

UNIVERSITÁ DEGLI STUDI "ROMA TRE"  
Dipartimento di Matematica e Fisica  
Corso di Laurea Magistrale in Scienze Computazionali

Tesi di Laurea Magistrale

**Ricerca della topologia ottimale di  
un sistema di deep learning per  
identificazioni di oggetti architettonici**

Candidato  
Dèsirée Adiutori

Relatore  
Prof. Alberto Paoluzzi

Anno Accademico 2017/2018  
Luglio 2018

# Indice

Introduzione . . . . .	3
<b>1 Algoritmi di apprendimento</b>	<b>4</b>
1.1 Costruzione di un algoritmo di apprendimento . . . . .	4
1.2 Apprendimento supervisionato . . . . .	9
1.2.1 Regressione . . . . .	9
1.3 Apprendimento non supervisionato . . . . .	10
1.4 Apprendimento per rinforzo . . . . .	11
<b>2 Reti neurali</b>	<b>12</b>
<b>3 Classificazione</b>	<b>15</b>
3.1 K-Nearest Neighbor . . . . .	15
3.2 Alberi di decisione . . . . .	17
<b>4 TensorFlow</b>	<b>18</b>
<b>Bibliografia</b>	<b>20</b>

## Introduzione

Dall'invenzione dei computer, l'uomo fa sempre più affidamento sulle macchine per risolvere problemi complessi di calcolo. Con l'aumentare delle prestazioni dei computer, man mano si sono sviluppati algoritmi di calcolo sempre più efficienti. Nel 1959 l'ingegnere del MIT, Arthur Samuel coniò il termine "*machine learning*", descrivendo l'apprendimento automatico come un "campo di studio che dà ai computer la possibilità di apprendere senza essere programmati esplicitamente per farlo".[1]

Definiamo l'apprendimento automatico come un insieme di metodi in grado di rilevare automaticamente i modelli tramite dei dati e quindi utilizzare i modelli scoperti per prevedere i dati futuri o per eseguire altri tipi di processi decisionali in condizioni di incertezza. L'insegnamento alla macchina è, pertanto, imprescindibile dai dati. Generalmente, più dati si passano alla macchina, più può imparare. Per questo motivo con l'avvento di Internet, dagli anni '90 ad oggi il tema del "*machine learning*" è diventato sempre più attuale, la mole di dati reperibile dal web è cospicuo e ha permesso che questo campo sia esponenzialmente progredito.

Il "*deep learning*" è un tipo particolare di machine learning, che riguarda l'emulazione di come gli esseri umani apprendono. Esso affronta i problemi del machine learning, rappresentando il mondo come una gerarchia di concetti annidati: ogni concetto è definito in relazione a concetti più semplici e le rappresentazioni astratte vengono calcolate in termini di concetti meno astratti. Il Deep Learning implica l'utilizzo di reti neurali artificiali (*deep artificial neural networks*), algoritmi e sistemi computazionali, ispirati al cervello umano, per affrontare i problemi del Machine Learning.

L'analogia di Shehzad Noor Taus Priyo può aiutare a capire meglio cosa siano le reti neurali:

"Immaginiamole come una serie di porte da oltrepassare, dove l'input è l'uomo che le deve oltrepassare e ogni volta che lo fa cambia qualcosa nel suo comportamento finché, all'ultima porta oltrepassata, l'uomo è diventato una persona del tutto differente, rappresentando l'output di questo processo." [2]  
Questa tesi si focalizza su un problema particolare di algoritmo di machine learning: la Classificazione (*Classification*), in particolar modo della classificazione di immagini. L'obiettivo principale è trovare un'architettura ottimale per l'algoritmo che identifica le immagini di oggetti architettonici, per fare questo bisogna trovare la giusta topologia, la giusta profondità e la giusta larghezza di ogni livello della rete neurale.

# Capitolo 1

## Algoritmi di apprendimento

Gli algoritmi di machine learning sono solitamente divisi in tre tipi principali:

- Supervised learning (apprendimento supervisionato)
- Unsupervised learning (apprendimento non supervisionato)
- Reinforcement learning (apprendimento per rinforzo)

Quali usare? Perchè sceglierne uno piuttosto che un altro?

La scelta dell'algoritmo da utilizzare dipende dal tipo di dati di cui si dispone. Ma la scelta finale va fatta solo esclusivamente dopo aver testato l'algoritmo, e si sceglie in base a quello più performante: un insieme di ipotesi che funziona bene in un dominio, potrebbe funzionare male in un altro.

**Teorema del No Free Lunch [3, Wolper,1996]**

*Non esiste una definizione universale di algoritmo "migliore".*

### 1.1 Costruzione di un algoritmo di apprendimento

Per costruire un algoritmo di apprendimento bisogna avere:

- processi(*task*), compiti che l'algoritmo deve eseguire;
- misuratori di rendimento, rilevatori delle caratteristiche dei processi;
- esperienze, quantità di dati dal quale imparare.

I processi di apprendimento automatico descrivono come il sistema dovrebbe elaborare un esempio.

**Definizione 1** Un *esempio* è una raccolta di caratteristiche che sono state misurate quantitativamente da alcuni oggetti o eventi elaborati.

Di solito, un esempio viene rappresentato da un vettore  $x \in \mathbb{R}^n$ , dove ogni elemento  $x_i$  rappresenta una caratteristica. Dato un processo si cerca di capire quale sia la caratteristica principale, sulla quale si deve misurare il suo rendimento. Infine, dobbiamo dare all'algoritmo un'esperienza sulla quale apprendere, che è quella che lo classificherà in uno dei tre tipi principali. Questa esperienza l'apprende dai *dataset*: una collezione di esempi.

I dataset possono essere di vari tipi:

- insieme di addestramento ( *training set* )
- insieme di prova ( *test set* )
- insieme di validazione ( *validation set* )

L' **insieme di addestramento** è una parte dell'insieme di dati che vengono utilizzati per addestrare un sistema di apprendimento supervisionato. Da questo insieme, l'algoritmo deve costruire una funzione che capisca, dai parametri, quali caratteristiche descrivono le varie categorie.

L' **insieme di prova** è un insieme di dati che, con l'insieme di addestramento, forma una partizione del dataset di partenza. Questi nuovi dati vengono utilizzati per valutare l'apprendimento dell'algoritmo "addestrato".

L' **insieme di validazione** è usato in maniera analoga all'insieme di prova, ma dei dati inseriti per testare l'algoritmo già si conosce la risposta (una parte di essi può far parte dell'insieme di addestramento) e da questa si valuta se l'output ottenuto è ottimale o meno.

Questi tre insiemi possono essere usati anche tutti e tre contemporaneamente. La scelta della cardinalità dei vari sottoinsiemi non è universale e dipende dal tipo di problema che viene affrontato.

Vediamo ora come valutare l'efficienza di un algoritmo:

**Definizione 2** L'*errore di allenamento* (*training error*) è una misura di errore che si può calcolare sul set di allenamento. Indica quanto l'algoritmo sta apprendendo.

**Definizione 3** La *generalizzazione* è la capacità di un algoritmo di essere ottimale in seguito ad un input proveniente dall'insieme di prova.

**Definizione 4** L'*errore di generalizzazione* (*generalization error*) è una misura di errore che si può calcolare sull'insieme di prova. Verifica se l'algoritmo ha imparato o solo memorizzato. Esso viene detto anche errore di test (*test error*).

Ipotizziamo che tutti gli esempi siano eventi indipendenti e che tutti gli insiemi, in cui partizioniamo l'insieme di dati, hanno la stessa distribuzione di probabilità uniforme.

**Definizione 5** Una **funzione di perdita** (loss function)  $L(y, \hat{y})$  è una funzione che misura la distanza (o l'errore) tra i valori di output previsto  $\hat{y}$  e i valori effettivi  $y$ .

Si possono usare varie misure, per esempio nel caso dell'errore quadratico medio (MSE):

$$L(y, \hat{y})_{train} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_{(train)} - y_{(train)})_i^2 \quad (1.1)$$

La nostra funzione di predizione dipenderà da dei parametri, rappresentati da un vettore  $w$ , lo scopo è di minimizzare l'errore di allenamento variando  $w$ . In base al tipo di apprendimento e al problema da affrontare verranno usati vari algoritmi per risolvere problemi di minimizzazione libera. Ovvero gli algoritmi per risolvere il problema:

$$f(x^*) = \min_{x \in \mathbb{R}^n} f(x), \quad f \in C^2$$

Ad esempio verrà spesso utilizzato il metodo di discesa del gradiente.[?] La struttura generale di un metodo di discesa iterativo di minimizzazione è:

$$x_{k+1} = x_k + \beta_k d_k \quad (1.2)$$

dove  $x_0$  è assegnato,  $\beta_k \in \mathbb{R}^+$  è il passo e  $d_k \in \mathbb{R}^n$  è la direzione lungo la quale ci si muove, che essendo una direzione di discesa sarà  $(d_k, \nabla f(x_k)) < 0$ . Sia il passo che la direzione vanno scelti opportunamente ad ogni passo, in modo che  $f(x_{k+1}) < f(x_k)$ .

Il passo viene scelto in modo che si abbia:

$$f(x_k + \beta_k d_k) = \min_{\beta} \{f(x_k + \beta d_k)\} \quad (\text{strategia di ricerca esatta})$$

Mentre la scelta della direzione è  $d_k = -\nabla f(x_k)$ . La derivata direzionale di  $f$  nella direzione  $d_k$  vale

$$\frac{\partial f}{\partial d_k}(x_k) = \frac{(d_k, \nabla f(x_k))}{\|d_k\|} = -\|\nabla f(x_k)\|$$

e per la disuguaglianza di Cauchy-Schwartz si ha anche:

$$\frac{|(d_k, \nabla f(x_k))|}{\|d_k\|} \leq \frac{\|d_k\| \|\nabla f(x_k)\|}{\|d_k\|} = \|\nabla f(x_k)\|$$

che mostra come la direzione di ricerca sia quella in cui la derivata direzionale di  $f$  è negativa e di modulo massimo.

Le condizioni di arresto del metodo sono:  $\|x_{k+1} - x_k\| \leq m$ ,  $\|\nabla f(x_{k+1})\| \leq m'$  oppure  $k > k_{max}$ , dove  $m$  e  $m'$  sono soglie date e  $k_{max}$  il numero massimo di iterazioni da effettuare.

I risultati ottenuti sono garantiti dai seguenti teoremi di convergenza:

**Teorema 1** Sia  $f(x) \in C^1$ , strettamente convessa sull'insieme  $\Sigma_0 = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ , e la successione  $\{x_k\}$  sia generata tramite l'algoritmo (1.2). Si supponga

1. che l'insieme  $\Sigma_0$  sia compatto;
2. che le direzioni  $d_k$  siano t.c.  $\frac{(d_k, \nabla f(x_k))}{\|d_k\| \|\nabla f(x_k)\|} \leq -\cos \theta$  per  $k \in I$ , con  $I$  insieme illimitato di indici;
3. che per  $k \in I$ ,  $\beta_k$  sia ottenuto tramite ricerca esatta.

Allora la successione  $\{x_k\}$  converge all'unico punto  $x^*$  di minimo per  $f$ .

**Teorema 2** Sia  $f(x) \in C^2$ , strettamente convessa sull'insieme (che si suppone compatto)  $\Sigma_0 = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ , e la successione  $\{x_k\}$  sia generata tramite l'algoritmo (1.2), con  $d_k = -\nabla f(x_k)$ .

Allora, se i passi  $\beta_k$  sono determinati tramite ricerca esatta, la successione  $x_k$  converge all'unico punto  $x^*$  di minimo per  $f$ .

Minimizzare l'errore di allenamento non necessariamente comporta l'ottimizzazione di apprendimento dell'algoritmo, potrebbe verificarsi il fenomeno di adattamento insufficiente (*underfitting*), ovvero non si hanno abbastanza dati per creare un modello di predizione accurato. Bisogna quindi valutare anche altri fattori: analizzare l'insieme di prova.

Ricordandoci dell'eq.1.1, calcoliamo l'errore di prova:

$$L(y, \hat{y})_{test} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_{(test)} - y_{(test)})_i^2 \quad (1.3)$$

Vorremmo che, con i parametri trovati per minimizzare l'errore di allenamento, anche questo errore sia minimo (l'ottimalità è 0). Ma come detto in precedenza non sempre questo accade, vorremmo quindi che il divario tra i due errori sia minimo. In caso contrario si verifica il fenomeno di adattamento eccessivo (*overfitting*) del modello all'insieme di dati che descrive, tramite un eccessivo numero di parametri. Il modello quindi non sarà generalizzabile ad un nuovo insieme di dati.

Consideriamo il valore atteso dell'errore di prova, calcolato prendendo una coppia di punti  $(X, Y)$  dall'insieme di prova:

$$\mathbb{E}[L(y, \hat{y})_{test}] = \mathbb{E}[(Y - \hat{y}(X))^2] \quad (1.4)$$

e definiamo la funzione dell'output effettivo come:

$$y(X) = \mathbb{E}(Y|X)$$

la quale avrà sicuramente un errore, dovuto da qualche interferenza che chiameremo: *distorzione stimata (estimation bias)*.

Ma con diversi insiemi di allenamento, possiamo costruire diverse funzioni  $\hat{y}$ , e anche questo è un'altra fonte di errore: la *varianza stimata (estimation variance)*. Possiamo quindi scrivere l'output come:

$$Y = y(X) + \epsilon$$

con  $\epsilon$  indipendente da  $X$  tale che  $\mathbb{E}[X] = 0$  e  $Var(X) = \sigma^2$ .

Possiamo quindi riscrivere l'equazione 1.4 come:

$$\begin{aligned} \mathbb{E}[L(y, \hat{y})_{test}] &= \mathbb{E}[(Y - \hat{y}(X))^2|X = x] \\ &= \mathbb{E}[(Y - y(x))^2|X = x] + \mathbb{E}[(y(x) - \hat{y}(x))^2|X = x] \\ &= \sigma^2 + \mathbb{E}[(y(x) - \hat{y}(x))^2] \end{aligned} \quad (1.5)$$

dove  $\sigma^2$  è chiamato errore Bayes e

$$\begin{aligned} \mathbb{E}[(y(x) - \hat{y}(x))^2] &= (\mathbb{E}[\hat{y}(x)] - y(x))^2 + \mathbb{E}[(\hat{y}(x) - \mathbb{E}[\hat{y}(x)])^2] \\ &= Bias(\hat{y}(x))^2 + Var(\hat{y}(x)) \end{aligned} \quad (1.6)$$

Si ottiene così il compromesso *distorzione-varianza (bias-variance tradeoff)*:

$$\mathbb{E}[L(y, \hat{y})_{test}] = \sigma^2 + Bias(\hat{y}(x))^2 + Var(\hat{y}(x)) \quad (1.7)$$

Se la *distorzione* ha valori alti e la *varianza* bassi avremo un fenomeno di *adattamento insufficiente*, mentre se la *distorzione* ha valori bassi e la *varianza* alti avremo un *adatteamento eccessivo*.<sup>[?]</sup> Un modo per equilibrare questo compromesso è usare la *convalida incrociata (Cross-Validation)*, che consiste nel ripetere l'addestramento e il test dell'algoritmo ogni volta su sottoinsiemi scelti in maniera casuale.

Alcune varianti di *convalida incrociata* verranno analizzate in seguito.



## 1.2 Apprendimento supervisionato

Gli algoritmi di apprendimento supervisionato vengono utilizzati per risolvere problemi di classificazione e di regressione, che analizzeremo rispettivamente nel cap. 3 e nel paragrafo 1.2.1. Si parla di apprendimento supervisionato quando il dataset che si utilizza contiene delle variabili, una delle quali è un'etichetta. Dato un vettore di input  $x = (x_1, \dots, x_n)$ , ogni  $x_i$  è un vettore d-dimensionale di numeri rappresentanti una caratteristica, da questi dati si costruisce l'insieme di addestramento di cardinalità  $N$ :  $D = \{(x_i, y_i)\}_{i=1}^N$ , dove  $y = (y_1, \dots, y_m)$  è l'output dei risultati desiderati e  $y_i$  è l'etichetta. Lo scopo è di apprendere una regola generale che colleghi i dati in ingresso con quelli in uscita, in modo che l'algoritmo apprenda a classificare un esempio completamente nuovo, non contenente l'etichetta.

Se  $y_i$  è di tipo testuale si parla di classificazione, quando invece è di tipo numerico si parla di regressione. Se indichiamo con  $C$  il numero delle classi a cui può appartenere l'output:  $y \in \{1, \dots, C\}$ , se  $C = 2$  la classificazione sarà binaria (in questo caso spesso  $y \in \{0, 1\}$ ); se  $C > 2$  sarà multiclasse.

### 1.2.1 Regressione

La Regressione prevede il valore futuro di un dato, avendo noto il suo valore attuale. Un esempio è la previsione della quotazione delle valute o delle azioni di una società. Nel marketing viene utilizzato per prevedere il tasso di risposta di una campagna sulla base di un dato profilo di clienti; nell'ambito commerciale per stimare come varia il fatturato dell'azienda al mutare della strategia. Questo avviene costruendo una funzione che meglio si adatta ai punti che descrivono la distribuzione delle  $Y$  sulle  $X$ .

#### Regressione lineare

Preso un vettore  $x \in \mathbb{R}^n$  in input, l'algoritmo cerca di prevedere l'output:  $y \in \mathbb{R}$ . Dove  $y = f(x)$ , con  $f$  una funzione lineare:

$$y = \alpha + \beta x \quad (\text{retta di regressione})$$

Sia  $\hat{y}$  il valore di output che l'algoritmo prevede. Definiamo l'output come:

$$\hat{y} = \alpha + \beta x + \epsilon$$

dove  $\epsilon$  è la componente di errore.

Ora per identificare la retta che meglio si adatta ai punti, che descrivono la distribuzione delle  $y$  sulle  $x$ , bisogna stimare i valori dei parametri  $\alpha$  e  $\beta$ ,

tramite i dati osservati su un esempio.

Usiamo il metodo dei minimi quadrati (*least squares*), che minimizza l'errore  $\epsilon$ .

$$\begin{aligned}\sum_{i=1}^n (\hat{y}_i - y_i)^2 &= \sum_{i=1}^n (\hat{y}_i - (\alpha + \beta x_i))^2 \\ &= \sum_{i=1}^n (\hat{y}_i - \alpha - \beta x_i)^2 = \min\end{aligned}$$

Per trovare i valori di  $\alpha$  e  $\beta$  risolviamo il sistema:

$$\begin{cases} \frac{d(\sum_{i=1}^n (\hat{y}_i - \alpha - \beta x_i)^2)}{d\alpha} = 0 & (1.8) \\ \frac{d(\sum_{i=1}^n (\hat{y}_i - \alpha - \beta x_i)^2)}{d\beta} = 0 & (1.9) \end{cases}$$

Dalla (1.8) otteniamo:

$$\alpha = \mathbb{E}[y] - \beta \mathbb{E}[x]$$

e dalla (1.9):

$$\beta = \frac{\sum_{i=1}^n (x_i - \mathbb{E}[x])(y_i - \mathbb{E}[y])}{\sum_{i=1}^n (x_i - \mathbb{E}[x])^2} = \frac{Cov(x, y)}{Var(x)}$$

ottenendo così i valori dei parametri che cercavamo.

## 1.3 Apprendimento non supervisionato

Gli algoritmi di apprendimento non supervisionato vengono utilizzati per risolvere problemi di raggruppamento. All'algoritmo viene passato solo l'input:  $D = \{x_i\}_{i=1}^N$  e cerca una relazione tra i dati per capire se e come essi siano collegati tra di loro. Non contenendo alcuna informazione preimpostata, l'algoritmo è chiamato a creare una "nuova conoscenza" (*knowledge discovery*). A differenza del caso supervisionato, questo apprendimento non ha una classificazione o un risultato finale con il quale determinare se il risultato è attendibile, ma generalizza le caratteristiche dei dati e in base ad esse attribuisce ad un input un output: serve generalmente ad estrarre informazioni non ancora note, "creando" esso stesso delle classi in cui dividere i dati. Questa tecnica si chiama *clustering*.

## 1.4 Apprendimento per rinforzo

Gli algoritmi di apprendimento per rinforzo vengono utilizzati per risolvere problemi di regressione. Lo scopo di questo algoritmo è di realizzare un sistema in grado di apprendere ed adattarsi ai cambiamenti dell'ambiente in cui si trovano, attraverso la distribuzione di una "ricompensa" detta rinforzo, data dalla valutazione delle prestazioni. Questi algoritmi sono costruiti sull'idea che i risultati corretti dovrebbero essere ricordati, per mezzo di un segnale di rinforzo, in modo che diventino più probabili e quindi più facilmente riottenuti nelle volte future; viceversa se il risultato è errato, il segnale sarà una penalità, ovvero si avrà una probabilità più bassa legata a quel determinato output.[4]

## Capitolo 2

# Reti neurali

Le reti neurali sono i modelli di deep learning per eccellenza. Sono un sistema di elaborazione di informazioni ispirato al funzionamento del sistema nervoso umano. La rete è strutturata come un grafo aciclico orientato in un verso solo. I nodi sono raggruppati in strati (*layers*): il primo strato contiene i nodi di input  $x_1, \dots, x_n$ , ognuno di essi forma con lo strato successivo un grafo fortemente connesso, dove ad ogni arco è associato un peso  $w_i$ . L'ultimo strato contiene i nodi di output. Gli strati tra il primo e l'ultimo strato sono chiamati strati nascosti (*hidden layers*). La lunghezza complessiva del percorso conferisce profondità al modello, da cui deriva il nome dell'apprendimento: "deep learning". In base all'architettura scelta, esistono vari modelli di reti neurali; la scelta dell'architettura della rete è molto importante, poichè in base al numero di nodi usati per ogni strato e alla profondità, il costo computazionale cresce o diminuisce: per esempio la scelta di un'architettura poco profonda e con una elevata quantità di nodi per strato causa un costo computazionale elevato e un massiccio utilizzo della memoria.

Lo scopo di questa tesi è di trovare l'architettura ottimale per un algoritmo di classificazione di immagini considerando anche la geolocalizzazione dell'oggetto e del suo visualizzatore. Ritornando alle reti neurali, in generale, abbiamo detto che si ispirano al nostro sistema nervoso; vediamo brevemente in che modo:

Ogni neurone, nel nostro cervello, riceve un'intera serie di segnali da altri neuroni, li somma all'interno del suo corpo cellulare e, sulla base della somma, aggiusta la frequenza delle scariche da inviare alla cellula successiva. I neuroni ricevono sia segnali eccitatori, ovvero che tendono ad aumentare la loro frequenza di scarica, che segnali inibitori, che tendono invece a diminuirli, ma nonostante ricevano due tipi di segnali, ne emettono poi di un solo tipo. Analogamente: ogni neurone artificiale, rappresentato da un nodo, diventa attivo se la quantità totale di segnale che riceve supera la soglia

di attivazione, definita dalla cosiddetta funzione di attivazione. Se un nodo diventa attivo, emette un segnale che viene trasmesso lungo i canali di trasmissione fino all'altra unità a cui è collegato.

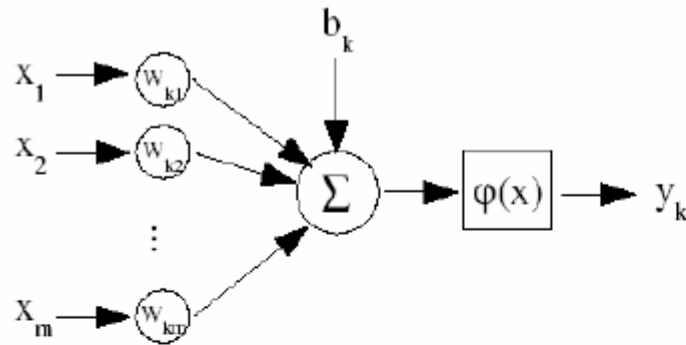


Figura 2.1: modello non lineare di un neurone artificiale

La figura 2.1, indipendentemente dal modello di rete utilizzato, mostra l'elaborazione eseguita da un neurone artificiale.

Siano  $x_1, \dots, x_n$  i dati in input, rappresentati dagli  $n$  nodi del primo strato, nel  $k$ -esimo neurone l'informazione viene elaborata come:

$$y_k = f(b_k + \sum_{i=1}^n w_{ki}x_i)$$

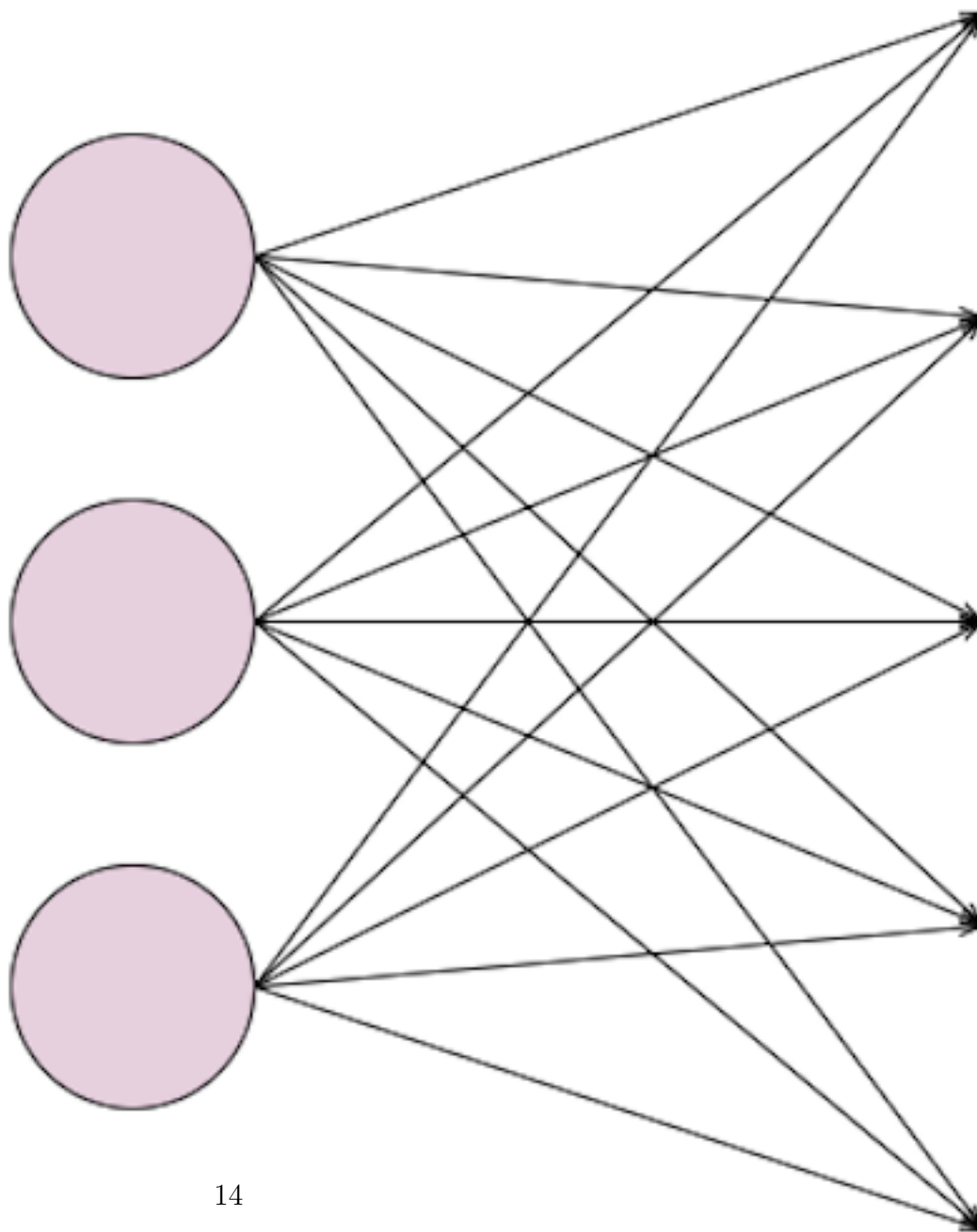
dove:

- $y_k$  è l'output generato dal neurone  $k$ ;
- $b_k$  è il valore soglia del neurone  $k$ ;
- $w_{ki}$  è il peso associato all'arco che collega il nodo  $i$ -esimo al neurone  $k$ ;
- $f(\cdot)$  è la funzione di attivazione.

La funzione di attivazione è una funzione non lineare, usata per limitare il valore dell' output, così da non consumare eccessiva memoria; vedremo in seguito il suo effettivo utilizzo.

Il motivo principale per cui vengono scelte le reti neurali è la possibilità di parallelizzare i calcoli.

Input



# Capitolo 3

## Classificazione

La Classificazione viene usata quando è necessario decidere a quale categoria appartiene un determinato dato. Per esempio, data una foto capire a quale categoria appartiene, in questa tesi vogliamo classificare immagini, più precisamente: capire a quale tipo di monumento corrisponde una determinata immagine.

Questo tipo di algoritmo deve specificare a quale delle  $k$  categorie appartiene un input. Crea una funzione  $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$ , quando  $y = f(x)$ , il modello assegna l'input descritto dal vettore  $x$  ad una categoria identificata dal codice numerico  $y$ . Esistono altre varianti dell'attività di classificazione, ad esempio, dove  $f$  genera una distribuzione di probabilità su classi.

Vediamo ora qualcuno degli algoritmi usati per classificare dati.

### 3.1 K-Nearest Neighbor

Questo è un algoritmo non parametrico, ovvero il numero di parametri cresce con la quantità di dati di addestramento. Dato un set di dati di addestramento  $(x_1, x_2, \dots, x_n)$ , corrispondenti ai risultati  $(y_1, y_2, \dots, y_n)$ , questo metodo, dato un nuovo punto  $z$ , cerca di prevedere la sua classe di appartenenza osservando, tra l'insieme di punti adiacenti, quelli a lui più vicini. Il numero di punti adiacenti da considerare dipende dal parametro  $k$ : si osservano le classi a cui appartengono i  $k$  punti più vicini e la classe più ricorrente sarà assegnata al punto  $z$ .

Vediamo ora come affrontare il problema:

Supponiamo di avere un set di dati che comprende  $N_k$  punti appartenenti alla classe  $C_k$  con  $N$  punti in totale, ovvero:  $\sum_k N_k = N$ . Se vogliamo classificare un nuovo punto  $x$ , disegniamo una sfera centrata su  $x$  contenente  $K$  punti qualsiasi, indipendentemente da come siano classificati. Vogliamo ora calco-

lare la funzione di probabilità di  $x$  che (essendo in un caso di classificazione i valori sono discreti) sarà la densità discreta:

$$P = \int_R p(x) dx \quad (3.1)$$

Supponiamo ora di aver raccolto un set di dati comprendente  $N$  osservazioni con probabilità uniforme  $p(x)$  di essere all'interno della regione  $R$ , quindi la probabilità di avere  $K$  punti all'interno di  $R$  sarà data dalla distribuzione binomiale:

$$P(X = K) = \frac{N!}{K!(N - K)!} P^K (1 - P)^{N - K}, \quad (3.2)$$

dove il valore atteso e la varianza sono date da:

$$\mathbb{E}[K] = NP \quad \text{Var}[K] = NP(1 - P) \quad (3.3)$$

Per  $N$  grandi, applichiamo il Teorema di De Moivre-Laplace alla eq.3.2 e otteniamo che la distribuzione binomiale si comporta come una distribuzione normale con stessa media e varianza della binomiale, perciò possiamo assumere:

$$K \simeq NP \quad (3.4)$$

Ora se assumiamo che la regione  $R$  sia sufficientemente piccola e che la densità di probabilità  $p(x)$  sia approssimativamente costante in  $R$ , da 3.1 otteniamo:

$$P \simeq p(x)V \quad (3.5)$$

dove  $V$  è il volume della sfera  $R$ . Combinando le eq. 3.4,3.5 otteniamo:

$$p(x) = \frac{K}{NV} \quad (3.6)$$

che fornisce le seguenti stime:

$$p(xjC_k) = \frac{K_k}{N_k V} \quad (3.7)$$

$$p(x) = \frac{K}{NV} \quad (3.8)$$

$$p(C_k) = \frac{N_k}{N} \quad (3.9)$$

che sostituite nel Teorema di Bayes otteniamo:

$$p(C_k j x) = \frac{p(xjC_k)p(C_k)}{p(x)} = \frac{K_k}{K} \quad (3.10)$$



Quindi per minimizzare la probabilità di errore di classificazione, bisogna assegnare al nuovo punto  $x$  la classe con probabilità più alta ovvero quando il valore  $\frac{K_k}{K}$  è massimo.

L'obiettivo del metodo è quindi chiaro: per classificare un nuovo punto, identifichiamo i  $K$  punti più vicini all'insieme di allenamento e quindi assegniamo il nuovo punto alla classe che ha il maggior numero di rappresentanti tra i  $K$  punti. Rimane solo la scelta della metrica da usare per calcolare la distanza tra i punti, solitamente viene usata la distanza Euclidea, ma questo dipende dal problema e dalla tipologia del dato da analizzare. Lo svantaggio di questo algoritmo è chiaro: il numero di distanze da calcolare aumenta con l'aumentare dell'insieme di addestramento. Oltre a rallentare il tempo di calcolo, si usa anche una considerevole quantità di memoria, per ovviare a questo si divide l'insieme di allenamento in sottoinsiemi di cardinalità  $n$ , dove  $n$  di solito è un divisore della cardinalità dell'insieme totale e prende il nome di *batch size*. Diminuendo il numero di dati per l'addestramento i calcoli e la memoria necessari diminuiscono notevolmente, in quanto i paragoni e i dati significativi da mantenere sono minori.

## 3.2 Alberi di decisione

# Capitolo 4

## TensorFlow

TensorFlow (TF) è una libreria software open source, sviluppata da Google, utilizzata per implementare l'apprendimento automatico e i sistemi di deep learning.

TensorFlow fornisce API native in linguaggio: Python, C/C++, Java, Go, e RUST. Noi useremo il linguaggio di programmazione Python.

In generale un algoritmo scritto in TF rispetta la seguente struttura:

1. Importare ed analizzare l'insieme di dati
2. Creare colonne di caratteristiche per descrivere i dati
3. Selezionare il tipo di modello
4. Provare il modello
5. Valutare l'efficacia del modello
6. Lasciare che il modello addestrato faccia previsioni (test)

che è in linea con la descrizione di un generico algoritmo di apprendimento descritto in (1.1). La vera innovazione di TF è come descrive il modello, poichè lo fa costruendo un grafico computazionale: *Data Flow Graph*. In questo grafico ogni nodo rappresenta l'istanza di un'operazione matematica, mentre ogni spigolo rappresenta un tensore, su cui vengono eseguite le operazioni.

**Definizione 6** Un *tensore* in TF è una matrice  $n$ -dimensionale di tipi di dati di base (es: *float32*, *int32*, *string*, ecc..). Viene chiamato *tf.Tensor* ed è descritto da tre parametri:

1. *grado (rank)*

2. *corpo (shape)*

3. *tipo (type)*

**1)** Il grado di un oggetto `tf.Tensor` è il suo numero di dimensioni. Come mostra la seguente figura, ogni rank in TensorFlow corrisponde a una diversa entità matematica:

**2)** Il corpo di un `tf.Tensor` è il numero di elementi in ogni dimensione. TF automaticamente deduce il corpo durante la costruzione del grafico. **3)** Il tipo è il tipo di dato a cui appartengono gli elementi del tensore.

I principali tipi di tensori sono:

- Variabili (*tf.Variable*): i parametri dell'algoritmo che verranno cambiati per ottimizzare l'algoritmo
- Costanti (*tf.constant*)
- Segnaposto (*tf.placeholder*): consentono di inserire dati e di creare operazioni per costruire il grafico computazionale, possono dipendere da altri dati ad esempio il risultato previsto di un calcolo. Possono essere usati più volte e non dare lo stesso risultato.
- Tensore sparso (*tf.SparseTensor*)

# Bibliografia

- [1] Arthur Samuel. [https://www.ibm.com/developerworks/community/blogs/jfp/entry/What\\_Is\\_Machine\\_Learning?lang=en](https://www.ibm.com/developerworks/community/blogs/jfp/entry/What_Is_Machine_Learning?lang=en), 1959.
- [2] Shehzad Noor Taus Priyo. <https://towardsdatascience.com/intro-to-deep-learning-d5caceedcf85>, 2017.
- [3] David H. Wolper and William G. Macready. No free lunch theorems for optimization, 1996.
- [4] Frank L Lewis. *Reinforcement learning and approximate dynamic programming for feedback control*. Wiley, 2013.
- [5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [6] K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning. MIT Press, 2012.