
# *Deep Learning*

Alessandro Aere

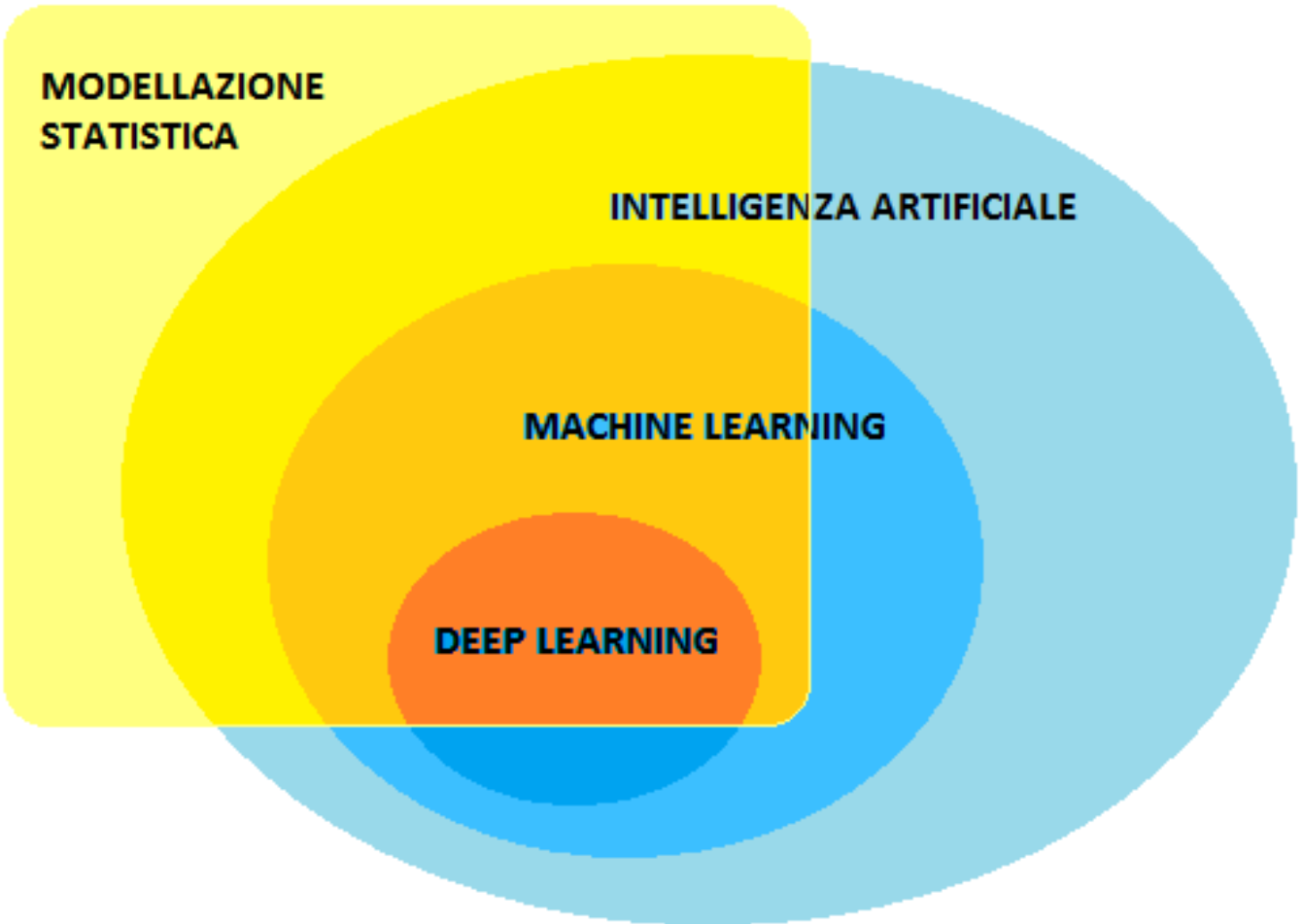
Università degli Studi di Padova

24 maggio 2019

# Indice

# Descrizione del contesto

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



# Descrizione del contesto

I più rilevanti utilizzatori di *deep learning*:

# Alcuni campi di applicazione del *deep learning*

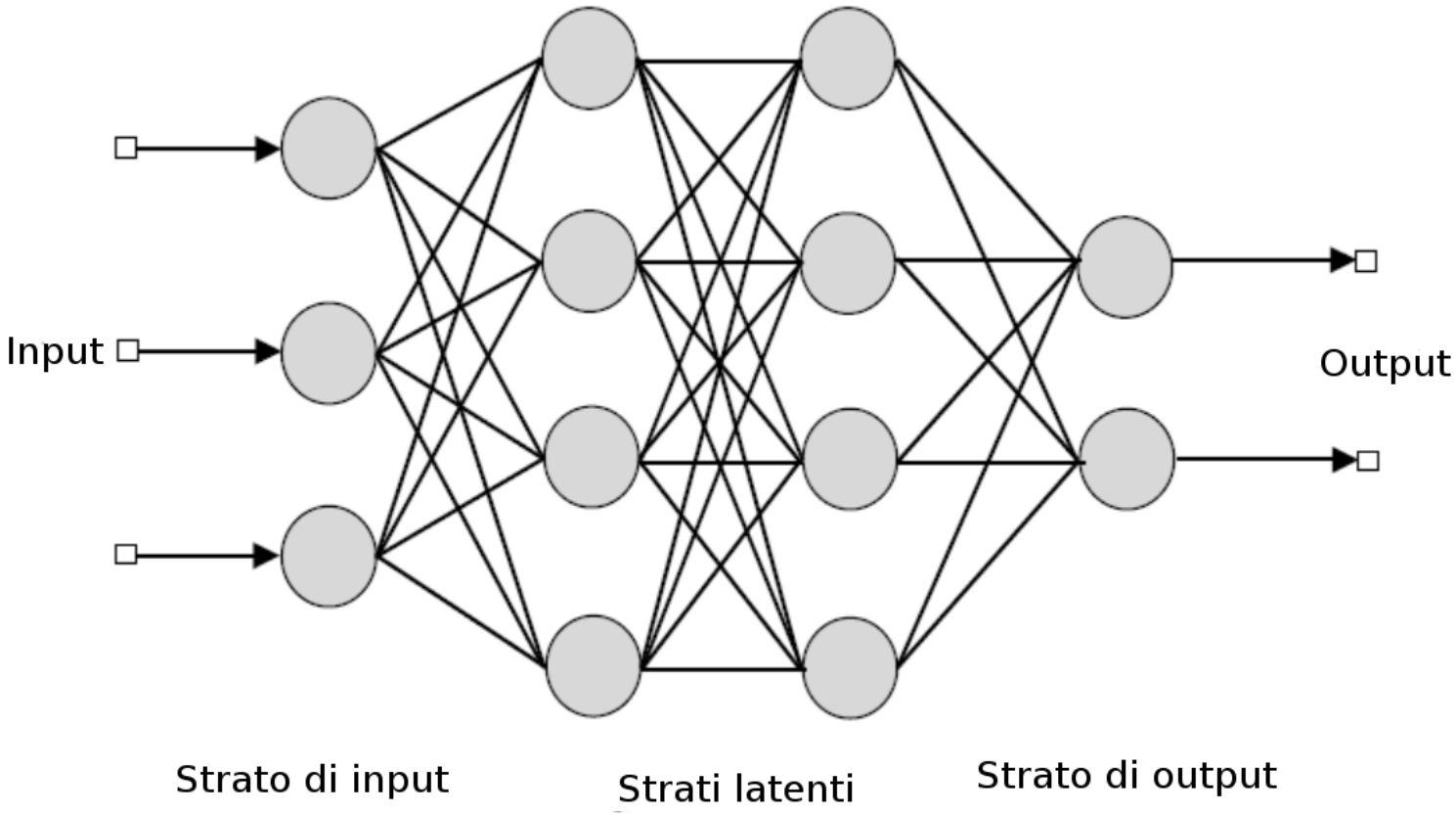
Alcuni campi applicativi del *deep learning* sono:

- Riconoscimento di immagini
- Elaborazione del linguaggio naturale (NLP)
- Riconoscimento vocale
- Traduzione automatica
- Conversione *text-to-speech*
- *Chatbot* (risposta automatica alle domande)
- Assistenti digitali (come *Amazon Alexa*)
- Sistemi di raccomandazione
- Motori di ricerca
- Guida autonoma
- *AlphaGO*

# 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

- $a_j^{(l)}$: valore del nodo $j$-esimo dello strato $l$-esimo;
- $w_{ij}^{(l)}$: coefficiente associato all'arco che collega il nodo $i$-esimo dello strato $l$-esimo con il nodo $j$-esimo dello strato $(l+1)$-esimo.

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

$$z_j^{(l)} = w_{0j}^{(l-1)} + \sum_{i=1}^{p_{l-1}} w_{ij}^{(l-1)} a_i^{(l-1)}$$

$$a_j^{(l)} = g^{(l)}(z_j^{(l)})$$

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 logistica multinomiale** (softmax):

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

# Stima dei parametri

Stima dei parametri $\hat{\mathbf{W}}$ della rete neurale:

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

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

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

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

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

# 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 del gradiente primo, il quale si ottiene in modo efficiente nel *passo all'indietro*.

# Logica dell'algoritmo di *backpropagation*

**Passo in avanti:**

1. Calcolare il valore dei nodi, utilizzando i valori correnti dei parametri.

**Passo all'indietro:**

- Fase di propagazione:
  2. L'obiettivo è quello di ricavare le derivate parziali della funzione di perdita rispetto ai parametri. Per alleggerire il costo computazionale, si ricavano prima quelle rispetto a $\mathbf{z}$, definite $\delta_2, ..., \delta_L$. Di queste è necessario calcolare solamente $\delta_L$, quella riferita all'ultimo strato.
  3. Il gradiente viene propagato all'indietro, in modo ricorsivo, attraverso l'**equazione di back-propagation** (operazione lineare).
  4. Avendo $\delta_2, ..., \delta_L$ è possibile ricavare le derivate parziali della funzione di perdita, rispetto ai parametri, con una semplice operazione lineare.
- Fase di aggiornamento:
  5. Aggiornare i parametri usando il **gradient descent** e le derivate calcolate al punto 4.

6. Usare i nuovi valori per i parametri nell'iterazione successiva (**epoca**).

# Fase di aggiornamento: *gradient descent*

## Definizione

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, al passo 5, avviene secondo la formula

$$W_{t+1}^{(l)} = W_t^{(l)} - \eta \cdot \Delta L(W_t^{(l)}; x, y), \qquad \text{per } l = 1, ..., L-1$$

dove

$$\Delta L(W_t^{(l)}; x, y) = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L[y_i, f(x_i; W)]}{\partial W_t^{(l)}}.$$

## *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, attraverso la formula

$$W_{t+1}^{(l)} = W_t^{(l)} - \eta \cdot \Delta L(W_t^{(l)}; x^{(i:i+m)}, y^{(i:i+m)}),$$

dove $(i : i + m)$ è l'indice per riferirsi al sottoinsieme di osservazioni che vanno dalla $i$-esima alla $(i + m)$-esima.

# Ulteriori sviluppi del *gradient descent*

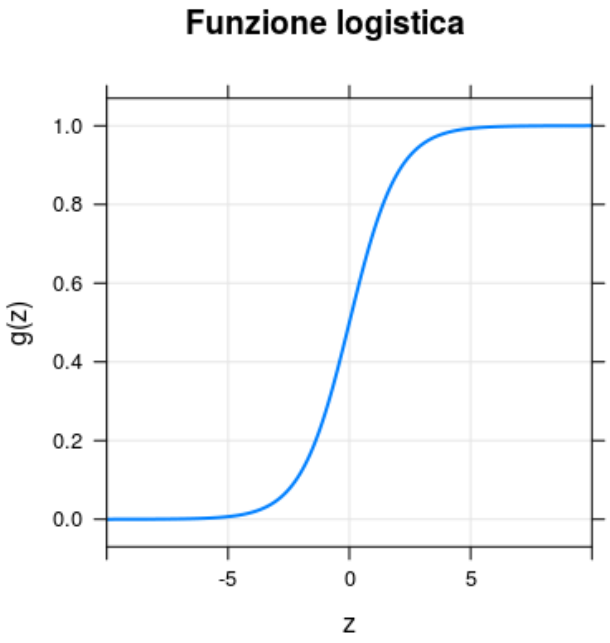
In seguito, sono stati sviluppati altri ottimizzatori per effettuare l'aggiornamento dei parametri, in modo da:

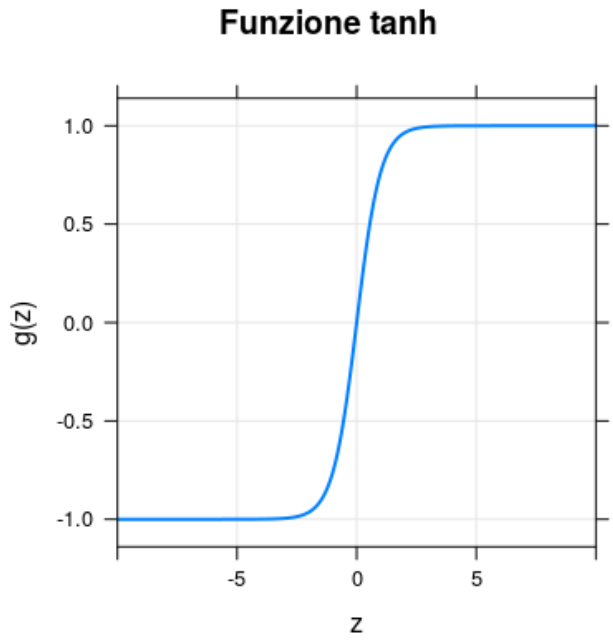
- scegliere il *learning rate* in modo adattivo, evitando la fase di regolarizzazione;
- permettere l'uso di diversi *learning rate* a seconda del parametro a cui sono affiancati;
- ridurre la propensione a rimanere intrappolati in minimi locali.

I principali ottimizzatori utilizzati nel *deep learning* sono:

- Ottimizzatori adattivi (e.g. *Adam*)
- *Stochastic gradient descent* con *learning rate scheduling*

# Le classiche funzioni di attivazione delle reti neurali




## Funzione logistica (sigmoidale)

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

## Tangente iperbolica

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

# La funzione di attivazione ReLU

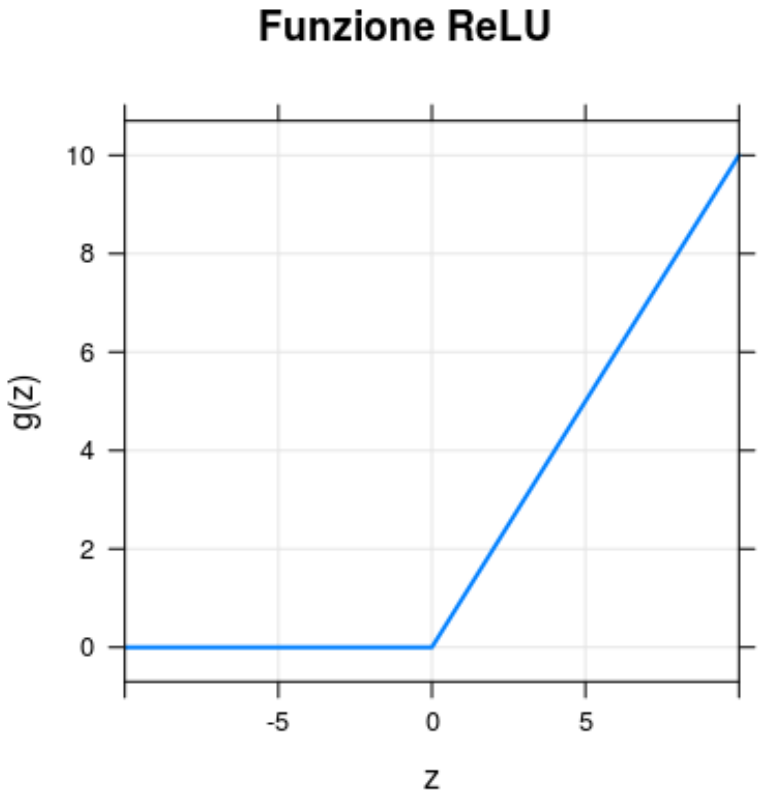
## Problema 1

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

## Soluzione

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

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

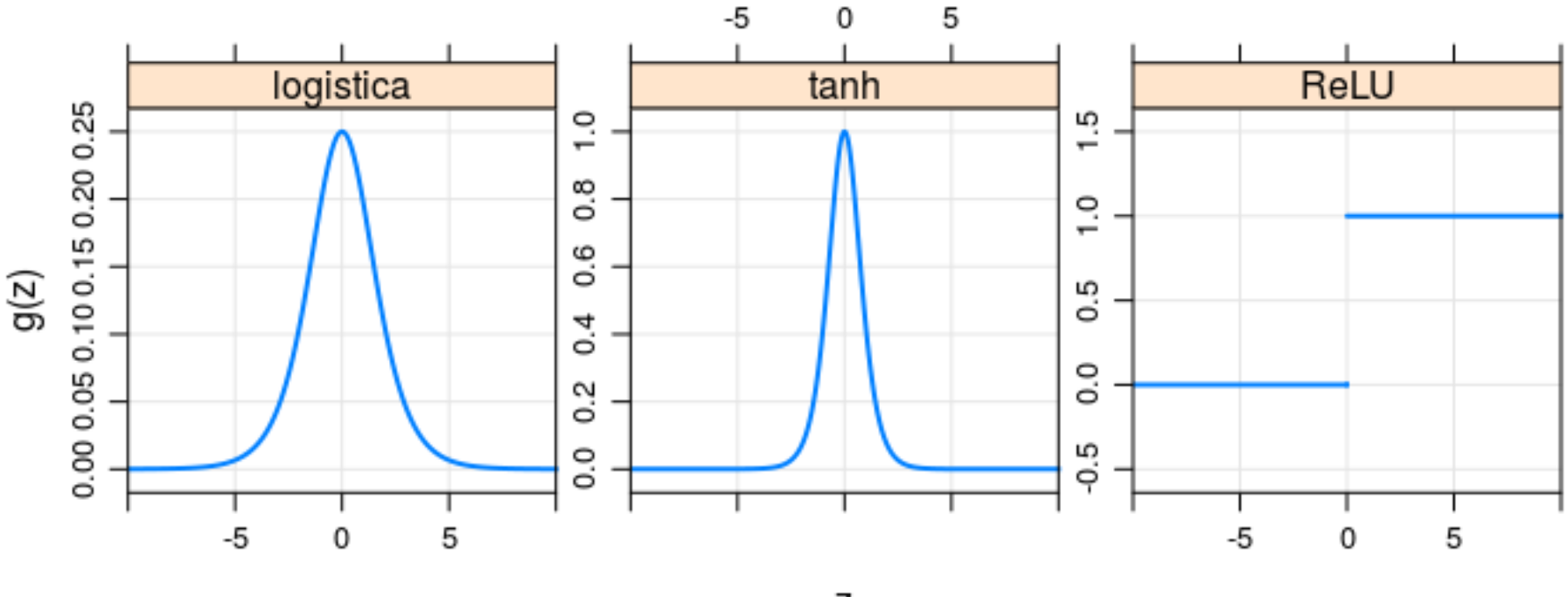


# La funzione di attivazione ReLU

## 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.

## Soluzione: $g'(z) = 1$ per $z \to \infty$



# Compromesso varianza-distorsione

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

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*.

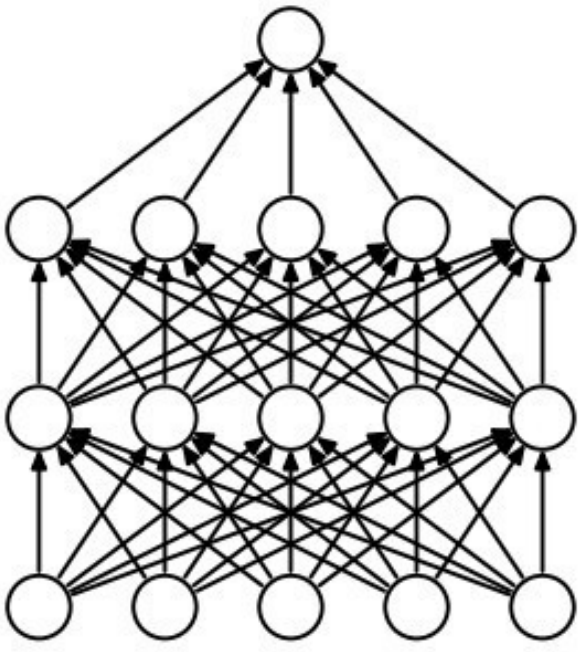
La penalità più utilizzata: *weight decay*, o penalità $L_2$

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

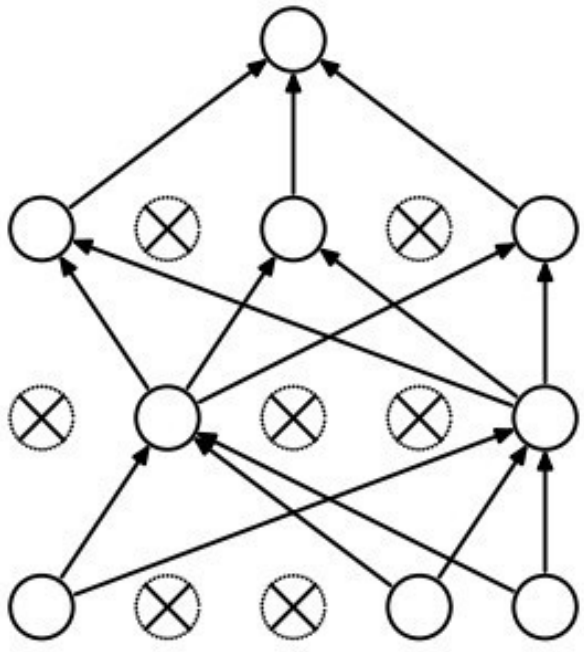
# *Dropout*

## 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.

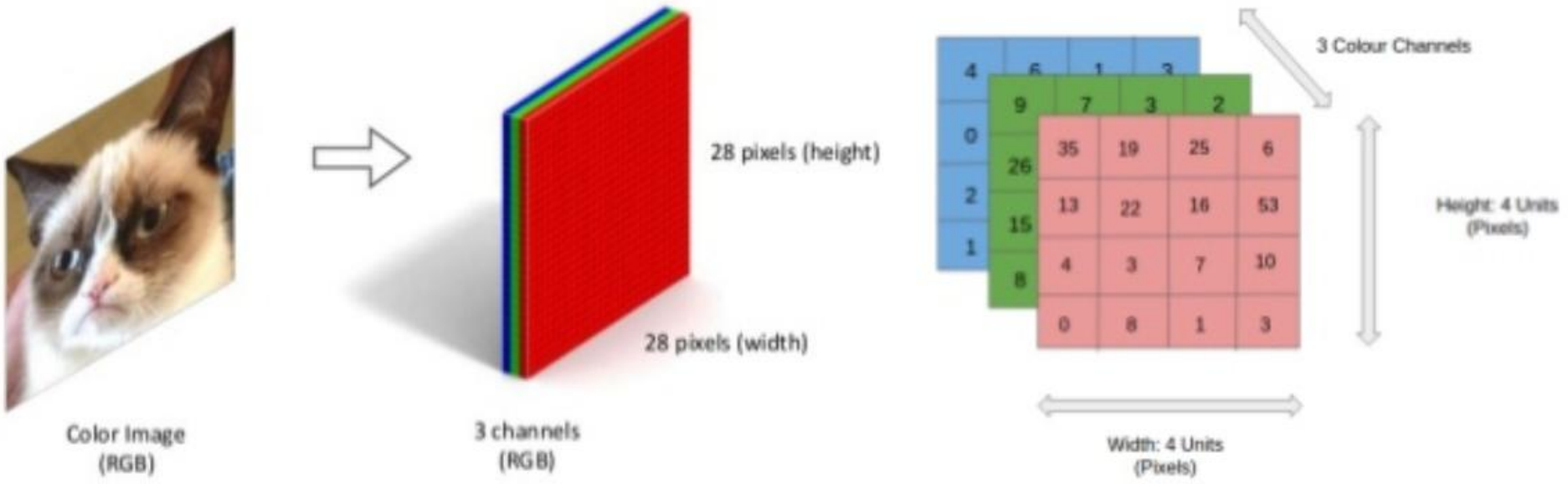
# Indice

# *Convolutional neural network* - La struttura del dato

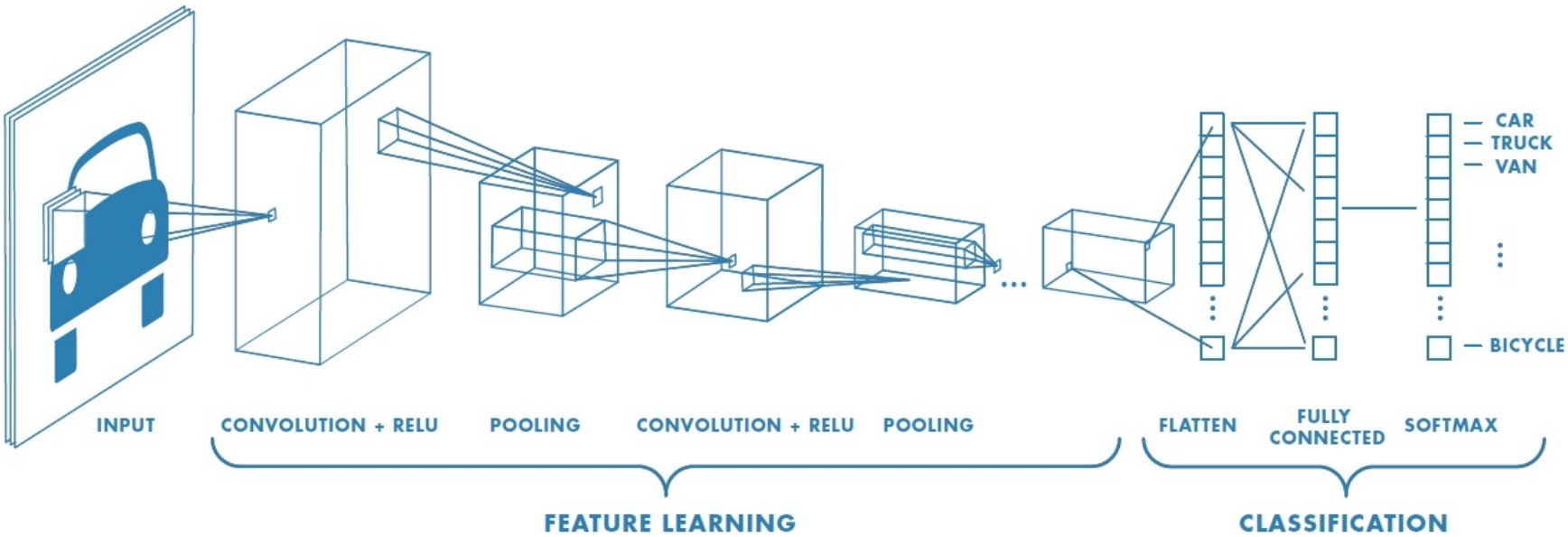
Le *convolutional neural networks* (CNN) sono una classe di reti neurali, che funziona in modo ottimale nella classificazione di immagini.

## 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).



# *Convolutional neural network* - La struttura della rete




La struttura della CNN è divisa in due parti:

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

# *Convolutional neural network* - La struttura della rete

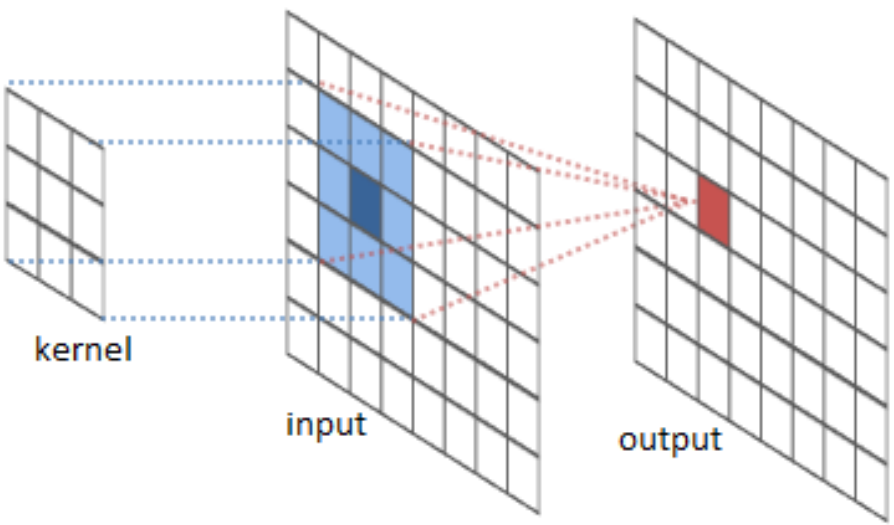
Lo strato convoluzionale è costituito da $p$ "versioni" differenti dell'immagine in entrata. Ognuna di queste "versioni" è il risultato dell'applicazione di un **filtro**. Il filtro viene moltiplicato (prodotto-interno) ad ogni sotto-immagine delle stesse dimensioni del filtro. I valori del filtro sono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

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.

# La convoluzione

## L'operazione di convoluzione

- $x \rightarrow$ immagine di dimensioni $k \times k \times 3$;
- $f \rightarrow$ filtro di dimensioni $q \times q$;
- $\tilde{x}_{i,j} = \sum_{h=1}^{3} \sum_{l=1}^{q} \sum_{l'=1}^{q} x_{i+l,j+l',h} f_{l,l'} \rightarrow$ generico elemento della "versione" trasformata dell'immagine.



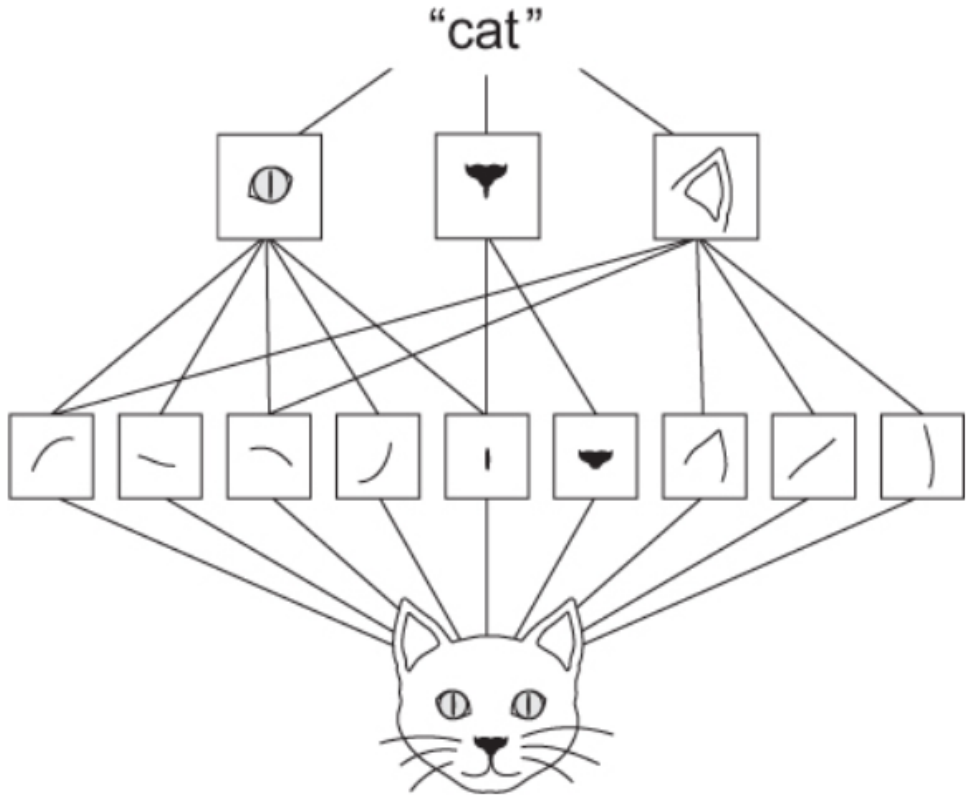

Sono sottoposti ad una fase di **regolarizzazione**:

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

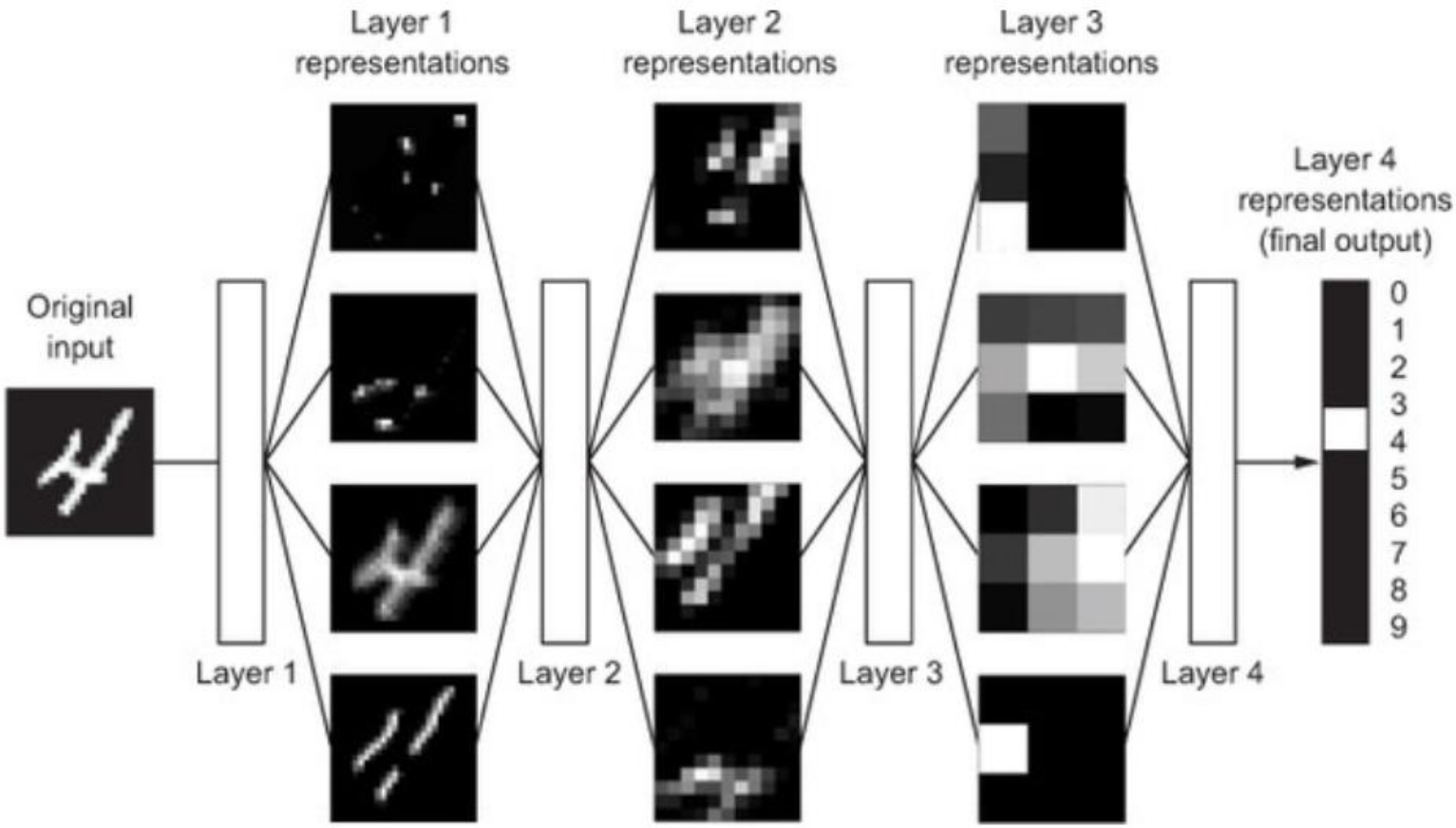
# *Convolutional neural network* - Le caratteristiche

Le principali caratteristiche dell'operazione di **convoluzione** sono:

- riesce ad identificare **pattern locali**;
- i *pattern* identificati sono **invarianti per traslazione**;
- apprendono **gerarchie spaziali** di *pattern*.



# *Convolutional neural network* - Le caratteristiche



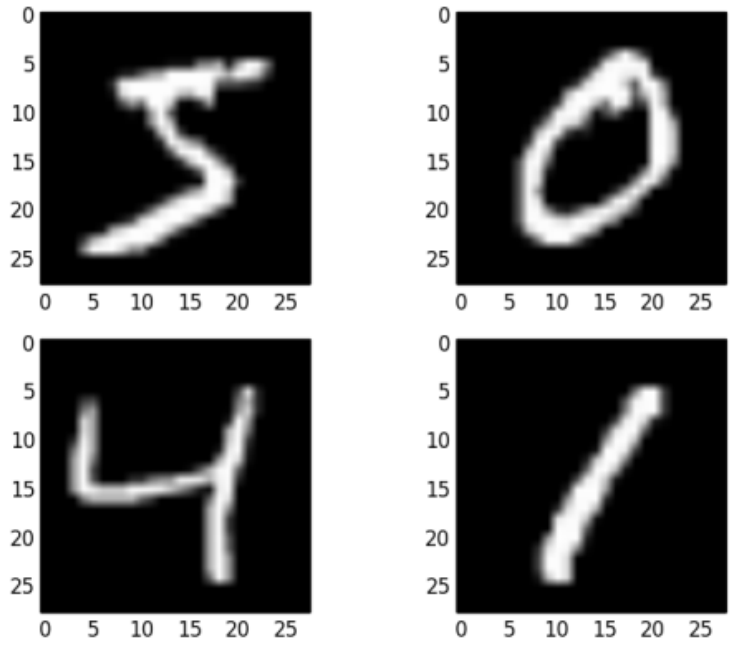

## La "distillazione" dell'informazione

La rete trasforma l'immagine in **rappresentazioni** che sono progressivamente differenti dall'immagine originale e progressivamente informative ai fini del risultato finale.

# Caso di studio: classificazione di immagini

Verrà effettuato un confronto di modelli utilizzando il dataset *MNIST*, una raccolta di immagini rappresentanti cifre scritte a mano.

- Lo scopo è classificare la cifra rappresentata (classificazione multinomiale a 10 classi).
- L'insieme di stima è composto da 21 000 osservazioni (immagini).
- Verrà calcolata l'accuratezza nell'insieme di verifica, composto da 10 500 osservazioni.



# Risultati della classificazione di immagini

| **Modello** | **Accuratezza** |
|---|---|
| Convolutional neural network | 99.0 % |
| Deep neural network | 98.3 % |
| Support vector machine | 97.6 % |
| Gradient boosting | 97.1 % |
| K-nearest neighbours | 96.3 % |
| Random forest | 96.1 % |
| Albero di classificazione | 87.6 % |

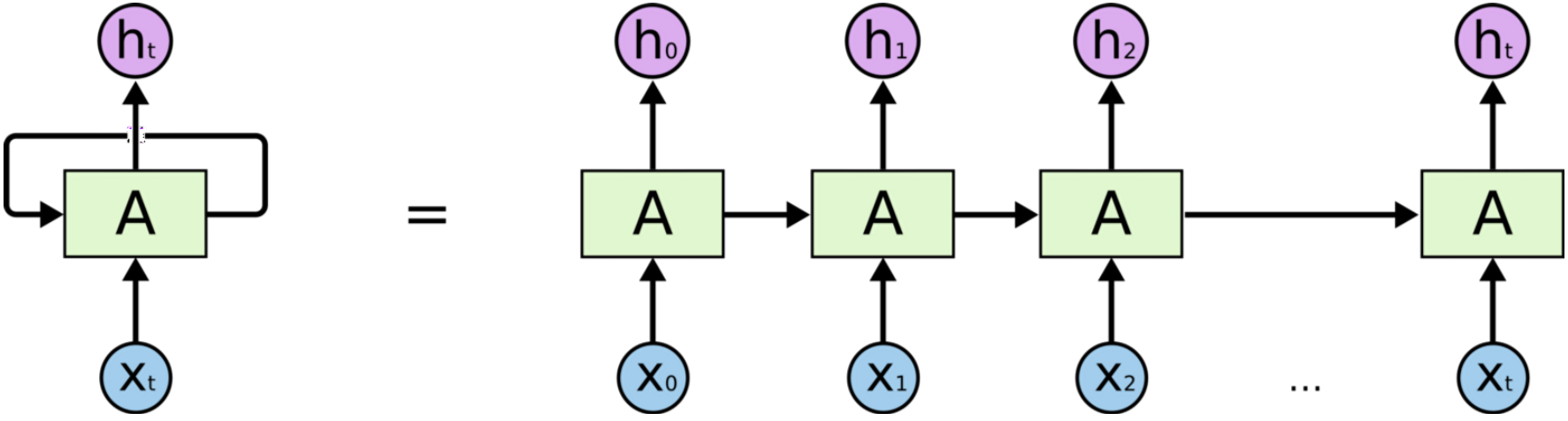
# Indice

# *Recurrent neural network* - La struttura della rete

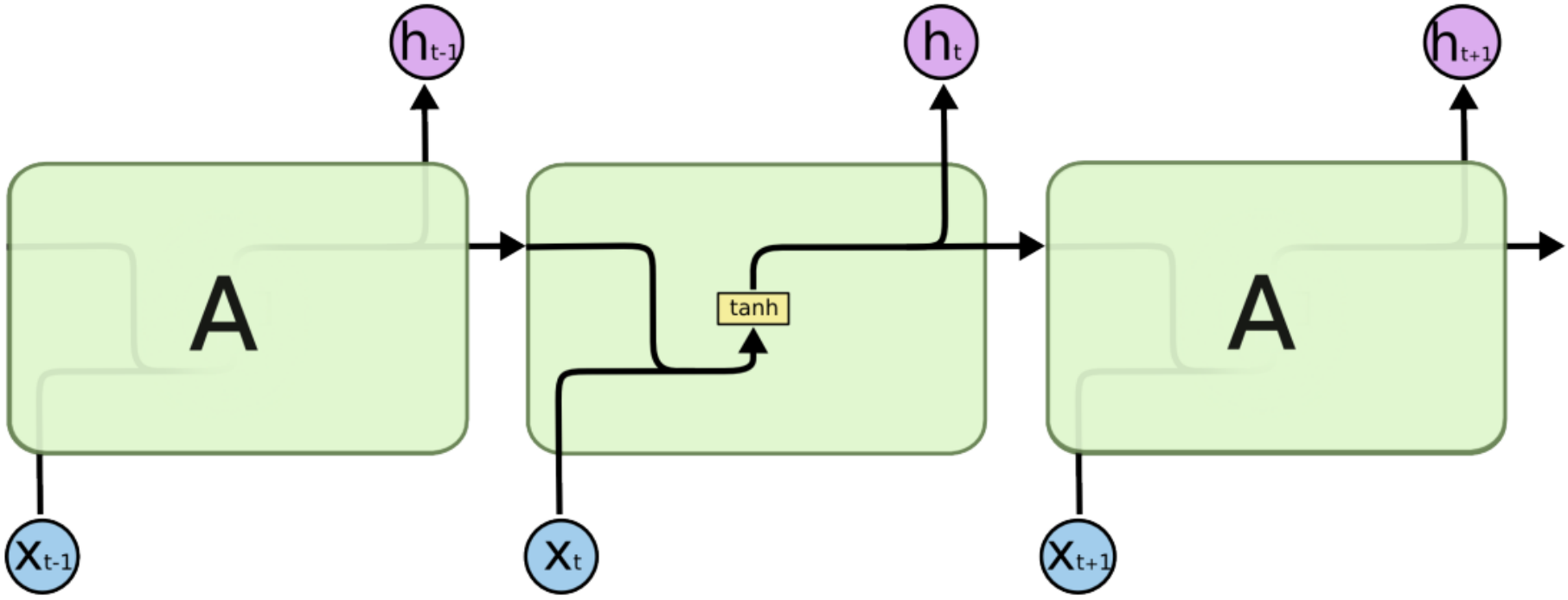
Una *recurrent neural network* (RNN) è un tipo di rete neurale, che funziona in modo ottimale nell'analisi di dati sequenziali, come serie storiche o dati testuali.

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



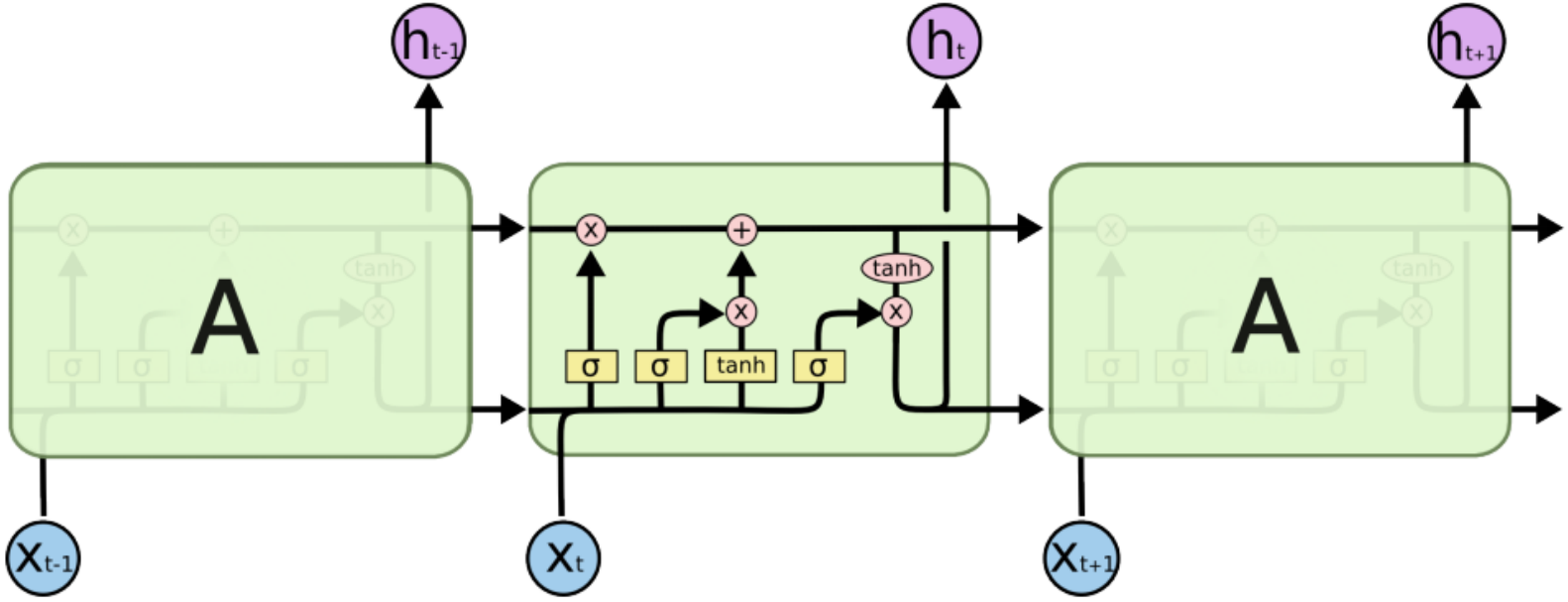

Al vettore in input al tempo $t$ viene applicata la *funzione di attivazione* (generalmente la *tangente iperbolica*), ottenendo come vettore di output l'**hidden state** al tempo $t$ ($h_t$). Questo vettore viene concatenato a $x_{t+1}$ e usato come input dello strato latente al tempo $t + 1$.

# Architetture alternative alle *recurrent neural networks*

## Problema

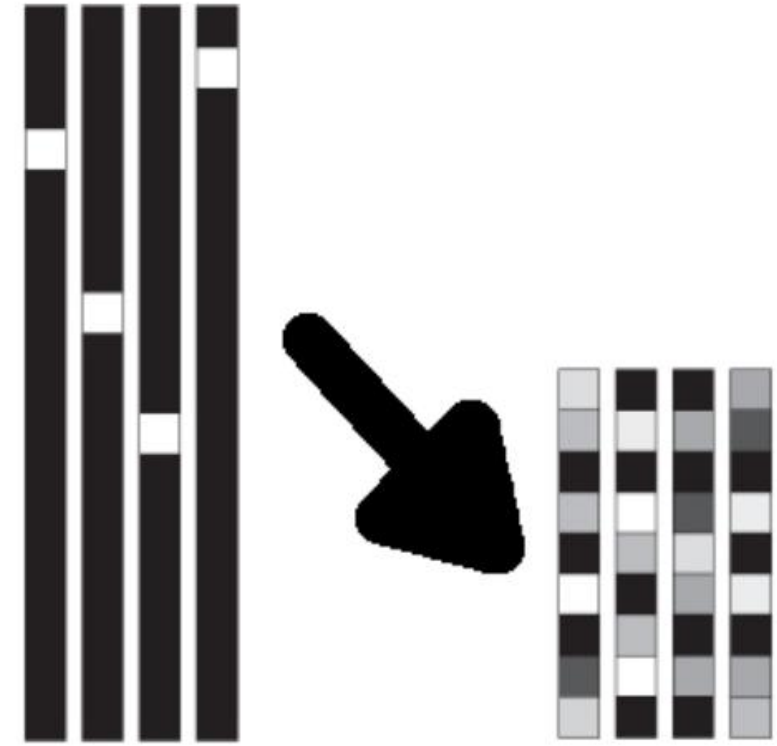
- L'utilizzo di strati ricorrenti con una finestra temporale (insieme di strati latenti passati connessi con lo strato latente corrente) ampia fa esplodere il numero di parametri da stimare;
- le informazioni importanti faticano a propagarsi a lungo nel tempo.

## Una soluzione: *Long-Short Term Memory* (LSTM)



# Lo strato di *embedding*

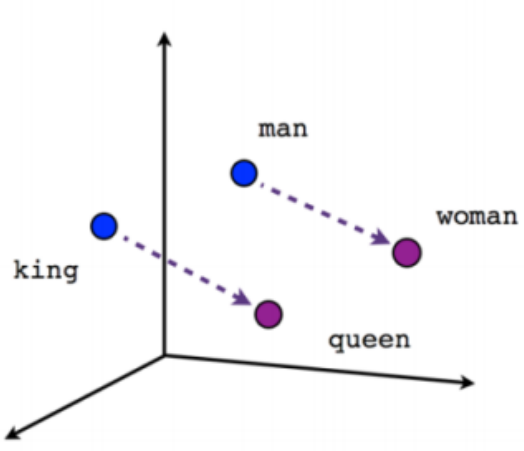
Se i dati sono **testuali**, come primo strato di una RNN viene spesso inserito uno **strato di embedding**



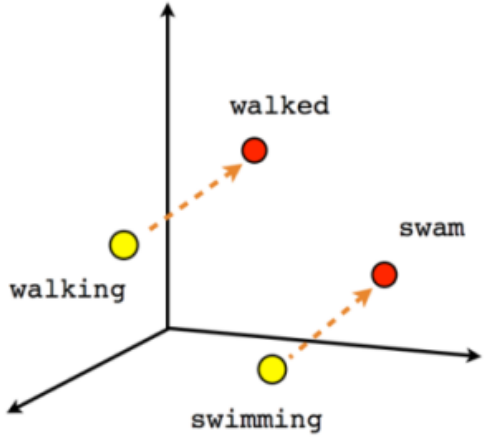

- comprime la rappresentazione dicotomica delle parole in uno spazio dimensionale inferiore
- gli *embedding* sono vettori densi, ossia ogni loro elemento è diverso da zero
- sono ottenuti nella fase di *training* e si adattano ai dati

# *Word embedding*

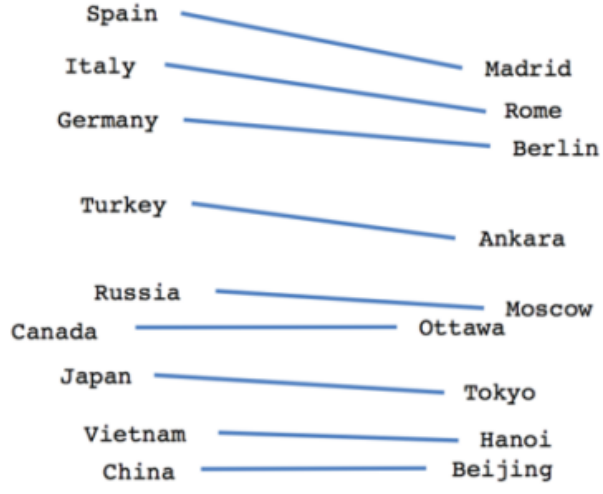
La rappresentazione vettoriale delle parole (*word embedding*) mantiene una coerenza semantica. Essendo i vettori composti da numeri, utilizzando i *word embedding* si possono confrontare due o più parole in modo quantitativo.




Male-Female



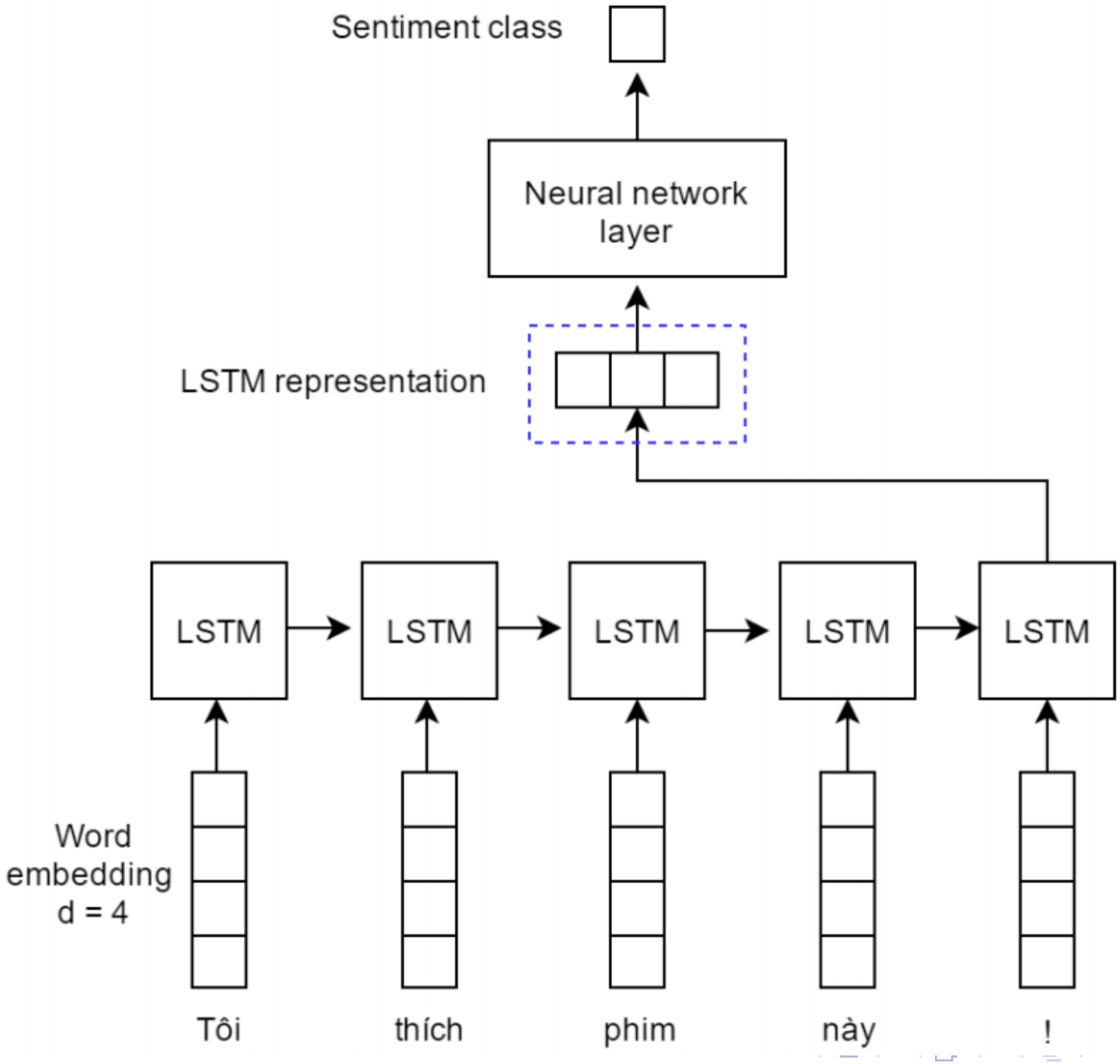

Verb tense




Country-Capital

Sono disponibili online, inoltre, i vettori di word embedding *pre-trainati*, ovvero che sono già stati calcolati precedentemente e che si possono utilizzare come punto di partenza.

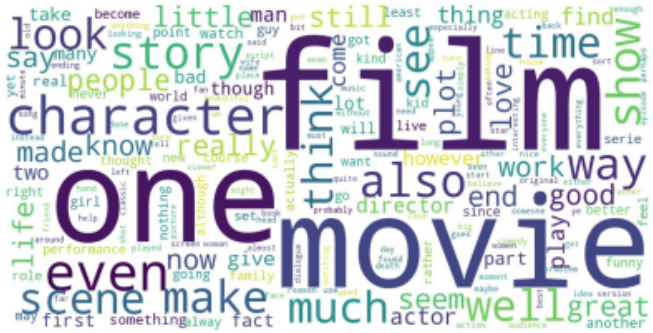
# Un esempio di rete neurale per la *sentiment analysis*



# Caso di studio: *sentiment analysis*

Verrà effettuato un confronto di modelli utilizzando il dataset *IMDB*, una raccolta di recensioni di film.

- Lo scopo è determinare se una recensione è *positiva* o *negativa* (classificazione binomiale).
- L'insieme di stima è composto da 25 000 osservazioni (recensioni).
- Ci sono 10 000 variabili esplicative e rappresentano le parole presenti nell'articolo; ogni variabile indica la presenza o assenza di una determinata parola. (eccetto *LSTM neural network*)
- Verrà calcolata l'accuratezza nell'insieme di verifica, composto da 25 000 osservazioni.



# Risultati della *sentiment analysis*

| **Modello** | **Accuratezza** |
|---|---|
| LSTM neural network | 88.0 % |
| Lasso | 87.0 % |
| Deep neural network | 86.5 % |
| Regressione logistica | 86.2 % |
| Random forest | 84.7 % |
| Bagging | 77.0 % |
| Adaboost | 72.5 % |
| Gradient boosting | 70.1 % |

# Indice

# 1) Non funziona così bene con pochi dati

Proviamo a ristimare i modelli dell'ultimo caso di studio, caricando solamente 5 000 delle 25 000 recensioni del training set di *IMDB*.

| **Modello** | **Accuratezza** |
|---|---|
| Lasso | 85.0 % |
| Regressione logistica | 85.0 % |
| Deep neural network | 83.5 % |
| LSTM neural network | 83.4 % |
| Random forest | 83.3 % |
| Bagging | 75.2 % |
| Adaboost | 70.8 % |
| Gradient boosting | 69.4 % |

# 2) Nella pratica è difficile e costoso

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

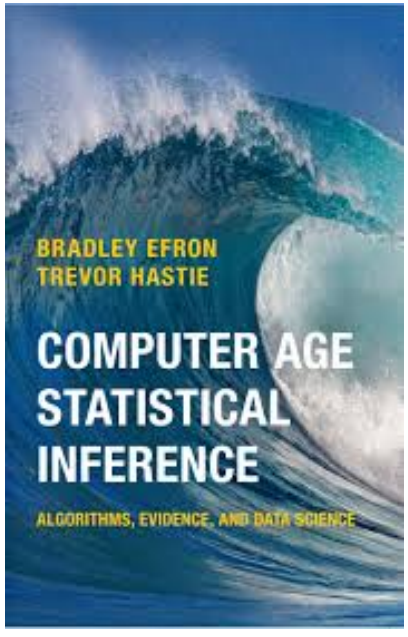
- ogni volta che si vuole affrontare un problema con il *deep learning*, una grossa parte del tempo viene utilizza per la fase di "ricerca";
- implementare una rete neurale è molto complesso e richiede solide basi di programmazione;
- effettuare il *training* di una rete neurale spesso può portare via 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.

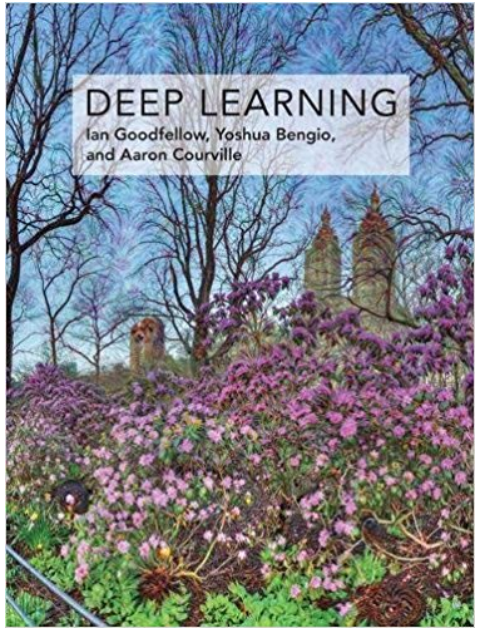
# 3) Non è interpretabile

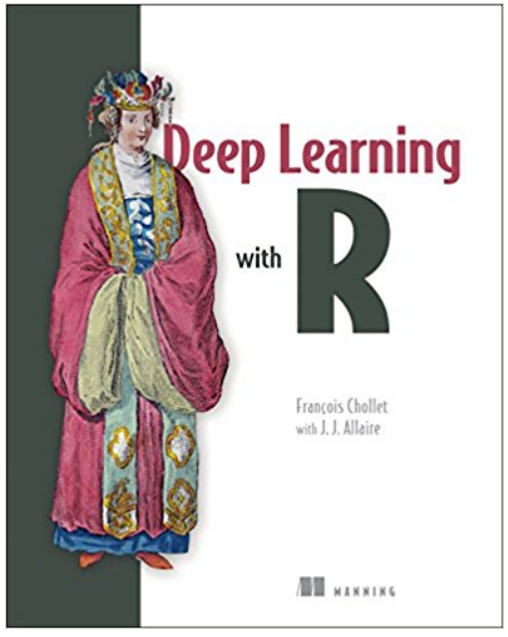
Le *deep neural networks* sono delle vere e proprie **black box**:

- 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;

# Riferimenti utili



# 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`.

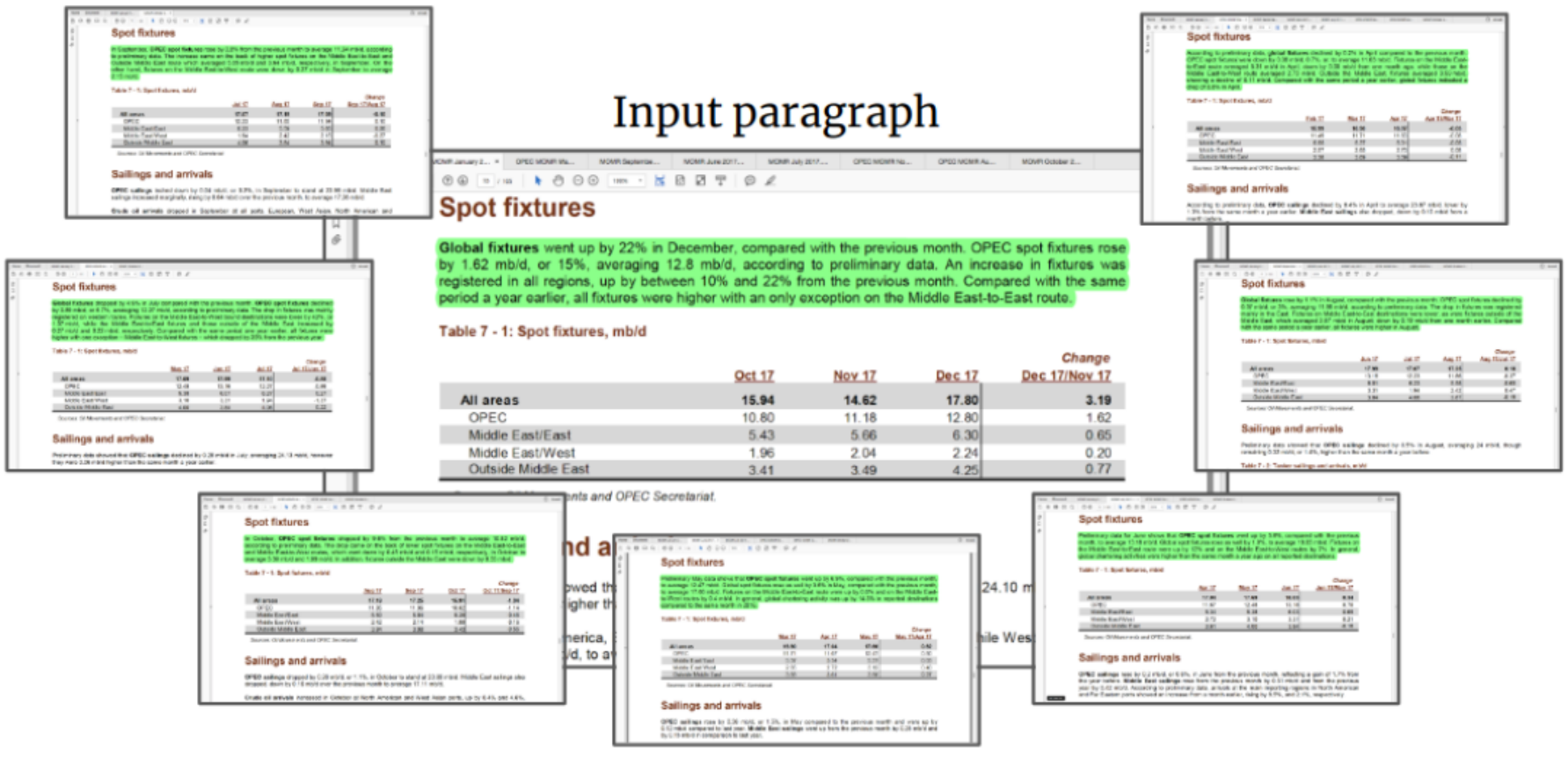
Consiglio!

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

# Qualche caso d'uso reale...

# Segmentazione di immagini

# Clustering di frasi



# Sistema di raccomandazione musicale