Introduzione al Deep Learning

Alessandro Aere

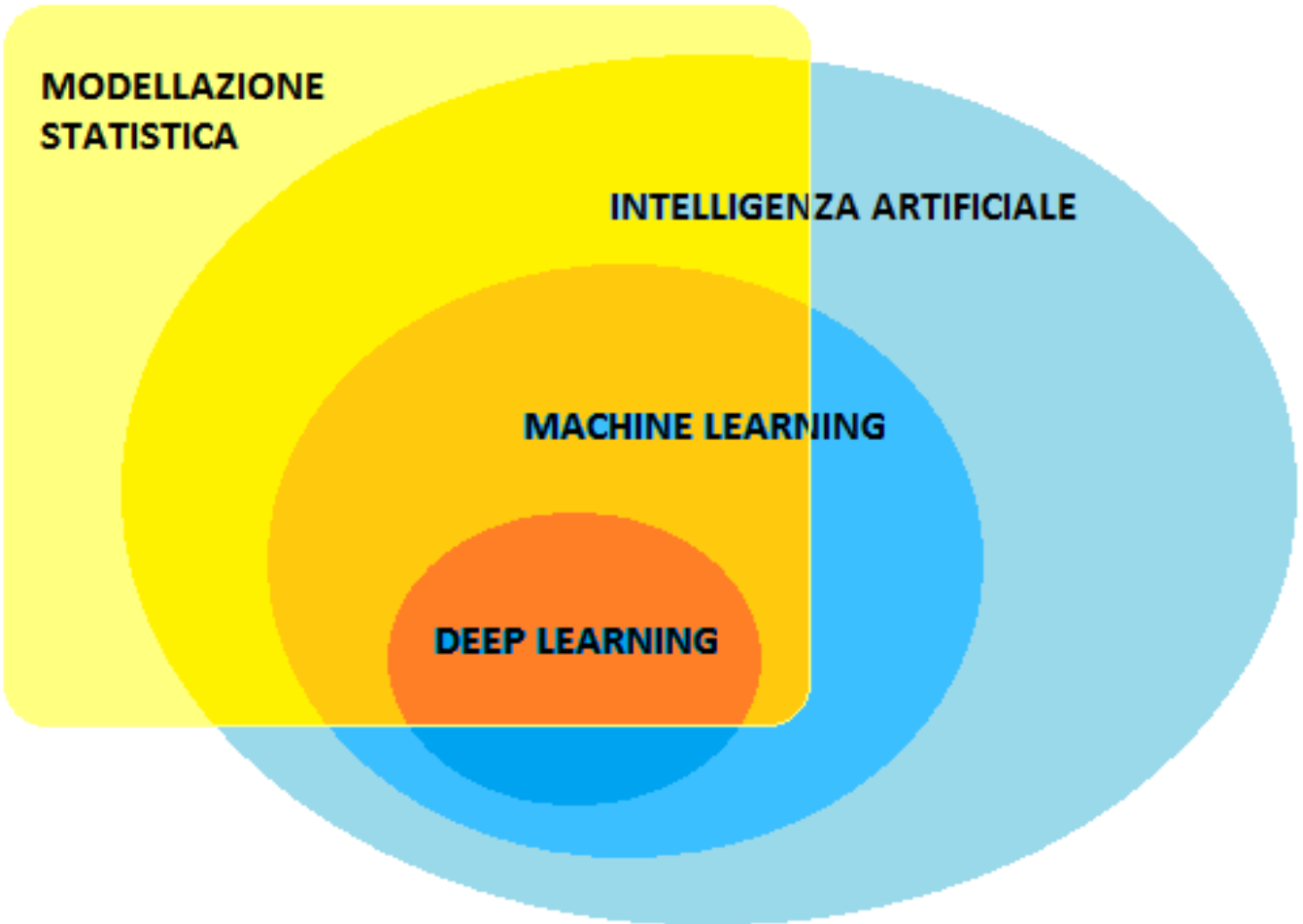
Università degli Studi di Padova

22 maggio 2020

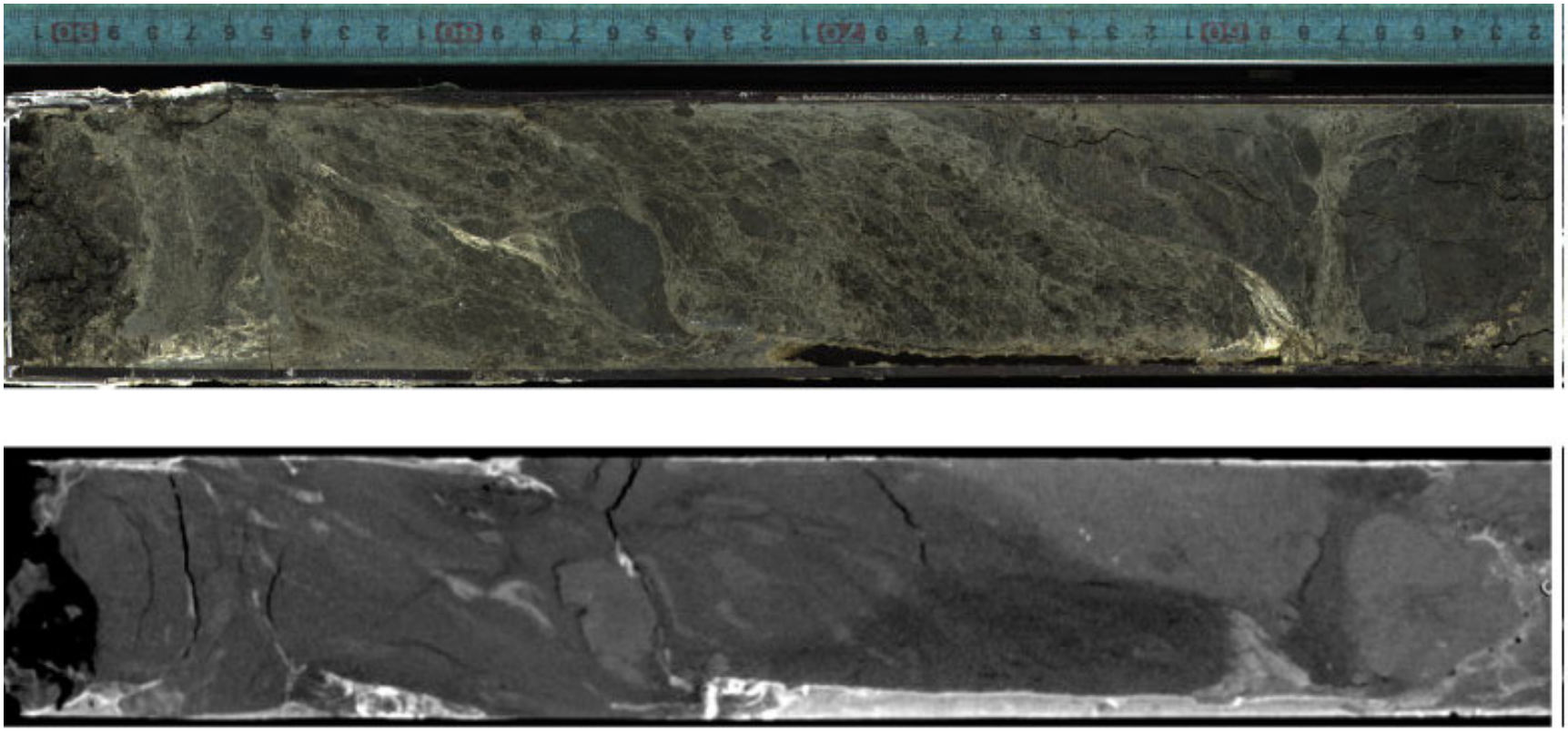
Indice

Descrizione del contesto

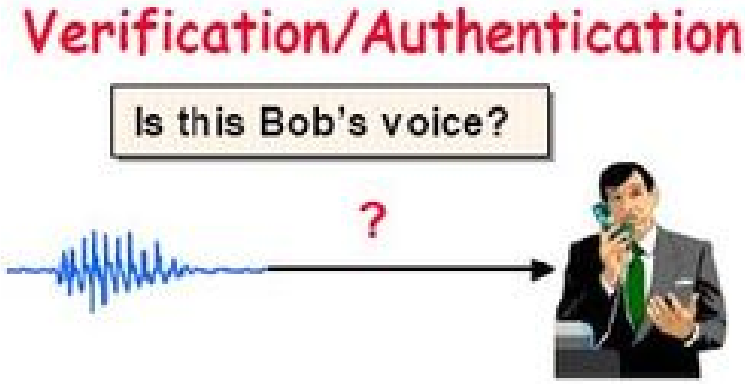
Il *deep learning* ha cominciato a svilupparsi a partire dal 2010, ed è nato in un contesto informatico.



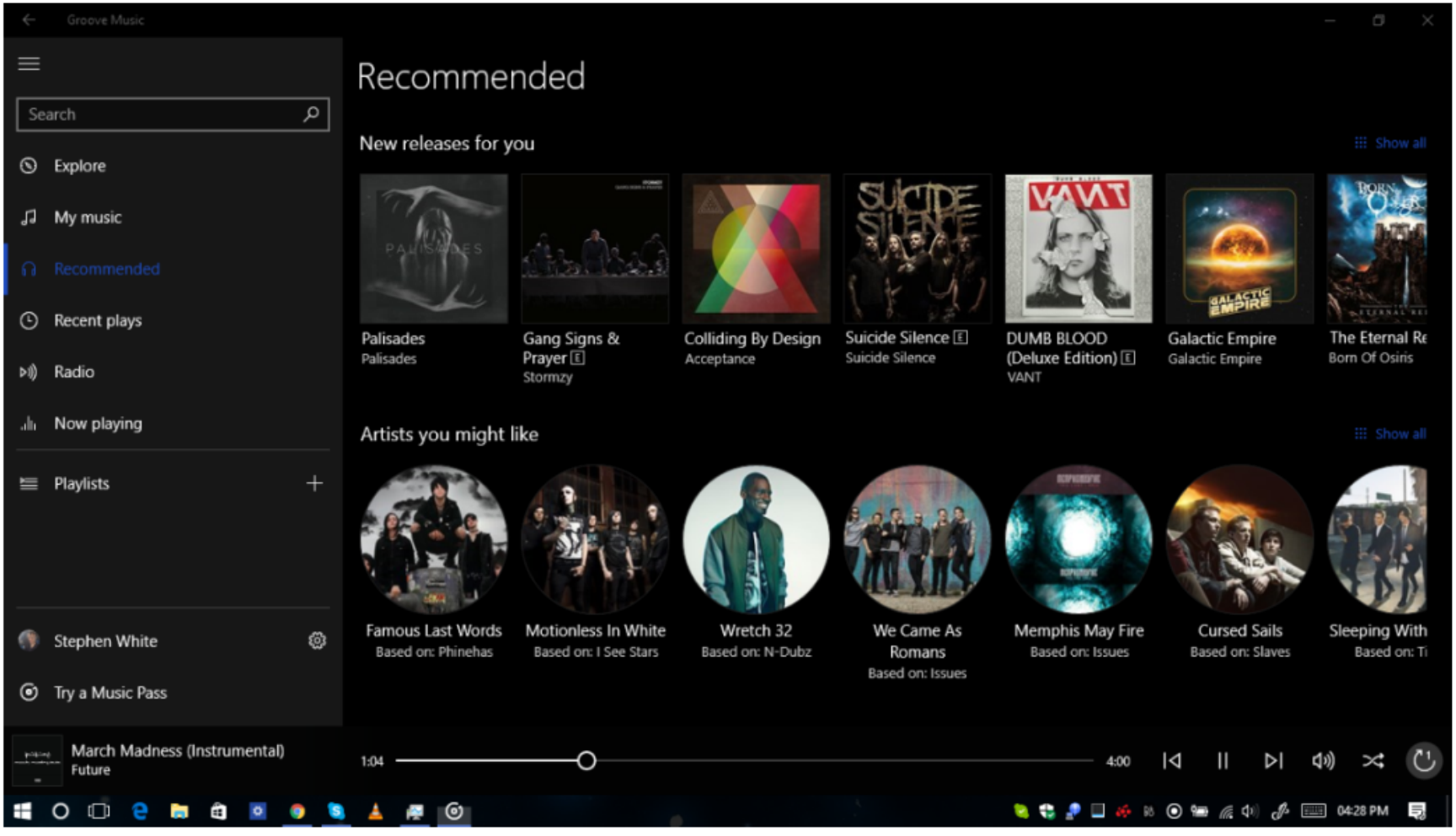
Allineamento di immagini



Riconoscimento vocale



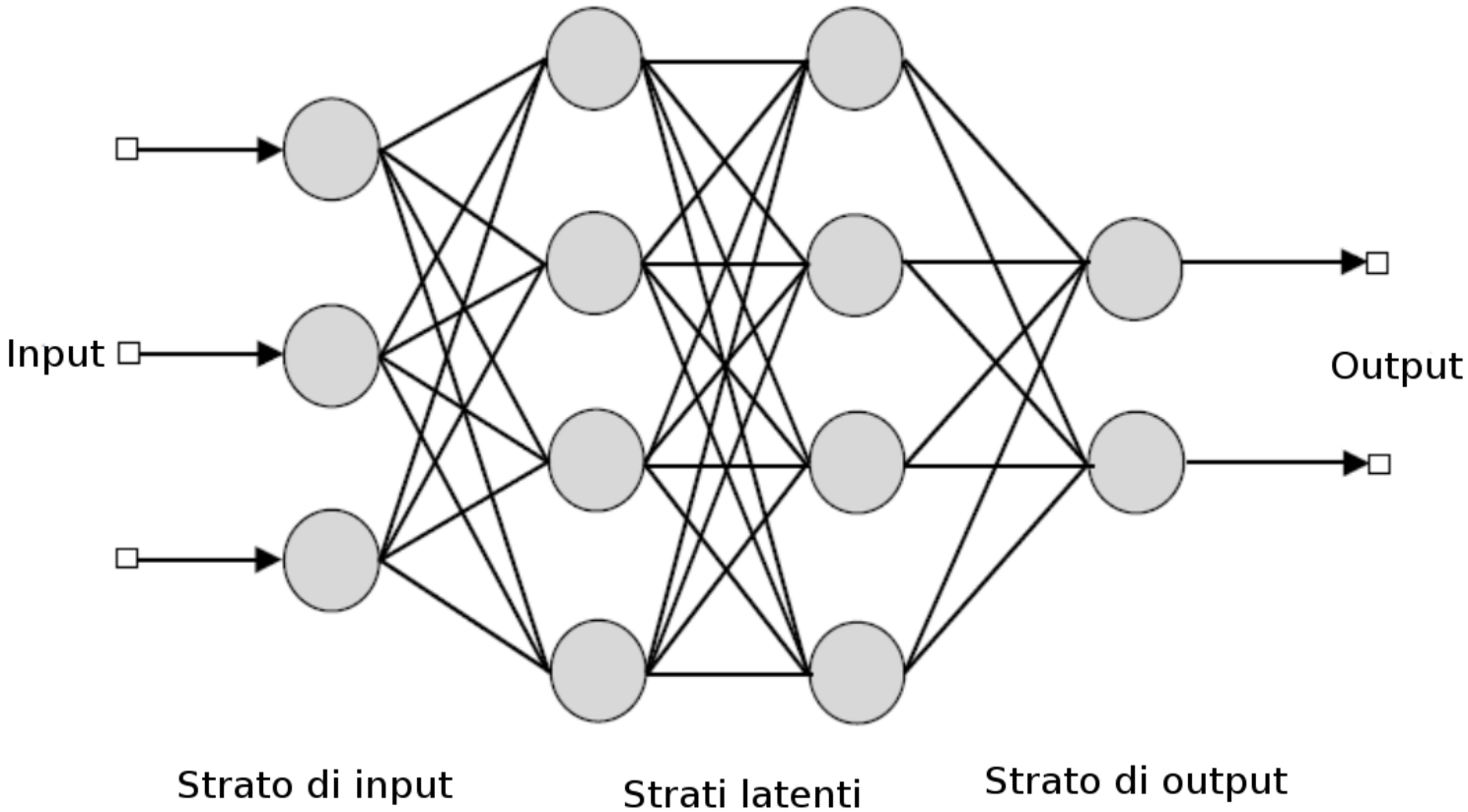
Sistema di raccomandazione



Indice

La struttura di una *deep neural network*

In seguito è raffigurata la struttura della *feed-forward neural network* (FFNN), i cui principali elementi sono i **nodi** e gli **archi**. Ad ogni arco è associato un **parametro**.



La struttura di una *deep neural network*

Sia

- $\mathbf{a}^{(l)}$: vettore di nodi corrispondenti allo strato l-esimo;
- $\mathrm{W}^{(l)}$: matrice di parametri associati agli archi che collegano lo strato l-esimo con lo strato successivo.

Come è legato il generico strato l con lo strato precedente $l-1$?

$$\mathbf{z}^{(l)} = \mathrm{W}^{(l-1)}\mathbf{a}^{(l-1)}$$

$$\mathbf{a}^{(l)} = g^{(l)}(\mathbf{z}^{(l)})$$

dove $g^{(l)}(\cdot)$ viene chiamata **funzione di attivazione**.

La struttura di una *deep neural network*

Problema di regressione univariato

- Si ha tipicamente un solo nodo di output.
- Una opportuna scelta della funzione di attivazione applicata all'ultimo strato latente è la **funzione identità**:

$$g^{(L)}(\mathbf{z}^{(L)}) = \mathbf{z}^{(L)}$$

Problema di classificazione

- Il numero di nodi di output coincide con il numero di classi della variabile risposta.
- Una opportuna scelta della funzione di attivazione applicata all'ultimo strato latente è la **funzione softmax** (logistica multinomiale):

$$g^{(L)}(\mathbf{z}^{(L)}) = \frac{e^{\mathbf{z}^{(L)}}}{\sum_{j=1}^{K} e^{\mathbf{z}^{(L)}}}$$

Stima dei parametri

Sia $\mathbf{W} = [\mathrm{W}^{(1)}\mathrm{W}^{(2)} \dots \mathrm{W}^{(L-1)}]$, la stima dei parametri $\hat{\mathbf{W}}$ della rete neurale è data da:

$$\hat{\mathbf{W}} = \arg\min \left\{ \frac{1}{n} \sum_{i=1}^{n} L[y_i, f(x_i; \mathbf{W})] \right\}$$

Principale funzioni di perdita per problemi di **regressione**:

- **Errore quadratico medio**: $\mathrm{MSE}(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i; \mathbf{W}))^2$

Principale funzioni di perdita per problemi di **classificazione**:

- **Cross-entropia**: $\mathrm{H}(\mathbf{W}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik} \log f_k(x_i; \mathbf{W})$

Algoritmo di *backpropagation*

L'algoritmo largamente più utilizzato per stimare le reti neurali, sia a strato singolo che multi-strato, è la *backpropagation*.
Questo algoritmo:

- ha la capacità di stimare i parametri con un basso costo computazionale;
- è iterativo (ogni iterazione dell'algoritmo viene chiamata **epoca**);
- è composto da due fasi: con il *passo in avanti* si ottiene $\hat{f}(x_i; \mathbf{W})$, tenendo fisso $\mathbf{W}$, mentre con il *passo all'indietro* vengono aggiornati i parametri;
- necessita solamente del calcolo della **derivata prima**, la quale si ottiene in modo efficiente nel *passo all'indietro*, che a sua volta si suddivide in *fase di propagazione* e *fase di aggiornamento*.

Fase di propagazione: *equazione di back-propagation*

Obiettivo: ricavare le derivate parziali della funzione di perdita rispetto ai parametri.

1. Data la generica osservazione i, si ricavano le derivate parziali della funzione di perdita rispetto a $\mathbf{z}$, definite $\delta_i^{(2)}(\mathbf{z}), ..., \delta_i^{(L)}(\mathbf{z})$.
   Di queste, è necessario calcolare solamente $\delta_i^{(L)}(\mathbf{z})$, quella riferita allo strato di output.
2. Il gradiente viene propagato all'indietro, in modo ricorsivo, attraverso l'**equazione di back-propagation** (operazione lineare).
3. Calcolati $\delta_i^{(2)}(\mathbf{z}), ..., \delta_i^{(L)}(\mathbf{z})$, è possibile ricavare le derivate parziali della funzione di perdita rispetto ai parametri, $\delta_i^{(1)}(\mathrm{W}), ..., \delta_i^{(L-1)}(\mathrm{W})$, con una semplice operazione lineare.

Fase di aggiornamento: *gradient descent*

Obiettivo: aggiornare i parametri usando le derivate calcolate nella *fase di propagazione*.

Il **gradient descent** è una tecnica numerica iterativa, che permette di trovare il punto di ottimo di una funzione in più variabili. L'aggiornamento dei parametri, quindi, avviene secondo la formula

$$\mathrm{W}_{t+1}^{(l)} = \mathrm{W}_{t}^{(l)} - \eta \cdot \Delta L(\mathrm{W}_{t}^{(l)}), \qquad \text{per } l = 1, ..., L-1$$

dove

$$\Delta L(\mathrm{W}^{(l)}) = \frac{1}{n} \sum_{i=1}^{n} \delta_i^{(l)}(\mathrm{W}).$$

Learning rate

Il parametro di regolarizzazione $\eta \in (0, 1]$ viene chiamato *learning rate*, e determina la grandezza dello spostamento.

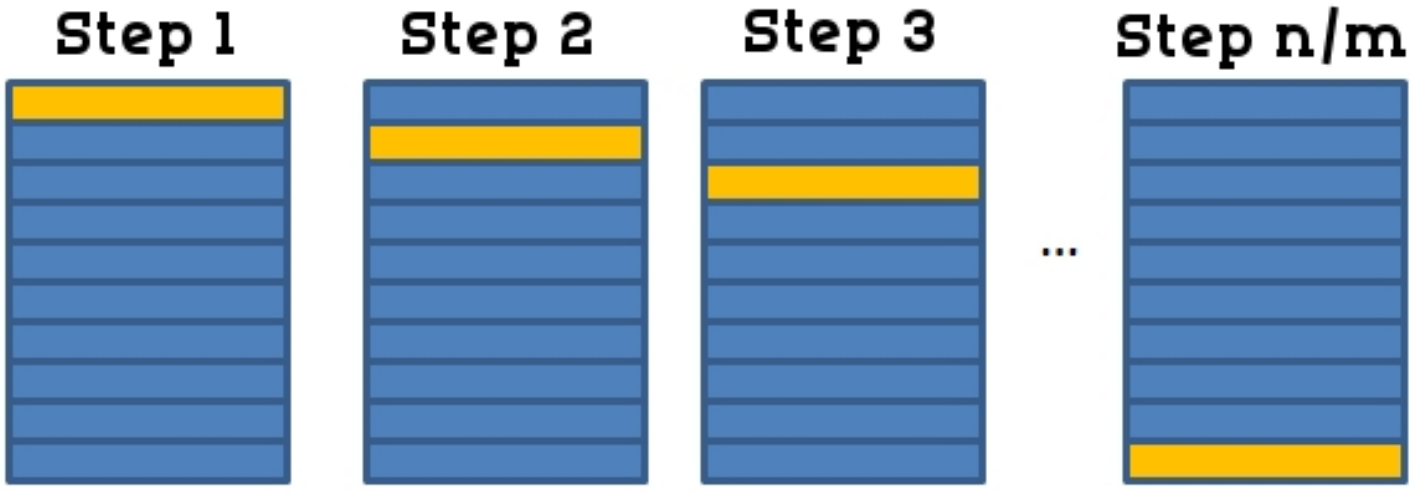
Mini-batch gradient descent

Problema

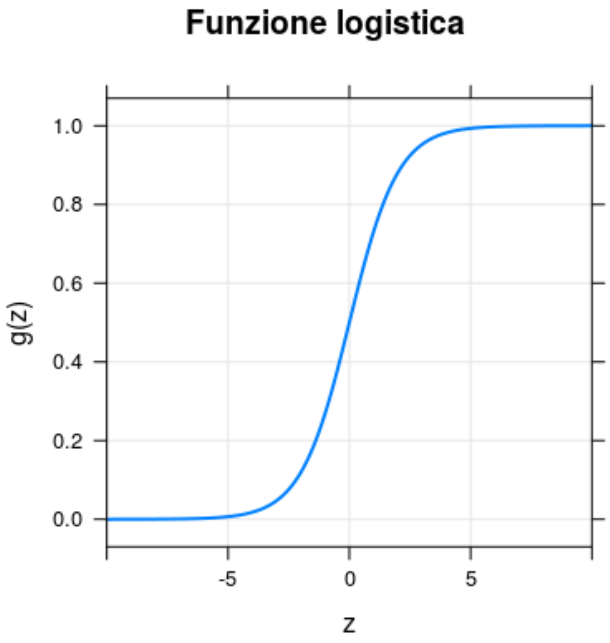
L'utilizzo di tutti i dati per effettuare un solo passo di aggiornamento comporta costi computazionali notevoli e rallenta di molto la procedura di stima.

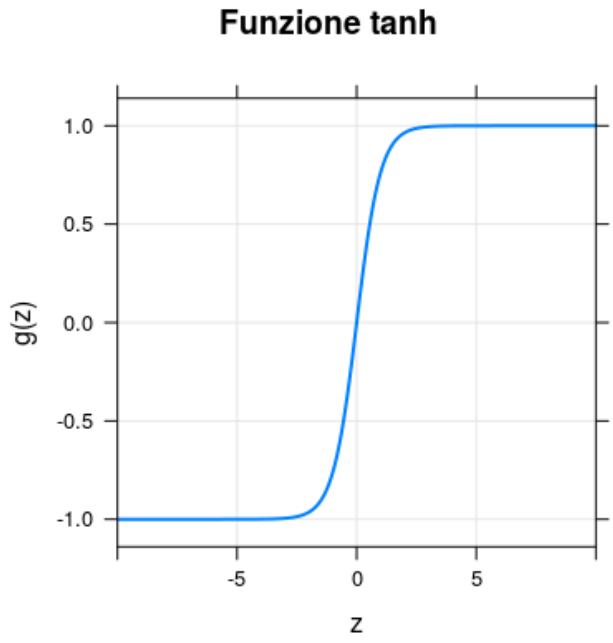
Soluzione

Viene introdotta la tecnica del *mini-batch (o stochastic) gradient descent*. Ciò consiste nel suddividere il *dataset* in sottocampioni di numerosità fissata $m \ll n$, dopo una permutazione casuale dell'intero insieme di dati. L'aggiornamento viene quindi attuato utilizzando ciascuno di questi sottoinsiemi.



Le classiche funzioni di attivazione delle reti neurali





Funzione logistica (sigmoidale)

$$\text{logistica}(z) = \frac{1}{1 + e^{-z}}.$$

Tangente iperbolica

$$\begin{aligned}\tanh(z) &= \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \\ &= 2 \cdot \text{logistica}(2z) - 1.\end{aligned}$$

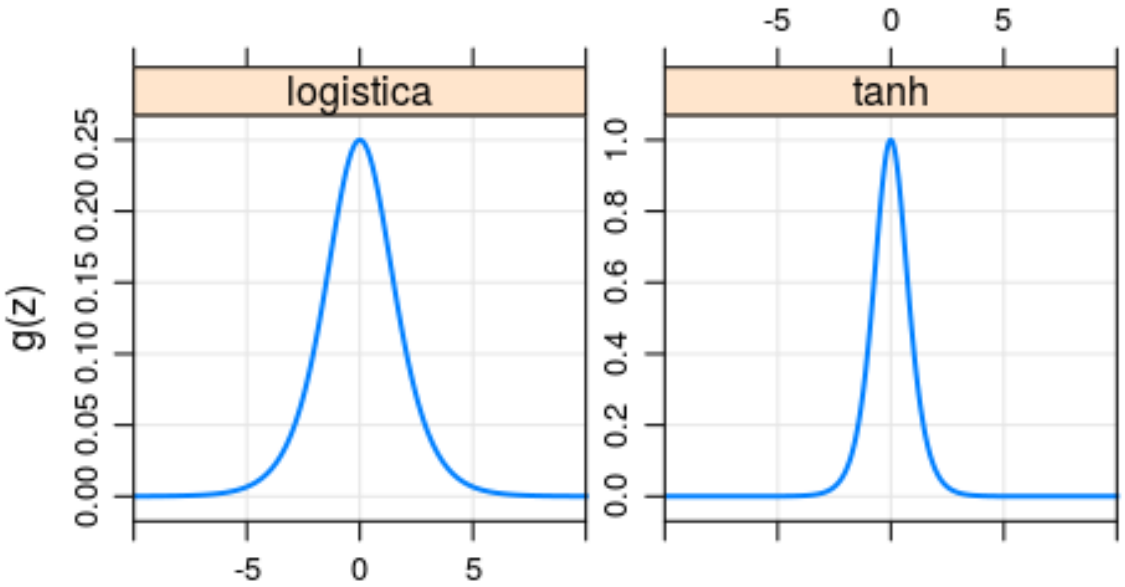
Le problematiche delle funzioni di attivazione classiche

Problema 1

Una funzione di attivazione limitata può ridurre la flessibilità del modello. Cambiamenti anche rilevanti di **z**, ma lontani dallo 0, corrispondono a variazioni quasi inesistenti della funzione.

Problema 2: "scomparsa del gradiente"

Nella fase di stima, quando i valori di **z** si avvicinano agli asintoti orizzontali della funzione di attivazione, il gradiente di questa funzione tende a 0.



La funzione di attivazione ReLU

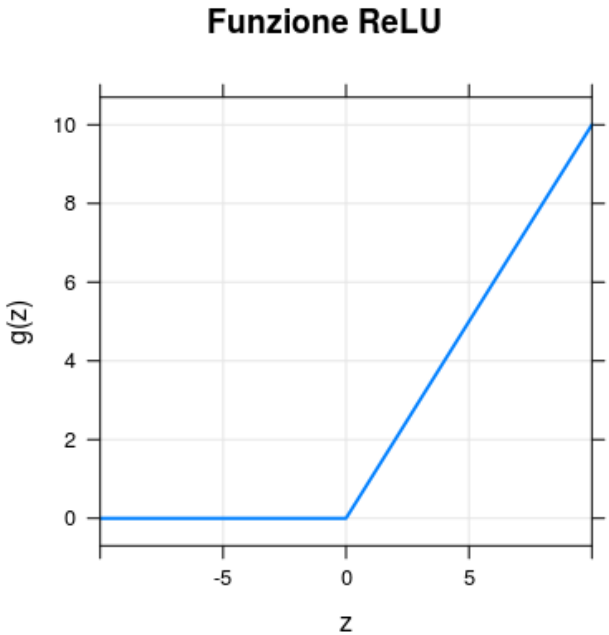
Soluzione

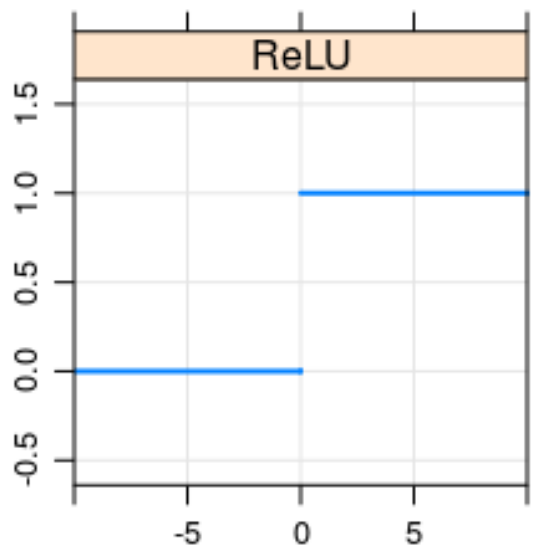
Si può utilizzare la funzione di attivazione *rectified linear unit* (ReLU). Quest'ultima è definita come

$$g(z) = z_+ = \max(0, z).$$

La funzione ReLU non soffre della *scomparsa del gradiente*:

$$g'(z) = \begin{cases} 0, & \text{se } z \leq 0 \\ 1, & \text{se } z > 0 \end{cases}$$





Compromesso varianza-distorsione

La selezione del modello ottimale, in termini di accuratezza previsiva, deve essere condotta facendo un *compromesso* tra *varianza* e *distorsione*.

Complessità del modello

Un primo modo per effettuare questo compromesso è quello di regolare la **complessità del modello** scegliendo quello che minimizza l'errore di previsione nell'*insieme di verifica*.
La complessità del modello è stabilita dal numero di parametri, il quale è legato in modo deterministico al numero di **nodi** e di **strati latenti**.

Early stopping

Un secondo modo è bloccare l'algoritmo di *backpropagation* dopo un certo numero di epoche.

Altri metodi di regolarizzazione

Esistono altri metodi per effettuare questo compromesso, come ad esempio i **metodi di penalizzazione** ed il **dropout**.

Metodi di penalizzazione

Applicando il metodo di penalizzazione, la stima dei parametri $\hat{\mathbf{W}}$, diventa quindi

$$\hat{\mathbf{W}} = \arg\min \left\{ \frac{1}{n} \sum_{i=1}^{n} L[y_i, f(x_i; \mathbf{W})] + \lambda J(\mathbf{W}) \right\},$$

dove $J(\mathbf{W})$ è un *termine di regolarizzazione* non negativo, mentre $\lambda \geq 0$ è un *parametro di regolarizzazione*.

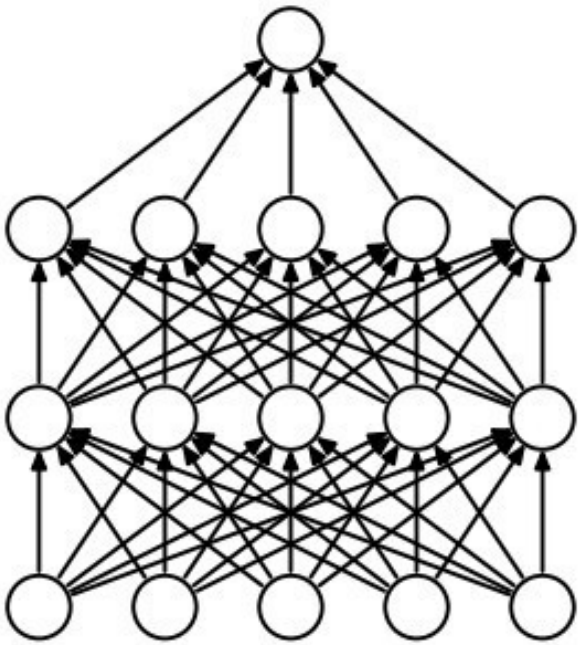
La penalità più utilizzata: *weight decay*, o penalità L_2

$$J(\mathbf{W}) = \frac{1}{2}\|\mathbf{W}\|_2^2 = \frac{1}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{p_l} \sum_{j=1}^{p_{l+1}} \left(w_{ij}^{(l)} \right)^2 .$$

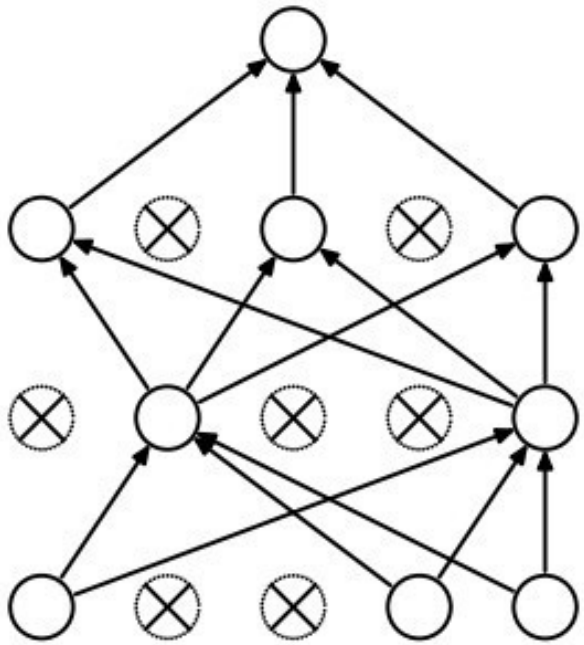
Dropout

Il metodo *dropout*

La tecnica del *dropout* consiste nel porre, ad ogni iterazione della procedura di *backpropagation*, una porzione di nodi pari a zero. Questa porzione viene scelta casualmente.



Rete neurale standard



Dopo l'applicazione del *dropout*

Dropout

Caratteristiche del *dropout*

- il *dropout* è un'approssimazione del risultato che si otterrebbe attraverso la combinazione di classificatori;
- il *dropout* è nato pensando alla logica del campionamento casuale delle variabili della *random forest*;
- viene risolto il problema dell'*overfitting* e si ha anche un netto miglioramento dell'accuratezza previsiva;
- la probabilità p di conservare un nodo nel modello è un parametro di regolarizzazione;
- Il *dropout* esprime tutta la sua efficacia con un elevato numero di osservazioni.

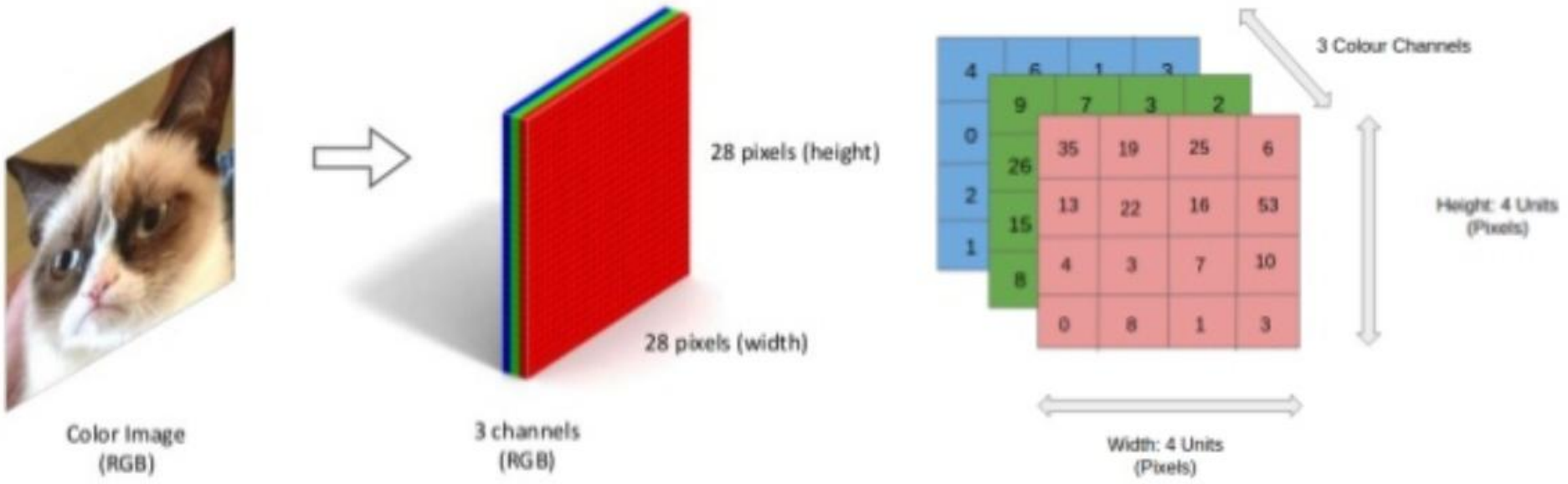
Indice

Convolutional neural network - La struttura del dato

Le *convolutional neural networks* (CNN) sono una classe di reti neurali, che funziona in modo ottimale nella classificazione di immagini.

Struttura del dato

Un'immagine possiede la struttura di un tensore a 3 dimensioni: le prime due dimensioni rispecchiano la disposizione dei *pixel*, mentre la terza dimensione è la rappresentazione del colore (RGB).



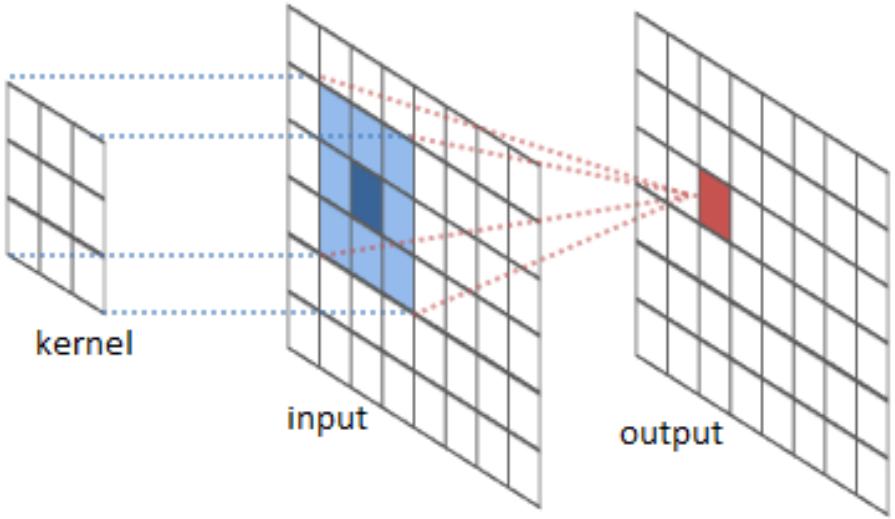
La convoluzione

L'operazione di convoluzione:

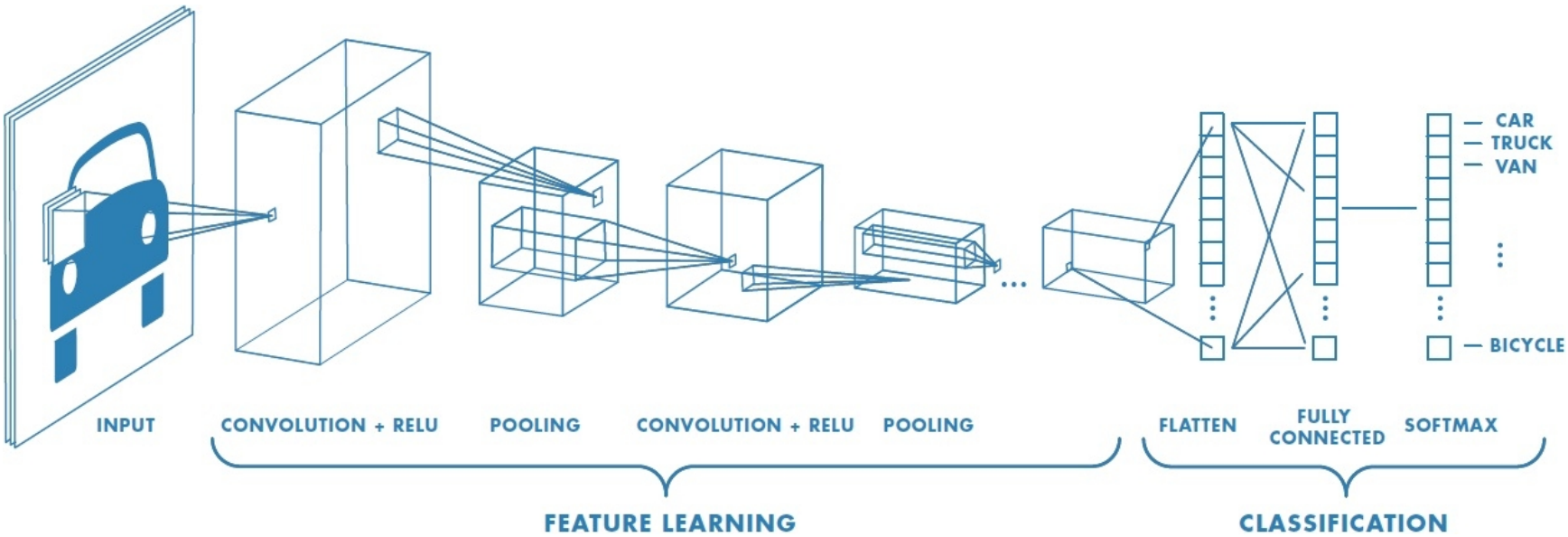
- $x \rightarrow$ immagine (input) di dimensioni $k \times k \times 3$;
- $f \rightarrow$ filtro (kernel) di dimensioni $q \times q$;
- $\tilde{x} \rightarrow$ immagine (output) di dimensioni $k \times k \times 1$.

Il generico elemento (i, j) di $\tilde{x}$ si ottiene dal *prodotto-interno* tra f e la corrispondente sotto-immagine di x:

$$\tilde{x}_{i,j} = \sum_{h=1}^{3} \sum_{l=0}^{q-1} \sum_{l'=0}^{q-1} x_{i+l,j+l',h} \cdot f_{l,l'}$$



Convolutional neural network - La struttura della rete



La struttura della CNN è divisa in due parti:

- *Parte convoluzionale*: la prima serie di strati dopo l'input alterna uno **strato convoluzionale** con uno **strato di pooling**.
- *Parte fully-connected*: la seconda serie di strati sono **fully-connected**, cioè esattamente come quelli delle *feed-forward neural networks*.

Convolutional neural network - La struttura della rete

Lo strato convoluzionale è costituito da p "versioni" differenti dell'immagine in entrata. Ognuna di queste "versioni" è il risultato dell'applicazione di un **filtro**. Il filtro viene moltiplicato (prodotto-interno) ad ogni sotto-immagine delle stesse dimensioni del filtro. I valori del filtro sono i parametri della rete.
Sono sottoposti ad una fase di **regolarizzazione**:

- il *numero di filtri* prodotti dalla convoluzione;
- la *dimensione dei filtri* (tipicamente 3×3 o 5×5);
- lo *spostamento del filtro* (*stride*).

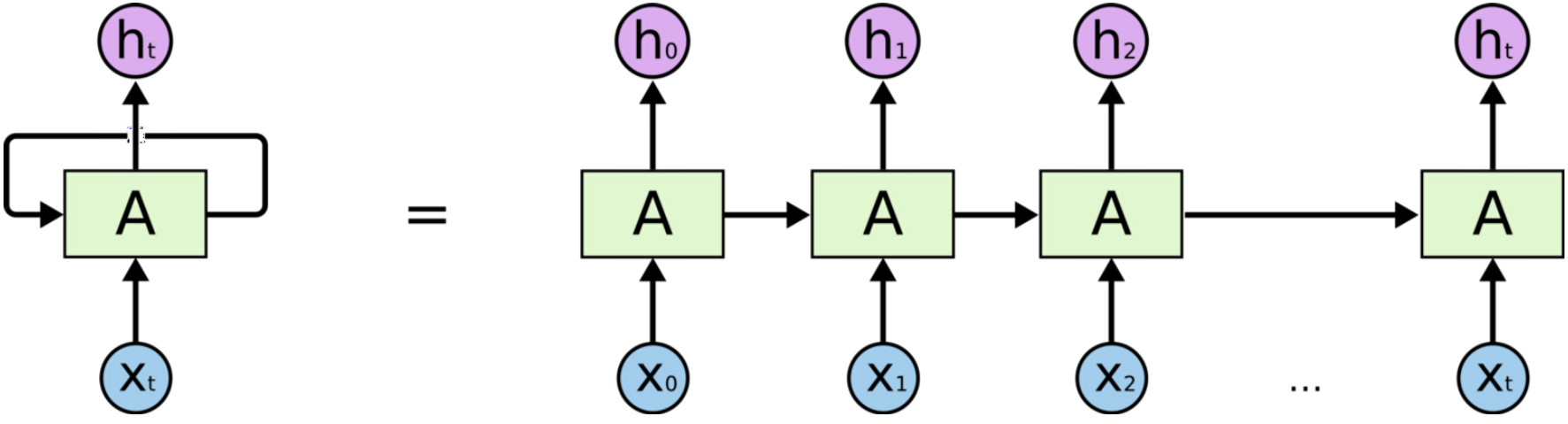
Lo strato di *pooling* suddivide ogni immagine in piccole parti di dimensione $r \times r$, e di ognuna di queste prende il valore **massimo**. Con ciascun valore massimo ricostruisce un'immagine di dimensioni ridotte. Questo permette di ridurre il numero di parametri senza perdere informazione.

Recurrent neural network - La struttura della rete

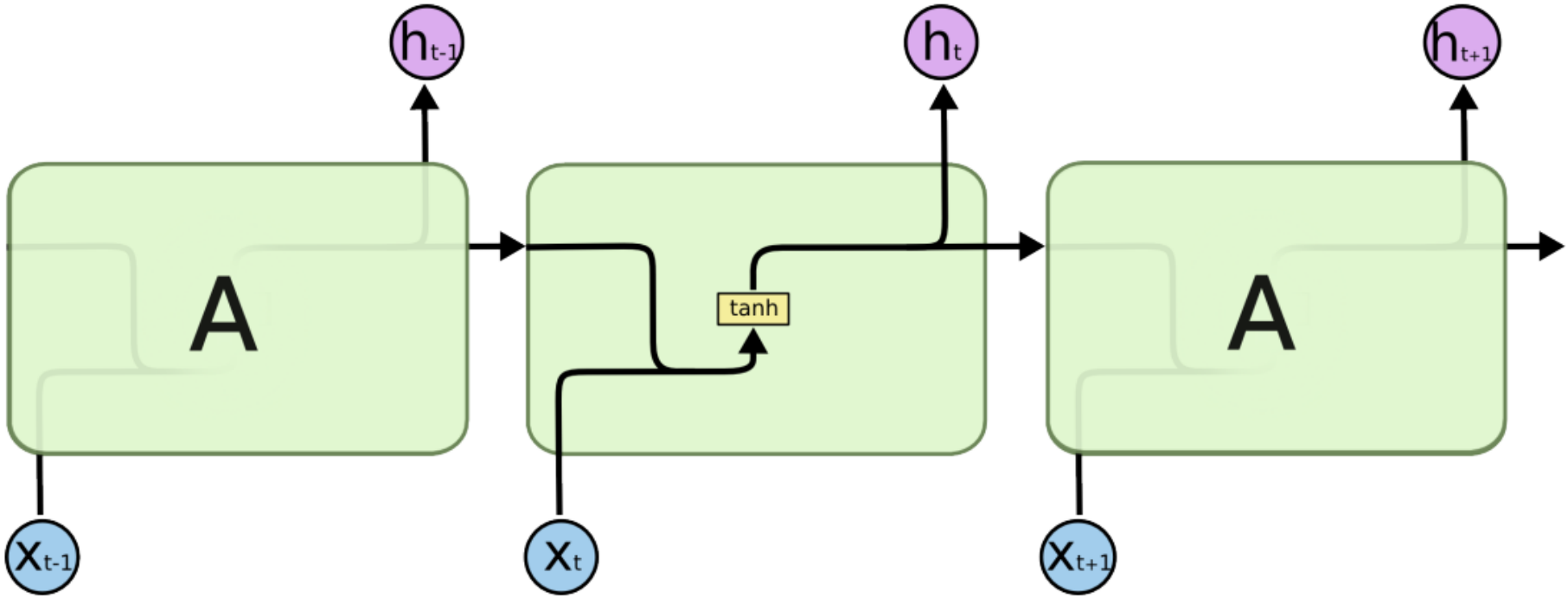
Una *recurrent neural network* (RNN) è un tipo di rete neurale, che funziona in modo ottimale nell'analisi di dati sequenziali, come serie storiche o dati testuali.

Definizione: Una RNN possiede uno o più **strati ricorrenti**

Uno **strato ricorrente** è uno strato latente che, non solo connette lo strato di input con lo strato latente successivo, ma connette anche lo strato latente al tempo precedente con lo strato latente al tempo successivo.

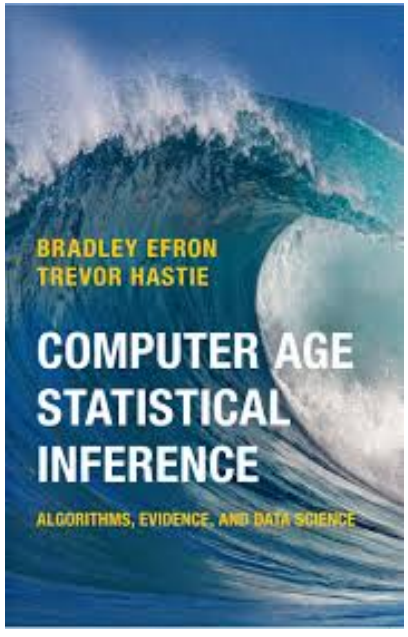


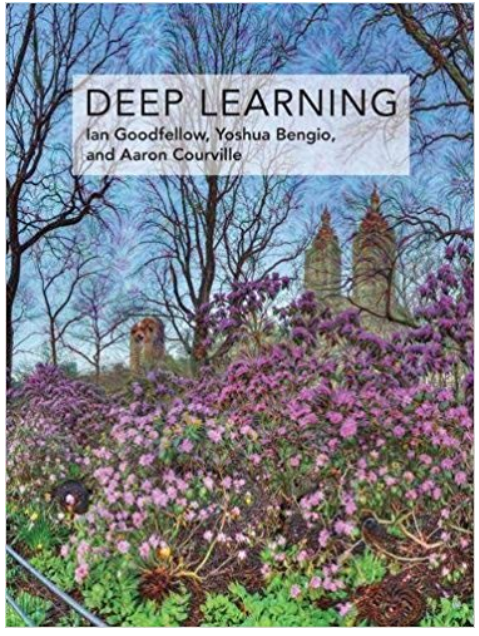
Lo strato ricorrente

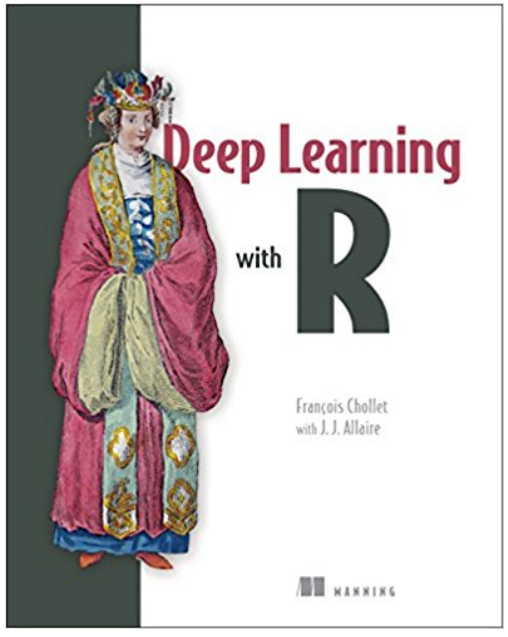


Al vettore in input al tempo t viene applicata la *funzione di attivazione* (generalmente la *tangente iperbolica*), ottenendo come vettore di output l'**hidden state** al tempo t (h_t). Questo vettore viene concatenato a x_{t+1} e usato come input dello strato latente al tempo $t+1$.

Riferimenti utili







Quando il *deep learning* può non funzionare?

1) Non funziona così bene con pochi dati.

Confronto tra modelli stimati con 25 000 recensioni del dataset di *IMDB*.

Modello	**Accuratezza**
RNN	**88.0 %**
Lasso	87.0 %
FFNN	**86.5 %**
Regressione logistica	86.2 %
Random forest	84.7 %
Bagging	77.0 %
Adaboost	72.5 %
Gradient boosting	70.1 %

Confronto tra modelli stimati con 5 000 recensioni del dataset di *IMDB*.

Modello	**Accuratezza**
Lasso	85.0 %
Regressione logistica	85.0 %
FFNN	**83.5 %**
RNN	**83.4 %**
Random forest	83.3 %
Bagging	75.2 %
Adaboost	70.8 %
Gradient boosting	69.4 %

Quando il *deep learning* può non funzionare?

2) Le *deep neural networks* sono delle vere e proprie **black box**:

- non si può interpretare ciò che accade all'interno di una *deep neural network*;
- sono potenti strumenti, utili quando lo scopo è solamente ottimizzare la capacità previsiva, ma totalmente inutili se ci cerca in qualche modo un'interpretazione del problema;
- recentemente in letteratura si possono trovare soluzioni a questo problema, ma a mio avviso sono delle forzature poco affidabili rispetto ad altri modelli statistici;
- in questo caso, è molto meglio ricorrere a modelli lineari, modelli ad albero, o combinazioni di classificatori, che offrono potenti strumenti per individuare i fattori che influenzano la varibile risposta, come ad esempio l'interpretazione dei coefficienti, le regole decisionali o l'importanza delle variabili;

Quando il *deep learning* può non funzionare?

3) Il *deep learning* è particolarmente oneroso in termini di tempo e risorse:

- ogni volta che si vuole affrontare un problema con il *deep learning*, una grossa parte del tempo viene utilizza per la fase di "ricerca";
- implementare una rete neurale è molto complesso e richiede solide basi di programmazione;
- effettuare il *training* di una rete neurale spesso può portare via molte ore, ed è totalmente inutile se ci serve una stima affidabile nel breve tempo;
- la quantità di elementi da *regolarizzare* è elevatissima;
- spesso i costi sono elevati, in quanto richiede una potenza di calcolo superiore agli altri modelli statistici: è necessario essere in possesso di una costosa scheda grafica, oppure noleggiare potenti macchine in *cloud*. Questi strumenti richiedono anche molto tempo per essere configurati.

La libreria `Keras`

Le caratteristiche principali della libreria `Keras` sono:

- è implementata sia in `R` che in `python`;
- permette la stima di tutte le classi di modelli per il *deep learning* supervisionato e non supervisionato;
- si serve del calcolo parallelo;
- può utilizzare l'unità di elaborazione grafica (GPU) per la stima del modello;
- si interfaccia alla libreria `TensorFlow` di `python`.

Consiglio!

Rispetto ad `R`, `python` ha una miglior gestione della memoria ed una maggior velocità computazionale.