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

Dèsirèe Adiutori

Relatore

Prof. Luciano Teresi

Anno Accademico 2017/2018

Luglio 2018

Indice

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 puòaiutare 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? Perché 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 contemporaneamente e la scelta della loro cardinalità non è universale: 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'algortimo 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 tuti 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, ad esempio l'errore quadratico medio(MSE):

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

La nostra funzione di predizione dipenderá da dei parametri, rappresentati da un vettore w, lo scopoè 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. Gli algoritmi che risolverono il problema:

$$f(x^*) = \min_{x \in \mathbb{R}^n} f(x), \qquad f \in C^2 \tag{1.2}$$

Per risolvere problemi di questo tipo, spesso viene utilizzato il metodo di discesa del gradiente.[**?**]
La struttura generale di un metodo di discesa iterativo di minimizzazione è:

$$x_{k+1} = x_k + \beta_k d_k \tag{1.3}$$

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 di discesa, sarà tale che: $(d_k, \nabla f(x_k)) < 0$. Sia il passo che la direzione vanno scelti opportunamente ad ogni iterazione, in modo che $f(x_{k+1}) < f(x_k)$.
Per ogni passo imponiamo:

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

Mentre per la direzione: $d_k = -\nabla f(x_k)$. 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: $\| x_{k+1} - xk \| \leq m$, $\| \nabla f(x_{k+1}) \| \leq m\prime$ oppure $k > k_{max}$, dove m e $m\prime$ 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 (1.3). Si supponga*

1. *che l'insieme Σ_0 sia compatto;*
2. *che le direzioni d_k siano t.c. $\frac{(d_k, \nabla f(x_k))}{\|d_k \quad \| \|\nabla f(x_k)\|} \leq -\cos\theta$ per $k \in I$, con I insieme illimitato di indici;*
3. *che per $k \in I$, β_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 (1.3), con $d_k = -\nabla f(x_k)$.*
Allora, se i passi β_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:

$$L(y, \hat{y})_{test} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_{(test)} - y_{(test)})_i^2 \tag{1.4}$$

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:

$$\mathbb{E}[L(y, \hat{y})_{test}] = \mathbb{E}[(Y - \hat{y}(X))^2] \tag{1.5}$$

e definiamo la funzione dell'output effettivo come:

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

la quale avrá 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 ϵ indipendente da X tale che $\mathbb{E}[X] = 0$ e $Var(X) = \sigma^2$.
Possiamo quindi riscrivere l'equazione 1.5 come:

$$\begin{aligned} \mathbb{E}[L(y, \hat{y})_{test}] &= \mathbb{E}[(Y - \hat{y}(X))^2 | X = x] \\ &= \mathbb{E}[(Y - y(x))^2 | X = x] + \mathbb{E}[(y(x) - \hat{y}(x))^2 | X = x] \\ &= \sigma^2 + \mathbb{E}[(y(x) - \hat{y}(x))^2] \end{aligned} \tag{1.6}$$

dove σ^2 è chiamato errore Bayes e

$$\begin{aligned} \mathbb{E}[(y(x) - \hat{y}(x))^2] &= (\mathbb{E}[\hat{y}(x)] - y(x))^2 + \mathbb{E}[(\hat{y}(x) - \mathbb{E}[\hat{y}(x)])^2] \\ &= Bias(\hat{y}(x))^2 + Var(\hat{y}(x)) \end{aligned} \tag{1.7}$$

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

$$\mathbb{E}[L(y, \hat{y})_{test}] = \sigma^2 + Bias(\hat{y}(x))^2 + Var(\hat{y}(x)) \tag{1.8}$$

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.[**?**] Un modo per equilibrare questo compromesso ï¿½ 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 $x = (x_1, \cdots, x_n)$, 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 $y = (y_1, \cdots, y_m)$ 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: $y \in \{1, ..., C\}$, 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:

$$y = \alpha + \beta x \qquad (\textit{retta di regressione})$$

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

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

dove ϵ 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 α e β,

tramite i dati ossrvati su un esempio.
Usiamo il metodo dei minimi quadrati (*least squares*), che minimizza l'errore ϵ.

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

Per trovare i valori di α e β risolviamo il sistema:

$$\begin{cases} \dfrac{d(\sum_{i=1}^{n}(\hat{y}_i - \alpha - \beta x_i)^2)}{d\,\alpha} = 0 & (1.9) \\ \dfrac{d(\sum_{i=1}^{n}(\hat{y}_i - \alpha - \beta x_i)^2)}{d\,\beta} = 0 & (1.10) \end{cases}$$

Dalla (1.9) otteniamo:

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

e dalla (1.10):

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

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 rusultato è errato , il segnale sarà una penalità, ovvero si avrà una probabilità più bassa legata a quel determinato output.[4]

Capitolo 2

Reti neurali artificiali

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 orientato. I nodi sono raggruppati in strati (*layers*): il primo strato contiene i nodi di input $x_1, \cdots, x_n$, connessi con lo strato successivo, 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 nasconti (*hidden layers*). La lunghezza complessiva del percorso determina la profondità del 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, poiché in base al numero di nodi usati per ogni strato ed 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 ed 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 le geolocalizzazione dell'oggetto da riconoscere (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 diminuirla, ma nonostante ricevano due tipi di segnali, ne emettono poi di un solo tipo. Analogamente: ogni neurone aritificiale, rappresentato da un no-

do, 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.

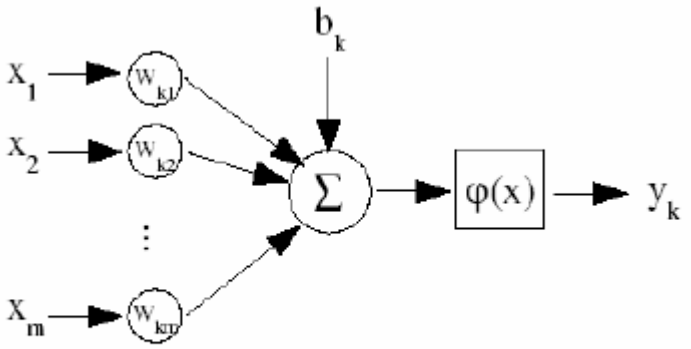

Figura 2.1: modello non lineare di un neurone artificiale

La figura 2.1, indipendentemente dal modello di rete utilizzato, mostra l'elaborazione eseguita da un neurone artificiale.
Siano $x_1, \cdots, x_n$ i dati in input, rappresentati dagli n nodi del primo strato, nel k-esimo neurone l'informazione viene elaborata come:

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

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.

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

2.1 Funzioni di attivazione

La funzione di attivazione è una funzione usata per normalizzare, quindi limitare, l'ampiezza dell' output, così da non consumare eccessiva memoria e di velocizzare il processo di calcolo .
Generalmente le reti neurali sono utilizzate per implementare funzioni complesse e le funzioni di attivazione non lineari consentono loro di approssimare funzioni arbitrariamente complesse. Le funzioni più utilizzate a tale scopo sono 3: Vediamo nel dettaglio quali sono.

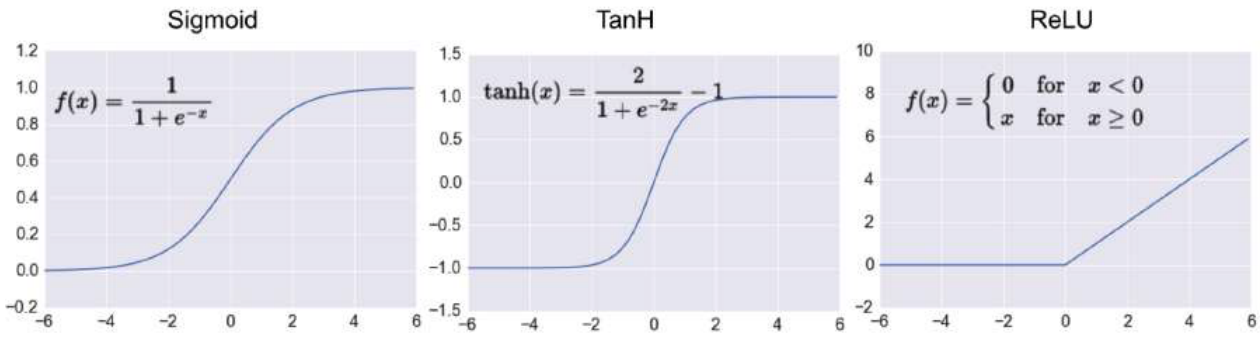

Figura 2.2: Grafici delle funzioni di attivazione più usate.

Sigmoid

La funzione Sigmoide viene usata sopartutto nei modelli in cui si deve prevedere la probabilità come output, poichè il suo codominio è l'intervallo: $(0, 1)$ ed ed è definita come: $S : \mathbb{R} \longrightarrow (0, 1)$, tale che:

$$S(x) = \frac{1}{1 + \exp^{-x}}$$

La Sigmoide è una funzione monotona e differenziabile, infatti la sua derivata:

$$\frac{dS}{dx} = \frac{\exp^{-x}}{(1+\exp^{-x})^2}$$

è continua, positiva e derivabile in ogni punto del dominio.

tanh

La tangente iperbolica: $\tanh : \mathbb{R} \longrightarrow (-1, 1)$,

$$\tanh(x) = \frac{2}{1+\exp^{-2x}} - 1$$

è simile alla Sigmoide, infatti sono legate dalla relazione:

$$\tanh(x) = 2 \cdot S(2x) - 1$$

ma ha un intervallo di output più ampio. Questo le consente di produrre anche output di segno negativo.
Anche la tangente iperbolica è una funzione monotona e differenziabile.
Purtroppo essendo funzioni limitate, per valori alti tendono ad un valore specifico: $\lim_{x\to+\infty} \tanh(x) = 1$, questo comportamento può portare ad una perdita di informazioni: se x corrisoponde ad un valore elevato e subisce una grande variazione, per la funzione invece avrà avuto una variazione minima. Lo stesso problema si presenta con le derivate, provocando la "scomparsa del gradiente"(*vanishing gradient*), fondamentale per approssimare al meglio la funzione; ma questo argomento verrà affrontato in 2.2.

ReLu

La funzione ReLu (Rectified Linear Unit) è la funzione piú utilizzata nel deep learning e, in particolare, nelle reti convoluzionali. È definita:

$$Relu : \mathbb{R} \longrightarrow [0, \infty)$$

$$Relu(x) = \begin{cases} x & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases}$$

Sia la funzione, che la sua derivata sono monotone. A differenza delle funzioni precedenti, la sua derivata è molto semplice da calcolare:

$$Relu(x)' = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases}$$

e non rischia di incorrere nella sparizione del gradiente. Si potrebbero avere problemi nell'origine, per la presenza di un punto angoloso, poiché la derivata è indefinita ma, per convenzione, viene definita uguale a zero.
Risulta immediato il motivo per cui sono molto utilizzate nelle reti formate da tanti strati: mappando i valori negativi in zero, permette di tenere solo i valori positivi, riducendo di molto il numero di neuroni attivati.

2.2 Addestramento di una rete

Come visto in 1.1, per addestrare un algoritmo si ha bisogno di una funzione di perdita e di un metodo per minimizzare l'errore di valutazione. L'addestramento di una rete neurale si basa sugli stessi principi:
si definisce una mappa

$$y = f(x, \theta)$$

e si cerca il valore del parametro θ che più accuratamente approssima la funzione. Bisogna quindi trovare dei parametri che minimizzano la funzione di perdita:

$$L[y, f(x, \theta)]$$

ovvero, trovare $\hat{\theta}$ tali che:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} L[y_i, f(x_i, \theta)] \right\}$$

Questo processo è necessario per approssimare una determinata funzione f^*, che descrive il comportamento della rete.
Durante l'addestramento, la rete viene provata piú volte, ogni volta con un inpur diverso, fino a che l'errore di addestramento è molto piccolo. In questa fase, ad ogni iterazione viene passato in input un insieme di addestramento, viene fissato un valore θ_0 iniziale, e tramite la funzione di perdita, si calcola l'errore di addestramento. In questo modo la rete ha il valore di quanto ciascun neurone di output sia lontano dal proprio valore atteso, e in che direzione (positiva o negativa).Per esempio, nel caso di riconoscimento dei numeri scritti a mano (MNIST):
Se il risultato atteso è 6, ci aspettiamo il valore 1 nel neurone 6 e 0 in tutti gli altri. Se al neurone 6 è associato il valore 0.7, allora la correzione da fare è di 0.3. Quindi la rete ripete l'addestramento aggiornando il peso corrispondente al neurone valutato con θ_1 e cosí via, fino a che il valore corrispondente al neurone 6 è molto vicino a 1. Viceversa, se il neurone 4 invece di 0 presenta

0.8, allora la correzione sarà -0.8 e si aggiorneranno i pesi relativi a questo neurone in modo da abbassarne drasticamente l'output.
L'algoritmo appena descritto a parole prende il nome di *back propagation*.

Algoritmo di back propagation

Consideriamo una rete neuronale composta da L strati, dove l'output è composto da un valore solamente. L'algoritmo di back propagation si divide in due fasi:

- propagazione in avanti
- propagazione all'indietro

Durante la propagazione in avanti, si calcolano tutti i valori dei nodi presenti nella rete a_i^k:

$$a_i^k = b_i^k + \sum_{j=1}^{r_{k-1}} w_{ji}^k \cdot o_j^{k-1} \tag{2.1}$$

dove:

- w_{ij}^k è il peso associato all'arco che collega il nodo i del k-esimo strato con il nodo j;
- b_i^k è il valore soglia del nodo i nel k-esimo strato;
- o_i^k è l'output del nodo i nel k-esimo strato;
- r_k è il numero di nodi presenti nel k-esimo strato.

Calcolato l'ultimo valore a^L, corrispondente all'output della rete, inizia la seconda fase. Per calcolare il gradiente della funzione di perdita $L[y, f(x, \theta)]$, si applica la regola della catena per il calcolo e poiché $\theta = (W^1, W^2, \cdots, W^{L-1})$, abbiamo:

$$\frac{\partial L}{\partial w_{ij}^k} = \frac{\partial L}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k}. \tag{2.2}$$

Ora definiamo il punto di forza di questo algoritmo, che consiste nell'introdurre una quantitá di costo (anche chiamata errore):

$$\delta_j^k \equiv \frac{\partial L}{\partial a_j^k} \tag{2.3}$$

attraverso la quale si calcolano le derivate parziali in modo iterativo, ripercorrendo la rete all'indietro.
Dall'eq 2.1, ricaviamo:

$$\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{l=0}^{r_{k-1}} w_{lj}^k o_l^{k-1}\right) = o_i^{k-1} \tag{2.4}$$

che sostituita nell'eq 3.4 otteniamo:

$$\frac{\partial L}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} \tag{2.5}$$

Partendo dallo strato dell'output, considerando come funzione di perdita:

$$L = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(g_o(a^L) - y)^2,$$

$$\delta^L = (g_o(a^L) - y)g_o'(a^L) = (\hat{y} - y)g_o'(a^L). \tag{2.6}$$

dove g_o è la funzione di perdita per l'output. Quindi la derivata parziale della funzione di perdita è:

$$\frac{\partial L}{\partial w_{iL-1}^L} = \delta^L o_i^{L-1} \tag{2.7}$$

Per gli strati nascosti invece abbiamo:

$$\delta_j^k = \sum_{l=1}^{r_{k+1}} \frac{\partial L}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{a_j^k} = \sum_{l=1}^{r_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{a_j^k}, \tag{2.8}$$

Sostituendo in 2.8 la 2.1 e denotando con g la funzione di perdita dello strato nascosto, si ottiene la formula di propagazione all'indietro:

$$\delta_j^k = g'(a_j^k) \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l k + 1 \tag{2.9}$$

e la derivata parziale della funzione di perdita rispetto ai pesi degli strati nascosti w_{ij}^{k+1} , $1 \leq k < L$, si ottiene con:

$$\frac{\partial L}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = g'(a_j^k) o_i^{k-1} \sum_{l=1}^{r_{k+1}} w_{jl}^{k+1} \delta_l k + 1 \tag{2.10}$$

Ora puó avvenire l'aggiornamento dei pesi tramite il metodo SGD (Stochastic Gradient Descent), ovvero in formule:

$$w_{ij}^{k+1} = w_{ij}^k + \eta \frac{\partial L}{\partial w_{ij}} \tag{2.11}$$

dove $\eta \in (0, 1]$ rappresenta il fattore di apprendimento (*learning rate*).
La scelta del fattore di apprendimento influenza molto il comportamento dell'algoritmo, infatti se scegliamo valori troppo piccoli, la convergenza sarà lenta, mentre se scegliamo valori troppo grandi si rischia di avere una rete instabile con comportamento oscillatorio.
Un metodo semplice per incrementare il fattore di apprendimento, senza il rischio di rendere la rete instabile, è quello di modificare la regola di aggiornamento inserendo un ulteriore parametro α, detto momento nell'equazione 2.11:

$$w_{ij}^{k+1} = \alpha w_{ij}^{k} + \eta \frac{\partial C}{\partial w_{ij}}. \tag{2.12}$$

Se si espande ricorsivamente la formula si ottiene:

$$w_{ij}^{k+1} = \eta \sum_{l=1} k + 1 \alpha_{k+1-l} \frac{\partial L}{\partial w_{ij}^{l}} \tag{2.13}$$

Inserendo il momento si ha il vantaggio che se la derivata parziale tende a mantenere lo stesso segno su iterazioni consecutive, grazie alla sommatoria, l'aggiornamento produce valori piú ampi e quindi tende ad accelerare nelle discese del gradiente. Se invece la derivata ha segni opposti ad iterazioni consecutive, la sommatoria tende a diminuire l'ampiezza dell'aggiornamento, in questo modo non si corre il rischio di avere dei loop infiniti nel caso di minimi locali della funzione d'errore.
A tal proposito generalmente i criteri di arresto dell'algoritmo possono essere:

- $\|\nabla L\| < \epsilon$
- $L[y, f(x, \theta)] = 0$;
- $L_i[y, f(x, \theta_i)] - L_j[y, f(x, \theta_j)] << \epsilon'$

dove L_i e L_j sono due epoche consecutive. Un'epoca corrisponde alle due fasi di propagazione necessarie per un aggiornamento dei pesi.

2.3 Reti Deep Feed-forward

Le reti Feed-forward sono le reti neurali profonde con la struttura più semplice, composte da almeno uno strato nascosto. La loro struttura è rappresentata da un grafo aciclico diretto in un'unica direzione, dove ogni nodo di uno strato è connesso con tutti i nodi dello strato successivo e nessun nodo è connesso con un nodo appartanente allo stesso strato.

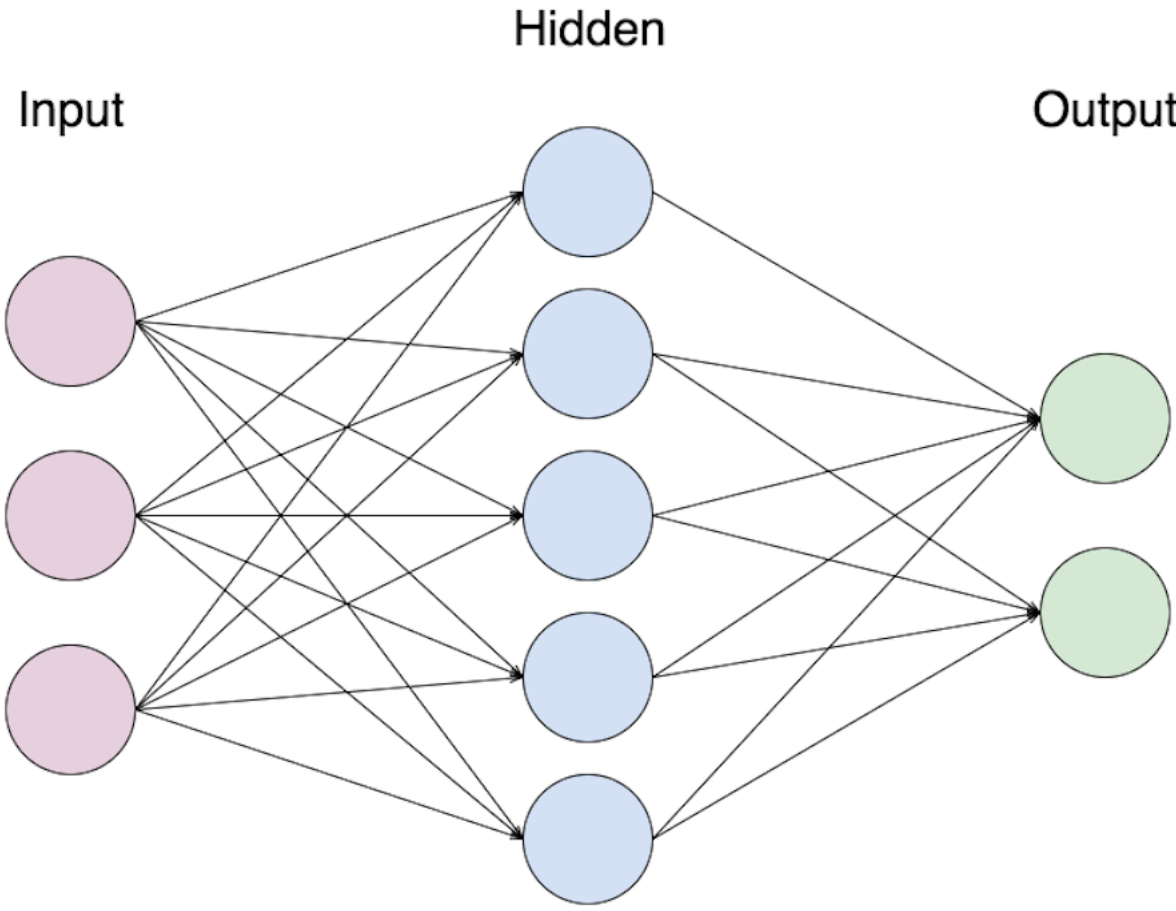

Figura 2.3: modello di rete deep feedforward con uno strato

2.4 Reti Convoluzionali

2.5 ImageNet

In Internet oramai sono presenti milioni di milioni di immagini e di video, usati per i modelli e gli algoritmi piï¿½ sofisticati e robusti, per aiutare gli utenti ad indicizzare, recuperare, organizzare e interagire con questi dati. Un esempio banale ï¿½ la ricerca dell'immagine di un cane su Google: se noi volessimo ricercare una razza canina in particolare, come fa l'algoritmo a farci visualizzare un cane di razza Labrador piuttosto che Beagle? Tramite categorie e sottocategorie. Ma esattamente come tali dati possono essere utilizzati e organizzati ï¿½ un problema che non ï¿½ ancora stato risolto.
Di solito i motori di ricerca possono trovare una determinata immagine solo se il testo inserito corrisponde al testo con cui ï¿½ stato etichettato. Ma le

etichette possono essere inaffidabili, inutili (nel caso di dialetti e nomignoli) o semplicemente inesistenti. Si vuole quindi cercare un metodo per far imparare alle macchine come riconoscere oggetti simili senza etichetta, rendendo possibile un notevole aumento dell'accuratezza del riconoscimento.
Si ï¿½ creato cosï¿½ **ImageNet**: un grande database visivo contenente oltre 14 milioni di immagini etichettate, progettato per l'utilizzo nella ricerca di software per il riconoscimento di oggetti visivi.
ImageNet si basa sulla struttura gerarchica fornita da un altro database: *WordNet*. WordNet ï¿½ un database semantico-lessicale per la lingua inglese elaborato dal linguista George Armitage Miller presso l'Universitï¿½ di Princeton, che si propone di organizzare, definire e descrivere i concetti espressi dai vocaboli. L'organizzazione del lessico si avvale di raggruppamenti di termini con significato affine, chiamati *"synset"* (dalla contrazione di synonym set), e del collegamento dei loro significati attraverso diversi tipi di relazioni chiaramente definite. All'interno dei synset le differenze di significato sono numerate e definite.[**?**]
Un'immagine digitale ï¿½ formata da diversi quadratini disposti in modo regolare, su una griglia di punti equidistanti, questi quadratini sono detti *pixel (picture elements)*. I pixel presenti in um'immagine ne determinano la dimensione e la risoluzione, per esempio un'immagine di dimensione $320x240$ pixel indica che sono presenti 240 pixel orizzontali e 320 verticali. Piï¿½ i pixel sono numerosi, e quindi piï¿½ piccoli e fitti, piï¿½ la risoluzione ï¿½ alta. In ogni pixel risiede un'informazione espressa in bit riguardante (generalmente) il colore: un pixel da 1 bit puï¿½ avere solo 2^1colori, mentre uno da 1 byte (8bit) puï¿½ rappresentare $2^8 = 256$ colori. Il numero di colori possibili ï¿½ detto anche profonditï¿½. Le rappresentazioni delle immagini a colori variano a seconda dei campi di colore che si usano, solitamente, ogni campo viene rappresentato da 1 byte e rappresenta il livello di intensitï¿½ dei colori fondamentali. Il modello piï¿½ utilizzato ï¿½ quello RGB, dove per ogni pixel vengono utilizzati 3byte:

- 1 byte per la componente rossa (R)
- 1 byte per la componente verde (G)
- 1 byte per la componente blu (B)

ma ne esistono anche altri come, ad esempio, CMYK che considera come colori fondamentali: ciano, magenta, giallo e nero.
Tipicamente nel data set di ImageNet la dimensione media delle immagini è di circa $400x350$

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, piú precisamente: capire a quale tipo di monumento corrispone una determinata immagine.
Questo tipo di algoritmo deve specificare a quale delle k categorie appartiene un input. Crea una funzione $f : \mathbb{R}^n \to \{1, \cdots, k\}$, 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'attivitï¿½ di classificazione, ad esempio, dove f genera una distribuzione di probabilitï¿½ su classi.
Vediamo ora qualcuno degli algoritmi usati per classificare dati.

3.1 K-Nearest Neighbor

Questo ï¿½ un algoritmo non parametrico, ovvero il numero di parametri cresce con la quantitï¿½ di dati di addestramento. Dato un set di dati di addestramento $(x_1, x_2, \cdots, x_n)$, corrispondenti ai risultati $(y_1, y_2, \cdots, y_n)$, 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 piï¿½ vicini. Il numero di punti adiacenti da considerare dipende dal parametro k: si osservano le classi a cui appartengono i k punti piï¿½ vicini e la classe piï¿½ ricorrente sarï¿½ 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 probabilitï¿½ di x che (essendo in un caso di classificazione i valori sono discreti) sarï¿½ la densitï¿½ discreta:

$$P = \int_R p(x)dx \tag{3.1}$$

Supponiamo ora di aver raccolto un set di dati comprendente N osservazioni con probabilitï¿½ uniforme $p(x)$ di essere all'interno della regione R, quindi la probabiltï¿½ di avere K punti all'interno di R sarï¿½ data dalla distribuzione binomiale:

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

dove il valore atteso e la varianza sono date da:

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

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, perciï¿½ possiamo assumere:

$$K \simeq NP \tag{3.4}$$

Ora se assumiamo che la regione R sia sufficientemente piccola e che la densitï¿½ di probabilitï¿½ $p(x)$ sia approssimativamente costante in R, da 3.1 otteniamo:

$$P \simeq p(x)V \tag{3.5}$$

dove V ï¿½ il volume della sfera R. Combinando le eq. 3.4,3.5 otteniamo:

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

che fornisce le seguenti stime:

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

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

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

che sostituite nel Teorema di Bayes otteniamo:

$$p(C_k jx) = \frac{p(xjC_k)p(C_k)}{p(x)} = \frac{K_k}{K} \tag{3.10}$$

Quindi per minimizzare la probabilitï¿½ di errore di classificazione, bisogna assegnare al nuovo punto x la classe con probabilitï¿½ piï¿½ alta ovvero quando il valore $\frac{K_k}{K}$ ï¿½ massimo.
L'obiettivo del metodo ï¿½ quindi chiaro: per classificare un nuovo punto, identifichiamo i K punti piï¿½ 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 algortimo ï¿½ 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 quantitï¿½ di memoria, per ovviare a questo si divide l'insieme di allenamento in sottoinsiemi di cardinalitï¿½ n, dove n di solito ï¿½ un divisore della cardinalitï¿½ 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) ï¿½ 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 ï¿½ in linea con la descrizione di un generico algoritmo di apprendimento descritto in (1.1). La vera innovazione di TF ï¿½ come descrive il modello, poichï¿½ 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 ï¿½ una matrice n-dimensionale di tipi di dati di base (es: float32, int32,string, ecc..). Viene chiamato tf.Tensor ed ï¿½ descritto da tre parametri:*

1. *grado (rank)*

2. *corpo (shape)*

3. *tipo (type)*

1) Il grado di un oggetto tf.Tensor ï¿½ il suo numero di dimensioni. Come mostra la seguente figura , ogni rank in TensorFlow corrisponde a una diversa entitï¿½ matematica:

2)Il corpo di un tf.Tensore ï¿½ il numero di elementi in ogni dimensione. TF automaticamente deduce il corpo durante la costruzione del grafico. **3)** Il tipo ï¿½ il tipo di dato a cui appartongono 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 piï¿½ 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, 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.