UNIVERSITÀ DI PISA

MSc IN DATA SCIENCE AND BUSINESS INFORMATICS

Optimization for Data Science

Neural Network with Momentum descent approach and Deflected Subgradient methods

Mattia Arancio Febbo - 561930

Michele Dicandia - 657494

ANNO ACCADEMICO 2024-25

Indice

1 Introduzione

Le reti neurali sono modelli fondamentali per affrontare problemi complessi come la classificazione di immagini, il riconoscimento vocale e il linguaggio naturale. Il loro successo dipende dalla capacità di ottimizzare efficacemente i parametri del modello, minimizzando una funzione di perdita spesso non convessa e altamente complessa.

Questo processo di ottimizzazione è particolarmente impegnativo, poiché la convergenza degli algoritmi è influenzata dalla topologia della rete, dalla funzione di perdita e dai valori iniziali dei parametri.

In questo progetto, analizziamo il comportamento di due algoritmi di ottimizzazione: il metodo della discesa con momentum (*Heavy Ball*) e il metodo del subgradiente deflesso. Questi algoritmi sono stati scelti per esplorare diverse strategie di convergenza, incluse quelle applicabili a funzioni non differenziabili. Gli esperimenti sono condotti su una rete neurale progettata per risolvere un problema di classificazione multiclasse.

Il report è strutturato come segue:

- **Prima sezione**: descrizione del dataset adottato, descrizione della rete neurale e della funzione di perdita utilizzata.
- **Seconda sezione**: introduzione dei due algoritmi di ottimizzazione e delle loro proprietà teoriche.
- **Ultima sezione**: presentazione dei risultati degli esperimenti.

1.1 Dataset

Il dataset selezionato per questo progetto è il Digits, un dataset comunemente utilizzato per valutare algoritmi di classificazione. Il dataset è una copia del test set dei dataset di riconoscimento ottico dei numeri scritti a mano forniti dall'UCI Machine Learning Repository.

Esso è composto da 1797 immagini in scala di grigi, ciascuna con una risoluzione di 8×8 pixel (64 pixel totali). Ogni immagine rappresenta una cifra scritta a mano, suddivisa in 10 categorie corrispondenti ai numeri da 0 a 9.

Le immagini, originariamente in formato raster, vengono convertite in rappresentazioni numeriche. Ogni pixel rappresenta un livello di luminosità con un valore compreso tra 0 (bianco puro) e 16 (nero puro). Questo processo permette di trasformare ogni immagine in un vettore numerico di dimensione 1×64, dove ciascun elemento rappresenta l'intensità di luminosità di un pixel specifico.

Un esempio della griglia dei pixel per un'immagine è mostrato in Figura 1, dove si osservano i diversi livelli di luminosità.

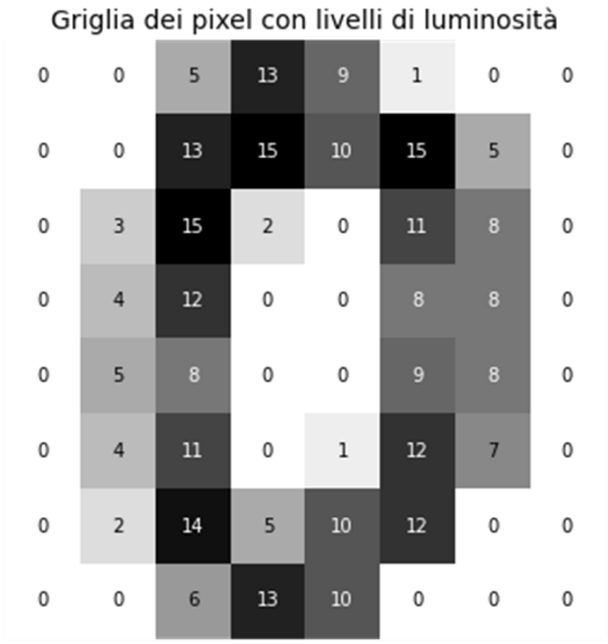


Figura 1: Esempio di griglia dei pixel per un'immagine.

Per uniformare i dati e migliorare l'efficienza degli algoritmi di ottimizzazione, i valori dei pixel vengono normalizzati dividendo ciascun valore per 16. Questo processo trasforma tutti i pixel in un range tra 0 e 1, rendendo pertanto i dati più adatti agli algoritmi di apprendimento automatico.

Per l'addestramento del modello, il dataset Digits, composto da 1797 immagini, è suddiviso come segue:

- **Training set**: pari a 1437 immagini.
- **Test set**: pari a 360 immagini.

Dopo il preprocessing, il dataset viene organizzato in matrici e vettori:

- **Training set** (X_{train}): una matrice di dimensione 1437×64, contenente le rappresentazioni numeriche delle immagini di addestramento
- **Test set** (X_{test}): una matrice di dimensione 360×64, contenente le rappresentazioni numeriche delle immagini di test.
- **Label del training set** (y_{train}): un vettore di dimensione 1437, contenente le classi associate alle immmagini del training set.
- **Label del test set** (y_{test}): un vettore di dimensione 360, contenente le classi associate alle immagini del test set.

1.1.1 Modello di Rete Neurale

Il modello utilizzato per l'analisi di questo dataset è una rete neurale composta da un input layer con 64 neuroni corrispondenti ai pixel di ciascuna immagine del dataset (8x8), un hidden layer di 32 neuroni e un output layer di 10 neuroni che corrispondono alle categorie. Poichè l'obiettivo non è eseguire un'analisi approfondita di machine learning, non ci focalizziamo sull'ottimizzazione della struttura dell'hidden layer.

Pertanto, ci soffermiamo solo a specificare che per i *hidden layers* è stata scelta la funzione di attivazione **ReLU** (Rectified Linear Unit), mentre per l'*output layer*, è stata utilizzata la funzione di attivazione **softmax**.

Invece è stata adottata la funzione di perdita **cross-entropy**, fondamentale per il suo impatto significativo sulle prestazioni della rete neurale.

Di seguito viene fornita una descrizione dettagliata del processo di apprendimento di una rete neurale, accompagnata dalle relative formule matematiche per supportare e chiarire ogni passaggio.

Innanzitutto, nella seguente scaletta sono riportate le informazioni necessarie per comprendere in modo più chiaro le formule matematiche che descrivono il processo stesso:

- x: input vettoriale di dimensione 64×1
- W^1: matrice dei pesi del primo layer di dimensione 32×64
- b^1: bias del primo layer di dimensione 32×1
- W^2: matrice dei pesi del secondo layer di dimensione 10×32
- b^2: bias del secondo layer di dimensione 10×1
- a^1: attivazione del primo layer
- z^1: output lineare del primo layer
- a^2, z^2: attivazione e output lineare del secondo layer

Per quanto riguarda la **Forward Pass**:

1. **Primo Layer (Hidden Layer)**:

$$z^1 = W^1 x + b^1$$
$$a^1 = \text{ReLU}(z^1) = \max(0, z^1)$$

2. **Secondo Layer (Output Layer)**:

$$z^2 = W^2 a^1 + b^2$$
$$a^2 = \text{Softmax}(z^2)$$

Dove la funzione **Softmax** è definita come:

$$\text{Softmax}(z_i^2) = \frac{e^{z_i^2}}{\sum_{j=1}^{10} e^{z_j^2}}$$

Per quanto riguarda la **Funzione di Loss**:

- **Cross-Entropy Loss** per classificazione multiclasse:

$$J(W^1, b^1, W^2, b^2) = -\sum_{i=1}^{m} \sum_{c=1}^{10} y_{i,c} \log(a_{i,c}^2)$$

Dove $y_{i,c}$ rappresenta un vettore di etichette binarie che indica se la classe c è la classe corretta per un determinato record i. Invece $a_{i,c}$ rappresenta l'output della rete neurale dopo l'applicazione della funzione softmax per il campione i e la classe c. Mentre m è il numero di campioni.

La scelta della funzione di perdita **cross-entropy**, combinata con la funzione di attivazione **softmax** in output, risulta particolarmente vantaggioso rispetto ad altre funzioni di perdita, come ad esempio il **Mean Squared Error (MSE)**. Questo perché la cross-entropy assegna penalizzazioni proporzionali all'entità dell'errore, rendendo gli aggiornamenti dei pesi più efficaci e favorendo un apprendimento più rapido.

Al contrario, l'MSE tende a generare gradienti molto piccoli quando le probabilità predette si avvicinano a 0 o 1, rallentando così il processo di apprendimento.

Per comprendere meglio questo comportamento, consideriamo uno scenario di previsione errata:

- **Previsione errata:** Supponiamo che la rete neurale stimi una probabilità $\hat{y} = a_{i,c} = 0.1$ per la classe corretta, mentre il valore corretto è $y = 1$. In questo caso:
  - Considerando la formula della **cross-entropy**, la perdita sarà circa 2.3. Questo valore deriva dal calcolo del logaritmo della probabilità predetta:

$$L_{CE} = -\log(0.1) \approx 2.3$$

  - Considerando la formula del **MSE**, la perdita sarà 0.81. Questo risultato si ottiene elevando al quadrato la differenza tra la probabilità predetta e il valore corretto:

$$L_{MSE} = (1 - 0.1)^2 = 0.81$$

Questi valori mostrano che la cross-entropy penalizza maggiormente le previsioni errate, spingendo il modello ad aggiornare i pesi in modo più significativo. Inoltre, genera **gradienti molto più grandi** quando la predizione è errata, facilitando un apprendimento più rapido rispetto all'MSE, che invece può risultare inefficace in questi casi.

Comunque, a questa funzione di loss verrà aggiunta una regolarizzazione L1, che penalizza la somma assoluta dei pesi della rete. Questo approccio tende a ridurre il valore dei pesi e, in molti casi, forza i pesi di connessioni meno rilevanti a essere esattamente pari a zero, eliminando di fatto tali connessioni dal modello. Riducendo il numero di pesi attivi, la regolarizzazione L1 semplifica il modello, contribuendo così a prevenire l'overfitting. Di seguito è riportata la formula:

$$J_{\text{reg}} = J + \lambda \left(\sum_{i,j} |W^1_{i,j}| + \sum_{k,l} |W^2_{k,l}| \right)$$

Dove λ è un parametro di regolarizzazione che determina il bilanciamento tra la funzione di loss originale e il termine di penalità della regolarizzazione L1. Esso controlla l'importanza relativa della semplificazione del modello rispetto all'accuratezza della previsione, permettendo di ridurre l'overfitting.

La presenza dello strato nascosto con funzione di attivazione **ReLU** rende comunque questa funzione **non convessa**, a causa della natura non lineare della ReLU. Di conseguenza, il passaggio della funzione di perdita presenterà numerosi **minimi locali**, che possono influenzare il processo di ottimizzazione.

La **funzione di perdita complessiva** J_{reg} (cross-entropy + L1 regularization), risulterà **non differenziabile** in corrispondenza di alcuni punti. In particolare, la regolarizzazione L1 introduce punti di **non differenziabilità** quando i pesi assumono valore nullo. In questi casi, ha senso parlare di **sottogradiente**, e per ottimizzare il modello sarà utile adottare metodi basati sul **sottogradiente**, che permettono di gestire efficacemente tali discontinuità.

Ottimizzare un modello significa determinare i valori ottimali dei parametri (in questo caso, i pesi) che minimizzano la funzione di perdita, indicata appunto come J_{reg}. Per ottenere questo risultato, si utilizza la **backpropagation**, un metodo fondamentale che calcola in modo efficiente i gradienti della funzione di perdita rispetto a tutti i parametri del modello. Questi gradienti vengono successivamente utilizzati per aggiornare i parametri tramite algoritmi di ottimizzazione, come la **discesa del gradiente**.

L'ottimizzazione si sviluppa in due fasi principali:

1. **Forward Pass**
   Durante il forward pass, l'input viene propagato attraverso la rete, layer per layer, fino a raggiungere lo strato di output. In ogni nodo, l'input viene trasformato combinando i pesi e i bias associati, seguito dall'applicazione di una funzione di attivazione appropriata. Per gli hidden layers, si utilizza comunemente la **ReLU**, mentre per lo strato di output si applica la **Softmax**. L'output del forward pass rappresenta le predizioni del modello, indicate come $\hat{y}$.

2. **Backward Pass**
   Il backward pass inizia calcolando il gradiente della funzione di perdita rispetto all'output del modello. Questo gradiente iniziale indica quanto piccole variazioni in ciascun output influenzano la perdita complessiva. Utilizzando la **regola della catena**, il gradiente viene propagato all'indietro attraverso la rete (da cui il termine "backpropagation"). In ogni layer, si calcolano i gradienti della perdita rispetto ai pesi e ai bias, consentendo di identificare come ciascun parametro contribuisce all'errore complessivo.

Per ogni peso w_{ij}, la **regola della catena** viene utilizzata per calcolare la derivata parziale della funzione di perdita rispetto a quel peso, secondo la seguente formula:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_k} \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{ij}}$$

Dove:

- J: rappresenta la funzione di perdita (loss).
- a_k: è l'attivazione del nodo k, ottenuta dopo l'applicazione della funzione di attivazione.
- z_k: è l'input lineare al nodo k, che dipende dal peso w_{ij}.

Quindi in altre parole questo formalismo descrive come si calcola il gradiente della perdita rispetto a un singolo peso, propagando i contributi attraverso i nodi del layer.

Con i gradienti calcolati per ogni parametro, i pesi e i bias vengono aggiornati nella direzione opposta al gradiente per ridurre la perdita, secondo la seguente formula:

$$w_i = w_{i-1} - \eta \cdot \nabla_w J$$

Dove η è il **tasso di apprendimento** che controlla appunto la velocità con cui il modello aggiorna i suoi parametri (pesi e bias) per ridurre la funzione di perdita.

2 Algoritmi

L'obiettivo principale di questo progetto è ottimizzare una rete neurale migliorando la precisione del modello tramite l'aggiustamento iterativo dei suoi parametri, in particolare i pesi. Questi ultimi sono fondamentali

poiché determinano la capacità della rete di apprendere dai dati di input e di modellare relazioni complesse per minimizzare la funzione di perdita.

Inizialmente, i pesi vengono assegnati in maniera casuale o seguendo specifiche distribuzioni probabilistiche. Questa scelta casuale è cruciale per rompere la simmetria tra i neuroni e garantire che la rete sia in grado di apprendere schemi complessi durante l'addestramento. Una cattiva inizializzazione potrebbe portare a problemi di convergenza o a soluzioni subottimali.

Durante l'addestramento, i pesi vengono aggiornati iterativamente in un processo noto come ottimizzazione, il cui scopo è minimizzare il valore della funzione di perdita. Questo processo si basa su tecniche che sfruttano gradienti o sottogradienti della funzione obiettivo, anche in casi in cui questa non sia completamente differenziabile.

Per garantire un'ottimizzazione efficace, in questo progetto vengono adottati due algoritmi principali: **l'heavy ball method** e **il deflected subgradient method**.

2.1 Heavy ball method

Il **Metodo Heavy Ball (HB)**, noto anche come *momentum classico*, è un algoritmo di ottimizzazione iterativo ideato per migliorare la velocità di convergenza rispetto al metodo di discesa del gradiente standard. Il principio alla base del metodo è l'introduzione di un termine di **momentum**, che stabilizza il processo di aggiornamento dei parametri e accelera il raggiungimento di un minimo [6].

Nel metodo HB, i parametri vengono aggiornati secondo la seguente regola iterativa:

$$x_{i+1} \leftarrow x_i - \alpha_i \nabla f(x_i) + \beta_i (x_i - x_{i-1})$$

dove:

- x_{i+1}: rappresenta i parametri aggiornati al passo $i + 1$;
- α_i: è il tasso di apprendimento (o *step size*);
- $\nabla f(x_i)$: è il gradiente della funzione di perdita calcolato rispetto ai parametri x_i;
- $\beta_i(x_i - x_{i-1})$: è il termine di *momentum*, che guida la direzione futura basandosi sul passo precedente.

Quando $\beta_i = 0$, il metodo HB si riduce al metodo standard di discesa del gradiente. Tuttavia, l'introduzione del momentum è fondamentale per migliorare la stabilità e l'efficienza dell'algoritmo, soprattutto nei problemi caratterizzati da superfici di perdita altamente non lineari o irregolari.

Questo algoritmo è particolarmente efficace in scenari in cui la **superficie di perdita** è **convessa e liscia**, poiché la presenza di un **gradiente ben definito e continuo** permette al termine di **momentum** di guidare l'ottimizzazione in modo più **stabile ed efficiente**.

Tuttavia, l'algoritmo può risultare meno efficace quando la funzione di perdita non è **differenziabile**, come nel caso del nostro modello, in cui la **regolarizzazione L1** introduce **discontinuità nei gradienti**, specialmente nei punti in cui i pesi diventano esattamente **zero**. In queste situazioni, l'**Heavy Ball (HB) method** potrebbe non comportarsi in modo ottimale, portando a **oscillazioni** o **stagnazioni** attorno ai punti di non differenziabilità.

Inoltre, poiché il metodo HB si basa sull'**accelerazione del gradiente** nelle iterazioni successive, la presenza di gradienti non definiti può **ostacolare la convergenza**, rendendolo meno efficace rispetto ad altri approcci più adatti a funzioni con regioni non differenziabili, come il **metodo del subgradiente deflesso**.

2.1.1 Proprietà di convergenza dell'HB

Il **metodo Heavy Ball (HB)** è particolarmente efficace nei problemi di ottimizzazione in cui la funzione obiettivo soddisfa specifiche proprietà matematiche che ne garantiscono una convergenza **rapida e stabile**. Tra le più rilevanti vi sono la **τ-convexity (convessità forte)** e la **L-smoothness (gradiente Lipschitz-continuo)**.

Prima di analizzare la funzione obiettivo specifica del nostro problema, definiamo le seguenti proprietà teoriche e il loro impatto sull'ottimizzazione:

1. **Convex**: Una funzione si dice **convessa** se soddisfa la seguente disuguaglianza per ogni coppia di punti x, y e per ogni $\alpha \in [0, 1]$:

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

   Questa proprietà implica che il segmento che congiunge due punti qualsiasi del grafico della funzione si trova sempre **sopra o coincidente** con il grafico stesso. Le funzioni convesse possiedono un **unico minimo globale**, rendendo il problema di ottimizzazione più semplice ed efficiente.

2. **τ-Convex**: Una funzione $f(x)$ è fortemente convessa con parametro $\tau > 0$ se soddisfa la seguente disuguaglianza:

   $$\alpha f(x) + (1-\alpha)f(y) \geq f\big(\alpha x + (1-\alpha)y\big) + \frac{\tau}{2}\alpha(1-\alpha)\|y-x\|^2$$

   Questa proprietà assicura che la funzione abbia una **curvatura sufficiente**, evitando che il minimo sia piatto, ma invece ben definito. Una funzione **fortemente convessa** facilita la **convergenza dell'ottimizzazione**, riducendo il rischio di regioni in cui il gradiente è quasi nullo per lunghi tratti, migliorando così l'efficacia degli algoritmi di discesa del gradiente.

3. **Smoothness**: Una funzione è detta **smooth** se il suo gradiente è **definito ovunque** e **continuo**. In termini matematici, ciò implica che la funzione sia **differenziabile in tutti i punti** e che la sua derivata non presenti **discontinuità**.

4. **L-smooth** (*L-Smoothness*): Una funzione $f(x)$ è L-smooth se il gradiente della funzione è lipschitziano, cioè:

   $$\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\|$$

   Questa proprietà garantisce che il gradiente della funzione non cambi troppo rapidamente, rendendo il processo di ottimizzazione più prevedibile e meno soggetto a oscillazioni. [2]

Per verificare se la nostra funzione obiettivo soddisfa queste condizioni, è necessario analizzarle singolarmente ed esaminarle una per una.

La **funzione obiettivo** (cross-entropy + regolarizzazione L1) è **convessa**, poiché:

- La **cross-entropy** è convessa rispetto ai logit della rete neurale.
- La **regolarizzazione L1** è convessa, in quanto la funzione $|w|$ è convessa.
- La somma di due funzioni convesse è anch'essa convessa.

Tuttavia, la **loss function** è convessa solo rispetto ai logit, ma **non** rispetto ai **pesi della rete neurale**, poiché le attivazioni **ReLU** negli hidden layer introducono **non linearità** e **minimi locali**.

La funzione obiettivo **non è fortemente convessa**, perché:

- La **cross-entropy** non soddisfa le condizioni di τ**-convexity**.
- La **regolarizzazione L1** non è fortemente convessa, in quanto la funzione valore assoluto **non** ha curvatura quadratica ovunque.

Poiché entrambe le componenti della funzione obiettivo **non** sono fortemente convesse, la funzione complessiva **non può essere fortemente convessa**.

Inoltre, la funzione obiettivo **non è smooth**, perché:

- La **cross-entropy** è differenziabile ovunque.
- La **regolarizzazione L1**, invece, **non è differenziabile nei punti in cui i pesi sono pari a zero**.

Di conseguenza, la funzione **non soddisfa la condizione di smoothness**, e quindi **non è L-smooth**, poiché la L-smoothness richiede la continuità e la limitatezza del gradiente.

Dato che la nostra funzione obiettivo **non soddisfa** né le condizioni di τ**-convexity** né quelle di **L-smoothness**, l'implementazione dell'algoritmo **Heavy Ball (HB)** potrebbe **non essere la scelta ottimale** per l'ottimizzazione dei pesi della rete neurale.

2.1.2 Velocità di convergenza dell'HB

Il metodo Heavy Ball (HB) raggiunge un tasso di convergenza ottimale sotto le ipotesi di τ**-convex** e **L-smooth** (*L-smoothness*). Il tasso di riduzione dell'errore euclideo è dato dalla formula:

$$r = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$$

dove $\kappa = \frac{L}{\tau}$ rappresenta il *numero di condizionamento* della funzione obiettivo. Un numero di condizionamento più basso (κ vicino a 1) indica una funzione più facile da ottimizzare, mentre valori elevati di κ suggeriscono una funzione più complessa e mal condizionata.

Nella pratica, il metodo HB richiede la scelta di due iperparametri chiave: il *tasso di apprendimento* (α_i) e il *coefficiente di momentum* (β_i). Il tasso di apprendimento controlla l'ampiezza del passo lungo la direzione del gradiente. Un valore comunemente utilizzato, che bilancia stabilità e velocità di convergenza, è dato da:

$$\alpha = \frac{4}{(\sqrt{L} + \sqrt{\tau})^2}$$

Il coefficiente di momentum, invece, influenza quanto il passo precedente contribuisce all'aggiornamento corrente. Un valore tipico è:

$$\beta = \left(\frac{\sqrt{L} - \sqrt{\tau}}{\sqrt{L} + \sqrt{\tau}} \right)^2$$

Questi valori di α e β sono progettati per garantire che l'algoritmo raggiunga il tasso di convergenza ottimale, massimizzando l'efficienza e la stabilità del processo di ottimizzazione. Grazie a queste scelte, il metodo HB si dimostra particolarmente efficace per problemi con funzioni obiettivo che soddisfano le condizioni teoriche di τ-convex e L-smooth.

Nel nostro caso specifico, la funzione di perdita **Cross-Entropy + Regolarizzazione L1 non** soddisfa né la **L-smoothness** né la τ**-convexity**, con un impatto diretto sulla **velocità di convergenza**.

Poiché la funzione **non è fortemente convessa**, **non** possiamo garantire una **contrazione esponenziale dell'errore**, poiché il parametro τ sarà **nullo o molto piccolo**. Di conseguenza, la **convergenza sarà sub-lineare**.

Inoltre, l'assenza di **L-smoothness** comporta oscillazioni più marcate, rendendo il **passo di aggiornamento (stepsize)** potenzialmente **troppo aggressivo in alcune regioni e inefficace in altre**.

In assenza di **forte convessità**, la velocità di convergenza si riduce a $\mathcal{O}(1/k)$. Mentre, in assenza di **L-smoothness**, la convergenza può diventare **irregolare e oscillante**, e in alcuni casi può risultare **più lenta** rispetto al metodo del gradiente standard.

Dato che **non possiamo applicare direttamente le formule teoriche precedenti**, è necessario adattare i parametri α (learning rate) e β (momentum) per evitare instabilità [1]:

- **Learning rate (α)**: Deve essere più **conservativo** rispetto al caso ideale, poiché la funzione **non è liscia**. Un valore empirico tipico è $\alpha \approx 1/L$, dove L è una stima massima della derivata seconda della funzione di perdita.
- **Momentum (β)**: In assenza di τ-convexity e L-smoothness, un valore di β **troppo alto** può amplificare le oscillazioni.

2.2 Deflected subgradient method

Il **Metodo del Subgradiente Deflesso** (*Deflected Subgradient Method*) rappresenta un'evoluzione avanzata del metodo subgradiente standard. Questo approccio è progettato per migliorare l'efficienza e la velocità di convergenza nei problemi di ottimizzazione caratterizzati da funzioni non differenziabili o con geometrie altamente irregolari. La caratteristica distintiva del metodo è l'uso di una direzione deflessa, ottenuta combinando il subgradiente corrente con le informazioni derivanti dalle iterazioni precedenti.

Questo approccio aiuta a **mitigare le variazioni brusche del gradiente** e a **ridurre le oscillazioni** che possono verificarsi in presenza di funzioni **non lisce**, migliorando così la **stabilità dell'ottimizzazione**.

2.2.1 Metodo del Subgradiente Standard

Per comprendere il **Metodo del Subgradiente Deflesso**, è utile iniziare dalla formula di aggiornamento del metodo subgradiente standard:

$$x_{i+1} = x_i - \alpha_i g_i$$

dove:

- x_i: rappresenta il punto corrente nel dominio della funzione obiettivo;
- g_i: è un subgradiente della funzione $f(x)$ calcolato in x_i;
- α_i: è la dimensione del passo (*stepsize*) utilizzata per l'aggiornamento.

Poiché la **funzione obiettivo** non è differenziabile ovunque, g_i **non** rappresenta un gradiente classico, ma appartiene all'insieme dei **sottogradienti** della funzione. Il metodo standard procede lungo la direzione **opposta** a g_i, cercando di **minimizzare** la funzione obiettivo.

Tuttavia, il **metodo del subgradiente standard** può soffrire di **oscillazioni significative** e di una **convergenza lenta**, specialmente in problemi con funzioni **non differenziabili** o **altamente irregolari**. Per superare queste difficoltà, è stato sviluppato il **Metodo del Subgradiente Deflesso (DSG)**.

2.2.2 Metodo del subgradiente Deflesso

Nel **Metodo del Subgradiente Deflesso**, la direzione di aggiornamento viene modificata introducendo un termine di deflessione, definito come:

$$d_i = \gamma_i g_i + (1 - \gamma_i) d_{i-1}$$

dove:

- d_i: rappresenta la direzione deflessa all'iterazione i;
- γ_i: è il parametro di deflessione che bilancia il contributo del subgradiente corrente g_i rispetto alla direzione precedente d_{i-1}.

L'aggiornamento dei parametri diventa quindi:

$$x_{i+1} = x_i - \alpha_i d_i$$

Questa combinazione lineare tra il subgradiente corrente e la direzione precedente migliora la stabilità del metodo, riducendo le oscillazioni che spesso emergono nei problemi non differenziabili o caratterizzati da vincoli complessi.

Il parametro γ_i regola il peso tra il gradiente corrente (g_i) e la direzione precedente (d_{i-1}):

- Se $\gamma_i = 1$, il metodo si riduce al metodo del subgradient standard;
- Se $\gamma_i < 1$, aumenta l'influenza della direzione precedente, favorendo una traiettoria di ottimizzazione più stabile.

Questa flessibilità rende il metodo del subgradiente deflesso particolarmente utile per problemi difficili, garantendo una maggiore robustezza e una convergenza più efficiente.

Per quanto riguarda la scelta dello **stepsize**, un aspetto cruciale poiché determina sia la **velocità di convergenza** sia la **stabilità** dell'algoritmo, esistono diverse strategie in base alle specifiche **condizioni di applicabilità**:

- **Step size costante**: Se il problema è **ben condizionato** e la funzione è τ**-convex** e **L-smooth**, è possibile utilizzare uno **step size costante**:

  $$\alpha = c$$

- **Step size secondo la regola di Polyak**: Se il valore ottimale f^* è noto o può essere stimato con buona precisione, lo **step size di Polyak** è efficace nei problemi **convessi** e **non differenziabili**:

  $$\alpha_i = \frac{f(x_i) - f^*}{\|g_i\|_2}$$

- **Step size decrescente nel tempo (Diminishing Step Size)**: Utilizzato in problemi **non differenziabili** e **convessi**, garantisce la **convergenza** se soddisfa le seguenti condizioni:

  $$\sum \alpha_i = \infty, \quad \sum \alpha_i^2 < \infty$$

  Lo **step size decrescente** è definito come:

  $$\alpha_i = \frac{c}{i}$$

  dove c è una costante positiva e i rappresenta il numero di iterazioni.

Poiché la nostra funzione obiettivo **non è smooth** e **non è fortemente convessa**, la scelta più adatta è lo **step size decrescente (Diminishing Step Size)**, poiché evita che gli aggiornamenti diventino **troppo aggressivi**, a differenza di uno **step size costante**.

Un'alternativa valida sarebbe lo **step size secondo Polyak**, a condizione che il valore ottimale f^* possa essere **stimato con precisione**.

2.2.3 Convergenza del modello SDG

Il **Metodo del Subgradiente Deflesso (DSG)** è progettato per affrontare problemi di ottimizzazione **non differenziabili** e **complessi**, migliorando la **convergenza** rispetto al metodo del **subgradiente standard**. Tuttavia, la sua efficacia dipende da specifiche **proprietà** della **funzione obiettivo** e dalla **scelta appropriata** dei **parametri**.

Il metodo **garantisce la convergenza** se soddisfa le seguenti condizioni [5]:

- **Diminishing Step Size Condition**: Come spiegato in precedenza, questa condizione assicura che lo **step size** sia inizialmente abbastanza grande da **esplorare lo spazio dei parametri**, ma **decresca rapidamente** per evitare oscillazioni indefinite. La condizione è la seguente:

$$\sum \alpha_i = \infty, \quad \sum \alpha_i^2 < \infty$$

- **Funzione convessa o pseudo-convessa**:
  - Se la funzione obiettivo è **convessa**, l'algoritmo **DSG** garantisce la **convergenza all'ottimo globale**.
  - Se la funzione è **pseudo-convessa**, DSG **converge** a un **punto critico** della funzione, che **potrebbe non essere un minimo globale**.
- **Limitatezza del sottogradiente**: Se il sottogradiente g_i è **limitato** ($\|g_i\| \leq G$ con $G > 0$), il metodo evita **aggiornamenti troppo grandi**, riducendo così il rischio di **instabilità**.
- **Scelta del parametro γ_i**:
  - Se $\gamma_i \to 0$ quando $i \to \infty$, la direzione di aggiornamento **si riduce** al **sottogradiente standard**, garantendo la convergenza nelle condizioni standard.
  - Se γ_i è **troppo grande** (ad esempio, $\gamma_i \approx 1$), il metodo potrebbe **non convergere**, poiché il peso delle direzioni precedenti potrebbe impedire un **adattamento efficace** alla traiettoria dell'ottimizzazione.

Nel nostro caso, considerando la **funzione obiettivo Cross-Entropy + Regolarizzazione L1**, la **condizione di step size decrescente** può essere soddisfatta scegliendo uno **step size che diminuisce nel tempo**.

Tuttavia, **la convergenza globale non è garantita**, poiché la nostra funzione **non è convessa**.

Inoltre, il **sottogradiente della funzione non è limitato** perchè, sebbene il **sottogradiente della Cross-Entropy** è **limitato**, in quanto la **Softmax** ha un gradiente contenuto, il **sottogradiente della regolarizzazione L1**, invece, **non è limitato**, a causa della **discontinuità in** $w = 0$ (dove cambia bruscamente da $-\lambda$ a $+\lambda$). Di conseguenza, il **DSG potrebbe soffrire di instabilità**.

Per garantire una maggiore **stabilità**, è consigliabile scegliere un γ_i **moderato** oppure un γ_i **decrescente**. [4]

2.2.4 Velocità di convergenza del modello SDG

La **velocità di convergenza** del **Metodo del Subgradiente Deflesso (DSG)** dipende fortemente dalla **convessità** della **funzione obiettivo** e dalla **scelta dello step size** α.

- **Se la funzione obiettivo $f(x)$ è τ-convex**, il metodo **DSG** ha un **tasso di convergenza** dato da:

  $$f(x_k) - f(x^*) = O(1/k)$$

  con **convergenza sub-lineare**, ma **più veloce** rispetto al **metodo del subgradiente standard**.

- **Se $f(x)$ è convessa ma non fortemente convessa**, DSG raggiunge un **tasso di convergenza più lento**:

  $$f(x_k) - f(x^*) = O(1/\sqrt{k})$$

  Questo implica una **convergenza più lenta** rispetto ai problemi **fortemente convessi**. Tuttavia, **DSG rimane vantaggioso** rispetto al **subgradiente standard**, grazie al **termine di deflessione**, che **riduce la variabilità degli aggiornamenti**.

- **Se la funzione da ottimizzare non è convessa**, **non esiste una garanzia formale** che il metodo DSG raggiunga il valore ottimale. Tuttavia, in alcune situazioni, può comunque convergere a un punto in cui le **variazioni della funzione sono minime**, spesso corrispondente a un **minimo locale**. Se la funzione presenta **molte irregolarità** e **minimi locali**, DSG potrebbe **fermarsi in una soluzione subottimale**, senza raggiungere necessariamente il miglior minimo possibile. [3]

Nel nostro caso, con la funzione **Cross-Entropy + Regolarizzazione L1**, la funzione **non è fortemente convessa** (a causa della regolarizzazione L1), quindi la **velocità di convergenza più probabile** è:

$$O(1/\sqrt{k})$$

3 Esperimenti

In questa sezione vengono illustrati i risultati delle implementazioni degli algoritmi *Heavy Ball* e *Deflected Subgradient*, applicati sia a funzioni matematiche standard che a al modello di rete neurale costruito utilizzando il dataset oggetto dell'analisi.

Per validare le implementazioni, gli algoritmi sono stati testati su due funzioni con caratteristiche differenti:

- **Funzione convessa**: La funzione di Matyas, definita come:

  $$f(x, y) = 0.26 \cdot (x^2 + y^2) - 0.48 \cdot x \cdot y$$

  Questa funzione è caratterizzata da un minimo globale ben definito in $(0, 0)$, dove il valore è $f(0, 0) = 0$. La sua forma è simmetrica e rappresenta un paraboloide le cui linee di livello sono ellissi concentriche. La funzione di Matyas è comunemente utilizzata per valutare la convergenza e la stabilità degli algoritmi di ottimizzazione.

- **Funzione non convessa**: La funzione di Himmelblau, definita come:

  $$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

  Questa funzione è caratterizzata dalla presenza di quattro minimi globali situati in: $(3, 2)$, $(-2.805, 3.131)$, $(-3.779, -3.283)$ e $(3.584, -1.848)$, tutti con valore $f(x, y) = 0$.

I principali parametri utilizzati per valutare le prestazioni degli algoritmi sono:

- **Valore della funzione**: Rappresenta una misura diretta dell'efficacia dell'ottimizzazione, ovvero il valore finale della funzione obiettivo raggiunto al termine delle iterazioni. Ad esempio, nel caso di una rete neurale, questo parametro corrisponde alla *loss* e indica la qualità della soluzione trovata. È importante notare che il valore finale potrebbe non coincidere con il minimo assoluto raggiunto durante il processo, specialmente in presenza di instabilità.

- **Instabilità**: Indica il grado di instabilità dell'algoritmo, misurato attraverso le variazioni relative del valore della funzione nel corso delle iterazioni. La formula utilizzata per calcolare l'instabilità è la seguente:

$$\text{unst}(x) = \frac{10^5}{|x|} \sum_{i=2}^{|x|} \max\left(0, \frac{x[i] - x[i-1]}{x[i-1]}\right)$$

  dove $x[i]$ rappresenta il valore della funzione alla i-esima iterazione e $|x|$ è il numero totale di iterazioni. Questo parametro normalizza gli incrementi relativi rispetto al numero di iterazioni, fornendo un'indicazione di quanto l'algoritmo sia soggetto a fluttuazioni indesiderate durante il processo di ottimizzazione. Tuttavia, si tratta di una misura euristica che presenta alcune limitazioni, come l'attribuzione dello stesso peso agli incrementi iniziali e a quelli finali, che potrebbero avere significati diversi nel contesto del processo di convergenza.

- **Velocità di convergenza**: Questa metrica quantifica il numero di iterazioni necessarie affinché l'algoritmo raggiunga, per la prima volta, un valore compreso entro un determinato intervallo rispetto al valore ottimale finale ottenuto. Il punteggio varia da 0 a 100, dove 100 indica che il valore della funzione era già all'interno dell'intervallo prestabilito sin dalla prima iterazione. Tuttavia, questa metrica da sola non è sufficiente per valutare le prestazioni complessive dell'algoritmo e deve essere considerata insieme al valore finale raggiunto.

$$cs(x) = 100 - \frac{\arg\max_i \left(x[i] \leq \min(x) \cdot \left(1 + \frac{\rho}{100}\right)\right) \cdot 100}{|x|}$$

  dove $\rho \geq 0$ rappresenta la tolleranza relativa al valore finale ottimale raggiunto. Un valore di velocità di convergenza pari a 100 indica che l'algoritmo ha raggiunto il range desiderato già alla prima iterazione. Questo parametro deve essere valutato insieme al valore finale raggiunto per ottenere un quadro completo delle prestazioni dell'algoritmo.

  L'esecuzione degli algoritmi è stata interrotta in base a uno dei seguenti criteri di arresto:

  1. **Norma del gradiente/subgradient**: L'algoritmo si arresta quando la norma del gradiente o del subgradiente scende al di sotto di una soglia predefinita ϵ, indicando che è stata trovata una soluzione ottimale.
  2. **Numero massimo di iterazioni**: L'esecuzione termina se viene raggiunto il limite massimo di iterazioni consentite, identificando una *"stopped solution"*.
  3. **Assenza di decrescita significativa**:L'algoritmo si arresta quando, per 10 iterazioni consecutive, la differenza tra il valore della funzione alla i-esima iterazione e quello alla $(i-1)$-esima iterazione è inferiore a ϵ, indicando che ulteriori miglioramenti sono trascurabili.
  4. **Instabilità prolungata dell'algoritmo**: Se per 20 iterazioni consecutive l'instabilità dell'algoritmo supera una soglia critica predefinita (adottando un criterio conservativo, ad esempio valore > 1000), l'ottimizzazione viene interrotta per evitare di sprecare risorse computazionali su una configurazione potenzialmente inefficace.

Da sottolineare che un'elevata instabilità non implica necessariamente che l'algoritmo non possa convergere, ma indica che la traiettoria dell'ottimizzazione è irregolare. Questo suggerisce che, con i parametri attuali, la configurazione potrebbe non essere ottimale in termini di efficienza. Per questo motivo, è preferibile interrompere il processo ed escludere tale configurazione, evitando di impiegare tempo e risorse computazionali su una soluzione potenzialmente meno stabile o efficace rispetto ad altre alternative.

Nella **Sezione 3.1** saranno eseguiti test di convergenza in cui i risultati degli algoritmi saranno analizzati utilizzando le funzioni di Matyas e Himmelblau, con l'obiettivo di valutarne la capacità di convergenza.

Nella **Sezione 3.2**, gli algoritmi saranno applicati a una rete neurale, approfondendo l'impatto della variazione dei parametri sul processo di ottimizzazione e sulle prestazioni complessive.

3.1 Test della convergenza

In questa sezione vengono analizzati i risultati delle implementazioni degli algoritmi **Heavy Ball** e **Deflected Subgradient Method**, applicati rispettivamente alle funzioni di **Matyas** (convessa) e di **Himmelblau** (non convessa). Gli esperimenti sono stati eseguiti partendo dal punto iniziale $(4, 4)$ e includendo una fase preliminare di **grid search** per ottimizzare i parametri principali. Le configurazioni esplorate sono state le seguenti:

- **Alpha** (α): $[0.01, 0.02, 0.05, 0.07, 0.1, 0.12, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 1.0]$
- **Momentum** (per Heavy Ball): $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$
- **Gamma** (γ, per Deflected Subgradient): $[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$

Di seguito vengono riportati gli pseudocodici degli algoritmi utilizzati nello studio:

Algorithm 1 Heavy Ball

$w_0 \leftarrow$ inizializza pesi
$m_0 \leftarrow 0$ `// Momento iniziale`
Imposta α (learning rate), μ (coefficiente di momentum), ε (tolleranza), max_iter
$i \leftarrow 0$
while $i <$ max_iter **do**
  $g_i \leftarrow \nabla f(w_i)$ `// Calcola gradiente`
  Calcola $\|g_i\|$ `// Norma del gradiente`
  Calcola instabilità su storico delle perdite
  Calcola velocità di convergenza
  $m_i \leftarrow \mu \cdot m_{i-1} - \alpha \cdot g_i$ `// Aggiorna momento`
  $w_{i+1} \leftarrow w_i + m_i$ `// Aggiorna pesi`
  if $\|g_i\| < \varepsilon$ **then**
    break
  if variazione media ultime 10 `losses` $< \varepsilon$ **then**
    break
  if instabilità media delle ultime `losses` > 1000 **then**
    break
  $i \leftarrow i + 1$
end

Algorithm 2 Deflected Subgradient

Inizializza pesi w_0, direzione precedente $d_0 \leftarrow 0$
Imposta α (iniziale), γ (coeff. deflessione), ε, `max_iter`
$i \leftarrow 1$
while $i \leq$ `max_iter` **do**
  Calcola subgradiente $g_i \in \partial f(w_i)$
  Calcola norma $\|g_i\|$
  Calcola instabilità sulla storia delle perdite
  Calcola velocità di convergenza
  Deflessione: $d_i \leftarrow \gamma \cdot g_i + (1 - \gamma) \cdot d_{i-1}$
  Step size decrescente: $\alpha_i = \alpha / i$
  Aggiorna pesi: $w_{i+1} \leftarrow w_i - \alpha_i \cdot d_i$
  if $\|g_i\| < \varepsilon$ **then**
    break
  if variazione media ultime 10 `losses` $< \varepsilon$ **then**
    break
  if instabilità media degli ultimi valori in `losses` > 1000 **then**
    break
  $i \leftarrow i + 1$
end

Sono state valutate numerose configurazioni attraverso la *grid search*. Grazie alla bassa complessità computazionale, è stato possibile analizzare i risultati di tutte le combinazioni per studiare il comportamento degli algoritmi al variare dei parametri.

Sono state testate diverse configurazioni e, per garantire una maggiore accuratezza evitando che l'algoritmo si arrestasse dopo una sola iterazione, è stato scelto un valore di ϵ pari a 1e-12

Per il metodo **Deflected Subgradient**, è stato preferito un passo costante rispetto a un passo **DSS** (Diminishing Step Size) o **Polyak** per le seguenti ragioni:

- La funzione di **Matyas** è τ-convex e **L-smooth**, quindi un passo costante è sufficiente per garantire un buon tasso di convergenza.
- Sebbene la funzione di **Himmelblau** non sia τ-convex, è stato comunque adottato un passo costante. Infatti, empiricamente si è osservato che l'utilizzo di un passo **DSS** comportava tempi di convergenza superiori alle **1000 iterazioni**. Questo fenomeno è dovuto al decremento progressivo di α a ogni iterazione, un problema noto come il **"problema di Scylla"**, che può rallentare significativamente l'ottimizzazione.

  I risultati ottenuti evidenziano differenze significative tra le due funzioni:

  - **Funzione di Matyas**: L'uso di un passo α grande risulta generalmente più efficiente. All'aumentare di α, il numero di iterazioni necessarie per raggiungere l'ottimo diminuisce. Tuttavia, valori di α eccessivamente elevati possono rendere l'algoritmo instabile, causando oscillazioni o divergenza.
  - **Funzione di Himmelblau**: A differenza della funzione di Matyas, la funzione di Himmelblau converge più facilmente con passi α piccoli. Inoltre, il metodo è in grado di individuare rapidamente uno dei minimi globali, tipicamente il punto $(3, 2)$.

– **Instabilità nelle configurazioni di Heavy Ball**: In entrambe le funzioni (soprattutto nel caso della funzione di Matyas), molte configurazioni testate con il metodo **Heavy Ball** si sono dimostrate efficaci nel raggiungere il minimo globale. Tuttavia, numerose combinazioni sono state scartate a causa dell'elevata instabilità, che ha reso il processo di ottimizzazione meno affidabile.

Le migliori configurazioni trovate si possono trovare nelle seguenti tabelle 1 e 2:

Algoritmo	**Parametri ottimali**	**Risultati principali**
Heavy Ball	$\alpha = 1$, $\beta = 0.6$	Iterazioni: 104 Instabilità: 0 Velocità di convergenza: 89.7 Loss finale: 0 Tempo di esecuzione: 0.06
Deflected Subgradient Method	$\alpha = 1$, $\gamma = 0.2$	Iterazioni: 11 Instabilità: 0 Velocità di convergenza: 99.0 Loss finale: 0 Tempo di esecuzione: 0.00

Tabella 1: Parametri ottimali applicati alla funzione di Matyas

Algoritmo	**Parametri ottimali**	**Risultati principali**
Heavy Ball	$\alpha = 0.01$, $\beta = 0.1$	Iterazioni: 46 Instabilità: 0 Velocità di convergenza: 95.5 Loss finale: 0 Tempo di esecuzione: 0.02
Deflected Subgradient Method	$\alpha = 0.02$, $\gamma = 0.9$	Iterazioni: 11 Instabilità: 0 Velocità di convergenza: 99.0 Loss finale: 0 Tempo di esecuzione: 0.00

Tabella 2: Parametri ottimali applicati alla funzione di Himmelblau

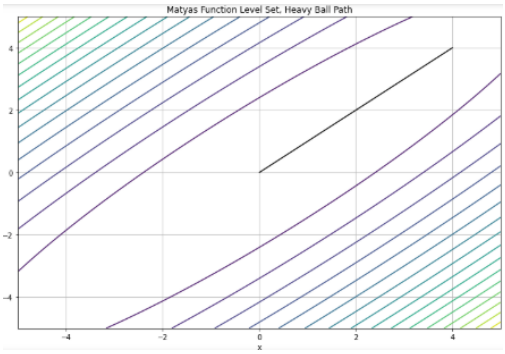
Dalle tabelle si osserva che, con una scelta ottimale dei parametri, il metodo **subgradiente deflesso** (con passo costante) risulta significativamente più efficiente rispetto al metodo **Heavy Ball**. Questo vantaggio è evidente in termini di **tempo di esecuzione**, **velocità di convergenza** e **numero di iterazioni** necessarie per raggiungere l'ottimo, che risultano sensibilmente inferiori rispetto al metodo **Heavy Ball**.

Per quanto riguarda la **convergenza** dei due algoritmi applicati alla **funzione di Matyas**, entrambi riescono a raggiungere il punto di ottimo $(0, 0)$ seguendo una traiettoria regolare. Tuttavia, il metodo **Heavy Ball** raggiunge il minimo con una precisione di $\epsilon = 1e{-}12$ in **104 iterazioni**, mentre il metodo **DSG** (Deflected Subgradient) impiega soltanto **11 iterazioni** per raggiungere lo stesso punto, dimostrando una maggiore efficienza in termini di velocità di convergenza.

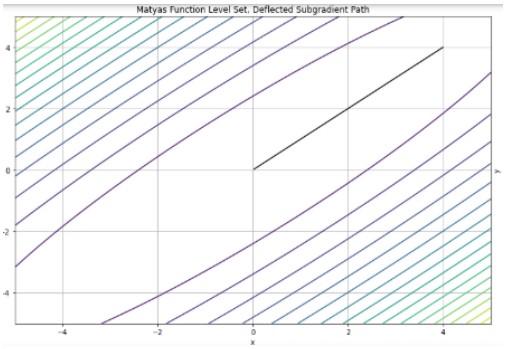
Nella figura 2 è mostrata la convergenza al punto di minimo dei due algoritmi nella funzione di Matyas.

Per quanto riguarda la **convergenza** dei due algoritmi applicati alla **funzione di Himmelblau**, il metodo **Heavy Ball (HB)** segue un percorso più diretto, sebbene leggermente smorzato a causa dell'effetto del momento, raggiungendo la convergenza in **46 iterazioni**.

Il metodo **Deflected Subgradient (DSG)**, invece, segue una traiettoria più irregolare, caratterizzata da angoli più pronunciati a causa dell'elevato parametro di deflessione. Nonostante il suo percorso possa



(a) Heavy Ball

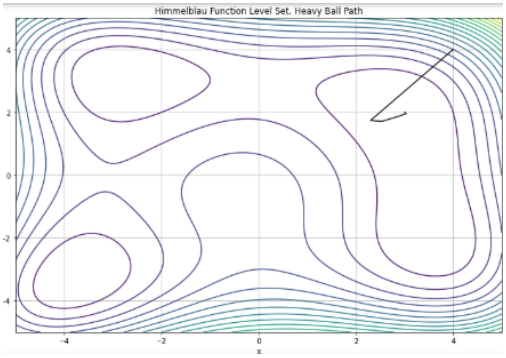


(b) Deflected Subgradient

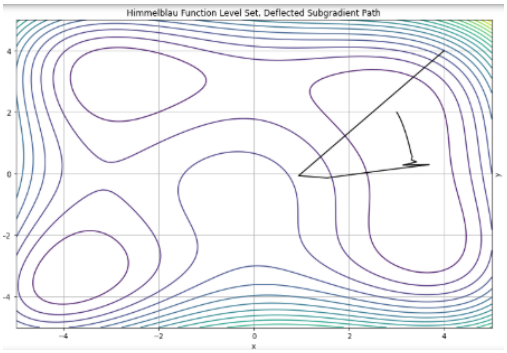
Figura 2: Matyas functions

sembrare più ampio, il DSG riesce a convergere in sole **11 iterazioni**, dimostrando, anche in questo caso, una maggiore **efficienza in termini di velocità di convergenza**.

In figura 3è mostrata la convergenza al punto di minimo (3,2) dei due algoritmi nella funzione di Himmelbalu.



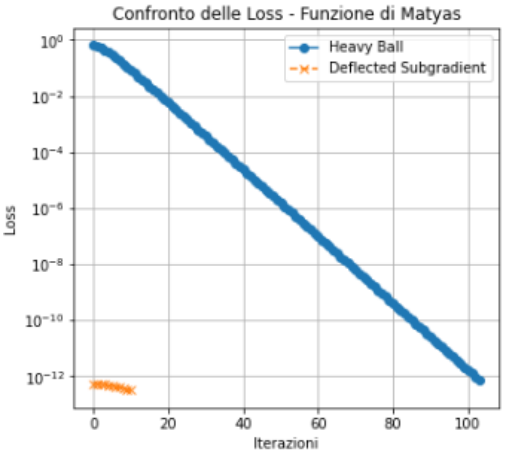
(a) Heavy Ball



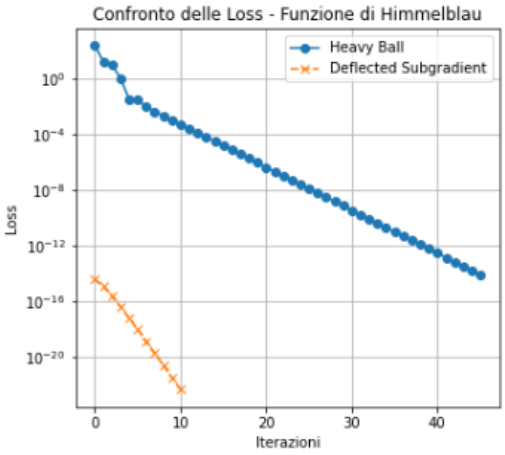
(b) Deflected Subgradient

Figura 3: Himmelblau functions

Infine, in figura 4 è possibile osservare un confronto tra la **velocità di convergenza** dei metodi **Heavy Ball** e **Deflected Subgradient** applicati alle funzioni di **Matyas** e **Himmelblau**. L'analisi viene effettuata valutando il **valore della funzione** in **scala logaritmica** rispetto al **numero di iterazioni**.



(a) Funzione di Matyas



(b) Funzione di Himmelblau

Figura 4: Velocità di convergenza

3.2 Rete Neurale

Dopo aver implementato e testato gli algoritmi di ottimizzazione su funzioni matematiche standard, abbiamo applicato i metodi **Heavy Ball (HB)** e **Deflected Subgradient Method (DSG)** a una rete neurale sviluppata in **Python**.

Il **dataset** utilizzato e la **struttura della rete neurale** sono descritti rispettivamente nelle **sezioni 1.1 e 1.1.1**.

Per garantire una corretta propagazione dei gradienti e stabilizzare il processo di addestramento, la rete neurale è stata inizializzata utilizzando il metodo **He Initialization** (noto anche come **Kaiming Initialization**), specificamente progettato per funzioni di attivazione **ReLU**.

La formula di **He Initialization** adottata è:

$$W \sim \mathcal{N}(0, \frac{2}{\text{fan\_in}})$$

dove:

- **fan_in** è il numero di neuroni in ingresso nel layer,
- i pesi sono inizializzati con una distribuzione normale a **media 0** e **varianza** $2/$**fan_in**,
- i **bias** sono stati inizializzati a **0**.

Questa scelta è stata adottata per prevenire il problema del **vanishing gradient**, migliorando la stabilità dell'ottimizzazione.

Per il **Deflected Subgradient Method (DSG)**, è stato utilizzato un **passo DSS** (Diminishing Step-Size) invece di un passo costante. Come discusso nella **sezione 2.2.2**, questa scelta è motivata dal fatto che le reti neurali non sono necessariamente τ-convesse, rendendo l'uso di un passo costante potenzialmente instabile. L'adozione di un **passo DSS** consente di adattare dinamicamente il learning rate, evitando oscillazioni indesiderate.

L'analisi è stata suddivisa in tre sezioni distinte, ognuna dedicata a un diverso valore del parametro di regolarizzazione λ (0.01, 0.1, 1.0). Questa scelta è motivata dall'esigenza di comprendere l'effetto della regolarizzazione L1 sulle prestazioni dei due algoritmi e individuare le configurazioni ottimali per ciascun valore di λ.

Per ogni algoritmo, verrà eseguita una *grid search* mantenendo fisso il parametro di regolarizzazione λ. Verranno considerate tutte le combinazioni possibili dei parametri α e β (o γ), al fine di determinare la configurazione che fornisce i migliori risultati in termini di prestazioni.

Per i parametri di ottimizzazione, sono stati selezionati i seguenti valori:

- α **(learning rate)**:
  - $[1, 2, 5, 10, 15, 20, 40]$ (nei casi del **Deflected Subgradient** con $\lambda = 0.01$ e $\lambda = 0.1$)
  - $[0.01, 0.05, 0.1, 0.2, 0.5, 0.75, 1]$ (in tutti gli altri casi)
- μ **(fattore di momento, solo per HB)**: $[0.1, 0.3, 0.5, 0.7, 0.9]$
- γ **(fattore di deflessione, solo per DSG)**: $[0.1, 0.3, 0.5, 0.7, 0.9]$

Nel caso del metodo **Deflected Subgradient (DSG)** con $\lambda = 0.01$ e $\lambda = 0.1$, si è scelto di utilizzare un **range di valori di α (stepsize) maggiori o uguali a 1**, applicando il **passo Diminishing Step-Size (DSS)**.

Questa decisione è stata presa sulla base di osservazioni empiriche, che hanno mostrato come valori di $\alpha < 1$ portassero a un **miglioramento inefficiente**, con una **convergenza eccessivamente lenta**.

L'esplorazione dei parametri è stata effettuata attraverso una **Grid Search**, valutando il comportamento degli algoritmi in termini di:

- **Valore funzione obiettivo**
- **Velocità di convergenza**
- **Instabilità**

Il numero massimo di iterazioni è stato fissato a 1000, mentre il criterio di arresto è stato impostato con un $\epsilon = 10e^{-4}$. Questa impostazione garantisce l'esecuzione della *grid search* in tempi ragionevoli per entrambi gli algoritmi, mantenendo al contempo un livello adeguato di accuratezza.

3.2.1 Analisi con $\lambda = 0.01$

Sono stati inizialmente testati gli algoritmi **Heavy Ball** e **Deflected Subgradient** utilizzando un valore di regolarizzazione molto basso ($\lambda = 0.01$). Questa scelta è stata effettuata per ridurre al minimo l'effetto della regolarizzazione e valutare le capacità di ottimizzazione pura di entrambi i metodi.

Dai risultati ottenuti, l'algoritmo **Heavy Ball** ha mostrato un comportamento relativamente stabile. I valori di α compresi tra **0.1** e **0.2** tendono a favorire una convergenza più rapida, mentre valori più elevati di μ (compresi tra **0.7** e **0.9**) sembrano migliorare ulteriormente la velocità di convergenza. Tuttavia, valori di α più alti (come $\alpha = 0.5$) hanno introdotto un'elevata instabilità, con valori di instabilità che in alcuni casi hanno superato **500**.

La configurazione che ha prodotto i migliori risultati per l'algoritmo **Heavy Ball** è stata la seguente:

- $\alpha = 0.2$
- $\mu = 0.9$
- **Funzione obiettivo** $= 1.047$
- **Velocità di convergenza** $= 29.3$

- **Instabilità** = 19.67

Questi risultati suggeriscono che, con un basso livello di regolarizzazione, l'algoritmo è in grado di ottenere prestazioni soddisfacenti in termini di convergenza, anche se il valore della funzione obiettivo non scende mai al di sotto di **1.0**.

Nella tabella 3 seguente sono riportate le dieci configurazioni migliori trovate per l'algoritmo **Heavy Ball**, ordinate in base ai valori di loss, velocità di convergenza e instabilità, con $\lambda = 0.01$.

α	μ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.20	0.9	1.047	29.3	19.67
0.10	0.9	1.054	13.9	6.10
0.20	0.7	1.066	17.6	9.19
0.05	0.9	1.097	27.7	0.39
0.50	0.7	1.098	34	508.25
0.50	0.5	1.102	21	209.51
0.20	0.5	1.139	6.6	4.45
0.50	0.9	1.140	36.9	158.01
0.75	0.5	1.142	15.6	966.10
0.50	0.3	1.51	25.7	114.744

Tabella 3: Heavy Ball con $\lambda = 0.01$ e 1000 iterazioni

Il metodo **Deflected Subgradient (DSG)** ha ottenuto un valore finale della funzione obiettivo più alto rispetto a **Heavy Ball (HB)**, con la migliore configurazione che ha raggiunto un valore di **1.2359**. Questo suggerisce che, sebbene DSG offra una maggiore stabilità, potrebbe interrompersi prematuramente senza raggiungere un minimo più profondo.

Dal punto di vista della **velocità di convergenza**, DSG ha mostrato un comportamento più variabile rispetto a HB. Alcune configurazioni hanno soddisfatto il criterio di arresto prima di raggiungere il limite massimo di iterazioni, con la migliore che si è fermata a **638 iterazioni**, mentre altre hanno raggiunto il tetto massimo di **1000 iterazioni**.

Per quanto riguarda l'**instabilità**, il metodo si è rivelato estremamente stabile nella maggior parte delle configurazioni, con valori pari a **0** nella quasi totalità dei casi. L'unica eccezione è stata la configurazione con $\alpha = 5$, $\gamma = 0.5$, che ha registrato un valore di instabilità di **2.7236**, comunque inferiore rispetto alle configurazioni più instabili di HB.

A differenza di HB, DSG ha mostrato una **forte dipendenza dal parametro** α, mentre la sensibilità rispetto a γ è risultata inferiore. In particolare:

- Valori più elevati di α (10-20) hanno favorito una **convergenza più rapida** e un valore della **funzione obiettivo inferiore**.
- Valori più bassi di α (2-5) hanno invece prodotto risultati peggiori, con una funzione obiettivo superiore a **2.8**.
- Il parametro γ (fattore di deflessione) non ha avuto un impatto significativo sulle prestazioni complessive, con risultati simili indipendentemente dal valore scelto. Tuttavia, configurazioni con γ più basso (0.1 - 0.3) hanno mostrato una leggera tendenza a migliorare la velocità di convergenza rispetto a valori più elevati.

La configurazione che ha prodotto i risultati migliori per l'algoritmo **Deflected Subgradient** con $\lambda = 0.01$ è la seguente:

- $\alpha = 20$
- $\gamma = 0.1$
- **Funzione obiettivo** = 1.235
- **Velocità di convergenza** = 47.3
- **Instabilità** = 0.0

Nella tabella 4 sono riportate le dieci migliori configurazioni trovate per l'algoritmo **Deflected Subgradient**, ordinate in base ai valori della funzione obiettivo, velocità di convergenza e instabilità, con $\lambda = 0.01$.

α	γ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
20	0.1	1.235	47.3	0.0
15	0.1	1.373	40.8	0.0
10	0.1	1.497	13.7	0.0
5	0.3	1.823	10.1	0.0
5	0.5	1.828	8.4	2.72
5	0.1	1.978	9.4	0.0
2	0.9	2.801	11.8	0.0
2	0.7	2.803	12.0	0.0
2	0.5	2.811	12.0	0.0
2	0.3	2.874	11.7	0.0

Tabella 4: Deflected Subgradient con $\lambda = 0.01$ e 1000 iterazioni

Un aspetto cruciale da valutare è se i due algoritmi convergano allo stesso minimo o se uno dei due si arresti in un minimo locale meno profondo.

Sebbene HB ottenga un valore inferiore della funzione obiettivo, DSG utilizza un passo **Diminishing Step-Size (DSS)**, il che potrebbe permettergli, nel lungo periodo, di avvicinarsi allo stesso minimo. Tuttavia, il limite di **1000 iterazioni** potrebbe aver impedito a DSG di esplorare più a fondo lo spazio dei parametri, limitandone le capacità di ottimizzazione.

L'**assenza di instabilità** in DSG suggerisce che l'algoritmo potrebbe stabilizzarsi prematuramente in un minimo locale, mentre HB, pur mostrando un comportamento più irregolare, continua a migliorare la funzione obiettivo fino al termine delle iterazioni. Questo potrebbe indicare che HB ha una maggiore capacità di **sfuggire ai minimi locali meno profondi**, riuscendo a trovare soluzioni migliori nel lungo termine.

Dopo aver completato una prima fase di *grid search* con $\lambda = 0.01$, è emerso che le configurazioni migliori per gli algoritmi **Heavy Ball** e **Deflected Subgradient** hanno mostrato prestazioni promettenti. Tuttavia, il limite massimo di **1.000 iterazioni** potrebbe aver impedito agli algoritmi di raggiungere un livello di ottimizzazione più accurato.

Per approfondire l'analisi, si è deciso di affinare ulteriormente i risultati selezionando i parametri migliori emersi nella prima fase. In questa seconda fase, il numero massimo di iterazioni è stato aumentato a **10.000** e il valore di ϵ è stato ridotto a $1e^{-6}$, al fine di ottenere una maggiore precisione e migliorare la qualità della convergenza.

Le configurazioni selezionate per questa seconda fase di analisi sono le seguenti:

- **Per l'algoritmo Heavy Ball:**
  - α (learning rate): [0.1, 0.2]

  - μ (momentum): [0.7, 0.9]

- **Per l'algoritmo Deflected Subgradient:**
  - α (learning rate): [10, 15, 20]
  - γ (deflessione): [0.1, 0.3]

Dopo aver eseguito la *grid search* utilizzando questi parametri ottimizzati, i migliori risultati ottenuti dall'algoritmo **Heavy Ball** sono riportati nella seguente tabella 5:

α	μ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.2	0.7	0.9794	14.16	21.611
0.1	0.9	0.9812	43.79	14.001
0.2	0.9	0.9858	65.64	29.172
0.1	0.7	0.9937	33.22	8.089

Tabella 5: Heavy Ball con $\lambda = 0.01$ e 10000 iterazioni

I risultati evidenziano un chiaro miglioramento rispetto alla precedente *grid search* effettuata con **1.000 iterazioni**. Il valore della **funzione obiettivo** è sceso al di sotto di **1.00**, dimostrando che l'aumento del numero massimo di iterazioni ha permesso all'algoritmo di proseguire l'ottimizzazione, raggiungendo così una soluzione più precisa.

Un confronto tra le diverse configurazioni testate mostra quanto segue:

- La **migliore configurazione** in termini di minimizzazione della funzione è stata ottenuta con $\alpha = 0.2$ e $\mu = 0.7$, con una loss pari a **0.9794**.
- La configurazione con $\alpha = 0.1$ e $\mu = 0.9$ ha mostrato una buona stabilità, raggiungendo un valore della funzione di **0.9812** e una velocità di convergenza superiore (**43.79** rispetto a **14.16**).
- Con $\alpha = 0.2$ e $\mu = 0.9$, l'algoritmo ha evidenziato una convergenza ancora più rapida (**65.64**), ma a discapito della stabilità, con un aumento significativo dell'instabilità.

Questi risultati suggeriscono che, per ottenere un compromesso ottimale tra minimizzazione della funzione, stabilità e velocità di convergenza, la configurazione ideale potrebbe essere:

- $\alpha = 0.1$, $\mu = 0.9$: questa combinazione garantisce una bassa instabilità e una buona velocità di convergenza.

I migliori risultati ottenuti dall'algoritmo **Deflected Subgradient** sono riportati di seguito 6.

α	γ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
20	0.1	1.131	39.78	0.000020
15	0.1	1.234	28.64	0.0
10	0.1	1.357	27.50	0.0
10	0.3	3.721	99.80	463.89
15	0.3	5.345	99.80	1964.58
20	0.3	7.568	100	4301.579

Tabella 6: Deflected Subgradient con $\lambda = 0.01$ e 10000 iterazioni

La configurazione migliore individuata per il DSG presenta i seguenti parametri:

- $\alpha = 20$, $\gamma = 0.1$
- **Funzione obiettivo** = 1.131
- **Velocità di convergenza** = 39.78
- **Instabilità** = 0.000020

L'algoritmo **Deflected Subgradient (DSG)** ha mostrato risultati più eterogenei rispetto a **Heavy Ball (HB)**, con una valore della **funzione obiettivo finale di 1.131**, indicando un miglioramento rispetto ai valori ottenuti nella prima fase dell'analisi.

Dal punto di vista della **velocità di convergenza**, DSG ha dimostrato una buona capacità di ottimizzazione, con la migliore configurazione che ha raggiunto un valore di **39.78**. Tuttavia, alcune configurazioni hanno registrato valori estremamente elevati, suggerendo un comportamento **instabile** dell'algoritmo in determinati casi.

Per quanto riguarda l'**instabilità**, il metodo ha mantenuto la sua caratteristica di **elevata stabilità** nelle configurazioni con $\gamma = 0.1$, dove il valore massimo registrato è stato di **0.000020**. Tuttavia, con $\gamma = 0.3$, l'algoritmo ha mostrato un comportamento altamente instabile, con valori che hanno superato **4300**.

L'utilizzo di un passo **Diminishing Step-Size (DSS)** in DSG potrebbe consentire all'algoritmo di avvicinarsi progressivamente allo stesso minimo di HB, sebbene lo faccia in modo significativamente più lento.

3.2.1.1 Analisi tasso di convergenza con $\lambda = 0.01$

Dopo aver analizzato i risultati ottenuti con un parametro di regolarizzazione $\lambda = 0.01$, si procede con il confronto tra il **tasso di convergenza teorico** ed **empirico** per entrambi gli algoritmi: **Heavy Ball (HB)** e **Deflected Subgradient (DSG)**.

Il confronto viene effettuato utilizzando il **gap relativo**, definito come:

$$\text{gap relativo} = f(x_k) - f^*$$

dove:

- $f(x_k)$ rappresenta il valore della **funzione obiettivo** alla k-esima iterazione;
- f^* è il valore stimato del **minimo globale**, ottenuto eseguendo l'algoritmo (con gli stessi parametri) per **100.000 iterazioni** e considerando il valore minimo della funzione obiettivo raggiunto.

Questo confronto permette di valutare quanto rapidamente gli algoritmi si avvicinano al minimo globale e di confrontare l'efficienza empirica con le previsioni teoriche di convergenza.

L'utilizzo del **gap relativo** consente di effettuare un confronto più equo tra i due algoritmi, eliminando eventuali differenze di scala nei valori della **funzione obiettivo**. Inoltre, la rappresentazione dei risultati su una scala logaritmica (*log-log*) permette di verificare se il comportamento empirico degli algoritmi segue il tasso di convergenza teorico atteso.

I tassi di convergenza teorici previsti per i due algoritmi sono i seguenti:

- **Heavy Ball**: Teoricamente, questo metodo dovrebbe presentare un tasso di convergenza di $O(1/k)$. Tuttavia, la presenza della regolarizzazione L1 e la natura della funzione di perdita (che combina *cross-entropy* e regolarizzazione), non essendo *L-smooth*, potrebbero rallentare la convergenza reale rispetto alla stima teorica.

- **Deflected Subgradient**: Per questo algoritmo, il tasso di convergenza teorico atteso è $O(1/\sqrt{k})$. In generale, i metodi basati sul subgradiente tendono a mostrare una decrescita della loss più lenta rispetto ai metodi che sfruttano l'accelerazione del gradiente.

Nel primo grafico 5, viene effettuato un confronto tra il tasso di convergenza empirico dell'algoritmo **Heavy Ball** (con $\alpha = 0.1$ e $\mu = 0.9$) e il tasso di convergenza teorico $O(1/k)$. Questo confronto permette di valutare quanto il comportamento reale dell'algoritmo si avvicini alle aspettative teoriche.

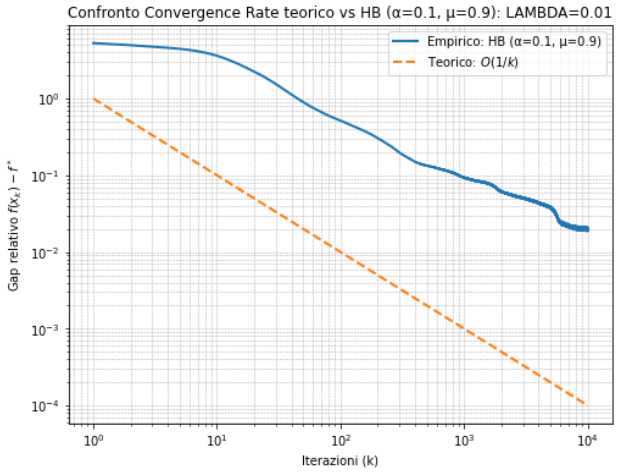


Figura 5: Confronto tra tasso di convergenza HB e il tasso di convergenza teorico.

Inizialmente, l'algoritmo segue un andamento coerente con le previsioni teoriche, mostrando una diminuzione progressiva della **funzione obiettivo**. Tuttavia, nelle fasi avanzate dell'ottimizzazione, si osserva una leggera deviazione rispetto alla curva teorica. Questo comportamento è attribuibile all'effetto della regolarizzazione L1, che introduce una penalizzazione sui pesi della rete neurale, e al fatto che la funzione di perdita non è *L-smooth*.

La rappresentazione su scala log-log conferma che il decadimento della funzione obiettivo non segue esattamente il comportamento teorico di $O(1/k)$, risultando invece leggermente più lento.

In generale, l'algoritmo **Heavy Ball** con $\lambda = 0.01$ si conferma efficace, mantenendo una buona velocità di convergenza e mostrando una diminuzione stabile della perdita.

Nel secondo grafico 6, viene effettuato un confronto analogo per l'algoritmo **Deflected Subgradient** con $\alpha = 20$ e $\gamma = 0.1$. Anche in questo caso, viene confrontato il comportamento empirico con il tasso teorico atteso di $O(1/\sqrt{k})$, consentendo di valutare la coerenza tra il comportamento osservato e le previsioni teoriche.

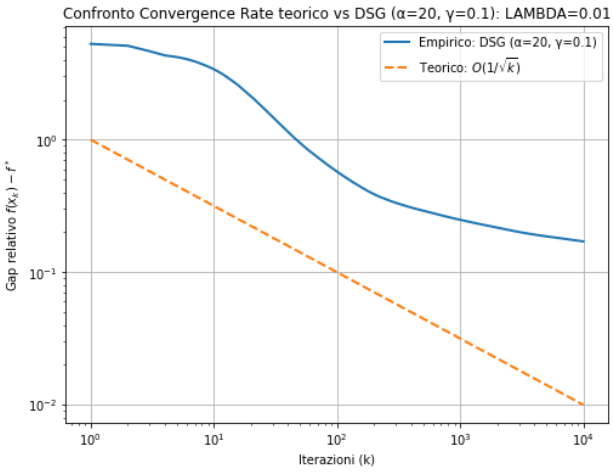


Figura 6: Confronto tra tasso di convergenza DSG e il tasso di convergenza teorico.

Il secondo grafico confronta il **tasso di convergenza empirico** con quello **teorico** per il metodo **Deflected Subgradient (DSG)**. Come previsto, la curva empirica segue una traiettoria più lenta rispetto a **Heavy Ball (HB)**, in linea con la previsione teorica di $O(1/\sqrt{k})$.

L'**andamento della curva empirica** mostra una progressiva stabilizzazione nelle iterazioni successive, suggerendo che l'algoritmo sta raggiungendo una regione della funzione obiettivo in cui le variazioni tra le iterazioni diventano sempre più ridotte. Tuttavia, rispetto alle previsioni teoriche, si osserva una **leggera deviazione nel comportamento reale**, con una convergenza più lenta del previsto. Questo potrebbe essere dovuto alla **sensibilità dell'algoritmo al parametro** γ, che potrebbe non essere stato ottimizzato al meglio in questa configurazione.

L'**assenza di oscillazioni significative** conferma che l'algoritmo mantiene un comportamento stabile. Tuttavia, la velocità di discesa della funzione obiettivo risulta inferiore rispetto ai metodi che sfruttano il **momentum**, come **Heavy Ball**.

Nel terzo grafico 7, viene effettuato un confronto diretto tra i due algoritmi: **Heavy Ball (HB)** con $\alpha = 0.1$ e $\mu = 0.9$ e **Deflected Subgradient (DSG)** con $\alpha = 20$ e $\gamma = 0.1$. Il confronto è basato sul valore del **gap relativo**, consentendo di valutare le differenze in termini di velocità di convergenza e stabilità tra i due metodi.

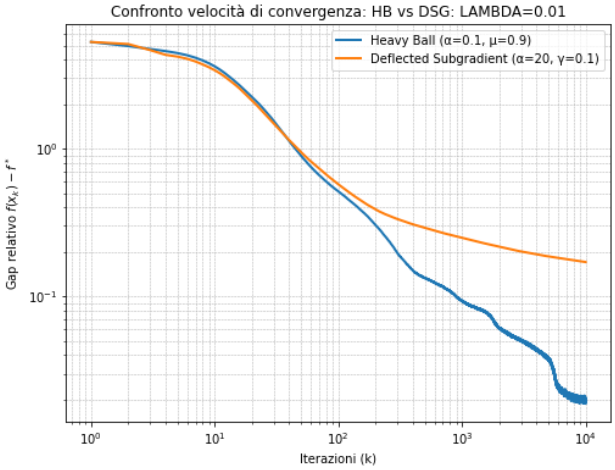


Figura 7: Confronto tra HB e DSG .

Il terzo grafico mostra un **confronto diretto** tra **Heavy Ball (HB)** e **Deflected Subgradient (DSG)** utilizzando lo stesso criterio di **gap relativo**.

Dall'analisi della curva, si osserva che **Heavy Ball converge più rapidamente** rispetto a **Deflected Subgradient**. Nelle prime iterazioni, entrambi i metodi mostrano un comportamento simile, ma dopo un certo numero di iterazioni, **HB continua a decrescere più velocemente**, mentre **DSG rallenta progressivamente**.

L'**andamento più regolare** di DSG conferma la sua **maggiore stabilità**, mentre HB, pur mostrando alcune **oscillazioni**, riesce a raggiungere un **valore inferiore della funzione obiettivo in meno tempo**. Questa differenza riflette le caratteristiche fondamentali dei due algoritmi:

- **HB** sfrutta il **momentum** per **accelerare** la discesa del gradiente.
- **DSG** adotta un aggiornamento più **conservativo**, limitando la velocità di discesa.

Le oscillazioni osservate nella curva di HB possono essere attribuite all'effetto del **momentum**, che amplifica la sensibilità alle variazioni del gradiente. Tuttavia, nonostante questa variabilità, **HB raggiunge un valore finale della funzione obiettivo più basso rispetto a DSG**.

L'**analisi del gap relativo** conferma che **Heavy Ball raggiunge il minimo della funzione obiettivo più rapidamente** rispetto a **Deflected Subgradient**, come previsto dalle teorie di ottimizzazione. Tuttavia, l'effetto della **regolarizzazione L1** e la struttura della funzione di perdita hanno introdotto alcune discrepanze rispetto alle previsioni teoriche.

DSG ha mostrato una **convergenza più lenta**, coerente con la previsione di $O(1/\sqrt{k})$, ma ha mantenuto una **traiettoria più stabile**, senza le oscillazioni presenti in HB. Questo suggerisce che, sebbene **HB sia generalmente più efficace nella minimizzazione della funzione obiettivo**, **DSG potrebbe essere preferibile in scenari in cui la stabilità dell'ottimizzazione è un requisito fondamentale**.

L'uso della **scala logaritmica** nel grafico ha permesso di evidenziare chiaramente come entrambi gli algoritmi seguano **trend compatibili con le loro previsioni teoriche**, sebbene con alcune **deviazioni dovute alle specifiche caratteristiche del problema di ottimizzazione**.

3.2.2 Analisi con $\lambda = 0.1$

Dopo aver analizzato il comportamento degli algoritmi di ottimizzazione con un valore di regolarizzazione $\lambda = 0.01$, si è deciso di testare un valore più elevato di $\lambda = 0.1$ per valutare come una penalizzazione più significativa sui pesi influenzi le prestazioni degli algoritmi **Heavy Ball (HB)** e **Deflected Subgradient (DSG)**.

L'obiettivo di questa fase è comprendere se un livello di regolarizzazione più elevato migliori la capacità di generalizzazione del modello e quale impatto abbia sul tasso di convergenza e sulla stabilità degli algoritmi.

Come nella fase precedente, l'analisi è stata condotta seguendo due passaggi distinti:

- Una prima *grid search* con un massimo di **1.000 iterazioni** e $\epsilon = 1e^{-4}$, per individuare rapidamente le configurazioni migliori.
- Successivamente, i migliori parametri emersi dalla prima fase sono stati selezionati per un'analisi più approfondita, eseguendo un test con **10.000 iterazioni** e $\epsilon = 1e^{-6}$, al fine di valutare in modo più accurato il comportamento degli algoritmi nel lungo termine.

Con $\lambda = 0.1$, l'algoritmo **Heavy Ball (HB)** ha mostrato un comportamento significativamente diverso rispetto a quanto osservato con $\lambda = 0.01$. I principali risultati emersi dalla *grid search* sono i seguenti:

- La **funzione obiettivo** risulta più alta rispetto al caso con $\lambda = 0.01$, con valori che oscillano tra **2.41** e **3.42**.
- Valori bassi di α (pari a **0.01**) garantiscono una maggiore stabilità, mantenendo l'instabilità contenuta fino a $\mu = 0.5$.
- All'aumentare di μ, la velocità di convergenza cresce, ma aumenta anche l'instabilità. Con $\mu = 0.9$, si osservano oscillazioni più marcate nei valori della loss.
- La velocità di convergenza è decisamente superiore rispetto a quanto osservato con $\lambda = 0.01$. In alcuni casi, con un passo pari a **0.05**, la velocità di convergenza può superare **90**.

La configurazione migliore per l'algoritmo **Heavy Ball** con un massimo di **1.000 iterazioni** è la seguente:

- $\alpha = 0.01$
- $\mu = 0.1$
- **Funzione obiettivo** = 2.413
- **Velocità di convergenza** = 43.0
- **Instabilità** = 15.42

In tabella 7 sono riportate le **10 migliori configurazioni** dell'algoritmo **Heavy Ball** con un massimo di **1.000 iterazioni** e $\epsilon = 1e^{-4}$.

α	μ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.01	0.1	2.413	43.0	15.42
0.01	0.3	2.414	55.5	4.91
0.01	0.5	2.421	68.1	15.51
0.01	0.7	2.471	80.7	28.35
0.01	0.9	2.587	85.9	63.70
0.05	0.1	2.858	88.7	38.86
0.05	0.3	2.863	91.0	247.89
0.05	0.5	2.899	93.1	144.09
0.05	0.7	3.134	95.7	246.91
0.10	0.1	3.425	94.3	346.94

Tabella 7: Heavy Ball con $\lambda = 0.1$ e 1000 iterazioni

Con $\lambda = 0.1$, il metodo **Deflected Subgradient (DSG)** ha ottenuto una valore finale della **funzione obiettivo inferiore** rispetto a **Heavy Ball (HB)**, con valori compresi tra **2.30 e 2.32**. Questo suggerisce che, in presenza di una maggiore **regolarizzazione**, DSG riesce a ottenere una **soluzione più stabile e meno sensibile** alle fluttuazioni dei gradienti rispetto a HB.

Dal punto di vista della **velocità di convergenza**, le prestazioni di DSG **variano significativamente** a seconda dei parametri scelti. Le **migliori configurazioni** hanno registrato valori superiori a **80.0**, mentre altre hanno raggiunto la convergenza in meno di **600 iterazioni**.

Per quanto riguarda l'**instabilità**, la maggior parte delle configurazioni ha mostrato **valori relativamente bassi**, con alcune eccezioni. In particolare, configurazioni con $\alpha = 5$, $\gamma = 0.3$ hanno registrato un livello di instabilità elevato (**54.20**), suggerendo che **valori troppo alti del parametro di deflessione** (γ) **possono compromettere la stabilità** dell'algoritmo.

Il parametro α ha un impatto significativo sulla **velocità di convergenza**. Valori più elevati, come $\alpha = 5$, tendono a **ridurre il numero di iterazioni necessarie per convergere**, ma al contempo **aumentano il rischio di instabilità**.

Il parametro γ (fattore di deflessione) influisce principalmente sulla **stabilità dell'algoritmo**. Valori **bassi** di γ (0.1 - 0.5) hanno garantito le **migliori prestazioni**, mentre valori più **alti** hanno mostrato una **tendenza all'aumento dell'instabilità**.

La configurazione ottimale per l'algoritmo **Deflected Subgradient** con un massimo di **1.000 iterazioni** è la seguente:

- $\alpha = 1.0$
- $\gamma = 0.5$
- **Funzione obiettivo** = 2.308
- **Velocità di convergenza** = 50.6
- **Instabilità** = 6.67

In Tabella 8 sono riportate le 10 migliori configurazioni del *Deflected Subgradient* con un massimo di 1.000 iterazioni e $\epsilon = 1 \times 10^{-4}$.

α	γ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
1	0.5	2.308	50.6	6.67
2	0.3	2.312	71.2	9.93
2	0.5	2.313	68.6	40.50
1	0.9	2.314	63.5	18.10
5	0.1	2.315	65.7	6.65
2	0.1	2.317	84.2	2.03
2	0.7	2.319	58.1	11.62
1	0.7	2.320	66.3	1.50
5	0.3	2.325	50.8	54.21
2	0.9	2.326	50.3	28.01

Tabella 8: Deflected Subgradient con $\lambda = 0.1$ e 1000 iterazioni

Dopo aver individuato i parametri migliori dalla *grid search* iniziale, si è proceduto con un'analisi più approfondita aumentando il numero massimo di iterazioni a **10.000** e riducendo ϵ a $1e^{-6}$, al fine di ottenere una stima più precisa della convergenza degli algoritmi.

Le configurazioni testate per ciascun algoritmo sono state le seguenti:

- **Per Heavy Ball (HB):**
  - $\alpha = 0.01$
  - $\mu = [0.1, 0.3, 0.5, 0.7, 0.9]$
- **Per Deflected Subgradient (DSG):**
  - $\alpha = [1, 2, 5]$
  - $\gamma = [0.1, 0.3, 0.5, 0.7, 0.9]$

Rispetto ai risultati ottenuti con **1.000 iterazioni**, l'ottimizzazione prolungata ha prodotto un leggero miglioramento nella **funzione obiettivo**, anche se non significativo:

- Il minimo valore della **funzione obiettivo** ottenuto è stato **2.412**, un valore molto vicino a quello raggiunto con un numero inferiore di iterazioni.
- La velocità di convergenza è migliorata notevolmente, passando da **43** a **94**, suggerendo che l'algoritmo riesce ad avvicinarsi all'ottimo più rapidamente rispetto al caso con $\lambda = 0.01$.
- L'instabilità è rimasta elevata, ma entro limiti accettabili.

La configurazione ottimale per l'algoritmo **Heavy Ball** con **10.000 iterazioni** è la seguente:

- $\alpha = 0.01$
- $\mu = 0.5$
- **Funzione obiettivo** = 2.419
- **Velocità di convergenza** = 96.76
- **Instabilità** = 25.67

Di seguito è riportata la tabella 9 con le migliori configurazioni dell'algoritmo **Heavy Ball** con **10.000 iterazioni** e $\epsilon = 1e^{-6}$.

α	μ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.01	0.1	2.412	94.28	37
0.01	0.3	2.417	95.50	53.54
0.01	0.5	2.419	96.76	25.67
0.01	0.7	2.469	98.04	36.76
0.01	0.9	2.568	98.03	67.82

Tabella 9: Heavy Ball con $\lambda = 0.01$ e 10000 iterazioni

Il metodo **Deflected Subgradient (DSG)** ha ottenuto un valore finale della **funzione obiettivo leggermente inferiore** rispetto a **Heavy Ball (HB)**, con un valore minimo di **2.3026**, migliorando rispetto ai **2.3082** registrati nella fase con **1000 iterazioni**.

Dal punto di vista della **velocità di convergenza**, DSG ha raggiunto **valori elevati**, con la maggior parte delle configurazioni comprese tra **90.0 e 97.76**. Questo suggerisce che l'algoritmo ha mantenuto una **traiettoria stabile** e ha continuato a **migliorare la funzione obiettivo** con l'aumento delle iterazioni.

L'**instabilità** è rimasta **estremamente contenuta** nella maggior parte delle configurazioni, con valori inferiori a **2.0** nella maggioranza dei casi. Solo per $\gamma = 0.5$ si osserva un valore massimo di **7.17**, comunque molto inferiore rispetto alle oscillazioni riscontrate in **HB**.

Le **differenze tra le configurazioni di** γ sono risultate più evidenti in questa fase. Valori più **bassi** di γ (0.1 - 0.3) hanno prodotto risultati **più stabili**, mentre valori **più alti** hanno mostrato una **maggiore instabilità**, pur mantenendosi su **livelli gestibili**.

La configurazione migliore per l'algoritmo **Deflected Subgradient** con **10.000 iterazioni** è la seguente:

- $\alpha = 2.0$
- $\gamma = 0.1$

- **Funzione obiettivo** = 2.302
- **Velocità di convergenza** = 97.76
- **Instabilità** = 0.53

Di seguito è riportata la tabella 10 con le migliori configurazioni dell'algoritmo **Deflected Subgradient** con **10.000 iterazioni** e $\epsilon = 1e^{-6}$.

α	γ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
2	0.1	2.302	97.76	0.53
1	0.1	2.3028	70.61	0.05
1	0.3	2.3030	90.83	0.35
2	0.3	2.3030	95.99	2.01
1	0.5	2.3032	94.54	2.02
5	0.1	2.3032	94.95	1.56
2	0.5	2.3033	95.56	7.17
1	0.7	2.3035	95.49	0.65
1	0.9	2.3039	95.25	3.91
2	0.7	2.3041	93.01	2.34

Tabella 10: Deflected Subgradient con $\lambda = 0.1$ e 10000 iterazioni

L'analisi della qualità del minimo raggiunto dai due algoritmi **Heavy Ball (HB)** e **Deflected Subgradient (DSG)** per un valore di regolarizzazione $\lambda = 0.1$ ha evidenziato differenze significative rispetto al caso con $\lambda = 0.01$. L'aumento della penalizzazione sui pesi ha avuto un impatto sia sulla **stabilità** che sulla **capacità degli algoritmi di individuare minimi profondi** della funzione obiettivo.

Dai risultati ottenuti dopo **10000 iterazioni**, emerge che **HB** raggiunge un valore della funzione obiettivo più altp rispetto a **DSG**, con un minimo pari a **2.413** contro **2.302**. Tuttavia, **HB mostra una maggiore instabilità**, con oscillazioni che aumentano all'aumentare del parametro di momentum μ. Al contrario, **DSG mantiene un comportamento più regolare**, con un valore di instabilità ridotto anche nelle configurazioni più aggressive. Questi risultati suggeriscono che, con un numero limitato di iterazioni, **HB riesce a scendere rapidamente verso un minimo locale**, mentre **DSG procede in modo più graduale ma stabile**.

L'analisi su un numero più elevato di iterazioni, fino a **10.000**, ha permesso di valutare in modo più accurato la convergenza degli algoritmi e l'eventuale presenza di differenze sostanziali. In questa fase, **DSG è riuscito a ridurre il valore della funzione obiettivo fino a 2.302**, superando il minimo trovato da **HB**, che si è fermato a **2.412**. Questo risultato indica che, a differenza di quanto osservato con $\lambda = 0.01$, in questo scenario **DSG non si blocca in un minimo locale subottimale**, ma continua a migliorare la soluzione, superando **HB** nel lungo periodo. Inoltre, **la stabilità di DSG rimane un punto di forza**, con livelli di instabilità trascurabili rispetto a quelli registrati per **HB**.

Il valore della funzione obiettivo dell'algoritmo **HB** oscilla tra 2.41 e 2.42 già a partire dall'iterazione 400 fino alla numero 100.000. Questo comportamento indica che l'algoritmo è rimasto intrappolato in un **minimo locale**, il quale risulta **meno vantaggioso** rispetto a quello ottenuto con il metodo del **Deflected Subgradient**.

La **maggiore stabilità di DSG**, unita alla capacità di raggiungere un valore più basso della funzione obiettivo, suggerisce che questo algoritmo sia **più robusto e meno incline a fermarsi in minimi locali di scarsa qualità**. **HB**, pur mostrando una velocità di convergenza inizialmente superiore, soffre di **instabilità crescente**, che potrebbe compromettere la sua capacità di individuare soluzioni ottimali nel lungo periodo.

In sintesi, **l'aumento del valore di regolarizzazione ha permesso a DSG di ottenere prestazioni migliori rispetto a HB, ribaltando il comportamento osservato con** $\lambda = 0.01$.

3.2.2.1 Analisi tasso di convergenza con $\lambda = 0.1$

Come nell'analisi condotta con $\lambda = 0.01$, verrà eseguito un confronto tra il **tasso di convergenza teorico** ed **empirico** degli algoritmi presi in considerazione, seguito da un breve confronto tra i due metodi. Le procedure utilizzate sono le stesse descritte nella sezione **3.3.2**.

Nel seguente grafico 8 viene analizzata la convergenza empirica dell'algoritmo **Heavy Ball (HB)**, utilizzando i parametri $\alpha = 0.01$ e $\mu = 0.5$, confrontandola con il suo tasso di convergenza teorico previsto, pari a $O(1/k)$.

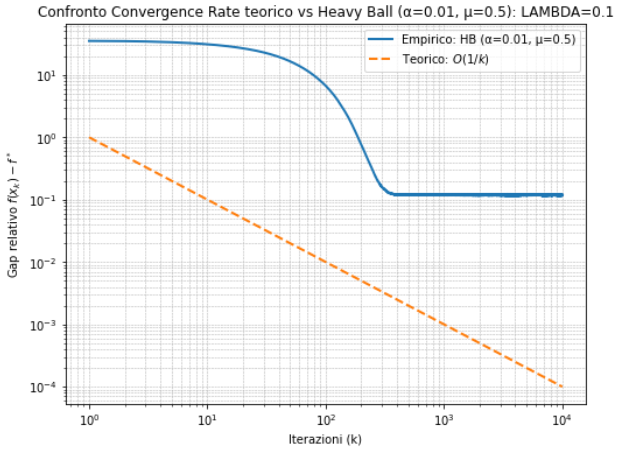


Figura 8: Confronto tra teorico e HB .

Nel **primo grafico** viene mostrato il confronto tra il **tasso di convergenza teorico** $O(1/k)$ e l'**andamento empirico** dell'algoritmo **Heavy Ball (HB)** con parametri $\alpha = 0.01$ e $\mu = 0.5$.

Nelle prime iterazioni, l'algoritmo segue una **traiettoria di riduzione della funzione obiettivo** coerente con le previsioni teoriche, mostrando una **diminuzione progressiva del gap relativo**. Tuttavia, dopo un certo numero di iterazioni, **HB entra in una fase di stallo**: la funzione obiettivo **smette di decrescere in modo significativo** e inizia a oscillare attorno a un valore compreso tra **2.41 e 2.42**. Questo comportamento suggerisce che l'algoritmo si sia **stabilizzato su un minimo locale**.

Il **tasso di convergenza osservato** risulta **inferiore a quello teorico** nelle iterazioni avanzate, evidenziando un comportamento **sub-ottimale** rispetto alla previsione di $O(1/k)$. Inoltre, **non si osservano miglioramenti significativi nelle iterazioni successive**, indicando che **HB potrebbe essersi bloccato in un minimo locale non ottimale**.

In questo altro grafico 9 viene effettuato un confronto analogo per l'algoritmo **Deflected Subgradient (DSG)**, utilizzando i parametri $\alpha = 2.0$ e $\gamma = 0.1$, confrontandolo con il suo tasso di convergenza teorico atteso, pari a $O(1/\sqrt{k})$.

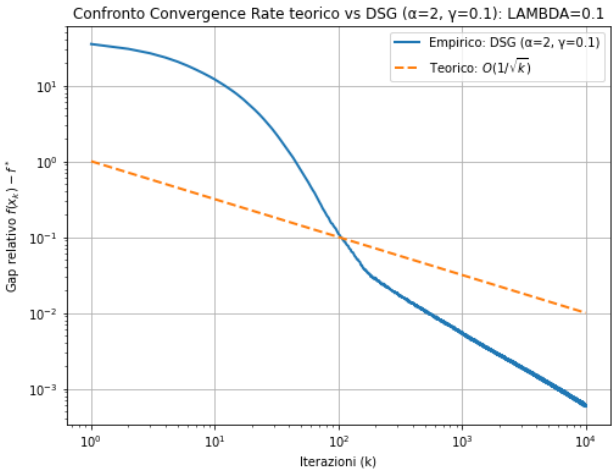


Figura 9: Confronto tra teorico e DSG .

L'algoritmo **Deflected Subgradient (DSG)**, sebbene inizialmente sembri avere una performance inferiore rispetto alle previsioni teoriche, riesce a decrescere in modo costante, superando persino il tasso di convergenza teorico atteso. Questo risultato evidenzia l'efficacia del metodo anche in presenza di regolarizzazione, dimostrando di essere un'ottima scelta per garantire stabilità durante il processo di ottimizzazione.

La regolarizzazione L1 con $\lambda = 0.1$ ha un impatto significativo sulla velocità di convergenza rispetto al caso con $\lambda = 0.01$. Questo risultato suggerisce che il metodo è in grado di mantenere un comportamento stabile ed efficace anche con penalizzazioni più elevate.

In figura 10 viene effettuato un confronto diretto tra i due algoritmi: **Heavy Ball (HB)**, con parametri $\alpha = 0.01$ e $\mu = 0.5$, e **Deflected Subgradient (DSG)**, con parametri $\alpha = 2.0$ e $\gamma = 0.1$, utilizzando come metrica il **gap relativo**.

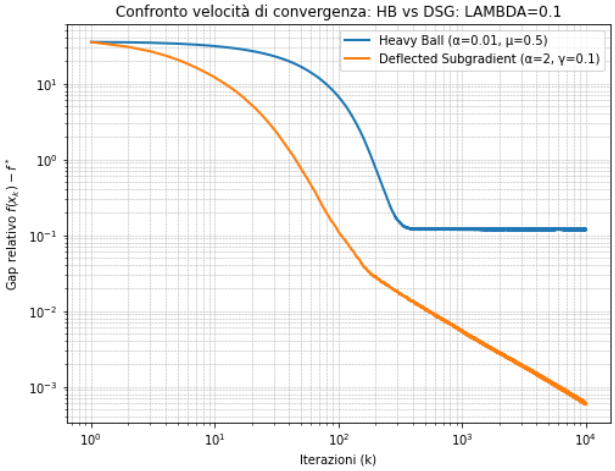


Figura 10: Confronto tra HB e DSG .

Il **terzo grafico** mostra un **confronto diretto** tra **Heavy Ball (HB)** e **Deflected Subgradient (DSG)** in termini di **velocità di convergenza**. Si osserva chiaramente che, **nonostante DSG abbia un tasso di convergenza teoricamente più lento rispetto a HB**, riesce comunque a ottenere **una soluzione migliore nel lungo periodo**.

Mentre **HB si stabilizza rapidamente** su un valore della funzione obiettivo compreso tra **2.41 e 2.42**, **DSG continua a migliorare**, raggiungendo un valore inferiore pari a **2.30**. Questo comportamento suggerisce che, **sebbene HB mostri una convergenza iniziale più rapida**, tende a **bloccarsi in un minimo locale meno profondo rispetto a DSG**.

In particolare, **HB** raggiunge velocemente una soluzione, ma **si stabilizza precocemente** su un valore della funzione obiettivo più elevato, suggerendo la presenza di un **minimo locale subottimale**. Al contrario, **DSG mantiene un miglioramento più graduale**, continuando a diminuire la funzione obiettivo anche nelle iterazioni avanzate e ottenendo un **valore finale inferiore rispetto a HB**.

3.2.3 Analisi con $\lambda = 1$

Dopo aver analizzato il comportamento degli algoritmi di ottimizzazione con $\lambda = 0.01$ e $\lambda = 0.1$, si è deciso di testare un valore ancora più elevato di regolarizzazione, $\lambda = 1.0$, al fine di valutare come una penalizzazione significativamente più forte sui pesi influenzi le prestazioni degli algoritmi **Heavy Ball (HB)** e **Deflected Subgradient (DSG)**.

L'obiettivo di questa analisi è determinare se un livello di regolarizzazione così elevato porti a una maggiore capacità di generalizzazione e stabilità del modello, e come influisca sul tasso di convergenza e sulla stabilità dei due algoritmi.

Come nei test precedenti, la valutazione è stata condotta in due fasi distinte:

- **Prima fase**: È stata eseguita una *grid search* con un massimo di **1.000 iterazioni** e $\epsilon = 1e^{-4}$, per individuare le migliori configurazioni iniziali.
- **Seconda fase**: Sono stati selezionati i parametri migliori emersi dalla prima fase e successivamente testati con **10.000 iterazioni** e $\epsilon = 1e^{-6}$, al fine di valutare con maggiore precisione il comportamento a lungo termine degli algoritmi.

Con $\lambda = 1.0$, l'algoritmo **Heavy Ball (HB)** ha mostrato un comportamento estremamente instabile, caratterizzato da oscillazioni significative della **funzione obiettivo** e valori molto elevati.

Le principali osservazioni emerse dall'analisi sono le seguenti:

- La **funzione obiettivo** è drasticamente più alta rispetto ai test condotti con $\lambda = 0.01$ e $\lambda = 0.1$, con valori che superano anche i **30-70** nelle configurazioni più instabili.
- Molte configurazioni risultano instabili, soprattutto per valori di $\mu \geq 0.7$, con livelli di instabilità superiori a **1000**.
- Per valori di $\mu = 0.1$ e $\mu = 0.3$, la stabilità migliora leggermente, con una **funzione obiettivo** compresa tra **13.24** e **13.43**, sebbene questi valori siano comunque significativamente più alti rispetto ai test precedenti.
- Un aumento di μ comporta una velocità di convergenza più elevata (superiore a **95**), ma a scapito della stabilità.
- Configurazioni con $\alpha = 0.05$ e $\alpha = 0.1$ risultano altamente instabili, soprattutto per valori di $\mu \geq 0.5$.

La configurazione migliore per l'algoritmo **Heavy Ball** con **1.000 iterazioni** (che rappresenta il miglior compromesso tra stabilità e velocità di convergenza) è la seguente:

- $\alpha = 0.01$

- $\mu = 0.1$
- **Funzione obiettivo** = 13.43
- **Velocità di convergenza** = 94.0
- **Instabilità** = 445.17

Di seguito è riportata la tabella 11 con le migliori configurazioni per l'algoritmo **Heavy Ball** con **1.000 iterazioni** e $\epsilon = 1e^{-4}$.

α	μ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.01	0.3	13.24	95.2	1138.70
0.01	0.1	13.43	94.0	445.17
0.01	0.5	14.53	93.8	795.79
0.01	0.7	19.19	97.4	1284.66
0.01	0.9	30.89	92.3	1004.09
0.05	0.1	58.31	98.6	671.05
0.05	0.3	59.07	99	1128.49
0.05	0.5	64.17	98.3	2343.45
0.05	0.7	73.99	98.7	7433.80
0.10	0.5	111.54	98.7	5985.84

Tabella 11: Heavy Ball con $\lambda = 1$ e 1000 iterazioni

Per l'algoritmo **Deflected Subgradient (DSG)**, il comportamento osservato con $\lambda = 1.0$ risulta nettamente diverso rispetto sia all'algoritmo **Heavy Ball (HB)** sia ai test condotti con penalizzazioni inferiori.

Le principali osservazioni emerse dall'analisi sono le seguenti:

- La **funzione obiettivo** è significativamente più bassa rispetto a quella di HB, con valori compresi tra **2.35** e **2.57**, indicando che il metodo è più efficace nel trovare soluzioni migliori anche con una forte penalizzazione.
- L'instabilità rimane contenuta, con valori generalmente molto bassi (inferiori a **20**), ad eccezione di alcune configurazioni isolate.
- Un aumento di α tende a migliorare la velocità di convergenza, raggiungendo valori fino a **66.3** nelle configurazioni migliori.

La configurazione ottimale per l'algoritmo **DSG** con **1.000 iterazioni** è la seguente:

- $\alpha = 0.20$
- $\gamma = 0.1$
- **Funzione obiettivo** = 2.35
- **Velocità di convergenza** = 34.9
- **Instabilità** = 18.97

Sotto è riportata la tabella 12 con le migliori configurazioni per l'algoritmo **Deflected Subgradient (DSG)** con **1.000 iterazioni** e $\epsilon = 1e^{-4}$.

α	γ	**Funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.20	0.1	2.35	34.9	18.97
0.20	0.3	2.40	20.6	124.22
0.50	0.1	2.42	16.9	82.61
0.10	0.7	2.43	12.4	32.80
0.10	0.5	2.43	9.4	73.05
0.10	0.9	2.44	11.1	562.50
0.20	0.7	2.46	12.6	86.78
0.75	0.1	2.49	12.9	158.40
1.00	0.1	2.54	8.1	239.36
0.10	0.3	2.57	5.3	5.00

Tabella 12: Deflected Subgradient con $\lambda = 1$ e 1000 iterazioni

Dopo aver individuato i parametri ottimali dalla *grid search* iniziale, si è proceduto con un'analisi più approfondita aumentando il numero di iterazioni a **10.000** e riducendo ϵ a $1e^{-6}$, al fine di ottenere una stima più precisa della convergenza degli algoritmi.

Le configurazioni testate per ciascun algoritmo sono state le seguenti:

- **Per Heavy Ball (HB):**
  - $\alpha = 0.01$
  - $\mu = [0.1, 0.3, 0.5, 0.7, 0.9]$
- **Per Deflected Subgradient (DSG):**
  - $\alpha = [0.1, 0.2]$
  - $\gamma = [0.1, 0.3, 0.5, 0.7, 0.9]$

Per l'algoritmo **Heavy Ball**, l'aumento del numero di iterazioni ha portato a un leggero miglioramento della **funzione obiettivo**, anche se non significativo rispetto ai risultati precedenti:

- La **funzione obiettivo** minima raggiunta è stata **13.57**, un valore molto vicino a quello ottenuto con un numero inferiore di iterazioni.
- La velocità di convergenza è migliorata in modo considerevole, passando da **43** a **73**, suggerendo che l'algoritmo è in grado di avvicinarsi più rapidamente all'ottimo.
- L'instabilità rimane elevata, con valori superiori a **900** anche nella configurazione migliore.

La configurazione ottimale per l'algoritmo **Heavy Ball** con **10.000 iterazioni** è risultata la seguente:

- $\alpha = 0.01$
- $\mu = 0.5$
- **funzione obiettivo** = 14.07
- **Velocità di convergenza** = 60.2
- **Instabilità** = 791.20

La tabella 13 riporta le migliori configurazioni dell'algoritmo **Heavy Ball (HB)** ottenute con **10.000 iterazioni** e $\epsilon = 1e^{-6}$.

α	μ	**funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.01	0.3	13.24	99.52	1138.70
0.01	0.1	13.57	73.05	912.98
0.01	0.5	14.07	60.20	791.20
0.01	0.7	19.19	99.74	1284.66
0.01	0.9	30.89	99.23	1004.09
0.05	0.1	57.92	98.42	571.85
0.05	0.3	59.07	99.90	1128.49
0.05	0.5	64.17	98.83	2243.45
0.05	0.7	73.99	98.87	7433.80
0.05	0.9	120.94	99.88	3624.98

Tabella 13: Heavy Ball con $\lambda = 1$ e 10000 iterazioni

L'algoritmo **Deflected Subgradient (DSG)** mostra un netto miglioramento rispetto all'algoritmo **Heavy Ball (HB)**, ottenendo risultati significativamente più stabili e una **funzione obiettivo** decisamente inferiore:

- La **funzione obiettivo** minima è scesa fino a **2.307**, avvicinandosi molto di più all'ottimo rispetto a quanto osservato con HB.
- La velocità di convergenza è aumentata sensibilmente, superando valori compresi tra **70** e **80** nelle configurazioni migliori.
- L'instabilità è praticamente nulla, confermando che il metodo rimane estremamente stabile anche con un numero di iterazioni molto elevato.

La configurazione ottimale per l'algoritmo **DSG** con **10.000 iterazioni** è la seguente:

- $\alpha = 0.2$
- $\gamma = 0.1$
- **funzione obiettivo** = 2.307
- **Velocità di convergenza** = 82.70
- **Instabilità** = 5.11

La tabella 14 riporta le migliori configurazioni ottenute per l'algoritmo **Deflected Subgradient (DSG)** con **10.000 iterazioni** e $\epsilon = 1e^{-6}$.

α	γ	**funzione obiettivo**	**Velocità di convergenza**	**Instabilità**
0.1	0.3	2.307	65.60	3.86
0.1	0.5	2.307	77.80	23.12
0.2	0.1	2.307	82.70	5.11
0.1	0.7	2.310	73.57	9.68
0.2	0.3	2.312	71.08	22.34
0.1	0.9	2.314	68.63	123.45
0.2	0.7	2.319	58.50	22.23
0.1	0.1	2.323	20.14	0.50

Tabella 14: Deflected Subgradient con $\lambda = 1$ e 10000 iterazioni

L'analisi della qualità della convergenza dei due algoritmi **Heavy Ball (HB)** e **Deflected Subgradient (DSG)**, con una regolarizzazione elevata pari a $\lambda = 1.0$, ha evidenziato differenze ancora più marcate rispetto ai test con valori inferiori di λ. L'aumento della penalizzazione ha avuto un impatto significativo sulle prestazioni, influenzando in particolare la **stabilità** e la **capacità di individuare minimi di qualità**.

L'algoritmo **HB** ha mostrato un comportamento **estremamente instabile**. Le configurazioni con un valore elevato di μ hanno generato oscillazioni considerevoli nella funzione obiettivo, con valori che in alcuni casi hanno **superato 70**, un risultato nettamente peggiore rispetto ai test con regolarizzazioni più basse. Anche nelle configurazioni relativamente più stabili, come quelle con $\mu = 0.1$ o $\mu = 0.3$, la funzione obiettivo si è attestata tra **13.24** e **13.57**, valori significativamente più elevati rispetto a quelli ottenuti con $\lambda = 0.1$ o $\lambda = 0.01$. Questi risultati suggeriscono che **HB non riesce ad adattarsi efficacemente a una regolarizzazione così elevata**, risultando fortemente penalizzato in termini di qualità della soluzione trovata.

L'aumento del numero di iterazioni a **10.000** non ha portato a un miglioramento significativo per **HB**. Il valore minimo della funzione obiettivo raggiunto è rimasto **13.57**, ancora molto distante dai risultati ottenuti con **DSG**. Inoltre, **l'instabilità è rimasta un problema critico**: anche nella migliore configurazione testata, i livelli di instabilità hanno superato **900**, rendendo l'algoritmo **altamente imprevedibile e poco affidabile** in presenza di regolarizzazione elevata.

Al contrario, il metodo **DSG** ha dimostrato **prestazioni nettamente migliori**. Con $\lambda = 1.0$, l'algoritmo ha raggiunto un valore della funzione obiettivo pari a **2.307**, **di gran lunga inferiore** rispetto a quello ottenuto da **HB**. Inoltre, **la stabilità di DSG è rimasta elevata**, con livelli di instabilità prossimi a **0** nella maggior parte delle configurazioni testate. Questi risultati suggeriscono che **DSG sia molto più adatto a scenari con regolarizzazioni elevate**, poiché riesce a mantenere **una traiettoria di ottimizzazione più regolare** e a **convergere verso soluzioni di qualità superiore**.

L'analisi comparativa tra i due algoritmi evidenzia che, con $\lambda = 1.0$, **DSG si dimostra nettamente più efficace** nel trovare un minimo più profondo e con maggiore stabilità. **HB**, al contrario, non solo si ferma in **minimi locali peggiori**, ma soffre anche di **forti oscillazioni** che ne compromettono la convergenza. Il peggioramento delle prestazioni di **HB** con l'aumento della regolarizzazione suggerisce che **questo algoritmo non sia adatto a scenari in cui la penalizzazione sui pesi è particolarmente elevata**.

In conclusione, con $\lambda = 1.0$, **DSG emerge come l'algoritmo nettamente superiore**, garantendo **una migliore qualità della soluzione finale** e **una stabilità significativamente più elevata**. Al contrario, **HB** non solo non riesce a migliorare il valore della funzione obiettivo, ma diventa **altamente instabile**, confermando di **non essere una scelta ottimale in contesti con regolarizzazioni elevate**.

3.2.3.1 Analisi tasso di convergenza con $\lambda = 1$

Dopo aver analizzato la convergenza degli algoritmi **Heavy Ball (HB)** e **Deflected Subgradient (DSG)** con $\lambda = 0.01$ e $\lambda = 0.1$, si considera ora il caso in cui la regolarizzazione L1 raggiunge il suo valore massimo, con $\lambda = 1$. L'obiettivo di questa analisi è esaminare come un livello così elevato di penalizzazione influenzi la velocità di convergenza e la stabilità dei due algoritmi.

Come nei casi precedenti, il confronto viene effettuato utilizzando il **gap relativo**.

In figura 11 viene mostrata la convergenza empirica dell'algoritmo **Heavy Ball (HB)**, con parametri $\alpha = 0.01$ e $\mu = 0.5$, confrontata con il suo tasso di convergenza teorico previsto, pari a $O(1/k)$.

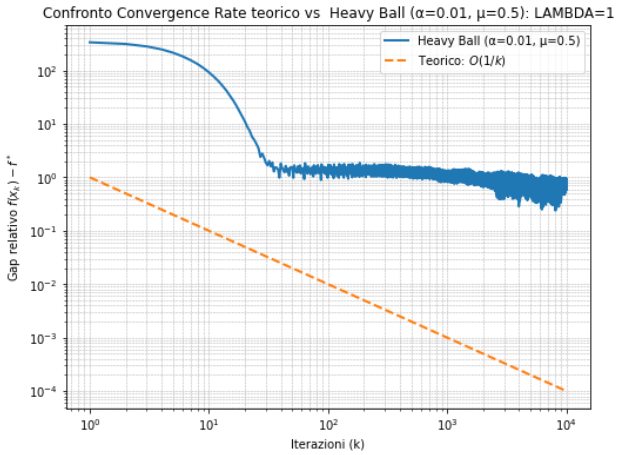


Figura 11: Confronto tra teorico e HB .

Nelle fasi iniziali, l'algoritmo segue un comportamento coerente con il tasso di convergenza teorico atteso, mostrando una riduzione progressiva del **gap relativo**. Tuttavia, nelle iterazioni avanzate, emerge una chiara instabilità, con oscillazioni significative. Questo comportamento è simile a quanto osservato con $\lambda = 0.1$, ma in questo caso le fluttuazioni risultano ancora più pronunciate.

L'instabilità è attribuibile a un'eccessiva accelerazione del metodo **Heavy Ball (HB)**, che causa fluttuazioni attorno al minimo piuttosto che una convergenza stabile e regolare.

Rispetto ai casi con valori di λ inferiori, l'algoritmo sembra risentire in modo più marcato dell'effetto della forte penalizzazione L1, compromettendo la sua efficacia nel raggiungere una soluzione stabile.

In figura 12 viene analizzato il comportamento empirico dell'algoritmo **Deflected Subgradient (DSG)**, utilizzando i parametri $\alpha = 0.2$ e $\gamma = 0.1$, e confrontato con il tasso di convergenza teorico atteso, pari a $O(1/\sqrt{k})$.

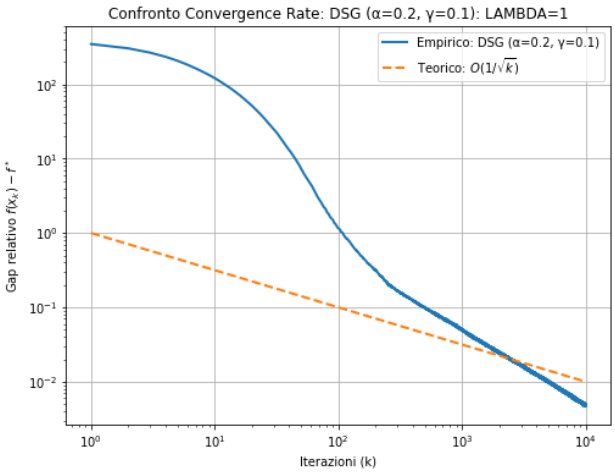


Figura 12: Confronto tra teorico e DSG .

A differenza dell'algoritmo **Heavy Ball (HB)**, il metodo **Deflected Subgradient (DSG)** mantiene una riduzione costante del **gap relativo** senza mostrare oscillazioni. Sebbene nelle prime iterazioni la convergenza sembri leggermente più lenta rispetto a quanto previsto dal tasso teorico, nelle fasi successive l'algoritmo migliora il suo andamento, allineandosi o addirittura superando il tasso di convergenza teorico.

L'assenza di oscillazioni rappresenta un chiaro vantaggio di **DSG** rispetto a **HB**, dimostrando una maggiore robustezza e stabilità.

La presenza di una regolarizzazione L1 elevata non compromette in modo significativo l'efficienza del metodo, confermando che **DSG** è una scelta stabile e affidabile anche in presenza di penalizzazioni elevate.

Anche qua in figura 13 viene effettuato un confronto diretto tra i due algoritmi: **Heavy Ball (HB)**, con parametri $\alpha = 0.01$ e $\mu = 0.5$, e **Deflected Subgradient (DSG)**, con parametri $\alpha = 0.2$ e $\gamma = 0.1$, sulla base del **gap relativo**.

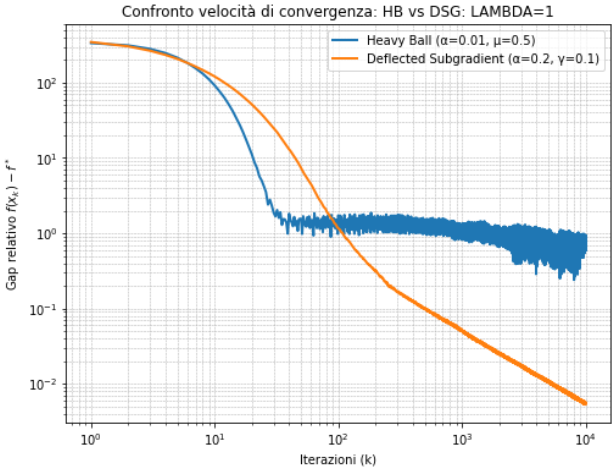


Figura 13: Confronto tra HB e DSG .

L'algoritmo **Heavy Ball (HB)** mostra una fase iniziale più rapida, con una riduzione della perdita più veloce rispetto al **Deflected Subgradient (DSG)**. Tuttavia, nelle fasi avanzate, **HB** diventa instabile e le oscillazioni impediscono una convergenza regolare verso il minimo.

Al contrario, **DSG** risulta inizialmente più lento, ma garantisce una decrescita costante e priva di fluttuazioni significative. La velocità di convergenza finale di **DSG** è superiore a quella di **HB**, poiché il suo **gap relativo** continua a ridursi, mentre **HB** oscilla intorno a valori instabili senza miglioramenti significativi.

L'elevata penalizzazione con $\lambda = 1$ accentua ulteriormente i limiti di **HB**, che diventa sempre più instabile all'aumentare delle iterazioni. Al contrario, **DSG** mantiene un comportamento stabile e regolare durante tutto il processo di ottimizzazione.

In presenza di una forte regolarizzazione ($\lambda = 1$), l'algoritmo **Deflected Subgradient** si dimostra preferibile rispetto a **Heavy Ball**. Non solo offre maggiore stabilità, ma la sua velocità di convergenza finale è migliore. Inoltre, **DSG** mantiene un comportamento regolare anche in scenari con penalizzazioni elevate, dimostrando una maggiore robustezza rispetto a **HB**.

4 Conclusioni

In questo lavoro è stata condotta un'analisi approfondita sulle prestazioni degli algoritmi di ottimizzazione **Heavy Ball (HB)** e **Deflected Subgradient (DSG)**, applicati a una rete neurale con funzione di perdita basata sulla **cross-entropy** e regolarizzazione L1. L'obiettivo principale era confrontare la **velocità di convergenza**, la **stabilità** e l'**efficacia** dei due metodi in presenza di diversi livelli di penalizzazione sui pesi, variando il parametro di regolarizzazione λ tra **0.01**, **0.1** e **1.0**.

L'analisi è stata strutturata in diverse fasi:

1. **Grid search iniziale**: è stata condotta una ricerca dei parametri ottimali con un massimo di **1.000 iterazioni** e $\epsilon = 1e^{-4}$ per individuare le migliori configurazioni di ciascun algoritmo.
2. **Ottimizzazione avanzata**: utilizzando i parametri migliori emersi dalla fase precedente, gli algoritmi sono stati testati con un massimo di **10.000 iterazioni** e $\epsilon = 1e^{-6}$ per valutare il comportamento a lungo termine.
3. **Analisi del tasso di convergenza**: è stato confrontato il comportamento empirico di **HB** e **DSG** con i rispettivi tassi di convergenza teorici:
   - $O(1/k)$ per **HB**, con possibili deviazioni dovute alla natura non L-smooth della funzione di perdita.
   - $O(1/\sqrt{k})$ per **DSG**, caratteristico degli algoritmi basati sul subgradiente.

Dai risultati ottenuti emergono alcune osservazioni chiave sulle prestazioni dei due algoritmi:

L'algoritmo **Heavy Ball (HB)** mostra una convergenza iniziale più rapida rispetto al **Deflected Subgradient (DSG)**, in particolare per valori bassi del parametro di regolarizzazione ($\lambda = 0.01$). Tuttavia, **DSG** si dimostra più lento nelle prime iterazioni, mantenendo però un andamento più costante e regolare nel tempo.

Con l'aumentare delle iterazioni, **HB** tende a diventare instabile, specialmente con valori elevati di λ. Questa instabilità si manifesta con oscillazioni significative nel valore della **funzione obiettivo**, probabilmente dovute all'eccessiva accelerazione introdotta dal metodo. Al contrario, **DSG** mantiene un comportamento molto più stabile in tutti i casi, con una traiettoria di convergenza regolare e priva di fluttuazioni anche per valori elevati di λ.

In termini di effetto della regolarizzazione:

- Con $\lambda = 0.01$, entrambi gli algoritmi mostrano risultati competitivi. **HB** ha una leggera superiorità in termini di velocità di convergenza, ma soffre di instabilità nelle iterazioni successive.
- Con $\lambda = 0.1$, **DSG** diventa più competitivo, offrendo una convergenza più regolare e stabile rispetto a **HB**, che continua a soffrire di oscillazioni più marcate.

- Con $\lambda = 1$, **DSG** si dimostra nettamente superiore: **HB** diventa completamente instabile e inefficace, mentre **DSG** continua a garantire una convergenza affidabile e un valore della **funzione obiettivo** inferiore.

Bibliografia

[1] Vassilis Apidopoulos, Nicolò Ginatta e Silvia Villa. ≪Convergence rates for the heavy-ball continuous dynamics for non-convex optimization, under Polyak–Łojasiewicz condition≫. *Journal of Global Optimization* 84.3 (2022), pp. 563–589.

[2] Jean-François Aujol, Charles Dossal e Aude Rondepierre. ≪Convergence rates of the Heavy-Ball method with Lojasiewicz property≫. Tesi di dott. IMB - Institut de Mathématiques de Bordeaux; INSA Toulouse; UPS Toulouse, 2020, pp. 6–15.

[3] Stephen Boyd e Jonghyuk Park. ≪Subgradient Methods: Notes for EE364b, Stanford University, Spring 2013–14≫. Tesi di dott. Stanford University, 2014, pp. 4–12.

[4] G. D'Antonio. ≪Porting ed estensione di codice C++ per l'ottimizzazione non differenziabile≫. Tesi di dott. Università di Pisa, 2006. Cap. 2.1, pp. 24–36. URL: https://commalab.di.unipi.it/files/Theses/dAntonio.pdf.

[5] A. Frangioni. *Nonsmooth Unconstrained Optimization*. Università di Pisa. 2024-25.

[6] A. Frangioni. *Smooth Unconstrained Optimization*. Università di Pisa. 2024-25.