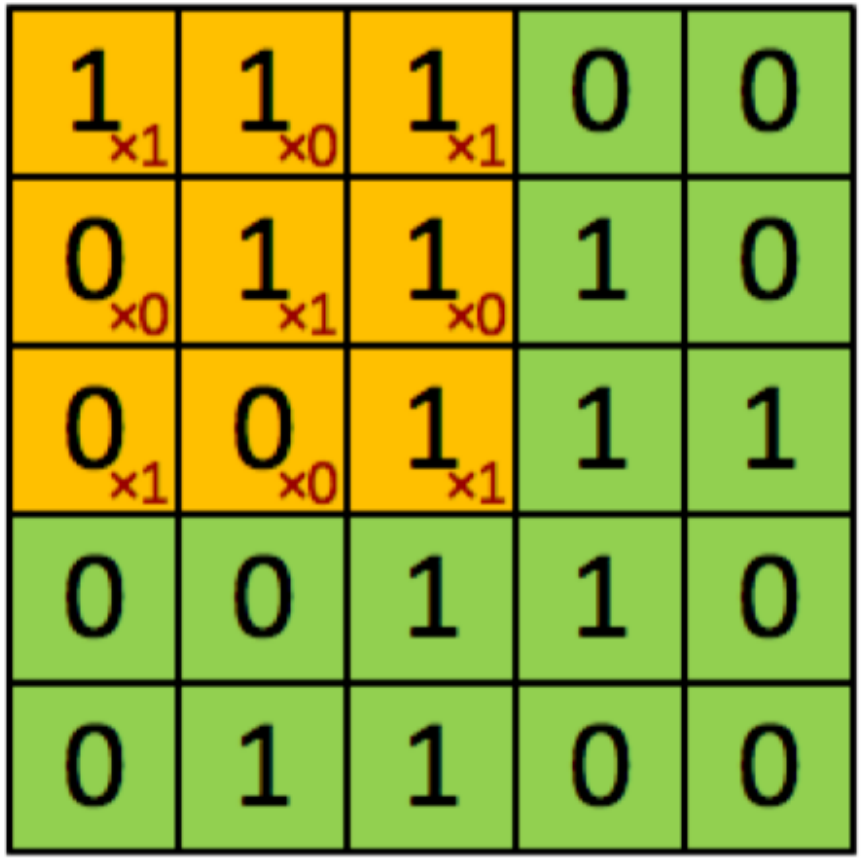
- La convoluzione è una delle più importanti operazioni di image processing attraverso la quale si applicano filtri digitali.
- Un filtro digitale (un piccola maschera 2D di pesi) è fatta scorrere sulle diverse posizioni di input; per ogni posizione viene generato un valore di output, eseguendo il prodotto scalare tra la maschera e la porzione dell'input coperta (entrambi trattati come vettori).

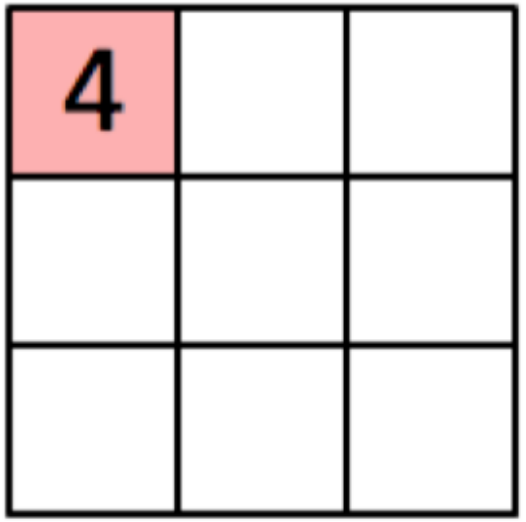
Convoluzione



Esempio (animato): pesi del filtro in rosso, porzione coperta in giallo

Convoluzione





Convoluzione

0	¼	0
¼	0	¼
0	¼	0

matrice di convoluzion

100	100	200	200
120	120	220	100
140	140	220	80
160	200	252	40

convoluzione

100	100	200	200
120	145	160	100
140	170	173	80
160	200	252	40

Convoluzione

0	¼	0
¼	0	¼
0	¼	0

matrice di convoluzione

100	100	200	200
120	120	220	100
140	140	220	80
160	200	252	40

convoluzione

100	100	200	200
120	145	160	100
140	170	173	80
160	200	252	40

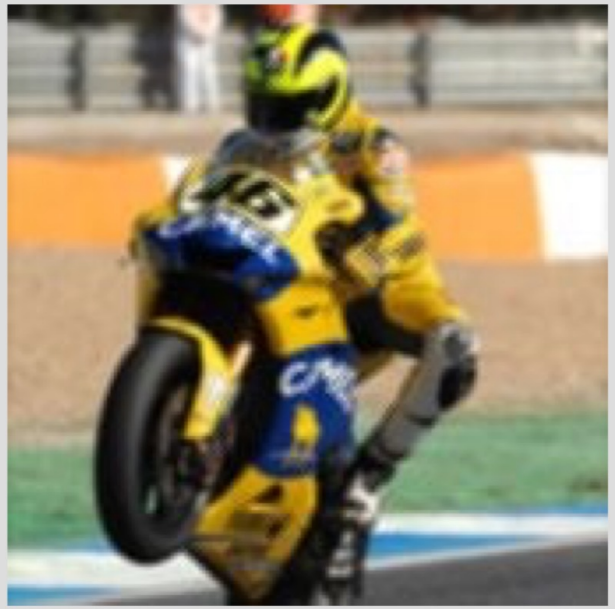
$$100\times0+100\times¼+200\times0+$$
$$120\times¼+120\times0+220\times¼+$$
$$140\times0+140\times¼+220\times0 \quad =145$$

Convoluzione

0	¼	0
¼	0	¼
0	¼	0

matrice di convoluzion e

convoluzione



Esempi applicazioni filtri a immagini

Immagine input

Filtro

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

Immagine output

Esempi applicazioni filtri a Immagini

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Esempi applicazioni filtri a Immagini

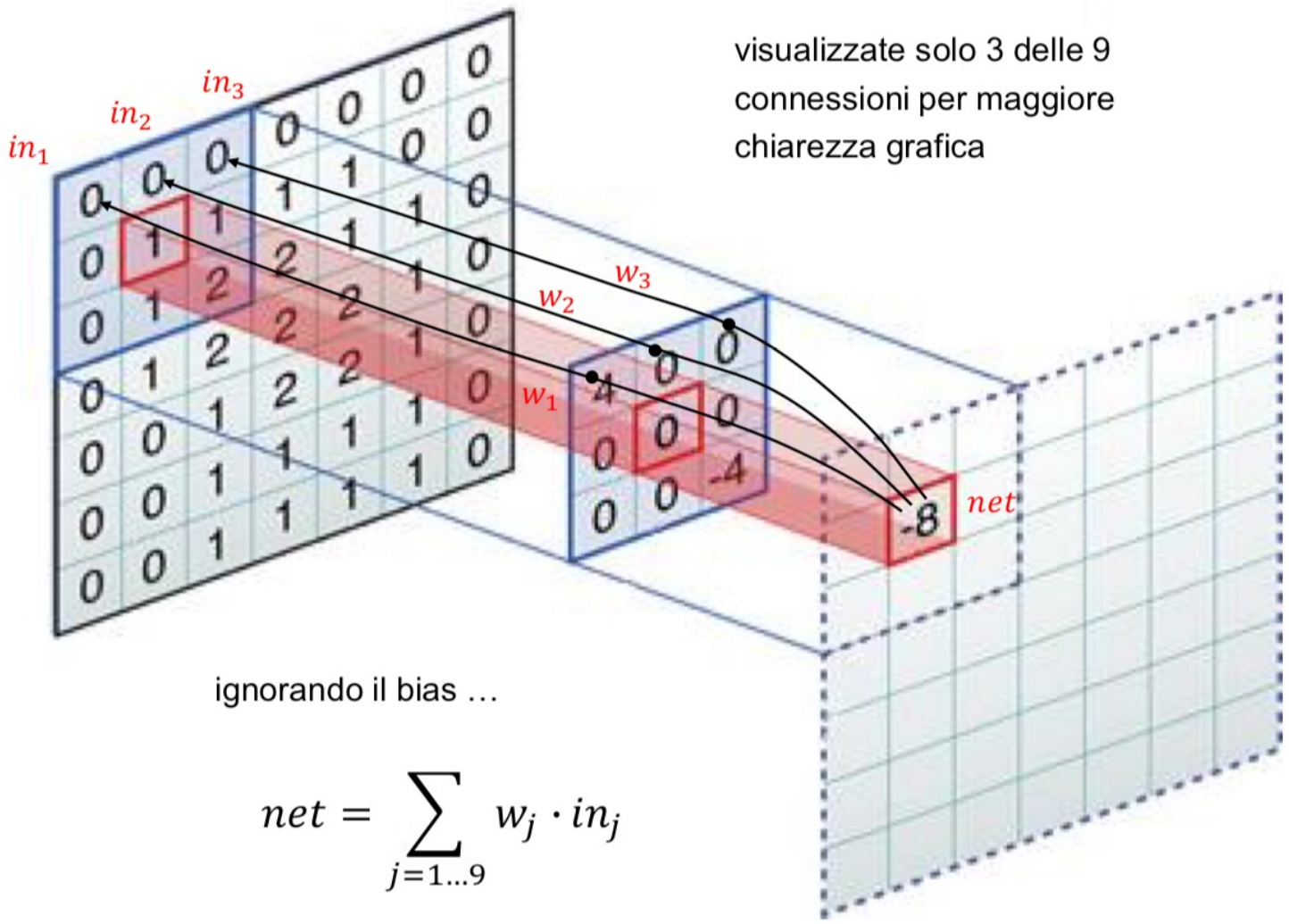
Dilata

Erodi

Neuroni come convolutori

- Consideriamo i pixel come neuroni e le due immagini di input e di output come livelli successivi di una rete. Dato un filtro 3×3, se colleghiamo un neurone ai 9 neuroni che esso «copre» nel livello precedente, e utilizziamo i pesi del filtro come pesi delle connessioni w, notiamo che un classico neurone (di una MLP) esegue di fatto una convoluzione.

Neuroni come convolutori

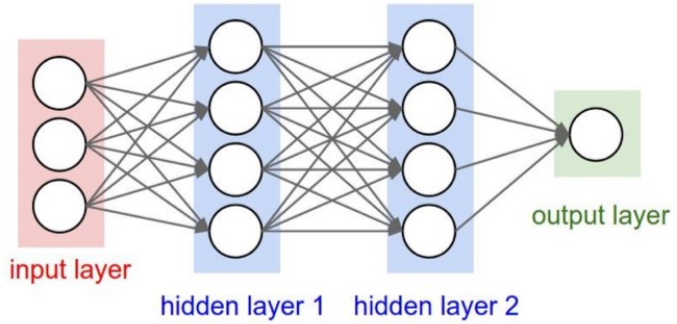


ignorando il bias …

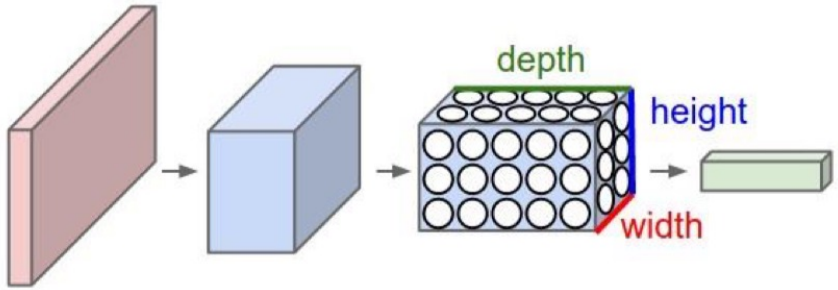
$$net = \sum_{j=1\ldots9} w_j \cdot in_j$$

CNN: Volumi

- Volumi: I neuroni di ciascun livello sono organizzati in griglie o volumi 3D (si tratta in realtà d i una notazione grafica utile per la comprensione delle connessioni locali).



MLP: organizzazione lineare dei neuroni nei livelli



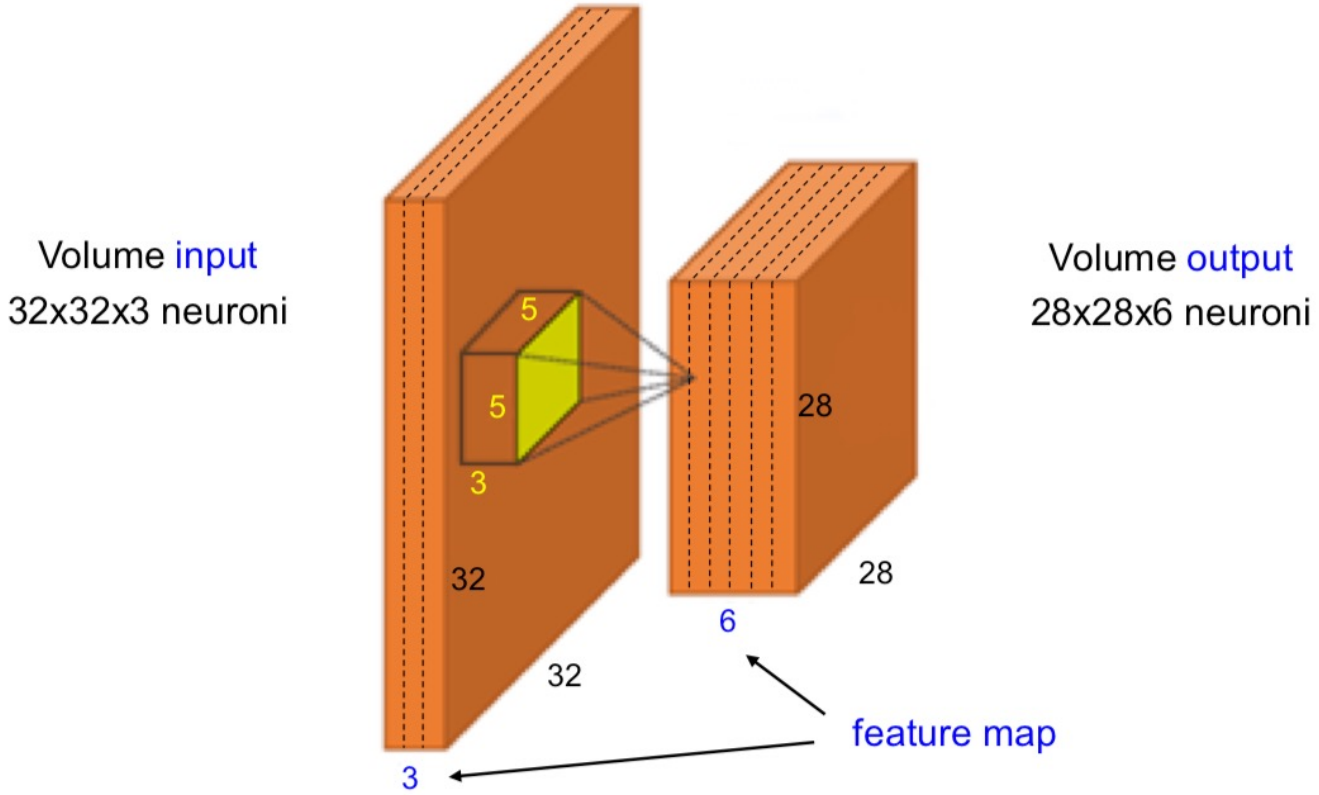
CNN: i livelli sono organizzati come griglie 3D di neuroni

CNN: Volumi

- sui piani width - height si conserva l'organizzazione spaziale «retinotipica» dell'immagine di input.
- la terza dimensione depth (come vedremo) individua le diverse feature map.

CNN: Convoluzione 3D

- Il filtro opera su una porzione del volume di input. Nell'esempio ogni neurone del volume di output è connesso a 5x5x3=75 neuroni del livello precedente.



CNN: Convoluzione 3D

- Ciascuna «fetta» di neuroni (stessa depth) denota una feature map. Nell'esempio troviamo:
  - 3 feature map (dimensione 32x32) nel volume di input.
  - 6 feature map (dimensione 28x28) nel volume di output.
- I pesi sono condivisi a livello di feature map. I neuroni di una stessa feature map processano porzioni diverse del volume di input nello stesso modo. Ogni feature map può essere vista come il risultato di uno specifico filtraggio dell'input (filtro fisso).
- Nell'esempio il numero di connessioni tra i due livelli è (28x28x6) x (5x5x3) = 352800, ma il numero totale di pesi è 6 x (5x5x3 + 1) = 456. *In analoga porzione di MLP quanti pesi?*

bias

CNN: Convoluzione 3D

- Quando un filtro 3D viene fatto scorrere sul volume di input, invece di spostarsi con passi unitari (di 1 neurone) si può utilizzare un passo (o $Stride$) maggiore. Questa operazione riduce la dimensione delle feature map nel volume di output e conseguentemente il numero di connessioni.
  - Sui livelli iniziali della rete per piccoli stride (es. 2, 4), è possibile ottenere un elevato guadagno in efficienza a discapito di una leggera penalizzazione in accuratezza.
- Ulteriore possibilità (per regolare la dimensione delle feature map) è quella di aggiungere un bordo (valori zero) al volume di input. Con il parametro $Padding$ si denota lo spessore (in pixel) del bordo.

Funzione di attivazione: ReLu

- Nelle reti MLP la funzione di attivazione (storicamente) più utilizzata è la sigmoide. Nelle reti profonde, l'utilizzo della sigmoide è problematico per la retro propagazione del gradiente (problema del vanishing gradient):
  - La derivata della sigmoide è tipicamente minore di 1 e l'applicazione della regola di derivazione a catena porta a moltiplicare molti termini minori di 1 con la conseguenza di ridurre parecchio i valori del gradiente nei livelli lontani dall'output. Per approfondimenti ed esempi:

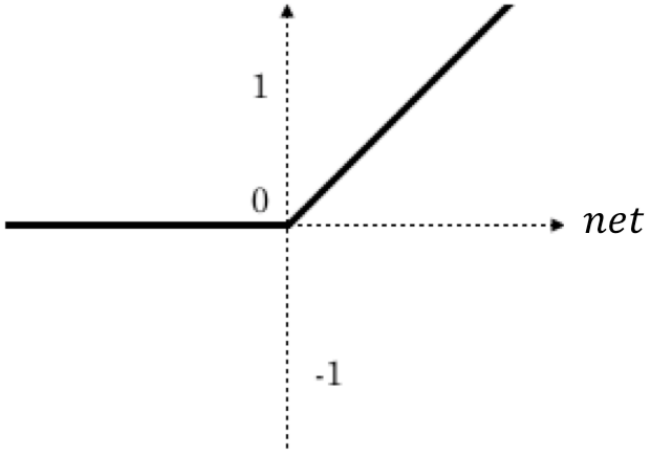
    http://neuralnetworksanddeeplearning.com/chap5.html

Funzione di attivazione: ReLu

- La sigmoide ha un comportamento saturante (allontanandosi dallo 0). Nelle regioni di saturazione la derivata vale 0 e pertanto il gradiente si annulla.

- L'utilizzo di Relu (Rectified Linear) come funzione di attivazione risolve il problema:

$$f(net) = max(0, net)$$



La derivata vale 0 per valori negativi o nulli di net e 1 per valori positivi

Per valori positivi nessuna saturazione

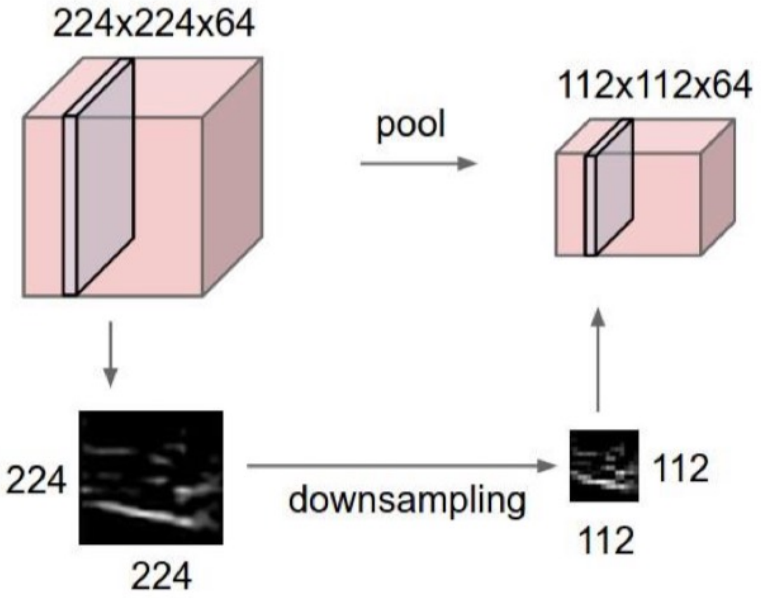
Porta ad attivazioni sparse (parte dei neuroni sono spenti) che possono conferire maggiore robustezza

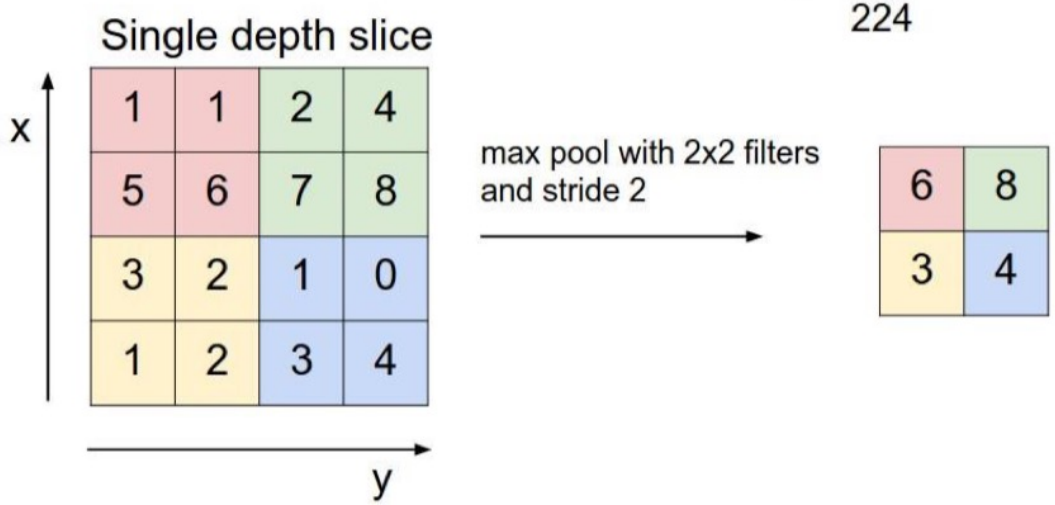
CNN: Pooling

- Un livello di pooling esegue un'aggregazione delle informazioni nel volume di input, generando feature map di dimensione inferiore. Obiettivo è conferire invarianza rispetto a semplici trasformazioni dell'input mantenendo al tempo stesso le informazioni significative ai fini della discriminazione dei pattern.
- L'aggregazione opera (generalmente) nell'ambito di ciascuna feature map, cosicché il numero di feature map nel volume di input e di output è lo stesso. Gli operatori di aggregazione più utilizzati sono la media (Avg) e il massimo (Max): entrambi «piuttosto» invarianti per piccole traslazioni. Questo tipo di aggregazione non ha parametri/pesi da apprendere.
- Nell'esempio un max-pooling con Filtri 2×2 e $Stride = 2$.

CNN: Pooling

L'equazione precedente per il calcolo di W_{out} è valida anche per pooling (considerando $Padding$ = 0)





Ricomponiamo i pezzi

- Esempio 1: Cifar-10 (Javascript running in the browser).

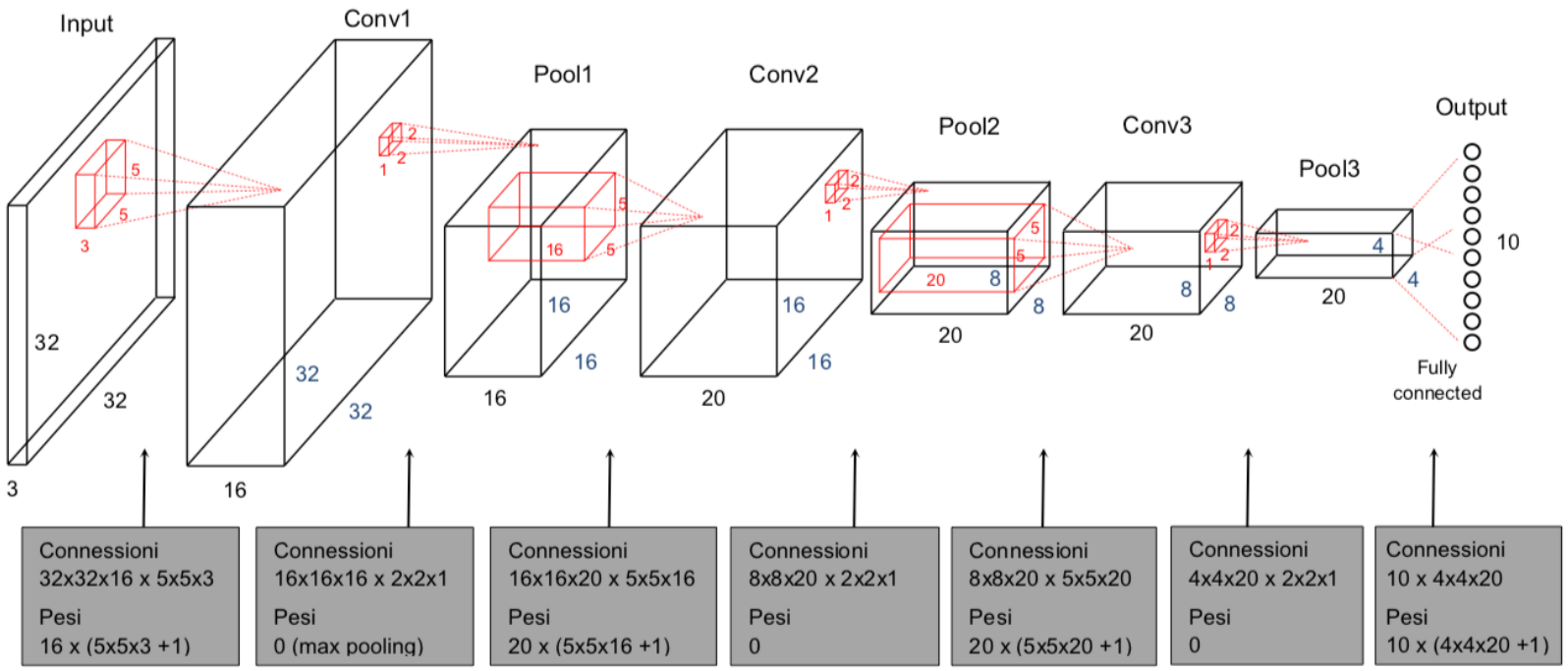
http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Architettura

- Input: Immagini RGB 32x32x3;
- Conv1: Filtri:5x5, FeatureMaps:16, stride:1, pad:2, attivazione: Relu
- Pool1: Tipo: Max, Filtri 2x2, stride:2
- Conv2: Filtri:5x5, FeatureMaps:20, stride:1, pad:2, attivazione: Relu
- Pool2: Tipo: Max, Filtri 2x2, stride:2
- Conv3: Filtri:5x5, FeatureMaps:20, stride:1, pad:2, attivazione: Relu
- Pool3: Tipo: Max, Filtri 2x2, stride:2
- Output: Softmax, NumClassi: 10

Ricomponiamo i pezzi

Disegniamo la rete e calcoliamo neuroni sui livelli, connessioni e pesi



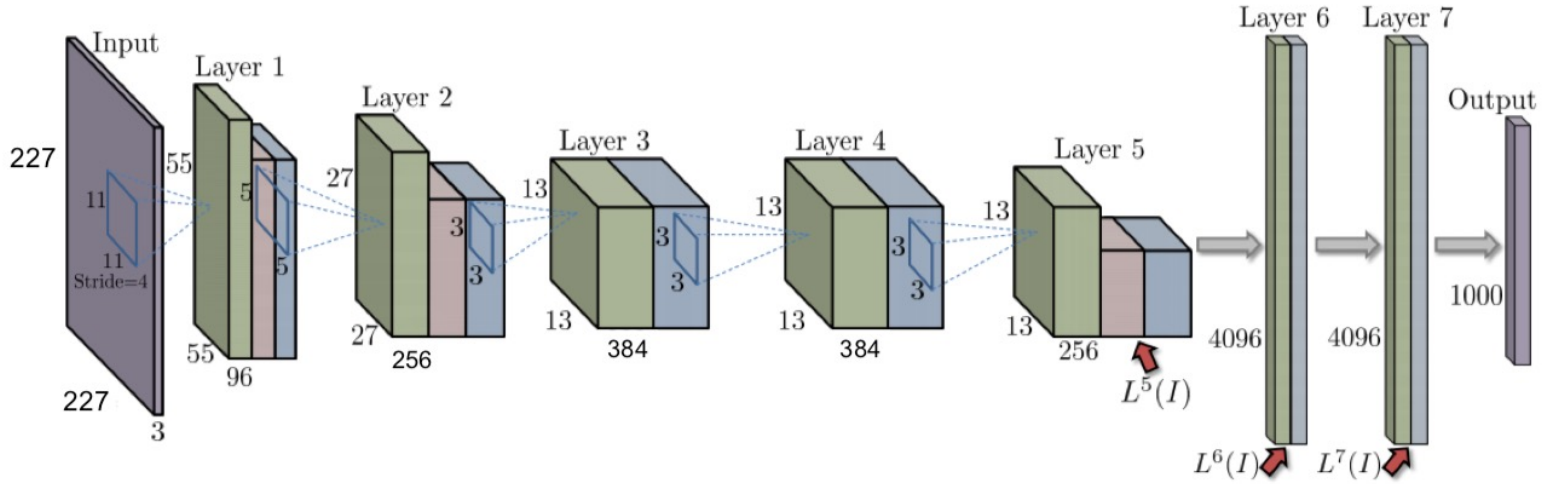
Neuroni totali: 31.562 (incluso livello input)

Connessioni totali: 3.942.784

Pesi totali: 22.466 (inclusi bias)

Ricomponiamo i pezzi

- Esempio 2: CaffeNet (AlexNet porting in Caffe).



Codici colore

- Viola: Input (immagini 227x227x3) e Output (1000 classi di ImageNet)
- Verde: Convoluzione
- Rosa: Pooling (max)
- Blu: Relu activation

Ricomponiamo i pezzi

Note

- Layer 6, 7 e 8: Fully connected
- Layer 8: Softmax (1000 classi)
- Stride: 4 per il primo livello di convoluzione, poi sempre 1
- Filtri: Dimensioni a scalare: da 11x11 a 3x3
- Feature Map: Numero crescente muovendosi verso l'output
- L^5, L^6, L^7, denotano feature riutilizzabili per altri problemi (vedi transfer-learning e [1]).
- Numero totale di parametri: 60M circa

[1] Babenko et al., *Neural Codes for Image Retrieval*, 2014.

In pratica

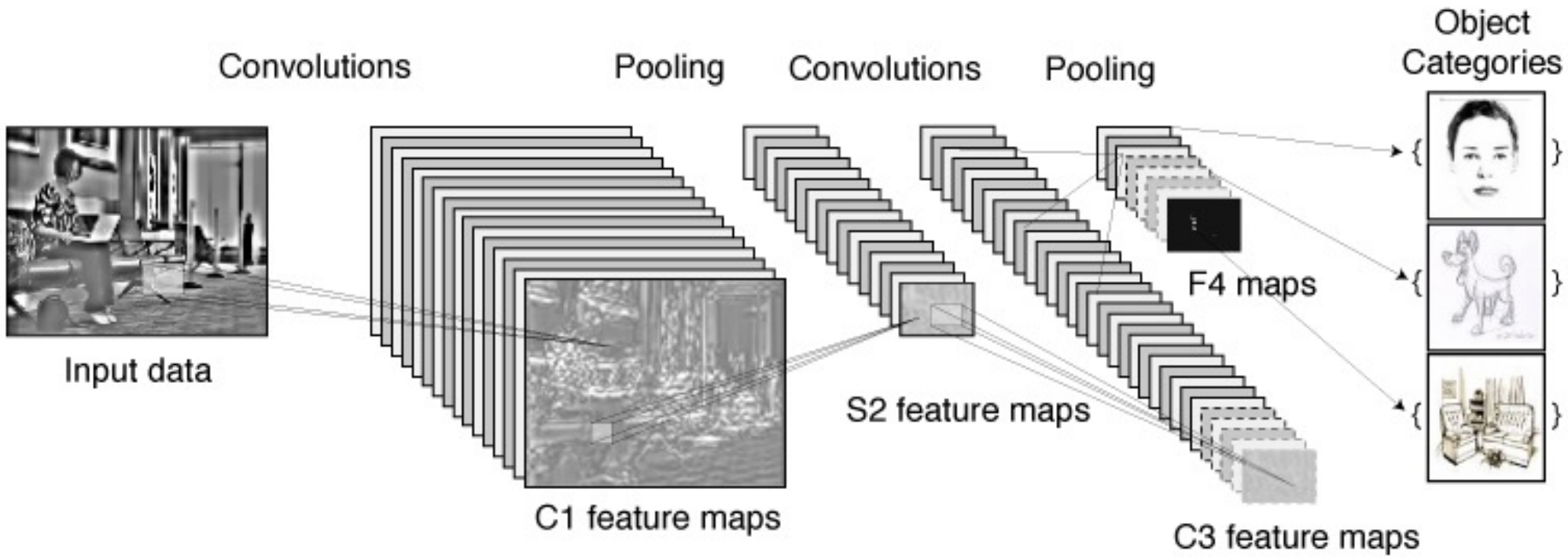
- L'implementazione di CNN (da zero) è certamente possibile. Il passo forward non è nemmeno complesso da codificare. D'altro canto la progettazione/sviluppo di software che consente:
  - il training/inference di architetture diverse a partire da una loro descrizione di alto livello (non embedded nel codice)
  - di effettuare il training con backpropagation del gradiente (rendendo disponibili le numerose varianti, parametrizzazioni e tricks disponibili)
  - ottimizzare la computazione su GPU (anche più di una)

richiede molto tempo/risorse per lo sviluppo/debug.

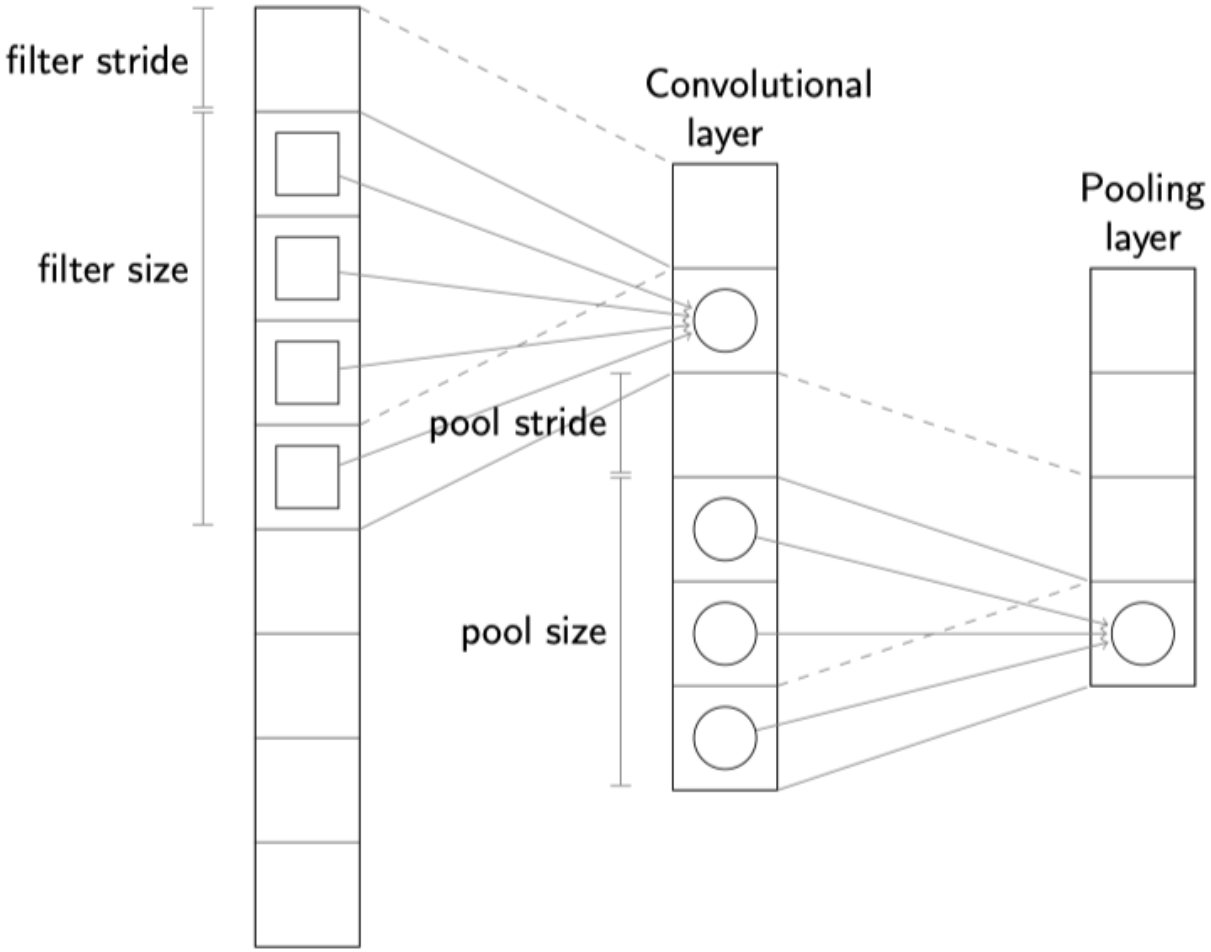
In Pratica

- Fortunatamente sono disponibili numerosi framework (spesso open-source) che consentono di operare su DNN. Tra i più utilizzati:
  - TensorFlow (Google) – *lo useremo in laboratorio*
  - PyTorch (Facebook)
  - Caffe (Berkley)
  - Theano (Bengio's group – Montreal) – oramai abbandonato
  - Digits (Nvidia) – *anche senza programmazione*

Convolutional Neural Network



Convolutional Neural Network



Convolutional Neural Network

In altre parole, abbiamo il:

- **Livello di input**: rappresenta l'insieme di numeri che rappresenta, per il computer, l'immagine da analizzare. Essa è rappresentata come un insieme di pixel. Ad esempio, 32 x 32 x 3 indica la larghezza (32), altezza (32) e profondità (3, i tre colori Red, Green e Blue nel formato RGB) dell'immagine.
- **Livello convoluzionale** (**Conv**): è il livello principale della rete. Il suo obiettivo è quello di individuare **schemi**, come ad esempio curve, angoli, circonferenze o quadrati raffigurati in un'immagine con elevata precisione. Sono più di uno, e ognuno di essi si concentra nella ricerca di queste caratteristiche nell'immagine iniziale. Maggiore è il loro numero e maggiore è la complessità della caratteristica che riescono ad individuare.

Convolutional Neural Network

- **Livello ReLU** (**Rectified Linear Units**): si pone l'obiettivo di annullare valori negativi ottenuti nei livelli precedenti e solitamente è posto dopo i livelli convoluzionali.

- **Livello Pool**: permette di identificare se la caratteristica di studio è presente nel livello precedente. Semplifica e rende più grezza l'immagine, mantenendo la caratteristica utilizzata dal livello convoluzionale.

- **Livello FC** (o **Fully connected**, completamente connesso): connette tutti i neuroni del livello precedente al fine di stabilire le varie classi identificative visualizzate nei precedenti livelli secondo una determinata probabilità. Ogni classe rappresenta una possibile risposta finale che il computer ti darà.

Convolutional Neural Network

Supponiamo che l'input di una CNN sia un'immagine rappresentante un 7 (esempio molto semplice, ma per far rendere l'idea), che accetta immagini dall'MNIST handwritten dataset.
Nel linguaggio macchina tale figura si può rappresentare tramite un array di 28 x 28 x 3 pixel.

Convolutional Neural Network

Per filtro generalmente, si intende una piccola matrice di poche righe e colonne che rappresenta una caratteristica (**feature**) che il livello convoluzionale vuole identificare, ad esempio le curve o una linea retta.

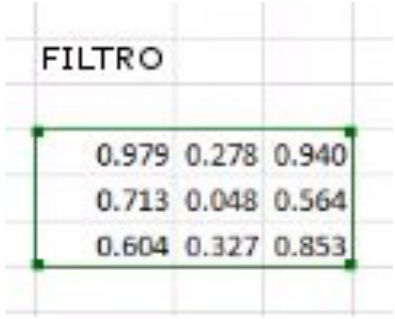
Inizialmente per i primi livelli si dice che il filtro rappresenta una caratteristica **di basso livello** perché identifica semplici oggetti come appunto curve o linee.

Per un livello convoluzionale il filtro identificherà le curve, per un altro linee orizzontali, per un altro ancora circonferenze, e così via negli ultimi livelli, fino a formare figure complesse che rappresentaranno oggetti più complicati.

In quest'ultimo caso si dice che il filtro rappresenta una caratteristica di **alto livello** perché identifica oggetti complessi, come il becco di un uccello, una mano o un volto.

Convolutional Neural Network

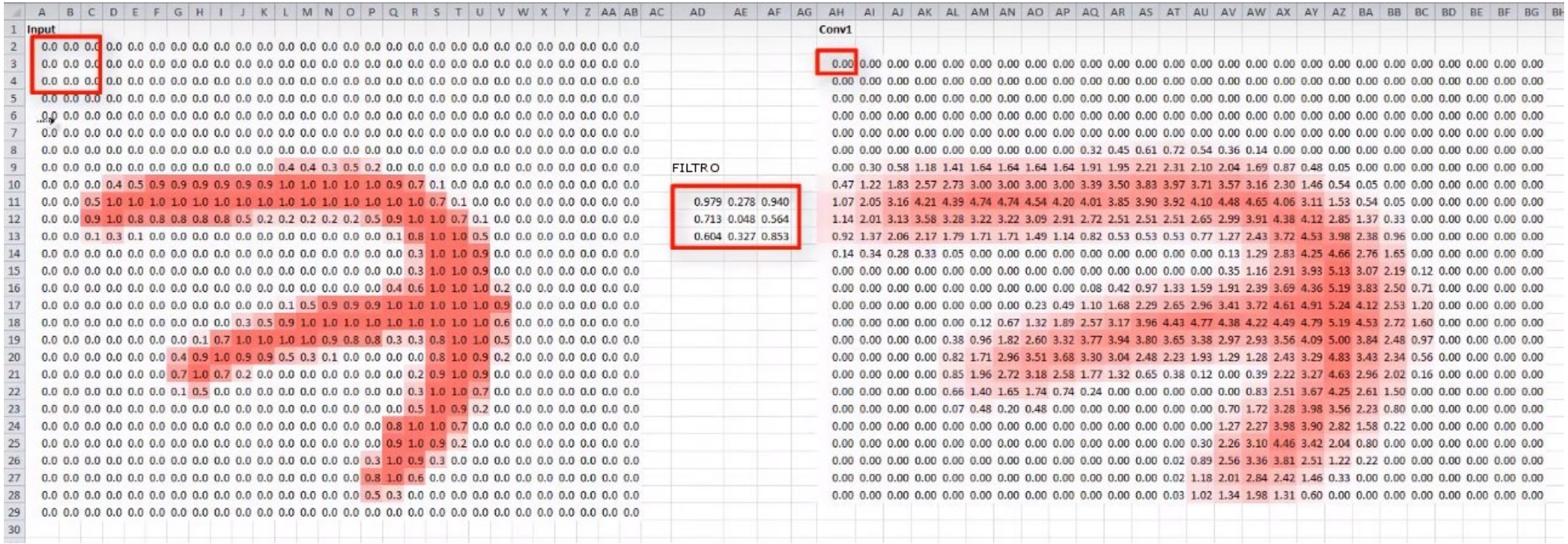
Ipotizziamo che il filtro sia un *rivelatore di curve*.
Identificata la caratteristica che il filtro identificherà nel livello convoluzionale, si decide la **dimensione del filtro** e il **numero di filtri** da utilizzare nel livello.
Nel nostro esempio si decide di utilizzare un filtro dalle dimensioni 3×3 (ossia 3 righe e 3 colonne), che per il primo livello convoluzionale assume valori casuali (detti anche **pesi**).



Convolutional Neural Network

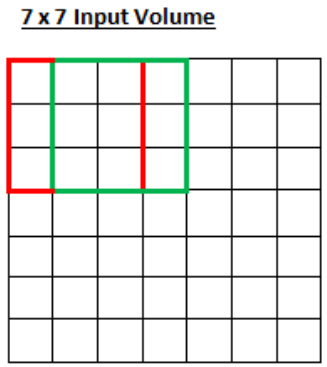
Per quanto riguarda il numero, ipotizziamo ora un solo filtro per semplicità, anche se in realtà quelli utilizzati sono diversi per ogni livello.
Successivamente si parte ad analizzare il cosiddetto **campo ricettivo**, che ha la stessa dimensione del filtro (quindi 3 x 3).
Esso viene inizialmente rappresentato dal primo blocco di pixel 3 x 3 in alto a sinistra del livello di input.
Il risultato, che otterremo in alto a sinistra nel livello successivo (**Conv1**) della rete neurale, si ottiene facendo un **prodotto scalare** dei valori del filtro con i valori di questo primo blocco (cioè facendo moltiplicazioni a livello di elemento per i pixel 3 x 3 del campo ricettivo e i pesi dei neuroni del filtro 3 x 3, e infine il risultato sarà la somma dei vari prodotti così ottenuti).
Questo ci dà un numero unico: tale valore sarà più alto in prossimità di curve, e più basso nel caso contrario.

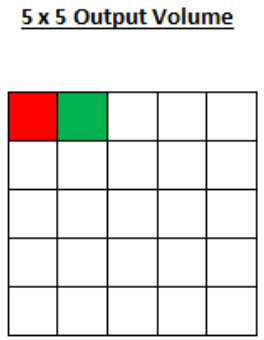
Convolutional Neural Network



Convolutional Neural Network

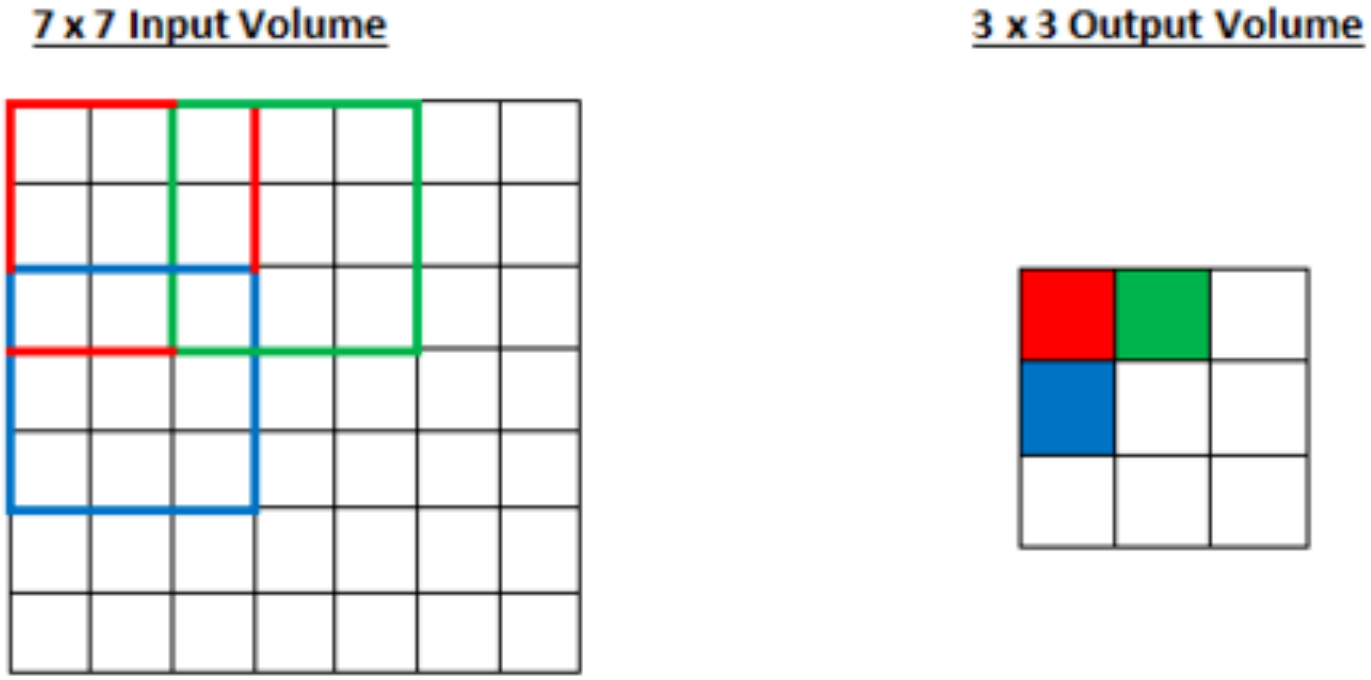
Nell'immagine sopra il primo risultato non è in prossimità di curve esso assumerà valore 0 (inoltre ogni pixel del volume di input assume valore nullo, quindi il risultato del prodotto scalare è pari a zero).
L'operazione appena vista va ripetuta per tutti i blocchi che l'immagine di input può contenere.
Di conseguenza, il campo ricettivo viene fatto spostare di un determinato **passo** (o **stride**) verso destra.
Ad esempio, ipotizzando di avere un volume di input 7 x 7 (numero di righe e di colonne del livello di input), un filtro 3 x 3 e un passo di 1, il campo ricettivo (sempre 3 x 3) si sposterà di una unità verso destra, come nell'immagine seguente:





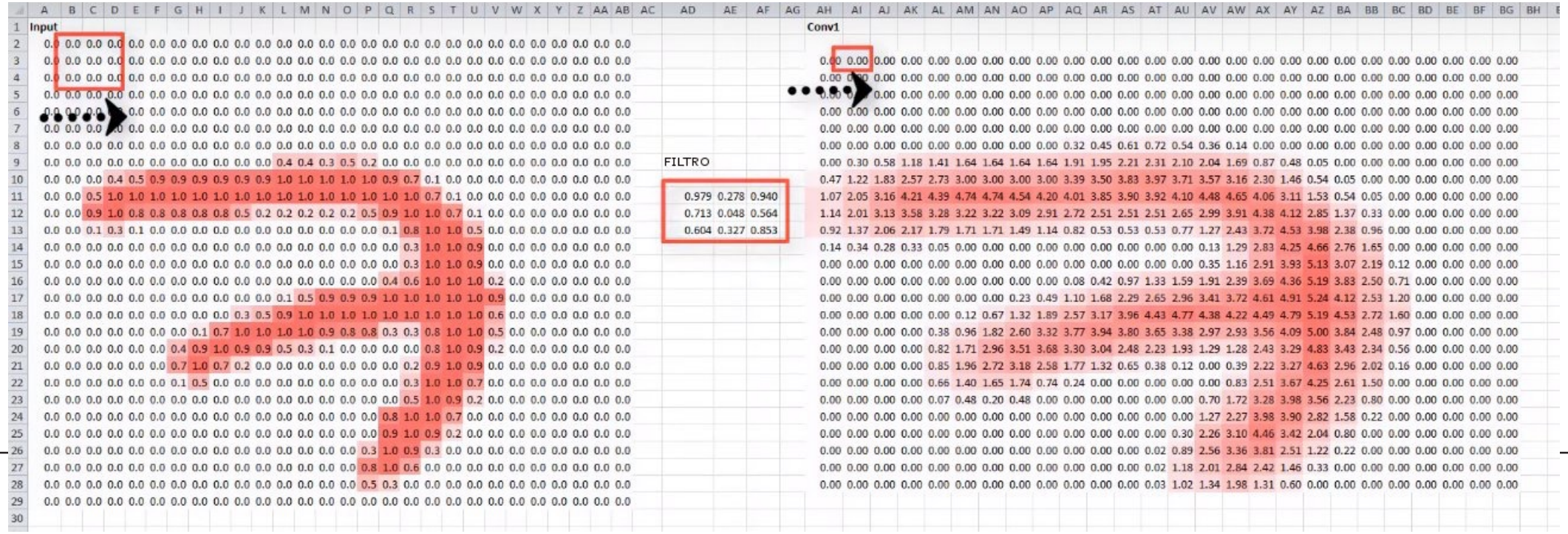
Convolutional Neural Network

Se il passo fosse di 2, la rappresentazione sarebbe la seguente:



Convolutional Neural Network

Il campo ricettivo si sta spostando di 2 unità ora e quindi il volume di output si restringe.
Per l'esempio iniziale, ipotizziamo un passo per ogni convoluzione (ossia passo = 1).
Il passaggio quindi sposta il campo ricettivo a destra di 1 unità, per analizzare il secondo blocco (e così via per tutti gli altri, fino a ricoprire l'intero volume di input).



Convolutional Neural Network

Dopo aver fatto scivolare il campo ricettivo su tutte le posizioni, otteniamo una matrice di numeri 26 x 26 x 1. La ragione per cui è 26 x 26 è che si ottengono 686 differenti pixel che un filtro di 3 x 3 può analizzare in un'immagine 28 x 28, come era quella proposta all'inizio.
La profondità, invece, risulta pari a 1 perché ho utilizzato un solo filtro in questo esempio.
L'insieme dei valori che si ottengono seguendo la procedura appena enunciata si dice **mappa di attivazione** (rappresentata in questo caso dalla matrice 26 x 26 x 1).
Se avessimo avuto più numeri di filtri, la stessa procedura andava ripetuta, per un numero di volte pari al numero dei filtri del livello (n), ottenendo n mappe di attivazione.

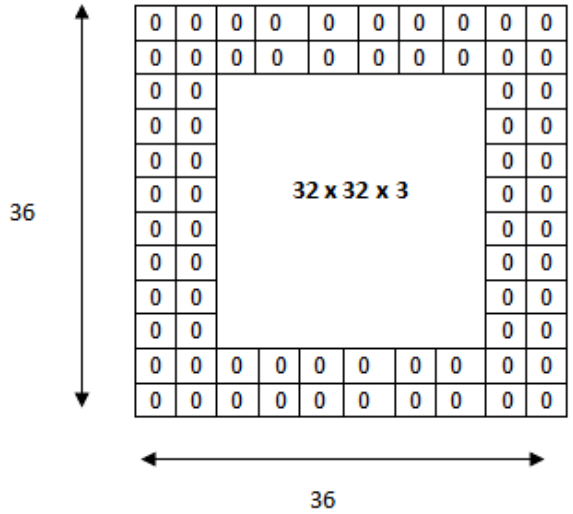
Convolutional Neural Network

Dimensione del filtro, passo e padding

Ci sono 4 parametri principali (definiti **iperparametri**) che influenzano il comportamento di un livello convoluzionale. Tre li abbiamo già visti: *passo*, *dimensione del filtro* e *numero*.
Il quarto è il **riempimento zero**, o **zero-padding** dall'inglese: esso identifica uno strato da apporre al volume di input iniziale al fine di evitare di perdere alcune informazioni al passaggio da un livello a un altro.
Vediamo un esempio.
Cosa succede quando si applicano tre filtri 5 x 5 x 3 a un volume di input 32 x 32 x 3?
Il volume di output sarebbe 28 x 28 x 3. Il motivo è sempre che si ottengono 784 differenti pixel che un filtro 5 x 5 può coprire in un'immagine 32 x 32. La profondità è data dal numero di filtri, quindi 3.
Si noti che le dimensioni spaziali diminuiscono.
Man mano che continuiamo ad applicare i livelli convoluzionali, la dimensione del volume diminuirà più velocemente di quanto vorremmo.

Convolutional Neural Network

Nei primi strati della nostra rete, però, è bene conservare il maggior numero di informazioni sul volume di input originale in modo da poter estrarre tali caratteristiche di basso livello, che altrimenti andrebbero perse e sarebbe poi impossibile recuperarle nei livelli successivi.
Quindi, quello che si vuole è che il volume di output rimanga 32 x 32 x 3.
Per fare ciò, si applica uno spessore zero di dimensione 2 al primo livello.
In altre parole, si riempe il volume di input con zeri attorno al bordo: otterremo così un volume di input 36 x 36 x 3.



Convolutional Neural Network

Per scegliere gli iperparametri di una rete neurale convoluzionale, non esiste uno standard stabilito che viene utilizzato da tutti i ricercatori, in quanto la rete dipende in gran parte dal tipo di dati a disposizione.

Convolutional Neural Network

Livello ReLU (Rectified Linear Units)

Quando attraversiamo un altro livello convoluzionale, l'output del primo livello convoluzionale diventa l'input del secondo livello.
Di conseguenza, l'output del livello convoluzionale diventa l'input del livello ReLU, che solitamente è collocato subito dopo il livello convoluzionale.
Il livello ReLU rappresenta un **livello non lineare**, il cui scopo è quello di introdurre la non linearità a un sistema che sostanzialmente sta calcolando operazioni lineari durante i livelli convoluzionali (tramite il prodotto scalare tra il filtro e il campo ricettivo).

Convolutional Neural Network

Livello ReLU (Rectified Linear Units)

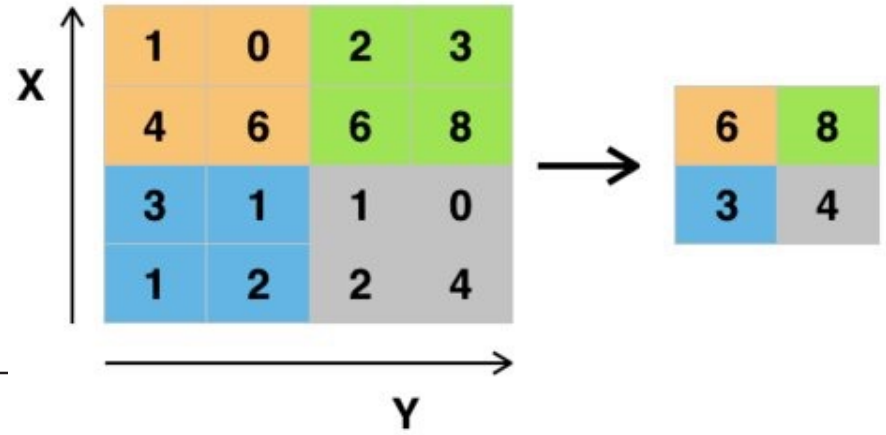
I ricercatori hanno scoperto che con questi livelli le reti neurali convoluzionali funzionano molto meglio perché la rete è in grado di allenarsi molto più velocemente (a causa dell'efficienza computazionale) senza impattare significativamente sull'accuratezza dei risultati.
Il livello ReLU applica la funzione **f (x) = max (0, x)** a tutti i valori nel volume di input.
In parole semplici, questo livello annulla tutti i valori negativi, aumentando le proprietà non lineari del modello e della rete globale senza influenzare i campi ricettivi del livello convoluzionale.

Convolutional Neural Network

Livello Pool

Dopo alcuni livelli ReLU, i programmatori possono scegliere di applicare un livello pool. Questo livello può essere eseguito secondo diversi risultati: il **massimo** è il più popolare, e per questo si considererà quello.

Secondo questa classificazione il livello richiede un filtro (normalmente di dimensione 2×2) e un passo della stessa lunghezza. Quindi lo applica al volume di input del livello precedente e genera il numero massimo in ogni campo ricettivo attorno al quale il filtro ruota.



Convolutional Neural Network

Ad esempio, per il primo riquadro in rosa: 6 risulta il massimo tra il primo blocco (1, 0, 4 e 6), e così viene riportato nella prima posizione. E lo stesso per le altre 3.
Il ragionamento intuitivo alla base di questo livello è che una volta che sappiamo che una caratteristica specifica è nel volume di input originale (ci sarà un alto valore di attivazione) e la sua posizione esatta non è importante quanto la sua posizione relativa rispetto alle altre caratteristiche.
Questo livello riduce drasticamente la dimensione spaziale (l'altezza e la larghezza cambiano ma non la profondità) del volume di input ed i requisiti computazionali per i livelli futuri.

Convolutional Neural Network

Livello FC (Full connection)

Solitamente è l'ultimo livello di una rete neurale convoluzionale.
Questo livello prende fondamentalmente un volume di input (qualunque sia l'output del livello convoluzionale o del ReLU o del livello pool che lo precede) e genera un vettore N dimensionale in cui N è il numero di classi tra cui il programma deve scegliere.

Convolutional Neural Network

Livello FC (Full connection)

Ad esempio, se si desidera un programma di classificazione delle cifre, N sarà 10 poiché 10 sono le cifre (0,1,2,3,4,5,6,7,8 e 9). Ogni numero in questo vettore di dimensione N rappresenta la probabilità di una certa classe.
Se il vettore risultante per un programma di classificazione di cifre è

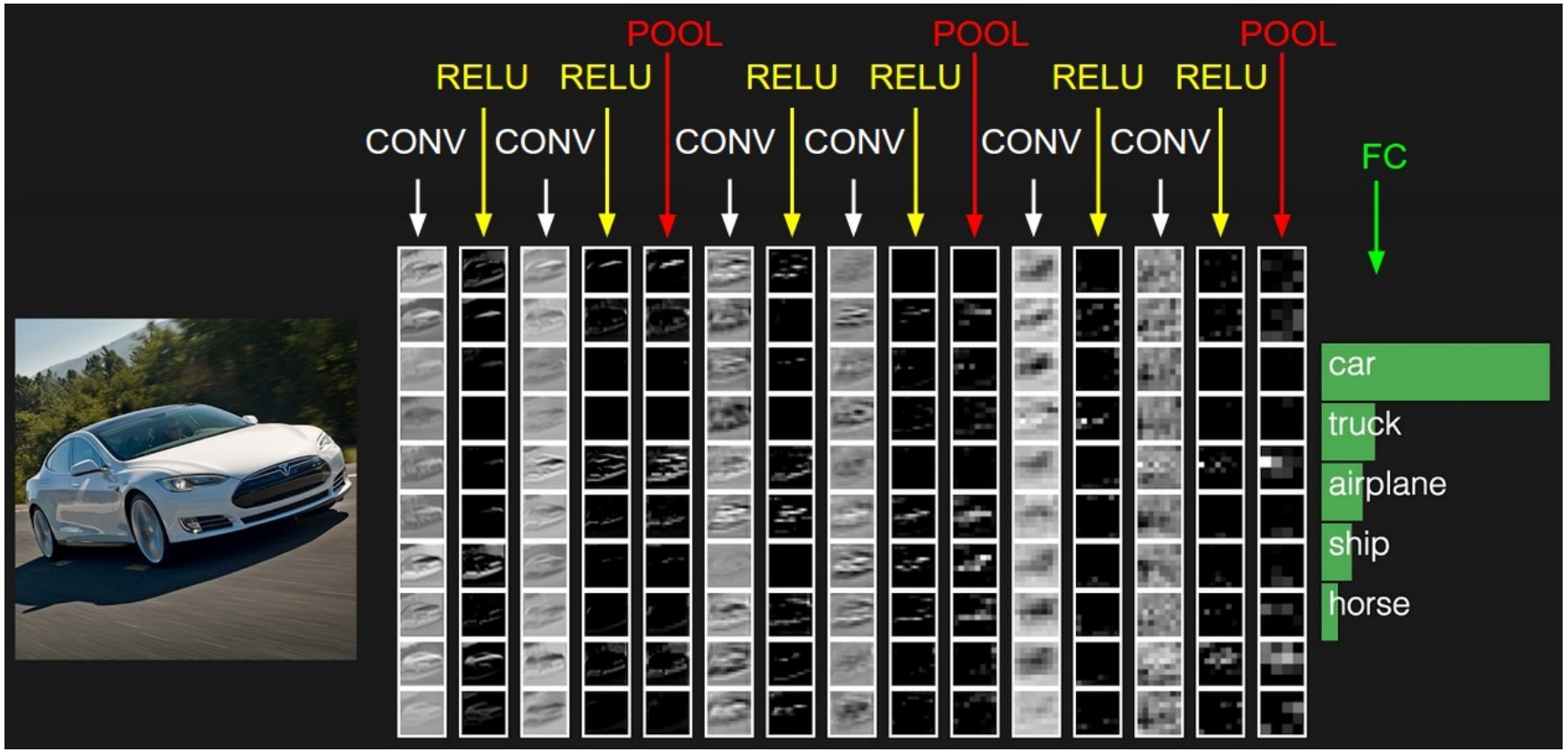
[0 0 15 10 0 0 0 65 0 10]

allora questo rappresenta una probabilità del 15% che l'immagine sia 2, una probabilità del 10% che l'immagine rappresenti un 3, una probabilità del 65% che l'immagine sia un 7 e una probabilità del 10% che l'immagine sia un 9 (tutti gli altri numeri hanno probabilità nulla di essere scelti).

Convolutional Neural Network

Il modo in cui questo livello completamente connesso funziona è che guarda l'output del livello precedente (che dovrebbe rappresentare le mappe di attivazione di caratteristiche di alto livello) e determina quali caratteristiche sono maggiormente correlate a una particolare classe.
Ad esempio, se il programma prevede che un'immagine sia un cane, avrà valori elevati nelle mappe di attivazione che rappresentano caratteristiche di alto livello come una zampa o 4 zampe, o il muso, e così via.
Analogamente, se il programma prevede che un'immagine sia un uccello, avrà valori alti nelle mappe di attivazione che rappresentano caratteristiche di alto livello come le ali o un becco, e così via.
Fondamentalmente, un livello FC guarda quali caratteristiche di alto livello sono maggiormente correlate ad una particolare classe e calcola i prodotti tra i pesi e il livello precedente per ottenere le probabilità corrette per le diverse classi.

Convolutional Neural Network



Convolutional Neural Network

Limiti di un MLP

Non tiene conto della struttura spaziale di un segnale all'ingresso della rete
Elevato numero di connessioni, a parità di dimensioni dei segnali di ingresso e uscita del layer (N x M pesi)

Vantaggi CNN

- Strati convoluzionali
- Campi ricettivi locali (Kernel)
- Pesi e bias condivisi
- Strati di Pooling

Convolutional Neural Network

layer	memoria richiesta	tempo di esecuzione
fully connected	elevata	basso
convoluzionale	bassa	alto

Convolutional Neural Network

I layer superficiali estraggono caratteristiche di basso livello (texture level extraction)

- estrazione contorni
- intensità media

I layer profondi estraggono caratteristiche di alto livello (object level extraction)

- object detection

Training e Transfer Learning

- Il training di CNN complesse (es. AlexNet) su dataset di grandi dimensioni (es. ImageNet) può richiedere giorni/settimane di tempo macchina anche se eseguito su GPU.
  - Fortunatamente, una volta che la rete è stata addestrata, il tempo richiesto per la classificazione di un nuovo pattern (propagazione forward) è in genere veloce (es. 10-100 ms).
- Inoltre il training di una CNN su un nuovo problema, richiede un training set etichettato di notevoli dimensioni (spesso non disponibile). In alternativa al training da zero, possiamo perseguire due strade (Transfer Learning):

Training e Transfer Learning

- Fine-Tuning: si parte con una rete pre-trained addestrata su un problema simile e:

1. si rimpiazza il livello di output con un nuovo livello di output softmax (adeguando il numero di classi).
2. come valori iniziali dei pesi si utilizzano quelli della rete pre-trained, tranne che per le connessioni tra il penultimo e ultimo livello i cui pesi sono inizializzati random.
3. si eseguono nuove iterazioni di addestramento (SGD) per ottimizzare i pesi rispetto alle peculiarità del nuovo dataset (non è necessario che sia di grandi dimensioni).

Fine Tuning e Transfer Learning

In certe situazioni occorre addestrare grandi modelli su piccoli dataset. In queste situazioni è necessario partire da modelli pre-addestrati su altri dataset.

Fine tuning:

- Training a partire da modelli pre-addestrati
- convergenza rapida (modelli pre-addestrati con iperparametri già ottimizzati)

Transfer learning:

- Estrazione delle feature da modelli pre-addestrati.
- Dopo di che, le feature estratte possono essere usate per la classificazione (SVM, kNN, Decision Tree, MLP,. . .)

Fine Tuning

Scelta dei modelli pre-addestrati

Dimensioni dataset	Modello addestrato su	
	dataset simile	**dataset differente**
piccolo	Fine tuning dell'output layer	Fine tuning dei layer profondi e dell'output layer
grande	Fine tuning di tutto il modello	Training da zero

Fine Tuning: quando?

In generale, se il nostro dataset non è drasticamente diverso nel contesto dal dataset su cui è addestrato il modello preaddesignato, dovremmo andare per la messa a punto.

Una rete preaddesignata su un set di dati ampio e diversificato come ImageNet cattura caratteristiche universali come curve e bordi nei suoi primi strati, che sono rilevanti e utili per la maggior parte dei problemi di classificazione.

Naturalmente, se il nostro set di dati rappresenta un dominio molto specifico, ad esempio, immagini mediche o caratteri scritti a mano in cinese, e che non è possibile trovare reti preaddesignate su tale dominio, dovremmo considerare la possibilità di addestrare la rete da zero.

Fine Tuning: quando?

Un'altra preoccupazione è che se il nostro dataset è piccolo, la messa a punto della rete preaddesignata su un piccolo dataset potrebbe portare ad un overfitting, specialmente se gli ultimi livelli della rete sono completamente connessi, come nel caso della rete VGG.

In genere, se abbiamo qualche migliaio di campioni grezzi, con le comuni strategie di incremento dei dati implementate (traduzione, rotazione, flipping, ecc.), la messa a punto ci darà di solito un risultato migliore.

Fine Tuning: quando?

Se il nostro dataset è davvero piccolo, diciamo meno di mille campioni, un approccio migliore è quello di prendere l'output dello strato intermedio prima dei livelli completamente collegati come caratteristiche (caratteristiche del collo di bottiglia) e addestrare un classificatore lineare (ad esempio SVM) su di esso.
SVM è particolarmente adatto a tracciare i limiti decisionali su un piccolo set di dati.

Fine Tuning: come?

Vediamo le linee guida generali:

1. La pratica comune è quella di troncare l'ultimo strato (strato softmax) della rete preallestita e sostituirlo con il nostro nuovo strato softmax che sono rilevanti per il nostro problema. Per esempio, la rete preaddesignata su ImageNet viene fornita con uno strato softmax con 1000 categorie. Se il nostro compito è una classificazione in 10 categorie, il nuovo strato softmax della rete sarà di 10 categorie invece di 1000 categorie. Poi si esegue di nuovo la propagazione sulla rete per mettere a punto i pesi preaddestrati. Assicuratevi che la convalida incrociata sia eseguita in modo che la rete sia in grado di generalizzare bene.

Fine Tuning: come?

2. Utilizzare un tasso di apprendimento inferiore per addestrare la rete. Dal momento che ci aspettiamo che i pesi preaddestrati siano già abbastanza buoni rispetto ai pesi inizializzati casualmente, non vogliamo distorcerli troppo velocemente e troppo.
Una pratica comune è rendere il tasso di apprendimento iniziale 10 volte inferiore a quello utilizzato per l'allenamento con i graffi.

Fine Tuning: come?

3. E' anche una pratica comune congelare i pesi dei primi strati della rete preaddesignata. Questo perché i primi strati catturano caratteristiche universali come curve e bordi che sono rilevanti anche per il nostro nuovo problema.
Vogliamo mantenere intatti questi pesi. Invece, faremo in modo che la rete si concentri sull'apprendimento delle caratteristiche specifiche del set di dati nei livelli successivi.

Fine Tuning

Dove troviamo reti preaddestrate?

Dipende dal framework di Deep Leanring. Per i framework più diffusi come Caffe, Keras, TensorFlow, Torch, MxNet, ecc., i rispettivi contributori solitamente tengono una lista dei modelli Covnet all'avanguardia (VGGG, Inception, ResNet, ResNet, ecc.) con le loro implementazioni e pesi pre-formati su un dataset comune come ImageNet o CIFAR.
Il modo migliore per trovare questi modelli pre-formati è cercare su google il modello e la struttura specifici. Tuttavia, per facilitare il processo di ricerca, ho messo insieme una lista con i comuni modelli Covnet pre-formati su framework popolari.

Fine Tuning

Dove troviamo reti pre-addestrate?

- **Caffe : Model Zoo** - Una piattaforma per i collaboratori di terze parti per condividere modelli di caffè pre addestrati.
- **Keras : Keras Application** - Implementazione di modelli Convnet all'avanguardia come VGG16/19, googleNetNet, Inception V3 e ResNet.
- **TensorFlow : VGG16 ResNet**
- **Torch : LoadCaffe** - Mantiene una lista di modelli popolari come AlexNet e VGGG .Weights portato da Caffe
- **MxNet: MxNet Model Gallery** - Mantiene i modelli preaddestrati Inception-BN (V2) e Inception V3.

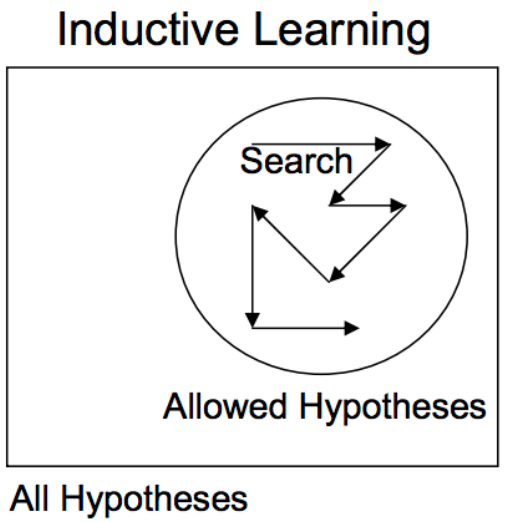
Transfer Learning

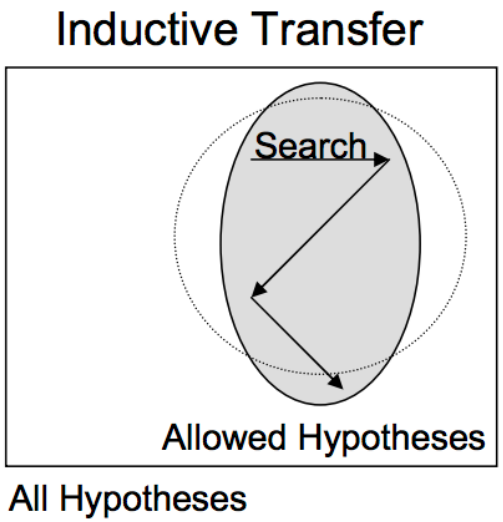
Il transfer learning è un metodo di apprendimento automatico in cui un modello sviluppato per un compito viene riutilizzato come punto di partenza per un modello su un secondo compito.

Si tratta di un approccio popolare nell'apprendimento profondo in cui modelli preformati sono utilizzati come punto di partenza per la visione artificiale e le attività di elaborazione del linguaggio naturale, date le vaste risorse di calcolo e di tempo necessarie per sviluppare modelli di rete neurale su questi problemi e dagli enormi salti di abilità che forniscono sui problemi correlati.

Transfer Learning

Il transfer learning è legato a problemi come l'apprendimento multitasking e la deriva concettuale e non è esclusivamente un'area di studio per il deep learning.
Tuttavia, il transfer learning è popolare nel deep learning, date le enormi risorse richieste per formare modelli di apprendimento profondo o i grandi e impegnativi set di dati sui quali vengono formati i modelli di apprendimento profondo.
Il trasferimento dell'apprendimento funziona nel deep learning do solo se le caratteristiche del modello apprese dal primo compito sono generali.





Transfer Learning

Come usarlo?

È possibile utilizzare il trasferimento di apprendimento sui propri problemi di modelli predittivi.

Sono presenti due approcci comuni sono i seguenti:

- Develop Model Approach
- Pre-trained Model Approach

Transfer Learning

Develop Model Approach

Selezionare Source Task. È necessario selezionare un problema di modellazione predittiva correlato con un'abbondanza di dati in cui esiste una qualche relazione tra i dati di input, i dati di output e/o i concetti appresi durante la mappatura dai dati di input a quelli di output.

Sviluppare il modello sorgente. Successivamente, è necessario sviluppare un modello abile per questa prima attività. Il modello deve essere migliore di un modello ingenuo per assicurare che sia stato eseguito l'apprendimento di alcune caratteristiche.

Transfer Learning

Develop Model Approach

Riutilizzare il modello. Il modello adatto al compito sorgente può quindi essere utilizzato come punto di partenza per un modello sul secondo compito di interesse. Ciò può comportare l'utilizzo di tutto o di parti del modello, a seconda della tecnica di modellazione utilizzata.
Sintonizzare il modello. Facoltativamente, il modello può aver bisogno di essere adattato o perfezionato sui dati della coppia input-output disponibili per il compito di interesse.

Transfer Learning

Pre-trained Model Approach

Selezionare il modello sorgente. Viene scelto un modello di sorgente preaddesignato tra i modelli disponibili. Molti istituti di ricerca rilasciano modelli su grandi e complessi insiemi di dati che possono essere inclusi nel pool di modelli candidati tra i quali scegliere.

Riutilizzare il modello. Il modello pre-trained può quindi essere utilizzato come punto di partenza per un modello sul secondo compito di interesse. Questo può comportare l'utilizzo di tutto o di parti del modello, a seconda della tecnica di modellazione utilizzata.

Transfer Learning

Pre-trained Model Approach

Sintonizzare il modello. Facoltativamente, il modello può aver bisogno di essere adattato o perfezionato sui dati della coppia input-output disponibili per il compito di interesse.

Questo secondo tipo di transfer learning è comune nel campo dell'apprendimento profondo.

Applicazioni

- Classificazione
- Image retrival
- Object detection and localizzation
- Segmentazione
- Denoising

Classificazione

Associazione di un pattern ad una classe.
Un pattern può essere un dato generico, un'immagine (o una parte del suo contenuto), una sequenza temporale, un testo, ecc.

Applicazioni
- Fault detection
- Face recognition
- Medical diagnosys
- Sentiment analysis
- Re Identification
- ...

→ **Sheep**

Classificazione di Immagini

Modelli noti
- VGG
- AlexNet
- ResNet
- InceptionNet
- DenseNet
- ...

Dataset pubblici
- ImageNet
- MNIST
- Fashion-MNIST
- Cifar-10
- Cifar-100
- ...

I più grandi modelli neurali per la classificazione di immagini sono stati valutati su **ImageNet** (ImageNet Large Scale Visual Recognition Challenge)

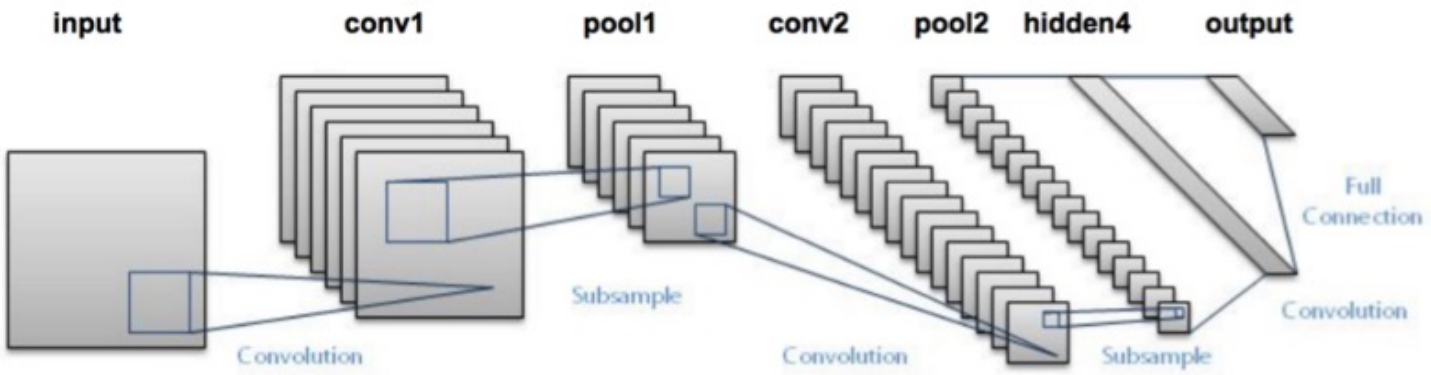
Classificazione di Immagini

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

ImageNet, è un set di oltre 15 milioni di immagini etichettate ad alta risoluzione con circa 22.000 categorie. ILSVRC utilizza un sottoinsieme di ImageNet di circa 1000 immagini in ciascuna delle 1000 categorie. In tutto, ci sono
circa 1,2 milioni di immagini di addestramento, 50.000 immagini di convalida e 150.000 immagini di test.

Classificazione di immagini

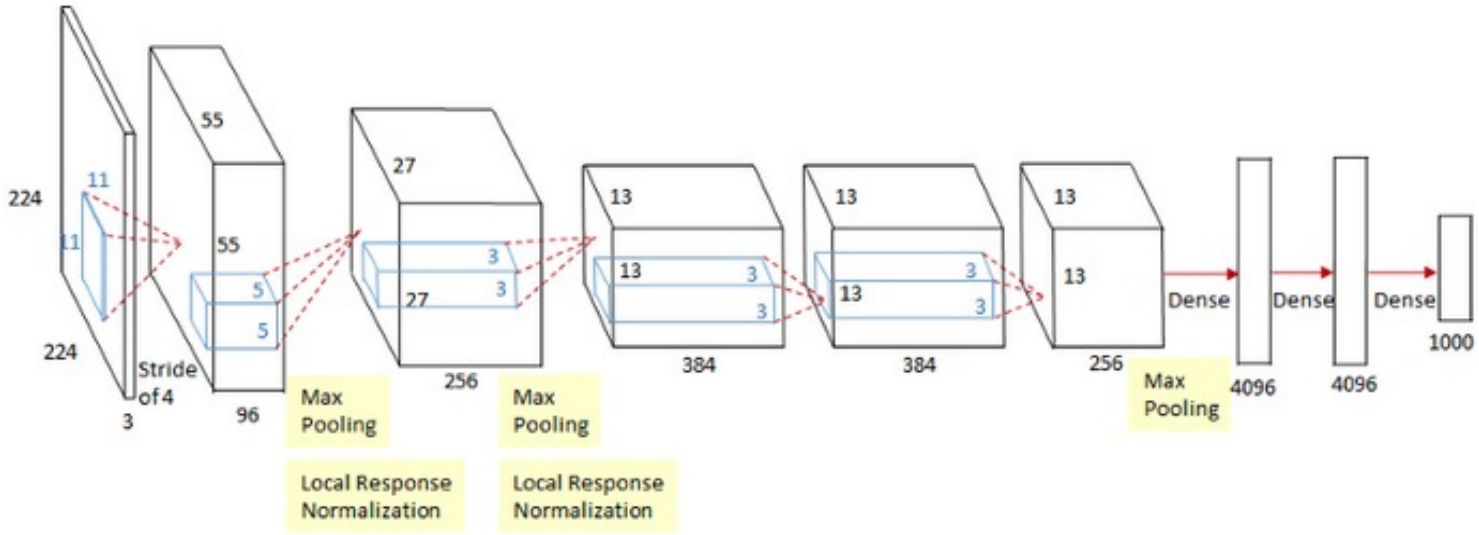
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



LeNet(1998)
LeNet-5, una pioniera rete convoluzionale profonda a 7 livelli di LeCun et al.
Nata per classificare le cifre, è stata applicata per riconoscere i numeri scritti a man
o su carta, digitalizzati in inputmap a scala di grigi 32x32 pixel.

Classificazione di Immagini

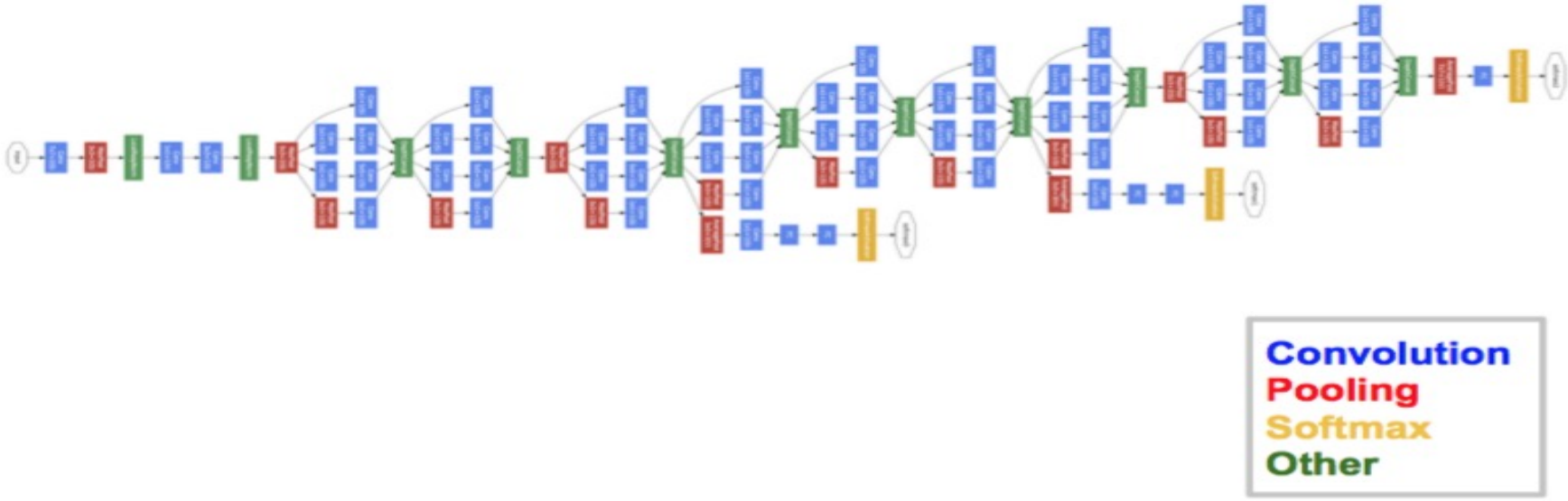
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



AlexNet/CaffeNet (2012)
Molto simile a LeNet di Yann LeCun ed altri, ma era più profonda, con più filtri per s trato e con strati convoluzionali sovrapposti

Classificazione di Immagini

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



GoogleNet / Inception (2014). Ispirata a LeNet ma implementa un modulo soprannominato inception. Questo modulo si basa su numerose convoluzioni molto piccole al fine di ridurre drasticamente il numero di parametri. L'architettura consiste in una CNN profonda 22 strati, ma ha ridotto il numero di parametri da 60 milioni (AlexNet) a 4 milioni.

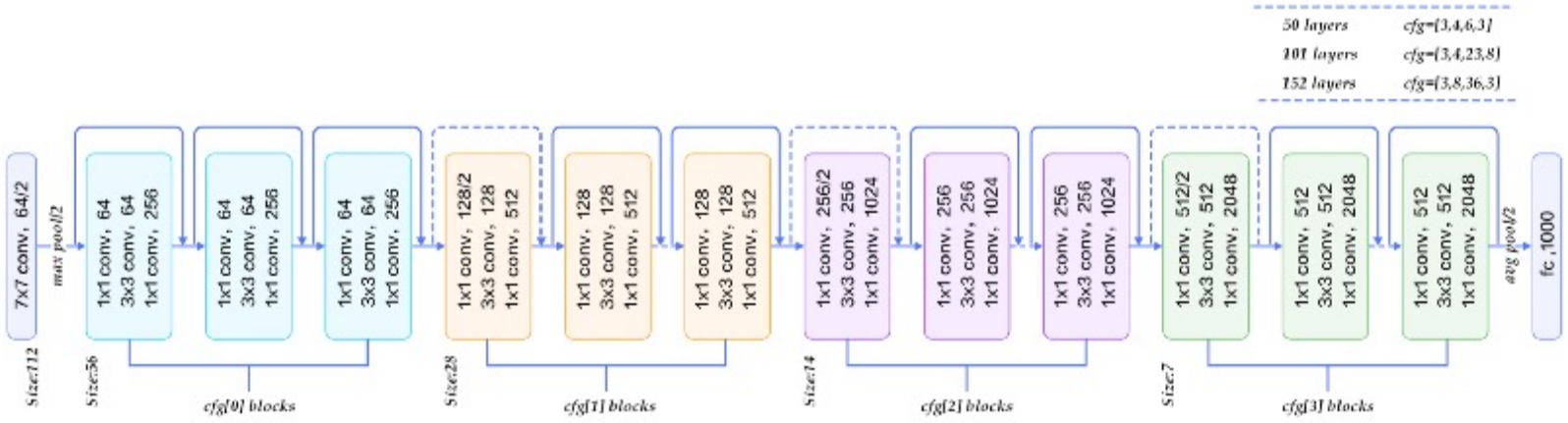
Classificazione di Immagini

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

VGG16(2014)
Simile a AlexNet, contiene solo convoluzioni 3x3, ma molti filtri. VGGNet è composto da 16 strati convoluzionali e comprende 138 milioni di parametri.

Classificazione di Immagini

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



ResNet (2015)
Rete neurale residua (ResNet) di Kaiming He et al. Ha introdotto una innovativa architettura con "**skip connections**". Con questa tecnica è stato possibile superare il problema del **vanishing gradient** e addestrare DCNN con 152 livelli, pur avendo una complessità inferiore a GoogleNet.

Classificazione di Immagini

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Metriche di Classificazione

Matrice di confusione

		Prediction	
		$\widehat{P}$	$\widehat{N}$
Ground Truth	P	TP	FN
	N	FP	TN

TP: true positive
FN: false negative
FP: false positive
TN: true negative

$$Accuracy = \frac{TP}{TP + FP + FN + TN}$$

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\widehat{P}}$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{P}$$

$$F_1 - Score = 2\frac{PrecisionRecall}{Precision + Recall}$$

- L'accuratezza è un indice globale
- Precision, Recall, F 1 -Score possono essere definite per ciascuna classe (più informative dell'Accuracy)

Image retrieval

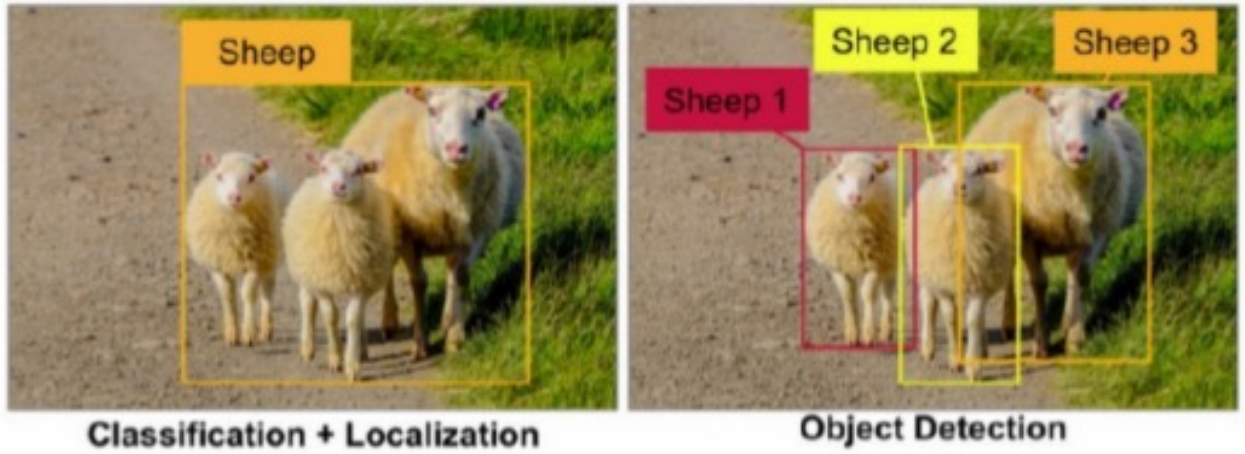
Recupero di immagini simili basato sul contenuto semantico.

Calcolo della similarità (tra immagine di query e immagini del dataset)

- distanza tra le features
- selezione delle features
  - layer superficiali → texture matching
  - layer profondi → object matching

Object Detection

- Ricerca di un oggetto nell'immagine



Formato da due compiti distinti

- riconoscimento dell'oggetto (classificazione)
- localizzazione dell'oggetto nell'immagine (trovare la sua posizione, bounding box)

localizzazione vs detection

- localizzazione: rileva il singolo oggetto.
- detection: rileva tutti gli oggetti appartenenti ad una classe.

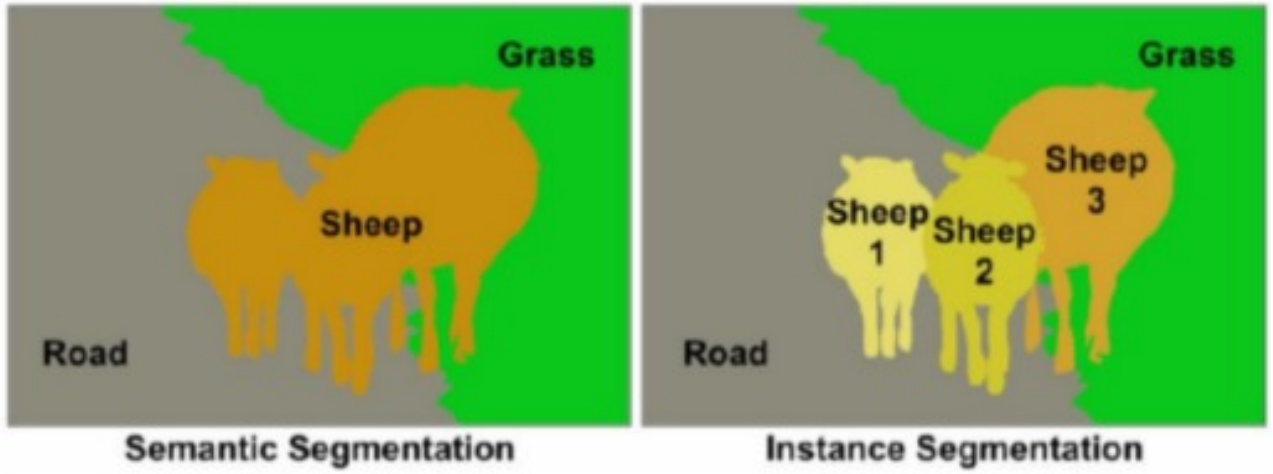
Object Detection

Modelli noti
- SPP-net
- RPN
- Fast R-CNN
- ...

Dataset pubblici
- ImageNet
- Pascal VOC
- COCO
- ...

Segmentazione

Partizione di un'immagine in regioni significative



Tipi di segmentazione
semantica: qualsiasi oggetto della classe viene associato alla classe stessa (classificazione pixel per pixel)
distanza: ciascun oggetto della classe viene identificato come singolo (object detection pixel per pixel)

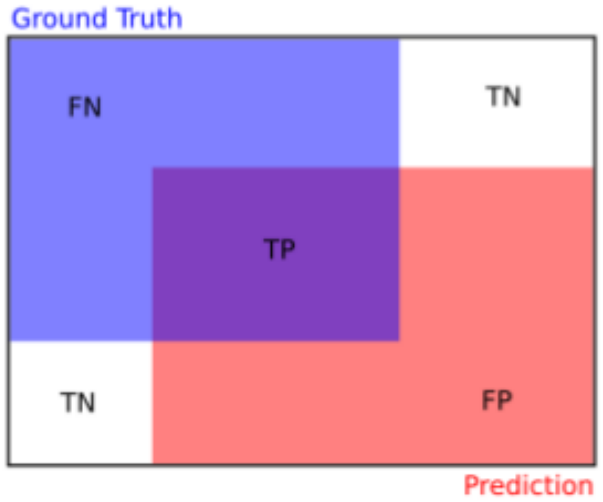
Segmentazione

Applicazioni
- diagnosi di immagini mediche
- misure di immagini satellitari
- visione dei robot

Modelli noti
- Fully Convolutionall Network (FCN)
- SegNet
- UNet
- ...

Dataset pubblici
- PASCAL
- COCO
- ...

Metriche di Object Detection e Segmentazione



$$|GT \cap Pred| = TP$$

$$|GT \cup Pred| = TP + FP + FN$$

$|\cdot|$: Operatore Cardinalità

Intersection over Union

$$IoU(Jaccard) = \frac{|GT \cap Pred|}{|GT \cup Pred|}$$

$$Dice = 2\frac{|GT \cap Pred|}{|GT| + |Pred|}$$

$$PixelAccuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Average Precision

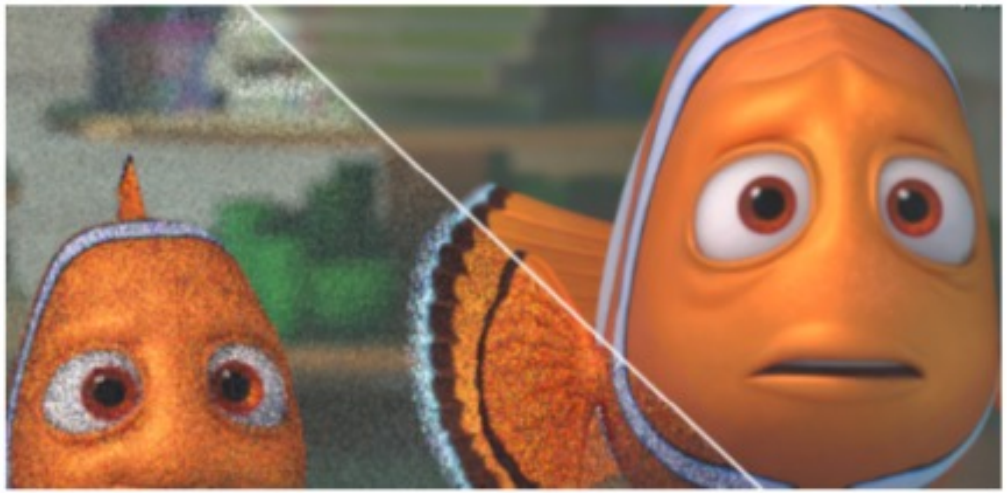
$$AP = \frac{1}{|tresholds|} \sum_{t} \frac{TP(t)}{TP(t) + FP(t) + FN(t)}$$

mean Average Precision

$$mAP = \frac{1}{|classes|} \sum_{c} AP(c)$$

Denoising

Rimozione del rumore da un immagine. (Rumore inteso come qualcosa di indesiderato)



Modelli:

- Denoising Autoencoder

Denoising Autoencoder

AutoEncoder

- Rete neurale avente gli strati di input e di output delle medesime dimensioni
- Introdotto nel contesto dell'apprendimento automatico non supervisionato, per apprend ere una rappresentazione compatta di un set di dati
- Costituito da due parti (tipicamente simmetriche):

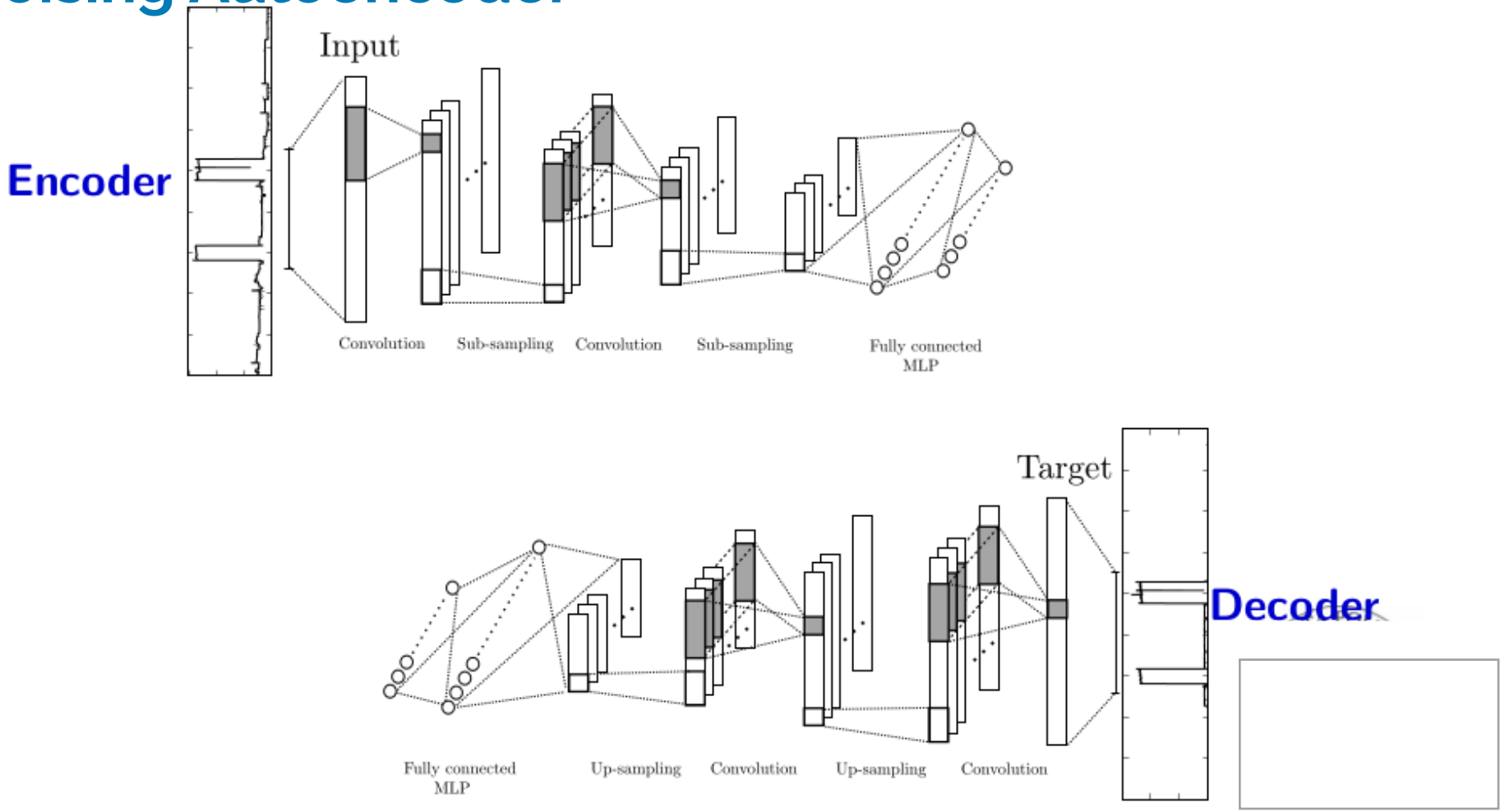
  Encoder (o codificatore) che analizza l'ingresso, per compattare l'informazione in uno strato di features di dimensione opportuna

  Decoder (o decodificatore), che ricostruisce, in uscita, l'ingresso a partire dallo strato di features

denoising AutoEncoder

- Stessa struttura dell'AE
- Allenati con tecniche supervisionate, per apprendere:
  - la migliore rappresentazione robusta ai rumori
  - Generare in uscita la versione priva di rumore del segnale di ingresso)

Denoising Autoencoder



UNIVERSITÀ DI MACERATA
l'umanesimo che innova
DIPARTIMENTO DI
SCIENZE POLITICHE,
DELLA COMUNICAZIONE,
E DELLE RELAZIONI INTERNAZIONALI

Deep Learning

Tricks of the Trade

- Introduzione
- Perché deep?
- Livelli e complessità
- Tipologie di DNN
- Ingredienti necessari
- Da MLP a CNN

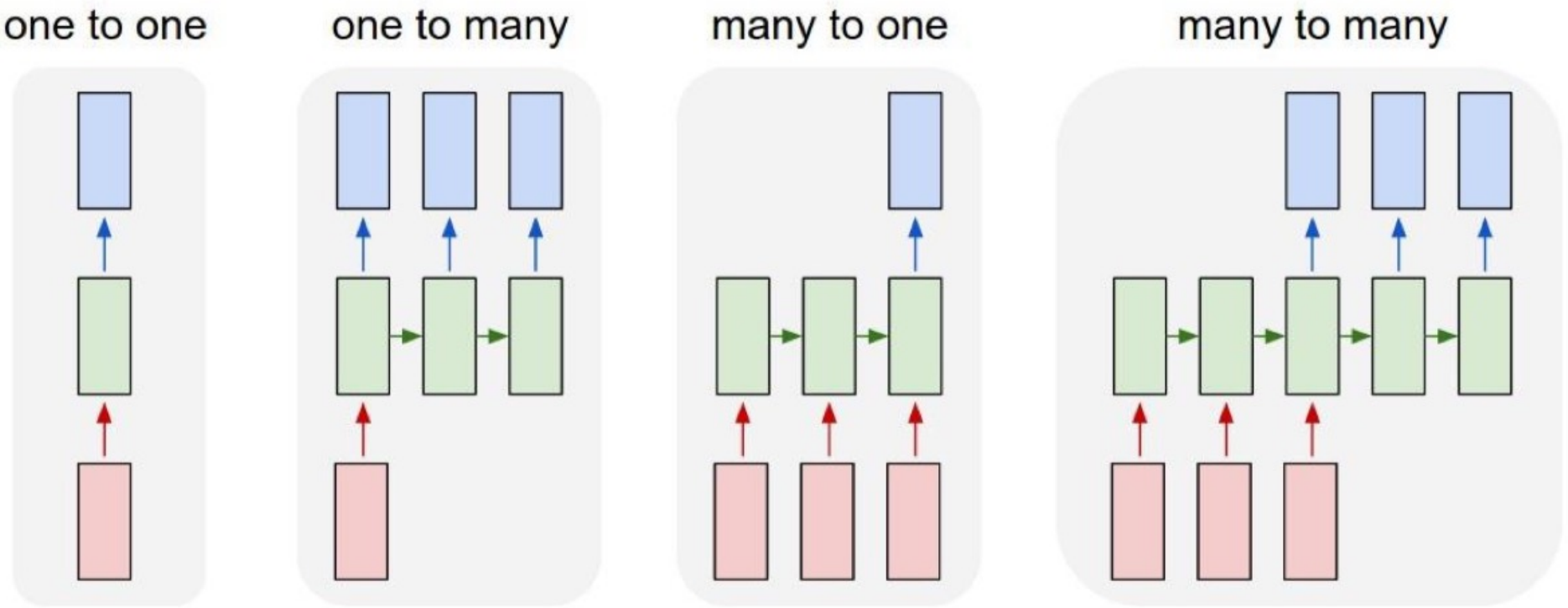
- Convolutional Neural Networks (CNN)
- Architettura
- Volumi e Convoluzione 3D
- Relu, Pooling
- Esempi di reti
- Training e Transfer Learning

Tricks of the Trade

- Recurrent Neural Networks (RNN)
- Sequence to Sequence
- Unfolding in Time
- Basic Cells, LSTM, GRU
- Natural Language Processing

- Reinforcement Learning
- Q-Learning
- Deep Q-Learning

Sequence to Sequence



Sequence to Sequence

Le reti feedforward (es. MLP, CNN) studiate fino a questo momento operano su vettori di dimensione prefissata. Ad esempio l'input è un vettore immagine è l'output un vettore di probabilità delle classi. Questo caso è rappresentato dalla prima colonna della figura come sequenza One to one.
Applicazioni diverse richiedono che l'input e/o l'output possano essere sequenze (anche di lunghezza variabile).

- One to many. Es. image captioning dove l'input un'immagine e l'output una frase (sequenza di parole) in linguaggio naturale che la descrive.
- Many to one. Es. sentiment analysis dove l'input è un testo (es. recensione di un prodotto) e l'output un valore continuo che esprime il sentiment o giudizio (positivo o negativo).
- Many to many. Es. language translation dove l'input è una frase in Inglese e l'output la sua traduzione in Italiano.

Recurrent neural network (RNN)

Limiti di un MLP

Non possono modellare informazioni sequenziali

Vantaggi di una RNN

Retroazione: l'uscita di un neurone viene riportata all'ingresso dello stesso neurone

- hanno uno stato interno
- simulano la memoria a breve termine

Modellano più fedelmente il cervello umano

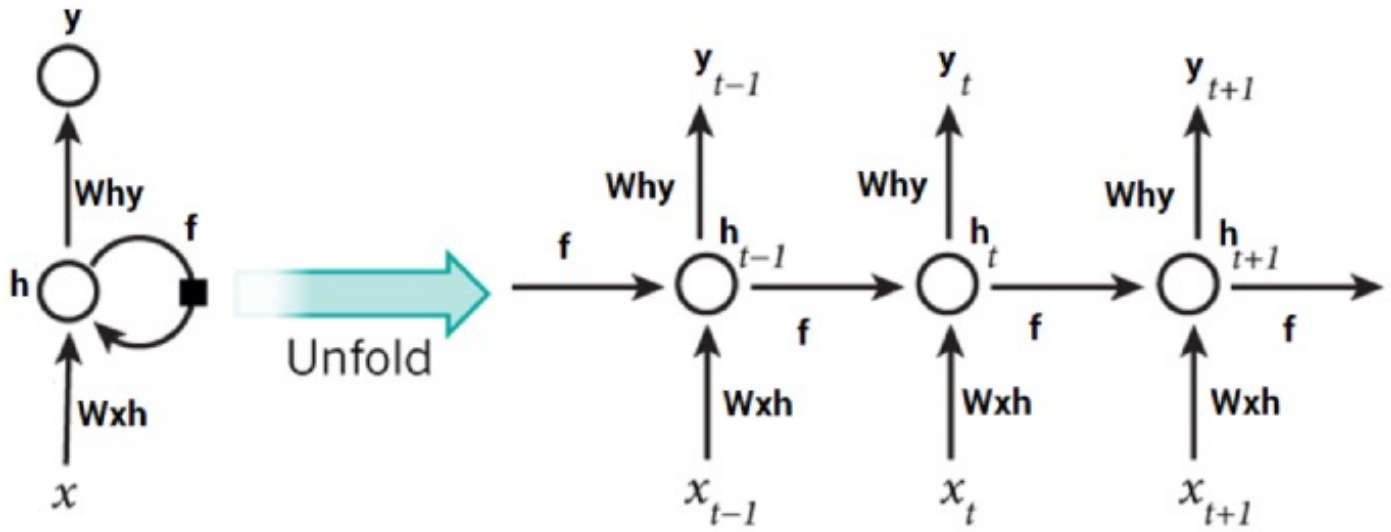
Recurrent neural network (RNN)

Tipo peculiare di rete neurale artificiale che, oltre alle vie di connessione ascendenti, possegga anche quelle discendenti o ricorrenti, ossia, che connetta le unità di output con quelle intermedie e con quelle di input. Queste connessioni addizionali producono una rete la cui risposta vettoriale (nello strato intermedio) all'input dello strato sensoriale è in parte una funzione del concomitante stato di attivazione (o cognitivo) del terzo strato il quale, a sua volta, è la risultante di input ed elaborazione precedenti. Una risposta ricorrente della rete a un determinato stimolo, quindi, non viene fissata soltanto dai caratteri strutturali della rete stessa, come si verifica nella rete *feedforward*. La risposta varia come funzione del precedente contesto dinamico o cognitivo in cui si è manifestato lo stimolo. Per un sistema ricorrente di questo tipo, la risposta dello strato intermedio si manifesta come una sequenza di vettori di attivazione.

Recurrent neural network (RNN)

Ogni rete ricorrente rappresenta una forma di memoria a breve termine: le informazioni raccolte nel vettore di attivazione dello strato intermedio vengono elaborate nello strato successivo e, quindi, rinviate alla propria origine, forse in forma modificata. Tali informazioni possono muoversi più volte lungo questo anello ricorrente, attenuandosi gradualmente. Se alcune informazioni sono importanti, la rete può anche avere una configurazione capace di conservarle senza attenuazione attraverso molti cicli, fino a che il contesto cognitivo non la ritenga più utile. Questo sistema fornisce automaticamente una forma di memoria a breve termine, sensibile al contenuto, con un tempo variabile di decadimento. Una rete ricorrente è altresì in grado di impegnarsi in un'attività cognitiva anche in assenza di una stimolazione in arrivo allo strato di input, perché gli impulsi che viaggiano sulle vie ricorrenti possono essere sufficienti a mantenere il sistema in attività. Essa può modulare inoltre, per mezzo di quelle stesse vie, la modalità di reazione agli stimoli dello strato sensoriale, e la salienza di taluni aspetti di quell'input. (*) → Neurofilosofia

Recurrent neural network (RNN)



Una rete ricorrente, in istanti successivi, può essere ricondotta ad una rete feed-forward. Questo permette di poter utilizzare l'algoritmo backpropagation per il training di reti ricorrenti.

- possibili problemi di instabilità
- difficoltà nel training (richiede più tempo)
- soffrono del problema del vanishing gradient

Long Short Time Memory (LSTM)

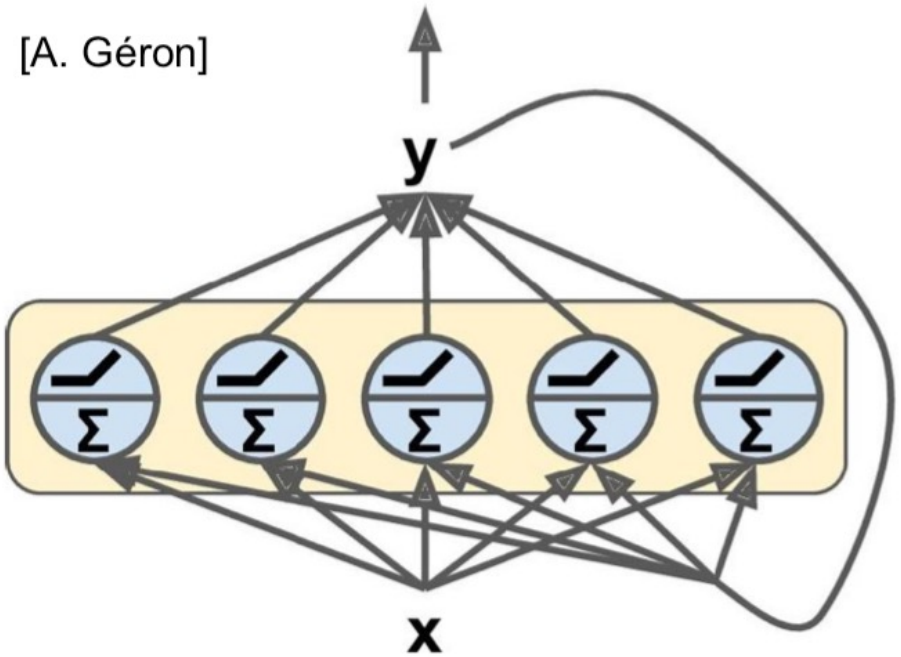
Limiti delle RNN

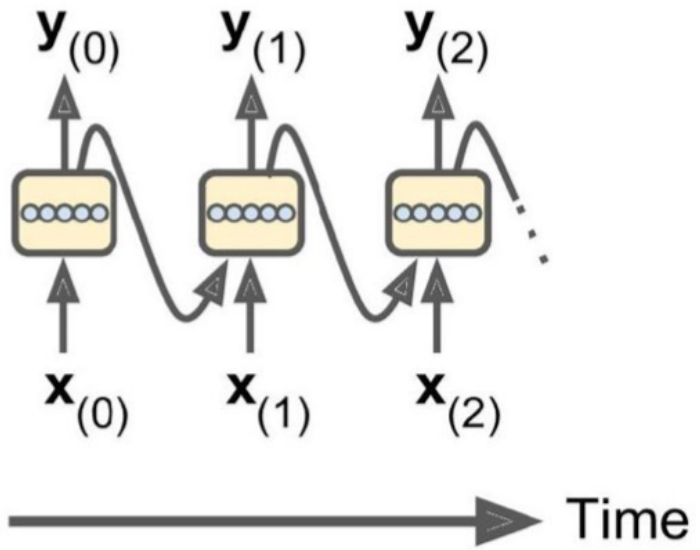
- soffrono del problema del vanishing gradient
- Sensibilità della rete decade nel tempo, come un nuovo ingresso sovrascrive l'attivazione del layer nascosto e la rete dimentica (forgot) il primo ingresso

Vantaggi di una RNN

- blocchi LSTM al posto dei neuroni del layer nascosto
- LSTM può sceglier se ricordare per un arbitrario periodo di tempo o dimenticare se necessario (dualità forget/retain)

Reti Ricorrenti (RNN)





Reti Ricorrenti (RNN)

Le reti ricorrenti prevedono «anche» collegamenti all'indietro o verso lo stesso livello. I modelli più comuni e diffusi (es. LSTM, GRA) prevedono collegamenti verso lo stesso livello.

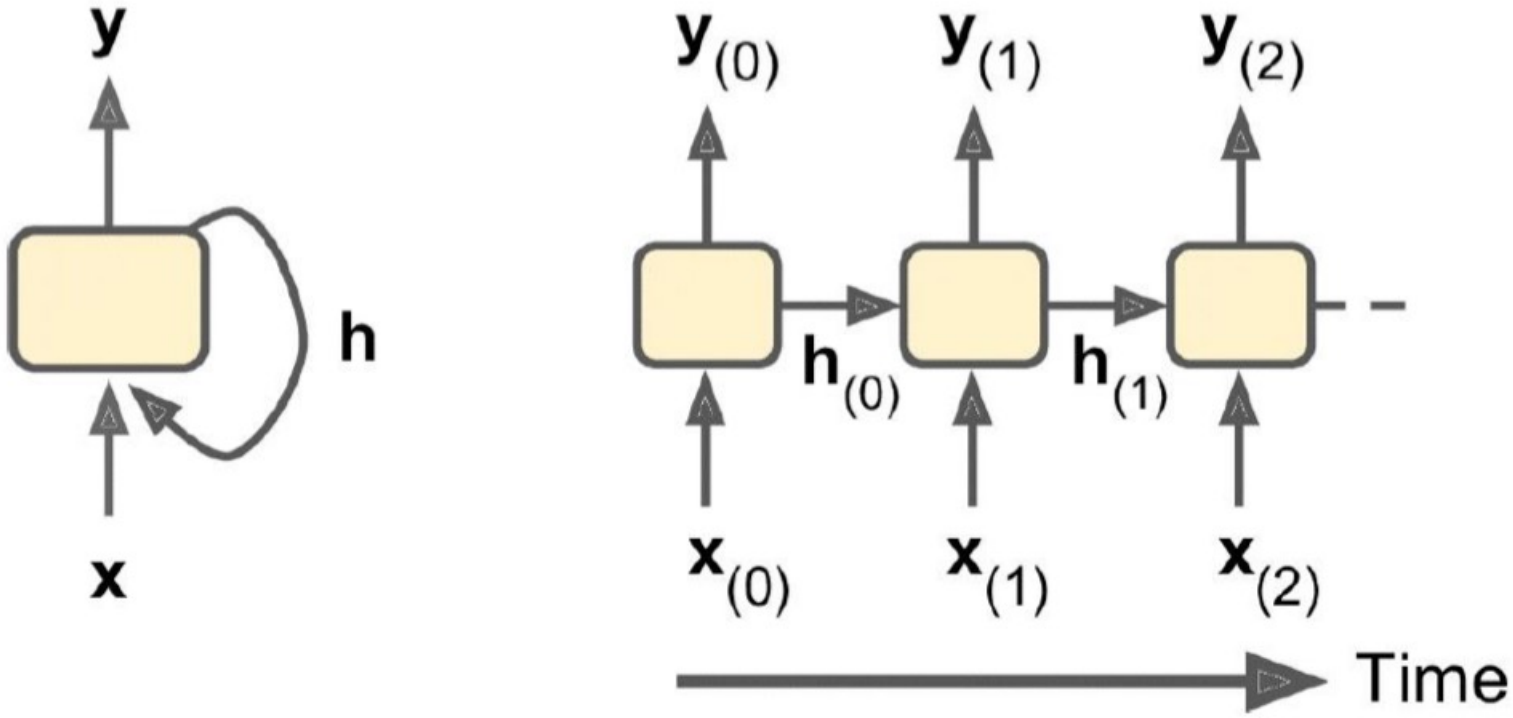
A ogni step della sequenza (ad esempio a ogni istante temporale t) il livello riceve oltre all'input $\mathbf{x}_{(t)}$ anche il suo output dello step precedente $\boldsymbol{y}_{(t-1)}$. Questo consente alla rete di basare le sue decisioni sulla storia passata (effetto memoria) ovvero su tutti gli elementi di una sequenza e sulla loro posizione reciproca.

Reti Ricorrenti (RNN)

- In una frase non è rilevante solo la presenza di specifiche parole ma è importante anche come le parole sono tra loro legate (posizione reciproca).
- La classificazione di video (sequenze di immagini) non necessariamente si basa sulle interrelazioni dei singoli frame. Esempio:
  - *per capire se un video è un documentario su animali è sufficiente la detection di animali in qualche frame (non occorre RNN).*
  - *per comprendere il linguaggio dei segni, è necessario analizzare le interrelazioni dei movimenti delle mani nei frame (utile RNN).*

Unfolding in Time

[A. Géron]



Unfolding in Time

Una cella è una parte di rete ricorrente che preserva uno stato (o memoria) interno $\mathbf{h}_{(t)}$ per ogni istante temporale. È costituita da un numero prefissato di neuroni (può essere vista come un layer).

- $\mathbf{h}_{(t)}$ dipende dall'input $\mathbf{x}_{(t)}$ e dallo stato precedente $\mathbf{h}_{(t-1)}$

$$\mathbf{h}_{(t)} = f\left(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)}\right)$$

Per poter addestrare (con backpropagation) una RNN è necessario eseguire il cosiddetto unfolding o unrolling in time (parte destra della figura), stabilendo a priori il numero di passi temporali su cui effettuare l'analisi.

Unfolding in Time

- Di fatto una RNN unfolded su 20 step equivale a una DNN feedforward con 20 livelli. Pertanto addestrare RNN che appaiono relativamente semplici può essere molto costoso e critico per la convergenza (problema del vanishing gradient).

- Da notare che in una RNN unfolded: l'input e l'output sono in generale collegati a tutte le istanze della cella e i pesi della cella (nascosti in f) sono comuni a tutte le istanze della cella.

Basic Cell

In una cella base di RNN:

Lo stato $\mathbf{h}_{(t)}$ dipende dall'input $\mathbf{x}_{(t)}$ e dallo stato precedente $\mathbf{h}_{(t-1)}$

$$\mathbf{h}_{(t)} = \phi\left(\mathbf{x}_{(t)}{}^{T} \cdot \mathbf{W}_{x} + \mathbf{h}_{(t-1)}{}^{T} \cdot \mathbf{W}_{h} + b\right)$$

dove:

- $\mathbf{W}_{x}$ e $\mathbf{W}_{y}$ sono i vettori dei pesi e b il bias da apprendere.
- ϕ è la funzione di attivazione (es. Relu).

e l'output $\mathbf{y}_{(t)}$ corrisponde allo stato $\mathbf{h}_{(t)}$:

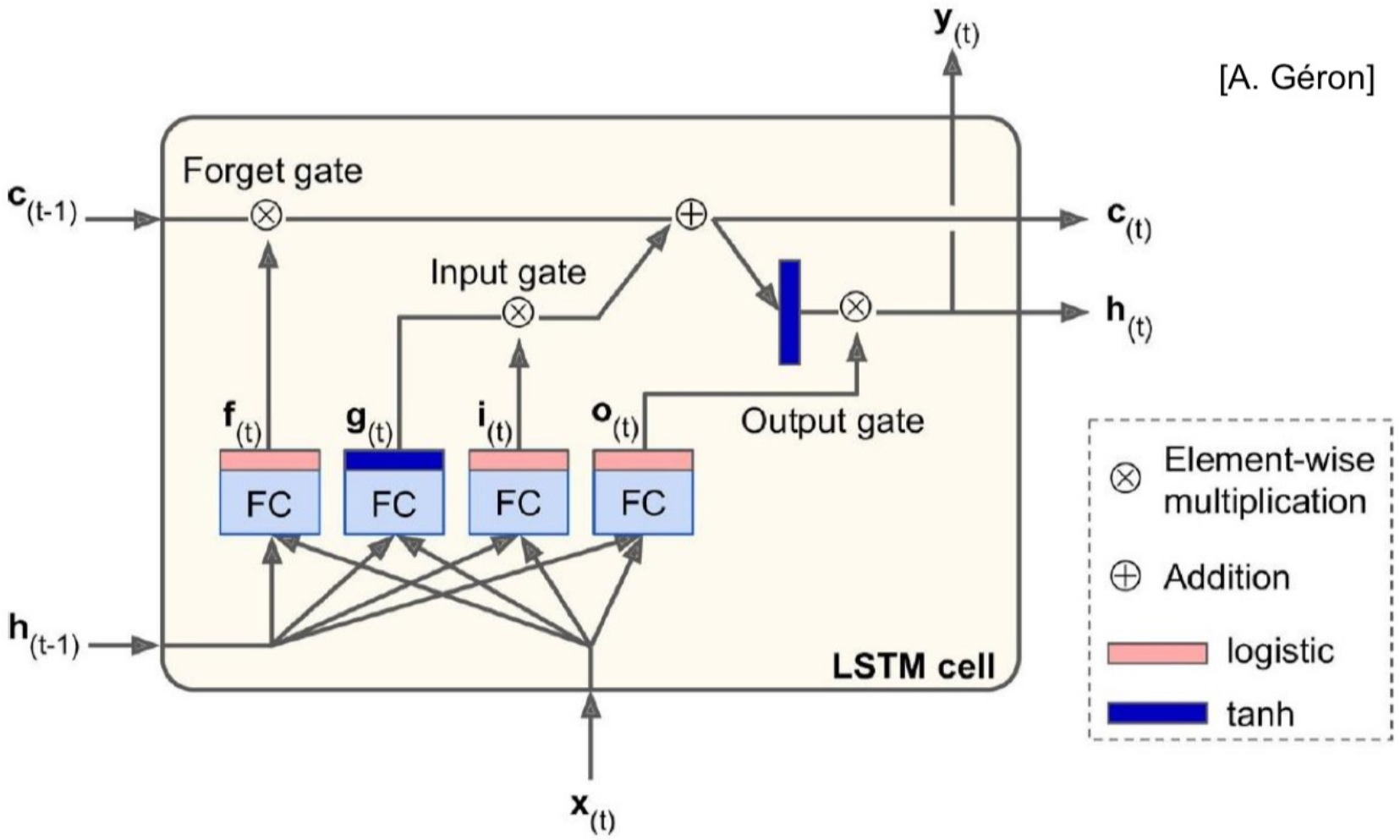
$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)}$$

Basic Cell

Le celle base hanno difficoltà a ricordare/sfruttare input di step lontani: la memoria dei primi input tende a svanire. D'altro canto sappiamo che in una frase anche le prime parole possono avere un'importanza molto rilevante.

Per risolvere questo problema e facilitare la convergenza in applicazioni complesse, sono state proposte celle più evolute dotate di un effetto memoria a lungo termine: LSTM e GRU sono le più note tipologie.

LSTM


[A. Géron]
$y_{(t)}$
$c_{(t-1)}$
$c_{(t)}$
$h_{(t)}$
$h_{(t-1)}$
$x_{(t)}$
Forget gate
Input gate
Output gate
$f_{(t)}$
$g_{(t)}$
$i_{(t)}$
$o_{(t)}$
FC
FC
FC
FC
LSTM cell
Element-wise multiplication
Addition
logistic
tanh

LSTM

In LSTM (Long Short-Term Memory) lo stato $\mathbf{h}_{(t)}$ è suddiviso in due vettori $\mathbf{h}_{(t)}$ e $\mathbf{c}_{(t)}$:

- $\mathbf{h}_{(t)}$ è uno stato (o memoria) a breve termine; anche in questo caso uguale all'output della cella $\mathbf{y}_{(t)}$.
- $\mathbf{c}_{(t)}$ è uno stato (o memoria) a lungo termine.
- la cella apprende durante il training cosa è importante dimenticare (forget gate) dello stato passato $\mathbf{c}_{(t-1)}$ e cosa estrarre e aggiungere (input gate) dall'input corrente $\mathbf{x}_{(t)}$.
- per il calcolo dell'output $\mathbf{y}_{(t)} = \mathbf{h}_{(t)}$ si combina l'input corrente (output gate) con informazioni estratte dalla memoria a lungo termine.

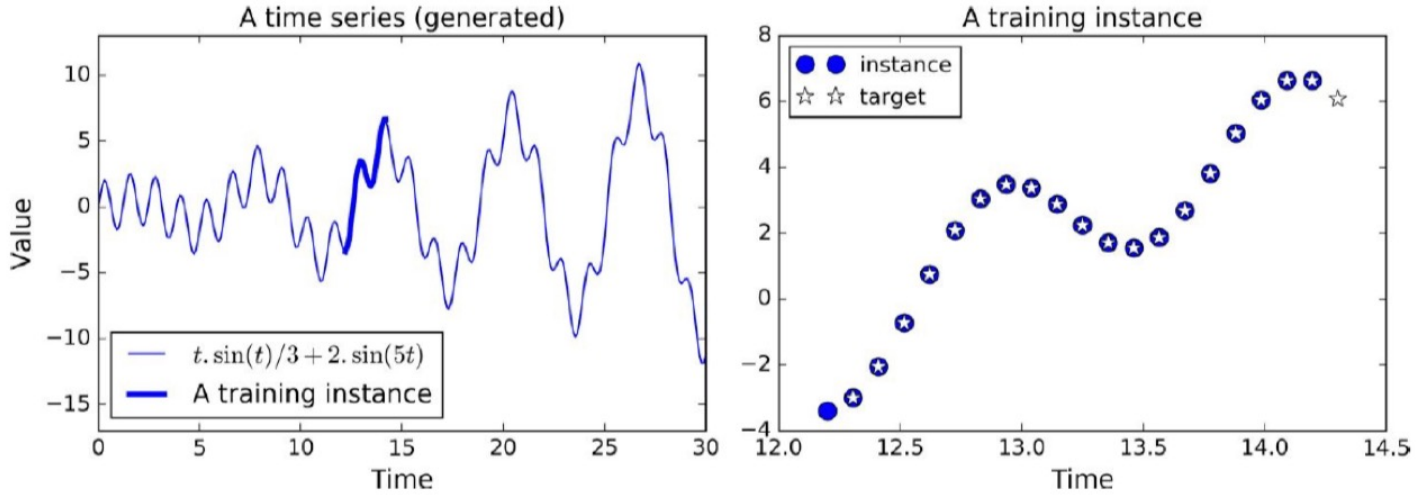
Esempio: predizione e serie temporali

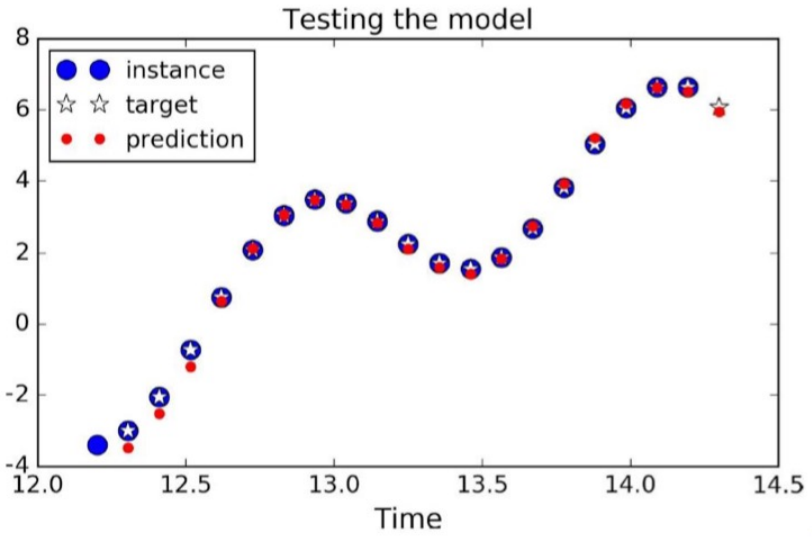
Una serie temporale può essere relativa all'andamento nel tempo: di un titolo di borsa, della temperatura di un ambiente, del consumo energetico di un impianto, ecc.

Possiamo considerare una serie temporale come una funzione campionata in più istanti temporali. Conoscendo i valori fino all'istante t siamo interessati a predirre il valore a t+1. Nota bene: non solo input 1D.

In [A. Géron] semplice esempio in Tensorflow di RNN applicate alla predizione di una funzione matematica (combinazione di sinusoidi).

Esempio: predizione e serie temporali





Natural Language Processing (NLP)

L'elaborazione del linguaggio naturale è uno dei settori cui il deep learning ha dato i maggiori contributi. Tra le principali applicazioni:

- Language Modeling: es. generare testi
- Text Classification: es. sentiment analysis
- Speech Recognition: es. traduzione parlato in testo
- Caption Generation: es. data un'immagine generare una descrizione in linguaggio naturale.
- Machine Translation: es. traduzione tra lingue, creare riassunti di documenti, generare commenti nel codice.
- Question Answering: es. rispondere in linguaggio naturale a domande poste in linguaggio naturale.

Word Embedding

Rappresentazione one-hot-vector:

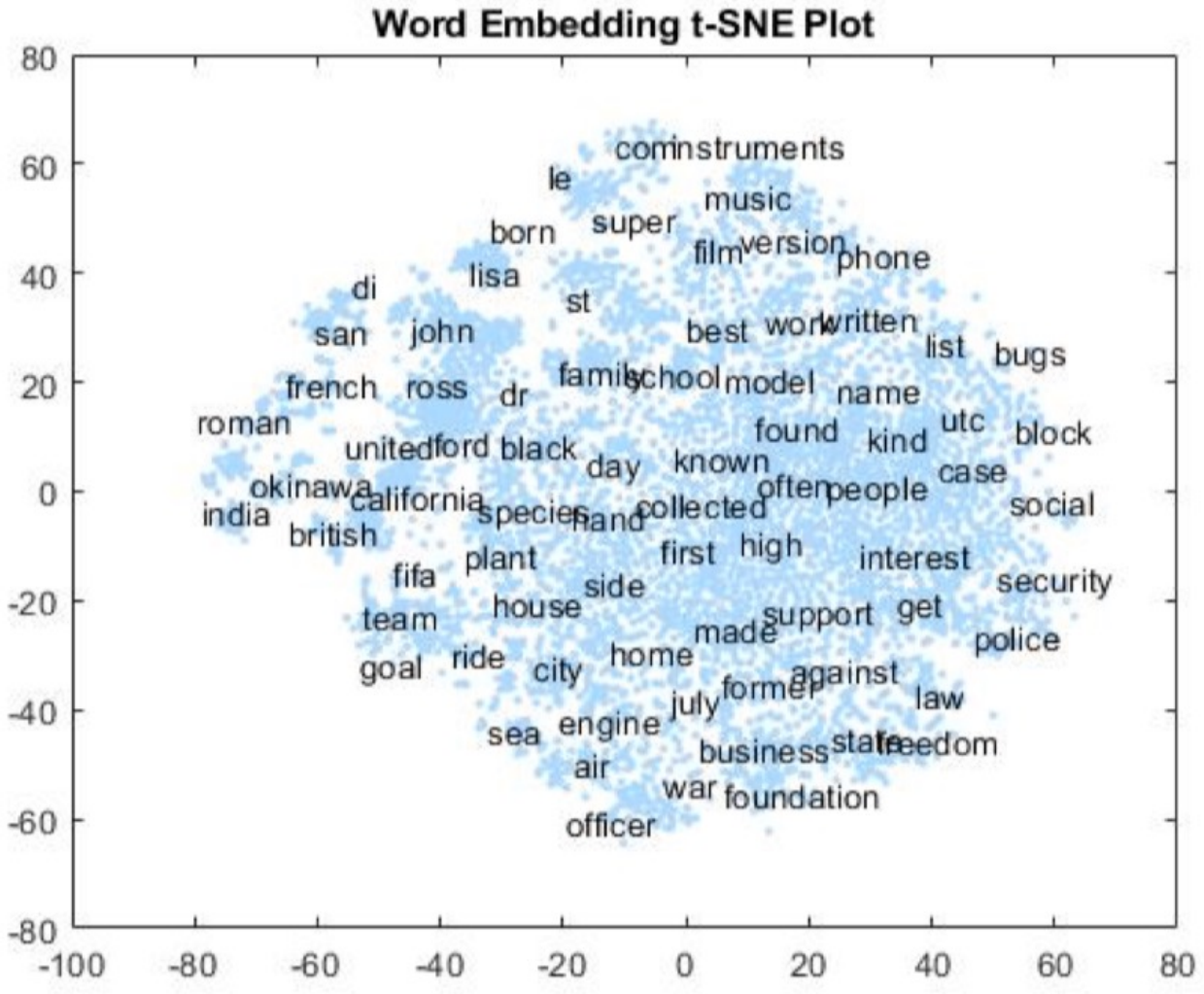
- Dato un dizionario di dimensione *m* parole (es. 50000), la parola corrispondente alla *k*-esima entry è codificata con un vettore di dimensione *m*, dove tutti gli elementi assumono valore 0 tranne l'elemento in posizione *k* che assume valore 1.
- Questa rappresentazione non è però utile perché non codifica la semantica e la similarità tra parole (es. sinonimi).

Word Embedding

La tecnica word embedding associa a ogni parola del dizionario un vettore di dimensionalità contenuta (es. 150). Parole di significato simile sono associate a vettori vicini nello spazio.

- Word2Vec è un modello di rete neurale (a due livelli) che può essere addestrato (unsupervised) per produrre l'embedding a partire da un corpus di testi.

Word Embedding



Language Models

Un modello di linguaggio (character based) può essere ottenuto con una RNN in modo abbastanza semplice a partire da un corpus di testi (es. La Divina Commedia):

- Trattiamo un testo come un sequenza temporale di caratteri ed estraiamo sequenze di lunghezza prefissata (es. 21 caratteri).
- Addestriamo una RNN a predire il 21-simo carattere dati i primi 20.

Language Models

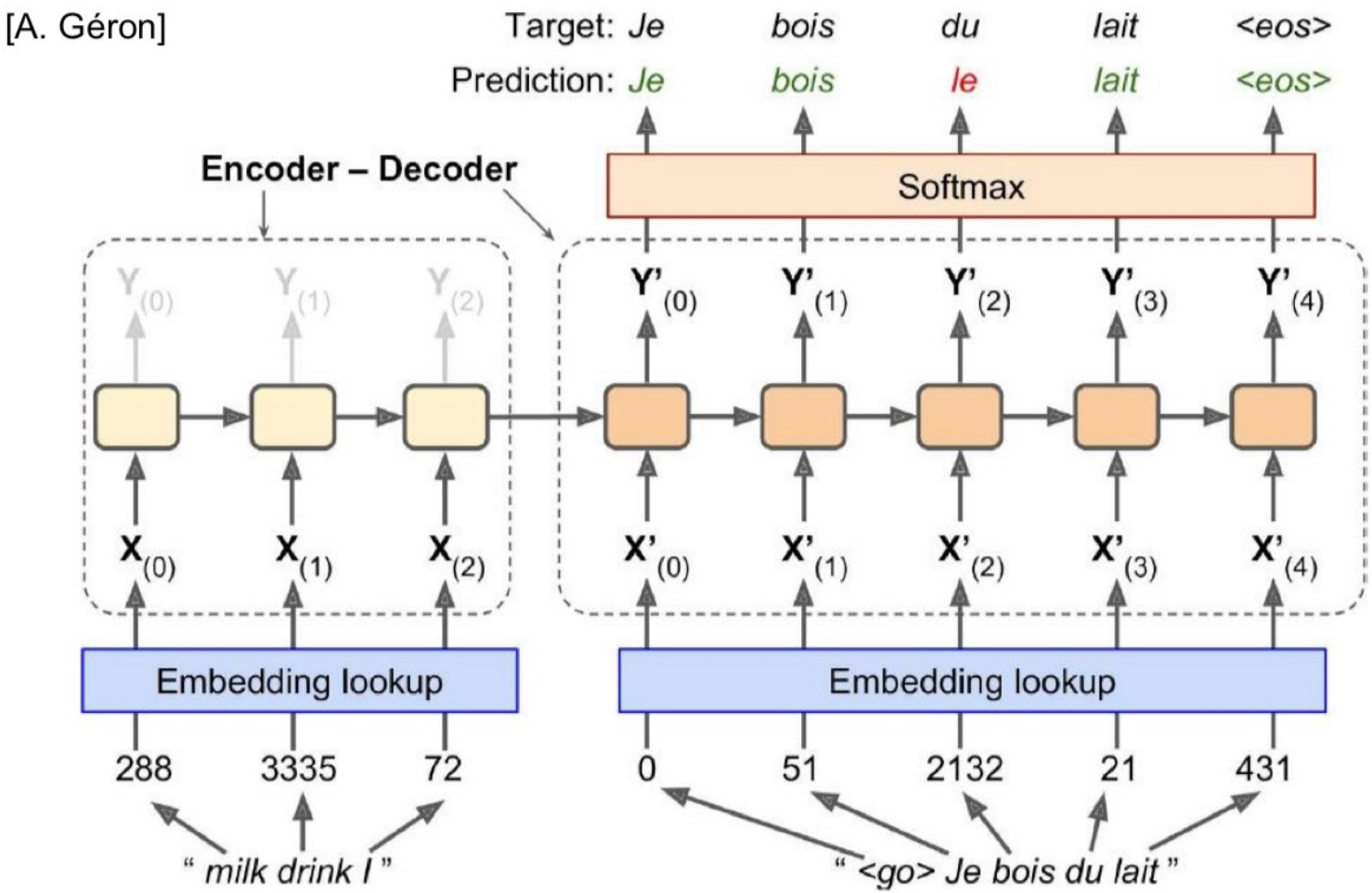
- Al termine dell'addestramento il modello può essere utilizzato in modo creativo per generare testi: si parte con uno o più caratteri random e si chiede alla rete di fare predizione; si appende l'ultimo carattere generato alla stringa corrente e si passano al modello gli ultimi 20 caratteri. Si itera in questo modo fino ad ottenere testo di lunghezza desiderata.

I risultati possono essere sorprendenti, vedi:

- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Modelli di linguaggio (word based) sono più potenti ma più complessi: richiedono parsing, stemming, word embedding.

Neural Machine Translation



Neural Machine Translation

Elaborazione di sequenze in modalità many-to-many (lunghezza variabile). La traduzione da una lingua a un'altra è l'applicazione più nota (e tra le più complesse).

- I modelli più utilizzati sono architetture RNN di tipo Encoder-Decoder con meccanismi di attenzione.
- Vedi Seq2Seq per TensorFlow.

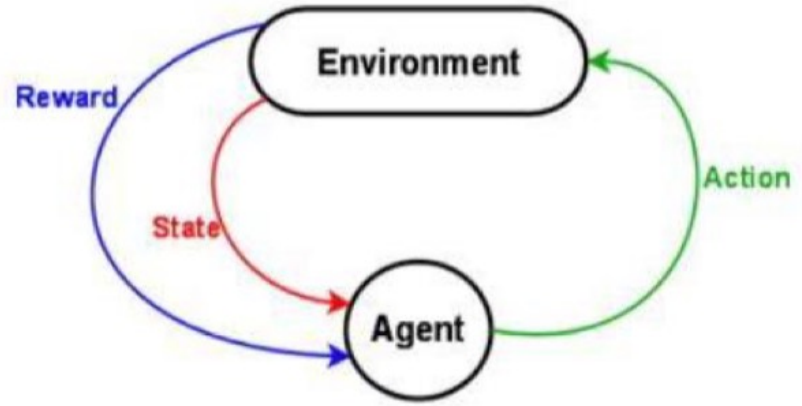
Per approfondimenti:

- Philipp Koehn, *Neural Machine Translation* https://arxiv.org/abs/1709.07809

Reinforcement Learning

L'obiettivo è apprendere un comportamento ottimale a partire dalle esperienze passate.

- Un agente esegue azioni (a) che modificano l'ambiente, provocando passaggi da uno stato (s) all'altro. Quando l'agente ottiene risultati positivi riceve una ricompensa o reward (r) che però può essere temporalmente ritardata rispetto all'azione, o alla sequenza di azioni, che l'hanno determinata.



Reinforcement Learning

- Un episodio (o game) è una sequenza finita di stati, azioni, reward:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2 \ldots, s_{n-1}, a_{n-1}, r_n, s_n$$

- In ciascun stato s_{t-1}, l'obiettivo è scegliere l'azione ottimale a_{t-1}, ovvero quella che massimizza il future reward R_t:

$$R_t = r_t + r_{t+1} + \cdots + r_n$$

- In molte applicazioni reali l'ambiente è stocastico (i.e., non è detto che la stessa azione determini sempre la stessa sequenza di stati e reward), pertanto applicando la logica del «*meglio un uovo oggi che una gallina domani*» si pesano maggiormente i reward temporalmente vicini (discounted future reward):

$$R_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots + \gamma^{n-t} \cdot r_n \quad \text{(con } 0 \leq \gamma \leq 1\text{)}$$