Università degli Studi di Milano Bicocca

DIPARTIMENTO DI MATEMATICA E APPLICAZIONI
Corso di Laurea Triennale in Matematica

L'algoritmo di Levenberg-Marquardt e il Deep Learning

Relatore:
Prof. Lourenco Beirao Da Veiga

Candidato:
Federica Madon
Matricola 825628

Anno Accademico 2020/2021

A nonno Giancarlo e a Luna.

Indice

Sommario

L'argomento principale di questa tesi è l'algoritmo di **Levenberg-Marquardt** (**LM**, capitolo **3**) in relazione alle sue applicazioni alle **reti neurali**. LM è un algoritmo iterativo, usato per la soluzione di problemi di ottimizzazione libera in forma di minimi quadrati non-lineari. Questo metodo trova solitamente applicazioni nei problemi che riguardano la costruzione di una curva, o di una funzione, che abbia la migliore corrispondenza con dei punti assegnati precedentemente.

In generale, l'algoritmo ha l'obiettivo di minimizzare la somma di quadrati

$$F(x) = \|f(x)\|^2 = f^T(x)f(x),$$

dove f è un vettore n-dimensionale, di funzioni regolari f_i, delle variabili indipendenti x_1, x_2, ..., x_p con $p < n$.

I vettori di discesa $h_i(\gamma)$, $i = 1, 2, \ldots$, sono scelti risolvendo, ad ogni passo, il problema dei minimi quadrati lineari, che minimizza $\|r_i(\gamma)\|^2$, dove

$$\begin{bmatrix} f(x_i) \\ 0 \end{bmatrix} + \begin{bmatrix} A_i \\ \gamma B_i \end{bmatrix} h_i(\gamma) = r_i(\gamma),$$

e $A_i = \nabla_x f(x_i)$, B_i è una matrice $p \times p$ di rango massimo (si può quindi assumere $\|B_i\| = 1$ e $\|B_i^{-1}\| \leq \alpha$ dove con $\|\cdot\|$ ci si riferisce alla norma spettrale) e $\gamma \geq 0$ è un parametro, variabile al passo, che regola la grandezza e la direzione della h_i.

Questo algoritmo è molto interessante poiché può essere interpretato come un metodo di *addestramento supervisionato*; risulta molto efficiente per l'addestramento di reti neurali artificiali di piccole dimensioni.

Le **reti neurali artificiali** (capitolo **2**) sono modelli di calcolo matematico-informatici composti da *neuroni artificiali*, basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni. I neuroni ricevono in ingresso una combinazione dei segnali provenienti dall'esterno o dalle altre unità e ne effettuano una trasformazione tramite una funzione, tipicamente non-lineare, detta *funzione di attivazione*. L'uscita di ciascun neurone viene poi inviata agli altri nodi oppure direttamente all'uscita della rete, attraverso connessioni orientate e pesate. Le reti neurali sono formate da strati, detti *layer*, e sono lo strumento principale del Deep Learning.

Il **Deep Learning** (capitolo **1**) è una tecnica di apprendimento automatico che insegna ai computer a svolgere un'attività naturale per l'uomo: imparare con l'esempio. Precisamente, l'apprendimento profondo è un sottoinsieme del Machine Learning

che programma una "macchina" a classificare autonomamente i dati ed a strutturarli gerarchicamente, trovando quelli più rilevanti e utili alla risoluzione di un problema, migliorando le proprie prestazioni con l'apprendimento continuo. Nei processi di Deep Learning la funzione da minimizzare, cioè la funzione obiettivo, definita come *funzione costo*, è potenzialmente molto complicata, ma la sua matrice jacobiana si può calcolare efficacemente, per esempio, con il **metodo di Backpropagation** (paragrafo **2.4**).

Nell'applicazione dell'algoritmo di Levenberg-Marquardt all'addestramento di reti neurali artificiali la funzione da minimizzare è quella di costo, che ha, infatti, una forma di tipo quadratico non-lineare. Supponendo di avere N dati come input in $\mathbb{R}^{n_1}$, $\{x^{\{i\}}\}_{i=1}^{N}$, per ognuno dei quali sono dati degli output target $\{y(x^{\{i\}})\}_{i=1}^{N}$ in $\mathbb{R}^{n_L}$, allora la funzione ha la forma:

$$Cost = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2,$$

dove $a^{[L]} \in \mathbb{R}^{n_L}$ è l'output della rete, calcolato mediante la funzione di attivazione.

La tesi è organizzata come segue. Nel *capitolo 1* l'argomento principale trattato è il Deep Learning, il suo funzionamento, le sue basi e applicazioni. Il *capitolo 2* introduce, più dettagliatamente, le reti neurali artificiali da un punto di vista matematico. Questa parte approfondisce, quindi, il concetto di funzione di attivazione e presenta l'assetto generale di una rete, con l'aiuto di un esempio. Il *terzo capitolo* illustra nei dettagli l'algoritmo di Levenberg-Marquardt, parte con un'introduzione generale arrivando poi ad analizzare il metodo dal punto di vista teorico, dimostrandone proprietà di convergenza globale, sotto opportune ipotesi. In questo capitolo è anche presente una lista dei punti chiave di come opera, in generale, LM. Infine, il *capitolo 4* consiste in test numerici del metodo di un problema modello di regressione non-lineare (usando il linguaggio **MATLAB**).

Introduzione

Molti di noi, almeno una volta, hanno sentito parlare di **Deep Learning** (capitolo **1**), un insieme di strumenti che sono diventati estremamente popolari in una vasta gamma di campi applicativi, dal riconoscimento di immagini, riconoscimento vocale ed elaborazione del linguaggio naturale alla pubblicità mirata e alla scoperta di farmaci. Al centro di questa rivoluzione del Deep Learning ci sono concetti familiari della matematica applicata e computazionale; in particolare, nel calcolo, nella teoria dell'approssimazione, nell'ottimizzazione e nell'algebra lineare.

Il Deep Learning è una sottocategoria del Machine Learning e indica una branca dell'Intelligenza Artificiale che fa riferimento agli algoritmi ispirati alla struttura e alla funzione del cervello chiamate **reti neurali artificiali** (capitolo **2**). Queste unità computazionali, i neuroni, sono connesse tra di loro con dei pesi variabili e delle costanti, che svolgono una funzione simile a quella delle sinapsi. I pesi e le costanti vengono aggiornati, attraverso un metodo di apprendimento; uno dei più usati, nel caso di reti di piccola o moderata dimensione, è il *metodo di Levenberg-Marquardt* (capitolo **3**), un algoritmo di ottimizzazione usato per la soluzione di problemi in forma di minimi quadrati non-lineari. Nei processi di Deep Learning la funzione da minimizzare viene linearizzata con un polinomio di Taylor, nel quale la matrice jacobiana si può calcolare, per esempio, con il *metodo di Backpropagation* (paragrafo **2.4**).

Capitolo 1

Il Deep Learning

Questo capitolo introduce il concetto di **Deep Learning** e il suo funzionamento con le possibili applicazioni.

1.1 Cos'è il Deep Learning?

Il Deep Learning è una tecnica di apprendimento automatico che insegna ai computer a svolgere un'attività naturale per l'uomo: imparare con l'esempio.

La traduzione letterale è: "**apprendimento profondo**".

Il Deep Learning è un sottoinsieme del *Machine Learning* (apprendimento automatico) e indica quella branca dell'Intelligenza Artificiale che fa riferimento agli algoritmi, ispirati alla struttura e alla funzione del cervello, chiamati **reti neurali artificiali** (capitolo **2**).

Raccogliendo le diverse interpretazioni, si potrebbe definire l'apprendimento profondo come un sistema che sfrutta una classe di algoritmi del Machine Learning che:

1. usano *vari livelli di unità non-lineari a cascata* per svolgere compiti di estrazione di caratteristiche e di trasformazione. Ciascun livello successivo utilizza l'uscita del livello precedente come input. Gli algoritmi possono essere sia tipo *supervisionato* sia *non supervisionato* e le applicazioni includono l'analisi di pattern (apprendimento non supervisionato) e la classificazione (apprendimento supervisionato);

2. sono basati *sull'apprendimento non supervisionato di livelli gerarchici multipli di caratteristiche (e di rappresentazioni) dei dati.* Le caratteristiche di più alto livello vengono derivate da quelle di livello più basso per creare una rappresentazione gerarchica;

3. fanno parte della più ampia classe di algoritmi di apprendimento della rappresentazione dei dati all'interno dell'apprendimento automatico;

4. apprendono multipli livelli di rappresentazione che corrispondono a differenti livelli di astrazione; questi livelli formano una *gerarchia di concetti.*

Applicando il Deep Learning, si avrà quindi una "macchina" che riesce autonomamente a classificare i dati ed a strutturarli gerarchicamente, trovando quelli più rilevanti e utili alla risoluzione di un problema (esattamente come fa la mente umana), migliorando le proprie prestazioni con l'apprendimento continuo.

1.2 La base del Deep Learning: "Le reti neurali artificiali"

L'apprendimento profondo basa il suo funzionamento sulla classificazione e "selezione" dei dati più rilevanti per giungere ad una conclusione, esattamente come fa il nostro cervello biologico che, per formulare una risposta ad un quesito, dedurre un'ipotesi logica, arrivare alla risoluzione di un problema, mette in moto i propri neuroni biologici e le connessioni neurali.

Il Deep Learning sfrutta le **reti neurali artificiali**, modelli di calcolo matematico-informatici composti da neuroni artificiali, basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni. Una rete neurale di fatto si presenta come un sistema "adattivo" in grado di modificare la sua struttura (i nodi e le interconnessioni), basandosi sia su dati esterni sia su informazioni interne che si connettono e passano attraverso la rete neurale durante la fase di apprendimento e ragionamento.

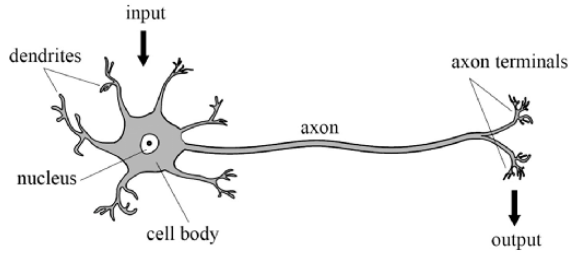

Figura 1.1: Neurone biologico.

Una rete neurale biologica riceve dati e segnali esterni, questi vengono elaborati in informazioni attraverso un imponente numero di neuroni interconnessi tra loro in una struttura non-lineare e variabile in risposta a quei dati e stimoli esterni stessi. È in quest'ottica che si parla dunque di "modello" matematico-informatico. Allo stesso modo, le reti neurali artificiali sono strutture non-lineari di dati, organizzate come strumenti di modellazione: ricevono segnali esterni su uno strato di nodi; ognuno di questi "*nodi di ingresso*" è collegato a svariati nodi interni della rete che, tipicamente, sono organizzati a più livelli in modo che ogni singolo nodo possa elaborare i segnali ricevuti trasmettendo ai livelli successivi il risultato delle sue elaborazioni. I neuroni sono connessi tra di loro con dei pesi variabili e delle costanti, che svolgono una funzione simile alle sinapsi.

In linea di massima, le reti neurali sono formate da tre strati:

1. lo **strato degli ingressi (I - Input)**: è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete;

2. il cosiddetto **strato H - hidden (strato nascosto)**: è quello che ha in carica il processo di elaborazione vero e proprio;

3. lo **strato di uscita (O - Output)**: qui vengono raccolti i risultati dell'elaborazione dello strato H e vengono adattati alle richieste del successivo livello-blocco della rete neurale.

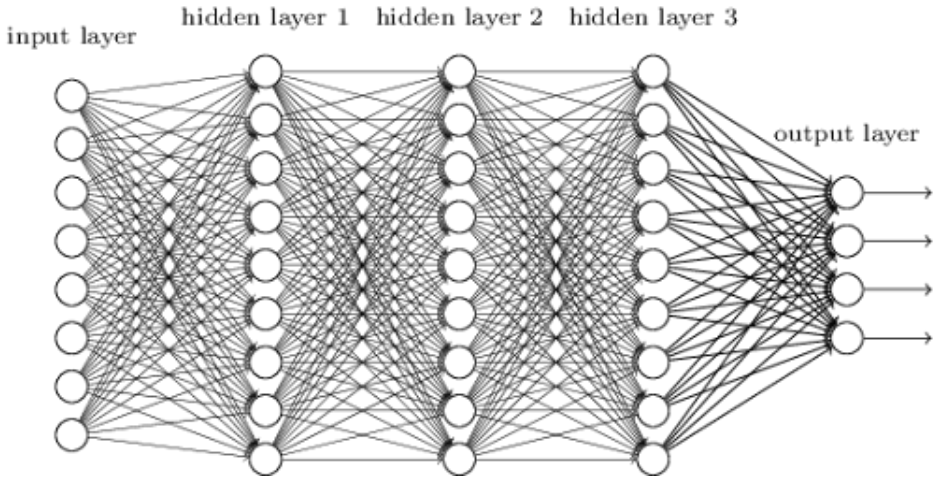

Figura 1.2: Esempio di composizione di una rete neurale artificiale.

1.3 Il funzionamento

Con il Deep Learning vengono simulati i processi di apprendimento del cervello biologico attraverso sistemi artificiali (le reti neurali artificiali, appunto) per insegnare alle macchine non solo ad apprendere autonomamente ma a farlo in modo più "profondo" come sa fare il cervello umano dove profondo significa "su più livelli" (vale a dire sul numero di layer nascosti nella rete neurale - chiamati **hidden layer**: quelle "tradizionali" contengono 2-3 layer, mentre le reti neurali profonde possono contenerne una quantità superiore) con lo scopo di costruire più livelli di astrazione.

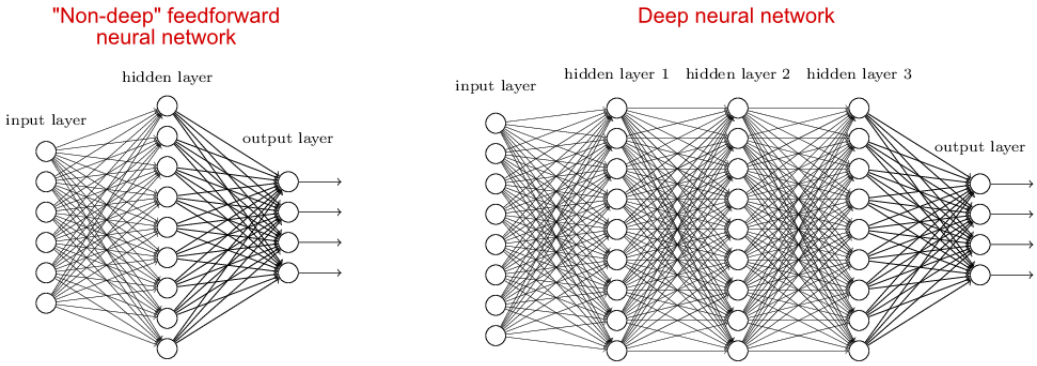

Figura 1.3: Differenza tra reti neurali profonde e non.

La conoscenza è acquisita dall'ambiente attraverso un processo adattivo di apprendimento, ed è immagazzinata nei parametri della rete, nei pesi e nelle costanti associati alle connessioni. I neuroni, che si possono vedere come nodi di una rete orientata provvisti di capacità di elaborazione, ricevono in ingresso una combinazione dei segnali provenienti dall'esterno o dalle altre unità e ne effettuano una trasformazione tramite una funzione, tipicamente non-lineare, detta **funzione di attivazione** (paragrafo **2.1**). L'uscita di ciascun neurone viene poi inviata agli altri nodi oppure direttamente all'uscita della rete, attraverso connessioni orientate e pesate. Una rete neurale, che può essere costituita da un sistema fisico dotato di un elevato grado di parallelismo e di

un'elevata connettività o, più comunemente, da un modello matematico che ne simuli il comportamento, consente di approssimare, in uno specifico contesto applicativo, la corrispondenza tra un ingresso e un'uscita di natura opportuna.

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 nonlinearità;
- dall'*architettura della rete*, ossia dal numero di nodi, dalla struttura e dall'orientamento delle connessioni;
- dai *valori dei parametri interni* associati alle unità elementari e alle connessioni, che devono essere determinati attraverso tecniche di apprendimento.

I molteplici livelli di astrazione possono dare alle reti neurali profonde un vantaggio enorme nell'imparare a risolvere complessi problemi di riconoscimento di schemi, proprio perché ad ogni livello intermedio aggiungono informazioni e analisi utili a fornire un output affidabile. È abbastanza facile intuire che quanti più livelli intermedi ci sono in una rete neurale profonda tanto più efficace è il risultato ma, di contro, la scalabilità della rete neurale è strettamente correlata ai data set, ai modelli matematici e alle risorse computazionali.

Seppur la richiesta di capacità computazionali enormi e, quindi, di costo elevato possa rappresentare un limite, i sistemi di Deep Learning migliorano le proprie prestazioni all'aumentare dei dati.

1.4 Addestrare un sistema di Deep Learning

Gli algoritmi di apprendimento automatico tradizionali sono lineari, gli algoritmi di apprendimento profondo sono impilati in una gerarchia di crescente complessità e astrazione.

Questi sistemi si basano, sostanzialmente, su un processo di apprendimento "**trial-and-error**", ma perché l'output finale sia affidabile sono necessarie enormi quantità di dati. Solitamente, nei sistemi di Deep Learning, l'unica accortezza degli scienziati è "*etichettare*" i dati senza spiegare al sistema come riconoscerne uno particolare: è il sistema stesso, attraverso livelli gerarchici multipli, che intuisce cosa caratterizza un particolare dato e quindi come riconoscerlo. Perciò, per l'accuratezza dell'output, almeno nella prima fase di addestramento, l'utilizzo di dati non strutturati potrebbe rappresentare un problema. I dati non strutturati possono essere analizzati da un modello di apprendimento profondo una volta formato e raggiunto un livello accettabile di accuratezza, ma non per la fase di training del sistema. L'insieme di dati etichettati (insieme di addestramento o **training set**) è costituto da un insieme di coppie ingresso-uscita, che si possono interpretare come esempi della relazione funzionale che si vuole approssimare. Una rete addestrata sulla base dei campioni dell'insieme di addestramento deve essere poi in grado di generalizzare, ossia di dare la risposta corretta in corrispondenza a ingressi non considerati nel training set e ciò costituisce l'uso applicativo della rete in problemi di classificazione o di regressione.

Si possono distinguere due paradigmi fondamentali di apprendimento:

- apprendimento **non supervisionato**, in cui i campioni di uscita non sono noti (oppure non vengono utilizzati), e i parametri della rete vengono determinati attraverso tecniche di clustering applicate ai soli campioni di ingresso;
- apprendimento **supervisionato**, in cui i parametri della rete vengono determinate tenendo conto anche delle uscite relative ai campioni di training.

Il problema della costruzione del modello si può formulare come un *problema di approssimazione funzionale*; la classe delle funzioni approssimanti è costituita dalle funzioni realizzabili dalla rete, al variare dei parametri interni. La specificità delle reti neurali, nell'ambito dei metodi di approssimazione di tipo più generale, sta nella particolare scelta delle funzioni approssimanti, che dipendono tipicamente in modo non-lineare dai parametri, e nel fatto che i dati, rappresentati dalle coppie del training set, sono discreti. Tali caratteristiche, che sono spesso alla base dei notevoli successi applicativi delle reti neurali, pongono difficili problemi di analisi e calcolo.

In termini generali, il problema dell'apprendimento è quello di definire in modo "ottimo" la complessità di un modello e il procedimento di identificazione dei parametri a partire dai dati, in modo da ottenere la migliore capacità di generalizzazione possibile. È, infatti, evidente che un modello troppo "semplice" potrebbe non essere in grado di descrivere con sufficiente accuratezza il fenomeno in esame, mentre un modello troppo "complesso" potrebbe portare a interpolare esclusivamente i dati di training, a scapito della capacità di generalizzazione. Ciò implica che la complessità del modello deve essere definita opportunamente, tenendo conto delle ipotesi a priori sul processo da identificare e della cardinalità del training set.

Quale sia l'impostazione concettuale a cui si faccia riferimento, gli algoritmi di addestramento per il calcolo dei parametri dei modelli neurali si basano invariabilmente sulla formulazione di un *problema di ottimizzazione* consistente nella minimizzazione di una funzione di errore di struttura opportuna rispetto ai parametri della rete. L'efficienza degli algoritmi di ottimizzazione nella soluzione dei problemi di addestramento condiziona, quindi, l'applicabilità dei modelli neurali.

Inoltre, i sistemi basati su Deep Learning sono difficili da addestrare anche a causa del numero stesso di strati della rete neurale. Il numero di strati e collegamenti tra i neuroni nella rete è tale che le "regolazioni", che devono essere apportate in ogni fase del processo di addestramento, possono diventare difficili; per questo per il training si usano spesso i cosiddetti algoritmi di retropropagazione dell'errore (**Backpropagation** paragrafo **2.4**) attraverso i quali si rivedono i pesi e le costanti della rete neurale, in caso di errori. Un processo che continua in modo iterativo finchè il gradiente è nullo.

1.5 Le possibili applicazioni

Già oggi ci sono casi d'uso ed ambiti di applicazione che possiamo notare anche come "comuni cittadini" non esperti di tecnologia.

Il Deep Learning è una tecnologia fondamentale alla base delle automobili a guida autonoma, poichè consente loro di riconoscere un segnale di stop o di distinguere un pedone da un lampione. É il fattore chiave del controllo vocale in dispositivi quali telefoni cellulari, tablet, TV e altoparlanti vivavoce. Si trovano delle applicazioni anche

in robot droni impiegati per la consegna di pacchi o anche per l'assistenza in casi di emergenza. Le architetture del Deep Learning, inoltre, sono molto utilizzate nel riconoscimento delle immagini per aiutare i radiologi a individuare i tumori nei raggi X, oppure per aiutare i ricercatori a individuare le sequenze genetiche correlate alle malattie e identificare le molecole che potrebbero portare a farmaci più efficaci o addirittura personalizzati; nella sorveglianza per il riconoscimento facciale e per il riconoscimento e la sintesi vocale e linguistica per chatbot e robot di servizio. L'apprendimento profondo è ugualmente impiegato nei sistemi di analisi per la manutenzione predittiva su un'infrastruttura o un impianto; in bioinformatica e nell'identificazione delle frodi.

Guardando invece ai tipi di applicazione (intesi come compiti che una macchina può svolgere grazie al Deep Learning), quelli di seguito sono quelli ad oggi più maturi:

- colorazione automatica delle immagini in bianco e nero;
- aggiunta automatica di suoni a filmati silenziosi;
- traduzione simultanea;
- classificazione degli oggetti all'interno di una fotografia;
- generazione automatica della grafia;
- generazione automatica di testo;
- generazione automatica di didascalie.

Capitolo 2

Le reti neurali artificiali in matematica

Nel capitolo **1** sono state introdotte le **reti neurali artificiali** da un punto di vista pratico. In questo capitolo verrà presentato l'assetto generale di una rete neurale artificiale, addestrata con un metodo di apprendimento *supervisionato*.

2.1 La funzione di attivazione

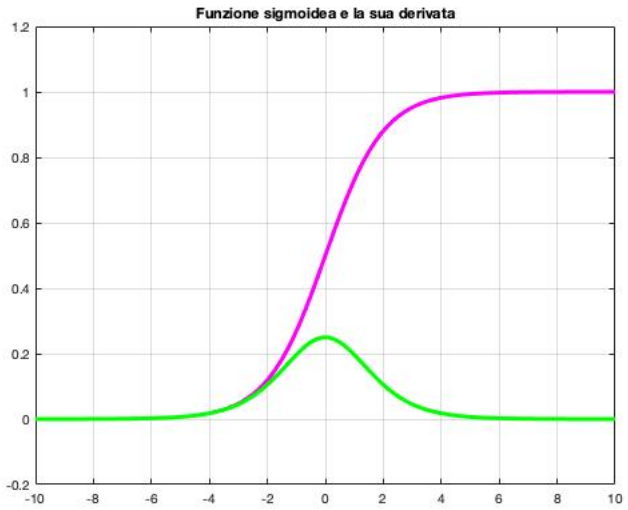

Figura 2.1: In magenta la funzione sigma e in verde la sua derivata.

L'approccio di una rete neurale artificiale consiste nell'utilizzo ripetuto di una semplice funzione non-lineare, detta **funzione di attivazione**, indicata con $f(x)$. Per esempio, si può considerare la *funzione sigma* (vedi figura **2.1**)

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

La funzione di attivazione imita il comportamento di un neurone nel cervello; riceve un vettore di *ingressi* x, fornendo in corrispondenza un'*uscita* scalare $y(x)$. Questa funzione deve soddisfare alcune proprietà:

- deve avere valori di output compresi nell'intervallo $\{0, 1\}$ oppure $\{-1, 1\}$;
- si attiva (dando un output uguale a 1) se l'input è sufficientemente grande, per propagare l'attività all'interno della rete;
- rimane inattiva (dando un output uguale a 0 o -1) se, invece, l'input non è abbastanza grande;
- ha la derivata di forma semplice, cioè $f'(x) = f(x)(1 - f(x))$ (in figura **2.1** è rappresentata la derivata della funzione $\sigma(x)$).

La funzione può essere modificata (fig. **2.2**) applicandole una trasformazione, per esempio, traslando l'argomento o facendo una dilatazione; nel linguaggio delle reti neurali, aggiungendo pesi e costanti all'input.

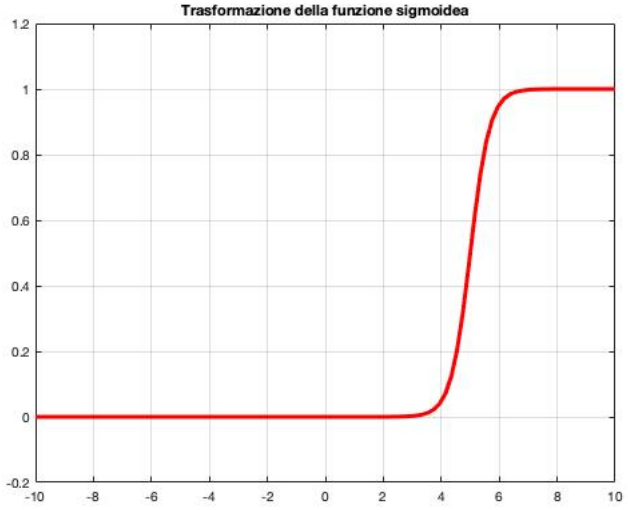

Figura 2.2: Trasformazione della funzione sigma: $\sigma(3(x - 5))$.

Per semplicità conviene interpretare la funzione di attivazione in senso vettoriale. Per $z \in \mathbb{R}^m$, $f : \mathbb{R}^m \to \mathbb{R}^m$ è definita nel modo seguente:

$$(f(z))_i = f(z_i).$$

2.2 Assetto generale

In un qualsiasi layer, ogni neurone riceve lo stesso input, un valore reale emesso da ogni neurone del precedente strato. Ciascun ricevente forma la sua combinazione pesata di questi valori, aggiunge le sue costanti e, applica la funzione di attivazione, producendo un output che invierà al livello successivo.

Ci sono due layers particolari: il primo, cioè il *layer di input*, non ha alcuno strato precedente e ogni suo neurone riceve un vettore di input, l'ultimo, cioè il *layer di output*, non ha alcuno strato successivo e i suoi neuroni forniscono l'output complessivo. I layers compresi tra il primo e l'ultimo si chiamano *hidden layers*.

Si consideri una rete neurale con L layers, dove i layers 1 e L sono lo strato di input e lo strato di output, rispettivamente. Si supponga che il layer l, per $l = 1, 2, 3, \ldots, L$, contenga n_l neuroni, così n_1 è la dimensione dei dati di input. Complessivamente, la rete mappa $\mathbb{R}^{n_1}$ in $\mathbb{R}^{n_L}$.

$W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$ denota la matrice dei pesi al livello l. Più precisamente, $w^{[l]}_{jk}$ è il peso che il neurone j del layer l applica all'output del neurone k dello strato $l-1$. Allo stesso modo, $b^{[l]} \in \mathbb{R}^{n_l}$ è il vettore delle costanti per il layer l, così che il neurone j dello strato l usa le costanti $b^{[l]}_j$.

In figura **2.3** è mostrato un esempio di una rete neurale artificiale formata da cinque layers, quindi $L = 5$. In questo caso $n_1 = 4$, $n_2 = 3$, $n_3 = 4$, $n_4 = 5$ e $n_5 = 2$, così $W^{[2]} \in \mathbb{R}^{3\times4}$, $W^{[3]} \in \mathbb{R}^{4\times3}$, $W^{[4]} \in \mathbb{R}^{5\times4}$, $W^{[5]} \in \mathbb{R}^{2\times5}$, $b^{[2]} \in \mathbb{R}^3$, $b^{[3]} \in \mathbb{R}^4$, $b^{[4]} \in \mathbb{R}^5$ e $b^{[5]} \in \mathbb{R}^2$.

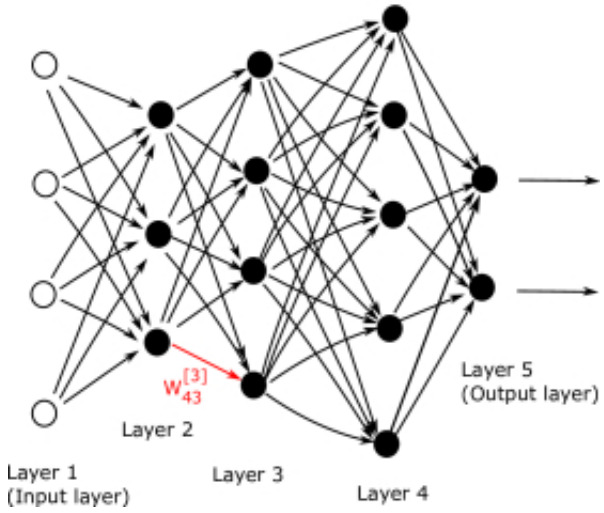

Figura 2.3: Una rete con cinque layers. Il bordo corrispondente al peso $w^{[3]}_{43}$ è evidenziato. L'output del neurone 3 dello strato 2 è pesato dal fattore $w^{[3]}_{43}$ quando è immesso nel neurone numero 4 dello strato 3.

Fornendo come input una $x \in \mathbb{R}^{n_1}$, si può, quindi, riassumere ordinatamente l'azione della rete lasciando che $a^{[l]}_j$ denoti l'output, o l'attivazione, dal neurone j del layer l. Così, si ha:

$$a^{[1]} = x \in \mathbb{R}^{n_1}, \tag{2.1}$$

$$a^{[l]} = f(W^{[l]}a^{[l-1]} + b^{[l]}) \in R^{n_l}, \text{per } l = 2, 3, \ldots, L. \tag{2.2}$$

Dovrebbe essere chiaro che (2.1) e (2.2) equivalgono ad un algoritmo che "alimenta" l'input in avanti, attraverso la rete, per produrre un output $a^{[L]} \in \mathbb{R}^{n_L}$; praticamente consistono nela **funzione "associata"**.

Ora si supponga di avere N dati come input in $\mathbb{R}^{n_1}$, $\{x^{\{i\}}\}_{i=1}^N$, per ognuno dei quali sono dati degli *output target* $\{y(x^{\{i\}})\}_{i=1}^N$ in $\mathbb{R}^{n_L}$ (i dati "etichettati" del paragrafo **1.4**). Perciò la **funzione costo** che vogliamo minimizzare ha la forma:

$$Cost = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2, \tag{2.3}$$

dove, per evitare di appesantire la notazione, non si è esplicitamente indicato che la (2.3) è la funzione di tutti i pesi e delle costanti. Il fattore $\frac{1}{2}$ è incluso in quanto si dovrà differenziare. La funzione (2.3), dove la discrepanza è misurata facendo la media della norma euclidea al quadrato sui punti dati, è spesso indicata come una funzione di costo quadratica. Nel linguaggio dell'ottimizzazione, è la **funzione obiettivo**.

Scegliere pesi e costanti che minimizzino la funzione costo è ciò in cui consiste l'*addestramento* di una rete neurale artificiale.

Osservazione 1. Quando si calcola l'output dell'ultimo layer si hanno due casi possibili:

- nei problemi di classificazione, per esempio, dove si richiede una risposta di tipo "decisionale", allora $a^{[L]}$ si calcolerà come i precedenti;
- se il problema richiede come risposta una "quantità", per esempio nel caso di approssimazione di funzioni, allora
$$a^{[L]} = W^{[L]}a^{[L-1]} + b^{[L]}.$$

2.3 Esempio di una rete neurale artificiale

In generale, si consideri il vettore a, contenente i valori reali prodotti dai neuroni di un qualsiasi layer, così che il vettore degli output dello strato successivo avrà la forma:

$$f(Wa + b). \tag{2.4}$$

Mantenendo la notazione del paragrafo **2.2**, W è la matrice dei pesi e b è il vettore delle costanti. Per enfatizzare il ruolo del i-esimo neurone in (2.4), si potrebbe scegliere la i-esima componente come

$$f\Big(\sum_j w_{ij}a_j + b_i\Big),$$

dove la somma è calcolata su tutte le entrate di a.

Si consideri una rete neurale artificiale come quella mostrata in figura **2.4**.

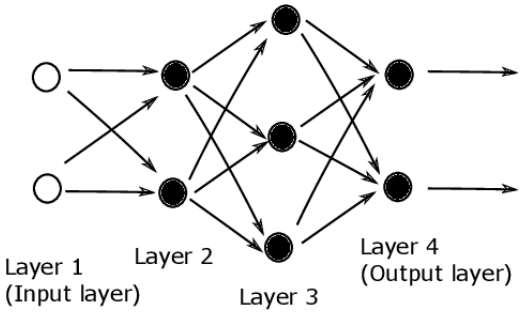

Figura 2.4: Una rete con 4 layers.

Poichè i dati di input hanno la forma $x \in \mathbb{R}^2$, i pesi e le costanti per il secondo layer possono essere rappresentati, rispettivamente, dalla matrice $W^{[2]} \in \mathbb{R}^{2\times 2}$ e dal vettore $b^{[2]} \in \mathbb{R}^2$. L'output del secondo strato avrà, allora, la seguente forma:

$$f(W^{[2]}x + b^{[2]}) \in \mathbb{R}^2.$$

Il layer numero 3 ha tre neuroni, ciascuno dei quali riceve un input in $\mathbb{R}^2$. Così i pesi e le costanti del terzo livello vengono rappresentati con una matrice $W^{[3]} \in \mathbb{R}^{3\times 2}$ e con un vettore $b^{[3]} \in \mathbb{R}^3$, rispettivamente. L'output del terzo strato avrà, allora, la seguente forma:

$$f\Big(W^{[3]}f(W^{[2]}x + b^{[2]}) + b^{[3]}\Big) \in \mathbb{R}^3.$$

Il quarto layer, cioè quello di output, ha due neuroni, ciascuno dei quali riceve un input in $\mathbb{R}^3$. Così i pesi e le costanti di questo livello possono essere rappresentati con una matrice $W^{[4]} \in \mathbb{R}^{2\times 3}$ e con un vettore $b^{[4]} \in \mathbb{R}^2$, rispettivamente. L'output del quarto layer, e quindi di tutta la rete, avrà la seguente forma:

$$F(x) = f\left(W^{[4]}f\Big(W^{[3]}f(W^{[2]}x + b^{[2]}) + b^{[3]}\Big) + b^{[4]}\right) \in \mathbb{R}^2. \tag{2.5}$$

L'espressione (2.5) definisce una funzione $F : \mathbb{R}^2 \to \mathbb{R}^2$.

Si supponga di fornire come input n dati in $\mathbb{R}^2$, indicati con $\{x^{\{i\}}\}_{i=1}^n$; $y(x^{\{i\}})$ indicano gli output target. La funziona costo avrà la forma:

$$Cost\Big(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]}\Big) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\|y(x^{\{i\}}) - F(x^{\{i\}})\|_2^2. \tag{2.6}$$

2.4 Algoritmo Backpropagation

Dato un generico $x^{\{i\}}$, sia

$$C_{x^{\{i\}}} = \frac{1}{2}\|y(x^{\{i\}}) - a^{[L]}(x^{\{i\}})\|_2^2. \tag{2.7}$$

Per rendere la notazione più comprensibile si elimina la dipendenza da $x^{\{i\}}$ e, quindi,

$$C = \frac{1}{2}\|y - a^{[L]}\|_2^2, \tag{2.8}$$

dove si ricorda da (2.2) che $a^{[L]}$ è l'output della rete neurale artificiale. La dipendenza di C dai pesi e dalle costanti deriva solo da $a^{[L]}$.

Per ricavare espressioni utili per le derivate parziali, è utile introdurre due ulteriori insiemi di variabili. Innanzitutto, sia

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}, \text{per } l = 2, 3, \ldots, L. \tag{2.9}$$

$z_j^{[l]}$ è l'input pesato per il neurone j del layer l. Inoltre, (2.2) può essere riscritta come:

$$a^{[l]} = f(z^{[l]}), \text{per } l = 2, 3, \ldots, L. \tag{2.10}$$

Successivamente, sia $\delta^{[l]} \in \mathbb{R}^{n_l}$ la seguente espressione:

$$\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}}, \text{ per } 1 \leq j \leq n_l \text{ e } 2 \leq l \leq L. \tag{2.11}$$

Quest'ultima espressione (2.11), spesso chiamata *errore* al j-esimo neurone del layer l, è una quantità intermedia, utile sia per l'analisi che per la computazione. Tuttavia, chiamare questa quantità errore, crea una certa ambiguità. In un qualsiasi hidden layer non è chiaro quanto "incolpare" ogni neurone per le discrepanze nell'output finale. Inoltre, nello strato finale, L, l'espressione (2.11) non quantifica queste discrepanze direttamente. Considerato che la funzione costo può essere minima solo se ha tutte le derivate parziali nulle, allora sembra utile l'idea di riferirsi a $\delta_j^{[l]}$, in (2.11), come ad un errore, così $\delta_j^{[l]} = 0$ è un risultato desiderabile. In seguito, può essere più utile tenere presente che $\delta_j^{[l]}$ misura la sensibilità della funzione di costo all'input pesato per il neurone j del layer l.

A questo punto è necessario definire il **Prodotto di Hadamard** (o *prodotto entrata per entrata*) di due vettori. Siano $x, y \in \mathbb{R}^n$, allora $x \circ y \in \mathbb{R}^n$ è definito come $(x \circ y)_i = x_i y_i$.
Con questa notazione, i risultati seguenti sono una conseguenza della *chain rule*.

Lemma 1.

$$\delta^{[L]} = f'(z^{[L]}) \circ (a^L - y), \tag{2.12}$$

$$\delta^{[l]} = f'(z^{[l]}) \circ (W^{[l+1]})^T \delta^{[l+1]} \qquad \textit{per } 2 \leq l \leq L-1, \tag{2.13}$$

$$\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]} \qquad \textit{per } 2 \leq l \leq L, \tag{2.14}$$

$$\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]} \qquad \textit{per } 2 \leq l \leq L. \tag{2.15}$$

Dimostrazione. Si veda, ad esempio, la dimostrazione riportata nell'articolo [6]. □

Ci sono molti aspetti del lemma **1** che meritano attenzione. Si ricorda da (2.1), (2.9) e (2.10) che $a^{[L]}$ può essere valutato con un *forward pass* attraverso la rete, calcolando $a^{[1]}$, $z^{[2]}$, $a^{[2]}$, $z^{[3]}$, ..., $a^{[L]}$ in ordine. Una volta calcolato $a^{[L]}$, si vede dalla (2.12), che $\delta^{[L]}$ è immediatamente disponibile. Allora, dalla (2.13), $\delta^{[L-1]}$, $\delta^{[L-2]}$, ..., $\delta^{[2]}$ possono essere calcolati in un *backward pass*. Dalla (2.14) e dalla (2.15), si ha, quindi, acesso alle derivate parziali. Calcolare i gradienti in questo modo è noto come *Back Propagation*.

Capitolo 3

L'Algoritmo di Levenberg-Marquardt

3.1 Introduzione

L'algoritmo di **Levenberg-Marquardt** (**LM**) è un algoritmo iterativo di ottimizzazione, tra quelli di maggior successo, usato per la soluzione di problemi in forma di minimi quadrati non-lineari. Questo metodo trova solitamente applicazioni nei problemi che riguardano la costruzione di una curva, o di una funzione, che abbia la migliore corrispondenza con dei punti assegnati precedentemente.

LM è un algoritmo che combina i vantaggi dei metodi *Quasi-Newton* [8] e di *discesa del gradiente* [6]. Fornisce un metodo, che dipende in modo non-lineare da un parametro γ, per calcolare la correzione al punto corrente.

Questo algoritmo può essere interpretato come un metodo di addestramento *supervisionato*; risulta molto efficiente per l'addestramento di reti neurali artificiali di piccole dimensioni. Per reti neurali con un numero di layer significativo, le dimensioni della matrice jacobiana aumentano considerevolmente la complessità computazionale, rendendo questo metodo praticamente inefficiente.

In generale, i problemi relativi all'addestramento di una rete neurale artificiale sono già stati introdotti nel capitolo **1**. Più precisamente, la funzione costo (2.3) può causare molteplici difficoltà computazionali:

- la funzione può avere una superficie con "valli ripide" e zone "piatte";
- l'Hessiana può essere mal condizionata;
- presenza di un'elevata dimensione dei dati su cui dover lavorare;
- esistenza di minimi locali non globali;
- gli insiemi di livello della funzione possono non essere compatti, non assicurando la convergenza globale di molti algoritmi.

3.2 Caratteristiche principali

In generale, l'algoritmo ha l'obiettivo di minimizzare la somma di quadrati

$$F(x) = \|f(x)\|^2 = f^T(x)f(x), \tag{3.1}$$

dove f è un vettore n-dimensionale, di funzioni regolari f_i, delle variabili indipendenti $x_1, x_2, \ldots, x_p$ con $p < n$.

I vettori di discesa $h_i(\gamma)$, $i = 1, 2, \ldots$, sono scelti risolvendo, ad ogni passo, il problema dei minimi quadrati lineari, che minimizza $\|r_i(\gamma)\|^2$, dove

$$\begin{bmatrix} f(x_i) \\ 0 \end{bmatrix} + \begin{bmatrix} A_i \\ \gamma B_i \end{bmatrix} h_i(\gamma) = r_i(\gamma), \tag{3.2}$$

e $A_i = \nabla_x f(x_i)$, B_i è una matrice $p \times p$ di rango massimo (si può quindi assumere $\|B_i\| = 1$ e $\|B_i^{-1}\| \leq \alpha$ dove con $\|\cdot\|$ ci si riferisce alla norma spettrale [1]) e $\gamma \geq 0$ è un parametro che regola la grandezza e la direzione della h_i.

La norma al quadrato dell'equazione (3.2) si può riscrivere, in forma meno compatta, come:

$$\|r_i\|^2 = f^T(x_i)f(x_i) + h_i^T A_i^T f(x_i) + f^T(x_i)A_i h_i + h_i^T A_i^T A_i h_i + \gamma^2 h_i^T B_i^T B_i h_i,$$

dove la dipendenza di r e di h da γ è stata omessa.

Osservazione 2. Il caso $B_i = I$, insieme ad una strategia adeguata alla scelta di γ, caratterizza la scelta più classica per l'algoritmo di Levenberg-Marquardt. In questo caso, si ottiene:

$$\|r_i\|^2 = f^T(x_i)f(x_i) + h_i^T A_i^T f(x_i) + f^T(x_i)A_i h_i + h_i^T A_i^T A_i h_i + \gamma^2 \|h_i\|^2,$$

dove la dipendenza di r e di h da γ è stata omessa. Da quest'ultima equazione si riesce già a intuire il legame tra γ e h_i; alla crescita del valore di γ corrisponde una diminuzione dei valori di h_i e quindi, ad una direzione di discesa inferiore, perciò l'algoritmo avrà un passo inferiore.

Osservazione 3. La soluzione $h_i(\gamma)$ di (3.2) è sempre univocamente determinata, a parte nel caso in cui $\gamma = 0$ e A ha rango minore di quello completo. In questo caso, h_i verrà calcolato prendendo l'unica soluzione dell'equazione di norma minima.

Un pregio di questo algoritmo è quello di *convergere globalmente*, sotto opportune ipotesi, a partire da un punto iniziale arbitrario, ad un punto stazionario di F.

Nell'applicazione dell'algoritmo all'addestramento di reti neurali artificiali la funzione da minimizzare è quella di **costo** (2.3) che ha, infatti, una forma di tipo quadratico non-lineare (3.1).

3.3 Funzionamento dell'algoritmo

Lo sviluppo dell'algoritmo può essere diviso in tre parti:

1. richiamo della caratterizzazione dei punti stazionari di $F(x)$;

[1] La norma spettrale della matrice M è definita come $\|M\| = \sqrt{\rho(M^*M)}$, dove ρ rappresenta il raggio spettrale di M^*M e M^* la trasposta coniugata di M.

2. introduzione di un teorema di convergenza, assumendo che la successione dei valori $\{F(x_i)\}$ soddisfi un criterio opportuno;

3. dimostrazione che, per x_i fissato, il criterio introdotto prima può sempre essere soddisfatto scegliendo γ sufficientemente grande in (3.2).

Lemma 2. *Sia γ limitata in* (3.2)*, allora le seguenti affermazioni sono equivalenti:*

i. $\begin{bmatrix} f_i \\ 0 \end{bmatrix} = r_i$,

ii. $\|f_i\| = \|r_i\|$,

iii. *x_i è un punto stazionario di $F(x)$.*

Dimostrazione. Si veda, ad esempio, la dimostrazione riportata nell'articolo [10]. □

Teorema 1. *Sia $\{\gamma_i\}$ una successione limitata in* (3.2) *tale che:*

$$0 < \sigma \leq \psi(x_i, \gamma_i), \tag{3.3}$$

dove σ è una costante fissata indipendente da i,

$$x_{i+1} = x_i + h_i(\gamma_i), \tag{3.4}$$

e

$$\psi(x_i, \gamma_i) = \frac{F(x_i) - F(x_{i+1})}{2(F(x_i) - \|r_i\|^2)}. \tag{3.5}$$

Allora la successione $\{F(x_i)\}$ è convergente e i punti limite della successione $\{x_i\}$ sono punti stazionari di $F(x)$.

Dimostrazione. Da (3.2) e ricordando il lemma **2** si può notare che, se x_i non è un punto stazionario di $F(x)$, allora $\|r_i\| < \|f_i\|$ e quindi, la (3.5) implica che $F(x_{i+1}) < F(x_i)$. La (3.3) si può riscrivere come:

$$\begin{aligned} \sigma &\leq \frac{(\|f(x_i)\| + \|f(x_{i+1})\|)(\|f(x_i)\| - \|f(x_{i-1})\|)}{2(\|f(x_i)\| + \|r_i\|)(\|f(x_i)\| - \|r_i\|)} \\ &\leq \frac{\|f(x_i)\| - \|f(x_{i+1})\|}{\|f(x_i)\| - \|r_i\|}, \end{aligned} \tag{3.6}$$

in quanto $\|f(x_i)\| \leq \|f(x_i)\| + \|r_i\|$ e $\|f(x_i)\| + \|f(x_{i+1})\| \leq 2\|f(x_i)\|$. Quindi possiamo riscrivere (3.6) come:

$$\|f(x_{i+1})\| \leq \|f(x_i)\| - \sigma(\|f(x_i)\| - \|r_i\|); \tag{3.7}$$

osserviamo anche che la successione non-negativa $\{\|f(x_i)\|\}$ è convergente, essendo $\|f(x_i)\| \geq \|f(x_{i+1})\|$. Inoltre, da (3.7), si può ricavare che

$$\|f(x_i)\| - \|r_i\| \leq \frac{1}{\sigma}(\|f(x_i)\| - \|f(x_{i+1})\|). \tag{3.8}$$

Siccome il membro destro di questa disuguaglianza tende a zero, applicando il lemma **2**, si ottiene la seconda implicazione del teorema. □

Teorema 2. *Per ogni $\sigma < 1$ e x_i esiste un γ che soddisfa la condizione* (3.3).

Dimostrazione. Si vuole mostrare che:

$$\psi(x_i, \gamma) \to 1 + O\Big(\frac{1}{\gamma^2}\Big) \text{ per } \gamma \to \infty,$$

in modo tale che il test sia soddisfatto per ogni x_i fissato, con la condizione che γ sia scelto abbastanza grande. L'equazione per determinare $h_i(\gamma)$ può essere scritta come:

$$[A_i^T A_i + \gamma^2 B_i^T B_i] h_i(\gamma) = -A_i^T f_i, \tag{3.9}$$

in modo che, avendo B rango massimo,

$$h_i(\gamma) = -\frac{1}{\gamma^2}(B_i^T B_i)^{-1} A_i^T f_i + O\Big(\frac{1}{\gamma^4}\Big). \tag{3.10}$$

Quindi $\|h_i(\gamma)\| \to 0$ quando $\gamma \to \infty$. Con un calcolo diretto si può ottenere:

$$\nabla F(x_i) = 2 f_i^T A_i, \tag{3.11}$$

ricavando il valore di h_i da (3.2) e ricordandosi come si era definito $F(x)$, allora:

$$\begin{aligned} \nabla F(x_i) h_i &= -2(\|f_i\|^2 - f_i^T r_i) \\ &= -2(\|f_i\|^2 - \|r_i\|^2). \end{aligned} \tag{3.12}$$

L'ultima uguaglianza di (3.12) deriva dal fatto che, utilizzando le relazioni (3.11) e (3.9), si può scrivere:

$$\nabla F(x_i) h_i = 2 h_i^T A_i^T f_i = -2 h_i^T [A_i^T A_i + \gamma^2 B_i^T B_i] h_i. \tag{3.13}$$

Considerando come è stato definito r_i, la sua norma si può scrivere come:

$$\|r_i\|^2 = r_i^T r_i = \|f_i\|^2 + h_i^T A_i^T A_i h_i + \gamma^2 h_i^2 B_i^T B_i h_i,$$

che implica

$$\|r_i\|^2 - \|f_i\|^2 = h_i^T [A_i^T A_i + \gamma^2 B_i^T B_i] h_i$$

e, sostituendo infine quest'ultima relazione a (3.13), si ottiene l'uguaglianza cercata. Ora utilizzando lo sviluppo di Taylor:

$$F(x_i) - F(x_{i+1}) = -\nabla F(x_i) h_i, \tag{3.14}$$

sostituendo all'espressione sopra (3.12) e utilizzando (3.10), insieme all'ipotesi che la funzione f_j sia regolare per $j = 1, \ldots, n$, otteniamo

$$\frac{F(x_i) - F(x_{i+1})}{2(F(x_i) - \|r_i\|^2)} = 1 + O\Big(\frac{1}{\gamma^2}\Big) \tag{3.15}$$

quando $\gamma \to \infty$, come si voleva dimostrare. □

Osservazione 4. La disuguaglianza (3.3) fornisce un metodo per scegliere un γ_i adatto, prendendo la decrescita del passo h_i e comparandola con quella prevista dall'equazione (3.2). Il teorema dice, quindi, che la convergenza è una conseguenza del fatto che queste due quantità sono simili, in modo tale che non ci si discosti troppo dalla regione in cui ha luogo l'approssimazione lineare.

Osservazione 5. Il caso particolare $\gamma = 0$ corrisponde al metodo di Newton. Nel caso in cui il metodo sia convergente e $\|f\|$ è sufficientemente piccolo, allora la velocità di convergenza del metodo può risultare molto soddisfacente. Questo suggerisce che risulta più vantaggioso lavorare con una γ piccola il più possibile.

Inoltre, scegliendo una σ piccola, ci sono maggiori possibilità di soddisfare la relazione (3.3) e probabilmente di favorire, anche, che γ abbia valori più piccoli.

Osservazione 6. La discrepanza nel risultato di convergenza globale è dovuta al fatto che il teorema **2** vale per ogni x_i assegnato, ma non dà informazioni riguardanti il $sup_i\gamma_i$, che potrebbe essere non limitato (mentre per applicare il teorema **1** servono anche i γ_i limitati). Quindi, per ora, si può solo affermare che, se $\{\gamma_i\}$ è limitato, allora i punti limite $\{x_i\}$ sono valori stazionari di $F(x)$.

L'elenco seguente mostra i *punti chiave* di come opera in generale il metodo di Levenberg-Marquardt.

1. Fissare i parametri iniziali: x_1, $\gamma_1^{(1)}$, σ, *DECR*, *EXPND*, *TOL*, *GMAX*, $i = 1$.
2. Porre $j = 1$ e calcolare f_i, A_i, B_i.
3. Determinare $h_i(\gamma_i^{(j)})$ e $\psi(x_i, \gamma_i^{(j)})$.
4. Se $\sigma \leq \psi(x_i, \gamma_i^{(j)})$ allora andare al punto 5, altrimenti porre $j = j + 1$, $\gamma_i^{(j)} = EXPND * \gamma_i^{(j-1)}$ e tornare al punto 3.
5. $\gamma_i = \gamma_i^{(j)}$ e $x_{i+1} = x_i + h_i(\gamma_i)$.
6. Se $\|f_i\| - \|r_i\| < TOL$ oppure $\gamma_i > GMAX$ allora interrompere il ciclo con il comando *EXIT*.
7. Se $j > 1$ allora porre $\gamma_{i+1}^{(1)} = \gamma_i$ altrimenti $\gamma_{i+1}^{(1)} = DECR * \gamma_i$.
8. Porre $i := i + 1$ e tornare al punto 2.

Osservazione 7. σ è solitamente scelta piccola, ad esempio dell'ordine di 10^{-4}, *TOL* e *GMAX* testano, rispettivamente, la convergenza e la limitatezza di $\{\gamma_i\}$; *EXPND* e *DECR* sono delle costanti che vengono utilizzate per modificare γ. Precisamente, *EXPND* assicura che al passo 4 il valore di γ aumenti, affinché sia soddisfatta la relazione (3.3) e *DECR* garantisce che al passo 7 γ decresca, per cercare di incrementare la velocità di convergenza nel caso in cui il test nel passo 4 sia soddisfatto con $j = 1$. In praticolare, è importante osservare che $EXPND > 1$ e $DECR < 1$.

Osservazione 8. Per calcolare il gradiente, nel caso di un problema di cui al capitolo **2** (paragrafo **2.4**), si usa l'algoritmo di **Backpropagation**.

3.4 Il caso non limitato

In questo paragrafo viene considerato, più nel dettaglio, il caso non limitato. In particolare, si può dedurre delle informazioni riguardanti $\nabla^2 F$.

Teorema 3. *Si supponga che la successione $\{\gamma_i\}$, trovata applicando l'algoritmo di Levenberg-Marquardt, sia illimitata. Allora la norma della matrice Hessiana $\nabla^2 F$ è anch'essa illimitata.*

Dimostrazione. Siccome $\{\gamma_i\}$ è illimitata, allora esiste un'altra successione illimitata $\{\gamma_i^*\}$ tale che:

$$\sigma > \psi\Big(x_i, \frac{\gamma_i^*}{EXPND}\Big),$$

poichè devono esserci infinite volte in cui il test del passo $\boxed{4}$ dell'algoritmo fallisce. Quindi, esiste una successione illimitata $\{\hat{\gamma}_i\}$ che soddisfa $\sigma > \psi(x_i, \hat{\gamma}_i)$. Per semplicità di notazione $h_i(\hat{\gamma}_i)$ è denotato come h_i. Il valor medio è indicato con una barra. Utilizzando, quindi, la relazione (3.12) e lo sviluppo di Taylor senza resto:

$$F(x_{i+1}) = F(x_i) + \nabla F_i h_i + \frac{1}{2} h_i^T \nabla^2 F(z_i) h_i,$$

con $z_i = x_i + \theta(x_{i+1} - x_i)$ e $\theta \in (0,1)$, e dunque $\overline{\nabla^2 F_i} = \nabla^2 F(z_i)$, si ottiene:

$$\sigma > \frac{|\nabla F_i h_i| - \frac{1}{2}|h_i^T \overline{\nabla^2 F_i} h_i|}{|\nabla F_i h_i|},$$

che implica:

$$|h_i^T \overline{\nabla^2 F_i} h_i| > 2(1-\sigma)|\nabla F_i h_i|. \tag{3.16}$$

Quindi

$$\|\nabla^2 F_i\| > 2(1-\sigma)\frac{|\nabla F_i h_i|}{\|h_i\|^2}. \tag{3.17}$$

Dalle uguaglianze (3.9) e (3.11) si può ricavare che:

$$\begin{aligned}\nabla F_i h_i &= -\frac{1}{2}\nabla F_i (A_i^T A_i + \hat{\gamma}_i^2 B_i^T B_i)^{-1} \nabla F_i^T \\ &= -\frac{1}{2} h_i^T (A_i^T A_i + \hat{\gamma}_i^2 B_i^T B_i) h_i,\end{aligned}$$

dove

$$|\nabla F_i h_i| \geq \frac{1}{2}\|h_i\|^2 \frac{\hat{\gamma}_i^2}{\alpha^2} \tag{3.18}$$

e $\frac{1}{\alpha^2}$ è, per ipotesi, il limite inferiore del più piccolo autovalore di $B_i^T B_i$.

Da tutte queste considerazioni segue che:

$$\|\overline{\nabla^2 F_i}\| > (1-\sigma)\frac{\hat{\gamma}^2}{\alpha^2}. \tag{3.19}$$

□

Corollario 1. *Sia $\|\nabla^2 F\|$ finito in un intorno di x, oppure, più in generale, sull'insieme di livello $\mathcal{L}_0 = \{x \in \mathbb{R}^p$ tali che $f(x) \leq f(x_0)\}$, allora ogni punto limite finito di $\{x_i\}$ è un punto stazionario di $F(x)$.*

Dimostrazione. Poiché $\|\nabla^2 F\|$ è finito in un intorno di x_0 su $\mathcal{L}_0$, allora la successione delle $\{\gamma_i\}$ non può essere illimitata perché ciò sarebbe in disaccordo con il teorema **3**.

Fatta questa osservazione, si può applicare i risultati visti in precedenza e, ricordandosi che la relazione $0 < \sigma < \psi(x_i, \gamma_i)$ risulta valida per costruzione, si può applicare il teorema **1** e concludere che i punti limite della successione sono valori stazionari di $F(x)$. □

Osservazione 9. Nelle ipotesi del corollario precedente, se inoltre $\mathcal{L}_0$ è compatto, si ricava con tecniche standard che la successione delle $\{x_i\}$ converge all'insieme dei punti stazionari di $F(x)$.

Capitolo 4

Test numerici

4.1 I risultati ottenuti

Per completare questo lavoro sono stati eseguiti alcuni test numerici (usando il linguaggio **MATLAB**).

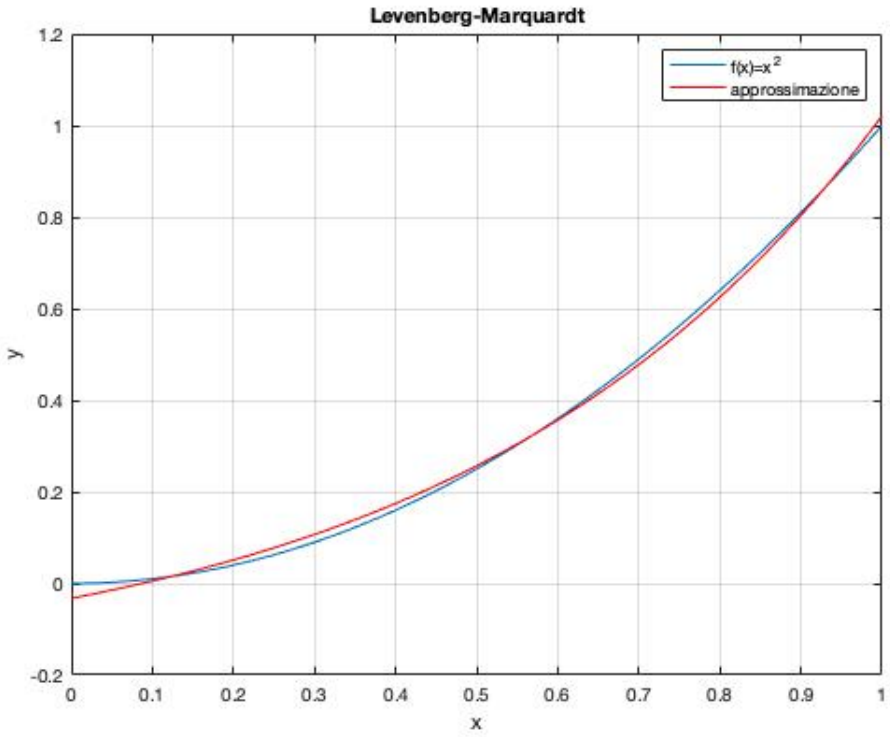

Figura 4.1: Il grafico blu rappresenta la funzione $g(x)$ da approssimare e il grafico rosso l'approssimazione ottenuta.

Si considera una generica funzione continua $g : [0, 1] \to \mathbb{R}$. Siano x_1, x_2, ..., x_d, con $x_1 = 0$, $x_d = 1$ e $d = 20$, i d nodi di una partizione uniforme dell'intervallo $[0, 1]$. Il problema modello di regressione non-lineare da risolvere consiste nella ricerca del

vettore $w = [w_1, w_2, w_3] \in \mathbb{R}^3$ tale che la funzione di tipo esponenziale

$$w_1 + w_2 e^{(w_3 x)}, \quad x \in [0, 1],$$

è quella che meglio approssima la funzione g, nel senso che minimizza il funzionale

$$F(w) = \sum_{i=1}^{d} |g(x_i) - w_1 - w_2 exp(w_3 x_i)|^2.$$

La funzione scelta è $g(x) = x^2$.

Il metodo che si è scelto di implementare, per la risoluzione del problema, è quello di Levenberg-Marquardt, come descritto in precedenza (capitolo **3**).

Considerando il listato dell'algoritmo (paragrafo **3.3**), come valori iniziali delle varie costanti si è scelto di usare: $DECR = 0.5$, $GMAX = 500$, $\sigma = 10^{-4}$, $EXPND = 1.5$ e $TOL = 10^{-8}$. La γ di partenza è stata inizializzata con un valore pari a 1. Le scelte sono concordi con ciò che è stato affermato in [9]. Come matrice B_i viene usata la matrice identità, per quanto scritto nell'osservazione **2**.

In figura **4.1** si disegnano i grafici di $g(x)$ e della soluzione numerica; si può apprezzare l'ottimo accordo tra le due funzioni.

w iniziale	n. iterate	γ finale	w finale
$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	8	0.0078	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$
$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$	9	0.0117	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$
$\begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$	23	0.0119	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$
$\begin{bmatrix} 10 \\ 20 \\ 15 \end{bmatrix}$	29	0.0113	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$
$\begin{bmatrix} 50 \\ 70 \\ 30 \end{bmatrix}$	49	0.0108	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$
$\begin{bmatrix} 100 \\ -50 \\ 60 \end{bmatrix}$	110	0.0122	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$
$\begin{bmatrix} 100 \\ 100 \\ 100 \end{bmatrix}$	425	0.0111	$\begin{bmatrix} -0.2090 \\ 0.1765 \\ 1.9415 \end{bmatrix}$

Tabella 4.1: Valori del vettore iniziale forniti e rispettivi risultati ottenuti.

La tabella **4.1** mostra i valori scelti dei vettori iniziali w. Si osserva, ovviamente, che il numero delle iterate aumenta allontanandosi dal punto di minimo. Il valore del

w finale è sempre lo stesso, questo implica che l'algoritmo funziona bene, poiché trova sempre lo stesso valore, corrispondente al punto di minimo. Inoltre, con questi test si è confermato quanto affermato nell'osservazione **5**, poiché il test con il numero di iterate inferiore è anche quello con la γ finale più piccola.

4.2 L'algoritmo

In seguito, sono riportati i codici implementati. Il primo che è stato inserito è quello principale e contiene l'algoritmo di LM, all'inizio del codice vengono inizializzati i valori che rimangono costanti durante le varie iterate. Le matrici `GAMMA` e `GAMMA_v` sono state utili per tenere in considerazione la dipendenza di γ dai vari passi.

```
x=[0:1/19:1];
GMAX=500;
sigma=10^(-4);
EXPND=1.5;
DECR=0.5;
TOL=10^(-8);
B=eye(3);

GAMMA=ones(GMAX,GMAX);
GAMMA_v=ones(1,GMAX);

%%%% START %%%%%%%%%%%%%%%%%%
wi=[0;0;0]; %punto iniziale
for i=1:GMAX
    j=1;
    [fi]=interno(wi,x);
    [Ai,Ati]=gradiente(wi,x);
    [hi]=discesa(Ati,fi,Ai,GAMMA(i,j),B);
    wi_1=wi+hi;
    [ri]=erre(fi,Ai,hi,GAMMA(i,j),B);
    [Fi]=valore(wi,x);
    [Fi_1]=valore(wi_1,x);
    [phi_i]=PHI(Fi,Fi_1,ri);
    while sigma > phi_i
        j=j+1;
        GAMMA(i,j)=EXPND*GAMMA(i,j-1);
        [hi]=discesa(Ati,fi,Ai,GAMMA(i,j),B);
        wi_1=wi+hi;
        [ri]=erre(fi,Ai,hi,GAMMA(i,j),B);
        [Fi]=valore(wi,x);
        [Fi_1]=valore(wi_1,x);
        [phi_i]=PHI(Fi,Fi_1,ri);
    end
    GAMMA_v(i)=GAMMA(i,j);
    if norm(fi)-ri<TOL || GAMMA_v(i)>GMAX
        break
```

```
    end
    if j>1
        GAMMA(i+1,1)=GAMMA_v(i);
    else
        GAMMA(i+1,1)=DECR*GAMMA_v(i);
    end
    wi=wi_1;
end
```

I codici seguenti contengono delle function implementate per alleggerire il codice principale.

```
function [f]=interno(w,x)
g=inline('x^2','x');
f=zeros(20,1);
for k=1:length(x)
    f(k)=g(x(k)) - w(1) - w(2)*exp(w(3)*x(k));
end

function [A,At]=gradiente(w,x)
A=zeros(length(x),3);
for i=1:length(x)
    A(i,1)=-1;
    A(i,2)=-exp(w(3)*x(i));
    A(i,3)=-w(2)*x(i)*exp(w(3)*x(i));
end
At=A';

function [h]=discesa(At,f,A,gamma,B)
h=(At*A+gamma^(2)*B)\(-At*f);

function [r]=erre(f,A,h,gamma,B)
vett1=[f;0;0;0];
mat2=[A;gamma*B];
vett2=mat2*h;
vettore_r=vett1+vett2;
r=norm(vettore_r);

function [F]=valore(w,x)
g=inline('x^2','x');
F=0;
for k=1:length(x)
    ff=g(x(k)) - w(1) - w(2)*exp(w(3)*x(k));
    F = F + ff^2;
end

function [phi]=PHI(F1,F2,r)
phi=(F1-F2)/(2*(F1-r^2));
```

Ringraziamenti

Prima di tutto, ringrazio il mio relatore, il Prof. Beirao Da Veiga Lourenco, per avermi affiancato durante la stesura di questa tesi con estrema disponibilità e per aver risposto sempre con celerità. Lo ringrazio, inoltre, per aver aumentato la mia passione per l'Analisi Numerica.

Ringrazio i miei genitori, ai quali dedico questa tesi. Senza di loro, i loro sacrifici e la loro complicità, non sarei chi sono e non sarei arrivata dove sono. Grazie per essere stati sempre al mio fianco, per non avermi fatto mancare niente e per aver sopportato il mio orribile carattere, che in questo periodo era peggiorato. Grazie mamma per non avermi mai fatto arrendere e autocommiserare, per essere un esempio per me. Grazie papà per tutti i consigli e per aver sempre creduto nelle mie capacità, per essere il punto di riferimento di casa e un esempio di perseveranza.

Un grazie va a mia nonna Anna, che è sempre stata un mio punto di riferimento.

Inoltre, non posso non ringraziare i miei zii, Giancarlo e Miriam, che nonostante la lontananza, mi sono sempre stati d'aiuto.

Un ringraziamento speciale va alla Prof.ssa Benna che mi ha sempre dato il consiglio giusto e le basi per affrontare questo percorso.

Bibliografia e Sitografia

[1] Jarosław Bilski et al. «Local Levenberg-Marquardt algorithm for learning feedforwad neural networks». In: *Journal of Artificial Intelligence and Soft Computing Research* 10 (2020).

[2] Nicoletta Boldrini. *Deep Learning: cos'è, come funziona e quali sono i casi d'uso*. 2019. URL: https://www.ai4business.it/intelligenza-artificiale/deep-learning/deep-learning-cose/.

[3] Nicoletta Boldrini. *Reti neurali: cosa sono, a cosa servono, la loro storia*. 2019. URL: https://www.ai4business.it/intelligenza-artificiale/deep-learning/reti-neurali/.

[4] *Deep Learning: 3 cose da sapere*. 2016. URL: https://it.mathworks.com/discovery/deep-learning.html.

[5] L Grippo e M Sciandrone. «Metodi di ottimizzazione per le reti neurali». In: *Università di Roma La Sapienza e Consiglio nazionale delle ricerche, Roma* (2006).

[6] Catherine F Higham e Desmond J Higham. «Deep learning: An introduction for applied mathematicians». In: *Siam review* 61.4 (2019), pp. 860–891.

[7] Chen Lv et al. «Levenberg–Marquardt backpropagation training of multilayer neural networks for state estimation of a safety-critical cyber-physical system». In: *IEEE Transactions on Industrial Informatics* 14.8 (2017), pp. 3436–3446.

[8] Jorge Nocedal e Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[9] MR Osborne. «Nonlinear least squares—the Levenberg algorithm revisited». In: *The ANZIAM Journal* 19.3 (1976), pp. 343–357.

[10] MR Osborne. «Some aspects of nonlinear least squares calculations». In: *Numerical methods for nonlinear optimization* (1972).

[11] *Apprendimento profondo*. 2012. URL: https://it.wikipedia.org/wiki/Apprendimento_profondo.

[12] *La Funzione sigmoidea*. 2006. URL: https://it.wikipedia.org/wiki/Funzione_sigmoidea.

[13] *Algoritmo di Levenberg-Marquardt*. 2020. URL: https://it.wikipedia.org/wiki/Algoritmo_di_Levenberg-Marquardt.