AI and Machine Learning

A quick introduction …

Methods in experimental particle physics - 17.01.2019

S. Giagu

References and further reading …

Machine Learning and Deep Learning (introduttivi non specialistici):

- Machine Learning Yearning: Andrew Ng (introduttivo, in preparazione alcuni capitoli già disponibili)
- Programming Collective Intelligence: T. Seagan, O'Reilly (applicativo, richiede conoscenza python)
- Pattern Classification: R.O. Duda, P.R. Hart, D.G. Stork, (2nd ed.) J.Wiley&Sons (completo con esercizi)
- Stat. Pattern Recognition: A. Webb, (3rd ed.), J.Wiley&Sons (molto chiaro e completo)
- Pattern Recognition and Machine Learning: C.M. Bishop (disponibile su web free)
- Deep Learning: I.Goodfellow, Y.Bengio, A.Courville, The MIT Press

Artificial Intelligence (introduttivi):

- Artificial Intelligence: A Modern Approach: P.Norvig. (disponibile su web free)
- Life 3.0 – Being Human in the Age of Artificial Intelligence: M. Tegmark (lettura serale)
- Fundamental Algorithms: 1 (Artificial Intelligence for Humans): J.Heaton (più avanzato)

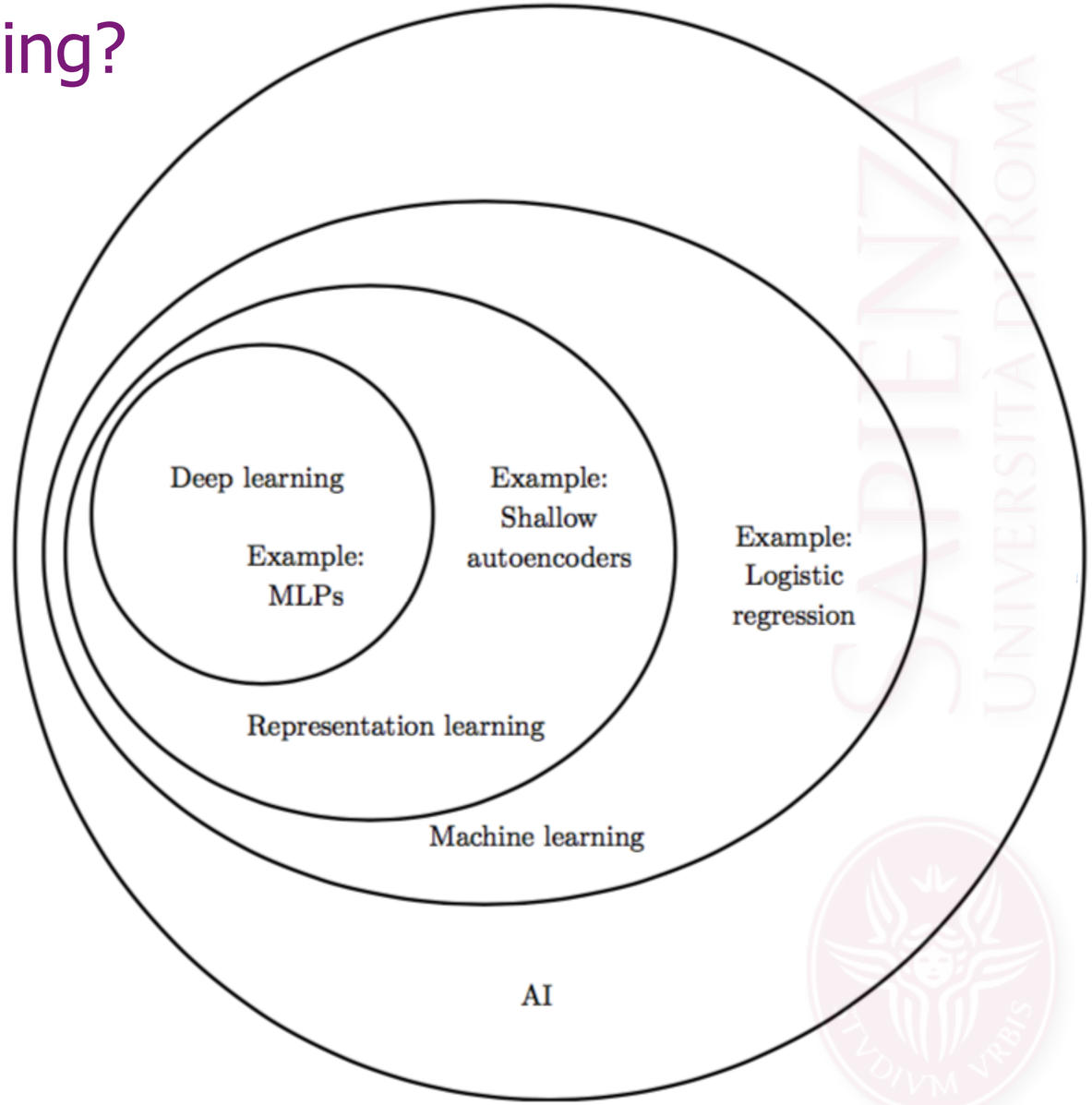
DeepNN tools/framework:

- PyTorch: https://pytorch.org
- Keras & TensorFlow: https://keras.io, https://www.tensorflow.org

Introduction

Che cosa si intende con Machine Learning?

- Il ML è una sotto-branca della Intelligenza Artificiale (AI),
  un campo della ricerca legato al tentativo di automatizzare compiti intellettuali normalmente svolti dall'uomo



AI

- Il concetto di AI esiste dalla metà degli anni ’50 e lo studio e sviluppo degli algoritmi utilizzati nei sistemi di AI è in corso da più di 20 anni, ma è solo negli ultimi 10 che applicazioni di AI si stanno diffondendo esponenzialmente nella vita sociale al di fuori dell’ambito della ricerca e sviluppo

Accelerazione esponenziale dovuta a tre sviluppi paralleli:

- migliori algoritmi (Machine & Deep Learning)
- disponibilità di maggiore potenza di calcolo (GPUs/TPUs/HPCs)
- abilità del settore tecnologico industriale di registrare enormi quantità di dati e renderli accessibili in rete (grid, clouds)

Machine Learning

- Definizione originale (Arthur Samuel, 1959):

  metodi computazionali (algoritmi) in grado di emulare i tipici comportamenti umani, e del mondo animale in generale, di apprendimento basato sull'esperienza (cioè tramite esempi), senza essere esplicitamente programmati

- algoritmi quindi in grado di risolvere quella classe di problemi (esempio: riconoscimento di immagini, comprensione di una lingua parlata) che non possono essere descritti in modo semplice da un set di regole matematiche formali (equazioni), e quindi troppo complessi da risolvere per un algoritmo computazionale di tipo tradizionale

ML: definizione moderna (Mitchell, 1998)

- un algoritmo (programma di un computer) si dice che apprendere da esempi (esperienza (E)) rispetto ad un dato compito (task (T)) e ad una data misura delle prestazioni (P), quando le prestazioni nel compiere il compito T, misurate da P, migliorano con l'esperienza E stessa

- Task T: vengono tipicamente descritte in termini di come il sistema di ML dovrebbe processare l'esempio E
  - Tipiche task nel ML:
    - classificazione ($f:R^n \rightarrow \{1,\ldots,k\}$), regressione ($f:R^n \rightarrow R$), trascrizione (ex. OCR), traduzione automatica (conversione di sequenze di simboli), anomaly detection, synthesis/sampling (es. Generatori di immagini, eventi), de-noising, stima delle densità di probablità …

Definizione Moderna di ML [2]

- Misura delle prestazioni P: per valutare le prestazioni di un sistema di ML e per confrontarlo con altri algoritmi è necessario definire una misura quantitativa delle prestazioni in relazione al Task T
  - accuracy (frazione di esempi per i quali l'algoritmo fornisce l'output corretto), error rate, costo statistico, …
  - da valutare sempre su un campione statisticamente indipendente (test sample)

- Esempi/Esperienza E:
  - l'insieme di informazioni empiriche dal quale l'algoritmo impara
    - training set (collezione di esempi … i dati)
    - conoscenze a priori: invarianti, correlazioni, ...

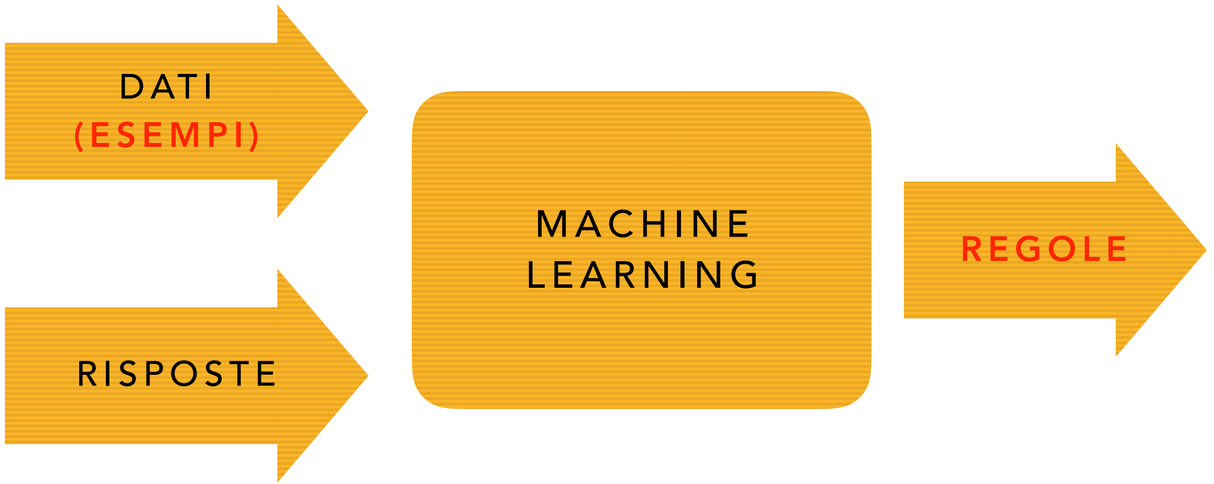
Machine Learning VS Programmazione Tradizionale

- programmazione tradizionale (AI simbolica): il programmatore (umano) carica un set di regole (il programma) nel processore e un set di dati che devono essere analizzati in accordo al set di regole per fornire la risposta al problema che si vuole risolvere

Il Machine Learning risolve il problema dell'apprendimento basato sull'esperienza attraverso un nuovo paradigma di programmazione

Machine Learning VS Programmazione Tradizionale

- ML: il programmatore fornisce al processore sia il set di dati che il set di risposte attese per quel set di dati, l'algoritmo produce un set di regole, che possono poi essere applicate a set indipendenti di dati per ottenere le risposte originali



un sistema per il ML viene "addestrato" più che programmato

- gli vengono presentati un certo numero di esempi significativi
- cerca strutture statistiche in tali esempi (assunzione: tali strutture devono ovviamente esistere)
- tali strutture statistiche permettono in ultima analisi al sistema di fornire delle regole per automatizzare il compito

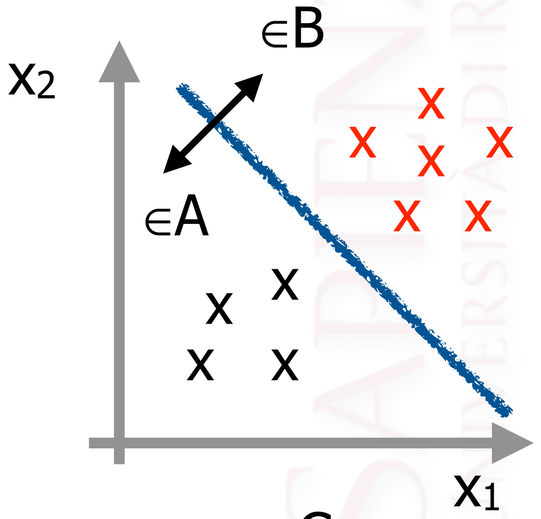
Paradigmi di apprendimento [1]

- Gli algoritmi di apprendimento possono essere suddivisi in diverse categorie che definiscono quale tipo di esperienza è permessa durante il processo di apprendimento
- categorie non formalmente definite, c'è un certo grado di sovrapposizione tra l'una e l'altra

- apprendimento supervised (i.e. esiste un insegnante):

  per ogni campione del training set è fornita la classe di appartenenza (label)

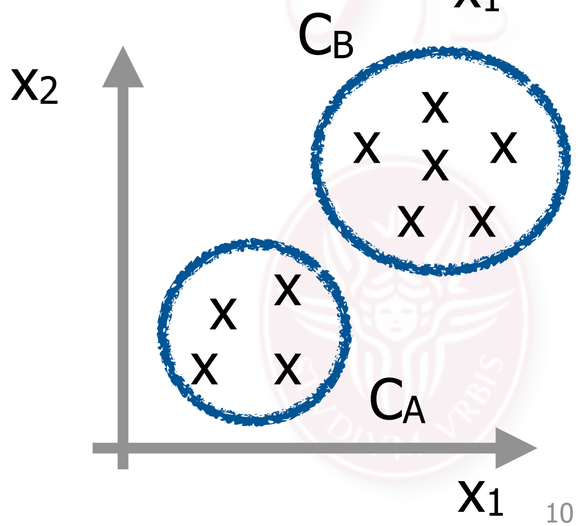
  obbiettivo dell'apprendimento: minimizzare gli errori o il costo di classificazione



- apprendimento unsupervised:

  non sono fornite esplicite informazioni sulla classe di appartenenza dei campioni del training set

  obbiettivo dell'apprendimento: formare dei raggruppamenti dei campioni, generalmente sulla base di similitudini, oppure apprendere l'intera distribuzione di probabilità che ha generato il dataset in input



Esempio di algoritmo unsupervised: Google News

Google News

Cerca argomenti, località e fonti

Notizie principali
Per te
Preferiti
Ricerche salvate
Italia
Dal mondo
Notizie locali
Affari
Scienza e tecnologia

Repubblica.it • oggi

Mostra altro

Anticorruzione, dal daspo per i corrotti allo stop alla prescrizione: cosa c'è nel ddl approvato dalla…

Il Fatto Quotidiano • oggi

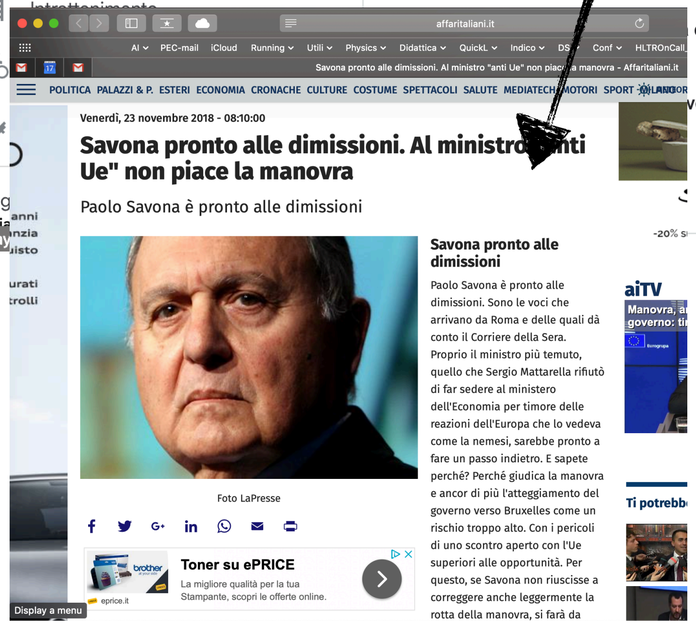
Savona pronto alle dimissioni. Al ministro "anti Ue" non piace la manovra

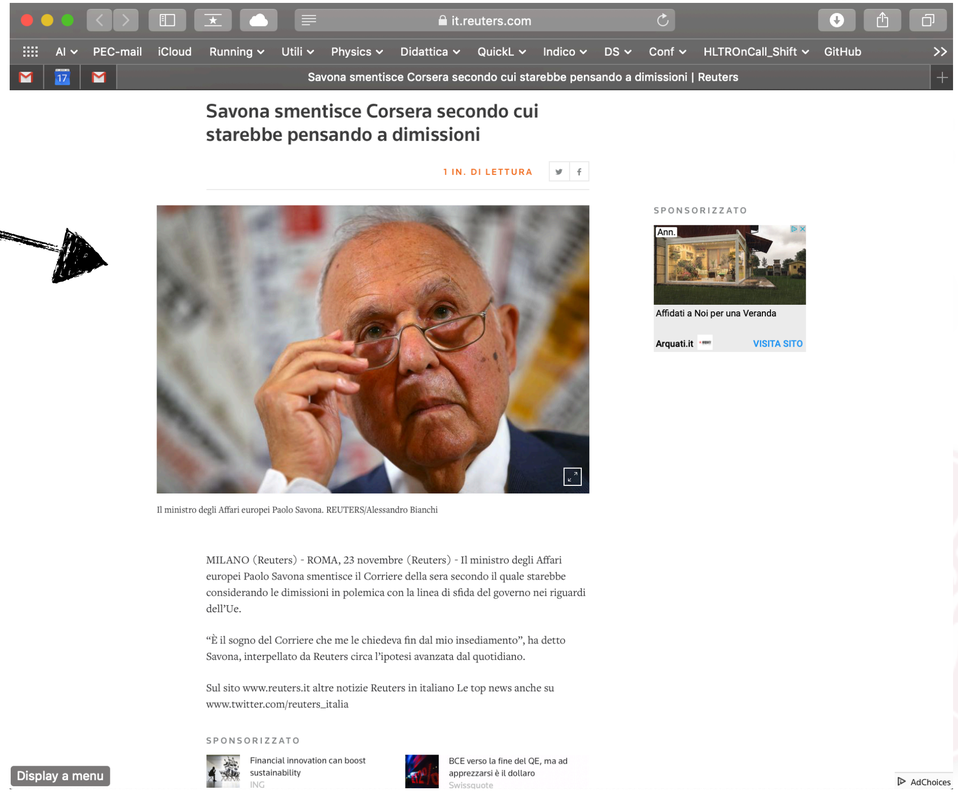
Affaritaliani.it • 3 ore fa

- La metamorfosi di Savona che adesso non esclude le dimissioni
  Corriere della Sera • oggi
- Savona smentisce Corsera secondo cui starebbe pensando a dimissioni
  Reuters Italia • 3 ore fa

…cambiato anche il governo"

…vona il vicolo cieco della maggioranza: Così non





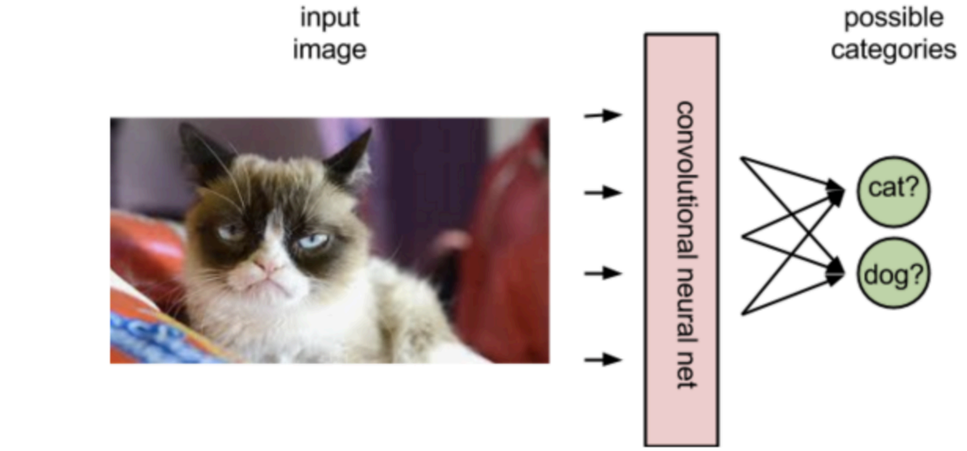
Paradigmi di apprendimento [2]

- Reinforcement learning:

  ispirato dalla psicologia comportamentale: non viene utilizzato un set di esempi fisso, ma l'algoritmo si adatta alle mutazioni dell'ambiente con cui interagisce attraverso un feedback continuo tra sistema e esempi e tramite la distribuzione di una sorta di ricompensa (rinforzo) che agisce su P

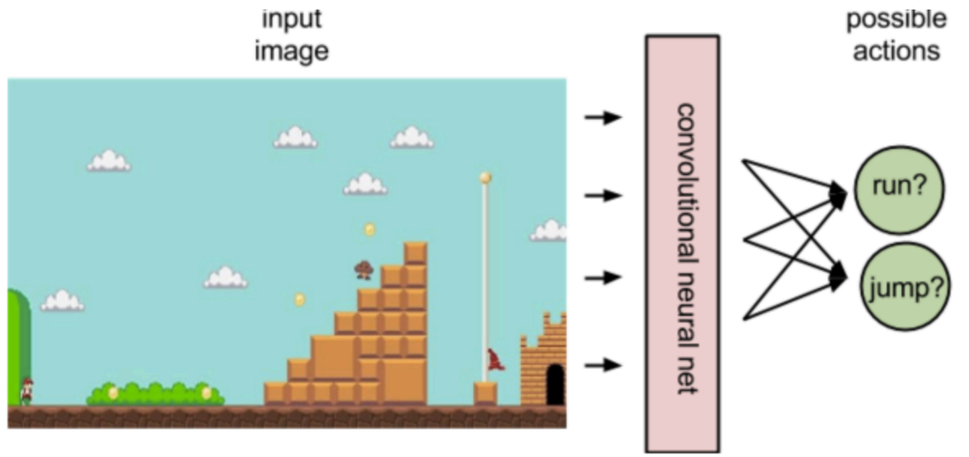
risolve il complicato problema di correlare azioni immediate con il ritardo nell'effetto che producono

esempio: massimizzare i punti vinti in un gioco che si sviluppa su molte mosse



ConvNet Supervisionato

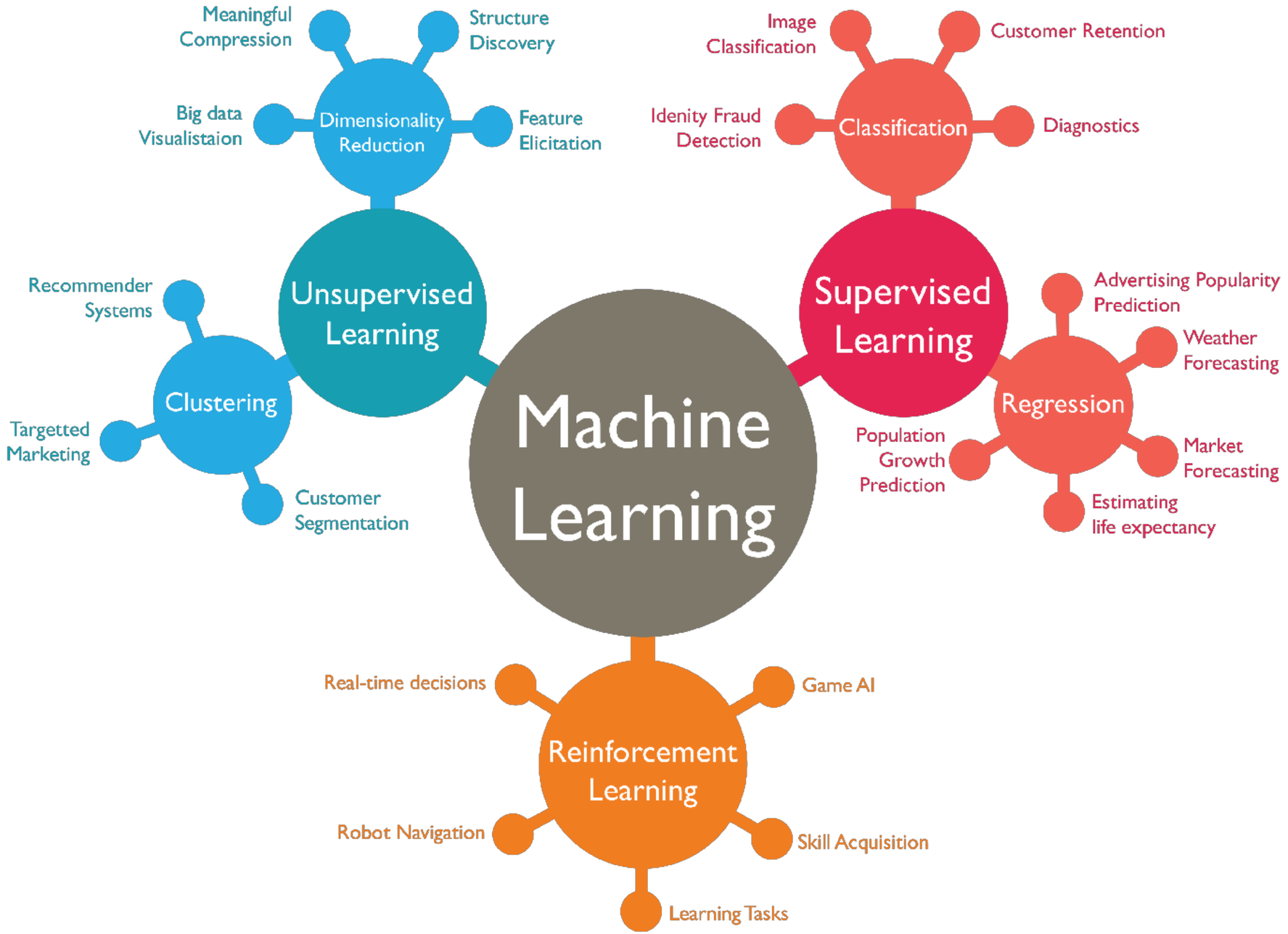
associa una label ad un'immagine



Agente Convoluzionale

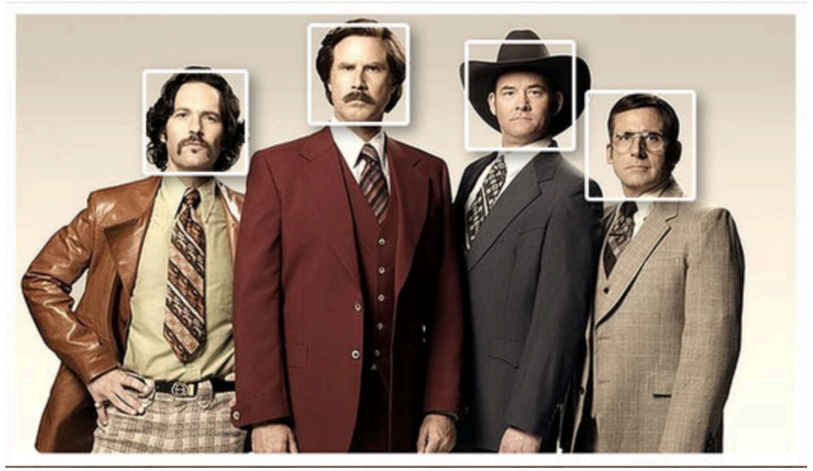
mappa uno stato nella migliore azione possibile

Tipi di apprendimento e task

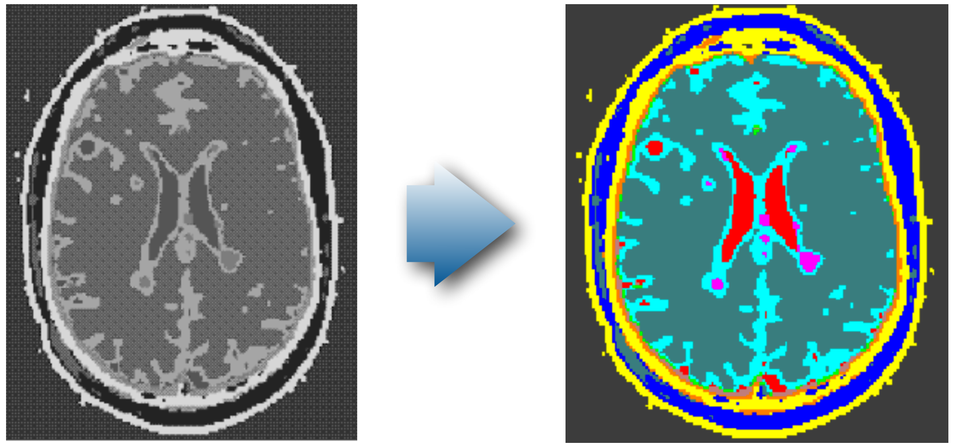


Esempi di applicazioni AI/ML

Face Detection:
- in tempo reale: macchine fotografiche
- su immagini: ex. foto su facebook
  - esperienza: porzioni di immagini
  - task: faccia o non-faccia



Medical Image Detection e Segmentation:
- esperienza: immagini (liste di pixel)
- task: identificare differenti tipi di tessuti biologici, componenti, disomogeneità …

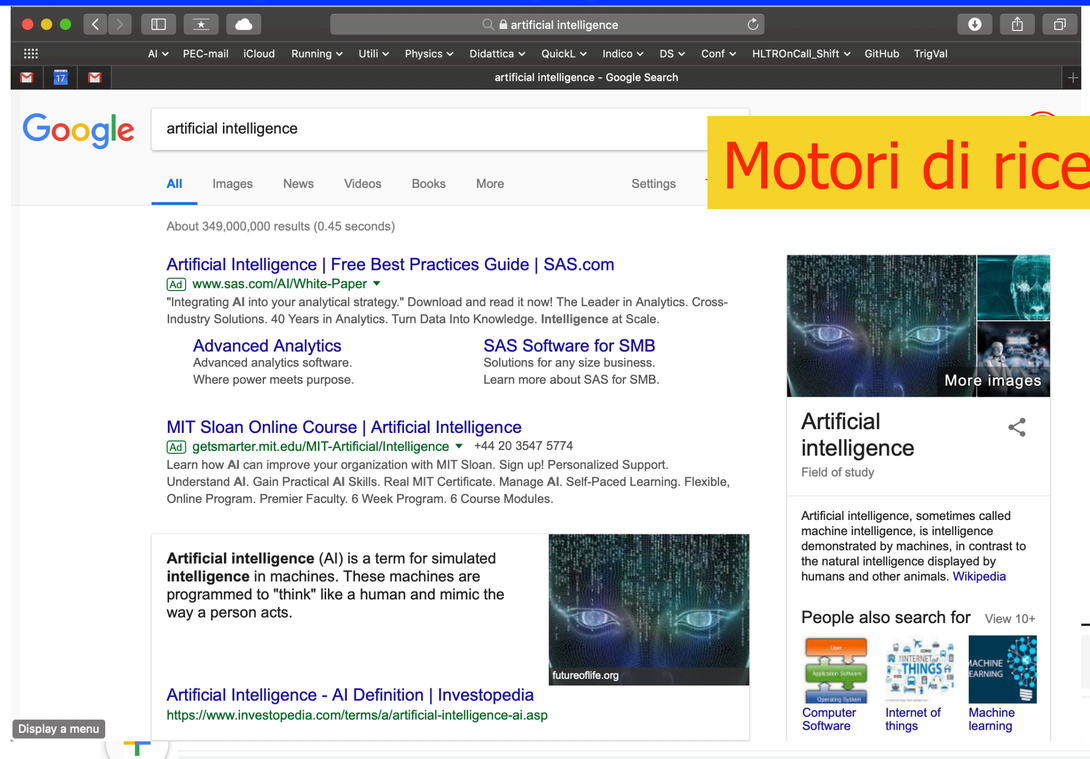


Riconoscimento della voce:
- esperienza: segnali acustici
- task: identificazione fonemi

ma-chin-le-ar-nin-g

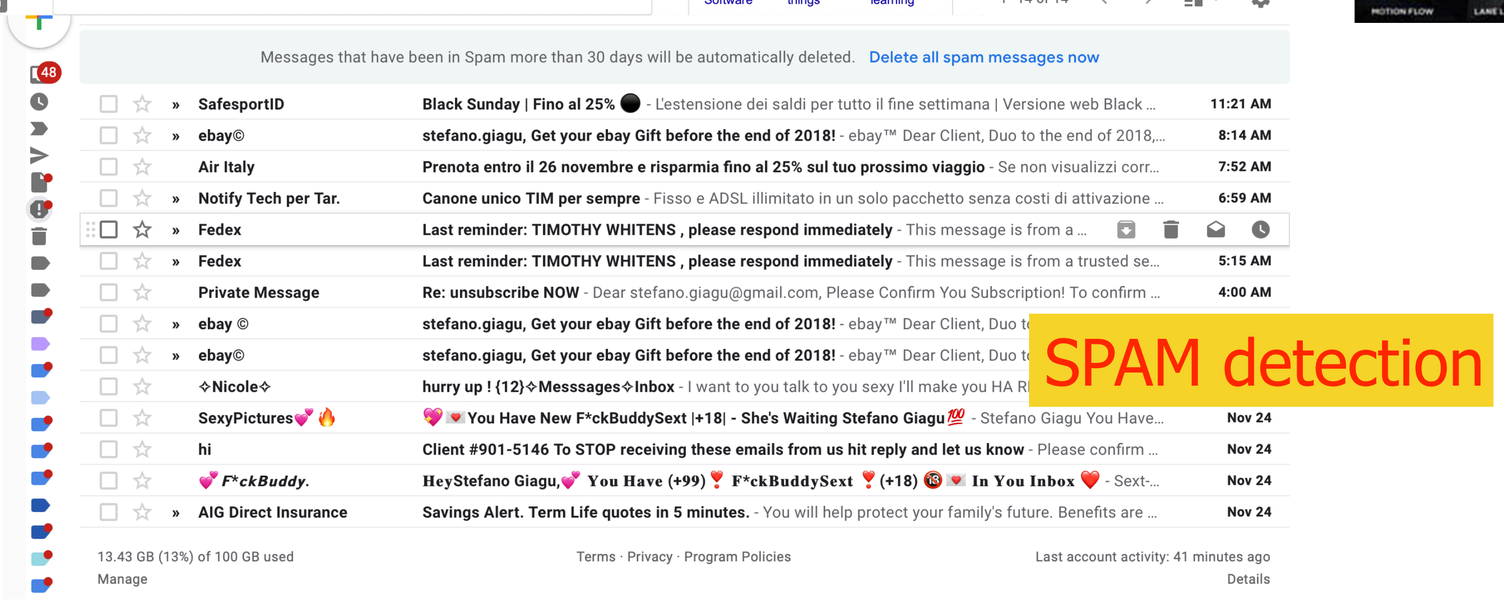
Esempi di applicazioni AI

Motori di ricerca web



Guida Autonoma

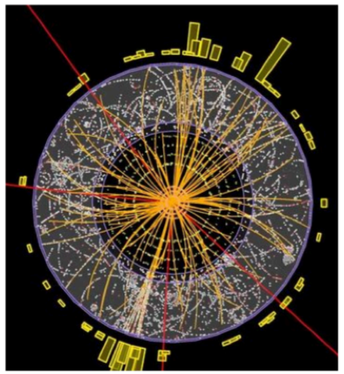
SPAM detection



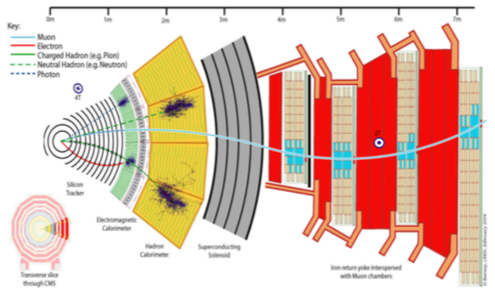
Droni Autonomi

Applicazioni in fisica (due esempi) ...

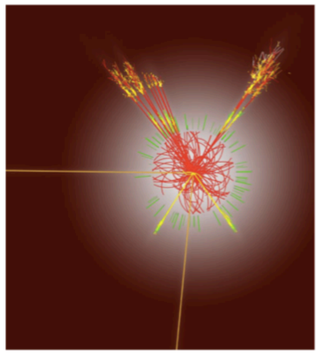
high energy physics



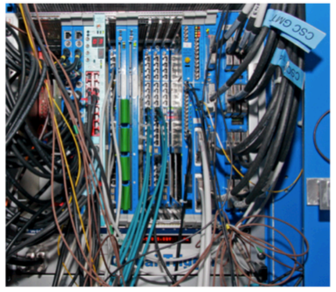
Deep Kalman
RNNs



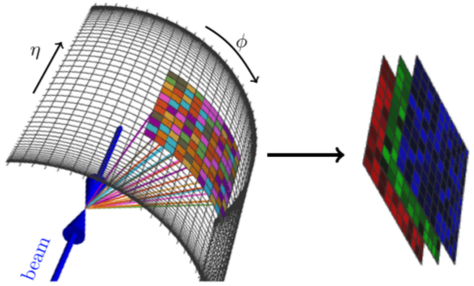
Generative Models,
Adversarial Networks



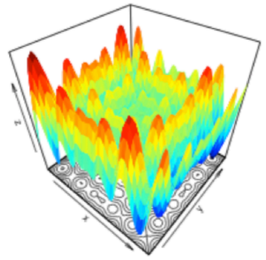
FCN, Recurrent,
LSTM NN



Deep ML +FPGA

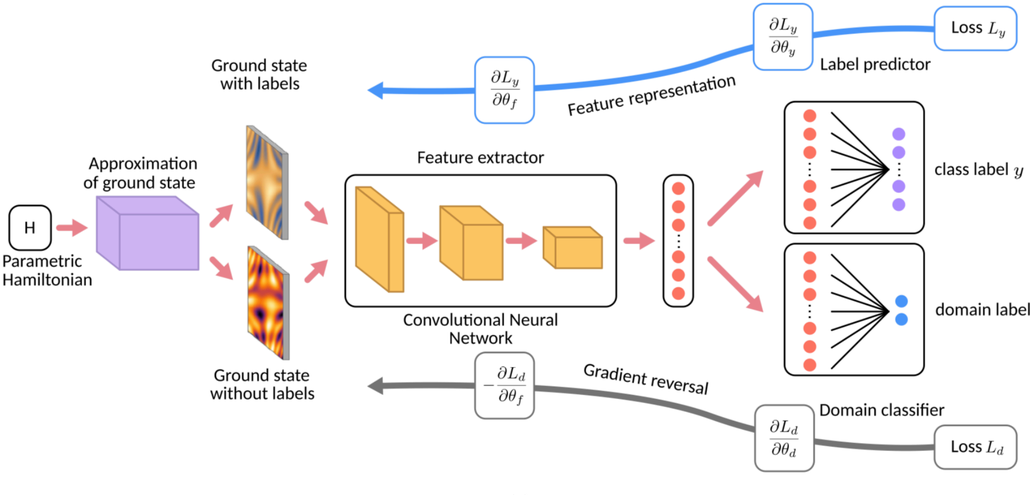


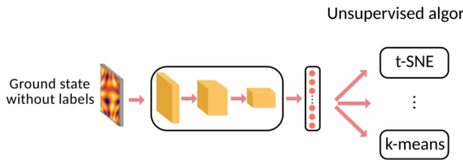
Convolutional DNN

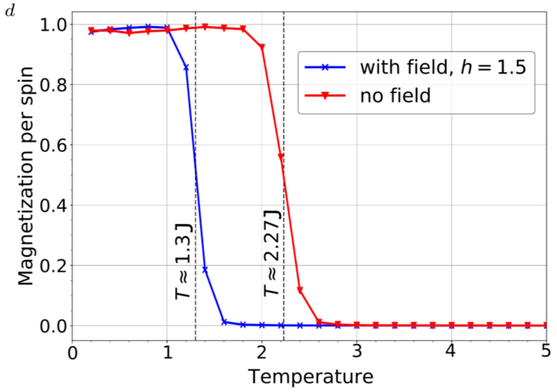


Multiobjective Regression

credits: D.Glayzer





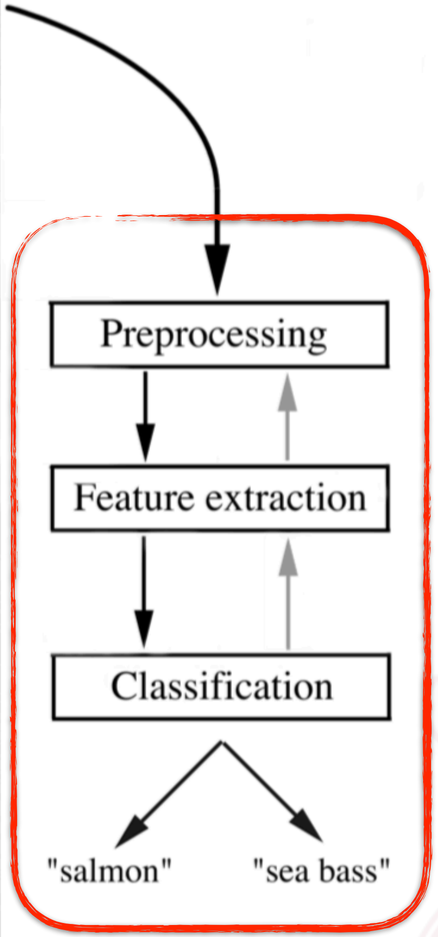


materia condensata

etc. etc...

Un esempio toy del più semplice sistema di Classificazione

esempio ittico da Duda et al.

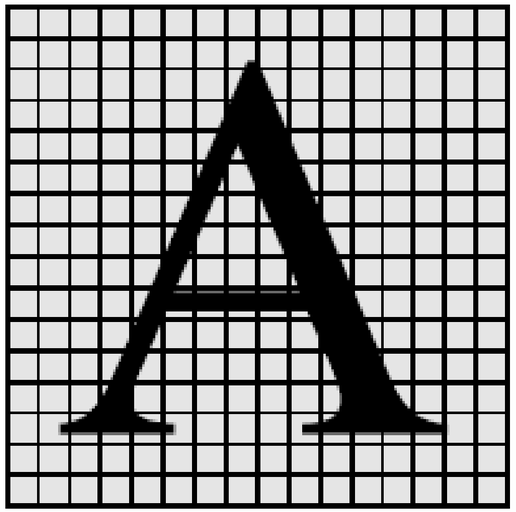


Algoritmo ML

Task:

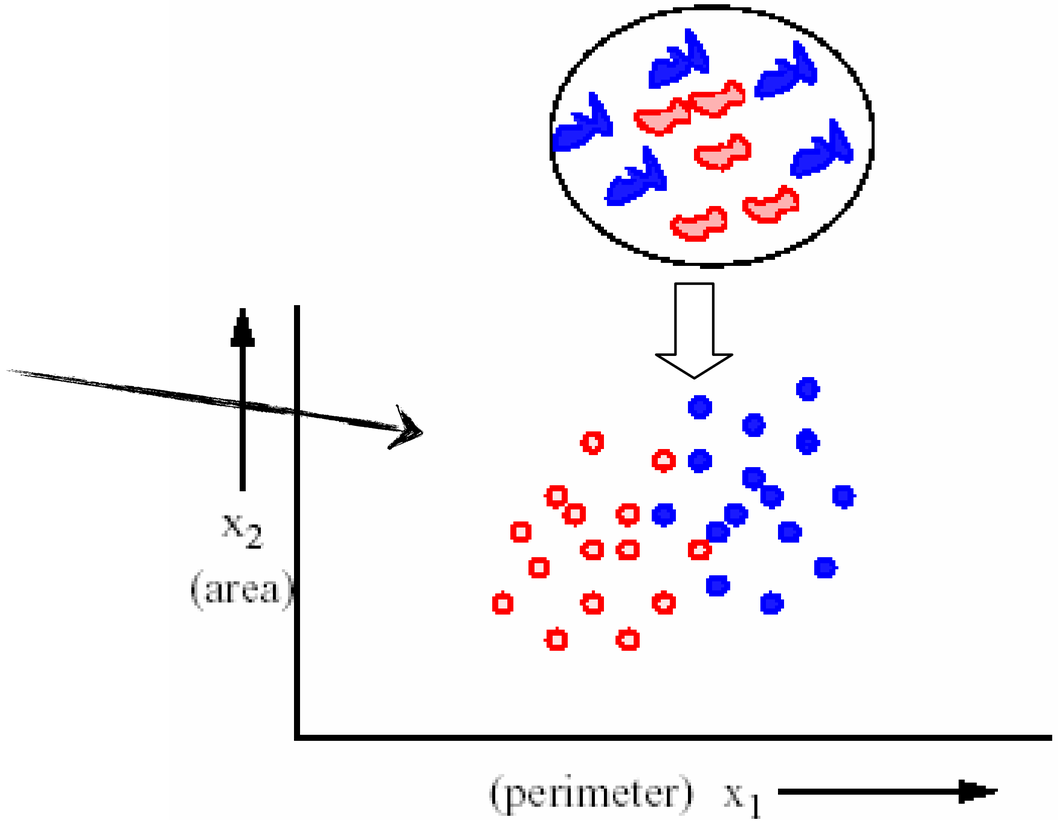
distinguere il tipo di pesce (salmone o spigola) che transita su un nastro trasportatore

Apprendere le Rappresentazioni dai dati

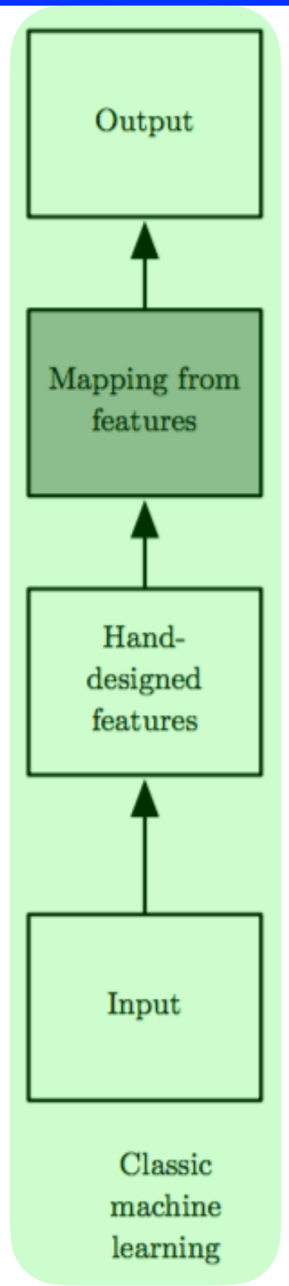


- 16x16=256 pixel rappresentano l'oggetto
- da questi si possono ricavare un insieme di misure sintetiche (tipicamente in numero minore) che descrivono l'oggetto (ex. rapporto tra pixel accesi/ totale, #cluster di pixel, area e/o perimetro del cluster, …): features

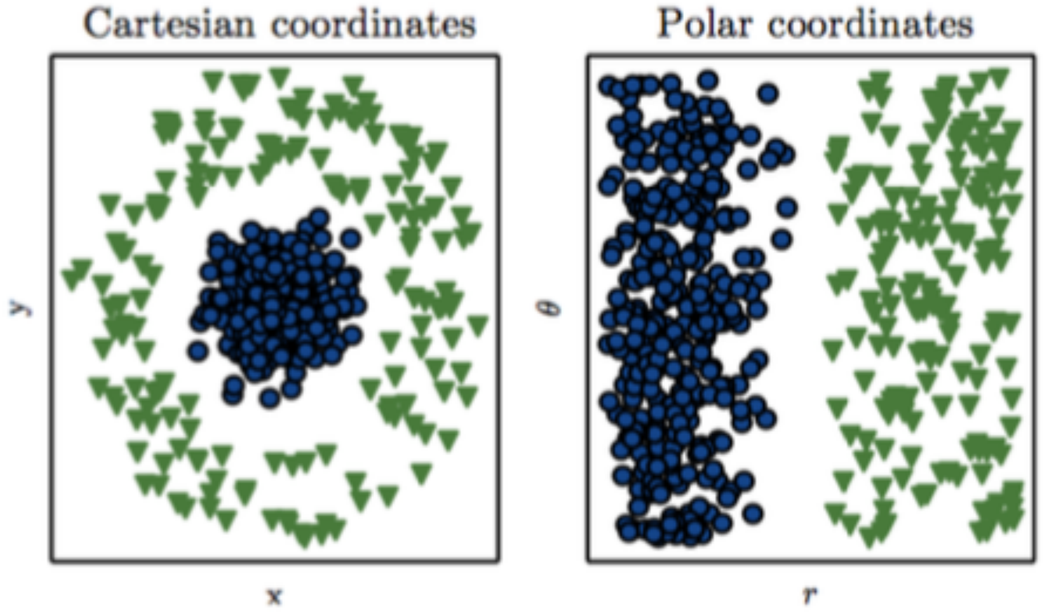
ad ogni oggetto corrisponde una descrizione in forma di vettore di features, che corrisponde ad un punto nello spazio delle features



Apprendere le Rappresentazioni dai dati



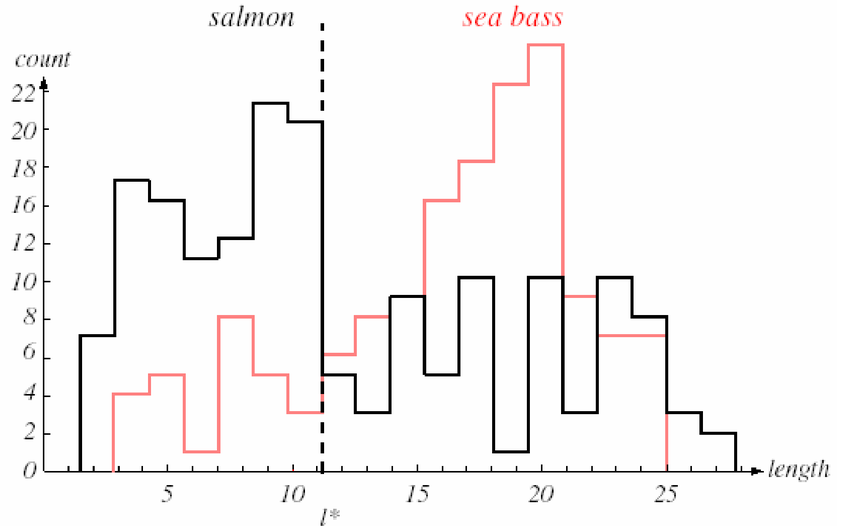
Nel ML di prima generazione l'insieme delle features era costruito e scelto dall'operatore sulla base del problema specifico e del bagaglio di conoscenze



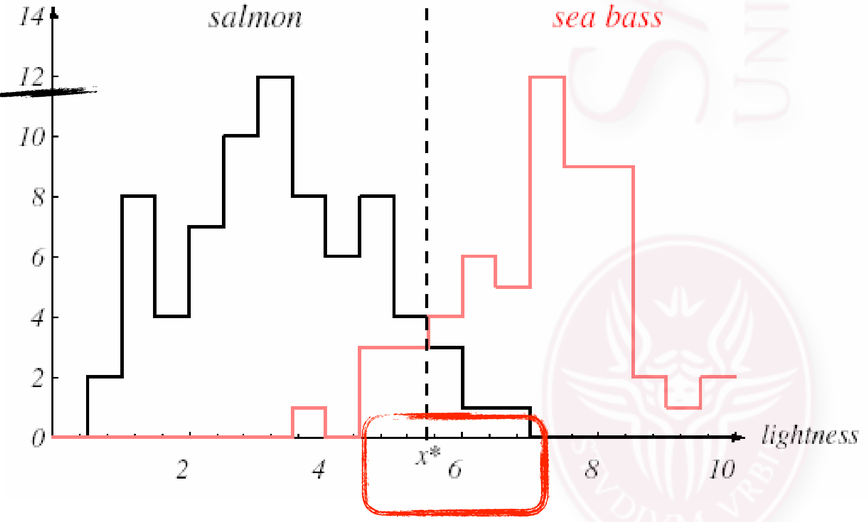
- uomo: identifica le feature
- algoritmo: identifica il mapping tra rappresentazione e output
- in questo approccio le prestazioni dell'algoritmo di ML dipendono fortemente dalla bontà della rappresentazione dell'oggetto fornita dal set di feature scelto

scelta delle features

- supponiamo che sulla base di informazioni ottenute sul problema si scelgano come feature la lunghezza e la luminosità (dell'immagine) del pesce
- quale delle due feature usare nella regola di classificazione?
- per valutare l'efficacia delle diverse scelte è necessario considerare un insieme di campioni rappresentativi dei due tipi di pesce, su cui effettuare la misura e fare le valutazioni (training set)



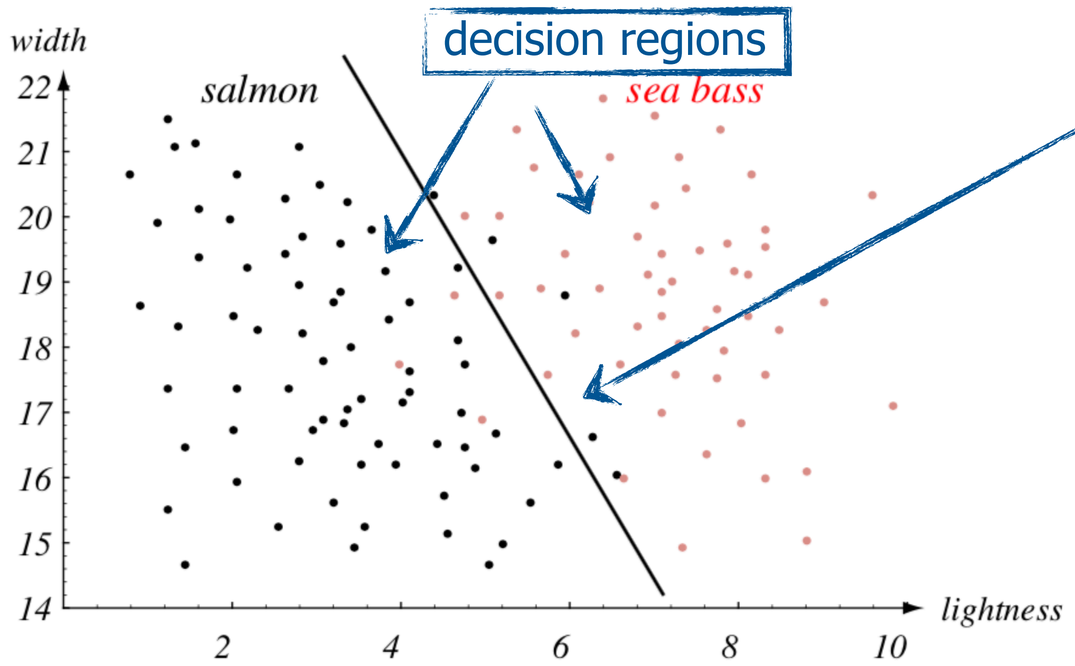
Regola di classificazione: if x > x*: spigola
else: salmone



la soglia x* è scelta ad esempio in modo da minimizzare la probabilità di classificazione errata

Regioni di decisione

- la feature scelta (luminosità), sebbene migliorativa rispetto alla lunghezza, potrebbe non portare ad una prestazione soddisfacente o comunque ottimale
- per migliorare ulteriormente, la soluzione più efficace è quella di scegliere un insieme di più features ed usarle contemporaneamente
- il problema diviene quello di partizionare lo spazio delle features in regioni, ognuna delle quali ascrivibile ad una delle classi note



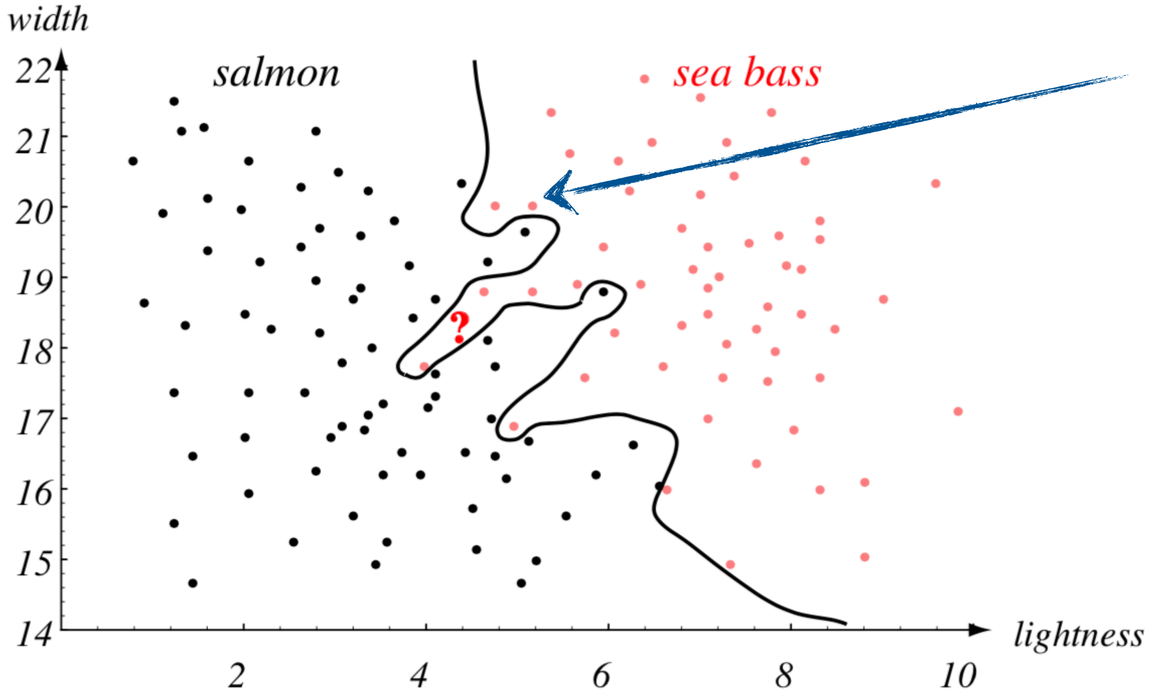
decision boundary o frontiera di decisione

- scelta più semplice: frontiera lineare
- i casi mal classificati sono inferiori rispetto al caso di una sola feature ma ancora presenti

Regola di classificazione: if $x_2 > w_0 + w_1 \cdot x_1$: spigola
else: salmone

Frontiere complesse …

- le regioni definite in base alla frontiera lineare implicano ancora degli errori
- domanda: è possibile eliminare del tutto gli errori con una frontiera meno semplice?



una frontiera di decisione più complessa che annulla gli errori sul training set (overfitting)

Regola di classificazione: if $x_2 > f(x_1;\mathbf{w})$: spigola
else: salmone

problema: in questo modo NON si garantisce una buona prestazione del sistema su campioni (non appartenenti al training set) che bisognerà classificare in fase operativa

- i campioni di training hanno necessariamente dimensione finita e la forma della frontiera dipenderà dalle fluttuazioni statistiche in tali campioni
- tale aspetto che prende il nome di capacità di generalizzazione del classificatore, costituisce il punto cruciale nella progettazione di qualsiasi algoritmo di machine learning

Teoria di Vapnik: minimizzazione del rischio strutturale

- in teoria statistica del ML si dimostra che vale la disuguaglianza (Vapnik):

errore sul campione di training

N: dimensione del campione di training

$$\text{Prob}\left(R(f) \le R_s(f) + \sqrt{\frac{h(\log(2N/h)+1) - \log(\eta/4)}{N}}\right) = 1-\eta$$

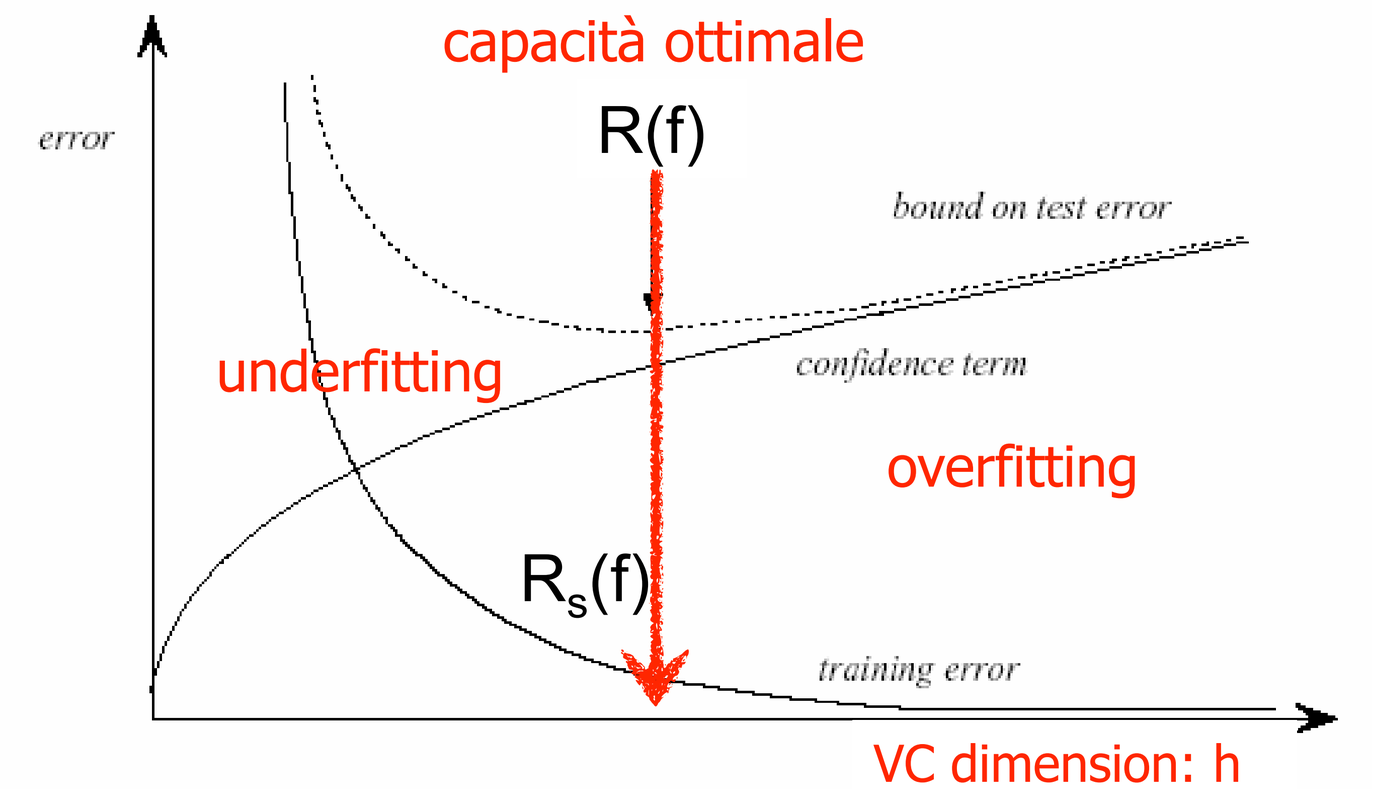
errore su un campione indipendente di test (quanto vogliamo realmente minimizzare per avere il classificatore ottimale)

termine di confidenza: non è altro che l'errore di generalizzazione

- h è un intero non negativo detto dimensione di Vapnik-Chervonenkis (VC-dimension) che misura la capacità (complessità, potere espressivo) del modello/algoritmo di ML utilizzato, maggiore h maggiore la capacità dell'algoritmo

Teoria di Vapnik: minimizzazione del rischio strutturale

bias-variance tradeoff: utilizzando un modello più complesso (i.e. h più grande) in grado di ridurre il bias, paghiamo la cosa con una varianza più grande …



1. Mantenere fisso il termine di confidenza (scegliendo una struttura appropriata del classificatore) e minimizzare il rischio empirico.
2. Mantenere fisso il rischio empirico (p.es. uguale a zero) e minimizzare il termine di confidenza.

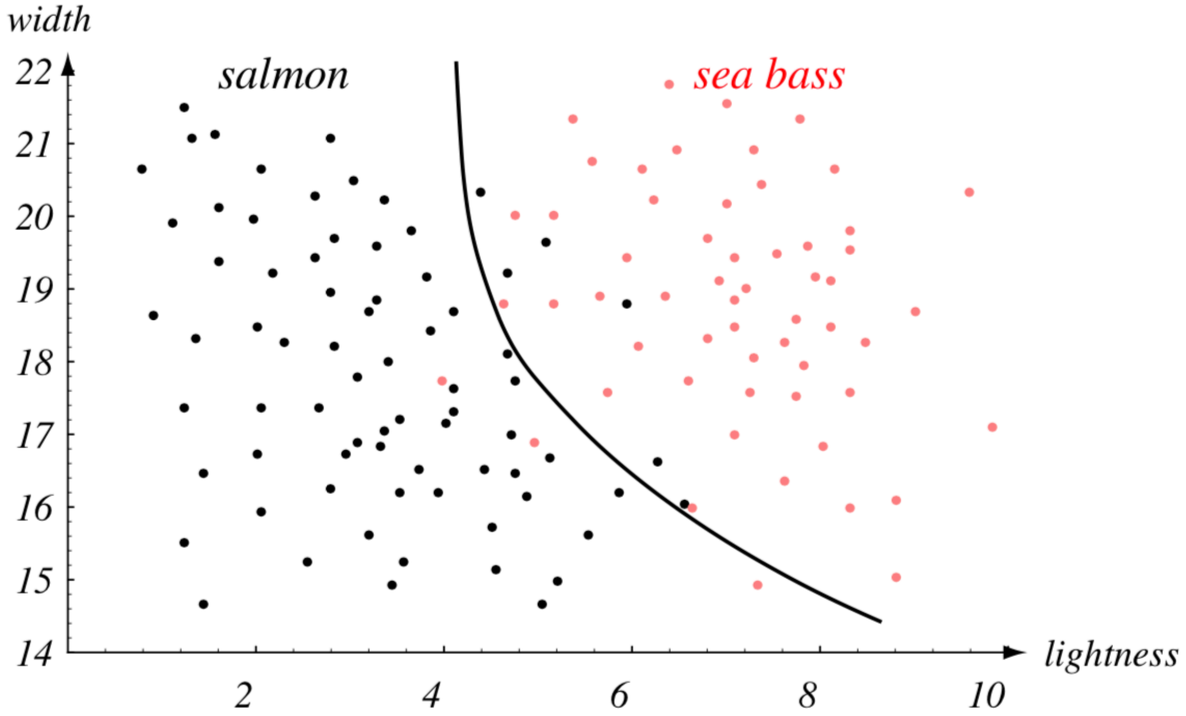
esempio NN

esempio SVM

… e problema della Generalizzazione

- Scelta della frontiera compromesso tra:
  - prestazioni del classificatore sul training-set
  - capacità di generalizzazione del classificatore
- è sempre preferibile tollerare un certo livello di errore sul training set se questo porta ad una migliore generalizzazione del classificatore

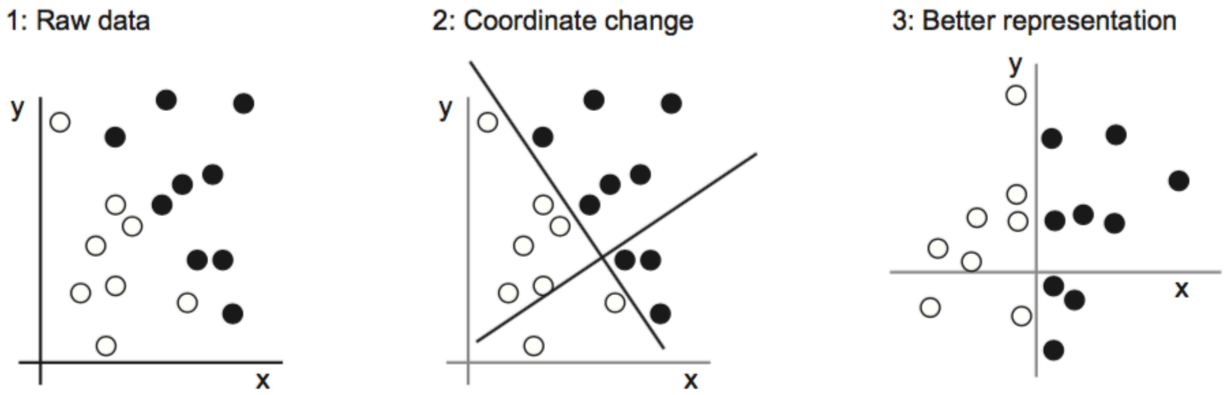
esempio: frontiera di decisione con buone prestazioni e buona capacità di generalizzazione



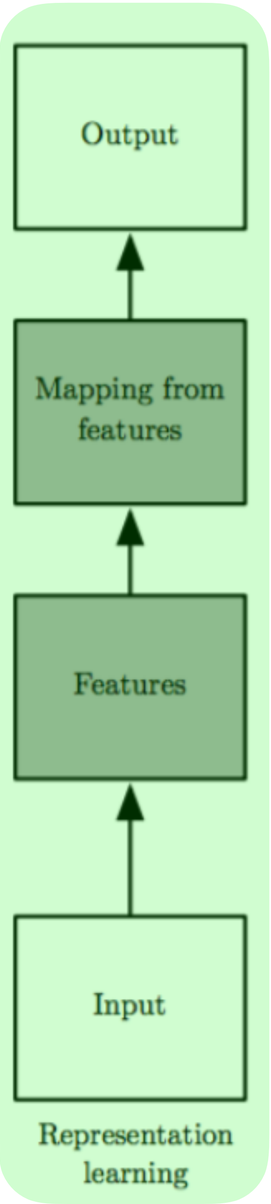
Representation Learning

- L'evoluzione più importante degli algoritmi di ML è stata l'introduzione del Representation Learning

- l'algoritmo identifica in modo automatico una migliore rappresentazione rispetto a quella fornita dalle feature in input all'algoritmo stesso

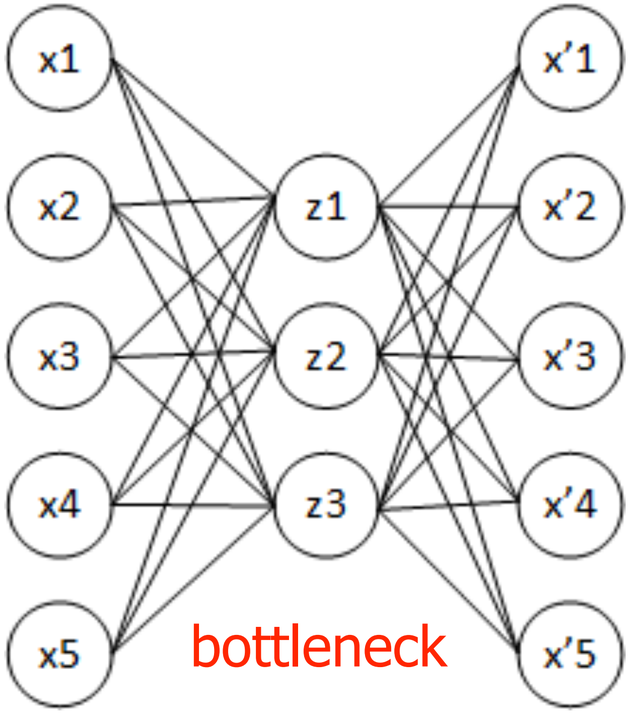


- ricerca all'interno di un insieme predefinito di operazioni chiamato "hypothesis space" usando come guida campioni di dati di esempio
- i diversi algoritmi di ML tradizionali corrispondono in pratica all'implementazione di differenti hypothesis space e di differenti tecniche di ricerca



Autoencoder: esempio di apprendimento di rappresentazioni

- apprendimento non-supervisionato in cui si cerca di scoprire caratteristiche comuni e generali nei dati di input
- combina un encoder che converte i dati di input in una rappresentazione differente, e un decoder che converte la nuova rappresentazione nel formato di input originale
- addestrato in modo da fornire in output la cosa più simile possibile all'input iniziale, in questo modo viene appresa la funzione identità



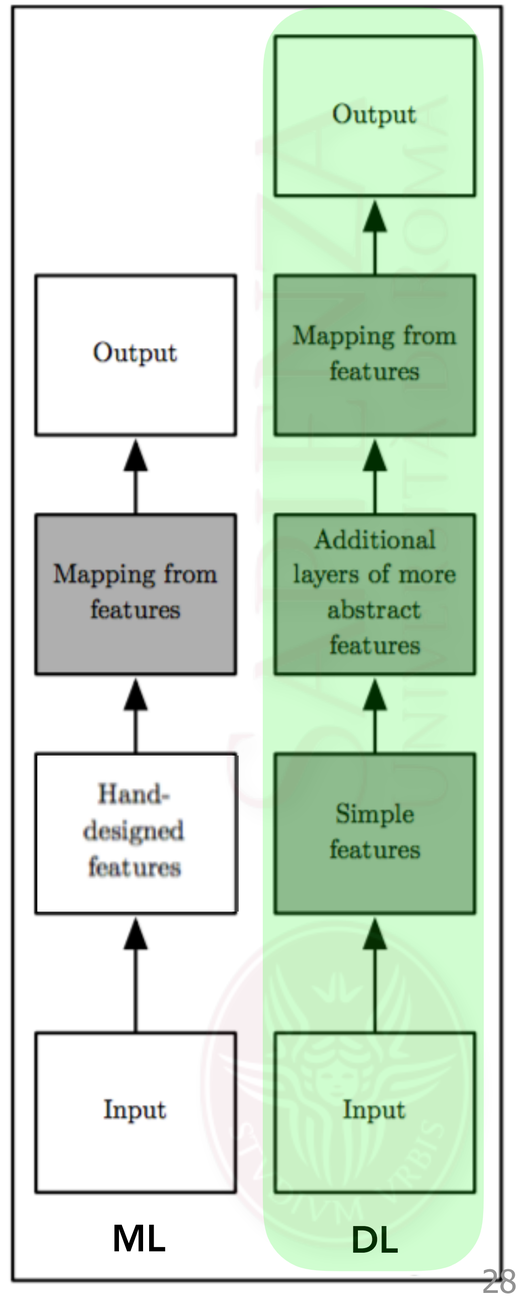
- "banale" a meno di non vincolare la rete ad avere un layer nascosto con un numero limitato di nodi (inferiore a quello dei nodi di input e output)
- in questo modo la rete costruisce (impara) rappresentazioni "compresse" delle feature di input

Deep Learning (DL)

- Gli algoritmi di ML tradizionali non sono "creativi" nel cercare le rappresentazioni, si limitano a cercarla tra quelle implementate nel hypothesis space

- Il Deep Learning risolve il problema organizzando idee e concetti in modo gerarchico e costruendo nuove rappresentazioni dei dati espresse in termini di altre rappresentazioni più semplici
  - esempio: il viso di una persona in una immagine fotografica è rappresentabile combinando caratteristiche più semplici: bordi, contorni, linee etc…
  - DL == representation learning gerarchico

estremamente potente ed efficace, ma richiede grandi campioni di dati esempi e grande potenza di calcolo

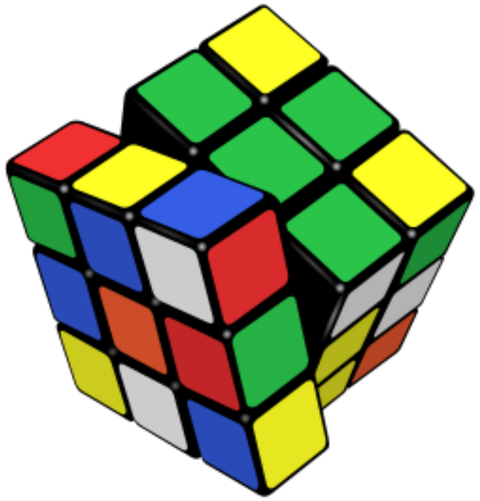


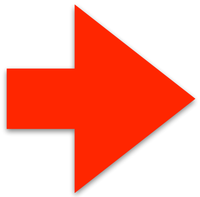
DL: interpretazione geometrico/intuitiva …

problema: classificare le faccette di colore rosso di un cubo di Rubik (classi: colori, dati: l'insieme delle 54 faccette colorate del cubo)

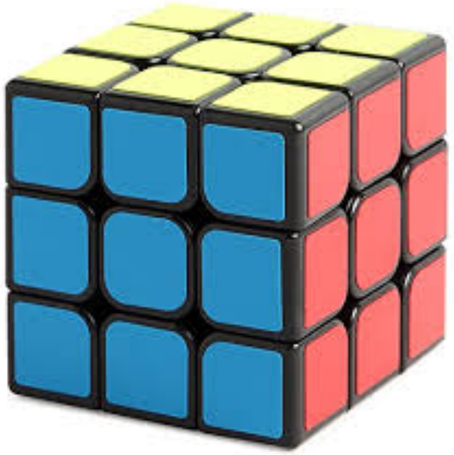
Il goal di qualunque algoritmo di rappresentation-learning è quello di trovare la trasformazione per cui la classificazione risulti la più semplice possibile

Nel DL questo viene implementato come una successione di semplici trasformazioni lineari (in questo caso rotazioni nello spazio tridimensionale), in modo molto simile a quello che faremo noi con le nostre mani per risolvere il cubo





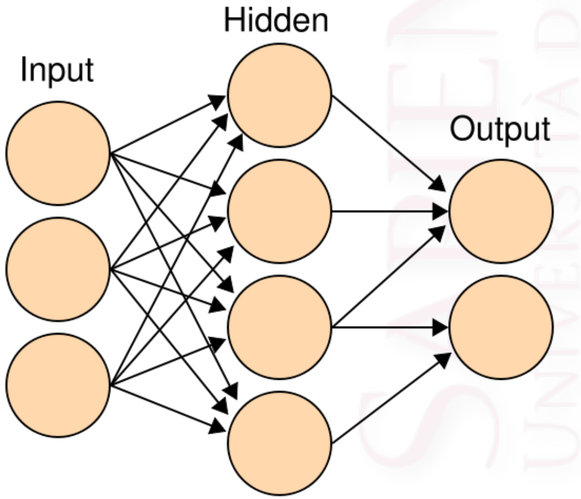
successione di rotazioni
$\in$ gruppo di Rubik



Artificial Neural Networks

- costituiscono l'esempio oggi più utilizzato di algoritmo di machine learning
- un ANN è un modello matematico basato sulla similitudine con i network neurali biologici:

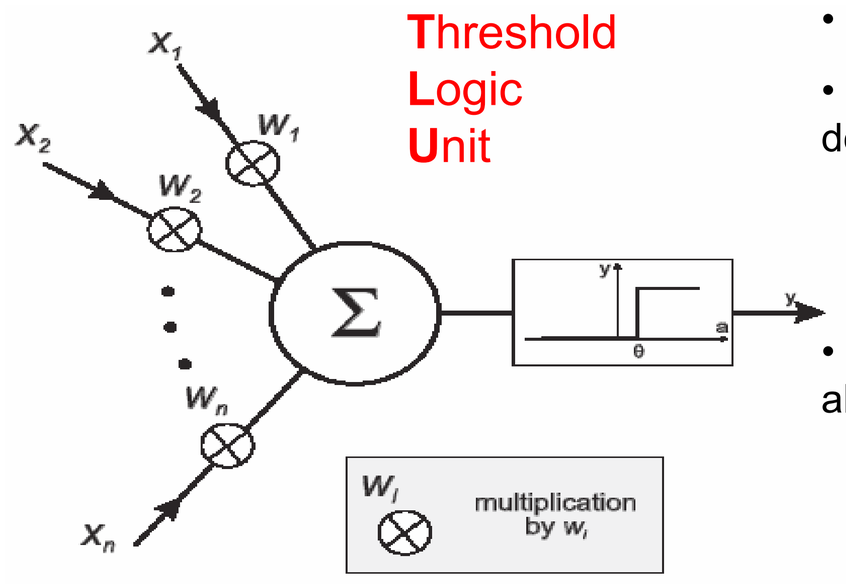
  - consiste in un gruppo interconnesso di neuroni artificiali
  - analizza informazioni in input in accordo ad un approccio computazionale di tipo connessionista → azioni eseguite in modo collettivo ed in parallelo da unità semplici (neuroni)
  - si comporta come un sistema adattivo: modifica la sua struttura basandosi su un insieme di informazioni che fluiscono attraverso la rete durante la fase di apprendimento



- la risposta non lineare viene ottenuta attraverso l'attivazione di nodi di output tramite funzioni non lineari
- Il representation learning gerarchico viene ottenuto attraverso l'utilizzo di architetture complesse costituite da molti strati di neuroni artificiali (deep-NN)

Il modello del neurone artificiale

Modello di McCulloch-Pitts (1943) / Rosenblatt (1962)



TLU:
Threshold
Logic
Unit

Caratteristiche:

- I segnali sono binari (x_i=0/1)
- La funzione di attivazione è definita come:

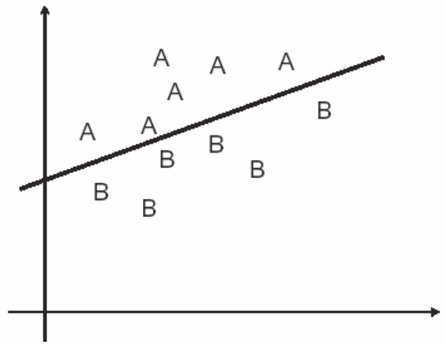
$$a = \sum_{i=1}^{n} w_i x_i$$

- L'uscita è definita in base alla regola:

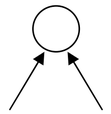
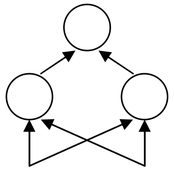
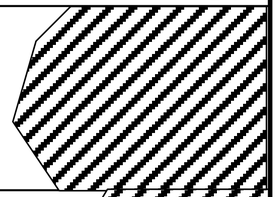
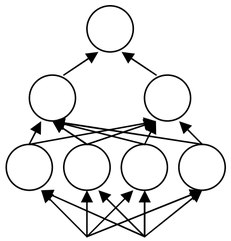
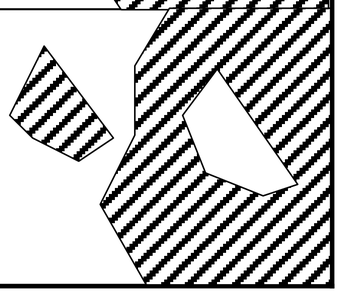
$$y = \begin{cases} 1 & if \quad a \geq \vartheta \\ 0 & if \quad a < \vartheta \end{cases}$$

con una TLU è possibile risolvere problemi in cui le classi siano linearmente separabili

Ex.: 2 classi linearmente separabili:



regioni di decisione delle reti neurali

Struttura	Regioni di decisione	Forma generale
	Semispazi delimitati da iperpiani	
	Regioni convesse	
	Regioni di forma arbitraria	

Universal Approximation Theorem

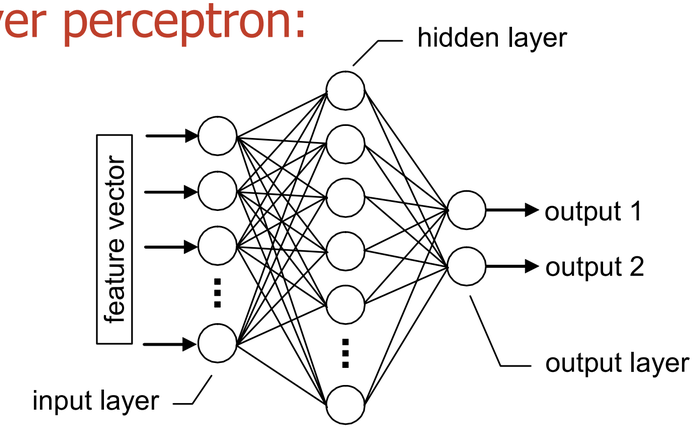
un Feed-Forward NN, con un singolo layer nascosto, contenente un numero finito di neuroni, può approssimare con la precisione voluta una funzione continua in qualunque sottoinsieme compatto di R^m sotto assunzioni minimali sulla funzione di attivazione

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

NOTA: il teorema non dice nulla sulla possibilità effettiva di apprendere facilmente i parametri della rete!

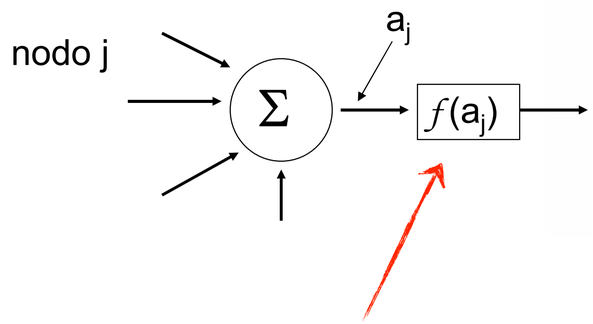
Feed-forward ANN

- Gli ANN più utilizzati hanno una struttura di tipo Feed-Forward multilayer perceptron:

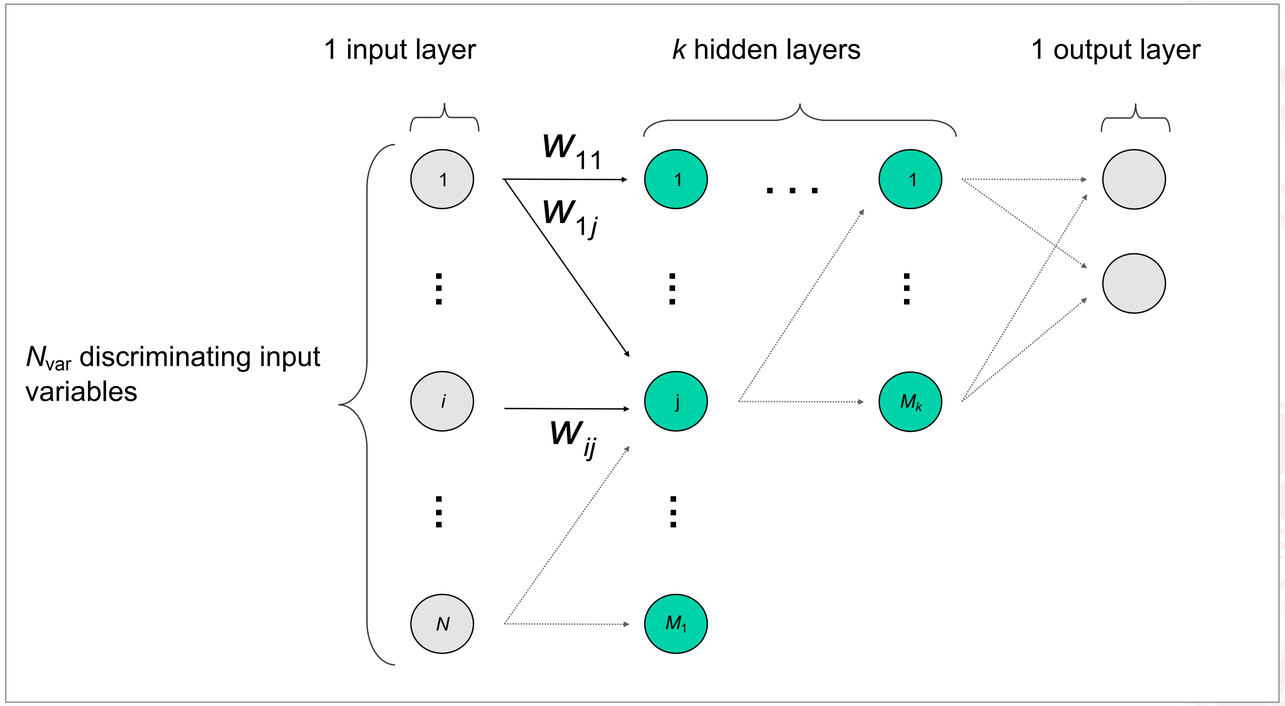


  - neuroni organizzati in layers: input, hidden-1, ... , hidden-K, output
  - sono possibili solo connessioni da un dato layer a quello immediatamente successivo

Nodo



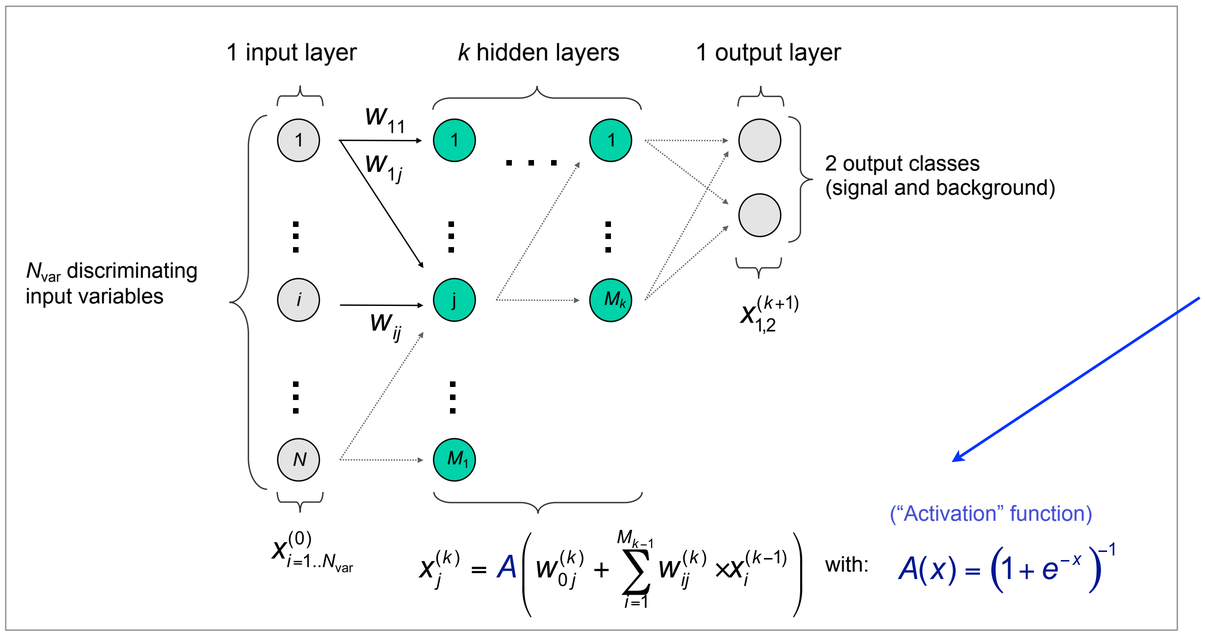
funzione di risposta (o di uscita)

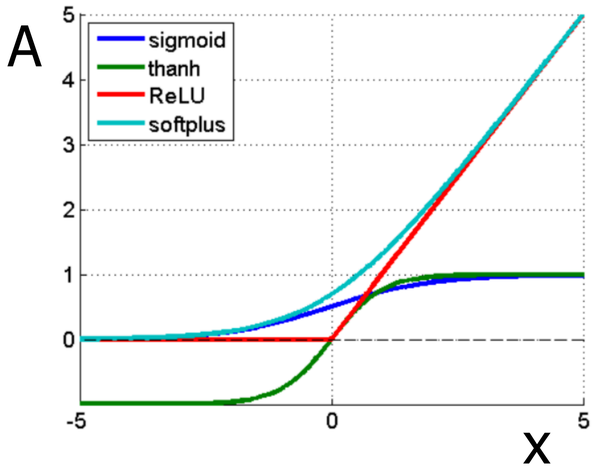


Feed-forward Multilayer Perceptron

Funzione di risposta

- comportamento della rete neurale determinato da:
  - struttura topologica dei neuroni
  - pesi associati alle connessioni tra neuroni
  - risposta dei neuroni ai dati di input
- Funzione di risposta ρ:
  - mappa l'input dal neurone $x^{(k-1)}_1,\ldots,x^{(k-1)}_n$ nell'output del neurone $x^{(k)}_j$
  - viene normalmente separata in una funzione sinaptica $k:R^n \rightarrow R$ ed una funzione di attivazione neurale $A:R \rightarrow R$: $\rho = k \bullet A$





$$A : x \rightarrow \begin{cases} \text{Lineare: } x \\ \text{Sigmoide: } 1/(1+e^x) \\ \text{Tanh(x)} \\ \text{ReLU: max(0,x)} \\ \text{softplus: } \log(1+e^x) \end{cases}$$

Training del NN

- l'operazione di aggiustamento dei pesi di ogni singola sinapsi è chiamata training della rete neurale
- nel training i pesi vengono aggiustati in modo da ottimizzare la risposta del classificatore (minimo errore di classificazione, massima separazione S/B, ...)
- tecnica piu utilizzata per il training: Back-propagation

Output per un ANN con:
- singolo layer nascosto con A: tanh
- output layer con A: lineare

n_h: numero neuroni layer nascosto

n_{var}: numero neuroni input layer

$$y_{ANN} = \sum_{j=1}^{n_h} x_j^{(2)} w_{j1}^{(2)} = \sum_{j=1}^{n_h} \tanh\left(\sum_{i=1}^{n_{var}} x_i w_{ij}^{(1)}\right) w_{j1}^{(2)}$$

peso tra j-esimo neurone layer nascosto e neurone di output

peso tra i-esimo neurone layer input e j-esimo neurone layer nascosto

Training del NN

- durante il processo di addestramento del NN vengono forniti al network N eventi: $\mathbf{x}_a$ (a=1,..,N)
- per ogni evento di training viene calcolato $y_{ANN}(a)$ e confrontato con l'output aspettato: $Y_a \in \{0,1\}$ (0 per eventi fondo, 1 per eventi segnale)
- viene definita una funzione di perdita (loss function), per esempio come misura della distanza tra $y_{ANN}(a)$ e Y_a:

$$\Delta(x_1,...,x_N|\mathbf{w}) = \sum_{\mathbf{a=1}}^{\mathbf{N}} \boldsymbol{\Delta}_{\mathbf{a}}(\mathbf{x_a}|\mathbf{w}) = \sum_{\mathbf{a=1}}^{\mathbf{N}} \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{y_{ANN}(a)} - \mathbf{Y_a})^{\mathbf{2}}$$

MSE

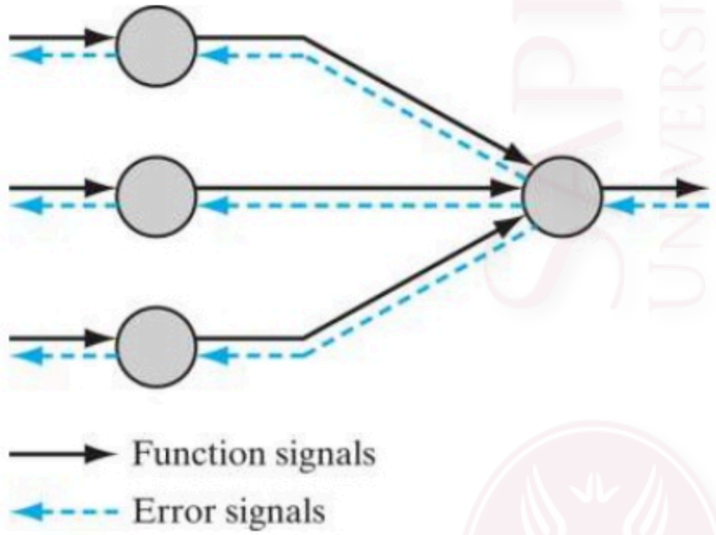
- il set di pesi w ottimizzato è quello che minimizza la loss function Δ
  - minimizzazione ottenuta con metodi steepest descent/gradient descent

$$\mathbf{w}^{(\rho+\mathbf{1})} = \mathbf{w}^{(\rho)} - \eta \nabla_{\mathbf{w}} \boldsymbol{\Delta}$$

dato un set random di pesi $w^{(\rho)}$, i pesi sono aggiornati muovendosi di una piccola distanza in direzione $\nabla_w \Delta$

Back-propagation

- Il training di un FFNN procede in 2 fasi
  - fase Forward: i pesi della rete sono fissati e il vettore di input viene propagato attraverso la rete layer per layer fino a raggiungere l'uscita (function signal)
  - fase Backward: l'errore Δ ottenuto confrontando l'output della rete con la risposta attesa viene propagato attraverso la rete, ancora layer per layer, ma in direzione opposta a quella della fase forward (error signal)

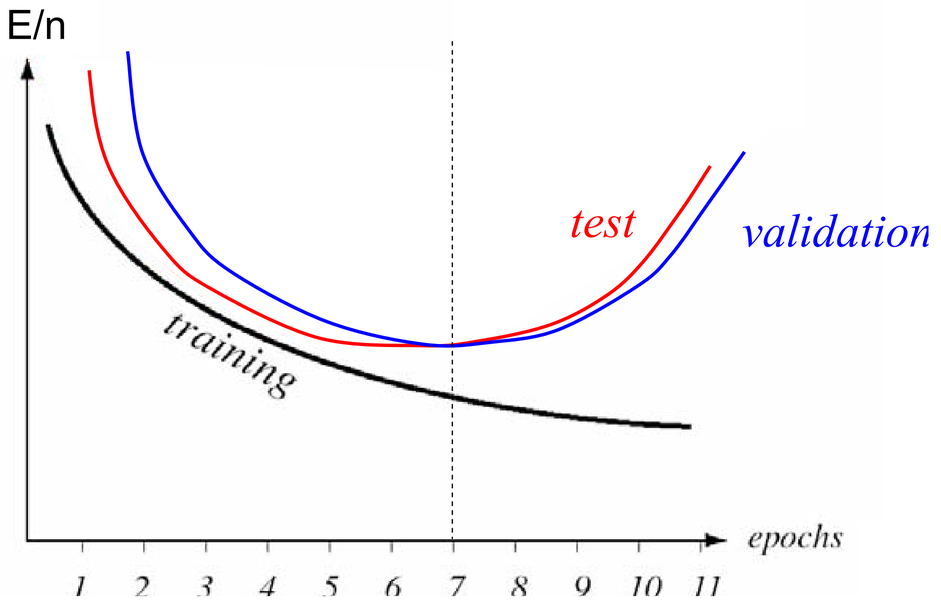


- ogni neurone (hidden o output) riceve e calcola i segnali function e error

- la procedura di back-propagation consiste in una semplificazione della discesa lungo il gradiente ottenuta applicando consecutivamente la regola di derivazione di funzione di funzione

Curve di apprendimento

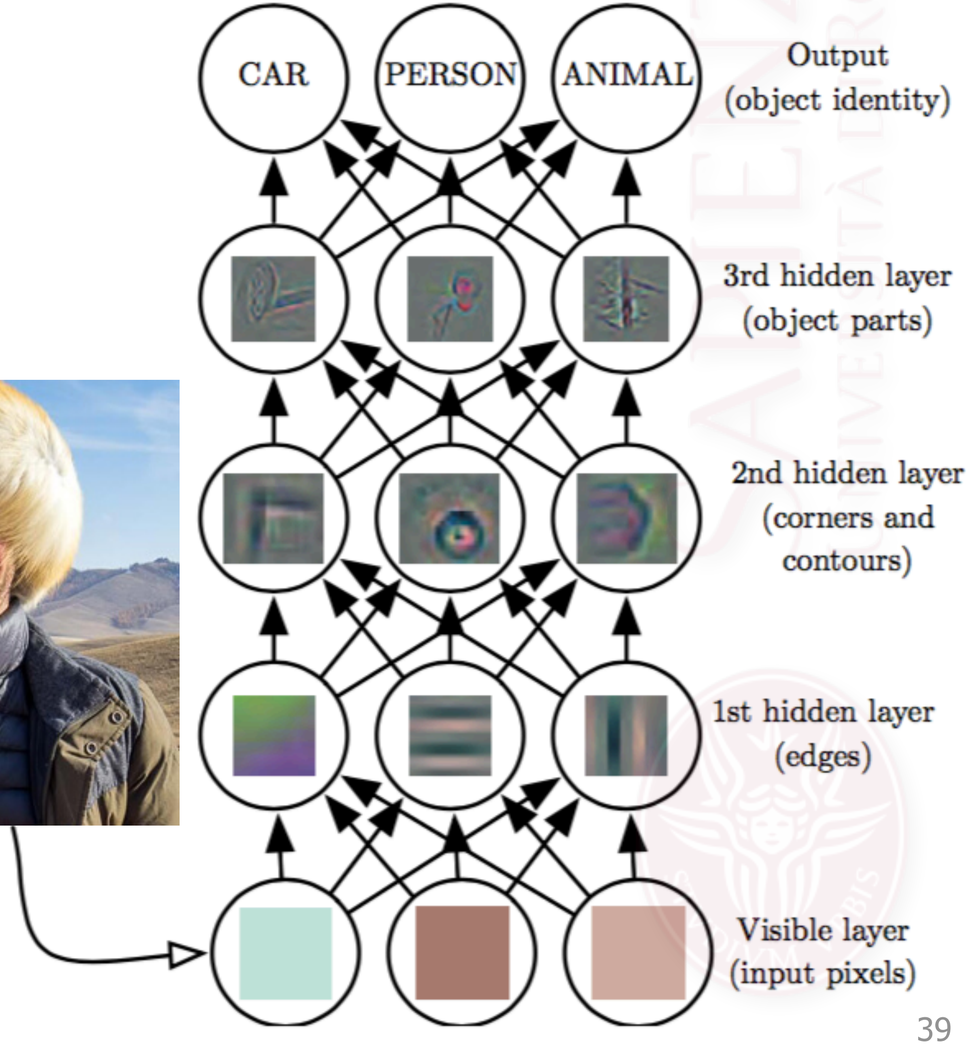
- all'inizio della fase di training, l'errore sul training set è tipicamente alto
- con il procedere delle iterazioni, l'errore tende a diminuire, raggiungendo un valore asintotico che dipende da:
  - dimensione del training set
  - numero dei pesi della rete
  - valore iniziale dei pesi
- l'andamento dell'errore rispetto al numero di epoche è visualizzato su una curva di apprendimento
- necessari più campioni indipendenti (o cross-validation) per decidere l'architettura, ottimizzare la rete, decidere il criterio di stop



Deep Neural Networks

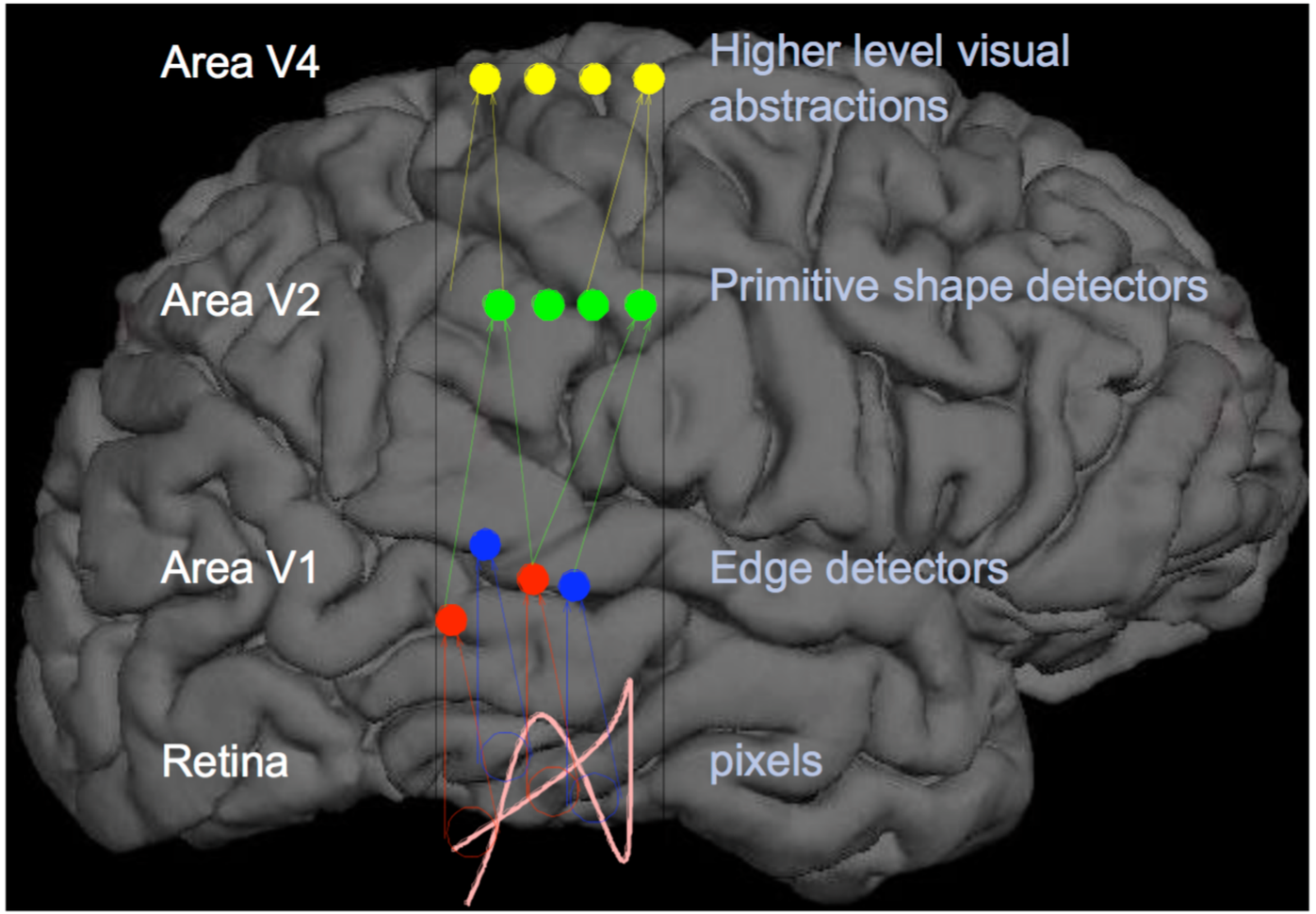
- idea base: tramite la composizione di un numero sufficientemente grande di moduli/layer di trasformazione è possibile apprendere funzioni (superfici di separazione) di estrema complessità ed eliminare nel processo variazioni irrilevanti

- immagine → array di pixel raw
- → primo layer di rappresentazione: cerca presenza /assenza di forti variazioni tonali in punti e direzioni particolari dell'immagine
- → secondo layer di rappresentazione: cerca particolari pattern di variazioni a prescindere dalle piccole variazioni tonali
- → terzo layer: combina i pattern in combinazioni più estese che corrispondono a parti di oggetti familiari
- → etc. etc..



Concetto noto da più di 50 anni (primo DNN del 1965), praticamente si è sviluppato solo a partire dalla seconda metà degli anni 2000 grazie a GPU potenti e campioni di addestramento di grande dimensione

Architettura di tipo deep del Cervello



- organizziamo idee e concetti gerarchicamente
- prima impariamo concetti semplici e poi li componiamo per rappresentare concetti più astratti
- il deep-learning cerca di emulare questo comportamento …

Potenziali criticità in un DNN

- oltre alle problematiche legate alla velocità di training e alla scelta dell'architettura ottimale, il problema più serio per una rete neurale profonda è legato all:

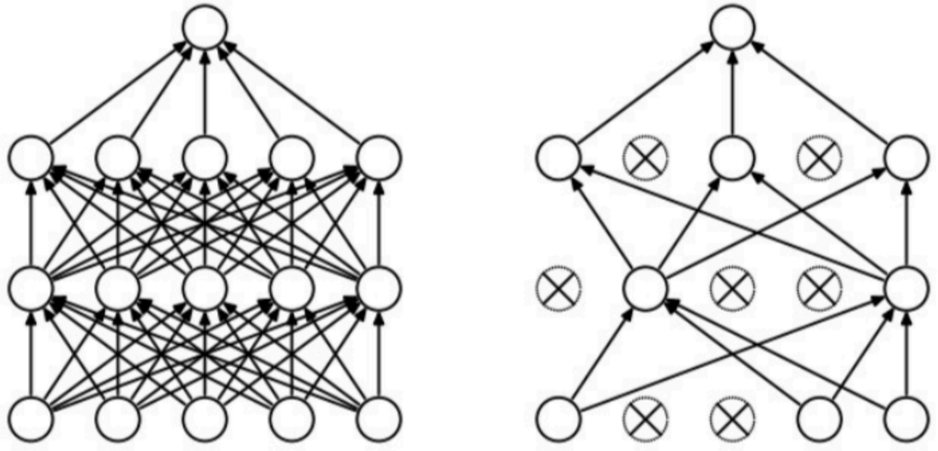
Hardcore Overfitting

- Soluzione: Regolarizzazione:
- 2 metodologie classiche: regolarizzazione L1 e/o L2, e Dropout

  Regolarizzazione L1/L2: si vincolano i valori dei pesi w_i ad essere piccoli (al limite zero) nelle dimensioni dello spazio delle feature meno informative riducendo la complessità del modello

Dropout:

prima



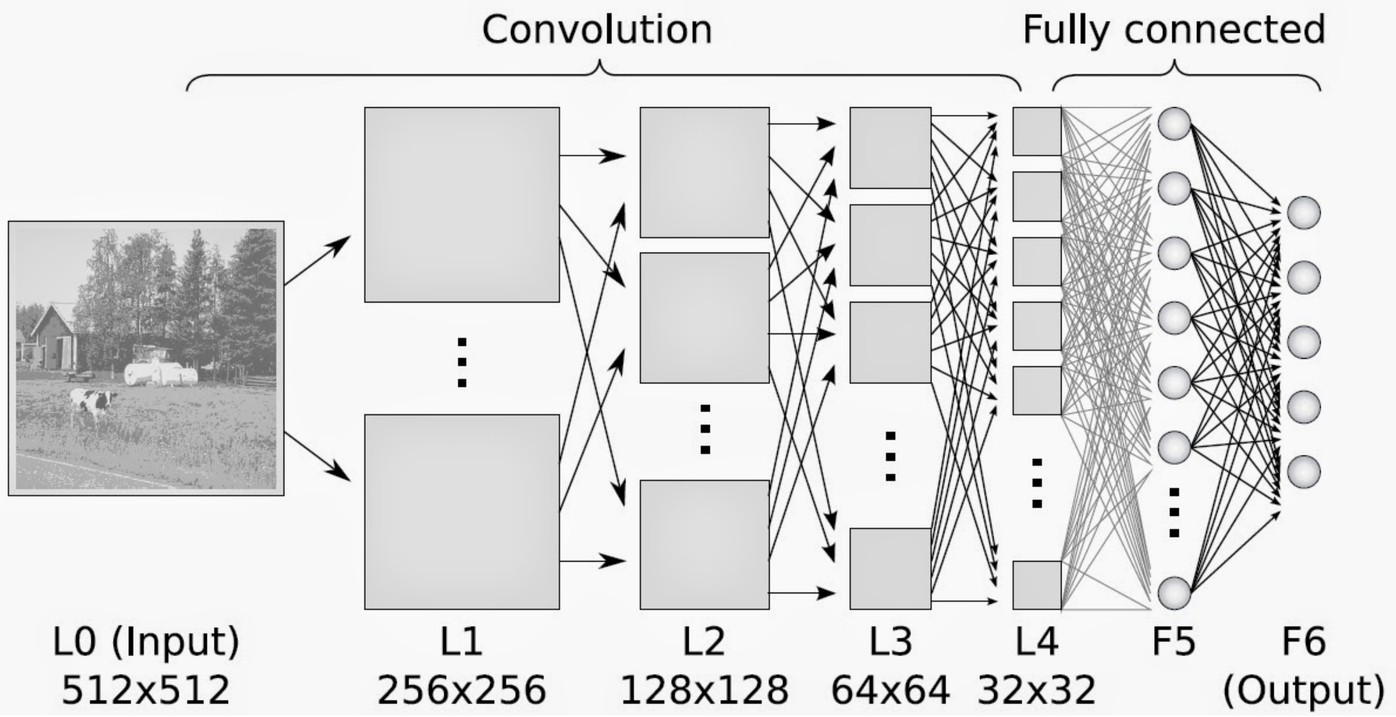
dopo

vengono messi a zero i pesi della rete con una probabilità prestabilita ex. ~30%

migliora drasticamente la capacità di reti profonde

La più nota architettura DNN: ConvNet

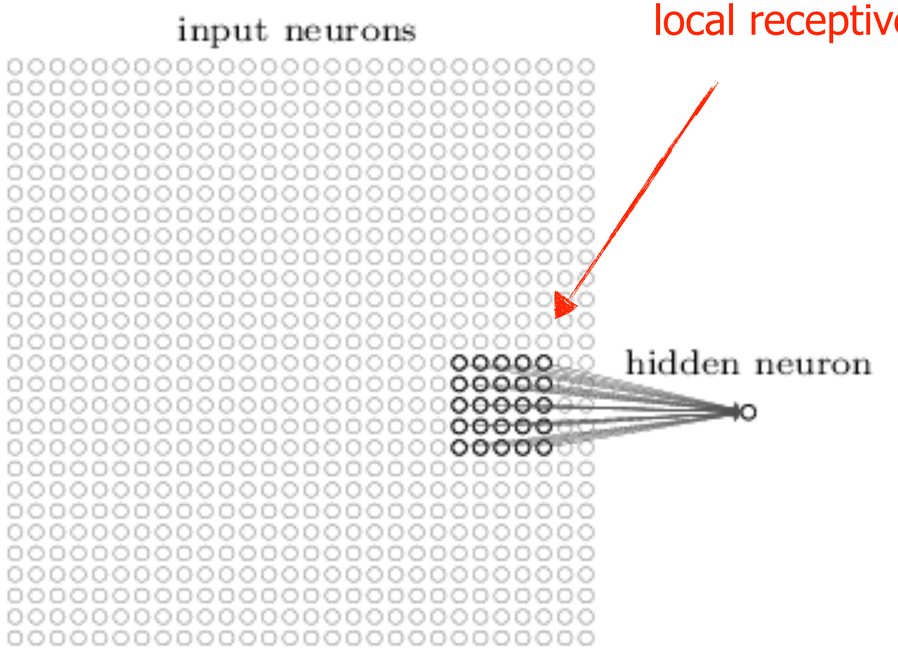
- architettura che ha ottenuto eccellenti prestazioni in applicazioni pratiche tipo object-recognition di immagini, etc.
  - le CNN sono progettate per elaborare dati rappresentati sotto forma di tensori: immagini (array bi-dimensionali contenenti le intensità dei pixel)
  - operano direttamente sulle immagini. L'input di una CNN quindi, a differenza di quello di una rete neurale ordinaria, sarà bidimensionale e le features saranno i pixel stessi delle immagini

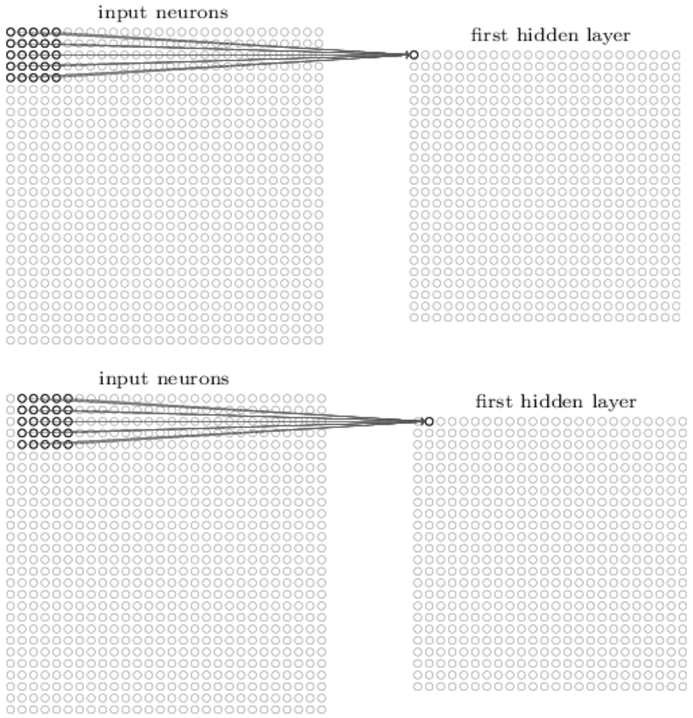


- Convolutional layers:
  - sfruttano tre idee di base
    - local receptive fields
    - kernels (shared weights)
    - pooling layers

Convolutional-NN

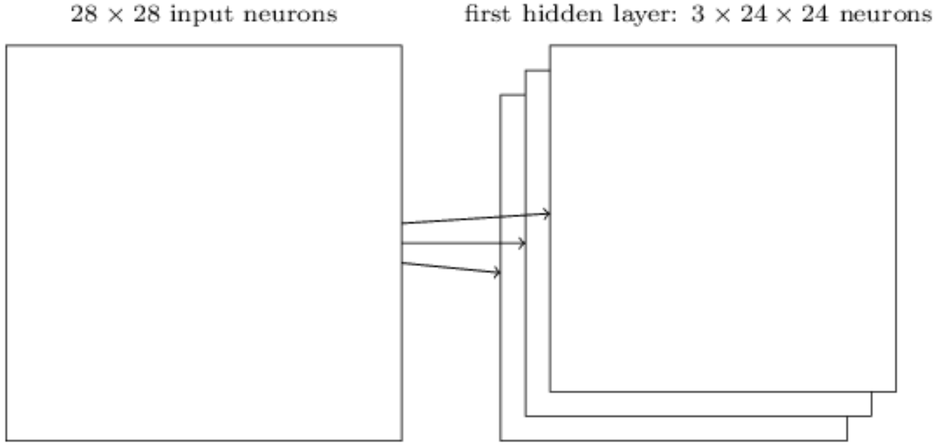
- local receptive fields:
  - i neuroni di input (corrispondenti agli NxN pixel che costituiscono l'immagine) non sono tutti connessi a tutti i neuroni di uno dei layer nascosti. Le connessioni vengono fatte solo con regioni localizzate e piccole dell'immagine di input
  - successivamente il local receptive field viene fatto scorrere attraverso tutta l'immagine. Per ogni local receptive filed shiftato ci sarà un neurone hidden nel layer nascosto



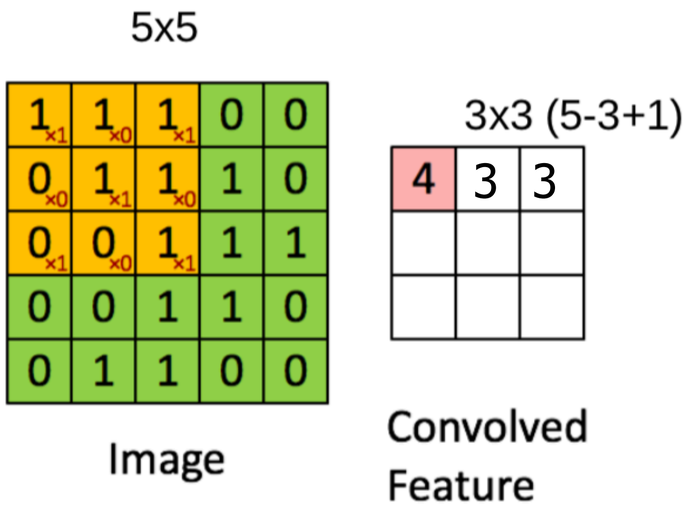


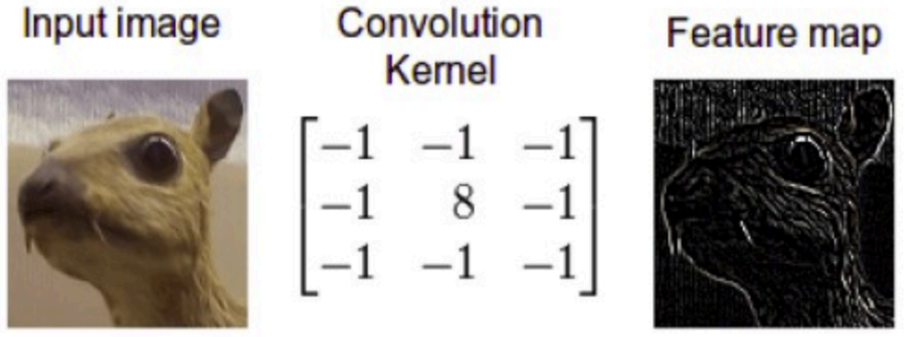
Convolutional-NN

- shared-weights (detti filtri o kernel):
  - tutti i neuroni nascosti di un local receptive field usano gli stessi pesi e bias. Questo significa che tutti i neuroni del layer nascosto rivelano la stessa identica sub-feature, solo in punti diversi dell'immagine
  - funziona grazie all'invarianza traslazionale di un immagine: se traslo l'immagine di un gatto rimarrà sempre l'immagine di un gatto
  - visto che la rete dovrà rivelare più sub-feature, di layer nascosti ve ne saranno molteplici, ognuno si occuperà di riconoscere una determinata sub-feature
  - il grande vantaggio è con questo metodo il numero di pesi della rete si è ridotto notevolmente



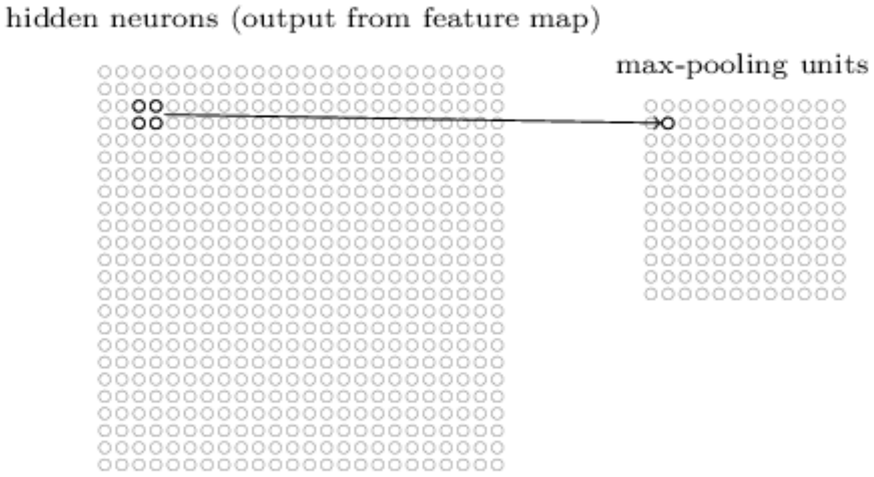
Esempio applicazione di un filtro

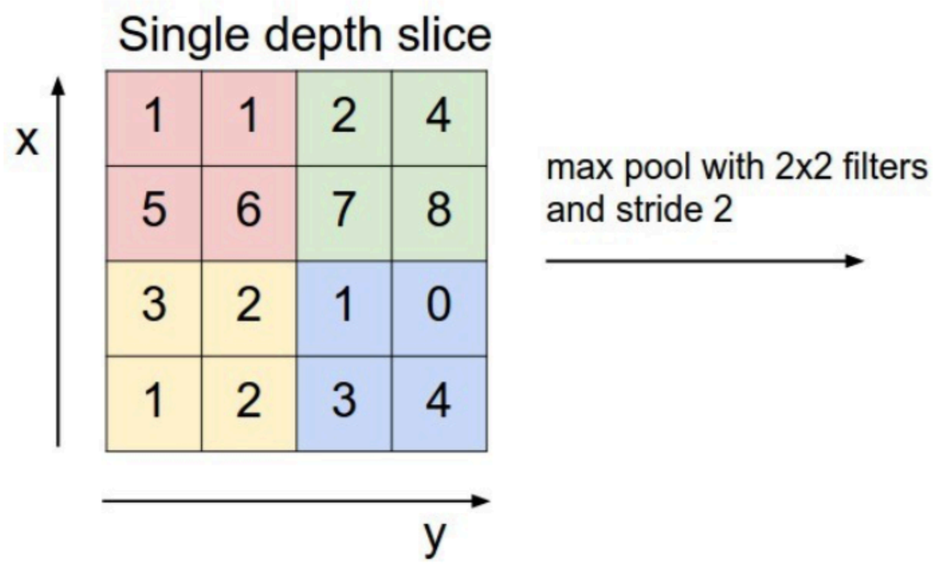




Convolutional-NN

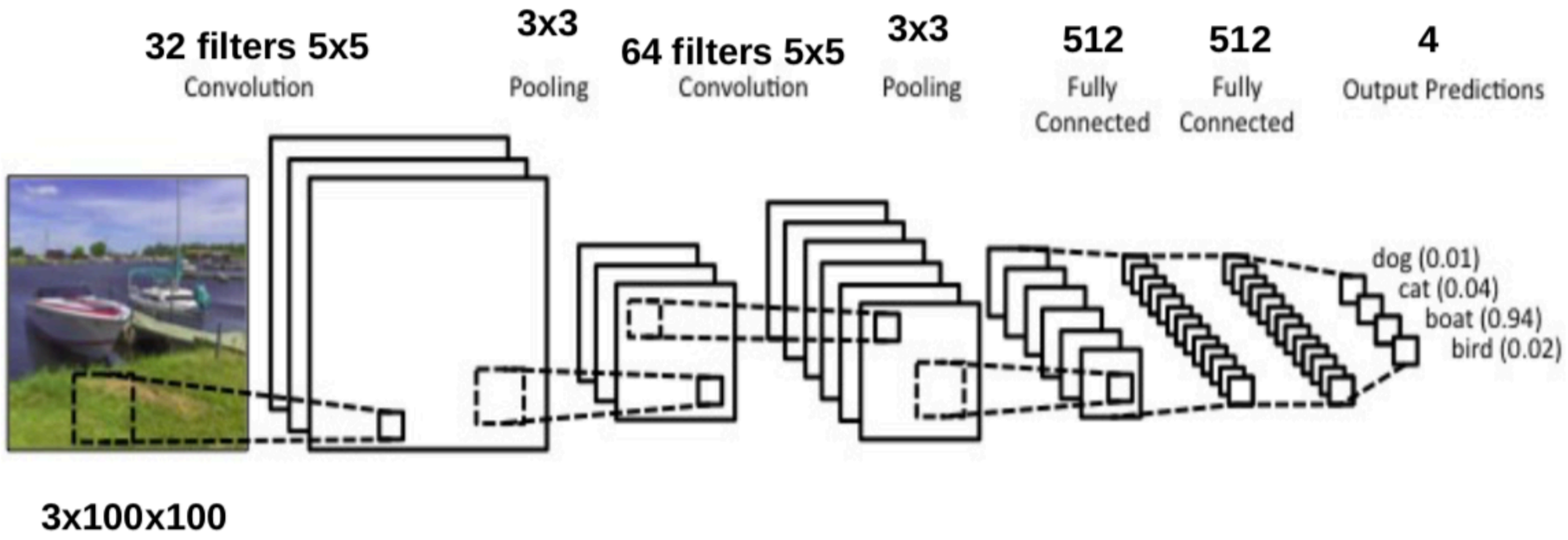
- pooling layers:
  - oltre a layer di convoluzione, un CNN contiene anche layer detti di pooling
  - scopo: semplificare l'informazione in uscita dal layer convoluzionale (meno pesi) + rendere il NN meno sensibile a piccole traslazioni dell'immagine
  - il max-pooling funziona come un modo per la rete di scoprire se una data sub-feature sia stata trovata in qualche parte dell'immagine, e una volta scoperto ciò di buttare l'informazione posizionale esatta contando sul fatto che una volta trovata la sub-feature la sua posizione esatta non sia importante quanto conoscerne la posizione relativa rispetto alle altre sub-feature





Convolutional-NN

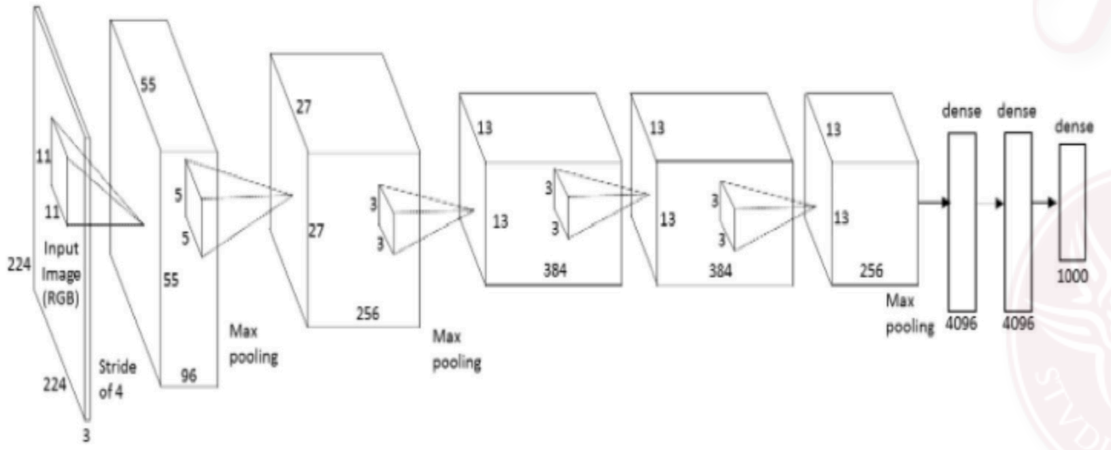
- infine l'output dei layer convoluzionali viene connesso (attraverso un layer di flattening) con uno o più layer densi (una NN tradizionale) che si occupa di risolvere il problema specifico (classificazione, regressione, etc.) utilizzando come input il set di feature prodotto dal CNN



Il primo CNN ad alte prestazioni: AlexNet

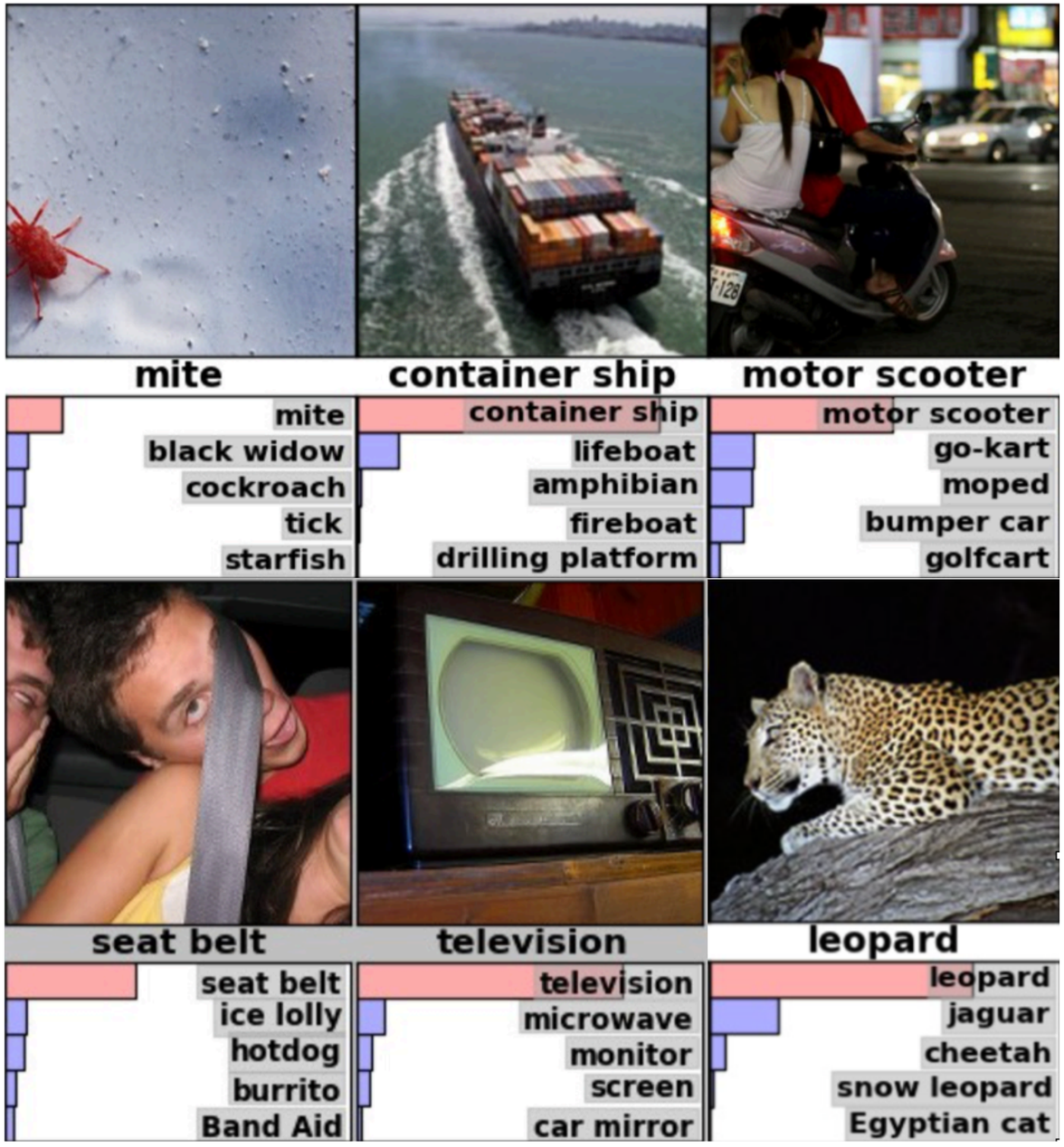
- NN basato su architettura Krizhevsky et Al. vincitore del Imagnet 2012 Contest
- sviluppato in ambiente Caffe (Berkeley Vision Deep Learning framework: http://caffe.berkeleyvision.org)
  - Istruzioni per installare una macchina virtuale sul proprio pc (mac os x) con caffe: https://vimeo.com/101582001

- Usa un Convolutional-NN in 2D
  - 7 hidden weight layers
  - tutti i feature extractor inizializzati con rumore gaussiano bianco e addestrati basandosi su i dati di input
  - addestramento totalmente supervisionato
  - training su GPU NVIDIA per ~1settimana
  - 650K neuroni
  - 60M parametri
  - 630M connessioni
  - layer finale: a 4096 dimensioni



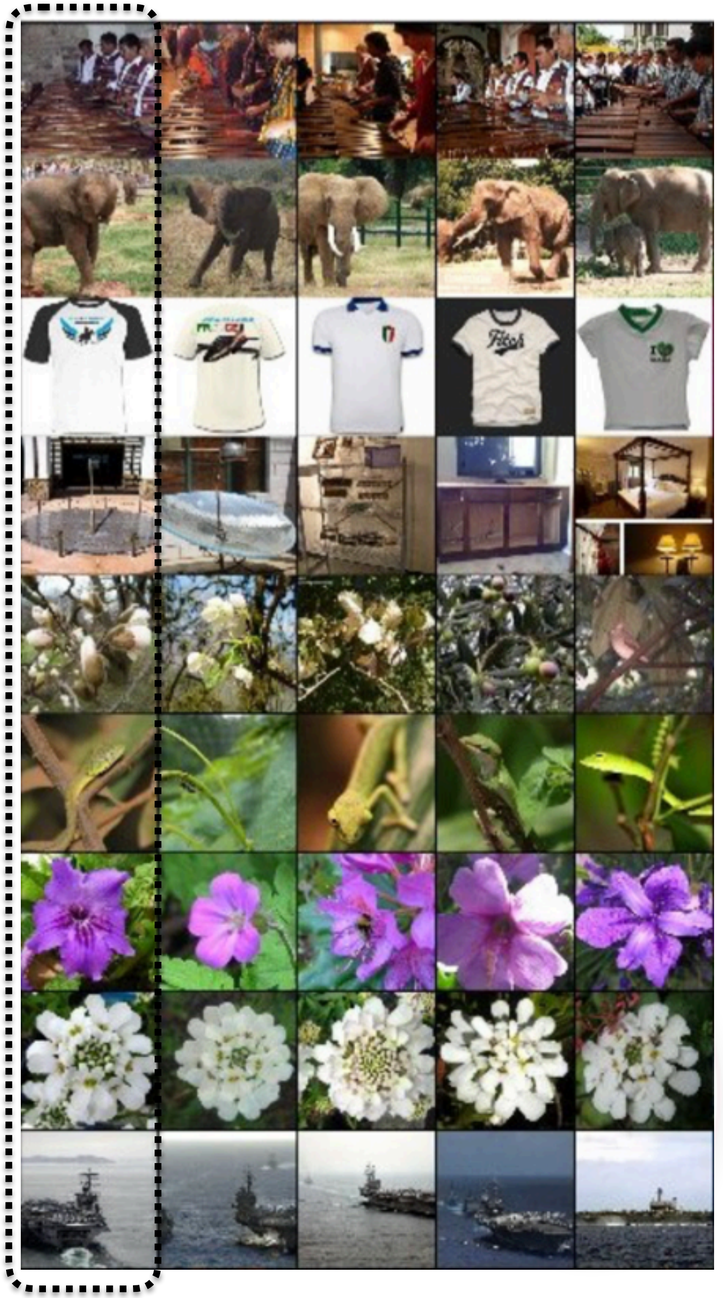
(Krizhevsky et al.)

AlexNet



classification

retrieval

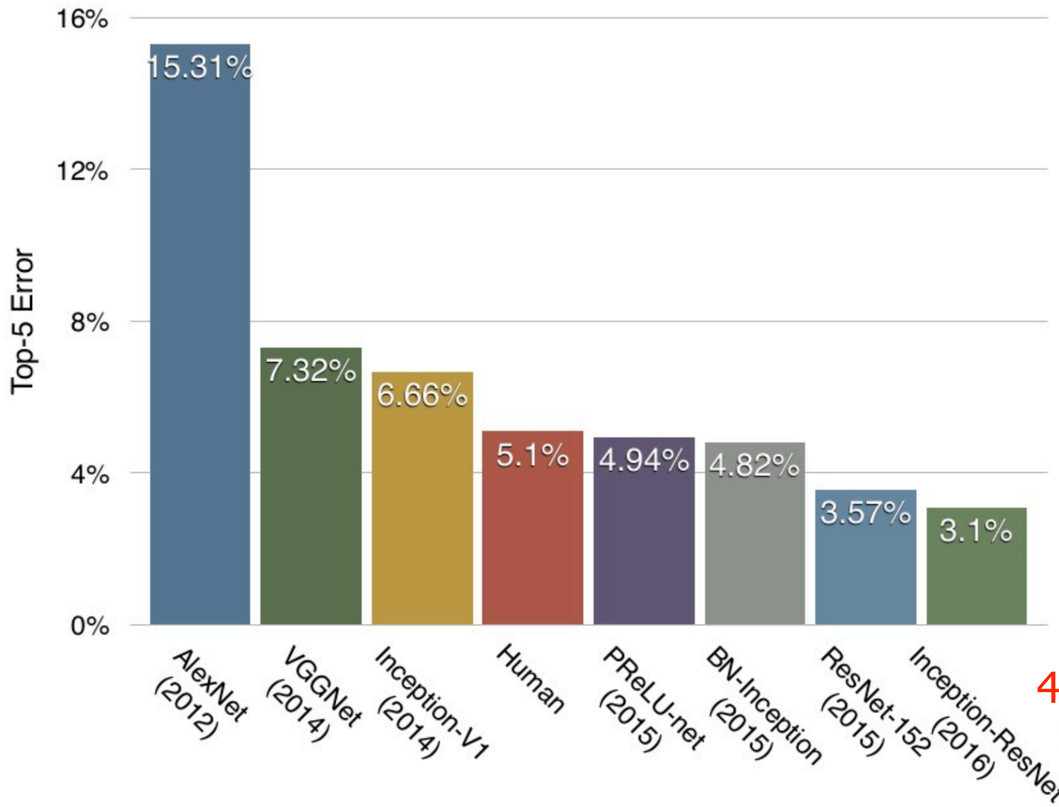


AlexNet→VGG→Inception→ResNet→…

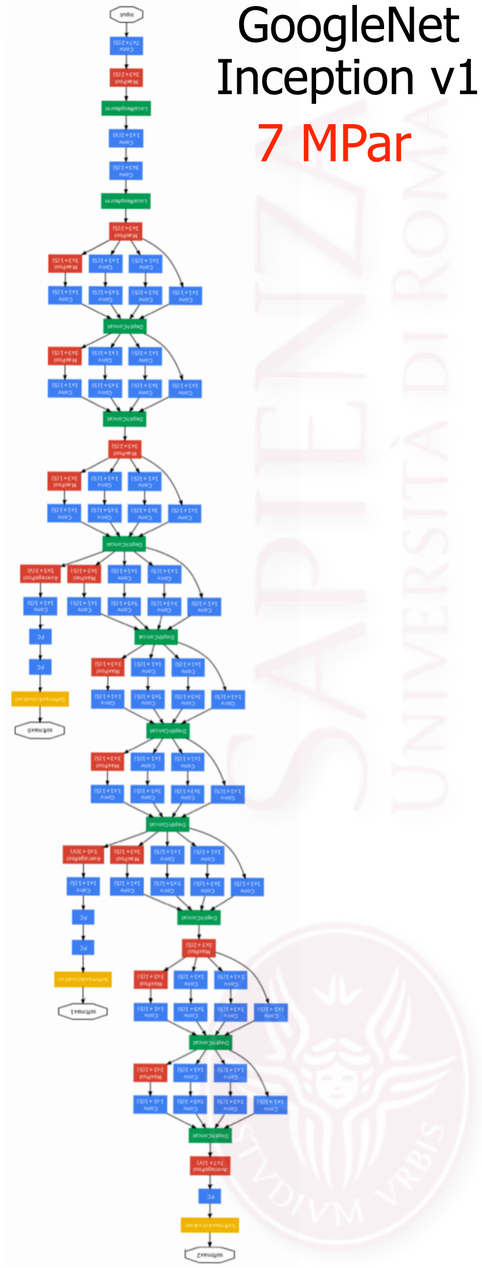
AlexNet, 8 layers
(ILSVRC 2012)
61 MPar

VGG, 19 layers
(ILSVRC 2014)
138 MPar

ResNet, 152 layers
(ILSVRC 2015)
60 MPar



467 layers
54 MPar



GoogleNet
Inception v1
7 MPar

Esempio: addestrare una CNN con Keras/Tensorflow sul vostro laptop

Istallazione Keras/Tensorflow: https://keras.io/#installation

```
import tensorflow as tf
import keras
```

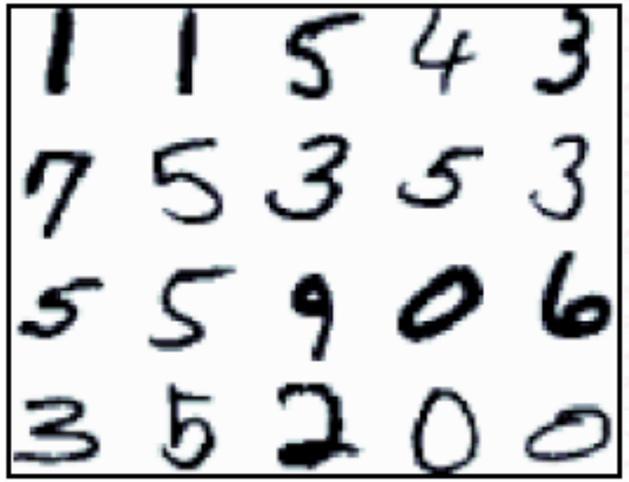
```
#import models & layers from keras
from keras import models
from keras import layers
```

```
#retrieve training data (MNIST dataset included in keras/tenswrflow
)
#60k images fro training and 10k images for training
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.lo
ad_data()

#verify:
train_images.shape
```

```
(60000, 28, 28)
```

```
train_labels
```

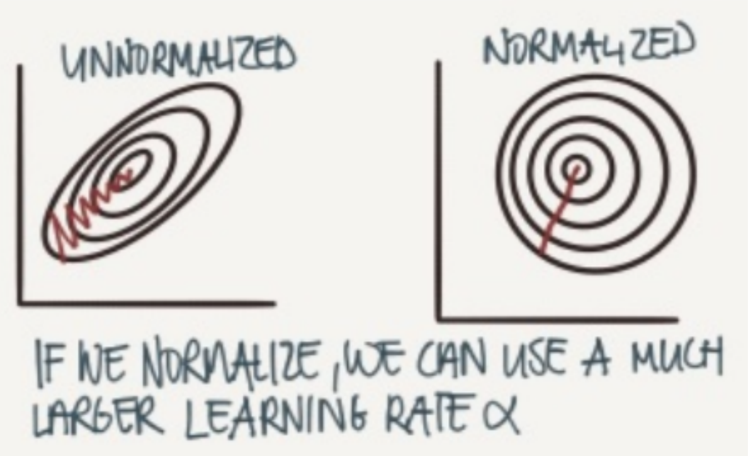
```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```



Data Preprocesssing

```
#training data preprocessing
# convert images to 28x28x1 pixelx X pixely X channels
train_images = train_images.reshape((60000, 28, 28, 1))
# normalize intensity to [0,1]: original image is uint8 (0,255)
train_images = train_images.astype('float32') / 255
#same for test images
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
```

Dense layers si aspettano inputs normalizzati in [0,1]

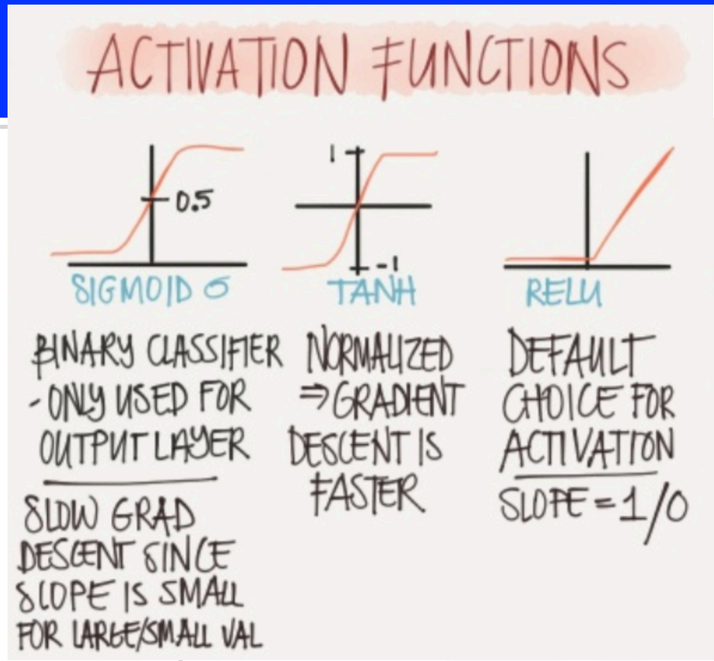



```
# encode the labels in category using one_hot encoding: builtin met
hod available in keras
from keras.utils import to_categorical


train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

one-hot encoding delle label di categoria … 10 categorie: (1,0,…,0); (0,1,…,0)…(0,0,…1)

CNN Model Definition

```
# Model definition (aka define the architecture of the nextwork)
from keras import optimizers
from keras import losses
from keras import metrics
from keras.optimizers import SGD, RMSprop, Adam
#sequential (aka Feed-Forward Convolutional Neural Network)
network = models.Sequential()

# Convnet architecture: 3 Conv2D layers followed by MaxPooling layers
network.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
network.add(layers.MaxPooling2D((2, 2)))
network.add(layers.Conv2D(64, (3, 3), activation='relu'))
# followed by a Classifier (FFNN)
network.add(layers.Flatten())   #flatten input in order to be used as input for the c
# one hidden dense layer
network.add(layers.Dense(64, activation='relu'))
# output layer: with 10 neurons (10 characters: 0 to 0)
# activation function is softmax: means each one of the neurons output a probability
# with the constraint sum(outputs) = 1 (i.e. the output neurons add up to 1)
network.add(layers.Dense(10, activation='softmax'))
```



CNN Model Definition

```
network.compile(optimizer=RMSprop(lr=LRDEF),
                loss=losses.categorical_crossentropy,
                metrics=['acc'])

#print summary of the model
network.summary()
```

```
# default learning rate
LRDEF = 0.001
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

NOTA: una struttura senza pooling layers non avrebbe permesso alla rete di apprendere le feature in modo gerarchico

È fondamentale disegnare la rete in modo tale che ogni layer successivo "guardi" a porzioni sempre più grandi dell'immagine

la struttura CNN permette di limitare il numero di parametri della rete

CNN Training

```
# Train phase
# define batch_size for the stcastic gradient and number of epochs ...

history = network.fit(part_train_images, part_train_labels, epochs=N_EPOCHS,
                      batch_size=BATCH_SIZE,
                      verbose=VERBOSE)
```

```
Epoch 5/5
48000/48000 [==============================] - 56s 1ms/step - loss: 0.0206 - acc: 0.9937 -
```

```
#checks on test sample
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
10000/10000 [==============================] - 2s 244us/step
test_acc: 0.9902
```

99% accuracy con CNN semplice e senza nessuna ottimizzazione specifica!

```
#number of epochs
N_EPOCHS    = 5
# batch size (number of trainqin events
BATCH_SIZE = 64      after each weight update)
VERBOSE = 1
```