SAPIENZA
UNIVERSITÀ DI ROMA

Algoritmi di Apprendimento delle Reti Neurali

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Matematica

Candidato

Gabriele D'Andrea
Matricola 1744433

Relatori

Prof.ssa Elena Agliari
Prof. Fabrizio Frezza

Anno Accademico 2018/2019

Indice

Capitolo 1

Introduzione

L'obiettivo di questo elaborato è presentare i principali risultati dei principali algoritmi usati nel *Machine Learning*. Il machine learning è stato usato in origine per problemi legati alle immagini in 2D tra cui il riconoscimento delle immagini (per distinguere figure), image detection (per rilevare la posizione di un oggetto e il tipo di immagine, usato negli algoritmi delle guide automatiche) e nell'arte (utilizzate per creare nuove forme artistiche). Tra le applicazioni più recenti ricordiamo le tecniche di diagnosi assistita che consente di distinguere le cellule malate da quelle sane grazie alla elaborazione delle immagini (che vengono migliorate in modo da avere una rappresentazione di funzionalità di alto livello) e il georadar che, ispezionando il sottosuolo, può essere utilizzato per l'identificazione di danni strutturali e per l'indagine di strutture nascoste come condutture dell'acqua e del gas. Inoltre, nell'ambito dell'archeologia, il GPR (Ground Penetrating Radar) può permettere di identificare le aree con presunti resti sepolti interessanti, in modo da evitare uno scavo completo e a volte troppo costoso.

Nella prima parte di questo elaborato verrà fornita un'introduzione al concetto di reti neurali e di apprendimento.

Nella seconda verranno presentati i principali modelli matematici delle reti neurali, partendo da quello di McCulloch-Pitts per poi arrivare a modelli più complessi come le *reti neurali multi-strato*.

Verranno poi trattati gli aspetti teorici della disciplina e messi a confronto i principali algoritmi di apprendimento, di cui si discuteranno i vantaggi e gli svantaggi sulla base della velocità di convergenza alla soluzione ottima e sul costo computazionale.

1.1 Definizione di rete neurale

Una *rete neurale artificiale* è un modello ispirato dal sistema nervoso umano e dalla struttura del cervello costituita da unità elementari (dette *neuroni*) in grado di apprendere automaticamente attraverso l'elaborazione di segnali provenienti dall'esterno e dai neuroni stessi, in questo modo la conoscenza viene acquisita dalla rete tramite un processo di *apprendimento* e viene immagazzinata dalle incognite della rete.

Da un punto di vista matematico, una rete neurale può essere rappresentata da un *grafo* pesato ed orientato in cui i nodi e gli archi sono rappresentati, rispettivamente, dai neuroni e dalle connessioni tra i neuroni. Ai nodi possiamo associare un'informazione aggiuntiva che può essere una costante o una funzione a seconda del tipo di rete indicante lo stato del neurone. Infatti, per ogni neurone, possiamo avere tre tipi di stato: stato *eccitato*, in cui il neurone può trasmettere il segnale, stato *quiesciente*, in cui il neurone non può trasmettere il segnale e stato *refrattario*, in cui il neurone, dopo aver trasmesso l'impulso, impiega del tempo per poter nuovamente inviarne altri. Nel caso continuo lo stato è rappresentato da funzioni continue $s_i(t)$ che descrivono il passaggio da uno stato all'altro, mentre nel caso discreto associamo dei valori binari. Ad ogni arco viene associato un peso, che può essere maggiore, minore o uguale a 0 a seconda che il collegamento sia *eccitatorio* (ossia renda attivo il neurone), *inibitorio* (ossia renda quiesciente il neurone) o non esista alcuna connessione tra i due neuroni. Ogni neurone riceve in ingresso una combinazioni di segnali e ne effettua una trasformazione tramite delle *funzioni di attivazione*. Il segnale di uscita può essere trasmesso se la somma pesata degli ingressi supera un valore chiamato soglia, il quale di viene poi o inviato ad altri neuroni della reta o restituito dalla rete stessa.

Il legame ingresso-uscita realizzato dalla rete dipende essenzialmente:

- *dal tipo di unità elementari*, che possono avere struttura interna più o meno complessa ed avere funzioni di attivazione caratterizzate da differenti tipi di non linearità;
- *dall'architettura della rete*, ossia dal numero di nodi, dalla struttura e dall'orientamento delle connessioni;
- *dai valori dei parametri interni*, che devono essere determinati attraverso tecniche di apprendimento.

Una prima distinzione tra le reti neurali, legata al tipo di dinamica degli impulsi nervosi, è la seguente:

- *Reti dinamiche*, in cui alla rete sono associate delle variabili, dipendenti dal tempo, che possono essere descritte da equazione differenziali o alle differenze finite;
- *Reti statiche*, in cui il legame ingresso-uscita è considerato istantaneo.

Una seconda distinzione, basata sul tipo di architettura della rete, e la seguente:

- *Reti neurali feedforward*, in cui la rete è organizzata in livelli e il flusso di informazione può avvenire in una sola direzione dal livello di ingresso a quello d'uscita passando, eventualmente, per tutti gli altri livelli (figura 1.1);
- *Reti neurali ricorsive*, in cui il flusso è bidirezionale cioè è possibile inviare il segnale dal neurone i al neurone j e viceversa;
- *Reti neurali a base radiale*, in cui le funzioni di attivazione sono funzioni a base radiali, ossia funzioni di variabile reale il cui valore dipende unicamente dalla distanza in norma tra x e un punto prefissato c;

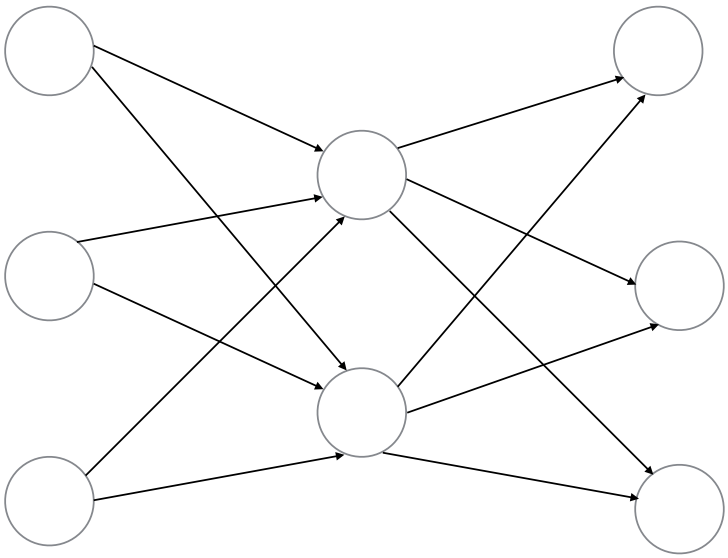

Figura 1.1. Esempio di rete neurale feedforward

L'implementazione di una rete neurale è composta dai seguenti passi:

1. Acquisire l'addestramento e analizzare l'insieme di dati;
2. Allenare la rete;
3. Effettuare previsioni con i dati attuali.

1.2 Definizione di apprendimento

Per *apprendimento* si intende la capacità di un programma di generalizzare ossia di estrarre un algoritmo in grado di dare la risposta corretta in corrispondenza di un input che non ha mai visto; partendo da un insieme di coppie, chiamato insieme di addestramento, il programma deve avere buone proprietà di generalizzazione in modo da poter considerare un insieme di input più grande. Una definizione di Mitchell (si veda [2]) afferma che: "Si dice che un programma impara da una certa *esperienza* E rispetto a una classe di *compiti* T ottenendo una *prestazione* P, se la sua capacità nel realizzare i compiti T, quantificata in termini di una misura di prestazione P, migliora con l'esperienza E." Dove:

- E: Esperienza indica il numero di volte in cui si è svolto quel lavoro;
- T: Task indica il lavoro da eseguire;
- P: Prestazione che può rappresentare, ad esempio, la probabilità che il programma riesca a completare il lavoro con successo.

L'apprendimento viene usato principalmente nella risoluzione di tre problemi:

- *classificazione*: il problema consiste, dato un insieme di input N dimensionali, associare ogni dato ad una categoria che viene indicata con un numero da 1 a k; l'obiettivo è trovare una funzione $f : \mathbb{R}^N \to \{1, 2, \ldots, k\}$ che associ ad ogni vettore $\boldsymbol{x}$ un numero appartenente all'insieme discreto di cardinalità k;
- *raggruppamento*: il problema consiste nel raggruppare in un certa categoria i dati che presentano caratteristiche simili tra loro;
- *regressione*: il problema consiste nel predire un valore futuro conoscendone il suo valore attuale.

Si possono distinguere tre paradigmi fondamentali di apprendimento:

- *Apprendimento Supervisionato*, in cui alla macchina vengono forniti le coppie di ingressi e uscite, i dati vengono prima etichettati e la macchina ricava una regola in grado di elaborare i dati non etichettati;
- *Apprendimento non Supervisionato*, in cui i dati non sono etichettati e la rete deve estrarre una regola che raggruppi i casi presentati secondo caratteristiche che ricava dai dati stessi senza una descrizione a priori del dato;
- *Apprendimento per Rinforzo*, in cui, a differenza dell'apprendimento supervisionato, in cui ad ogni input è associato un output, sono possibili diversi output per ogni input. In quest'ultimo caso l'algoritmo viene punito o premiato in base allo stato provvisto per ogni input fornito all'algoritmo. Inoltre, questo tipo di apprendimento non termina ma continua in modo indeterminato.

Per quanto riguarda le tecniche algoritmiche utilizzate nei problemi di apprendimento, le principali sono:

- *apprendimento batch*: si suppone che sia noto tutto l'insieme di addestramento e l'algoritmo comincia solo dopo che sono stati presentati tutti i campioni
- *apprendimento on-line*: l'algoritmo calcola le incognite della rete tenendo conto delle informazioni relative ad un singolo campione.

La seconda tecnica risulta più generale poiché può essere usata anche se i dati vengono acquisiti durante il processo. L'obiettivo dei problemi di apprendimento è quello di definire i parametri della rete, ossia i pesi associati alle interconnessioni e le soglie associate ai neuroni, che dovranno essere determinati partendo da un insieme finito di dati. Inoltre, come detto in precedenza, individuando i valori incogniti della rete, il programma dovrebbe essere in grado di trovare una regola generale, tenendo conto del legame ingresso-uscita, che consenta di restituire la risposta corretta. Nei problemi di approssimazione discreta tale problema è mal posto, poiché esistono più funzioni che realizzano i legami tra gli ingressi e le uscite, inoltre i dati possono essere affetti da errori di campionamento. In termini generali, il problema dell'apprendimento consiste nel definire in maniera "ottimale" (che definiremo nel seguito) la *complessità* di un modello in modo che

abbia buone proprietà di generalizzazione; infatti un modello troppo semplice potrebbe non essere in grado di descrivere un fenomeno in esame mentre uno troppo complesso potrebbe essere valido solo per un numero limitato di fenomeni reali. La complessità di un modello dipenderà dai parametri liberi della rete e dovranno essere determinati tenendo conto delle ipotesi della rete (per esempio le proprietà del grafo) e del numero dei campioni appartenenti all'insieme di addestramento. Il calcolo di tali parametri si basa sulla formulazione di un problema di ottimizzazione, in cui deve essere minimizzata un'opportuna funzione di errore della struttura.

Capitolo 2

Studio dei modelli neurali

In questa sezione viene descritta una delle versioni più comuni del neurone formale, ispirata ai principi di funzionamento del neurone biologico. Viene mostrato come il neurone formale si possa interpretare come un classificatore lineare i cui parametri possono essere determinati, sotto opportune ipotesi, risolvendo un sistema di disequazioni lineari; vengono quindi messe in evidenza le principali limitazioni di questo modello, che rendono necessaria la considerazione di strutture più complesse.

2.1 Modello di McCulloch-Pitts

Consideriamo uno dei più semplici modelli di reti neurali ossia il *modello di McCulloch-Pitts* (si veda [1]): supponiamo di avere N neuroni, a cui abbiamo associato dei valori di ingresso $x_i \in \mathbb{R}$ con $i = 1, \ldots, N$, collegati ad un singolo neurone di uscita y mediante sinapsi aventi peso $w_i \in \mathbb{R}$. La funzione y dovrà restituire valore 1 se la somma pesata dei valori di ingresso supera la soglia ϑ, -1 altrimenti; quindi y avrà una forma del tipo:

$$y = g(\sum_{i=1}^{N} w_i x_i - \vartheta) = g(\boldsymbol{w}^T \boldsymbol{x} - \vartheta)$$

dove g, detta *funzione di attivazione*, può assumere valori *booleani* o *continui* a seconda dei casi (in certe situazioni risulta conveniente scegliere una funzione continua in modo da poter usare le tecniche del calcolo differenziale ed integrale), si veda figura 2.1. Tra le funzione booleane le più usate sono la *funzione segno*

$$g(t) = sgn(t) = \begin{cases} +1 & t \geq 0 \\ -1 & t < 0, \end{cases}$$

che restituisce una risposta binaria $y \in \{-1, +1\}$, e la *funzione di Heaviside*

$$H(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$$

che restituisce una risposta binaria $y \in \{0, 1\}$. Per quanto riguarda le funzione continue, le più usate sono le funzioni *sigmoidali* ossia funzioni monotone crescenti

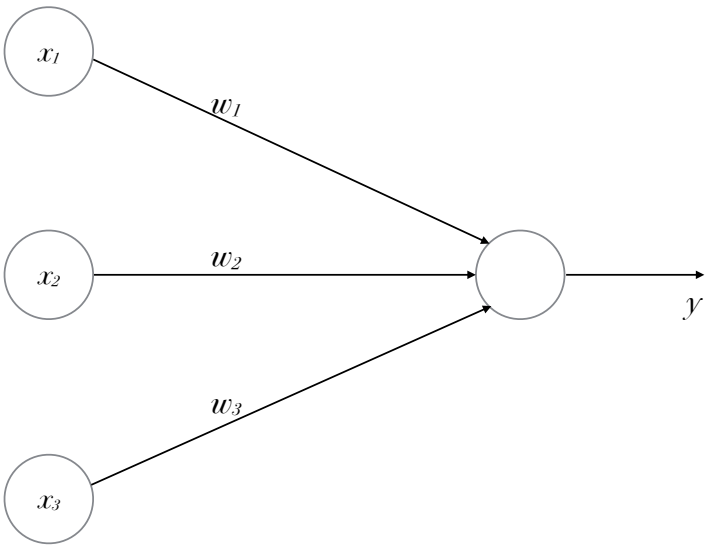

Figura 2.1. Schema di un neurone formale costituita da N=3 neuroni

tali che

$$\lim_{t\to+\infty} g(t) = 1 \qquad \lim_{t\to-\infty} g(t) = -1$$

come, ad esempio, la funzione tangente iperbolica

$$\tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}.$$

Con un'opportuna scelta dei pesi e della soglia, assumendo che g sia una funzione booleana, il neurone formale di McCulloch-Pitts è in grado di realizzare le funzione logiche AND,NOT e OR.

Il modello del neurone formale fu poi perfezionato da Rosenblatt, la cui idea fu di considerarlo come un *classificatore lineare* che associa ad ogni vettore $\boldsymbol{x}$ (che rappresenta le caratteristiche dell'oggetto da classificare) un valore binario (e quindi effettua una classificazione binaria) in base al segno della funzione lineare data da $\boldsymbol{w}^T\boldsymbol{x} - \vartheta$. L'aspetto più rilevante e innovativo rispetto agli altri modelli di calcolo consiste nel fatto che i valori dei pesi e della soglia possono essere determinati attraverso un processo di apprendimento a partire da un insieme di addestramento

$$T = \{(\boldsymbol{x}^p, y^p), \boldsymbol{x}^p \in \mathbb{R}^n, y^p \in \{-1, 1\} \quad p = 1, \ldots, P\}$$

formato dalle coppie ingresso-uscita in cui all'ingresso $\boldsymbol{x}^p$ viene associata la classificazione corretta y^p. I campioni dell'insieme di addestramento saranno classificati in maniera corretta se i pesi e la soglia sono determinati di modo che risultino vere le seguenti disuguaglianze:

$$\begin{cases} \sum_{i=1}^N w_i {x_i}^p - \vartheta \geq 0 & y^p = 1 \\ \sum_{i=1}^N w_i {x_i}^p - \vartheta < 0 & y^p = -1 \end{cases} \tag{2.1}$$

La risoluzione si basa sulla ricerca di un iperpiano H che separi i due insiemi A e B definiti come segue

$$A = \{\boldsymbol{x}^p, (\boldsymbol{x}^p, y^p) \in T, y^p = 1\} \quad B = \{\boldsymbol{x}^p, (\boldsymbol{x}^p, y^p) \in T, y^p = -1\}$$

ossia A è l'insieme degli ingressi della rete che restituiscono l'uscita 1 mentre B è l'insieme degli ingressi che restituiscono l'uscita -1.

L'esistenza di $\boldsymbol{w}$ e ϑ che risolvano il sistema (2.1) può quindi essere garantita se e solo se gli insiemi A e B sono *linearmente separabili* ossia se e solo se i rispettivi involucri convessi sono disgiunti. Si verifica anche facilmente che le condizioni (2.1) ammettono soluzione se e solo se ammette soluzione il sistema

$$\begin{cases} \boldsymbol{w}^T \boldsymbol{x}^p - \vartheta > 0 & \boldsymbol{x}^p \in A \\ \boldsymbol{w}^T \boldsymbol{x}^p - \vartheta < 0 & \boldsymbol{x}^p \in B \end{cases} \tag{2.2}$$

Tale scrittura può essere resa più compatta introducendo due valori x_0=-1 e $w_0 = \vartheta$, di modo che il sistema diventi:

$$\begin{cases} \boldsymbol{w}^T \boldsymbol{x}^p > 0 & \boldsymbol{x}^p \in A \\ \boldsymbol{w}^T \boldsymbol{x}^p < 0 & \boldsymbol{x}^p \in B \end{cases} \tag{2.3}$$

dove $\boldsymbol{w}$ e $\boldsymbol{x}^p$ sono vettori N+1-dimensionali e, senza perdita di generalità, $\|\boldsymbol{x}^p\| = 1$.

2.2 Algoritmo del percettrone

Il seguente algoritmo, sviluppato da Rosenblatt (si veda [1]), consente di determinare le componenti del vettore $\boldsymbol{w}$ partendo dall'insieme di addestramento T definito come sopra in maniera tale che risulti corretta la classificazione. Il vettore viene aggiornato sommando una certa quantità che sarà pari a $\boldsymbol{x}^p$ oppure a $-\boldsymbol{x}^p$ a seconda del valore di y^p.

Pseudo-codice:

Dati: Input $\boldsymbol{x}^p$, Target y^p, $p = 1, \ldots, P$.

Inizializzazione. Poni $\boldsymbol{w}(0) = \boldsymbol{0}$, $k = 0$, nclass = 0.

While nclass $< P$ do

  For $p = 1, \ldots, P$ do

    If $sgn(\boldsymbol{w}(k)^T \boldsymbol{x}^p) = y^p$ then

      poni nclass = nclass + 1

    else

      poni $\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + y^p \boldsymbol{x}^p$

      $k = k + 1$

  If nclass $< P$ then poni nclass = 0

Teorema1

Se A e B sono linearmente separabili, l'algoritmo restituisce in un numero finito di passi un vettore $\bar{\boldsymbol{w}}$ tale che tutti i campioni risultino correttamente classificati.

Dimostrazione:

Supponiamo che esista un vettore $\bar{\boldsymbol{w}}$ (che senza perdita di generalità ha norma unitaria) tale che soddisfi il sistema di disequazioni (2.3). Supponiamo per assurdo che l'algoritmo non termini in un numero finito di passi, ossia che per ogni iterazione k esista un vettore $\boldsymbol{x}^p$ (dipendente da k) non correttamente classificato. Calcoliamo il prodotto scalare tra $\bar{\boldsymbol{w}}$ e $\boldsymbol{w}(k+1)$, ricordando che $\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + y^p\boldsymbol{x}^p$:

$$\boldsymbol{w}(k+1)^T\bar{\boldsymbol{w}} = \boldsymbol{w}(k)^T\bar{\boldsymbol{w}} + y^p\boldsymbol{x}^p\bar{\boldsymbol{w}} > \boldsymbol{w}(k)^T\bar{\boldsymbol{w}} + \delta,$$

avendo posto $\delta{=}\min_{p=1,\dots P} y^p\boldsymbol{x}^p\bar{\boldsymbol{w}}{>}0$.

Usando l'induzione, si ha che:

$$\boldsymbol{w}(k+1) = (k+1)y^p\boldsymbol{x}^p$$

quindi, usando la disuguaglianza di Schwarz, si ha che la norma di $\boldsymbol{w}(k+1)$ è pari a:

$$\|\boldsymbol{w}(k+1)\| \geq \boldsymbol{w}(k+1)^T\bar{\boldsymbol{w}} \geq (k+1)\delta.$$

D'altra parte, tenendo conto che $y^p\boldsymbol{x}^p\boldsymbol{w}(k){<}0$ (poiché $\boldsymbol{x}^p$ non è correttamente classificato) e che la norma di $y^p\boldsymbol{x}^p$ è pari a 1 si ha che:

$$\|\boldsymbol{w}(k+1)\|^2 = \|\boldsymbol{w}(k)\|^2 + 2y^p\boldsymbol{w}(k)^T\boldsymbol{x}^p + 1 \leq \|\boldsymbol{w}(k)\|^2 + 1.$$

Da cui deriva che:

$$k+1 \leq \frac{1}{\delta^2}$$

da cui segue l'assurdo avendo supposto che l'algoritmo non convergesse in un numero finito di passi.

2.3 Limiti del percettrone

Il percettrone fu il prototipo dei modelli di reti neurali. Esso permetteva di riconoscere semplici schemi e modelli grazie ai suoi due livelli formati da neuroni in grado di processare le informazioni. Successivamente si scoprì che il percettrone non era in grado di realizzare la funzione logica binaria XOR che restituisce 1 se e soltanto se i valori in ingresso sono l'uno il NOT dell'altro (ma questo risultava ovvio dal teorema di convergenza, essendo XOR una funzione non linearmente separabile); infatti, è possibile dimostrare che non esiste un'opportuna scelta delle incognite della rete per implementare tale funzione. Questo portò alla nascita delle *reti neurali multistrato*, modello più complesso e sofisticato del percettrone, ma più efficiente e generale che tratteremo nei prossimi capitoli.

2.4 Algoritmi di discesa

Gli algoritmi risolutivi che verranno trattati nella prossima sezione sono *algoritmi di minimizzazione* basati sul metodo della discesa; rivediamo brevemente lo schema generale di un algoritmo di discesa (ricordando che $N(x)$ è un intorno di x)

Input: $\{\min_{x\in X} f(x)\}$; $\quad x^0 \in X$

Output: $x^* \in X$ soluzione ottima locale

Inizializzazione $x := x^0$(soluzione ottima corrente)

Repeat

seleziona $x^{'} \in N(x) \cap X$

modifica $N(x) = N(x) - x^{'}$

if $f(x^{'}) < f(x)$

$x = x^{'}$

$x^* = x$

finché $N(x) \cap X = \emptyset$

L'algoritmo considera (se esiste) un punto appartenente all'intersezione dell'intorno di x (che è la soluzione ottima corrente) e di X (ossia un "vicino" di x appartenente a X), confronta la funzione valutata in $x^{'}$ e x e , tra le due, sceglie il valore corrente che minimizzi la funzione e ripete finché l'insieme non sia vuoto. Tale metodo restituisce un ottimo locale, poiché considera i vicini di x.

Negli algoritmi che tratteremo il valore $\boldsymbol{x}^{'}$ appartiene ad $\mathbb{R}^n$ e sarà dato da una formula ricorsiva del tipo

$$\boldsymbol{x}^{'} = \boldsymbol{x} + \lambda \boldsymbol{d},$$

dove $\boldsymbol{d}$ è una *direzione di discesa* per $f(\boldsymbol{x})$ in $\boldsymbol{x}$ ossia un vettore di $\mathbb{R}^n$ (diverso dal vettore nullo) tale che esista $\lambda^* > 0$ tale che

$$f(\boldsymbol{x} + \lambda \boldsymbol{d}) \leq f(\boldsymbol{x}) \quad \forall \lambda \in (0, \lambda^*).$$

Nei metodi iterativi si sceglie come direzione di discesa un qualsiasi vettore $\boldsymbol{p}^k$ tale che risulti $(\boldsymbol{p}^k)^T \nabla f(\boldsymbol{x}^k) < 0$; questa scelta assicura che la funzione decresca in corrispondenza di piccoli spostamenti del valore minimo corrente $\boldsymbol{x}^k$, in base al teorema di Taylor. In base a questa definizione, l'antigradiente (se non nullo) è sempre una direzione di discesa, poiché $-\nabla f(\boldsymbol{x}^k)^T \nabla f(\boldsymbol{x}^k) = -\|\nabla f(\boldsymbol{x}^k)\|^2 < 0$.

Capitolo 3

Introduzione al Deep Learning

Per *Deep Learning*, come suggerisce il nome stesso, si intende una rete neurale formata da un elevato numero di livelli in grado di elaborare i segnali e di inviarli lungo la rete. Il passaggio da reti neurali "poco profonde" (ossia reti senza livelli nascosti) a reti "più profonde" (ossia reti con uno o più livelli nascosti) ha permesso a più funzioni di essere realizzate dalle reti neurali. In questa sezione tratteremo il modello delle reti multistrato e i principali algoritmi risolutivi.

3.1 Modello delle reti multistrato

Si tratta di un grafo formato da (si veda anche fig 3.1):

- un insieme di nodi d'ingresso associati agli n ingressi $x_i \in \mathbb{R}$ con $i = 1, \ldots, n$;
- un insieme di neuroni organizzati su $L \geq 2$ strati di cui:
  - L-1 *strati nascosti*, ossia sono formati da neuroni le cui uscite contribuiscono agli ingressi dei neuroni appartenenti agli strati successivi;
  - uno *strato d'uscita*, le cui uscite costituiscono l'uscita della rete;
- un insieme di archi orientati e pesati che rappresentano le interconnessioni fra i nodi e i neuroni.

Poniamo $w_{ij}^{(l)}$ il peso dell'arco uscente dal neurone i appartenente allo strato nascosto l-1 ed entrante nel neurone j appartenente allo strato nascosto l (che rappresenta il peso della connessione).

Sia poi $g_j^l : \mathbb{R} \to \mathbb{R}$ la funzione di attivazioni associate al neurone j appartenente allo strato nascosto l, che prende in input la somma pesata degli ingressi (che indicheremo con $a_j^{(l)}$) sottratta per la soglia (che indicheremo con w_{0j}); in conclusione, porremo per l=1

$$a_j^{(1)} = \sum_{i=1}^{n} w_{ij}^{(1)} x_i - w_{0j}^{(1)} \qquad z_j^{(1)} = g_j^{(1)}(a_j^{(1)})$$

e per $l \geq 2$ si avrà:

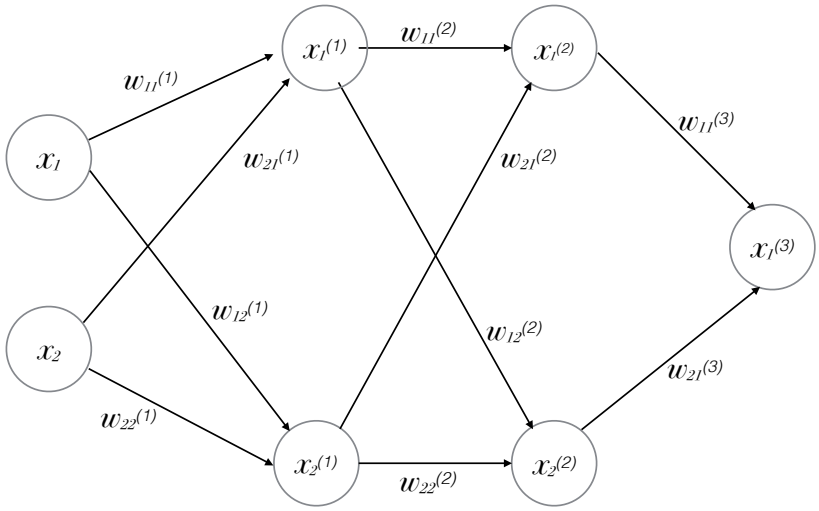

Figura 3.1. Rete neurale a 3 strati con 2 strati nascosti

$$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} z_i^{(l-1)} - w_{0j}^{(l)} \qquad z_j^{(l)} = g_j^{(l)}(a_j^{(l)}),$$

avendo indicato con $N^{(l)}$ il numero di neuroni dello strato nascosto l. Nel modello più semplice di rete multistrato, a cui faremo riferimento nel seguito, si assume che lo strato nascosto sia uno e lo strato d'uscita sia formato da un singolo neurone; in questo caso le notazioni si semplificano come segue:

- w_{ij} rappresenta i pesi delle connessioni tra gli ingressi e i neuroni dello strato nascosto;
- ϑ_j rappresenta la soglia associata al neurone j dello strato nascosto;
- v_j rappresenta i pesi delle connessioni tra i neuroni dello strato nascosto e il neurone dello strato d'uscita;
- g rappresenta la funzione di attivazione condivisa dai neuroni j appartenenti allo strato nascosto;

Sotto queste ipotesi l'uscita della rete y sarà data da:

$$y = \sum_{j=1}^{N} v_j g(\sum_{i=1}^{n} w_{ij} x_i - \vartheta_j).$$

Si assume che g sia differenziabile e sigmoidale. Le scelte più comuni sono la funzione logistica

$$g(t) = \frac{1}{1 + e^{-ct}}$$

che fornisce un'uscita (0,1), oppure la funzione tangente iperbolica

$$g(t) = \tanh(t/2) = \frac{1 - e^{-t}}{1 + e^{t}}$$

che restituisce una risposta (-1,1).

3.2 Teorema di approssimazione

Prima di trattare i principali algoritmi, premettiamo alcuni risultati riguardanti le reti a 2 strati con uno strato nascosto. Le reti a 2 strati consentono, in linea di principio, di interpolare esattamente i dati, nel senso precisato nel teorema successivo.

Teorema 2

Sia $g \in C(\mathbb{R})$ e si assuma che g non sia un polinomio. Dati k vettori distinti $\boldsymbol{x}^i \in \mathbb{R}^n$ e k numeri $\alpha_i \in \mathbb{R}$, esistono k vettori $\boldsymbol{w}^j \in \mathbb{R}^n$ e $2k$ numeri $v_j, \vartheta_j \in \mathbb{R}$ tali che

$$\sum_{j=1}^{k} v_j g(\boldsymbol{w}^j \boldsymbol{x}^i - \vartheta_j) = \alpha_i \quad i = 1, \ldots, k. \tag{3.1}$$

Prima di tutto, premettiamo la seguente definizione e le seguenti proposizioni:

Definizione 1

Assumiamo che Λ e Θ siano sottoinsiemi di $\mathbb{R}$ e assumiamo che g sia continua. Chiameremo $N(g, \Lambda, \Theta)$ la combinazione lineare delle funzioni ottenute componendo la funzione g con le rette $\lambda t - \vartheta$, in formule

$$N(g, \Lambda, \Theta) = span\{g(\lambda t - \vartheta); \lambda \in \Lambda, \vartheta \in \Theta\}.$$

Proposizione 1

Sia $g \in C^{\infty}(\mathbb{R})$ ed assumiamo che g non sia un polinomio. Allora $N(g, \mathbb{R}, \mathbb{R})$ è denso in C(ℝ).

Dimostrazione

Sappiamo che, se una funzione g derivabile infinite volte con continuità su un intervallo aperto non è un polinomio, allora esiste un punto $-\vartheta_0$ nell'intervallo tale che $g^{(k)}(-\vartheta_0) \neq 0$ per ogni k=0,1,2,....

Dal momento che $g \in C^{\infty}(\mathbb{R})$ e $[g((\lambda + h)t - \vartheta_0) - g(\lambda t - \vartheta_0)]/h \in N(g, \mathbb{R}, \mathbb{R})$ per ogni $h \neq 0$ segue che

$$\frac{d}{d\lambda} g(\lambda t - \vartheta_0)\big|_{\lambda=0} = t g^{'}(-\vartheta_0)$$

è contenuto nella chiusura di $N(g, \mathbb{R}, \mathbb{R})$. Allo stesso modo

$$\frac{d^k}{d\lambda^k} g(\lambda t - \vartheta_0)\big|_{\lambda=0} = t^k g^{(k)}(-\vartheta_0)$$

è contenuto nella chiusura per ogni k. Poiché $g^{(k)}(-\vartheta_0) \neq 0$, la chiusura di $N(g, \mathbb{R}, \mathbb{R})$ contiene tutti i monomi e quindi tutti i polinomi. Per il teorema di approssimazione di Weierstrass questo implica che $N(g, \mathbb{R}, \mathbb{R})$ è denso in C(K) per ogni compatto $K \subset \mathbb{R}$.

Proposizione 2
Sia $g \in C(\mathbb{R})$ e assumiamo che g non sia un polinomio. Allora N(g,$\mathbb{R}$,$\mathbb{R}$) è denso in C($\mathbb{R}$).

Dimostrazione
Sia $\varphi \in C^\infty(\mathbb{R})$ una funzione a supporto compatto. Definiamo

$$g_\varphi(t) = \int_{-\infty}^{+\infty} g(t-y)\varphi(y)\,dy$$

ossia $g_\varphi(t)$ è la convoluzione tra g e φ. Dato che $g, \varphi \in C(\mathbb{R})$ e φ è una funzione a supporto compatto , la convoluzione converge per ogni t, inoltre, come è facile dimostrare considerando le somme di Riemann, g_φ apparitene alla chiusura di $N(g, \{1\}, \mathbb{R})$.

Inoltre la chiusura di $N(g_\varphi, \mathbb{R}, \mathbb{R})$ è contenuta nella chiusura di N(g,$\mathbb{R}$,$\mathbb{R}$) dato che

$$g_\varphi(\lambda t - \vartheta) = \int_{-\infty}^{+\infty} g(\lambda t - \vartheta - y)\varphi(y)\,dy$$

per ogni $\lambda \in \mathbb{R}$. Poiché $g_\varphi \in C^\infty(\mathbb{R})$ abbiamo, dal metodo di verifica della proposizione 1, che $t^k g_\varphi^{(k)}(-\vartheta)$ appartiene alla chiusura di $N(g_\varphi, \mathbb{R}, \mathbb{R})$ per ogni $\vartheta \in \mathbb{R}$ e per ogni k. Adesso se $N(g, \mathbb{R}, \mathbb{R})$ non è denso in C($\mathbb{R}$) allora t^k non appartiene alla chiusura di $N(g, \mathbb{R}, \mathbb{R})$ per un qualche k. Così t^k non appartiene in $N(g_\varphi, \mathbb{R}, \mathbb{R})$ per ogni $\varphi \in C^\infty(\mathbb{R})$. Questo implica che $g_\varphi^{(k)}(-\vartheta) = 0$ per ogni $\vartheta \in \mathbb{R}$ e per ogni $\varphi \in C^\infty(\mathbb{R})$. Così possiamo concludere che g_φ è un polinomio di grado al più k-1. Sappiamo che esiste una sequenza di funzioni $\varphi_n \in C^\infty$ tale che g_{φ_n} converga uniformemente ad una funzione g su ogni compatto di $\mathbb{R}$. Dal momento che g_{φ_n} è un polinomio di grado al più k-1 per ogni φ_n, segue necessariamente che anche g è un polinomio di grado al più k-1. Questo contraddice l'ipotesi.

Dimostrazione del Teorema 2
Sia $\boldsymbol{w}$ un qualche vettore di di $\mathbb{R}^n$ tale che $\boldsymbol{w}\boldsymbol{x}^i = t_i$ con t_i valori distinti per $i = 1, \ldots, k$. Sia $\boldsymbol{w}^j = \lambda_j \boldsymbol{w}$ per $\lambda_j \in \mathbb{R}$, $j = 1, \ldots, k$. Sostituendo le quantità nella (3.1), otteniamo la tesi equivalente

$$\sum_{j=1}^{k} v_j g(\lambda_j t_i - \vartheta_j) = \alpha_i \quad i = 1, \ldots, k. \tag{3.2}$$

Risolvere la (3.2) è equivalente a dimostrare l'indipendenza lineare (al variare di λ e ϑ) delle k funzione continue $g(\lambda t_i - \vartheta)$, $i = 1, \ldots, k$. Se queste funzioni sono linearmente indipendenti, allora esistono λ_j,ϑ_j,$j = 1, \ldots, k$, tale che

$$det(g(\lambda_j t_i - \vartheta_j)) \neq 0$$

e la (3.2) può essere risolta con questa scelta delle costanti λ_j,ϑ_j. Se, per assurdo, le funzioni fossero linearmente dipendenti, esisterebbero dei coefficienti d_i tali che

$$\sum_{i=1}^{k} d_i g(\lambda t_i - \vartheta) = 0 \tag{3.3}$$

per ogni $\lambda, \vartheta \in \mathbb{R}$.

Potremmo riscrivere la (3.3) nella forma

$$\int_{-\infty}^{+\infty} g(\lambda t - \vartheta)\, d\mu(t) = 0 \tag{3.4}$$

con

$$d\mu = \sum_{i=1}^{k} d_i d\delta_i$$

(dove $d\delta_i$ vale 1 in $t = t_i$, zero altrimenti). La misura $d\mu$ è una misura di Borel non banale con supporto compatto. In altre parole, essa rappresenta un funzionale lineare non banale su C($\mathbb{R}$). Abbiamo costruito, in (3.4), una funzionale lineare non banale $g(\lambda t - \vartheta)$ al variare di $\lambda, \vartheta \in \mathbb{R}$. Questo implica che

$$span\{g(\lambda t - \vartheta) : \lambda, \vartheta \in \mathbb{R}\}$$

non è denso in C($\mathbb{R}$), e questo contraddice la proposizione 2.

3.3 Algoritmi di apprendimento per reti neurali multistrato

La costruzione di una rete multistrato con n ingressi e K uscite consiste nello scegliere la struttura della rete (ossia il numero di strati e il numero di neuroni) e nell'addestrare la rete ossia trovare i valori incogniti della rete (che sarebbero i pesi delle connessioni e le soglie associate ad ogni neurone) usando opportuni algoritmi partendo da un insieme di addestramento

$$T = \{(\boldsymbol{x}^p, \boldsymbol{y}^p), \boldsymbol{x}^p \in \mathbb{R}^n, \boldsymbol{y}^p \in \mathbb{R}^K p = 1, \ldots, P\}$$

dove, in generale, $\boldsymbol{x}^p$ sono vettori n dimensionali mentre $\boldsymbol{y}^p$ sono vettori K dimensionali. Per trovare le incognite della rete, come avevamo preannunciato, bisogna minimizzare un funzione di errore della rete stessa definita come

$$E(\boldsymbol{w}) = \sum_{p=1}^{P} E_p(\boldsymbol{w})$$

dove E_p è il termine di errore relativo al p-esimo campione e misura la distanza tra l'uscita desiderata $\boldsymbol{y}^p$ e l'uscita fornita dal sistema $\boldsymbol{y}^p(\boldsymbol{x}^p, \boldsymbol{w})$. La misura più usata è l'errore quadratico

$$E_p(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{y}^p - \boldsymbol{y}^p(\boldsymbol{x}^p, \boldsymbol{w})\|^2$$

Come si è detto in precedenza, lo scopo dell'addestramento è di risalire al processo che ha generato i dati; per questo motivo la scelta dell'architettura, la scelta dei campioni da inserire in T, la definizione di E e la strategia di addestramento dovranno assicurare buone capacità di generalizzazione. Per le reti multi-strato è stato stabilito un numero minimo di campioni affinché la rete abbia buone proprietà di generalizzazione. Nonostante questo importante risultato teorico, la scelta della struttura e del metodo di addestramento viene effettuata tramite algoritmi euristici,

poiché le stime teoriche possono rivelarsi inefficienti. In genere vengono seguite due strategie fondamentali: la prima, detta stabilizzazione strutturale, consiste nello scegliere il numero di unità elementari tramite l'addestramento di una sequenza di rete in cui viene fatto variare il numero di neuroni nascosti. Per ogni rete i parametri liberi vengono determinati minimizzando funzioni di errore e le prestazioni di ogni rete vengono misurate sulla base degli errori che la rete commette in corrispondenza di un campione non appartenente all'insieme di addestramento. La rete restituita è quella che minimizza l'errore. La seconda, detta tecnica di regolarizzazione, consiste nell'aggiungere alla funzione di errore un termine di penalità sulla norma di $\boldsymbol{w}$ in modo da ridurre l'insieme delle possibili scelte. Quindi la funzione E sarà data da

$$E(\boldsymbol{w}) = \sum_{p=1}^{P} E_p(\boldsymbol{w}) + \gamma \|\boldsymbol{w}\|^2$$

Il problema della minimizzazione di E è un difficile problema di ottimizzazione non lineare poiché la complessità del problema è dovuta alle seguenti difficoltà computazionali:

- forti non linearità di E;
- possibile mal condizionamento dell'Hessiana (ossia variazioni infinitesimali delle componenti della matrice Hessiana causano grandi variazioni della soluzione del corrispondente sistema lineare);
- elevata numero W dei parametri $\boldsymbol{w}$ ed elevato numero P di campioni p;
- presenza di minimi locali non globali (ossia $\boldsymbol{w}$ è una soluzione del problema di minimizzazione solo se consideriamo un intorno di $\boldsymbol{w}$ stesso).

Per questi motivi si preferisce usare algoritmi euristici poiché assicurano una soluzione, in generale non ottima ma "vicina" ad essa, in termini computazionali accettabili.

3.4 Backpropagation

Il primo metodo che tratteremo è il *Backpropagation*, uno dei più diffusi (si veda per esempio [2],[5]).

Consideriamo una rete neurale formata da L strati, in cui $\boldsymbol{x} \in \mathbb{R}^n$ è il vettore d'ingresso, $\boldsymbol{y} \in \mathbb{R}^K$ è il vettore d'uscita e poniamo

$$z_i^{(0)} = x_i \quad per \quad i = 1, \ldots, n \quad z_i^{(L)} = y_i \quad per \quad i = 1, \ldots, K$$

Sia $w_{ij}^{(l)}$ il peso dell'arco entrante nel neurone j. Calcoliamo la quantità $\partial E / \partial w_{ij}^{(l)}$,tenendo conto del fatto che E_p dipende da $w_{ij}^{(l)}$, solo attraverso la dipendenza dall'ingresso $a_j^{(l)}$ al neurone j. Utilizzando la regola di derivazione delle funzione composte si ha che

$$\frac{\partial E_p}{\partial w_{ij}^{(l)}} = \frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial E_p}{\partial a_j^{(l)}}$$

Definendo

$$\delta_j^{(l)} := \frac{\partial E_p}{\partial a_j^{(l)}}$$

e ricordando che $a_j^{(l)} = \sum_{h=1}^{n} w_{hj}^{(l)} z_h^{(l-1)}$, si ha che

$$\frac{\partial E_p}{\partial w_{ij}^{(l)}} = \delta_j^l z_i^{(l-1)} \tag{3.5}$$

dove i $z_i^{(l)}$ possono essere ricavati dagli ingressi. Il prossimo obiettivo sarà ricavare i valori delle $\delta_j^{(l)}$, e per farlo distinguiamo due casi:

- j è un neurone appartenente allo strato di uscita
- j appartiene ad uno strato nascosto

Nel primo caso, ricordando che l'uscita dello strato nascosto è pari a $z_j^{(L)} = g^{(L)}(a_j^{(L)})$ si ha che:

$$\delta_j^{(L)} = \frac{\partial E_p}{\partial a_j^{(L)}} = \frac{dg_j^{(L)}}{da_j^{(L)}} \frac{\partial E_p}{\partial z_j^{(L)}}$$

dove il primo termine è la derivata di $g_j^{(L)}$ e il secondo termine è calcolabile analiticamente.

Nel secondo caso, l'errore $\delta_j^{(l)}$ è data dalla sommatoria estesa su tutti i neuroni (nascosti o di uscita) che ricevono impulsi elettrici dal neurone j appartenente allo strato nascosto l, quindi

$$\delta_j^{(l)} = \sum_k \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial E_p}{\partial a_k^{(l+1)}}$$

poiché la funzione E_p dipende da $a_j^{(l)}$ solo tramite la dipendenza da $a_k^{(l+1)}$ (essendo definita ricorsivamente a partire dai valori degli ingressi dello strato precedente). Ricordando che $a_k^{(l)} = \sum_h w_{hk}^{(l)} g_h^{(l-1)}(a_h^{(l-1)})$, si ha che

$$\frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} = w_{kj}^{(l+1)} \frac{dg_j^{(l)}(a_j^{(l)})}{da_j^{(l)}}$$

e sapendo che

$$\frac{\partial E_p}{\partial a_k^{(l+1)}} = \delta_k^{(l+1)}$$

mettendo assieme tutto, otteniamo la seguente formula ricorsiva per l'errore j dello strato l

$$\delta_j^{(l)} = \sum_k w_{jk}^{(l+1)} \frac{dg_j^{(l)}(a_j^l)}{da_j^{(l)}} \delta_k^{(l+1)}$$

quindi l'errore si ottiene "propagando" all'indietro gli errori dello strato successivo.

Riassumendo, l'algoritmo di Backpropagation si basa sulle seguenti fasi:

1. Immettere un vettore di ingresso $\boldsymbol{x}$ e calcolare ricorsivamente gli ingressi e le uscite dei livelli successivi;

2. Valutare la quantità $\delta_k^{(l)}$ a seconda del tipo di neurone;

3. Usare la propagazione all'indietro per ottenere gli errori di tutti gli strati;

4. Usare la (3.5) per valutare l'errore richiesto e quindi il gradiente.

Dati. Input $\boldsymbol{x}^p \in R^n$, Target $\boldsymbol{y}^p \in R^K$.

Poni $z_i^{(0)}$=x_i^p, i=1,. . . ,n, $z_{n+1}^{(0)}$=-1

For $l = 1, \ldots, L$

For $j = 1, \ldots, N^{(l)}$

$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} z_i^{(l-1)}, \quad z_j^{(l)} = g_j^{(l)}(a_j^{(l)})$

poni $z_{N^{(l)}+1}^{(l)}$=-1

For $i = 1, \ldots, K$

poni $e_i = z_i^{(L)} - y_i^{(L)}$

For $j = 1, \ldots, K$

calcola $\delta_j^{(L)} = e_j g_j'(a_j^{(L)})$

poni $\frac{\partial E_p}{\partial w_{ij}^{(L)}} = \delta_j^{(L)} z_i^{(L-1)}$

For $l = L-1, \ldots, 1$

For $j = 1, \ldots, N^{(l)}$

calcola $\delta_j^{(l)} = g_j'(a_j^{(l)}) \sum_{k=1}^{N^{(l+1)}} \delta_k^{(l+1)} w_{jk}^{(l+1)}$

For $i = 1, \ldots, N^{(l-1)} + 1$

poni $\frac{\partial E_p}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} z_i^{(l-1)}$

Il costo dell'algoritmo è O(W), dove W è la dimensione dei parametri. Questo algoritmo deve essere eseguito per ogni funzione E_p (poiché ∇E=$\sum_{p=1}^{P} \nabla E_p$) quindi il costo totale per calcolare ∇E è pari a O(PxW).

3.5 Metodo del gradiente

Il metodo precedentemente trattato può essere posto in relazione con il metodo del gradiente, uno degli di algoritmi di ottimizzazione più noti ed usato, in cui il vettore $\boldsymbol{w}$ viene modificato nel modo seguente

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta^k \nabla E(\boldsymbol{w}^k)$$

con η^k>0 che deve essere determinato dalle condizioni sulla funzione E. Nel caso particolare in cui la funzione E sia una funzione quadratica

$$E(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T A \boldsymbol{w} - \boldsymbol{b}^T \boldsymbol{w},$$

il cui problema di minimo corrisponde a risolvere un sistema lineare della forma $A\boldsymbol{w} = \boldsymbol{b}$, con A matrice simmetrica e definita positiva, il passo di discesa η^k è dato da

$$\eta^k = -\frac{\nabla E(\boldsymbol{w}^k)^T \boldsymbol{r}^k}{\nabla E(\boldsymbol{w}^k)^T A \nabla E(\boldsymbol{w}^k)},$$

con $\boldsymbol{r}^k = \boldsymbol{b} - A\boldsymbol{w}^k$, che si ottiene rendendo stazionaria la funzione

$$E(\eta^k) = E(\boldsymbol{w}^{k+1} - \eta^k \nabla E(\boldsymbol{w}^k))$$

Supponendo il passo di discesa costante, la convergenza può essere garantita, imponendo che la funzione E soddisfi una condizione di Lipschitz ossia che esista una costante L>0, chiamata costante di Lipschitz (che sarebbe la più piccola costante di Lipschitz), tale che

$$\forall \boldsymbol{u}, \boldsymbol{v} \in R^m \quad \|\nabla E(\boldsymbol{u}) - \nabla E(\boldsymbol{v})\| < L\|\boldsymbol{u} - \boldsymbol{v}\|.$$

Supponendo che il passo di discesa dipenda dall'iterazione k, esso può essere ricavato usando tecniche euristiche in modo da soddisfare due condizioni:

- variazione "contenuta" dell'ottimo corrente;
- miglioramento della funzione in corrispondenza del passo k+1.

Un possibile metodo euristico è il *metodo di Armijo*, in cui si assume che la direzione $\boldsymbol{d}^k$ sia di discesa:

Metodo di Armijo:

Dati: $a^k > 0, \quad \delta \in (0,1), \quad \gamma \in (0, 1/2)$

poni $\eta = a^k \quad j$=0

while $E(\boldsymbol{w}^k + \eta \boldsymbol{d}^k) \leq E(\boldsymbol{w}^k) + \gamma\eta\nabla E(\boldsymbol{w}^k)\boldsymbol{d}^k$

poni $\eta = \delta\eta, \quad j = j + 1$

poni $\eta^k = \eta$

Per poter usare il metodo bisogna definire un valore iniziale a^k, che è scelto di modo che soddisfi la seguente disuguaglianza

$$a^k \geq \frac{1}{\|\boldsymbol{d}^k\|}\sigma\left(\frac{\nabla E(\boldsymbol{w}^k)^T\boldsymbol{d}^k}{\|\boldsymbol{d}^k\|}\right)$$

dove $\sigma : \mathbb{R}^+ \to \mathbb{R}^+$ è una funzione di forzamento. L'algoritmo potrebbe risultare troppo costoso in termini computazionali poiché il numero di valutazioni della funzione σ potrebbe influire pesantemente sul costo del metodo. Per rendere il metodo più veloce si preferisce usare criteri di accettabilità, ossia, se risulta che $\nabla E(\boldsymbol{w}^k)^T\boldsymbol{d}^k < 0$, η^k può essere scelto tra i valori che soddisfano le seguenti disuguaglianze

$$\begin{cases} E(\boldsymbol{w}^k + \eta^k\boldsymbol{d}^k) \leq E(\boldsymbol{w}^k) + \gamma_1\eta^k\nabla E(\boldsymbol{w}^k)^T\boldsymbol{d}^k \\ E(\boldsymbol{w}^k + \eta^k\boldsymbol{d}^k) \geq E(\boldsymbol{w}^k) + \gamma_2\eta^k\nabla E(\boldsymbol{w}^k)^T\boldsymbol{d}^k \end{cases}$$

con $0<\gamma_1<\gamma_2<1/2$. Anche in questo caso, però, l'algoritmo risulta inefficiente e le operazioni possono richiedere un numero troppo elevato di iterazioni.

Posto $\boldsymbol{d}^k = -\nabla E(\boldsymbol{w}^k)$ e utilizzando il metodo di Armijo per il calcolo di η^k, si può dimostrare che ogni punto limite della sequenza generata è un punto stazionario. L'impiego di una tecnica di ricerca unidimensionale nel metodo del gradiente risulta notevolmente vantaggioso rispetto alla scelta di un passo costante o all'utilizzo di una tecnica euristica. Tuttavia, tutte le implementazioni tradizionali del metodo del gradiente sono in generale inefficienti e possono richiedere un numero molto elevato di iterazioni e di valutazioni della funzione e del gradiente anche per problemi di piccola dimensione. La rapidità di convergenza dell'algoritmo può essere migliorata aggiungendo alla direzione di ricerca un termine dipendente dalla direzione del passo precedente

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta\nabla E(\boldsymbol{w}^k) + \beta(\boldsymbol{w}^k - \boldsymbol{w}^{k-1})$$

con $\eta>0$ e $\beta>0$ scelti in maniera opportuna; la scelta ottima corrisponde al caso

$$\eta = \frac{4}{(\sqrt{L}+\sqrt{l})^2} \quad \beta = \frac{(\sqrt{L}-\sqrt{l})^2}{(\sqrt{L}+\sqrt{l})^2}.$$

3.6 Metodo stocastico del gradiente

Il metodo standard del gradiente, per la maggior parte delle funzione di errore, garantisce che il vettore dei pesi $\boldsymbol{w}$ converga ad un minimo locale, a patto che ne esista uno e il passo di discesa sia piccolo. Da un punto di vista computazionale, l'algoritmo di discesa del gradiente può risultare troppo oneroso se il numero di pesi e di campioni è molto elevato oppure se la forma delle funzioni non è facilmente derivabile. Un modo per ridurre il costo è usare un algoritmo stocastico. Il metodo stocastico del gradiente, a differenza del metodo standard, il quale elabora tutti i campioni dell'insieme di addestramento prima di aggiornare il vettore dei pesi, approssima la funzione di errore, data dalla somma degli errori associati ad ogni campione, come la somma di un numero $m < P$ di funzioni di errore. Quindi l'iterazione è data da

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta\nabla\sum_{p=1}^{m} E_p(\boldsymbol{w}^k).$$

Il metodo on-line, sulla base della formula precedente, risulta essere un caso particolare del metodo stocastico poiché E viene approssimata con un singolo addendo. Poiché i pesi vengono aggiornati più frequentemente rispetto al metodo standard, l'algoritmo può convergere più velocemente ad un minimo.

3.7 Metodo del gradiente coniugato

I principali vantaggi del metodo del gradiente coniugato (metodo preso in prestito dall'analisi numerica) che tratteremo in questa sezione sono i seguenti:

- La rapidità di convergenza è maggiore rispetto al metodo del gradiente;
- Non serve conoscere la matrice Hessiana e non bisogna effettuare operazioni matriciali.

Prima di definire l'iterazioni, diamo la definizione di vettori coniugati: data una matrice Q(mxm) simmetrica e definita positiva, k vettori linearmente indipendenti $d^1, \ldots, d^k \in R^m$ si dicono coniugati rispetto alla matrice Q se risultano soddisfatte le seguenti :

$$\boldsymbol{d}^{i^T} Q \boldsymbol{d}^j = 0 \quad i, j = 1, \ldots, k \quad i \neq j$$

Nel caso di una funzione di errore quadratica della forma

$$E(\boldsymbol{w}) = \frac{1}{2} \boldsymbol{w}^T Q \boldsymbol{w} + \boldsymbol{c}^T \boldsymbol{w}$$

dove Q è la matrice Hessiana simmetrica e definita positiva, il vettore $\boldsymbol{w}^{k+1}$ è dato dalla seguente iterazione:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \eta^k \boldsymbol{d}^k$$

con

$$\boldsymbol{d}^k = \begin{cases} -\nabla E(\boldsymbol{w}^k) & \text{per} \quad k = 0 \\ -\nabla E(\boldsymbol{w}^k) + \beta^k \boldsymbol{d}^{k-1} & \text{per} \quad k > 1 \end{cases}$$

Il passo η^k è ottenuto dall'interazione

$$\eta^k = -\frac{\nabla E(\boldsymbol{w}^k)^T \boldsymbol{d}^k}{(\boldsymbol{d}^k)^T Q \boldsymbol{d}^k}$$

e lo scalare β^k è definito dalla formula

$$\beta^k = \frac{\nabla E(\boldsymbol{w}^k)^T Q \boldsymbol{d}^k - 1}{(\boldsymbol{d}^{k-1})^T Q \boldsymbol{d}^{k-1}}.$$

Altre possibili scelte per la definizione di β^k(che consentono di ottenere una convergenza maggiore) sono la formula di Fletcher e Reeves(FR)

$$\beta^k_{FR} = \frac{\|\nabla E(\boldsymbol{w}^k)\|^2}{\|\nabla E(\boldsymbol{w}^{k-1})\|^2}$$

oppure quella di Polak e Ribière (PR)

$$\beta_{PR}^k = \frac{\nabla E(\boldsymbol{w}^k)^T(\nabla E(\boldsymbol{w}^k - \nabla E(\boldsymbol{w}^{k-1})}{\|\nabla E(\boldsymbol{w}^{k-1})\|^2}.$$

S'è dimostrato che la formula di Fletcher e Reeves soddisfa le seguenti disuguaglianze

$$\begin{cases} E(\boldsymbol{w}^k + \eta^k \boldsymbol{d}^k) < E(\boldsymbol{w}^k) + \gamma\eta^k \nabla E(\boldsymbol{w}^k)\boldsymbol{d}^k \\ \|\nabla E(\boldsymbol{w}^k + \eta^k \boldsymbol{d}^k)^T \boldsymbol{d}^k\| < \beta \|\nabla E(\boldsymbol{w}^k)^T \boldsymbol{d}^k\| \end{cases}$$

con $\gamma < \beta < 1/2$, assicura la proprietà

$$\lim_{k\to\infty} \inf \|\nabla E(\boldsymbol{w}^k)\| = 0.$$

Nella pratica, il metodo di Polak e Ribière risulta essere più efficiente. Se la funzione obiettivo è fortemente convessa, allora il la convergenza globale del metodo PR è assicurata. Nel caso generale si considera una variante del metodo PR, con

$$\beta^k = max(\beta_{PR}^k, 0),$$

la quale assicura la convergenza globale del metodo.

3.8 Metodi Gauss-Newton

Come abbiamo detto nelle sezioni precedenti, una possibile scelta, nonché la più usata, per la misura di errore è la funzione errore quadratica

$$E(\boldsymbol{w}) = \sum_{p=1}^{P} E_p(\boldsymbol{w}) = \frac{1}{2}\sum_{p=1}^{P} \|\boldsymbol{y}^p - \boldsymbol{y}^p(\boldsymbol{x}^p, \boldsymbol{w})\|^2.$$

Definendo il vettore

$$\boldsymbol{e}(\boldsymbol{w}) = (e_1(\boldsymbol{w}), \ldots, e_p(\boldsymbol{w}))$$

con $e_p : \mathbb{R}^n \to \mathbb{R}$ definito da la differenza in norma dell'uscita desiderata $\boldsymbol{y}^p$ e l'uscita fornita dal sistema $\boldsymbol{y}^p(\boldsymbol{x}^p, \boldsymbol{w})$, la funzione E assume la seguente forma

$$E(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{e}(\boldsymbol{w})\|^2.$$

Ponendo $J(\boldsymbol{w}) = \nabla \boldsymbol{e}(\boldsymbol{w})$ ed usando la forma appena ricavata, si ha che

$$\nabla E(\boldsymbol{w}) = J^T(w)\boldsymbol{e}(\boldsymbol{w})$$

che ci fornisce il gradiente di E, mentre per l'Hessiana si avrà la seguente formula

$$\nabla^2 E(\boldsymbol{w}) = \sum_{p=1}^{P} \nabla e_p(w) \nabla e_p(\boldsymbol{w})^T + \sum_{p=1}^{P} \nabla e_p(\boldsymbol{w}) \nabla^2 e_p(\boldsymbol{w}) = J^T(\boldsymbol{w})J(\boldsymbol{w}) + Q(\boldsymbol{w}).$$

Nei metodi di tipo Gauss-Newton e, in generale, nei metodi ai minimi quadrati, si assume che il primo termine della funzione E, ossia $J^T(\boldsymbol{w})J(\boldsymbol{w})$, sia dominante

rispetto a Q($\boldsymbol{w}$), in modo da poter trascurare i termini del secondo ordine. Nel metodo Gauss-Newton si considera la seguente formula ricorsiva

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \boldsymbol{d}_{GN}^k$$

dove $\boldsymbol{d}_{GN}^k$ è soluzione del sistema

$$(J^k)^T J^k \boldsymbol{d} = -(J^k)^T \boldsymbol{e}^k$$

che risulta essere un caso particolare dell'equazione

$$\nabla^2 E(\boldsymbol{w}^k)\boldsymbol{d} = -\nabla E(\boldsymbol{w}^k),$$

che caratterizza la direzione di discesa del metodo di Newton, sotto le ipotesi che $J^T(\boldsymbol{w})J(\boldsymbol{w})$ sia dominante rispetto a Q($\boldsymbol{w}$). I potenziali vantaggi sono i seguenti:

- l'approssimazione $\nabla^2 E^k \approx (J^k)^T J^k$ consente di evitare il calcolo delle singole matrici Hessiane $\nabla^2 e^p$ per $p = 1, \ldots, P$;
- in molte situazioni il termine $J^T J$ è effettivamente dominante, per cui l'efficienza del metodo è paragonabile al metodo di Newton anche se il secondo termine non è considerato nell'approssimazione dell'Hessiana;
- la direzione $\boldsymbol{d}_{GN}^k$ è la soluzione del problema ai minimi quadrati lineari

$$\min_{\boldsymbol{d} \in R^n} \|J\boldsymbol{d} + \boldsymbol{e}^k\|^2,$$

per cui possono essere usati diversi algoritmi risolutivi efficienti (tipo gradiente coniugato). Per assicurare la convergenza globale il metodo di Gauss-Newton è spesso implementato nella seguente forma

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta^k (J(\boldsymbol{w}^k)^T J(\boldsymbol{w}^k) + D^k)^{-1} J(\boldsymbol{w}^k)^T e(\boldsymbol{w}^k),$$

dove η^k è il passo scelto con un opportuno metodo di ricerca unidimensionale (tipo Armijo) e D^k è una matrice diagonale tale che la matrice

$$J(\boldsymbol{w}^k)^T J(\boldsymbol{w}^k) + D^k$$

risulti essere uniforme definita positiva.

3.9 Il metodo di Barzilai-Borwein

Una versione del metodo del gradiente, nota come il metodo di Barzilai-Borwein(BB), risulta promettente per la risoluzione di problemi addestramento "difficili". Il metodo è descritto dalla seguente iterazione

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \frac{1}{\alpha^k} \nabla E(\boldsymbol{w}^k)$$

dove lo scalare α è definito da una delle due formule

$$\alpha_1 = \frac{\boldsymbol{s}^T \boldsymbol{y}}{\boldsymbol{s}^T \boldsymbol{s}} \quad \alpha_2 = \frac{\boldsymbol{y}^T \boldsymbol{y}}{\boldsymbol{s}^T \boldsymbol{y}}$$

con

$$\boldsymbol{s} = \boldsymbol{w}^k - \boldsymbol{w}^{k-1}, \quad \boldsymbol{y} = \nabla E(\boldsymbol{w}^k) - \nabla E(\boldsymbol{w}^{k-1}).$$

Le possibili scelte di α sono legate all'equazione

$$B\boldsymbol{s} = \boldsymbol{y}$$

con B matrice mxm simmetrica definita positiva che approssima la matrice Hessiana $\nabla^2 E(\boldsymbol{w}^k)$.Infatti, gli scalari α_1 e α_2 minimizzano, rispettivamente, gli errori $\|\alpha I\boldsymbol{s}-\boldsymbol{y}\|$ e $\|\boldsymbol{s}-(1/\alpha)I\boldsymbol{y}\|$, qualora si assuma $B = \alpha I$. Nel caso di funzione di errore quadratica il metodo BB può essere visto come un metodo del gradiente (essendo la direzione di discesa data dal gradiente di E) con tasso di apprendimento dato dall'inverso di un'approssimazione di un autovalore della matrice Hessiana. In un caso ideale, ossia se B fosse la matrice Hessiana, si avrebbe una sequenza di autovalori esatti per definire ogni passo k, e l'algoritmo restituirebbe il punto minimo di una funzione quadratica in al più un numero k di iterazioni. Per questo motivo il metodo BB offre buone proprietà locali. La convergenza è stata dimostrata solo nel caso di funzione quadratica convessa. Nel caso generale il metodo potrebbe non convergere; per questo motivo conviene usare una versione modificata del metodo

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta^k \frac{1}{\alpha^k} \nabla E(\boldsymbol{w}^k)$$

dove η^k è calcolato con il metodo di Armijo e soddisfa la condizione di accettabilità

$$E(\boldsymbol{w}^k + \eta \boldsymbol{d}^k) \leq \max_{j \in [0, min(k,M)]} E(\boldsymbol{w}^{k-j}) + \gamma\eta \nabla E(\boldsymbol{w}^k)^T \boldsymbol{d}^k$$

dove $\boldsymbol{d}^k = -(1/\alpha^k)\nabla E(\boldsymbol{w}^k)$ e α^k è dato dal metodo BB (eventualmente modificato).

Il metodo generalizzato consente di migliorare notevolmente il comportamento del metodo del gradiente ed è competitivo con il metodo del gradiente coniugato anche nei problemi convessi a grande dimensione. Tuttavia esso può richiedere un costo computazionale elevato in problemi difficili in cui può essere necessario un ulteriore rilassamento della non monotonicità.

3.10 Metodi on-line

I metodi on-line si basano sull'aggiornamento del vettore $\boldsymbol{w}$ in corrispondenza di ciascuna funzione E_p, relativa ad ogni campione $(\boldsymbol{x}^p, \boldsymbol{y}^p)$. Nei problemi di apprendimento reali i metodi online risultano essere migliori rispetto a quelli batch, sia perché possono essere usati anche quando i dati vengono acquisiti durante il processo, sia quando non conviene considerare tutta la funzione E data dalla somma di ogni errore relativo al p-esimo campione; infatti, il calcolo esplicito della funzione E e del suo gradiente può risultare inefficiente o, ad esempio, se la cardinalità dell'insieme di addestramento T è molto elevata, o se quest'ultimo è ridondante (ossia contiene termini ripetuti). Da un punto di vista deterministico, i metodi on-line consistono nel decomporre la funzione E e il suo gradiente e nel valutare ogni singola componente generando una sequenza di P iterazioni (chiamata "epoca"). Come abbiamo già visto,

il metodo on-line più noto e usato nei problemi di addestramento di una rete neurale è la versione on-line dell'algoritmo del backpropagation che si basa sull'iterazione

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta^k \nabla E_{p(k)}(\boldsymbol{w}^k)$$

in cui, per assicurare la convergenza, si richiede cha il passo di discesa η^k tenda a 0 non troppo rapidamente, ossia che soddisfi le seguenti condizioni

$$\sum_{k=0}^{\infty} \eta^k = +\infty, \quad \sum_{k=0}^{\infty} (\eta^k)^2 < +\infty.$$

Una possibile scelta è data da $\eta^k = c/k$ con $c>0$. La principale limitazione di questo metodo consiste proprio nel fatto che la scelta di un passo di discesa tendente a 0 non troppo rapidamente condiziona la velocità di convergenza, che quindi risulta essere più lenta rispetto ad un metodo batch. Per questo motivo si preferisce usare o tecniche euristiche o metodi misti batch-on-line, in cui la backpropagation on-line BP è utilizzata per una o più epoche ed il passo è ricalcolato periodicamente valutando la funzione di errore complessiva E.

3.11 Metodi di ottimizzazione globali

I metodi di ottimizzazione globale sono stati introdotti per ovviare alle difficoltà costituite dai minimi locali. Tutti gli algoritmi fino a qui trattati convergono a dei minimi locali della funzione, ossia al valore minimo assunto dalla funzione se la si suppone definita su un intorno del punto restituito dagli algoritmi.

Tali algoritmi si suddividono in metodi stocastici e deterministici; i primi consistono nell'approssimare il gradiente della funzione, essendo data dalla somma di più funzioni, da minimizzare alla somma di un numero più piccolo di addendi, in modo da ridurre il tempo di esecuzione, mentre i secondi considerano tutti i termini della funzione da minimizzare. Tra i primi rientrano gli algoritmi on-line, poiché sostituiscono la funzione E con un su addendo E_p, mentre tra i secondi rientrano gli algoritmi batch. Sono inoltre stati sviluppati dei metodi ibridi che uniscono la velocità di convergenza al "minimo desiderato" con la possibilità di evitare minimi locali.

I principali metodi ibridi si basano sul concetto di omotopia, detti metodi di omotopia, che deformano la funzione di errore originaria in modo da ottenere una funzione che sia più semplice da minimizzare. Il metodo "Expanded Range Approximation" (ERA) fa parte di questa classe. Innanzitutto si definisce la media delle uscite data da

$$\boldsymbol{y}_m = \frac{1}{P} \sum_{p=1}^{P} \boldsymbol{y}^p$$

e poi tale valore viene modificato dalla seguente formula

$$\boldsymbol{y}^p(\lambda) = y_m + \lambda(\boldsymbol{y}^p - \boldsymbol{y}_m)$$

con $\lambda \in [0, 1]$. Sperimentalmente il metodo è risultato efficiente nella risoluzione di problemi di apprendimento.

Un'altra tecnica di omotopia consiste nell'introdurre un termine che renda non lineare la funzione di attivazione. Tale omotopia è definita dalla trasformazione

$$g(t, \lambda) = (1 - \lambda)t + \lambda tanh(t)$$

con $\lambda \in [0, 1]$

3.12 Principali difetti degli algoritmi di apprendimento

I principali difetti della maggior parte degli algoritmi fin qui trattati sono:

- Sparizione ed esplosione del gradiente: Le reti neurali multistrato sono inclini a far svanire (o a far esplodere) il gradiente a causa del modo con cui vengono calcolate le derivate parziali livello per livello in una maniera a "cascata", con cui ogni livello contribuisce a incrementare o diminuire le derivate. Allo stesso modo, i pesi vengono aumentati o diminuiti in base ai gradienti in modo da ridurre la funzione costo;

- Minimi locali: i minimi locali sono sempre dei minimi globali per funzioni convesse. Per funzioni non convesse i metodi basati sul metodo del gradiente sono vulnerabili a convergere prematuramente ad un minimo locale;

- Regioni piatte: come per i minimi locali, i metodi basati sul metodo del gradiente potrebbe spostarsi su zone piatte in cui si trovano punti di sella ossia punti in cui la funzione cambia convessità;

- Tempo di addestramento: molti modelli richiedono un tempo esorbitante ed un grande insieme di addestramento, rendendo oneroso il tempo computazionale. Molto spesso i campioni dell'insieme di addestramento non aggiungono valore al processo di apprendimento ed introducono disturbo al processo.

- Eccesso: se aggiungiamo più neuroni alla rete neurale, la rete può indubbiamente modellare più problemi complessi. La rete neurale può prestarsi maggiormente all'insieme di training. Ma c'è un alto rischio di eccedere a valori anomali e disturbi nell'insieme di addestramento. Questo può incrementare il costo computazionale per addestrare la rete e comportare una previsione di qualità inferiore a quella attesa.

3.13 Ottimizzazione degli algoritmi di apprendimento

Gli algoritmi di apprendimento mirano a raggiungere l'obiettivo riducendo il costo della funzione di errore. La causa principale di tre svantaggi degli algoritmi consiste nel fatto che i metodi di apprendimento assumono che la funzione sia convessa. Un altro problema è l'alto numero di nodi e archi che condiziona il numero dei pesi. I principali miglioramenti per le tecniche di apprendimento sono:

- Tecnica di inizializzazione dei parametri: i parametri della rete (ossia i pesi del grafo) vengono inizializzati in maniera casuale, supponendo che siano delle

variabili aleatorie uniformemente distribuite secondo la regola

$$w \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

- Tasso di apprendimento adattivo: questa tecnica consente di cambiare il tasso di apprendimento per i parametri in risposta al gradiente. A sua volte si divide in
  - Algoritmo Delta-bar:il tasso di apprendimento viene incrementato se la derivata parziale rispetto ad esso mantiene lo stesso segno, altrimenti viene diminuito;
  - AdaGrad: prescrive un ridimensionamento inversamente proporzionale dei tassi di apprendimento alla radice quadrata del gradiente quadrato cumulativo.
- Normalizzazione batch: gli input vengono aggiornati in modo che la distribuzione degli input abbia una varianza unitaria ed una media nulla;
- Pre-addestramento supervisionato: un problema di addestramento complesso viene diviso in parti più piccole che vengono allenate e dopo combinate per risolvere il problema più grande;
- Tecnica del dropout: le unità vengono scelte in maniera casuale e vengono annullati i pesi e le uscite così che non influenzino il metodo di backpropagation. Ciò permette di ridurre il rischio di eccesso e di accelerare il processo di addestramento.

Bibliografia

[1] L.Grippo, M. Sciandrone, *Metodi di Ottimizzazione per le Reti Neurali (con tutte le citazioni)*, Aracne Editrice.

[2] C. M. Bishop (2006), *Pattern Recognition and Machine Learning*, Springer.

[3] A. Shrestha, A. Mahmood (2019), *Review of Deep Learning Algorithms and Architectures*, IEEE Access.

[4] A. Lesne (2006), *Complex Networks:from Graph Theory to Biology*, Letters in Mathematical Physics.

[5] O. K. Ernst, *Stochastic Gradient Descent Learning and the Backpropagation Algorithm*,

[6] A. Pinkus (1999), *Approximation theory of the MLP model in neural networks*, Acta Numerica