Extreme Learning Machine

Project n.10

Optimization for Data Science
University of Pisa
A.Y. 2023/2024

Author: *Bombino Biancamaria 561745, Mastrorilli Alessandro 657939*

Professor:	*Antonio Frangioni*
Master's degree:	Data Science & Business Informatics
Department:	Computer Science
Date:	August 17, 2024

Contents

1. Extreme Learning Machine

L'**Extreme Learning Machine** è un algoritmo di apprendimento con un *single hidden layer feedforward neural network* che permette di superare i vari problemi che esistono nei vari algoritmi di *Back Propagation.* [2] Esso è un metodo di apprendimento veloce rispetto ai metodi tradizionali, in quanto i pesi di input e gli hidden biases sono scelti in modo randomico e i pesi di output sono determinati analiticamente usando la *Moore-Penrose pseudo-inverse.* Questo algoritmo non ha parametri da regolare e ciò permette di evitare situazioni in cui, durante la regolazione del tasso di apprendimento, si può visualizzare una divergenza o convergenza molto lenta, oppure un blocco in minimi locali sub-ottimali.
In questo progetto ci poniamo l'obiettivo di addestrare una rete neurale ELM, considerando il nostro obiettivo di ottimizzazione e utilizzando i seguenti approcci di ottimizzazione:

- **Metodo di Quasi-Newton: BFGS**
- **Soluzione in forma chiusa con equazioni normali e fattorizzazione di Cholesky**

La differenza principale tra le due tecniche è il metodo considerato per ottimizzare i pesi del modello. Il primo metodo utilizza un approccio iterativo basato sulla discesa del gradiente, mentre il secondo adopera una soluzione chiusa basata su equazioni normali e fattorizzazione di Cholesky. Entrambi mirano a trovare i pesi ottimali del modello che minimizzano l'errore tra le previsioni del modello e i target desiderati, ovvero la funzione obiettivo.

1.1 Single hidden layer feedforward networks with random hidden nodes

Sia (X, T) un insieme di *N* campioni, dove $X = \{x_i[n]\} \in \mathbb{R}^{N \times I}$ rappresenta la matrice contenente gli input relativi all'*i-esimo* neurone al tempo *n*; e $T = \{t_l[n]\} \in \mathbb{R}^{N \times L}$ rappresenta la matrice degli output desiderati associati all'*l-esimo* neurone di output al tempo *n*. Questi campioni sono usati per addestrare un SLFN, con I neuroni di input, K neuroni nascosti, L neuroni di output e una funzione di attivazione $g(\cdot)$. In ELM i pesi di input $\{w_{ik}\}$ e i bias nascosti $\{b_k\}$ sono generati randomicamente.
$\{w_{ik}\}$ è il peso che connette l'*i-esimo* neurone di input al *k-esimo* neurone nascosto, ed esso incorpora $\{b_k\}$ bias del *k-esimo* neurone nascosto tale che $\{w_{0k} = b_k\}$ e $\{x_0[n] = 1\}$.
La matrice di output dell'hidden-layer $H = \{h_k[n]\} \in \mathbb{R}^{N \times K}$ può essere ottenuta con:

$$h_k[n] = g\left(\sum_{i=0}^{I} x_i[n] \cdot w_{ik}\right)$$

Sia $W_2 = \{W_{2kl}\} \in \mathbb{R}^{K \times L}$ la matrice dei pesi di output, dove W_{2kl} denota il peso di connessione tra il *k-esimo* neurone nascosto e l'*l-esimo* neurone di output; e sia $Y = \{y_l[n]\} \in \mathbb{R}^{N \times L}$ la matrice

dei dati di output della rete, in cui $y_l[n]$ è l'output dell'*l-esimo* neurone di output all'*n-esimo* istante di tempo. La seguente equazione può essere ottenuta per i neuroni di output:

$$y_l[n] = \sum_{k=1}^{K} h_k[n] \cdot \beta_{kl}$$

Oppure $Y = H \cdot W_2$.
Data la matrice di output dell'hidden layer H e la matrice target T, per minimizzare $\|Y - T\|_2^2$, i pesi di output possono essere calcolati utilizzando la soluzione del sistema lineare:

$$W_2' = H^\dagger \cdot T$$

dove $H^\dagger$ è la *Moore–Penrose generalized inverse* della matrice H.

1.2 Algoritmo ELM

L'algoritmo di apprendimento ELM per Single Layer Feedforward Networks (SLFN) comprende i seguenti passaggi [1]:
Dato un training set (X, T) con $X = \{\mathbf{x}_i[n]\} \in \mathbb{R}^{N \times I}$ che denota i dati di input e $T = \{\mathbf{t}_l[n]\} \in \mathbb{R}^{N \times L}$ i target, con funzione di attivazione $g(\cdot)$, e con *K* neuroni nascosti:

1. Assegnare dei pesi di input arbitrari w_k e bias $b_k = w_{0k}$ con $k = 1, \ldots, K$.
2. Calcolare gli hidden layer della matrice di output H.
3. Calcolare i pesi di output:
$$W_2' = H^\dagger \cdot T$$
dove H, W_2 e T sono definiti come in precedenza.

2. Funzione obiettivo

Nel nostro caso specifico, la funzione obiettivo $\boldsymbol{f(\mathbf{W_2})}$ da minimizzare è costituita dalla somma dell'errore quadratico medio *(MSE)* tra le previsioni del modello e i valori target, e un termine di regolarizzazione *L2* sui pesi W_2. Quindi essa può essere rappresentata come:

$$f(\mathbf{W}_2) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x}_i) - t_i\right)^2 + \lambda \|\mathbf{W}_2\|^2$$

dove:

- $\boldsymbol{W}_2$ rappresenta la matrice dei pesi di output,
- $\boldsymbol{W}_1$ rappresenta i pesi di input,
- $\boldsymbol{x}_i$ è l'*i-esimo* campione di input,
- $\boldsymbol{\sigma}$ è la funzione di attivazione,

- $\boldsymbol{t_i}$ è il target per l'*i-esimo* campione,
- $\boldsymbol{N}$ è il numero totale dei campioni,
- $\boldsymbol{\lambda}$ è il parametro di regolarizzazione,
- $\boldsymbol{\sigma}$ è la funzione di attivazione element-wise,
- $\|\boldsymbol{W_2}\|^{\boldsymbol{2}}$ è il termine di regolarizzazione *L2*.

2.1 Proprietà della funzione obiettivo

- **Convessità:** La funzione obiettivo è caratterizzata dalla norma euclidea che gode della proprietà di essere sempre convessa.[1] Inoltre, questa norma contiene componenti lineari, e aggiungendo il termine di regolarizzazione *L2*, il risultato non cambia poiché anche questo è rappresentato da un termine quadratico lineare $\|\mathbf{W}_2\|^2$. In generale, essendo la funzione composta anche da termini quadratici, possiamo concludere che essa è convessa.
- **Continuità:** La funzione obiettivo è continua in $\mathbf{W}_2$ poiché il primo termine di errore quadratico è una somma di funzioni continue, ovvero una composizione di funzioni lineari e non lineari continue. Il secondo termine di regolarizzazione è anch'esso continuo. In generale, essendo la funzione la somma di questi due termini continui, anch'essa è continua.
- **Derivabilità:** Una funzione vettoriale è derivabile se esiste il gradiente. Per il calcolo del gradiente consideriamo le derivate di ciascun termine rispetto a $\mathbf{W}_2$. Entrambi i termini sono derivabili poiché il primo è una somma di funzioni quadrate e continue, quindi derivabili ovunque; mentre il secondo termine rappresenta la norma quadrata di $\mathbf{W}_2$ e per questo motivo è anch'essa continua e derivabile ovunque. Combinando entrambi i termini, otteniamo il gradiente complessivo della funzione obiettivo:

$$\nabla_{\mathbf{W}_2} f(\mathbf{W}_2) = \frac{2}{N} \sum_{i=1}^{N} \left(\mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{x}_i) - t_i\right) \sigma(\mathbf{W}_1 \mathbf{x}_i) + 2\lambda \mathbf{W}_2$$

3. Metodo di Quasi-Newton

I metodi Quasi-Newton sono algoritmi di ottimizzazione utilizzati per risolvere problemi di minimizzazione di funzioni non lineari. Questi metodi sono progettati per ottimizzare funzioni obiettivo senza richiedere la valutazione diretta della matrice Hessiana, che può essere costosa da calcolare o memorizzare, specialmente per problemi di grandi dimensioni. L'idea chiave di questi metodi è di approssimare la matrice Hessiana inversa, che rappresenta la seconda derivata della funzione obiettivo, senza doverla calcolare esplicitamente ad ogni iterazione. Infatti, utilizzano una serie di aggiornamenti iterativi basati sui gradienti della funzione obiettivo per approssimare l'inversa della matrice Hessiana.

[1] Norma

3.1 BFGS

Il metodo della classe dei Quasi-Newton scelto per risolvere il problema è il metodo **BFGS**. Questo algoritmo si basa su aggiornamenti successivi dei pesi W_2, guidati dal gradiente della funzione obiettivo. Nel nostro contesto, come detto precedentemente, la funzione obiettivo è rappresentata come una combinazione di un termine di perdita e una penalizzazione di regolarizzazione λ.
Di seguito elenchiamo i passaggi fondamentali di tale algoritmo:

1. Inizializzazione dei pesi dell'output layer $\mathbf{W}_2$ in modo casuale.
2. Inizializzazione della matrice Hessiana approssimata inversa $\mathbf{B}$:
   (a) Definizione di un parametro scalare α_1 con il valore più appropriato per il nostro problema.
   (b) Inizializzare la matrice Hessiana $\mathbf{B}$ con:
   $$\mathbf{B} = \alpha_1 \cdot \mathbf{I}$$
3. Calcolo l'output del hidden layer:
   $$\mathbf{H} = \sigma(\mathbf{W}_1\mathbf{X})$$
4. Ripetizione dei seguenti passi fino alla convergenza (in base a un valore di tolleranza impostato) o al raggiungimento del numero massimo di iterazioni:
   (a) Calcolo del gradiente della funzione di costo (il Mean Square Error tra le previsioni del modello e i target desiderati, aggiungendo un termine di regolarizzazione λ per prevenire l'overfitting) rispetto a W_2, considerando che l'output della rete neurale è $\mathbf{Y} = \mathbf{H} \cdot \mathbf{W}_2$, e l'errore tra le previsioni della rete e i target è $\mathbf{E} = \mathbf{Y} - \mathbf{T}$:
   $$\nabla J(\mathbf{W}_2) = \frac{2}{N}\mathbf{H}^T \cdot \mathbf{E} + 2\lambda\mathbf{W}_2$$
   (b) Aggiornamento della direzione di discesa utilizzando la matrice Hessiana approssimata inversa $\mathbf{B}$ e il gradiente ottenuto ∇J:
   $$\mathbf{d} = -\mathbf{B}\nabla J$$
   dove J è la funzione di costo rispetto ai pesi $\mathbf{W}_2$.
   (c) Line Search in forma chiusa:
   Essendo la nostra funzione obiettivo quadratica, la line search in forma chiusa è il metodo adatto per determinare il passo α ottimale, senza dover iterare; da utilizzare nella direzione di discesa d durante l'aggiornamento dei pesi W_2. Per giungere alla formula finale vengono seguite le seguenti fasi:
   - Espandiamo la funzione obiettivo $\boldsymbol{J(\mathbf{W_2})}$ sostituendola con la parametrizzazione della funzione:
   $$J(\mathrm{W}_2 + \alpha \cdot \mathrm{d})$$

- Deriviamo la funzione rispetto a:

$$\frac{d}{d\alpha} J(W2 + \alpha \cdot d)$$

- Calcoliamo il prodotto interno tra la derivata appena appena trovata e la direzione, ponendo il tutto pari a 0:

$$\phi'(\alpha) = \langle \nabla f(W_2 + \alpha d), d \rangle = 0$$

- Isoliamo α nell'equazione e otteniamo:

$$\alpha = -\frac{\frac{1}{N}\sum_{i=1}^{N}\left[\left\langle W_2 H_i H_i^T, d\right\rangle - \left\langle t_i, H_i^T d\right\rangle\right] + \lambda \left\langle W_2, d\right\rangle}{\frac{1}{N}\sum_{i=1}^{N}\left\langle dH_i H_i^T, d\right\rangle + \lambda \left\langle d, d\right\rangle}$$

(d) Aggiornamento dei pesi di output:

$$W_2(i+1) = W_2(i) + \alpha \cdot d \tag{1}$$

(e) Definzione del fattore di scala $\rho_i = \frac{1}{\mathbf{y_i}^T \mathbf{s_i}}$, dove:

- y_i è la differenza tra i gradienti della funzione obiettivo in due iterazioni successive.
- s_i è la differenza tra i vettori delle variabili in due iterazioni successive.

(f) Aggiornamento della matrice hessiana inversa $\mathbf{B}$:

$$B_{i+1} = B_i + \rho_i \left[\left(1 + \rho_i \mathbf{y_i}^T B_i \mathbf{y_i}\right) \mathbf{s_i} \mathbf{s_i}^T - \left(B_i \mathbf{y_i} \mathbf{s_i}^T + \mathbf{s_i} \mathbf{y_i}^T B_i\right)\right] \tag{2}$$

(g) Bisogna verificare la convergenza controllando se la norma del gradiente e la norma della differenza dei pesi di output W_2 tra due iterazioni sono inferiori alla tolleranza, o verificando il raggiungimento del numero massimo di iterazioni considerato. Se una di queste condizioni è verificata l'algoritmo si interrompe.

5. Infine, vengono restituiti i pesi del output layer W_2 ottimizzati.

3.2 Complessità computazionale

Il metodo BFGS utilizzato per trovare il minimo della funzione, come affermato in precedenza, non richiede il calcolo esplicito della matrice Hessiana. Per analizzare la sua complessità, esaminiamo ogni componente chiave presente in una singola iterazione:

- Il **calcolo del gradiente** ha una complessità di $O(n)$, poiché si devono calcolare le derivate parziali rispetto a ciascuna delle *n* variabili di input.
- Per l'**aggiornamento della direzione di discesa e dalla matrice Hessiana inversa**, la complessità è pari a $O(n^2)$ poiché comporta operazioni su matrici di dimensione *n*×*n*.

In sintesi, combinando questi fattori, per ogni iterazione del metodo BFGS, la complessità è pari a $\mathbf{O(n^2)}$; e perciò tale metodo richiede una complessità computazionale quadratica. Se il metodo BFGS richiede *k* iterazioni per convergere, la complessità totale sarà $\mathbf{O(k * n^2)}$.

3.3 Velocità di convergenza

La velocità di convergenza di un metodo di ottimizzazione si riferisce a quanto rapidamente il metodo si avvicina alla soluzione ottimale, ovvero al minimo della funzione obiettivo man mano che si procede con le iterazioni. Nella fase iniziale, ipoteticamente, durante le prime iterazioni, il metodo può convergere lentamente, poiché la matrice hessiana approssimata H non è ancora ben calibrata.
Infatti, quest'ultima viene approssimata inizialmente come la matrice identità I e quindi l'algoritmo assume che la funzione obiettivo sia uniforme. Ciò può essere lontano dalla realtà se la funzione ha una curvatura variabile, ovvero la matrice hessiana reale ha autovalori molto diversi tra loro. Di conseguenza, questa assunzione potrebbe portare a passi sub-ottimali.
Fortunatamente, il metodo BFGS aggiorna iterativamente la matrice Hessiana approssimata, migliorandone la precisione ad ogni iterazione. Questo processo consente all'algoritmo di adattarsi gradualmente alla reale curvatura della nostra funzione obiettivo. Dopo diverse iterazioni, l'algoritmo potrà quindi beneficiare della matrice Hessiana approssimata maggiormente accurata e i pesi W_2 si aggiorneranno in modo più efficiente, avvicinandosi al minimo della funzione obiettivo in modo più rapido.
Questo comportamento è caratteristico della **convergenza superlineare**, in cui la velocità di convergenza aumenta man mano che ci si avvicina alla soluzione. Per verificare l'effettiva convergenza superlineare del nostro metodo, dimostriamo prima la convergenza globale e successivamente la condizione aggiuntiva per la superlineare.
Per garantire una **convergenza globale** del metodo BFGS è necessario che la funzione obiettivo segua le seguenti assunzioni: [3]

Assunzione 8.1

1. La funzione obiettivo f è due volte continuamente differenziabile.
2. Il level set $\{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$ è convesso e lì esistono le costanti positive m e M tali che:
$$m\|z\|^2 \leq z^T G(x) z \leq M\|z\|^2$$
per tutti i $z \in \mathbb{R}^n$ e x nel level set.

La seconda parte di questa assunzione implica che $G(x)$ sia definita positiva sul level set e che f abbia un unico minimizer x^* nel level set.

Applicazione dell'Assunzione 8.1

1. Due volte continuamente differenziabile:
$f(W_2)$ è composta da due termini, ovvero il termine di errore quadratico $\|W_2\sigma(W_1x) - y\|_2^2$ e il termine di regolarizzazione $L_2\lambda\|W_2\|_2^2$.
Entrambi questi termini sono funzioni quadratiche, quindi la loro somma è una funzione quadratica e di conseguenza sono due volte differenziabili.
2. Convessità del level set:
per dimostrare che il level set è convesso, dobbiamo dimostrare che la funzione obiettivo

è convessa. Come riportato nella Sezione 2.1, la funzione risulta essere convessa e di conseguenza il level set $\{W_2 : f(W_2) \leq f(W_2^0)\}$ è convesso.

Successivamente definiamo:

- La matrice Hessiana ∇^2 della funzione obiettivo $f(W_2)$ come:

$$\nabla^2 \|W_2\sigma(W_1x) - y\|_2^2 = 2\sigma(W_1x)^T\sigma(W_1x)$$

- La matrice Hessiana del termine di regolarizzazione $L_2\lambda\|W_2\|_2^2$ come:

$$\nabla^2 \lambda\|W_2\|_2^2 = 2\lambda I$$

- Perciò, la matrice Hessiana finale è delineata come:

$$2\sigma(W_1x)^T\sigma(W_1x) + 2\lambda I$$

Poiché $2\sigma(W_1x)^T\sigma(W_1x)$ è una matrice *semidefinita positiva* (prodotto di una matrice con il suo trasposto) e $2\lambda I$ è una matrice *definita positiva* (ricordando che $\lambda > 0$ e perciò essendo una matrice identità tutti i suoi autovalori sono strettamente positivi), la matrice Hessiana totale è **definita positiva**. Possiamo quindi affermare che esistono due costanti positive m e M tali che per ogni z in $\mathbb{R}^n$ vale che:

$$m\|z\|^2 \leq z^T\nabla^2 f(W_2)z \leq M\|z\|^2$$

Inoltre, la definita positività dell'Hessiana implica che la funzione obiettivo sia **strettamente convessa** ed abbia un **unico minimo globale** x^* all'interno del level set. Questo è fondamentale per garantire che l'algoritmo converga verso una soluzione.

Teorema 8.5

Sia B_0 una matrice iniziale simmetrica e definita positiva, e sia x_0 il punto iniziale per cui l'assunzione 8.1 è soddisfatta. Allora la sequenza $\{x_k\}$ generata dall'algoritmo 8.1 converge al minimizer x^* di f.
Consideriamo la nostra matrice Hessiana ∇^2, ricordando che essa è definita positiva come dimostrato in precedenza; e ne verifichiamo la sua simmetria:
Essa è composta da:

- $2\sigma(W_1x)^T\sigma(W_1x)$, che è simmetrica poiché il prodotto di un vettore per se stesso è sempre una matrice simmetrica.

- $2\lambda I$, che è simmetrica per via delle proprietà della matrice identità che rimangono invariate quando viene moltiplicato il termine scalare.

Concludendo quindi la matrice Hessiana è **simmetrica**, e attraverso tali nozioni la **convergenza globale** è stata verificata.
Successivamente, per garantire una **convergenza superlineare** è necessario verificare la *continuità Lipschitziana* della Matrice Hessiana.

Assunzione 8.2

La matrice Hessiana G è Lipschitz continua in x^*, tale che:

$$\|G(x) - G(x^*)\| \leq L\|x - x^*\|$$

per tutti gli x vicini a x^*, dove L è una costante positiva.

Applicazione dell'Assunzione 8.2

Per dimostrare la *continuità Lipschitziana* della matrice Hessiana, dobbiamo verificare che esiste una costante M tale che per ogni coppia di punti W_2 e W_2^*:

$$\|\nabla^2 f(W_2) - \nabla^2 f(W_2^*)\| \leq M\|W_2 - W_2^*\|$$

La norma della differenza tra le matrici Hessiane è:

$$\|\nabla^2 f(W_2) - \nabla^2 f(W_2^*)\| = \left\|2\sigma(W1x)^T\sigma(W1x) - 2\sigma(W1x^*)^T\sigma(W1x^*)\right\|$$

Essa può essere riscritta come:

$$\|\nabla^2 f(W_2) - \nabla^2 f(W_2^*)\| = 2\left\|\sigma(W1x)^T\sigma(W1x) - \sigma(W1x^*)^T\sigma(W1x^*)\right\|$$

Generalmente, la differenza $\left\|\sigma(W1x)^T\sigma(W1x) - \sigma(W1x^*)^T\sigma(W1x^*)\right\|$ può essere stimata come $M\|W_2 - W_2^*\|$, dove M è una costante positiva dipendente dalla funzione di attivazione e dalla matrice $W1$. Quindi utilizzando questa stima è possibile sostituire il secondo termine ottenendo la seguente disuguaglianza:

$$\|\nabla^2 f(W_2) - \nabla^2 f(W_2^*)\| \leq 2M\|W_2 - W_2^*\|$$

Allora possiamo concludere che $\nabla^2 f(W_2)$ è **Lipschitz continua** rispetto a W_2 con una costante di Lipschitz *2M*. La scelta di tale costante si basa sulla stima della massima variazione della matrice Hessiana tra due punti $\|W_2 - W_2^*\|$.

Assunzione 8.52

Il tasso di convergenza delle iterazioni è lineare. In particolare è possibile mostrare che la sequenza $\|x_k - x^*\|$ converge a zero rapidamente abbastanza che:

$$\sum_{k=1}^{\infty} |x - x_k| < \infty \tag{3}$$

- Avendo verificato che f è due volte continuamente differenziabile, fortemente convessa e la matrice Hessiana è Lipschitz-continua, possiamo concludere che esiste un tasso di convergenza lineare con costante $c \in (0, 1)$. Questo implica che:

  $$\|\mathbf{W}_{2_{k+1}} - \mathbf{W}_*\| \leq c\|\mathbf{W}_{2_k} - \mathbf{W}_*\|$$

  E quindi l'assunzione 8.52 può essere verificata teoricamente.

Teorema 8.6

Supponiamo che f sia due volte continuamente differenziabile e che le iterazioni generate dall'algoritmo BFGS convergono al minimizer x^* al quale regge l'assunzione 8.2. Supponiamo anche che regga l'assunzione 8.52.

- Allora x_k converge a x^* ad una *velocità superlineare*.

Applicazione Teorema 8.6

La nostra $f(W_2)$ è due volte continuamente differenziabile come dimostrato in precedenza e inoltre è stata verificata anche l'assunzione 8.2. Supponendo la validità dell'assunzione 8.52, possiamo concludere che W_2 converge a W_{2_*} ad una **velocità superlineare**.

4. Cholesky Factorization

La fattorizzazione di Cholesky è una tecnica utilizzata per decomporre una matrice simmetrica e definita positiva in un prodotto di due matrici: una matrice triangolare inferiore e la sua trasposta. Questo tipo di decomposizione è utile poiché consente di risolvere sistemi lineari in modo efficiente e stabile rispetto alla risoluzione diretta del sistema. Per tali motivi esso è anche meno sensibile agli errori di approssimazione.
Nel nostro problema il sistema lineare è rappresentato da:

$$\|HW_2 - T\|_2^2$$

in cui

- H: Rappresenta la matrice di output dell'hidden layer.
- T: Rappresenta il vettore dei valori target.

Di seguito illustriamo i passaggi per risolvere il nostro problema attraverso questo approccio:

- Definiamo la matrice Q da passare in input all'algoritmo come:

  $$H^T H + N\lambda I$$

  dove N è il numero di campioni.

Per applicare la fattorizzazione di Cholesky [4], dobbiamo prima verificarne la sua esistenza. Sappiamo che essa esiste ed è unica se la matrice Q è simmetrica e definita positiva, ovvero $Q = Q^T$ e tutti gli autovalori sono strettamente positivi.

- **Q è simmetrica:**

  $$Q^T = (H^T H + N\lambda I)^T = (H^T H)^T + (N\lambda I)^T = HH^T + N\lambda I = Q$$

  Questo è verificato poiché la trasposta della somma di due matrici è la somma delle trasposte delle singole matrici, e la trasposta del prodotto di due matrici è il prodotto delle trasposte di queste matrici. La matrice identità è invece simmetrica per definizione.

- **Q è definita positiva:** La matrice Q è composta dal primo termine $H^T H$ che rappresenta una matrice di *Gram*.[2] Per definizione, tutti gli autovalori di una matrice di *Gram* sono

[2]Matrice di Gram

reali e non negativi, quindi $H^T H$ è semidefinita positiva. Il secondo termine invece è $N\lambda I$, e per le proprietà della matrice identità, possiamo affermare che essa è definita positiva poiché tutti i suoi autovalori sono positivi con $\lambda > 0$ e $N > 0$. Quest'ultimo parametro è positivo poiché la regolarizzazione L_2 aggiunge un termine quadratico ai pesi durante l'ottimizzazione del modello, il che aiuta a prevenire l'eccessiva complessità del modello e il fenomeno di overfitting. Infine, la somma di una matrice semidefinita positiva e una definita positiva è definita positiva.

Concludendo, possiamo asserire che la **fattorizzazione di Cholesky esiste per il nostro problema**.
Dopo tale verifica, è possibile quindi proseguire con le restanti fasi:

- Applichiamo la fattorizzazione di Cholesky alla matrice *Q* per ottenere *L* tale che

$$Q = LL^T$$

Si procede quindi calcolando gli elementi di *L* utilizzando le seguenti formule:

- Per determinare gli elementi sulla diagonale:

$$L_{i,i} = \sqrt{Q_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2}$$

- Per determinare gli elementi fuori dalla diagonale:

$$L_{i,j} = \frac{1}{L_{j,j}} \left(Q_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right) \quad \text{per } j < i$$

Questo procedimento viene ripetuto per tutte le righe e colonne di *L*, fino a quando la matrice sarà completamente piena.
In prosieguo, si continua con la risoluzione del seguente sistema lineare:

$$H^T H W_2 = H^T T \Leftrightarrow LL^T W_2 = H^T T \Leftrightarrow Ly = H^T T$$

- Per tale operazione si risolve prima $Ly = H^T T$ (*sostituzione in avanti*) e successivamente, detta y^* la sua soluzione, viene risolto $L^T W_2 = y^*$ (*sostituzione all'indietro*).

Infine, i pesi di output ottimizzati W_2 vengono restituiti.

4.1 Complessità computazionale

Il costo computazionale per la risoluzione del sistema con Cholesky comprende:

- $\frac{n^2 m}{2}$ operazioni moltiplicative per il calcolo di $H^T H$ (si tiene conto della simmetria di tale matrice);

- nm operazioni moltiplicative per il calcolo di $H^T T$;
- $\frac{n^3}{6}$ operazioni per la risoluzione del sistema lineare $H^T H W_2 = H^T T$ col metodo di Cholesky.

Dato che usualmente $n^2 m^2 \gg nm$, il numero di operazioni moltiplicative risulta essere:

$$O(n^2 m^2 + nm + \frac{n^3}{6}) \approx O(n^2 m^2 + \frac{n^3}{6})$$

dove n è il numero di variabili di input e m è il numero di campioni.

5. Esperimenti

In questa Sezione, vengono descritti gli esperimenti condotti per il nostro problema e i risultati generati tramite l'esecuzione dei metodi di ottimizzazione adoperati. Inizialmente è stata effettuata la generazione di un dataset sintetico con 1000 features e 30,000 samples, basato su un problema di classificazione binaria; e a tal proposito è possibile definire la struttura della nostra rete neurale:

- Numero di input: 1000
- Numero di neuroni nell'hidden layer: 10
- Numero di pesi di input W_1: 1000 x 10 = 10000
- Numero di pesi di output W_2: 10 x 1 = 10

Di conseguenza, il numero totale di variabili nella rete risulta essere pari a 10010.
La funzione di attivazione scelta per il calcolo dell'output nell'ELM è la *sigmoide*, grazie alle sua facilità di calcolo delle derivate (utile per il BFGS) e la stabilità numerica. Inoltre, per evitare l'overfitting, il parametro di regolarizzazione utilizzato è l'*L2* per ridurre l'overfitting penalizzando i pesi grandi. A tale fattore è stato associato un valore pari a *lambda_reg = 0.01*.
L'ELM richiede la scelta casuale dei pesi di input e degli hidden biases, per tale motivo i pesi W_1 e i biases b_k sono stati generati in maniera random; e per garantire un'inizializzazione pseudo-casuale, è stata adottata la *Xavier Initialization* espressa nella maniera seguente:

$$W \sim \mathcal{U}\left(-\frac{1}{\sqrt{n_{\text{in}}}}, \frac{1}{\sqrt{n_{\text{in}}}}\right)$$

dove n_{in} indica il numero di input.
Questa inizializzazione è stata garantita anche per i pesi di output W_2 per l'algoritmo BFGS. Inoltre, è stato impostato un parametro *seed* per garantire la riproducibilità dei risultati quando si inizializzano i pesi in modo casuale. Il *seed* garantisce che ogni esecuzione dell'algoritmo inizi con gli stessi pesi iniziali. Questo è utile per confrontare i risultati tra diverse esecuzioni e tra diverse tecniche, assicurando che eventuali differenze nei risultati siano dovute agli algoritmi stessi e non a diverse inizializzazioni random.
Successivamente i passaggi condotti sono i seguenti:

- Implementazione della funzione obiettivo.

- Implementazione dell'ELM.
- Implementazione del BFGS.
- Implementazione della fattorizzazione Cholesky e risoluzione delle equazioni normali.

Le descrizioni delle analisi condotte con gli algoritmi di ottimizzazione BFGS e Cholesky sono riportate nei paragrafi successivi.

5.1 BFGS

Il BFGS è stato implementato scegliendo i parametri in modo da selezionare la combinazione di valori più adatti, considerando la tolleranza e alpha_1.
In particolare, i valori acquisiti sono: *tol = 1e-15, alpha_1 = 2.5*.
Una prima analisi condotta per il BFGS è il calcolo del *Relative Gap*, per valutare la precisione dell'implementazione, identificare eventuali errori e assicurare l'attendibilità dei risultati. Esso è definito come:

$$\frac{\|W_2 - W_2^*\|}{\|W_2^*\|}$$

dove W_2 è la soluzione restituita dall'algoritmo implementato, e W_2^* è la soluzione ottimale trovata dal comando *backslash* di Matlab per la risoluzione del sistema lineare ottenuto attraverso i termini della funzione obiettivo. Per quest'ultimo, considerando le assunzioni discusse nella parte teorica riguardo l'unicità di un minimo globale, sono stati eseguiti i seguenti passaggi:

- Calcolo del gradiente rispetto a W_2, posto uguale a 0:

$$H^T HW2 - H^T Y + N\lambda W2 = 0$$

- Risoluzione del sistema:

$$(H^T H + N\lambda I)W_2 = H^T Y$$

  in cui H è la matrice delle attivazioni dei dati di input X rispetto a W_1 e la funzione di attivazione, Y è il vettore dei target e I è la matrice identità.

Inizialmente dagli esiti ottenuti, è stato possibile notare come il *relative gap* della prima iterazione è elevato rispetto alle successive, in cui si nota una drastica diminuizione; passando da un valore pari a 1.97608497318934 ad un valore uguale a 1.5047135836575e-14.
Una sintesi di quanto affermato, si può evincere dal *line plot* in Figura 1, utilizzato per rappresentare l'andamendo del *Relative Gap*, dei valori della *Funzione Obiettivo* e della *Norma del Gradiente*. Tale grafico è utile per monitorare il progresso dell'algoritmo BFGS, consentendo di osservare come le diverse metriche cambiano con le iterazioni e di assicurarsi che l'algoritmo stia effettivamente convergendo.

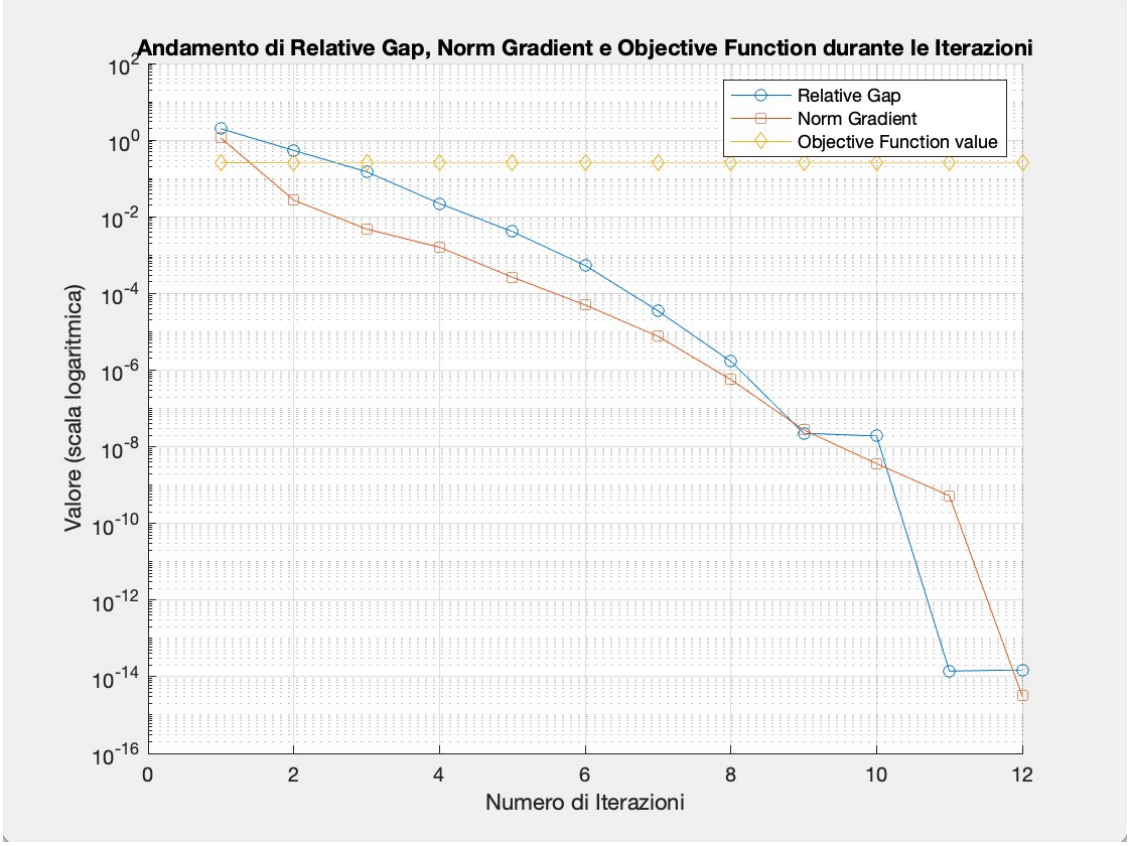

Figure 1: Progresso del BFGS durante l'allenamento

Infatti è possibile notare che:

- La *Funzione obiettivo* rimane relativamente costante nel tempo, giungendo ad un valore pari a 0.252015477303778. Questo comportamento suggerisce che il valore della funzione obiettivo potrebbe essere già vicino al minimo, e inoltre le variazioni essendo molto piccole potrebbero non essere visibili in scala logaritmica.
- La *Norma del Gradiente* diminuisce drasticamente con le iterazioni, indicando che questo approccio si sta avvicinando ad un minimo. Infatti un gradiente molto prossimo allo zero evidenzia il raggiungimento di un punto stazionario, ovvero un potenziale minimo.
- Il *Relative Gap* diminuisce con il numero di iterazioni e verso la fine decrementa drasticamente per poi stabilizzarsi nelle ultime due iterazioni, sottolineando che l'algoritmo sta convergendo verso il valore ottimale.

Infine, è possibile dedurre che l'algoritmo BFGS converge rapidamente entro circa 12 iterazioni, nonostante esso sia stato impostato per un numero massimo di iterazioni maggiore. I comportamenti osservati (velocità rapida di stabilizzazione della funzione obiettivo e riduzione rigorosa del gradiente e del gap) sono indicatori di una velocità di convergenza superiore a quella attesa per i metodi di convergenza più tradizionali. Questa analisi empirica è supportata dalle caratteristiche teoriche della convergenza superlineare, che designano una riduzione dell'errore in modo più rapido e efficace.

5.2 Cholesky

In Cholesky, dopo aver eseguito l'implementazione, è stato calcolato nuovamente il *Relative Gap*, nella stesso modo descritto nella Sezione 5.1 per il BFGS. Il valore di tale metrica ottenuto per questa tecnica è pari a 9.3674420719433e-15.

5.3 Confronto

Il confronto tra i vari metodi di ottimizzazioni, è stato effettuato in termini di valore ottenuto dalla funzione obiettivo, valori dei pesi di output W_2 che minimizzano la funzione e la precisione della soluzione misurabile attraverso i rispettivi gap. Inizialmente nella Tabella 1, sono riportati i pesi di output W_2 ottenuti dai diversi approcci, considerando le dimensioni del problema riportate precedentemente all'inizio della Sezione 5. Il paragone tra i pesi di output estrapolati con con la soluzione ottimale, con BFGS e con Cholesky mostra l'importanza della scelta dell'algoritmo di ottimizzazione.

W_2 ottimali	W_2 con BFGS	W_2 con Cholesky
0.0749458467963751	0.0749458467963736	0.0749458467963758
0.0679874760384262	0.0679874760384255	0.0679874760384262
0.0381851005228706	0.0381851005228718	0.0381851005228709
0.0897442962175778	0.0897442962175789	0.0897442962175794
0.0920264188959944	0.0920264188959957	0.0920264188959942
0.142758001594618	0.142758001594618	0.142758001594616
0.0261635398260668	0.0261635398260676	0.0261635398260664
0.114723374851061	0.114723374851063	0.114723374851062
0.0860104552679081	0.0860104552679047	0.0860104552679093
0.154246728014711	0.154246728014711	0.154246728014709

Table 1: Confronto pesi di output W_2

I pesi ottenuti da queste tre tecniche sono molto simili, con differenze molto piccole tra i valori corrispondenti. Questo suggerisce che entrambi i metodi di ottimizzazione conducono a un insieme di pesi molto simile, se non lo stesso. Tali risultati mostrano che la funzione obiettivo è ben condizionata e la soluzione trovata è robusta rispetto al metodo di ottimizzazione utilizzato. Il valore della funzione obiettivo ottenuto da Cholesky risulta essere pari a 0.252015477303778, e quindi uguale al valore ottenuto per il BFGS. Mentre con la soluzione ottimale ottenuta, la funzione obiettivo risulta essere nuovamente uguale a 0.252015477303778. Questi esiti riferiscono che entrambi i metodi di ottimizzazione sono in grado di trovare una soluzione ottimale per il problema specifico. Per quanto riguarda il relative gap, Cholesky conduce ad un valore inferiore rispetto al BFGS, sottolineando quindi una maggiore precisione rispetto a quest'ultimo nel trovare la soluzione ottimale.
Infine è stato analizzato il tempo di esecuzione dei metodi al variare del numero di variabili di input. Tale analisi è stata raffigurata tramite un *line plot*, presente in Figura 2.

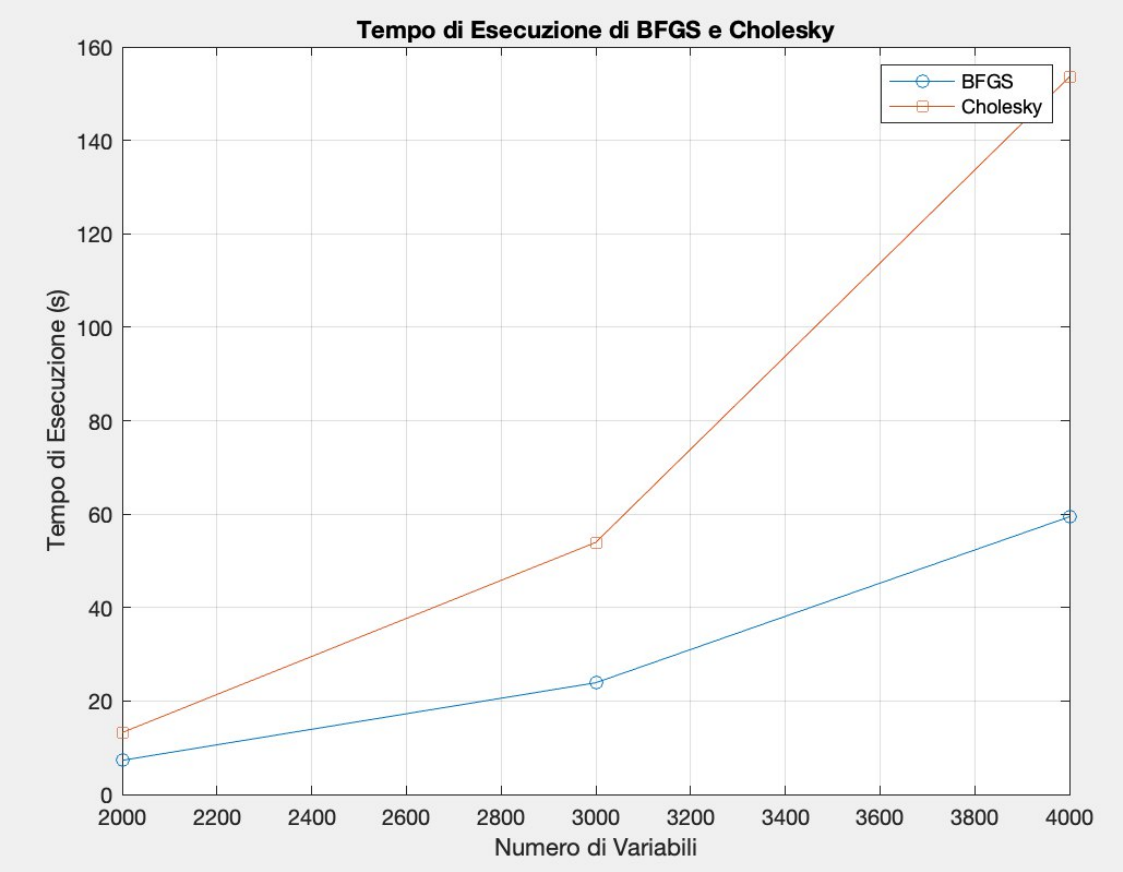

Figure 2: Tempo di Esecuzione al variare del numero di variabili di input e di iterazioni

In particolare è stato aumentanto il numero di samples e il numero di neuroni nascosti, rendendo la matrice H (output dell'hidden layer) quadrata, in modo da visualizzare meglio la crescita della durata dell'esecuzione. Di conseguenza, con tale aumento, il numero di pesi e il numero totale delle variabili della rete sono incrementati.
Il grafico, rappresentato nell'immagine 2, riporta quindi l'andamento del tempo di esecuzione del metodo di ottimizzazione BFGS (in questo esperimento senza l'impostazione di un numero massimo di iterazioni) e di Cholesky.
Il costo computazionale atteso per la risoluzione del sistema attraverso Cholesky è $O(n^2m^2 + \frac{n^3}{6})$, dove *n* è il numero di variabili di input e *m* è il numero di campioni. La curva del tempo di esecuzione per Cholesky mostra un andamento crescente, e per problemi di dimensioni minori (inferiori o uguali a 2000) esso ha un'efficienza pressochè simile al BFGS. Con l'aumentare delle dimensioni del problema, l'efficienza computazionale di Cholesky diminuisce per via della sua complessità. Infatti, quest'ultima, osservabile nel grafico, sembra rispecchiare la complessità cubica attesa per valori grandi di *n*; poiché all'aumentare della dimensione di input incrementa in modo cubico anche il tempo di esecuzione.
In base al numero di iterazioni del metodo BFGS, la complessità è $O(k * n^2)$ dove *n* è il numero di variabili di input e *k* è il numero di iterazioni che l'algoritmo effetua fino alla convergenza. Il tempo di esecuzione mostra una variazione significativa con l'aumento del numero di variabili. Questo comportamento può indicare che il metodo BFGS è sensibile alla dimensione del problema

e al numero di riprocessamenti, con alcune configurazioni che richiedono un tempo di calcolo significativamente maggiore. Inoltre, tale comportamento evidenzia che aumentando il numero di variabili di input l'algoritmo BFGS necessita di un numero di iterazioni superiore per trovare una soluzione prossima a quella ottimale, rispetto sia agli esperimenti condotti inizialmente sia alle varie dimensioni testate. Nel grafico in Figura 1, la curva piatta della funzione obiettivo dopo poche iterazioni designa che esso converge rapidamente al valore ottimale, il che è un buon segno di efficienza dell'algoritmo. Tuttavia, l'aumento del tempo di esecuzione suggerisce che, per grandi dimensioni, il costo per iterazione si accumula, ma comunque risulta essere più veloce rispetto a Cholesky, per k relativamente piccolo.
Attraverso la Tabella 2, sono evidenziati i risultati riportarti dalle due tecniche in base alle dimensioni, in modo da effettuare un confronto in termini di complessità computazionale e di relative gap. Inoltre, per il BFGS viene sottolineato anche il numero di iterazioni utili per la convergenza per ogni casistica.

Dimensioni H	Tempo Cholesky	Tempo BFGS	Iterazioni BFGS	Gap BFGS	GAP Cholesky
2000x2000	13.247937656	7.297810256	62	6.9089e-13	7.1402e-13
3000x3000	53.95553652	23.910604316	67	1.0386e-12	1.1032e-12
4000x4000	153.573317223	59.421995947	68	1.4294e-12	1.5349e-12

Table 2: Confronto Metodi di Ottimizzazione

Proseguendo, è ragionevole che con un numero elevato di iterazioni k, il BFGS può essere in grado di trovare una soluzione precisa e sempre più comparabile con quella ottenuta tramite il metodo di Cholesky, come evidenziato dai gap presenti nella Tabella 2. Conseguentemente il tempo di esecuzione cresce progressivamente, ma in questi esperimenti rimane comunque inferiore a quello richiesto dal metodo diretto (il BFGS impiega circa la metà del suo tempo).
In seguito, per estendere queste ultime analisi, è stata eseguita un'ulteriore verifica basata sulla variazione del parametro di regolarizzazione *L2*, denominato *lambda_reg*. Nello specifico sono stati testati tre valori differenti di tale incognita, e gli esiti suddivisi in base alle dimensioni di input osservate sono visibili tramite le tabelle 3, 4 e 5.

lambda_reg	Tempo Cholesky	Tempo BFGS	Iterazioni BFGS	Gap BFGS	GAP Cholesky
1e+2	14.0346	0.8213	7	3.9354e-15	3.4624e-15
1e+0	13.6551	1.2729	12	6.9255e-14	5.9292e-14
1e-2	16.2427	8.3697	62	6.9095e-13	7.1402e-13

Table 3: Dimensione della matrice quadrata H: 2000x2000

lambda_reg	Tempo Cholesky	Tempo BFGS	Iterazioni BFGS	Gap BFGS	GAP Cholesky
1e+2	76.3097	2.4170	7	5.8988e-15	5.2050e-15
1e+0	78.3276	4.8315	12	1.2073e-13	1.1291e-13
1e-2	82.9769	27.9409	67	1.0387e-12	1.1032e-12

Table 4: Dimensione della matrice quadrata H: 3000x3000

lambda_reg	Tempo Cholesky	Tempo BFGS	Iterazioni BFGS	Gap BFGS	GAP Cholesky
1e+2	310.7548	6.9003	7	9.9161e-15	7.2609e-15
1e+0	220.9988	13.2311	13	1.7052e-13	1.5680e-13
1e-2	190.1181	145.9135	68	1.4294e-12	1.5349e-12

Table 5: Dimensione della matrice quadrata H: 4000x4000

Attraverso questa indagine è possibile dedurre come varia il comportamento delle tecniche di ottimizzazione e quali sono i fattori che maggiormente le influenzano. Infatti, è osservabile come i tempi di calcolo tendono a crescere con l'aumentare delle dimensioni della matrice, ma in generale la complessità di Cholesky rimane abbastanza stabile per un dato *lambda_reg*. Questo potrebbe indicare che tale approccio è meno sensibile alla variazione del parametro di regolarizzazione e più sensibile alla dimensione di input.
Al contrario, la complessità computazionale per il BFGS aumenta drasticamente quando il paramatro di regolarizzazione viene ridotto. Questo è particolarmente evidente per *lambda_reg = 1e-2*, dove i tempi passano da pochi secondi a centinaia o migliaia di secondi al crescere delle variabili di input. Tale risultato suggerisce che il BFGS è molto più sensibile alla regolarizzazione, richiedendo molto più tempo per convergere quando questo fattore è piccolo. A tal proposito infatti, è stato osservato anche il comportamento con il valore *lambda_reg = 1e-4*. Con tale osservazione non è stato possibile riportare un esito per via dell'eccessiva complessità temporale.
Inoltre, il numero di iterazioni necessarie per l'approccio iterativo incrementa in modo significativo quando l'indicatore *L2* diminuisce. Ciò è coerente con l'aumento del tempo di esecuzione e indica che un valore di *lambda_reg* inferiore rende il problema più difficile da risolvere per il BFGS, richiedendo più riprocessamenti per raggiungere la convergenza.
Per quanto riguarda i relative gap per entrambi i metodi, essi tendono a crescere con la diminuzione della variabile *lambda*, il che è prevedibile poiché una regolarizzazione minore porta a soluzioni meno stabili e più sensibili agli errori numerici. Tuttavia, i gap rimangono comunque molto piccoli, denotando che entrambi i metodi sono capaci di trovare soluzioni con alta precisione, anche se il tempo di calcolo e il numero di iterazioni necessari variano significativamente.
Concludendo, si può affermare che Cholesky è più robusto in termini di complessità rispetto alle variazioni del parametro di regolarizzazione, ma richiede più tempo al crescere della dimensione del problema. Invece, il BFGS è più veloce per valori più alti di *lambda_reg*, ma diventa molto meno efficiente per valori più bassi, soprattutto in termini di esecuzione e numero di iterazioni.

Conclusioni

Nella valutazione dei metodi di ottimizzazione per il nostro problema, il focus principale è stato il *gap relativo* rispetto al valore ottimale. Questo parametro ha permesso di misurare quanto vicino l'algoritmo è riuscito a raggiungere il minimo globale desiderato.
Inoltre, è stato osservato che, per il nostro problema, il numero di variabili ha un impatto significativo sulla complessità computazionale. Questa osservazione è stata confermata dai grafici in Figura 2, dove è stato confrontato il tempo di esecuzione dei metodi al variare del numero di variabili di input. Questo suggerisce che la dimensione del problema, in termini di numero di variabili, è cruciale per determinare l'efficienza e la scalabilità degli algoritmi di ottimizzazione.
Per quanto riguarda il metodo Cholesky, ci si aspetta una complessità computazionale teorica

cubica. Nei risultati, è stato visualizzato un aumento graduale del tempo di esecuzione con l'aumentare del numero di variabili. Per BFGS, la complessità teorica per singola iterazione è quadratica, e generalmente dipende anche dal numero k di riprocessamenti eseguiti. Il BFGS non è stato configurato per un massimo di iterazioni durante l'analisi temporale, osservando che l'algoritmo converge spesso entro poche iterazioni effettive. Tuttavia, l'aumento delle iterazioni può influenzare il tempo complessivo di esecuzione, soprattutto per dimensioni più grandi.
L'analisi empirica ha mostrato che entrambi i metodi di ottimizzazione, sebbene con approcci e complessità differenti, sono stati efficaci nel minimizzare la funzione obiettivo e nel ridurre il gap rispetto al valore ottimale. Infatti, tale metrica rispetto alla soluzione ottimale è stata costantemente bassa e simile tra loro, indicando che entrambi gli approcci risultano essere abbastanza precisi.
In conclusione, la scelta dell'algoritmo di ottimizzazione dipende dalle specifiche del problema (come per esempio il valore del fattore di regolarizzazione e la dimensione di input), bilanciando la complessità computazionale con l'obiettivo di ottenere soluzioni accurate e efficienti. Infatti, per matrici più grandi e regolarizzazione forte (*lambda* elevato), BFGS è competitivo, ma Cholesky diventa preferibile per problemi con regolarizzazione ridotta. In aggiunta, il BFGS può essere prediletto per problemi in cui il numero di iterazioni non è eccessivamente elevato, poiché BFGS scala meglio con il numero di variabili grazie alla sua crescita quadratica rispetto alla crescita cubica di Cholesky. Il metodo di Cholesky, invece, è più adatto per problemi con piccole dimensioni di dati dove la prevedibilità e la stabilità del calcolo sono cruciali. Infine, essendo la fattorizzazione di Cholesky un metodo diretto e non iterativo esso risulta essere vantaggioso nei casi in cui è fondamentale una maggior precisione.

References

[1] Glenn Paul Gara. Build an extreme learning machine in python. Towards Data Science, 2020.

[2] Qin-Yu Zhu Guangbin Huang and Chee Kheong Siew. *Extreme learning machine: a new learning scheme of feedforward neural networks.* IEEE International Joint Conference on Neural Networks, 2004. (IEEE Cat. No.04CH37541), 2:985–990 vol.2, 2004.

[3] Nocedal J. and Wright S.J. *Numerical Optimization.* Springer, 1999. ISBN 0-387-98793-2 Springer-Verlag New York Berlin Heidelberg SPIN 10764949.

[4] Sommariva A. Algebra Lineare Numerica, 2021. URL `https://www.math.unipd.it/alvise/CN/TEORIA/ALGEBRALINEARETEORIA/algebralineare PDF.pdf`.