POLITECNICO DI TORINO

Corso di Laurea
in Matematica per le Scienze dell'Ingegneria

Reti Neurali

Trattazione sull'architettura, il funzionamento e l'applicazione delle reti neurali al deep learning

Relatrice
prof.ssa Valeria Chiadò Piat

Candidato
Riccardo Kiefer

Anno Accademico 2020-2021

Ai miei nonni,

Sommario

In un mondo ed una società il cui grado di complessità è in costante aumento le reti neurali artificiali offrono un modello per studiare, cercare di comprendere e simulare quanto forse di più complesso e misterioso sia mai apparso sul nostro pianeta: il cervello umano. Migliaia di anni di evoluzione hanno plasmato il funzionamento di questa intricata rete di neuroni che ogni giorno permettono all'uomo di percepire l'ambiente circostante, ricavare informazioni da esso, adattarsi a situazioni differenti nonché formulare un pensiero complesso e la straordinaria capacità di astrarre. Operazioni che sembrano spontanee ad ognuno di noi ma di cui ignoriamo la profonda complessità.
Le reti neurali, da cui il nome, in qualche modo cercano di riprodurre la plasticità del nostro cervello al fine di creare quella che può essere definita un'intelligenza artificiale. Il vantaggio di questo strumento è la possibilità di apprendere tramite "esperienza", una facoltà che sinora si pensava relegata al mondo biologico. Vedremo infatti come una rete neurale possa essere sottoposta ad un processo di apprendimento che l'alleni a produrre una risposta rispetto a stimoli completamente nuovi riconoscendo gli stessi pattern che le sono stati proposti in fase di insegnamento. Una capacità che le rende uno strumento estramemente elastico con un elevato grado di accuretezza. Negli ultimi anni infatti esse hanno trovato sempre più eterogenea applicazione in ambiti come economia, cybersecurity, data science, biologia e molti altri.

Indice

Elenco delle figure

Il vero problema non è se le macchine sappiano pensare ma se gli uomini lo facciano.

[Burrhus F. Skinner]

Capitolo 1

Introduzione generale

1.1 Reti Neurali

Le reti neurali artificiali sono la struttura base nell'ambito dell'apprendimento automatico e quindi dell'intelligenza artificiale. Come suggerisce il nome si tratta di un tipo di modello computazionale composto da "neuroni" artificiali ispirato vagamente dalla semplificazione di una rete neurale biologica. È possibile determinare delle connessioni tra questi neuroni fornendo in input come segnale di attivazione l'output di altri neuroni adiacenti in modo da determinare una trasmissione del segnale all'interno della rete. Lo scopo consiste nell'allenare la rete neurale in modo che risponda in maniera corretta agli stimoli che le vengono proposti in fase di apprendimento e generi un output corretto. Nel capitolo due vedremo bene nel dettaglio come è possibile modellizzare una rete neurale descrivendone l'architettura a partire dai singoli "neuroni" che la compongono ed analizzeremo cosa sia e come funzioni il processo di apprendimento.

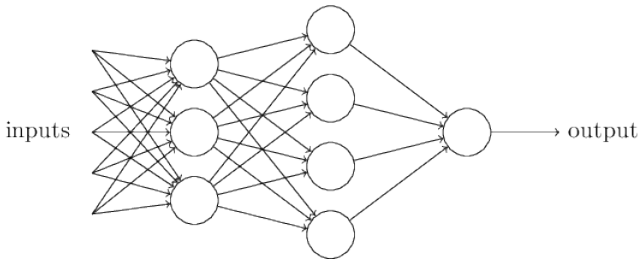


Figura 1.1: Rappresentazione grafica di una rete neurale.

1.2 Deep Learning

Il deep learning (apprendimeto profondo in inglese) consiste in una particolare applicazione delle rete neurali a problemi dall'elevato grado di complessità ed astrazione come il

riconoscimento vocale o l'identificazione di volti. Questo risultato viene ottenuto utilizzando reti multistrato dove l'informazione in input viene analizzata sempre più in profondità mentre viaggia all'interno del network costruendo un livello di gerarchia nei concetti da interpretare prima non possibile. Come vedremo la difficoltà per questo tipo di reti consiste nel generare processi di apprendimento veloci e dunque utilizzabili nelle applicazioni pratiche. Sono state quindi sviluppate nuove idee ed architetture che migliorino la capicità di apprendimento delle *deep neural network*. Ad oggi sono lo strumento più utilizzato per produrre esempi primordiali di intelligenza artificiale Nielsen [2019]. Vedremo i dettagli nel capitolo tre.

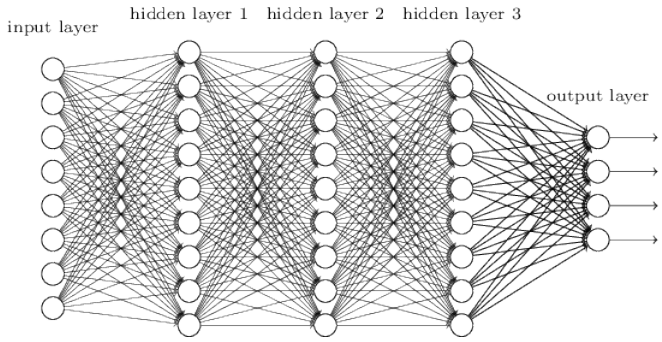


Figura 1.2: Rappresentazione grafica di una rete neurale multistrato.

1.3 Obbiettivi

L'obbiettivo principale di questa trattazione è offrire una breve, seppur abbastanza dettagliata, descrizione dell'architettura e del ruolo svolto dalle reti neurali nel processo di apprendimento automatico (*machine learning*) ed apprendimento profondo (*deep learning*). Il campo delle reti neurali è molto vasto ed alcuni concetti risultano estremamente difficili. Durante la stesura di questa tesi triennale sono dunque rimasto per lo più in superficie e ho cercato di offrire una chiave di lettura sui concetti principali piuttosto che un'analisi approfondita e specifica di ogni tipo di rete neurale e dello loro applicazioni più recenti. L'obbiettivo era rendere il lettore consapevole dell'argomento e guidarlo nel comprendere la logica di base, in particolare riguardo:

1. *L'architettura ed il funzionamento dei network neurali*: partire dalle unità fondamentali che le compongono: i neuroni artificiali, per arrivare a descrivere la loro organizzazione all'interno della rete.

2. *Il processo di apprendimento e le funzioni che lo regolano*: cosa significa per una rete neurale "imparare" e quali sono i metodi numerici usati nell'implementazione di algoritmi di apprendimento autonomo.

3. *Deep Learning*: come è possibile utilizzare le reti neurali per risolvere problemi dall'elevato grado di complessità e gli albori dell'intelligenza artificiale.

Capitolo 2

Reti Neurali

2.1 Neuroni artificiali

Un neurone artificiale può essere considerato l'unità logica all'interno della rete neurale. È in grado di produrre un output in base agli stimoli cui viene sottoposto in input.

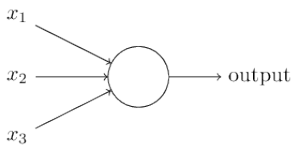


Figura 2.1: Modellizzazione grafica di un neurone artificiale

2.1.1 Perceptrons

Il perceptron è il modello più semplice di neurone artificiale sviluppato tra gli anni 50 e 60 nel Novecento dallo scienziato Frank Rosenblatt. Nonostante questo tipo di neuroni sia ormai superato è un utile strumento per capirne il funzionamento. Un perceptron prende in input diversi stimoli binari e produce un singolo output: 1 o 0.
Rosenblatt ideò un utile stratagemma al fine di determinare l'output: introdusse dei numeri reali chiamati pesi indicati con la lettera minuscola w (dall'inglese weights) che esprimono l'importanza del rispettivo input rispetto l'output. L'output del neurone risulta 1 o 0 a seconda che la somma del prodotto input per il rispettivo peso sia maggiore o minore uguale di un certo valore soglia (thresold).

$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq threshold \\ 1, & \text{if } \sum_j w_j x_j > threshold \end{cases} \tag{2.1}$$

È possibile semplificare ulteriormente questa notazione considerando la sommatoria come un prodotto scalare ed introducendo un nuovo termine b (bias $\doteq$ -threshold)

$$output = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases} \tag{2.2}$$

Si può pensare al bias come ad una stima di quanto facilmente il perptron emetta un output non nullo. Un neurone con bias molto grande restituirà come output 1 molto più facilmente di un neurone con bias molto negativo. Agendo sul valore dei pesi o dei bias è possibile settare il tipo di risposta desiderata dal neurone .
Come vedremo sarà possibile escogitare algoritmi di apprendimento automatico in grado di regolare autonomamente questi due valori in un network di neuroni artificiali. Affinché ciò sia possibile però è necessario utilizzare un diverso tipo di neurone che risponda in maniera più regolare agli stimoli esterni.

2.1.2 Sigmoid Neurons

Per avere un processo di apprendimento efficace è necessario elaborare un tipo di rete neurale dove piccole modifiche dei pesi o dei bias comportano piccoli cambiamenti nell'output in modo da avere un comportamento prevedibile ad avvicinarci "a piccoli passi", variando leggermente pesi e bias ad ogni ciclo di insegnamento, verso un output del network sempre più corretto.
Se la nostra rete fosse composta da Perceptrons questo risulterebbe impossibile in quanto essi hanno solo due possibili valori di output ed una piccola modifica dei pesi o dei bias potrebbe capovolgere nettamente l'output portandolo da 0 a 1. Questa mancanza di sensibilità dei Perceptrons potrebbe alterare il comportamento dell'intera rete in maniera caotica. Questo renderebbe complicato modificare gradualmente la rete neurale in modo che si avvicini al risultato voluto.
L'obbiettivo quindi é quello di linearizzare l'output, questo è possibile utilizzando un nuovo tipo di neurone il Sigmoid Neuron. I Sigmoid Neurons sono strutturalmente uguali ai Perceptrons ma con una sostanziale differenza che li rende adatti per essere utilizzari in un algoritmo di apprendimento. Quest'ultimi ammettono in input qualsiasi numero reale tra 0 ed 1 anziché i soli due valori binari dei primi. L'output infatti risulta essere $\sigma(wx + b)$ dove σ rappresenta la funzione di Sigmoid definita come:

$$\sigma(z) \doteq \frac{1}{1 + e^{-z}} \tag{2.3}$$

In questo contesto σ prende talvolta il nome di funzione di attivazione del neurone.

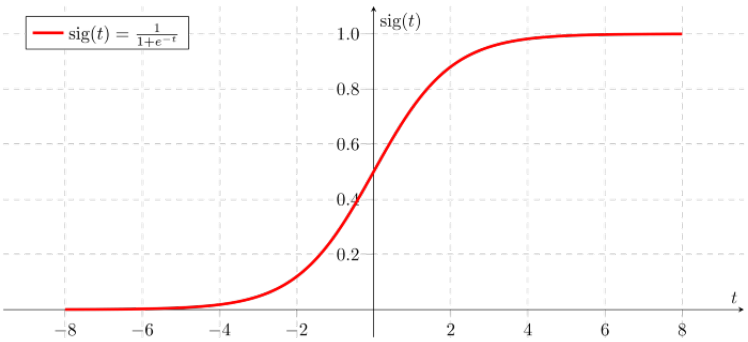


Figura 2.2: Funzione di Sigmoid.

Osservando il grafico di σ notiamo che per valori molto positivi o molto negativi di $z \equiv w \cdot x + b$ l'output del Sigmoid Neuron risulta essere lo stesso del Perceptron (che invece può essere schematizzato come una funzione a gradino). Tuttavia le proprietà di regolarita della funzione di Sigmoid garantiscono che piccoli variazioni delle variabili indipendenti producano piccole variazioni di quella dipendente. Differenziando:

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b \tag{2.4}$$

In conclusione i Sigmoid Neurons pur mantenendo lo stesso comportamento qualitativo dei Perceptrons rendono molto più semplice capire come modificare pesi e bias andrà ad influire sull'output finale.

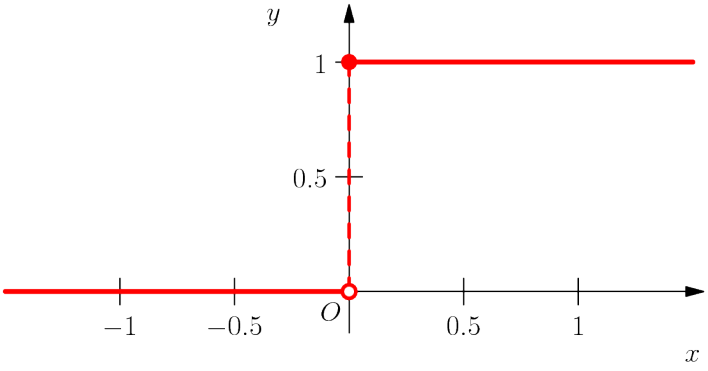


Figura 2.3: Funzione a gradino.

2.2 Architettura delle reti neurali

Dopo aver visto il funzionamento dei singoli neuroni analizziamo ora nel dettaglio come possano essere organizzati all'interno di una rete neurale. Una rete neurale sostanzialmente è un network di molti neuroni artificiali collegati tra loro tramite i pesi w. Usualmente viene rappresentata come una serie di neuroni ordinati in più colonne detti strati (layers in inglese). In questa stesura ci occuperamo solamente di reti neurali in cui l'output di un layer viene usato come input del layer succesivo, questo tipo di network viene denominato *feedforward neural networks*. Non ci occuperemo quindi di reti che presenato loop interni: le cosiddette *recurrent neural networks*.

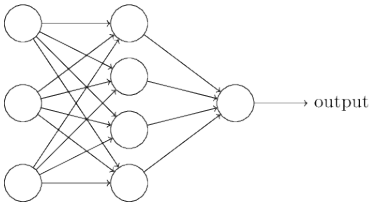


Figura 2.4: Rete neurale elementare.

I neuroni (input neurons) del primo strato hanno il compito di pesare le informazioni ricevute in input dall'esterno della rete. Questo strato viene infatti chiamato input layer. Lo strato più a destra invece contiene i neuroni di output (output neurons) della rete ovvero quelli settati in modo tale da restituire l'output dell'intera rete. Infine tutti gli strati nel mezzo costituiscono gli hidden layers. L'informazione viaggiando da sinistra verso destra all'interno della rete può essere analizzata ad un livello sempre più complesso ed astratto.
Mentre il design degli strati di input ed output risulta spesso intuitivo non è possibile riassumere la costruzione degli hidden layers basandosi su regole generali. I ricercatori di reti neurali hanno quindi sviluppato numerosi ipotesi di design in maniera euristica per aiutare le persone ad ottenere il comportamento voluto dalle loro reti Nielsen [2019].

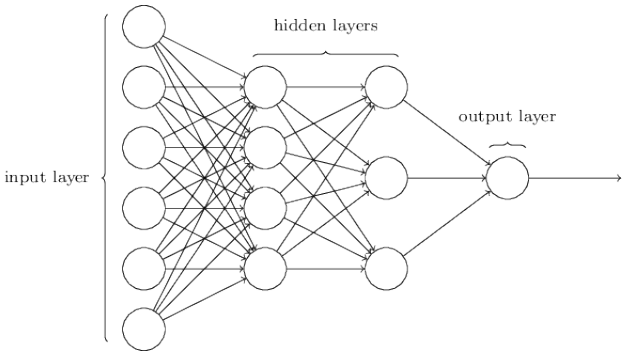


Figura 2.5: Rete neurale complessa.

2.3 Apprendimento automatico

Una volta compreso il funzionamento dei neuroni artificiali e l'architettura dei network neurali occupiamoci ora di capire cosa si intende per apprendimento automatico di una rete neurale e quali siano le equazioni che lo descrivono.
L'apprendimento di rete neurale è divisa in due fasi. La prima serve a regolare in maniera corretta i pesi e bias, questo risultato viene ottenuto utilizzando una grande mole di dati come input, chiamati *training data set*. L'obbiettivo è quello di sviluppare un algortimo in grado di regolare autonomamente pesi e bias in modo tale che l'output della rete neurali approssimi al meglio possibile il valore corretto per ogni dato di allenamento in input. La seconda fase invece consiste nel verificare l'efficienza della rete, a tal proposito si utilizza un insieme di dati, detti *test data set*, diverso da quello precedente e si misura la percentuale di accuratezza ovvero quanti dati hanno restituito il corretto output rispetto al totale.
Vediamo ora nello specifico quali sono i metodi più usati per settare una rete neurale in fase di apprendimento.

2.3.1 Metodo di discesa del gradiente

Il metodo di discesa del gradiente è un ben noto algoritmo dell'analisi numerica per minimizzare il valore di una data funzione.
Data una funzione sufficientemente regolare in due incognite (ν_1, ν_2) sappiamo dall'analisi che piccole variazione delle variabili indipendenti comportano una variazioni della variabile dipendente secondo la legge:

$$\Delta C \approx \frac{\partial C}{\partial \nu_1}\Delta\nu_1 + \frac{\partial C}{\partial \nu_2}\Delta\nu_2 \tag{2.5}$$

Introducendo il vettore gradiente:

$$\nabla C \equiv (\frac{\partial C}{\partial \nu_1}, \frac{\partial C}{\partial \nu_2})^T \tag{2.6}$$

L'equazione (2.5) può essere riscritta come:

$$\Delta C \approx \nabla C \cdot \Delta\nu \tag{2.7}$$

Volendo minimizzare la funzione C è importante scegliere $\Delta\nu$ in modo che ΔC sia negativo. Supponiamo quindi di scegliere:

$$\Delta\nu = -\eta\nabla C \tag{2.8}$$

Dove η è un paramentro positivo chiamato in inglese *learning rate*. Questa scelta di $\Delta\nu$ ci garantisce che $\Delta C \leq 0$ in quanto $\Delta C \approx -\eta\nabla C \cdot \nabla C = -\eta \left\|\nabla C\right\|^2$ e quindi che C sia sempre in decremento. Usiamo quindi l'equazione (2.8) per calcolare il valore di $\Delta\nu$ e poi utilizziamo questo valore per calcolare una nuova "posizione" delle variabili indipendenti:

$$\nu \to \nu' = \nu - \eta\nabla C \tag{2.9}$$

in modo che $C(\nu_1', \nu_2') < C(\nu_1, \nu_2)$. Eseguendo iterativamente questi passaggi si converge infine ad un minimo locale della funzione.
Riassumendo: l'algoritmo di discesa del gradiente funziona calcolando ripetutamente il gradiente ∇C e poi utilizzando questo valore per muoversi in direzione opposto nel dominio della funzione facendola decrescere sino ad arrivare ad un minimo locale per C.

Vediamo ora come questo metodo possa essere utilizzato nel contesto dell'apprendimento automatico per una rete neurale. Prima di tutto è necessario introdurre una funzione che quantifica quanto bene l'output della rete stia approssimando il valore esatto dei training data. Definiamo una funzione di costo, detta funzione quadratica di costo:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \tag{2.10}$$

In (2.10) le varie grandezze hanno i seguenti significati:

1. w l'insieme dei pesi del network;
2. b l'insieme dei bias;
3. n il numero totale degli input;
4. a il vettore dell'output dato x;
5. *y(x)* il valore esatto dell'output dato x;

La somma è su tutti gli input di allenamento x.
L'idea è quella di usare il metodo di discesa del gradiente per trovare pesi w_k e bias b_l che minimizzino l'equzione (2.10). In altre parole la nostra "posizione" nel dominio della funzione ha ora componenti w_k e b_l mentre le corrispondendi componenti del gradiente ∇C sono $\partial C/\partial w_k$ e $\partial C/\partial b_l$. Scrivendo quindi la regola di discesa del gradiente componente per componente abbiamo:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k} \tag{2.11}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{2.12}$$

Metodo stocastico di discesa del gradiente

Torniamo un momento alla nostra funzione quadratica di costo (2.10). Questa può essere riscritta nella forma $C = \frac{1}{n} \sum_x C_x$ che altro non è che una media delle funzioni costo $C_x \equiv \frac{\|y(x)-a\|^2}{2}$ per ogni singolo input x di apprendimento. Sostanzialmente per calcolare il gradiente ∇C dobbiamo trovare i gradienti ∇C_x separatamente per poi farne la media ($\nabla C = \frac{1}{n} \sum_x \nabla C_x$). Sfortunatamente quando il campione di apprendimento è molto numeroso questo processo può richiedere tempo compromettendo la velocità di apprendimento della rete neurale.

Per incrementare la velocità di apprendimento si è deciso di fare ricorso al *metodo stocastico di discesa del gardiente*. L'idea è quella di ridurre il campione di apprendimento selezionando un piccolo sottoinsieme di elementi scelti casualmente per poi stimare ∇C facendo la media dei ∇C_x sugli elementi di tale sottoinsieme. Così facendo si ottiene una buona stima del vero gradiente ∇C, inoltre si riduce molto il costo computazionale e di conseguenza i tempi di apprendimento Nielsen [2019].
Precisamente questo metodo funziona scegliendo aleatoriamente un numero intero m di input di allenamento. Ci riferiamo a questo insieme $X_1, X_2, ..., X_m$ come ad un mini lotto. Supposto m sufficientemente grande ci aspettiamo, per la legge dei grandi numeri, che la media campionaria di ∇C_{X_j} approssimi il valore atteso di tutta la popolazione ovvero:

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C \tag{2.13}$$

Considerando solo primo ed ultimo membro abbiamo:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j} \tag{2.14}$$

Il metodo quindi lavora sulla rete utilizzando la stima del gradiente per correggere pesi e bias:

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{2.15}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}. \tag{2.16}$$

Dove la somma è su tutti gli input di allenamento X_j del mini lotto corrente. Successivamente viene selezionato casualmente un altro mini lotto e ripetuti gli stessi passaggi. Una volta terminati gli input di apprendimento disponibili si dice che è stato completato un *epoch* di allenamento.
Non è necessario che la stima del gradiente sia perfetta quello che importa è muoversi nella direzione generale di decresita per C, questo significa che il calcolo esatto del gradiente non è richiesto. In altre parole le fluttuazioni statistiche non compromettono la potenza di questo metodo che è alla base di molte tecniche di apprendimento autonomo Nielsen [2019].

2.3.2 Algoritmo di backpropagation

Abbiamo visto come il metodo di discesa del gradiente venga utilizzato per l'apprendimento di una reta neurale. Soffermiamoci ora su come il gradiente possa essere calcolato efficientemente.
Il metodo usuale consiste nello sfruttare la definizione di rapporto incrementale per calcolarne le componenti:

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j) - C(w)}{\epsilon} \tag{2.17}$$

Dove $\epsilon > 0$ è un numero piccolo a piacere e e_j è il vettore unitario nella *j-esima* direzione. Sfortunatamente questo approccio seppur semplice da implementare risulta molto lento soprattutto se applicato ad una rete neurale con migliaia e migliaia di pesi e bias. Per questa ragione è stato ideato un altro tipo di algoritmo che abbassa di molto i tempi computazionali per il calcolo di ∇C detto algoritmo di *backpropagation* (retropropagazione). L'algoritmo si basa su quattro equazioni fondamentali:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)^1 \tag{2.18}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{2.19}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{2.20}$$

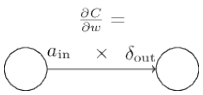
$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{2.21}$$

I equazione: Descrive l'errore δ^L nello strato di output. L'errore è definito come: $\delta^L \doteq \frac{\partial C}{\partial z^L_j}$ dove z^l_l è la componente *j-esima* dell input pesato ($z^l \doteq w^l a^{l-1} + b^l$). Il primo termine esprime quanto rapidamente il costo cambia come funzione del *j-esimo* output di attivazione. Il secondo termine invece misura quanto rapitamente la funzione di attivazione varia rispetto z^L_j. Nel caso della funzione di costo quadratica: $\nabla_a C = (a^L - y)$ e quindi $\delta^L = (a^L - y) \odot \sigma'(z^L)$

II equazione: Mette in relazione l'errore δ^l con l'errore nello strato successivo δ^{l+1}. In particolare quando applicchiamo la matrice trasposta a δ^{l+1} è come se spostassimo l'errore indietro di uno strato all'interno del network. Combinando questa equazione con la prima è possibile calcolare l'errore in qualsiasi strato partendo da δ^L, da questa possibilità di retropropagare l'errore fino l'inizio della rete deriva il nome dell'algoritmo.

III equazione: Sostanzialmente afferma che l'errore δ^l_j altro non è che la derivata parziale di C rispetto b^l_j ($\partial C / \partial b^l_j$).

IV equazione: Fornisce un modo semplice per calcolare la derivata parziale C rispetto il peso w^l_{jk}. Volendo alleggerire la notazione possiamo pensarla come $\frac{\partial C}{\partial w} = a_{in}\delta_{out}$:



Quando l'attivazione $a_{in} \approx 0$ anche la componente del gradiente $\partial C / \partial w \approx 0$ ovvero il peso non si modifica molto durante la discesa del gradiente.

[1] $\odot$ rappresenta il prodotto di Hadamard: dati due vettori s e t: $(s \odot t)_j = s_j t_j$.

Dimostrazione delle quattro equazioni fondamentali

Dimostriamo ora brevemente le equzioni viste sopra:

1. Partendo dalla definizione di δ^l:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} \tag{2.22}$$

Appliccando la regola di derivazione a catena possiamo esprimere la derivata partiale di sopra in funzione delle attivazioni dei neuroni output:

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \tag{2.23}$$

Quando $k \neq j$ l'attivazione a_k^L non dipende da z_j^L possiamo dunque scrivere:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \tag{2.24}$$

Ricordando che $a_j^L = \sigma(z_j^L)$ il secondo termine dell'equazione può essere riscritto come $\sigma'(z_j^L)$:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{2.25}$$

che altro non è che la forma in componenti della (2.18).

2. Per dimostrare la seconda equazione partiamo sempre dalla definizione di δ_j^L ed applichiamo la regola di derivazione a catena:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \tag{2.26}$$

Inoltre sappiamo che:

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{2.27}$$

differenziando:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l) \tag{2.28}$$

sostituendo nella (2.26) otteniamo:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) \tag{2.29}$$

ovvero la forma in componenti della (2.19).

3. Partiamo di nuovo dalla definizione di errore δ_j^l ed applichiamo la regola di derivazione a catena rispetto b_k^l:

$$\delta_j^l = \sum_k \frac{\partial C}{\partial b_k^l} \frac{\partial b_k^l}{\partial z_j^l} \tag{2.30}$$

come nella dimostrazione della prima equazione quando $k \neq j$, ${}^{\partial b_k^l}/_{\partial z_j^l} = 0$:

$$\delta_j^l = \frac{\partial C}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} \tag{2.31}$$

Ricordando che:

$$b_j^l = \sigma^{-1}(a_j^l) - \sum_k w_{jk}^l a_k^{l-1} = \sigma^{-1}(\sigma(z_j^l)) - \sum_k w_{jk}^l z_k^{l-1} = z_j^l - \sum_k w_{jk}^l z_k^{l-1} \tag{2.32}$$

differenziando:

$$\frac{\partial b_j^l}{\partial z_j^l} = 1 \tag{2.33}$$

e sostituendo nella (2.31) otteniamo:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.34}$$

ovvero la (2.20).

4. Similmente a quanto fatto precedentemente:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{2.35}$$

Dalla definizione di z_j^l e δ_j^l:

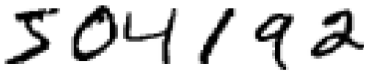
$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial (a_k^{l-1} w_{jk}^l + b_j^l)}{\partial w_{jk}^l} \delta_j^l = a_k^{l-1} \delta_j^l \tag{2.36}$$

ovvero la (2.21)

Riassumendo: l'algoritmo di backpropagation sfrutta la possibilità di calcolare la retropropagazione dell'errore all'interno della rete grazie all'equazione (2.19) per poi utilizzare questi valori per il calcolo delle componenti del gradiente della funzione di costo. Complessivamente il costo di tale algoritmo è lo stesso di quello richiesto per il passaggio in avanti dell'informazione dallo strato in input a quello di output Nielsen [2019].

2.4 Un'applicazione: riconoscimento di numeri scritti a mano

Mettiamo ora in pratica la teoria per vedere come una rete neurale possa riconoscere caratteri numerici scritti a mano del tipo:



Si tratta di dati modificati di 250 persone diverse raccolti dal NIST [2]. Sono immagini di 28x28 pixel in scala di grigi. La prima sfida consiste nel separare le singole cifre, in questo testo tuttavia ometteremo come affrontare tale problematica ed utilizzeremo singole cifre già separate.
La rete neurale si occuperà quindi di classificare individualmente le scritte. Useremo una rete neurale con architettura a tre strati: l'input layer, un hidden layer e l'output layer.

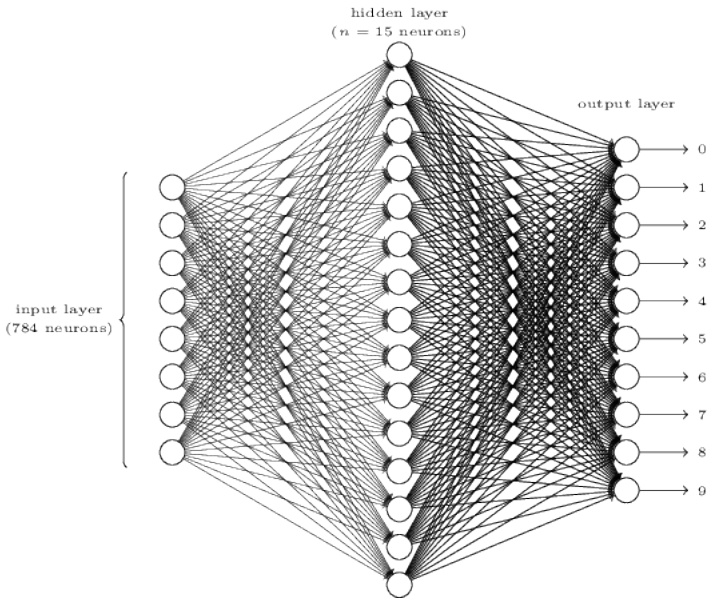
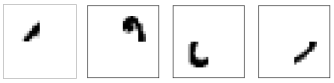


Figura 2.6: Rete neurale per il riconoscimento di cifre scritte a mano.

- **Input layer**: denotato con la lettera x può essere rappresentato come un vettore di dimensione 28x28 = 784, un elemento per ogni pixel dell'immagine presa in esame. Il valore di ogni componente del vettore è un numero compreso tra zero ed uno ad indicare il livello di grigio del pixel corrispondente dove 0 sta per bianco ed 1 per nero.

[2] United States' National Institute of Standards and Technology.

- **Hidden Layer**: Come evidenziato durante la trattazione non esiste una legge precisa per indicare il numero di neuroni che andranno a comporre lo strato intermedio, nella Figura 2.4 ne sono stati messi 15 ma è possibile provare configurazioni diverse con più neuroni in fase di implementazione e trovare empiricamente quella che fornisce risultati migliori. Non è nemmeno possibile definire con certezza la funzione che svolgono, euristicamente si può pensare che essi riconoscano un immagine vedendo da che parti è composta Nielsen [2019]. Per esempio supponiamo che i primi quattro neuroni si occupino di rilevare se nell'immagine da identificare sono presenti questi quattro pattern:

  In caso affermativo i quattro neuroni andranno ad "accendere" il neurone collegato all'output zero nell'ultimo strato e possiamo concludere che la cifra:

  è stata riconosciuta correttamente come zero.

- **Output layer**: composto da 10 neuroni ognuno dei quali indica una cifra da 0 a 9. I neuroni che si attivano indicano il valore finale dell'output della rete.

Come lavora questa rete neurale

In fase di allenamento vengono utilizzare in input 60,000 immagini di cifre scritte a mano. La rete utilizzando il metodo di discesa stocastico del gradiente e l'algoritmo di backpropagation regola autonomamente pesi e bias minimizzando la funzione di costo quadratica (2.10). Successivamente in fase di test vengono utilizzate altre 10,000 immagini diverse e si verifica quanti output corretti si hanno sul totale.

Capitolo 3

Deep Learning

3.1 Network neurali profondi

I netwrok neurali profondi (*deep neural networks*) sono un tipo di rete neurale dall'architettura multistrato. Precisamente, si considera una rete neurale profonda se è dotata di almeno due hidden layers. Il vantaggio rispetto ai network poco profondi sta nella possibilità di eseguire compiti anche molto complessi il che li rende strumenti molto più potenti. Inoltre rendono anche il design della rete più semplice in quanto è possibile stratificare e distribuire meglio attraverso gli hidden layer i vari compiti da gestire in fase di programmazione:

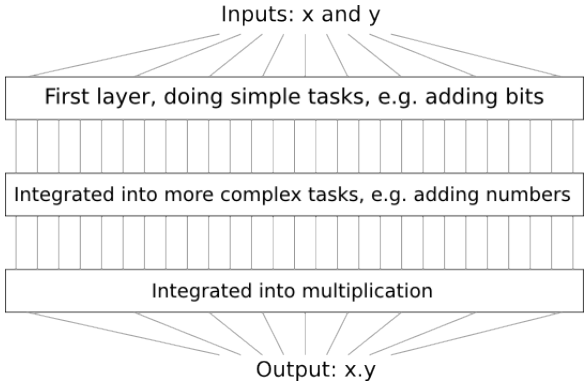
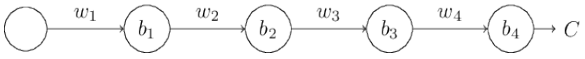


Figura 3.1: Esempio di rete multistrato per la moltiplicazione di due numeri.

Tuttavia la potenza di questi strumenti viene ad un caro prezzo. Vedremo infatti quali difficoltà la loro struttura comporta in fase di apprendimento.

3.2 Perché è difficile allenare i network neurali profondi

La difficoltà maggiore nell'allenare reti neurali profonde utilizzando gli strumenti a noi conosciuti, quali il metodo stocastico di discesa del gradiente e l'algoritmo di backpropagation, consiste nella scomparsa del gradiente.
La scomparsa del gradiente è il fenomeno per cui durante la fase di apprendimento le componenti del gradiente della funzione di costo negli strati più interni della rete risultano progressivamente minori nel tempo rispetto a quelle degli strati esterni, questo comporta velocità di apprendimento differenti, compromettendo l'apprendimento omogeneo della rete e rendendo il processo instabile.
Vediamo perché la cancellazione del gradiente è inevitabile. Consideriamo il caso di una rete neurale elementare formata da soli 5 neuroni in serie:



Per capire il fenomeno sopra descritto studiamo la componente $^{\partial C}/_{\partial b_1}$ del gradiente. Immaginiamo di effettuare una piccola variazione Δb_1 nel bias b_1 questo genererà un effetto a cascata che propagandosi all'interno della rete andrà ad influenzare la funzione di costo stessa. Abbiamo che:

$$\frac{\partial C}{\partial b_1} \approx \frac{\Delta C}{\Delta b_1} \tag{3.1}$$

Inoltre, ricordando che $a_1 = \sigma(z_1) = \sigma(w_1 a_0 + b_1)$, la variazione Δb_1 influira sull'output a_1 del neurone nel primo hidden layer provocandone una variazione Δa_1 secondo la legge:

$$\Delta a_1 \approx \frac{\partial \sigma(w_1 a_0 + b_1)}{\partial b_1} \Delta b_1 = \sigma'(z_1) \Delta b_1 \tag{3.2}$$

Questo cambiamento Δa_1 a sua volta influenzera l'input pesato, $z_2 = w_2 a_1 + b_2$, del neurone nel secondo hidden layer:

$$\Delta z_2 \approx \frac{\partial z_2}{\partial a_1} \Delta a_1 = w_2 \Delta a_1 \tag{3.3}$$

Sostituendo la (3.2) nella (3.3) vediamo come la variazione del bias b_1 si propaga all'interno del network fino a coinvolgere z_2:

$$\Delta z_2 \approx \sigma'(z_1) w_2 \Delta b_1 \tag{3.4}$$

Tracciando in avanti il modo in cui Δb_1 si propaga arriviamo infine ad includere l'effetto su C:

$$\Delta C = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4} \Delta b_1 \tag{3.5}$$

E dividendo per Δb_1:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)w_2\sigma'(z_2)w_3\sigma'(z_3)w_4\sigma'(z_4)\frac{\partial C}{\partial a_4} \tag{3.6}$$

Per capire la scomparsa del gradiente osserviamo il grafico di σ' (in rosso tratteggiato):

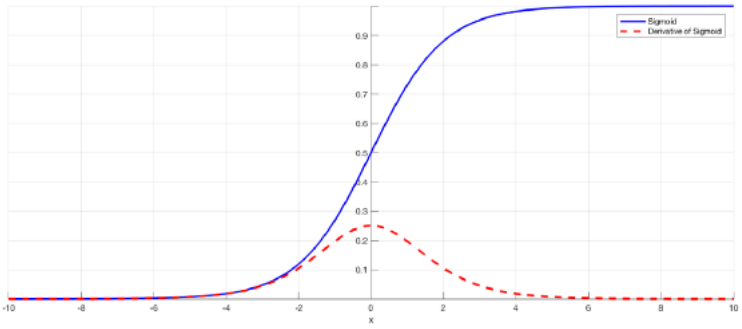


Figura 3.2: Derivata della funzione di Sigmoid

Come si può facilmente intuire il massimo della derivata si ottiene in 0 ed è pari ad $1/4$ ($\sigma'(0) = \frac{1}{4}$). Inoltre i pesi vengoni inizializzati causalmente secondo un distribuzione normale di media 0 e varianza 1 quindi $|w_j| < 1$ ma allora $|w_j\sigma'(z_j)| < 1/4$. Questo significa che più termini $w_j\sigma(z_j)$ sono presenti nell'espressione del gradiente più esso verrà scalato di un fattore $1/4$, è evidente quindi che:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)w_2\sigma'(z_2)w_3\sigma'(z_3)w_4\sigma'(z_4)\frac{\partial C}{\partial a_4} < \frac{\partial C}{\partial b_3} = \sigma'(z_3)w_4\sigma'(z_4)\frac{\partial C}{\partial a_4} \tag{3.7}$$

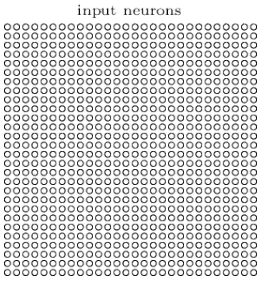
.

Nel prossimo sottocapitolo analizzeremo come aggirare questa problematica e sviluppare approcci alternativi al deep learning.

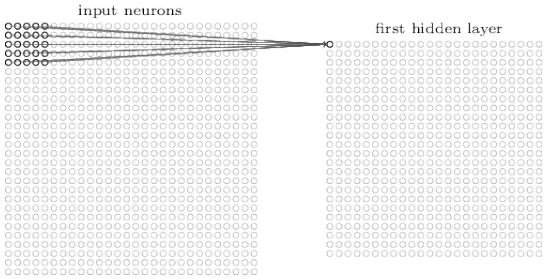
3.3 Network Convolutivi

I network convolutivi profondi utilizzano una speciale architettura che li rende particolarmente adatti all'analisi di immagini, oggigiorno infatti sono tra i più usati nell'ambito del riconoscimento di immagini Nielsen [2019]. Architettura che ha inoltre il vantaggio di aumentare considerevolmente la velocità di apprendimento evitando la cancellazione del gradiente. Questo tipo di network si basa su tre idee fondamentali: *campi ricettivi locali*, *pesi e bias condivisi* e *rappruppamento*.

1. **Campi ricettivi locali:** la prima novità consiste nella diversa organizzazione dei dati in input, nell'applicazione delle reti neurali al riconoscimento di cifre scritte a mano abbiamo visto come l'input poteva essere considerato un vettore di dimensione 28x28 = 784 che non teneva minimamente conto della struttura spaziale dei pixel. In una rete convolutiva possiamo invece ripensare l'input come una distribuzione bidimensionale di 28x28 pixel:



A differenza di prima ogni neurone dell'hidden layer non sarà connesso con tutti i singoli input pixel bensì con delle piccole aree circoscritte nella griglia di input. Per essere più precisi selezioneremo un area di 5x5 pixel corrispondente a 25 input e la collegheremo ad un neurone specifico:



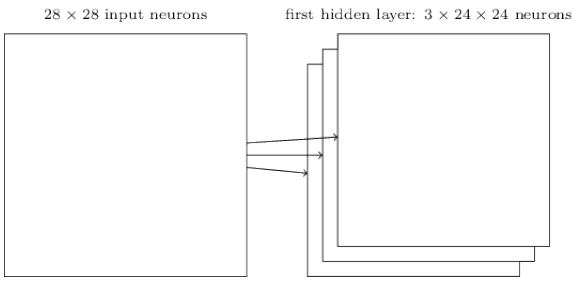
Quest'area è chiamata campo ricettivo locale (*local receptive field*) del neurone. Neuroni diversi saranno connessi a campi ricettivi diversi fino ad esaurire ogni possibili input. Considerando ad esempio campi spostati di un solo pixel verso destra e in basso gli uni rispetto agli altri otteniamo un hidden layer composto da 24x24 neuroni.

2. **Pesi e bias condivisi:** Ogni neurone collegato al campo ricettivo locale ha un proprio bias oltre a 25 pesi (un per ogni pixel in input). La novità consiste nel fatto che usiamo gli stessi pesi e bias (*shared weights and biases*) per tutti i neuroni appartenenti al medesimo strato. In altre parole per il *j,k-esimo* neurone l'outout sara:

$$\sigma(b + \sum_{l=0}^{4} \sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}) \tag{3.8}$$

L'equazione (3.8), in quanto una convoluzione, dà il nome a questa categoria di network, può essere infatti pensata come $a^1 = \sigma(b + w * a^0)$ dove a^1 sono gli output, a^o gli input di attivazione e $*$ l'operatore convoluzione.
La ragione per cui utilizzare pesi e bias comuni risulta chiara quando si pensa allo scopo della rete: l'identificazione di immagini. Un' immagine può presentare la stessa caratteristica, ad esempio una linea verticale, in più parti, i neuroni di uno stesso strato hanno quindi il compito di individuare quella particolare caratteristica attraverso tutta l'immagine. In termini più matematici si può dire che le reti neurali

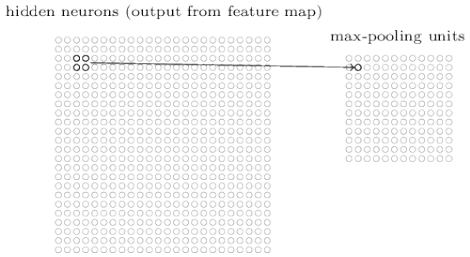
convolutive tengono conto dell'invarianza per traslazioni delle immagine. Per questa ragione ci si riferisce spesso al collegamento tra input ed hidden layer come ad una *mappa caratteristica*.
Per individuare proprietà diverse dell'immagine occorrono allora più mappe caratteristiche. Ad esempio la rete nella figura seguente è in grado di individuare tre proprietà diverse dell'input.



.

Il grande vantaggio dei pesi condivisi consiste nel ridotto numero di parametri: per ogni mappa caratteristica ne avremo solamente 26 (25 pesi + 1 bias) questo aumentanta di molto la velocità di apprendimento poichè riduce i costi computazionali.

3. **Raggruppamento:** Il raggruppamento (*pooling*) consiste in un ulteriore layer, intermedio tra quello convolutivo e l'output, utilizzato per semplificare le informazioni in output dallo strato precendente. Ad esempio il max-pooling semplicemente restituisce in output il massimo dell'attivazione in una regione 2x2 come illustrato nella figura di sotto:



.

Sostanzialmente serve a confermare la presenza o meno della proprietà individuata dalla mappa caratteristica in una determinata regione dell'immagine con il vantaggio di diminuire ancora una volta il numero di parametri per gli strati successivi.

Il risultato finale di un network convolutivo risulta essere:

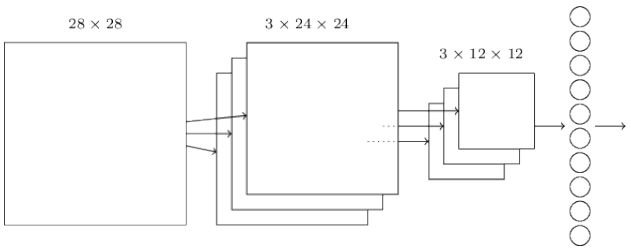


Figura 3.3: Network convolutivo completo.

.

3.4 Deep Learning ed intelligenza artificiale

Parlare di intelligenza è una questione delicata in quanto non esiste una definizione unanimemente condivisa di intelligenza Ienca [2019]. L'argomento è talmente vasto e profondo da poter riempire da solo le pagine di un libro.
Dare dunque un'interpretazione esaustiva di intelligenza o comportamento intelligente esula dalle priorità di quest'ultimo sottocapitolo che vuol essere invece una breve raccolta di recenti applicazioni del deep learning per lo sviluppo di tecnologie che potremmo definire intelligenti o almeno più intelligenti di quelle che le hanno precedute. Il deep learning è infatti alla base di:

- Algoritmi di riconoscimento vocale utilizzati negli assistenti virtuali di smartphone come Siri o Google Assistant.
- Algoritmi per la guida autonoma di veicoli.
- Algoritmi sofisticati per il riconoscimento di volti come DeepFace, una rete neurale sviluppata da Facebook che ha un grado di accuratezza del 97%.
- Un particolare tipo di rete neurale chiamato GAN in grado di generare immagini realistiche di persone mai esistite.
- Algoritmi utilizzati per i giochi di strategia come DeepBlue della IBM in grado di battere il campione del mondo di scacchi dell'epoca Gary Kasparov o AlphaGo di Google capace di sconfiggere l'uomo in quello che è considerato il gioco più complesso al mondo con opprossimativamente 2.08×10^{170} posizioni possibili.

Capitolo 4

Conclusioni

In conclusione spero che questo lavoro di tesi abbia risposto agli obbiettivi prefissati e saziato la fame di curiosità per chi, come me, è nuovo in questo campo.
Il motivo che mi ha spinto a scegliere tale argomento, nonostante io venga da un percorso matematico perlopiù teorico, è stato l'interesse personale di volere comprendere meglio le basi dei meccanismi che regolano il machine learning e la straordinaria potenza di calcolo che questo strumento offre. Mentre procedevo nella ricerca e visionavo i materiali a mia disposizione mi sono accorto di quanto l'argomento sia vasto ed un campo di studi ancora aperto. Soprattutto per quanto riguarda l'elaborazione di un'intelligenza artificiale evoluta, in grado di autogenerare propri concetti e rielaborarli. Come persona affascinata dal funzionamento del nostro cervello trovo che questo possa essere di grande aiuto per riuscire a comprendere meglio il nostro organo più importante.

4.1 Limite

Abbiamo elogiato le potenzialità che le reti neurali possono avere. Tuttavia non ne abbiamo analizzato i limiti. Il limite principale è nella complessità strutturale a cui reti neurali molto profonde possono arrivare. Tale complessità ci permette magari di ottenere risultati sraordinari ma ci impedisce di avere una visione chiara di come l'informazione venga pesata, scomposta ed analizzata all'interno della rete perché frutto di un processo di apprendimento automatico su cui non abbiamo controllo. Il rischio è quello di creare un cervello ma proprio come col nostro di non riuscirne a capire il funzionamento profondo.

Bibliografia

M. Ienca. *Intelligenza^2, per un'unione di intelligenza naturale ed artificiale.* Rosenberg & Sellier, 2019.

M. Nielsen. *Neural Networks and Deep Learning.* neuralnetworksanddeeplearning.com, 2019.