Allenamento di Reti Neurali Artificiali con Discesa Stocastica del Gradiente e Metodi Quasi-Newton

Computational Mathematics for Learning and Data Analysis + Machine Learning

progetto 2

Carlo Alessi

25 febbraio 2019

Sommario

Un semplice framework per definire, allenare e valutare reti neurali artificiali è stato sviluppato. In particolare sono stati implementati gli algoritmi di apprendimento di discesa stocastica del gradiente (SGD), BFGS e L-BFGS, nonché una line-search che soddisfa le condizioni di Armijo-Wolfe. Le prestazioni degli algoritmi sono state valutate su quattro dataset, utilizzando le metriche di accuratezza, errore quadratico medio e distanza Euclidea. Infine gli approcci sono stati messi a conforto mettendo in luce efficacia, efficienza e capacità di generalizzazione.

Indice

1 Introduzione

Il problema di minimizzare una funzione obiettivo può essere risolto con diversi algoritmi. Gli algoritmi basati sulla discesa del gradiente, come la discesa stocastica del gradiente e le sue varianti, sono onnipresenti nel caso di ottimizzazione a grande scala (i.e. datasets grandi), in quanto è possibile approssimare il gradiente vero calcolandolo su mini-batch, invece che sull'intero insieme di dati. Invece gli algoritmi basati sul metodo di Newton sono efficaci quando il dataset ha dimensioni limitate ed è quindi possibile calcolare il vero gradiente, talvolta approssimando le derivate seconde, con il fine di ottenere direzioni di discesa più accurate. Se il primo approccio è migliore del secondo o viceversa è ancora oggetto di ricerca. Nessun algoritmo è dominante sugli altri, e ogni applicazione deve avere un algoritmo su misura. Per questa ragione è interessante discutere entrambi gli approcci in questa relazione.

Gli obiettivi di questo progetto sono i seguenti:

1. Implementare un semplice framework che permette di definire ed allenare reti neurali artificiali.
2. Implementare tre algoritmi di ottimizzazione (discesa del gradiente, BFGS e L-BFGS) confrontandone le prestazioni ottenute su diversi datasets.
3. Implementare vari algoritmi di line-search che soddisfano diverse condizioni (i.e. condizioni di Armijo, Wolfe, confrontando i risultati.
4. Analizzare l'algoritmo L-BFGS, in modo più o meno approfondito, studiando come il variare dei singoli iperparametri influenza le prestazioni.

Il resto della relazione è così sviluppato. In sezione 2 viene descritto il problema di ottimizzazione risolto, descrivendo i datasets utilizzati e introducendo la notazione matematica utilizzata per definire la rete neurale. Vengono inoltre definite le funzioni di attivazione impiegate nello studio, nonché la funzione obiettivo. In sezione 3 vengo recensiti gli aspetti fondamentali degli algoritmi di apprendimento e di line-search implementati. In sezione 4 vengono riassunti i risultati teorici di convergenza degli algoritmi e viene discusso se questi si applicano o meno al problema in questione. La sezione 5 descrive la metodologia adottata in questo progetto. In particolare le librerie utilizzate, il software design, le scelte implementative e la tecnica di validazione. sezione 6 e sezione 7 riportano i risultati degli esperimenti rilevanti dal punto di vista della Matematica Computazionale e del Machine Learning rispettivamente. sezione 8 riassume i risultati principali e conclude.

2 Descrizione del problema di ottimizzazione

Il problema di ottimizzazione non vincolata da risolvere è l'allenamento di una rete neurale artificiale:

$$min_{w \in \mathbb{R}^n} E(w) \tag{1}$$

Di seguito vengono brevemente descritti i dataset su cui viene fatto il benchmark degli algoritmi di apprendimento. Inoltre viene definita la notazione e definite le principali componenti della rete neurale. Successivamente vengono elencate le funzioni di attivazione coinvolte. Infine viene definita la funzione obiettivo E e le sue proprietà.

2.1 Dataset

Sono stati usati 4 datasets: MONK-1, MONK-2 e MONK-3 [5] per il benchmark della rete neurale, ML-CUP per la sfida. Proprietà, task e funzioni obiettivo associate sono riassunte in Tabella 1.

2.2 Notazione utilizzata e definizioni

In questa sezione viene descritta la notazione matematica utilizzata e definite le componenti principali della rete neurale relativamente agli input, output, parametri e funzioni di attivazione.

Una rete neurale feed-forward fully-connected è una funzione $F : \mathbb{R}^a \to \mathbb{R}^b$. Graficamente può essere vista come un DAG (Directed Acyclic Graph), dove ogni neurone del layer i-esimo è

Tabella 1: Proprietà dei dataset, task e funzione obiettivo. Si ottimizza l'Errore Quadratico Medio (MSE) per i MONK, e la Distanza Euclidea Media (MEE) per ML-CUP.

Dataset	Dim.	n. Input	n. Output	Tipo variabili	Task	Funzione obiettivo
MONK-1	124	17	1	binarie	classificazione	MSE
MONK-2	169	17	1	binarie	classificazione	MSE
MONK-4	122	17	1	binarie	classificazione	MSE
ML-CUP	1016	10	2	reali	regressione	MEE/MSE

collegato solamente ad ogni neurone del layer $i+1$-esimo, senza formazione di cicli. Una rete neurale feed-forward di l layers è cosi definita:

$$F(x; w, l) = \begin{cases} g^1(W_1 \cdot x) & , \ l = 1 \\ g^l(W_l \cdot F(x; w, l-1)) & , \ otherwise \end{cases} \tag{2}$$

dove:

- $x \in \mathbb{R}^a$ generico input pattern. Alternativamente si indica con x_{ji} l'i-esimo input del neurone j. Quest'ultima notazione vale per qualsiasi neurone della rete.
- $w \in \mathbb{R}^n$ vettore dei parametri/pesi della rete neurale. In particolare w_{ji} è il peso associato a x_{ji}. Talvolta i parametri sono definiti per strati della rete, ottenendo un insieme di matrici $\{W_i, i = 1, \dots l\}$, dove l indica il numero di strati della rete. La matrice W_i ha un numero di righe pari al numero di neuroni nel layer i e un numero di colonne pari al numero di neuroni del layer $i-1$.
- $net_j(w) = \sum_i w_{ji} \cdot x_{ji}$ la somma pesata degli input del neurone j.
- $g_j : \mathbb{R} \to \mathbb{R}$ la funzione di attivazione usata nel neurone j. Alternativamente con $g^l : \mathbb{R}^{n(l)} \to \mathbb{R}^{n(l)}$ si indica la funzione di attivazione del layer l, dove $n(l)$ è il numero di neuroni del layer l.
- $o_j(w) = g_j(net_j(w))$ è l'output del neurone j. $\hat{\boldsymbol{y}}$ è il vettore di output della rete.
- y_j è il target output per il j-esimo neurone di output della rete. $\boldsymbol{y}$ è il vettore corrispondente.
- $downstream(j)$ è l'insieme dei neuroni i cui inputs includono l'output del neurone j (i.e. tutti i neuroni del layer successivo, escluso il bias).

2.3 Funzioni di attivazione

Tabella 2 definisce le funzioni di attivazione implementate. A seconda del task (regressione o classificazione) e del layer in questione sono impiegate funzioni di attivazione diverse. La funzione identità viene usata nel primo layer (InputLayer), il quale serve solo da segnaposto per le variabili di input. Viene inoltre usata nell'ultimo layer (OutputLayer) se il problema è di regressione. Nulla vieta l'utilizzo della funzione identità negli hidden layer, ma cosi facendo la rete neurale non apprenderebbe relazioni non-lineari tra input e output. Le funzioni sigmoid e *tanh* vengono utilizzate negli hidden layers, e nell'OutputLayer se il task è di classificazione binaria. La funzione ReLU viene usata negli hidden layers per velocizzare l'apprendimento di reti neurali profonde e diminuire il rischio del problema del *vanishing gradient* (i.e. il gradiente diventa prossimo a zero) presente nelle funzioni sigmoid e *tanh*.

Proprietà funzioni di attivazione Le attivazioni Lineare, Sigmoid e Tanh sono di classe C^∞ in quanto derivabili con continuità infinite volte. Invece ReLU è di classe C^0 perché non è derivabile nell'origine, dove presenta un punto angoloso. Le attivazioni Lineare, Sigmoid e Tanh sono Lipschitz-continue in $\mathbb{R}$ in quanto le loro derivate sono limitate. Anche le loro derivate n-esime sono Lipschitziane. Invece ReLU e le sue derivate non sono Lipschitziane su tutto $\mathbb{R}$ ma solo in $\mathbb{R} \setminus \{0\}$.

Tabella 2: Funzioni di attivazione utilizzate.

Nome	Dominio/Codominio	Definizione	Derivata $\frac{dg}{dx} = g'(x)$
Identità	$g : \mathbb{R} \to \mathbb{R}$	$g(x) = x$	1
Sigmoid	$g : \mathbb{R} \to [0, 1]$	$g(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$\sigma'(x) = \sigma(x)(1 - \sigma(x))$
Tangente iperbolica	$g : \mathbb{R} \to [-1, +1]$	$g(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$tanh'(x) = 1 - tanh^2(x)$
ReLU	$g : \mathbb{R} \to \mathbb{R}_+ \cup \{0\}$	$g(x) = max(0, x)$	1 if $x > 0$, 0 altrimenti[a]

[a] Per semplificare l'implementazione non si è considerato che in realtà la derivata non è definita in $x = 0$.

2.4 Funzione obiettivo

Come riassunto in Tabella 1, le funzioni obiettivo da minimizzare sono l'Errore Quadratico Medio (MSE) e la distanza Euclidea Media (MEE)[1], a cui si aggiunge la regolarizzazione di Tikhonov (norma L_2). La norma L_2 ha due effetti: **(i)** Dal punto di vista del Machine Learning serve per penalizzare le soluzioni con $||w||_2$ elevata, con il fine di evitare l'overfitting ed ottenere migliori capacità di generalizzazione della rete [3]. **(ii)** Dal punto di vista dell'ottimizzazione ha un effetto di stabilizzazione, in quanto migliora il condizionamento dei problemi [2]. Quest'ultima proprietà sarà molto importante nei risultati riportati in sezione 6, in quanto il suo effetto è particolarmente visibile nel dataset ML-CUP. Si ha la seguente funzione obiettivo:

$$E(w) = \frac{1}{\mid D \mid} \sum_{d \in D} E_d(w) + \lambda \sum_{i,j} w_{ij}^2 \tag{3}$$

Dove il primo addendo è detto *errore sui dati*, il secondo è detto *errore di regolarizzazione*, mentre λ è il coefficiente di regolarizzazione/penalizzazione. Invece $E_d(w)$ è l'errore su un solo esempio d di allenamento, definito diversamente a seconda del task:

$$E_d(w) = MSE_d(w) = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2^2 \tag{4}$$

$$E_d(w) = MEE_d(w) = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2 \tag{5}$$

Le derivate di (4) e (5) sono rispettivamente:

$$\nabla_{\hat{\boldsymbol{y}}} E = 2(\hat{\boldsymbol{y}} - \boldsymbol{y}) \tag{6}$$

$$\nabla_{\hat{\boldsymbol{y}}} E = \frac{\hat{\boldsymbol{y}} - \boldsymbol{y}}{||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2} \tag{7}$$

2.4.1 Studio della funzione obiettivo

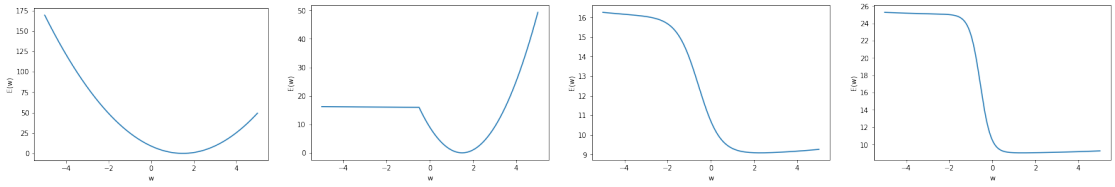
Le proprietà della funzione obiettivo (3) (convessità, regolarità, continuità e Lipschitzianità) vengono studiate per comprendere le proprietà di convergenza dei vari algoritmi, con l'ausilio delle definizioni in Appendice A e dell'esempio mostrato in Figura 1.

Per le proprietà di (4) e (5), E è limitata inferiormente dal valore 0, ed illimitata superiormente, ovvero $E : \mathbb{R} \to [0, +\infty)$.

Derivate Errore Si osservino le derivate di MSE (6) e MEE (7). Nel primo caso, si ha che $\nabla_{\hat{\boldsymbol{y}}} E \to \boldsymbol{0}$ per $\hat{\boldsymbol{y}} \to \boldsymbol{y}$. Nel secondo caso si potrebbero avere errori numerici quando $\hat{\boldsymbol{y}} \simeq \boldsymbol{y}$, in quanto il numeratore $\hat{\boldsymbol{y}} - \boldsymbol{y} \to \boldsymbol{0}$ e il denominatore si annulla.

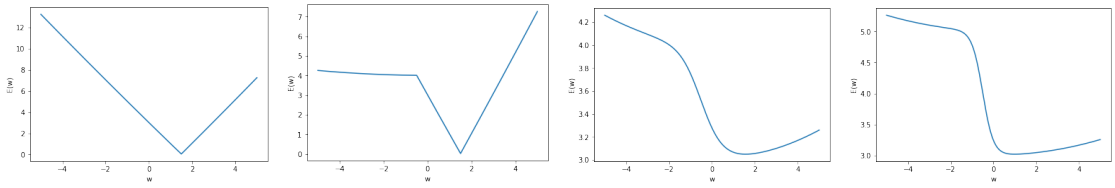
[1] In realtà si ottimizza solo la MSE perché MEE è non differenziabile.
[2] https://en.wikipedia.org/wiki/Tikhonov_regularization

MSE



(a) lineare (b) relu (c) sigmoid (d) tanh

MEE



(e) lineare (f) relu (g) sigmoid (h) tanh

Figura 1: Studio qualitativo della funzione obiettivo, $E(w) : \mathbb{R} \to \mathbb{R}$, variando le funzioni di attivazione. Prima riga: $E(w) = (t - g(xw))^2 + \lambda w^2$. Seconda riga: $E(w) = ||t - g(xw)||_2 + \lambda w^2$. Parametri: bias $b = 1$, regolarizzazione $\lambda = 0.01$, pesi w casuali. Curve ottenute su un dataset di un elemento $x = 2$ con target $y = 4$.

Continuità e differenziabilità Per le proprietà di composizione di funzioni continue e derivabili, se la funzione obiettivo è $E := MSE$, allora (3) è di classe C^∞ usando le attivazioni Lineare, Sigmoid e Tanh. Nel caso di $E := MEE$, (3) è di classe C^∞ solo usando le attivazioni Sigmoid e Tanh, mentre è di classe C^0 usando l'attivazione Lineare (vedi 1e). Usando ReLU $E \in C^0$ in ogni caso (vedi 1b, 1f), in quanto E presenta punti di non derivabilità, esattamente come la funzione di attivazione stessa.

Lipschitzianità di E, ∇E e $\nabla^2 E$ Per le proprietà di composizione di funzioni Lipschitziane, si ha che E, (3), non è Lipschitziana perché il secondo addendo, essendo un quadrato, non è Lipschitz-continuo su tutto $\mathbb{R}$, ma solo su un sottoinsieme compatto $I \subseteq R$. Invece ∇E e $\nabla^2 E$ sono Lipschitziane in $\mathbb{R}$ solo nel caso in cui vengano usate attivazioni Lipschitziane.

Convessità Facendo uno studio qualitativo della convessità di E, osservando Figura 1, si nota che (3) è convessa solo nel caso di $E := MSE$ con attivazione Lineare (vedi 1a). Questo si dimostra servendoci del fatto che la composizione di una funzione convessa (e.g. elevazione al quadrato) e una trasformazione affine (e.g. $g(wx) = wx$) preserva la convessità [1, p. 79]. La funzione (3) è non convessa in tutti gli altri casi mostrati.

3 Algoritmi di apprendimento

In questa sezione vengono descritti nel dettaglio gli algoritmi di apprendimento che sono oggetto di studio in questa relazione, ovvero discesa del gradiente e quasi-Newton.

3.1 Discesa del gradiente

La discesa del gradiente fonda le sue basi sull'algoritmo di back propagation [7], l'algoritmo standard per calcolare il gradiente di una rete neurale. Di seguito si descrive come viene usata la *chain-rule* per calcolare $\nabla_{\boldsymbol{w}} E$, e la regola di aggiornamento di un singolo peso.

Calcolo del gradiente Usando la chain rule, la derivata dell'errore per uno specifico esempio d rispetto ad un determinato peso w_{ji} viene scomposta in tre termini come segue:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial o_j(w)} \times \frac{\partial o_j(w)}{\partial net_j(w)} \times \frac{\partial net_j(w)}{\partial w_{ji}} \tag{8}$$

Per semplicità si pone $\delta_j = \frac{\partial E_d}{\partial o_j(w)} \times \frac{\partial o_j(w)}{\partial net_j(w)}$. Le tre componenti dell'errore in (8) sono cosi calcolate:

1. $\frac{\partial E_d}{\partial o_j(w)}$ è differente a seconda se si tratta di hidden o output layer:

$$\frac{\partial E_d}{\partial o_j(w)} = \begin{cases} -(t_j - o_j(w)) & , \; OutputLayer \\ \sum_{k \in downstream(j)} w_{kj}\delta_k & , \; HiddenLayer \end{cases} \tag{9}$$

2. $\frac{\partial o_j(w)}{\partial net_j(w)} = \frac{\partial f_j(net_j(w))}{\partial net_j(w)}$, ovvero la derivata della funzione di attivazione del neurone rispetto al suo input pesato. La formula specifica è determinata in base alla scelta delle funzioni di attivazione descritte in sottosezione 2.3.

3. $\frac{\partial net_j(w)}{\partial w_{ji}} = x_{ji}$, ovvero l'$i$-esima variabile di input del neurone j associata al peso w_{ji} da aggiornare.

Aggiungiamo dunque la derivata della componente dell'errore relativo alla regolarizzazione, ovvero $\frac{\partial}{\partial w_{ji}} \lambda ||w||^2 = 2\lambda w_{ji}$. Mettendo assieme le componenti precedenti si ottiene

$$\frac{\partial E_d}{\partial w_{ji}} = \delta_j x_{ji} + 2\lambda w_{ji}$$

Avendo definito la componente del gradiente relativa ad un singolo peso w_{ji}, mettendo assieme le varie derivate $\frac{\partial E_d(w)}{\partial w_{ij}}$, si ottiene il gradiente:

$$\nabla_{\boldsymbol{w}} E_d(w) = [\ldots \frac{\partial E_d}{\partial w_{ji}} \ldots] \quad \forall w_{ji} \in w$$

Aggiornamento dei pesi L'aggiornamento dei pesi avviene all'iterazione k con la seguente formula:

$$w_{ji} = w_{ji} + \Delta w_{ji}^k$$

dove Δw è definito come:

$$\Delta w_{ji}^k = -\alpha \cdot \frac{\partial E_d(w)}{\partial w_{ji}} + \mu \cdot \Delta w_{ji}^{k-1}$$

Lo scalare $\alpha \in [0, 1]$ è lo step size o learning rate, ed $\mu \in [0, 1]$ è la frazione dell'aggiornamento dei pesi effettuato al passo $k - 1$, ovvero il *momentum*. Entrambi sono fissati arbitrariamente.

3.2 Algoritmo BFGS

L'algoritmo BFGS è un algoritmo Quasi-newton [8, cap. 6]. Sia $f : \mathbb{R} \to \mathbb{R}$ la funzione obiettivo, il suo comportamento in un intorno dell'iterato x_k viene approssimato dal seguente modello quadratico convesso:

$$m_k(p) = f(x_k) + \nabla f(x_k)^\intercal p + \frac{1}{2} p^\intercal B_k p \tag{10}$$

dove $B_k \in \mathbb{R}^{n \times b}$ è una matrice simmetrica e definita positiva, approssimazione dell'Hessiano $\nabla^2 f(x_k)$, aggiornata ad ogni iterazione. Si noti che $m_k(0) = f(x_k)$ e $\nabla m_k(0) = \nabla f(x_k)$. Il minimizzatore p_k del modello quadratico (10) può essere scritto esplicitamente come:

$$p_k = -B_k^{-1} \nabla f(x_k) \tag{11}$$

e il nuovo iterato diventa

$$x_{k+1} = x_k + \alpha_k p_k \tag{12}$$

dove α_k è scelto per soddisfare le condizioni di Wolfe. Si consideri ora il modello quadratico m_{k+1} all'iterato x_k:

$$m_{k+1}(p) = f(x_{k+1}) + \nabla f(x_{k+1})^\intercal p + \frac{1}{2} p^\intercal B_{k+1} p \tag{13}$$

Come per (10), anche (10) è tale che $m_{k+1}(0) = f(x_{k+1})$ e $\nabla m_{k+1}(0) = \nabla f(x_{k+1})$. Viene inoltre imposto il vincolo $\nabla m_{k+1}(x_k) = \nabla f(x_k)$, cioè

$$m_{k+1}(-\alpha_k p_k) = \nabla f(x_{k+1} - \alpha_k B_{k+1} p_k = \nabla f(x_k) \tag{14}$$

il quale redistribuendo i termini diventa:

$$B_{k+1} \alpha_k p_k = \nabla f(x_{k+1}) - \nabla f(x_k) \tag{15}$$

Ponendo:

- $s_k = x_{k+1} - x_k = \alpha_k p_k$
- $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$

(15) diventa:

$$B_{k+1} s_k = y_k \tag{16}$$

la quale viene riferita come *equazione della secante*, la quale richiede che la matrice definita positiva B_{k+1} mappi s_k in y_k. Questo è possibile solo se i due vettori soddisfano la *condizione di curvatura*:

$$s_k^\intercal y_k > 0 \tag{17}$$

Per motivi di efficienza si opera con l'inversa dell'approssimazione dell'Hessiana, ponendo $H = B^{-1}$, la quale deve essere simmetrica e definita positiva, e rispettare l'*equazione della secante*: $H_{k+1} y_k = s_k$. Per derivare l'approssimazine dell'inversa dell'Hessiana alla iterazione successiva, si risolve il seguente problema di ottimizzazione:

$$H_{k+1} = arg \min_H ||H - H_k|| \quad subject\ to\ H = H^\intercal\ ,\ H y_k = s_k \tag{18}$$

la quale soluzione unica H_{k+1}, ponendo $\rho_k = \frac{1}{y_k^\intercal s_k}$, definisce la formula di aggiornamento BFGS:

$$H_{k+1} = (I - \rho_k s_k y_k^\intercal) H_k (I - \rho_k y_k s_k^\intercal) + \rho_k s_k s_k^\intercal \tag{19}$$

La *condizione di curvatura*, (17), rappresenta un aspetto fondamentale per la converganza dell'algoritmo. È soddisfatta per ogni due punti x_k, x_{k+1} quando la funzione obiettivo è fortemente convessa. In caso di funzione non convessa la condizione va forzata esplicitamente ponendo vincoli sulla line-search. Inoltre (17) gioca un ruolo importante dal punto di vista numerico. Se $y_k^\intercal s_k \sim 0$, il termine $\rho_k \to \infty$, e dalla equazione di aggiornamento (19) si può notare che $||H_{k+1}||$ diverrebbe

enorme. Questo può causare errori di arrotondamento, a causa dell'aritmetica a precisione finita del calcolatore, i quali risultano in una matrice H che è una cattiva approssimazione della vera matrice Hessiana. Come risultato la line-search viene rallentata in quanto non riesce a trovare un buono step α lungo la direzione di discesa, ormai distorta dagli arrotondamenti.

Inoltre l'algoritmo si comporta al meglio quando la line-search impiegata soddisfa le condizioni di Wolfe (o strong Wolfe), la quale ha capacità autocorrettive nel caso in cui si H sia una cattiva approssimazione. In aggiunta la line-search dovrebbe sempre testare il passo di ottimizzazione $\alpha = 1$ al primo tentativo, il quale eventualmente verrà sempre accettato risultando in una convergenza superlineare, che è sufficientemente veloce in pratica.

Ogni iterazione di questo metodo costa $\mathcal{O}(n^2)$ operazioni aritmetiche, in quanto non sono presenti operazioni con complessità $\mathcal{O}(n^3)$ come la risoluzione di sistemi lineari o prodotti matrice-matrice. Il BFGS è un algoritmo robusto, in quanto si comporta bene quando applicato a problemi di varia complessità. Tuttavia esso diventa impraticabile per problemi con molte variabili perché richiede $\mathcal{O}(n^2)$ memoria per la memorizzazione dell'approssimazione dell'Hessiana.

3.3 Algoritmo L-BFGS

L'algoritmo L-BFGS [8] è una versione a memoria limitata del metodo BFGS, utile quando il numero di variabili n è grande, il quale renderebbe il calcolo dell'Hessiana un'operazione costosa. Invece di memorizzare l'intera approssimazione dell'Hessiana $H \in \mathbb{R}^{n\times n}$, L-BFGS salva solo $m \ll n$ coppie di vettori $\{s_i, y_i\}_{i=k-m}^{k-1}$ ottenuti nelle ultime m iterazioni. Le coppie di vettori $\{s_i, y_i\}$, costantemente aggiornate, rappresentano implicitamente l'informazione di curvatura, da cui è possibile costruire l'approssimazione dell'Hessiana. Valori $m \in \{3, 20\}$ sono sufficienti per ottenere buoni risultati.

Il prodotto $H_k \nabla f(x_k)$ è calcolato efficientemente, partendo da $\nabla f(x_k)$ e le coppie $\{s_i, y_i\}$, con una sub-routine iterativa chiamata *two-loop recursion*, consistente in una serie di somme e prodotti scalari[8, pag. 178]. Nella sub-routine si scieglie un'approssimazione dell'Hessiana iniziale, $H_k^0 = \gamma_k I$, dove lo scalare γ_k varia ad ogni iterazione ed è definito come:

$$\gamma_k = \begin{cases} 1 & k = 1 \\ \frac{s_{k-1}^\intercal y_{k-1}}{||y_{k-1}||^2} & k \geq 2 \end{cases} \tag{20}$$

La scelta di γ_k assicura che la direzione p_k sia ben scalata, garantendo che il passo $\alpha_k = 1$ venga accettato definitivamente. Come in BFGS, è importante che la line-search soddisfi le condizioni di Wolfe, cosicché l'aggiornamento BFGS (19) sia stabile. Infine si scarta la coppia (s_{k-m}, y_{k-m}) se $k > m$ e si aggiornano e salvano i vettori s_k e y_k ottenuti all'iterazione corrente. Durante le prime $m-1$ iterazioni, BFGS e L-BFGS sono equivalenti se entrambi scelgono la stessa la matrice iniziale H_0, e se L-BFGS sceglie $H_k^0 = H_0$ ad ogni iterazione[3].

L'algoritmo è meno robusto quando il numero m di vettori salvati è piccolo. Incrementando m si ha un comportamento più stabile, diminuendo il numero di valutazioni di funzione, ma con lo svantaggio di aumentare il costo di ogni iterazione e la memoria utilizzata. Quindi il valore ottimale di m dipende dal problema e dalle risorse computazionali disponibili, ma è solitamente un valore piccolo. L'algoritmo L-BFGS è inefficiente quando applicato a problemi mal-condizionati, in particolare quando la matrice Hessiana ha una larga distribuzione degli autovalori. Il costo computazionale di ogni iterazione a $\mathcal{O}(m \cdot n)$ contro i $\mathcal{O}(n^2)$ per iterazione di BFGS. Quindi dal momento che in pratica $m << n$, un iterazione di L-BFGS costa meno. La validità di questo risultato in termini di tempo di esecuzione dipende anche dal costo del calcolo della funzione, del gradiente e della line search. L'occupazione di memoria è $\mathcal{O}(m \cdot n)$ per la memorizzazione delle coppie di vettori $\{s_i, y_i\}$.

3.4 Line search

Si consideri l'aggiornamento (12), $x_{k+1} = x_k + \alpha_k p_k$, l'algoritmo di line search calcola ad ogni iterazione il passo di ottimizzazione α_k da fare lungo la direzione discendente (11), $p_k = -B_k^{-1} \nabla f(x_k)$, dove $B_k \in \mathbb{R}^{n\times n}$ è una matrice simmetrica, non-singolare, definita positiva, che approssima la matrice Hessiana. La direzione discendente p_k garantisce che il valore della funzione f può essere diminuito lungo questa direzione. Vale in fatti la seguente relazione:

[3] Questa proprietà è stata verificata con un unit-test.

$$p_k^\intercal \nabla f(x_k) = -\nabla f(x_k)^\intercal B_k^{-1} f(x_k) < 0 \tag{21}$$

Il valore α ottimo sarebbe il minimizzatore globale della funzione

$$\phi(\alpha) = f(x_k + \alpha p_k) \qquad , \alpha > 0 \tag{22}$$

In generale trovare un minimizzatore locale o globale di ϕ è un processo costoso, perché potrebbe richiedere un gran numero di valutazioni di f e ∇f. Pertanto ci si accontenta di un α che riduca il valore di f adeguatamente, con un costo minimo. Essendo una ricerca locale, anziché globale, la line search è detta *inesatta*. Per garantire il successo dei metodi Quasi-newton, la line search deve trovare un α_k che soddisfi una o più delle seguenti condizioni: condizione di Armijo, condizione di Wolfe e condizione forte di Wolfe [8, cap. 3].

Condizione di Armijo. La condizione di Armijo, chiamata anche *condizione di decremento sufficiente*, è definita dalla seguente disuguaglianza:

$$\phi(\alpha) \leq \phi(0) + c_1 \alpha \phi'(0) \tag{23}$$

dove la parte destra della disuguaglianza è una funzione lineare in α con coefficiente angolare $c_1\phi'(0) = c_1 \nabla f(x_k)^T p_k < 0$, il quale garantisce che il valore della funzione f diminuisca, e $c_1 \in (0, 1)$ è una costante. La condizione di Armijo verrebbe soddisfatta da ogni valore α sufficientemente piccolo, con lo svantaggio che si avrebbe una lenta convergenza dell'algoritmo. D'altra parte un valore di α troppo grande potrebbe andare oltre il punto di minimo cercato. Per questi motivi è opportuno che α soddisfi anche la seguente condizione.

Condizione di Wolfe. La condizione di Wolfe, conosciuta anche come *condizione di curvatura*, è definita dalla seguente disuguaglianza:

$$\phi'(\alpha) \geq c_2 \phi'(0) \tag{24}$$

dove $c_2 \in (c_1, 1)$ è una costante, $\phi'(\alpha_k) = \nabla f(x_k + \alpha_k p_k)^\intercal p_k$, e $\phi'(0) = \nabla f(x_k)^\intercal p_k$. Per evitare che $\phi'(\alpha) \gg 0$ sia troppo positivo, andrebbero esclusi i punti che vanno troppo oltre un punto stazionario di ϕ. Per soddisfare questo criterio si può modificare la condizione di curvatura, ottenendo quella che è chiamata la condizione di Wolfe forte:

$$| \phi'(\alpha) | \leq c_2 | \phi'(0) | \tag{25}$$

3.4.1 Algoritmo per il calcolo del passo di ottimizzazione

L'algoritmo di line search implementato [8, cap. 3] soddisfa le condizioni forti di Wolfe, (23) & (25), ed è composto di due fasi:

1. **Bracketing**. Si inizia con un passo di ottimizzazione di prova, $\alpha_1 = 1$, e si continua ad incrementarlo finché non si trova un α che soddisfa le condizioni di forti di Wolfe, o un intervallo $[\alpha_{low}, \alpha_{high}]$ contenenti potenziali step. Il ciclo principale viene eseguito per $max\_iterations = 100$ iterazioni, fissato arbitrariamente. (vedi algorithm 1)

2. **Selezione**. Si decrementa la dimensione dell'intervallo $[\alpha_{low}, \alpha_{high}]$ bisezionandolo e riassegnando gli estremi, fino a trovare un α soddisfacente le condizioni forti di Wolfe. Il ciclo viene ripetuto per $\max\_iterations = 100$ iterazioni, fissato arbitrariamente. (vedi algorithm 2)

L'algoritmo assume che la direzione di ricerca p_k sia una direzione di discesa (i.e. $\phi'(0) < 0$), altrimenti si termina il programma sollevando un'eccezione. Inoltre se nella prima o nella seconda fase si supera il numero massimo di iterazioni, la line search termina restituendo $\alpha = -1$. In questo caso anche l'ottimizzazione termina.

Algorithm 1: Armijo-strong Wolfe Line Search. Fase di bracketing. Il passo di ottimizzazione di prova α_i viene incrementato ad ogni iterazione per mezzo di $\alpha_i \leftarrow \alpha_i/\theta$, con $\theta \in (0,1)$.

$\alpha_0 \leftarrow 0$
$\alpha_1 \leftarrow 1$
for *i = 1 to max_iterations* **do**
  if ***not*** $\phi(\alpha_i) \leq \phi(0) + c_1\alpha_i\phi'(0)$ ***or*** $[\phi(\alpha_i) \geq \phi(\alpha_{i-1})$ ***and*** $i > 1]$ **then**
    $\alpha^* \leftarrow zoom(\alpha_{i-1}, \alpha_i)$
    return α^*
  end if
  if $|\phi'(\alpha_i)| \leq -c_2\phi'(0)$ **then**
    $\alpha^* \leftarrow \alpha_i$
    return α^*
  end if
  if $\phi'(\alpha_i \geq 0)$ **then**
    $\alpha^* \leftarrow zoom(\alpha_i, \alpha_{i-1})$
    return α^*
  end if
  $\alpha_i \leftarrow \alpha_i/\theta$
end for
return -1

Algorithm 2: Zoom. Fase di selezione.

for *j = 0 to max_iterations* **do**
  $\alpha_j \leftarrow (\alpha_{low} + \alpha_{high})/2$
  if ***not*** $\alpha_j \leq \phi(0) + c_1\alpha_j\phi'(0)$ ***or*** $\phi(\alpha_j) \geq \phi(\alpha_{low})$ **then**
    $\alpha_{high} \leftarrow \alpha_j$
  end if
  else
    if $|\phi'(\alpha_j)| \leq -c_2\phi'(0)$ **then**
      $\alpha^* \leftarrow \alpha_j$
      return α^*
    end if
    if $\phi'(\alpha_j)(\alpha_{high} - \alpha_{low}) \geq 0$ **then**
      $\alpha_{high} \leftarrow \alpha_{low}$
    end if
    $\alpha_{low} \leftarrow \alpha_j$
  end if
  if $|\alpha_{high} - \alpha_{low}| < 10^{-16}$ **then**
    return -1
  end if
end for
return -1

3.5 Iperparametri di allenamento e criteri di stop

Tabella 3 riassume gli iperparametri di allenamento e i criteri di stop usati per ogni algoritmo. Per lo SGD si fa la messa a punto del learning rate α, momentum μ, forza di regolarizzazione λ, dimensione della *batch* e numero iterazioni. L'algoritmo BFGS ha come iperparametri la forza di regolarizzazione λ e il numero di iterazioni. L-BFGS ha in aggiunta il numero m di differenze di gradienti e soluzioni. Entrambi hanno come criteri di stop il raggiungimento del numero massimo di iterazioni, oppure la vanificazione del gradiente ad una norma di minore di ϵ, deciso empiricamente relativamente al dataset e all'algoritmo utilizzato con il fine di ottenere modelli più robusti. Infine la line-search viene messa a punto tramite i parametri della condizione di decremento sufficiente, c_1, e della condizione di curvatura, c_2. Sono inoltre iperparametri il numero massimo di iterazioni della prima e seconda fase, e il divisore θ. I criteri di stop sono il soddisfacimento delle condizioni forti di Wolfe o il raggiungimento del numero massimo di iterazioni.

Tabella 3: Iperparametri di allenamento e criteri di stop per ogni algoritmo.

Algoritmo	Iperparametri					Criteri stop
SGD	α	μ	λ	batch_size	iterazioni	iterazioni; $\|\|\nabla E\|\| < \epsilon$
BFGS			λ		iterazioni	iterazioni; $\|\|\nabla E\|\| < \epsilon$
L-BFGS		m	λ		iterazioni	iterazioni; $\|\|\nabla E\|\| < \epsilon$
A-W line search		c_1	c_2	θ	iterazioni	iterazioni; condizioni di Wolfe forte

4 Analisi della convergenza

In questa sezione vengono in primo luogo riassunti i risultati teorici di convergenza degli algoritmi SGD, BFGS e L-BFGS. Infine viene discusso se i risultati si applicano al problema in questione.

4.1 Preliminari sulla convergenza

Sia $f : \mathbb{R}^n \to \mathbb{R}$ la funzione obiettivo, gli algoritmi di ottimizzazione senza vincoli convergono ad un punto stazionario x^* dove il gradiente della funzione si annulla (vedi Appendice B).

4.1.1 Convergenza dei metodi line search

La convergenza della sequenza di iterati $\{x_k\}$ dipende dalla scelta della direzione p_k e dalla dimensione dello step α_k, ed è condizionata dal soddisfacimento di alcune assunzioni [8, cap. 3]. Sia θ_k l'angolo tra p_k e la direzione di massimo decremento, $-\nabla f(x_k)$, definito come:

$$\cos\theta_k = \frac{-\nabla f(x_k)^\intercal p_k}{||\nabla f(x_k)|| \cdot ||p_k||} \tag{26}$$

Teorema 1 (Condizione di Zoutendijk) *Si consideri un'iterazione del tipo $x_{k+1} = x_k + \alpha_k p_k$, dove p_k è una direzione di discesa e α_k soddisfa le condizioni di Wolfe. Supporre che f sia limitata inferiormente in $\mathbb{R}^n$ e che f sia continuamente differenziabile in un insieme N aperto contenente l'insieme di livello $\{x : f(x) \leq f(x_0)\}$, dove x_0 è il punto iniziale dell'iterazione. Supporre inoltre che il gradiente ∇f sia Lipschitz-continuo in N. Allora*

$$\sum_{k=0}^{\infty} \cos^2\theta_k ||\nabla f(x_k)||^2 < \infty \tag{27}$$

La condizione di Zoutendijk (27) implica che

$$\cos^2\theta_k ||\nabla f(x_k)||^2 \to 0 \tag{28}$$

Inoltre se la direzione p_k forma un angolo $\theta_k < 90°$ con la direzione di massimo decremento, $-\nabla f(x_k)$, esiste una costante $\delta > 0$ tale che

$$\cos\theta_k \geq \delta > 0 \quad \forall k \tag{29}$$

Segue direttamente da (28) che

$$\lim_{k\to\infty} ||\nabla f(x_k)|| = 0 \tag{30}$$

In breve, il gradiente $||\nabla f_k||$ converge a zero se le direzioni di ricerca p_k non sono ortogonali a $-\nabla f(x_k)$.

Condizione di Zoutendijk per i metodi quasi-Newton Nel caso dei metodi quasi-Newton, supporre che le matrici B_k siano definite positive, $B_k \succ 0 \quad \forall k$, e che abbiano un numero di condizionamento uniformemente limitato, ovvero:

$$k(B_k) = ||B_k|| \cdot ||B_k^{-1}|| = \frac{\sigma_1}{\sigma_n} \leq M \quad \forall k \tag{31}$$

dove $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$ sono i valori singolari della matrice B. È possibile dimostrare (vedi Appendice C) che da (26) segue che

$$\cos\theta_k \geq 1/M > 0 \tag{32}$$

Combinando (32) con (28) si ottiene che

$$\lim_{x\to\infty} ||\nabla f(x_k)|| = 0$$

Pertanto si è dimostrato che i metodi quasi-Newton convergono ad un punto stazionario x^* se vengono soddisfatte le seguenti condizioni [8, cap. 3.2, pag.40]:

1. Le matrici B_k hanno un numero di condizionamento limitato.
2. Le matrici B_k sono definite positive (necessario per avere direzioni p_k di discesa).
3. I passi di ottimizzazione α_k soddisfano le condizioni di Wolfe .

4.2 Convergenza discesa del gradiente

In questa sezione vengono riportati i risultati di convergenza del metodo di discesa del gradiente con passo di ottimizzazione α fisso. I risultati che seguono si basano sulla seguente assunzione.

Assunzione 1 *Il gradiente ∇f è Lipschitz-continuo con constante $L > 0$, ovvero*

$$||\nabla f(x) - \nabla f(y)|| \leq L \cdot ||x - y|| \quad \forall x, y \in \mathbb{R}^n$$

4.2.1 Convergenza globale SGD

Nel caso di funzione obiettivo convessa si dispone del seguente teorema

Teorema 2 *Sia x^* un punto di minimo globale. Se la funzione obiettivo $f : \mathbb{R}^n \to \mathbb{R}$ è convessa e differenziabile, allora la discesa del gradiente con passo di ottimizzazione fisso $\alpha \leq 1/L$ soddisfa:*

$$f(x_k) - f(x^*) \leq \frac{||x_0 - x^*||}{2\alpha k}$$

Le conseguenze del Teorema 2 sono le seguenti:

1. Se f è convessa, l'algoritmo ha convergenza sub-lineare, ovvero la relazione $f(x_k) - f(x^*) \leq \epsilon$ viene soddisfatta in $O(1/\epsilon)$ iterazioni.
2. Se f è strettamente convessa, l'algoritmo ha convergenza lineare, e la relazione $f(x_k) - f(x^*) \leq \epsilon$ viene soddisfatta in $O(\log(1/\epsilon))$ iterazioni.

4.2.2 Convergenza locale SGD

In generale se f non è convessa l'algoritmo non garantisce la convergenza ad un minimo globale, ma solo ad un minimo locale. I seguenti risultati, studiati da Lee et Al [4, sez. 4] affermano che la discesa del gradiente, con inizializzazione casuale e passo di ottimizzazione α sufficientemente piccolo, converge ad un minimo locale o $-\infty$ quasi sicuramente. Oltre a valere l'assunzione 1, si assume che f sia di classe C^2.

Teorema 3 (Discesa del gradiente non converge mai ad un punto di sella) *Sia $f \in C^2$, x^* un punto di sella stretto, e $\alpha < 1/L$ il passo di ottimizzazione. Allora*

$$P(\lim_k x_k = x^*) = 0$$

Corollario 1 *Sia S l'insieme dei punti di sella, assumendo che siano tutti stretti. Se S è al più infinitamente numerabile, allora*

$$P(\lim_k x_k \in S) = 0$$

Teorema 4 (Discesa del gradiente converge ad un minimo locale) *Assumendo le stesse condizioni del corollario 1 e che $\lim_k x_k$ esiste, allora*

$$P(\lim_k x_k = x^*) = 1$$

dove x^* è un minimo locale.

Lee at Al [4] riportano inoltre le seguenti condizioni sufficienti affinché il limite $\lim_k x_k$ esista. La seguente proposizione 1 garantisce che la sequenza di soluzioni $\{x_k\}$ non converga a ∞ imponendo che f abbia insiemi di sottolivello compatti.

Proposizione 1 *Se f è continuamente differenziabile, ha punti stazionari isolati e insiemi di sottolivello compatti, allora il limite $\lim_k x_k$ esiste e tale limite è un punto stazionario di f.*

La seconda condizione sufficiente per l'esistenza del limite $\lim_k x_k$ è basata sulla disuguaglianza del gradiente di Lojasiewicz, la quale caratterizza la pendenza del gradiente in un intorno di un punto stazionario, assicurando che $x_{k+1} - x_k$ sia una quantità finita. Inoltre permette di derivare la velocità di convergenza ad un minimo locale.

Definizione 1 (Diseguaglianza di Lojasiewicz) *Un punto stazionario x^* soddisfa la diseguaglianza del gradiente di Lojasiewicz se esiste un intorno N, $\beta, \epsilon > 0$ e $\gamma \in (0, 1)$ tali che*

$$||\nabla f(x)|| \geq \beta |f(x) - f(x^*)|^\gamma$$

per tutti gli x in $\{x \in N : f(x^*) < f(x) < f(x^* + \epsilon)\}$

Proposizione 2 *Se le condizioni del corollario 1 sono soddisfatte e la sequenza $\{x_k\}$ non converge ad ∞, allora esiste il limite $\lim_k x_k = x^*$, con x^* minimo locale. Se inoltre x^* soddisfa la diseguaglianza di Lojasiewicz, allora*

$$||x_k - x^*|| \leq \begin{cases} Cd^k & \gamma \in (0, 1/2] \\ \frac{C}{k^{(1-\gamma)/(2\gamma-1)}} & \gamma \in (1/2, 1) \end{cases}$$

per qualche C e $d < 1$ indipendente da k.

4.3 Convergenza quasi-Newton

Si riportano i risultati generali di convergenza dei metodi quasi-Newton [8, cap. 3]. Si ricorda che per i metodi quasi-Newton, la condizione (30) è valida se le matrici B_k hanno un numero di condizionamento limitato e se sono definite positive. I risultati di questa sezione si basano sulla seguente assunzione:

Assunzione 2

i. Il passo di ottimizzazione α_k, calcolato tramite line search inesatta, soddisfa le condizioni di Wolfe o Wolfe forte.

ii. La line search prova sempre $\alpha = 1$ al primo passo, accettandolo se soddisfa le condizioni di Wolfe.

Teorema 5 e Teorema 6 affermano che se la direzione quasi-Newton p_k approssima la direzione di Newton abbastanza bene, allora il passo di ottimizzazione $\alpha = 1$ soddisferà le condizioni di Wolfe mentre gli iterati x_k convergono alla soluzione.

Teorema 5 *Sia $f : \mathbb{R}^n \to \mathbb{R}$ con $f \in C^3$. Si consideri l'iterazione $x_{k+1} = x_k + \alpha_k p_k$, dove $p_k = -B_k^{-1} \nabla f(x_k)$ è una direzione di discesa e α_k soddisfa le condizioni di Wolfe con $c_1 \leq 1/2$. Se la sequenza $\{x_k\}$ converge ad un punto x^* tale che $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*) \succ 0$, e se p_k soddisfa*

$$\lim_{k \to \infty} \frac{||(B_k - \nabla^2 f(x^*))p_k||}{||p_k||} = 0 \tag{33}$$

allora,

i. $\exists k_0 : \alpha_k = 1$ è ammissibile $\forall k > k_0$

ii. Se $\alpha_k = 1$ $\forall k > k_0$, $\{x_k\}$ converge a x^* superlinearmente.

Teorema 6 *Sia $f : \mathbb{R}^n \to \mathbb{R}$ con $f \in C^3$. Si consideri l'iterazione $x_{k+1} = x_k + p_k$ ($\alpha_k = 1$ $\forall k$) e $p_k = -B_k^{-1} \nabla f(x_k)$. Supporre che la sequenza $\{x_k\}$ converga ad un punto x^* tale che $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*) \succ 0$. Allora $\{x_k\}$ converge superlinearmente se e solo se (33) è soddisfatta.*

4.4 Convergenza BFGS

L'analisi della convergenza del BFGS viene articolata con teoremi di convergenza *globale* e *locale*, e le relative assunzioni fatte sulla funzione obiettivo f e le soluzioni x [8, cap. 8].

4.4.1 Convergenza globale BFGS

Lo studio della convergenza globale viene fatto nel caso in cui l'algoritmo BFGS viene applicato ad funzione obiettivo convessa e smooth, partendo da un punto arbitrario x_0 e da un'approssimazione dell'hessiana B_0 che sia simmetrica e definita positiva. Vengono inoltre fatte le seguenti assunzioni:

Assunzione 3 *1. La funzione obiettivo f è continuamente differenziabile due volte.*

2. L'insieme di livello $\Omega = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ è convesso.

3. Esistono costanti positive m, M tali che, $\forall z \in \mathbb{R}^n$ ed $x \in \Omega$ si ha:

$$m \cdot ||z||^2 \leq z^\intercal \nabla^2 f(x) z \leq M \cdot ||z||^2$$

L'assunzione 3 implica che $\nabla^2 f(x)$ è definita positiva in Ω e che f ha un unico punto di minimo $x^* \in \Omega$.

Teorema 7 *Sia $B_0 \in \mathbb{R}^{n \times n}$ simmetrica e definita positiva. Sia x_0 la soluzione iniziale, per la quale assunzione 3 è soddisfatta. Allora la sequenza $\{x_k\}$ generata dall'algoritmo BFGS converge ad un minimo x^* di f.*

Il teorema 7 è stato generalizzato all'intera famiglia Broyden. Pertanto questo risultato vale anche per l'algoritmo L-BFGS. Inoltre si ha che la sequenza $\{||x_k - x^*||\}$ converge a zero linearmente. La convergenza è invece superlineare se la sequenza soddisfa:

$$\sum_{k=1}^{\infty} ||x_k - x^*|| < \infty \tag{34}$$

4.4.2 Convergenza locale BFGS

Lo studio della convergenza locale del BFGS è valido per tutte le funzioni obiettivo non lineari, anche non convesse. Alle assunzioni 3 viene aggiunta la seguente:

Assunzione 4 *La matrice Hessiana $\nabla^2 f(x)$ è Lipschitz continua in x^*. Ovvero, per ogni x in un intorno di x^*, esiste una costante positiva L tale che:*

$$||\nabla^2 f(x) - \nabla^2 f(x^*)|| \leq L \cdot ||x - x^*||$$

Teorema 8 *Sia f due volte differenziabile. Sia la sequenza delle soluzioni $\{x_k\}$ convergente ad un minimo x^*, per il quale l'assunzione 4 è soddisfatta. Sia inoltre soddisfatta la relazione (34), allora $\{x_k\}$ converge a x^* superlinearmente.*

4.5 Convergenza L-BFGS

L'enunciato e la dimostrazione del seguente teorema sono riportate in Liu et Al. [6].

Teorema 9 *Sia il punto x_0 il punto iniziale per il quale f soddisfa assunzione 3. Supporre che le metrici iniziali $B_k^{(0)}$ siano tali che $\{||B_k^0||\}$ e $\{||B_k^{(0)^{-1}}||\}$ siano bounded. Allora per qualsiasi $B_0 \succ 0$, L-BFGS genera una sequenza $\{x_k\}$ che converge al minimo globale x^*. Inoltre esiste una costante $r \in [0, 1)$ tale che*

$$f(x_k) - f(x^*) \leq r^k (f(x_0) - f(x^*))$$

cioè $\{x_k\}$ converge $R-$linearmente.

4.6 Problemi nell'ottimizzazione di reti neurali

L'ottimizzazione di reti neurali può presentare varie complicazioni tali da renderlo un task difficile.

Malcondizionamento Il malcondizionamento della matrice Hessiana H può risultare in un apprendimento molto lento anche in presenza di un forte gradiente, perché il *learning rate* deve essere scalato per compensare la forte curvatura [3, sec. 8.2.1]. Si consideri il prodotto $H^{-1}\nabla f(x)$. Se H è stimata perfettamente ma è malcondizionata, moltiplicare per H amplifica gli errori pre-esistenti in $\nabla f(x)$. Pertanto piccole perturbazioni $\delta\nabla f(x)$ causano grandi cambiamenti in $H^{-1}\nabla f(x)$. Nei metodi Quasi-newton la matrice H è solo un'approssimazione della vera matrice Hessiana, quindi il prodotto potrebbe contenere anche errori maggiori [3, p. 273].

Minimi locali La superfice della funzione obiettivo potrebbe presentare molti minimi locali x^*, anche con valori $f(x^*)$ simili, se non uguali, o buoni quanto il minimo globale. Inoltre soluzioni con costo simile possono avere errori di generalizzazione molto differenti. I minimi locali possono essere problematici quando hanno un costo molto maggiore rispetto a quello del minimo globale. Tuttavia per reti neurali sufficientemente grandi, la maggior parte dei minimi locali hanno un costo buono quanto un minimizzatore globale. Pertanto non è importante trovare un minimo globale [3, sec. 8.2.2].

Punti di sella e regioni piatte Per molte funzioni non-convesse multi-dimensionali, i minimi locali sono rari rispetto ai punti di sella. I punti di sella possono essere considerati come minimi locali lungo una direzione, e massimi locali lungo un'altra direzione. Per i problemi di ottimizzazione del primo ordine, il gradiente spesso diventa molto piccolo vicino ad un punto di sella. Tuttavia, da risultati empirici sembra che l'algoritmo di Discesa del Gradiente sia in grado di fuggire dai punti di sella in molti casi. Secondo un risultato teorico riportato in Teorema 3, sotto certe condizioni l'algoritmo di Discesa del Gradiente non converge mai ad un punto di sella. Invece per i metodi di Newton, i punti di sella costituiscono un problema, in quanto sono esplicitamente progettati per trovare un punto dove il gradiente si annulla. Pertanto senza appropriate modificazioni, possono finire in un punto di sella. Questo è un problema perché nei punti di sella, la matrice Hessiana ha autovalori positivi e negativi, ma (come discusso in sottosottosezione 4.1.1) i metodi Quasi-newton necessitano di una matrice definita positiva per avere una direzione p di discesa. Questi motivi potrebbero spiegare perché in problemi multi-dimensionali i metodi del secondo ordine non hanno rimpiazzato completamente i metodi di discesa del gradiente per l'allenamento di reti neurali [3, sec. 8.2.3].

Inizializzazione dei parametri Molti algoritmi sono fortemente influenzati dalla scelta dei parametri iniziali. Il punto iniziale può determinare del tutto se l'algoritmo converge. Certi punti iniziali possono essere così instabili che l'algoritmo incontra errori numerici e fallisce. Quando l'ottimizzazione converge, il punto iniziale può determinare quanto velocemente si converge, se si converge ad un buon minimo locale o meno; può anche influenzare la capacità di generalizzazione. Un euristica da rispettare nell'inizializzazione dei parametri è quella di "rompere la simmetria" tra neuroni differenti, cosicché nessuna risulti ridondante. Questo motiva l'inizalizzazione casuale. Spesso si inizializzano i parametri con valori campionati casualmente da una distribuzione Gaussiana o uniforme. Anche il magnitudo dei parametri iniziali influenza la procedura di ottimizzazione e l'abilità di generalizzare. Dal punto di vista dell'ottimizzazione, una larga distribuzione risulta in una più marcata rottura della simmetria, la quale aiuta a mantenere il segnale durante la propagazione in avanti e all'indietro dell'input e del gradiente rispettivamente. Tuttavia, parametri troppo grandi possono risultare in *exploding values*, che possono causare la saturazione delle funzioni di attivazione (e.g. Sigmoid, TanH) e la successiva vanificazione del gradiente rispetto ai neuroni saturati. Dal punto di vista del Machine Learning, si prediligono parametri piccoli per aumentare la capacità di generalizzazione. Detto questo, anche se esistono metodi sofisticati per l'inizializzazione (e.g [2]), in questo progetto si è scelto di inizializzare i pesi dall'intervallo $[-0.5, 0.5]$ seguendo una distribuzione uniforme.

4.7 Discussione sulla convergenza

La discussione sulla convergenza concerne il caso in cui vengano usate le attivazioni TanH negli hidden layers, mentre nel layer di output venga usata TanH per il task di classificazione (dataset MONK), e Lineare per il task di regressione (dataset ML-CUP). A seguito dello studio della funzione obiettivo svolto in sezione 2, si ha che E è non-Lipschitz-continua, non convessa. Inoltre ∇E e $\nabla^2 E$ sono Lipschitz continue. Si hanno poi due possibilità:

- Se $E := MSE$, allora E è regolare, cioè $E \in C^\infty$.
- Se $E := MEE$, allora $E \in C^0$, in quanto la derivata di (5) è indefinita in $\mathbf{0}$ (vedi anche figura 1e).

In ogni caso E è limitata inferiormente dal valore 0, e $dom(E) = \mathbb{R}^n$ è un insieme convesso. Lo studio dell'applicabilità dei teoremi di convergenza globale viene subito scartato in quanto la funzione obiettivo in esame non è convessa.

SGD Innanzi tutto l'assunzione 1 è soddisfatta in quanto il gradiente ∇E è Lipschitz continuo. I risultati riportati in sottosottosezione 4.2.2 suppongono che la funzione obiettivo sia di classe C^2. Anche questa assunzione è soddisfatta se $E := MSE$ perché $E \in C^\infty$. Non è soddisfatta se $E := MEE$. Per quanto riguarda l'applicabilità di Teorema 3, non si sa se la funzione obiettivo abbia punti di sella stretti o meno. In ogni caso, selezionando un passo di ottimizzazione $\alpha < 1/L$ sufficientemente piccolo, dove L è la costante di Lipschitz (anche essa ignota), l'algoritmo convergerà ad un punto di sella con probabilità zero. Se si sceglie un passo di ottimizzazione troppo elevato, $\alpha \geq 1/L$, allora c'è il rischio di convergere ad un punto di sella. Inoltre per il Teorema 4 la discesa del gradiente converge ad un punto di minimo locale con probabilità 1, se il limite $\lim_k x_k$ non tende ad ∞. Per dimostrare l'esistenza del limite si deve verificare che le ipotesi della proposizione 1 siano soddisfatte. Si può dimostrare che la funzione E ha insiemi di sottolivello compatti, ma non si sa se abbia punti stazionari isolati o meno. Tuttavia possiamo assicurarci che la sequenza degli iterati $\{x_k\}$ rimanga all'interno di un sottonsieme compatto di $\mathbb{R}^n$ grazie alla regolarizzazione di Tikhonov, che penalizza le soluzioni con norma elevata. Per quanto riguarda la velocità di convergenza, non si è in grado di concludere nulla, perché non si sa se i punti stazionari di E soddisfano o meno la disuguaglianza di Lojasiewicz.

BFGS e L-BFGS Le ipotesi della condizione di Zoutendijk (27) sono soddisfatte se $E := MSE$, in quanto **(i)** p_k è una direzione di discesa (viene effettuato il controllo $\phi'(0) < 0$), **(ii)** α_k soddisfa le condizioni di Wolfe, **(iii)** la funzione obiettivo E è limitata inferiormente dal valore 0, **(iv)** E è continuamente differenziabile in $\mathbb{R}^n$, e **(v)** ∇E è Lipschitz-continuo in $\mathbb{R}^n$. Se invece $E := MEE$, la condizione (iv) salta in quanto $E := MEE \in C^0$, e quindi la condizione di Zoutendijk non vale. Pertanto la convergenza ad una soluzione x^* tale che $||\nabla E(x^*)|| = 0$ non è garantita.

Inoltre, come dimostrato in sottosottosezione 4.1.1, la direzione p_k forma un angolo di $\theta_k < 90°$ con la direzione di massimo decremento solo se **(a)** $B_k \succ 0$ e **(b)** $k(B_k) = \frac{\sigma_1}{\sigma_n}$ è limitato. Innanzitutto la condizione (a) è garantita dai metodi Quasi-newton in questione. Invece la condizione (b) vale se il rapporto tra il massimo e il minimo valore singolare è contenuto, ovvero se σ_{max} e σ_{min} sono relativamente molto grandi e molto piccoli rispettivamente. Se (a) e (b) valgono, allora la condizione (29) è soddisfatta, da cui segue (30), ovvero $\lim_{k\to\infty} ||\nabla E(x_k)|| = 0$. Quindi gli algoritmi BFGS e L-BFGS convergono ad un punto stazionario x^* tale che $\nabla E(x^*) = 0$ se si verificano (i-v) e (a-b). In particolare, la convergenza ad un minimizzatore locale è garantita dai requisiti imposti su $p_k = -B_k^{-1}\nabla E(x_k)$ dall'informazione di curvatura dell'approssimazione dell'Hessiana. Si aggiunge che eventuali problemi numerici potrebbero compromettere la convergenza nel caso in cui il condizionamento di B diventi elevato, o che B diventi gradualmente non definita positiva, con autovalori prossimi a zero (o addirittura negativi).

Per quanto riguarda la velocità di convergenza (Teorema 5 e Teorema 6) la relazione (33), ovvero $\lim_{k\to\infty} \frac{||(B_k - \nabla^2 f(x^*))p_k||}{||p_k||} = 0$, è condizione necessaria e sufficiente per avere convergenza superlineare. Le ipotesi del Teorema 5 sono soddisfatte se $E := MSE$ in quanto **(i)** $E := MSE \in C^\infty$, **(ii)** p_k è direzione di discesa, **(iii)** α_k soddisfa le condizione di Wolfe con $c_1 \leq 0.5$, e **(iv)** $\{x_k\} \to x^*$ con $\nabla E(x^*) = 0$, e $\nabla^2 E(x^*) \succ 0$. Se infine la condizione (33) è rispettata, il passo di ottimizzazione $\alpha_k = 1$ sarà ammissibile (e scelto dalla line search) definitivamente ogni iterazione $k > k_0$, ottenendo una convergenza superlineare. Se invece $E := MEE$ la convergenza sarà lenta perché la scelta del passo di ottimizzazione $\alpha = 1$ non sarà garantita.

5 Metodo

Vengono descritti gli strumenti utilizzati per la realizzazione del progetto, le principali scelte di design ed implementazione, e la metodologia usata per gli esperimenti.

Tools e librerie Il progetto è stato sviluppato con i seguenti strumenti: Python come linguaggio base, per la facilità di utilizzo e la grande quantità di librerie open source a disposizione. In particolare le librerie Pandas e Scikit-learn sono state usate per il pre-processing dei dataset. La libreria Numpy è stata usata per i calcoli matematici (sopratutto per i prodotti scalari). Infine Matplotlib è stata usata per il rendering dei risultati degli esperimenti.

Software design Lo sviluppo del software ha seguito un approccio *Object Oriented*. Il software è composto di tre classi principali: NeuralNetwork, Layer, e Neuron. In particolare la classe NeuralNetwork è costituita da una lista di Layer, a loro volta aventi una lista di Neuron. Ogni Neuron ha una lista contenente i pesi associati ai suoi input, e implementa le funzioni di attivazione e le relative derivate. Sebbene lo scopo del progetto non era di implementare un framework, è stato cercato di riutilizzare il più possibile ogni porzione di codice, parametrizzando ed automatizzando tutti gli esperimenti. Infine per assicurare la "correttezza" di quanto implementato, i metodi più importanti della rete neurale (per esempio le funzioni che implementano *forward-propagation*, *back-propagation* e il *training*) sono state supportate da unit-test costantemente aggiornati.

Scelte implementative Sono state fatte le seguenti scelte implementative. L'architettura e la topologia della rete neurale sono completamente determinate dai due parametri del costruttore della classe NeuralNetwork, *architettura* e *neuroni*. In particolare il primo è un array di interi in cui l'i-esima entry indica il numero di neuroni presenti all'i-esimo layer, mentre il secondo è un array, della stessa lunghezza del primo, che indica il tipo di neuroni presenti in ogni layer.

La rete neurale implementa i metodi *train_SGD()*, *train_BFGS()* e *train_LBFGS()* per effettuare l'allenamento con i vari algoritmi di apprendimento. Per lo SGD il tipo di allenamento (batch, mini-batch, online) è interamente definito dal parametro *batch_size* della funzione *train_SGD()*. Inoltre è possibile salvare lo stato dell'allenamento scrivendo il tipo di architettura e i pesi della rete su file tramite la funzione *dump_weights()*. Successivamente è possibile caricare lo stato della rete tramite la funzione *load_weights()*. Queste funzioni sono per avere diversi "snapshot" di vari modelli al fine di confrontarli tra loro.

Validazione Il rischio empirico è stato calcolato usando la tecnica di validazione *holdout*, dividendo i dati di allenamento in due partizioni disgiunte, *training set* e *validation set*. La prima partizione viene usata per allenare la rete neurale, mentre la seconda per validarne le prestazioni e decidere la miglior impostazione di iper-parametri. Ogni esperimento viene effettuato 5 volte arbitrariamente. Infine la capacità di generalizzazione dei modelli è stata valutata misurando le prestazioni ottenute su un *test set* indipendente dai due precedenti.

La selezione di migliori modelli è stata fatta tramite grid search. Le metriche usate per valutare le prestazioni dei modelli sono state Accuracy, Mean Squared Error e Mean Euclidean Error.

Esperimenti preliminari Gli esperimenti preliminari hanno individuato a grandi linee, per ogni iper-parametro di allenamento, gli ordini di grandezza e intervalli di valore che risultavano in migliori prestazioni e/o tempo di convergenza. Inoltre uno studio più o meno approfondito è stato svolto per individuare il numero di hidden layer e neuroni necessari per il *fitting* dei dataset.

6 Risultati Ottimizzazione

I primi esperimenti sono rilevanti dal punto di vista della Matematica Computazionale. Ci si concentra quindi nel fare il fitting dei training set, senza preoccuparsi delle capacità di generalizzazione. Thrun et Al. [9, pag. 102] riportano che sono sufficienti reti neurali molto piccole per fittare i training set dei MONK. In particolare è sufficiente un singolo hidden layer con 3, 2 e 4 neuroni rispettivamente per MONK-1 , MONK-2 e MONK-3. Mentre per il dataset CUP si è usata una rete neurale con un solo hidden layer con 5 neuroni (vedi Tabella 4).

Tabella 4: Architettura reti neurali usate per gli esperimenti.

	MONK-1	MONK-2	MONK-3	ML-CUP
architettura	[17, 3, 1]	[17, 2, 1]	[17, 4, 1]	[10, 5, 2]
attivazioni	[Input, Tanh*]	[Input, Tanh*]	[Input, Tanh*]	[Input, Tanh, Lineare]

I criteri di stop sono i seguenti: $||\boldsymbol{\nabla E(w)}|| < \boldsymbol{\epsilon = 10^{-5}}$ e $\boldsymbol{iterazioni = 10^4}$. Gli iperparametri di allenamento usati sono quelli solitamente consigliati. Particolari eccezioni sono rese note. Per SGD è stato usato $\boldsymbol{\alpha = 0.01}$, $\boldsymbol{\mu = 0.9}$, $\boldsymbol{batch\_size = size(data)}$[4]. Per L-BFGS si ha $\boldsymbol{m = 5}$. Il coefficiente di regolarizzazione è $\boldsymbol{\lambda \in \{0, 0.05\}}$. Per la Armijo-Wolfe line-search si è usato $\boldsymbol{c_1 = 10^{-4}}$, $\boldsymbol{c_2 = 0.9}$ e $\boldsymbol{\theta = 0.9}$, mentre per la backtracking $\boldsymbol{c_1 = 10^{-4}}$ e $\boldsymbol{\theta = 0.9}$. Con questi settaggi si sono prodotti i seguenti risultati:

Esperimento 1 Il comportamento dei metodi Quasi-newton con *Backtracking* e *Armijo-Wolfe* line search, per $\boldsymbol{iterazioni = 10^3}$, è mostrato in **??**. I successivi esperimenti usano solo la *Armijo-Wolfe*.

Esperimento 2

- Tabella 5 riassume le prestazioni in termini di numero iterazioni, errore sul training set e tempo di esecuzione.
- Figura 4 mostra le curve dell'andamento della norma del gradiente, $||\nabla E(x)||$.
- Figura 5 mostra le curve dell'andamento dell'errore relativo, $|E(x) - E(x^*)|$.
- Figura 6 mostra l'andamento del numero di condizionamento della matrice $H = B^{-1}$ vs numero di iterazioni, per l'algoritmo BFGS[5]. Gli autovalori della matrice H_{BFGS} in un caso particolare sono mostrati in Figura 7.
- Figura 8 mostra i passi di ottimizzazione α scelti dalla Armijo-Wolfe line search, ad ogni iterazione, impiegata negli algoritmi BFGS ed L-BFGS.

Esperimento 3 È stato fatto un confronto degli algoritmi BFGS e L-BFGS variando il parametro $m \in \{1, 3, 5, 7, 10, 15, 20, 25, 30\}$. Per entrambi gli algoritmi sono stati impostati i seguenti iperparametri: $c_1 = 10^{-4}$, $c_2 = 0.9$, $\theta = 0.9$, $\lambda = 0.05$, $\epsilon = 10^{-5}$. L'esperimento è stato svolto sul dataset MONK-3 per tre problemi di ottimizzazione rispettivamente in $\mathbb{R}^{20}$, $\mathbb{R}^{115}$ e $\mathbb{R}^{1141}$, per meglio comprendere il vantaggio dell'algoritmo L-BFGS all'aumentare del numero di variabili $n \in \{20, 115, 1141\}$. Le reti neurali usate per ciascun problema (sia per BFGS che per L-BFGS) sono state inizializzate con lo stesso insieme di pesi/parametri iniziali. Vengono riportati i seguenti risultati:

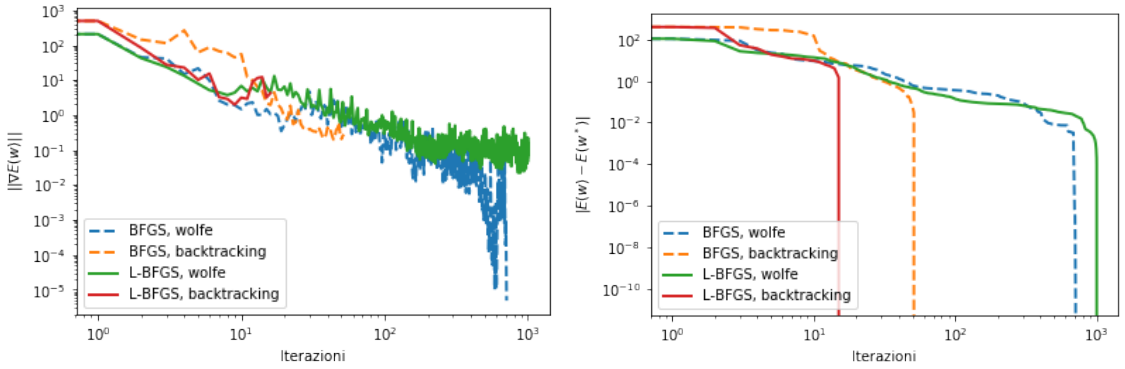
- Tabella 6 riassume i risultati in termini di numero iterazioni, tempo di allenamento e tasso tempo/iterazioni.
- Figura 9 mostra le curve dell'errore relativo e della norma del gradiente.

[4] Ottenendo di fatto l'algoritmo standard di discesa del gradiente con momentum.

[5] Per l'algoritmo L-BFGS non si è calcolato il numero di condizionamento perché la matrice H non è disponibile esplicitamente, in quanto viene usata la sub-routine iterativa *two-loop recursion* per ottenere direttamente il prodotto $H_k \nabla f(x_k)$.

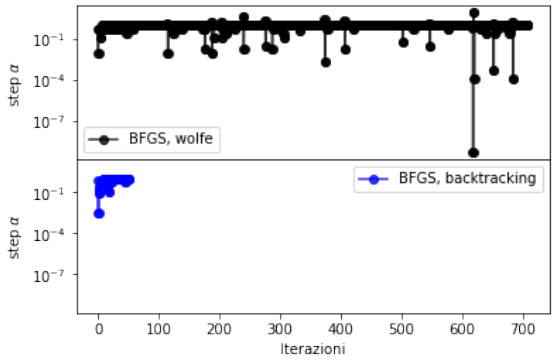
Esperimento 1

ML-CUP $\lambda = 0$

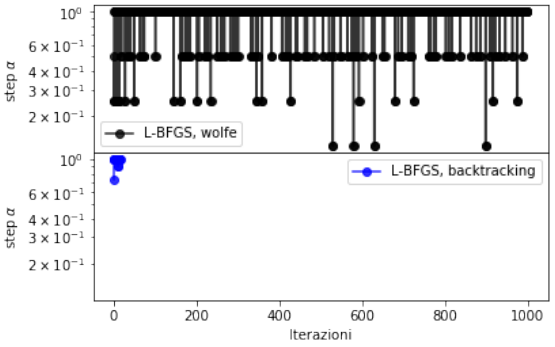


(a) norma gradiente

(b) gap relativo



(c) step size BFGS

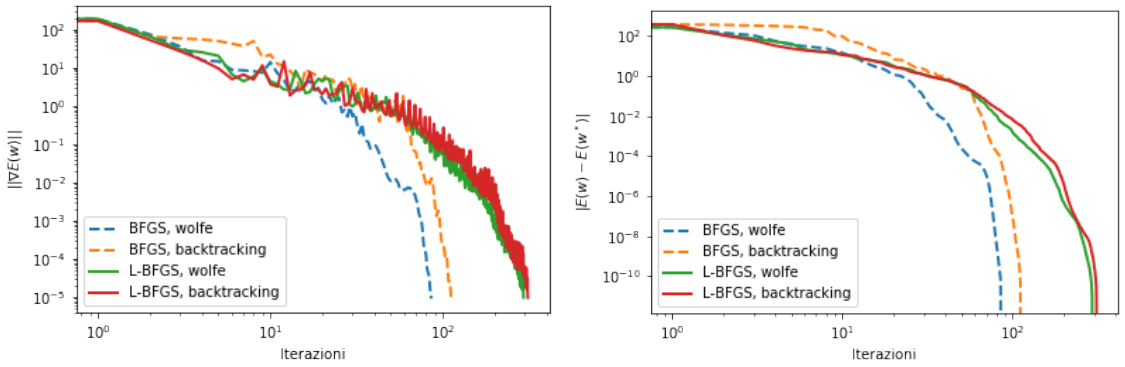


(d) step size L-BFGS

Figura 2: Quasi-newton a confronto con *backtracking* e *Armijo-Wolfe* line search con $\boldsymbol{\lambda = 0}$. Con *Armijo-Wolfe*, solo BFGS è converso ad un minimizzatore, mentre L-BFGS è arrivato al numero max di iterazioni senza raggiungere una norma del gradiente sufficientemente piccola, 2a. Con *backtracking*, l'ottimizzazione è stata terminata manualmente quando H è divenuta non definita positiva, con un solo autovalore negativo. Secondo la discussione riportata in sottosezione 4.6, gli algoritmi sono conversi ad un punto di sella.
Ci sono alcune curiosità riguardo figura 2b a cui non è stata data risposta. Uno studio più approfondito delle potrebbe rispondere alle seguenti domande: (i) perché usando la *backtracking* gli algoritmi hanno ridotto il gap relativo in meno iterazioni che usando *Armijo-Wolfe*. (ii) perché usando la *backtracking* L-BFGS è stato più veloce di BFGS, in termini di iterazioni, e viceversa usando *Armijo-Wolfe*.

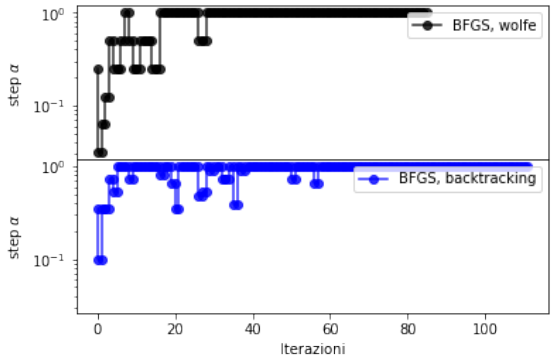
Esperimento 1

ML-CUP $\lambda = 0.05$

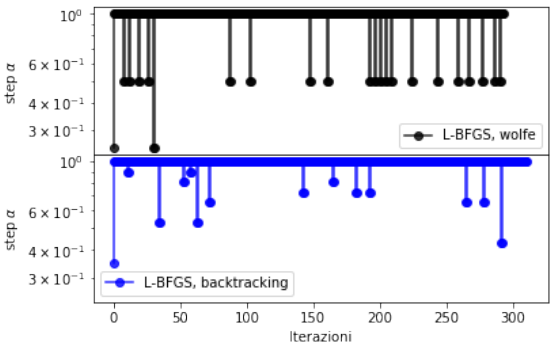


(a) norma gradiente

(b) gap relativo



(c) step size BFGS



(d) step size L-BFGS

Figura 3: Quasi-newton a confronto con *backtracking* e *Armijo-Wolfe* line search, con $\boldsymbol{\lambda = 0.05}$. La regolarizzazione fa da stabilizzatore, migliorando il funzionamento delle line search. Non ci sono problemi e le line search si comportano similmente, scegliendo $\boldsymbol{\alpha = 1}$ nelle ultime iterazioni (3c, 3d).

Esperimento 2

		GD	BFGS	L-BFGS
MONK-1 $\boldsymbol{\lambda = 0}$	Iterazioni	10000	145	92
	MSE	**0.00027**	0.0879	0.0967
	$\|\|\nabla E\|\|$	0.00057	$8 \cdot 10^{-6}$	$6 \cdot 10^{-6}$
	Tempo (s)	161	11	5
MONK-2 $\boldsymbol{\lambda = 0}$	Iterazioni	10000	56	70
	MSE	**0.414**	0.4854	0.5689
	$\|\|\nabla E\|\|$	0.0033	$9 \cdot 10^{-6}$	$2 \cdot 10^{-6}$
	Tempo (s)	207	3	4
MONK-3 $\boldsymbol{\lambda = 0}$	Iterazioni	10000	47	39
	MSE	**0.0322**	0.0983	0.0983
	$\|\|\nabla E\|\|$	0.0019	$7 \cdot 10^{-6}$	$6 \cdot 10^{-6}$
	Tempo (s)	213	3	2
CUP $\boldsymbol{\lambda = 0}$	Iterazioni	10000	1173	10000
	MSE	2.5975	2.6351	**2.5405**
	$\|\|\nabla E\|\|$	0.0012	0.1610	0.0086
	Tempo (s)	1090	325	2432
CUP $\boldsymbol{\lambda = 0.05}$	Iterazioni	1592	172	288
	MSE	**6.1089**	**6.1089**	6.4437
	$\|\|\nabla E\|\|$	$9 \cdot 10^{-6}$	$7 \cdot 10^{-6}$	$7 \cdot 10^{-6}$
	Tempo (s)	208	62	83

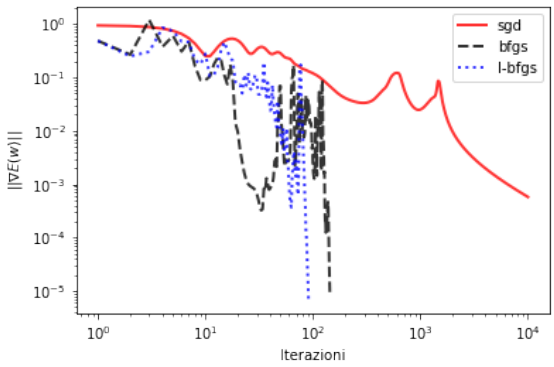
Tabella 5: Risultati ottenuti sul training set. Nei dataset MONK, I metodi Quasi-Newton convergono molto prima della discesa del gradiente ad un minimo locale, ottendendo un errore leggermente peggiore della discesa del gradiente. Con $\boldsymbol{\lambda = 0}$ nel dataset ML-CUP, BFGS termina prematuramente perché la matrice H è diventata non definita positiva, avendo un autovalore negativo, convergendo in un punto di sella. Invece con $\boldsymbol{\lambda = 0.05}$ gli algoritmi convergono molto più velocemente ad un minimo locale, ottenendo valori di loss maggiori. I migliori valori dell'errore per dataset sono mostrati in grassetto.

Esperimento 3

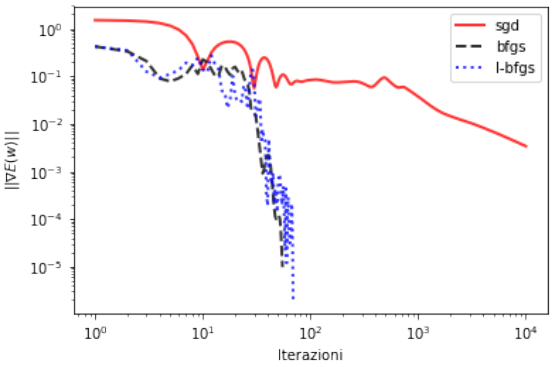
Tabella 6: Confronto algoritmi Quasi-newton al variare del parametro m, per tre problemi in $\mathbb{R}^{20}$, $\mathbb{R}^{115}$ e $\mathbb{R}^{1141}$. I migliori tempi di esecuzione per ogni n sono mostrati in grassetto. Gli andamenti delle curve durante l'ottimizzazione sono mostrate in Figura 9.

		n=20			**n=115**			**n=1141**		
	m	iters	time	time/iter	iters	time	time/iter	iters	time	time/iter
L-BFGS	1	76	3.27	0.0430	278	22.32	0.0803	967	1060.87	1.0971
	3	59	2.18	0.0370	172	13.87	0.0807	513	520.91	1.0154
	5	42	1.63	0.0389	161	13.09	0.0813	645	653.35	1.0129
	7	37	1.36	0.0369	140	11.40	0.0814	357	367.63	1.0298
	10	38	1.38	0.0362	121	9.37	0.0775	447	477.00	1.0671
	15	33	1.24	0.0376	108	7.98	0.0739	479	511.05	1.0669
	20	32	1.14	0.0357	96	7.14	0.0744	393	380.86	0.9691
	25	31	**1.11**	0.0357	91	6.89	0.0758	268	**254.92**	0.9512
	30	31	**1.11**	0.0359	87	**6.25**	0.0719	271	256.85	0.9478
BFGS	//	32	1.25	0.0392	132	10.08	0.0764	291	420.62	1.4454

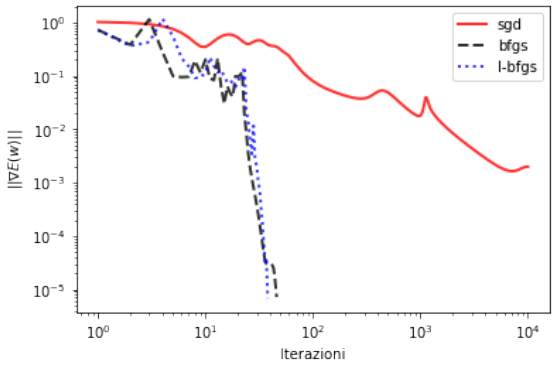
Esperimento 2



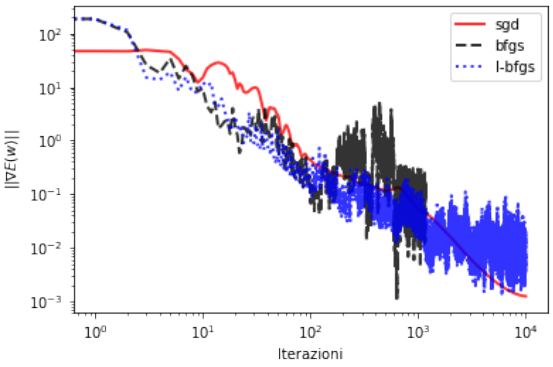
(a) MONK-1



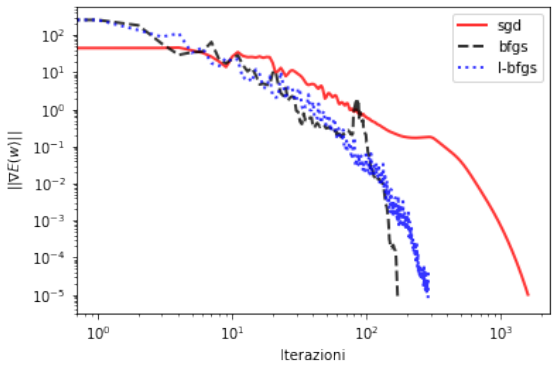
(b) MONK-2



(c) MONK-3



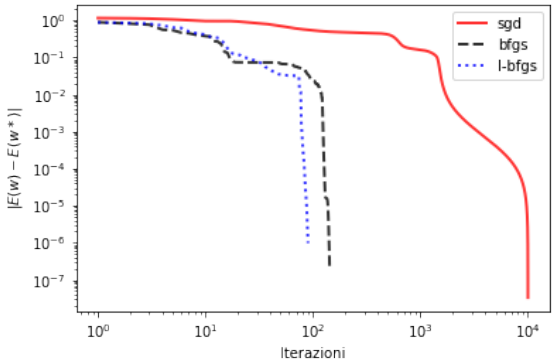
(d) ML-CUP $\boldsymbol{\lambda = 0}$



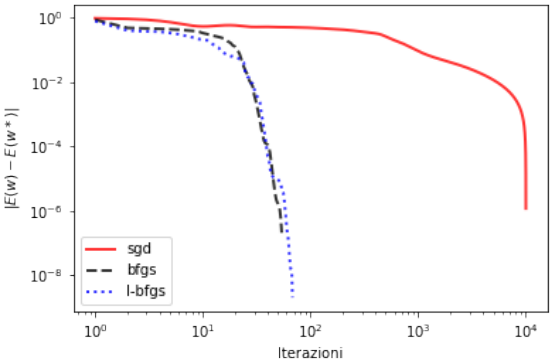
(e) ML-CUP $\boldsymbol{\lambda = 0.05}$

Figura 4: Grafici della variazione della norma del gradiente. Nei dataset MONK i metodi Quasi-newton sono conversi con un numero di iterazioni molto minore (di 1 o 2 ordini di grandezza) rispetto alla Discesa del Gradiente. Notare l'andamento caotico di BFGS ed L-BFGS nel dataset ML-CUP con $\boldsymbol{\lambda = 0.0}$, dovuto probabilmente al grave malcondizionamento della matrice H, (vedi fig. 6d), mentre con $\boldsymbol{\lambda = 0.05}$ riappare la convergenza veloce dei metodi Quasi-newton rispetto alla Discesa del Gradiente. La regolarizzazione ha velocizzato la convergenza anche della Discesa del Gradiente.

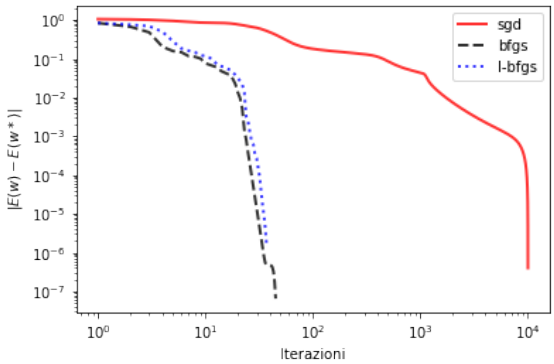
Esperimento 2



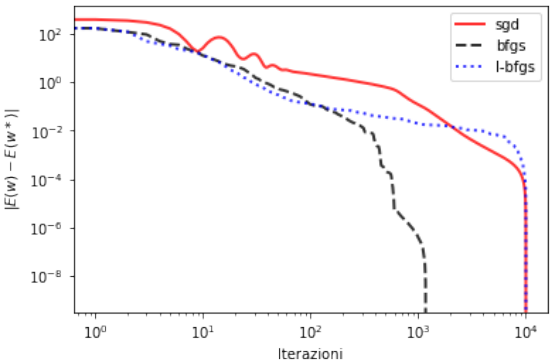
(a) MONK-1



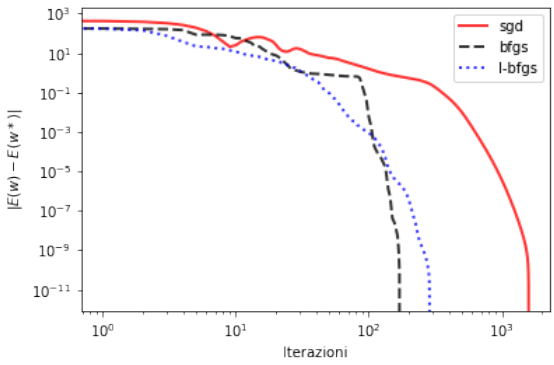
(b) MONK-2



(c) MONK-3



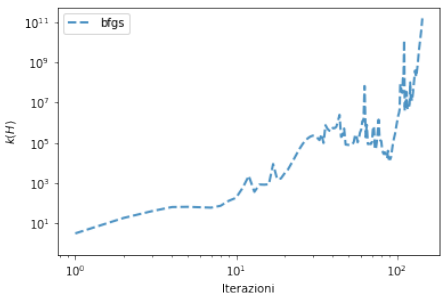
(d) ML-CUP $\boldsymbol{\lambda = 0}$



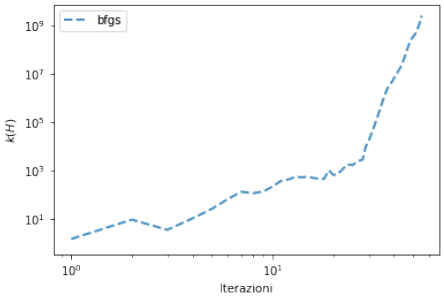
(e) ML-CUP $\boldsymbol{\lambda = 0.05}$

Figura 5: Grafici dell'andamento dell'errore relativo. Gli algoritmi Quasi-newton convergono molto più velocemente della discesa del gradiente nei MONK. Come in Figura 4 si può notare l'effetto positivo della regolarizzazione.

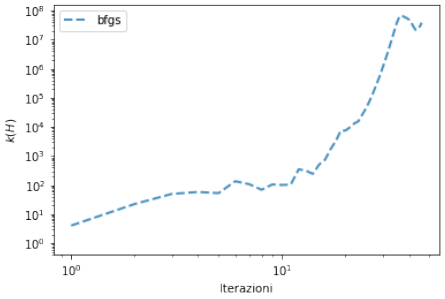
Esperimento 2



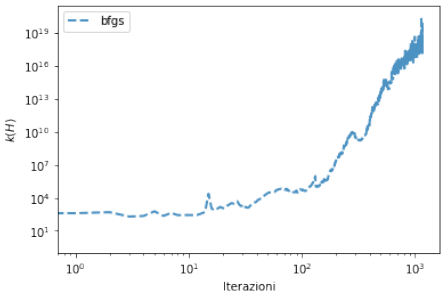
(a) MONK-1



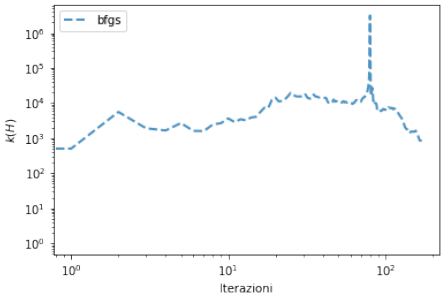
(b) MONK-2



(c) MONK-3



(d) ML-CUP $\boldsymbol{\lambda = 0}$



(e) ML-CUP $\boldsymbol{\lambda = 0.05}$

Figura 6: Numero di condizionamento della matrice $H_{BFGS} = B^{-1}$ vs numero di iterazioni. Il condizionamento è alto ma contenuto nei MONK, mentre è gravemente mal-condizionato per ML-CUP con $\boldsymbol{\lambda = 0}$, 6d, (si vedano gli autovalori della H_{BFGS} in Figura 7). Invece con $\boldsymbol{\lambda = 0.05}$ il problema è ben-condizionato, con 10^6 massimo picco del numero di condizionamento, 6e.

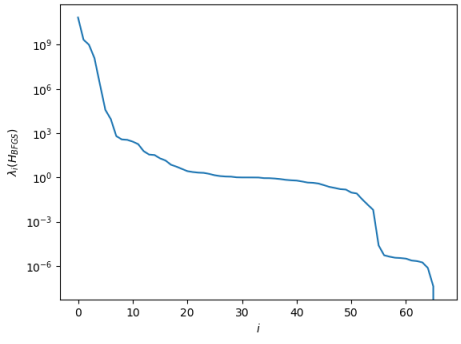
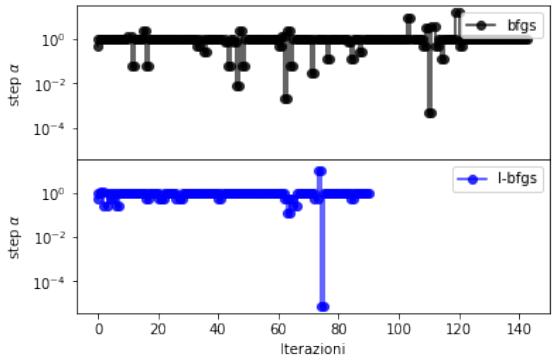
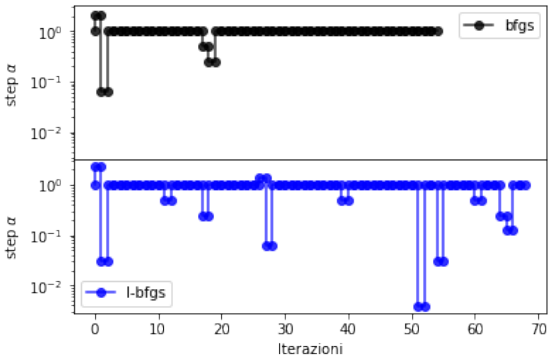


Figura 7: Distribuzione log-lin degli autovalori della matrice H_{BFGS}, ordinati decrescentemente, per algoritmo BFGS nel dataset ML-CUP con $\boldsymbol{\lambda = 0}$. La distribuzione è larga ed è presente un autovalore negativo. Pertanto l'algoritmo è converso in un punto di sella.

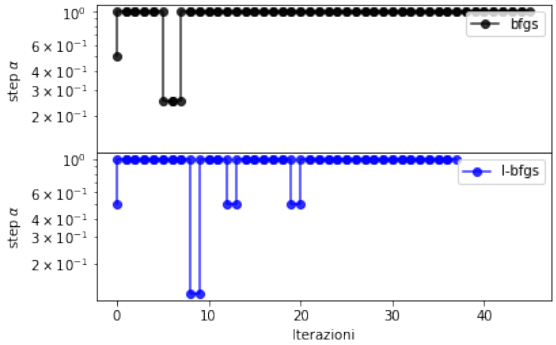
Esperimento 2



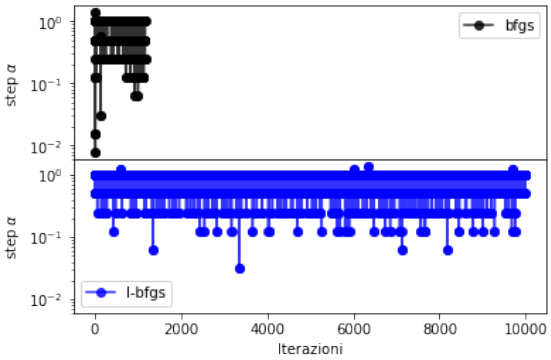
(a) monk 1



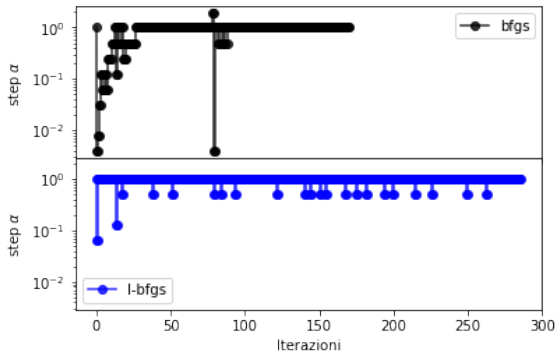
(b) monk 2



(c) monk 3



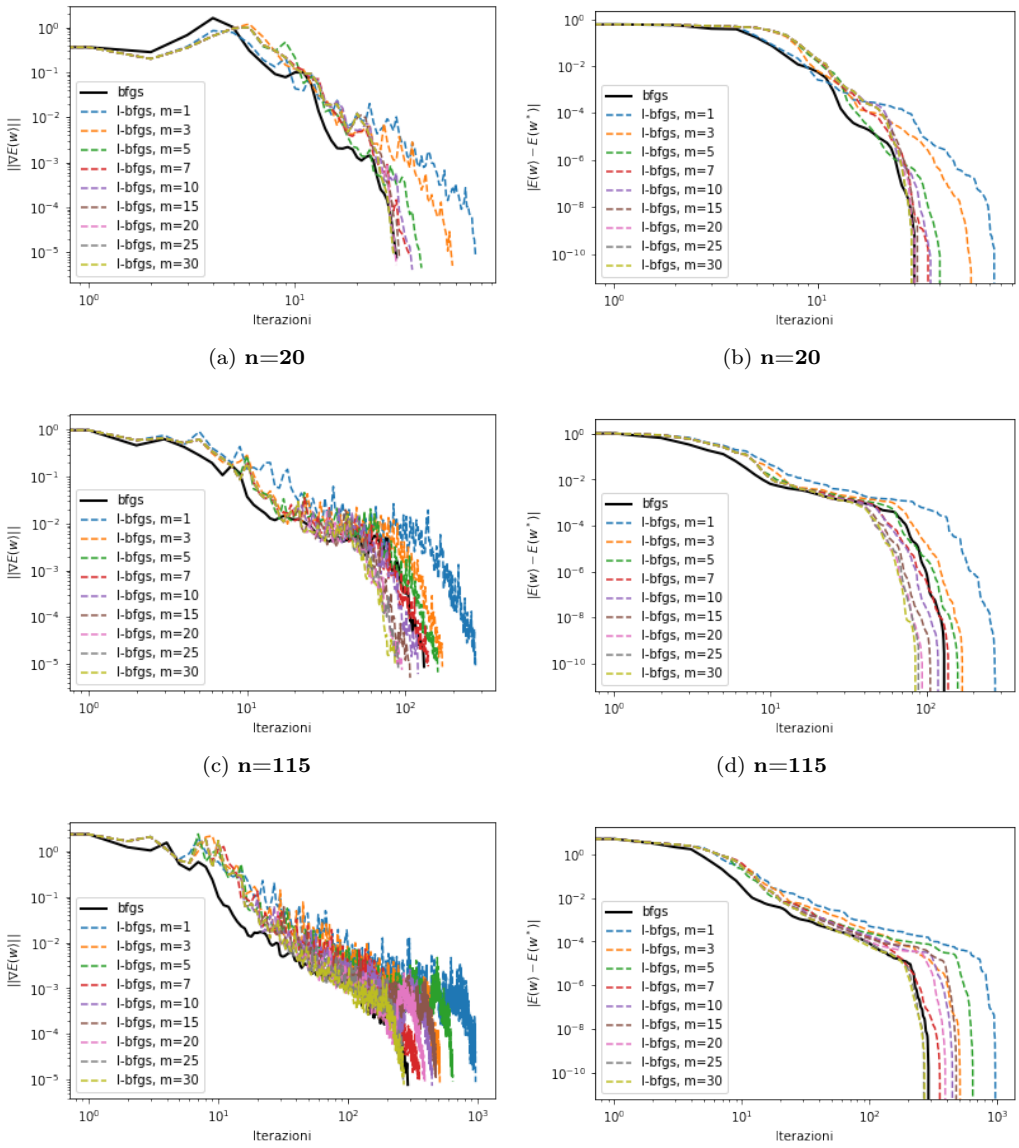
(d) ML-CUP $\boldsymbol{\lambda = 0}$



(e) ML-CUP $\boldsymbol{\lambda = 0.05}$

Figura 8: Passo di ottimizzazione α scelto dalla line search ad ogni iterazione. La line-search sceglie $\alpha = 1$ nelle fasi finali dell'allenamento fino al raggiungimento del minimo locale, tranne che nel caso di ML-CUP con $\boldsymbol{\lambda = 0}$.

Esperimento 3



(a) **n=20** (b) **n=20**

(c) **n=115** (d) **n=115**

(e) **n=1141** (f) **n=1141**

Figura 9: Confronto BFGS e L-BFGS al variare del parametro m. Per $n = 20$ BFGS converge in meno iterazioni di L-BFGS per tutti i valori di m tranne per $m \in \{25, 30\}$. Per $n = 115$ BFGS converge in meno iterazioni solo per $m \in \{1, 3, 5, 7\}$. In 9b e 9d BFGS tende a convergere in più iterazioni di L-BFGS al crescere di n e al decrescere di m. Una possibile motivazione è che, all'aumentare di m, la matrice H_{LBFGS} tenda ad approssimare la vera matrice Hessiana meglio di H_{BFGS}. Tuttavia questo andamento viene interrotto in quanto per $n = 1141$ sembrano valere le stesse considerazioni fatte per $n = 20$. Quindi si potrebbe concludere che ci siano determinati compremessi tra m ed n tali da far convergere L-BFGS più velocemente di BFGS in termini di iterazioni.

7 Risultati Machine Learning

L'obiettivo dei seguenti esperimenti è ottenere buone capacità di generalizzazione, usando le architetture di Tabella 4. I risultati sono riassunti in Tabella 7, Tabella 8 e Tabella 9 [6]. Le curve di apprendimento sono mostrate in Figura 10 e Figura 11. Una versione alternativa delle stesse curve è mostrata in Appendice D.

SGD	MONK-1	MONK-2	MONK-3	ML-CUP
α	0.01	0.01	0.01	0.01
μ	0.9	0.9	0.9	0.9
λ	0.01	0.01	0.05	0.05
batch_size	32	32	32	32
ϵ	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Loss (TR)	0.0681	0.0615	0.2821	5.3456
Loss (TS)	0.1017	0.0722	0.2282	3.7970
Accuracy (TR)	1	1	0.9338	N/A
Accuracy (TS)	1	1	0.9722	N/A
Iterazioni	1000	1000	1000	1000
Tempo (s)	38	43	41	158

Tabella 7: Iperparametri e risultati SGD. L'algoritmo non converge ad un minimo locale (max *iterazioni=1000*) ma ottiene, in un tempo ragionevole, buoni risultati su tutti i MONK, generalizzando bene sia su MONK-3 che su ML-CUP.

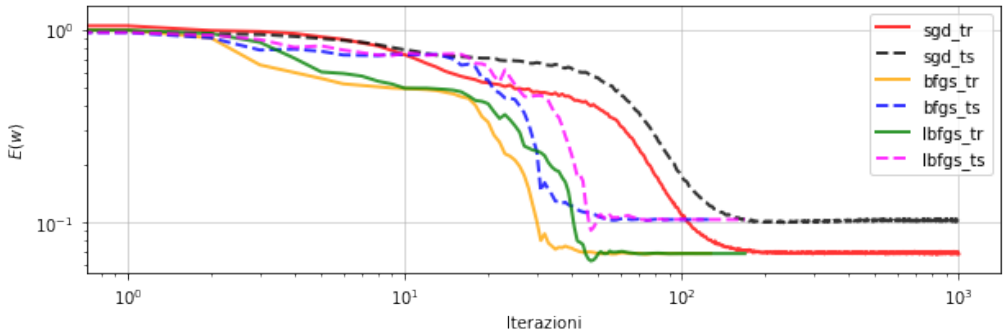
BFGS	MONK-1	MONK-2	MONK-3	ML-CUP
c_1	10^{-4}	10^{-4}	10^{-4}	10^{-4}
c_2	0.9	0.9	0.9	0.9
θ	0.9	0.9	0.9	0.9
λ	0.01	0.01	0.05	0.05
ϵ	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Loss (TR)	0.0684	0.0615	0.2785	2.9681
Loss (TS)	0.1028	0.0714	0.2217	3.0770
Accuracy (TR)	1	1	0.9344	N/A
Accuracy (TS)	1	1	0.9722	N/A
Iterazioni	128	98	95	79
Tempo (s)	9	7	6	31

Tabella 8: Iperparametri e risultati BFGS. L'algoritmo converge molto velocemente (relativamente a SGD) ad un minimo locale su tutti i dataset, mostrando buone capacità di generalizzazione sul MONK-3, ma non altrettanto buone sul ML-CUP, in quanto l'errore sul test set è leggermente maggiore dell'errore sul training set.

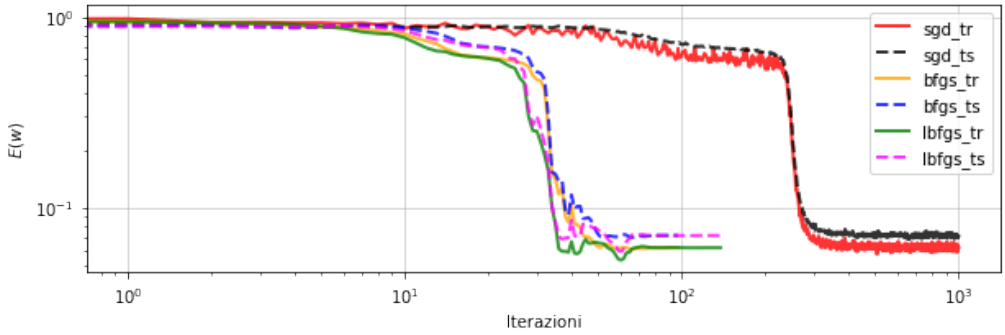
L-BFGS	MONK-1	MONK-2	MONK-3	ML-CUP
c_1	10^{-4}	10^{-4}	10^{-4}	10^{-4}
c_2	0.9	0.9	0.9	0.9
θ	0.9	0.9	0.9	0.9
λ	0.01	0.01	0.05	0.05
ϵ	10^{-5}	10^{-5}	10^{-5}	10^{-5}
m	5	5	5	5
Loss (TR)	0.0684	0.0615	0.2785	2.9681
Loss (TS)	0.1028	0.0714	0.2217	3.0770
Accuracy (TR)	1	1	0.9344	N/A
Accuracy (TS)	1	1	0.9722	N/A
Iterazioni	170	139	126	208
Tempo (s)	10	9	8	75

Tabella 9: Iperparametri e risultati L-BFGS. Valgono le stesse considerazioni fatte per BFGS in Tabella 8, in quanto i due algoritmi ottengono gli stessi identici risultati. Non è stato verificato che gli algoritmi siano conversi anche alla stessa soluzione. L'unico svantaggio è che L-BFGS impiega circa il doppio del tempo di BFGS sul ML-CUP.

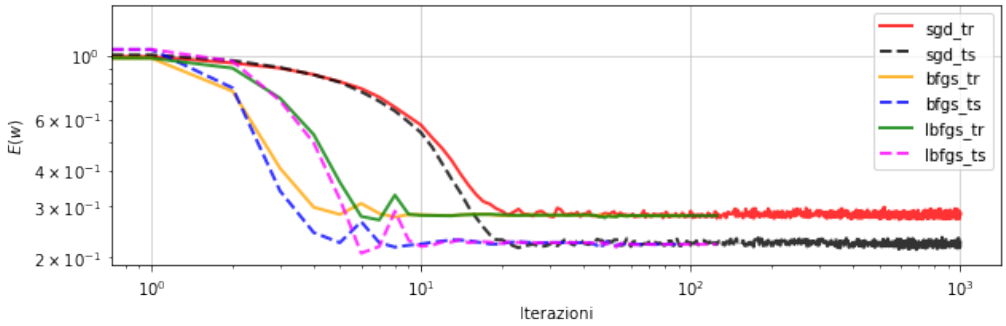
[6] I valori di Loss riportati fanno riferimento solo all'*errore sui dati*.



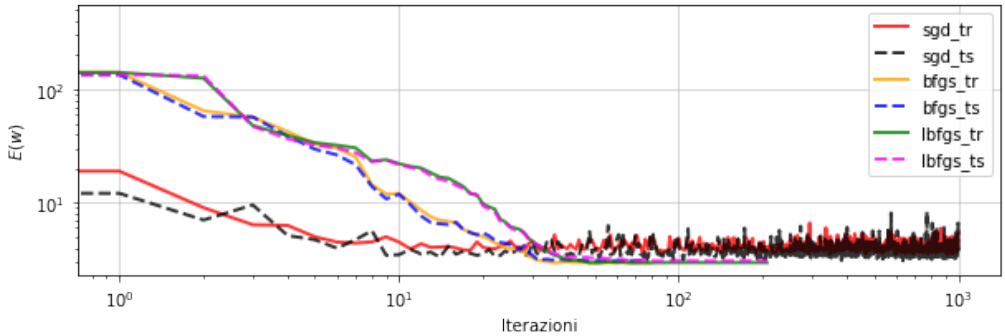
(a) **MONK-1**



(b) **MONK-2**

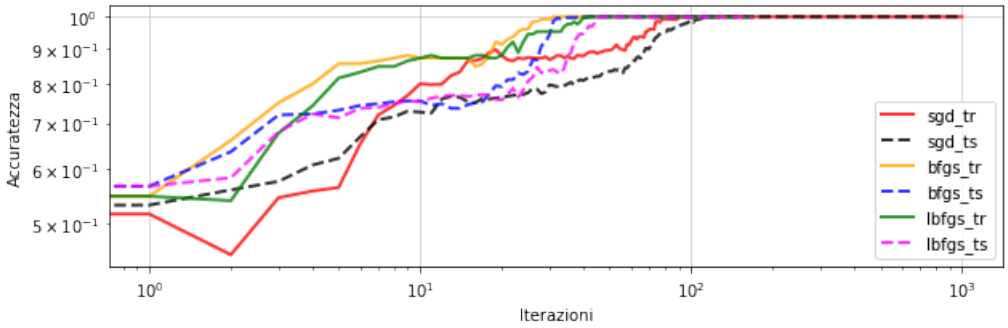


(c) **MONK-3**

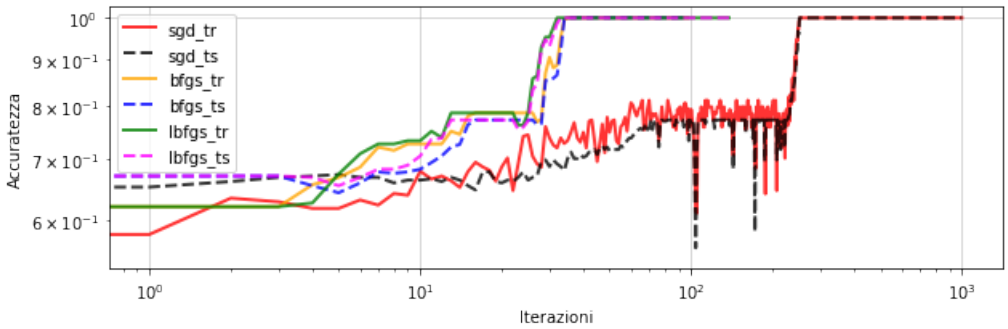


(d) **ML-CUP**

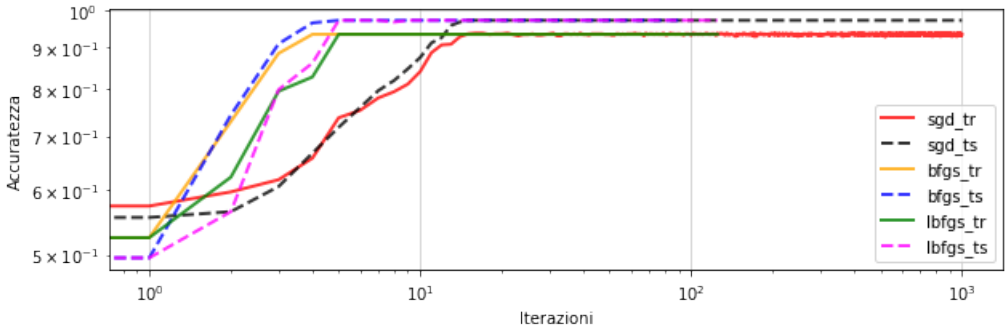
Figura 10: Curve di apprendimento dell'errore sul training e test set. Dalle curve non si evince la presenza di *overfitting* in quanto generalmente l'errore sul test set non tende a crescere mentre l'errore sul training set diminuisce. Le curve di SGD sono leggeremente più a zig-zag delle altre, probabilmente dovuto ad un α appena troppo elevato. Notiamo nel ML-CUP come le curve dei metodi Quasi-newton siano al di sotto delle curve del SGD.



(a) **MONK-1**



(b) **MONK-2**



(c) **MONK-3**

Figura 11: Curve di apprendimento dell'accuratezza sul training e test set. In generale valgono le stesse considerazioni fatte in Figura 10. Vale la pena notare il temporaneo picco negativo di prestazioni del SGD sul MONK-2 verso l'iterazione 10^2, non visibilmente presente nel grafico dell'errore, che ci suggerisce che tra errore e accuratezza non vi è una vera e propria correlazione.

8 Conclusioni

In questa relazione si sono confrontati gli algorimi Discesa del Gradiente, BFGS ed L-BFGS. Dal punto di vista dell'ottimizzazione gli algoritmi Quasi-newton surclassano l'algoritmo di discesa del gradiente in termini di iterazioni e tempo di esecuzione, mentre sono comparabili dal punto di vista dell'errore raggiunto. Tuttavia se il problema è malcondizionato le prestazioni degli algoritmi Quasi-newton degradano molto, fino a compromettere del tutto la convergenza. Nei problemi malcondizionati la Discesa del Gradiente diventa più performante dal punto di vista delle iterazioni. È stato discusso e verificato empiricamente che aggiungere un po' di regolarizzazione migliora il condizionamento del problema a beneficio degli algoritmi Quasi-newton. Si è visto che la line search Armijo-Wolfe si comporta meglio della backtracking in generale, anche se ci sono determinate situazioni, che meriterebbero uno studio più approfondito, in cui la backtracking risulta più efficiente dal punto di vista delle iterazioni totali dell'algoritmo. Il confronto di BFGS ed L-BFGS in funzione di n ed m suggerisce che non nessun algoritmo è dominante sull'altro, e che valori di m maggiori migliorano la convergenza in termini di iterazioni. Infine dal punto di vista del Machine Learning, gli algoritmi hanno ottenuto prestazioni identiche o simili nei task di classificazione binaria, ma nel task di regressione gli algoritmi Quasi-newton hanno manifestato migliori capacità di generalizzazione della Discesa Stocastica del Gradiente. Tuttavia non è stata fatta una ricerca approfondita degli iperparametri. Lavori futuri potrebbero esplorare maggiormente l'effetto della regolarizzazione su problemi malcondizionati.

Riferimenti bibliografici

[1] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[4] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent converges to minimizers. *arXiv preprint arXiv:1602.04915*, 2016.

[5] M. Lichman. UCI machine learning repository, 2013.

[6] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[7] Tom M Mitchell et al. Machine learning. wcb, 1997.

[8] JORGE. NOCEDAL. *NUMERICAL OPTIMIZATION*. Springer, 2006.

[9] Sebastian B Thrun, Jerzy W Bala, Eric Bloedorn, Ivan Bratko, Bojan Cestnik, John Cheng, Kenneth A De Jong, Saso Dzeroski, Douglas H Fisher, Scott E Fahlman, et al. The monk's problems: A performance comparison of different learning algorithms. Technical report, 1991.

[10] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[11] Stephen Wright and Jorge Nocedal. Numerical optimization. *Springer Science*, 35(67-68):7, 1999.

A Background

Definizione 2 (Funzione Lipschitz-continua) *Sia $A \subseteq \mathbb{R}$ e sia $f : A \to \mathbb{R}$ una funzione. Si dice che f è Lipschitziana in A se esiste una costante L tale che*

$$|f(y) - f(x)| \leq L|x - y| \ \forall x, y \in A$$

Teorema 10 (Condizione necessaria e sufficiente per la Lipschitzianità) *Sia I un intervallo di $\mathbb{R}$ e sia $f : I \to \mathbb{R}$ una funzione derivabile. Allora f è Lipschitziana in I se e solo se $f'(x)$ è limitata in I.*

Definizione 3 (Funzione liscia) *Una funzione f si dice liscia se $f \in C^\infty$, ovvero è differenziabile infinite volte avendo tutte le derivate continue.*

Definizione 4 (Insieme convesso) *Un insieme $S \subseteq \mathbb{R}^n$ è convesso se $\forall x, y \in S$ e $\forall \alpha \in [0, 1]$*

$$\alpha x + (1 - \alpha)y \in S$$

Definizione 5 (Funzione convessa) *Una funzione f si dice convessa se il suo dominio S è un insieme convesso e se $\forall x, y \in S$ e $\forall \alpha \in [0, 1]$*

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

B Ottimalità

Definizione 6 (Punto stazionario)

1. *Un punto $x^* \in \mathbb{R}^n$ è un **punto stazionario** di f se $\nabla f(x^*) = 0$.*
2. *Un punto stazionario x^* è un punto di **minimo globale** se $f(x^*) \leq f(x)\ \forall x \in \mathbb{R}^n$.*
3. *Un punto stazionario x^* è un punto di **minimo locale (debole)** se esiste un intorno N di x^* tale che $f(x^*) \leq f(x)\ \forall x \in N$*
4. *Un punto stazionario x^* è un punto di **minimo locale stretto** se esiste un intorno N di x^* tale che $f(x^*) < f(x)\ \forall x \in N$ con $x \neq x^*$.*
5. *Un punto stazionario x^* è un punto di **minimo locale isolato** se esiste un intorno N di x^* tale che x^* è l'unico minimizzatore locale in N.*
6. *Un punto stazionario x^* è un punto di **sella** se, per ogni intorno N di x^*, esistono $x, y, \in N$ tali che $f(x) \leq f(x^*) \leq f(y)$.*
7. *Un punto stazionario x^* di f è un punto di **sella stretto** se $\lambda_{min}\nabla^2 f(x^*) < 0$, i.e. se $\nabla^2 f$ ha almeno un autovalore strettamente negativo.*

Di seguito vengono riportate le condizioni necessarie e sufficienti di ottimalità [8, cap. 2].

Teorema 11 (Condizione necessaria del primo ordine) *Se x^* è un minimo locale ed f è continuamente differenziabile in un intorno aperto di x^*, allora $\nabla f(x^*) = 0$.*

Teorema 12 (Condizione necessaria del secondo ordine) *Se x^* è un minimo locale e $\nabla^2 f$ è continuo in un intorno aperto di x^*, allora $\nabla f(x^*) = 0$ e $\nabla^2 f(x^*) \succcurlyeq 0$.*

Teorema 13 (Condizione sufficiente del secondo ordine) *Se $\nabla f(x^*) = 0$ e $\nabla^2 f$ è continuo in un intorno aperto di x^* e $\nabla^2 f(x^*) \succ 0$, allora x^* è un minimo locale stretto.*

C Dimostrazione $\cos\theta \geq 1/M > 0$

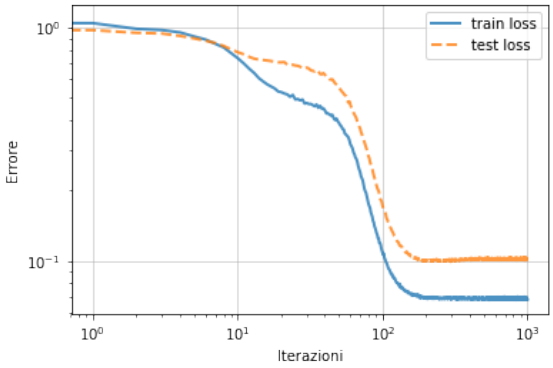
Dimostrazione 1

$$
\begin{aligned}
\cos\theta_k &= \frac{-\nabla f(x_k)^\intercal p_k}{||\nabla f(x_k)|| \cdot ||p_k||} = && \{\textit{def. } \cos\theta_k \text{ (26)}\} \\
&= \frac{-\nabla f(x_k)^\intercal(-B_k^{-1}\nabla f(x_k))}{||\nabla f(x_k)|| \cdot || - B_k^{-1}\nabla f(x_k)||} = && \{\textit{def. } p_k, \text{ (11)}\} \\
&= \frac{\nabla f(x_k)^\intercal B_k^{-1}\nabla f(x_k)}{||\nabla f(x_k)|| \cdot ||B_k^{-1}\nabla f(x_k)||} \geq \\
&\geq \frac{\nabla f(x_k)^\intercal B_k^{-1}\nabla f(x_k)}{||\nabla f(x_k)|| \cdot ||B_k^{-1}|| \cdot ||\nabla f(x_k)||} = && \{||AB|| \leq ||A|| \cdot ||B|| \ \} \\
&= v^\intercal B_k^{-1} v \cdot \frac{1}{||B_k^{-1}||} \geq && \{\ \nabla f(x_k)/||\nabla f(x_k)|| := v \ \} \\
&\geq \frac{1}{||B_k^{-1}||} \cdot \lambda_{min}(B_k^{-1}) = && \{\lambda_{min} \leq v^\intercal A v \leq \lambda_{max} \ \} \\
&= \frac{1}{||B_k^{-1}||} \cdot \frac{1}{\lambda_{max}(B_k)} = \\
&= \frac{1}{||B_k^{-1}||} \cdot \frac{1}{||B_k||} = && \{\ ||A|| = \lambda_{max} \textit{ per } A \textit{ simmetrica e definita positiva } \} \\
&\geq 1/M > 0 && \{\ \textit{def. } k(B_k), \text{ (31)} \ \}
\end{aligned}
$$

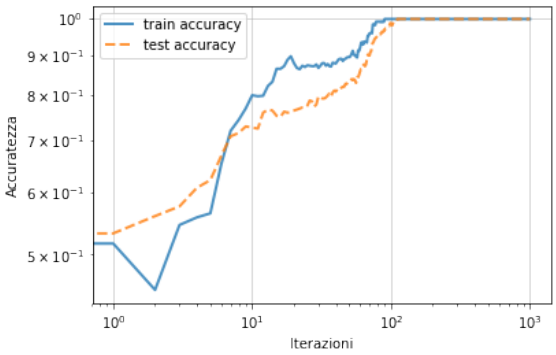
□

D Curve Apprendimento Machine Learning

SGD

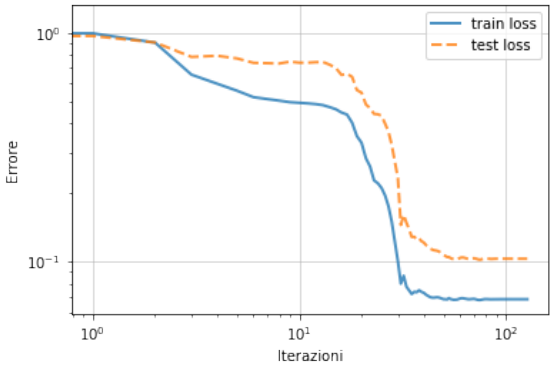


(a) loss

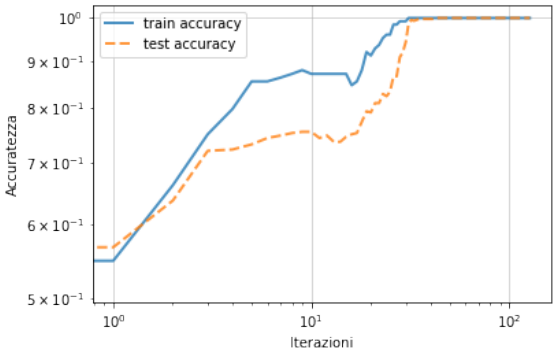


(b) accuracy

BFGS

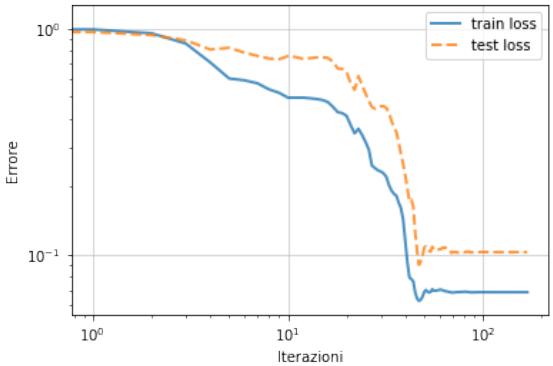


(c) loss

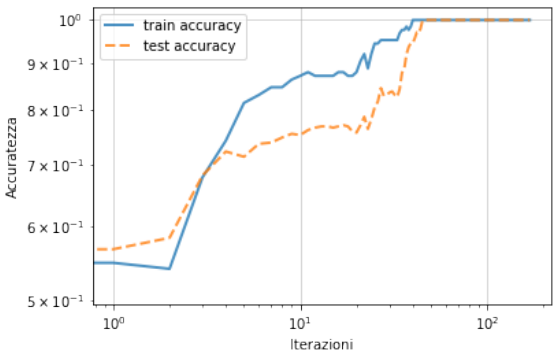


(d) accuracy

L-BFGS



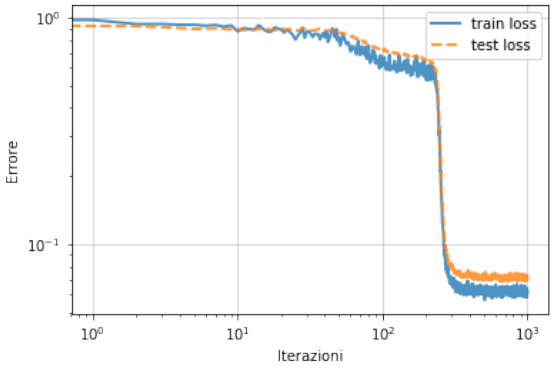
(e) loss



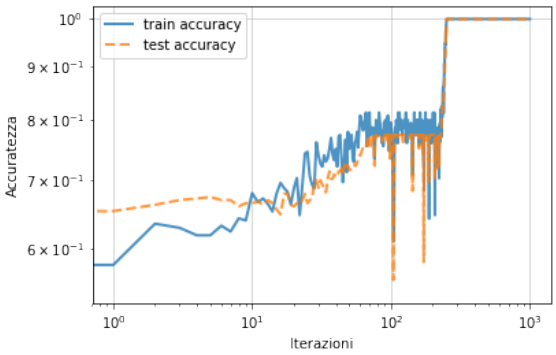
(f) accuracy

Figura 12: Curve di apprendimento sul dataset MONK 1.

SGD

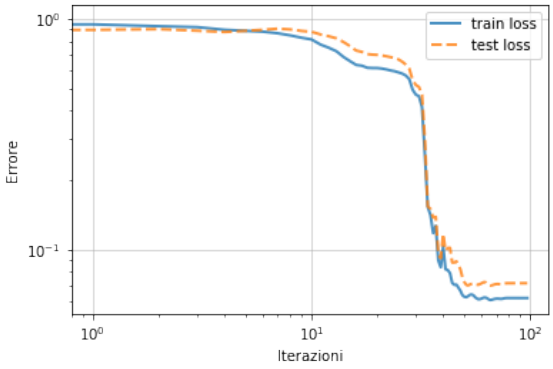


(a) loss

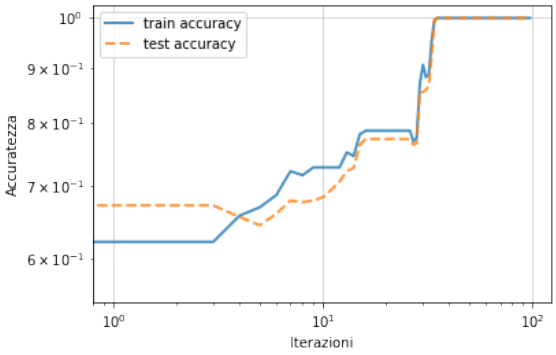


(b) accuracy

BFGS

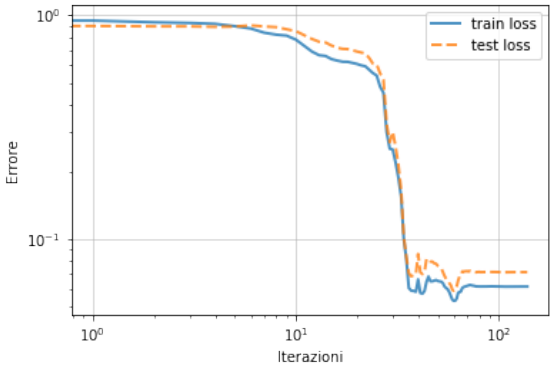


(c) loss

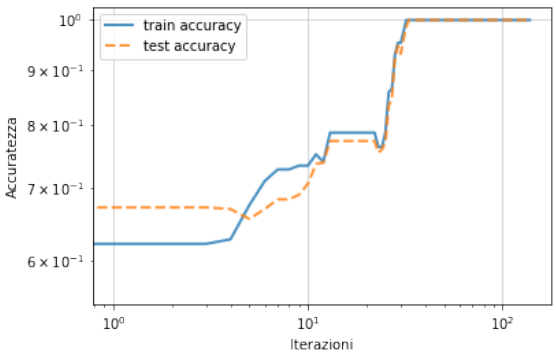


(d) accuracy

L-BFGS



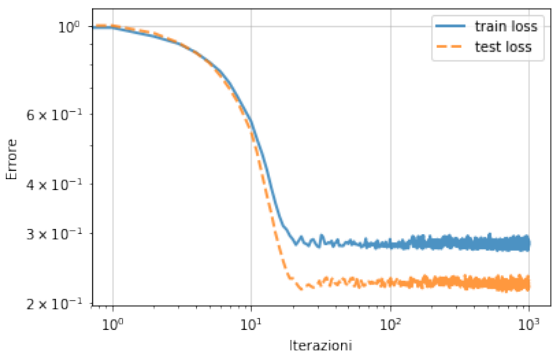
(e) loss



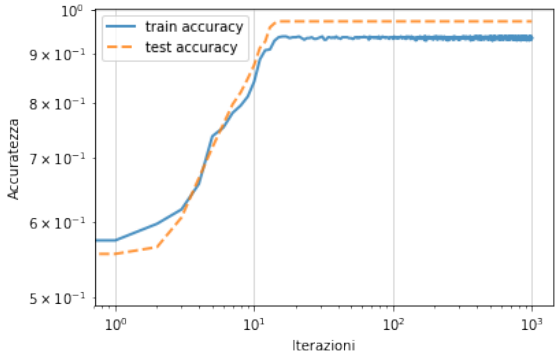
(f) accuracy

Figura 13: Curve di apprendimento sul dataset MONK 2.

SGD

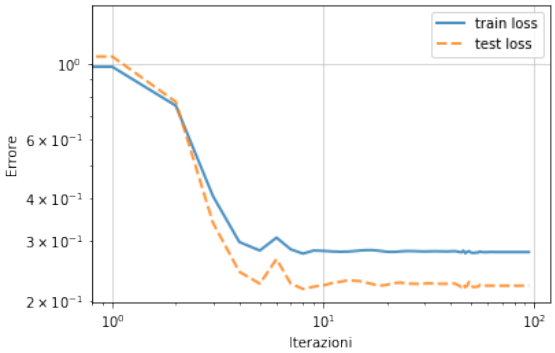


(a) loss

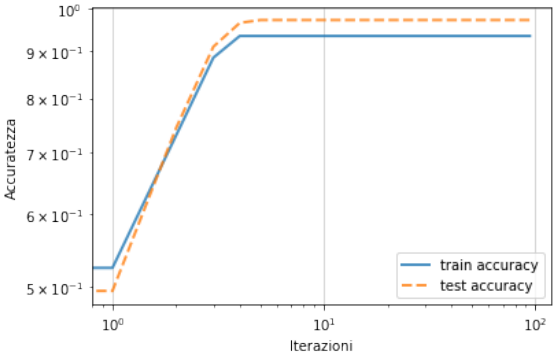


(b) accuracy

BFGS

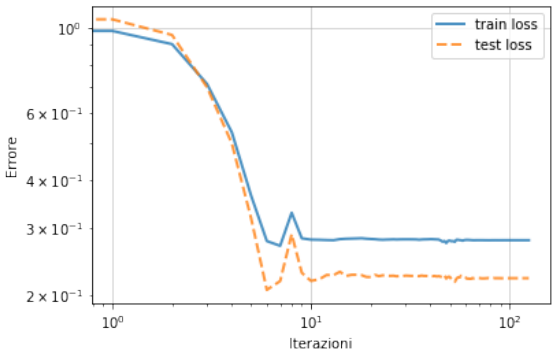


(c) loss

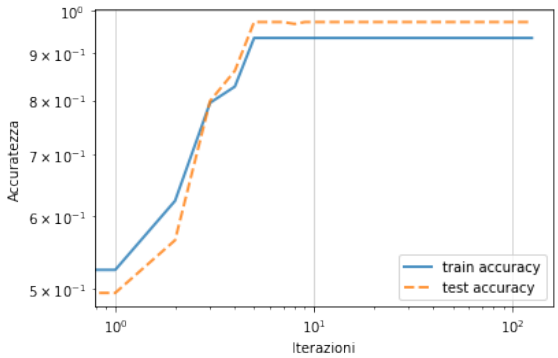


(d) accuracy

L-BFGS

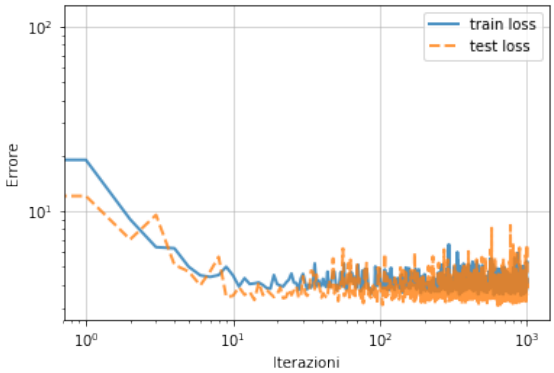


(e) loss

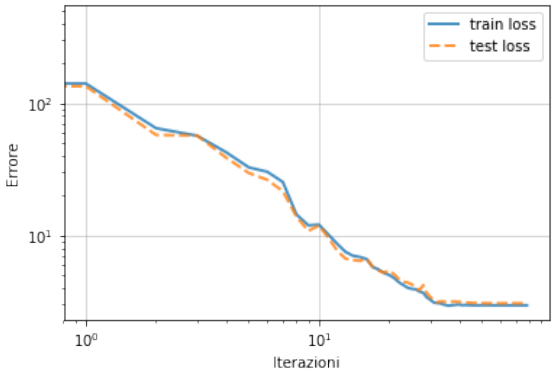


(f) accuracy

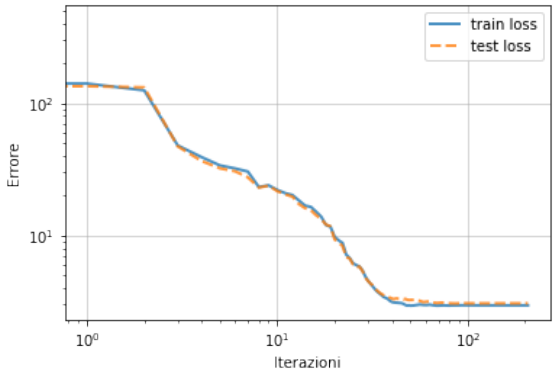
Figura 14: Curve di apprendimento sul dataset MONK 3.



(a) SGD



(b) BFGS



(c) L-BFGS

Figura 15: Curve di apprendimento sul dataset ML-Cup.