Università degli Studi di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea Magistrale in Matematica

LISA-AIS: Controllo Dinamico del Batch Size ed Adaptive Importance Sampling nel Deep Learning

Relatore:
Prof.ssa Giorgia Franchini

Laureanda:
Marta Munari

Correlatore:
Dott. Matteo Lombardi

Anno Accademico 2024/2025

Sommario

La necessità di gestire problemi su larga scala rende fondamentale lo sviluppo di algoritmi di Deep Learning che operino in tempi computazionali sostenibili. La presente tesi affronta il problema di minimizzazione del rischio empirico per l'addestramento dei modelli attraverso il *metodo di Discesa del Gradiente Stocastico (SGD)*, focalizzandosi sulle strategie di riduzione della varianza del gradiente stocastico al fine di migliorarne l'efficienza e la stabilità. Tra queste, si distinguono l'estrazione di un *mini-batch* di campioni all'interno della stessa iterazione, il controllo della sua dimensione e la scelta della distribuzione di campionamento utilizzata. In particolare, sono state studiate e integrate le seguenti strategie: la riduzione progressiva della varianza attraverso l'aumento dinamico della dimensione del mini-batch, presentata nell'*algoritmo DeepLISA*, e la *distribuzione di campionamento Adaptive Importance Sampling (AIS)*, un metodo Monte Carlo che approssima la minimizzazione della varianza rispetto alla scelta delle distribuzioni di probabilità.

In principio, sono state sviluppate alcune varianti di *DeepLISA*, che, aggiornando puntualmente la dimensione del mini-batch, permettono di ottenere risultati iniziali migliori grazie alla velocizzazione del processo nelle prime epoche.

Successivamente, attraverso l'integrazione della *strategia AIS* all'interno dell'*algoritmo DeepLISA*, la tesi evidenzia come il metodo risultante, l'*algoritmo DeepLISA-AIS*, permetta di ottenere un rapido incremento delle prestazioni in poche iterazioni, migliori rispetto a quelle dell'algoritmo originale, riducendone inoltre il comportamento oscillatorio. Tuttavia, l'analisi ha mostrato che l'accuratezza tende a stabilizzarsi intorno all'87.9%, risultando inferiore rispetto al metodo originale, il quale raggiunge l'89.0%.

Le due difficoltà principali riscontrate nell'implementazione dell'algoritmo in ambiente PyTorch riguardano la saturazione della memoria del calcolatore e il proibitivo

tempo computazionale legato al calcolo dei gradienti per campione, necessari per la *strategia AIS*. Tali operazioni ritardano il processo di circa 50 secondi per ogni epoca, rendendo la qualità delle prestazioni non paragonabile a quella di *DeepLISA* in termini di tempo.

Per superare queste limitazioni, sono state sviluppate ulteriori varianti, tra cui l'aggiornamento della distribuzione di campionamento con cadenza per epoca e l'aggiornamento puntuale della dimensione del mini-batch. Queste modifiche hanno permesso di risparmiare rispettivamente circa 4 secondi per epoca e 5 minuti sul tempo totale, peggiorando tuttavia il comportamento oscillatorio del metodo.

Risultati di maggior rilievo sono stati ottenuti attraverso l'alternanza del campionamento casuale e della distribuzione adattiva, che ha permesso di stabilizzare l'accuratezza ad una soglia superiore rispetto a *DeepLISA* attraverso un tempo computazionale adeguato, raggiungendo circa l'89.4%.

Infine, si è deciso di provare ad eliminare completamente il calcolo dei gradienti per campione, cogliendo informazioni unicamente dal gradiente del rischio empirico sull'intero mini-batch. Attraverso tempi computazionali molto simili a quelli di *DeepLISA*, questa soluzione permette di ottenere un'accuratezza dell'89.0%, garantendo le stesse prestazioni ma con meno oscillazioni.

Gli algoritmi proposti sono stati validati per l'addestramento di una particolare rete neurale convoluzionale, per la classificazione del dataset *CIFAR10*: la *Residual Network* (*ResNet18*). Essa è costituita da diciotto strati separati in blocchi dalle cosiddette *shortcut connections*, che permettono di ridurre il problema di degradazione dell'accuratezza presente nelle reti neurali molto profonde.

Indice

Capitolo 1

Fondamenti di Machine Learning e Deep Learning

Il *metodo del Gradiente Stocastico (SGD)* è un algoritmo iterativo che permette la risoluzione di particolari tipologie di problemi di ottimizzazione, la cui funzione obiettivo dipende da una distribuzione di probabilità: i *problemi di Ottimizzazione Stocastica*.

Definizione 1.0.1 *Un **problema di Ottimizzazione Stocastica** non vincolato è un problema di minimizzazione (o massimizzazione) la cui funzione obiettivo dipende da una variabile aleatoria. Esso assume la seguente forma*

$$\min_{x \in \mathbb{R}^d} F(x) \equiv \mathbb{E}\left[f(x, \xi)\right]$$

dove

- ξ *è un vettore aleatorio;*
- $f : \mathbb{R}^d \times \xi \to \mathbb{R}$*, funzione di costo che misura la "bontà" dei parametri* x *rispetto ad una specifica realizzazione del vettore aleatorio* ξ*;*
- *l'aspettazione* $\mathbb{E}$ *è costruita rispetto a* ξ *nello spazio di probabilità* $(\Xi, \mathcal{F}, \mathcal{P})$*.*

Tale problema è ampiamente ricorrente nell'ambito del *Machine Learning* e, in particolare, del *Deep Learning*. Il seguente capitolo descrive il contesto teorico e applicativo di riferimento, servendosi dei documenti [14, 15, 31].

1.1 Machine Learning e Problema di Apprendimento Supervisionato

Di seguito, si introducono i concetti fondamentali del Machine Learning, con riferimento a [14, 31].

Definizione 1.1.1 *Si definisce* ***Machine Learning*** *un insieme di metodi in grado di individuare automaticamente pattern nei dati (****"learning from examples"****) e, successivamente, utilizzarli per prevedere dati futuri o per eseguire ulteriori tipi di processi decisionali [23].*

Dunque, è possibile definire il *problema di apprendimento*. Siano due insiemi di variabili $X \subseteq \mathbb{R}^{d_x}$ e $Y \subseteq \mathbb{R}^{d_y}$, legati secondo la distribuzione di probabilità $p_{X,Y}(\mathbf{x}, \mathbf{y})$, definita su $X \times Y$: solitamente, un elemento di X non è associato in modo univoco ad un elemento di Y.

Si consideri un insieme di coppie input-output, chiamato **training set**,

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) \in X \times Y \, : \, i = 1, 2, \ldots, n\}$$

dove $n \in \mathbb{N}$ rappresenta il numero degli esempi ottenuti campionando $X \times Y$ secondo la distribuzione generalmente sconosciuta $p_{X,Y}(\mathbf{x}, \mathbf{y})$. $\mathbf{x}_i$ e $\mathbf{y}_i$ sono chiamati rispettivamente **features** (o **attributi**) e **response variable**.

Il **problema di apprendimento** consiste nell'individuare uno stimatore a partire da $\mathcal{S}$

$$h_{\mathcal{S}} : X \to Y$$

appartenente ad uno spazio di ipotesi adatto $\mathcal{H}$, che può essere utilizzato per prevedere il valore $\mathbf{y} \in Y$ corrispondente a $\mathbf{x} \in X$, ossia

$$h_{\mathcal{S}}(\mathbf{x}) \sim \mathbf{y}$$

Esso può essere suddiviso in due principali tipologie di apprendimento: *apprendimento supervisionato* e *apprendimento non supervisionato*. In particolare, il primo può essere ricondotto alla risoluzione di un problema di ottimizzazione stocastica. Dato il training set di esempi input-output

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) \in X \times Y \, : \, i = 1, 2, \ldots, n\},$$

l'**apprendimento supervisionato** ha lo scopo di dedurre le informazioni sulla probabilità sconosciuta che genera i dati e utilizzarla per prevedere dati futuri: dato un nuovo input, fornire un output che abbia la maggior probabilità di correttezza. La funzione $h_{\mathcal{S}}$ che approssima la relazione teorica tra X e Y è detta **funzione di predizione** (o **predittore**).

Inoltre, si supponga che la funzione di predizione $h_{\mathcal{S}}$ abbia una forma predefinita che dipende dalla scelta di un vettore di parametri $\mathbf{w} \in \mathbb{R}^d$; si ricorrerà alla forma $h(\mathbf{w}; \cdot)$ per esprimere tale dipendenza. Il *problema di apprendimento* consiste nello scegliere dall'insieme di parametri di $\mathbf{w} \in \mathbb{R}^d$ quello in grado di ottenere la funzione di predizione $h(\mathbf{w}; \cdot)$ che meglio approssimi gli output dei dati, ossia che minimizzi gli errori di predizione:

$$h(\mathbf{w}; \mathbf{x}) \sim \mathbf{y}.$$

Tra i problemi di apprendimento supervisionato, si distinguono: **classificazione binaria**, se $Y = \{y_1, y_2\}$; **classificazione multipla**, se $Y = \{y_1, y_2, \dots, y_c\}$; **regressione**, se $Y = \mathbb{R}$.

Osservazione 1.1.2 *Così come appena mostrato, le definizioni e gli enunciati successivi faranno riferimento a un predittore scelto in un insieme di funzioni aventi forma predefinita, delle quali sarà esplicitata la dipendenza da un parametro* $\mathbf{w} \in \mathbb{R}^d$.

1.1.1 Problema di Minimizzazione del Rischio

Allo scopo di scegliere la funzione più adatta, che permetta di minimizzare gli errori di predizione, è necessario introdurre la definizione di *funzione loss*. Essa misura la discrepanza tra il valore reale dell'output e quello predetto dallo stimatore stesso. Siano due spazi $X \subseteq \mathbb{R}^{d_x}$ e $Y \subseteq \mathbb{R}^{d_y}$, rispettivamente di input e output. Data una funzione di predizione $h(\mathbf{w}; \cdot) : X \to Y$, il cui comportamento è determinato dal vettore dei parametri $\mathbf{w} \in \mathbb{R}^d$, è definita **funzione loss** (o **funzione costo**) un'applicazione

$$\begin{aligned} L : X \times Y \times Y &\longrightarrow [0, +\infty) \\ \big(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}\big) &\longmapsto L\big(\mathbf{x}, \mathbf{y}, \hat{\mathbf{y}}\big), \end{aligned}$$

dove $\hat{\mathbf{y}} := h(\mathbf{w}; \mathbf{x})$ è la predizione corrispondente all'input $\mathbf{x}$ tramite $h(\mathbf{w}; \cdot)$, tale che

$$L(\mathbf{x}, \mathbf{y}, \mathbf{y}) = 0 \qquad \forall \mathbf{x} \in X \quad \forall \mathbf{y} \in Y.$$

In particolare, data una coppia input-output $(\mathbf{x}, \mathbf{y})$, il valore $L\big(\mathbf{x}, \mathbf{y}, h(\mathbf{w}; \mathbf{x})\big)$ rappresenta il costo associato al divario tra la predizione del modello e il valore reale. La scelta della funzione loss da utilizzare è strettamente legata al tipo di problema in analisi.

Servendosi di tale funzione, che rappresenta l'errore compiuto da un predittore per una particolare coppia input-output, si definisce la misura della sua capacità globale. Sia lo spazio input-output $X \times Y$ caratterizzato da una probabilità $P : X \times Y \to [0, 1]$, dove $X \subseteq \mathbb{R}^{d_x}$ e $Y \subseteq \mathbb{R}^{d_y}$.

Data la funzione di predizione $h(\mathbf{w}, \cdot)$ e la funzione loss L, si definisce **rischio atteso**

$$R : \mathbb{R}^d \to \mathbb{R},$$

dipendente da un vettore di parametri $\mathbf{w} \in \mathbb{R}^d$ relativo a h,

$$R(\mathbf{w}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P}\left[\, L\big(\mathbf{x}, \mathbf{y}, f(\mathbf{w}; \mathbf{x})\big)\,\right] = \int_{\mathbb{R}^{d_x}\times\mathbb{R}^{d_y}} L\big(\mathbf{x}, \mathbf{y}, f(\mathbf{w}; \mathbf{x})\big)\, P(\mathbf{x}, \mathbf{y}) \quad d\mathbf{x}\, d\mathbf{y}.$$

A partire da questo, è possibile enunciare lo scopo del Machine Learning, su cui si basa la scelta del modello migliore per il problema di apprendimento considerato:

Proposizione 1.1.3 (Principio di Minimizzazione del Rischio Atteso) *Sia lo spazio input-output $X \times Y$ dotato della misura di probabilità $P : X \times Y \to [0, 1]$, dove $X \subseteq \mathbb{R}^{d_x}$ e $Y \subseteq \mathbb{R}^{d_y}$. Si consideri un insieme di funzioni $h(\mathbf{w}; \cdot)$, dipendenti dal vettore di parametri $\mathbf{w} \in \mathbb{R}^d$.*

La scelta della funzione di predizione migliore per il problema di apprendimento ricade su $h(\mathbf{w}^*; \cdot)$ tale che $\mathbf{w}^*$ sia soluzione del problema di minimizzazione del rischio atteso $R(\mathbf{w})$

$$\min_{\mathbf{w}\in\mathbb{R}^d} R(\mathbf{w}) \tag{1.1}$$

Tuttavia, generalmente la distribuzione di probabilità associata allo spazio $X \times Y$ è sconosciuta. Dunque, non essendo possibile minimizzare il rischio atteso (1.1), è necessario applicare il seguente *principio induttivo*: sostituire il rischio atteso $R(\mathbf{w})$ col cosiddetto *rischio empirico* $R_{\text{emp}}(\mathbf{w})$ (1.3) e approssimare la funzione di predizione $h(\mathbf{w}^*; \cdot)$ attraverso il nuovo problema di ottimizzazione.

Definizione 1.1.4 *Dato un insieme di dati $\mathcal{S} = \{z_1, z_2, \ldots, z_n\}$, è chiamata* ***densità di probabilità empirica*** *la densità di probabilità*

$$p_{emp}(z) := \frac{1}{N} \sum_{i=1}^{n} \delta_{z_i}(z)$$

dove $\delta_{\xi'}$ è la ***distribuzione delta di Dirac****, tale che, per ogni funzione*

$$\int \delta_{\xi'}(\xi) f(\xi)\, d\xi = f(\xi) \tag{1.2}$$

Si consideri un training set finito,

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) \in X \times Y \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \ : \ i = 1, \ldots, n\},$$

generato secondo la distribuzione di probabilità $P : X \times Y \to [0, 1]$.

Scelto un insieme di funzioni di predizione $h(\mathbf{w}; \cdot)$, definite da un vettore di parametri $\mathbf{w} \in \mathbb{R}^d$ e una funzione loss $L : X \times Y \times Y \to \mathbb{R}$, il **rischio empirico**

$$R_{\text{emp}} : \mathbb{R}^d \to \mathbb{R}, \tag{1.3}$$

dipendente da $\mathbf{w} \in \mathbb{R}^d$, è ottenuto come

$$\begin{aligned} R_{\text{emp}}(\mathbf{w}) &= \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{\text{emp}}} \left[L\big(\mathbf{x}, \mathbf{y}, h(\mathbf{w}; \mathbf{x})\big) \right] \\ &= \int_{X \times Y} L\big(\mathbf{x}, \mathbf{y}, h(\mathbf{x}; \mathbf{w})\big)\, p_{\text{emp}}(\mathbf{x}, \mathbf{y})\, d\mathbf{x}\, dy \\ &= \frac{1}{n} \sum_{i=1}^{n} L\big(\mathbf{x_i}, \mathbf{y_i}, h(\mathbf{w}; \mathbf{x_i})\big) \end{aligned}$$

dove p_{emp} è la *densità empirica*:

$$p_{\text{emp}}(\mathbf{x}, \mathbf{y}) := \frac{1}{N} \sum_{i=1}^{n} \delta_{\mathbf{x}_i}(\mathbf{x}) \delta_{\mathbf{y}_i}(\mathbf{y}).$$

Si osservi che la terza espressione è conseguenza diretta della proprietà (1.2). Essa, ponendo

$$f_i(\mathbf{w}) := L\big(\mathbf{x}_i, \mathbf{y}_i, h(\mathbf{w}; \mathbf{x}_i)\big) \quad \forall i \in \{1, 2, \ldots, n\}$$

può essere riscritta come

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w})$$

Infine, è possibile enunciare il *principio induttivo di minimizzazione del rischio empirico (ERM principle)*:

Proposizione 1.1.5 (Principio di Minimizzazione del Rischio Empirico) *Sia lo spazio input-output $X \times Y$ dotato della misura di probabilità sconosciuta $P : X \times Y \to [0,1]$, dove $X \subseteq \mathbb{R}^{d_x}$ e $Y \subseteq \mathbb{R}^{d_y}$. Si consideri un insieme di funzioni $f(\mathbf{w}; \cdot)$, dipendenti dal vettore di parametri $\mathbf{w} \in \mathbb{R}^d$.*

Dato il training set, costituito secondo la densità di probabilità P,

$$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) \in X \times Y \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \ : \ i = 1, \dots, n\},$$

è possibile scegliere come approssimazione della funzione di predizione per il problema di apprendimento $h(\mathbf{w}^*; \cdot)$ tale che $\mathbf{w}^*$ sia soluzione del problema di minimizzazione del rischio empirico $R_{emp}(\mathbf{w})$

$$\min_{\mathbf{w} \in \mathbb{R}^d} R_{emp}(\mathbf{w}) \tag{1.4}$$

Infatti, il rischio empirico può essere utilizzato per approssimare il rischio atteso, considerando un numero di esempi del training set sufficientemente grande. Ciò emerge dalla seguente proprietà, dimostrabile mediante il *teorema dei Grandi Numeri*:

Proprietà 1.1.6 *Sia n il numero di esempi all'interno del training set e sia h la funzione di predizione utilizzata, allora il rischio empirico converge in probabilità al rischio atteso:*

$$R_{emp}[h] \xrightarrow[n \to \infty]{P} R[h]$$

ossia

$$\forall \varepsilon > 0 \quad \lim_{n \to \infty} P\left(|R_{emp}[h] - R[h]| > \varepsilon\right) = 0$$

Tuttavia, la minimizzazione dell'errore compiuto sul training set, il rischio empirico, non può essere l'unico requisito per la determinazione del modello migliore per il problema di apprendimento: si parla della *capacità di generalizzazione*.

1.1.2 Bias-Variance Dilemma e Regolarizzazione del Rischio Empirico

I testi di riferimento per tale sezione sono [14, 15].

Lo scopo principale del Machine Learning è trovare un modello dotato di capacità di **generalizzazione**, ossia dell'abilità di predire correttamente gli output anche per nuovi input, non utilizzati per l'addestramento. Mentre il *principio ERM*

(1.4) permette di diminuire la misura dell'errore sul training set, il *training error*, è necessario trovare una strategia che garantisca ciò anche per il cosiddetto *errore di generalizzazione*, anche chiamato *test error*.

In generale, può presentarsi uno dei seguenti casi:

1. **underfitting**, in cui si introduce un **bias** nel modello, il quale non è in grado di ottenere un errore di training sufficientemente basso;

2. **overfitting**, dove il modello si adatta perfettamente ai dati del training set, ma aumenta la sua **varianza**: piccole variazioni sui dati forniti per l'addestramento, possono portare a grandi modifiche del modello. In tal caso, si perde la capacità di generalizzazione, aumentando la distanza tra training error e test error.

Dunque, è necessario sviluppare delle strategie che permettano di raggiungere il giusto compromesso tra i due casi. Nella statistica classica, questo problema è chiamato **Bias-Variance Dilemma**.

Per controllare tale situazione è necessario selezionare un adeguato **spazio di ipotesi**, insieme di funzioni che possono essere selezionate per il modello di apprendimento. Scegliere un insieme molto limitato, ad esempio considerando solo funzioni lineari, potrebbe portare ad una situazione di underfitting: la dipendenza funzionale tra input e output non emerge dai dati, ma da un bias introdotto dallo spazio scelto. Viceversa, attraverso un insieme troppo grande si potrebbe ottenere overfitting: esisterà sempre una funzione in grado di adattarsi perfettamente al training set.

Ulteriori strategie che permettono di raggiungere lo scopo sono i **metodi di regolarizzazione**. Essi permettono di migliorare la capacità di generalizzazione del modello a discapito dell'errore compiuto sull'insieme di addestramento. Tali metodi sono considerati efficienti se permettono di ridurre la varianza, non aumentando eccessivamente il bias introdotto. In particolare, metodi di regolarizzazione consistono nell'aggiungere alla funzione obiettivo $R_{\text{emp}}(\mathbf{w}; \mathcal{S})$, dipendente dal training set $\mathcal{S}$ e dai parametri del modello $\mathbf{w} \in \mathbb{R}^d$, un **termine di penalizzazione** $\Omega(\mathbf{w})$ costruito attraverso la norma dei parametri stessi. Il problema di minimizzazione diventa:

$$\min_{\mathbf{w} \in \mathbb{R}^d} R_{\text{emp}}(\mathbf{w}; \mathcal{S}) + \lambda \Omega(\mathbf{w}) \tag{1.5}$$

dove

- $\mathcal{S}$ è l'insieme di addestramento;
- $\Omega(\mathbf{w}) = ||\mathbf{w}||_q^q$, con $q > 0$, è la penalità, la cui scelta influenza il tipo di soluzioni. Solitamente si sceglie $q = 1$ oppure $q = 2$ e $\lambda = \frac{1}{2}$. In quest'ultimo caso, λ è chiamato **weight decay**;
- $\lambda \in [0, +\infty)$ è l'iperparametro che stabilisce il peso di $\Omega(\mathbf{w})$ nella funzione obiettivo. Se $\lambda = 0$, non è presente regolarizzazione; viceversa, maggiore sarà il valore di λ, minori dovranno essere le componenti del vettore $\mathbf{w}$ che determina il modello.

Questa modifica della funzione obiettivo permette di esprimere preferenze sul parametro da scegliere e corrisponde ad una forma più generale di restrizione dello spazio di ipotesi. Infatti, il problema (1.5) è equivalente alla sua forma lagrangiana

$$\begin{aligned} &\min_{\mathbf{w} \in \mathbb{R}^d} \quad R_{\text{emp}}(\mathbf{w}; \mathcal{S}) \\ &\Omega(\mathbf{w}) \leq \rho \end{aligned}$$

per una qualche costante $\rho > 0$.

1.2 Deep Learning e Reti Neurali Convoluzionali

1.2.1 Artificial Neural Networks (ANN)

Il *Deep Learning* è una branca fondamentale del Machine Learning, il cui nome deriva da una particolare caratteristica dell'architettura dei modelli Artificial Neural Networks: la profondità ("depth") data dall'*organizzazione in strati* [15]. Le informazioni riportate in questa sezione sono tratte dai testi [14, 15].

Definizione 1.2.1 *Le **Artificial Neural Networks (ANN)** sono modelli di ispirazione biologica, il cui funzionamento simula l'elaborazione delle informazioni del cervello umano. Esse sono costituite dall'unione di **neuroni artificiali**, o **processing elements** (PE), organizzati in strati, i **layers**, e collegati attraverso coefficienti, detti **pesi**. Tale struttura neurale fonda la sua conoscenza individuando schemi e relazione nascoste tra i dati, apprese attraverso l'esperienza.*

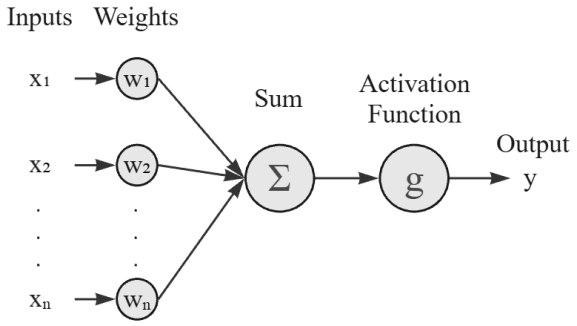


Figura 1.1: Struttura del neurone artificiale [14]

Il primo modello di questo tipo fu sviluppato da *W. S. McCulloch* e *W. Pitts* [22]. Esso è caratterizzato da un unico elemento, il **neurone di McCulloch e Pitts**, in grado di restituire come output uno dei due possibili stati $\{0, 1\}$, in base all'eventualità che la somma dei valori in input raggiunga una certa soglia prefissata. Tale modello pone le basi per la nascita e lo sviluppo del concetto di neurone artificiale.

I **processing elements (PE)**, o **neuroni artificiali**, presentano una particolare struttura, mostrata in *Figura 1.1*, che richiama il *neurone biologico*. Essi sono definiti da una **funzione di attivazione** $g : \mathbb{R}^d \to \mathbb{R}$, da **pesi** $\mathbf{w} = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$ e da una certa soglia, il **bias**, $b \in \mathbb{R}$, che vengono utilizzata in tal modo: considerando un campione in input $\mathbf{x} = (x_1, x_2, \ldots x_d) \in \mathbb{R}^d$, il neurone artificiale restituirà come output il valore assunto da g valutano nel cosiddetto **segnale di attivazione**:

$$y(\mathbf{x}) = g\left(\sum_{i=1}^{d} w_i x_i + b\right). \tag{1.6}$$

Tra le funzioni di attivazione più comuni, si ricordano

- la **funzione segno**

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

  il cui modello corrispondente, **Linear Threshold Unit (LTU)** o **percettrone**, è in grado matematicamente di rappresentare una classificazione binaria mediante un iperpiano di $\mathbb{R}^d$,

$$y(\mathbf{x}) = \begin{cases} +1 & \text{se } \sum_{i=1}^{d} w_i x_i + b \geq 0 \\ -1 & \text{se } \sum_{i=1}^{d} w_i x_i + b < 0 \end{cases}$$

- **funzione sigmoide o logistica**, principalmente utilizzata per modelli il cui output rappresenta una probabilità,

$$g(t) = \frac{1}{1+e^{-t}} \in (0,1) \quad \forall t \in \mathbb{R}.$$

- **funzione ReLU (Rectified Linear Unit)**

$$g(t) = \max(0,t) \in [0,\infty) \quad \forall t \in \mathbb{R}. \tag{1.7}$$

  Grazie ad essa, tutti i valori negativi vengono trasformati in 0. Per questo motivo, esiste una versione simile che prende in considerazione la negatività degli input:

$$g(t) = \begin{cases} at & \text{se } t < 0 \\ t & \text{se } t \geq 0 \end{cases}$$

  Se $a = 0.01$ si parla di **Leaky ReLu**, altrimenti, per $a > 0$, **Randomized ReLU**.

- **funzione identità**, che semplicemente restituisce il valore in input stesso:

$$g(t) = t \in (-\infty, +\infty) \qquad \forall t \in \mathbb{R}.$$

Tuttavia, una ANN costituita da un unico neurone artificiale è spesso troppo semplice per stimare la natura dei dati secondo il grado di accuratezza desiderato. Ad esempio, nell'ambito della classificazione binaria, le due classi potrebbero non essere linearmente separabili attraverso il modello *LTU*. Dunque, per ovviare a tale problema è possibile considerare una struttura più complessa, che permetta la mappatura degli input iniziali in uno spazio appropriato, dove la separabilità lineare degli input trasformati è garantita. Matematicamente, si tratta di applicare al vettore in input $\mathbf{x} \in \mathbb{R}^d$ la composizione di funzioni $g^{(i)}$ che definiscono i neuroni artificiali, la cui legge, inizialmente dipendente da vettori di parametri $\mathbf{w}^{(i)}$, può essere scelta attraverso la fase di addestramento della rete stessa:

$$g(\mathbf{x}) = \left(g^{(L)} \circ g^{(L-1)} \circ \cdots \circ g^{(1)}\right)(\mathbf{x}).$$

La motivazione alla base di questa scelta è data dal *Teorema di Approssimazione Universale* che garantisce che una rete neurale feedforward, con almeno uno strato

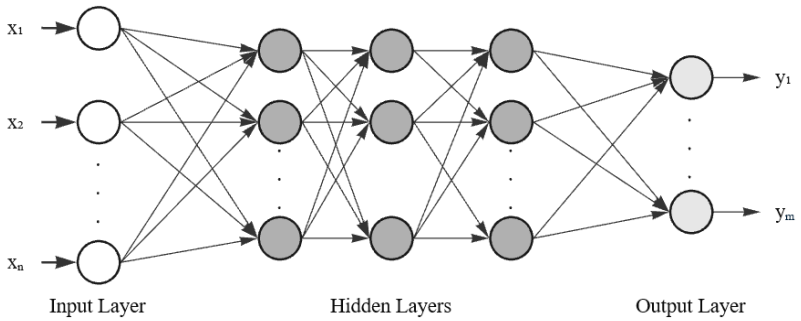


Figura 1.2: Struttura stratificata della Rete Neurale Artificiale.

nascosto dotato di un numero sufficiente di neuroni e di funzioni di attivazione limitate e non lineari, può approssimare arbitrariamente bene la funzione desiderata, supponendo che essa verifichi certe ipotesi, come la misurabilità o la continuità [18].

Tale architettura definisce le **Multilayer Neural Networks**, la cui struttura *a strati* è legata alla catena di composizioni di processing elements (PE). Essa può essere rappresentata attraverso il *grafo diretto aciclico* 1.2 costituito da

- d nodi iniziali che rappresentano le componenti dell'input

$$\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d;$$

- un insieme di nodi per i neuroni artificiali, organizzati in L strati che si susseguono. I primi $(L-1)$ sono **hidden layers**, in quanto i risultati non saranno restituiti, ma verranno utilizzati dallo strato successivo, mentre l'ultimo sarà l'**output layer**;

- un insieme di archi orientati e pesati, che rappresentano le connessioni tra i nodi.

Definizione 1.2.2 *Una rete neurale è detta* ***feedforward*** *se le connessioni avvengono solo dai neuroni di uno strato a quelli dello strato immediatamente successivo. Inoltre, essa è detta* ***fully connected*** *se ogni coppia di neuroni appartenenti a strati consecutivi è collegata.*

Viceversa, si parla di reti ***ricorrenti*** *se sono presenti anche connessioni tra neuroni dello stesso strato o orientati verso strati precedenti. Queste sono chiamate* ***connessioni feedback****.*

Considerando una rete neurale feedforward, il suo funzionamento può essere matematicamente formalizzato in questo modo: si considerino

- $l \in \{1, 2, \dots L\}$ indice relativo allo strato l della rete;
- $w_{ji}^{(l)}$ peso dell'arco orientato entrante nel neurone j dello strato l e uscente dal neurone i-esimo dello strato $(l-1)$;
- $g_j^{(l)} : \mathbb{R} \to \mathbb{R}$ funzione di attivazione relativa al neurone j dello strato l. Essa permetterà di calcolare l' output relativo al neurone attraverso la formula (1.6), che coinvolge i pesi $w_{ji}^{(l)}$, gli input dati dai neuroni dello strato precedente e una soglia $w_{j0}^{(l)}$.

In particolare, indicando con $a_j^{(l)}$ la somma pesata descritta in precedenza e con $z_j^{(l)}$ l'output del neurone j nello strato l, la formula di aggiornamento di una rete neurale può essere espressa in tal modo:

- allo stato $l = 1$, per ogni $j \in \{1, 2, \dots, N^{(1)}\}$

$$a_j^{(1)} = \sum_{i=1}^{d} w_{ji}^{(1)} x_i - w_{j0}^{(1)}$$
$$z_j^{(1)} = g_j^{(1)} \left(a_j^{(1)} \right)$$

- allo stato $l \in \{2, 3, \dots, L\}$, , per ogni $j \in \{1, 2, \dots, N^{(l)}\}$

$$a_j^{(l)} = \sum_{i=1}^{N^{(l-1)}} w_{ji}^{(l)} z_i^{(l-1)} - w_{j0}^{(l)}$$
$$z_j^{(l)} = g_j^{(l)} \left(a_j^{(l)} \right)$$

dove $N^{(l)}$ è il numero di neuroni presenti nello strato (l).

Inoltre, essa può essere espressa in *forma matriciale*. Siano

- $\mathbf{W}^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l-1)}}$ matrice definita dai pesi dello l,

$$\mathbf{W}^{(l)} = \left\{ w_{ji}^{(l)} \right\}_{j=1,2,\dots N^{(l)};\, i=1,2,\dots,N^{(l-1)}}$$

- il vettore delle soglie, *bias*, dello strato l,

$$\mathbf{w}_0^{(l)} = \left\{ w_{j0}^{(l)} \right\}_{j=1,2,\dots N^{(l)}}$$

- la mappa costituita dalle funzioni di attivazione dello strato l, che rappresenta gli output di tutti i neuroni presenti,

$$\mathbf{g}^{(l)} : \mathbb{R}^{N^{(l-1)}} \to \mathbb{R}^{N^{(l-1)}} \tag{1.8}$$

$$\mathbf{z} \mapsto \left(g_1^{(l)}(\mathbf{z}), g_2^{(l)}(\mathbf{z}), \ldots, g_{N^{(l)}}^{(l)}(\mathbf{z})\right) \tag{1.9}$$

allora, per ogni $l \in \{1, 2, \ldots L\}$,

$$\mathbf{A}^{(l)} = \mathbf{W}^{(l)}\mathbf{z}^{(l-1)} - \mathbf{w}_0^{(l)} \tag{1.10}$$

$$\mathbf{z}^{(l)} = \mathbf{g}^{(l)}\left(\mathbf{A}^{(l)}\right) \tag{1.11}$$

L'ultimo strato di una rete neurale e la loss considerata per il calcolo dell'errore compiuto dipendono dal tipo di problema in analisi. Ad esempio, si consideri il caso della *classificazione multiclasse* su C classi, in cui, dato in input un campione $\mathbf{x} \in \mathbb{R}$, si denota con $\mathbf{z} = (z_1, z_2, \ldots, z_C) \in \mathbb{R}^C$ l'output corrispondente generato dal modello, grazie all'ultimo strato costituito da C neuroni, uno per ogni classe disponibile. Inizialmente, sarà applicata la funzione di attivazione Softmax (1.12) per ottenere una distribuzione di probabilità discreta in cui la componente k-esima $\frac{e^{z_k}}{\sum_{c=1}^{C} e^{z_c}}$ rappresenta la probabilità di appartenenza del campione alla classe k.

Definizione 1.2.3 *La funzione $g : \mathbb{R}^m \to \mathbb{R}^m$ è detta* ***funzione Softmax*** *se definita come*

$$g(x_1, x_2, \ldots, x_m) = \left(\frac{e^{x_1}}{A}, \frac{e^{x_2}}{A}, \ldots, \frac{e^{x_m}}{A}\right) \in \mathbb{R}^d \tag{1.12}$$

dove A rappresenta il fattore di normalizzazione $A = \sum_{i=1}^{m} e^{x_i}$.

Essa viene principalmente utilizzata per restituire una distribuzione di probabilità attraverso le componente del vettore in input $\mathbf{x} = (x_1, x_2, \ldots, x_m)$.

Successivamente, tale distribuzione sarà confrontata con una funzione target p prefissata attraverso la *Cross-Entropy Loss*, che rappresenta la sua differenza da p:

Definizione 1.2.4 *Date due distribuzioni discrete P e Q, è definita* ***Cross-Entropy*** *la misura*

$$H(p, q) = -\mathbb{E}_{\alpha \sim P}[\log(Q(\alpha))] = -\sum_{\alpha} P(\alpha)\, \log\left(Q(\alpha)\right)$$

Invece, per problemi di *regressione*, denotando con $z_i \in \mathbb{R}$ e $y_i \in \mathbb{R}$ rispettivamente l'output generato dalla rete per il campione i e il valore realmente associato, è possibile adottare, ad esempio, la **loss quadratica**

$$L(y_i, z_i) = (y_i - z_i)^2 \quad \forall i \in \{1, 2, \ldots, n\}$$

il cui rischio empirico rappresenta la misura **Mean Square Error (MSE)**.

Scelta di Parametri e Iperparametri del Modello

L'architettura di una rete neurale multistrato dipende dalla scelta degli **iperparametri** prefissati, quali il numero di strati L e la quantità $N^{(l)}$ di neuroni presente in ognuno di essi, e da **parametri** incogniti (pesi degli archi e soglie).

I primi devono essere scelti attraverso opportune tecniche che coinvolgono un validation set, ricavato selezionando casualmente una parte del training set. Solitamente, si ricorre alla scelta di un valore che riduca l'errore compiuto sull'insieme di validazione dalla rete, addestrata sul restante insieme di addestramento.

Dopo aver fissato opportunamente gli iperparametri della rete neurale, è necessario calcolare i parametri attraverso la **fase di addestramento** del modello. Essa prevede la risoluzione del problema di minimizzazione del rischio empirico (1.4), eventualmente regolarizzato per ridurre l'overfitting sul training set (1.5). Dunque, è necessario definire una funzione loss opportuna al problema in analisi.

Considerando le *feedforward neural networks*, il cui flusso delle informazioni è lineare (dal primo verso l'ultimo strato), la risoluzione del problema di minimizzazione per la scelta dei parametri può avvenire attraverso un processo iterativo, il *metodo del di discesa del Gradiente Stocastico (SGD)*, di cui si parlerà nella *Sezione 2.2*. In particolare, ad ogni iterazione k di addestramento del modello saranno compiuti i seguenti passi:

1. **forward pass**, in cui, fissati i parametri correnti $\mathbf{w}^{(k)}$, si calcola il rischio empirico del modello applicando la *forward propagation* su un sottoinsieme del training set;

2. **backward pass**, dove si ottiene il corrispondente gradiente rispetto ai parametri della rete grazie all'*algoritmo back-propagation*;

3. **metodo SGD** per la scelta di un nuovo vettore di parametri $\mathbf{w}^{(k+1)}$, di cui si parlerà nel *capitolo 2*.

Infatti, la particolare struttura di questa rete permette di distinguere due differenti processi.

Dato un input $\mathbf{x} \in \mathbb{R}^d$ e fissati i parametri del modello, la **forward propagation** rappresenta il flusso di informazioni attraverso gli strati nascosti, che, a partire da $\mathbf{x}$, permette di ottenere l'output ad esso associato. Nel processo di training, esso comprenderà anche il calcolo dell'errore (funzione loss) corrispondente.

Viceversa, l'algoritmo di **back-propagation** (o di **retropropagazione del gradiente**), permette il calcolo del gradiente dell'errore risultante rispetto ai parametri della rete neurale, valutato numericamente in quelli correnti. Questo avviene attraverso un flusso a ritroso (letteralmente, *backward*) della catena costruita dalla funzione loss stessa e dalla composizione delle funzioni che costituiscono la rete (1.10). Questo procedimento, semplice e poco costoso, è giustificato dalla cosiddetta *regola della catena* per il calcolo del gradiente di una funzione composta.

In particolare, considerato un campione in input $\mathbf{z}^{(0)}$, la funzione loss ℓ e la struttura composta che definisce la rete (1.10), il calcolo dell'errore nella classificazione (o regressione) sarà ottenuto attraverso le seguenti operazioni

$$\mathbf{z}^{(0)} \xrightarrow{\mathbf{g}^{(1)}(\mathbf{W}^{(1)})} \mathbf{z}^{(1)} \xrightarrow{\mathbf{g}^{(2)}(\mathbf{W}^{(2)})} \dots \xrightarrow{\mathbf{g}^{(L)}(\mathbf{W}^{(L)})} \mathbf{z}^{(L)} \xrightarrow{\ell(\mathbf{z}^{(L)},y)} \ell(\mathbf{W}) \tag{1.13}$$

dove $\mathbf{W}$ rappresenta l'insieme di tutti i parametri. Fissato uno strato l, la *regola della catena per il calcolo del gradiente delle funzioni composte* rispetto a $\mathbf{W}^{(l)}$ afferma che

$$\nabla_{\mathbf{W}^{(l)}} \ell(\mathbf{W}) = \left(J_{\mathbf{W}^{(l)}} \mathbf{z}^{(l)}\right)^{\mathrm{T}} \left(J_{\mathbf{z}^{(l)}} \mathbf{z}^{(l+1)}\right)^{\mathrm{T}} \cdots \left(J_{\mathbf{z}^{(L-1)}} \mathbf{z}^{(L)}\right)^{\mathrm{T}} \left(\nabla_{\mathbf{z}^{(L)}} \ell(\mathbf{W})\right) \tag{1.14}$$

dove $J_p q$ è la matrice Jacobiana di q rispetto ad p,

$$[J_p q]_{ij} = \frac{\partial [q]_i}{\partial [p]_j}.$$

Da essa ne deriva

$$\nabla_{\mathbf{W}^{(l)}} \ell(\mathbf{W}) = \left(J_{\mathbf{W}^{(l)}} \mathbf{z}^{(l)}\right)^{\mathrm{T}} \left(\nabla_{\mathbf{z}^{(l)}} \ell(\mathbf{W})\right). \tag{1.15}$$

Allo stesso modo,

$$\nabla_{\mathbf{z}^{(l-1)}} \ell(\mathbf{W}) = \left(J_{\mathbf{z}^{(l-1)}} \mathbf{z}^{(l)}\right)^{\mathrm{T}} \nabla_{\mathbf{z}^{(l)}} \ell(\mathbf{W}). \tag{1.16}$$

Grazie alla struttura ricorsiva data da 1.15 e 1.16, è possibile calcolare $\nabla_{\mathbf{W}^{(l)}}\ell(\mathbf{W})$ per ogni strato $l \in \{1, 2, \dots L\}$ ripercorrendo a ritroso il grafo computazionale (1.13): ogni $\nabla_{\mathbf{z}^{(l-1)}\ell(\mathbf{W})}$ permette di calcolare $\nabla_{\mathbf{W}^{(l)}}\ell(\mathbf{W})$ (1.15) e $\nabla_{\mathbf{z}^{(l-1)}}\ell(\mathbf{W})$ (1.16), che a sua volta sarà utilizzato per l'iterazione a ritroso successiva. Ad ogni passo, per l'implementazione della retropropagazione del gradiente, sarà necessario conoscere unicamente come moltiplicare per uno Jacobiano [11].

Fenomeno di Vanishing ed Exploding Gradient

Siccome per ogni strato l, con $l \in \{1, 2, \dots L\}$ il calcolo dei gradienti $\nabla_{\mathbf{W}^{(l)}}\ell(\mathbf{w}^{(k)})$, ovvero $\nabla_{\mathbf{W}^{(l)}}\ell(W)$ calcolato in $\mathbf{w}^{(k)}$, è determinato dal prodotto di $L - l + 2$ termini, la cui formula è riportata in (1.14), negli strati iniziali potrebbero verificarsi due particolari fenomeni diametralmente opposti: l'annullamento o l'esplosione del gradiente, rispettivamente denotati come *vanishing gradient* ed *exploding gradient*.

Il fenomeno del **vanishing gradient** si verifica quando i termini del prodotto sono prevalentemente molto piccoli, inferiori ad uno. Dopo ripetute moltiplicazioni, i gradienti calcolati potrebbero assumere valori talmente ridotti da rendere difficile la determinazione della direzione di discesa per l'aggiornamento dei parametri tramite *metodo SGD*.

Al contrario, termini troppo grandi coinvolti nella moltiplicazione, **exploding gradient**, renderebbero instabile l'apprendimento a causa di aggiornamenti caratterizzati da passi troppo grandi.

Il fenomeno del *vanishing gradient* può essere contrastato, ad esempio, adottando la *Batch Normalization*, descritta successivamente, o introducendo le *shortcut connections*, presentate in *Sezione 1.2.3*. Al contrario, per eliminare l'*exploding gradient*, è possibile usare le **euristiche di gradient clipping**, ricordando che il gradiente serve per ottenere informazioni sulla direzione da percorrere e non sulla lunghezza di passo. Esistono differenti strategie, tra le quali si ricorda la più semplice [25]: definita una soglia positiva τ e indicato con $\mathbf{W}$ l'insieme dei parametri di ogni strato, se $\|\nabla_{\mathbf{W}}\ell(\mathbf{w}^{(k)})\| > \tau$, $\nabla_{\mathbf{W}}\ell(\mathbf{w}^{(k)})$ sarà riscalato attraverso l'operazione

$$\nabla_{\mathbf{W}}\ell(\mathbf{w}^{(k)}) \leftarrow \tau \frac{\nabla_{\mathbf{W}}\ell(\mathbf{w}^{(k)})}{\|\nabla_{\mathbf{W}^{(l)}}\ell(\mathbf{w}^{(k)})\|}.$$

Batch Normalization

Al fine di migliorare la difficoltà di addestramento di reti neurali molto profonde, si introduce la **Batch Normalization** [19], tecnica che prevede la *riparametrizzazione adattiva* del valore assunto dalle immagini date in input alle funzioni di attivazione $\mathbf{g}^{(l)}$. Sia $\mathcal{N}_k$ il mini-batch all'iterazione k dell'addestramento, sia $\mathbf{z}_i^{(l)}$ l'output generato dallo strato l-esimo per il campione $i \in \mathcal{N}_k$, definito secondo l'espressione (1.10), il metodo prevede la normalizzazione di ogni $\mathbf{A}_i^{(l)}$ attraverso le operazioni elemento per elemento, denotate con $\mathrm{BN}_l(\,\cdot\,)$, tali che

$$\mathrm{BN}_l\left(\mathbf{A}_i^{(l)}\right) = \frac{\mathbf{A}_i^{(l)} - \boldsymbol{\mu}^{(l)}}{\boldsymbol{\sigma}^{(l)}},$$

dove $\boldsymbol{\mu}^{(l)}$ e $(\boldsymbol{\sigma}^{(l)})^2$ sono rispettivamente la media e la varianza calcolate separatamente per ogni componente di $\{\mathbf{A}_i^{(l)} : i \in \mathcal{N}_k\}$, ossia

- $\boldsymbol{\mu}^{(l)} = \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} \mathbf{A}_i^{(l)}$;
- $\boldsymbol{\sigma}^{(l)} = \sqrt{\delta + \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} \left[\mathbf{A}_i^{(l)} - \boldsymbol{\mu}^{(l)}\right]^2}$, in cui $\delta > 0$ è una costante per evitare problemi legati al calcolo del gradiente, indefinito per $\sqrt{z}$ quando $z = 0$; solitamente, δ è posta come $\delta = 10^{-8}$.

Attraverso la Batch Normalization, l'output del campione i relativo allo strato l, indicato con $\bar{\mathbf{z}}_i^{(l)}$, sarà ottenuto secondo la formula ricorsiva

$$\begin{aligned} \mathbf{A}_i^{(l)} &= \mathbf{W}^{(l)} \bar{\mathbf{z}}^{(l-1)} - \mathbf{w}_0^{(l)} \\ \bar{\mathbf{z}}_i^{(l)} &= \mathbf{g}^{(l)} \left(\mathrm{BN}_l(\mathbf{A}_i^{(l)}) \right) \end{aligned}$$

per ogni $i \in \mathcal{N}_k$, per ogni $l \in \{1, 2, \ldots, L\}$.

Questa strategia permette di apportare differenti vantaggi. Innanzitutto, essa ostacola aggiornamenti dei parametri con l'unica funzione di aumentare la media o la varianza dei dati, in quanto queste operazioni verrebbero comunque rimosse. Inoltre, la Batch Normalization aggiunge rumore additivo e moltiplicativo agli output degli strati nascosti, che può ridurre l'overfitting della rete. Infine, essa permette di aumentare la stabilità del gradiente, diminuendo la probabilità che si verifichi il fenomeno del *vanishing gradient*. Infatti, gli output degli strati sono riportati in un insieme controllato, evitando zone delle funzioni di attivazione completamente prive

di pendenza, come, ad esempio, valori fortemente negativi per la *funzione ReLU (1.7)*.

In realtà, al fine di mantenere la stessa potenza espressiva della rete neurale, solitamente si sceglie di rimodellare la distribuzione gaussiana ottenuta dalla Batch Normalization, attraverso la formula

$$\bar{\mathbf{z}}_i^{(l)} = g_m^{(l)}\Big(\boldsymbol{\gamma}_m^{(l)} \odot \mathrm{BN}_l(\mathbf{A}_i^{(l)}) + \boldsymbol{\beta}_m^{(l)}\Big),$$

in cui le operazioni prodotto ($\odot$) e somma sono compiute elemento per elemento. Si ricordi che $g_m^{(l)}$ rappresenta la funzione di attivazione relativa all'm-esimo neurone artificiale dello strato l, che definisce la mappa $\mathbf{g}^{(l)}$ (formula (1.8)) per ogni $m \in \{1, 2, \ldots, N^{(l)}\}$. $\boldsymbol{\gamma}_m^{(l)}$ e $\boldsymbol{\beta}_m^{(l)}$ sono tensori di parametri aggiuntivi, della stessa dimensione di $\mathbf{A}_i^{(l)}$, anch'essi aggiornati attraverso la procedura di addestramento. Essi rappresentano rispettivamente la varianza e la media dei dati soggetti alla riparametrizzazione. In tal caso, i *bias* presenti in $\mathbf{A}_i^{(l)}$ devono essere omessi, in quanto ridondanti con il termine $\boldsymbol{\beta}_m^{(l)}$.

1.2.2 Convolutional Neural Networks (CNN)

A partire dall'operazione matematica di convoluzione è possibile costruire una particolare tipologia di reti neurali: le *Convolutional Neural Networks*. Esse vengono utilizzate nei casi in cui i dati in input siano espressi in forma di "griglia" 1D, come le time-series, ossia sequenze temporali tra le quali i file audio, o 2D, immagini intese come griglia di pixel. Di seguito, sarà considerato il caso bidimensionale.

Definizione 1.2.5 *Una **Convolutional Neural Network (ConvNet, CNN)** è una particolare rete neurale artificiale (ANN), utilizzata per la classificazione di immagini. Essa è caratterizzata dalla presenza di particolari strati che, sfruttando l'operazione di convoluzione (1.2.6), permettono di assegnare un certo grado di importanza per la classificazione ad alcuni aspetti dell'immagine, che comprendono caratteristiche sia ad alto che a basso livello. Questo riconoscimento avviene attraverso parametri presenti nei filtri che partecipano alle operazioni sull'immagine, apprendibili attraverso la classica fase di addestramento di una rete.*

La struttura di una CNN è mostrata in *Figura 1.3*.

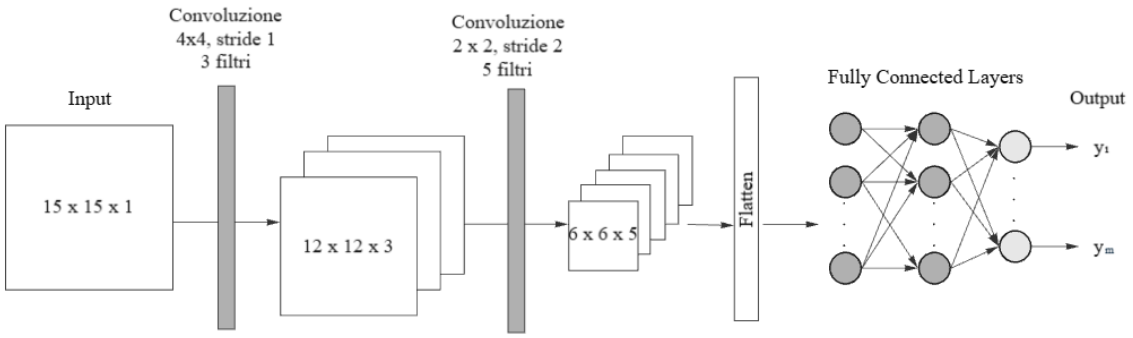


Figura 1.3: Architettura di una CNN.

Innanzitutto, si definisce l'operatore di convoluzione per immagini utilizzato nell'ambito del machine learning:

Definizione 1.2.6 *Si considerino due matrici bidimensionali*

- $I \in \mathbb{R}^{d_1 \times d_2}$*, immagine in input;*
- $K \in \mathbb{R}^{k_1 \times k_2}$*, detta* ***kernel*** *(o* ***filtro****), dove* $k_i \leq d_i$ *per* $i \in \{1, 2\}$*.*

È chiamata ***cross-correlation*** *tra* I *e* K *la matrice* $\tilde{I} \in \mathbb{R}^{(d_1-k_1+1)\times(d_2-k_2+1)}$ *tale che, per ogni* $i \in \{1, 2, \ldots, (d_1 - k_1 + 1)\}$ *e* $j \in \{1, 2, \ldots, (d_2 - k_2 + 1)\}$*,*

$$\tilde{I}_{(i,j)} = \sum_{m=1}^{k_1} \sum_{n=1}^{k_2} I_{(i+m,\, j+n)}\, K_{(m,n)}$$

Il corrispondente operatore è indicato con $(I * K)$*.*

Tale formula non corrisponde alla vera operazione di *convoluzione bidimensionale* nota in letteratura. Tuttavia, essa permette di ottenere la medesima matrice risultante, a discapito della proprietà di commutatività dell'operatore. Per questo motivo, nell'ambito del Machine Learning, essa viene utilizzata per l'implementazione della formula di convoluzione stessa. In particolare, la nuova matrice costruita in questo modo:

- il kernel K viene spostato lungo la matrice I per un numero di volte pari a $\mathbb{R}^{(d_1-k_1+1)\times(d_2-k_2+1)}$, in modo che ad ogni passo esso sia sovrapposto ad una sottomatrice differente;
- ogni componente della nuova matrice è ottenuta mediante la somma dei prodotti elemento per elemento tra il kernel e la porzione di I sovrapposta.

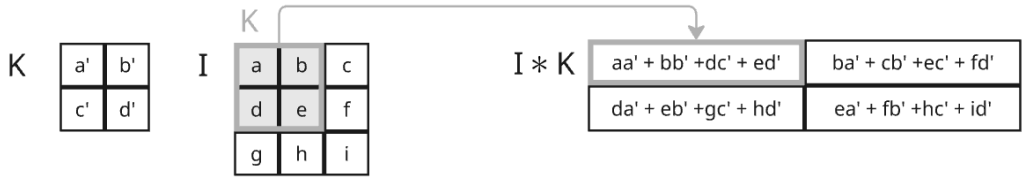


Figura 1.4: Esempio di operazione di convoluzione tra una matrice I e un kernel K.

Si osservi la *Figura 1.4* per maggiore chiarezza nell'operazione.

È possibile generalizzare questa operazione introducendo la lunghezza di passo (**stride**) che il kernel compie durante questo spostamento, riducendo ulteriormente la dimensione dell'immagine risultante.

Inoltre, se un kernel K è moltiplicato $(*)$ per un input $I \in \mathbb{R}^{d_x \times d_y \times C}$ costituito da C canali, ossia C immagini della stessa dimensione $d_x \times d_y$ impilate, il risultato corrisponderà alla somma componente per componente delle griglie ottenute dalla convoluzione di K per ogni canale di I:

$$(I * K)_{(i,j)} = \sum_{c=1}^{C} \sum_{m=1}^{k_1} \sum_{n=1}^{k_2} I_{(i+m,\, j+n,\, c)}\, K_{(m,n)}$$

Esistono due tipologie di strati che caratterizzano una CNN: *convolutional* e *pooling* layers.

Il **Convolutional Layer** è un particolare tipo di strato che prevede l'utilizzo dell'operazione di convoluzione attraverso uno o più filtri K_l, $l = 1, 2, \ldots, m$, sull'immagine I in input. Il risultato di questo strato sarà una "pila" di m griglie bidimensionali, ottenute da $I * K_l$ per ogni $l \in \{1, 2, \ldots, m\}$.

Le componenti delle matrici K saranno parametri calcolabili attraverso la classica fase di addestramento delle reti neurali, anch'essi soggetti alla retropropagazione del gradiente, in quanto sono coinvolti in relazioni di tipo lineare coi valori in input.

Solitamente, il kernel è scelto tra le matrici quadrate di dimensione dispari, il cui valore fa parte degli iperparametri del modello.

Compiere operazioni di questo tipo sulle immagini è vantaggioso per la comprensione dell'immagine nel suo complesso, catturando le caratteristiche di alto e basso livello. Convenzionalmente, la scelta della sequenza di questi strati prevede una prima ricerca delle caratteristiche di basso livello, di carattere generale, seguita da operazioni che permettono di cogliere aspetti più specifici dell'immagine.

Inoltre, ogni singolo parametro di questo strato viene riutilizzato più volte, rendendo il modello maggiormente adattabile all'intero dataset di addestramento.

Tra gli strati convoluzionali, è possibile scegliere l'impostazione **Valid Padding** (o *Zero Padding*) o **Same Padding**. La prima rappresenta l'operazione di convoluzione precedentemente descritta, con conseguente riduzione di dimensionalità della matrice risultate. Al contrario, la seconda prevede l'aumento delle dimensioni dell'immagine in input attraverso una cornice di zeri, scelta in modo tale che la matrice risultante abbia la sua stessa dimensione iniziale.

Un secondo layer è il **Pooling Layer**, che prevede l'utilizzo del kernel K come nello strato di convoluzione. Tuttavia, in tal caso le componenti della matrice risultante saranno ottenute, ad esempio, attraverso il valore massimo (**Max Pooling**) o medio (**Average Pooling**) della porzione di immagine I sovrapposta a K. L'utilizzo di questo strato è vantaggioso per molteplici aspetti. Innanzitutto, esso permette una riduzione di dimensionalità della matrice, diminuendo la potenza di calcolo richiesta e il rumore presente nei dati. Inoltre, esso non introduce alcun parametro o iperparametro da scegliere, perciò lo strato non sarà soggetto al processo di retropropagazione del gradiente.

A terminazione della CNN, il risultato bidimensionale degli strati precedentemente descritti, dopo un'opportuna trasformazione in forma vettoriale, può essere introdotto come input di una tradizionale *Fully Connected, Feed-Forward Neural Network*, seguita dalla funzione *Softmax*, trattate in precedenza. Questa rete permette di generare una dipendenza non lineare tra i parametri e gli input, grazie ad opportune funzioni di attivazione.

In conclusione, le *Reti Neurali Convoluzionali* permettono di classificare efficacemente le immagini apprendendo e valutando le loro caratteristiche spaziali (immagini) e temporali (time series) che determinano la figura nel suo complesso.

1.2.3 Residual Neural Networks (ResNet)

Utilizzare reti neurali più profonde permette generalmente di ottenere maggiori benefici. Tuttavia, la scelta del numero di strati genera un fenomeno non indifferente per le performance del metodo: la **degradazione dell'accuratezza sul training set**. Infatti, all'aumentare del numero di strati, è possibile che l'accuratezza sull'in-

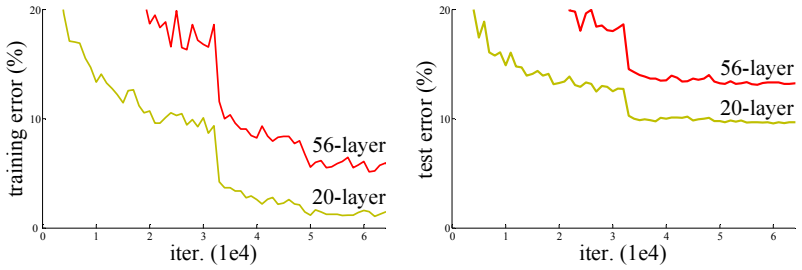


Figura 1.5: Confronto dei risultati di una classica rete neurale convoluzionale, con 20 o 56 strati, utilizzata per il problema di classificazione del dataset CIFAR-10 [16].

sieme di training saturi e si riduca rapidamente. Ovviamente, questo fenomeno non è causato dall'overfitting, poiché in tal caso l'errore commesso dal modello dovrebbe contemporaneamente diminuire sul trainig set e aumentare sull'insieme test. Le considerazioni precedenti sono osservabili in *Figura (1.5)*.

Nel 2015, un gruppo di ricercatori di Microsoft ha sviluppato una particolare classe di Reti Neurali Convoluzionali, le *Residual Neural Networks*, proposte nel paper [16] per risolvere il problema della degradazione dell'accuratezza sul training set.

Considerando modelli in cui non è presente il fenomeno svantaggioso di *vanishing* del gradiente, l'ipotesi principale è che questo errore sia causato dal fatto che non tutti i sistemi siano facilmente ottimizzabili. Ad esempio, si consideri una rete neurale poco profonda e si aggiungano ad essa identity layers. Teoricamente, questo nuovo modello non dovrebbe presentare prestazioni peggiori di quello iniziale, siccome gli strati identità mappano gli input in loro stessi. Tuttavia, il risultato ottenuto attraverso gli esperimenti dell'articolo [16] è ben diverso, suggerendo la difficoltà nel replicare mappature identità attraverso strati non lineari. Il **Deep Residual Learning** rappresenta una soluzione per questo problema, basata sull'ipotesi che l'ottimizzazione del modello residuo (1.17) sia più semplice dell'originale.

Si definisca $\mathcal{H}(\mathbf{x})$ una porzione del modello feedforward desiderato per l'apprendimento, costituita da uno o più strati consecutivi. Inoltre, si costruisca $\mathcal{F}(\mathbf{x})$ come la **funzione residua** di $\mathcal{H}$, ottenuta attraverso l'operazione

$$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x} \tag{1.17}$$

Osservazione 1.2.7 *Affinché questa operazione sia ben definita, si suppone che* $\mathbf{x}$ *e* $\mathcal{H}(\mathbf{x})$*, rispettivamente input e output del modello, presentino la stessa dimensio-*

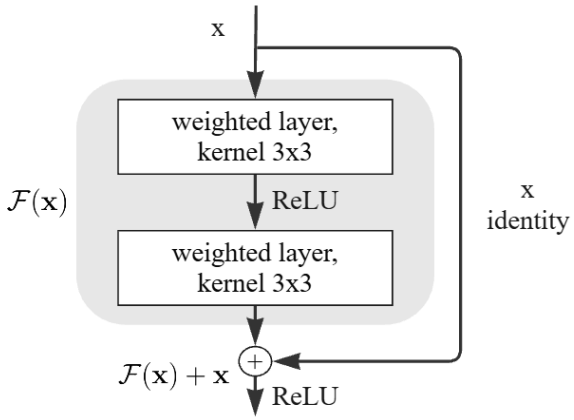


Figura 1.6: Esempio di building block del Residual Learning: struttura utilizzata per ResNet-18 e Resnet-34 [16].

nalità. Tuttavia, successivamente sarà presentata una generalizzazione di questa struttura.

La funzione $\mathcal{H}(\mathbf{x})$ può essere ottenuta ottimizzando $\mathcal{F}(\mathbf{x})$ e, solo successivamente, sommando ad essa $\mathbf{x}$. Infatti,

$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$$

Questa operazione può essere implementata attraverso le **Shortcut Connections**, connessioni strutturali che saltano uno o più strati, connettendo input e output della porzione di modello. In questo contesto, attraverso la mappa identità (**Identity Shortcut Connections**), esse copieranno il valore in input, il quale sarà sommato all'output degli strati presenti all'interno del cosiddetto **building block**, la cui struttura è mostrata in figura (1.6). Grazie a questa modifica, il ruolo degli strati presenti sarà differente: prima essi permettevano l'apprendimento del modello ottimale $\mathcal{H}(\mathbf{x})$, ora si adattano alla funzione residua $\mathcal{F}(\mathbf{x})$.

L'esplicitazione della mappa identità porta ad un vantaggio fondamentale nel caso in cui l'identity layer sia l'operazione migliore per il building block modellato rispetto al dataset considerato. Infatti, ottenere $\mathcal{F}(\mathbf{x}) \simeq 0$, sostituendo così gli strati non lineari con un unico identity layer $\mathcal{H}(\mathbf{x}) \simeq \mathbf{x}$, sarà più semplice rispetto a cercare di ottenere una mappa identità attraverso strati non lineari.

Inoltre, siccome la connessione shortcut *non dipende da parametri aggiuntivi*, sarà ancora possibile sfruttare il processo di retropropagazione del gradiente per

l'addestramento della nuova rete attraverso il metodo di Discesa del Gradiente Stocastico. Siccome la complessità computazionale del blocco non aumenta, il nuovo modello sarà confrontabile con la sua versione originale.

Attraverso un'analisi esaustiva sul ImageNet, è stato mostrato come questa nuova architettura, all'aumentare della profondità di una rete neurale convoluzionale, permetta di risolvere il problema di degradazione dell'accuratezza di training.

Sulla base di queste considerazioni, i ricercatori hanno proposto le *Residual Neural Networks (ResNet)*.

Definizione 1.2.8 *Le **Residual Neural Networks (ResNet)** sono reti neurali convoluzionali formate da una successione di building blocks, come quello mostrato in Figura 1.6. In particolare, questi ultimi sono costituiti da pochi strati convoluzionali, seguiti eventualmente dall'operazione di Batch Normalization (BN) e aventi funzione di attivazione ReLU (1.7). Inoltre, è presente una shortcut connection definita dalla mappa identità.*

In conclusione, escludendo la presenza di bias, ogni building block costituito da s strati convoluzionali avrà una struttura del tipo

$$\mathbf{y} = \mathcal{F}(\mathbf{x}; \{\mathbf{W}^{(i)}\}_{i=1}^{s}) + \mathbf{x} \tag{1.18}$$

dove

- $\{\mathbf{W}^{(i)}\}_{i=1}^{s}$ *sono le matrici dei parametri degli s strati;*
- *la funzione che definisce il blocco è*

$$\mathcal{F}(\mathbf{x}; \{\mathbf{W}^{(i)}\}_{i=1}^{s}) = \mathbf{W}^{(s)} * \Big(\Phi_{s-1} \circ \Phi_{s-2} \circ \cdots \circ \Phi_1\Big)(\mathbf{x}),$$

  dove Φ_i, per $i \in \{1, 2, \ldots, s-1\}$ è l'operatore (risp. con Batch Normalizzazione (BN))

$$\Phi_i(\mathbf{z}) := \sigma\Big(\mathbf{W}^{(i)} * \mathbf{z}\Big) \qquad \Big(risp. \quad \Phi_i(\mathbf{z}) := \sigma\Big(\mathrm{BN}_i\big(\mathbf{W}^{(i)} * \mathbf{z}\big)\Big)\Big);$$

- σ *è la funzione di attivazione Rectified Linear Unit (ReLU);*
- $*$ *rappresenta l'operazione di convoluzione (1.2.6);*

Osservazione 1.2.9 *Si consiglia di utilizzare $s > 1$. Infatti, per $s = 1$,*

$$\mathbf{y} = \mathbf{W}^{(1)} * \mathbf{x} + \mathbf{x}$$

presenta unicamente relazioni lineari, che non portano ad alcun vantaggio.

Come sottolineato in precedenza (*Osservazione 1.2.7*), queste reti riservate al caso in cui le immagini mantengano la stessa dimensione durante le operazioni. È possibile affrontare il problema della riduzione di dimensionalità causata dai convolutional layer attraverso tre differenti connessioni:

(A) **Zero-Padding Shortcut**, che sfrutta la strategia di Valid-Padding (*Sezione 1.2.2*) prima di ogni strato convoluzionale per rendere l'output $\mathcal{F}(\mathbf{x})$ della dimensione desiderata;

(B) **Projection Shortcut**, che diminuisce la dimensione di $\mathbf{x}$ ricorrendo alla convoluzione con la *proiezione identità* $\mathbf{W}_I$, matrice di parametri aggiuntivi di dimensione opportuna. Dunque, il risultato di ogni building block è

$$\mathbf{y} = \mathcal{F}(\mathbf{x}; \{\mathbf{W}^{(i)}\}_{i=1}^{s}) + \mathbf{W}_I * \mathbf{x} \tag{1.19}$$

(C) approccio ibrido, che alterna i metodi precedenti.

Confrontando i risultati delle strategie (*Tabella 1.1*), si può notare che l'aumento dei parametri attraverso le proiezioni identità porta ad un leggero miglioramento sull'addestramento del modello, a discapito della della dimensione del modello e della complessità computazionale, in termini di tempo e memoria utilizzati. Infatti, aggiungendo coefficienti pari a 0 su $\mathbf{x}$, si perde in essi la struttura di residual learning. Tuttavia, data la poca differenza nelle prestazioni, i metodi proposti successivamente presenteranno la prima versione proposta.

In particolare, per il dataset ImageNet[1], i ricercatori hanno sviluppato due Residual Networks, di 18 e 34 strati, ispirandosi alle reti neurali *VGG nets*, caratterizzate

[1]Il dataset ImageNet 2012 [28] è costituito da 1000 classi di immagini. Nell'articolo [16], sul training set è stata compiuta la *Data Augmentation*, per migliorare la generalizzazione del modello: ridimensionamento, selezionando casualmente un valore $[256, 480]$ come dimensione inferiore nell'intervallo; crop 224×224 su immagine stessa o sul suo capovolgimento orizzontale.

model	top-1 error	top-5 error
ResNet-34 A	25.03	7.76
ResNet-34 B	**24.19**	**7.40**
ResNet-34 C	24.52	7.46

Tabella 1.1: Tasso di errore (%, 10-crop testing) su ImageNet Validation [28]. Confronto tra varianti di ResNet-34, per trattare la riduzione della dimensionalità [16].

da una sequenza di strati convoluzionali con filtri di dimensione 3×3 e proposte nell'articolo [29]. Queste Residual Networks sono costruite attraverso building blocks aventi forma come in *Figura 1.6*, i cui strati che li determinano sono esplicitati nella *Tabella 1.2*.

Più nel dettaglio, la successione di strati e shortcut connections è mostrata in *Figura 1.7*. Per ottenere le dimensioni dell'immagine e il numero di canali desiderati dopo ogni strato, si utilizzano differenti strategie.

- **Projection Shortcut**, convoluzione 1×1 (1.19): per aumentare il numero di canali di $\mathbf{x}$ quando avviene il passaggio ad una nuova tipologia di building blocks (ad esempio, da c a $\tilde{c}$), si utilizza come primo shortcut una convoluzione 1×1, che non modifica la dimensione dell'immagine. Il numero di filtri presenti sarà pari alla nuova quantità di canali (ossia, $\tilde{c}$) generata dagli strati successivi. Si osservi che questa operazione introduce $\tilde{c}$ nuovi parametri che definiscono i filtri 1×1 utilizzati. Le restanti connessioni, rimarranno *Shortcuts Connections* classiche, definite dalla mappa identità (1.18).

- **Zero-Padding** precede ogni strato, in modo da ottenere la dimensione dell'immagine richiesta dalla struttura espressa in *Tabella 1.2*.

- In generale, ogni operazione convoluzionale presenterà **stride** 1. Tuttavia, il primo strato di ogni gruppo di buiding blocks, dal secondo gruppo in poi, avrà stride pari a 2 e Zero-Padding adeguato per dimezzare la dimensione dell'immagine rispetto alla sequenza di building bocks che lo precede.

Inoltre, per eliminare le difficoltà dell'ottimizzazione del modello legata ai *vanishing gradients*, si inserisce la *Batch Normalization* dopo ogni strato convoluzionale.

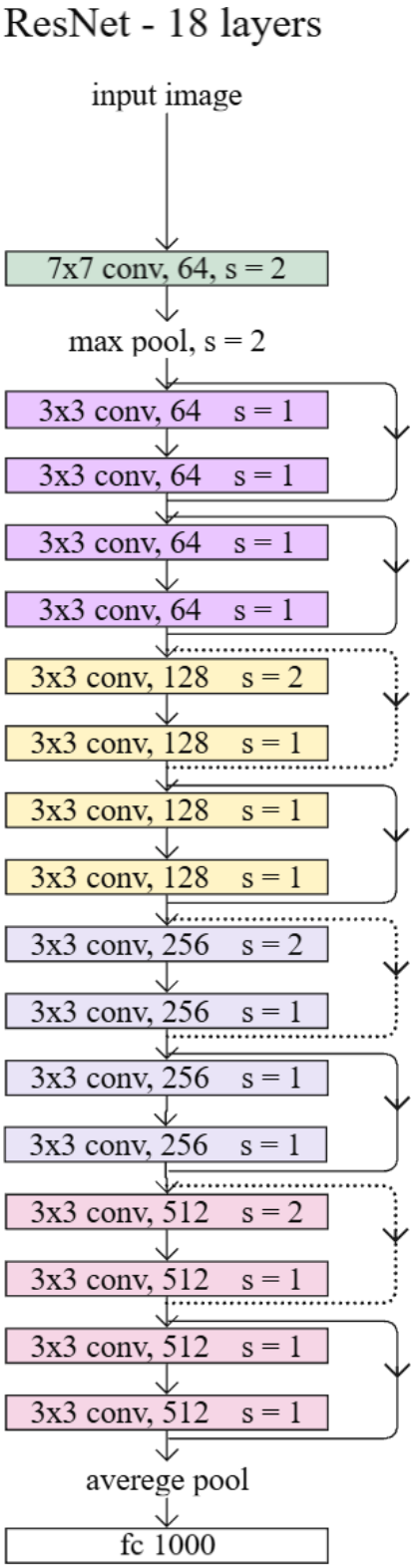


Figura 1.7: Architettura della ResNet-18. Le connessioni dei building blocks sono rappresentate attraverso frecce a lato. Se esse sono tratteggiate, sono presenti projection shortcuts*, altrimenti* identity shortcuts *classici. s rappresenta lo* stride *scelto per le operazioni di convoluzione dello stato.*

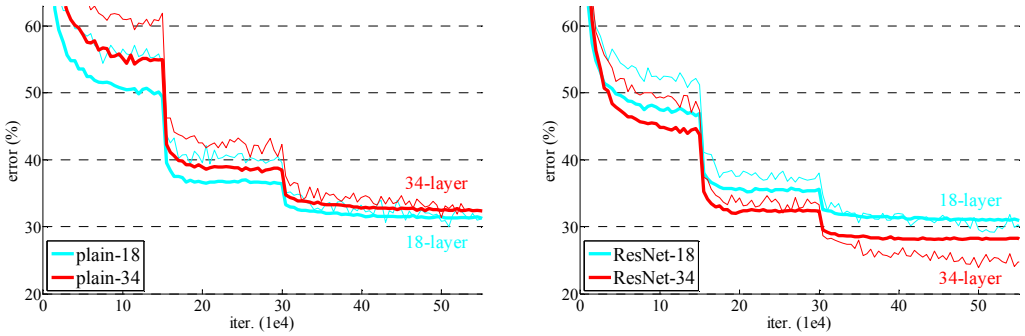


Figura 1.8: Errore (%) nell'addestramento di ImageNet. Le curve sottili rappresentano l'errore di training, mentre quelle più marcate mostrano l'errore di validazione calcolato sui ritagli centrali (center crops). A sinistra sono riportate le reti plain a 18 e 34 livelli; a destra le ResNet con 18 e 34 livelli. Nel grafico, le ResNet non presentano parametri aggiuntivi rispetto alle corrispondenti reti plain [16].

Infine, i risultati della classificazione sono ottenuti attraverso una rete Neurale Full Connected costituita da un unico strato avente 1000 neuroni artificiali, uno per ogni classe, seguita dalla funzione Softmax (1.12), per ottenere la distribuzione di probabilità di appartenenza dell'input ad una delle classi possibili.

L'aggiunta delle *shortcut connections* alla versione "plain", costruita con gli stessi strati ma senza connessioni identità, migliora l'adattamento del modello ai dati, risolvendo il problema della degradazione dell'accuratezza sul training set, dove presente. In particolare, la *Tabella 1.3* mostra come l'errore compiuto sul validation set sia molto inferiore nel caso della rete neurale più profonda (34 strati). Questo si rispecchia nell'aumento della training accuracy (*Figura 1.8*). Questa enorme differenza non è presente nel caso meno profondo, 18 strati. Tuttavia la ResNet-18 converge più velocemente rispetto alla corrispondente versione "plain" (*Figura 1.8*).

Deeper Residual Networks

Se si considerano Reti Neurali Convoluzionali troppo profonde, l'utilizzo di proiezioni shortcut e il conseguente aumento del numero di parametri da addestrare causano maggiore complessità, in termini di tempo e dimensione del modello. Dunque, è possibile adottare una nuova struttura per i building blocks, costituita unicamente da connessioni identità: **Bottleneck Design** mostrato in *Figura 1.9*. Questa parti-

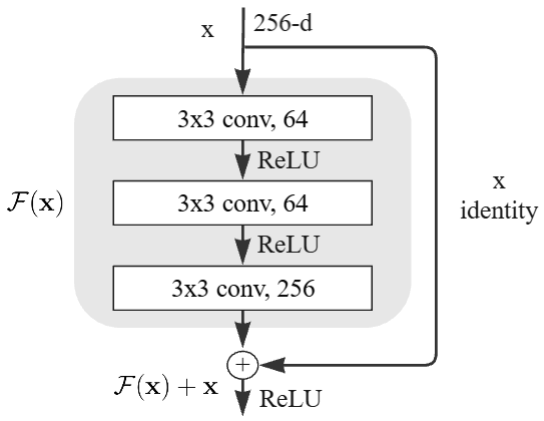


Figura 1.9: Building blocks con struttura a collo di bottiglia [16].

colare architettura è caratterizzata da una sequenza di tre strati, in cui il primo e l'ultimo sono utilizzati rispettivamente per diminuire ed aumentare la dimensionalità della pila di immagini, mentre il secondo permette l'esplorazione spaziale attraverso la convoluzione 3×3. La sequenza dei blocchi scelti per 50, 101 e 152 strati è schematizzata all'interno della *Tabella 1.4*.

Residual Networks di questo tipo permettono di ottenere risultati migliori in termini di accuratezza rispetto alle ResNet meno profonde. Questo risultato è dovuto alla loro maggiore profondità, che allo stesso tempo non causa la degradazione dell'accuratezza sul training set.

layer name	**output size**	**18-layer**	**34-layer**
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2	
		3×3 max pool, stride 2	
conv2_x	$56 \times 56 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
1×1		average pool, 1000-d fc, softmax	
FLOPs		1.8×10^9	3.6×10^9

Tabella 1.2: Architettura ResNet progettata per ImageNet, con 18 o 34 strati. Gli strati dei building blocks sono disposti in verticale all'interno delle parentesi quadre, riportando per ogni strato la dimensione dei filtri, accompagnata dal numero di filtri utilizzati; ad esempio $\left[3 \times 3, 64\right]$ *indicherà un unico strato costituito da 64 filtri di dimensione* 3×3*. Inoltre, queste parentesi sono seguite dal numero di ripetizioni consecutive dei blocchi di quel tipo: se si susseguono due building blocks uguali, si scriverà* $\left[3 \times 3, 64\right] \times 2$*. La dimensione dell'output presenta la forma* $d_x \times d_y \times c$*, dove* $d_x \times d_y$ *indica la dimensione delle immagini risultanti e c rappresenta il numero di canali contenenti un output, uno per filtro. Inoltre, il primo tra gli strati di ogni sezione "conv* i*_*x*", con* $i > 2$*, di building blocks ha stride pari a 2, mentre nelle restanti esso sarà 1. Infine, in ogni strato è presente lo Zero Padding per ottenere la grandezza dell'immagine in output desiderata. Per le connessioni, si osservi la Figura 1.7. Si ricordi che "FLOPs" (FLoating-point Operations) rappresenta il numero di operazioni in virgola mobile totali per l'elaborazione di un singolo input [16].*

Model	Plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Tabella 1.3: Top-1 error (%, 10-crop testing) sul validation set di ImageNet. In questo caso le ResNet non presentano parametri aggiuntivi rispetto alla loro versione "plain" [16].

layer name	**output size**	**50-layer**	**101-layer**	**152-layer**
conv1	112×112	7×7, 64, stride 2		
		3×3 max pool, stride 2		
conv2_x	56×56	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
1×1		average pool, 1000-d fc, softmax		
FLOPs		3.8×10^9	7.6×10^9	11.3×10^9

Tabella 1.4: Architettura ResNet profonde progettata per ImageNet: 50, 101 e 152 layer [16].

Capitolo 2

Il Metodo del Gradiente Stocastico (SGD) e la Versione Mini-Batch

I metodi di ottimizzazione svolgono un ruolo fondamentale all'interno del Machine Learning. Infatti, nell'ambito dell'apprendimento supervisionato, un esempio tipico di applicazione è la determinazione dei parametri del modello nella fase di addestramento attraverso la risoluzione di uno tra questi problemi:

- se la distribuzione di probabilità che genera i dati

  $$P : X \times Y \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to [0, 1]$$

  è nota, si ricorre alla minimizzazione stocastica del Rischio Atteso $R(\mathbf{w})$ (1.1);

- altrimenti, fornito un training set $\mathcal{S}$ generato secondo la distribuzione sconosciuta,

  $$\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) \in X \times Y \, : \, i = 1, 2, \ldots, n\}$$

  si richiede la minimizzazione deterministica del Rischio Empirico $R_{\text{emp}}(\mathbf{w})$ (1.4), eventualmente regolarizzato (1.5).

I tradizionali metodi basati sul gradiente, come il *Full Gradient Method* descritto nella sezione 2.1, sono efficienti per problemi di *dimensioni limitate* e per problemi di apprendimento *offline*, in quanto il calcolo del gradiente della funzione obiettivo

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w}) \quad \text{dove} \quad f_i(\mathbf{w}) := L\big(\mathbf{x}_i, \mathbf{y}_i, h(\mathbf{w}; \mathbf{x}_i)\big) \quad \forall i \in \{1, 2, \ldots, n\}$$

coinvolge tutti gli n dati.

Dunque, grazie alla crescita esponenziale del numero di dati a disposizione, il mondo accademico si è interessato allo sviluppo di nuovi metodi in grado di migliorare il costo computazionale in termini di tempo di esecuzione: nasce il *metodo del Gradiente Stocastico*, in cui si coinvolge un solo dato, estratto casualmente ad ogni iterazione. Esso è descritto nella sezione 2.2. Oltre ad un vantaggio per il *Machine Learning su larga scala*, esso permette di trattare problemi di apprendimento *online*, i cui dati non vengono forniti nel corso del tempo. Infatti, la natura del metodo SGD permette di proseguire l'addestramento anche aggiungendo ulteriori campioni [14].

Ove non specificato, le nozioni di questo capitolo appartengono alla pubblicazione [8].

2.1 Metodo di Discesa del Gradiente (GD)

Di seguito è riportato un breve richiamo ai *Metodi di Discesa del Gradiente*. Per maggiori dettagli, consultare il libro [24].

I **metodi di Discesa del Gradiente (Gradient Descent, GD)** sono **algoritmi iterativi** applicabili a problemi di minimizzazione non vincolata, e più in generale di ottimizzazione, con funzione obiettivo differenziabile con continuità:

$$\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w})$$

dove $f : \mathbb{R}^d \to \mathbb{R}$ è tale che $f \in \mathcal{C}^1(\mathbb{R}^d)$.

In particolare, a partire da un punto iniziale $\mathbf{w}^{(0)} \in \mathbb{R}^d$, esse generano una successione $\{\mathbf{w}^{(k)}\}_{k\in\mathbb{N}} \subset \mathbb{R}^d$ tali che permettono di determinare un punto di stazionarietà $\mathbf{w}^*$ della funzione obiettivo, che corrisponde al suo punto di minimo aggiungendo l'ipotesi di convessità. Si ricordi che un $\mathbf{w}^*$ è detto **punto di stazionarietà** per f se $\nabla f(\mathbf{w}^*) = 0$.

Lo schema generale di aggiornamento dell'iterazione di questi metodi è

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha_k \mathbf{d}^{(k)},$$

dove $\mathbf{d}^{(k)} \in \mathbb{R}^d$ e $\alpha_k \in \mathbb{R}_{>0}$ sono chiamati rispettivamente **direzione di ricerca** e **parametro di lunghezza di passo** (o **step length**). Il vettore $\mathbf{d}^{(k)} \in \mathbb{R}^d$ è scelto tra le *direzioni di discesa* della funzione f in $\mathbf{x}^{(k)}$.

Definizione 2.1.1 *Dato un punto* $\mathbf{w} \in \mathbb{R}^d$ *e una funzione* $f \in \mathcal{C}^1(\mathbb{R}^d)$*, un vettore* $\mathbf{d} \in \mathbb{R}^d$ *è detto* ***direzione di discesa*** *per* f *in* $\mathbf{w}$ *se soddisfa la disuguaglianza*

$$\nabla f(\mathbf{w})^T \mathbf{d} < 0$$

Essa garantisce localmente la discesa della funzione f valutata in $\mathbf{w}$ lungo la direzione $\mathbf{d}$. In particolare, è possibile dimostrare che

Proprietà 2.1.2 *Siano* $\mathbf{w} \in \mathbb{R}^d$*, una funzione* $f \in \mathcal{C}^1(\mathbb{R}^d)$ *e* $\mathbf{d} \in \mathbb{R}^d$ *direzione di discesa per* f *in* $\mathbf{w}$*, allora esiste uno scalare* $\tilde{\alpha} > 0$ *sufficientemente piccolo tale che*

$$f(\mathbf{w} + \alpha \mathbf{d}) < f(\mathbf{w}) \qquad \forall \alpha \in (0, \tilde{\alpha}]$$

Inoltre, lo scalare $\alpha_k \in \mathbb{R}_{>0}$ è determinato in modo tale che sia garantita la discesa della funzione per l'iterazione $k+1$, la quale caratterizza i cosiddetti **metodi di discesa**:

$$f(\mathbf{w}^{(k+1)}) = f(\mathbf{w}^{(k)} + \alpha_k \mathbf{d}^{(k)}) < f(\mathbf{w}^{(k)}) \qquad \forall k \in \mathbb{N}_0 \tag{2.1}$$

Osservazione 2.1.3 *Geometricamente, la direzione di discesa* $\mathbf{d} \in \mathbb{R}^d$ *forma un angolo ottuso con il gradiente della funzione* f *calcolato in* $\mathbf{x} \in \mathbb{R}^d$*, il quale rappresenta la direzione di massima crescita della funzione. Infatti,*

$$\nabla f(\mathbf{w})^T \mathbf{d} < 0 \quad \iff \quad ||\nabla f(\mathbf{w})||\, ||\mathbf{d}||\, cos(\eta) < 0 \quad \iff \quad cos(\eta) < 0$$

È possibile dimostrare la ben posizione dell'algoritmo, il cui arresto avviene in prossimità di uno dei punti di stazionarietà dell'applicazione f. Infatti, valgono le seguenti proprietà:

Proprietà 2.1.4 *Sia* $\mathbf{w} \in \mathbb{R}^d$*,*

- *se* $\mathbf{w}$ *non è un punto di stazionarietà per* f*, allora esiste almeno una direzione di discesa in* $\mathbf{w}$*;*
- *se non esistono direzioni di discesa per* f *in* $\mathbf{w}$*, allora* $\mathbf{w}$ *è un punto di stazionarietà.*

Vi sono differenti **criteri di arresto** per l'algoritmo, tra i quali si ricorda, fissata una tolleranza $\tau > 0$,

$$||\nabla f(\mathbf{w}^{(k)})|| \leq \tau.$$

Infine, un quesito fondamentale riguarda la *scelta di direzioni di discesa* e *di lunghezze di passo* adeguate. Un esempio comune di direzione di discesa è

$$\mathbf{d}^{(k)} = -\nabla f(\mathbf{w}^{(k)}),$$

che definisce il cosiddetto **metodo di Steepest Descent**, la cui formula di aggiornamento diventa

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f(\mathbf{w}^{(k)}). \tag{2.2}$$

Uno dei suoi vantaggi riguarda il costo computazionale: ad ogni iterazione è necessario unicamente calcolare il $\nabla f(\mathbf{w}^{(k)})$, evitando le derivate seconde, presenti in altre tipologie di direzioni di discesa. Tuttavia, la convergenza del metodo potrebbe essere lenta per alcuni problemi.

2.2 Metodo del Gradiente Stocastico (SGD)

2.2.1 Formulazione del Metodo SGD

Con la crescita esponenziale dei dati a disposizione nei maggiori ambiti di applicazione del machine learning, come la sanità, e lo sviluppo del Deep Learning, la necessità di gestire problemi su larga scala diventa cruciale. Per la risoluzione dei problemi di ottimizzazione necessari, come la minimizzazione del rischio empirico per l'addestramento delle reti neurali (*Sezione 1.2.1*), non è più efficiente a livello di tempo computazionale sfruttare il metodo tradizionale di discesa del gradiente. Questa necessità permette la nascita di un metodo semplice ed efficace che utilizza ad ogni iterazione solo un numero ridotto di dati: il *metodo di Discesa del Gradiente Stocastico (Stochastic Gradient Descent, SGD)*. Il primo metodo che segna l'inizio dell'era dell'*approssimazione stocastica* risale al 1951. Si tratta dell'articolo [27] pubblicato da H. Robbins e S. Monro, in cui si propone una catena di Markov per la risoluzione di equazioni contenenti il valore atteso di una funzione rispetto ad una distribuzione di probabilità sconosciuta. Per lo sviluppo di tale sezione è stato consultato il documento [8].

Il metodo del Gradiente Stocastico viene sviluppato per funzioni obiettivo non necessariamente convesse e può essere utilizzato sia per problemi di minimizzazione del rischio atteso, sia del rischio empirico, con opportune differenze nella scelta dei

campioni per ogni iterazione. Indicando con ξ una variabile aleatoria definita dalla distribuzione P, la cui realizzazione rappresenta un singolo campione $(\mathbf{x}, \mathbf{y}) \in X \times Y$, esse possono essere espresse in questo modo:

$$R(\mathbf{w}) = \mathbb{E}_{\xi \sim P}[f(\mathbf{w}; \xi)]$$

dove $f(\mathbf{w}; \cdot)$ è la composizione della funzione loss L e della funzione di predizione h, ossia $f(\mathbf{w}; (\mathbf{x}, \mathbf{y})) := L(\mathbf{x}, \mathbf{y}, h(\mathbf{w}; \mathbf{x}))$;

$$R_{\text{emp}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} f_i(\mathbf{w}) \quad \text{dove} \quad f_i(\mathbf{w}) := f(\mathbf{w}; \xi_{[i]}) = L(\mathbf{x}_i, \mathbf{y}_i, h(\mathbf{w}; \mathbf{x}_i)) \quad \forall i \in \{1, 2, \ldots, n\} \tag{2.3}$$

in cui $\{\xi_{[i]}\}_{i=1}^{n}$ rappresenta la sequenza di n realizzazioni di ξ, che generano il training set $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$. Successivamente, saranno indicate con ξ_k, dove $k \in \mathbb{N}_0$, variabili aleatorie indipendenti e identicamente distribuite, copie di ξ.

Indicando con $F : \mathbb{R}^d \to \mathbb{R}$ la funzione obiettivo del problema, tale che $F \in \mathcal{C}^1(\mathbb{R}^d)$, la formula di aggiornamento che definisce il **metodo del Gradiente Stocastico (Stochastic Gradient Descent, SGD)** è

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \, g(\mathbf{w}^{(k)}, \xi_k), \tag{2.4}$$

dove

- $g(\mathbf{w}^{(k)}, \xi_k) := \nabla f_{i_k}(\mathbf{w}^{(k)})$ è uno stimatore *non distorto* (2.3.2) di $\nabla F(\mathbf{w}^{(k)})$, gradiente di F calcolato in $\mathbf{w}^{(k)}$;
- $\alpha_k \in \mathbb{R}_{>0}$ è la **lunghezza di passo** (o **learning rate**) lungo la direzione scelta;
- i_k è l'indice del campione ottenuto tramite una realizzazione di ξ_k, generata attraverso una distribuzione uniforme per il rischio empirico e la distribuzione P per il rischio atteso.

Osservazione 2.2.1 *Si noti che, affinché il metodo sia valido, è sufficiente scegliere un qualsiasi stimatore non distorto di $\nabla F(\mathbf{w}^{(k)})$, anche differente da quello presente in (2.4).*

Siccome la scelta degli indici ad ogni iterazione segue una distribuzione di probabilità P, i parametri del modello $\{\mathbf{w}^{(k)}\}_{k \in \mathbb{N}_0} \subseteq \mathbb{R}^d$ appartengono ad un processo stocastico, non deterministico.

2.2.2 Confronto tra i Metodi GD e SGD

L'articolo [8] mette in luce tre motivazioni principali per cui si dovrebbe prediligere il metodo classico del Gradiente Stocastico rispetto alla versione Full Gradient nel caso di *Large-Scale Machine Learning*.

La prima **motivazione intuitiva** riguarda la gestione della **ridondanza** presente sui campioni. Si consideri il caso peggiore, in cui il training set $\mathcal{S}$ è ottenuto dall'unione di m copie distinte di un insieme più piccolo $\mathcal{S}_{\text{sub}}$. È banale dimostrare che minimizzare il rischio empirico rispetto a $\mathcal{S}$ corrisponda allo stesso problema costruito su $\mathcal{S}_{\text{sub}}$. Utilizzare il *metodo del Gradiente Stocastico* significa considerare in entrambi i casi un unico campione per ogni iterazione, in cui la probabilità di estrazione dei dati appartenenti a $\mathcal{S}_{\text{sub}}$ è la stessa. Viceversa, il *metodo Full Gradient Descent* considererà ad ogni iterazione tutti i possibili indici. Dunque, il costo computazionale per $\mathcal{S}$ sarà m volte superiore rispetto alla considerazione di una singola copia. In conclusione, il metodo del Gradiente Stocastico permette di considerare le stesse informazioni con costo computazionale inferiore, nel caso in cui sia presente una certa ridondanza nei dati, seppur non esattamente copie tra loro, maggiormente possibile nei problemi su larga scala.

La seconda motivazione considera i risultati dal punto di vista **pratico**. Dato training set $\mathcal{S}$ di cardinalità n, la successione di iterazioni dell'algoritmo in cui avvengono n accessi a $\mathcal{S}$ è chiamata **epoca**. Fissata un'epoca, mentre nel *metodo Full Gradient Descent* avviene un unico passo di aggiornamento dei parametri, il *metodo del Gradiente Stocastico* presenta n step successivi. Le prime epoche di quest'ultimo sono caratterizzate solitamente da un drastico miglioramento dell'accuratezza rispetto ai pochi passi compiuti dal metodo GD. Tuttavia, esso è seguito da un da un forte arresto, che sottolinea la necessità di scegliere una successione di lunghezze di passo $\{\alpha_k\}_{k\in\mathbb{N}_0}$ adeguata, tipicamente tale che sia decrescente. Queste considerazioni possono essere osservate parlando dell'*esempio di Bertsekas*:

Esempio 2.2.2 (esempio di Bertsekas (2015)) *Si consideri il caso in cui ogni f_i in (2.3) è un'applicazione quadratica convessa rispetto a $\mathbf{w} \in \mathbb{R}^d$, tale che il suo punto di minimo $\mathbf{w}_{i,*}$ sia uniformemente distribuito in $[-1, 1]$, per $i = 1, 2, \ldots, n$. Supponendo che $\mathbf{w}^{(0)} >> 1$, durante la prima iterazione avverrà sicuramente uno spostamento di $\mathbf{w}^{(1)}$ verso sinistra. Successivamente, con una certa probabilità, ad*

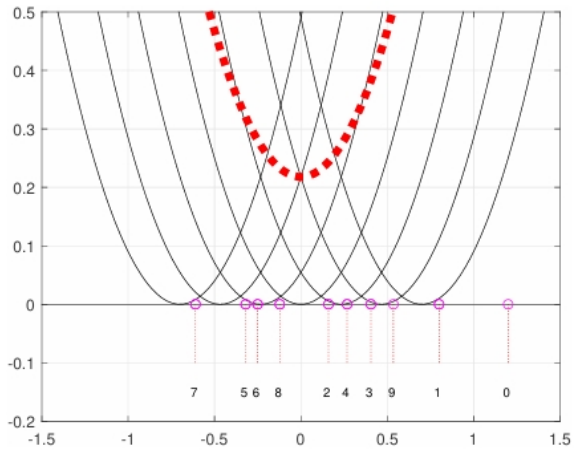


Figura 2.1: Illustrazione del comportamento del metodo SGD nell'esempio di Bertsekas [14].

ogni iterazione $\mathbf{w}^{(k)}$ *seguirà la direzione che permette l'avvicinamento al vero punto di minimo* $\mathbf{w}_*$ *del problema. Tuttavia, l'iniziale avvicinamento sarà seguito da un drastico rallentamento all'interno di una zona di confusione contenente* $\mathbf{w}_*$*, in quanto le informazioni fornite da un unico indice risultano essere insufficienti per ottenere maggior precisione. Questo comportamento è mostrato dalla Figura 2.1.*

Dunque, se si è obbligati a compiere solo un numero di epoche limitato, è possibile ottenere migliori prestazioni utilizzando il *metodo SGD*.

Questo vantaggio iniziale può essere dimostrato anche **teoricamente** attraverso l'analisi dei risultati di convergenza dei due metodi, fissando particolari ipotesi per la legittimità dei teoremi.

Definizione 2.2.3 *Una funzione* $F : \mathbb{R}^d \to \mathbb{R}$ *è* ***fortemente convessa*** *se soddisfa la seguente condizione*

$$\exists c > 0 \ : \ F(y) > F(x) + \nabla F(x)^T (y - x) + \frac{1}{2} c ||y - x||_2^2 \quad \forall (x, y) \in \mathbb{R}^d \times \mathbb{R}^d,$$

Supponendo che la funzione obiettivo $F : \mathbb{R}^d \to \mathbb{R}$ sia fortemente convessa, il tasso di convergenza del *metodo Full Gradient Descent* è **R-lineare**, ovvero

$$\exists \rho \in (0, 1) \ : \ F(\mathbf{w}^{(k)}) - F(\mathbf{w}_*) \leq \mathcal{O}(\rho^k) \quad \forall k = 0, 1, \ldots$$

dove $\mathbf{w}_*$ è il punto di minimo di F. Dunque, il numero di iterazioni k affinché sia garantita la ε-ottimalità, fissato $\varepsilon > 0$, cioè per cui

$$R_{\text{emp}}(\mathbf{w}^{(k)}) - R_{\text{emp}}(\mathbf{w}_*) \leq \varepsilon,$$

è proporzionale a $\log\left(\frac{1}{\varepsilon}\right)$. Ricordando che il costo per iterazione è pari ad n, il costo computazionale del metodo è proporzionale a $n\log\left(\frac{1}{\varepsilon}\right)$. Al contrario, aggiungendo opportune ipotesi sulla sequenza lunghezza di passo (2.5.5) $\{\alpha_k\}_{k\in\mathbb{N}_0}$, il *metodo Stochastic Gradient Descent* ha tasso di convergenza atteso **sub-lineare**, ossia

$$R_{\text{emp}}(\mathbf{w}^{(k)}) - R_{\text{emp}}(\mathbf{w}_*) \leq \mathcal{O}\left(\frac{1}{k}\right) \quad \forall k = 0, 1, \dots$$

Quindi, siccome il costo per iterazione è indipendente da n, il costo computazionale del metodo SGD è proporzionale a $\frac{1}{\varepsilon}$. Confrontando i due risultati, se il numero di esempi presenti nel dataset considerato è sufficientemente grande, come nel caso del *Large-Scale Machine Learning*, la ε-ottimalità sarà raggiunta attraverso un minor costo computazionale dal metodo del Gradiente Stocastico, in quanto in tal caso, nonostante il tasso di convergenza sia peggiore,

$$n\log\left(\frac{1}{\varepsilon}\right) >> \frac{1}{\varepsilon}.$$

2.3 Versione Mini-Batch e Proprietà Statistiche dello Stimatore del Gradiente

2.3.1 Proprietà Statistiche dello Stimatore del Gradiente

Per poter comprendere alcuni miglioramenti possibili da apportare al metodo, è necessario definire uno stimatore e le caratteristiche necessarie perché possa essere considerato valido.

Definizione 2.3.1 *[15] Si consideri un parametro θ sconosciuto. Dato un insieme di n campioni $\mathcal{S} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots \mathbf{x}^{(m)}\}$, con $m \in \mathbb{N}$, identicamente distribuiti (i.i.d.), una qualsiasi funzione*

$$\hat{\theta}(\mathcal{S}) = g(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots \mathbf{x}^{(m)})$$

*è detta **stimatore** per θ.*

Siccome i dati sono generati da un processo stocastico, $\hat{\theta}$ è una variabile aleatoria.

Esistono alcune principali proprietà che determinano la bontà di uno stimatore. Si tratta di *consistenza*, *bias* e *varianza*.

Definizione 2.3.2 *[23] Dato un parametro θ, un suo stimatore $\hat{\theta}(\mathcal{S})$ è* ***consistente*** *se*

$$\hat{\theta}(\mathcal{S}) \xrightarrow[|\mathcal{S}|\to\infty]{P} \theta$$

secondo la convergenza in probabilità.

Definizione 2.3.3 *[23] Si consideri un parametro θ e il suo stimatore $\hat{\theta}(\mathcal{S})$. Il* ***bias*** *di $\hat{\theta}(\mathcal{S})$ è definito come*

$$\text{bias}(\,\hat{\theta}(\cdot)\,) = \mathbb{E}_{P(\mathcal{S}|\theta)}\big[\hat{\theta}(\mathcal{S}) - \theta\big] = \mathbb{E}_{P(\mathcal{S}|\theta)}\big[\hat{\theta}(\mathcal{S})\big] - \theta$$

Si parla di stimatore ***non distorto*** *se* $\text{bias}(\,\hat{\theta}(\cdot)\,) = 0$*, rappresenta il caso in cui la distribuzione $\hat{P} = P(\cdot|\hat{\theta})$, determinata dalla stima di θ, è centrata nei veri valori del parametro stesso.*

In particolare, nel metodo SGD la distribuzione di probabilità sconosciuta P è approssimata mediante il campionamento uniforme di un esempio tra quelli appartenenti al training set $\mathcal{S}$.

Sebbene sia credibile che lo stimatore $g(\mathbf{w}^{(k)}, \xi_k)$ scelto debba essere non distorto, è anche importante che la *varianza* dello stimatore sia sufficientemente basso. Infatti, la stima di θ potrebbe comunque essere troppo differente rispetto ai valori reali che questo assume, rendendo $\hat{\theta}$ inefficace.

Definizione 2.3.4 *[23] La* ***varianza*** *dello stimatore $\hat{\theta}(\mathcal{S})$ del parametro $\theta \in \mathbb{R}$ è calcolabile attraverso la formula*

$$\text{Var}[\,\hat{\theta}\,] = \mathbb{E}\Big[\,(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2\,\Big]$$

in cui varianza e aspettazione sono espresse secondo distribuzione di probabilità $P(\mathcal{S}|\theta)$.

Similmente, se il parametro θ è multidimensionale, supponiamo $\theta \in \mathbb{R}^m$ vettore colonna, la varianza del suo stimatore $\hat{\theta}(\mathcal{S}) \in \mathbb{R}^m$ è la ***matrice di covarianza***

$$\text{Var}[\,\hat{\theta}\,] = \mathbb{E}\Big[\,(\hat{\theta} - \mathbb{E}[\hat{\theta}])(\hat{\theta} - \mathbb{E}[\hat{\theta}])^{\mathrm{T}}\,\Big] \in \mathbb{R}^{m\times m}$$

La sua riduzione può essere ottenuta diminuendo

$$\text{Var}[\,\hat{\theta}\,] = \mathbb{E}\Big[\,\|\hat{\theta} - \mathbb{E}[\hat{\theta}]\|^2\,\Big].$$

Dunque, è ragionevole introdurre alcune strategie che permettano di ridurre tale *rumore* dello stimatore.

2.3.2 Metodo del Gradiente Stocastico Mini-Batch

Come affermato inizialmente, la scelta di un unico indice i_k per ogni iterazione permette di ridurre notevolmente il tempo di esecuzione del processo rispetto al caso deterministico, *metodo di Steepest Descent*. Infatti, quest'ultimo, anche noto come *Full Gradient Descent Method*, considera tutti gli indici per il calcolo del gradiente di R_{emp}. Tuttavia, $-\nabla f_{i_k}$ potrebbe non essere direzione di discesa per la funzione obiettivo. Se $-\nabla f_{i_k}$ è uno stimatore non distorto di una direzione di discesa, la successione $\{\mathbf{w}^{(k)}\}_{k\in\mathbb{N}_0}$ tenderà a muoversi comunque verso il punto di minimo R_{emp}, ma con maggiore difficoltà a causa della varianza elevata. Per ottenere un miglioramento, è possibile trovare il giusto compromesso tra i due metodi, che permette di ridurre il rumore dello stimatore, pur rimanendo poco costoso.

Per questo motivo, è stata sviluppata la **versione Mini-Batch** del metodo del Gradiente Stocastico (SGD). Essa prevede che ad ogni iterazione $k \in \mathbb{N}$ sia estratto un sottoinsieme di indici $\mathcal{N}_k \subseteq \{1, 2, \ldots, n\}$, il **mini-batch**, secondo una distribuzione uniforme. Ricordando la formula del rischio empirico (2.3) e che ogni indice i è associato all'i-esimo campione del training set $\mathcal{S}$, la scelta del **gradiente stocastico** per la versione mini-batch ricade su

$$g(\mathbf{w}^{(k)}; \xi_k) := \frac{1}{|\mathcal{N}_k|} \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$$

dove ξ_k rappresenta l'insieme delle realizzazioni della variabile aleatoria ξ definita dalla distribuzione uniforme all'iterazione k. Esso rappresenta il gradiente di f nel problema (3.1), calcolato considerando solo sottoinsieme di campioni $\mathcal{N}_k \subseteq \mathcal{S}$.

In particolare, grazie a questa strategia, la varianza dello stimatore $g(\mathbf{w}^{(k)}; \xi_k)$ sarà ridotta di un fattore $\frac{1}{|\mathcal{N}_k|}$ rispetto al gradiente stocastico $\nabla f_i(\mathbf{w}^{(k)})$ ottenuto da un singolo indice, copia i.i.d. della stessa variabile aleatoria per ogni $i \in \{1, 2, \ldots, n\}$. Aumentando il numero di indici considerati, la varianza diminuirà.

La versione Mini-Batch apre la strada a nuove estensioni che permettono di migliorare il tasso di convergenza del metodo, come l'*aggiornamento dinamico della dimensione del batch* e la *scelta della distribuzione* per il suo campionamento. Siccome anch'esse presentano una varianza dello stimatore $g(\mathbf{w}^{(k)}, \xi_k)$ minore rispetto al metodo del Gradiente Stocastico (SGD) tradizionale, fanno parte dei cosiddetti **metodi di Riduzione della Varianza**. Essi saranno approfonditi nei *capitoli 3*

e *4*. Oltre, al miglioramento della velocità di convergenza del metodo, diminuire il rumore dello stimatore $g(\mathbf{w}^{(k)}, \xi_k)$ facilita la scelta di una lunghezza di passo α_k adeguata.

2.4 Pseudocodice dell'Algoritmo SGD e Mini-Batch

Lo pseudocodice del Metodo del Gradiente Stocastico (SGD) nella sua versione tradizionale o con mini-batch è riportato in *Algorithm 1*, dove

Algorithm 1 Stochastic Gradient Descent (SGD)

Require: Dataset $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$; $T \in \mathbb{N}$.

1: Scelta del parametro iniziale $\mathbf{w}^{(0)}$;
2: **for** $k = 0, 1, \ldots, T-1$ **do**
3: Generare una realizzazione della variabile aleatoria ξ_k (estrazione uniforme da $\mathcal{S}$);
4: Calcolo del vettore gradiente stocastico $g(\mathbf{w}^{(k)}, \xi_k)$;
5: Scelta del parametro lunghezza di passo $\alpha_k > 0$;
6: Aggiornamento del vettore di parametri $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \, g(\mathbf{w}^{(k)}, \xi_k)$.
7: **end for**
8: **return** $\mathbf{w}^{(T)}$

- ξ_k è la variabile aleatoria che genera uniformemente un indice o una sequenza $\mathcal{N}_k$, utilizzati nell'iterazione k;
- il gradiente stocastico scelto come stimatore del gradiente della funzione obiettivo $R_{\text{emp}}(\mathbf{w})$ all'iterazione k:

$$g(\mathbf{w}^{(k)}, \xi_k) := \begin{cases} \nabla f_{i_k}(\mathbf{w}^{(k)}) \\ \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) \end{cases}$$

- $\alpha_k \in \mathbb{R}_{>0}$ parametro lunghezza di passo, opportunamente scelto.

2.5 Selezione degli Iperparametri e Analisi della Convergenza

2.5.1 Selezione di Learning Rate e Mini-Batch Size

La scelta degli iperparametri, quali il *learning rate* α_k e il *micro-batch size* N_k, cardinalità del sottoinsieme dei campioni estratti per ogni iterazione, è cruciale per la convergenza e la performance ottenute dal metodo del Gradiente Stocastico. Tuttavia, questo problema non è banale e il costo computazionale necessario potrebbe essere molto elevato.

Esistono differenti valide strategie per la loro determinazione, le quali influenzano i risultati teorici di convergenza ottenibili dal metodo.

Tra quelle relative alla determinazione di α_k per ogni iterazione k, si ricorda la **regola dello step size fisso**,

$$\alpha_k := \alpha \quad \forall k \in \mathbb{N}_0, \text{ con } \alpha \in \mathbb{R}_{>0} \text{ fissato.}$$

Tuttavia, questo metodo presenta molte criticità. Infatti, la scelta di uno scalare α troppo elevato potrebbe causare una forte oscillazione esterna ad un intorno contenente il punto di stazionarietà senza mai convergere a causa del passo compiuto troppo grande. Viceversa, se α_k è eccessivamente piccolo, la convergenza effettiva sarà troppo lenta.

Dunque, per cercare di superare questa limitazione, è possibile ricorrere ad un aggiornamento dello **step size decrescente** all'aumentare del numero di iterazioni. Solitamente, si utilizza

$$\alpha_k = \mathcal{O}\left(\frac{1}{k}\right) = \frac{\alpha_0}{1+\alpha k} \qquad \forall k \in \mathbb{N}_0 \qquad \text{con } \alpha_0, \alpha > 0 \text{ fissati.}$$

Purtroppo, in questo caso la successione $\{\alpha_k\}_k$ è predeterminata dagli iperparametri α_0 e α, rendendola fortemente dipendente dalla loro scelta. Dunque, è banale immaginare che la loro ricerca possa essere lenta e costosa. Una scelta inadeguata potrebbe compromettere i risultati del modello.

Per diminuire il costo e aumentare la velocità nella ricerca degli iperparametri adeguati, è stato sviluppato un adattamento della strategia relativa ai metodi di

Discesa del Gradiente (GD): **Line Search Procedure con condizione di sufficiente decrescita di Armijo**, spiegata nella *Sezione 3.2*. Essa permette una ricerca adattiva della lunghezza di passo ad ogni iterazione.

Allo stesso modo, la dimensione dei batch N_k può essere prefissata o crescente all'aumentare dell'iterazione k, aggiungendo ad ogni iterazione informazioni. Ad esempio, la convergenza lineare del metodo SGD è garantita scegliendo una crescita di tipo geometrico. Inoltre, la legge che determina N_k può essere utilizzata come mezzo per ridurre la varianza dello stimatore della direzione di discesa corrente, diminuendone così il rumore (*Sezione 3.3*).

2.5.2 Risultati di Convergenza

Attraverso ipotesi restrittive, la scelta della legge che definisce questi iperparametri influenza in risultati di convergenza del metodo del Gradiente Stocastico. In particolare, avendo una funzione obiettivo $F : \mathbb{R}^d \to \mathbb{R}$ nel caso di minimizzazione del rischio atteso o empirico, si considerino le seguenti ipotesi:

Assunzione 2.5.1 (Lipschitz-Continuità del Gradiente) *F è differenziabile con continuità, ossia $F \in \mathcal{C}^1(\mathbb{R}^d)$, e $\nabla F : \mathbb{R}^d \to \mathbb{R}^d$ è Lipschitz continuo, cioè*

$$\exists L > 0 \quad : \quad \|\nabla F(\mathbf{w}) - \nabla F(\tilde{\mathbf{w}})\|_2 \leq L\|\mathbf{w} - \tilde{\mathbf{w}}\|_2 \quad \forall \mathbf{w}, \tilde{\mathbf{w}} \in \mathbb{R}^d.$$

Assunzione 2.5.2 (Forte Convessità) *F è fortemente convessa, ossia*

$$\exists c > 0 \quad : \quad F(\tilde{\mathbf{w}}) \geq F(\mathbf{w}) + \nabla F(\mathbf{w})^T(\tilde{\mathbf{w}} - \mathbf{w}) + \frac{1}{2}c\|\tilde{\mathbf{w}} - \mathbf{w}\|_2^2 \quad \forall \mathbf{w}, \tilde{\mathbf{w}} \in \mathbb{R}^d$$

Conseguenza diretta della forte convessità è l'esistenza e unicità di un unico punto di minimo $\mathbf{w}^*$. Si denota il valore di minimo $F^* := F(\mathbf{w}^*)$

Assunzione 2.5.3 (Limitazioni su Momento Primo e Secondo) *F soddisfa le seguenti disuguaglianze:*

1. *La successione $\{\mathbf{w}^{(k)}\}_k$ è contenuta in un insieme aperto $A \subset \mathbb{R}^d$ tale che*

$$\exists F_{inf} > 0 \quad : \quad F(w) \geq F_{inf} \quad \forall \mathbf{w} \in A;$$

2. *Esistono scalari* $\eta_G \geq \eta > 0$ *tali che, per ogni* $k \in \mathbb{N}$,

$$\nabla F(\mathbf{w}^{(k)})^T \mathbb{E}[g(\mathbf{w}^{(k)}, \xi^{(k)})] \geq \eta \|\nabla F(\mathbf{w}^{(k)})\|_2^2$$

e

$$\|\mathbb{E}[g(\mathbf{w}^{(k)}, \xi^{(k)})]\|_2 \leq \eta_G \|\nabla F(\mathbf{w}^{(k)})\|_2.$$

3. *Esistono scalari* $M \geq 0$ *e* $M_V \geq 0$ *tali che, per ogni* $k \in \mathbb{N}$,

$$Var[g(\mathbf{w}^{(k)}, \xi^{(k)})] \leq M + M_V \|\nabla F(\mathbf{w}^{(k)})\|_2^2.$$

Considerando la scelta della lunghezza di passo fisso, è possibile dimostrare un lower bound per il *gap di ottimalità*.

Teorema 2.5.4 (Forte Convessità, Lunghezza di Passo Fissa [8]) *Si considerino le ipotesi 2.5.1, 2.5.2 e 2.5.3, dove* $F^* = F_{inf}$. *Se il metodo del Gradiente Stocastico utilizza la regola della lunghezza di passo fisso,*

$$\alpha_k := \alpha \quad \forall k \in \mathbb{N}_0,$$

tale che

$$0 < \alpha \leq \frac{\eta}{LM_G}$$

dove $M_G := M_V + \eta_G^2 \geq \eta^2 > 0$.

Allora, l'optimality gap atteso soddisfa la seguente disuguaglianza

$$\mathbb{E}[F(\mathbf{w}^{(k)}) - F^*] \leq \frac{\alpha LM}{2c\eta} + (1 - \alpha c\eta)^k \left(F(\mathbf{w}^{(0)}) - F^* - \frac{\alpha LM}{2c\eta} \right) \xrightarrow[k\to\infty]{} \frac{\alpha LM}{2c\eta}$$

per ogni $k \in \mathbb{N}_0$.

Allo stesso modo, considerando una particolare legge di aggiornamento dello stepsize decrescente, tale che

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \qquad \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

Teorema 2.5.5 (Forte Convessità, Lunghezza di Passo Decrescente [8]) *Si considerino le ipotesi 2.5.1, 2.5.2 e 2.5.3, dove* $F^* = F_{inf}$. *Se il metodo del Gradiente Stocastico utilizza la regola della lunghezza di passo decrescente definita come*

$$\alpha_k = \frac{\beta_0}{\beta_1 + k},$$

per qualche $\beta_0 > \frac{1}{c\eta}$ *e* $\beta_1 > 0$ *tale che*

$$\alpha_0 \leq \frac{\eta}{LM_G}.$$

Allora, posto

$$\nu := \max\left\{\frac{\beta_0^2 LM}{2(\beta_0 c\eta - 1)}, (\beta_1 + 1)\big(F(\mathbf{w}^{(0)}) - F^*\big)\right\},$$

per ogni $k \in \mathbb{N}_0$*, il gap di ottimalità atteso soddisfa*

$$\mathbb{E}[F(\mathbf{w}^{(k)}) - F^*] \leq \frac{\eta}{\beta_1 + k}.$$

Inoltre, nel caso più generale della funzione obiettivo, essa potrebbe non essere fortemente convessa, non garantendo così che esista un unico punto di minimo globale, e neanche convessa, causando l'esistenza di punti di stazionarietà che potrebbero non corrispondere al minimo globale della funzione. È comunque possibile ottenere informazioni sull'avvicinamento della successione a uno dei punti di stazionarietà.

Teorema 2.5.6 (Non Convessità, Stepsize Fisso [8]) *Supponendo che siano verificate le ipotesi 2.5.1 e 2.5.3 e che la regola per l'aggiornamento della lunghezza di passo utilizzata dal metodo del Gradiente Stocastico sia*

$$\alpha_k = \alpha \quad \forall k \in \mathbb{N}_0,$$

tale che

$$0 < \alpha < \frac{\eta}{LM_G}.$$

Allora, per ogni $K \in \mathbb{N}$*, il valore atteso della media dei quadrati delle norme dei gradienti per le prime* K *iterazioni soddisfa la seguente disuguaglianza*

$$\mathbb{E}\Big[\frac{1}{K}\sum_{k=0}^{K-1}\|\nabla F(\mathbf{w}^{(k)})\|_2^2\Big] = \frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\Big[\|\nabla F(\mathbf{w}^{(k)})\|_2^2\Big] \leq \frac{\alpha LM}{\eta} + \frac{2\big(F(\mathbf{w}^{(0)}) - F_{\text{inf}}\big)}{K\eta\alpha} \xrightarrow[k\to\infty]{} \frac{\alpha LM}{\eta}$$

Teorema 2.5.7 (Non Convessità, Stepsize Decrescente [8]) *Supponendo che siano verificate le ipotesi 2.5.1 e 2.5.3 e che il metodo del Gradiente Stocastico sia definito con una regola per l'aggiornamento della lunghezza di passo* $\{\alpha_k\}_{k\in\mathbb{N}_0}$ *che soddisfi*

$$\sum_{k=0}^{\infty}\alpha_k = \infty, \quad \sum_{k=0}^{\infty}\alpha_k^2 < \infty.$$

Allora, la successione $\{\mathbf{w}^{(k)}\}_{k\in\mathbb{N}_0}$ *generata dal metodo è tale che*

$$\liminf_{k\to\infty}\mathbb{E}\big[\|\nabla F(\mathbf{w}^{(k)})\|_2^2\big] = 0.$$

Le dimostrazioni dei teoremi precedenti sono presenti in [8].

Infine, anche la scelta dinamica della dimensione del batch size può influenzare il tasso di convergenza. Un esempio è riportato dal seguente teorema, che utilizza la crescita geometrica di N_k per garantire la convergenza (in aspettazione) R-lineare. In questo caso, sarà considerato il problema di minimizzazione del rischio empirico.

Teorema 2.5.8 ([9]) *Considerando la versione mini-batch del metodo del Gradiente Stocastico, si indichi con N_k la dimensione del mini-batch $\mathcal{N}_k$ scelto all'iterazione k. Siano verificate le ipotesi*

- $F \in \mathcal{C}^2(\mathbb{R}^d)$;
- *F è uniformemente convessa, ovvero esistono due costanti λ e L tali che $0 < \lambda < L$, per cui vale*

$$\lambda\|\mathbf{v}\|^2 \leq \mathbf{v}^T\nabla^2 F(\mathbf{w})\mathbf{v} \leq L\|\mathbf{v}\|^2, \qquad \forall\, \mathbf{w} \in \mathbb{R}^d,\ \forall\, \mathbf{v} \in \mathbb{R}^d;$$

  essa garantisce l'unicità del punto di minimo $\mathbf{w}^*$ di F.

Si scelga lunghezza di passo fissa

$$\alpha_k := \frac{1}{L}.$$

Inoltre, si scelga come regola di aggiornamento dinamico della dimensione del batch

$$N_k := \lceil a^k \rceil, \qquad \text{con } a > 1,$$

che definisce una crescita geometrica.
Supponiamo che esista una costante $\varepsilon > 0$ tale che

$$\left\|\mathrm{Var}\left[\nabla f_i(\mathbf{w}^{(k)})\right]\right\|_1 \ \leq\ \varepsilon \qquad \forall k \in \mathbb{N}_0.$$

Allora, la successione $\{\mathbf{w}^{(k)}\}_{k\in\mathbb{N}_0}$ verifica la seguente disuguaglianza

$$\mathbb{E}\left[F(\mathbf{w}^{(k)}) - F(\mathbf{w}^*)\right] \ \leq\ C\,\rho^k$$

dove

$$\rho = \max\left\{1 - \frac{\lambda}{4L},\, \frac{1}{a}\right\} < 1 \quad e \quad C = \max\left\{F(\mathbf{w}^{(0)}) - F(\mathbf{w}^*),\, \frac{2\varepsilon}{\lambda}\right\}.$$

per ogni $k \in \mathbb{N}_0$.

Capitolo 3

DeepLISA e Riduzione della Varianza Attraverso la Scelta del Mini-Batch Size

Come accennato nella *Sezione 2.5*, il metodo del Gradiente Stocastico tradizionale presenta molte limitazioni sia per quanto riguarda la velocità di convergenza, migliorabile attraverso una regola di aggiornamento della dimensione dei mini-batch, sia per la forte dipendenza dalla scelta degli iperparametri, il cui costo per l'ottimizzazione non può essere ignorato.

L'articolo [13] descrive un nuovo metodo in grado di superare questi vincoli: l'**algoritmo Deep-LISA**. Esso apporta due modifiche fondamentali per ottenere migliori risultati:

1. la regola di aggiornamento del learning rate α_k segue la procedura di *line search* monotona o non monotona, che riduce significativamente la dipendenza dei risultati dalla scelta degli iperparametri grazie ad una strategia di selezionamento adattiva (*Sezione 3.2*);

2. la scelta della *dimensione del mini-batch* ad ogni iterazione segue una legge monotona debolmente crescente, caratterizzata da una crescita lenta che permette una riduzione sempre maggiore della varianza dello stimatore della direzione di discesa, il gradiente stocastico (*Sezione 3.3*).

Queste modifiche porteranno miglioramenti sia rispetto alla convergenza, che nel caso di funzioni fortemente convesse diventa di tipo *R-lineare*, sia attraverso la diminuzione del costo del tuning degli iperparametri e l'aumento della loro robustezza.

3.1 Definizione del Problema e Assunzioni Strutturali

In molti ambiti di Machine Learning e Deep Learning sorge la necessità di risolvere problemi di minimizzazione del rischio empirico, eventualmente regolarizzato, della forma

$$\min_{\mathbf{w}\in\mathbf{R}^d} f(\mathbf{w}) \equiv \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}) \quad \left(\text{risp.} \quad \min_{\mathbf{w}\in\mathbf{R}^d} f(\mathbf{w}) \equiv \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2\right), \tag{3.1}$$

dove

- $\mathcal{S}$ rappresenta il training set, avente cardinalità pari ad n;
- $f_i : \mathbb{R}^d \to \mathbb{R}$ è una funzione differenziabile con continuità in $\mathbb{R}^d$, ovvero

  $$f_i \in \mathcal{C}^1(\mathbb{R}^d) \quad \forall i \in \{1, 2, \ldots, n\},$$

  relativa al campione i-esimo di $\mathcal{S}$. Si osservi che, grazie a questa ipotesi, il suo gradiente rispetto a $\mathbf{w}$ è ben definito e permette di utilizzarlo per il *metodo del Gradiente Stocastico*.

Per la sua risoluzione, è possibile utilizzare il *metodo del Gradiente Stocastico, versione mini-batch* (*Sezione 2.3.2*), definito dalla regola di aggiornamento iterativo

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k+1)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(\mathbf{k})}) \tag{3.2}$$

dove, fissata l'iterazione $k \in \mathbb{N}_0$,

- $\mathcal{N}_k \subseteq \mathcal{S}$ è un sottoinsieme di campioni estratti uniformemente dal training set $\mathcal{S}$. Sarà indicata con N_k la cardinalità di questo insieme;
- $f_{\mathcal{N}_k} : \mathbb{R}^d \to \mathbb{R}$ rappresenta il rischio empirico costruito su $\mathcal{N}_k$, eventualmente regolarizzato,

  $$f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k}\sum_{i=1}^{N_k} f_i(\mathbf{w}) \qquad \left(\text{risp.} \quad f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k}\sum_{i=1}^{N_k} f_i(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2\right),$$

il cui gradiente è calcolato come

$$\nabla f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k}\sum_{i=1}^{N_k}\nabla f_i(\mathbf{w}) \qquad \left(\text{risp.} \quad \nabla f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k}\sum_{i=1}^{N_k}\nabla f_i(\mathbf{w}) + \lambda\,\mathbf{w}\right);$$

- il learning rate α_k è uno scalare positivo scelto attraverso la *line search procedure* monotona o non monotona (*Sezione 3.2*), dipendente da un intero positivo M.

Alle assunzioni annunciate in precedenza, si aggiungono quelle classiche che permettono di dimostrare i successivi risultati di ben posizione e di convergenza.

Notazione 3.1.1 *Siano uno spazio di probabilità $(\Theta, \mathcal{F}, \mathcal{P})$, ed una variabile aleatoria X, indicando con $\mathcal{F}_k$ la σ-algebra generata dai primi k valori ottenuti dal processo stocastico $\mathbf{x}^{(0)}$, $\mathbf{x}^{(1)}$,..., $\mathbf{x}^{(k)}$, allora:*

- $\mathbb{E}[X]$ *rappresenta l'aspettazione totale di X;*

- $\mathbb{E}_k[X] := \mathbb{E}[X|\mathcal{F}_k]$ *è l'aspettazione condizionata di X rispetto a $\mathcal{F}_k$.*

Assunzione 3.1.2 *La funzione $f : \mathbb{R}^d \to \mathbb{R}$ ammette un punto di minimo ed è inferiormente limitata da uno scalare f^*, ovvero*

$$f(\mathbf{w}) \geq f^* \qquad \forall \mathbf{w} \in \mathbb{R}^d$$

$$X^* = argmin_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}) \neq \emptyset$$

Assunzione 3.1.3 *f_i, $i \in \{1, 2, \ldots, n\}$ ha gradiente L_i-Lipschitz continuo, dunque*

$$\exists \bar{L} > 0 \quad : \quad \|\nabla f_i(\mathbf{w}) - \nabla f_i(\tilde{\mathbf{w}})\|_2 \leq \bar{L}\|\mathbf{w} - \tilde{\mathbf{w}}\|_2 \quad \forall \mathbf{w}, \tilde{\mathbf{w}} \in \mathbb{R}^d.$$

L_i è il più piccolo valore di $\bar{L}$ che soddisfa tale disuguaglianza.

Conseguenza diretta di questa ipotesi è che anche ∇f è L-Lipschitz continuo, con

$$L \leq \frac{1}{n}\sum_{i=1}^{n} L_i.$$

Inoltre, posto $L_{\max} := \max_{0\leq i\leq n} L_i$,

$$L \leq L_{\max}.$$

Allo stesso modo, è possibile dimostrare che $\nabla f_{\mathcal{N}_k}$ è $L_{\mathcal{N}_k}$-Lipschitz continuo, con

$$L_{\mathcal{N}_k} \leq \frac{1}{N_k}\sum_{i=1}^{N_k} L_i \leq L_{\max}$$

Assunzione 3.1.4 *Il gradiente stocastico $\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ è uno stimatore non distorto (Definizione 2.3.2) di $\nabla f(\mathbf{w}^{(k)})$, cioè*

$$\mathbb{E}_k\Big[\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\Big] = \nabla f(\mathbf{w}^{(k)}).$$

Inoltre,

$$\mathbb{E}_k\Big[f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\Big] = f(\mathbf{w}^{(k)}).$$

Assunzione 3.1.5 *Data una sequenza propria e sommabile $\{\varepsilon_k\}_k$, $\sum_{k=0}^{\infty}\varepsilon_k < +\infty$, i gradienti stocastici non distorti $\{\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\}_k$ devono soddisfare la seguente disuguaglianza*

$$\mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|_2^2\Big] \leq \varepsilon_k \qquad \forall k \in \mathbb{N}_0, \tag{3.3}$$

costruita appositamente per il contesto di Deep Learning.
Questa condizione forza la ***riduzione progressiva della varianza****.*

Osservazione 3.1.6 *Ricordando che lo stimatore è non distorto,*

$$\mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|_2^2\Big] = \mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|_2^2\Big] - \|\nabla f(\mathbf{w}^{(k)})\|_2^2. \tag{3.4}$$

Dunque, la disuguaglianza 3.3 è equivalente a

$$\mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|_2^2\Big] \leq \|\nabla f(\mathbf{w}^{(k)})\|_2^2 + \varepsilon_k. \tag{3.5}$$

Questa ipotesi è meno restrittiva rispetto alla richiesta di Strong Growth Condition (SGC)

$$\exists \rho \geq 1 \quad : \quad \mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w})\|_2^2\Big] \leq \rho\|\nabla f(\mathbf{w})\|_2^2,$$

che deve valere per ogni $\mathbf{w} \in \mathbb{R}^d$, presente in alcuni teoremi per dimostrare i risultati di convergenza per funzioni obiettivo anche non convesse.

Infine, è necessario introdurre un lemma utilizzato per la dimostrazione dei teoremi legati ai risultati di convergenza del metodo.

Lemma 3.1.7 ([13, Robbins-Siegmund Lemma, Lemma 2.1, Sec. 2]) *Si considerino variabili aleatorie non negative ν_k, u_k, η_k, μ_k tali che*

$$\mathbb{E}\Big[\nu_{k+1} \mid \mathcal{S}_k\Big] \leq (1+\eta_k)\nu_k - u_k + \mu_k \quad a.s.,$$

$$\sum_{k=0}^{\infty} \eta_k < \infty \quad a.s., \qquad \sum_{k=0}^{\infty} \mu_k < \infty \quad a.s.,$$

dove $\mathbb{E}\Big[\nu_{k+1} \mid \mathcal{S}_k\Big]$ *indica l'aspettazione condizionata dati* $\nu_0, \ldots, \nu_k$, $u_0, \ldots, u_k$, $\eta_0, \ldots, \eta_k$, $\mu_0, \ldots, \mu_k$.

Allora, esiste una variabile aleatoria $\nu \geq 0$ *tale che*

$$\nu_k \to \nu \quad a.s.$$

Inoltre,

$$\sum_{k=0}^{\infty} u_k < \infty \quad a.s.$$

3.2 Line Search Procedure per il Calcolo del Learning Rate e Risultati di Convergenza

3.2.1 Line Search Procedure per il Calcolo del Learning

La **Procedura di Line Search con condizione di Backtracking di Armijo** per la determinazione ad ogni passo di α_k deriva dall'adattamento al caso stocastico della versione omonima per il Metodo di Discesa del Gradiente (GD). Esistono due tipoligie di versioni, *monotona* o *non monotona*, a seconda del valore che assume l'iperparametro $M \in \mathbb{N}$.

Lo pseudocodice della procedura è mostrato in *Algorithm 2*. In particolare, ad

Algorithm 2 Stochastic Line Search Procedure per α_k [13]

Require: $\alpha_{\max} > 0$, $M > 0$, $\beta \in (0,1)$, $\gamma \in (0,1)$

1: Si pone $\alpha := \alpha_{\max}$;
2: Si calcola il gradiente stocastico $\mathbf{g}_k \leftarrow \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;
3: **while** $f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha \mathbf{g}_k) > \max_{0 \leq j \leq \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) - \gamma \alpha \|\mathbf{g}_k\|^2$ **do**
4: $\quad \alpha \leftarrow \beta \alpha$;
5: **end while**
6: $\alpha_k \leftarrow \alpha$;
7: **return** α_k.

ogni iterazione compiuta dal ciclo *while*, $\alpha_{\max}$ viene moltiplicato per $\beta \in (0,1)$,

ottenendo dopo le prime m iterazioni

$$\alpha := \alpha_{\max}\beta^m.$$

Aumentando m, $\alpha_{\max}$ viene ridotto. La scelta del learning rate α_k ricadrà su $\bar{\alpha}$ calcolato secondo il più piccolo numero di iterazioni $\bar{m}$ tale che sia verificata la condizione di sufficiente discesa

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \bar{\alpha}\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) \leq \max_{0\leq j\leq \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) - \gamma\bar{\alpha}\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2$$

Siccome $-\gamma\bar{\alpha}\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2$ è strettamente negativo, per definizione di massimo, sarà garantito che

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \bar{\alpha}\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) < f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) \qquad \forall j \in [0, \min\{k, M\}] \cap \mathbb{N}_0$$

Se $k \leq M$, caso **monotono**, si richiede che il nuovo iterato $\mathbf{w}^{(k+1)}$ sia scelto in modo che $f_{\mathcal{N}_k}$ valutato in quel punto sia strettamente minore rispetto a tutti i valori $f_{\mathcal{N}_j}(\mathbf{w}^{(j)})$, $j = 1, 2, \ldots, k$ ottenuti in precedenza. Viceversa, se $k > M$, caso **non monotono**, la decrescita è richiesta solo rispetto agli M valori precedenti ad esso.

3.2.2 Ben Posizione di SGD con Line Search Procedure

È possibile dimostrare la *ben posizione* della procedura di Line Search per la selezione di α_k, servendosi del *Descent Lemma*. In questa sezione presentiamo i risultati principali del modello.

Lemma 3.2.1 ([13, Descent Lemma]) *Sia $F : \mathbb{R}^n \to \mathbb{R}$ una funzione continuamente differenziabile avente gradiente L-Lipschitz continuo, ossia*

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

Allora, per ogni $x, y \in \mathbb{R}^n$, vale

$$F(y) \leq F(x) + \nabla F(x)^\top (y - x) + \frac{L}{2}\|y - x\|^2. \tag{3.6}$$

Osservazione 3.2.2 *Si osservi che, grazie alle ipotesi $f_i \in \mathcal{C}^1(\mathbb{R}^d)$ e 3.1.3, questa disuguaglianza è verificata per f, $f_{\mathcal{N}_k}$ e f_i, per ogni $i \in \{1, 2, \ldots, n\}$ e per ogni $\mathcal{N}_k \subseteq \mathcal{S}$.*

In particolare, la ben posizione del metodo del Gradiente Stocastico dotato di Line Search Procedure è ben posto, come verificato nel *Lemma 3.2.3*.

Lemma 3.2.3 (Ben Posizione della Line Search Procedure) *[13, Lemma 2.2, Sec. 2] Verificata l'ipotesi 3.1.3, la procedura di Line Search presentata nell'Algorithm 2 è ben posta. Inoltre, posto*

$$\tilde{\alpha} = \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)}{L_{\max}}\right\},$$

vale la seguente disuguaglianza:

$$0 < \tilde{\alpha} \leq \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)}{L_{\mathcal{N}_k}}\right\} \leq \alpha_k \leq \alpha_{\max} \qquad \forall k \in \mathbb{N}_0 \tag{3.7}$$

Osservazione 3.2.4 *Dall'equazione (3.7) è banale dedurre che, se* $\gamma \leq \frac{1}{2}$ *e* $\alpha_{max} > \frac{2\beta(1-\gamma)}{L_{\mathcal{N}_k}}$*, allora*

$$\alpha_k \geq \frac{\beta}{L_{\mathcal{N}_k}}.$$

Per favorire la fluidità della trattazione, la dimostrazione del *Lemma 3.2.3* è riportata in *Appendice 7.6.3*.

3.2.3 Risultati di Convergenza

Questa sezione è volta all'analisi della convergenza del metodo SGD *(3.2)* dotato di Line Search Procedure (*Algorithm 2*), dipendente dalle assunzioni fatte sulla funzione obiettivo. In particolare, si analizzeranno tre casistiche: caso generale, non convesso, di f_i; convessità di f_i; proprietà di Polyak-Lojasiewicz su f. La dimostrazioni dei *Teoremi 3.2.5 - 3.2.6 - 3.2.7 - 3.2.11* sono riportate in *Appendice 7.6.3*.

Caso Non Convesso. Si consideri il caso di *funzioni obiettivo non convesse*. Il *Teorema 3.2.5* permette di dimostrare che la successione generata da

$$\left\{\min_{k=0,\dots,T-1} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2]\right\}_T$$

converge **sublinearmente** a zero. Inoltre, la successione $\{\mathbf{w}^{(k)}\}_k$ converge ad un punto di stazionarietà, in quanto $\lim_{k\to+\infty} \|\nabla f(\mathbf{w}^{(k)})\| = 0$ quasi certamente.

Teorema 3.2.5 (Convergenza per Funzioni Obiettivo Non Convesse) *[13, Teorema 2.1, Sec. 2] Si supponga che siano verificate le assunzioni 3.1.2, 3.1.3, 3.1.4 e che la varianza del gradiente sia progressivamente ridotta grazie a una successione positiva e sommabile $\{\varepsilon_k\}_k$ (Assunzione 3.1.5). La sequenza $\{\mathbf{w}^{(k)}\}_k$ ottenuta attraverso il metodo SGD (3.2) dotato di Line Search Procedure (Algorithm 2), con $\alpha_{max} < \frac{2}{L}$, ammette tasso di convergenza sublineare a un punto di stazionarietà, in quanto*

$$\min_{k=0,\dots,T-1} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2] \leq \frac{1}{\delta T}\big(f(\mathbf{w}^{(0)}) - f^*\big) + \frac{\xi}{\delta T}\sum_{k=0}^{T-1}\varepsilon_k, \tag{3.8}$$

dove $\delta = 2\tilde{\alpha} - L\alpha_{\max}^2 > 0$. Inoltre,

1. $\sum_{k=0}^{\infty} \|\nabla f(\mathbf{w}^{(k)})\|^2 < +\infty$ *a.s.;*
2. $\lim_{k\to+\infty} \|\nabla f(\mathbf{w}^{(k)})\| = 0$ *a.s.*

Caso Convesso. Aggiungendo l'ipotesi di *convessità dei termini della funzione obiettivo*, è garantita la convergenza della successione $\{\mathbf{w}^{(k)}\}_k$ a un punto di minimo (*Teorema 3.2.6*). Aggiungendo ulteriori ipotesi, il *Teorema 3.2.7* dimostra che il **tasso di convergenza** di $\{\mathbf{w}^{(k)}\}_k$ o della sua media

$$\left\{\frac{1}{T+1}\sum_{k=0}^{T}\mathbf{w}^{(k)}\right\}_T$$

è **sublineare**.

Teorema 3.2.6 (Convergenza per Funzioni Obiettivo Convesse) *[13, Teorema 2.2, Sec. 2] Date le assunzioni 3.1.2, 3.1.3, 3.1.4, se inoltre*

- *l'Assunzione 3.1.5 coinvolge una successione positiva e sommabile $\{\varepsilon_k\}_k$ tale che*
$$\sum_{k=0}^{\infty}\sqrt{\varepsilon_k} < +\infty;$$
- *f_i è convessa per ogni $i \in \{1, 2, \dots, n\}$;*

allora, la sequenza $\{\mathbf{w}^{(k)}\}_k$ generata dal metodo SGD (3.2) dotato di Line Search Procedure (Algorithm 2), con $\alpha_{max} < \frac{2}{L}$, converge a.s. alla soluzione del problema (3.1).

In particolare, è possibile dimostrare il tasso di convergenza nel caso convesso.

Teorema 3.2.7 (Tasso di Convergenza per Funzioni Obiettivo Convesse) *[13, Teorema 2.3, Sec. 2] Supponendo che siano verificate le assunzioni 3.1.2, 3.1.3, 3.1.4, e che*

- *l'Assunzione 3.1.5 coinvolge una successione positiva e sommabile* $\{\varepsilon_k\}_k$ *tale che*
$$\sum_{k=0}^{\infty} \sqrt{\varepsilon_k} < +\infty;$$
- f_i *è convessa per ogni* $i \in \{1, 2, \dots, n\}$*;*

allora, la sequenza $\{\mathbf{w}^{(k)}\}$ *generata dal metodo SGD (3.2) con Line Search Procedure (Algorithm 2), tale che* $\alpha_{max} < \frac{2}{L}$*,*

$$\mathbb{E}[f(\bar{\mathbf{w}}^{(T)}) - f(\mathbf{w}^*)] = \mathcal{O}\left(\frac{1}{T}\right),$$

dove

$$\bar{\mathbf{w}}^{(T)} = \frac{1}{T+1} \sum_{k=0}^{T} \mathbf{w}^{(k)}.$$

Se inoltre si ha

$$\sum_{k=0}^{\infty} k\, \varepsilon_k < \infty,$$

allora

$$\mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] = \mathcal{O}\left(\frac{1}{k}\right).$$

Proprietà di Polyak-Lojasiewicz. Infine, aggiungendo come ipotesi su f la *proprietà di Polyak-Lojasiewicz*, garantita nel caso di *funzioni fortemente convesse*, il metodo è caratterizzato da un **tasso di convergenza R-lineare** (*Teorema 3.2.11*).

Definizione 3.2.8 *Sia una funzione* $f \in \mathcal{C}^1(\mathbb{R}^d)$*, supponendo che esista un punto di minimo, essa verifica la* ***proprietà di Polyak-Lojasiewicz*** *se*

$$\|\nabla f(\mathbf{x})\|^2 \geq 2c\big(f(\mathbf{x}) - f^*\big) \quad \forall \mathbf{x} \in \mathbb{R}^d, \tag{3.9}$$

in cui $c > 0$ *e* $f^* = \inf_{\mathbf{x} \in \mathbb{R}^d} f(x)$*.*

Innanzitutto, si ricordino due lemmi che permettono di dimostrare il teorema relativo alla convergenza.

Lemma 3.2.9 ([13, Lemma 2.3, Sec. 2]) *Sia $\{\varepsilon_k\}_k$ una successione positiva che converge R-linearmente a zero. Allora, per ogni $\phi \in (0,1)$ e $q \in \mathbb{N}$,*

$$r_k = \sum_{j=1}^{k+1} \phi^{j-1} \varepsilon_{q+k-j}$$

converge R-linearmente a zero.

Lemma 3.2.10 ([13, Lemma 2.4, Sec. 2]) *Sia una funzione $h : \mathbb{R}^d \to \mathbb{R}$ tale che*

- *h soddisfa la proprietà di Polyak-Lojasiewicz, con relativa costante $c > 0$;*
- *∇h è L-Lipschitz continuo.*

Allora,

$$c \leq L.$$

Dunque, vale il seguente risultato di convergenza.

Teorema 3.2.11 (Convergenza R-Lineare per Funzioni con Proprietà PL) *[13, Teorema 2.4, Sec. 2] Sotto le assunzioni 3.1.2, 3.1.3, 3.1.4, si supponga che*

- *f_i sia convessa, per ogni $i \in \{1, \ldots, n\}$;*
- *la funzione obiettivo f soddisfi la condizione di Polyak–Łojasiewicz;*
- *sia verificata l'Assunzione 3.1.5 attraverso una successione $\{\varepsilon_k\}$ positiva che converge a zero R-linearmente.*

Allora, la successione $\{\mathbf{w}^{(k)}\}_k$ costruita attraverso il metodo SGD (3.2) dotato di procedura Line Search (Algorithm 2), con $\alpha_{\max} < \frac{1}{L}$, è caratterizzata da un tasso di convergenza R-lineare. In particolare,

$$\mathbb{E}\big[f(\mathbf{w}^{(k+1)}) - f^*\big] \leq (1 - \delta c)^k \big(f(\mathbf{w}^{(0)}) - f^*\big) + \tau \rho^k, \qquad \forall k \in \mathbb{N}_0 \tag{3.10}$$

dove $\delta = 2\tilde{\alpha} - L\alpha_{\max}^2$, $\delta c < 1$, $\tau > 0$ e $\rho \in [0,1)$.

3.3 Strategia di Mini-Batch Size per la Riduzione della Varianza

L'idea alla base della legge che guida la scelta della dimensione N_k del mini-batch ad ogni iterazione k deriva dall'*algoritmo LISA*, proposto nell'articolo [12].

Affinché la convergenza del metodo SGD dotato di strategia di Line Search sia garantita, è necessario considerare il problema di minimo (3.1) in cui f è una funzione obiettivo generica, tale che

(A) $0 \leq \alpha_k \leq \alpha_{\max} < \frac{2}{L}$, dove L è la costante di Lipschitz del gradiente della funzione obiettivo f (*Assunzione 3.1.3*);

(B) vale la condizione presentata nell'*Assunzione 3.1.5*:

$$\mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|_2^2\Big] \leq \varepsilon_k \qquad \forall k \in \mathbb{N}_0, \tag{3.11}$$

dove $\{\varepsilon_k\}_k$ è una successione sommabile, positiva e non crescente. La monotonia imposta a questa successione permette la riduzione progressiva della varianza dello stimatore.

La *Condizione (B)* può essere ottenuta attraverso una scelta opportuna dei $N_k \in \{N_0, 2, \ldots, n\}$ ad ogni iterazione, dove N_0 è un iperparametro che rappresenta il limite inferiore della dimensione dei mini-batch. Infatti, la varianza del gradiente stocastico costruito attraverso il batch $\mathcal{N}_k$ è

$$\mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] \overset{(1)}{=} \frac{1}{N_k^2}\sum_{i\in\mathcal{N}_k}\mathbb{E}_k[\|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] \tag{3.12}$$

$$\overset{(2)}{=} \frac{N_k}{N_k^2}\mathbb{E}_k\Big[\|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] \tag{3.13}$$

$$= \frac{1}{N_k}\mathbb{E}_k[\|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] \tag{3.14}$$

per un qualsiasi $i \in \mathcal{N}_k$. L'equazione è conseguenza diretta del fatto che gli stimatori $\nabla f_i(\mathbf{w}^{(k)})$ sono copie i.i.d. di una stessa variabile aleatoria (utilizzo in equazione (2)) e della proprietà relativa alla varianza di una combinazione lineare di variabili aleatorie (utilizzo in equazione (1)). Inoltre, supponendo che la varianza del singolo stimatore $\nabla f_i(\mathbf{w}^{(k)})$ sia limitata da una costante $C > 0$ per ogni $i \in \mathcal{N}_k$,

$$\mathbb{E}_k\Big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] = \frac{1}{N_k}\mathbb{E}_k\Big[\|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] \leq \frac{C}{N_k}$$

Dunque, affinché sia verificata la disuguaglianza *(3.11)*, è sufficiente scegliere N_k tale che

$$\frac{C}{N_k} \leq \varepsilon_k,$$

cioè

$$N_k \geq \frac{C}{\varepsilon_k} \qquad \forall k \in \mathbb{N}_0. \tag{3.15}$$

3.3.1 Selezione del Mini-Batch Size dell'algoritmo LISA

Sulla base delle considerazioni precedenti, l'*algoritmo LISA* [12] prevede che ad ogni iterazione k sia scelto esattamente

$$N_k := \left\lceil \frac{C}{\varepsilon_k} \right\rceil, \tag{3.16}$$

più piccolo intero per cui la disequazione *3.15)* è verificata. In particolare, per ogni $k \in \mathbb{N}$, è possibile approssimare C attraverso la varianza campionaria di $\nabla f_i(\mathbf{w}^{(k)})$

$$\mathbb{E}_k\Big[\|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2\Big] \approx \frac{1}{N_k - 1} \sum_{i \in \mathcal{N}_k} \Big(\|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\Big),$$

la quale, divisa per N_k, determina la varianza campionaria dell'intero stimatore $\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$

$$V^{(k)} \equiv \frac{1}{N_k(N_k - 1)} \sum_{i \in \mathcal{N}_k} \|\nabla f_i(\mathbf{w}^{(k)}) - \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2. \tag{3.17}$$

Dunque, sarà sufficiente scegliere una dimensione N_k e verificare se $V^{(k)} \leq \varepsilon_k$; in caso contrario, si segue la formula per la determinazione della dimensione successiva da testare

$$N_k^{\text{next}} = \min\left\{ n,\ \max\left(\left\lceil \frac{N_k^{\text{prev}} V_{\text{prev}}^{(k)}}{\varepsilon_k} \right\rceil, N_k + 1 \right) \right\},$$

in cui gli apici 'prev' e 'next' rappresentano rispettivamente i valori della prova precedente e di quella successiva. In tal modo, N_k^{next} è maggiore o uguale a $\lceil \frac{C}{\varepsilon_k} \rceil$, dove C è relativo al mini-batch size provato precedentemente (N_k^{prev}). Se la varianza campionaria della nuova dimensione scelta verifica $V_{\text{next}}^{(k)} \leq \varepsilon_k$, la ricerca per l'iterazione corrente k termina, altrimenti si ripete il procedimento.

L'**algoritmo LISA** è un metodo che segue lo schema SGD *(3.2)* con Line Search Procedure (*Algorithm 2*), in cui la successione crescente $\{N_k\}_k$ viene determinata dall'*Algoritmo 3*.

Algorithm 3 LISA Dynamic Mini-batch Selection, iteration k [12]

Require: $\mathbf{w}^{(k)} \in \mathbb{R}^d$; ε_k elemento della sequenza sommabile non negativa $\{\varepsilon_k\}_k$; N_{k-1} grandezza del mini-batch dell'iterazione precedente.

1: Si pone $N_k = N_{k-1}$;

2: Campionamento del mini-batch $\mathcal{N}_k$ di dimensione iniziale N_k;

3: Calcolo del gradiente stocastico

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{N_k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$$

4: Calcolo della varianza campionaria, dipendente da N_k,

$$V^{(k)} = \frac{1}{N_k(N_k - 1)} \sum_{i \in \mathcal{N}_k} \left\| \nabla f_i(\mathbf{w}^{(k)}) - \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) \right\|^2$$

5: **if** $V^{(k)} \leq \varepsilon_k$ **or** $N_k \geq N$ **then**

6: Scelta di α_k con *Algorithm 2*;

7: **else**

8: $N_k = \min\left\{n,\ \max\left(\left\lceil \frac{N_k V^{(k)}}{\varepsilon_k} \right\rceil, N_k + 1\right)\right\}$;

9: Ripetere a partire dalla *riga 2*.

10: **end if**

Tuttavia, questo metodo presenta **criticità** sia in termini di **costo computazionale** che in termini di **memoria**. In particolare, ad ogni iterazione è necessario calcolare l'espressione (3.17), che richiede la memorizzazione della matrice dei $\{\nabla f_i(\mathbf{w}^{(k)})\}_{i\in\mathcal{N}_k}$ di dimensione $d \times N_k$, dove d è il numero di parametri del modello ($\mathbf{w} \in \mathbb{R}^d$). Inoltre, questa operazione potrebbe essere ripetuta più volte in un'unica iterazione, per riuscire a soddisfare il criterio di arresto $V^{(k)} \leq \varepsilon_k$.

3.3.2 Selezione del Mini-Batch Size dell'algoritmo DeepLISA

L'articolo [13] propone una soluzione intelligente per eliminare la necessità di grandi risorse di calcolo e di memoria da parte dei calcolatori, rispondendo comunque alla necessità di scegliere la dimensione del mini-batch che verifichi 3.15, al fine di garantire la veridicità della *condizione (B)*. Supponendo di conoscere C, indipendente dall'iterazione k, e considerando la successione debolmente decrescente $\{\varepsilon_k\}_k$, porre

$$m_k := \left\lceil \frac{C}{\varepsilon_k} \right\rceil \qquad \forall k \in \mathbb{N}_0$$

significa costruire una successione $\{m_k\}_{k\in\mathbb{N}_0}$ monotona debolmente crescente, che rappresenta i più piccoli interi per cui la condizione (3.15) è verificata per ogni k.

Si indichi con il termine **epoca** la sequenza di iterazioni che accede al training set $\mathcal{S}$ un numero di volte pari alla sua cardinalità ($|\mathcal{S}| = n$). È possibile costruire i mini-batch per ogni epoca t attraverso una *partizione del training set* rispetto a una dimensione fissata N_t. In tal modo, ogni epoca sarà costituita da $\lceil \frac{n}{N_t} \rceil$ mini-batch con N_t campioni, ad eccezione dell'ultimo che conterrà una quantità pari al resto della divisione stessa. Si pongano i batch corrispondenti secondo la notazione

$$\left\{\mathcal{N}_{\bar{t}+0}, \mathcal{N}_{\bar{t}+1}, \ldots, \mathcal{N}_{\bar{t}+\lceil \frac{n}{N_t} \rceil - 1}\right\},$$

dove $\bar{t}$ è il numero di iterazioni considerate fino all'inizio della nuova epoca t, che corrisponde all'indice del suo primo mini-batch.

Fissato $N_0 \in \mathbb{N}$, la dimensione N_t dei mini-batch, per $t \geq 1$, sarà scelta in modo tale che la condizione (3.15) sia soddisfatta per ogni sua iterazione. Siccome la successione $\{m_k\}_{k\in\mathbb{N}_0}$ è monotona debolmente crescente, sarà sufficiente scegliere la componente il cui indice è successivo a quelli relativi alle iterazioni dell'epoca t. In

particolare, si considererà

$$N_t := m_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil} \qquad \forall t \in \mathbb{N},$$

ossia

$$N_t := \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \qquad \forall t \in \mathbb{N}.$$

Infatti, siccome $0 < N_{t-1} \leq N_t$ per ogni epoca t, il numero di iterazioni compiute scegliendo N_t come dimensione sarà al massimo pari a quello ottenuto attraverso N_{t-1}, in quanto

$$\left\lceil \frac{n}{N_t} \right\rceil \leq \left\lceil \frac{n}{N_{t-1}} \right\rceil.$$

Dalla crescenza della successione $\{m_k\}_k$, la dimensione del mini-batch $\mathcal{N}_k$ dell'epoca t verificherà la condizione richiesta (3.15): siccome

- $|\mathcal{N}_k| := N_t$;
- $N_t = m_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil} \geq m_{\bar{t}+\left\lceil \frac{n}{N_t} \right\rceil} \geq m_k$, cioè $N_t \geq \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \geq \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_t} \right\rceil}} \right\rceil \geq \left\lceil \frac{C}{\varepsilon_k} \right\rceil$;

allora

$$|\mathcal{N}_k| \geq \frac{C}{\varepsilon_k}$$

per ogni $k \in \{\bar{t},\, \bar{t}+1, \ldots,\, \bar{t}+\lceil \frac{n}{N_t} \rceil - 1\}$.

Inoltre, è necessario porre delle limitazioni strutturali alla scelta della dimensione dei batch. Indicando con N_0 e $n_{\max}$ rispettivamente il limite inferiore e superiore di N_t, la sua **regola di aggiornamento dinamico del mini-batch size** sarà

$$N_t := \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil ;\, N_0 \right\} \right\} \qquad \forall t \in \mathbb{N}. \tag{3.18}$$

Mentre N_0 ammette qualsiasi valore, n_{max} è obbligato dal vincolo di memoria a disposizione del calcolatore.

3.4 Algoritmo DeepLISA e Pseudocodice

L'algoritmo sviluppato nella pubblicazione [13] segue lo schema del Gradiente Stocastico *(3.2)* con procedura di Line Search (*Algorithm 2*) e utilizza la regola di

aggiornamento della dimensione dei batch (3.18). Esso è chiamato **DeepLISA**, in quanto progettato per diminuire la quantità di risorse necessarie per i problemi su larga scala sfruttando un linguaggio comune nell'ambito del Deep Learning.

Algorithm 4 DeepLISA Algorithm [13]

Require: $T > 0$, $n_{\max} > 0$, $\mathbf{w}^{(0)} \in \mathbb{R}^d$, $0 < N_0 < n$, $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$, $\beta, \beta_2 \in (0,1)$, $\gamma > 0$, $M > 0$, una sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$, $C > 0$, $\bar{t} = 0$.

1: **for** $t = 1, 2, \ldots, T$ **do**

Step 1: Scelta della Dimensione e Creazione dei Mini-Batch Size

2: Scelta di N_t come

$$N_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\};$$

3: Partizione del training set $\mathcal{S}$ in $\left\lceil \frac{n}{N_t} \right\rceil$ mini-batch di dimensione N_t;

4: **for** $k = \bar{t}, \ldots, \bar{t} + \left\lceil \frac{n}{N_t} \right\rceil - 1$ **do**

5: Si sceglie il mini-batch $\mathcal{N}_k$ di cardinalità N_t.

6: Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e il gradiente stocastico $\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;

Step 2: Line Search Procedure per la Scelta del Learning Rate

7: Si calcola $\bar{\mathbf{w}} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;

8: **while** $f_{\mathcal{N}_k}(\bar{\mathbf{w}}) > \max_{0\le j\le \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) - \gamma\alpha_k \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2$ **do**

9: $\alpha_k = \beta\alpha_k$;

10: $\bar{\mathbf{w}} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;

11: **end while**

Step 3: Aggiornamento per l'Iterazione Successiva

12: Si pongono

$$\mathbf{w}^{(k+1)} = \bar{\mathbf{w}};$$
$$\alpha_{k+1} = \min\left\{ \alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min} \right) \right\}.$$

13: **end for**

14: Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.

15: **end for**

L'*algoritmo LISA (Algorithm 3*) costruisce uno o più batch ad ogni iterazione, fino a quando la condizione di arresto per la selezione della dimensione N_k non è soddisfatta. Questo determina un vero e proprio *collo di bottiglia del traffico in memoria* e, conseguentemente, del tempo computazionale, in quanto il numero di accessi all'insieme di addestramento in memoria è molto grande. Al contrario l'implementazione di *DeepLISA (Algorithm 4)* si serve della classe *PyTorch* che costruisce una partizione dell'insieme di dati, scegliendo in modo causale il loro ordine e accedendo al training set un'unica volta all'inizio di ogni epoca: il **Data Loader**. Questo permette di diminuire il traffico in memoria e il tempo computazionale.

Inoltre, a differenza di *LISA*, la strategia (3.18) di *DeepLISA* permette di avere **totale controllo sulla dimensione dei mini-batch** scelta per ogni epoca. Difatti, $\{N_t\}_t$ è predeterminata dalla scelta dell'iperparametro N_0 e della successione non crescente $\{\varepsilon_k\}_k$ che stabilisce la pendenza della sua crescita, diminuendo il tempo necessario per il fine-tuning degli iperparametri attraverso la procedura *trial-and-error*:

- N_0 non deve essere nè troppo grande, nè troppo piccolo per $\{\varepsilon_k\}_k$: nel primo caso, N_t rimarrà uguale a N_0 per molte iterazioni in quanto limite inferiore per la dimensione, mentre nel secondo la sequenza $\{N_t\}_t$ cresce troppo rapidamente, superando il vincolo di capacità $n_{\max}$. Inoltre, maggiore è la dimensione del batch, minore sarà la varianza del gradiente stocastico ad essa associato; un valore N_0 sufficientemente piccolo permette di sfruttare inizialmente l'alta varianza dello stimatore.

- Al contrario, n_{max} è un vincolo di capacità imposto dal calcolatore GPGPU (General Purpose Graphic Process Unit), che deve avere sufficiente memoria per contenere un singolo batch.

- Anche la scelta della successione $\{\varepsilon_k\}_k$ non crescente e tale che $\sum_{k=1}^{+\infty} \varepsilon_k < +\infty$ non richiede molto tempo computazionale pre-iterazione; esso sarà scelto in modo tale che N_t sia compreso tra N_0 e $n_{\max}$, avvicinandosi a quest'ultimo senza mai raggiungerlo.

Line Search Procedure in DeepLISA (Step 2)

Infine, per garantire che sia verificata la *condizione (A)*, ossia $0 \leq \alpha_k \leq \alpha_{\max} < \frac{2}{L}$, è sufficiente ricordare i risultati del *Lemma 3.2.3.* Esso asserisce la ben posizione del metodo SGD (*Step 3*) con procedura di Line Search per la selezione della lunghezza di passo (*Step 2*) e la disuguaglianza 3.7:

$$0 < \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)}{L_{\max}}\right\} \leq \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)}{L_{\mathcal{N}_k}}\right\} \leq \alpha_k \leq \alpha_{\max} \qquad \forall k \in \mathbb{N}_0.$$

Il limite superiore $\frac{2}{L}$ può essere approssimato grazie ad un'opportuna scelta dell'iperparametro $\alpha_{\max}$.

Inoltre, la successione $\{\alpha_k\}_k$ non è forzata ad essere strettamente decrescente, evitando il problema di una riduzione drastica dello stepsize già dalle prime epoche. In particolare, α_{k+1} è inizializzato secondo la legge

$$\alpha_{k+1}^{\text{initialization}} = \min\left\{\alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min}\right)\right\}$$

dove $\beta_2 \in (0,1)$ è un iperparametro opportunamente fissato. Di conseguenza, il primo tentativo di lunghezza di passo sarà un valore compreso tra $\alpha_{\min}$ e $\alpha_{\max}$ superiore a quello considerato al passo precedente. Minore sarà il valore di β_2, maggiore sarà $\alpha_{k+1}^{\text{initialization}}$.

3.4.1 Analisi di Robustezza ed Efficienza di DeepLISA

All'interno della pubblicazione [13], è presente l'analisi dell'efficacia e della robustezza dell'algoritmo DeepLISA rispetto alla scelta degli iperparametri coinvolti ed alle prestazioni su un problema di larga scala in un framework di Deep Learning.

Innanzitutto, l'articolo mostra come l'introduzione della *Line Search Procedure* per la scelta della lunghezza di passo α_k determini la **robustezza** del modello: piccole variazioni sulla scelta degli iperparametri, generano performance simili tra loro. Un vantaggio di questo fenomeno è la riduzione della necessità di un tempo elevato per il fine-tuning. Al contrario, nel *metodo del Gradiente Stocastico* con lunghezza di passo e dimensione del mini-batch fissati, [13] mostra che le prestazioni dipendono fortemente dal valore assunto dagli iperparametri.

Inoltre, le **prestazioni** migliori dell'algoritmo DeepLISA sono **comparabili** con quelle ottenute dal metodo del Gradiente Stocastico citato in precedenza e da quello

in cui il learning rate è determinato secondo la procedura di Line Search regola di *Armijo*

$$\alpha_k := \min\{\alpha_{\max}, \alpha_{k-1}\delta^{\frac{N_0}{n}}\} \quad \text{con} \quad \delta > 1.$$

Capitolo 4

Importance Sampling e Riduzione della Varianza del Gradiente Stocastico

Considerando il problema di minimizzazione (3.1), il metodo iterativo SGD (*Algorithm 1*) prevede che il gradiente stocastico sia costruito attraverso la formula

$$\bar{g}(\mathbf{w}^{(k)}) := \begin{cases} \nabla f_{i_k}(\mathbf{w}^{(k)}) & \text{SGD classico} \\ \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) & \text{SGD con mini-batch} \end{cases} \tag{4.1}$$

dove i campioni considerati ad ogni iterazione sono scelti secondo una distribuzione uniforme. Tuttavia, la distribuzione utilizzata per l'estrazione potrebbe essere modificata per ottenere la diminuzione della varianza dello stimatore $g(\mathbf{w}^{(k)})$: strategia *Importance Sampling*.

In questo capitolo saranno esposti i risultati della pubblicazione [10], che costruisce alcune varianti degli algoritmi presentati in [21], che sfruttano la distribuzione *Adaptive Importance Sampling* per il campionamento degli indici. In particolare, in [10] si propone la procedura di *Line Search* per α_k, sostituendo la legge strettamente decrescente

$$\alpha_k = \frac{\alpha_0}{1 + \bar{\alpha} k} \qquad \forall k \in \mathbb{N}. \tag{4.2}$$

Questo permette di diminuire la forte dipendenza dagli iperparametri, come riportato nei risultati del capitolo precedente (*Capitolo 3*).

Per comodità, in questo capitolo si ricorrerà alla seguente notazione:

Notazione 4.0.1 *Siano una variabile aleatoria* $\mathbf{X} \in \mathbb{R}^d$*, dipendente dall'estrazione degli indici* i *con distribuzione di probabilità* p^k *(*$i \sim p^k$*), ed una* σ*-algebra* $\mathcal{F}_k$*, generata dai risultati delle iterazioni del processo* $\mathbf{w}^{(0)}$*,* $\mathbf{w}^{(1)}$*,...,* $\mathbf{w}^{(k)}$*. Allora,*

- $\mathbb{E}_{i\sim p^k}[\mathbf{X}|\mathcal{F}_k]$ *sarà l'aspettazione condizionata di* $\mathbf{X}$ *rispetto alla* σ*-algebra* $\mathcal{F}_k$*;*
- $\mathbb{E}_{i\sim p^k}[\mathbf{X}]$ *sarà l'aspettazione totale di* X*.*

4.1 Adaptive Importance Sampling per la Riduzione della Varianza

Nell'ambito di campionamento statistico, la strategia **Importance Sampling (IS)** [23] è un *metodo Monte Carlo* comunemente utilizzato per approssimare un parametro I ottenibile attraverso l'aspettazione di una funzione $\bar{g}$ rispetto ad una distribuzione di probabilità q:

$$I = \mathbb{E}_q[\bar{g}(\mathbf{X})] = \int \bar{g}(\mathbf{x})q(\mathbf{x})\mathrm{d}\mathbf{x},$$

Il metodo IS costruisce uno stimatore g di I, definito attraverso una nuova densità di probabilità p, tale che

$$I = \mathbb{E}_p[g(\mathbf{X})] = \mathbb{E}_q[\bar{g}(\mathbf{X})], \qquad \mathrm{Var}_p[g(\mathbf{X})] < \mathrm{Var}_q[\bar{g}(\mathbf{X})].$$

p è chiamata **importance distribution**. Questa strategia nasce dall'idea di considerare regioni dello spazio dove $\mathbf{x}$ è maggiormente significativo per il calcolo di I, aumentandone la probabilità che sia scelto per la creazione del nuovo stimatore, la cui varianza viene ridotta.

In particolare, dallo sviluppo

$$\mathbb{E}_q[\bar{g}(\mathbf{X})] = \int \bar{g}(\mathbf{x})q(\mathbf{x})\,\mathrm{d}\mathbf{x} = \int \bar{g}(\mathbf{x})\frac{q(\mathbf{x})}{p(\mathbf{x})}p(\mathbf{x})\,\mathrm{d}\mathbf{x} = \mathbb{E}_p\left[\bar{g}(\mathbf{X})\frac{q(\mathbf{X})}{p(\mathbf{X})}\right], \tag{4.3}$$

ne deriva la scelta dello stimatore non distorto

$$g(\mathbf{x}) := \bar{g}(\mathbf{x})\frac{q(\mathbf{x})}{p(\mathbf{x})}$$

e di una distribuzione di probabilità p opportuna, in modo che la varianza di $\bar{g}$ sia effettivamente ridotta e che

$$p(\mathbf{x}) > 0 \quad \text{se} \quad \bar{g}(\mathbf{x})q(\mathbf{x}) \neq 0,$$

affinchè g sia ben definito.

Nel contesto discreto in analisi, per ogni iterazione k, il parametro da stimare è $\nabla f(\mathbf{w}^{(k)})$. Inoltre, considerando il metodo SGD, lo stimatore $\bar{g}_k$ sarà definito come (4.1), i cui indici sono campionati secondo la distribuzione uniforme $q = (q_1, q_2, \ldots, q_n)$ tale che

$$q_i = \frac{1}{n} \qquad \text{per ogni indice } i \in \{1, 2, \ldots, n\}.$$

Il nuovo stimatore sarà costruito attraverso una probabilità di estrarre i campioni $p^k = (p_1^k, p_2^k, \ldots, p_n^k)$ tale che

$$p_i^k > 0 \quad \forall i \in \{1, 2, \ldots, n\}, \qquad \sum_{i=1}^{n} p_i = 1$$

dove la componente i-esima p_i^k rappresenta la probabilità che sia estratto il campione i. La distribuzione p^k funge da **importance distribution discreta** sugli indici i del training set $\mathcal{S}$, analogamente a quanto accade nel caso continuo, permettendo di campionare con maggiore frequenza i dati più "significativi" per il gradiente. Si osservi che la distorsione (*bias*) dello stimatore viene evitata grazie al peso $\frac{q_i}{p_i^k}$, ossia $\frac{1}{np_i^k}$.

In particolare, il nuovo stimatore *non distorto* sarà

$$g(\mathbf{w}^{(k)}) := \begin{cases} \frac{1}{np_{i_k}^k} \nabla f_{i_k}(\mathbf{w}^{(k)}) & \text{SGD classico} \\ \frac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) & \text{SGD con mini-batch} \end{cases} \tag{4.4}$$

i cui indici sono campionati attraverso la distribuzione importance sampling stessa. In particolare, il primo sarà ottenuto basandosi su quanto affermato nel caso continuo, mentre il secondo sarà adattato attraverso la somma pesata di $\frac{1}{np_{i_k}^k} \nabla f_{i_k}(\mathbf{w}^{(k)})$ i cui pesi sono

$$\delta_i = \frac{p_i^k}{\sum_{i \in \mathcal{N}_k} p_i^k}.$$

$g(\mathbf{w}^{(k)})$ definito in (4.4) sarà il **gradiente stocastico** per la versione *SGD* che utilizza la distribuzione *Importance Sampling*. È facile dimostrare che si tratta di

stimatori non distorti di $\nabla f(\mathbf{w}^{(k)})$: per il metodo SGD basterà adattare lo sviluppo (4.3) al caso discreto; la versione mini-batch è conseguenza diretta della media pesata di stimatori non distorti di $\nabla f(\mathbf{w}^{(k)})$.

Si consideri il metodo del gradiente stocastico classico. Come dimostrato in [33], la **distribuzione Importance Sampling** ottimale, che minimizza la varianza dello stimatore $g(\mathbf{w}^{(k)}) = \frac{1}{np_{i_k}^k}\nabla f_{i_k}(\mathbf{w}^{(k)})$, è definita da

$$(p_i^k)^* = \frac{\|\nabla f_i(\mathbf{w}^{(k)})\|}{\sum_{j=1}^n \|\nabla f_j(\mathbf{w}^{(k)})\|}, \tag{4.5}$$

soluzione dimostrabile attraverso le condizioni di ottimalità di Karush–Kuhn–Tucker del problema di minimizzazione

$$\min_{\substack{p_i^k\in[0,1]\\ \sum_{i=1}^n p_i^k=1}} \mathbb{E}_k\left[\left\|\frac{1}{np_i^k}\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\right\|^2\right] \iff \min_{\substack{p_i^k\in[0,1]\\ \sum_{i=1}^n p_i^k=1}} \frac{1}{n^2}\sum_{i=1}^n \frac{1}{np_i^k}\|\nabla f_i(\mathbf{w}^{(k)})\|^2,$$

la cui aspettazione dipende dalla distribuzione p^k con cui gli indici i vengono estratti. Le due forme sopra indicate sono equivalenti, in quanto vale l'uguaglianza

$$\begin{aligned}\mathbb{E}_k\left[\left\|\frac{1}{np_i^k}\nabla f_i(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\right\|^2\right] &= \mathbb{E}_k\left[\left\|\frac{1}{np_i^k}\nabla f_i(\mathbf{w}^{(k)})\right\|^2\right] - \|\nabla f(\mathbf{w}^{(k)})\|^2 \\ &= \frac{1}{n^2}\sum_{i=1}^n \frac{1}{np_i^k}\|\nabla f_i(\mathbf{w}^{(k)})\|^2 - \|\nabla f(\mathbf{w}^{(k)})\|^2\end{aligned}$$

dove il secondo termine è indipendente dalla scelta di p^k.

Tuttavia, questa formula è **computazionalmente inefficiente** per problemi di larga scala, in quanto, per ogni iterazione k, è necessario il calcolo dei n gradienti differenti, $\nabla f_i(\mathbf{w}^{(k)})$ per $i \in \{1, 2, \ldots, n\}$. Perciò, è ragionevole considerare un nuovo approccio, che aggiorni il valore della norma $\|\nabla f_{i_k}(\mathbf{w}^{(k)})\|$ solo se l'indice i_k è estratto nell'iterazione k. In tal caso, si parla di distribuzione *Adaptive Importance Sampling (AIS)*. Quest'ultima è definita nel seguente modo:

Definizione 4.1.1 *Fissata un'iterazione $k \in \{0, 1, \ldots, maxit\}$, la* ***distribuzione Adaptive Importance Sampling (AIS)*** $p^k = (p_1^k, p_2^k, \ldots, p_n^k)$*, dove l'$i$-esima componente rappresenta la probabilità di estrazione del campione i, è definita attraverso la combinazione convessa*

$$p_i^k = \xi_k \frac{\pi_i^k}{\sum_{j=1}^n \pi_j^k} + (1 - \xi_k)\frac{1}{n} \tag{4.6}$$

in cui

- $\xi_k \in [\xi_{min}, \xi_{max}] \subseteq (0,1)$ *è una sequenza crescente, da* ξ_{min} *a* ξ_{max}*, costruita come*

$$\begin{aligned}\xi_k &= \left(1 - \frac{k}{maxit}\right)\xi_{\min} + \frac{k}{maxit}\xi_{\max} \\ &= \xi_{\min} + \frac{k}{maxit}\left(\xi_{\max} - \xi_{\min}\right)\end{aligned}$$

dove $maxit + 1$ *rappresenta il numero massimo di iterazioni;*

- $\pi^k = (\pi_1^k, \pi_2^k, \ldots, \pi_n^k)$ *è il vettore contenente le norme dei gradienti calcolati. Inizializzata come* $\pi^0 = (1, 1, \ldots, 1)$*, è aggiornata ad ogni iterazione*

$$\pi_i^k = \begin{cases} \|\nabla f_i(\mathbf{w}^{(k)})\| & \text{se } i = i_k \quad (\text{risp. } i \in \mathcal{N}_k), \\ \pi_i^{k-1} & \text{altrimenti.} \end{cases} \qquad \forall i \in \{1, 2, \ldots, n\} \quad (4.7)$$

nel caso in cui sia estratto un unico indice o un mini-batch $\mathcal{N}_k$.

La distribuzione di probabilità p^k è costruita attraverso la media pesata tra la distribuzione uniforme e quella ottenuta normalizzando π^k. Siccome la successione $\{\xi_k\}_{k=0,1,\ldots,\text{maxit}}$ è crescente, p^k sarà realizzata a partire dalla distribuzione uniforme, esplorazione causale degli indici, dando gradualmente più importanza all'approssimazione della distribuzione ottimale (4.5) ottenuta in modo adattivo tramite π^k.

In pratica, la distribuzione AIS assegna una probabilità di campionamento superiore agli elementi i con norma del gradiente ∇f_i elevata. Di conseguenza, in quanto esempi con costo avente massima pendenza rispetto ai parametri $\mathbf{w}$, si intravede una potenziale marcata decrescita della loro loss, riducendo significativamente il rischio empirico totale: il modello potrebbe imparare maggiormente da questi errori, rispetto a campioni già ben classificati.

4.2 Procedura di Line Search generalizzata per il Calcolo del Learning Rate

Si consideri la versione mini-batch $\mathcal{N}_k$ del metodo SGD, (*Sezione 2.3.2*), dove

$$f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k}\sum_{i \in \mathcal{N}_k} f_i(\mathbf{w}) \qquad \text{con} \qquad N_k = |\mathcal{N}_k|.$$

Nella *Sezione 3.2* è descritta la procedura di Line Search con regola di sufficiente decrescita di Armijo, formulata nel caso particolare in cui la direzione di discesa $d_k = -g_k(\mathbf{w}^{(k)})$ è esattamente $-\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$. Tuttavia, modificando la probabilità di estrazione degli indici, il gradiente stocastico viene modificato.

Si consideri il gradiente stocastico (4.4) per la versione mini-batch, i cui indici sono estratti attraverso una nuova distribuzione differente da quella uniforme. La **procedura di Line Search con condizione Armijo** per la sufficiente decrescita è riportata nell'*Algoritmo 5*. Questo permette di determinare la lunghezza di passo α_k ad ogni iterazione, diminuendo il costo computazionale necessario per la ricerca degli iperparametri migliori. Infatti, come affermato nel *Capitolo 3*, quest'ultimo calcola α_k in modo adattivo, diminuendo la dipendenza delle performance dalla scelta degli iperparametri, fortemente presente seguendo la legge decrescente (4.2).

Algorithm 5 Stochastic Line Search Procedure per α_k [10]

Require: $\alpha_{\max} \geq 1$, $\beta \in (0,1)$, $\beta \in (0,1)$, $\gamma \in (0, \frac{1}{\alpha_{\max}})$, p^k probabilità di estrazione degli indici.

1: Si pone $\alpha := \alpha_{\max}$;

2: Si calcola il gradiente stocastico (4.4)

$$\mathbf{g}_k = \frac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$$

3: **while** $f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha \mathbf{g}_k) > f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma \alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} \mathbf{g}_k$ **do**

4: $\quad \alpha = \beta \alpha$;

5: **end while**

6: $\alpha_k = \alpha$;

7: **return** α_k.

In particolare, per ogni iterazione k, sarà scelta la lunghezza di passo

$$\alpha_k = \alpha_{\max} \beta^{\bar{m}},$$

dove $\bar{m}$ è il più piccolo intero che soddisfa

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha g_k) \leq f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\, \alpha_k\, \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} g_k.$$

4.3 Estensioni Adaptive Importance Sampling del Metodo del Gradiente Stocastico (SGD-AIS e Mini-Batch SGD-AIS)

Dato il problema di minimizzazione (3.1), una nuova versione del metodo del Gradiente Stocastico può essere ottenuta iterativamente dalla regola di aggiornamento

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k+1)} - \alpha_k \, g_k(\mathbf{w}^{(k)}),$$

in cui

- la lunghezza di passo α_k è scelta attraverso la *procedura di Line Search* descritta nella *Sezione 4.2*, che garantisce la **robustezza** del metodo rispetto alla scelta dei suoi iperparametri iniziali;
- gli indici estratti ad ogni iterazione seguono la distribuzione *Adaptive Importance Sampling (AIS)* (4.6), che, per come è costruita, permette di ridurre la varianza dello stimatore $g_k(\mathbf{w}^{(k)})$ di $\nabla f(\mathbf{w}^{(k)})$;
- il gradiente stocastico $g_k(\mathbf{w}^{(k)})$ è calcolato seguendo la formula (4.4).

In particolare, in *Algorithm 6* è riportato lo pseudocodice del metodo SGD in versione classica e mini-batch con dimensione m fissata, in cui sono compiute le modifiche precedenti: **Mini-Batch SGD-AIS** ($m \neq 1$) e **SGD-AIS** ($m := 1$, che elimina il ciclo *for* dello step 2). Scegliere la strategia mini-batch è vantaggioso in quanto è anch'esso un metodo per ridurre la varianza dello stimatore e, conseguentemente, permette di migliorare la **stabilità della convergenza** del metodo, diminuendo il comportamento oscillatorio della funzione rischio empirico dell'insieme di addestramento. Viceversa, se $m := 1$, le sue prestazioni saranno fortemente dipendenti dalla scelta casuale dell'unico campione, causando maggiori oscillazioni nei risultati.

4.4 Analisi Teorica: Ben Posizione e Risultati di Convergenza

In questa sezione sono riportati i teoremi relativi alla ben posizione e ad alcuni risultati del metodo *Mini Batch SGD-AIS*, in cui il gradiente stocastico è costruito

Algorithm 6 Mini-batch SGD-AIS (SGD-AIS se $m = 1$) [10]

Require: Numero di iterazioni $maxit$; numero di indici n; limiti del peso AIS $\{\xi_{\min}, \xi_{\max}\}$; dimensione del mini-batch m; lunghezza di passo iniziale α_0; parametri dell'*Algoritmo Line Search 5* $\{\alpha_{\max}, \beta, \gamma\}$.

Inizializzazione di $\mathbf{w}^{(0)}$, $\pi_i = 1$ per ogni $i \in \{1, \dots, n\}$

for $k = 0, 1, \dots, maxit$ **do**

Step 1: Aggiornamento della Distribuzione AIS

$\xi_k = \xi_{\min} + \dfrac{k}{maxit}(\xi_{\max} - \xi_{\min})$;

for $z = 1, 2, \dots, n$ **do**

$p_z^k = \xi_k \dfrac{\pi_z}{\sum_{j=1}^{n} \pi_j} + (1 - \xi_k)\dfrac{1}{n}$;

end for

Step 2: Selezione del Mini-Batch e Aggiornamento di π

$\mathcal{N}_k = \emptyset$;

for $j = 1, 2, \dots, m$ **do**

Si seleziona casualmente $i_j \in \{1, \dots, n\}$ secondo la distribuzione p^k;

Si aggiunge i_j a $\mathcal{N}_k$;

Si aggiorna π: $\pi_{i_j} = \|\nabla f_{i_j}(\mathbf{w}^{(k)})\|$;

end for

Step 3: Calcolo del Gradiente Stocastico

Se $m \neq 1$, $g_k = \dfrac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \nabla f_i(\mathbf{w}^{(k)})$; se $m = 1$, $g_k = \dfrac{1}{n\, p_i^k} \nabla f_i(\mathbf{w}^{(k)})$;

Step 4: Calcolo del Learning Rate

Calcolo di α_k tramite l'*Algoritmo Line Search 5*

Step 5: Aggiornamento di $\mathbf{w}^{(k)}$

$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k g_k$

end for

attraverso 4.4.

Le dimostrazioni corrispondenti sono presentate in [10].

4.4.1 Assunzioni

Si considerino le assunzioni 3.1.2, 3.1.3, già presentate nel capitolo precedente. Ad esse, si aggiungano quelle indicate successivamente.

Assunzione 4.4.1 *La sequenza* $\{\mathbf{w}_k\}_k \subset \Omega \subseteq \mathbb{R}^d$ *è un insieme compatto (equivalentemente, chiuso e limitato).*

Assunzione 4.4.2 *Si definisca lo scalare*

$$G := \max_{\mathbf{w}\in\Omega} \left\{ \max_{i=1,2,\dots,n} \|\nabla f_i(\mathbf{w})\| \right\} \quad \left(\Rightarrow \|\nabla f_i(\mathbf{w})\| \leq G \quad \forall i \in \{i = 1, 2, \dots, n\} \, \forall \mathbf{w} \in \Omega \right).$$

Esso è ben definito, ossia $G < +\infty$*, grazie alla compattezza di* Ω *ed alla continuità della funzione* $\mathbf{w} \mapsto \|\nabla f_i(\mathbf{w})\|$ *(Teorema di Weierstrass).*

Si ipotizzi l'esistenza di due costanti $\delta > 0$ *e* $\rho > 0$ *tali che valgono le seguenti disuguaglianze:*

$$\min_{\mathbf{w}\in\Omega} \left\{ \frac{1}{n} \sum_{i=1}^{n} \|\nabla f_i(\mathbf{w})\| \right\} \geq \delta G, \quad \min_{\mathbf{w}\in\Omega} \left\{ \frac{1}{2n^2} \sum_{j=1}^{n} \sum_{i=1}^{n} \left(\|\nabla f_i(\mathbf{w})\| - \|\nabla f_j(\mathbf{w})\| \right)^2 \right\} \geq \rho G^2.$$

In [33, Assumption 3, Section III], si afferma che è possibile dimostrare che l'*Assunzione 4.4.2* è verificata se

$$f_i(\mathbf{w}) = g(\mathbf{w}^\top \mathbf{x}_i) + \frac{\lambda}{2} \|\mathbf{w}\|^2,$$

dove g è una funzione loss continua e gli esempi $\mathbf{x}_i$ seguono una distribuzione gaussiana o uniforme.

Assunzione 4.4.3 *Si consideri* $N \geq n$*. Per ogni iterazione* $k \geq N$*, il campione* $j \in \{1, \dots, n\}$ *è presente nelle* N *estrazioni precedenti compiute seguendo la distribuzione ottimale (4.6), ossia*

$$\forall N \geq n \quad \forall j \in \{1, \dots, n\} \quad j \in \{i_{k-1}, i_{k-2}, \dots, i_{k-N}\}$$

dove i_m *rappresenta l'indice estratto all'iterazione* m*.*

Questa assunzione garantisce che ogni componente di π, definito in (4.7), sia aggiornata, considerando N iterazioni consecutive. Inoltre, in [33, Proposition B.1, Supplementary Material], si dimostra che questa è verificata con alta probabilità per N sufficientemente grande, tale che

$$N \geq 4n \log\left(\frac{n}{1-\xi_{\max}}\right).$$

4.4.2 Ben Posizione della Procedura di Line Search

Innanzitutto, si dimostri la ben posizione dell'algoritmo di Line Search con condizione di Armijo 5.

Lemma 4.4.4 ([10, Lemma 1. Sec. 4]) *Si consideri l'Assunzione 3.1.3, la procedura di line search descritta nell'Algoritmo 5, con probabilità di campionamento Adaptive Importance Sampling p^k (4.6), è ben definita e vale*

$$0 < \alpha_{\min} \leq \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)(1-\xi_{max})}{L_{\mathcal{N}_k}}\right\} \leq \alpha_k \leq \alpha_{\max}, \tag{4.8}$$

dove

$$\alpha_{\min} = \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)(1-\xi_{max})}{L_{\max}}\right\}.$$

La dimostrazione di questo lemma è riportata in *Appendice 7.6.3.*

4.4.3 Risultati di Convergenza

Riduzione della Varianza dello Stimatore. Innanzitutto, sotto le assunzioni descritte in precedenza, è possibile dimostrare che la scelta della distribuzione di probabilità ottimale (4.6) garantisce la *riduzione della varianza* dello stimatore gradiente stocastico, rispetto al caso classico che presenta la distribuzione uniforme: *Teorema 4.4.5.*

Teorema 4.4.5 ([33, Section III.A]) *Sotto le Assunzioni 3.1.3, 4.4.1,4.4.2, 4.4.3, sia $k \geq N$. Se la lunghezza di passo α_k è limitato da*

$$0 < \alpha_k < \bar{\alpha} \quad dove \quad \bar{\alpha} := \frac{(1-\xi_{max})^3 \xi_{min} \delta \rho}{(1-\xi_{max})^2 \xi_{min} NL\rho + \xi_{max} NL},$$

allora vale

$$\mathrm{Var}_{i\sim p^k}\left[\frac{1}{np_i^k}\nabla f_i(\mathbf{w}^{(k)})\right] \leq \mathrm{Var}_{i\sim\mathcal{U}}\left[\nabla f_i(\mathbf{w}^{(k)})\right] - \nu G^2,$$

dove $\mathcal{U}$ *è la distribuzione uniforme su* $\{1, 2, \ldots, n\}$ *e* $\nu \in (0, 1)$ *è lo scalare*

$$\nu = \frac{\xi_{max}\rho - \alpha L \alpha_k N}{(1 - \xi_{max})^3 \delta - (1 - \xi_{max})^2 L \alpha_k N}.$$

Ipotesi di Forte Convessità. Grazie al teorema precedente (*Teorema 4.4.5*) ed all'ipotesi di *forte convessità* della funzione obiettivo, il *Teorema 4.4.9* dimostra che il gap di ottimalità atteso converge *R-linearmente* ad una costante positiva.

Si ricordino il *Descent Lemma 3.2.1*, la *proprietà di Polyak-Lojasiewicz (3.9)* e la definizione di forte convessità di una funzione:

Definizione 4.4.6 *Sia una funzione* $f : \mathbb{R}^d \to \mathbb{R}$*. Essa è* ***c-fortemente convessa*** *se esiste una costante* $c > 0$ *tale che*

$$f(\mathbf{x}_1) \geq f(\mathbf{x}_2) + \nabla f(\mathbf{x}_2)^\top (\mathbf{x}_1 - \mathbf{x}_2) + \frac{c}{2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2, \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d.$$

Inoltre, è possibile dimostrare la seguente proprietà:

Proprietà 4.4.7 ([10, Property 3. Sec.. 4]) *Se una funzione* $f : \mathbb{R}^d \to \mathbb{R}$ *è inferiormente limitata da una costante* f^* *e c-fortemente convessa, allora è verificata la proprietà di Polyak-Lojasiewicz (3.9), ovvero*

$$\|\nabla f(\mathbf{w})\|^2 \geq 2c\big(f(\mathbf{w}) - f^*\big) \quad \forall \mathbf{w} \in \mathbb{R}^d,$$

Infine, ricordando il *lemma 3.2.10* e la proprietà (4.4.7), è facile dimostrare:

Proprietà 4.4.8 ([10, Property 3. Sec. 4]) *Sia una funzione* $f : \mathbb{R}^d \to \mathbb{R}$ *tale che* f *è c-fortemente convessa e* ∇f *è L-Lipschitz continuo; allora,*

$$c \leq L.$$

Grazie a questi risultati, è possibile dimostrare il risultato di convergenza per l'optimality gap $\mathbb{E}\big[f(\mathbf{w}^{(k)}) - f^*\big]$ atteso nel caso di **funzioni fortemente convesse**: *Teorema 4.4.9.*

Teorema 4.4.9 ([10, Theorem 2. Sec. 4]) *Date le Assunzioni 3.1.2, 3.1.3, 4.4.1, 4.4.2 e 4.4.3, sia* $\{\mathbf{w}^{(k)}\}_{k\in\mathbb{N}}$ *la sequenza determinata dall'algoritmo mini-batch SGD-AIS (Algorithm 6), dove* α_k *è ottenuto attraverso la procedura di line search (Algorithm 5). Supponendo che*

- *f sia c-fortemente convessa;*
- *il massimo valore assumibile da α_k soddisfi*

$$\alpha_{\max} < \tilde{\alpha} \quad dove \quad \tilde{\alpha} := \min\left\{\bar{\alpha}, \frac{1}{L}\right\}.$$

Allora vale

$$\mathbb{E}\big[f(\mathbf{w}^{(k)}) - f^\star\big] \leq (1 - c\alpha_{\min})^{k-N}\mathbb{E}\big[f(\mathbf{w}^{(N)}) - f^*\big] + M\frac{1 - (1 - c\alpha_{\min})^{k-N}}{c\alpha_{\min}},$$

dove

$$M = \frac{\alpha_{\max}^2}{2}(1-\nu)G^2, \qquad 0 < 1 - c\,\alpha_{\min} < 1,$$

e $\alpha_{\min} > 0$ è definito come (4.8).

Il *Teorema 4.4.9* esige che sia verificata una **forte assunzione su** $\alpha_{\max}$, difficilmente realizzabile a causa della sua dipendenza da molte costanti teoriche:

$$\alpha_{\max} < \min\left\{\bar{\alpha}, \frac{1}{L}\right\}.$$

Questa problematica è comune a molti risultati di convergenza per il gradiente stocastico, come DeepLISA (*Sezione 3.2.3*).

Inoltre, il *Teorema 4.4.9* permette di fare alcune considerazioni sul **tasso di convergenza** del metodo. In particolare, asintoticamente l'*optimality gap* è limitato da una costante $\tilde{M}$, che impedisce la convergenza a zero, similmente al caso di metodo SGD con lunghezza di passo fisso:

$$\lim_{k\to+\infty} \mathbb{E}[f(\mathbf{w}^{(k)}) - f^*] \leq \frac{M}{c\alpha_{\min}} = \alpha_{\max}\frac{(1-\nu)G^2}{2c\,\alpha_{\min}} := \tilde{M}.$$

Ciò è causato dalla procedura di line search per la determinazione della lunghezza di passo, che, secondo i risultati teorici di convergenza, risulta essere peggiore rispetto ad una legge che decresce rapidamente e garantisce la convergenza a zero. Tuttavia, è possibile abbassare tale soglia $\tilde{M}$ sfruttando la sua dipendenza dal fattore $\alpha_{\max}$: scegliendo uno scalare sufficientemente piccolo, è comunque possibile ridurre il gap, a discapito della velocità di convergenza effettiva.

Caso Non Convesso. Infine, è possibile ottenere un risultato di convergenza simile per *funzioni obiettivo non convesse*: la media delle aspettazioni delle norme al quadrato dei gradienti calcolati per la funzione stessa converge *sub-linearmente* ad una costante strettamente positiva.

Teorema 4.4.10 ([10, Theorem 3. Sec. 4.]) *Date le Assunzioni 3.1.2, 3.1.3, 4.4.1, 4.4.2 e 4.4.3, sia* $\{\mathbf{w}^{(k)}\}_{k\in\mathbb{N}}$ *la sequenza ottenuta dall'algoritmo mini-batch SGD-AIS (Algorithm 6), dove* α_k *è determinato con la procedura di line search (Algorithm 5). Se si ha*

$$\alpha_{\max} < \tilde{\alpha}, \quad \textit{dove} \quad \tilde{\alpha} := \min\left\{\bar{\alpha}, \frac{1}{L}\right\}$$

allora, per ogni $K \geq N$,

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\Big[\|\nabla f(\mathbf{w}^{(k)})\|^2\Big] \leq \frac{(K-N)\alpha_{\max}}{K\alpha_{\min}}G^2(1-\nu) + \frac{1}{K}(M_1+M_2),$$

dove

$$M_1 = \frac{2\mathbb{E}[f(\mathbf{w}^{(N)}) - f^*]}{\alpha_{\min}}, \qquad M_2 = NG^2.$$

Analizzando asintoticamente questa disuguaglianza, si ottiene

$$\lim_{K\to+\infty}\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\big[\|\nabla f(\mathbf{w}^{(k)})\|^2\big] \leq \alpha_{\max}\frac{(1-\nu)G^2}{\alpha_{\min}}.$$

Analogamente le considerazioni fatte per il caso di forte convessità, sia sull'assunzione relativa a $\alpha_{\max}$, sia per la mancata convergenza a zero, in questo caso, della norma del gradiente. Si osservi che anche in questo caso è possibile controllare questo limite attraverso una scelta opportuna di $\alpha_{\max}$.

Capitolo 5

DeepLISA-LISA: Variazioni su Dimensione dei Mini-Batch e Distribuzione di Campionamento

Nel *Capitolo 3*, si descrive l'*algoritmo DeepLISA 4*, variante di mini-batch SGD utilizzata per la risoluzione del problema di minimizzazione (3.1). Ad ogni epoca, insieme di iterazioni successive in cui sono estratti totalmente n indici, esso sfrutta la *classe Pytorch Data Loader*, che permette di ridurre al minimo la latenza della GPU, creando un flusso continuo di dati. In particolare, indicando con N_t la dimensione dei mini-batch scelta per l'epoca t, il training set $\mathcal{S}$ sarà partizionato in $\lceil \frac{n}{N_t} \rceil$ mini-batch disgiunti a coppie: siano $\left\{\mathcal{N}_{\bar{t}}, \mathcal{N}_{\bar{t}+1}, \ldots, \mathcal{N}_{\bar{t}+\lceil \frac{n}{N_t} \rceil - 1}\right\}$, i sottoinsiemi costruiti da Data Loader per l'epoca t, in cui $\bar{t}$ è l'indice della prima iterazione per l'epoca corrente, allora

$$\bigcup_{i=0}^{\lceil \frac{n}{N_t} \rceil - 1} \mathcal{N}_{\bar{t}+i} = \mathcal{S}, \quad \text{e} \quad \mathcal{N}_{\bar{t}+i} \cap \mathcal{N}_{\bar{t}+j} = \emptyset, \tag{5.1}$$

$$|\mathcal{N}_{\bar{t}+i}| = \begin{cases} N_t & i \in \{1, 2, \ldots, \lfloor \frac{n}{N_t} \rfloor\} \\ \text{resto di } \frac{n}{N_t} & \text{altrimenti} \end{cases} \tag{5.2}$$

per ogni $i, j \in \{0, 1, \ldots, \left\lceil \frac{n}{N_t} \right\rceil - 1\}$, con $i \neq j$.

In questo scenario, le estrazioni all'interno di un'epoca risultano dipendenti dalle precedenti, poiché l'assenza di reinserimento vincola la scelta dei campioni successivi.

Tale approccio si discosta dalle ipotesi teoriche di convergenza, che spesso assumono campionamenti completamente i.i.d. (indipendenti e identicamente distribuiti). Dunque, diventa interessante valutare se una differente strategia di campionamento possa influenzare il comportamento del metodo.

Inoltre, l'aggiornamento della dimensione dei mini-batch N_t avviene all'inizio di ogni epoca, secondo la legge predeterminata (3.18) che garantisce la riduzione della varianza del gradiente stocastico per ogni sua iterazione.

In particolare, si analizzeranno le seguenti modifiche:

(A) **campionamento con reinserimento tra mini-batch**: ridurre la dipendenza tra iterazioni successive, permettendo la ripetizione di campioni in batch distinti, pur mantenendo l'unicità degli elementi all'interno dello stesso batch;

(B) **aggiornamento della dimensione del mini-batch ad ogni iterazione**, modificando opportunamente la strategia di DeepLISA.

Notazione 5.0.1 *Nel seguito, $\mathcal{N}_k$ indicherà il mini-batch relativo all'iterazione k nel caso in cui la sua dimensione N_t è aggiornata all'inizio di ogni epoca t; al contrario, se l'aggiornamento avviene puntualmente, esso sarà denotato con $\mathcal{B}_k$, e B_k sarà la sua dimensione.*

Modifica (A). Sia *(A')* la procedura standard di costruzione dei mini-batch, presente in DeepLISA e descritta da (5.1). La *modifica (A)* introduce il campionamento con reinserimento degli indici già utilizzati in mini-batch precedenti. Tuttavia, anche in questo caso si impone che, all'interno dello stesso mini-batch, gli indici debbano essere distinti. Perciò, siano $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_m$ i mini-batch generati all'interno della stessa epoca, il processo di campionamento segue una distribuzione uniforme tale che

- potrebbe esistere un indice in comune tra mini-batch differenti, ossia $\mathcal{B}_i \cap \mathcal{B}_j \neq \emptyset$ per qualche $i \neq j$;
- all'interno di ogni batch $\mathcal{B}_i$, gli indici sono estratti uniformemente e senza ripetizioni.

Modifica (B). Sia *(B')* la regola di aggiornamento della dimensione del mini-batch, utilizzata all'inizio di ogni epoca t secondo la legge (3.18)

$$N_t := \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil ;\, N_0 \right\} \right\} \qquad \forall t \in \mathbb{N},$$

dove: $\{\varepsilon_k\}_k$ è una sequenza sommabile, positiva e non crescente; $C > 0$ è la costante che limita la varianza di ogni stimatore $\nabla f_i(\mathbf{w}^{(k)})$, per $i \in \mathcal{N}_k$; N_0 e $n_{\max}$ sono limitazioni inferiori e superiori. Questa è costruita per garantire ad ogni iterazione che la varianza dello stimatore sia ridotta secondo la disuguaglianza (3.3), verificata se la dimensione di ogni mini-batch soddisfa (3.15), ossia

$$|\mathcal{N}_k| \geq \frac{C}{\varepsilon_k} \qquad \text{per ogni iterazione } k.$$

Seguendo l'approccio iniziale dell'*algoritmo LISA*, in cui l'aggiornamento del batch size avviene ad ogni iterazione attraverso (3.16), la riduzione della varianza viene garantita imponendo che la successione sia debolmente crescente

$$B_k := \left\lceil \frac{C}{\varepsilon_k} \right\rceil,$$

dove, per ogni iterazione k,

- B_k rappresenta la dimensione scelta per il mini-batch $\mathcal{B}_k$;
- $\{\varepsilon_k\}_k$ è la stessa sequenza sommabile, positiva e non crescente definita per l'*algoritmo DeepLISA*;

Tuttavia, a differenza di *LISA (Algorithm 3)*, B_k viene determinato modificando direttamente la legge (3.18) utilizzata da *DeepLISA*. In particolare, si definisce

$$B_k := \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_k} \right\rceil ;\, B_0 \right\} \right\} \qquad \forall k \in \mathbb{N}_0. \tag{5.3}$$

Tale strategia eredita tutti i vantaggi enunciati per *DeepLISA*.

Innanzitutto, anch'essa garantisce il **totale controllo sulla dimensione dei mini-batch**. Infatti, a differenza di *LISA*, $\{B_k\}_k$ è deterministica e indipendente dalla sequenza di indici generata. Dunque, è possibile garantire una lenta crescita della dimensione del batch, più graduale rispetto a DeepLISA: fissata un'epoca t,

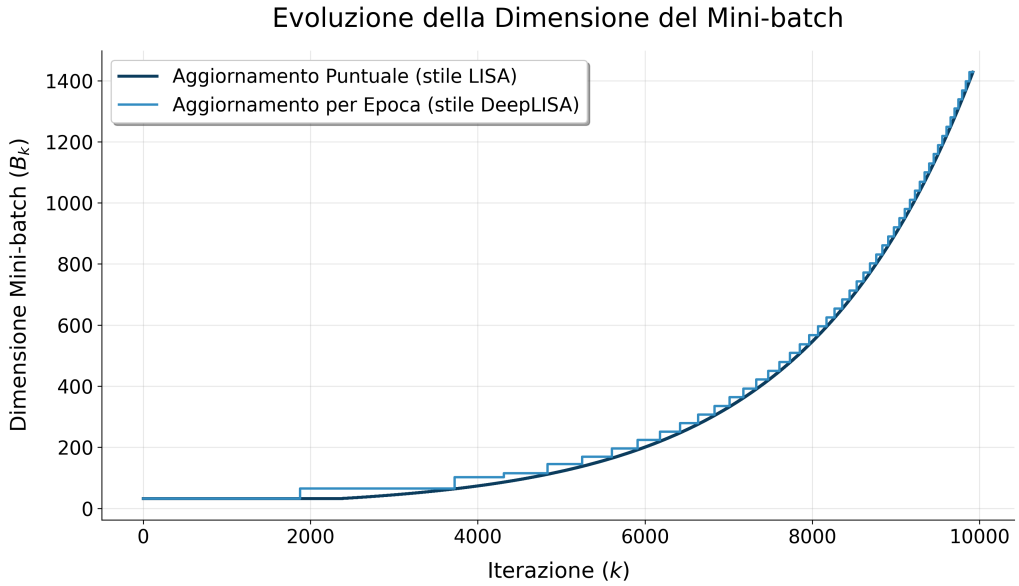


Figura 5.1: Confronto dell'evoluzione della dimensione dei mini-batch per iterazione: aggiornamento puntuale (stile LISA) contro aggiornamento per epoca (stile DeepLISA).

mentre DeepLISA utilizza un'unica dimensione, sufficientemente grande affinché si verifichi (3.15) anche per l'ultima iterazione, ma risultando potenzialmente eccessiva per quelle precedenti, la *modifica (B)* introduce un adattamento puntuale che determina una crescita graduale e ad hoc per ogni iterazione.

Inoltre, questo approccio potrebbe aumentare l'**efficienza computazionale** di DeepLISA. Infatti, poiché la dimensione del mini-batch aumenta più lentamente, il numero totale dei gradienti da calcolare, uno per campione, sarà inferiore rispetto a DeepLISA, riducendo il tempo di esecuzione delle prime epoche della fase di training, dove la differenza delle dimensioni è maggiormente evidente. La *Figura 5.1* mostra la crescita della dimensione dei mini-batch utilizzata per ogni iterazione, confrontando *Modifica (B)* con *Modifica (B')*.

Infine, dopo opportune modifiche, anch'essa sfrutterà la classe **Data Loader** di PyTorch per la creazione dei mini-batch. Attraverso la sua natura **multiprocessing** (parametro *'num_workers'* maggiore di 0), è possibile parallelizzare la preparazione dei mini-batch sulla CPU, garantendo un flusso di dati continuo verso la GPU e, di conseguenza, riducendo i suoi tempi di inattività. In questo caso, al fine di creare ogni mini-batch, sarà aggiornata la dimensione B_k richiesta e saranno campionati uniformemente i corrispondenti indici senza reinserimento.

L'implementazione di questa variante viene effettuata considerando un **unico** *Data Loader PyTorch*, in grado di generare una successione di mini-batch $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K\}$, dove K è il numero di iterazioni complessive, tali che:

- sia rispettata una tra le *Modifiche (A)* e *(A')*;
- ogni mini-batch $\mathcal{B}_k$ abbia dimensione B_k, ottenuta attraverso la legge (5.3).

Questo è possibile grazie alla determinazione a priori di $\{B_k\}_k$, indipendente da risultati ottenuti durante il processo.

Algorithm 9 formalizza lo pseudocodice per **DeepLISA con Aggiornamento Puntuale del Batch Size**.

5.1 Varianti del metodo DeepLISA

Di seguito sono riportate alcune varianti di *DeepLISA (Algorithm 4)*, metodi del Gradiente Stocastico dotati di Line Search Procedure che differiscono dall'algoritmo originale secondo le modifiche dichiarate in precedenza.

5.1.1 DeepLISA con Mini-Batch Non Disgiunti

Innanzitutto, si consideri l'algoritmo *DeepLISA* dotato di *Modifica (A)* sulla distribuzione di campionamento degli indici. In questo caso, lo pseudocodice è analogo a quello proposto per *DeepLISA (Algorithm 4)*, con regola di aggiornamento della dimensione dei mini-batch per epoca (3.18) (*condizione (B')*). Tuttavia, la sua implementazione prevede una modifica del campionatore (*Batch Sampler PyTorch*), che sostituisce la classica partizione degli indici $\{1, 2, \ldots, n\}$ all'interno di *Data Loader*. Per ogni epoca t, il nuovo campionatore genera $\lceil \frac{n}{N_t} \rceil$ gruppi di indici di dimensione (5.2), in cui

- ogni gruppo è estratto uniformemente senza ripetizioni interne;
- l'estrazione di ogni gruppo è indipendente dalle altre.

Per emulare Data Loader standard, che considera esattamente n indici in un'epoca, l'ultimo batch avrà cardinalità pari al resto della divisione $\frac{n}{N_t}$, se questo è maggiore di zero (parametro *'drop_last* impostato come *'False'*).

Lo pseudocodice di questa prima variante è presentato in *Algorithm 10*.

5.1.2 DeepLISA con Aggiornamento Puntuale del Batch Size

In questa seconda variante, si applica la *Modifica (B)* descritta nel paragrafo precedente. A differenza della versione originale, l'algoritmo non attende il termine dell'epoca per modificare la dimensione dei mini-batch, ma esegue un aggiornamento dinamico di B_k ad ogni iterazione k. Questo approccio permette un *aumento graduale del carico computazionale*, garantendo che la condizione di riduzione della varianza sia soddisfatta con la minima dimensione di batch necessaria in ogni istante.

L'implementazione di questa variante viene effettuata considerando un unico *Data Loader PyTorch*, in grado di generare una successione di mini-batch $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_K\}$, dove K è il numero di iterazioni complessive. Questi sono costruiti attraverso una sequenza di partizioni del training set $\mathcal{S}$, in modo tale che, analogamente al Data Loader standard, gli indici di una stessa epoca saranno estratti casualmente senza reinserimento. Una volta terminati gli elementi di una partizione, si darà inizio all'epoca successiva, generandone una differente. In particolare, il *Batch Sampler PyTorch* corrispondente compie le seguenti operazioni:

- **Fase 1.** Si genera una permutazione $\mathcal{P}(\mathcal{S})$ dell'insieme di indici $\{1, 2, \ldots, n\}$;
- **Fase 2.** Ad ogni iterazione, si calcola la dimensione B_k del mini-batch, il quale viene costruito attraverso gli indici dalla partizione non ancora utilizzati; se la quantità disponibile è troppo piccola, si ripete la *Fase 1.*

Questa procedura è formalizzata in *Algorithm 8*, dove, per semplicità, si farà riferimento all'opzione *'drop_last'* impostata su *'True'*.

La seconda variante di *DeepLISA*, dotata di *modifica (B)*, è descritta in *Algorithm 9*, combinato con il Data Loader *Algorithm 8*.

5.1.3 DeepLISA con Mini-Batch Indipendenti e Aggiornamento Puntuale del Batch Size

L'ultima variante analizzata nasce dalla combinazione delle due strategie precedentemente descritte, integrando sia la *Modifica (A)* sulla distribuzione di campionamen-

to, sia la *Modifica (B)* relativa all'aggiornamento della dimensione dei mini-batch. La prima scelta permette di avvicinarsi maggiormente all'ipotesi di indipendenza statistica (i.i.d.), eliminando l'utilizzo di partizioni del training set, mentre la seconda dovrebbe migliorare il tempo di esecuzione.

Dal punto di vista implementativo, si definirà un unico *Data Loader* che non utilizzerà più le permutazioni degli indici (*Fase 1* della variante precedente). Al contrario, il suo *Batch Sampler* eseguirà i seguenti passaggi, per ogni iterazione k:

1. calcolo della dimensione puntuale B_k secondo la legge (5.3);
2. estrazione uniforme di B_k indici distinti appartenenti a $\{1, 2, \ldots, n\}$, indipendentemente dalle estrazioni effettuate nelle iterazioni precedenti.

Lo pseudocodice che formalizza questo procedimento è riportato in *Algorithm 7*. Esso, verrà utilizzato all'interno di *Algorithm 9* per la costruzione di questa terza variante.

Algorithm 7 Data Loader PyTorch per DeepLISA con Mini-Batch Indipendenti e Aggiornamento Puntuale del Batch Size

Require: Numero totale di iterazioni K; training set $\mathcal{S}$; $\{\varepsilon_k\}_k$ successione sommabile, positiva, non crescente; $C > 0$; vincoli per il batch size $\{B_0, n_{\max}\}$.

1: **for** $k = 1, 2, \ldots, K$ **do**
2: Calcolo di B_k, dimensione per il mini-batch $\mathcal{B}_k$:

$$B_k = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_k} \right\rceil ; B_0 \right\} \right\};$$

3: Estrazione B_k indici distinti, estratti uniformemente da $\{1, 2, \ldots, n\}$, dove $n = |\mathcal{S}|$;
4: Creazione di $\mathcal{B}_k$ attraverso gli indici estratti.
5: **end for**
6: **return** $\mathcal{B}_1$, $\mathcal{B}_2$,..., $\mathcal{B}_K$

Algorithm 8 Data Loader PyTorch per DeepLISA con aggiornamento puntuale del batch size

Require: Numero totale di iterazioni K; training set $\mathcal{S}$; $\{\varepsilon_k\}_k$ successione sommabile, positiva, non crescente; $C > 0$; vincoli per il batch size $\{B_0,\, n_{\max}\}$.

Fase 1: Inizializzazione della Sequenza di Indici

Si genera una permutazione $\mathcal{P}(\mathcal{S})$ dell'insieme di indici $\{1, 2, \ldots, n\}$;

Fase 2: Creazione dei Mini-Batch

for $k = 1, 2, \ldots, K$ **do**

Calcolo di B_k, dimensione per il mini-batch $\mathcal{B}_k$:

$$B_k = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_k} \right\rceil ;\, B_0 \right\} \right\};$$

if $B_k > |\mathcal{P}(\mathcal{S})|$ **then** ▷ Se i campioni residui sono insufficienti

Fase 1: Inizializzazione della Sequenza di Indici

Reinizializzazione della permutazione $\mathcal{P}(\mathcal{S})$ (nuova epoca);

end if

Estrazione di $\mathcal{B}_k$ dai primi B_k indici di $\mathcal{P}(\mathcal{S})$;

$\mathcal{P}(\mathcal{S})$ = rimozione da $\mathcal{P}(\mathcal{S})$ dei B_k indici utilizzati.

end for

return $\mathcal{B}_1$, $\mathcal{B}_2$,..., $\mathcal{B}_K$

Algorithm 9 Varianti 2, 3: Algoritmo DeepLISA con Aggiornamento Puntuale della Dimensione dei Mini-Batch

Require: $K > 0$, numero totale di iterazioni, $n_{\max} > 0$, $\mathbf{w}^{(0)} \in \mathbb{R}^d$, $0 < B_0 < n$, $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$, $\beta, \beta_2 \in (0,1)$, $\gamma > 0$, $M > 0$, una sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$, $C > 0$, $\bar{t} = 0$.

Step 1: Creazione dei Mini-Batch Size

Generazione dei K mini-batch attraverso

- *Algorithm 8*, se *Modifica (A')* $\rightarrow$ variante 2, **DeepLISA con Aggiornamento Puntuale del Batch Size**;
- *Algorithm 7*, se *Modifica (A)* $\rightarrow$ variante 3, **DeepLISA con Mini-Batch Indipendenti e Aggiornamento Puntuale del Batch Size**.

for $k = 1, 2, \ldots, K$ **do**

Si sceglie il mini-batch $\mathcal{B}_k$ di cardinalità B_k.

Si calcola $f_{\mathcal{B}_k}(\mathbf{w}^{(k)})$ e il gradiente stocastico $\nabla f_{\mathcal{B}_k}(\mathbf{w}^{(k)})$;

Step 2: Line Search Procedure per la Scelta del Learning Rate

Si calcola $\bar{\mathbf{w}} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{B}_k}(\mathbf{w}^{(k)})$;

while $f_{\mathcal{B}_k}(\bar{\mathbf{w}}) > \max_{0\le j\le \min\{k,M\}} f_{\mathcal{B}_{k-j}}(\bar{\mathbf{w}}^{(k-j)}) - \gamma\alpha_k \|\nabla f_{\mathcal{B}_k}(\mathbf{w}^{(k)})\|^2$ **do**

$\alpha_k = \beta\alpha_k$

end while

Step 3: Aggiornamento per l'Iterazione Successiva

Si pone

$$\mathbf{w}^{(k+1)} = \bar{\mathbf{w}};$$
$$\alpha_{k+1} = \min\left\{\alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min}\right)\right\}.$$

end for

Algorithm 10 Variante 1: Algoritmo DeepLISA con Mini-Batch Non Disgiunti

Require: $T > 0$, $n_{\max} > 0$, $\mathbf{w}^{(0)} \in \mathbb{R}^d$, $0 < N_0 < n$, $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$, $\beta, \beta_2 \in (0,1)$, $\gamma > 0$, $M > 0$, una sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$, $C > 0$, $\bar{t} = 0$.

1: **for** $t = 1, 2, \ldots, T$ **do**

Step 1: Scelta della Dimensione e Creazione dei Mini-Batch Size

2: Scelta di N_t come

$$N_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\};$$

3: Generazione $\left\lceil \frac{n}{N_t} \right\rceil$ mini-batch di dimensione N_t, secondo la *Modifica (A)*;

4: **for** $k = \bar{t}, \ldots, \bar{t} + \left\lceil \frac{n}{N_t} \right\rceil - 1$ **do**

5: Si sceglie il mini-batch $\mathcal{N}_k$ di cardinalità N_t.

6: Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e il gradiente stocastico $\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;

Step 2: Line Search Procedure per la Scelta del Learning Rate

7: Si calcola $\bar{\mathbf{w}} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;

8: **while** $f_{\mathcal{N}_k}(\bar{\mathbf{w}}) > \max_{0\le j\le \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\bar{\mathbf{w}}^{(k-j)}) - \gamma\alpha_k \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2$ **do**

9: $\alpha_k = \beta\alpha_k$

10: **end while**

Step 3: Aggiornamento per l'Iterazione Successiva

11: Porre

$$\mathbf{w}^{(k+1)} = \bar{\mathbf{w}};$$
$$\alpha_{k+1} = \min\left\{ \alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min} \right) \right\}.$$

12: **end for**

13: Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.

14: **end for**

5.2 Analisi dei Risultati

Impostazioni Base dei Modelli. Le varianti proposte in precedenza sono confrontate con l'algoritmo *DeepLISA* originale, attraverso l'analisi delle prestazioni ottenute sul dataset di letteratura *CIFAR10*[1]. Questo problema su larga scala di classificazione multiclasse di immagini viene affrontato attraverso la rete neurale *ResNet18*, descritta nella *Sezione 1.2.3*. Siccome l'architettura originale è progettata per immagini di dimensione $224 \times 224 \times 3$, è necessario apportare alcune modifiche per adattarla al caso $32 \times 32 \times 3$ di CIFAR10, proposte dall'articolo [13]:

- rimozione del primo strato di Max-Pooling;
- riduzione del primo strato convoluzionale: da 64 filtri 7×7 e stride 2, a 64 filtri 3×3 con stride pari a 1.

Inoltre, a causa del grande numero di parametri da addestrare, pari a 11 169 162, è necessario adottare alcune strategie per prevenire l'overfitting del processo [13]:

- aggiunta del parametro di regolarizzazione L_2, con termine $\lambda := 5 \cdot 10^{-4}$;
- applicazione di tecniche di *Data Augmentation* sulle immagini (Random Orizzontal Flip, Random Crop e Normalizzazione).

Infine, è stata rimossa la Batch-Normalization.

Gli iperparametri utilizzati per tutti i metodi, derivanti dal fine-tuning di [13], saranno: $C = 10$, $\varepsilon_k = 0.9995^k$, $\beta = 0.5$, $\beta_2 = \frac{1}{3}$, $\alpha_0 = \alpha_{\max} = 1$, $M = 1$, $\gamma = 1$ e $N_0 = B_0 = 32$.

Strategia di Confronto dei Risultati. Per misurare la **stabilità del metodo**, ogni processo è stato ripetuto 5 volte sul test set, variando il seme che rende deterministica la stocasticità del processo. Per ogni epoca, i risultati saranno visualizzati attraverso

- la *media* m delle accuratezze ottenute;

[1] Il *dataset CIFAR10* [20] contiene immagini 32×32 in scala RGB (3 canali), suddivise in 10 classi. Esso è costituito da 60 000 immagini: 50 000 campioni per l'insieme di addestramento; 10 000 campioni per l'insieme di test.

- la *banda della deviazione standard* σ, rappresentata dall'intervallo $[m - \sigma, m + \sigma]$, che serve per visualizzare la dispersione dei risultati.

L'ampiezza della banda permette di misurare la stabilità del metodo: minore sarà la dispersione, maggiore sarà la stabilità del metodo, il quale risulta capace di fornire prestazioni tendenzialmente analoghe, indipendentemente dalla stocasticità del processo, ossia dalla modifica dei seed.

Le accuratezze saranno confrontate rispetto al **tempo di esecuzione**, in quanto questo permette di valutare contemporaneamente due aspetti fondamentali:

- **efficienza del traffico di memoria**, che rappresenta l'impatto della quantità di volte in cui si accede al dataset all'interno della GPU;
- **velocità di esecuzione**, che mostra il risparmio computazionale dato dal calcolo dei gradienti su un numero minore di dati per ogni iterazione delle *varianti con aggiornamento puntuale del batch size*.

Per l'algoritmo *DeepLISA* originale e la sua *variante con mini-batch non disgiunti* (*Modifica (A), Algorithm 10*) saranno considerate 50 epoche, che corrispondono a 9 562 iterazioni per le varianti con *aggiornamento puntuale della dimensione del mini-batch* (*Modifica (B), Algorithm 9*). Siccome queste ultime non presentano il concetto di epoca come partizione del dataset, si è scelto di valutare le prestazioni sul test set nelle iterazioni corrispondenti a quelle dei test nei metodi precedenti, per garantire la confrontabilità rispetto al numero di iterazioni.

Gli esperimenti di questo capitolo sono stati eseguiti su un sistema avente CPU Intel i7-12700K (20 core), 32 GB di RAM e una GPU NVIDIA GeForce RTX 3060 Ti.

Le analisi svolte sono strutturate attraverso tre differenti paragrafi:

1. approfondimento sulla differenza temporale data dalla strategia di aggiornamento della dimensione del mini-batch size;
2. confronto delle accuratezze medie, con banda relativa alla deviazione standard, rispetto alle epoche considerate;
3. medesimo confronto rispetto al tempo medio di esecuzione sulle cinque ripetizioni dei processi.

Analisi del Tempo Computazionale per Epoca (Modifica (B) contro Modifica (B')). Innanzitutto, sono stati analizzati i tempi di esecuzione per le prime 50 epoche, confrontando l'algoritmo *DeepLISA* classico (*Modifica (B')*) con la seconda variante, avente aggiornamento puntuale della dimensione del mini-batch (*Modifica (B), stile LISA*), descritta in *Sezione 5.1.2*).

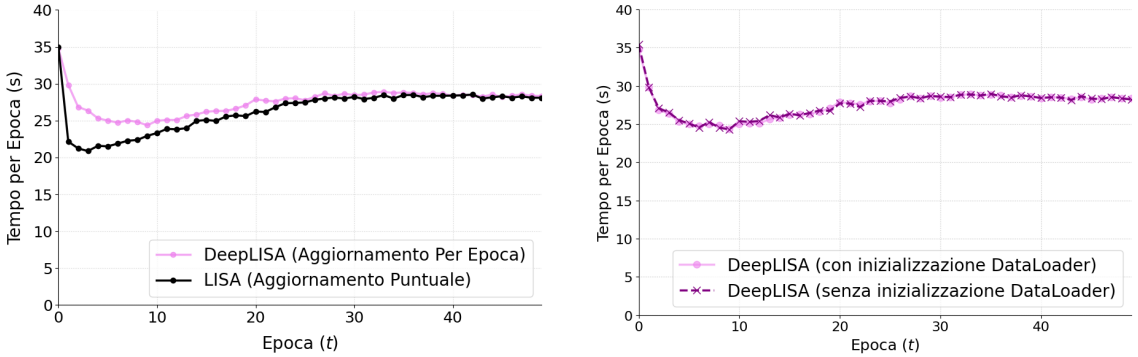
La *Figura 5.2a* conferma l'**aumento dell'efficienza computazionale** preventivato, dato dall'approccio in stile LISA (*Modifica (B)*). Nonostante l'utilizzo di batch di piccole dimensioni aumenti tipicamente il numero di accessi al dataset in memoria, e dunque il tempo di esecuzione legato a questa operazione [13], l'algoritmo proposto mantiene inalterato il numero di iterazioni totali per "epoca", i cui campioni non ricoprono tutto il dataset. Quindi, il risparmio temporale è dato dal minor numero di campioni processati durante il *backward pass*. Infatti, poiché la dimensione del mini-batch cresce più lentamente rispetto al metodo classico (*Figura 5.1*), il numero totale di gradienti da calcolare è inferiore, soprattutto nelle prime epoche, dove in effetti si riscontra un beneficio maggiore (*Figura 5.2a*).

Tuttavia, considerando le prime 50 epoche, si osserva che il miglioramento medio, calcolato su 5 seed, è di soli 62.83 *secondi*, come riportato in *Tabella 1.1*.

Infine, si osservi che tale riduzione è dovuta unicamente alla scelta di considerare le epoche dettate dallo stesso numero di iterazioni di *DeepLISA* e non all'inizializzazione di un unico Data Loader, ulteriore modifica apportata sulle varianti aventi *Modifica (B)*. Infatti, come evidenziato nella *Figura 5.2b*, escludendo i tempi di inizializzazione dall'*algoritmo DeepLISA*, il suo tempo computazionale rimane invariato.

Metodo	**Tempo Medio (s)**	**Dev. Std (s)**	**Risparmio (s)**
DeepLISA	1381.29	6.93	-
LISA (Variante 2)	1318.46	5.90	62.83

Tabella 5.1: Confronto dei tempi di esecuzione medi su 50 epoche, ripetute con 5 semi differenti. Il risparmio è calcolato come differenza temporale tra la versione originale e la variante LISA.



(a) Confronto DeepLISA vs LISA.

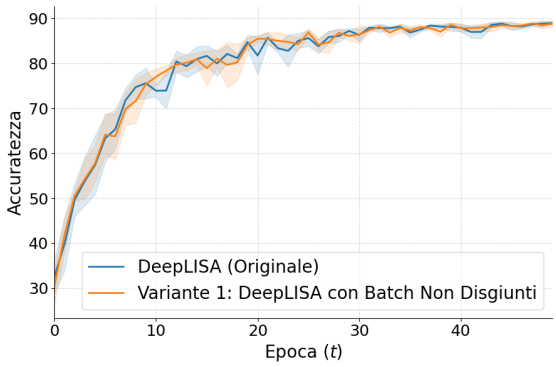
(b) Impatto del DataLoader in DeepLISA.

Figura 5.2: Analisi dei tempi di esecuzione per epoca: (a) risparmio del tempo di esecuzione della strategia LISA (aggiornamento puntuale della dimensione del mini-batch) rispetto a DeepLISA (aggiornamento per epoca della dimensione del mini-batch); (b) verifica del tempo necessario per l'inizializzazione dei Data Loader.

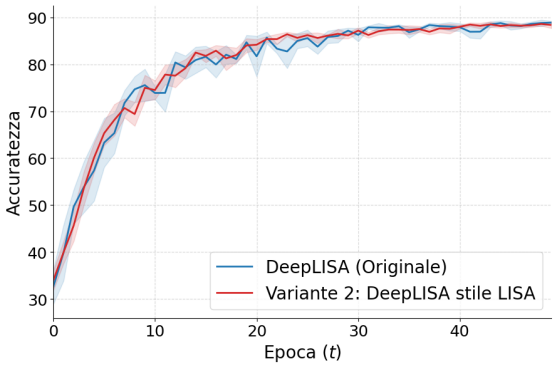
Analisi delle Accuratezze per Epoca. Siccome le varianti di DeepLISA con aggiornamento puntuale della dimensione del mini-batch (*Modifica (B)*) sono costituite da epoche aventi meno indici rispetto alla cardinalità del training set, è necessario stabilire quanto ciò influenzi le **prestazioni finali per ogni epoca**. Essa è rappresentata dallo stesso numero di iterazioni compiute dall'algoritmo originale. Analogamente, è significativo osservare il comportamento ottenuto dall'eliminazione della dipendenza delle estrazioni, se appartenenti a batch differenti (*Modifica (A)*).

Osservando le *Figure 5.3a - 5.3b - 5.3c*, si può notare che le prestazioni seguono tendenzialmente l'andamento ottenuto dall'algoritmo originale. Di conseguenza, le *Modifiche (A)-(B)* non peggiorano, nè migliorano, le prestazioni del modello in modo evidente. In particolare, pur non considerando tutte le informazioni del training set, un numero inferiore di campioni per iterazione, tipico delle varianti di tipo *LISA*, è comunque sufficiente per ottenere le stesse buone prestazioni (*Figure 5.3b - 5.3c*).

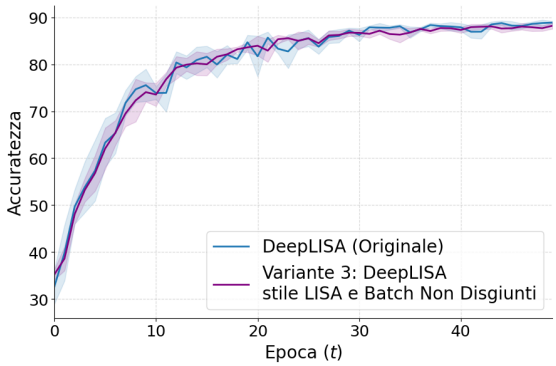
Inoltre, è possibile analizzare la **stabilità delle varianti di DeepLISA** attraverso l'ampiezza della banda determinata dalla deviazione standard. Innanzitutto, si consideri la variante DeepLISA in cui i batch non sono disgiunti (*Modifica (A)*). Dalla *Figura 5.3a*, si nota che la sua varianza è molto ridotta nelle prime epoche. Tuttavia, sono le *varianti stile LISA* che garantiscono una stabilità maggiore per l'intero processo (*Figure 5.3b - 5.3c*).



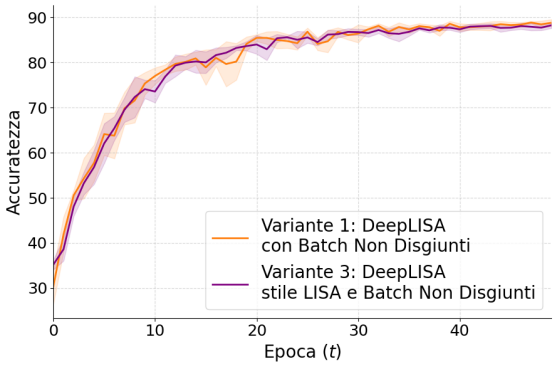
(a) DeepLISA vs Variante 1.



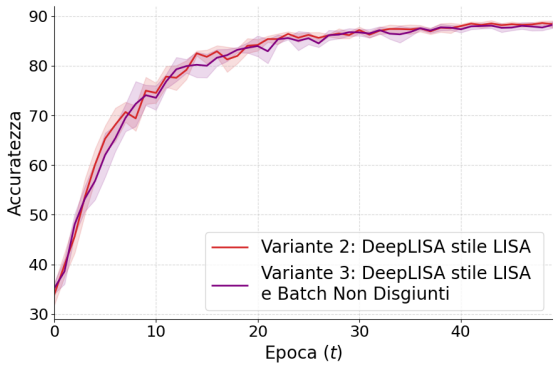
(b) DeepLISA vs Variante 2.



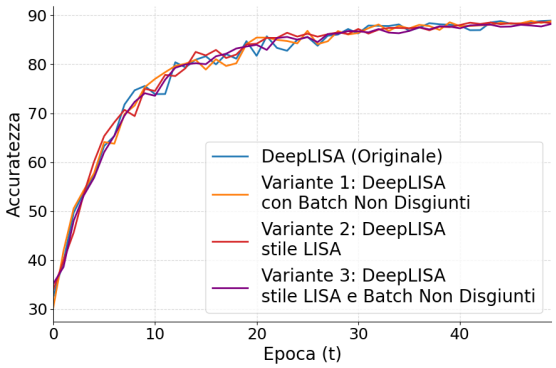
(c) DeepLISA vs Variante 3.



(d) Varianti con batch non disgiunti.

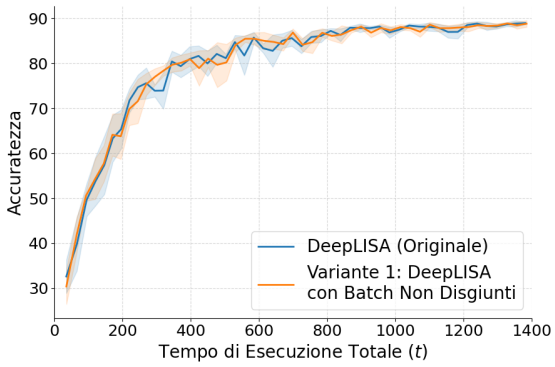


(e) Varianti stile LISA.

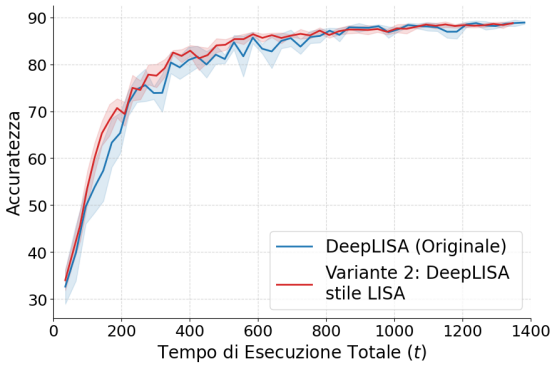


(f) Confronto della media delle accuratezze.

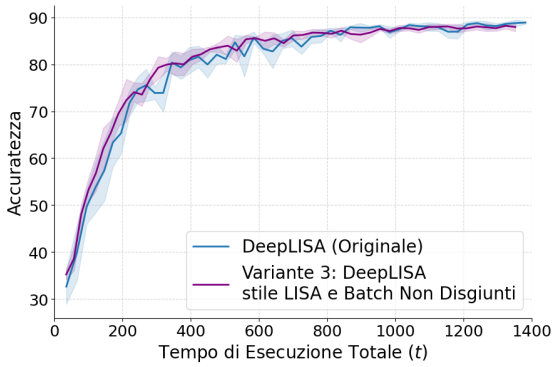
Figura 5.3: Analisi dell'accuratezza (%) media per epoca con bande di deviazione standard. I dati sono stati ottenuti attraverso la ripetizione degli algoritmi con 5 semi differenti.



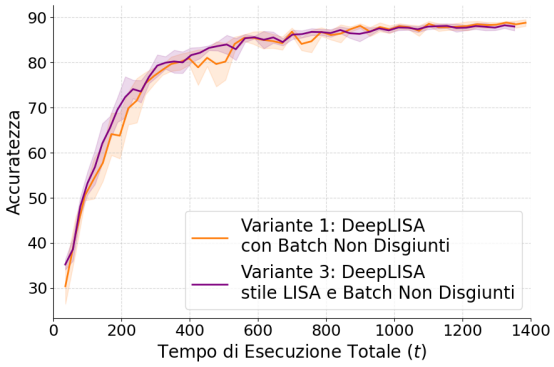
(a) DeepLISA vs Variante 1.



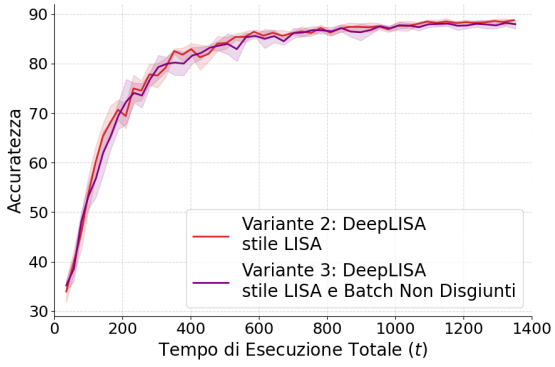
(b) DeepLISA vs Variante 2.



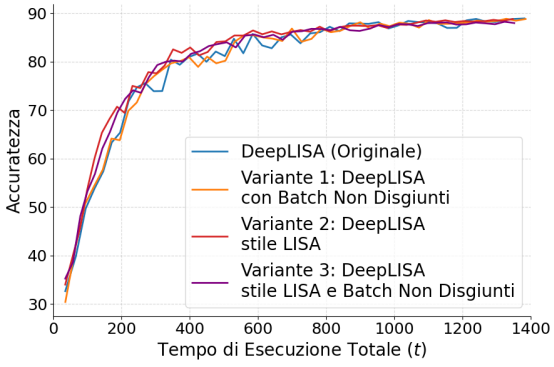
(c) DeepLISA vs Variante 3.



(d) Varianti con batch non disgiunti.



(e) Varianti stile LISA.



(f) Confronto della media delle accuratezze.

Figura 5.4: Analisi dell'accuratezza (%) media per tempo medio di esecuzione, con bande di deviazione standard. I dati sono stati ottenuti attraverso la ripetizione degli algoritmi con 5 semi differenti.

Infine, si analizzi il **comportamento oscillatorio** delle accuratezze. In generale, le *varianti LISA* tendono ad avere meno oscillazioni dell'accuratezza rispetto a DeepLISA, soprattutto nelle epoche in cui la crescita diventa ridotta (*Figure 5.3b - 5.3c*). Dunque, in qualsiasi epoca avvenga l'arresto, è garantito ottenere risultati simili a quelli delle epoche vicine. In forma meno evidente, anche la prima variante sembra avere meno oscillazioni nella fase di crescita dell'accuratezza (*Figura 5.3a*).

Analisi delle Accuratezze per Tempo Computazionale. Infine, per confrontare la bontà dei metodi, è necessario analizzare le **prestazioni degli algoritmi rispetto al tempo di esecuzione**. Siccome i tempi per iterazione di uno stesso algoritmo sono molti simili tra loro, per ogni epoca sarà considerato il tempo medio di esecuzione delle 5 ripetizioni eseguite. Inoltre, ricordando che le varianti di DeepLISA con *Modifica (B)* sono più veloci, saranno valutate 9645 iterazioni, equivalenti a 51 epoche, ottenendo lo stesso intervallo temporale degli altri metodi.

Come dichiarato in precedenza, la bontà delle varianti di DeepLISA è confrontabile con quella dell'algoritmo originale. Dunque, osservando le *Figure 5.4b - 5.4c*, è evidente che la velocità della **strategia LISA** giochi un ruolo fondamentale nelle prestazioni dei metodi. Infatti, grazie al risparmio temporale iniziale ed alla loro stabilità nelle oscillazioni, hanno prestazioni iniziali migliori. Tuttavia, nelle fasi finali, è evidente che solo la seconda variante, con mini-batch disgiunti tra loro, eguagli DeepLISA nei risultati. Attraverso il confronto diretto, presentato in *Figura 5.4e*, è possibile stabilire che l'aggiornamento puntuale della dimensione del batch avente *Modifica (A')*, ossia campionamento senza reinserimento, è tendenzialmente migliore in tutto il processo. Dunque, utilizzando meno campioni per iterazione, è necessario disporre del maggior numero di informazioni possibili, distinte tra loro.

Al contrario, questa differenza non è percepita tra i metodi con ***Modifica (B')***, in cui per ogni epoca sono presenti n indici (*Figura 5.4a*). Inoltre, siccome temporalmente simili, non è presente alcun miglioramento dettato dalla velocità per epoca.

Infine, è possibile confrontare le accuratezze medie di ogni metodo proposto all'interno di questo capitolo (*Figura 5.4f*): mediamente la strategia *LISA* (*Modifica (A')-(B)*) garantisce risultati migliori, mentre l'*algoritmo DeepLISA* accresce la

propria bontà solo successivamente. Tuttavia, quest'ultimo è l'unico in grado di toccare punti leggermente superiori di accuratezza nelle epoche finali.

Capitolo 6

Algoritmo DeepLISA con Distribuzione Adaptive Importance Sampling

Ricordando le tecniche di riduzione della varianza alla base degli algoritmi *DeepLISA* (*Capitolo 3*) e *Mini-Batch SGD-AIS* (*Capitolo 4*), è possibile sviluppare un metodo ibrido, che sfrutti entrambe le strategie.

6.1 Algoritmo DeepLISA-AIS e Pseudocodice

L'**algoritmo DeepLISA-AIS** è un *metodo del Gradiente Stocastico (versione Mini-Batch)* sviluppato per risolvere il problema di minimizzazione del rischio empirico (*formulazione 3.1*), eventualmente regolarizzato, per la ricerca dei parametri ottimali dei modelli di Deep Learning. La regola di aggiornamento iterativo che permette di generare la sequenza di parametri del modello $\{\mathbf{w}^{(k)}\}_k$ sarà

$$\mathbf{w}^{(k+1)} := \mathbf{w}^{(k)} - \alpha_k g_{\mathcal{N}_k}(\mathbf{w}^{(\mathbf{k})})$$

dove, fissata l'iterazione $k \in \mathbb{N}_0$, $-g_{\mathcal{N}_k}(\mathbf{w}^{(\mathbf{k})})$ rappresenta lo stimatore della direzione di discesa $-\nabla f(\mathbf{w}^{(\mathbf{k})})$. Esso agisce sulla varianza di $g_{\mathcal{N}_k}(\mathbf{w}^{(\mathbf{k})})$ attraverso due strategie che intervengono su due aspetti differenti della sua costruzione:

1. la **crescita progressiva della dimensione del mini-batch**, aumentando la quantità di informazioni utilizzate per $g_{\mathcal{N}_k}(\mathbf{w^{(k)}})$ (*riduzione quantitativa della varianza*);

2. fissata una dimensione del mini-batch, scegliere i campioni migliori attraverso una **distribuzione di campionamento non uniforme**, che approssima quella ottimale per la minimizzazione della varianza stessa nel caso di un singolo campione, come affermato nel *Teorema 4.4.5*, (*riduzione qualitativa della varianza*).

L'azione combinata di questi due metodi permette una riduzione progressiva della varianza dello stimatore di *DeepLISA-AIS*.

In particolare, questo metodo è sviluppato in modo tale che

- $\mathcal{N}_k \subseteq \mathcal{S}$ è un sottoinsieme di campioni estratti dal training set $\mathcal{S}$, secondo la **distribuzione Adaptive Importance Sampling** p^k (*Sezione 4.1, legge (4.6)*);

- N_k, cardinalità dell'insieme $\mathcal{N}_k$, sarà costruita attraverso la strategia di aggiornamento ad ogni epoca descritta in **DeepLISA** (*Sezione 3.3, legge (3.18)*);

- il learning rate α_k è uno scalare positivo scelto attraverso la **procedura di Line Search con Regola di Backtracking di Armijo**, adattata al caso in cui l'estrazione degli indici non avviene uniformemente (*Sezione 4.2*);

- $f_{\mathcal{N}_k} : \mathbb{R}^d \to \mathbb{R}$ rappresenta il rischio empirico costruito su $\mathcal{N}_k$, eventualmente regolarizzato,

  $$f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k} \sum_{i \in \mathcal{N}_k} f_i(\mathbf{w}) \qquad \left(\text{risp.} \quad f_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{N_k} \sum_{i \in \mathcal{N}_k} f_i(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \right),$$

  il cui **gradiente stocastico**, stimatore non distorto di $\nabla f(\mathbf{w}^{(k)})$, segue la formulazione (4.4), i.e.

  $$g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) \quad \left(\text{risp.} \quad g_{\mathcal{N}_k}(\mathbf{w}) = \frac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) + \lambda \, \mathbf{w}^{(k)} \right).$$

L'*algoritmo DeepLISA-AIS* è formalizzato attraverso lo pseudocodice *Algorithm 12*, spiegato nel dettaglio successivamente. È stata omessa la regolarizzazione, che tuttavia può essere facilmente aggiunta.

Aggiornamento Dinamico della Dimensione dei Mini-Batch. Come affermato in precedenza, l'algoritmo si basa sulla strategia di aggiornamento del mini-batch approfondita in *Sezione 3.3*. Seguendo il ragionamento della sezione, si definisce un'**epoca** t come la sequenza di iterazioni successive in cui si accede ad n campioni del training set $\mathcal{S}$, dove $n = |\mathcal{S}|$. Per ogni epoca t, si costruiscono $\left\lfloor \frac{n}{N_t} \right\rfloor$ **mini-batch** (*Step 3*) aventi dimensione fissata N_t

$$\left\{ \mathcal{N}_{\bar{t}}, \mathcal{N}_{\bar{t}+1}, \ldots, \mathcal{N}_{\bar{t}+\lfloor \frac{n}{N_t} \rfloor - 1} \right\},$$

dove $\bar{t}$ è il numero di iterazioni compiute prima dell'epoca t e corrisponde all'indice del suo primo mini-batch. A differenza di *DeepLISA*, i loro campioni non saranno estratti uniformemente, ma seguiranno per ogni iterazione la *distribuzione Adaptive Importance Sampling* (4.6) corrente: rispettivamente

$$\left\{ p^{\bar{t}}, p^{\bar{t}+1}, \ldots, p^{\bar{t}+\lfloor \frac{n}{N_t} \rfloor - 1} \right\}.$$

Per semplicità, nel caso in cui il training set $\mathcal{S}$ non sia perfettamente divisibile in mini-batch di dimensione N_t, si è deciso di eliminare l'ultimo sottoinsieme incompleto dell'epoca. Dunque, saranno costruiti $\left\lfloor \frac{n}{N_t} \right\rfloor$ mini-batch. Nell'implementazione standard di un Data Loader, questo corrisponderà all'impostazione

$$\textit{'drop\_last = True'}.$$

Coerentemente con la *Sezione 3.3*, la regola per l'**aggiornamento dinamico della dimensione del mini-batch del metodo DeepLISA-AIS** (*Step 1*) sarà

$$N_t := \min \left\{ n_{\max};\ \max \left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil ;\ N_0 \right\} \right\} \qquad \forall t \in \mathbb{N}. \tag{6.1}$$

dove N_0 e $n_{\max}$, vincolo di memoria a disposizione del calcolatore, sono rispettivamente il limite inferiore e superiore di N_t. Pur non essendo garantita la riduzione progressiva della varianza

$$\mathrm{Var}_{i \sim p^k} \left[g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) \right]$$

come nel caso di distribuzione uniforme, in cui si verifica la *disuguaglianza 3.11*, intuitivamente, l'aumento della quantità di informazioni utilizzate per il gradiente stocastico dovrebbe portare comunque ad un miglioramento dello stimatore. All'interno della *Sezione 6.3*, è stato verificato sperimentalmente sul dataset *CIFAR10* che ciò permette di ottenere prestazioni iniziali migliori.

Distribuzione AIS per la Costruzione dei Mini-Batch. Per ogni iterazione compiuta dall'algoritmo, il mini-batch $\mathcal{N}_k$ corrispondente sarà ottenuto attraverso il campionamento di $\{1, 2, \ldots, n\}$ secondo una particolare densità di probabilità che approssima la distribuzione Importance Sampling *(Step 3)*: **distribuzione Adaptive Importance Sampling (AIS)** p^k, aggiornata ad ogni passo seguendo la *Definizione 4.1.1 (Step 2)*.

Essa è definita come una combinazione convessa tra la distribuzione uniforme e la distribuzione basata sulla norma dei gradienti, divisi per campione. All'aumentare delle iterazioni compiute, quest'ultima assume un peso crescente grazie ad ξ_k, permettendo all'algoritmo di passare da una fase iniziale di esplorazione uniforme a una fase di campionamento mirata sugli esempi più informativi. Per questi ultimi, la massima pendenza di $f_i(\mathbf{w})$, misura dell'errore compiuto dal modello, è elevata. Dunque, si può sperare di apprendere maggiormente da questi campioni, in quanto piccole variazioni sui parametri potrebbero diminuire significativamente il costo $f_i(\mathbf{w})$ e, conseguentemente, il rischio empirico relativo a $\mathcal{S}$.

Affinché lo stimatore della direzione di discesa sia non distorto rispetto alla nuova distribuzione, il **gradiente stocastico** assume forma (4.4), che si ricordi essere

$$g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) \quad \textit{(Step 4)}.$$

Si osservi che la procedura di aggiornamento di p^k richiede la conoscenza a priori dell'indice *maxit* relativo all'ultima iterazione compiuta dall'algoritmo, ricordando che gli indici delle iterazioni sono numerati da 0 a *maxit*. Sebbene l'algoritmo sia scandito dalle epoche, *maxit* risulta deterministico e facilmente calcolabile grazie alla predeterminazione della sequenza $\{N_t\}_{t=1}^T$ delle dimensioni utilizzate per i mini-batch, vantaggio già dichiarato per il metodo DeepLISA. L'*Algoritmo 11* è sviluppato per calcolare *maxit*, simulando la costruzione dei mini-batch per ogni epoca e conteggiando il numero di iterazioni previste.

Line Search Procedure per il Calcolo del Learning Rate. Infine, per ogni iterazione k, la lunghezza di passo α_k è determinata attraverso la *procedura di Line Search con Backtracking di Armijo*, descritta nella *Sezione 4.2* per il caso in cui la direzione di discesa non corrisponde a $-\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$. Essa è presentata in *Algorithm*

5 ed è utilizzata all'interno dello *Step 5* dell'algoritmo DeepLISA-AIS (*Algorithm 12*).

Tuttavia, integrando la ricerca della lunghezza di passo all'algoritmo DeepLISA originale, si è scelto di inizializzare α_{k+1} per la procedura di Line Search (*Step 6*) secondo la formula

$$\alpha_{k+1}^{\text{initialization}} = \min\left\{\alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min}\right)\right\},$$

dove $\beta_2 \in (0,1)$ garantisce che $\alpha_{k+1}^{\text{initialization}} \geq \alpha_k$. Questa scelta permette di assecondare l'andamento decrescente del learning rate, tentando un incremento all'inizio di ogni iterazione, qualora la condizione backtracking di Armijo risulti soddisfatta.

Come osservato nei capitoli precedenti, la procedura Line Search adattiva conferisce **robustezza al modello** rispetto alle altre strategie per la scelta del learning rate. Infatti, questo approccio diminuisce drasticamente la dipendenza delle prestazioni dalla scelta dei relativi iperparametri e, conseguentemente, il tempo necessario per il fine-tuning.

Algorithm 11 Algoritmo per il Calcolo di *maxit*

Require: numero di epoche $T > 0$; $n = |\mathcal{S}|$; $N_0, n_{\max} \in \mathbb{N}$ t.c. $N_0 < n_{\max}$; $C > 0$; $\{\varepsilon_k\}_k$, successione sommabile, non crescente, positiva.

1: $\bar{t} := 0$ (indice dell'iterazione corrente per *Sezione 7.3*);
2: num_batches := 0, inizializzata per il conteggio del numero di batch costruiti;
3: **for** $t = 1, 2, \ldots, T$ **do**

Scelta della Dimensione dei Mini-Batch

4: $N_t = \min\left\{n_{\max}; \max\left\{\left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0\right\}\right\}$;

Aggiornamento del numero di Mini-Batch

5: num_batches = num_batches + $\left\lfloor \frac{n}{N_t} \right\rfloor$ ▷ equivalente a drop_last = True
6: **end for**
7: $maxit$ = num_batches − 1 ▷ indice dell'ultimo mini-batch
8: **return** $maxit$

Algorithm 12 Algoritmo DeepLISA-AIS

Require: $T > 0$; $\mathbf{w}^{(0)} \in \mathbb{R}^d$; $n = |\mathcal{S}|$; $N_0, n_{\max} \in \mathbb{N}$ t.c. $N_0 < n_{\max}$; sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$; $C > 0$; $\bar{t} = 0$; limiti del peso AIS $\{\xi_{\min}, \xi_{\max}\}$; α_0 t.c. $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$; parametri $\{\alpha_{\min}, \alpha_{\max}, \beta, \beta_2, \gamma\}$ con $\gamma \in \left(0, \frac{1}{\alpha_{\max}}\right)$, $\beta, \beta_2 \in (0,1)$.

Inizializzazione di $\mathbf{w}^{(0)}$, $\pi_i = 1$ per ogni $i \in \{1, \dots, n\}$

Calcolo del numero totale di iterazioni *maxit* (*Algorithm 11*);

for $t = 1, 2, \dots, T$ **do**

Step 1: Scelta della Dimensione dei Mini-Batch

Scelta di N_t come $N_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\}$;

for $k = \bar{t}, \dots, \bar{t} + \left\lfloor \frac{n}{N_t} \right\rfloor - 1$ **do** ▷ equivalente a drop_last = True

Step 2: Aggiornamento della Distribuzione AIS

$\xi_k = \xi_{\min} + \dfrac{k}{maxit}(\xi_{\max} - \xi_{\min})$;

for $z = 1, 2, \dots, n$ **do**

$p_z^k = \xi_k \dfrac{\pi_z}{\sum_{j=1}^n \pi_j} + (1 - \xi_k)\dfrac{1}{n}$;

end for

Step 3: Selezione del Mini-Batch e Aggiornamento di π

$\mathcal{N}_k = \emptyset$;

for $j = 1, 2, \dots, N_t$ **do**

Si seleziona casualmente $i_j \in \{1, \dots, n\}$ secondo la distribuzione p^k;

Si aggiunge i_j a $\mathcal{N}_k$;

Si aggiorna π: $\pi_{i_j} = \|\nabla f_{i_j}(\mathbf{w}^{(k)})\|$;

end for

Step 4: Calcolo del Gradiente Stocastico

Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e $g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \dfrac{1}{n \sum_{i\in\mathcal{N}_k} p_i^k} \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$

Step 5: Line Search Procedure per la Scelta del Learning Rate

while $f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) > f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} g_k$ **do**

$\alpha_k = \beta\alpha_k$

end while

Step 6: Aggiornamento per l'Iterazione Successiva

$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$; $\alpha_{k+1} = \min\left\{ \alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min} \right) \right\}$.

end for

Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.

end for

6.2 Implementazione PyTorch e Analisi degli Svantaggi Computazionali

L'implementazione di *DeepLISA-AIS* è stata realizzata in ambiente *PyTorch* [26], solitamente utilizzato nel contesto del Deep Learning. Il lavoro ha previsto l'estensione del codice originale dell'algoritmo dell'algoritmo DeepLISA (*Algorithm 4*), fornitomi dagli autori di [13]. La sua struttura si basa sulla coordinazione tra due componenti principali:

- la classe **Trainer** gestisce l'intero processo del metodo, alternando le fasi di addestramento (forward pass, backward pass e utilizzo dell'Optimizer, descritti in *Sezione 1.2.1*) e di test;
- la classe *Interpol*, chiamata **Optimizer**, che rappresenta il metodo SGD, integrando la procedura di *Line Search*, l'aggiornamento dei parametri e l'inizializzazione della lunghezza di passo per l'iterazione successiva (*Step 5, Step 6*).

6.2.1 Data Loader ed Inattività della GPU

La prima modifica fondamentale riguarda il campionamento dei mini-batch di dimensione N_t per ogni epoca t fissata *(Step 3)*. Siccome la distribuzione Adaptive Importance Sampling p^k viene modificata ad ogni iterazione e dipende dai risultati dei processi precedenti, cioè non è predeterminata, è necessario costruire un **Batch Sampler PyTorch dinamico**. A differenza del campionatore standard, esso non memorizza una versione statica della distribuzione di probabilità, ma crea $\left\lfloor \frac{n}{N_t} \right\rfloor$ mini-batch secondo la seguente procedura:

- **Passo 1.** si richiede la distribuzione AIS corrente, salvata all'interno di una variabile in Trainer;
- **Passo 2.** si estrae un nuovo mini-batch attraverso *Weighted Random Sampler PyTorch* con reinserimento, secondo i pesi dati dalla distribuzione AIS.

La prima operazione è facilmente ottenibile attraverso un funzione definita all'interno della classe *Trainer*, che restituisce la distribuzione AIS corrente:

```
def get_distribution_AIS(self):
 """
 Restituisce il tensore distribution_AIS per il Batch Sampler dinamico.
 """
 return self.distribution_AIS
```

Il Batch Sampler dinamico avrà in input la funzione *self.get_distribution_AIS*, che sarà richiamata ad ogni iterazione per ottenere sempre l'ultima versione della distribuzione AIS.

Il Batch Sampler definito in precedenza viene impiegato all'interno di un *Data Loader PyTorch* per la generazione dei mini-batch. Tuttavia, siccome la distribuzione AIS è strettamente dipendente dagli esiti delle iterazioni precedenti e soggetta a continui aggiornamenti, non è più possibile sfruttare la sua natura **multiprocessing**, impostando il parametro *'num_workers'* maggiore di 0. Infatti, attraverso questa modalità, la parallelizzazione sulla CPU preparerebbe i batch in anticipo, basandosi però su una versione della distribuzione non ancora aggiornata dallo *Step 3*. Dunque, è indispensabile impostare *'num_workers=0'*. In questo modo, le operazioni per la creazione di un batch avvengono solo nel momento dell'estrazione effettiva dal *DataLoader*, garantendo l'impiego della distribuzione AIS più recente. Tuttavia, questa scelta determina un rallentamento significativo del processo, poichè introduce una latenza nel caricamento dei dati che **aumenta il tempo di inattività della GPU**, la quale deve attendere il batch dalla CPU per compiere le operazioni richieste.

6.2.2 Gradienti per Campione, Saturazione della Memoria e Tempo Computazionale

Considerando il training set $\mathcal{S}$ avente cardinalità n, il fulcro dell'implementazione della distribuzione AIS consiste nella scelta della strategia di calcolo dei gradienti della funzione loss rispetto ai parametri del modello $\mathbf{w}^{(k)}$, distinguendo i contributi dei singoli campioni di $\mathcal{S}$. Questi sono fondamentali per l'aggiornamento del tensore

$$\pi = \Big[\|\nabla f_1(\mathbf{w}^{(k_1)})\|,\ \|\nabla f_2(\mathbf{w}^{(k_2)})\|, \ldots, \|\nabla f_n(\mathbf{w}^{(k_n)})\| \Big] \in \mathbb{R}^n$$

tale che $\|\nabla f_i(\mathbf{w}^{(k_i)})\|$ rappresenti la norma della funzione loss calcolata per il campione i-esimo di $\mathcal{S}$, aggiornata per l'ultima volta durante l'iterazione k_i, che può essere differente per componenti differenti *(Step 3)*. In particolare, per ogni mini-batch $\mathcal{N}_k$ estratto dal Data Loader saranno calcolati $\left\{\nabla f_i(\mathbf{w}^{(k)})\right\}_{i\in\mathcal{N}_k}$ e, successivamente, le loro norme, prima di procedere con la retropropagazione del gradiente (*backward pass*) e l'aggiornamento del vettore di parametri $\mathbf{w}^{(k)}$.

Tuttavia, la memorizzazione dei gradienti individuali rappresenta il principale **collo di bottiglia** del metodo in termini di **utilizzo di memoria**. Si consideri una qualsiasi iterazione k. Mentre in un addestramento standard dei metodi SGD la memoria GPU richiesta per i gradienti è costante e pari a d, dove d è il numero dei parametri del modello considerato (memorizzazione di $\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ attraverso il *backward pass*), per la versione AIS è necessaria la memorizzazione di un tensore di dimensione $|\mathcal{N}_k| \times d$, della forma

$$\mathbf{M}_{\text{gradients}} = \begin{pmatrix} \nabla f_{i_1}(\mathbf{w}^{(k)}) \\ \nabla f_{i_2}(\mathbf{w}^{(k)}) \\ \vdots \\ \nabla f_{i_{|\mathcal{N}_k|}}(\mathbf{w}^{(k)}) \end{pmatrix} \in \mathbb{R}^{|\mathcal{N}_k|\times d} \quad \text{dove} \quad \nabla f_i(\mathbf{w}^{(k)}) = \begin{pmatrix} \frac{\partial f_i(\mathbf{w}^{(k)})}{\partial w_1} \\ \frac{\partial f_i(\mathbf{w}^{(k)})}{\partial w_2} \\ \vdots \\ \frac{\partial f_i(\mathbf{w}^{(k)})}{\partial w_d} \end{pmatrix}^{\mathrm{T}} \in \mathbb{R}^d$$

e $\{i_1, i_2, \ldots, i_{|\mathcal{N}_k|}\}$ rappresentano gli indici che costituiscono il mini-batch $\mathcal{N}_k$, seguendo l'ordine di estrazione.

Si osservi che, a causa della crescita progressiva della dimensione dei mini-batch $|\mathcal{N}_k|$ imposta dalla legge (6.1), la necessità di spazio in memoria aumenta significativamente, proporzionalmente al valore elevato che d può assumere (ad esempio, nel caso di *ResNet18* si ha che $d = 11\,169\,162$).

Un'ulteriore problematica riscontrata riguarda l'**inefficienza computazionale del calcolo dei gradienti per campione** $\left\{\nabla f_i(\mathbf{w}^{(k)})\right\}_{i\in\mathcal{N}_k}$. Infatti, l'ambiente PyTorch non è progettato per calcolare il gradiente per campione della funzione loss: applicando la funzione *.backward()* sul tensore $\left[f_{i_1}(\mathbf{w}^{(k)}), f_{i_2}(\mathbf{w}^{(k)}), \ldots, f_{i_{|\mathcal{N}_k|}}(\mathbf{w}^{(k)})\right]$, sarà restituito esclusivamente $\sum_{i\in\mathcal{N}_k} \nabla_w f_i(\mathbf{w}^{(k)})$ per ogni parametro w, al fine di risparmiare memoria accumulando i gradienti durante la retropropagazione invece che memorizzare l'intera matrice inutilmente. Per ovviare a questo problema, PyTorch propone differenti approcci che svolgono quanto richiesto, questi sono stati sviluppati seguendo ciò che è riportato all'interno della documentazione [4,5], con opportune

modifiche che si adattano alla necessità. In particolare, sono state testate tre differenti librerie, al fine di scegliere quale di queste è più rapida per le operazioni richieste:

- la **funzione PyTorch *autograd.grad*** richiede che le operazioni siano svolte sequenzialmente su ogni campione $i \in \mathcal{N}_k$, calcolando singolarmente $f_i(\mathbf{w}^{(k)})$ ed utilizzando *autograd.grad* per ottenere il tensore $\nabla f_i(\mathbf{w}^{(k)})$. Nonostante questa strategia sia vantaggiosa in termini di memoria, in quanto permette di calcolare le norme individualmente senza dover allocare l'intera matrice $\mathbf{M}_{\text{gradients}}$, il suo tempo di esecuzione risulta proibitivo, poiché esclude la possibilità di parallelizzazione sulla GPU, ripetendo il procedimento $|\mathcal{N}_k|$ volte per ogni iterazione.

- la **libreria *functorch*** [17], ora integrata in *PyTorch* tramite il modulo *torch.func* [3], consente la parallelizzazione delle operazioni, compiendole contemporaneamente su tutti i dati di $\mathcal{N}_k$ tramite la vettorializzazione del gradiente della funzione loss. Dunque, $\nabla f_i(\mathbf{w}^{(k)})$ viene calcolato parallelamente per ogni campione i in $\mathcal{N}_k$ (input). Tuttavia, il limite principale dell'approccio è che l'intera matrice $\mathbf{M}_{\text{gradients}}$ deve essere memorizzata.

- la **libreria *Opacus*** [32] introduce una nuova strategia, chiamata *Fast Gradient Clipping [4]*, che permette di ricavare direttamente la norma $\|\nabla f_i(\mathbf{w}^{(k)})\|$ calcolando le norme al quadrato per ogni strato, i.e. $\|\nabla_w f_i(\mathbf{w}^{(k)})\|^2$, ed accumulandole [30]. Ciò evita la memorizzazione di $\mathbf{M}_{\text{gradients}}$ nella sua interezza, ma necessita comunque di spazio disponibile per la sua architettura. Inoltre, la libreria impone due rigidi obblighi: non tutti gli strati sono supportati; non devono essere presenti operazioni in-place per ottenere le predizioni attraverso il modello e per il calcolo della loss corrispondente.

Per scegliere quale approccio è migliore per *DeepLISA-AIS*, esso è stato implementato considerando le tre varianti ed è stato applicato per l'addestramento della *ResNet18* sul dataset CIFAR10. Le modifiche apportate a questa rete neurale sono riportate in *Sezione 5.2*.

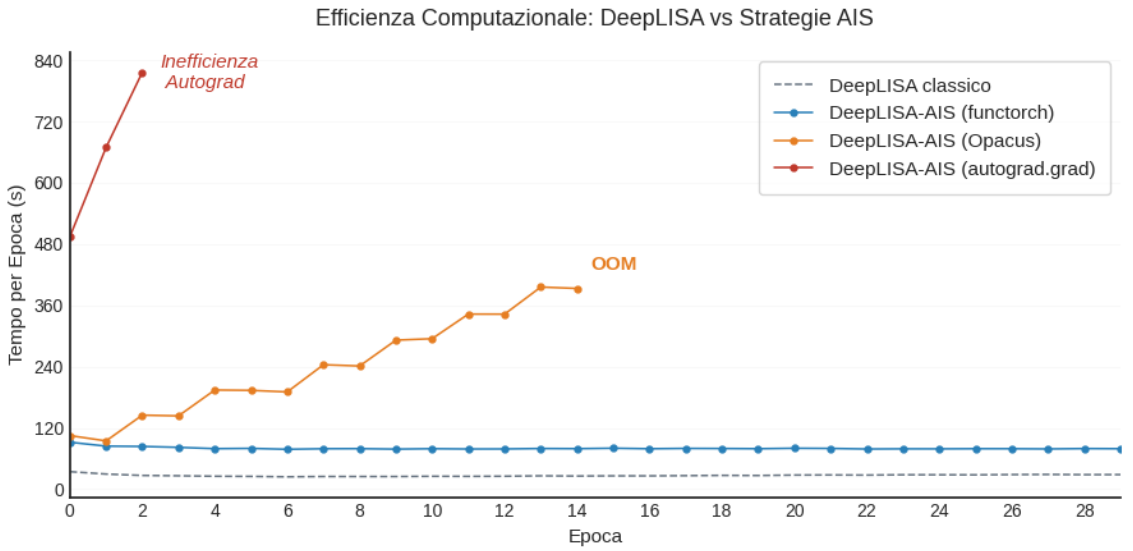


Figura 6.1: Tempo computazionale di DeepLISA-AIS per epoca, confrontando le strategie autograd.grad, functorch e Opacus per il calcolo dei gradienti per campione. Per completezza, è riportato anche il tempo impiegato da DeepLISA.

Si osservi che, in questo specifico caso, gli strati presenti sono supportati da *Opacus*. Tuttavia, è stato necessario modificare i suoi *Basic Blocks*[1] in modo tale che le operazioni degli strati non fossero in-place. Inoltre per ridurre il problema della limitatezza della memoria a disposizione (errore *Out of Memory*, *OOM*) per *functorch* e *Opacus* è stato necessario suddividere i mini-batch attraverso sottoinsiemi di 50 campioni[2], aumentando così il tempo dovuto alla parallelizzazione frammentata di $\mathcal{N}_k$. Questo permette di memorizzare solo una sottomatrice di $\mathbf{M}_{\text{gradients}}$ alla volta, dalla quale sono calcolate subito dopo le norme.

Dall'analisi sperimentale, riportata in *Figura 6.1*, si osserva come l'approccio basato su *functorch* risulti il più efficiente, mantenendo tempi per epoca ridotti e simili nonostante l'incremento progressivo di N_k, e dunque l'aumento di sottoinsiemi del mini-batch di 50 campioni. Al contrario, *autograd.grad* mostra una crescita dei tempi insostenibile. Infine, *Opacus* presenta un tempo di esecuzione in costante

[1] Un *Basic Block* è una classe PyTorch che compie le operazioni di un "blocco" di strati di una rete neurale, determinata dalla loro concatenazione. Essi compongono le Residual Networks e la loro implementazione, che subisce una modifica da in-place ad out-of-place, è consultabile in [6].

[2] Scelta vincolata dalle risorse hardware disponibili, in particolare dalla memoria della GPU, in un sistema dotato di CPU Intel i7-12700K (20 core), 32 GB di RAM e GPU NVIDIA GeForce RTX 3060 Ti.

crescita, probabilmente causato dalla frammentazione del mini-batch in un numero sempre maggiore sottogruppi. L'ipotesi è che il costo per un solo *passo di backward di Opacus* sia costante, indipendentemente dal numero di esempi presenti al suo interno, mentre il tempo di *functorch* potrebbe dipendere solo dal numero di campioni considerati, il quale è costante e pari a n per ogni epoca. Come affermato in [5], la singola operazione di *Opacus* dovrebbe essere più veloce rispetto a *functorch*. Dunque, il numero di ripetizioni della singola operazione è determinante. Inoltre, nonostante la suddivisione del mini-batch, la libreria *Opacus* non è stata in grado di concludere le epoche prefissate, arrestandosi alla quindicesima a causa dell'insufficiente spazio in memoria (errore *OOM*). Queste considerazioni giustificano la scelta di utilizzare *functorch* per l'implementazione.

Si osservi che, confrontando il tempo computazionale per *DeepLISA-AIS* con *functorch* e quello speso per *DeepLISA* (*Algorithm 4*), il primo impiega circa un minuto in più per ogni epoca (*Figura 6.1*).

Infine, si osservi che, per aggiornare le componenti $\|\nabla f_i(\mathbf{w}^{(k)})\|$ di π nelle posizioni corrette, ossia rispettando l'ordine di campionamento di $\mathcal{N}_k$, è stata implementata una personalizzazione della classe **Dataset PyTorch** in modo tale che sia restituita la terna *(tensore campioni, tensore dei target, tensore degli indici dei campioni)*.

6.2.3 Implementazione del Gradiente Stocastico

Un'ultima considerazione riguarda il calcolo del gradiente stocastico

$$g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}).$$

Così come per l'algoritmo *DeepLISA*, sarà sfruttata l'operazione *.backward()* su

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} f_i(\mathbf{w}^{(k)}),$$

che permette di calcolare

$$\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}).$$

Dunque, ogni componente di quest'ultimo dovrà essere moltiplicato per il fattore

$$\frac{|\mathcal{N}_k|}{n \sum_{i \in \mathcal{N}_k} p_i^k},$$

al fine di ricavare $g_{\mathcal{N}_k}(\mathbf{w}^{(k)})$.

6.3 Analisi delle Prestazioni del Metodo

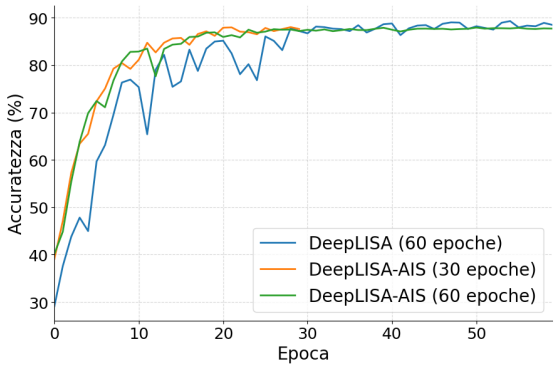
In questa sezione è riportata l'analisi delle prestazioni dell'algoritmo *DeepLISA-AIS*, con 30 e 60 epoche, a confronto con il metodo *DeepLISA*, considerando fino a 100 epoche. Anche in questo caso, il modello scelto per l'addestramento è la *Residual Network* di 18 strati, descritta in *Sezione 1.2.3*. Su di essa, sono apportate le modifiche dichiarate nella *Sezione 5.2*, per adattare il modello al problema di overfitting ed alla classificazione multiclasse del dataset *CIFAR10* (immagini $32 \times 32 \times 3$).

Gli esperimenti di questo capitolo sono stati eseguiti su un sistema avente CPU Intel i7-12700K (20 core), 32 GB di RAM e una GPU NVIDIA GeForce RTX 3060 Ti. I risultati sono riportati in *Figura 6.2*. Inoltre, la *Tabella 6.1* analizza le differenze temporali dei due algoritmi confrontati.

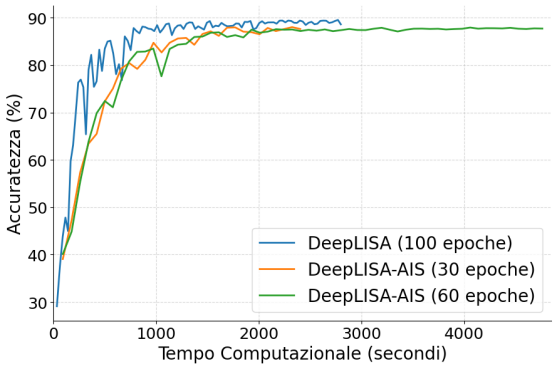
Epoche	**DeepLISA**	**DeepLISA-AIS**	**Differenza per Epoca**
30	838.74 s	2404.64 s	53.15 s
60	1693.83 s	4762.52 s	51.62 s
100	2799.64 s	-	-

Tabella 6.1: Confronto dei tempi computazionali (in secondi) di DeepLISA e DeepLISA-AIS, per 30, 60 *e* 100 *epoche. Siccome i tempi per epoca presentano circa la stessa distanza, in "Differenza per Epoca" sarà riportata la media della loro differenza durante il processo.*

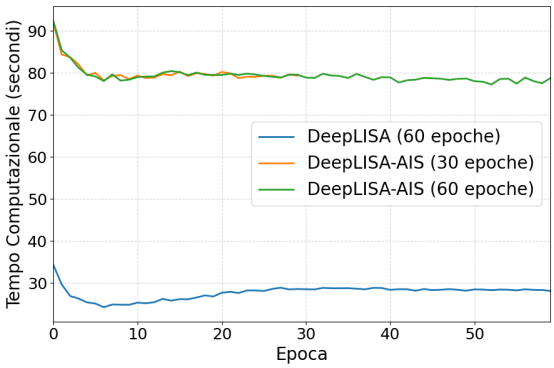
Accuratezza per Epoca. La *Figura 6.2a* permette di confrontare l'evoluzione dell'*accuratezza dei metodi rispetto al numero di iterazioni* compiute, per un massimo di 60 epoche totali. È evidente che, nelle fasi iniziali, la strategia *Adaptive Importance Sampling* apporta un notevole beneficio: la selezione mirata di esempi con maggiore contenuto informativo si rivela più efficace di un'estrazione casuale ed eterogenea dal grande insieme di addestramento. Tuttavia, al raggiungimento di livelli di accuratezza elevati, la gerarchia tende ad invertirsi: mentre l'algoritmo *DeepLISA-AIS* (60 epoche) stabilizza le proprie prestazioni entro il piccolo intervallo $[87.6\%, 87.9\%]$, *DeepLISA* riesce a superare tale soglia, raggiungendo fino all'89.0% di accuratezza nelle stesse epoche. Tale risultato suggerisce che, dopo un iniziale



(a) Accuratezza per epoca.



(b) Accuratezza per tempo computazionale.



(c) Tempo computazionale per epoca.

Figura 6.2: Confronto tra i metodi DeepLISA e DeepLISA-AIS rispetto ad accuratezza (%) e tempo computazionale (secondi). Risultati ottenuti sul dataset CIFAR10.

vantaggio nella scelta ragionata dei campioni, questi non sono più in grado di guidare verso un miglioramento. Al contrario, nel tempo, la scelta casuale si rivela migliore. Dunque, nasce l'idea dei cosiddetti *metodi ibridi*, descritti in *Sezione 7.3*, per usufruire dei benefici di entrambi i metodi.

Inoltre, l'*algoritmo DeepLISA-AIS* ottiene prestazioni iniziali migliori se utilizzato su meno iterazioni. Infatti, se ripetuto su 30 epoche, esso presenta tendenzialmente prestazioni migliori rispetto al medesimo sviluppato su 60 epoche. Inoltre, limitando la strategia AIS a 30 epoche, le sue accuratezze finali sono contenute nell'intervallo $[87.5, 88.0]$.

Stabilità Temporale dei Metodi. Osservando la *Figura 6.2a*, è possibile analizzare la *stabilità* dei metodi considerati. Durante l'intero processo, l'*algoritmo DeepLISA-AIS* presenta oscillazioni significativamente ridotte rispetto a *DeepLISA*. Dunque, in termini applicativi, esso è considerato più affidabile, in quanto, fissata un'epoca di arresto, si avrà un'alta probabilità di ottenere valide prestazioni, minimizzando il rischio di concludere il processo in un punto di minimo locale dell'accuratezza.

Questo comportamento è giustificato dallo scopo della *strategia AIS*: ridurre maggiormente la varianza dello stimatore rispetto all'utilizzo della distribuzione uniforme, caratteristica di DeepLISA, rendendolo dunque maggiormente simile alla massima direzione di discesa deterministica $-\nabla f(\mathbf{w}^{(k)})$.

Accuratezza per Tempo Computazionale. Purtroppo, dalla *Figura 6.2b*, si evince che il tempo computazionale riservato al calcolo dei gradienti per campione rende nullo il beneficio iniziale generato dalla *strategia AIS* nelle prime epoche. Tramite una scelta casuale, l'*algoritmo DeepLISA* raggiunge velocemente ciò che *DeepLISA-AIS* ottiene in un tempo elevato, seguendo una scelta ragionata.

Osservando la *Tabella 6.1*, data la rapidità del primo metodo, per ottenere un'analisi sullo stesso intervallo di tempo, sono state considerate 100 sue epoche, temporalmente simili a 30 epoche di *DeepLISA-AIS*. La possibilità di compiere un numero molto maggiore di iterazioni, permette a *DeepLISA* di stabilizzare le proprie prestazioni ad una soglia ancora più elevata, all'interno dell'intervallo $[88.6\%, 89.6\%]$, aumentando il gap con la soglia che *DeepLISA-AIS* non è in grado di oltrepassare.

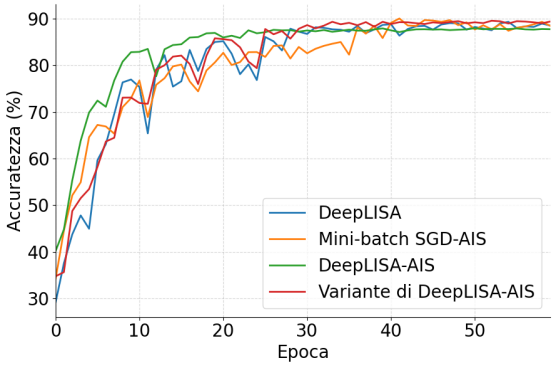


Figura 6.3: Confronto delle prestazioni ottenute attraverso Mini-Batch SGD-AIS, con dimensione del mini-batch pari a 32*, DeepLISA, DeepLISA-AIS e una delle sue varianti, descritta nella presentata in Sezione come Setting B.*

In effetti, dalla *Figura 6.2c* e dai risultati della *Tabella 6.1*, si nota che ad ogni epoca il calcolo dei *gradienti per campione* impiega circa 50 secondi, i quali, accumulandosi, costituiscono un problema non trascurabile. Dunque, all'interno del capitolo successivo saranno sviluppate alcune varianti con lo scopo di ridurre questo tempo computazionale.

Confronto delle Prestazioni con Mini-Batch SGD-AIS. Infine, si è deciso di confrontare le prestazioni del metodo *DeepLISA-AIS* con il caso in cui la dimensione del mini-batch è fissata, descritto nel *Capitolo 4*. Dalla *Figura 6.3*, si deduce che l'aumento progressivo della dimensione del mini-batch incentiva la *strategia AIS* ad ottenere risultati iniziali nettamente migliori. Al contrario, nelle ultime venti epoche utilizzare un mini-batch di dimensioni ridotte ha portato ad un'accuratezza paragonabile a quella di *DeepLISA*, che tuttavia può essere superata attraverso alcune delle varianti sviluppate nel *Capitolo 7*.

Impostazione degli Iperparametri del Modello. Gli iperparametri utilizzati per l'*algoritmo DeepLISA* deriva dal fine-tuning dell'articolo [13]. In particolare, essi sono: $C = 10$, $\varepsilon_k = 0.9995^k$, $\beta = 0.5$, $\beta_2 = \frac{1}{3}$, $\alpha_0 = \alpha_{\max} = 1$, $M = 1$, $N_0 = 32$ e $n_{\max} = 5000$.

Gli stessi iperparametri sono stati utilizzati per l'implementazione di *DeepLISA-AIS*, ad eccezione di γ. Infatti, esso è utilizzato all'interno della *regola di Backtrac-*

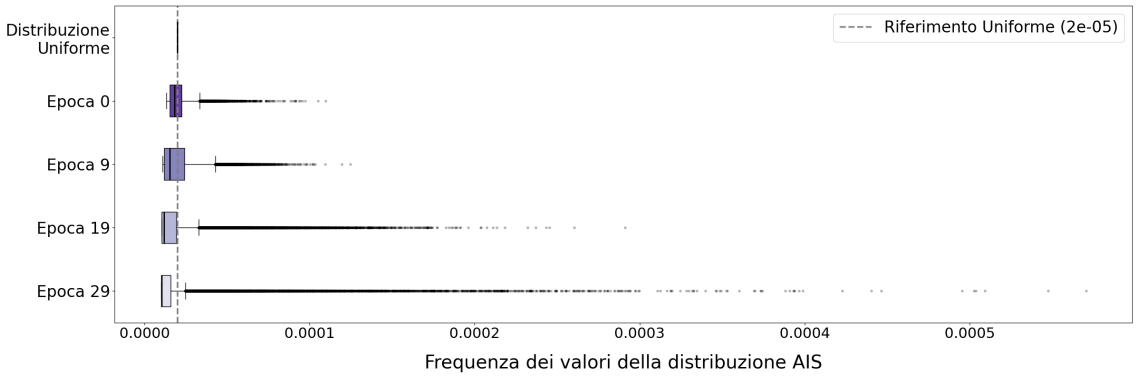


Figura 6.4: Box plot della distribuzione AIS: frequenza dei valori che la distribuzione assume durante il processo DeepLISA-AIS sul dataset CIFAR10.

king di Armijo

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) \leq f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} g_k,$$

dove viene moltiplicato per il fattore di g_k, i.e. $\dfrac{1}{n \sum_{i \in \mathcal{N}_k} p_i^k}$. Inoltre, ad esso si aggiungono due nuovi iperparametri: $\xi_{\min}$ e $\xi_{\max}$.

In particolare, considerando un *insieme di validazione* pari al 10% del training set, sono state confrontate le prestazioni del modello per differenti valori. Fissato $\xi_{\min} := 0.3$, gli iperparametri ottenuti dal processo di fine-tuning sono

$$\gamma := 0.15; \quad \xi_{\max} := \begin{cases} 0.5 & \text{se 30 epoche;} \\ 0.8 & \text{se 60 epoche.} \end{cases}$$

6.4 Analisi dell'Evoluzione del Metodo e della Distribuzione AIS

Dati i promettenti risultati ottenuti dall'*algoritmo DeepLISA-AIS*, è utile indagare sulla dinamica operativa della *strategia AIS*, la quale permette di ottenere un rapido miglioramento in poche iterazioni, attraverso una scelta ragionata dei campioni. Lo scopo di questa sezione è analizzare il comportamento della procedura rispetto al dataset *CIFAR10* nell'arco di trenta epoche di addestramento.

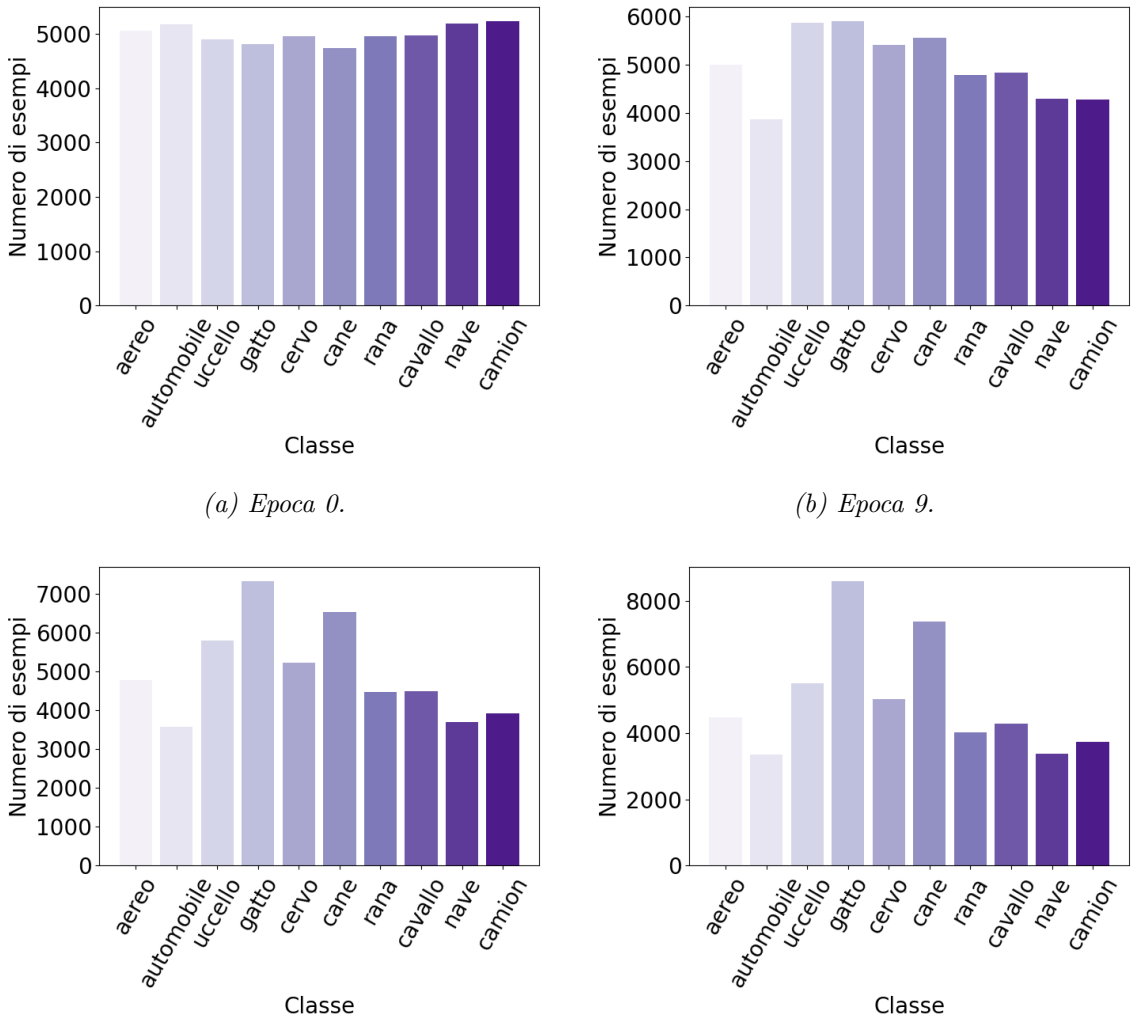
La *Figura 6.4* rappresenta l'andamento dei valori assunti dalla *distribuzione AIS* durante il processo. Essa mostra come la distribuzione di probabilità si discosti

progressivamente dalla distribuzione uniforme, costituita da valori pari a $2 \cdot 10^{-5}$. Con lo scorrere delle epoche si verificano due differenti fenomeni: oltre il 75% degli esempi ha probabilità di essere campionata sempre più piccola, discostandosi maggiormente dal valore uniforme; avviene un allungamento della "coda" destra del box plot. Di conseguenza, un numero ridotto di campioni (meno del 25%) acquisisce pesi sempre più grandi, aumentando la frequenza con cui questi vengono selezionati per la costruzione dei mini-batch. Probabilmente, questa dinamica è legata alla riduzione di un numero sempre più grande delle componenti di π, vettore delle norme dei gradienti della loss per campione, oltre all'aumento di importanza attribuito alla *distribuzione AIS* stessa rispetto a quella uniforme.

Per come la distribuzione è costruita, gli esempi più estratti saranno quelli in cui è possibile ottenere una maggiore decrescita dell'errore attraverso piccoli spostamenti dei parametri. Intuitivamente, esempi in cui si compie un errore minimo, potrebbero avere possibilità inferiore di decrescita locale, lasciando spazio principalmente al campionamento di dati su cui si sbaglia maggiormente.

Tale affermazione trova riscontro sperimentalmente. Innanzitutto, si osservi la frequenza delle classi estratte all'interno di alcune epoche (*Figura 6.5*). Confrontando questo andamento con lo sviluppo delle prestazioni del modello sull'insieme di addestramento, di cui sono riportati i risultati dopo la prima e l'ultima epoca (*Figura 6.6*), si nota come il campionamento insista sulle classi la cui accuratezza è effettivamente inferiore. In particolare, all'interno dell'insieme di addestramento, diventano sempre più presenti le classi *'gatto'*, *'cane'* e *'uccello'*: ad esempio, durante tutto il processo, il modello tende a scambiare spesso le classi *'gatto'* e *'cane'*, mentre la classe *'uccello'* è principalmente confusa come *'aereo'*. Viceversa, le classi meno selezionate sono quelle aventi accuratezza maggiore (*'automobile'*, *'nave'*, *'camion'*).

Si analizzi ora la distribuzione di campionamento ottenuta al termine delle trenta epoche, mostrata in parte in *Figura 6.7*. Considerando i 7500 esempi con probabilità più bassa di essere selezionati e la loro appartenenza alle classi (*Figura 6.8a*), è evidente la dominanza dei mezzi di trasporto (*'nave'*, *'automobile'* e *'camion'*), che coincidono con la classi maggiormente indovinate durante tutto il processo. In particolare, selezionando le 50 immagini con meno probabilità di essere scelte, tra le



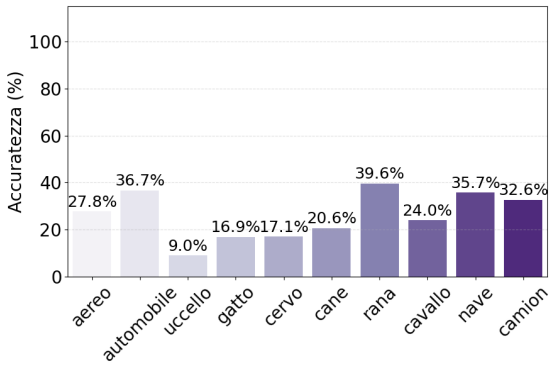
(a) Epoca 0.

(b) Epoca 9.

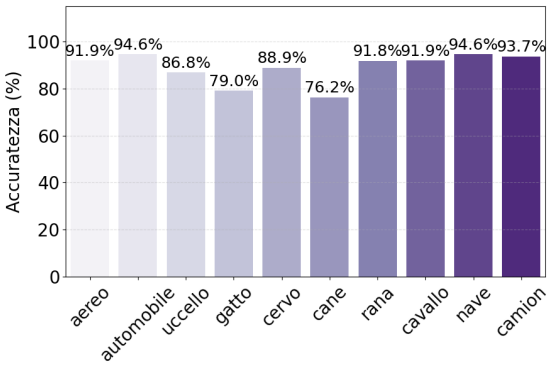
(c) Epoca 19.

(d) Epoca 29.

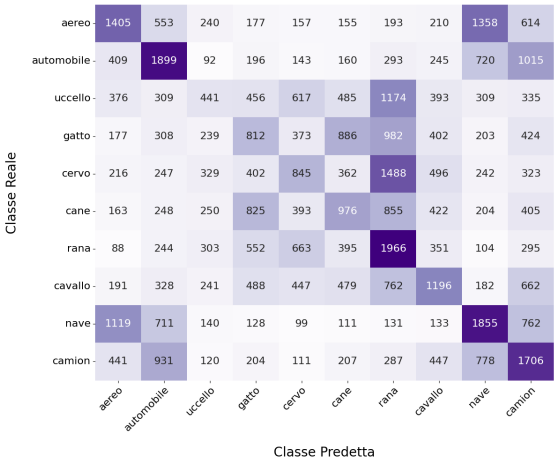
Figura 6.5: Evoluzione della frequenza di campionamento per classe durante l'addestramento su CIFAR10. Sono riportate le epoche 0, 9, 19 *e* 29.



(a) Accuratezza, epoca 0.



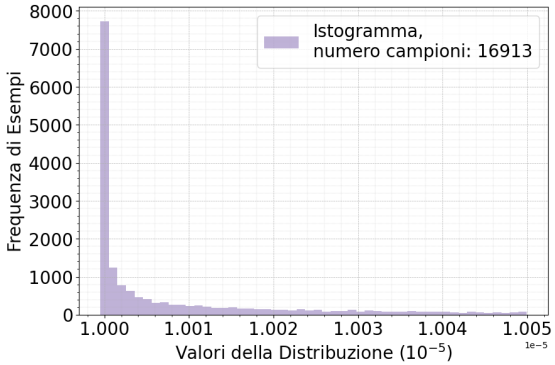
(b) Accuratezza, epoca 29.



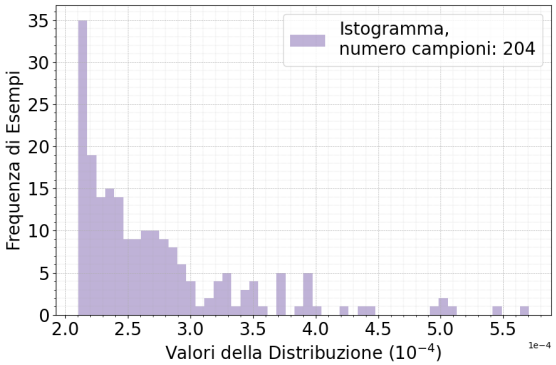
(c) Matrice di confusione, epoca 0.

(d) Matrice di confusione, epoca 29.

Figura 6.6: Accuratezza per classe e matrice di confusione risultante dalla classificazione dell'insieme di addestramento dopo la prima e l'ultima epoca, rispettivamente epoca 0 *ed epoca* 29.

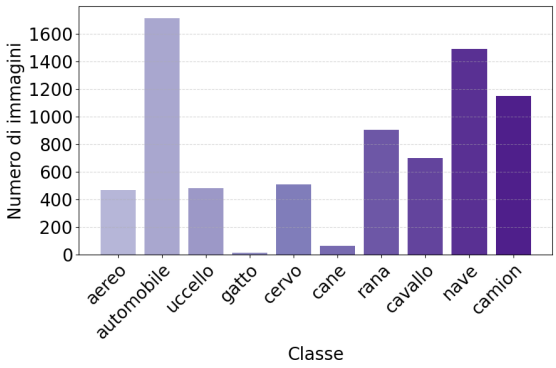


(a) Valori più bassi della distribuzione AIS.

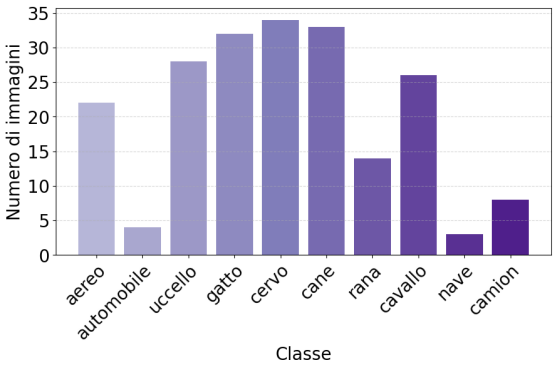


(b) Valori più alti della distribuzione AIS.

Figura 6.7: Istogramma della frequenza dei valori all'interno della distribuzione AIS al termine della trentesima epoca: (a) valori inferiori alla soglia $1.005 \cdot 10^{-5}$*; (b) valori superiori alla soglia* 0.00021*.*



(a) 7500 *esempi con probabilità inferiore.*



(b) 204 *esempi con probabilità superiore.*

Figura 6.8: Numero di campioni divisi per classe con valori più alti o più bassi della probabilità di campionamento al termine della trentesima epoca.

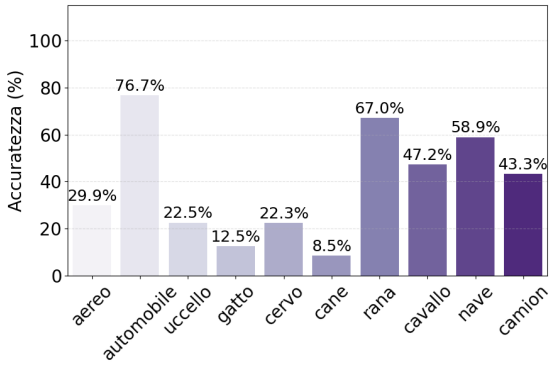
quali quelle riportate in *Figura 6.9*, si nota una prevalenza di veicoli e, in generale, soggetti tendenzialmente ben visibili e non disposti frontalmente.

Al contrario, si considerino le immagini appartenenti alla "coda" della distribuzione (*Figura 6.7b*). La *Figura 6.8* mostra una forte presenza di classi che sono maggiormente sbagliate dal modello. In *Figura 6.10*, sono riportati alcuni esempi rappresentativi: immagini ingrandite, con soggetti poco chiari o colori che si discostano dalla consuetudine (ad esempio, una nave in giardino, dove lo sfondo si discosta dai colori marini convenzionali).

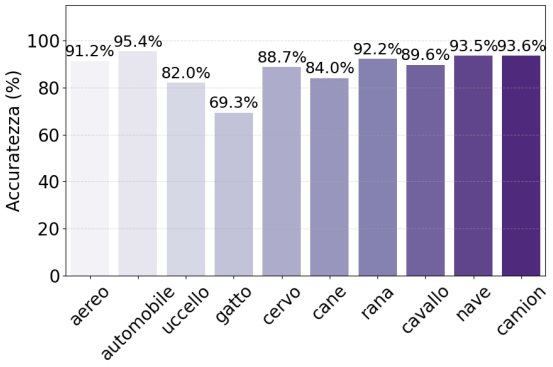
Figura 6.9: Esempi visivi di immagini CIFAR10 con probabilità di campionamento bassa.

Figura 6.10: Esempi visivi di immagini CIFAR10 con probabilità di campionamento alta.

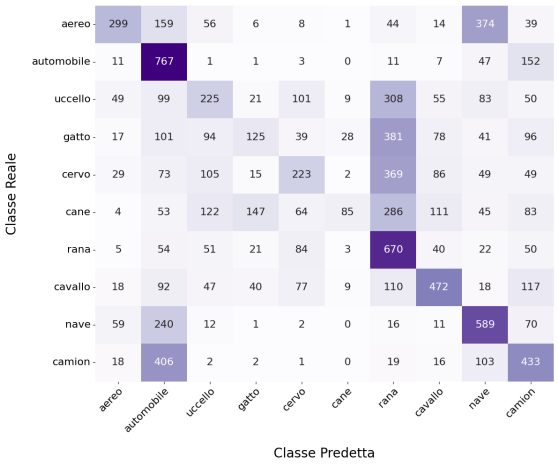
Per completezza, in *Figura 6.11* sono riportati i risultati della classificazione sul test set alla fine della prima e dell'ultima epoca. Questi presentano le stesse problematiche della classificazione dell'insieme di addestramento.



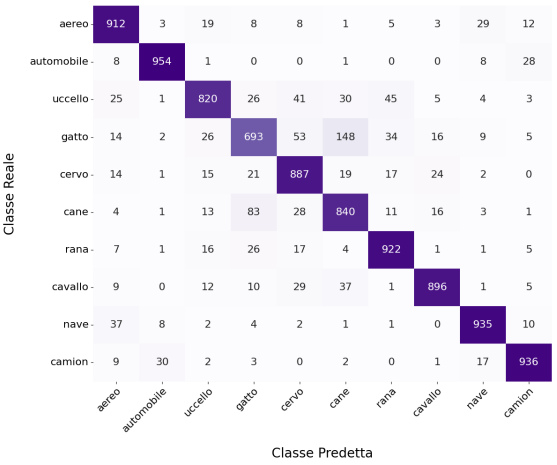
(a) Accuratezza, epoca 0.



(b) Accuratezza, epoca 29.



(c) Matrice di confusione, epoca 0.



(d) Matrice di confusione, epoca 29.

Figura 6.11: Accuratezza per classe e matrice di confusione risultante dalla classificazione dell'insieme di test dopo la prima e l'ultima epoca, rispettivamente epoca 0 *ed epoca* 29.

Capitolo 7

Varianti del Metodo DeepLISA-AIS

I risultati dell'*algoritmo DeepLISA-AIS* evidenziano due differenti problematiche nelle sue prestazioni: da una parte, la lentezza delle iterazioni rovina le prestazioni dell'algoritmo; dall'altra, l'accuratezza arresta la propria crescita ad una soglia inferiore rispetto all'algoritmo *DeepLISA*. In questo capitolo, sono riportate alcune varianti di *DeepLISA-AIS*, sviluppate nella speranza di superare, almeno in parte, tali limitazioni.

7.1 DeepLISA-AIS con Aggiornamento per Epoca della Distribuzione AIS

Innanzitutto, al fine di sfruttare la natura *multi-processing* della *classe DataLoader* di PyTorch, si è deciso di aggiornare la *distribuzione Adaptive Importance Sampling* p^t unicamente all'inizio di ogni epoca t. Nonostante non venga eliminato il calcolo dei gradienti per campione, operazione molto costosa, è possibile impostare il parametro *'num_workers'*= 2, per garantire un flusso continuo di dati all'interno della GPU e diminuire la sua inattività. Infatti, i mini-batch saranno creati parallelamente all'interno della CPU, senza attendere che questi siano richiesti dall'utente.

Lo pseudocodice che formalizza questo processo è mostrato in *Algorithm 13*: **DeepLISA-AIS con Aggiornamento per Epoca della Distribuzione AIS**. In tal caso, l'aggiornamento del peso ξ, utilizzato per il calcolo della *distribuzione AIS*, dipenderà dal numero di epoche compiute, una per ogni distribuzione differente nel processo. In particolare, all'interno dello *Step 1*, esso sarà calcolato come

$$\xi_t = \xi_{\min} + \frac{t}{T}(\xi_{\max} - \xi_{\min}) \quad \forall t \in \{1, 2, \dots T\}.$$

Algorithm 13 Variante DeepLISA-AIS: Aggiornamento di p^t per Epoca

Require: $T > 0$; $\mathbf{w}^{(0)} \in \mathbb{R}^d$; $n = |\mathcal{S}|$; $N_0, n_{\max} \in \mathbb{N}$ t.c. $N_0 < n_{\max}$; sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$; $C > 0$; $\bar{t} = 0$; limiti del peso AIS $\{\xi_{\min}, \xi_{\max}\}$; α_0 t.c. $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$; parametri $\{\alpha_{\min}, \alpha_{\max}, \beta, \beta_2, \gamma\}$ con $\gamma \in \left(0, \frac{1}{\alpha_{\max}}\right)$, $\beta, \beta_2 \in (0,1)$.

Inizializzazione di $\mathbf{w}^{(0)}$, $\pi_i = 1$ per ogni $i \in \{1, \dots, n\}$
for $t = 1, 2, \dots, T$ **do**
Step 1: Aggiornamento della Distribuzione AIS
$\xi_t = \xi_{\min} + \dfrac{t}{T}(\xi_{\max} - \xi_{\min})$;
for $z = 1, 2, \dots, n$ **do**
$p_z^t = \xi_t \dfrac{\pi_z}{\sum_{j=1}^n \pi_j} + (1 - \xi_t)\dfrac{1}{n}$;
end for
Step 2: Scelta della Dimensione dei Mini-Batch
Scelta di N_t come $N_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \dfrac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\}$;
for $k = \bar{t}, \dots, \bar{t} + \left\lfloor \frac{n}{N_t} \right\rfloor - 1$ **do** ▷ equivalente a drop_last = True
Step 3: Selezione del Mini-Batch e Aggiornamento di π
$\mathcal{N}_k = \emptyset$;
for $j = 1, 2, \dots, N_t$ **do**
Si seleziona casualmente $i_j \in \{1, \dots, n\}$ secondo la distribuzione p^t;
Si aggiunge i_j a $\mathcal{N}_k$;
Si aggiorna π: $\pi_{i_j} = \|\nabla f_{i_j}(\mathbf{w}^{(k)})\|$;
end for
Step 4: Calcolo del Gradiente Stocastico
Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e $g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \dfrac{1}{n \sum_{i\in\mathcal{N}_k} p_i^t} \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$
Step 5: Line Search Procedure per la Scelta del Learning Rate
while $f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) > f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} g_k$ **do**
$\alpha_k = \beta\alpha_k$
end while
Step 6: Aggiornamento per l'Iterazione Successiva
$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$; $\alpha_{k+1} = \min\left\{ \alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min} \right) \right\}$.
end for
Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.
end for

7.2 LISA-AIS: DeepLISA-AIS con Calcolo Puntuale del Batch Size

Seguendo il ragionamento sviluppato nel *Capitolo 5*, è possibile aggiornare la dimensione dei mini-batch all'inizio di ogni iterazione k, secondo la formula (5.3), che si ricordi essere

$$N_k := \min\left\{n_{\max}; \max\left\{\left\lceil\frac{C}{\varepsilon_k}\right\rceil; N_0\right\}\right\} \qquad \forall k \in \mathbb{N}_0, \tag{7.1}$$

dove $\{\varepsilon_k\}_k$ è la stessa sequenza sommabile, positiva e non crescente definita per l'*algoritmo DeepLISA*. *DeepLISA-AIS con aggiornamento puntuale del mini-batch size* sarà chiamato come **algotitmo LISA-AIS**, in quanto l'idea deriva dalla fusione dell'*algoritmo LISA* e della sua variante *DeepLISA*.

Non essendo vincolata da un vero e proprio concetto di epoca, così come nel *Capitolo 5*, si sceglie di testare il modello nelle stesse iterazioni di *DeepLISA-AIS*. Affinché questo sia possibile, si è deciso di introdurre un'ulteriore successione $\{n_t\}_t$ che, avendo la stessa legge di DeepLISA-AIS (6.1) per l'aggiornamento del mini-batch

$$n_t := \min\left\{n_{\max}; \max\left\{\left\lceil\frac{C}{\varepsilon_{\bar{t}+\left\lceil\frac{n}{n_{t-1}}\right\rceil}}\right\rceil; N_0\right\}\right\} \qquad \forall t \in \mathbb{N}, \tag{7.2}$$

scandisce le epoche considerate attraverso $\left\lfloor\frac{n}{n_t}\right\rfloor$ mini-batch per epoca.

Grazie alla formula (7.1), a parità di numero di iterazioni, *LISA-AIS* considererà meno esempi rispetto all'algoritmo originale, riducendo il tempo computazionale legato al lento calcolo dei gradienti per campione. Questa differenza sarà evidente soprattutto nelle prime fasi di addestramento, in cui il divario del numero di esempi sarà maggiore (*Figura 5.1*).

Algorithm 14 formalizza il metodo descritto in precedenza. Data la scelta di individuare le epoche attraverso la dimensione (7.2), a parità di epoche, il numero complessivo di iterazioni coincide con quello di *DeepLISA-AIS*. Dunque, anche in questo caso sarà considerato *Algorithm 11*.

Algorithm 14 Algoritmo LISA-AIS (calcolo puntuale del mini-batch size)

Require: $T > 0$; $\mathbf{w}^{(0)} \in \mathbb{R}^d$; $n = |\mathcal{S}|$; $N_0, n_{\max} \in \mathbb{N}$ t.c. $N_0 < n_{\max}$; sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$; $C > 0$; $\bar{t} = 0$; limiti del peso AIS $\{\xi_{\min}, \xi_{\max}\}$; α_0 t.c. $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$; parametri $\{\alpha_{\min}, \alpha_{\max}, \beta, \beta_2, \gamma\}$ con $\gamma \in \left(0, \frac{1}{\alpha_{\max}}\right)$, $\beta, \beta_2 \in (0,1)$.

Inizializzazione di $\mathbf{w}^{(0)}$, $\pi_i = 1$ per ogni $i \in \{1, \dots, n\}$

Calcolo del numero totale di iterazioni *maxit* (*Algorithm 11*);

for $t = 1, 2, \dots, T$ **do**

$$n_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{n_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\};$$

for $k = \bar{t}, \dots, \bar{t} + \left\lfloor \frac{n}{n_t} \right\rfloor - 1$ **do** ▷ equivalente a drop_last = True

Step 1: Scelta della Dimensione dei Mini-Batch

Scelta di N_k come $N_k = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_k} \right\rceil ; N_0 \right\} \right\}$;

Step 2: Aggiornamento della Distribuzione AIS

$\xi_k = \xi_{\min} + \dfrac{k}{maxit}(\xi_{\max} - \xi_{\min})$;

for $z = 1, 2, \dots, n$ **do**

$p_z^k = \xi_k \dfrac{\pi_z}{\sum_{j=1}^n \pi_j} + (1 - \xi_k)\dfrac{1}{n}$;

end for

Step 3: Selezione del Mini-Batch e Aggiornamento di π

$\mathcal{N}_k = \emptyset$;

for $j = 1, 2, \dots, N_k$ **do**

Si seleziona casualmente $i_j \in \{1, \dots, n\}$ secondo la distribuzione p^k;

Si aggiunge i_j a $\mathcal{N}_k$ e si aggiorna π: $\pi_{i_j} = \|\nabla f_{i_j}(\mathbf{w}^{(k)})\|$;

end for

Step 4: Calcolo del Gradiente Stocastico

Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e $g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \dfrac{1}{n \sum_{i\in\mathcal{N}_k} p_i^k} \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$

Step 5: Line Search Procedure per la Scelta del Learning Rate

while $f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) > f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} g_k$ **do**

$\alpha_k = \beta\alpha_k$

end while

Step 6: Aggiornamento per l'Iterazione Successiva

$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$; $\alpha_{k+1} = \min\left\{ \alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min} \right) \right\}$.

end for

Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.

end for

7.3 DeepLISA e DeepLISA-AIS: Alternanza dei Metodi

Un terzo tentativo può essere un metodo ibrido basato sull'alternanza di *DeepLISA* e della sua variante *DeepLISA-AIS*. Nonostante non segua la distribuzione di campionamento ottimale (*distribuzione AIS*), DeepLISA permette di abbattere sensibilmente i tempi di esecuzione, aumentando in poco tempo il numero di iterazioni compiute. Inoltre, si ipotizza un possibile miglioramento alternando fasi di esplorazione dei dati (campionamento uniforme) e fasi che si focalizzano sui campioni in grado di dare un maggior contributo alla discesa del rischio empirico (campionamento AIS): reinizializzazioni periodiche della distribuzione evitano che il metodo si concentri eccessivamente sugli stessi campioni. Il periodo esplorativo può avvenire sia attraverso *DeepLISA classico*, sia reinizializzando l'algoritmo DeepLISA-AIS, in cui il passaggio da campionamento uniforme a distribuzione ottimale diventa graduale.

7.3.1 Metodo Ibrido tra DeepLISA e DeepLISA-AIS

Algorithm 15 offre un esempio di implementazione di questa **strategia ibrida**: fissato il numero di epoche per ciascun metodo (T_{DL} e T_{AIS}), anche nullo se questo non viene utilizzato, è possibile alternarli, fino a che una certa condizione di arresto non è soddisfatta. Ad esempio, si può imporre una soglia T come limite superiore per il numero di epoche totali. Si osservi che i parametri fissati, tra cui $\{\alpha_{\min}, \alpha_{\max}, \beta, \beta_2, \gamma\}$, possono essere distinti per le due strategie.

7.3.2 Metodo Ibrido con Accumulo delle Norme dei Gradienti

Lo stesso l'*algoritmo DeepLISA* può essere fonte di informazioni sulla rilevanza dei campioni. Infatti, grazie all'operazione *.backward()* sul rischio empirico del mini-batch corrente, i.e.

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} f_i(\mathbf{w}^{(k)}),$$

azione già compiuta dall'algoritmo per calcolare il gradiente stocastico, è possibile ricavare la sua norma

$$\left\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\right\| = \frac{1}{|\mathcal{N}_k|}\left\| \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) \right\|,$$

attraverso uno sforzo computazionale aggiuntivo estremamente ridotto.

Sebbene i contributi dei singoli campioni ($\|\nabla f_i(\mathbf{w}^{(k)})\|$) non siano isolati, il termine $\left\| \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) \right\|$ permette di valutare complessivamente la rilevanza informativa del mini-batch, attraverso un tempo di esecuzione quasi nullo. Dividendo il risultato per $|\mathcal{N}_k|$, si ottiene un contributo approssimativo dei singoli dati in $\mathcal{N}_k$.

Queste informazioni possono essere accumulate durante l'esecuzione delle iterazioni di *DeepLISA* ed utilizzate come inizializzazione del tensore π, indicata con π^0. In tal modo, la distribuzione Adaptive Importance Sampling (4.6) generata alla prima iterazione di *DeepLISA-AIS*, non sarà uniforme, ma rifletterà lo storico dei gradienti osservati in *DeepLISA* opportunamente normalizzato, ossia $\frac{\pi^0}{\sum_{m=1}^{n}\pi_m^0}$. Al termine delle epoche dell'algoritmo, la componente j-esima di π^0 è ottenuta attraverso la formula

$$\pi_j^0 = \sum_{k\in\mathcal{I}_j} \left(\frac{1}{|\mathcal{N}_k|}\left\| \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)}) \right\| \right), \tag{7.3}$$

dove $\mathcal{I}_j$ rappresenta l'insieme degli indici delle iterazioni di *DeepLISA* in cui il campione j viene selezionato.

Questa strategia permette di iniziare la fase AIS avendo già a disposizione informazioni preliminari approssimative, che possono guidare la scelta dei campioni nei primi passi.

Il processo di aggiornamento di π segue quanto descritto successivamente.

Definizione 7.3.1 *Sia K il numero di iterazioni svolte dall'algoritmo DeepLISA, il tensore $\pi^0 = (\pi_1^0, \pi_2^0, \ldots, \pi_n^0)$, contenente lo storico delle norme dei gradienti ottenuti durante il processo, uno per ogni campione, è definito iterativamente in questo modo:*

- *$\tilde{\pi}^0$ è inizializzato come $(0, 0, \ldots, 0) \in \mathbb{R}^n$;*
- *ad ogni iterazione k di DeepLISA, con $k \in \{1, 2, \ldots K\}$,*

$$\tilde{\pi}_j^k = \begin{cases} \tilde{\pi}_j^{k-1} + \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\| & \text{se } j \in \mathcal{N}_k, \\ \tilde{\pi}_j^{k-1} & \text{altrimenti.} \end{cases} \qquad \forall j \in \{1, 2, \ldots, n\}$$

Al termine delle iterazioni, $\pi^0 := \tilde{\pi}^K$ *è tale che la sua componente* j*-esima, relativa al campione* j*, assume forma (7.3), per ogni* $j \in \{1, 2, \ldots, n\}$*.*

Si osservi che, rispetto alla versione ibrida descritta nella *Sezione 7.3.1*, π sarà inizializzato come tensore nullo. Infatti, ogni sua componente sarà aggiornata esattamente una volta per ogni epoca di *DeepLISA*, grazie alla scelta dei mini-batch come partizione dell'insieme di addestramento, assumendo così valori strettamente positivi. In particolare, per ogni iterazione, l'aggiornamento di π avverrà per ogni campione di $\mathcal{N}_k$.

Il processo di aggiornamento di π è mostrato nello *Step 2* di *Algorithm 16*. Questo, integrato all'interno di *Algorithm 15*, rappresenta una nuova variante di *DeepLISA-AIS*: **algoritmo ibrido tra DeepLISA (con Accumulo della Norma del Gradiente del Mini-Batch) e DeepLISA-AIS**.

Algorithm 15 Metodo Ibrido tra DeepLISA e DeepLISA-AIS

Require: $T > 0$; $\mathbf{w}^{(0)} \in \mathbb{R}^d$; $n = |\mathcal{S}|$; $N_0, n_{\max} \in \mathbb{N}$ t.c. $N_0 < n_{\max}$; sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$; $C > 0$; $\bar{t} = 0$; limiti del peso AIS $\{\xi_{\min}, \xi_{\max}\}$; α_0 t.c. $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$; parametri $\{\alpha_{\min}, \alpha_{\max}, \beta, \beta_2, \gamma\}$ con $\gamma \in \left(0, \frac{1}{\alpha_{\max}}\right)$, $\beta, \beta_2 \in (0,1)$, uno per ogni metodo; numero di epoche $T_{DL}, T_{AIS} \in \mathbb{N}$, risp. per DeepLISA e DeepLISA-AIS.

1: Inizializzazione di $\mathbf{w}^{(0)}$;

2: Inizializzazione di π tale che, per ogni $i \in \{1, 2, \ldots, n\}$,

$$\pi_i := \begin{cases} 1 & \text{per } \textit{Sezione 7.3.1} \text{ (distribuzione AIS è uniforme);} \\ 0 & \text{per } \textit{Sezione 7.3.2}. \end{cases}$$

3: Inizializzazione del numero di epoche T_{DL} e T_{AIS}; inizializzazione del numero di epoche totali T.

4: $num\_epochs := T_{DL} + T_{AIS}$;

5: **while** $num\_epochs \leq T$ **do**

6: **Algoritmo DeepLISA** per T_{DL} epoche, attraverso

- *Algorithm 4* per *Sezione 7.3.1*;
- *Algorithm 16* per *Sezione 7.3.2*;

7: **Algoritmo DeepLISA-AIS** per T_{AIS} epoche (*Algorithm 12*);

8: Reinizializzazione di π tale che, per ogni $i \in \{1, 2, \ldots, n\}$,

$$\pi_i := \begin{cases} 1 & \text{per } \textit{Sezione 7.3.1} \text{ (distribuzione AIS ritorna uniforme);} \\ 0 & \text{per } \textit{Sezione 7.3.2}. \end{cases}$$

9: Eventuale modifica di T_{DL} e T_{AIS};

10: Aggiornamento di num_epochs: $num\_epochs = num\_epochs + T_{DL} + T_{AIS}$.

11: **end while**

Algorithm 16 Algoritmo DeepLISA con Accumulo della Norma del Gradiente del Mini-Batch

Require: $T > 0$, $n_{\max} > 0$, $\mathbf{w}^{(0)} \in \mathbb{R}^d$, $0 < N_0 < n$, $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$, $\beta, \beta_2 \in (0,1)$, $\gamma > 0$, $M > 0$, una sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$, $C > 0$, $\bar{t} = 0$.

Inizializzazione di $\pi = (0, 0, \ldots, 0) \in \mathbb{R}^n$;
for $t = 1, 2, \ldots, T$ **do**
Step 1: Scelta della Dimensione e Creazione dei Mini-Batch Size
Scelta di N_t come

$$N_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\};$$

Partizione del training set $\mathcal{S}$ in $\left\lceil \frac{n}{N_t} \right\rceil$ mini-batch di dimensione N_t;
for $k = \bar{t}, \ldots, \bar{t} + \left\lceil \frac{n}{N_t} \right\rceil - 1$ **do**
Si sceglie il mini-batch $\mathcal{N}_k$ di cardinalità N_t.

Step 2: Calcolo del Gradiente Stocastico e Aggiornamento di π
Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e il gradiente stocastico $\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;
Si calcola $\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|$;
for $j \in \mathcal{N}_k$ **do**
$\pi_j = \pi_j + \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|$ (aggiornamento di π);
end for

Step 3: Line Search Procedure per la Scelta del Learning Rate
Si calcola $\bar{\mathbf{w}} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;
while $f_{\mathcal{N}_k}(\bar{\mathbf{w}}) > \max_{0\le j\le \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) - \gamma\alpha_k \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2$ **do**
$\alpha_k = \beta\alpha_k$;
$\bar{\mathbf{w}} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$;
end while
Step 4: Aggiornamento per l'Iterazione Successiva
Si pongono $\mathbf{w}^{(k+1)} = \bar{\mathbf{w}}$; $\alpha_{k+1} = \min\left\{\alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min}\right)\right\}$.
end for
Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.
end for

7.4 DeepLISA con Accumulo delle Norme dei Gradienti per AIS

Grazie alla strategia descritta nella *Sezione 7.3.2*, si osserva un leggero miglioramento nelle prime epoche in cui si applica la strategia AIS. Questo suggerisce l'idea di considerare le norme dei gradienti di $f_{\mathcal{N}_k}$ proprio come base per la distribuzione di campionamento. Sebbene questa non sia equivalente alla distribuzione AIS ottimale, lo storico delle norme di $\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|$ può fornire comunque informazioni significative per individuare i campioni in grado di ridurre maggiormente il rischio empirico. Il vantaggio di questa strategia è la riduzione significativa dei tempi computazionali, eliminando il calcolo delle norme dei gradienti per campione e, così, superando la principale limitazione che penalizza *DeepLISA-AIS*.

A differenza di quanto scritto nella *Sezione 7.3.2*, in tal caso viene introdotto un ulteriore iperparametro $\bar{\beta} \in (0, 1]$, che permette di modellare la perdita di rilevanza delle informazioni memorizzate in passato. Un valore di $\bar{\beta}$ molto piccolo determina una rapida "dimenticanza" dei valori accumulati. Al contrario, quando $\bar{\beta}$ è prossimo a 1, le informazioni sono maggiormente preservate.

In particolare, il tensore π^k, contenente lo storico dei gradienti accumulati fino all'iterazione k, è definito come

$$\pi_j^k = \sum_{i=0}^{m} \left(\bar{\beta}^{m-i} \frac{1}{|\mathcal{N}_{k_i}|} \Big\| \sum_{z \in \mathcal{N}_{k_i}} \nabla f_z(\mathbf{w}^{(k)}) \Big\| \right), \tag{7.4}$$

dove $\{k_0, k_1, \ldots, k_m\}$ è l'insieme ordinato degli indici delle iterazioni precedenti a k in cui il campione j viene selezionato per il mini-batch.

Il processo di aggiornamento di π segue quanto descritto successivamente.

Definizione 7.4.1 *Sia $maxit + 1$ il numero di iterazioni svolte dall'algoritmo, il tensore $\pi^k = (\pi_1^k, \pi_2^k, \ldots, \pi_n^k)$ memorizza lo* ***storico delle norme dei gradienti*** *ottenuti durante il processo, dove la sua componente j-esima è relativa al campione j dell'insieme di addestramento. Esso è definito iterativamente in questo modo:*

- *π^0 è inizializzato come $(1, 1, \ldots, 1) \in \mathbb{R}^n$;*
- *ad ogni iterazione k, con $k \in \{1, 2, \ldots maxit\}$,*

$$\pi_j^k = \begin{cases} \bar{\beta}\,\pi_j^{k-1} + \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\| & \textit{se } j \in \mathcal{N}_k, \\ \pi_j^{k-1} & \textit{altrimenti.} \end{cases} \qquad \forall j \in \{1, 2, \ldots, n\}$$

dove $\bar{\beta} \in (0, 1]$ è un fattore che diminuisce l'importanza data alle norme accumulate in passato.

*La **distribuzione di campionamento** p^k sarà calcolata come nel caso Adaptive Importance Sampling (4.6).*

Si osservi che, rispetto alla versione ibrida descritta nella *Sezione 7.3.2*, ogni componente di π sarà inizializzata come 1. Questo è necessario per evitare errori durante i calcoli nelle prime iterazioni: le componenti di π non sono necessariamente diverse da 0, causando errori nel calcolo di $\frac{\pi}{\sum_{i=1}^{|\mathcal{N}_k|}\pi_i}$, il cui denominatore potrebbe essere pari a 0. Tuttavia, grazie all'iperparametro parametro $\bar{\beta}$, questo non causa alcuna alterazione di π a lungo termine.

L'implementazione del metodo è formalizzata in *Algorithm 17* (*Step 4*). Così come *Algorithm 16*, sarà sfruttata l'operazione PyTorch *.backward()*, già presente all'interno del processo SGD.

Algorithm 17 Variante di DeepLISA-AIS, Accumulo delle Norme dei Gradienti

Require: $T > 0$; $\mathbf{w}^{(0)} \in \mathbb{R}^d$; $n = |\mathcal{S}|$; $N_0, n_{\max} \in \mathbb{N}$ t.c. $N_0 < n_{\max}$; sequenza positiva e sommabile $\{\varepsilon_k\}_{k\in\mathbb{N}}$; $C > 0$; $\bar{t} = 0$; limiti del peso AIS $\{\xi_{\min}, \xi_{\max}\}$; α_0 t.c. $0 < \alpha_{\min} < \alpha_0 < \alpha_{\max}$; parametri $\{\alpha_{\min}, \alpha_{\max}, \beta, \beta_2, \gamma\}$ con $\gamma \in \left(0, \frac{1}{\alpha_{\max}}\right)$, $\beta, \beta_2 \in (0,1)$; $\bar{\beta} \in (0,1]$.

Inizializzazione di $\mathbf{w}^{(0)}$, $\pi_i = 1$ per ogni $i \in \{1, \dots, n\}$

Calcolo del numero totale di iterazioni *maxit* (*Algorithm 11*);

for $t = 1, 2, \dots, T$ **do**

Step 1: Scelta della Dimensione dei Mini-Batch

Scelta di N_t come $N_t = \min\left\{ n_{\max}; \max\left\{ \left\lceil \frac{C}{\varepsilon_{\bar{t}+\left\lceil \frac{n}{N_{t-1}} \right\rceil}} \right\rceil \; N_0 \right\} \right\}$;

for $k = \bar{t}, \dots, \bar{t} + \left\lfloor \frac{n}{N_t} \right\rfloor - 1$ **do** ▷ equivalente a drop_last = True

Step 2: Aggiornamento della Distribuzione AIS

$\xi_k = \xi_{\min} + \dfrac{k}{maxit}(\xi_{\max} - \xi_{\min})$;

$p_z^k = \xi_k \dfrac{\pi_z}{\sum_{j=1}^n \pi_j} + (1 - \xi_k)\dfrac{1}{n}$ per ogni $z \in \{1, 2, \dots, n\}$;

Step 3: Selezione del Mini-Batch e Aggiornamento di π

$\mathcal{N}_k$ = estrazione di N_t campioni secondo la distribuzione p^k;

Step 4: Calcolo del Gradiente Stocastico e Aggiornamento di π

Si calcola $f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$ e $g_{\mathcal{N}_k}(\mathbf{w}^{(k)}) = \dfrac{1}{n \sum_{i\in\mathcal{N}_k} p_i^k} \sum_{i\in\mathcal{N}_k} \nabla f_i(\mathbf{w}^{(k)})$

Si calcola $\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|$;

for $j \in \mathcal{N}_k$ **do**

$\pi_j = \bar{\beta}\,\pi_j + \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|$ (aggiornamento di π);

end for

Step 5: Line Search Procedure per la Scelta del Learning Rate

while $f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) > f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}} g_k$ **do**

$\alpha_k = \beta\alpha_k$

end while

Step 6: Aggiornamento per l'Iterazione Successiva

Poni $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$; $\alpha_{k+1} = \min\left\{\alpha_{\max}, \max\left(\frac{\alpha_k}{\beta_2}, \alpha_{\min}\right)\right\}$.

end for

Aggiornamento del numero di iterazioni totali $\bar{t} = k + 1$.

end for

7.5 Analisi delle Prestazioni delle Varianti di DeepLISA-AIS

In questa sezione sono riportate le analisi delle prestazioni delle varianti di *DeepLISA* sviluppate in precedenza. I risultati derivano dal problema di classificazione multiclasse del dataset *CIFAR10* [20] attraverso la *Residual Neural Network* descritta nella *Sezione 1.2.3*. Questa è riadattata al problema attraverso le modifiche riportate un *Sezione 5.2*.

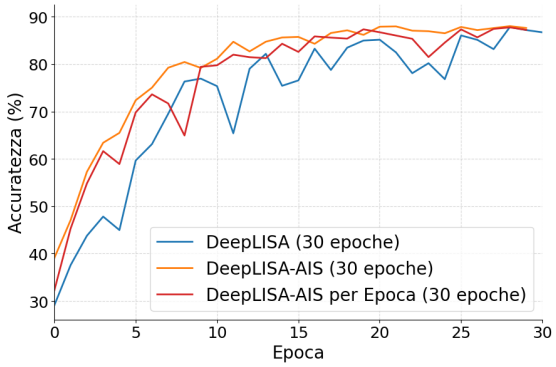
Per queste varianti, sono stati utilizzati gli stessi iperparametri indicati nell'ultimo paragrafo di *Sezione 6.3*. Inoltre, nel caso in cui siano utilizzate 10 epoche (risp. 100 epoche) sarà considerato $\xi_{\max} := 0.5$ (risp. $\xi_{\max} := 0.8$). Infine, per la variante descritta nella *Sezione 7.4*, si considererà $\bar{\beta} := 0.3$, ottenuto attraverso il processo di fine-tuning.

7.5.1 LISA-AIS ed Aggiornamento per Epoca della Distribuzione

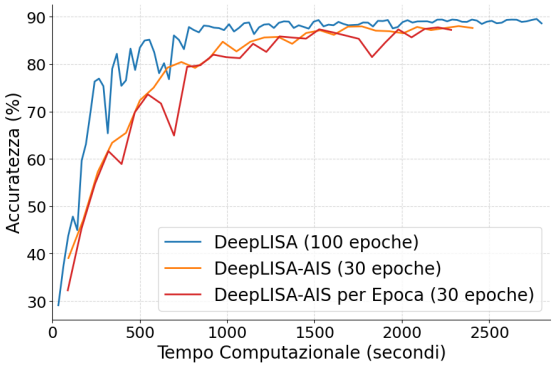
Si considerino le varianti sviluppate nelle *Sezioni 7.1* e *7.2*. Esse, pur nascendo da due approcci distinti, sono accomunate dallo stesso obiettivo: superare i limiti computazionali di *DeepLISA-AIS*, cercando di allineare i tempi di esecuzione a quelli di *DeepLISA* senza compromettere le prestazioni.

Da un lato, l'aggiornamento della *distribuzione AIS* all'inizio di ogni epoca consente di sfruttare la *natura multi-processing* della classe *Data Loader PyTorch*, attivata impostando il parametro *'num_workers'* a 2; dall'altro, la selezione di un numero ridotto di indici permette di limitare il tempo richiesto dal calcolo dei gradienti per campione.

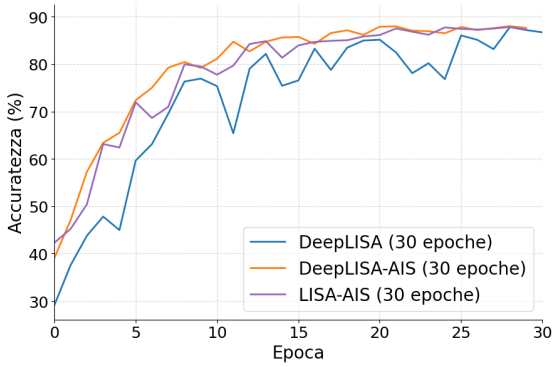
I risultati ottenuti dalla classificazione di *CIFAR10* sono riportati in *Figura 7.1*, in cui si analizza l'accuratezza sul test set rispetto al numero di iterazioni e al tempo computazionale dei processi *DeepLISA*, *DeepLISA-AIS* e delle sue varianti. Inoltre, la *Tabella 7.1* mostra il risparmio temporale corrispondente. Mentre per gli algoritmi con *strategia AIS* sono considerate 30 epoche, *DeepLISA* sarà ripetuto per 100 epoche, poiché compie le iterazioni più velocemente.



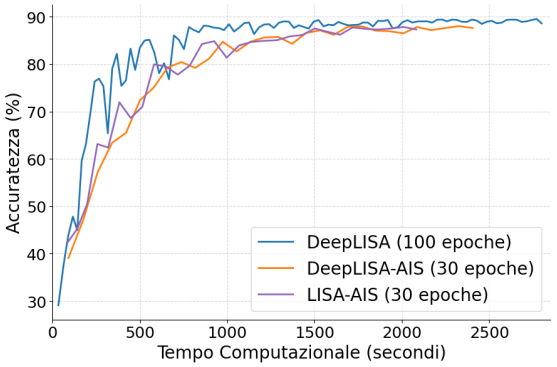
(a) Aggiornamento di p^k per epoca: accuratezza per epoca.



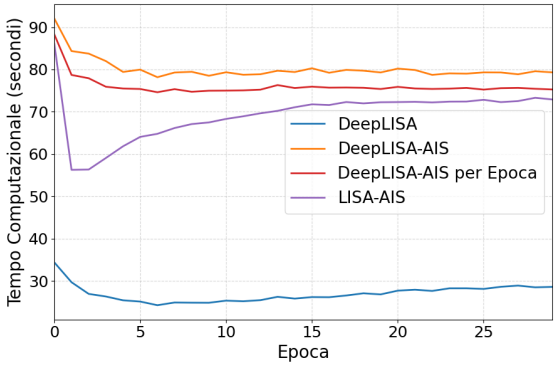
(b) Aggiornamento di p^k per epoca: accuratezza per tempo computazionale.



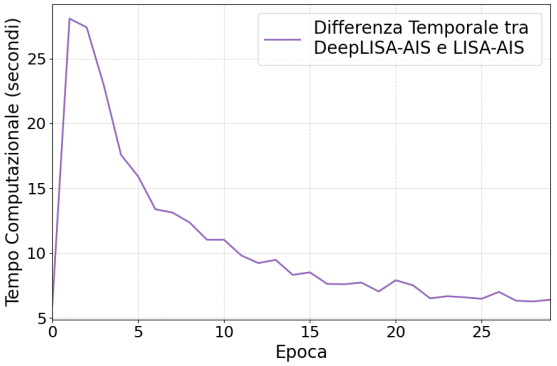
(c) LISA-AIS: accuratezza per epoca.



(d) LISA-AIS: accuratezza per tempo.



(e) Tempo computazionale per epoca.



(f) LISA-AIS: differenza di tempo per epoca.

Figura 7.1: Confronto tra i metodi DeepLISA, DeepLISA-AIS e le sue varianti con aggiornamento di p^k per epoca e con aggiornamento puntuale della dimensione del mini-batch, rispetto ad accuratezza (%) e tempo computazionale (secondi). Risultati ottenuti sul dataset CIFAR10.

	DeepLISA-AIS	**DeepLISA-AIS per Epoca**	**LISA-AIS**
Tempo totale	2404.64 s	2282.08 s	2082.70 s
Risparmio	-	4.09 s per epoca	321.94 s totali

Tabella 7.1: Confronto dei tempi computazionali (in secondi) per 30 *epoche di DeepLISA-AIS e delle sua varianti con aggiornamento per epoca della distribuzione e con aggiornamento puntuale della dimensione del mini-batch. Siccome nel primo caso i tempi per epoca presentano circa la stessa distanza, in "Risparmio per Epoca" sarà riportata la media della loro differenza durante il processo.*

Accuratezza per Epoche ed Instabilità. Innanzitutto, dalla *Figura 7.1a*, si osserva che aggiornare la distribuzione solo una volta per epoca non altera significativamente le prestazioni del modello, le quali subiscono solo un lieve peggioramento. Tuttavia, le oscillazioni dell'accuratezza aumentano, compromettendo la stabilità temporale che caratterizza l'*algoritmo DeepLISA-AIS*. Questo è dovuto al fatto che la varianza dello stimatore non è effettivamente ridotta attraverso la scelta della distribuzione AIS, quindi è meno probabile che la direzione ottenuta si avvicini a quella ottimale $-\nabla f(\mathbf{w}^{(k)})$.

Un comportamento analogo si osserva anche aggiornando puntualmente la dimensione del mini-batch, probabilmente riconducibile all'impiego di una minor quantità di informazioni per ogni iterazione (*Figura 7.1c*).

Nonostante ciò, considerando il numero di passi compiuti, queste varianti rimangono più stabili e con efficacia predittiva migliore rispetto all'*algoritmo DeepLISA*, caratterizzato da una scelta casuale degli esempi.

Accuratezza per Tempo Computazionale. Da *Figura 7.1e* e *Tabella 7.1*, si evince che la parallelizzazione delle operazioni per la creazione dei mini-batch permette di ridurre il tempo computazionale di circa 4.09 secondi per ogni epoca, attenuando la durata complessiva del processo di due minuti. Tuttavia, questo lieve miglioramento, unito all'aumento dell'instabilità dei risultati, si rivela insufficiente per raggiungere le prestazioni di *DeepLISA* o almeno migliorare quelle di *DeepLISA-AIS* (*Figura 7.1b*). In effetti, il distacco del tempo computazionale tra *DeepLISA* e questa variante rimane elevato: circa 49.06 secondi per singola epoca.

Al contrario, come evidenziato nelle *Figure 7.1d - 7.1e*, la velocità del processo *LISA-AIS* nelle prime epoche di addestramento permette di ottenere accuratezze superiori all'algoritmo originale, risparmiando complessivamente 321.94 secondi.

7.5.2 Metodi Ibridi: Alternanza di DeepLISA e DeepLISA-AIS

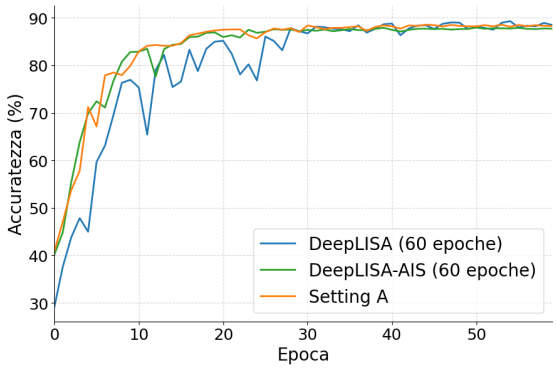
Un'ulteriore idea è quella di alternare fasi di esplorazione tramite distribuzione uniforme e fasi in cui si segue la scelta adattiva, al fine di non focalizzarsi eccessivamente sugli stessi esempi difficili. Seguendo quanto descritto nella *Sezione 7.3*, sono state testate differenti strategie, tra le quali si distinguono per i loro risultati, riportati in *Figura 7.2*:

- **Setting A.** 10 epoche di *DeepLISA-AIS*, ripetute per un totale di 60 epoche;
- **Setting B.** 30 epoche di *DeepLISA*, seguite da tre ripetizioni di 10 epoche con la *strategia AIS*, per un totale di 60 epoche;
- **Setting C.** alternanza dei due metodi, 10 epoche ciascuno;

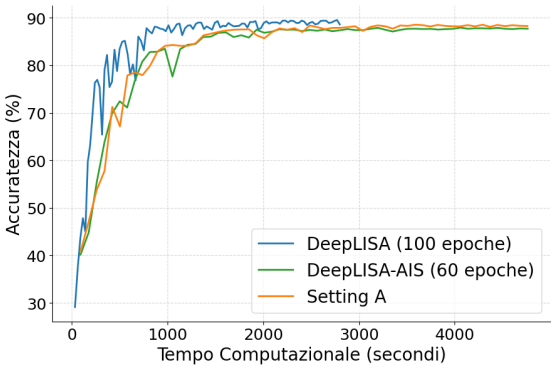
Ulteriori strategie implementate, che tuttavia non hanno portato a risultati significativi, sono:

- due ripetizioni di 30 epoche di *DeepLISA-AIS*;
- alternanza dei due metodi, 30 epoche ognuno;
- 10 epoche di *DeepLISA* (con e senza accumulo delle norme), seguite da 30 epoche di *DeepLISA-AIS*, dove si osserva che la scelta di inizializzare π come descritto nella *Sezione 7.3.2* permette di ottenere un visibile miglioramento iniziale della *strategia AIS*.

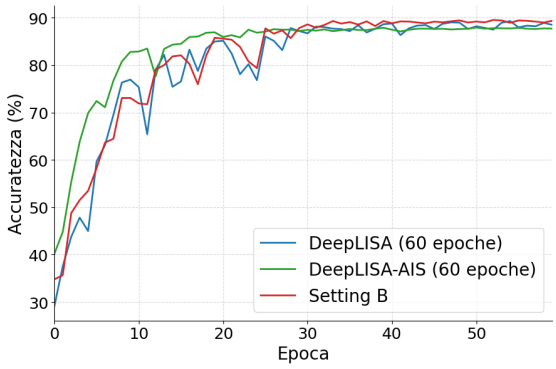
Si analizzino le *Figure 7.2a - 7.2b* relative al modello ottenuto attraverso il **Setting A**. Le sue prestazioni, pur essendo molto simili al caso di *DeepLISA-AIS* su 60 epoche, verificano un lieve incremento alla soglia dell'accuratezza raggiunta dalle ultime epoche, compresa tra 88.0% e 88.5%. Dunque, reinizializzare il processo con piccole fasi di esplorazione (distribuzione uniforme all'inizio di ogni processo) risulta essere vantaggioso.



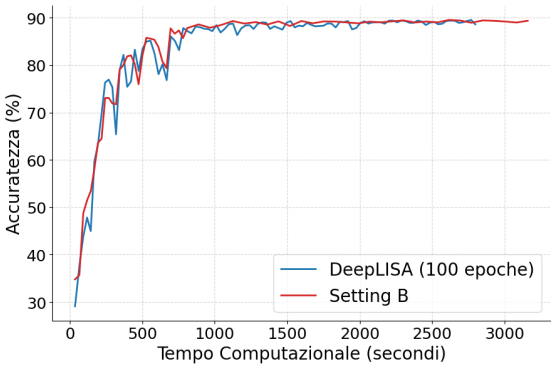
(a) Setting A: accuratezza per epoca.



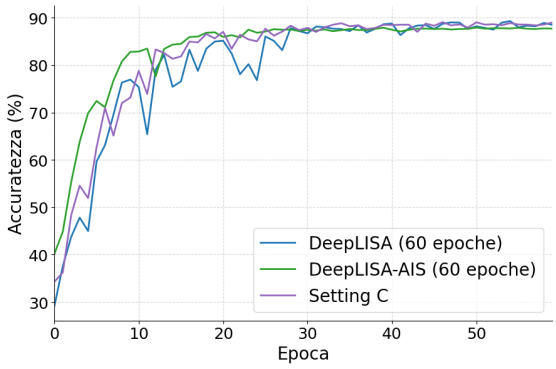
(b) Setting A: accuratezza per tempo.



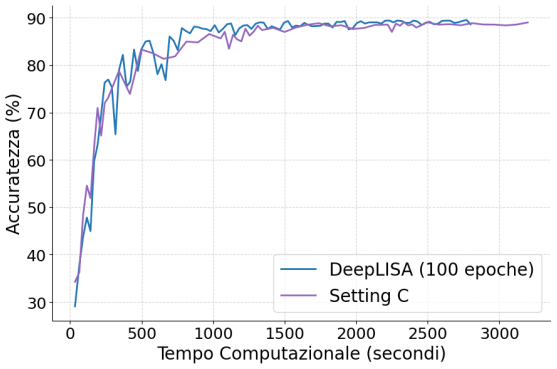
(c) Setting B: accuratezza per epoca.



(d) Setting B: accuratezza per tempo.



(e) Setting C: accuratezza per epoca.



(f) Setting C: accuratezza per tempo.

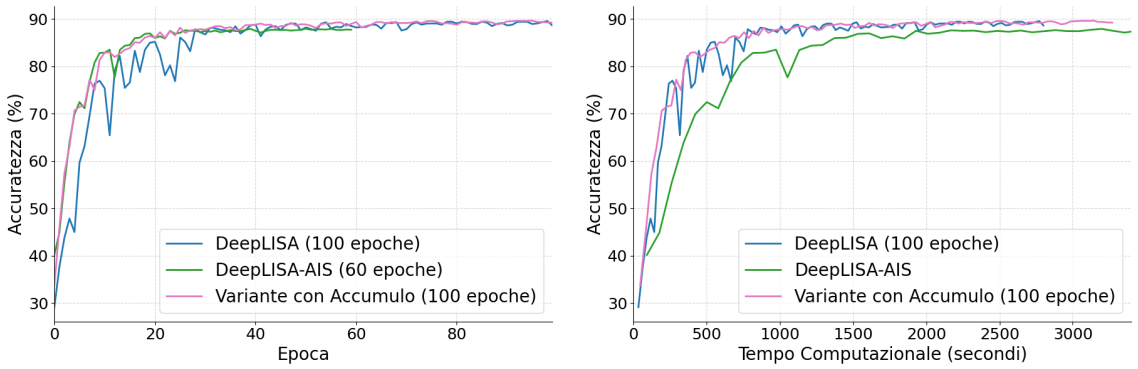
Figura 7.2: Confronto dell'accuratezza per epoca e per tempo computazionale (in secondi), valutando i metodi DeepLISA, DeepLISA-AIS e alcuni tra metodi ibridi.

Tuttavia, il risultato più interessante si ottiene attraverso il **Setting B**. Dopo la crescita iniziale data dalle estrazioni casuali, l'utilizzo della *strategia AIS* permette finalmente di ottenere un "salto" delle prestazioni, stabilizzando l'accuratezza ad un livello elevato: da 89.0% a 89.5%. Dunque, a parità di numero di iterazioni compiute, esso diventa più convincente dell'*algoritmo DeepLISA*, che raggiunge questi risultati solo dopo decine di epoche successive e con maggiori oscillazioni delle prestazioni (*Figura 7.2c*). Inoltre, l'utilizzo iniziale di *DeepLISA* e le ridotte oscillazioni delle prestazioni, rendono questa variante migliore in tutto l'intervallo temporale considerato (*Figura 7.2d*).

Infine, attraverso i risultati di **Setting C** osservabili nelle *Figure 7.2e - 7.2f*, si mostra che, quando viene utilizzata la *strategia AIS*, si ottengono risultati superiori rispetto ad una scelta causale, buone anche valutandole per tempo di esecuzione.

7.5.3 DeepLISA con Accumulo delle Norme dei Gradienti per AIS

Infine, si analizzano i risultati dell'ultima variante, descritta nella *Sezione 7.4*: *DeepLISA* con distribuzione di campionamento ottenuta dall'accumulo delle norme dei gradienti del rischio empirico sul mini-batch.



(a) Accuratezza per epoca. *(b) Accuratezza per tempo computazionale.*

Figura 7.3: Confronto dell'accuratezza per epoca e per tempo computazionale (in secondi), valutando i metodi DeepLISA, DeepLISA-AIS e con distribuzione ottenuta attraverso l'accumulo della norma del gradiente della funzione rischio empirico.

Dalla *Figura 7.3*, si osserva che questo metodo può rappresentare il giusto com-

promesso tra l'*efficienza computazionale* generata dalle informazioni del mini-batch e la necessità di costruire un processo rapido, superando la limitazione principale del calcolo dei gradienti per campione.

Nonostante non si segua propriamente la *strategia AIS*, su questo dataset, le prestazioni del metodo sono molto buone, sia rispetto al numero di iterazioni compiute, sia rispetto al tempo computazionale. Nel primo caso, mostrato nella *Figura 7.3a*, esso presenta ottime prestazioni iniziali, che si stabilizzano nell'intervallo $[88.8\%, 89.6\%]$ e completamente in linea con quelli ottenuti dalla *strategia AIS*. Attraverso questo comportamento, esso raggiunge le prestazioni elevate di *DeepLISA*, riducendo il suo comportamento oscillatorio. Un possibile merito di questo risultato è legato al fatto che, tramite dimensioni dei mini-batch molto elevate, sempre più campioni tendono ad avere le stesse componenti della distribuzione, paragonabile quasi a quella uniforme. Inoltre, la *Figura 7.3b* e la *Tabella 7.2* mostrano come il processo, nonostante sia leggermente più lento, permetta di ottenere risultati lievemente migliori, anche grazie alla riduzione delle oscillazioni nelle prestazioni finali.

Epoche	**DeepLISA**	**DeepLISA-AIS**	**Variante con Accumulo**
100	2799.64 s	-	3269.03 s
60	-	4762.52 s	-

Tabella 7.2: Confronto dei tempi computazionali (in secondi) per 60 *epoche di DeepLISA-AIS e* 100 *epoche di DeepLISA e della variante descritta nella Sezione 7.4.*

7.6 Applicazione ad un Problema di Classificazione Tumorale: il Mesotelioma

Al fine di valutare l'efficacia dei metodi proposti in un contesto applicativo reale, sono stati condotti alcuni test su un problema di classificazione multiclasse per il riconoscimento dei sottotipi istologici di una forma rara di cancro: il *mesotelioma*. È importante sottolineare che, in tal caso, non è stata scelta una rete neurale più efficiente per il tipo di problema, ad esempio in grado di cogliere al meglio dettagli

locali microscopici, tralasciando quelli globali, poco informativi. Ad esempio, l'ultilizzo degli strati di *pooling* riduce la risoluzione delle informazioni microscopiche. Inoltre non è stata eseguita alcuna operazione di fine-tuning. Dunque, l'obiettivo non è presentare un modello pronto per l'uso clinico, ma fornire un base di riferimento (*baseline*) ed una prima prova ad una sperimentazione più completa volta all'ottimizzazione della classificazione. Pertanto, si tratta di una *prova di fattibilità*, volta a verificare se le strategie proposte all'interno della tesi siano in grado di apprendere informazioni da un dataset complesso e non risultino unicamente utilizzabili sui dataset "giocattolo", come *CIFAR10*. Naturalmente, una fase successiva del lavoro dovrebbe prevedere la ricerca di reti neurali e iperparametri migliori, *fine-tuning*, per massimizzare le prestazioni finali ottenibili sul dataset.

7.6.1 CAMEL Dataset: Classification of Asbestos-related Mesothelioma Explainable Learning

Il **mesotelioma** è un raro tumore che si sviluppa a partire dalle cellule del *mesotelio*, sottile membrana che riveste gli organi interni, il cui principale fattore di rischio è l'esposizione all'amianto [2]. È possibile distinguerlo in quattro sottotipi istologici, dipendenti dalle caratteristiche delle cellule che lo costituiscono:

- **epitelioide**, caso più comune ed associato ad una prognosi mediamente migliore;
- **sarcomatoide** o *fibroso*, maggiormente aggressivo rispetto al precedente;
- **bifasico** o *misto*, caratterizzato dalla coesistenza dei primi due casi;
- **desmoplastico**, più raro e difficile da diagnosticare, non è presente all'interno del dataset considerato.

La *Figura 7.4* mostra alcuni campioni istologici che evidenziano le diverse morfologie cellulari: il mesotelioma epitelioide presenta cellule cuboidali, il sottotipo sarcomatoide è di tipo fusiforme e la variante bifasica comprende entrambe le componenti strutturali. Proprio queste caratteristiche devono essere apprese dal modello per fornire una corretta classificazione.

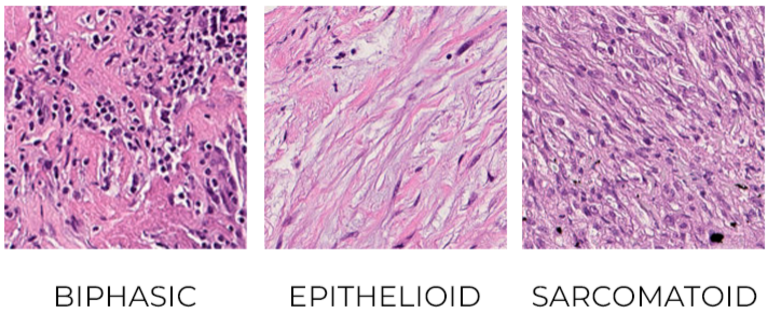


Figura 7.4: Vetrini di campioni istologici per alcuni sottotipi di mesotelioma pleurico: epitelioide, sarcomatoide, bifasico [?].

Il dataset in analisi, **dataset CAMEL (Classification of Asbestos-related Mesothelioma Explainable Learning)** [?], contiene immagini di vetrini relativi a **mesoteliomi pleurici**, ovvero presenti nella membrana polmonare esterna, la *pleura*. Essi sono forniti dal *Politecnico di Torino*, in collaborazione con l'*ospedale San Luigi di Torino*. La tecnica utilizzata per il contrasto di questi vetrini è la **colorazione Ematossilina-Eosina (H&E)**, che evidenzia i nuclei cellulari attraverso sfumature blu-viola date dall'ematossilina, separandoli con una colorazione rosa-rosso data dall'eosina, in grado di colorare citoplasma e strutture circostanti [1]. Tale contrasto cromatico permette di cogliere la differente morfologia dei sottotipi di mesotelioma.

Le immagini contenute nel dataset sono estratte da 15 vetrini differenti (*Whole Slide Images, WSIs*), suddivisi in 109 305 patch a colori di dimensioni 224×224:

- 5 WSIs per il mesotelioma bifasico, *classe 0*, da cui sono estratti 34 696 patch;
- 5 WSIs per il mesotelioma epitelioide, *classe 1*, suddivisi in 44 425 patch;
- 5 WSIs per il mesotelioma sarcomatoide, *classe 2*, per un totale di 30 184 patch.

Dunque, si tratta di un **problema di classificazione multiclasse**, che può essere affrontato attraverso la *ResNet18* descritta nella *Sezione 5.2*.

La principale limitazione di questo dataset risiede nello scarso numero di pazienti campionati, che determina una **ridotta varietà biologica e morfologica**. Infatti, persone con lo stesso sottotipo potrebbero presentare tumori differenti, di-

pendentemente dalla loro storia clinica e dal tipo di esposizione all'amianto. Inoltre, nonostante il controllo dichiarato dagli autori, non tutte le immagini possiedono un buon contenuto informativo; ad esempio, la *Figura 7.5* evidenzia alcuni artefatti presenti all'interno dei vetrini. Infine, considerando pazienti diversi le sfumature della *colorazione H&E* possono subire alcune variazioni cromatiche a causa di fattori esterni, come la concentrazione dei reagenti e il tempo di esposizione ai coloranti.

(a) Bifasico. *(b) Epitelioide.* *(c) Sarcomatoide.*

Figura 7.5: Esempi di vetrini con contenuto informativo corrotto: (a) mesotelioma bifasico che presenta solo due nuclei di cellule tumorali, confondibili con il caso epitelioide; (b) immagine non nitida; (c) immagine artefatta, priva di tessuto biologico informativo.

7.6.2 Preparazione del Dataset: Training Set e Test Set

Per addestrare il modello ed analizzarne le prestazioni, è necessario suddividere l'insieme di immagini in due sottoinsiemi: il *training set* e il *test set*.

In primo luogo, si è deciso di suddividere casualmente ogni classe, tale che circa l'80% dei suoi esempi sia utilizzato per l'addestramento e il restante 20% formi l'insieme di test. Nonostante la correttezza teorica di tale suddivisione su un generico dataset, in questo caso il modello sarà compromesso e presenterà prestazioni gonfiate rispetto alla realtà a causa di "scorciatoie" nell'apprendimento. Infatti, utilizzare i vetrini di uno stesso paziente per entrambi gli insiemi introduce una sorta di correlazione tra i due: il modello classificherà gli insiemi di test riconoscendo le caratteristiche che accomunano i vetrini dello stesso paziente, come, ad esempio, le stesse sfumature della colorazione. Si tratta di un caso di un fenomeno più ampio, noto come *Data Leakage*. Con il termine **Data Leakage** si intende l'utilizzo di informazioni illecite durante l'addestramento del modello, che gonfiano le perfor-

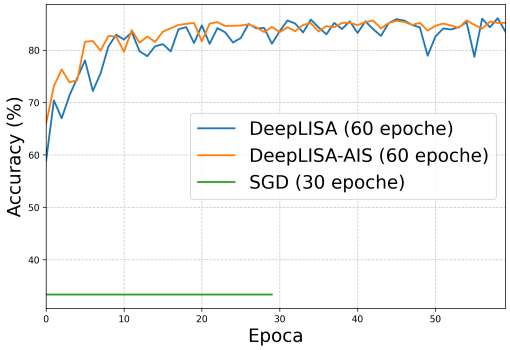


Figura 7.6: Accuratezze per epoca di ResNet18, addestrata con differenti metodi, nel caso in cui, erratamente, si distinguono gli esempi di uno stesso paziente in insieme di training e di test.

mance sul test set, ma non sono in grado di generalizzare sui dati futuri (*momento di inferenza*) [7]. In questo caso, il problema sorge dalla correlazione tra esempi del training set e test set. In effetti, osservando i risultati ottenuti dall'addestramento dei modelli sul test set (*Figura 7.6*), le prestazioni sono molto più elevate rispetto a ciò che si otterrà dalla seconda strategia di separazione training-test, 50% di accuratezza, raggiungendo velocemente un'accuratezza dell'85%.

Per evitare tale compromissione, si è deciso di separare il dataset in questo modo: per ogni classe, quattro pazienti sono scelti casualmente per il training set, un paziente comporrà il test set. Purtroppo, la scarsità di pazienti rende labile il test set: le prestazioni dipendono fortemente dai tre pazienti che lo compongono e dalla qualità delle sue immagini. Anche il training set, contenente un totale di 12 pazienti, non presenta sufficiente varietà per una robusta classificazione. La *Tabella 7.3* mostra il numero di campioni per ognuno di questi.

	Bifasico	**Epitelioide**	**Sarcomatoide**	**Totale**
Training Set	28 658 (33.7%)	33 588 (39.5%)	22 810 (26.8%)	85 056
Test Set	6 038 (24.9%)	10 837 (44.7%)	7 374 (30.4%)	24 249

Tabella 7.3: Tabella riassuntiva della suddivisione del dataset in training set e test set. Per ogni classe, è riportato il numero di patch all'interno degli insiemi, accompagnato dalla percentuale (%) rispetto all'insieme stesso di appartenenza.

7.6.3 Risultati di Classificazione del Mesotelioma

Informazioni sul Predittore e sui Metodi di Addestramento. Il *dataset CAMEL* rappresenta un problema di classificazione, su tre classi, di immagini a colori di dimensione 224×224. Siccome si è deciso di utilizzare lo stesso modello dei capitoli precedenti, adattato ad immagini a colori di dimensione 32×32, queste sono state opportunamente ridimensionate. Si osservi che questa modifica potrebbe ridurre il contenuto informativo utile per la classificazione. Dunque, in analisi future, si potrebbe sfruttare la *ResNet18* originale, nata per immagini 224×224.

Le modifiche apportante alla *Residual Network con 18 strati* sono riportante in *Sezione 5.2*. Inoltre, siccome a differenza di *CIFAR10* non conta l'orientazione verticale delle cellule, al fine aumentare la capacità di generalizzazione del modello, si aggiunge alle tecniche di *Data Augmentation* anche il *Random Vertical Flip*. Infine, la *normalizzazione* dei tre canali RGB degli esempi è stata riadattata al nuovo training set di immagini.

Siccome gli insiemi di training e di test sono lievemente sbilanciati, per uno dei metodi si è deciso di ricorrere anche a pesi specifici per i loro errori. In particolare, per ogni classe i, alla *Cross Entropy Loss* sarà aggiunto il peso

$$\frac{n_{\text{train}}}{c\, n_{\text{i}}}$$

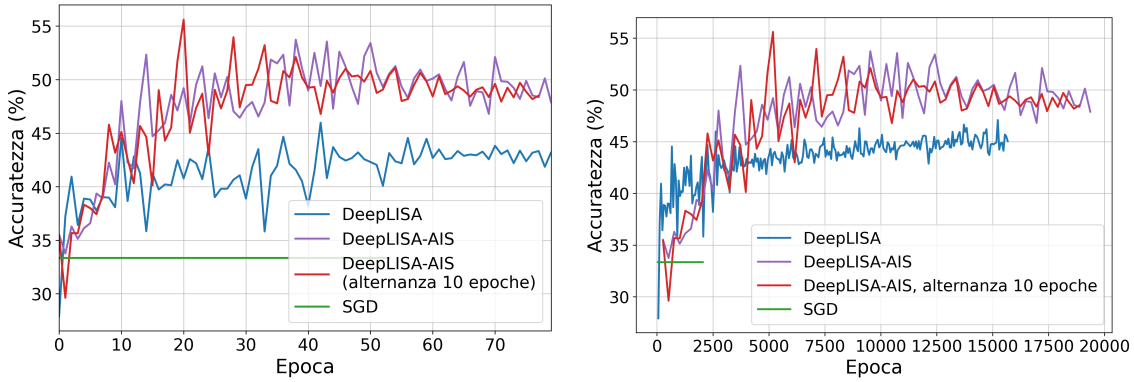
dove $c = 3$ è il numero di classi, n_{train} è il numero di esempi del training set, n_i di essi appartenenti alla classe i, con $i \in \{0, 1, 2\}$.

Senza compiere alcuna operazione per il *fine-tuning*, sono stati confrontati i risultati della *ResNet18* addestrata attraverso

- *SGD* con dimensione del mini-batch fissa, pari a 32, e lunghezza di passo 0.05;
- *DeepLISA*;
- *DeepLISA-AIS*, considerando anche il bilanciamento sopracitato;
- *DeepLISA-AIS* reinizializzato ogni 10 epoche.

Gli iperparametri utilizzati sono quelli impostati per il dataset *CIFAR10*.

Infine, dato lo sbilanciamento, le prestazioni saranno confrontate attraverso la metrica *accuratezza bilanciata*, media delle accuratezze per ogni classe.

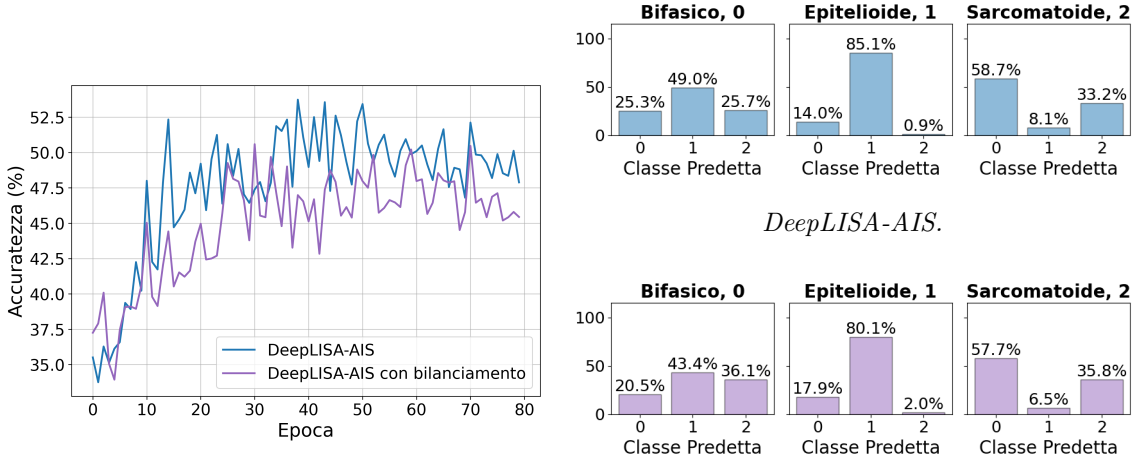


(a) Accuratezza per epoca. *(b) Accuratezza per tempo computazionale.*

Figura 7.7: Confronto dell'accuratezza bilanciata ottenuta attraverso differenti metodi di addestramento del modello. Per l'algoritmo DeepLISA, per il metodo SGD e per le strategie AIS sono considerate rispettivamente 100*,* 56 *e* 80 *epoche.*

Analisi dei Risultati. In *Figura 7.7*, sono mostrati i risultati dell'*accuratezza bilanciata* del modello sulle prime 80 epoche. Si osserva immediatamente che, attraverso il *metodo SGD*, il modello diverge, a causa della lunghezza di passo fissa troppo elevata. Questo sottolinea il notevole vantaggio nella scelta della *procedura di Line Search*, che riduce la dipendenza delle prestazioni dalla scelta dei suoi iperparametri e, di conseguenza, il tempo computazionale per il fine-tuning.
Inoltre, al contrario di quanto riscontrato in *CIFAR10*, la *strategia AIS* porta ad un rapido miglioramento solo dopo la decima epoca, le cui prestazioni terminano attorno ad un'accuratezza bilanciata del 48%. In tal caso, nonostante i tempi computazionali dei gradienti per campione siano limitativi, aumentando il processo di 3.09 minuti per epoca, la bontà delle prestazioni rende il metodo *DeepLISA-AIS* migliore già dopo 40 minuti. Infine, a differenza dei risultati ottenuti per il dataset *CIFAR10*, il comportamento oscillatorio delle prestazioni non è ridotto.

Successivamente, siccome il training set è lievemente sbilanciato, sono state confrontate le accuratezze ottenute per ogni classe attraverso *DeepLISA-AIS* e la sua versione con bilanciamento. Sperimentalmente, all'interno della *Sezione 6.4*, si è mostrato come il metodo tenda a concentrare le estrazioni sulle classi maggiormente sbagliate. Dunque, aggiungere pesi al dataset potrebbe non essere necessario e, anzi, dannoso per i risultati. In realtà, dalla *Figura 7.8* si osserva un leggero miglioramento per la classe meno presente, i mesoteliomi sarcomatoidi, a discapito degli altri



Accuratezza per epoca.

DeepLISA-AIS.

DeepLISA-AIS con bilanciamento.

Figura 7.8: Confronto dei risultati ottenuti attraverso il metodo DeepLISA-AIS e la sua versione con bilanciamento attraverso l'attribuzione di pesi nella funzione loss: il grafico a sinistra mostra le accuratezze bilanciate ottenute durante il processo; i grafici a destra mostrano le predizioni compiute dal modello dopo 80 *epoche per ogni insieme di esempi appartenente alla stessa classe.*

due sottotipi.

Inoltre, questi risultati evidenziano la difficoltà principale della classificazione: la classe 0, sottotipo bifasico, è quella più difficile da distinguere a causa della coesistenza di caratteristiche delle restanti classi. Essa rende anche difficile la classificazione del mesotelioma sarcomatoide.

Conclusioni

Nella presente tesi, è stata studiato e implementato un metodo di Discesa del Gradiente Stocastico basato sulla fusione di due differenti strategie per la riduzione della varianza, con l'obiettivo di migliorare l'addestramento dei modelli nel Deep Learning. In particolare, sono stati integrati tre approcci distinti: l'aumento dinamico della dimensione del mini-batch, presentato nell'*algoritmo DeepLISA*, la procedura di Line Search per la determinazione della lunghezza di passo e la selezione dei campioni tramite la *distribuzione Adaptive Importance Sampling (AIS)*, caratteristiche del metodo *Mini-Batch SGD-AIS*.

Dall'analisi sperimentale condotta sulla rete neurale convoluzionale *ResNet18* sul *dataset CIFAR10*, è emersa la notevole rilevanza della *strategia AIS* sulle prestazioni del modello, che garantisce un rapido incremento nelle prime epoche, riducendone inoltre il comportamento oscillatorio. Tuttavia, il tempo computazionale necessario per l'esecuzione, dovuto alla necessità di calcolare le norme dei gradienti per campione, rappresenta il suo principale limite.

Per tale motivo, sono state sviluppate alcune strategie volte alla riduzione del tempo computazionale e all'incremento delle prestazioni. In particolare, si distinguono l'aggiornamento della distribuzione per epoca, l'aggiornamento puntuale della dimensione del mini-batch (*variante LISA*) e l'eliminazione del calcolo dei gradienti per campione, sfruttando informazioni ricavate dall'intero mini-batch. Tuttavia, i risultati più significativi sono stati ottenuti alternando fasi di campionamento uniforme a fasi di campionamento adattivo (*metodo ibrido*).

Gli sviluppi futuri di questo lavoro si muoveranno in due direzioni. Da una parte, sarà studiata una strategia adattiva capace di alternare in modo automatico il campionamento uniforme ed adattivo, oltre all'aggiornamento puntuale della dimensione del mini-batch, evitando di forzare questi cambiamenti attraverso iper-

parametri prefissati. Dall'altra, ci si occuperà di fornire una dimostrazione formale della riduzione della varianza dello stimatore AIS nel caso di mini-batch.

Infine, dati i risultati ottenuti sul dataset medico per la classificazione del mesotelioma, sarà necessaria la ricerca di modelli migliori per la classificazione di questo tipo.

Appendice

In questa appendice sono riportate alcune dimostrazioni relative a teoremi del *Capitolo 3* e del *Capitolo 4*.

Dimostrazione del Lemma 3.2.3 [13, Lemma 2.2, Sec. 2]. Innanzitutto, si dimostri che, per ogni iterazione $k \in \mathbb{N}$, la procedura si arresta. Banalmente, se $\alpha_{\max}$ soddisfa la condizione

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})) \leq \max_{0 \leq j \leq \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) - \gamma\alpha \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2, \quad (5)$$

allora si pone $\alpha_k = \alpha_{\max}$ e la procedura termina. Inoltre, in tal caso la disuguaglianza (3.7) è verificata.

Supponiamo ora che $\alpha_{\max}$ non soddisfi la disuguaglianza (5). Siccome la funzione $f_{\mathcal{N}_k}$ verifica il *Descent Lemma 3.2.1* (*Osservazione 3.2.2*), si ha che, per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, vale

$$f_{\mathcal{N}_k}(\mathbf{y}) \leq f_{\mathcal{N}_k}(\mathbf{x}) + \nabla f_{\mathcal{N}_k}(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{L_{\mathcal{N}_k}}{2} \|\mathbf{y} - \mathbf{x}\|^2.$$

In particolare, ponendo $\mathbf{y} := \mathbf{w}^{(k+1)}$ e $\mathbf{x} := \mathbf{w}^{(k)}$ e ricordando la regola di aggiornamento $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$, si ottiene

$$\begin{aligned} f_{\mathcal{N}_k}\big(\mathbf{w}^{(k)} - \alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\big) &\leq f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) + \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^\top \big(-\alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\big) \\ &\quad + \frac{L_{\mathcal{N}_k}}{2} \| - \alpha \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 \\ &\leq f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \alpha \left(1 - \frac{L_{\mathcal{N}_k}\alpha}{2}\right) \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 \\ &\overset{(A)}{\leq} \max_{0 \leq j \leq \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}) - \alpha \left(1 - \frac{L_{\mathcal{N}_k}\alpha}{2}\right) \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 \end{aligned} \quad (6)$$

dove la disuguaglianza (A) è conseguenza della definizione di massimo, tale che

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) \leq \max_{0 \leq j \leq \min\{k,M\}} f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)}).$$

Di conseguenza, se vale

$$-\Big(1-\frac{L_{\mathcal{N}_k}\alpha}{2}\Big)\leq-\gamma \qquad \Longleftrightarrow \qquad \alpha\leq\frac{2(1-\gamma)}{L_{\mathcal{N}_k}}$$

la condizione di arresto (5) è verificata.
Siccome $\beta\in(0,1)$ e $\alpha_{\max}$ non soddisfa la disuguaglianza (5),

$$\exists m>0 \qquad : \qquad \alpha_{\max}\beta^m\leq\frac{2(1-\gamma)}{L_{\mathcal{N}_k}}.$$

Dunque, da quanto verificato in precedenza, la procedura si arresta, ponendo $\alpha_k:=\alpha_{\max}\beta^m$. In conclusione, la procedura di Line Search è ben posta.

Quindi, si dimostri che anche in tal caso vale la disuguaglianza (3.7). Siccome $m>0$ è il più piccolo intero per cui vale (5), $\frac{\alpha_{\max}}{\beta}=\alpha_{\max}\beta^(m-1)$ sarà tale che

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)}-\frac{\alpha_{\max}}{\beta}\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}))\leq\max_{0\leq j\leq\min\{k,M\}}f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)})-\gamma\frac{\alpha_{\max}}{\beta}\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2,$$

da cui,

$$\begin{aligned}\gamma\frac{\alpha_k}{\beta}&>\frac{\max_{0\leq j\leq\min\{k,M\}}f_{\mathcal{N}_{k-j}}(\mathbf{w}^{(k-j)})-f_{\mathcal{N}_k}\big(\mathbf{w}^{(k)}-\frac{\alpha_k}{\beta}\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\big)}{\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2}\\&\overset{(B)}{\geq}\frac{\alpha_k}{\beta}\left(1-\frac{L_{\mathcal{N}_k}\alpha_k}{2\beta}\right)\end{aligned}$$

dove (B) è conseguenza diretta della disuguaglianza (6). Dunque,

$$\alpha_k>\frac{2\beta(1-\gamma)}{L_{\mathcal{N}_k}}\geq\min\left\{\alpha_{\max},\frac{2\beta(1-\gamma)}{L_{\mathcal{N}_k}}\right\}.$$

Infine, siccome $L_{\mathcal{N}_k}\leq L_{\max}$, si ottiene la tesi.

□

Dimostrazione del Teorema 3.2.5 [13, Teorema 2.1, Sec. 2]. Innanzitutto, si dimostri che $\delta=2\tilde{\alpha}-L\alpha_{\max}^2>0$.
Se $\alpha_{\max}\leq\frac{2\beta(1-\gamma)}{L_{\max}}$, dalla definizione di $\tilde{\alpha}$, si ha che $\tilde{\alpha}=\alpha_{\max}$. Dunque, dall'ipotesi $\alpha_{\max}<\frac{2}{L}$, si ottiene che

$$\delta=2\tilde{\alpha}-L\alpha_{\max}^2=2\alpha_{\max}-L\alpha_{\max}^2>0.$$

Altrimenti, supponiamo che valga

$$\alpha_{\max}>\frac{2\beta(1-\gamma)}{L_{\max}}. \tag{7}$$

Per definizione di $\tilde{\alpha}$, si ha $\tilde{\alpha} = \frac{2\beta(1-\gamma)}{L_{\max}}$, da cui

$$\delta = \frac{4\beta(1-\gamma)}{L_{\max}} - L\alpha_{\max}^2.$$

Esso è strettamente positivo se

$$0 < \alpha_{\max} < \sqrt{\frac{4\beta(1-\gamma)}{L_{\max}L}}.$$

ovvero, ricordando l'ipotesi (7),

$$\frac{2\beta(1-\gamma)}{L_{\max}} < \alpha_{\max} < \sqrt{\frac{4\beta(1-\gamma)}{L_{\max}L}} \quad \iff \quad \frac{2}{L}\frac{L\beta(1-\gamma)}{L_{\max}} < \alpha_{\max} < \frac{2}{L}\sqrt{\frac{L\beta(1-\gamma)}{L_{\max}L}}$$

Ciò si verifica quando

$$\alpha_{\max} < \frac{2}{L} \quad \text{e} \quad \sqrt{\frac{L\beta(1-\gamma)}{L_{\max}}} < 1.$$

Mentre la prima condizione è un'ipotesi del teorema, la seconda, osservando che $\frac{L}{L_{\max}} \leq 1$ e $\beta \in (0,1)$ si ottiene se $0 < \gamma < 1$.

Ora, si dimostri che vale la disuguaglianza (3.8). Dall'*Assunzione 3.1.3*, si ha che f è *L-Lipschitz continua*. Dunque, vale la disuguaglianza (3.6) del *Descent Lemma 3.2.1*. In particolare, ponendo $y := \mathbf{w}^{(k+1)}$ e $x := \mathbf{w}^{(k)}$ e ricordando la regola di aggiornamento $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$, si dimostra che

$$\begin{aligned}
f(\mathbf{w}^{(k+1)}) - f(\mathbf{w}^{(k)}) &\leq -\alpha_k \nabla f(\mathbf{w}^{(k)})^\top \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) + \frac{L\alpha_k^2}{2}\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 \\
&= \frac{\alpha_k}{2}\Big(\|\nabla f(\mathbf{w}^{(k)}) - \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 - \|\nabla f(\mathbf{w}^{(k)})\|^2 - \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\Big) + \\
&\quad + \frac{L\alpha_k^2}{2}\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 \\
&\overset{(A)}{\leq} \frac{\alpha_{\max}}{2}\|\nabla f(\mathbf{w}^{(k)}) - \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 - \frac{\tilde{\alpha}}{2}\Big(\|\nabla f(\mathbf{w}^{(k)})\|^2 + \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\Big) + \\
&\quad + \frac{L\alpha_{\max}^2}{2}\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2,
\end{aligned}$$

dove (A) è conseguenza del fatto che $0 < \tilde{\alpha} \leq \alpha_k \leq \alpha_{\max}$ (*Lemma 3.2.3*).
Considerandone l'aspettazione condizionata rispetto alla σ-algebra $\mathcal{F}_k$, per ogni $k \in \mathbb{N}_0$, grazie alle sue proprietà di linearità e monotonia, si ottiene

$$\begin{aligned}
2\mathbb{E}_k\big[f(\mathbf{w}^{(k+1)}) - f(\mathbf{w}^{(k)})\big] &\leq \alpha_{\max}\mathbb{E}_k\big[\|\nabla f(\mathbf{w}^{(k)}) - \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\big] + \\
&\quad - \tilde{\alpha}\left(\mathbb{E}_k\big[\|\nabla f(\mathbf{w}^{(k)})\|^2\big] + \mathbb{E}_k\big[\|\nabla\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\big]\right) + \\
&\quad + L\alpha_{\max}^2\mathbb{E}_k\big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\big],
\end{aligned}$$

da cui, siccome vale (3.4),

$$2\mathbb{E}_k\big[f(\mathbf{w}^{(k+1)}) - f(\mathbf{w}^{(k)})\big] = (\alpha_{\max} + L\alpha_{\max}^2 - \tilde{\alpha})\mathbb{E}_k\big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\big] - (\alpha_{\max} + \tilde{\alpha})\|\nabla f(\mathbf{w}^{(k)})\|^2.$$

Siccome $\mathbb{E}_k\big[\|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2\big]$ verifica la disuguaglianza 3.5, si ottiene

$$\begin{aligned}2\mathbb{E}_k\big[f(\mathbf{w}^{(k+1)}) - f(\mathbf{w}^{(k)})\big] &\leq (\alpha_{\max} + L\alpha_{\max}^2 - \tilde{\alpha})\big(\|\nabla f(\mathbf{w}^{(k)})\|^2 + \varepsilon_k\big) - (\alpha_{\max} + \tilde{\alpha})\|\nabla f(\mathbf{w}^{(k)})\|^2 \\ &= -(2\tilde{\alpha} - L\alpha_{\max}^2)\|\nabla f(\mathbf{w}^{(k)})\|^2 + (\alpha_{\max} + L\alpha_{\max}^2 - \tilde{\alpha})\varepsilon_k.\end{aligned}$$

Ponendo $\xi := (\alpha_{\max} + L\alpha_{\max}^2 - \tilde{\alpha})$, strettamente positiva poiché $0 < \tilde{\alpha} \leq \alpha_{\max}$, e ricordando che $\delta := 2\tilde{\alpha} - L\alpha_{\max}^2 > 0$, si ottiene

$$2\mathbb{E}_k\big[f(\mathbf{w}^{(k+1)}) - f(\mathbf{w}^{(k)})\big] \leq -\delta\|\nabla f(\mathbf{w}^{(k)})\|^2 + \xi\varepsilon_k,$$

e, equivalentemente,

$$\|\nabla f(\mathbf{w}^{(k)})\|^2 \leq \frac{2}{\delta}\mathbb{E}_k\big[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^{(k+1)})\big] + \frac{\xi}{\delta}\varepsilon_k \qquad \forall k \in \mathbb{N}_0 \tag{8}$$

Si calcoli l'aspettazione della disuguaglianza, che, attraverso la sua monotonia e la legge dell'aspettazione totale, permette di ricavare

$$\mathbb{E}\big[\|\nabla f(\mathbf{w}^{(k)})\|^2\big] \leq \mathbb{E}\big[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^{(k+1)})\big] + \frac{\xi}{\delta}\varepsilon_k \tag{9}$$

Poi, sommando per $k \in \{0, 1, \ldots, T-1\}$ e dividendo entrambi i membri per T,

$$\begin{aligned}\frac{1}{T}\sum_{k=0}^{T-1}\mathbb{E}\big[\|\nabla f(\mathbf{w}^{(k)})\|^2\big] &\leq \frac{2}{\delta T}\sum_{k=0}^{T-1}\mathbb{E}\big[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^{(k+1)})\big] + \frac{\xi}{\delta T}\sum_{k=0}^{T-1}\varepsilon_k \\ &\overset{(B)}{\leq} \frac{2}{\delta T}\big(f(\mathbf{w}^{(0)}) - \mathbb{E}[f(\mathbf{w}^{(T)})]\big) + \frac{\xi}{\delta T}\sum_{k=0}^{T-1}\varepsilon_k \\ &\overset{(C)}{\leq} \frac{2}{\delta T}(f(\mathbf{w}^{(0)}) - f^*) + \frac{\xi}{\delta T}\sum_{k=0}^{T-1}\varepsilon_k,\end{aligned} \tag{10}$$

dove

- (B) è ottenuto dalla linearità dell'aspettazione, dallo sviluppo della serie telescopica e dalla deterministicità di $f(\mathbf{w}^{(0)})$, da cui $\mathbb{E}[f(\mathbf{w}^{(0)})] = f(\mathbf{w}^{(0)})$;
- (C) è conseguenza diretta dell'*Assunzione 3.1.2* e della linearità dell'aspettazione $\big(\mathbb{E}[f(\mathbf{w}^{(T)})] \geq f^*\big)$.

Banalmente, per definizione di minimo, la tesi è verificata.

Poi, si dimostri

1. $\sum_{k=0}^{+\infty} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2] < +\infty$.

Anch'esso è conseguenza diretta di (10), eliminando il fattore $\frac{1}{T}$ da entrambi i membri.

Infine, si dimostri

2. $\lim_{k\to+\infty} \|\nabla f(\mathbf{w}^{(k)})\| = 0$ a.s.

Sviluppando la disuguaglianza (9) e ricordando che $f^* \leq f(\mathbf{w})$ per ogni $\mathbf{w} \in \mathbb{R}^d$, si ottiene

$$\mathbb{E}_k[f(\mathbf{w}^{(k+1)}) - f^*] \leq \big(f(\mathbf{w}^{(k)}) - f^*\big) - \frac{\delta}{2}\|\nabla f(\mathbf{w}^{(k)})\|^2 + \frac{\xi}{2}\varepsilon_k, \tag{11}$$

da cui, per il *Lemma 3.1.7*,

$$\sum_{k=0}^{\infty} \|\nabla f(\mathbf{w}^{(k)})\|^2 < +\infty \quad \text{a.s.}$$

ponendo

- $\nu_k := f(\mathbf{w}^{(k)}) - f^*$, non negativo per *Assunzione 3.1.2*;
- $u_k := \frac{\delta}{2}\|\nabla f(\mathbf{w}^{(k)})\|^2$, non negativo poichè $\delta > 0$, come dimostrato in precedenza;
- $\mu_k := \frac{\xi}{2}\varepsilon_k$, non negativo e sommabile grazie alle ipotesi sulla successione $\{\varepsilon_k\}_k$.

Dunque, dalla condizione necessaria per la convergenza delle serie di termini non negativi, si ottiene

$$\lim_{k\to+\infty} \|\nabla f(\mathbf{w}^{(k)})\| = 0 \quad \text{a.s.}$$

□

Dimostrazione del Teorema 3.2.6 [13, Teorema 2.2, Sec. 2]. Sia $\mathbf{w}^*$ una soluzione del problema (3.1), si dimostri che la successione $\{\mathbf{w}^{(k)}\}_k$ converge quasi sicuramente ad un punto di minimo. Ricordando la legge di aggiornamento $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})$, vale

$$\begin{aligned}\|\mathbf{w}^{(k+1)} - \mathbf{w}^*\|^2 &= \|\mathbf{w}^{(k)} - \alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \mathbf{w}^*\|^2 \\ &= \|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2 + \alpha_k^2 \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 + 2\alpha_k \nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T(\mathbf{w}^* - \mathbf{w}^{(k)})\end{aligned}$$

Sommando e sottraendo $2\alpha_k \nabla f(\mathbf{w}^{(k)})^T(\mathbf{w}^* - \mathbf{w}^{(k)})$ al secondo membro,

$$\begin{aligned}\|\mathbf{w}^{(k+1)} - \mathbf{w}^*\|^2 &= \|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2 + \alpha_k^2 \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 + 2\alpha_k \nabla f(\mathbf{w}^{(k)})^T(\mathbf{w}^* - \mathbf{w}^{(k)}) + \\ &\quad + 2\alpha_k (\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)}))^T(\mathbf{w}^* - \mathbf{w}^{(k)}) \\ &\overset{(A)}{\leq} \|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2 + \alpha_k^2 \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 + 2\alpha_k (f(\mathbf{w}^*) - f(\mathbf{w}^{(k)})) + \\ &\quad + 2\alpha_k \frac{\sqrt{\epsilon_k}}{2} \|\mathbf{w}^* - \mathbf{w}^{(k)}\|^2 + 2\alpha_k \frac{1}{2\sqrt{\epsilon_k}} \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2 \\ &\overset{(B)}{\leq} (1 + \alpha_{\max}\sqrt{\varepsilon_k}) \|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2 + \alpha_{\max}^2 \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})\|^2 + \\ &\quad + \alpha_{\max} \frac{1}{\sqrt{\varepsilon_k}} \|\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \nabla f(\mathbf{w}^{(k)})\|^2,\end{aligned}$$

dove

(A) è conseguenza della convessità di f, per cui

$$f(\mathbf{w}^*) \geq f(\mathbf{w}^{(k)}) + \nabla f(\mathbf{w}^{(k)})^T(\mathbf{w}^* - \mathbf{w}^{(k)}),$$

e della *disuguaglianza di Young*

$$\mathbf{x}^T\ \mathbf{y} \leq \frac{\|\mathbf{x}\|^2}{2\tau} + \frac{\tau\|\mathbf{y}\|^2}{2}, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^d, \tau \neq 0,$$

con $\tau := \sqrt{\varepsilon_k}$;

(B) è dovuto al fatto che $0 < \alpha_k \leq \alpha_{\max}$ e $f(\mathbf{w}^*) - f(\mathbf{w}^{(k)}) \leq 0$.

Considerando il suo valore atteso condizionato rispetto alla σ-algebra $\mathcal{F}_k$, si ottiene

$$\mathbb{E}_k[\|\mathbf{w}^{(k+1)} - \mathbf{w}^*\|^2] \leq (1 + \alpha_{\max}\sqrt{\varepsilon_k}) \|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2 + \alpha_{\max}^2 \|\nabla f(\mathbf{w}^{(k)})\|^2 + \alpha_{\max}^2 \varepsilon_k + \alpha_{\max}\sqrt{\varepsilon_k}.$$

Dunque, è possibile applicare il *Lemma 3.1.7*, in quanto sono verificate corrispondenti ipotesi, considerando

- $\eta_k := \alpha_{\max}\sqrt{\varepsilon_k}$, sommabile in quanto, per ipotesi $\sqrt{\varepsilon_k}$ è sommabile;
- $u_k = 0$;
- $\mu_k = \alpha_{\max}^2\|\nabla f(\mathbf{w}^{(k)})\|^2 + \alpha_{\max}^2\varepsilon_k + \alpha_{\max}\sqrt{\varepsilon_k}$, sommabile poiché $\sum_{k=0}^{+\infty}\|\nabla f(\mathbf{w}^{(k)})\|^2 < +\infty$ quasi sicuramente per il *Teorema 3.2.5*, $\{\varepsilon_k\}_k$ e $\{\sqrt{\varepsilon_k}\}_k$ sono sommabili per ipotesi.

Allora, $\{\|\mathbf{w}^{(k)} - \mathbf{w}^*\|\}_k$ converge quasi sicuramente per ogni $\mathbf{w}^* \in X^*$. Da questo risultato, è possibile dimostrare che esiste $\bar{\mathbf{w}} \in X^*$ tale che

$$\exists \tilde{\mathbf{w}} \in X^* \quad : \quad \mathbf{w}^{(k)} \to \tilde{\mathbf{w}} \quad \text{per } k \to \infty \quad \text{quasi sicuramente.} \tag{12}$$

Infatti, sia $\{\mathbf{w}_i^*\}_i$ un sottoinsieme numerabile dell'interno relativo di X^*, $\mathrm{ri}(X^*)$, denso in X^*. Siccome $\{\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\|\}_k$ converge quasi sicuramente per ogni i, si deduce la seguente probabilità

$$\mathbb{P}(\{\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\| \text{ non converge}\}) = 0.$$

Dunque,

$$\mathbb{P}\Big(\forall i, \exists c_i \text{ t.c. } \lim_{k\to\infty}\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\| = c_i\Big) = 1,$$

in quanto

$$\mathbb{P}\Big(\forall i, \exists c_i \text{ t.c. } \lim_{k\to\infty}\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\| = c_i\Big) \leq 1$$

e

$$\begin{aligned}\mathbb{P}\Big(\forall i, \exists c_i \text{ t.c. } \lim_{k\to\infty}\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\| = c_i\Big) &= 1 - \mathbb{P}\Big(\{\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\| \text{ non converge}\}\Big) \\ &\geq 1 - \sum_i \mathbb{P}\Big(\{\|\mathbf{w}^{(k)} - \mathbf{w}_i^*\| \text{ non converge}\}\Big) \\ &= 1\end{aligned}$$

Ora, si dimostri che

$$\exists \tilde{\mathbf{w}} \in X^* \quad : \quad \|\mathbf{w}^{(k)} - \tilde{\mathbf{w}}\| \to 0 \quad \text{per } k \to \infty \quad \text{quasi sicuramente,}$$

grazie a cui la tesi (12) è verificata. Supponiamo per assurdo che esistano sottosuccessioni convergenti $\{\mathbf{w}^{(k)}\}$, indicate come $\{\mathbf{a}_{k_j}\}_j$ e $\{\mathbf{b}_{k_j}\}_j$, che convergono rispettivamente ad $\mathbf{a}^*$ e $\mathbf{b}^*$, con $\|\mathbf{a}^* - \mathbf{b}^*\| = r > 0$. Per il *Teorema 3.2.5*, $\mathbf{a}^*$ e $\mathbf{b}^*$ sono punti

di stazionarietà e, in particolare, punti di minimo, in quando f è convessa. Quindi, $\mathbf{a}^*, \mathbf{b}^* \in X^*$.
Siccome $\{\mathbf{w}_i^*\}_i$ è denso in X^*, per ogni $\varepsilon > 0$, esistono $\mathbf{w}_{i_a}^*$ e $\mathbf{w}_{i_b}^*$ tali che $\|\mathbf{w}_{i_a}^* - \mathbf{a}^*\| < \varepsilon$ e $\|\mathbf{w}_{i_b}^* - \mathbf{b}^*\| < \varepsilon$. Inoltre, per k_j sufficientemente grande, $\|\mathbf{a}_{k_j} - \mathbf{a}^*\| < \varepsilon$ e $\|\mathbf{b}_{k_j} - \mathbf{b}^*\| < \varepsilon$. Allora,

$$\|\mathbf{a}_{k_j} - \mathbf{w}_{i_a}^*\| = \|\mathbf{a}_{k_j} - \mathbf{a}^* - (\mathbf{w}_{i_a}^* - \mathbf{a}^*)\| \leq \|\mathbf{a}_{k_j} - \mathbf{a}^*\| + \|\mathbf{a}^* - \mathbf{w}_{i_a}^*\| < 2\varepsilon$$

e, dunque, per l'arbitrarietà di $\varepsilon > 0$, $\{\|\mathbf{a}_{k_j} - \mathbf{w}_{i_a}^*\|\}_{k_j}$ converge quasi sicuramente a 0. Tuttavia,

$$\|\mathbf{b}_{k_j} - \mathbf{w}_{i_a}^*\| \geq \|\mathbf{b}^* - \mathbf{a}^*\| - \|\mathbf{a}^* - \mathbf{w}_{i_a}^*\| - \|\mathbf{b}_{k_j} - \mathbf{b}^*\| > r - 2\varepsilon.$$

Quest'ultima disuguaglianza contraddice la convergenza quasi sicura a 0 di $\|\mathbf{w}^{(k)} - \mathbf{w}_{i_a}^*\|$. In conclusione, deve essere $\mathbf{a}^* = \mathbf{b}^*$ e, quindi, deve esiste $\tilde{\mathbf{w}} \in X^*$ tale che

$$\|\mathbf{w}^{(k)} - \tilde{\mathbf{w}}\| \to 0 \quad \text{per } k \to \infty \text{ quasi sicuramente.}$$

□

Dimostrazione del Teorema 3.2.7 [13, Teorema 2.3, Sec. 2]. Ripetendo i calcoli svolti per la disuguaglianza (13) senza considerare $f(\mathbf{w}^*) - f(\mathbf{w}^{(k)}) < 0$, si ricava

$$\begin{aligned}\mathbb{E}_k[\|\mathbf{w}^{(k+1)} - \mathbf{w}^*\|^2] \leq (1 + \alpha_{\max}\sqrt{\varepsilon_k})\|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2 + \alpha_{\max}^2\|\nabla f(\mathbf{w}^{(k)})\|^2 + \\ + \alpha_{\max}^2\varepsilon_k + \alpha_{\max}\sqrt{\varepsilon_k} + 2\tilde{\alpha}\,\mathbb{E}_k\big(f(\mathbf{w}^*) - f(\mathbf{w}^{(k)})\big).\end{aligned}$$

Si sommi la disuguaglianza per $k \in \{0, 1, \ldots, T\}$ e ne si calcoli l'aspettazione totale. Sfruttando la proprietà di linearità dell'aspettazione e la legge dell'aspettazione totale, si ottiene

$$\begin{aligned}\mathbb{E}[\|\mathbf{w}^{(k+1)} - \mathbf{w}^*\|^2] \leq \sum_{k=0}^{T}\mathbb{E}[\|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2] + \alpha_{\max}\sum_{k=0}^{T}\sqrt{\varepsilon_k}\mathbb{E}[\|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2] + \\ + \alpha_{\max}^2\sum_{k=0}^{T}\mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2] + \\ + \alpha_{\max}^2\sum_{k=0}^{T}\varepsilon_k + \alpha_{\max}\sum_{k=0}^{T}\sqrt{\varepsilon_k} + 2\tilde{\alpha}\sum_{k=0}^{T}\mathbb{E}\big[f(\mathbf{w}^*) - f(\mathbf{w}^{(k)})\big],\end{aligned}$$

da cui, ricavando $\sum_{k=0}^{T} \mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)]$, si ottiene

$$\begin{aligned}\sum_{k=0}^{T} \mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] \leq \frac{1}{2\tilde{\alpha}}\Big(\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 - \mathbb{E}[\|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2]\Big) \\ + \frac{\alpha_{\max}^2}{2\tilde{\alpha}} \sum_{k=0}^{T} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2] \\ + \frac{\alpha_{\max}}{2\tilde{\alpha}} \sum_{k=0}^{T} \sqrt{\varepsilon_k}\, \mathbb{E}[\|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2] \\ + \frac{\alpha_{\max}^2}{2\tilde{\alpha}} \sum_{k=0}^{T} \varepsilon_k + \frac{\alpha_{\max}}{2\tilde{\alpha}} \sum_{k=0}^{T} \sqrt{\varepsilon_k}.\end{aligned}$$

Poi, utilizzando $\mathbb{E}[\|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2] \geq 0$,

$$\begin{aligned}\sum_{k=0}^{T} \mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] \leq \frac{1}{2\tilde{\alpha}}\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\alpha_{\max}^2}{2\tilde{\alpha}} \sum_{k=0}^{T} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2] + \\ + \frac{\alpha_{\max}}{2\tilde{\alpha}} \sum_{k=0}^{T} \sqrt{\varepsilon_k}\, \mathbb{E}[\|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2] + \\ + \frac{\alpha_{\max}^2}{2\tilde{\alpha}} \sum_{k=0}^{T} \varepsilon_k + \frac{\alpha_{\max}}{2\tilde{\alpha}} \sum_{k=0}^{T} \sqrt{\varepsilon_k}.\end{aligned}$$

Ponendo

$$S = \sum_{k=0}^{+\infty} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2], \quad \bar{\varepsilon} = \sum_{k=0}^{+\infty} \varepsilon_k, \quad \tilde{\varepsilon} = \sum_{k=0}^{+\infty} \sqrt{\varepsilon_k},$$

dove

- $S < +\infty$ per il *Teorema 3.2.5*;
- $\bar{\varepsilon}$ e $\tilde{\varepsilon}$ sono finite per l'ipotesi di sommabilità delle corrispondenti successioni;

e osservando che

- $\sum_{k=0}^{T} \mathbb{E}[\|\nabla f(\mathbf{w}^{(k)})\|^2] \leq S$;
- $\mathbb{E}[\|\mathbf{w}^{(k)} - \mathbf{w}^*\|^2]$ è convergente e, dunque, limitata da una costante $M > 0$;

si ottiene

$$\sum_{k=0}^{T} \mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] \leq \frac{1}{2\tilde{\alpha}}\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\alpha_{\max}^2}{2\tilde{\alpha}} S + \frac{\alpha_{\max}}{2\tilde{\alpha}} \tilde{\varepsilon} M + \frac{\alpha_{\max}^2}{2\tilde{\alpha}} \bar{\varepsilon} + \frac{\alpha_{\max}}{2\tilde{\alpha}} \tilde{\varepsilon}. \tag{13}$$

Poi, si ponga

$$\bar{\mathbf{w}}^{(T)} = \frac{1}{T+1}\sum_{k=0}^{T}\mathbf{w}^{(k)},$$

e, dividendo la disuguaglianza (13) per $(T+1)$, si applichi la *disuguaglianza di Jensen*, ovvero $\mathbb{E}[f(\bar{\mathbf{w}}^{(T)})] \leq \frac{1}{T+1}\sum_{k=0}^{T}\mathbb{E}[f(\mathbf{w}^{(k)})]$, ottenendo

$$\mathbb{E}[f(\bar{\mathbf{w}}^{(T)}) - f(\mathbf{w}^*)] \leq \frac{1}{T+1}\left(\frac{1}{2\tilde{\alpha}}\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\alpha_{\max}^2}{2\tilde{\alpha}}S + \frac{\alpha_{\max}}{2\tilde{\alpha}}\tilde{\varepsilon}M + \frac{\alpha_{\max}^2}{2\tilde{\alpha}}\bar{\varepsilon} + \frac{\alpha_{\max}}{2\tilde{\alpha}}\tilde{\varepsilon}\right).$$

Dunque, è dimostrato che $\mathbb{E}[f(\bar{\mathbf{w}}^{(T)}) - f(\mathbf{w}^*)]$ abbia un tasso di convergenza $\mathcal{O}(\frac{1}{T})$.
Poi, si sviluppi il primo membro della disuguaglianza (13). Siccome
$0 \leq f(\mathbf{w}^{(0)}) - f(\mathbf{w}^*)$, si ha

$$\begin{aligned}
\sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)})] - Tf(\mathbf{w}^*) &= \sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] \\
&\leq \sum_{k=0}^{T}\mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] \\
&\leq \sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^*)] \\
&\leq \frac{1}{2\tilde{\alpha}}\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\alpha_{\max}^2}{2\tilde{\alpha}}S + \frac{\alpha_{\max}}{2\tilde{\alpha}}\tilde{\varepsilon}M + \frac{\alpha_{\max}^2}{2\tilde{\alpha}}\bar{\varepsilon} + \frac{\alpha_{\max}}{2\tilde{\alpha}}\tilde{\varepsilon}.
\end{aligned} \tag{14}$$

Poi, attraverso la disuguaglianza (8), si determini il limite inferiore per $\sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)})]$.
Infatti, si ottiene

$$\mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^{(k+1)})] + \frac{\xi}{2}\varepsilon_k \geq 0.$$

Moltiplicando entrambi i membri per k e sommando i risultati per $k \in \{1, 2, \ldots, T\}$,

$$\begin{aligned}
0 &\leq \sum_{k=1}^{T} k\,\mathbb{E}[f(\mathbf{w}^{(k)}) - f(\mathbf{w}^{(k+1)})] + \frac{\xi}{2}\sum_{k=1}^{T}k\varepsilon_k \\
&= \sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)})] - T\mathbb{E}[f(\mathbf{w}^{(T+1)})] + \frac{\xi}{2}\sum_{k=1}^{T}k\varepsilon_k.
\end{aligned}$$

Poi, ponendo $\bar{S} = \frac{\xi}{2}\sum_{k=0}^{+\infty}k\varepsilon_k$, si ha

$$T\,\mathbb{E}[f(\mathbf{w}^{(T+1)})] \leq \sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)})] + \bar{S} \quad \Longleftrightarrow \quad T\,\mathbb{E}[f(\mathbf{w}^{(T+1)})] - \bar{S} \leq \sum_{k=1}^{T}\mathbb{E}[f(\mathbf{w}^{(k)})] \tag{15}$$

Infine, combinando i risultati (14) e (15),

$$\mathbb{E}[f(\mathbf{w}^{(T+1)}) - f(\mathbf{w}^*)] \leq \frac{1}{T}\left(\frac{1}{2\tilde{\alpha}}\|\mathbf{w}^{(0)} - \mathbf{w}^*\|^2 + \frac{\alpha_{\max}^2}{2\tilde{\alpha}}S + \frac{\alpha_{\max}}{2\tilde{\alpha}}\tilde{\varepsilon}M + \frac{\alpha_{\max}^2}{2\tilde{\alpha}}\bar{\varepsilon} + \frac{\alpha_{\max}}{2\tilde{\alpha}}\tilde{\varepsilon} + \bar{S}\right).$$

La tesi è verificata per l'arbitrarietà di T.

□

Dimostrazione del Teorema 3.2.11 [13, Teorema 2.4, Sec. 2]. Innanzitutto, si dimostri che

$$\delta c < 1.$$

Dalla definizione della costante δ e dal fatto che $\tilde{\alpha} \leq \alpha_{\max} < \frac{1}{L}$, in cui l'ultima disuguaglianza è fornita come ipotesi, si ha

$$\delta = 2\tilde{\alpha} - L\alpha_{max}^2 < \frac{2}{L} - L\left(\frac{1}{L}\right)^2 = \frac{1}{L}$$

Inoltre, il *Lemma 3.2.10* garantisce che $c \leq L$. Dunque, è facile dimostrare $\delta c < 1$.

Ora, si dimostri la disuguaglianza (3.10), ovvero

$$\mathbb{E}\big[f(\mathbf{w}^{(k+1)}) - f^*\big] \leq (1-\delta c)^k\big(f(\mathbf{w}^{(0)}) - f^*\big) + \tau\rho^k, \qquad \forall k \in \mathbb{N}_0$$

Dalla disuguaglianza (11) e dal fatto che f verifica *condizione PL*, si ottiene

$$\begin{aligned}\mathbb{E}_k[f(\mathbf{w}^{(k+1)}) - f^*] &\leq f(\mathbf{w}^{(k)}) - f^* - \frac{\delta}{2}\|\nabla f(\mathbf{w}^{(k)})\|^2 + \frac{\xi}{2}\varepsilon_k \\ &\leq f(\mathbf{w}^{(k)}) - f^* - \delta c(f(\mathbf{w}^{(k)}) - f^*) + \frac{\xi}{2}\varepsilon_k \\ &= (1-\delta c)(f(\mathbf{w}^{(k)}) - f^*) + \frac{\xi}{2}\varepsilon_k\end{aligned}$$

Considerandone il valore atteso, applicando la legge dell'aspettazione totale, si ottiene

$$\mathbb{E}[f(\mathbf{w}^{(k+1)}) - f^*] \leq (1-\delta c)\mathbb{E}[f(\mathbf{w}^{(k)}) - f^*] + \frac{\xi}{2}\varepsilon_k,$$

che, sviluppato ricorsivamente, diventa

$$\begin{aligned}\mathbb{E}[f(\mathbf{w}^{(k+1)}) - f^*] &\leq (1-\delta c)\left[(1-\delta c)\mathbb{E}[f(\mathbf{w}^{(k-1)}) - f^*] + \frac{\xi}{2}\varepsilon_{k-1}\right] + \frac{\xi}{2}\varepsilon_k \\ &\dots \\ &\leq (1-\delta c)^k (f(\mathbf{w}^{(0)}) - f^*) + \frac{\xi}{2}\sum_{j=0}^{k}(1-\delta c)^j \varepsilon_{k-j} \\ &\overset{(A)}{=} (1-\delta c)^k (f(\mathbf{w}^{(0)}) - f^*) + \frac{\xi}{2}\sum_{\bar{j}=1}^{k+1}(1-\delta c)^{\bar{j}-1}\varepsilon_{1+k-\bar{j}},\end{aligned} \tag{16}$$

dove (A) è ottenuto attraverso il cambiamento di variabile $\bar{j} := j + 1$ Infine, si osservi che il *Lemma 3.2.9* è verificato ponendo $\phi := 1-\delta c$ e $q := 1$, in quanto, come dimostrato precedentemente, $\delta c \in (0, 1)$; allora,

$$\exists \eta > 0,\ \rho \in [0,1) \quad : \quad \sum_{\bar{j}=1}^{k+1}(1-\delta c)^{\bar{j}-1}\varepsilon_{k+1-\bar{j}} \leq \eta\rho^k, \quad \forall k \in \mathbb{N}.$$

Dunque, inserendo tale disuguaglianza in (16) e ponendo $\tau := \frac{\xi}{2}\eta$, la tesi è verificata.

Dimostrazione del Teorema 4.4.4 [10, Lemma 1. Sec. 4]. Supponiamo che $\alpha_{\max}$ soddisfi la condizione di Armijo la condizione di Armijo

$$f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha\mathbf{g}_k) \leq f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \gamma\,\alpha_k\,\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^{\mathrm{T}}\mathbf{g}_k; \tag{17}$$

allora, $\alpha_k = \alpha_{\max}$ e la tesi è verificata.

Ora, supponiamo che $\alpha_{\max}$ non soddisfi questa condizione. Siccome $f_{\mathcal{N}_k}$ verifica le ipotesi del *Descent Lemma 3.2.1*, si ha

$$\begin{aligned}f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \alpha\mathbf{g}_k) &\leq f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \alpha\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T\mathbf{g}_k + \frac{\alpha^2 L_{\mathcal{N}_k}}{2}\|\mathbf{g}_k\|^2 \\ &= f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \alpha\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T\mathbf{g}_k + \frac{\alpha^2 L_{\mathcal{N}_k}}{2}(\mathbf{g}_k^T\mathbf{g}_k) \\ &= f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \alpha\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T\mathbf{g}_k + \frac{\alpha^2 L_{\mathcal{N}_k}}{2}\frac{m}{n\sum_{i\in\mathcal{N}_k} p_i^k}\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T\mathbf{g}_k \\ &= f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - \alpha\left(1 - \frac{\alpha L_{\mathcal{N}_k}}{2}\frac{m}{n\sum_{i\in\mathcal{N}_k} p_i^k}\right)\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T\mathbf{g}_k\end{aligned} \tag{18}$$

Dunque, la condizione di Armijo (17), condizione di arresto per la procedura di Line Search, è soddisfatta se

$$1 - \frac{\alpha L_{\mathcal{N}_k}}{2}\frac{m}{n\sum_{i\in\mathcal{N}_k} p_i^k} \geq \gamma \iff \alpha \leq \frac{2(1-\gamma)n\sum_{i\in\mathcal{N}_k} p_i^k}{mL_{\mathcal{N}_k}}$$

Per come costruita, la procedura di backtracking di *Algorithm 5* pone $\alpha_k = \alpha_{\max}\beta^m$, dove $m > 0$ esiste ed è il più piccolo intero per cui (17) è verificata, ovvero

$$\alpha_{\max}\beta^m \leq \frac{2(1-\gamma)n\sum_{i\in\mathcal{N}_k} p_i^k}{mL_{\mathcal{N}_k}}.$$

Dunque, essa non è soddisfatta per $\frac{\alpha_k}{\beta} = \alpha_{\max}\beta^{m-1}$ (A). In più, si ricordi la disuguaglianza (18) (B), si ottiene

$$\gamma\frac{\alpha_k}{\beta} \overset{(A)}{>} \frac{f_{\mathcal{N}_k}(\mathbf{w}^{(k)}) - f_{\mathcal{N}_k}(\mathbf{w}^{(k)} - \frac{\alpha_k}{\beta}\mathbf{g}_k)}{\nabla f_{\mathcal{N}_k}(\mathbf{w}^{(k)})^T\mathbf{g}_k} \overset{(B)}{\geq} \frac{\alpha_k}{\beta}\left(1 - \frac{\alpha_k L_{\mathcal{N}_k}}{2\beta}\frac{m}{n\sum_{i\in\mathcal{N}_k} p_i^k}\right).$$

Da questa disuguaglianza e dalla definizione di p_i^k, per cui $p_i^k \geq \frac{1-\xi_{\max}}{n}$ per ogni $i \in \{1, 2, \ldots, n\}$, per ogni $k \in \mathbb{N}$, si ottiene

$$\alpha_k > \frac{2\beta(1-\gamma)n\sum_{i\in\mathcal{N}_k} p_i^k}{L_{\mathcal{N}_k}m} \geq \frac{2\beta(1-\gamma)(1-\xi_{\max})}{L_{\mathcal{N}_k}} \geq \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)(1-\xi_{\max})}{L_{\mathcal{N}_k}}\right\},$$

Infine, siccome $L_{\mathcal{N}_k} \leq L_{\max} := \max_i L_i$, si ottiene

$$0 < \alpha_{\min} = \min\left\{\alpha_{\max}, \frac{2\beta(1-\gamma)(1-\xi_{\max})}{L_{\max}}\right\} \leq \alpha_k \leq \alpha_{\max}.$$

□

Bibliografia

[1] H&E – MyPathologyReport. URL: https://www.mypathologyreport.ca/it/tag/he/.

[2] Mesotelioma: Sintomi, prevenzione, cause, diagnosi | AIRC. URL: https://www.airc.it/cancro/informazioni-tumori/guida-ai-tumori/mesotelioma.

[3] Migrating from functorch to torch.func — PyTorch main documentation. URL: https://docs.pytorch.org/docs/stable/func.migrating.html.

[4] Opacus · Train PyTorch models with Differential Privacy · GradSampleModuleFastGradientClipping. URL: https://opacus.ai/api/grad_sample_module_fast_gradient_clipping.html.

[5] Per-sample-gradients — functorch nightly documentation. URL: https://docs.pytorch.org/functorch/stable/notebooks/per_sample_grads.html.

[6] torchvision.models.resnet — Torchvision master documentation. URL: https://docs.pytorch.org/vision/0.9/_modules/torchvision/models/resnet.html.

[7] Judith Bernett, David B. Blumenthal, Dominik G. Grimm, Florian Haselbeck, Roman Joeres, Olga V. Kalinina, and Markus List. Guiding questions to avoid data leakage in biological machine learning applications. *Nature Methods*, 21(8):1444–1453, 2024. doi:10.1038/s41592-024-02362-y.

[8] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, 60(2):223–311, 2018. _eprint: https://doi.org/10.1137/16M1080173. doi:10.1137/16M1080173.

[9] Richard H. Byrd, Gillian M. Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Math. Program.*, 134(1):127–155, 2012. doi:10.1007/s10107-012-0572-5.

[10] Filippo Camellini, Serena Crisci, Anna De Magistris, and Giorgia Franchini. A line-search based SGD algorithm with Adaptive Importance Sampling. *Journal of Computational and Applied Mathematics*, 477:117120, 2026. doi:10.1016/j.cam.2025.117120.

[11] Felix Dangel, Frederik Kunstner, and Philipp Hennig. BackPACK: Packing more into backprop, 2020. doi:10.48550/arXiv.1912.10985.

[12] Giorgia Franchini, Federica Porta, Valeria Ruggiero, Ilaria Trombini, and Luca Zanni. Learning rate selection in stochastic gradient methods based on line search strategies. *Applied Mathematics in Science and Engineering*, 31, 2023. doi:10.1080/27690911.2022.2164000.

[13] Giorgia Franchini, Federica Porta, Valeria Ruggiero, Ilaria Trombini, and Luca Zanni. A stochastic gradient method with variance control and variable learning rate for Deep Learning. *Journal of Computational and Applied Mathematics*, 451:116083, 2024. doi:10.1016/j.cam.2024.116083.

[14] Giorgia Franchini and Marco Prato. Dispense di Computational and Statistical Learning, 2021.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive computation and machine learning series. MIT Press, 2016. URL: https://www.deeplearningbook.org/.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, 2015. doi:10.48550/arXiv.1512.03385.

[17] Richard Zou Horace He. functorch: JAX-like composable function transforms for PyTorch, 2021. URL: https://github.com/pytorch/functorch.

[18] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. doi:10.1016/0893-6080(91)90009-T.

[19] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 2015. PMLR. URL: https://proceedings.mlr.press/v37/ioffe15.html.

[20] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-10 dataset (Canadian Institute for Advanced Research). URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[21] Huikang Liu, Xiaolu Wang, Jiajin Li, and Anthony Man-Cho So. Low-Cost Lipschitz-Independent Adaptive Importance Sampling of Stochastic Gradients. pages 2150–2157. IEEE Computer Society, 2021. doi:10.1109/ICPR48806.2021.9413313.

[22] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bull Math Biol*, 52(1-2):99–115; discussion 73–97, 1990. doi:10.1007/BF02478259.

[23] Kevin P. Murphy. *Machine learning - a probabilistic perspective*. Adaptive computation and machine learning series. 2012.

[24] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, ii edition, 2006. doi:10.1007/978-0-387-40065-5.

[25] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks, 2013. URL: http://arxiv.org/abs/1211.5063, doi:10.48550/arXiv.1211.5063.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit

Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[27] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951. doi:10.1214/aoms/1177729586.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2015. doi:10.48550/arXiv.1409.0575.

[29] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015. doi:10.48550/arXiv.1409.1556.

[30] Enayat Ullah, Huanyu Zhang, Will Bullock, and Ilya Mironov. Enabling Fast Gradient Clipping and Ghost Clipping in Opacus – PyTorch. URL: https://pytorch.org/blog/clipping-in-opacus/.

[31] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, ii edition, 2000. doi:DOI:10.1007/978-1-4757-3264-1_1.

[32] Ashkan Yousefpour, Igor Shilov, Alexandre Sablayrolles, Davide Testuggine, Karthik Prasad, Mani Malek, John Nguyen, Sayan Ghosh, Akash Bharadwaj, Jessica Zhao, Graham Cormode, and Ilya Mironov. Opacus: User-Friendly Differential Privacy Library in PyTorch. 2021. doi:10.48550/arXiv.2109.12298.

[33] Peilin Zhao and Tong Zhang. Stochastic Optimization with Importance Sampling for Regularized Loss Minimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*,

volume 37 of *Proceedings of Machine Learning Research*, pages 1–9, Lille, France, July 2015. PMLR. URL: https://proceedings.mlr.press/v37/zhaoa15.html.