POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea

PARAFAC e Applicazioni

Relatore
prof. Stefano Berrone
firma del relatore

. . . . . .
. . . . . .

Candidato
Davide Leo
firma del candidato

. . . . . .

Anno Accademico 2022-2023

Sommario

La *parallel factorization* o, più comunemente, *PARAFAC*, è una tecnica di decomposizione tensoriale per la rappresentazione e compressione di tensori multimodali. Il metodo può essere considerato come un'estensione della *Principal Component Analysis* (*PCA*) ed è ampiamente impiegato nell'analisi di dati multidimensionali, tra cui i risultati di test chemiometrici, psicometrici e spettroscopie a fluorescenza.
La presente tesi si propone di fornire una trattazione teorica della decomposizione PARAFAC, adottando un approccio bottom-up, al fine di evidenziarne le potenziali applicazioni nell'ambito dell'analisi dei dati. In particolare, si propone un nuovo metodo per la costruzione di una feature map tensoriale da introdurre nei modelli di deep learning, sfruttando le potenzialità della decomposizione PARAFAC.
Attraverso questo studio, si intende dimostrare come le tecniche di decomposizione tensoriale offrano soluzioni efficaci per affrontare la *curse of dimensionality*, consentendo di ottenere rappresentazioni compatte e informative di dati multidimensionali complessi, aprendo così nuove prospettive per una loro migliore comprensione e interpretazione.

Ringraziamenti

Prima di procedere con la trattazione, desidero dedicare un sentito ringraziamento a tutti coloro che mi hanno accompagnato lungo questo significativo percorso accademico.
Innanzitutto, desidero esprimere la mia gratitudine al Prof. Stefano Berrone, il mio relatore, per i Suoi consigli ed il Suo supporto durante la stesura di questo elaborato.
Un sentito ringraziamento va alla mia famiglia, la quale mi ha sostenuto lungo tutto il mio percorso universitario. Grazie per il vostro costante incoraggiamento e per la fiducia che mi avete offerto.
Desidero inoltre ringraziare di cuore il mio collega Alessandro Licciardi. Abbiamo condiviso tante esperienze, idee e progetti durante questi anni universitari. Grazie per essere stato al mio fianco nei momenti di difficoltà e per aver condiviso con me questa avventura.
Un sentito ringraziamento va al mio amico Andrea Bonafortuna, il quale è stato una preziosa fonte di incoraggiamento lungo tutto il mio percorso accademico. Grazie di cuore per i continui confronti, sfoghi e conversazioni che abbiamo condiviso.
Infine, vorrei ringraziare chiunque abbia contribuito, anche indirettamente, al mio percorso di studio e ricerca. Ogni incontro, conversazione ed esperienza vissuta hanno contribuito alla mia formazione personale ed accademica. Il vostro contributo è stato fondamentale e siete tutti parte integrante della mia esperienza universitaria.

Indice

Parte I

PARAFAC

Capitolo 1

Decomposizione poliadica canonica

1.1 Introduzione

Il calcolo tensoriale fu introdotto nella geometria differenziale verso la fine del 1800, mentre l'analisi tensoriale si sviluppò nel contesto della teoria della relatività generale di Einstein nei primi anni del 1900, con l'introduzione dell'omonima notazione degli indici. Quest'ultima permise di semplificare e sintetizzare in una forma più compatta le equazioni della fisica e, successivamente, le equazioni coinvolte nei calcoli statistici multivariati.
Dopo i primi sviluppi relativi alla teoria dei tensori da parte di matematici e fisici, la necessità di un'analisi efficiente ed efficace di dati multilineari ha portato ai lavori pionieristici di Tucker e Harshman, rispettivamente, in psicometria e fonetica, i quali hanno proposto quelle che oggi sono la decomposizione di Tucker e la fattorizzazione parallela (PARAFAC). Parallelamente Carroll e Chang svilupparono e proposero anch'essi la PARAFAC con il nome di decomposizione poliadica canonica (CANDECOMP o CPD). A partire dagli anni '90, l'analisi a più vie ha riscosso un crescente successo in chimica e, in particolare, in spettrometria e chemiometria, dimostrando un alto grado di interpretabilità dovuto alla capacità di sintetizzare ed estrarre relazioni complesse dai dati. Al giorno d'oggi, i tensori hanno un ruolo importante in molti campi applicativi per la rappresentazione e l'analisi di dati multidimensionali, non solo in chimica, ma anche nell'industria alimentare, nelle scienze ambientali, nell'elaborazione di segnali e immagini, nella computer vision, nelle neuroscienze, nelle scienze dell'informazione, nel data mining, nella pattern recognition e così via.
Nella seguente trattazione si presenterà e descriverà la PARAFAC con un approccio bottom-up, ripercorrendo lo sviluppo del modello a partire dalla decomposizione di Kruskal di tensori trilineari fino all'applicazione nel campo dell'analisi dati e del deep learning.

1.2 Nozioni preliminari

Per chiarezza di notazione, i tensori con 3 o più modi verranno indicati in carattere italico, per esempio $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, mentre le matrici ed i vettori, rispettivamente, in maiuscolo e minuscolo grassetto, ovvero $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$ e $\boldsymbol{x} \in \mathbb{R}^{I_1}$. Gli scalari, invece, saranno rappresentati in corsivo minuscolo, per esempio $a \in \mathbb{R}$.
Come introdotto precedentemente, per *tensore* si intende la generalizzazione dei concetti di vettore e matrice per spazi multimodali. Per questo motivo, è necessario estendere le definizioni di prodotto scalare e norma per quest'ultimi.

Definizione 1. *Prodotto di Hadamard*
Il *prodotto di Hadamard*, indicato con "$\odot$", è un operatore che mappa una coppia di tensori $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ in un terzo tensore $\mathcal{Z} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ tale che

$$z_{i_1,i_2,\ldots,i_N} = x_{i_1,i_2,\ldots,i_N} y_{i_1,i_2,\ldots,i_N}, \ \forall i_1, i_2, \ldots, i_N \in I_1, I_2, \ldots, I_N. \tag{1.1}$$

Il prodotto di Hadamard è anche detto *prodotto elemento a elemento*.

Definizione 2. *Prodotto scalare di due tensori*
Dati due tensori $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, l'operatore bilineare $\langle \cdot, \cdot \rangle$ tale che

$$\langle \mathcal{X}, \mathcal{Y} \rangle = \sum_{i_1,i_2,\ldots,i_N} (\mathcal{X} \odot \mathcal{Y})_{i_1,i_2,\ldots,i_N} \in \mathbb{R}, \tag{1.2}$$

è detto *prodotto scalare tra i tensori* $\mathcal{X}, \mathcal{Y}$.

Dimostrazione. Per dimostrare che $\langle \cdot, \cdot \rangle$ è un prodotto scalare è necessario verificare che siano soddisfatte le seguenti proprietà:

- $\langle \mathcal{X}, \mathcal{Y} \rangle, \ \forall \mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$;
- $\alpha \langle \mathcal{X}, \mathcal{Y} \rangle = \langle \alpha \mathcal{X}, \mathcal{Y} \rangle = \langle \mathcal{X}, \alpha \mathcal{Y} \rangle, \ \forall \mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}, \ \forall \alpha \in \mathbb{R}$;
- $\langle \mathcal{X} + \mathcal{Z}, \mathcal{Y} \rangle = \langle \mathcal{X}, \mathcal{Y} \rangle + \langle \mathcal{Z}, \mathcal{Y} \rangle, \ \forall \mathcal{X}, \mathcal{Y}.\mathcal{Z} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$:
- $\langle \mathcal{X}, \mathcal{X} \rangle \geq 0, \ \forall \mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$;
- $\langle \mathcal{X}, \mathcal{X} \rangle = 0, \iff \mathcal{X}$ è il tensore nullo;

Infatti:

- $\langle \cdot, \cdot \rangle$ è commutativo, poiché lo sono il prodotto di Hadamard e la somma tra scalari;
- sia $\alpha \in \mathbb{R}$ uno scalare, allora

$$\begin{aligned} \alpha \langle \mathcal{X}, \mathcal{Y} \rangle &= \alpha \cdot \sum_{i_1,i_2,\ldots,i_N} x_{i_1,i_2,\ldots,i_N} y_{i_1,i_2,\ldots,i_N} \\ &= \sum_{i_1,i_2,\ldots,i_N} (\alpha x_{i_1,i_2,\ldots,i_N}) y_{i_1,i_2,\ldots,i_N} \\ &= \sum_{i_1,i_2,\ldots,i_N} x_{i_1,i_2,\ldots,i_N} (\alpha y_{i_1,i_2,\ldots,i_N}), \end{aligned}$$

  dove le ultime due uguaglianze sono, rispettivamente, $\langle \alpha \mathcal{X}, \mathcal{Y} \rangle$ e $\langle \mathcal{X}, \alpha \mathcal{Y} \rangle$.

- la distributività rispetto alla somma, invece, si dimostra sviluppando $\langle \mathcal{X} + \mathcal{Z}, \mathcal{Y} \rangle$:

$$\begin{aligned}
\langle \mathcal{X} + \mathcal{Z}, \mathcal{Y} \rangle &= \sum_{i_1, i_2, \ldots, i_N} \left(x_{i_1, i_2, \ldots, i_N} + z_{i_1, i_2, \ldots, i_N} \right) y_{i_1, i_2, \ldots, i_N} \\
&= \sum_{i_1, i_2, \ldots, i_N} \left(x_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} + z_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} \right) \\
&= \sum_{i_1, i_2, \ldots, i_N} x_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} + \sum_{i_1, i_2, \ldots, i_N} z_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} \\
&= \langle \mathcal{X}, \mathcal{Y} \rangle + \langle \mathcal{Z}, \mathcal{Y} \rangle;
\end{aligned}$$

- per costruzione $\langle \mathcal{X}, \mathcal{X} \rangle \geq 0$ ed è nullo se e solo se $\mathcal{X}$ è il tensore con tutte le entrate nulle.

□

Definizione 3. *Norma di Frobenius di un tensore*
Il prodotto scalare introdotto nella Definizione 2 induce la *norma di Frobenius* per un tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, indicata con $\|\cdot\|_F$ e definita come segue:

$$\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}. \tag{1.3}$$

Dimostrazione. Essendo indotta dal prodotto scalare $\langle \cdot, \cdot \rangle$, rimane da dimostrare che $\|\mathcal{X} + \mathcal{Y}\|_F \leq \|\mathcal{X}\|_F + \|\mathcal{Y}\|_F$, con $\mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. Esplicitando il quadrato della norma si osserva che

$$\begin{aligned}
\|\mathcal{X} + \mathcal{Y}\|_F^2 &= \sum_{i_1, i_2, \ldots, i_N} \left(x_{i_1, i_2, \ldots, i_N} + y_{i_1, i_2, \ldots, i_N} \right)^2 \\
&= \sum_{i_1, i_2, \ldots, i_N} \left(x_{i_1, i_2, \ldots, i_N}^2 + 2 x_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} + y_{i_1, i_2, \ldots, i_N}^2 \right) \\
&= \sum_{i_1, i_2, \ldots, i_N} x_{i_1, i_2, \ldots, i_N}^2 + \sum_{i_1, i_2, \ldots, i_N} 2 x_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} + \sum_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N}^2 \\
&\leq \sum_{i_1, i_2, \ldots, i_N} x_{i_1, i_2, \ldots, i_N}^2 + 2 \left| \sum_{i_1, i_2, \ldots, i_N} x_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N} \right| + \sum_{i_1, i_2, \ldots, i_N} y_{i_1, i_2, \ldots, i_N}^2 \\
&= \|\mathcal{X}\|_F^2 + 2 \left| \langle \mathcal{X}, \mathcal{Y} \rangle \right| + \|\mathcal{Y}\|_F^2 .
\end{aligned}$$

Applicando così la disuguaglianza di Cauchy-Schwarz al secondo termine della somma si dimostra che vale la disuguaglianza triangolare per $\|\cdot\|_F$, infatti

$$\|\mathcal{X}\|_F^2 + 2 \left| \langle \mathcal{X}, \mathcal{Y} \rangle \right| + \|\mathcal{Y}\|_F^2 \leq \|\mathcal{X}\|_F^2 + 2 \|\mathcal{X}\|_F \|\mathcal{Y}\|_F + \|\mathcal{Y}\|_F^2 = (\|\mathcal{X}\|_F + \|\mathcal{Y}\|_F)^2,$$

da cui

$$\|\mathcal{X} - \mathcal{Y}\|_F^2 \leq (\|\mathcal{X}\|_F + \|\mathcal{Y}\|_F)^2.$$

□

Definizione 4. *Trasposizione di un tensore*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore, e si definisca la permutazione p dell'insieme di indici $\{1, 2, \ldots, N\}$ come

$$\begin{aligned} p : \{1, 2, \ldots, N\} &\longrightarrow \{1, 2, \ldots, N\} \\ i &\longmapsto p(i). \end{aligned} \tag{1.4}$$

Allora, il tensore $\mathcal{X}^{T_p} \in \mathcal{R}^{I_{p(1)} \times I_{p(2)} \times \cdots \times I_{p(N)}}$ è detto *tensore trasposto rispetto a p di* $\mathcal{X}$ e l'operazione di scambio dei modi mediante la permutazione p è detta *trasposizione*.

Esempio 1. Sia $\mathcal{X} \in \mathbb{R}^{3 \times 4 \times 2}$ il tensore

$$\begin{aligned} \mathcal{X}_{:,:,1} &= \begin{bmatrix} 20 & 11 & 27 & 14 \\ 2 & 7 & 0 & 5 \\ 6 & 12 & 29 & 18 \end{bmatrix} \\ \mathcal{X}_{:,:,2} &= \begin{bmatrix} 12 & 11 & 21 & 9 \\ 10 & 1 & 22 & 27 \\ 27 & 9 & 25 & 22 \end{bmatrix}, \end{aligned}$$

dove $I_1 = 3, I_2 = 4, I_3 = 2$. Si definisca la permutazione p come

$$\begin{aligned} p : \{1, 2, 3\} &\longrightarrow \{1, 2, 3\} \\ 1 &\longmapsto p(1) = 3 \\ 2 &\longmapsto p(2) = 2 \\ 3 &\longmapsto p(3) = 1. \end{aligned}$$

Allora, $\mathcal{X}^{T_p} \in \mathbb{R}^{2 \times 4 \times 3}$ risulta essere

$$\begin{aligned} \mathcal{X}^{T_p}_{:,:,1} &= \begin{bmatrix} 20 & 11 & 27 & 14 \\ 12 & 11 & 21 & 9 \end{bmatrix} \\ \mathcal{X}^{T_p}_{:,:,2} &= \begin{bmatrix} 2 & 7 & 0 & 5 \\ 10 & 1 & 22 & 27 \end{bmatrix} \\ \mathcal{X}^{T_p}_{:,:,3} &= \begin{bmatrix} 6 & 12 & 29 & 18 \\ 27 & 9 & 25 & 22 \end{bmatrix}. \end{aligned}$$

Si osserva facilmente come, nel caso di tensori bidimensionali, la trasposizione tensoriale coincide con la classica trasposizione matriciale. Infatti, data una matrice $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$ e la permutazione

$$\begin{aligned} p : \{1, 2\} &\longrightarrow \{1, 2\} \\ 1 &\longmapsto p(1) = 2 \\ 2 &\longmapsto p(2) = 1, \end{aligned}$$

la matrice trasposta è $\boldsymbol{X}^{T_p} \in \mathbb{R}^{I_2 \times I_1}$.

Definizione 5. *Reshape di un tensore*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore e $I = \{I_i : i \leq N,\ i, N \in \mathbb{N}\}$ l'insieme degli apici rappresentanti la dimensione di ciascuno dei modi. Si definisca analogamente l'insieme $L = \{L_i : i < M,\ i, M \in \mathbb{N}\}$, per i cui elementi vale la relazione

$$\prod_{i=1}^{N} I_i = \prod_{i=1}^{M} L_i. \tag{1.5}$$

Allora, l'operatore

$$\begin{aligned} \mathcal{R}_L :\ \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} &\longrightarrow \mathbb{R}^{L_1 \times L_2 \times \cdots \times L_M} \\ \mathcal{X} &\longmapsto \mathcal{R}_L(\mathcal{X}). \end{aligned} \tag{1.6}$$

è detto *reshaping di un tensore*.

L'operatore $\mathcal{R}_L$ modifica la dimensionalità del tensore a cui viene applicato riorganizzandone gli elementi, com'è possibile osservare nel seguente esempio.

Esempio 2. Si consideri il tensore $\mathcal{X}$ definito nell'esempio 1 e si definisca il reshaping

$$\mathcal{R}_{\{6,4\}} :\ \{3, 4, 2\} \longrightarrow \{6, 4\}.$$

Si osservi che vale la relazione (1.5), infatti

$$3 \cdot 4 \cdot 2 = 6 \cdot 4 = 24.$$

Allora,

$$\mathcal{R}_{\{6,4\}}(\mathcal{X}) = \begin{bmatrix} 20 & 11 & 27 & 14 \\ 2 & 7 & 0 & 5 \\ 6 & 12 & 29 & 18 \\ 12 & 11 & 21 & 9 \\ 10 & 1 & 22 & 27 \\ 27 & 9 & 25 & 22 \end{bmatrix}.$$

Definizione 6. *Unfolding di un tensore*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore, $I = \{I_i : i \leq N,\ i, N \in \mathbb{N}\}$ l'insieme delle dimensioni dei modi e p_i la permutazione

$$\begin{aligned} p_i :\ I &\longrightarrow I \\ 1 &\longmapsto p_i(1) = i \\ i &\longmapsto p_i(i) = 1 \\ j &\longmapsto p_i(j) = j,\ \forall j \in \{1, 2, \ldots, N\} \setminus \{1, i\}. \end{aligned} \tag{1.7}$$

Sia inoltre $\mathcal{R}_{L_i}$ l'operatore di reshaping rispetto all'insieme $L_i = \{I_i, \prod_{j \neq i} I_j\}$.
Si definisce l'*unfolding di un tensore rispetto al suo i-esimo modo* l'operatore $\mathcal{U}_i$ tale che

$$\begin{aligned} \mathcal{U}_i :\ \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} &\longrightarrow \mathbb{R}^{I_i \times \prod_{j \neq i} I_j} \\ \mathcal{X} &\longmapsto \mathcal{R}_{L_i}(\mathcal{X}^{T_{p_i}}). \end{aligned} \tag{1.8}$$

$$\mathcal{U}_1(\mathcal{X}) = \left[\begin{array}{c|c|c|c} \mathcal{X}_{:,:,1} & \mathcal{X}_{:,:,2} & \cdots & \mathcal{X}_{:,:,I_3} \end{array}\right] \in \mathbb{R}^{I_1 \times I_2}$$

Figura 1.1: Esempio di unfolding del tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ lungo il suo primo asse.

L'unfolding di un tensore è un'operazione fondamentale dal punto di vista applicativo poiché permette di costruire una rappresentazione matriciale per dati ed oggetti multimodali, rendendo così possibile l'applicazione di metodi e algoritmi comuni per le matrici. Esso può essere considerato come un processo di "srotolamento" del tensore lungo un modo specifico. Il risultato, come mostrato in Figura 1.1, è una sequenza di matrici, ognuna delle quali corrisponde ad uno "strato" del tensore originale. Ciascuna di queste matrici viene poi concatenata per colonne sequenzialmente alle precedenti.

Esempio 3. Si vuole calcolare l'unfolding $\mathcal{U}_2(\mathcal{X})$ del tensore $\mathcal{X}$ definito nell'esempio 1. Applicando la Definizione 6, si ottiene che

$$\mathcal{U}_2(\mathcal{X}) = \mathcal{R}_{L_2}(\mathcal{X}^{T_2}), \quad L_2 = \{4, 2 \cdot 3\},$$

da cui

$$U_2(\mathcal{X}) = \begin{bmatrix} 20 & 2 & 6 & 12 & 10 & 27 \\ 11 & 7 & 12 & 11 & 1 & 9 \\ 27 & 0 & 29 & 21 & 22 & 25 \\ 14 & 5 & 18 & 9 & 27 & 22 \end{bmatrix} \in \mathbb{R}^{4 \times 6}.$$

1.3 Decomposizioni tensoriali a rango ridotto

La *decomposizione poliadica canonica* è un particolare tipo di decomposizione che permette di generalizzare il concetto di rango per tensori multimodali. Affinché sia possibile fornire una corretta definizione della suddetta decomposizione, è necessaria l'introduzione di alcuni concetti fondamentali sui quali essa si basa.

1.3.1 Decomposizione di Kruskal

Definizione 7. *Prodotto poliadico*
Si consideri l'insieme di vettori $\mathcal{V} = \{\boldsymbol{v}^{(j)} \in \mathbb{R}^{I_j} : j \leq N,\ j, N \in \mathbb{N}\}$, detto *poliade*. L'operatore indicato con "$\bigcirc_{j=1}^{N}$" tale che

$$\mathcal{X} = \bigcirc_{j=1}^{N} \boldsymbol{v}^{(j)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}, \tag{1.9}$$

per il quale vale la relazione

$$X_{i_1, i_2, \ldots, i_N} = \prod_{i=1}^{N} v_{i_j}^{(j)}, \tag{1.10}$$

è detto *prodotto poliadico*.
In particolare, nel caso in cui il prodotto fosse ristretto a solo due vettori $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}$, si utilizza il simbolo "$\circ$" ed è detto *prodotto diadico*, il quale coincide con la definizione del classico prodotto esterno vettoriale.

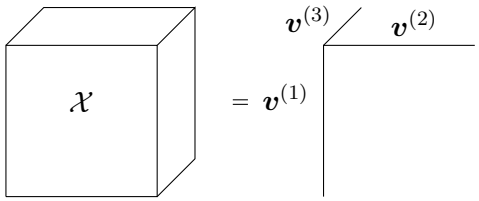

Figura 1.2: Rappresentazione del tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ ottenuto tramite un prodotto triadico.

Il tensore costruito nella precedente definizione ha rango 1 poiché è il risultato del prodotto poliadico degli N vettori di una poliade, i quali, non sono che tensori di rango unitario anch'essi.
In generale, è possibile costruire tensori come combinazioni lineari di prodotti poliadici, estendendo la definizione di rango per vettori e matrici multidimensionali come segue.

Definizione 8. *Rango di una combinazione lineare di prodotti poliadici*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, tale che

$$\mathcal{X} = \sum_{i=1}^{M} (\lambda_i \bigcirc_{j=1}^{N} \boldsymbol{v}^{(i,j)}), \tag{1.11}$$

con $\boldsymbol{\lambda} \in \mathbb{R}^M$ vettore di coefficienti reali e $\boldsymbol{v}^{(i,j)} \in \mathbb{R}^{I_j}$ $j-$esimo vettore dell'$i-$esima poliade. Inoltre, si supponga che gli i vettori appartenenti ad uno stesso modo siano linearmente indipendenti. Allora, il *rango* di un tensore costruito come combinazione lineare di M prodotti poliadici è esattamente uguale ad M.

Il primo risultato relativo alla decomposizione di un tensore mediante combinazione lineare di prodotti poliadici è il seguente.

Definizione 9. *Decomposizione di Kruskal*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ un tensore di rango R rappresentabile come

$$\mathcal{X} = \sum_{r=1}^{R} \lambda_r \cdot (\boldsymbol{v}^{(r,1)} \circ \boldsymbol{v}^{(r,2)} \circ \boldsymbol{v}^{(r,3)}) \tag{1.12}$$

e siano i vettori $\boldsymbol{v}^{(r,j)}$ di norma unitaria e linearmente indipendenti all'interno di ciascun modo.
Allora, la rappresentazione del tensore mediante combinazione lineare di prodotti triadici è detta *decomposizione di Kruskal* di $\mathcal{X}$.

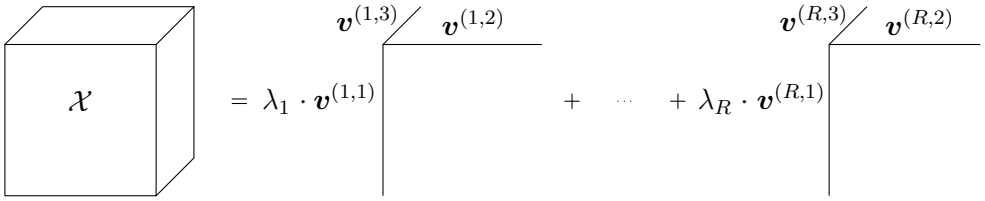

Figura 1.3: Rappresentazione della decomposizione di Kruskal del tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

L'importanza di tale decomposizione è legata al fatto che è unica sotto condizioni poco stringenti, come enunciato nel seguente teorema.

Teorema. *Unicità della decomposizione di Kruskal*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ un tensore di rango R e $\sum_{r=1}^{R^*}(\lambda_r \boldsymbol{v}_1^{(r)} \circ \boldsymbol{v}_2^{(r)} \circ \boldsymbol{v}_3^{(r)})$ la sua decomposizione di Kruskal sviluppata fino all'R^*-esima poliade con i coefficienti λ_r tali che $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{R^*}$. Siano inoltre K_1, K_2, K_3 i ranghi di ciascun modo, ovvero il rispettivo numero di vettori linearmente indipendenti.
Allora tale decomposizione è unica a se vale la seguente condizione:

$$K_1 + K_2 + K_3 \geq 2R^* + 2. \tag{1.13}$$

Se i vincoli relativi all'ordinamento dei coefficienti λ_r e alla norma dei vettori delle triadi fossero rilassati, la decomposizione sarebbe comunque unica a meno di riscalamenti e permutazioni delle componenti. Infatti, nelle applicazioni, al fine di semplificare il problema di ottimizzazione da risolvere per approssimare tale decomposizione, i coefficienti vengono spesso fissati o si assume che siano noti, in modo da calcolarne le rispettive triadi.

1.3.2 Generalizzazione della decomposizione di Kruskal per tensori N-modali

I risultati ottenuti da Kruskal, però, sono limitati nelle applicazioni poiché ristretti a tensori tridimensionali. Sotto ulteriori ipotesi, Harshman e, parallelamente, Carroll e Chang estesero la decomposizione a tensori N-modali, definendola *decomposizione poliadica canonica*.

Definizione 10. *Decomposizione poliadica canonica* (*CPD*)
Siano $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore di rango R scrivibile come combinazione lineare di prodotti poliadici

$$\mathcal{X} = \sum_{r=1}^{R}(\lambda_r \bigcirc_{j=1}^{N} \boldsymbol{v}^{(r,j)}) \tag{1.14}$$

e siano i vettori $\boldsymbol{v}^{(r,j)}$ di norma unitaria e ortogonali all'interno di ciascun modo. Analogamente alla definzione 9, tale rappresentazione è detta *decomposizione poliadica canonica* di $\mathcal{X}$.

Assumendo ulteriori ipotesi, anche la CPD, come la decomposizione di Kruskal, è unica, permettendo così di rappresentare in maniera univoca *canonica* tensori risultanti dalla combinazione lineare di poliadi, come avviene per i vettori, descrivibili univocamente una volta fissata la base.

Definizione 11. *Teorema di unicità della CPD*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore e $\sum_{r=1}^{R}(\lambda_r \bigcirc_{j=1}^{N} \boldsymbol{v}^{(r,j)})$ la sua decomposizione poliadica canonica. Allora, se valgono le ipotesi introdotte nella Definizione 10 e se i prodotti poliadici sono ordinati per λ_r decrescenti, tale decomposizione è unica.

Inoltre, conoscendo la decomposizione poliadica canonica di un tensore è possibile valutarne le proprietà sfruttando la sinteticità della rappresentazione. Un esempio immediato può essere il calcolo del rango, il quale, applicando la Definizione 8, risulta essere esattamente il numero di poliadi. Anche il calcolo della norma, se nota la decomposizione, risulta più semplice ed efficiente da calcolare, ma è prima necessario introdurre alcune proprietà della CPD.

Proposizione 1. Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ tale che $\mathcal{X} = \bigcirc_{j=1}^{N} \boldsymbol{v}^{(j)}$, con $\boldsymbol{v}^{(j)}$ vettori di norma unitaria, allora, $\|\mathcal{X}\|_F = 1$.

Dimostrazione. Riprendendo l'equazione (1.10), è possibile esplicitare il quadrato della norma di Frobenius di $\mathcal{X}$ in funzione dei vettori $\boldsymbol{v}^{(j)}$, ottenendo

$$\begin{aligned} \|\mathcal{X}\|_F^2 &= \sum_{i_1, i_2, \ldots, i_N} \left(\prod_{j=1}^{N} v_{i_j}^{(j)} \right)^2 \\ &= \left(\sum_{i_1} \left(v_{i_1}^{(1)} \right)^2 \right) \left(\sum_{i_2} \left(v_{i_2}^{(2)} \right)^2 \right) \cdots \left(\sum_{i_N} \left(v_{i_N}^{(N)} \right)^2 \right) \\ &= \|\boldsymbol{v}^{(1)}\|_F^2 \|\boldsymbol{v}^{(1)}\|_F^2 \ldots \|\boldsymbol{v}^{(1)}\|_F^2 = 1. \end{aligned} \tag{1.15}$$

Di conseguenza, $\|\mathcal{X}\|_F = \sqrt{1} = 1$. □

Proposizione 2. Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore di rango R rappresentabile mediante la sua decomposizione poliadica canonica. Allora $\|\mathcal{X}\|_F = \|\boldsymbol{\lambda}\|_F$, con $\boldsymbol{\lambda} \in \mathbb{R}^R$ vettore dei coefficienti moltiplicanti ciascuna poliade della decomposizione.

Dimostrazione. Si sviluppi il quadrato di $\|\mathcal{X}\|_F$ sostituendo l'equazione (1.9) ad $\mathcal{X}$.

$$\|\mathcal{X}\|_F^2 = \|\sum_{r=1}^{R}(\lambda_r \bigcirc_{j=1}^{N} \boldsymbol{v}^{(r,j)})\|_F^2 = \langle \sum_{r=1}^{R}(\lambda_r \bigcirc_{j=1}^{N} \boldsymbol{v}^{(r,j)}), \sum_{r=1}^{R}(\lambda_r \bigcirc_{j=1}^{N} \boldsymbol{v}^{(r,j)}) \rangle \tag{1.16}$$

Per semplificare la notazione, si indichi con $\mathcal{V}^{(r)}$ l'$r-$esimo prodotto poliadico, quindi

$$\begin{aligned} \|\mathcal{X}\|_F^2 &= \langle \sum_{r_1=1}^{R} \lambda_r \mathcal{V}^{(r_1)}, \sum_{r_2=1}^{R} \lambda_r \mathcal{V}^{(r_2)} \rangle \\ &= \sum_{r_1, r_2} \lambda_{r_1} \lambda_{r_2} \langle \mathcal{V}^{(r_1)}, \mathcal{V}^{(r_2)} \rangle \\ &= \sum_{r=1}^{R} \lambda_r^2 \langle \mathcal{V}^{(r)}, \mathcal{V}^{(r)} \rangle + \sum_{r_1 \neq r_2} \lambda_{r_1} \lambda_{r_2} \langle \mathcal{V}^{(r_1)}, \mathcal{V}^{(r_2)} \rangle \end{aligned} \tag{1.17}$$

Si osservi che i termini della prima sommatoria sono nient'altro che il quadrato dei coefficienti λ_r moltiplicati per la norma dei rispettivi prodotti poliadici, i quali, applicando la Proposizione 1, hanno norma unitaria. Di conseguenza, quest'ultimo risulta essere il quadrato della norma di Frobenius del vettore dei coefficienti, infatti

$$\sum_{r=1}^{R} \lambda_r^2 \langle \mathcal{V}^{(r)}, \mathcal{V}^{(r)} \rangle = \sum_{r=1}^{R} \lambda_r^2 \langle \mathcal{V}^{(r)}, \mathcal{V}^{(r)} \rangle = \sum_{r=1}^{R} \lambda_r^2 = \|\boldsymbol{\lambda}\|_F^2 .$$

Siccome i vettori delle poliadi di uno stesso modo sono tra di essi ortogonali, il prodotto $\langle \mathcal{V}^{(r_1)}, \mathcal{V}^{(r_2)} \rangle$ per $r_1 \neq r_2$ è nullo, dimostrando così la proposizione. □

Nella trattazione tornerà utile la seguente definizione, che permette di scrivere in maniera più chiara e sintetica l'equazione (1.14).

Definizione 12. *Notazione matriciale della CPD*
Si consideri il tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ di rango R rappresentabile mediante decomposizione poliadica canonica. Si introducano le matrici $\boldsymbol{V}^{(j)} \in \mathbb{R}^{I_j \times R}$ definite come

$$\boldsymbol{V}^{(j)} = \left[\boldsymbol{v}^{(1,j)} \ \boldsymbol{v}^{(2,j)} \ \dots \ \boldsymbol{v}^{(R,j)} \right], \ \forall j = 1,2,\dots,N, \tag{1.18}$$

i cui vettori colonna $\boldsymbol{v}^{(r,j)} \in \mathbb{R}^{I_j}$ sono di norma unitaria appartenenti al $j-$esimo modo della decomposizione. Allora $\mathcal{X}$ può essere indicato con la forma matriciale contratta

$$\mathcal{X} = \left\| \boldsymbol{\lambda}; \boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}, \dots, \boldsymbol{V}^{(N)} \right\|, \tag{1.19}$$

con $\boldsymbol{\lambda} \in \mathbb{R}^R$ vettore dei coefficienti.

Osservazione. Siccome gli R vettori colonna $\boldsymbol{v}^{(i,j)}$ di ciascuna matrice $\boldsymbol{V}^{(j)}$ sono tra di essi ortogonali, anche il rango delle matrici $\boldsymbol{V}^{(j)}$ risulta essere R.

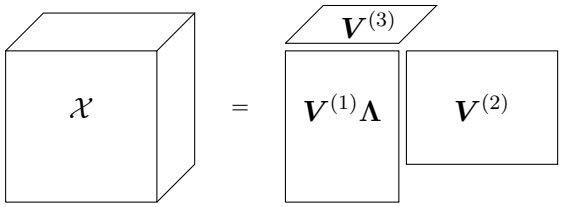

Figura 1.4: Notazione matriciale della decomposizione del tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, indicando con $\boldsymbol{\Lambda} \in \mathbb{R}^{R \times R}$ la matrice diagonale i cui elementi non nulli sono i pesi λ_r.

1.4 CPD e decomposizione SVD

Teorema. *Decomposizione ai valori singolari*
Sia $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$ una matrice, allora esistono tre matrici $\boldsymbol{U} \in \mathbb{R}^{I_1 \times I_1}$, $\boldsymbol{\Sigma} \in \mathbb{R}^{I_1 \times I_2}$, $\boldsymbol{V} \in \mathbb{R}^{I_2 \times I_2}$ tali che

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T, \tag{1.20}$$

e:

- $\boldsymbol{U}, \boldsymbol{V}$ sono ortogonali;
- $\boldsymbol{\Sigma}$ è una matrice diagonale i cui valori diagonali sono non negativi.

Tale decomposizione è detta *ai valori singolari* (*SVD*), identificando con il termine *valori singolari* gli elementi diagonali della matrice $\boldsymbol{\Sigma}$, mentre le colonne delle matrici $\boldsymbol{U}, \boldsymbol{V}$ sono dette, rispettivamente, *vettori singolari sinistri* e *destri*.

La decomposizione SVD gode delle seguenti proprietà:

- i valori singolari di $\boldsymbol{X}$ coincidono con gli autovalori della matrici $\boldsymbol{X}^T\boldsymbol{X}$ e $\boldsymbol{X}\boldsymbol{X}^T$;
- le matrici $\boldsymbol{U}, \boldsymbol{V}$ sono, rispettivamente, le matrici degli autovettori di $\boldsymbol{X}^T\boldsymbol{X}$ e di $\boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}}$;
- il numero di valori singolari non nulli coincide con il rango di $\boldsymbol{X}$.

Si osserva che, nel caso in cui $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$, la decomposizione $\hat{\boldsymbol{X}}$ si riduce alla somma di prodotti diadici

$$\hat{\boldsymbol{X}} = \sum_{r=1}^{R} \lambda_r (\boldsymbol{v}^{(r,1)} \circ (\boldsymbol{v}^{(r,2)})^T), \tag{1.21}$$

riscrivibile nella forma matriciale

$$\hat{\boldsymbol{X}} = \boldsymbol{V}^{(1)} \boldsymbol{\Lambda} (\boldsymbol{V}^{(2)})^T, \tag{1.22}$$

dove $\boldsymbol{\Lambda} \in \mathbb{R}^{R \times R}$ è una matrice diagonale i cui elementi non nulli sono i coefficienti λ_r.
Se il rango di $\boldsymbol{X}$ è esattamente R e se le matrici $\boldsymbol{V}^{(i)},\ i \in \{1, 2\}$ sono matrici ortonormali, allora la decomposizione poliadica canonica è unica per il teorema 1.3.1 e coincide necessariamente con la decomposizione ai valori singolari. In questo caso particolare, i coefficienti λ_r coincidono con i valori singolari e, siccome ogni matrice ammette una decomposizione SVD, deve valere necessariamente l'uguaglianza $\boldsymbol{X} = \hat{\boldsymbol{X}}$.
La CPD può essere quindi considerata come un'estensione della decomposizione ai valori singolari, ereditandone direttamente i numerosi campi di applicazione. Un esempio possono essere i metodi bilineari utilizzati nell'analisi dati, tra cui la *Principal Component Analysis* o la *Linear Discriminant Analysis*, nelle quali l'SVD permette di ridurre la dimensionalità dei dataset di partenza conservandone la maggior parte dell'informazione spiegata. La decomposizione poliadica canonica può essere introdotta negli algoritmi citati per analizzare dataset multilineari, sintetizzandone le features ed estraendone le principali relazioni.

Capitolo 2

Decomposizione PARAFAC

2.1 Decomposizione CPD per problemi ad alta dimensionalità

In generale, la decomposizione CPD permette di sintetizzare, analizzare e visualizzare dati multivariati senza dover ricorrere ad una loro vettorizzazione monodimensionale (*flattening*), garantendone comunque un'accurata rappresentazione ed una facile interpretazione. Un esempio possono essere i dati raccolti durante test ed esperimenti chimici tra cui la spettrometria di massa, il cui obiettivo è quello di analizzare lo spettro di massa di un determinato campione al fine di distinguerne gli elementi che lo compongono. In genere, in quest'ultimo caso, i risultati delle spettroscopie vengono collezionati sotto forma di tensori multimodali, per cui l'applicazione della decomposizione CPD risulta naturale. In particolare, in funzione della complessità del campione in analisi, si assume che quest'ultimo sia composto da R elementi e che il tensore che ne rappresenta lo spettro sia di rango R, dove ciascuna poliade caratterizza il contributo del rispettivo elemento della decomposizione, permettendone quindi l'identificazione.
Tuttavia, non sempre è possibile conoscere a priori il numero di elementi di cui il campione è composto, calcolare esattamente il rango di un tensore in modo efficiente e, soprattutto, non è sempre possibile assumere che il tensore sia il risultato di una combinazione lineare di prodotti poliadici per via della naturale stocasticità dei dati.
Per questo motivo, è necessario introdurre il concetto di $\epsilon-$vicinanza, al fine di poter generalizzare i risultati ottenuti nella precedente sezione.

Definizione 13. *$\epsilon-$vicinanza*
Due tensori $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_e \times \cdots \times I_N}$ per i quali vale la relazione

$$\|\mathcal{X} - \mathcal{Y}\|_F = \sqrt{\sum_{j_1, j_2, \ldots, j_N} (x_{j_1, j_2, \ldots, j_N} - y_{j_1, j_2, \ldots, j_N})^2} \leq \epsilon, \ \epsilon \geq 0, \tag{2.1}$$

sono detti *$\epsilon-$vicini*, ovvero $\mathcal{X} \overset{\epsilon}{\approx} \mathcal{Y}$.

Due tensori delle stesse dimensioni, quindi, si dicono $\epsilon-$vicini se la norma di Frobenius della loro differenza componente a componente è minore di una certa soglia fissata ϵ.

L'estensione della definizione di decomposizione poliadica canonica ad un qualsiasi tensore multimodale è ora intuitiva e viene formalizzata dal seguente enunciato.

Definizione 14. *$\epsilon-CPD$*
Siano $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore di rango R e $\hat{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore rappresentabile mediante decomposizione poliadica canonica, anch'esso di rango R. Se $\mathcal{X} \overset{\epsilon}{\approx} \hat{\mathcal{X}}$, allora $\hat{\mathcal{X}}$ è detta *decomposizione poliadica canonica $\epsilon-$vicina* di $\mathcal{X}$ ($\epsilon-CPD$).

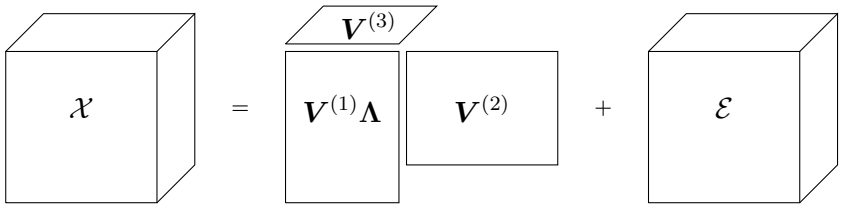

Figura 2.1: Rappresentazione del tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ in funzione della sua decomposizione $\epsilon-$CPD.

Bhaskara, Charikary e Vijayaraghavanz hanno esteso i risultati di Kruskal, Harshman e di Carroll e Chang anche per decomposizioni $\epsilon-$vicine [1], per le quali, sotto ulteriori ipotesi, ne è verificata l'unicità se vale la nuova condizione

$$\sum_{i=1}^{N} K_i \geq 2R + N - 1. \tag{2.2}$$

Inoltre, anche la Proposizione 2 può essere estesa come segue.

Proposizione 3. Siano $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore di rango R e $\hat{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ la sua unica decomposizione poliadica canonica $\epsilon-$vicina, con $\boldsymbol{\lambda} \in \mathbb{R}^R$ vettore dei coefficienti di $\hat{\mathcal{X}}$. Allora, la norma di $\boldsymbol{\lambda}$ è limitata e vale la seguente relazione:

$$\|\boldsymbol{\lambda}\|_F \in \left[\|\mathcal{X}\|_F - \epsilon, \|\mathcal{X}\|_F + \epsilon\right]. \tag{2.3}$$

Dimostrazione. Richiamando la Definizione 14, vale la relazione

$$\|\mathcal{X} - \hat{\mathcal{X}}\|_F \leq \epsilon, \ \epsilon \geq 0. \tag{2.4}$$

Applicando la disuguaglianza triangolare inversa al termine a sinistra della disequazione, si ottiene

$$|\|\mathcal{X}\|_F - \|\hat{\mathcal{X}}\|_F| \leq \|\mathcal{X} - \hat{\mathcal{X}}\|_F \leq \epsilon,$$

per cui

$$|\|\mathcal{X}\|_F - \|\hat{\mathcal{X}}\|_F| \leq \epsilon. \tag{2.5}$$

Applicando la definizione di modulo,

$$\|\mathcal{X}\|_F - \epsilon \leq \|\hat{\mathcal{X}}\|_F \leq \|\mathcal{X}\|_F + \epsilon,$$

e, siccome $\|\hat{\mathcal{X}}\|_F = \|\boldsymbol{\lambda}\|_F$ si verifica la Proposizione:

$$\|\mathcal{X}\|_F - \epsilon \leq \|\boldsymbol{\lambda}\|_F \leq \|\mathcal{X}\|_F + \epsilon, \tag{2.6}$$

□

Nelle applicazioni, quindi, l'obiettivo è quello di calcolare la decomposizione $\epsilon-$CPD $\hat{\mathcal{X}}$ del tensore dei dati $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$.
Supponendo noto il rango R di tale tensore, allora $\hat{\mathcal{X}}$ è soluzione del seguente problema di minimo.

$$\begin{aligned}
\min_{\hat{\mathcal{X}}} \quad & \|\mathcal{X} - \hat{\mathcal{X}}\|_F \\
\text{s.t.} \quad & \hat{\mathcal{X}} = \left\| \boldsymbol{\lambda}; \boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}, \ldots, \boldsymbol{V}^{(N)} \right\| \\
& \|\boldsymbol{V}^{(j)}_{:,r}\|_F = 1, \quad j = 1, \ldots, N \\
& \qquad\qquad\qquad r = 1, \ldots, R \\
& \langle (\boldsymbol{V}^{(j)}_{:,r_1})^T, \boldsymbol{V}^{(j)}_{:,r_2} \rangle = 0, \quad r_1 \neq r_2, \quad j = 1, \ldots, N \\
& \|\mathcal{X}\|_F \leq \|\boldsymbol{\lambda}\|_F \leq \|\mathcal{X}\|_F + \epsilon \\
& \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_R \\
& \boldsymbol{V}^{(j)} \in \mathbb{R}^{I_j \times R}, \quad j = 1, \ldots, N \\
& \boldsymbol{\lambda} \in \mathbb{R}^R.
\end{aligned} \tag{2.7}$$

Siccome la decomposizione ϵ-CPD viene solitamente applicata per l'analisi e la compressione di dati multivariati di elevata dimensionalità, la risoluzione del problema (2.7) risulta spesso molto complessa e, soprattutto, onerosa dal punto di vista computazionale. Inoltre, come precedentemente indicato, la determinazione del rango di un tensore prima del calcolo della decomposizione può risultare un'operazione non banale e spesso computazionalmente onerosa. Infatti, la complessità computazionale del calcolo del rango di un tensore è classificata come NP-hard, il che significa che non esistono attualmente algoritmi efficienti per risolvere il problema in tempi ragionevoli all'aumentare della dimensionalità. Per semplificare il calcolo di $\hat{\mathcal{X}}$, quindi, è necessario rilassare alcuni vincoli e considerare R non più come rango esatto del tensore, ma più semplicemente come il numero di componenti della decomposizione. Inoltre, venendo spesso applicata a tensori di grandi dimensioni per la loro rappresentazione a rango ridotto ed essendo poco stringenti le ipotesi che ne garantiscono l'unicità, le soluzioni ottenute con il rilassamento dei vincoli sono uniche anch'esse a meno di eventuali permutazioni delle poliadi e riscalamenti.
La riformulazione del problema (2.7) più comune in letteratura è la seguente:

$$\begin{aligned}
\min_{\hat{\mathcal{X}}} \quad & \|\mathcal{X} - \hat{\mathcal{X}}\|_F \\
\text{s.t.} \quad & \hat{\mathcal{X}} = \left\| \boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}, \ldots, \boldsymbol{V}^{(N)} \right\| \\
& \boldsymbol{V}^{(j)} \in \mathbb{R}^{I_j \times R}, \quad j = 1, \ldots, N,
\end{aligned} \tag{2.8}$$

dove $\hat{\mathcal{X}}$ è detta *decomposizione PARAFAC* (*decomposizione ai fattori paralleli* o *fattorizzazione parallela*).
In letteratura si utilizza il termine PARAFAC come sinonimo di decomposizione poliadica canonica, ma in questa trattazione si vogliono distinguere i risultati teorici da quello che ne è l'approccio pratico. Infatti, le decomposizioni di Kruskal e, più in generale, di Harshman sono applicabili a tensori che, per assunzione, sono esprimibili come combinazioni lineari di prodotti poliadici. Differentemente, la PARAFAC, ottenuta dal rilassamento dei vincoli del problema di minimo, è una generalizzazione non solo della decomposizione CPD, ma anche della decomposizione $\epsilon-$CPD, risultando quindi uno strumento più versatile ed efficiente.

2.1.1 Calcolo di $\hat{\mathcal{X}}$ e metodo ALS

La funzione obiettivo del problema di minimo (2.8) è la norma di Frobenius della differenza tra il tensore da decomporre e la sua decomposizione PARAFAC. Riprendendo la Definizione 3, essa può essere esplicitata come segue:

$$\|\mathcal{X} - \hat{\mathcal{X}}\|_F = \sqrt{\sum_{i_1,i_2,\dots,i_N} (x_{i_1,i_2,\dots,i_N} - \hat{x}_{i_1,i_2,\dots,i_N})^2}. \tag{2.9}$$

Essendo la radice quadrata una funzione monotona strettamente crescente, è sufficiente minimizzare l'argomento della radice, ottenendo così una somma di quadrati convessa nella variabili $\hat{x}_{i_1,i_2,\dots,i_N}$. La convessità della funzione obiettivo viene preservata anche per trasformazioni lineari, per cui la si moltiplica per il fattore $\frac{1}{2N_{\text{tot}}}$, con $N_{\text{tot}} = \prod_{j=1}^{N} I_j$ numero di elementi di $\mathcal{X}$.
Il problema (2.8), quindi, può essere riscritto come

$$\begin{aligned} \min_{\hat{\mathcal{X}}} \quad & \frac{1}{2} \cdot \text{MSE}(\mathcal{X}, \hat{\mathcal{X}}) \\ \text{s.t.} \quad & \hat{\mathcal{X}} = \left\| \boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}, \dots, \boldsymbol{V}^{(N)} \right\| \\ & \boldsymbol{V}^{(j)} \in \mathbb{R}^{I_j \times R}, \quad j = 1, \dots, N, \end{aligned} \tag{2.10}$$

indicando con *MSE* l'*errore quadratico medio* di approssimazione, ovvero il quadrato della norma del tensore dei residui $\mathcal{E} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ moltiplicata per $\frac{1}{N_{\text{tot}}}$.
Richiamando l'equazione (1.10), l'MSE risulta essere

$$\text{MSE}(\mathcal{X}, \hat{\mathcal{X}}) = \frac{1}{2N_{\text{tot}}} \sum_{i_1,i_2,\dots,i_N} (x_{i_1,i_2,\dots,i_N} - \sum_{r=1}^{R} (\prod_{j=1}^{N} v_{i_j,r}^{(j)}))^2. \tag{2.11}$$

La ricerca del minimo della funzione obiettivo è quindi analago alla minimizzazione dell'errore di approssimazione medio in funzione delle singole variabili $v_{(i_j,r)}^{(j)}$ le quali, per via dei prodotti poliadici, introducono non linearità negli argomenti degli elevamenti a potenza. Tuttavia, si osserva facilmente che, se la variabile rispetto alla quale minimizzare l'equazione (2.11) fosse una sola, allora il problema sarebbe convesso rispetto ad essa.

La verifica è rapida ed è sufficiente calcolare la derivata seconda dell'MSE$(\mathcal{X}, \hat{\mathcal{X}})$ rispetto a $v^{(J^*)}_{I^*,R^*}$ con $I^* = i_{J^*}$, J^* ed R^* indici fissati:

$$\begin{aligned}\frac{\partial^2 \text{MSE}(\mathcal{X}, \hat{\mathcal{X}})}{\partial (v^{(J^*)}_{I^*,R^*})^2} &= \frac{\partial^2}{\partial (v^{(J^*)}_{I^*,R^*})^2}\left[\frac{1}{2N_{\text{tot}}} \sum_{i_1,i_2,\dots,i_N} (x_{i_1,i_2,\dots,i_N} - \sum_{r=1}^{R}(\prod_{j=1}^{N} v^{(j)}_{i_j,r}))^2\right] \\ &= \frac{\partial}{\partial v^{(J^*)}_{I^*,R^*}}\left[\frac{1}{N_{\text{tot}}} \sum_{i_j, j\neq J^*} (\sum_{r=1}^{R}(\prod_{j=1}^{N} v^{(j)}_{i_j,r}) - x_{i_1,i_2,\dots,i_N})(\prod_{j\neq J^*} v^{(j)}_{i_j,R^*})\right] \\ &= \frac{1}{N_{\text{tot}}}(\prod_{j\neq J^*} v^{(j)}_{i_j,R^*})^2 \geq 0.\end{aligned} \tag{2.12}$$

Essendo la funzione convessa, quindi, la soluzione dell'equazione lineare

$$\frac{1}{N_{\text{tot}}} \sum_{i_j, j\neq J^*} (\sum_{r=1}^{R}(\prod_{j=1}^{N} v^{(j)}_{i_j,r}) - x_{i_1,i_2,\dots,i_N})(\prod_{j\neq J^*} v^{(j)}_{i_j,R^*}) = 0, \tag{2.13}$$

nella variabile $v^{(J^*)}_{I^*,R^*}$, per I^*, J^*, R^* fissati, è l'unico punto di minimo.
Applicando lo stesso procedimento alle restanti entrate della matrice $\boldsymbol{V}^{(J^*)}$ si ottiene un sistema di $N_{J^*}R$, equazioni in $N_{J^*}R$ incognite.
Senza perdita di generalità, si osserva che la norma indotta dal prodotto scalare (2) è invariante per reshaping dei tensori per i quali viene calcolata. Infatti, per esempio,

$$\|X\|_F = \sqrt{\sum_{i_1,i_2,\dots,i_N} x^2_{i_1,i_2,\dots,i_N}} = \|\text{vec}(\mathcal{X})\|_F, \tag{2.14}$$

indicando con vec$(\mathcal{X})$ la vettorizzazione del tensore, ovvero

$$\text{vec}(\mathcal{X}) = \mathcal{R}_{\{N_{\text{tot}}\}}(\mathcal{X}) \in \mathbb{R}^{N_{\text{tot}}}. \tag{2.15}$$

Per questo motivo, è possibile generalizzare l'equazione (2.11) non più in funzione delle singole entrate $v^{(j)}_{i_j,r}$, ma in un sistema di equazioni matriciali nelle variabili $\boldsymbol{V}^{(j)}$.
Siccome l'operatore $\|\cdot\|_F$ è invariante per reshaping, l'MSE è scrivibile in funzione delle matrici della decomposizione. Sia quindi

$$\boldsymbol{Z^{(J^*)}} = \mathcal{U}_{J^*}\left(\left\|\boldsymbol{V}^{(j_1)}, \boldsymbol{V}^{(j_2)}, \dots, \boldsymbol{V}^{(j_{N-1})}\right\|\right) \in \mathbb{R}^{R\times \prod_{j\neq J^*} I_j}, \quad j_i \neq J^* \ \forall i = 1, \dots, N-1 \tag{2.16}$$

l'unfolding della decomposizione PARAFAC privata della J^*-esima matrice, la funzione obiettivo diventa

$$\begin{aligned}\text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J^*)}) &= \frac{1}{2N_{\text{tot}}} \| \mathcal{U}_{J^*}(\mathcal{X}) - \boldsymbol{V}^{(J^*)}\boldsymbol{Z}^{(J^*)} \|^2_F \\ &= \frac{1}{2N_{\text{tot}}} \| \text{vec}(\mathcal{U}_{J^*}(\mathcal{X})) - \text{vec}(\boldsymbol{V}^{(J^*)}\boldsymbol{Z}^{(J^*)}) \|^2_F.\end{aligned} \tag{2.17}$$

Di conseguenza, siccome il problema matriciale è analogo al problema vettorizzato, la notazione

$$\nabla_{\boldsymbol{V}^{(J^*)}}\text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J^*)}) \in \mathbb{R}^{N_{J^*} \times R} \tag{2.18}$$

indicherà la matrice delle derivate prime dell'MSE rispetto alle componenti di $\boldsymbol{V}^{J^*}$, ovvero tale che

$$\nabla_{\boldsymbol{V}^{(J^*)}}\text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J^*)})_{i,r} = \frac{\partial \text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J^*)})}{\partial v_{i,r}^{(J^*)}}, \tag{2.19}$$

mentre la matrice delle derivate seconde sarà indicata come

$$\boldsymbol{H}_{\boldsymbol{V}^{(J^*)}}\text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J)^*}) \in \mathbb{R}^{N_{J^*}R \times N_{J^*}R}. \tag{2.20}$$

Quest'ultima è una matrice diagonale semidefinita positiva poiché

$$\begin{aligned} &\frac{\partial^2 \text{MSE}(\mathcal{X}, \hat{\mathcal{X}})}{\partial (v_{I^*,R^*}^{(J^*)})^2} = \frac{1}{N_{\text{tot}}}\Big(\prod_{j \neq J^*} v_{i_j,R^*}^{(j)}\Big)^2 \geq 0 \\ &\frac{\partial^2 \text{MSE}(\mathcal{X}, \hat{\mathcal{X}})}{\partial v_{I_1^*,R_1^*}^{(J^*)} \partial v_{I_2^*,R_2^*}^{(J^*)}} = 0, \qquad\qquad \forall I_1^* \neq I_2^*,\ R_1^* \neq R_2^*, \end{aligned} \tag{2.21}$$

per cui la funzione obiettivo è convessa rispetto alla matrice $\boldsymbol{V}^{(J^*)}$.
La soluzione del problema di minimo vincolato a $\boldsymbol{V}^{(J^*)}$ esiste e può essere calcolata risolvendo il sistema lineare

$$\nabla_{\boldsymbol{V}^{(J^*)}}\text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J^*)}) = 0 \tag{2.22}$$

il quale può essere riscritto nella seguente forma esplicita:

$$[\boldsymbol{Z}^{(J^*)}(\boldsymbol{Z}^{(J^*)})^T]^T(\boldsymbol{V}^{(J^*)})^T = \boldsymbol{Z}^{(J^*)}(\mathcal{U}_{J^*}(\mathcal{X}))^T. \tag{2.23}$$

Poiché la matrice $\boldsymbol{V}_{\text{opt}}^{(J^*)}$ che soddisfa la precedente equazione è soluzione del problema di minimo (2.10) per $\boldsymbol{V}^{(j \neq J^*)}$ fissate, deve valere necessariamente la disuguaglianza

$$\text{MSE}(\mathcal{X}, \boldsymbol{V}_{\text{opt}}^{(J^*)}) \leq \text{MSE}(\mathcal{X}, \boldsymbol{V}^{(J^*)}). \tag{2.24}$$

I risultati ottenuti possono essere estesi a ciascuna matrice $\boldsymbol{V}^j$ della decomposizione per via della simmetria della funzione obiettivo. Di conseguenza, risolvendo sequenzialmente l'uguaglianza (2.22) per ciascuna di esse, la funzione obiettivo decresce. Un minimo locale dell'MSE viene raggiunto solo quando la soluzione rimane costante ad ogni ciclo di minimizzazione. Dal momento che il problema di ottimizzazione ristretto alle singole matrici è detto dei minimi quadrati, l'algoritmo appena descritto prende il nome di *metodo dei minimi quadrati alternati* (*ALS*) ed è riassunto dall'Algoritmo 1.
Siccome non vi è garanzia della convergenza del metodo al minimo globale, ma, piuttosto, è garantita per minimi locali, il criterio d'interruzione dell'algoritmo più comune è

$$\text{MSE}(\hat{\mathcal{X}}_k, \hat{\mathcal{X}}_{k-1}) \leq \eta, , \tag{2.25}$$

Algoritmo 1 Metodo dei minimi quadrati alternati

Input:

$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, $R \in \mathbb{N}$

$\boldsymbol{V}_0^{(j)} \in \mathbb{R}^{I_j \times R}, \quad j = 1, \ldots, N$

Metodo:

$k \leftarrow 1$

while MSE decresce **do**

  for $j = 1, \ldots, N$ **do**

    $\boldsymbol{Z}_k^{(j)} \leftarrow \mathcal{U}_j\left(\left\|\boldsymbol{V}_{k-1}^{(j_1)}, \boldsymbol{V}_{k-1}^{(j_2)}, \ldots, \boldsymbol{V}_{k-1}^{(j_{N-1})}\right\|\right)$

    Calcolo di $\boldsymbol{V}_k^{(j)}$ t.c. $[\boldsymbol{Z}_k^{(j)}(\boldsymbol{Z}_k^{(j)})^T]^T(\boldsymbol{V}_k^{(j)})^T = \boldsymbol{Z}_k^{(j)}(\mathcal{U}_j(\mathcal{X}))^T$

  end for

  $\hat{\mathcal{X}}_k \leftarrow \left\|\boldsymbol{V}_k^{(1)}, \boldsymbol{V}_k^{(2)}, \ldots, \boldsymbol{V}_k^{(N)}\right\|$

  $\mathrm{MSE}_k \leftarrow \dfrac{1}{N_{\mathrm{tot}}}\|\mathcal{X} - \hat{\mathcal{X}}_k\|_F^2$

  $k \leftarrow k + 1$

end while

con η tolleranza sull'oscillazione della decomposizione rispetto all'iterazione precedente. Dal momento che il calcolo dello stopping criterion non è parte del processo di ottimizzazione e che il calcolo dell'MSE per tensori di grandi dimensioni può risultare oneroso dal punto di vista computazionale, in questa trattazione si propone la seguente alternativa:

$$|\mathrm{MSE}_k - \mathrm{MSE}_{k-1}| \leq \eta. \tag{2.26}$$

La variazione dell'MSE è infatti un ottimo indicatore della stabilità della soluzione alla $k-$esima iterazione. Questa scelta è giustificata dalla costruzione del metodo ALS per questo specifico problema, dove il tensore $\hat{\mathcal{X}}_k$ è il risultato di N aggiornamenti sequenziali, fortemente dipendenti dall'approssimazione ottenuta all'iterazione precedente.
Differentemente dal criterio (2.25), quindi, sarà solamente necessario memorizzare ad ogni iterazione l'errore quadratico medio.

2.1.2 Problema dei minimi quadrati

Il calcolo della decomposizione PARAFAC del tensore $\mathcal{X}$ prevede, ad ogni iterazione, l'aggiornamento sequenziale delle matrici $\boldsymbol{V}^{(j)}$ risolvendo i rispettivi problemi dei minimi quadrati. Le soluzioni dei singoli problemi si ottengono risolvendo i rispettivi sistemi lineari

$$\nabla_{\boldsymbol{V}^{(j)}}\mathrm{MSE}(\mathcal{X}, \boldsymbol{V}^{(j)}) = 0, \quad j = 1, \ldots, N. \tag{2.27}$$

Si ricorda che, per rendere l'algoritmo più versatile ed efficiente, sono stati rilassati i vincoli derivanti dalle ipotesi di Harshman, Carroll e Chang , motivo per cui non vi è più la garanzia di ortogonalità tra i vettori appartenenti ad uno stesso modo. Di conseguenza, non è possibile avere la certezza che ad ogni iterazione le matrici $\boldsymbol{V}^{(j)}$ siano di rango pieno e, soprattutto, nella maggior parte dei casi non è noto a priori neanche il rango $\mathcal{X}$.

L'equazione (2.23), quindi, non può essere risolta direttamente. Essendo la funzione obiettivo ristretta alle singole matrici $\boldsymbol{V}^{(j)}$ convessa, si propone di utilizzare il *metodo di più ripida discesa del gradiente*.

Metodo di più ripida discesa del gradiente

Si vuole ricercare un minimo locale di una funzione $f(\cdot)$ applicando un metodo iterativo che aggiorni ad ogni iterazione l'approssimazione della soluzione $\boldsymbol{x}_{k-1} \in \mathbb{R}^N$ mediante l'assegnazione:

$$\boldsymbol{x}_k \leftarrow \boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{p}_k. \tag{2.28}$$

La funzione $f(\boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{p}_k)$ può essere approssimata mediante il suo sviluppo di Taylor troncato al primo ordine:

$$f(\boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{p}_k) \approx f(\boldsymbol{x}_{k-1}) + \alpha_k \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_{k-1}). \tag{2.29}$$

Si osserva facilmente che il vettore che massimizza la minimizzazione della funzione è $\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_{k-1})$, prendendo il nome di *direzione di più ripida discesa*.

Metodo del gradiente per il calcolo di $\hat{\mathcal{X}}$

Come anticipato nella sezione 2.1.1, nonostante le incognite siano delle matrici e non dei vettori, la norma di Frobenius e, di conseguenza, l'MSE sono invarianti per reshaping, per cui la notazione rimane invariata.
Infatti,

$$\nabla_{\boldsymbol{V}^{(j)}} \mathrm{MSE}(\mathcal{X}, \boldsymbol{V}^{(j)}) = \mathcal{R}_{\{I_j, R\}}(\nabla_{\mathrm{vec}(\boldsymbol{V}^{(j)})} \mathrm{MSE}(\mathcal{X}, \boldsymbol{V}^{(j)})). \tag{2.30}$$

Sia quindi $\boldsymbol{V}^{(j)}_{k,t-1}$ l'approssimazione della matrice $\boldsymbol{V}^{(j)}$ alla $k-$esima iterazione di ALS, e alla $(t-1)-$esima iterazione di discesa del gradiente. Sia inoltre $\nabla_{\boldsymbol{V}^{(j)}_{k,t-1}} \mathrm{MSE}_{k,t}(\mathcal{X}, \boldsymbol{V}^{(j)}_{k,t-1})$ il valore del gradiente della funzione obiettivo rispetto alla variabile in esame. L'aggiornamento dell matrice si calcola come segue:

$$\boldsymbol{V}^{(j)}_{k,t} \leftarrow \boldsymbol{V}^{(j)}_{k,t-1} - \alpha_t \cdot \nabla_{\boldsymbol{V}^{(j)}_{k,t-1}} \mathrm{MSE}_{k,t}(\mathcal{X}, \boldsymbol{V}^{(j)}_{k,t-1}), \quad \alpha_t \in \mathbb{R}^+. \tag{2.31}$$

Il coefficiente α_t è detto step-length e pesa l'aggiornamento della variabile minimizzante durante il processo di ottimizzazione, limitando anche il rischio di overflow. In generale, soprattutto nel caso di funzioni molto complesse quali quelle descritte dalle architetture delle *reti neurali artificiali* (*ANN*), vengono utilizzate delle euristiche, tra cui:

- step-length *costante*;
- aggiornamento *time-based*, ovvero, fissato un *rate di decadimento* η, α_t si calcola ad ogni iterazione come $\alpha_t = \dfrac{1}{1+\eta t}$;
- *decadimento esponenziale*, simile all'aggiornamento time-based, ma lo step-length si calcola mediante l'equazione $\alpha_t = \alpha_0 e^{-\eta t}$.

Nel caso in esame, essendo l'MSE una funzione quadratica e convessa per le matrici $\boldsymbol{V}^{(j)}$, lo è anche per α_t essendo la (2.31) lineare in α_t, per cui è possibile calcolare ad ogni iterazione lo step-length ottimale di discesa.
Per semplificare la notazione, si considererà implicito l'indice k relativo alle iterazioni di ALS e si indicherà con $\boldsymbol{G}_t^{(j)}$ il gradiente dell'MSE rispetto, appunto, alla matrice $\boldsymbol{V}_t^{(j)}$.
Sostituendo la (2.31) nell'equazione (2.17) si ottiene la seguente funzione obiettivo:

$$\begin{aligned}
\text{MSE}(\mathcal{X}, \alpha_t) &= \frac{1}{2N_{\text{tot}}} \| \, \mathcal{U}_j(\mathcal{X}) - (\boldsymbol{V}_{t-1}^{(j)} - \alpha_t \boldsymbol{G}_t^{(j)}) \boldsymbol{Z}^{(j)} \, \|_F^2 \\
&= \frac{1}{2N_{\text{tot}}} \| \, \alpha_t \boldsymbol{G}_t^{(j)} \boldsymbol{Z}^{(j)} + \mathcal{U}_j(\mathcal{X}) - \boldsymbol{V}_{t-1}^{(j)} \boldsymbol{Z}^{(j)} \, \|_F^2 \\
&= \frac{1}{2N_{\text{tot}}} \| \, \alpha_t \boldsymbol{G}_t^{(j)} \boldsymbol{Z}^{(j)} + \mathcal{U}_j(\mathcal{X}) - \mathcal{U}_j(\mathcal{X} - \hat{\mathcal{X}}_t) \, \|_F^2 \\
&= \frac{1}{2N_{\text{tot}}} \| \, \alpha_t \boldsymbol{G}_t^{(j)} \boldsymbol{Z}^{(j)} + \mathcal{U}_j(\mathcal{E}_t) \, \|_F^2 .
\end{aligned} \tag{2.32}$$

Siano $\boldsymbol{\phi}_t^{(j)} := \boldsymbol{G}_t^{(j)} \boldsymbol{Z}^{(j)}$ e $\boldsymbol{\psi}_t^{(j)} := \mathcal{U}_j(\mathcal{E}_t)$, esplicitando l'equazione precedente si ottiene che

$$\begin{aligned}
\text{MSE}(\mathcal{X}, \alpha_t) &= \frac{1}{2N_{\text{tot}}} \sum_{i_1, i_2} (\alpha_t \phi_{t,i_1,i_2}^{(j)} + \psi_{t,i_1,i_2}^{(j)})^2 \\
&= \frac{1}{2N_{\text{tot}}} (\alpha_t^2 \sum_{i_1, i_2} (\phi_{t,i_1,i_2}^{(j)})^2 + 2\alpha_t \sum_{i_1, i_2} (\phi_{t,i_1,i_2}^{(j)} \psi_{t,i_1,i_2}^{(j)}) + \sum_{i_1, i_2} (\psi_{t,i_1,i_2}^{(j)})^2) \\
&= \frac{1}{2N_{\text{tot}}} (\|\boldsymbol{\phi}_t^{(j)}\|_F^2 \alpha_t^2 + 2\langle \boldsymbol{\phi}_t^{(j)}, \boldsymbol{\psi}_t^{(j)} \rangle \alpha_t + \|\boldsymbol{\psi}_t^{(j)}\|_F^2).
\end{aligned} \tag{2.33}$$

La norma della matrice $\boldsymbol{\psi}_t^{(j)}$ dipende direttamente dalla norma del gradiente dell'errore quadratico medio in funzione della matrice $\boldsymbol{V}^{(j)}$ all'iterazione corrente. Quest'ultima equazione dimostra la convessità dell'errore quadratico medio anche in funzione dello step-length. Infatti, fin quando l'algoritmo non è arrivato a convergenza, il gradiente ha norma non nulla, di conseguenza

$$\begin{aligned}
\frac{d^2}{d\alpha_t^2} \text{MSE}(\mathcal{X}, \alpha_t) &= \frac{d^2}{d\alpha_t^2} \left[\frac{1}{2N_{\text{tot}}} (\|\boldsymbol{\phi}_t^{(j)}\|_F^2 \alpha_t^2 + 2\langle \boldsymbol{\phi}_t^{(j)}, \boldsymbol{\psi}_t^{(j)} \rangle \alpha_t + \|\boldsymbol{\psi}_t^{(j)}\|_F^2) \right] \\
&= \frac{d}{d\alpha_t} \left[\frac{1}{N_{\text{tot}}} (\|\boldsymbol{\phi}_t^{(j)}\|_F^2 \alpha_t + \langle \boldsymbol{\phi}_t^{(j)}, \boldsymbol{\psi}_t^{(j)} \rangle) \right] \\
&= \frac{\|\boldsymbol{\phi}_t^{(j)}\|_F^2}{N_{\text{tot}}} \geq 0.
\end{aligned} \tag{2.34}$$

Essendo il minimo unico, è sufficiente risolvere l'equazione

$$\frac{1}{N_{\text{tot}}} (\|\boldsymbol{\phi}_t^{(j)}\|_F^2 \alpha_t + \langle \boldsymbol{\psi}_t^{(j)}, \boldsymbol{\phi}_t^{(j)} \rangle) = 0, \tag{2.35}$$

la cui soluzione è

$$\alpha_t = -\frac{\langle \boldsymbol{\phi}_t^{(j)}, \boldsymbol{\psi}_t^{(j)} \rangle}{\|\boldsymbol{\phi}_t^{(j)}\|_F^2}, \tag{2.36}$$

ovvero lo step-length ottimale per la $t-$esima iterazione di discesa del gradiente.
In questo modo, ad ogni iterazione di discesa del gradiente viene garantita la massima decrescita possibile dell'MSE. Lo stopping criterion per una tolleranza fissata η è

$$\|\nabla \text{MSE}(\mathcal{X}, \boldsymbol{V}_t^{(j)})\|_F \leq \eta, \tag{2.37}$$

poiché, riprendendo l'equazione (2.22), interrompe le iterazioni quando l'errore di approssimazione della soluzione è inferiore alla tolleranza accettabile.
L'algoritmo è riassunto nel seguente pseudocodice.

Algoritmo 2 Metodo di più ripida discesa del gradiente per la matrice $\boldsymbol{V}^{(j)}$

Input:
$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, $R \in \mathbb{N}$
$\boldsymbol{Z}^{(j)} \in \mathbb{R}^{R \times \prod_{k \neq j} I_k}$
$\boldsymbol{V}_0^{(j)} \in \mathbb{R}^{I_j \times R}, \quad j = 1, \ldots, N$
Metodo:
$\boldsymbol{G}_1^{(j)} \leftarrow \nabla \text{MSE}(\mathcal{X}, \boldsymbol{V}_0^{(j)})$
$t \leftarrow 1$
while $\|\boldsymbol{G}_t^{(j)}\|_F > \eta$ **do**
  $\boldsymbol{\phi}_t^{(j)} \leftarrow \boldsymbol{G}_t^{(j)} \boldsymbol{Z}^{(j)}$
  $\boldsymbol{\psi}_t^{(j)} \leftarrow \mathcal{U}_j(\mathcal{X}) - \boldsymbol{V}_{t-1}^{(j)} \boldsymbol{Z}^{(j)}$
  $\alpha_t \leftarrow -\dfrac{\langle \boldsymbol{\phi}_t^{(j)}, \boldsymbol{\psi}_t^{(j)} \rangle}{\|\boldsymbol{\phi}_t^{(j)}\|_F^2}$
  $\boldsymbol{V}_t^{(j)} \leftarrow \boldsymbol{V}_{t-1}^{(j)} - \alpha_t \boldsymbol{G}_t^{(j)}$
  $t \leftarrow t + 1$
end while

L'algoritmo completo di approssimazione della decomposizione PARAFAC di un tensore multimodale $\mathcal{X}$ è formalizzato nello Algoritmo 3.

2.2 Analisi di complessità

Il costo computazionale dell'algoritmo 3 dipende principalmente dal costo degli unfolding, dei prodotti matriciali, scalari, e dal calcolo dell'MSE. Il costo complessivo verrà quindi calcolato con un approccio bottom-up, analizzando prima le singole operazioni.

Unfolding

Nell'algoritmo l'unfolding viene applicato in due diverse casistiche:

- unfolding del tensore $\mathcal{X}$;
- calcolo della matrice $\boldsymbol{Z}^{(J^*)} = \left\| \boldsymbol{V}^{(j_1)}, \boldsymbol{V}^{(j_2)}, \ldots, \boldsymbol{V}^{(j_{N-1})} \right\|$.

Algoritmo 3 Metodo di approssimazione della decomposizione PARAFAC

Input:

$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, $R \in \mathbb{N}$

$\boldsymbol{V}_0^{(j)} \in \mathbb{R}^{I_j \times R}, \quad j = 1, \ldots, N$

$\eta_1, \eta_2 \in \mathbb{R}^+$

Metodo:

$\hat{\mathcal{X}}_0 \leftarrow \left\| \boldsymbol{V}_0^{(1)}, \boldsymbol{V}_0^{(2)}, \ldots, \boldsymbol{V}_0^{(N)} \right\|$

$\text{MSE}_0 \leftarrow \text{MSE}(\mathcal{X}, \hat{\mathcal{X}}_0)$

$\text{tol} \leftarrow \eta_1 + 1$

$k \leftarrow 1$

while $\text{tol} > \eta_1$ **do**

  for $j = 1, \ldots, N$ **do**

    $\boldsymbol{Z}_k^{(j)} \leftarrow \mathcal{U}_j \left(\left\| \boldsymbol{V}_{k-1}^{(j_1)}, \boldsymbol{V}_{k-1}^{(j_2)}, \ldots, \boldsymbol{V}_{k-1}^{(j_{N-1})} \right\| \right)$

    $\boldsymbol{M}_0 \leftarrow \boldsymbol{V}_{k-1}^{(j)}$

    $\boldsymbol{G}_1 \leftarrow \nabla_{\boldsymbol{M}_0} \text{MSE}(\mathcal{X}, \boldsymbol{M}_0)$

    $t \leftarrow 1$

    while $\|\boldsymbol{G}_t\|_F > \eta_2$ **do**

      $\boldsymbol{\phi}_t^{(j)} \leftarrow \boldsymbol{G}_t^{(j)} \boldsymbol{Z}_k^{(j)}$

      $\boldsymbol{\psi}_t^{(j)} \leftarrow \mathcal{U}_j(\mathcal{X}) - \boldsymbol{M}_{t-1} \boldsymbol{Z}_k^{(j)}$

      $\alpha_t \leftarrow -\dfrac{\langle \boldsymbol{\phi}_t^{(j)}, \boldsymbol{\psi}_t^{(j)} \rangle}{\|\boldsymbol{\psi}_t^{(j)}\|_F^2}$

      $\boldsymbol{M}_t \leftarrow \boldsymbol{M}_{t-1} - \alpha_t \boldsymbol{G}_t^{(j)}$

      $t \leftarrow t + 1$

    end while

    $\boldsymbol{V}_k^{(j)} \leftarrow \boldsymbol{M}_{t-1}$

    $\text{MSE}_k \leftarrow \text{MSE}(\mathcal{U}_N(\mathcal{X}), \boldsymbol{V}_k^{(N)} \boldsymbol{Z}_k^{(N)})$

    $\text{tol} \leftarrow |\text{MSE}_k - \text{MSE}_{k-1}|$

    $k \leftarrow k + 1$

  end for

end while

Nel primo caso, siccome $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ è il tensore da approssimare, esso non è soggetto ad alterazioni o aggiornamenti delle sue entrate, per cui l'operatore di unfolding coincide con la Definizione 6. Di conseguenza, il costo computazionale è dato dalla somma tra il costo della trasposizione rispetto alla direzione j fissata ed il costo del reshaping ad una matrice.

Dal momento che entrambe le operazioni prevedono un riarrangiamento degli elementi del tensore, il loro tempo di calcolo è lineare rispetto al numero di entrate, poiché è necessario scorrere una volta ognuna di esse e riposizionarle nella nuova forma corrispondente.

Assumendo che $I_1 = I_2 = \cdots = I_N = D$, il costo computazionale è dato da $\mathcal{O}(D^N + D^N) = \mathcal{O}(D^N)$.

Diversamente, nel caso della matrice $\boldsymbol{Z}^{(J^*)}$ sarebbe necessario calcolare il tensore $(N-1)-$modale ad ogni step di ottimizzazione sommando prima le R poliadi ($\mathcal{O}(RD(N-1))$), calcolandone successivamente il prodotto poliadico ($\mathcal{O}(RD^{N-1})$) ed infine applicandovi l'unfolding ($\mathcal{O}(D^{N-1})$). Per questo motivo si preferisce una pipeline unificata durante la quale si calcolano ed allocano sequenzialmente le matrici bidimensionali dell'unfolding, come descritto nei seguenti punti:

- date le matrici $\boldsymbol{V}^{(j)}$, si fissa l'indice J^* del modo rispetto al quale calcolare l'unfolding;
- si inizializza la matrice $\boldsymbol{Z}^{(J^*)} \in \mathbb{R}^{R \times D^{N-1}}$ nella quale allocare i risultati;
- ogni vettore colonna di $\boldsymbol{V}^{(j_1)}$ deve essere moltiplicato per tutte le combinazioni possibili di prodotti tra gli elementi dei restanti modi ristretti al proprio indice $r = 1, \dots, R$; ogni singola operazione è un $\mathcal{O}(DD^{N-2}) = \mathcal{O}(D^{N-1})$, per cui il costo totale è $\mathcal{O}(RD^{N-1})$;
- le $N-2$ matrici risultanti vengono allocate sequenzialmente nella matrice $\boldsymbol{Z}^{(J^*)}$ inizializzata precedentemente; quest'operazione è lineare nel numero di elementi, per cui ha un costo di $\mathcal{O}(RD^{N-1})$.

Il tempo di calcolo complessivo è dato da $\mathcal{O}(RD^{N-1} + RD^{N-1}) = \mathcal{O}(RD^{N-1})$.

Prodotti matriciali e scalari

Gli unici prodotti matriciali presenti nell'algoritmo vengono effettuati per il calcolo dei gradienti e per il calcolo dello step-length ottimale.
La matrice $\boldsymbol{G}^{(J^*)} \in \mathbb{R}^{I_{J^*} \times R}$ è data dall'equazione

$$\boldsymbol{G}^{(J^*)} = (\mathcal{U}_{J^*}(\mathcal{X}) - \boldsymbol{V}^{(J^*)}\boldsymbol{Z}^{(J^*)})(\boldsymbol{Z}^{(J^*)})^T,$$

il cui tempo di calcolo è dato da:

- il prodotto matriciale tra $\boldsymbol{V}^{(J^*)} \in \mathbb{R}^{D \times R}$ e $\boldsymbol{Z}^{(J^*)} \in \mathbb{R}^{R \times D^{N-1}}$ di $\mathcal{O}(DRD^{N-1}) = \mathcal{O}(RD^N)$;
- la sottrazione matriciale, la quale ha un costo lineare $\mathcal{O}(D^N)$;
- il prodotto matriciale dato dalla matrice risultante dalla parentesi e da $\boldsymbol{Z}^{(J^*)}$, per un costo di $\mathcal{O}(RD^N)$.

Per cui, il calcolo del gradiente richiede un numero di operazioni pari a $\mathcal{O}(RD^N + D^N + RD^N) = \mathcal{O}(RD^N)$.
Si osserva che nell'algoritmo i prodotti matriciali hanno tutti complessità pari a $\mathcal{O}(RD^N)$, poiché calcolati sempre tra una delle matrici $\boldsymbol{V}^{(j)}$ e la rispettiva matrice $\boldsymbol{Z}^{(j)}$.
Il prodotto scalare $\langle \cdot, \cdot \rangle$, è applicato unicamente per il calcolo dello step-length tra $\boldsymbol{\phi}^{(j)}$ e $\boldsymbol{\psi}^{(j)}$, entrambe matrici di $\mathbb{R}^{D \times D^{N-1}}$. Dalla Definizione 2 si osserva che l'operatore prevede di moltiplicare le matrici elemento a elemento ($\mathcal{O}(D^N)$) e di sommarne i risultati

($\mathcal{O}(D^N)$). Il costo complessivo è quindi $\mathcal{O}(D^N + D^N) = \mathcal{O}(D^N)$.
Dal momento che $\|\mathcal{X}\|_F = \sqrt{\langle \mathcal{X}, \mathcal{X} \rangle}$, anche il costo computazionale della norma di Frobenius risulta essere $\mathcal{O}(D^N)$.

MSE

Analogamente alla norma di Frobenius, il calcolo dell'MSE prevede la sottrazione tra due tensori composti da D^N elementi, il calcolo del quadrato della norma del tensore ottenuto diviso per uno scalare.
Anche in questo caso il costo computazionale è $\mathcal{O}(D^N)$.

Costo computazionale complessivo

Il costo computazionale dell'intero algoritmo è dato dai costi delle singole operazioni e dal numero di iterazioni di ALS, indicato con K_{ALS}, e di discesa del gradiente, indicato con K_{GD}.
In particolare, ad ogni iterazione di risoluzione dei singoli problemi di minimo vengono calcolati

- due prodotti matriciali ($\mathcal{O}(RD^N)$);
- un prodotto scalare, ($\mathcal{O}(D^N)$);
- due norme ($\mathcal{O}(D^N)$);
- due sottrazioni matriciali ($\mathcal{O}(D^N)$);
- un prodotto tra uno scalare e una matrice ($\mathcal{O}(D^N)$),

per un costo totale per problema di $\mathcal{O}(K_{\text{GD}}RD^N)$. L'unfolding di $\mathcal{X}$ non è stato considerato nel conto poiché viene effettuato una sola volta all'inizio dell'algoritmo.
Ad ogni iterazione di ALS, invece, le operazioni effettuate sono le seguenti:

- calcolo di $\boldsymbol{Z}^{(j)}$ ($\mathcal{O}(RD^{N-1})$);
- calcolo di un gradiente ($\mathcal{O}(D^N)$);
- K_{GD} iterazioni di discesa del gradiente ($\mathcal{O}(K_{\text{GD}}RD^N)$);
- calcolo dell'MSE ($\mathcal{O}(D^N)$).

Ogni iterazione di ALS richiede quindi un tempo di calcolo pari a $\mathcal{O}(K_{\text{GD}}RD^N)$, per un totale di $\mathcal{O}(K_{\text{ALS}}K_{\text{GD}}RD^N)$.
Prima delle iterazioni vengono calcolati gli N unfolding di $\mathcal{X}$ ed l'MSE relativo all'approssimazione iniziale $\hat{\mathcal{X}}_0$, ma quest'ultimi non influiscono sul costo computazionale totale, che rimane $\mathcal{O}(K_{\text{ALS}}K_{\text{GD}}RD^N)$.

2.3 Implementazione

A scopo dimostrativo si propone un'implementazione in Python dell'algoritmo 3 per il calcolo della decomposizione PARAFAC di tensori tridimensionali. La libreria utilizzata è *NumPy* poiché fornisce un'efficiente gestione degli array multidimensionali e una vasta gamma di funzioni per operazioni matriciali e tensoriali, rinominata nel codice con *np*.
Per una maggiore chiarezza e leggibilità, la funzione principale denominata *PARAFAC_3way* richiamerà i metodi per il calcolo degli unfolding e per le iterazioni di discesa del gradiente implementati esternamente.

2.3.1 Unfolding del tensore $\mathcal{X}$

L'unfolding di un tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ rispetto ad una specifica direzione $J^* \in \{1, 2, 3\}$ prevede l'esecuzione delle seguenti operazioni:

- calcolo della trasposizione $\mathcal{X}^{T_p} \in \mathbb{R}^{I_{J^*} \times I_{j_1} \times I_{j_2}}$, indicando con p la permutazione che scambia J^* con 1, mantenendo invariato l'indice rimanente;
- inizializzazione della matrice $\mathcal{U}_{J^*}(\mathcal{X}) \in \mathbb{R}^{I_{J^*} \times I_{j_1} I_{j_2}}$ nella quale verrà allocato l'unfolding del tensore;
- allocazione sequenziale delle I_{j_2} matrici di dimensione $I_{J^*} \times I_{j_1}$ nella matrice inizializzata precedentemente.

```python
def unfold_X(X, axis):

    # Trasposizione tensore
    if axis == 0:
        X = X.copy()
    elif axis == 1:
        X = np.swapaxes(X, 0, 1)
    else:
        X = np.swapaxes(X, 0, 2)

    # Estrazione della dimensionalità del tensore
    I, J, T = X.shape

    # X_unfold = np.ndarray (shape: (I, K = J * T))
    K = J * T
    X_unfold = np.empty((I, K))

    # Loop lungo l'asse 2
    for t in range(T):
        # Ad ogni iterazione si memorizza sequenzialmente la t-esima
        # matrice all'intero del nuovo np.ndarray X_unfold
        X_unfold[:, t * J : (t + 1) * J] = X[:, :, t]

    return X_unfold
```

2.3.2 Calcolo di $\boldsymbol{Z}^{(J^*)}$

Diversamente dal caso precedente, come introdotto nell'analisi della complessità, la matrice $\boldsymbol{Z}^{(J^*)}$ può essere calcolata direttamente a partire dalle matrici della decomposizione. Essendo il tensore da decomporre tridimensionale, è sufficiente moltiplicare ogni colonna di $\boldsymbol{V}^{(j_1)}$ per ogni scalare delle rispettive colonne di $\boldsymbol{V}^{(j_2)}$ e concatenare le matrici risultanti.

```
def unfold_tensors(V1, V2):

    # Estrazione delle dimensionalità di V1 e V2
    v1, v2 = V1.shape[0], V2.shape[0]

    # Z = np.ndarray (shape: (v1 * v2, R))
    # Per coerenza di notazione nel codice, l'asse di lunghezza
    # R viene mantenuto in posizione 1
    Z = np.empty((v1 * v2, V1.shape[1]))

    # Loop lungo l'asse 2
    for idx, row in enumerate(V2):
        # Ad ogni iterazione si moltiplicano le colonne di V1
        # per i rispettivi scalari delle colonne di V2 e le matrici
        # risultanti vengono allocate in T_unfold
        Z[idx * v1 : (idx + 1) * v1, :] = V1 * row

    return Z
```

2.3.3 Metodo del gradiente

La funzione in cui è implementato il metodo del gradiente richiede in input le seguenti variabili:

- $\boldsymbol{V}$, matrice rispetto alla quale minimizzare l'MSE;
- $\boldsymbol{Z}$, la matrice calcolata mediante la funzione *unfold_tensors*;
- $\boldsymbol{X}_{\text{unfold}}$, unfolding del tensore target $\mathcal{X}$;
- massimo numero di iterazioni da eseguire e la tolleranza sulla norma del gradiente.

```
def SteepestGradientDescent(V, Z, X_unfold, gd_iter_max, gd_tol):

    # Inizializzazione della matrice Z^T
    Z_trnsp = Z.T

    # Steepest Gradient Descent
    for k in range(gd_iter_max):
```

```
        # Calcolo di X_hat
        X_hat = V.dot(Z_trnsp)

        # Calcolo del gradiente dell'MSE rispetto a V
        psi = X_unfold - X_hat
        grad_V = psi.dot(Z) / X_unfold.size

        # Calcolo dello step-length ottimale
        phi = grad_V.dot(Z_trnsp)
        alpha = - (phi * psi).sum()
        alpha /= (phi ** 2).sum()

        # Update della matrice V
        V -= alpha * grad_V

        # Criterio di stop: norma del gradiente
        norm_V = np.linalg.norm(grad_V)
        if norm_V < gd_tol:
            break

    return V
```

2.3.4 Decomposizione PARAFAC

La funzione *PARAFAC_3way* combina i metodi descritti consentendo un'implementazione chiara ed efficiente dell'algoritmo di decomposizione tensoriale.
Le variabili in input sono:

- $\mathcal{X}$, tensore da approssimare;
- R, numero di fattori della decomposizione;
- massimo numero di iterazioni e tolleranze η_1, η_2 per le verifiche di convergenza dei metodi ALS e del gradiente;
- *n_max_nochange*, ovvero il massimo numero di iterazioni per cui considerare la soluzione stabile;
- *random_state*, seed per l'inizializzazione casuale delle triadi iniziali.

Per garantire la replicabilità dei risultati, la prima operazione eseguita all'interno della funzione è l'inizializzazione dello *pseudo random number generator*, dal quale vengono generate le triadi iniziali della decomposizione. Quest'ultime sono campionate da una distribuzione uniforme con valori compresi tra $\min(\mathcal{X})$ e $\max(\mathcal{X})$.
Successivamente, si inizializzano le variabili necessarie per:

- tenere traccia della funzione obiettivo durante le iterazioni;
- monitorare la convergenza del metodo.

Prima delle iterazioni di ALS, si calcolano gli unfolding di $\mathcal{X}$ rispetto a ciascun asse.
Il metodo ALS si interrompe quando il modulo della variazione della funzione obiettivo rispetto all'iterazione precedente è stabile entro η_1 per almeno *n_max_nochange* iterazioni, o quando si è raggiunto il massimo numero di iterazioni.

```
def PARAFAC(X, R = 1, als_iter_max = 50, als_tol = 1e-8, n_max_nochange = 5,
            gd_iter_max = 50, gd_tol = 1e-8, random_state = 300890):

    # Inizializzazione prng
    prng = np.random.default_rng(random_state)

    # Estrazione della dimensionalità del tensore
    t, n, m = X.shape

    # Numero di elementi del tensore
    N = t * n * m

    # Bounds per l'inizializzazione
    lb, ub = X.min(), X.max()

    # Calcolo degli unfolding di X
    X_u0 = unfold_X(X, axis = 0)
    X_u1 = unfold_X(X, axis = 1)
    X_u2 = unfold_X(X, axis = 2)

    # Inizializzazione della decomposizione
    V0 = prng.uniform(lb, ub, (t, R))
    V1 = prng.uniform(lb, ub, (n, R))
    V2 = prng.uniform(lb, ub, (m, R))

    # Inizializzazione delle variabili di controllo
    als_iter_max = int(als_iter_max)
    gd_iter_max = int(gd_iter_max)
    n_max_nochange = int(n_max_nochange)
    loss_track = np.empty((als_iter_max + 1))
    loss_track[0] = 0
    count_nochange = 0

    # Iterazioni ALS
    for k in range(als_iter_max):

        # Aggiornamento tensori
        Z = unfold_tensors(V1, V2)
        V0 = SteepestGradientDescent(V0, Z, X_u0, gd_iter_max, gd_tol)
        Z = unfold_tensors(V0, V2)
        V1 = SteepestGradientDescent(V1, Z, X_u1, gd_iter_max, gd_tol)
        Z = unfold_tensors(V1, V0)
        V2 = SteepestGradientDescent(V2, Z, X_u2, gd_iter_max, gd_tol)
```

```
        # Calcolo della funzione obiettivo
        loss_track[k + 1] = ((X_u2 - (V2.dot(Z.T))) ** 2).sum() / (2 * N)

        # Calcolo della variazione della soluzione
        delta_loss = loss_track[k + 1] - loss_track[k]
        if abs(delta_loss) < als_tol:
            count_nochange += 1
        elif count_nochange > 0:
            count_nochange = 0

        # Criterio di stop
        if count_nochange == n_max_nochange:
            break

    return (V0, V1, V2), loss_track[1: k + 2]
```

2.3.5 Test

L'algoritmo è stato testato su un tensore $\mathcal{X} \in \mathbb{R}^{50\times40\times30}$ generato randomicamente da una distribuzione uniforme con valori compresi nell'intervallo $[-1, 1]$. Si è scelto di inizializzare il generatore di numeri pseudo casuali *default_rng* del pacchetto *Random* di *NumPy* a partire dal seed 300890 per garantire la replicabilità dei risultati.
La decomposizione è stata testata per 10 iterazioni di ALS impostando il parametro n_max_nochange = 30 in maniera tale da confrontare la variazione della funzione obiettivo durante il processo di minimizzazione rispetto al numero di triadi $R \in \{5, 10, 15, 30\}$. Per ogni sotto-problema di minimo sono state eseguite fino a 150 iterazioni di metodo del gradiente, interrotte prematuramente nel caso lo stopping criterion fosse stato soddisfatto per una tolleranza di 10^{-8}.
Dalla Figura 2.2 si osserva come, a partire dall'inizializzazione (iterazione 0), la funzione obiettivo raggiunga rapidamente un plateau per ciascun valore di R, convergendo rapidamente ad un minimo locale. Inoltre, in questo caso, l'aumento del numero di triadi per rappresentare il tensore in forma canonica non porta ad un miglioramento significativo dell'approssimazione, mantenendo l'errore quadratico medio nello stesso ordine di grandezza. Ciò significa che il tensore in analisi può essere decomposto in poche, ma rappresentative, triadi.

La stessa pipeline è stata applicata ad un tensore $\mathcal{Y} \in \mathbb{R}^{50\times40\times30}$ inizializzato sempre randomicamente, ma come forma di Kruskal di rango 30, in Figura 2.3 i risultati del processo di minimizzazione.
In questo caso l'algoritmo è andato a convergenza entro le prime 3 iterazioni e l'approssimazione della decomposizione con 30 fattori ha portato ad un effettivo miglioramento del valore della funzione obiettivo di un ordine di grandezza.
Nella Tabella 2.3.5 sono stati riassunti i risultati dei due test, indicando l'MSE di approssimazione in funzione dei valori di R e l'errore di approssimazione dei vettori delle triadi in termini di ortogonalità. Nello specifico, per ogni matrice $\boldsymbol{V}^{(j)} \in \mathbb{R}^{I_j\times R}$ è stata calcolata

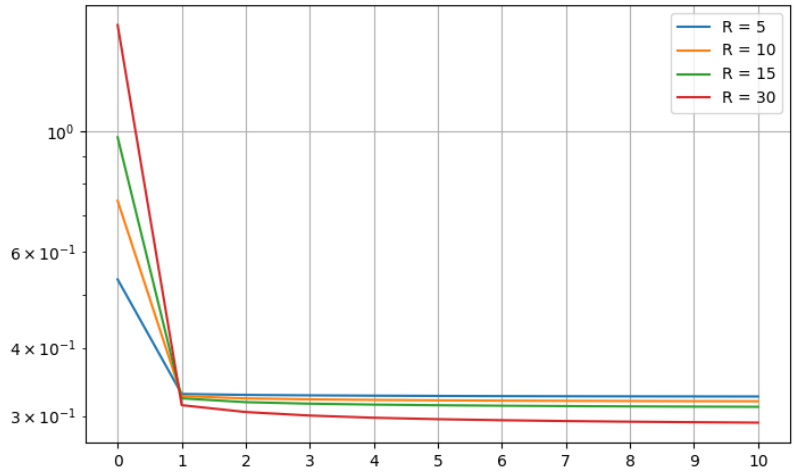

Figura 2.2: Decrescita dell'MSE di approssimazione del tensore $\mathcal{X}$ in funzione del numero delle iterazioni.

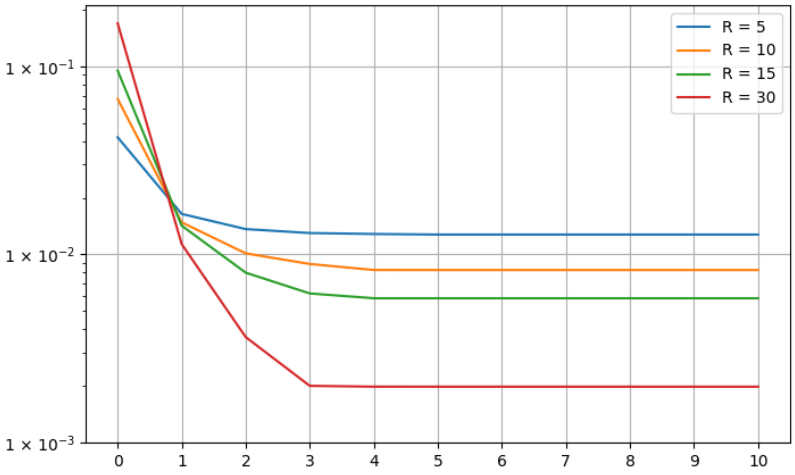

Figura 2.3: Decrescita dell'MSE di approssimazione del tensore $\mathcal{Y}$ in funzione del numero delle iterazioni.

la quantità

$$\mathrm{MSE}(\boldsymbol{I}_R, (\boldsymbol{V}^{(j)})^T \boldsymbol{V}^{(j)}) = \frac{1}{R^2} \sum_{i_1,i_2} ((\boldsymbol{I}_R)_{i_1,i_2} - ((\boldsymbol{V}^{(j)})^T \boldsymbol{V}^{(j)})_{i_1,i_2})^2, \tag{2.38}$$

ovvero l'errore di approssimazione della matrice identità, successivamente ad una rinormalizzazione delle colonne delle matrici in analisi.

Tensore	**R**	**MSE**	**Ortogonalità $\boldsymbol{V}^{(1)}$**	**Ortogonalità $\boldsymbol{V}^{(2)}$**	**Ortogonalità $\boldsymbol{V}^{(3)}$**
$\mathcal{X}$	5	0.326	0.014	0.018	0.007
	10	0.319	0.029	0.023	0.022
	15	0.311	0.042	0.034	0.028
	30	0.291	0.034	0.025	0.024
$\mathcal{Y}$	5	0.013	0	0	0
	10	0.008	0.016	0.016	0.018
	15	0.006	0.030	0.033	0.029
	30	0.002	0.036	0.035	0.032

Le decomposizioni ottenute mostrano comunque come una decomposizione composta da un numero ridotto di triadi possa risultare sufficientemente informativa per approssimare e descrivere il tensore target, come avviene applicando la PCA a dataset bidimensionali. Per questo motivo, la PARAFAC trova ampia applicazione nell'analisi e nella compressione dati quando quest'ultimi sono multilineari, come per esempio, dati multimediali, serie storiche e risultati sperimentali, per i quali può essere considerata come un'estensione della PCA.

Parte II

Applicazioni

Capitolo 3

PCA per dati multilineari

3.1 Principal Component Analysis

3.1.1 Il metodo

La *Principal Component Analysis* (*PCA*) è un metodo matematico che trova ampio utilizzo nell'ambito dell'analisi di dati bilineari, in particolare negli ambiti della features reduction e della data visualization. Infatti, mediante la PCA, i dati vengono proiettati nel sottospazio identificato dalle *componenti principali*, ovvero le direzioni lungo le quali ne viene spiegata la massima varianza, rendendone quindi la rappresentazione estremamente informativa. Il nuovo sistema di riferimento viene appunto calcolato a partire dalla matrice di varianza e covarianza delle *features*, ovvero degli attributi che caratterizzano le osservazioni del dataset in analisi, indicato con $\boldsymbol{X} \in \mathbb{R}^{N \times M}$, dove N è il numero di campioni ed M il numero di features.
In generale, l'algoritmo per il calcolo delle componenti principali prevede che le colonne di $\boldsymbol{X}$ vengano traslate di una quantità pari alle rispettive *medie campionarie* μ_j tali che

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}, \ j = 1, \ldots, N, \tag{3.1}$$

di modo che, successivamente a questa trasformazione, le features risultanti abbiano media nulla. Questo primo step permette di calcolare rapidamente anche la matrice di varianza e covarianza, indicata da $\boldsymbol{S} \in \mathbb{R}^{M \times M}$.
Infatti, sia

$$\mathrm{Cov}(\boldsymbol{X}_{:,j_1}, \boldsymbol{X}_{:,j_2}) = \frac{1}{N-1} \sum_{i=1}^{N} (x_{i,j_1} - \mu_{j_1})(x_{i,j_2} - \mu_{j_2}), \ j_1, j_2 = 1, \ldots, N \tag{3.2}$$

la *covarianza campionaria* tra le features indicizzate da j_1, j_2, e

$$\mathrm{Var}(\boldsymbol{X}_{:,j_1}) = \mathrm{Cov}(\boldsymbol{X}_{:,j_1}, \boldsymbol{X}_{:,j_2}) \tag{3.3}$$

l'equazione per il calcolo della varianza campionaria relativa alla j_1-esima feature, allora è possibile indicare la *matrice varianza e covarianza* $\boldsymbol{S} \in \mathbb{R}^{M\times M}$ come

$$\boldsymbol{S} = \frac{1}{N-1}\begin{bmatrix} \mathrm{Var}(\boldsymbol{X}_{:,\boldsymbol{j_1}}) & \mathrm{Cov}(\boldsymbol{X}_{:,j_1}, \boldsymbol{X}_{:,j_2}) & \dots & \mathrm{Cov}(\boldsymbol{X}_{:,j_1}, \boldsymbol{X}_{:,j_M}) \\ \mathrm{Cov}(\boldsymbol{X}_{:,j_2}, \boldsymbol{X}_{:,j_1}) & \mathrm{Var}(\boldsymbol{X}_{:,\boldsymbol{j_2}}) & \dots & \mathrm{Cov}(\boldsymbol{X}_{:,j_2}, \boldsymbol{X}_{:,j_M}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(\boldsymbol{X}_{:,j_M}, \boldsymbol{X}_{:,j_1}) & \mathrm{Cov}(\boldsymbol{X}_{:,j_2}, \boldsymbol{X}_{:,j_M}) & \dots & \mathrm{Var}(\boldsymbol{X}_{:,j_M}) \end{bmatrix}. \tag{3.4}$$

Indicando con

$$\boldsymbol{X}_\mu = \begin{bmatrix} \boldsymbol{X}_{:,1} - \mu_1\boldsymbol{1} & \boldsymbol{X}_{:,2} - \mu_2\boldsymbol{1} & \dots & \boldsymbol{X}_{:,N} - \mu_N\boldsymbol{1} \end{bmatrix} \tag{3.5}$$

la matrice $\boldsymbol{X}$ con le colonne traslate delle rispettive medie, allora vale la relazione:

$$\boldsymbol{S} = \frac{1}{N-1}\boldsymbol{X}_\mu^T\boldsymbol{X}_\mu. \tag{3.6}$$

Proposizione 4. *Morfismo diagonalizzabile*
Un morfismo $F : \mathcal{V} \longrightarrow \mathcal{V}$ è *diagonalizzabile* se esiste una base $\boldsymbol{V}$ di $\mathcal{V}$ tale che la rispettiva matrice di trasformazione $\boldsymbol{A_{VV}}(F)$ è diagonale.

Osservazione. Si osserva che, per ogni base $\boldsymbol{V'}$ di $\mathcal{V}$ la corrispondente matrice di trasformazione $\boldsymbol{A_{V'V'}}(F)$ è simile a $\boldsymbol{A_{VV}}(F)$ rispetto alla matrice di cambio base $\boldsymbol{M_{V'V}}$, ovvero

$$\boldsymbol{A_{V'V'}} = \boldsymbol{M_{VV'}}\boldsymbol{A_{VV}}\boldsymbol{M_{V'V}}. \tag{3.7}$$

Teorema. *Teorema spettrale*
Sia $\boldsymbol{X} \in \mathbb{R}^{M\times M}$ una matrice simmetrica, allora $\boldsymbol{X}$ è diagonalizzabile rispetto ad una matrice ortogonale $\boldsymbol{P} \in \mathbb{R}^{M\times M}$.

Conseguenza diretta del teorema è che, siccome anche la matrice di varianza e covarianza $\boldsymbol{S}$ è simmetrica, è anch'essa diagonalizzabile rispetto ad una matrice ortogonale $\boldsymbol{P}$, calcolabile mediante decomposizione agli autovalori di $\boldsymbol{S}$, risultando

$$\boldsymbol{S} = \boldsymbol{P}\boldsymbol{\Lambda}\boldsymbol{P}^T, \tag{3.8}$$

con $\boldsymbol{\Lambda} \in \mathbb{R}^{M\times M}$ matrice diagonale degli autovalori.
Inoltre, essendo $\boldsymbol{S}$ e $\boldsymbol{\Lambda}$ matrici simili, vale la relazione

$$\mathrm{Tr}(\boldsymbol{S}) = \mathrm{Tr}(\boldsymbol{\Lambda}), \tag{3.9}$$

per cui la matrice $\boldsymbol{\Lambda}$ pesa i contributi dei singoli autovettori in funzione della varianza da essi spiegata. Riordinando i termini della decomposizione per valore decrescente degli autovalori, gli autovettori saranno riordinati di conseguenza, indicando le direzioni lungo le quali è spiegato il $\left(\lambda_i / \sum_{j=1}^{M} \lambda_j\right) \cdot 100\%$ della varianza del dataset in analisi.
Richiamando la Proposizione 4, $\boldsymbol{P}$ coincide con la matrice di cambio base, per cui, la matrice dei dati rispetto al nuovo sistema di riferimento si calcola mediante il prodotto

$$\hat{\boldsymbol{X}}^T = \boldsymbol{P}^T\boldsymbol{X}^T. \tag{3.10}$$

Siccome le colonne di $\boldsymbol{P}$ sono ordinate per varianza spiegata, proiettare la matrice $\boldsymbol{X}$ rispetto ai primi m autovettori ne permette una rappresentazione a dimensionalità ridotta che ne conserva il $\left(\sum_{j=1}^{m} \lambda_j / \sum_{j=1}^{M} \lambda_j\right) \cdot 100\%$ della varianza spiegata, evidenziando e selezionando le relazioni tra le features più caratterizzanti.
Gli autovettori prendono quindi il nome di *componenti principali*.

Osservazione. Per costruzione, le componenti principali non sono altro che i vettori singolari destri della matrice $\boldsymbol{S}^{\frac{1}{2}} = \sqrt{\frac{1}{N-1}}\boldsymbol{X}_{\mu}$, le cui rispettive quantità di varianza spiegata sono i valori singolari, mentre la proiezione di $\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{P}$ è equivalente alla matrice dei vettori singolari sinistri moltiplicati per i rispettivi valori singolari.
Infatti, poiché

$$\boldsymbol{S}^{\frac{1}{2}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{P}^T$$

con

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Lambda} \\ \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{N\times M},\ \boldsymbol{0} \in \mathbb{R}^{(N-M)\times M},$$

ricordando che $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}_M$, si ottiene l'uguaglianza

$$\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{P} = \boldsymbol{U}\boldsymbol{\Sigma}. \tag{3.11}$$

3.1.2 Relazione tra PCA e PARAFAC

Richiamando la sezione 1.4, essendo l'SVD una restrizione della CPD per tensori bilineari, deve esistere una relazione tra la PCA e la PARAFAC. Applicando l'algoritmo 3 a $\boldsymbol{S}^{\frac{1}{2}}$ si ottiene infatti la decomposizione

$$\boldsymbol{S}^{\frac{1}{2}} = \boldsymbol{V}^{(1)}(\boldsymbol{V}^{(2)})^T$$

per la quale vale l'uguaglianza

$$\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{P}^T = \boldsymbol{V}^{(1)}(\boldsymbol{V}^{(2)})^T$$

Di conseguenza,

$$\begin{aligned} \boldsymbol{V}^{(1)} &= \boldsymbol{U}_{:,:M}\boldsymbol{\Lambda}^{\alpha} \\ \boldsymbol{V}^{(2)} &= \boldsymbol{P}\boldsymbol{\Lambda}^{\beta} \end{aligned}, \tag{3.12}$$

con $\alpha + \beta = 1$, per cui i vettori colonna della matrice $\boldsymbol{V}^{(2)}$ coincidono con le componenti principali a meno di riscalamenti e, analogamente, dall'uguaglianza (3.11) si osserva che vale la stessa relazione tra la matrice $\boldsymbol{V}^{(1)}$ e la proiezione $\boldsymbol{S}^{\frac{1}{2}}\boldsymbol{P}$.
Per mostrare l'analogia tra PARAFAC e PCA, entrambi i metodi sono stati applicati al dataset *Wine* disponibile nel pacchetto *datasets* di *Scikit-learn*. Quest'ultimo consta di 3 classi e 13 features, per cui performare una riduzione della dimensionalità rispetto alle

componenti principali ne permette la visualizzazione senza perdita di interpretabilità.
Il dataset è stato standardizzato per colonne, ovvero è stata applicata la trasformazione

$$\begin{aligned} \phi : \mathbb{R}^{I_2} &\longrightarrow \mathbb{R}^{I_2} \\ \boldsymbol{X}_{:,j} &\longmapsto \frac{\boldsymbol{X}_{:,j} - \mu_j}{\sigma_j}, \end{aligned} \tag{3.13}$$

dove

$$\sigma_j = \sqrt{\mathrm{Var}(\boldsymbol{X}_{:,j})} \tag{3.14}$$

è la *deviazione standard* della $j-$esima colonna. Il vantaggio di questa trasformazione è che le features del dataset risultante $\boldsymbol{X}_{std}$ hanno ciascuna media nulla e varianza unitaria, riscalandone i valori e rendendo le features tra loro confrontabili, evitando, quindi, che le componenti principali siano influenzate dagli originali ordini di grandezza.
Dalla Figura 3.1 si osserva l'estrema similarità dei risultati ottenuti applicando la PCA (in alto) e la PARAFAC (in basso). Infatti, la prima componente principale (PC_1) ed il fattore della prima diade (PF_1) risultano pressoché identici, mentre i secondi termini delle decomposizioni (PC_2 e PF_2 rispettivamente) differiscono solo per il segno, estraendo gli stessi pattern e contributi dalle features. I dati proiettati (scatter plots a sinistra) sono infatti speculari, dimostrando quanto descritto.
Dai barplots si osserva inoltre come le componenti principali e, di conseguenza, i fattori della decomposizione PARAFAC siano degli assi orientati all'interno dello spazio delle features, la cui direzione è data dal contributo di ciascuna di esse nella caratterizzazione complessiva del dataset. Questo garantisce un'alta interpretabilità dei due metodi e dei risultati da essi ottenuti, evidenziando le principali correlazioni tra le features e permettendone una descrizione sintetica ed intuitiva.

3.2 Parallel Factor Analysis

Nel contesto dell'analisi dati, calcolare la decomposizione PARAFAC di un dataset standardizzato $\boldsymbol{X} \in \mathbb{R}^{I_1 \times I_2}$ coincide con il calcolo delle componenti principali a meno di eventuali riscalamenti. Le matrici $\boldsymbol{V}^{(1)}$ e $\boldsymbol{V}^{(2)}$ ottenute mediante metodo ALS descrivono, rispettivamente, la distribuzione dei dati e l'orientamento del nuovo sistema di riferimento in funzione delle features.
Per costruzione del metodo, differentemente dalla PCA, i vettori colonna di $\boldsymbol{V}^{(2)}$ non hanno necessariamente norma unitaria, pertanto proiettare nuovi dati calcolando il prodotto matriciale $\boldsymbol{X}_{\text{new}}\boldsymbol{V}^{(2)}$ può portare a contrazioni e/o dilatazioni lungo le nuove direzioni. Infatti, richiamando i risultati (3.12),

$$\begin{aligned} \boldsymbol{X}\boldsymbol{V}^{(2)} &= \boldsymbol{V}^{(1)}(\boldsymbol{V}^{(2)})^T\boldsymbol{V}^{(2)} \\ &= \boldsymbol{V}^{(1)}\boldsymbol{\Lambda}^{\beta}\boldsymbol{P}^T\boldsymbol{P}\boldsymbol{\Lambda}^{\beta} \\ &= \boldsymbol{V}^{(1)}\boldsymbol{\Lambda}^{\beta}\boldsymbol{\Lambda}^{\beta} \\ &= \hat{\boldsymbol{X}}_{\text{PCA}}\boldsymbol{\Lambda}^{\beta}, \end{aligned}$$

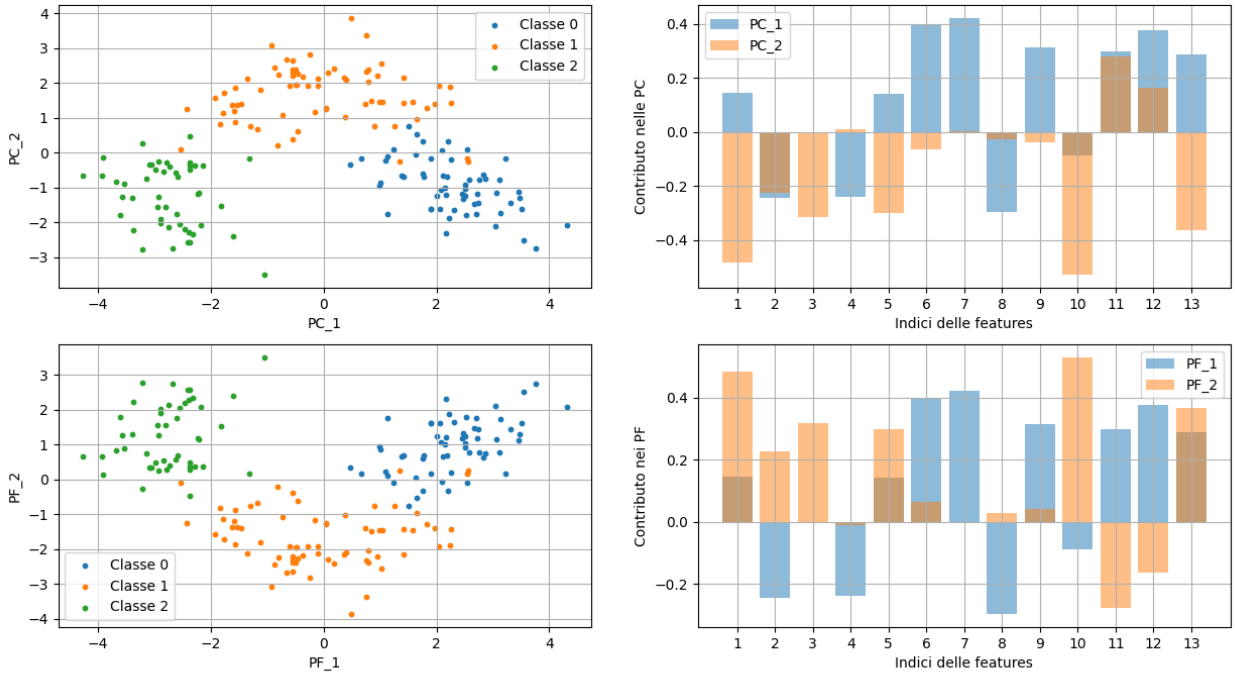

Figura 3.1: Applicazione di PCA (in alto) e PARAFAC (in basso) al Wine dataset. A sinistra sono riportate le proiezioni dei dati rispetto ai nuovi sistemi di riferimento, mentre a destra la caratterizzazione di quest'ultimi rispetto ai contributi delle features.

ovvero, proiettando i dati su cui sono stati calcolati i fattori nel nuovo sistema di riferimento, si commette un errore di riscalamento rispetto alla PCA di λ_i^{β} per l'$i-$esima feature, $\forall i = 1, \dots, M$.
Il problema, quindi, si traduce nel calcolare la matrice $\boldsymbol{V}_{\text{new}}^{(1)}$ che minimizza l'errore di approssimazione di $\boldsymbol{X}_{\text{new}}$ fissato $\boldsymbol{V}^{(2)}$, ovvero

$$\text{MSE}(\boldsymbol{X}_{\text{new}}, \boldsymbol{V}_{\text{new}}^{(1)}) = \frac{1}{N_{\text{new}}} \|\boldsymbol{X}_{\text{new}} - \boldsymbol{V}_{\text{new}}^{(1)} (\boldsymbol{V}^{(2)})^T\|_F^2. \tag{3.15}$$

Quest'ultimo è equivalente ad un singolo step dell'algoritmo 3 poiché, come nella PCA, si assume che le nuove osservazioni provengano dalla stessa distribuzione dei dati già a disposizione, considerando quindi il profilo delle features identificato dalle colonne di $\boldsymbol{V}^{(2)}$ fissato. La funzione (3.15) è quindi convessa e la matrice incognita $\boldsymbol{V}_{\text{new}}^{(1)}$ può essere approssimata mediante metodo del gradiente applicando l'algoritmo 2.
Questo tipo di approccio permette di calcolare l'approssimazione a rango ridotto delle osservazioni di un dataset senza che sia necessario costruire una matrice di proiezione a

partire dagli autovettori della matrice di varianza e covarianza, ma risolvendo sequenzialmente una serie di problemi dei minimi quadrati.
Inoltre, la PARAFAC è principalmente un efficiente metodo di decomposizione tensoriale, per cui. nell'analisi dati, essa generalizza la PCA permettendone l'applicazione a dataset multimodali.
Siano $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ il tensore dei dati $(N-1)-$dimensionali da decomporre, le cui I_1 osservazioni sono distribuite lungo il primo asse, e $\hat{\mathcal{X}} = \left\| \boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}, \ldots, \boldsymbol{V}^{(N)} \right\|$ la sua decomposizione PARAFAC troncata all'$R-$esima poliade. Si definiscono allora *fattori paralleli* i vettori colonna delle matrici $\boldsymbol{V}^{(j>1)}$, mentre, coerentemente con quanto descritto nella sezione precedente, la matrice $\boldsymbol{V}^{(1)}$ non è altro che la rappresentazione dei dati rispetto alle prime R poliadi.
Nel caso fosse necessario ridurre la dimensionalità di nuove osservazioni rispetto ad un set di fattori principali già calcolati, è sufficiente minimizzare la funzione (3.15) estesa al caso $N-$modale, ovvero:

$$\mathrm{MSE}(\mathcal{X}_{\mathrm{new}}, \boldsymbol{V}^{(1)}_{\mathrm{new}}) = \frac{1}{N_{\mathrm{new}}} \| \mathcal{U}_1(\mathcal{X}_{\mathrm{new}}) - \boldsymbol{V}^{(1)}_{\mathrm{new}} \boldsymbol{U}_1(\left\| \boldsymbol{V}^{(2)}, \boldsymbol{V}^{(3)}, \ldots, \boldsymbol{V}^{(N)} \right\|) \|^2_F. \tag{3.16}$$

Il metodo appena presentato prende il nome di *Parallel Factor Analysis* e nella trattazione verrà indicata con l'acronimo*PFA*.

3.2.1 Applicazione PFA a dataset di immagini

In questa sezione si applicherà la PFA al dataset *Fetch Olivetti Faces*, disponibile anch'esso nel pacchetto *datasets* di *Scikit-learn*. Si tratta di una collezione di 400 immagini 64×64 in scala di grigi rafFiguranti volti di 40 persone differenti, di cui alcuni esempi sono riportati in Figura 3.2.

Figura 3.2: Alcuni volti del dataset *Fetch Olivetti Faces*.

Ciascuna immagine è stata standardizzata indipendentemente applicando la trasformazione

$$\begin{aligned} \phi : \mathbb{R}^{N\times N} &\longrightarrow \mathbb{R}^{N\times N} \\ x_{i,j} &\longmapsto \frac{x_{i,j} - \mu_{\boldsymbol{X}}}{\sigma_{\boldsymbol{X}}} \end{aligned} \tag{3.17}$$

dove $N = 64$, mentre

$$\mu_{\boldsymbol{X}} = \frac{1}{N^2}\sum_{i,j} x_{i,j}$$

$$\sigma_{\boldsymbol{X}} = \sqrt{\frac{1}{N^2-1}\sum_{i,j}(x_{i,j} - \mu_{\boldsymbol{X}})^2}$$

sono, rispettivamente, la media e la deviazione standard campionarie dei pixel per $\boldsymbol{X}$. Questa scelta è motivata dal fatto che, trasformando indipendentemente ogni immagine, ciascuna osservazione del dataset è caratterizzata da una matrice di media nulla e varianza unitaria. Il vantaggio di questo tipo di standardizzazione sulle immagini è dato dal fatto che se ne conservano le rispettive proprietà senza alcun tipo di bias introdotto dalle altre osservazioni. Di conseguenza, i nuovi campioni da decomporre saranno anch'essi standardizzati indipendentemente, limitando eventuali errori di traslazione e riscalamento dei singoli valori assunti dai pixel.
Decomponendo il dataset con una singola triade, i pattern più significativi vengono identificati nei due fattori rappresentanti l'altezza e la larghezza dell'immagine, mentre la distribuzione dei dati rispetto alle features estratte è sintetizzata dal fattore parallelo approssimante il primo asse. Calcolando il prodotto diadico tra i fattori delle features è possibile visualizzare le aree più significative nelle immagini del dataset, denominando la mappa così ottenuta *factor-face*. I risultati descritti sono riportati in Figura 3.3.

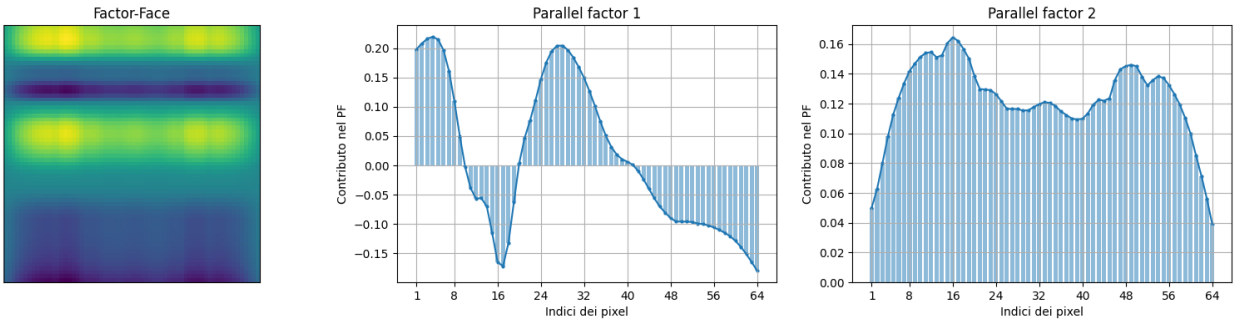

Figura 3.3: Features estratte applicando la PARAFAC al dataset *Fetch Olivetti Faces*; da sinistra verso destra, rispettivamente, *factor-face*, pattern estratto dal fattore relativo all'altezza dell'immagine e pattern estratto dal fattore relativo alla larghezza.

Dalla factor-face è rapido osservare come le aree più informative delle immagini risultano essere principalmente quelle attorno agli occhi, rispetto alle quali i dati sono rappresentati

dal rimanente fattore della triade.
I dati sono attualmente caratterizzati non più da N^2 pixel, ma da un singolo numero reale che pesa il contributo della factor-face per ciascun'immagine, la quale a sua volta sintetizza con soli $2N$ valori i pattern principali dell'intero dataset, ottenendo un modello estremamente sintetico ed interpretabile.
Aumentando il numero di poliadi aumenta proporzionalmente la quantità di informazione estratta, migliorando ulteriormente il profilo definito dalla factor-face. Calcolando esplicitamente il tensore $\hat{\mathcal{X}}$, ovvero l'approssimazione del dataset utilizzando R poliadi, è possibile visualizzare i volti del dataset nel nuovo sistema di riferimento, come in Figura 3.4. La factor-face ottenuta da questa nuova decomposizione con $R = 8$ diadi evidenzia in maniera più dettagliata i tratti caratterizzanti dei volti presenti nel dataset, tra cui occhi, sopracciglia, naso e bocca. Inoltre, la riduzione di dimensionalità ha avuto un effetto regolarizzante sulle immagini, centrando l'orientamento del volto ed esaltandone le aree più informative, come si osserva nella colonna centrale della Figura.
Nonostante le ricostruzioni non siano dettagliate, è comunque possibile distinguere facilmente i volti in funzione dei tratti sopra citati, in particolare, dalla lunghezza, dallo spessore e dall'inclinazione delle approssimazioni dei medesimi.
Si ricorda, però, che le immagini riportate in Figura 3.4 sono la ricostruzione 64×64 dei dati ridotti di dimensionalità rispetto al profilo definito dalla factor-face, ovvero dalle matrici $\boldsymbol{V}^{(2)}, \boldsymbol{V}^{(3)}$. Come nel caso bilineare, infatti, le osservazioni sono ora compresse nella matrice $\boldsymbol{V}^{(1)} \in \mathbb{R}^{400\times 8}$, le cui nuove features sono, per ogni campione del dataset, i pesi associati a ciascuna diade per la ricostruzione del volto a partire dalla factor-face.
Le features sono state quindi ridotte di circa il 99.8% ($(8/(64)^2)\cdot 100 \approx 0.2$) massimizzando l'informatività della nuova rappresentazione.

3.2.2 Classificazione di dati a dimensionalità ridotta

Nel contesto dell'analisi dati e del machine learning, la notevole compressione delle features consente di ridurre efficacemente la dimensionalità dei problemi di ottimizzazione da risolvere durante il fitting dei metodi di clustering e classificazione.
La nuova rappresentazione dei dati ottenuta mediante PFA sintetizza la maggior parte dell'informazione nelle componenti del nuovo sistema di riferimento, descrivendo quindi ciascuna osservazione come un vettore di dimensione R. Come avviene nel caso bilineare con la PCA, metodi come il *Support Vector Machines* (*SVM*) e la *regressione logistica* beneficiano dei risultati della decomposizione, riducendo il rischio di possibili bias dovuti dalla numerosità delle features. Relativamente all'SVM, è inoltre più efficiente e meno computazionalmente onerosa l'applicazione dei *kernel* al fine di migliorare la separazione dei dati dal momento che l'aumento di dimensionalità viene calcolato solo rispetto alle features più significative.
In particolare, il modello utilizzato per testare le performance di classificazione successivamente alla riduzione della dimensionalità ottenuta dall'applicazione della la PFA è un SVM con kernel *rbf*, ovvero

$$\begin{aligned} \mathcal{K} : \mathbb{R}^{N\times N} &\longrightarrow \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{x}^*) &\longmapsto e^{-\gamma\|\boldsymbol{x}-\boldsymbol{x}^*\|_F^2},\ \gamma \in \mathbb{R}^+. \end{aligned} \tag{3.18}$$

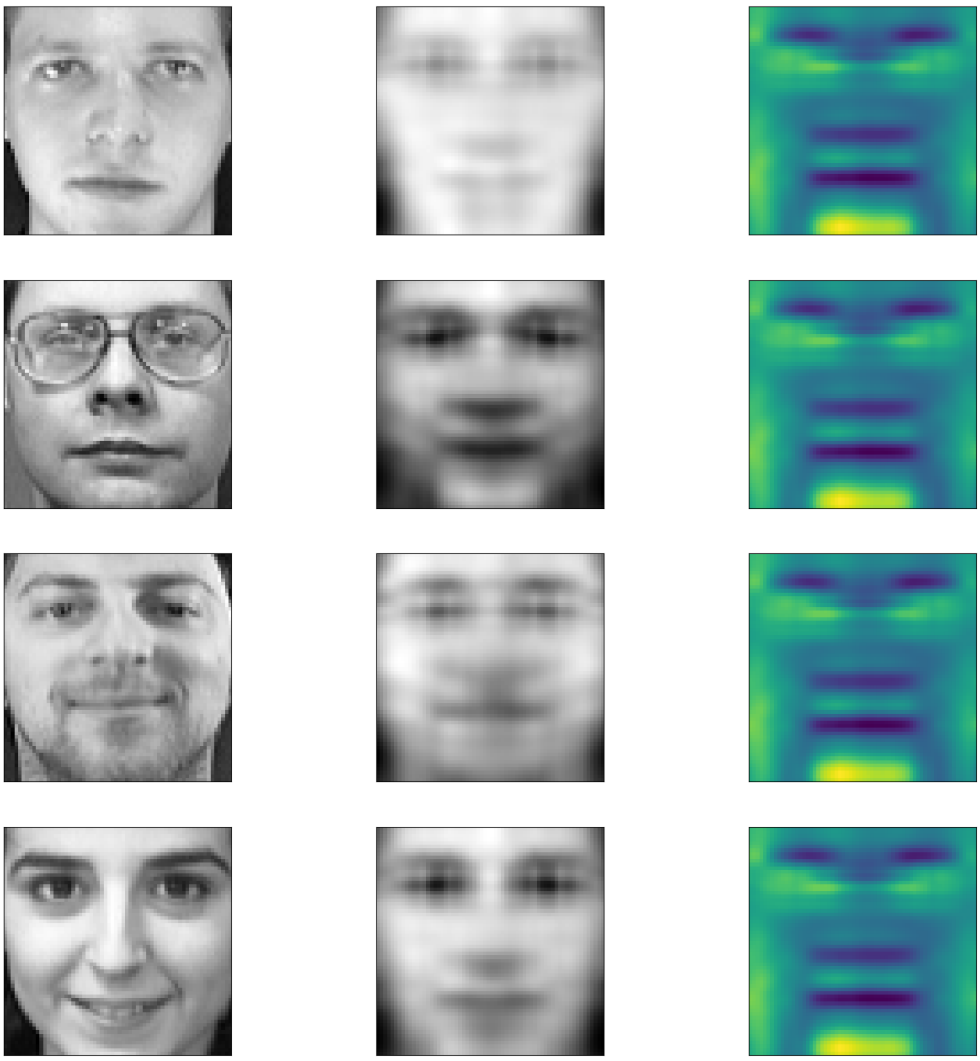

Figura 3.4: Volti della Figura 3.2 ricostruiti utilizzando 8 poliadi; a sinistra l'immagine originale, in centro l'immagine ricostruita a partire dalla factor-face, riportata come riferimento sulla destra.

Quest'ultimo è stato testato su quattro decomposizioni differenti per valore di R, rispettivamente $2, 4, 8, 16$, di modo da osservare come la quantità di informazione estratta dalle features scala al variare del numero di triadi. I parametri del modello su cui è stato effettuato il tuning sono:

- $C \in \{1, 10, 100, 1000\}$;
- $\gamma \in \{0.001, 0.01, 0.1\}$.

Il primo è detto *coefficiente di regolarizzazione* e, in funzione del valore che gli si assegna, penalizza gli errori di classificazione: bassi valori di C portano alla costruzione di margini *soft*, limitando l'overfitting, ma aumentando il rischio di commettere errori di previsione; viceversa, alti valori di C portano alla costruzione di margini *hard*, risultando così in performance più conservative. Il parametro γ introdotto nell'equazione (3.18) pesa invece la distanza tra due osservazioni: analogamente al coefficiente di regolarizzazione, alti valori di γ portano il modello a partizionare lo spazio in maniera più complessa e dettagliata, mentre, al contrario, bassi valori comportano una partizione più semplice e generalizzabile ai nuovi dati.
Il modello è stato addestrato sul 60% del dataset, quindi su 6 immagini per volto (si ricorda che il dataset è composto da 40 volti differenti per i quali sono disponibili 10 immagini in totale), e testato sul restante 40%. Le previsioni sono state valutate utilizzando la metrica *f1*, definita come

$$\frac{1}{N_{\text{classes}}} \sum_{i=1}^{N_{\text{classes}}} 2 \cdot \frac{\text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}, \tag{3.19}$$

con

$$\begin{cases} \text{Precision}_i = \dfrac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} \\ \qquad\qquad\qquad\qquad \forall i = 1, \ldots, N_{\text{classes}}, \\ \text{Recall}_i \quad = \dfrac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \end{cases} \tag{3.20}$$

ed indicando con TP_i, FN_i, FP_i, rispettivamente, il numero di previsioni corrette ed errate degli elementi dell'$i-$esima classe e di assegnazioni ad essa di campioni che non vi appartengono.

Le combinazioni di parametri sono state testate fissando prima R e C e variando γ in ordine crescente di valore; terminata questa prima fase, C scala al valore successivo e si testano nuovamente tutte i valori di γ. Analogamente vale per R. Dal grafico si osserva come già per $R = 4$, e alti coefficienti di regolarizzazione ($C = 1000, \gamma = 0.1$) è possibile partizionare perfettamente lo spazio dei campioni, risultando però poco robusto alla variabilità dei nuovi dati da classificare. Pur mantenendo alti i valori di C e γ, il classificatore ha migliorato notevolmente la sensibilità, ottenendo un *f1* maggiore di 0.8 sul test-set comprimendo il numero di features di più del 99%, raggiungendo la migliore performance con i parametri $R = 16, C = 1000, \gamma = 0.01$.
I risultati appena descritti sono stati ottenuti calcolando la decomposizione PARAFAC su

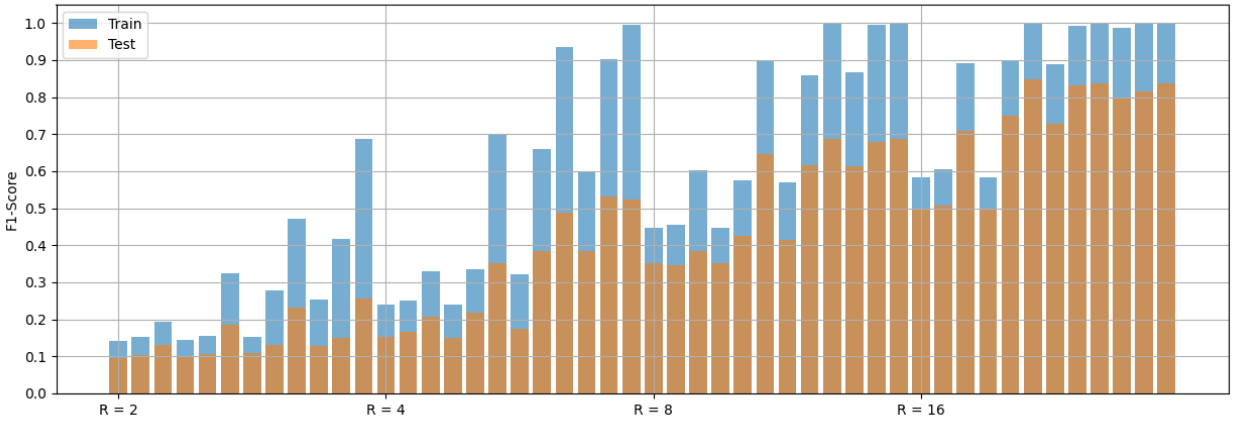

Figura 3.5: Risultati della grid search sui parametri dell'SVM. Ogni barplot rappresenta lo score *f1* raggiunto per una data combinazione di parametri nel formato $[R, C, \gamma]$.

tutto il dataset di training. Siccome nelle applicazioni reali i dataset sono drasticamente più numerosi, calcolare i fattori paralleli sull'intero tensore di training può essere inefficiente ed oneroso. Se i dati sono sufficientemente informativi, è possibile approssimare la factor-face su un sottoinsieme più piccolo dei dati a disposizione, per poi trasformare tutti i campioni in un secondo momento.

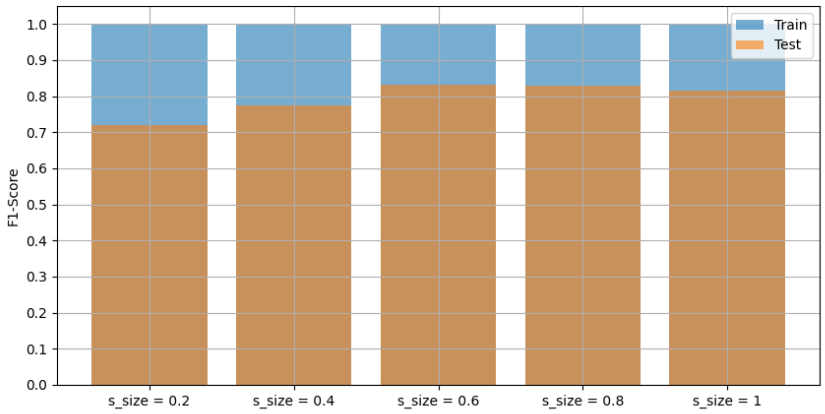

Figura 3.6: Risultati dell'SVM al variare della dimensione del sottoinsieme di dati su cui è stata applicata la PARAFAC.

Con *s_size*, in Figura 3.6, si indica la frazione di dataset utilizzata per calcolare la decomposizione tensoriale. Dalla Figura si deduce che, in questo caso, è sufficiente calcolare

i fattori paralleli da solo il 40% del tensore originale, garantendo una rappresentazione analoga al risultato della decomposizione di sottoinsiemi più numerosi.

3.2.3 Applicazione di PFA ed SVM al MNIST dataset

Il dataset *Fetch Olivetti Faces* è risultato utile per una migliore comprensione dell'applicazione della PARAFAC e del ruolo dei fattori paralleli nell'ambito dell'analisi dati. Si ricorda, però, che l'SVM è stato addestrato utilizzando 240 campioni di cui 6 per ogni classe, portando alla costruzione di un modello conservativo ed estremamente sensibile alla variabilità di nuovi dati.
Per questo motivo si è scelto di applicare la stessa pipeline ad un dataset con una numerosità maggiore, ovvero il *MNIST Digits*, sempre disponibile nel pacchetto *datasets* di *Scikit-learn*. Quest'ultimo è composto da 1797 immagini 8×8 nelle quali sono rafFigurati i numeri da 0 a 9 scritti a mano, come rappresentato in Figura 4.12, con una media di 180 campioni per classe.

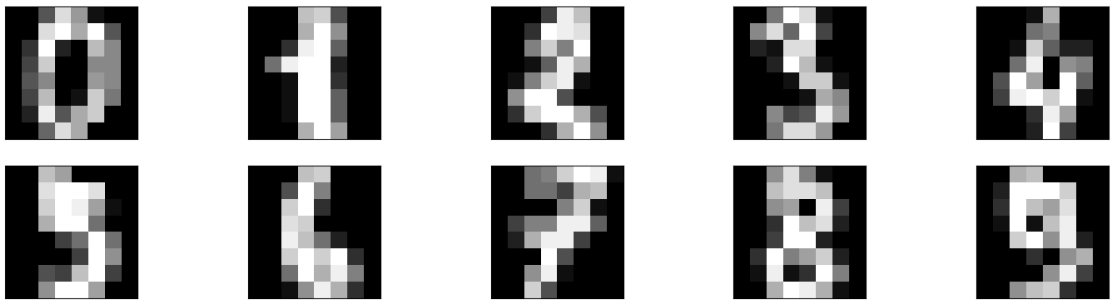

Figura 3.7: Samples del dataset *MNIST Digits*.

L'SVM è stato addestrato sempre sul 60% del dataset totale utilizzando gli stessi valori di C e γ testati per il *Fetch Olivetti Faces*, diversamente dal numero di triadi delle decomposizioni che variano nell'insieme $\{1, 2, 4, 8\}$. I risultati sono riportati in Figura 3.8, dalla quale si osserva come il maggiore numero di samples per classe ha migliorato la robustezza dell'SVM. Infatti, le performance relative a valori di $C \in \{1, 10\}$ sono preferibili per via della costruzione di margini soft, portando anche a valori di *f1* sul test set più alti rispetto a quelli sul training set.
Le due migliori combinazioni in termini di score di previsione e robustezza differiscono solo per il valore di $C \in \{1, 10\}$, con $R = 4$ e $\gamma = 0.001$. In entrambi i casi è stato replicato il test relativo al calcolo della decomposizione da un subsample del training set, ottenendo i risultati mostrati, rispettivamente, nelle Figura 3.9 e 3.10.
In particolare, relativamente al caso con $C = 1$, il fitting della decomposizione rispetto ad un numero ristretto di campioni ha permesso all'SVM di modellare eventuali errori di proiezione, ottenendo una partizione meno rigida dello spazio dei dati. Aumentando il

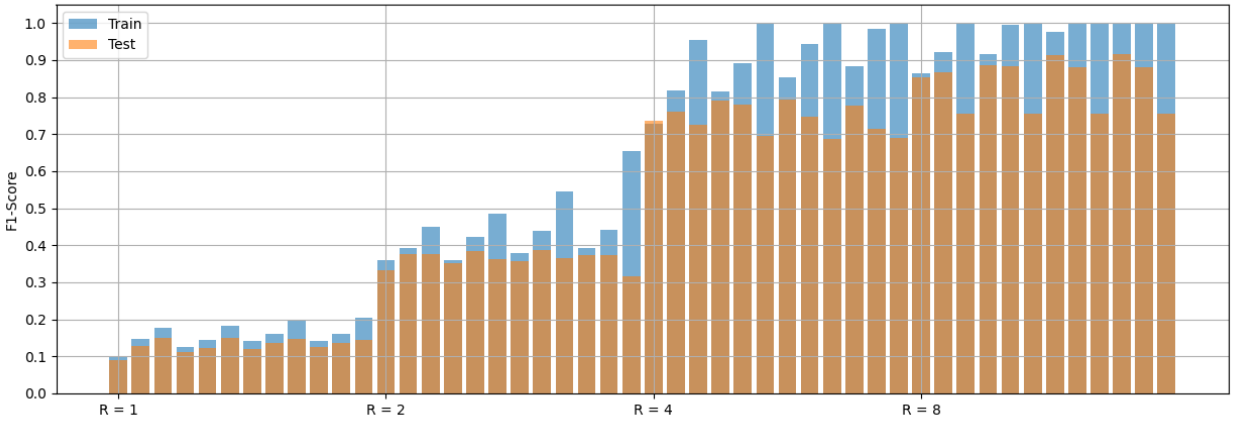

Figura 3.8: Risultati della grid search sui parametri dell'SVM relativi al dataset *MNIST Digits*.

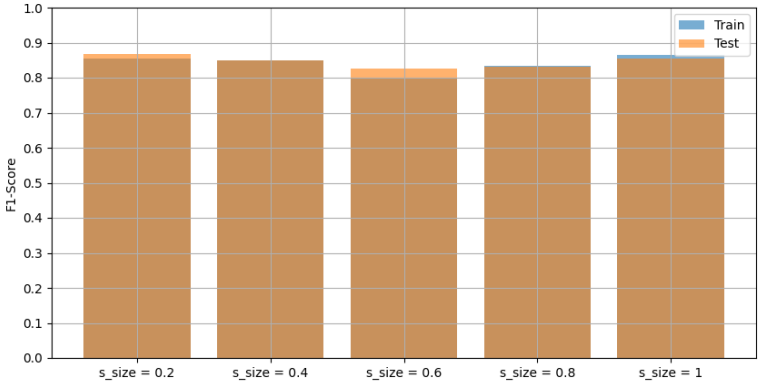

Figura 3.9: Risultati dell'SVM al variare della dimensione del sottoinsieme di dati su cui è stata applicata la PARAFAC per $C = 1$.

coefficiente di regolarizzazione a 10, gli score sono aumentati sopra a 0.9. In entrambi casi, l'applicazione della decomposizione PARAFAC a solamente il 20% del dataset ha portato a risultati migliori, mantenendo gli stessi score che si otterrebbero decomponendo l'intero tensore dei dati, ma riducendone notevolmente il costo computazionale dell'algoritmo ALS e, soprattutto, limitando il rischio di overfitting.

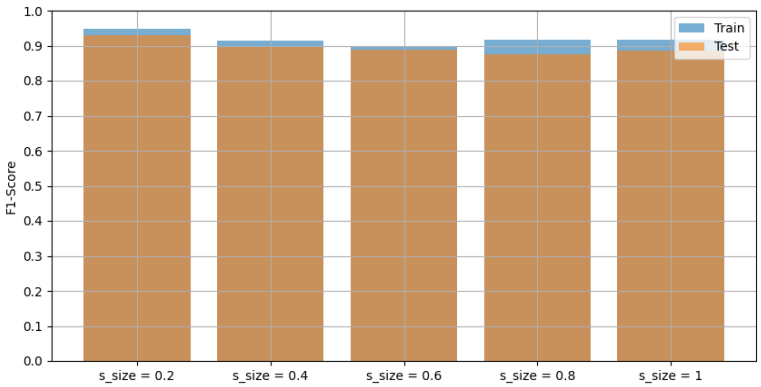

Figura 3.10: Risultati dell'SVM al variare della dimensione del sottoinsieme di dati su cui è stata applicata la PARAFAC per $C = 10$.

Capitolo 4

Feature mapping tensoriale

4.1 Feature mapping

4.1.1 Preprocessing e feature maps

Nell'analisi dati e nel machine learning, raramente gli algoritmi ed i modelli vengono addestrati sui dati grezzi, ma piuttosto rispetto a manipolazioni di quest'ultimi. Il motivo risiede nel fatto che non sempre i valori assunti dalle features sono confrontabili (es.: diversi ordini di grandezza), informativi (features superflue e/o ridondanti) o sufficienti (necessità di ulteriori features per rendere i samples separabili) affinché le ipotesi rispetto alle quali i modelli garantiscono il miglior *fitting* dei dati siano soddisfatte. Le tecniche utilizzate per trasformare i dati grezzi sono numerose e prendono il nome di *feature maps*.

Definizione 15. *feature map*
Sia $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ un tensore composto da I_1 samples multimodali caratterizzati da $I_2 + I_3 \cdots + I_N$ features. Si definisce *feature map* una funzione $\mathcal{F}$ tale che:

$$\begin{aligned} \mathcal{F} : \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N} &\longrightarrow \mathbb{R}^{I_1 \times J_1 \times \cdots \times J_M} \\ \mathcal{X} &\longmapsto \mathcal{F}(\mathcal{X}) = \mathcal{Y}. \end{aligned} \tag{4.1}$$

La funzione $\mathcal{F}$, quindi, mappa $\mathcal{X}$ in un nuovo tensore $\mathcal{Y}$ caratterizzato da nuove M features.

Alcuni esempi di feature maps possono essere i riscalamenti, come le standardizzazioni (3.13) e (3.17). Anche la PCA e, di conseguenza, la PFA sono feature maps, poiché consentono la costruzione di una mappa dallo spazio delle features allo spazio generato, rispettivamente, dalle componenti principali e dai fattori paralleli.

Proposizione 5. Siano $\mathcal{F}_i$, $\forall i = 1, \ldots, M$ feature maps tali che

$$\begin{aligned} \mathcal{F}_1 &: \mathcal{S} \longrightarrow \mathcal{S}_1 \\ \mathcal{F}_i &: \mathcal{S}_{i-1} \longrightarrow \mathcal{S}_i, \quad \forall i = 2, \ldots N, \end{aligned}$$

con $\mathcal{X} \in \mathcal{S}$. Allora, indicando con $\bigcirc_{i=1}^{N} \mathcal{F}_i$ la composizione di feature maps, anche la funzione

$$\begin{aligned} \mathcal{F} : \mathcal{S} &\longrightarrow \mathcal{S}_N \\ \mathcal{X} &\longmapsto \bigcirc_{i=1}^{N} \mathcal{F}_i(\mathcal{X}) = \mathcal{Y} \end{aligned} \tag{4.2}$$

è una feature map.

Il processo di manipolazione dei dati, includendo anche metodi per la selezione dei samples o di eventuali ricampionamenti, e di scelta ed applicazione della migliore feature map $\mathcal{F}$ è detto *preprocessing*. Le performance dei modelli sono strettamente correlate al preprocessing, il quale dipende a sua volta dai dati e dal task in questione. La scelta della mappa ottimale $\mathcal{F}^*$ è perciò cruciale ed è necessario sperimentare diversi approcci e combinazioni prima di definire la pipeline finale. Per questo motivo, sono stati introdotti algoritmi di deep learning tra cui le *reti neurali convoluzionali*, le cui architetture sono composte principalmente da due blocchi:

- blocco *convoluzionale*, il cui scopo è quello di apprendere la feature map ottimale in funzione dei dati ricevuti in input;
- blocco *denso*, ovvero il vero e proprio classificatore/regressore, generalmente un *percettrone multistrato*.

Il vantaggio principale di questa struttura è che, salvo alcune manipolazioni per la regolarizzazione dei dati, la feature map è parte della CNN e come tale viene appresa direttamente in fase di training, permettendo così al modello di riconoscere pattern e correlazioni complesse nei campioni. Il meccanismo di apprendimento delle CNN verrà spiegato più nel dettaglio nella sezione 4.2.1 analizzando l'architettura della nota rete neurale *LeNet-5*.

4.1.2 PARAFAC come feature map

Nel deep learning le feature map vengono apprese direttamente dai modelli con il fine di ottimizzarne le performance. Similmente, la decomposizione PARAFAC può essere utilizzata non solo per ridurre la dimensionalità di dati multilineari, ma per estrarre anche relazioni complesse tra i dati sfruttandone le proprietà ottenendone una rappresentazione univoca. Infatti, le ipotesi relative all'unicità della decomposizione a meno di permutazioni e riscalamenti sono deboli, per cui l'univocità è garantita dalla PARAFAC stessa. Inoltre, un ulteriore vantaggio dell'utilizzo della PARAFAC è legato al fatto che, come mostrato nel capitolo 3, è sufficiente una frazione del dataset affinché sia possibile ottenere una buona approssimazione dei fattori paralleli, addestrando così i modelli anche su possibili perdite di informazione causate dalla proiezione nel nuovo sistema di riferimento. L'approccio che si propone è una generalizzazione della semplice applicazione della PARAFAC sul tensore dei dati. Richiamando il dataset *Fetch Olivetti Faces* a scopo dimostrativo, si supponga di generare una griglia uniforme che suddivida un'immagine di N sotto-immagini, come mostrato in Figura 4.1.

Figura 4.1: Esempio di suddivisione in griglia delle immagini del dataset *Fetch Olivetti Faces*.

L'immagine è un quadrato il cui lato, in generale, è composto da W pixel, mentre ogni quadrato generato dalla griglia ha i lati di lunghezza K. Assumendo che le celle della griglia possano sovrapporsi a patto che la distanza tra due celle adiacenti sia sempre di S pixel, allora

$$N = \left(\left\lceil \frac{W-K}{S} \right\rceil + 1 \right)^2 . \tag{4.3}$$

Nel caso della Figura 4.1 il lato è lungo 64 pixel, per cui, per frazionare l'immagine con una griglia composta da 4×4 celle, necessariamente $K = 16$ e $S = 16$.

Osservazione. La griglia non è applicata su ogni singola immagine, ma bensì su tutto il dataset. Se quest'ultimo è un tensore $\mathcal{X} \in \mathbb{R}^{I_1 \times N \times N}$, allora ogni singolo quadrato della griglia si estende lungo la direzione dei samples, risultando un tensore di dimensione $I_1 \times K \times K$.

L'obiettivo, quindi, è quello di calcolare le decomposizioni di R fattori di ogni singolo sotto-tensore generato dalla griglia, ottenendo così per ognuna delle N regioni una rappresentazione che ne cattura i tratti più importanti e distintivi. Ogni cella $\mathcal{X}_n \in \mathbb{R}^{I_1 \times K \times K}$ viene quindi decomposta come:

$$\mathcal{X}_n \overset{\epsilon}{\approx} \left\| \boldsymbol{V}^{(1)}, \boldsymbol{V}^{(2)}, \boldsymbol{V}^{(3)} \right\|, \ \epsilon > 0, \tag{4.4}$$

con $\boldsymbol{V}^{(1)} \in \mathbb{R}^{I_1 \times R}$ e $\boldsymbol{V}^{(2)}, \boldsymbol{V}^{(3)} \in \mathbb{R}^{N \times R}$. In questo caso non si parla più di factor face, ma si utilizza la stessa nomenclatura delle CNN, per cui il prodotto $\boldsymbol{V}^{(2)}(\boldsymbol{V}^{(3)})^T \in \mathbb{R}^{N \times N}$ è detto *kernel*. Ogni kernel è quindi unico e ed è fondamentalmente il sistema di riferimento rispetto al quale le rispettive regioni dei nuovi campioni vengono proiettate. In funzione della griglia rappresentata in Figura 4.1, applicando il metodo descritto con decomposizioni di 4 fattori al dataset *Fetch Olivetti Faces* si ottengono quindi i 16 kernel riportati in Figura 4.2.

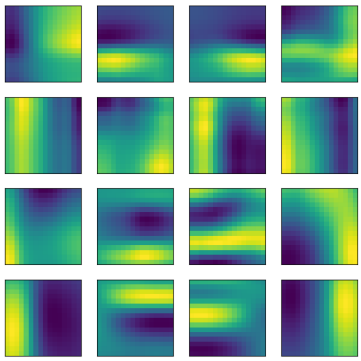

Figura 4.2: Kernels appresi decomponendo il dataset *Fetch Olivetti Faces* rispetto alla griglia della Figura 4.1.

Ogni kernel proietta ogni sotto tensore $I_1 \times K \times K$ in una matrice $I_1 \times R$. Di conseguenza, per ogni sample si ottiene un vettore di R elementi per cella, e lo stesso risulterà essere un tensore $R \times \sqrt{N} \times \sqrt{N}$. Applicando i kernel appresi nel precedente esempio al volto 4.1 si ottiene il tensore $4 \times 4 \times 4$ riportato in Figura 4.3.

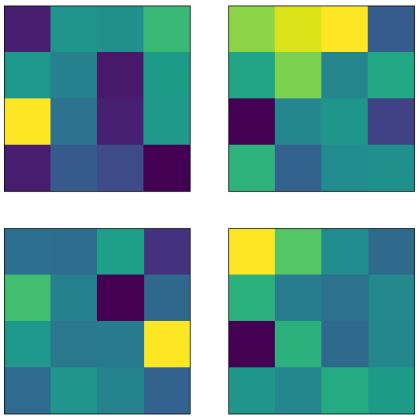

Figura 4.3: Risultato dell'applicazione dei kernel della Figura 4.2 al volto 4.1; le matrici rappresentate sono le matrici del tensore risultante in ordine crescente di indice rispetto al primo asse.

Anche in questo caso la dimensionalità delle singole immagini ha subito una notevole riduzione, risultando composte da $\frac{1}{64}$ del numero originale di pixel.
La feature map così descritta verrà indicata nella trattazione con il nome di *feature map*

tensoriale ed è definita come segue:

$$\begin{aligned} \mathcal{F}_R : \mathbb{R}^{I_1 \times I_2 \times I_3} &\longrightarrow \mathbb{R}^{I_1 \times R \times J_1 \times J_2} \\ \mathcal{X} &\longmapsto \mathcal{F}(\mathcal{X}) = \mathcal{Y}. \end{aligned} \tag{4.5}$$

4.1.3 Applicazione di feature mapping tensoriale ed SVM al MNIST dataset

In questa sezione si applicherà il feature map tensoriale al dataset *MNIST Digits*, comparandone le performance con quelle di un SVM con kernel radiale addestrato direttamente sui dati standardizzati.
Essendo le immagini 8×8 si sceglie di calcolare dei kernel di dimensione 4×4 per $R = 2$, la cui distanza tra essi (*stride*) è di 2 pixel. Applicando la (4.3) si ottengono un totale di 9 kernels, rappresentati in Figura 4.4 .

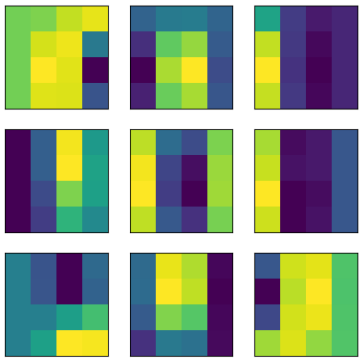

Figura 4.4: Kernels appresi decomponendo il dataset *MNIST Digits*.

Trasformando i dati si ottengono dei tensori di dimensione $2 \times 3 \times 3$, come mostrato in Figura 4.5, per un totale di 18 features, ovvero il 28% del numero di pixel originali. Si osserva come i dati trasformati siano caratterizzati da pattern distinti, differenti per ogni classe, indicando così che la maggior parte dell'informazione dei campioni sia stata estratta dai kernel ottenuti mediante l'applicazione della decomposizione PARAFAC.

In primo luogo, si mostrino i risultati relativi all'addestramento dell'SVM rispetto ai dati originali. Il problem set e le combinazioni di parametri testate sono gli stessi della sezione 3.2.3, ottenendo gli score riportati in Figura 4.6.
Per $\gamma = 0.001$ e $C \in \{1, 10, 100, 1000\}$ si ottiene uno score *f1* di 1.0 sul train set e di 0.98 sul test set.
Molto simili sono le performance del classificatore applicato ai dati trasformati, riassunte

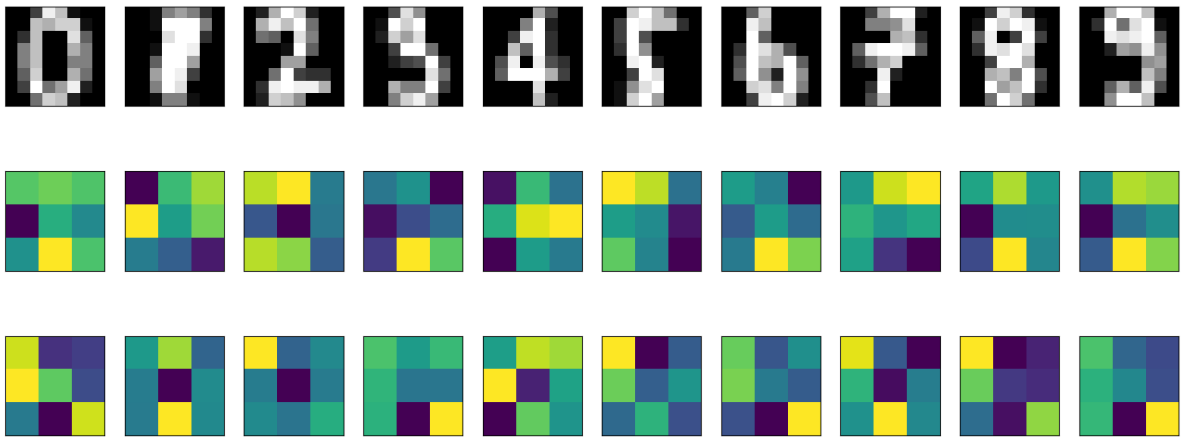

Figura 4.5: Samples del dataset *MNIST Digits* trasformati mediante feature map tensoriale.

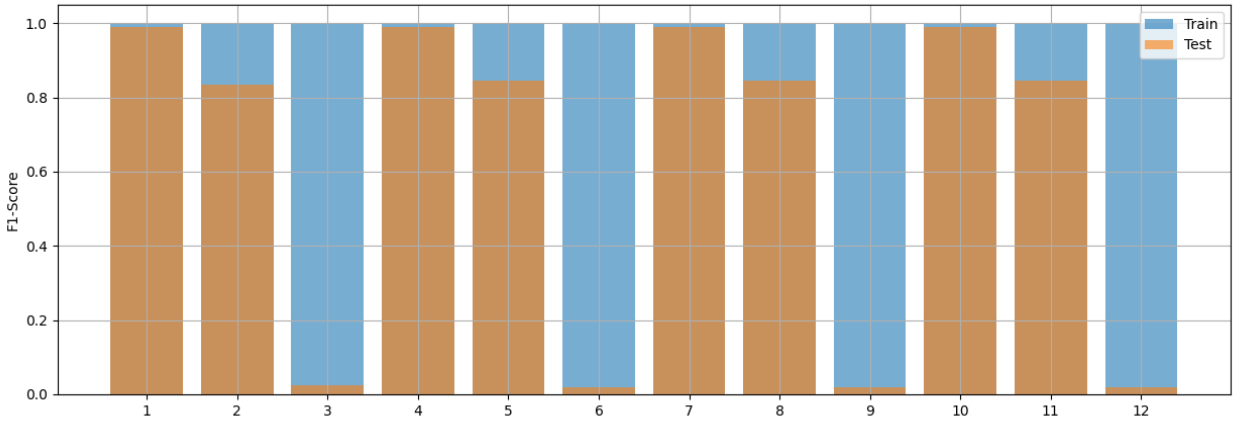

Figura 4.6: Risultati della grid search sui parametri dell'SVM testato sulle immagini standardizzate del dataset *MNIST Digits*.

dai barplot in Figura 4.7, i rispetto ai quali il massimo score sul train set è sempre 1.0, mentre sul test 0.97 per $C = 10, \gamma = 0.001$.
Anche in questo caso è stata testata la qualità del fitting dei kernel confrontandone gli score in funzione della frazione del dataset rispetto al quale sono stati calcolati.
Come mostrato in Figura 4.8 i kernel calcolati risultano estremamente stabili anche se

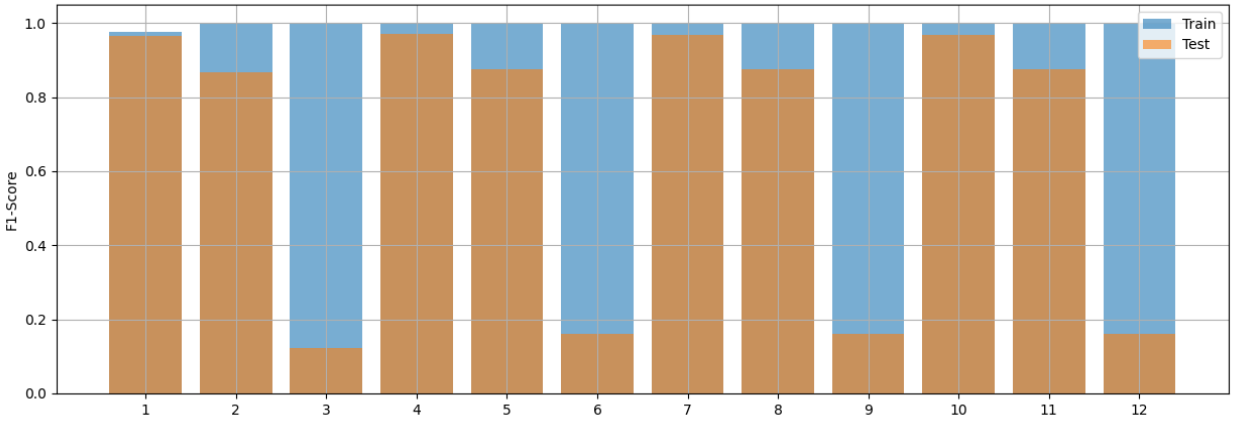

Figura 4.7: Risultati della grid search sui parametri dell'SVM testato sulle immagini trasformate mediante feature mapping tensoriale del dataset *MNIST Digits*.

appresi su piccole frazioni del dataset in questione, ottenendo sempre uno score sul test set che oscilla attorno a 0.96 e 0.97.

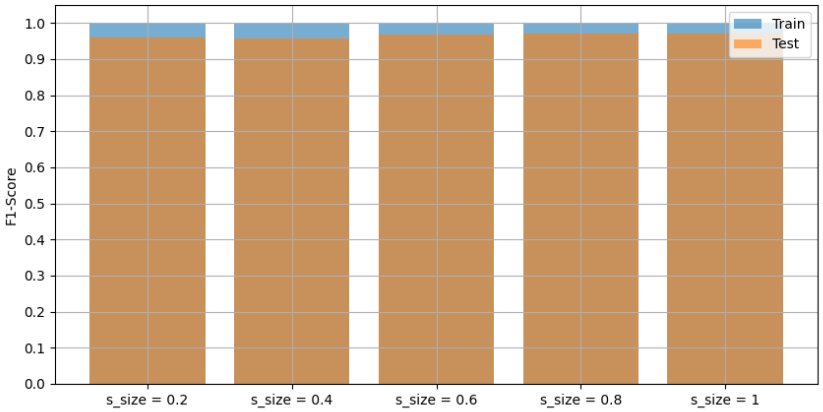

Figura 4.8: Risultati dell'SVM al variare della dimensione del sottoinsieme di dati su cui è stata applicata la feature map tensoriale per $C = 10, \gamma = 0.001$.

L'applicazione della PARAFAC a sotto-immagini di dimensione ridotta rispetto al dato originale ne riduce notevolmente il costo computazionale, convergendo a kernel tensoriali più stabili. Di conseguenza, se l'obiettivo è quello di ridurre la dimensionalità del dataset preservandone però la maggior parte dell'informazione, allora un approccio come il precedente permette una rappresentazione robusta e sintetica. Diversamente, la scelta di kernel

di dimensioni ridotte porta ad una minore riduzione del numero di features, ma ad una maggiore quantità di informazione estratta da ogni singola regione della griglia. Infatti, calcolando kernel tensoriali di dimensione $K = 2$ con stride $S = 1$ si ottiene uno score sul test set di 0.99 (Figura 4.9), mantenendo il risultato costante indipendentemente dal numero di samples rispetto al quale sono stati costruiti (Figura 4.10).

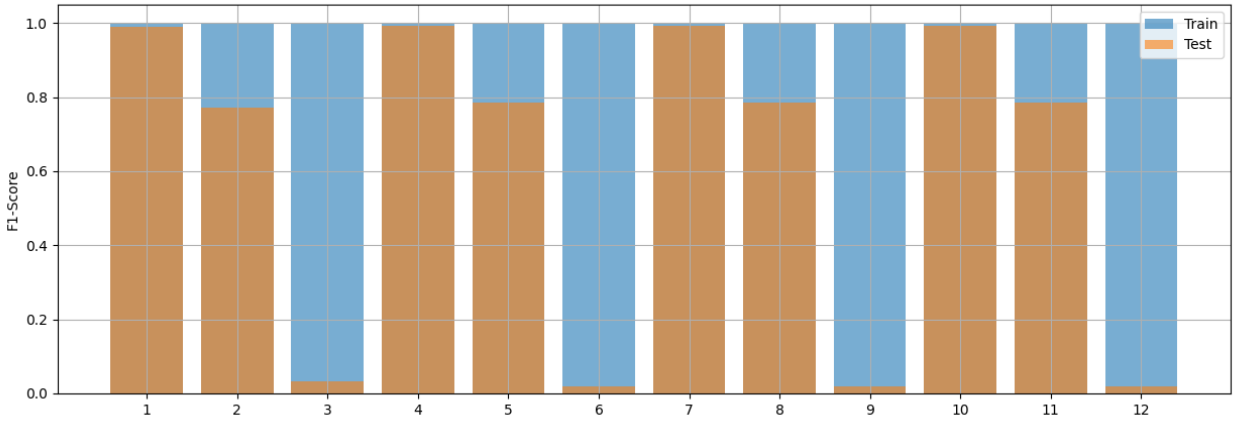

Figura 4.9: Risultati della grid search sui parametri dell'SVM testato sulle immagini trasformate mediante feature mapping tensoriale del dataset *MNIST Digits* con $K = 2, S = 1, R = 1$.

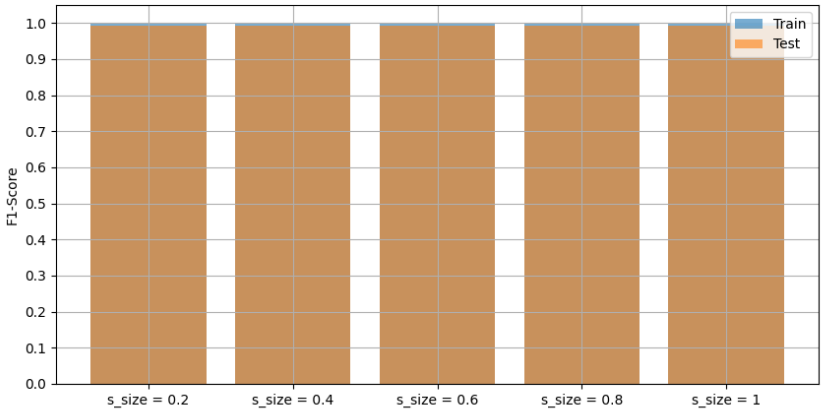

Figura 4.10: Risultati dell'SVM al variare della dimensione del sottoinsieme di dati su cui è stata applicata la feature map tensoriale per $C = 10, \gamma = 0.001$.

Quest'estensione dell'applicazione della PARAFAC, non unicamente al tensore complessivo, ma a sue sotto-regioni, permette di ottenere una rappresentazione robusta dei dati in funzione di pattern e features locali. Ogni kernel è caratterizzato dalla poliade (diade negli esempi presentati) che meglio rappresenta e sintetizza le informazioni del sotto-tensore decomposto. Un approccio simile è proposto nel deep learning nelle reti neurali convoluzionali, nel processo detto di *convoluzione*, durante il quale il modello calcola e costruisce la feature map ottimale.

4.2 Reti neurali convoluzionali

4.2.1 LeNet-5

Come anticipato nella sezione 4.1.1, le reti neurali convoluzionali sono dei modelli che permettono di semplificare la pipeline del preprocessing essendo composte non solo dal classificatore, ma anche dalla feature map, la quale viene appresa direttamente dai dati in fase di addestramento. La sua costruzione è molto simile a quanto descritto per la feature map tensoriale, differendo sulla composizione e struttura dei kernel, detti anche *filtri convoluzionali*.
Sia $\mathcal{X}^{(j)} \in \mathbb{R}^{D\times N\times M}$ il $j-$esimo campione di un dataset $\mathcal{X} \in \mathbb{R}^{I_1\times D\times N\times M}$, con I_1 numero di samples, D profondità e N, M, rispettivamente altezza e larghezza di ciascun sample. Un kernel di dimensione $K \leq \min\{N, M\}$ allora è un tensore $\mathcal{W} \in \mathbb{R}^{D\times K\times K}$. Differentemente dalla feature map tensoriale, lo stesso filtro viene applicato alle regioni di dimensione $D \times K \times K$ generate dalla griglia che suddivide il dato, mappando ciascuna regione in uno scalare mediante la funzione

$$\begin{aligned} \mathcal{F}_{\mathcal{W}} : \mathbb{R}^{D\times K\times K} &\longrightarrow \mathbb{R} \\ \mathcal{X}_n^{(j)} &\longmapsto \sum_{i_1,i_2,i_3} \left(\mathcal{X}_n^{(j)} \odot \mathcal{W}\right)_{i_1,i_2,i_3}. \end{aligned} \tag{4.6}$$

L'utilizzo di più filtri permette quindi di aumentare o diminuire la profondità dell'output della funzione concatenandone gli scalari ottenuti.
I kernel applicati ad uno stesso tensore costituiscono un *layer convoluzionale*. I tensori risultanti dall'applicazione di un filtro convoluzionale vengono a loro volta trasformati mediante una *funzione di attivazione*, la quale, introducendo non linearità nel modello, permette di apprendere ed estrarre relazioni più complesse tra i dati. Al termine di questo processo, i risultati sono soggetti ad un subsampling detto *pooling*, il quale, analogamente ad un kernel convoluzionale, permette di ridurre ulteriormente la dimensionalità degli output, selezionando il massimo valore per ogni regione (*max pooling*) o calcolandone la media (*average pooling*).
La struttura descritta costituisce un *blocco convoluzionale*. Più blocchi possono essere inseriti sequenzialmente al fine di ottimizzare le performance del classificatore, il quale in genere è un *percettrone multistrato*. L'intero modello viene addestrato mediante *backpropagation*, ovvero un metodo iterativo basato sulla *chain rule*, il quale aggiorna i parametri del modello in funzione del gradiente dell'errore di previsione.
Un esempio di rete neurale convoluzionale nota in letteratura è *LeNet*-5, la cui architettura è stata introdotta da *Yann LeCun* nel 1998, con lo scopo di migliorare le capacità di generalizzazione dei percettroni multistrato. LeNet-5 è specificatamente costruita per i dataset MNIST composti da 10 classi distinte ed immagini di dimensione $1 \times 32 \times 32$ ed è composta da:

- un layer convoluzionale composto da 6 kernel con $K = 5$, $S = 1$, mappando gli input in tensori $6 \times 28 \times 28$;
- un filtro di pooling con $K = S = 2$, riducendo la dimensione degli output del layer precedente in tensori $6 \times 14 \times 14$;

- un layer convoluzionale composto a sua volta da 16 kernel con $K = 5, S = 1$, dal quale si ottengono tensori $16 \times 10 \times 10$;
- un filtro di pooling di con $K = S = 2$, portando ad una dimensionalità finale di $16 \times 5 \times 5$;
- un percettrone multistrato composto da due layer centrali di dimensioni rispettivamente $120, 84$ ed un layer di output composto da 10 neuroni.

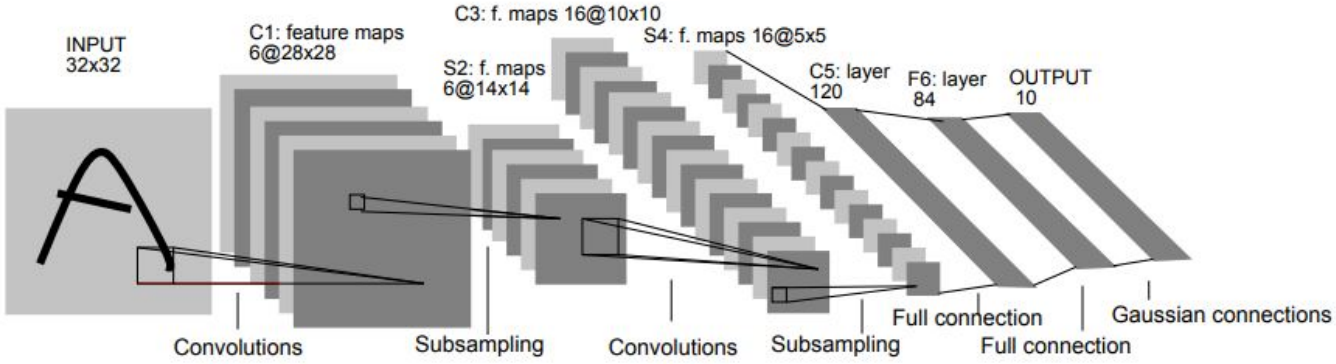

Figura 4.11: Architettura di LeNet-5.

Il numero di parametri da ottimizzare sono 60000, ma attualmente è comune l'utilizzo di reti neurali convoluzionali con centinaia di milioni di parametri, il cui addestramento è estremamente costoso sia in termini di tempo, costo ed energia.

4.2.2 PFNet-RBF

Nella sezione 4.1.2 è stato proposto un approccio alternativo della PARAFAC, rendendo possibile la costruzione di una feature map appresa dai dati sfruttando le proprietà della decomposizione. I kernel ottenuti permettono di descrivere le rispettive regioni del dataset in maniera univoca per via dell'unicità della PARAFAC, rappresentando ciascuna di esse rispetto al sistema di riferimento che meglio ne approssima i pattern principali. Inoltre, diversamente dalle reti neurali convoluzionali, i kernel non sono tensori multimodali, ma più semplicemente R poliadi, con R ordine della decomposizione, riducendo notevolmente il numero di parametri del modello. Un ulteriore vantaggio è dato dal fatto che ogni regione è associata ad un kernel tensoriale differente, adattato ad essa.
Si propone quindi un'architettura che sostituisce i blocchi convoluzionali di una rete neurale convoluzionale con una singola feature map tensoriale, seguita a sua volta da un percettrone multistrato. In particolare, il percettrone è composto da due layer di dimensione, rispettivamente, N_{features} e N_{classes}, ovvero il numero di features totali ottenute vettorizzando l'output del feature map tensoriale ed il numero di classi. Per questa semplice

architettura si è scelto di utilizzare come funzione di attivazione tra i due layer

$$\begin{aligned} \phi : \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto e^{-\frac{1}{2}x^2}, \end{aligned} \tag{4.7}$$

da cui prende il nome di *PFNet-RBF*. Infatti, visti i miglioramenti in termini di classificazione ottenuti dall'SVM successivamente all'applicazione di un kernel RBF ai campioni trasformati mediante feature map tensoriale, l'obiettivo del primo strato del percettrone è quello di emulare la non linearità introdotta dal kernel trick. Siccome la funzione (4.7) è pari, valori simmetrici rispetto all'origine vengono penalizzati ugualmente, per cui risulta utile standardizzare l'output del primo layer in modo da stabilizzarne i valori. In particolare, si osserva che l'argomento di ϕ risulta essere $(x - \mu)/\sigma$. Di conseguenza, la funzione di attivazione diventa

$$\begin{aligned} \phi : \mathbb{R} &\longrightarrow \mathbb{R} \\ x &\longmapsto e^{-\frac{1}{2\sigma}(x-\mu)^2}, \end{aligned} \tag{4.8}$$

ovvero la densità di una distribuzione standard a meno del fattore di correzione, in modo tale che l'output della funzione di attivazione sia compreso tra 0 e 1. Tuttavia, non è possibile conoscere a priori i valori assunti dalle attivazioni del layer dal momento che gli stessi parametri che lo compongono vengono ottimizzati per ogni batch e ad ogni epoca. Per affrontare tale problema si utilizza una tecnica chiamata *batch normalization*. La batch normalization è un'operazione che normalizza gli input di ogni batch all'interno di una rete neurale. Questa tecnica prevede il calcolo delle medie e delle varianze delle features di input all'interno di ogni batch al fine di standardizzarle. Questo processo garantisce una distribuzione simile a una variabile casuale con media 0 e varianza 1 (a meno di un fattore di correzione). Inoltre, vengono applicati ulteriori due fattori, rispettivamente, di scala e di shift appresi durante la fase di addestramento, al fine di adattare i valori normalizzati, se necessario.
L'equazione della batch normalization è:

$$\mathrm{BN}(x_i) = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta, \tag{4.9}$$

dove $\mathrm{BN}(x_i)$ rappresenta il valore normalizzato dell'input x_i, γ e β sono parametri appresi dal modello, μ_B è la media del batch corrente, σ_B^2 è la varianza del batch e ϵ è un valore molto piccolo aggiunto per evitare valori nulli a denominatore.
Utilizzando la batch normalization, si ottiene una maggiore stabilità e convergenza durante l'addestramento di reti neurali. Di conseguenza, applicando tale tecnica ad un layer denso con una funzione di attivazione RBF (4.7), si normalizzano gli output del layer e si mantiene una distribuzione stabile dei valori durante l'addestramento.
Quest'architettura è quindi composta dai $N_{\text{kernels}} \times 2KR$ parametri dei kernel tensoriali, sommati ai $N_{\text{features}}^3 N_{\text{classes}}$ del percettrone multistrato.

4.2.3 Confronto LeNet-5 e PFNet

Le due reti neurali sono state testate sul dataset *Fashion MNIST*, composto da 10 classi bilanciate. Il training set è composto da 60000 campioni, mentre il test set da 10000. Il

dataset contiene immagini 28×28 in scala di grigi rafFiguranti capi di abbigliamento.

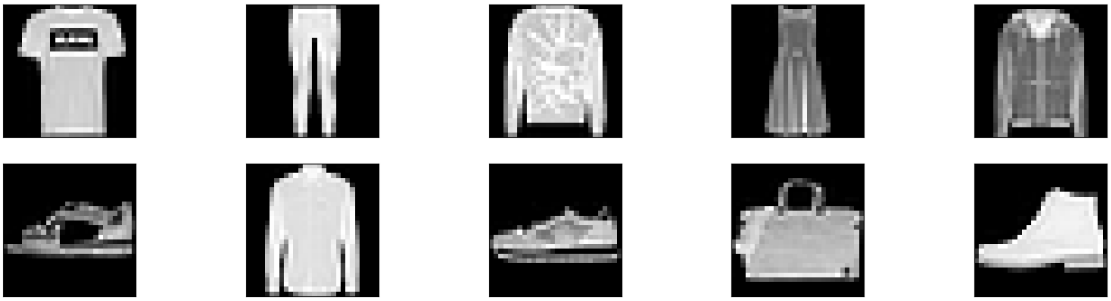

Figura 4.12: Samples del dataset *Fashion MNIST*.

Relativamente ai modelli, LeNet-5 è stata testata utilizzando come funzione di attivazione la tangente iperbolica, mentre PFNet-RBF con kernel tensoriali di ordine $R = 2$, di dimensione $K = 4$ e stride $S = 3$. Di conseguenza, dall'equazione (4.3) il numero totale di kernel risulta essere 81, trasformando ciascuna immagine in un tensore $2 \times 9 \times 9$. Il modello è composto quindi da 29160 parametri, 1296 dei quali appresi durante il fitting della PARAFAC, mentre i restanti 27864 mediante backpropagation. Richiamando i risultati precedenti, la feature map tensoriale è robusta anche se calcolata su un sottoinsieme del dataset, per cui, vista la numerosità dei campioni, si utilizzerà solo il 20% del dataset per il fitting. Per entrambi i modelli, l'output di ogni kernel (sia convoluzionale che tensoriale) è standardizzato, aggiornando le rispettive medie e deviazioni standard campionarie ad ogni batch utilizzando la batch normalization. Il training consta di 30 epoche con batch di 128 campioni. Per concludere, l'ottimizzatore utilizzato è l'*Adam* con i parametri di default di *PyTorch*.
I risultati sono riportati nelle Figure 4.13 e 4.14, dalle quali si osserva l'estrema similarità tra i due modelli in termini di prestazioni. PFNet-RBF, con meno della metà dei parametri di LeNet-5, ha ottenuto un'accuracy di circa il 5% meno rispetto ad essa, limitandone l'overfitting, mantenendo uno score molto simile a LeNet-5 sul test set (circa 1% in meno), ma con uno scarto di accuracy rispetto al train set di circa il 6% contro il 10% della CNN.

I risultati mostrano come il feature map tensoriale abbia permesso ad un percettrone multistrato composto da due layer di ottenere performance prossime a quelle di una rete neurale convoluzionale nota in letteratura. Il vantaggio dell'architettura è che l'apprendimento della feature map è indipendente dalla fase di training del percettrone, richiedendo un numero ridotto di samples e di parametri.

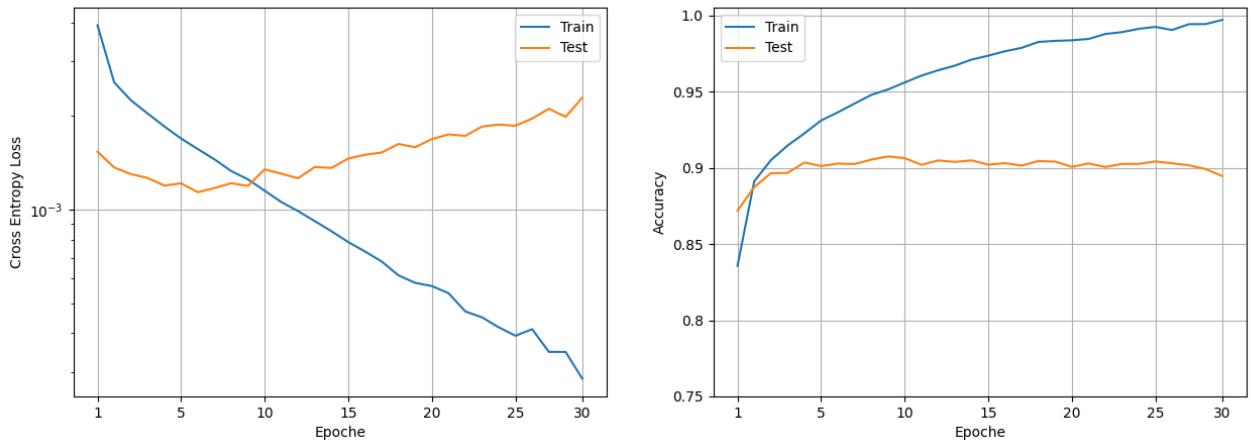

Figura 4.13: Risultati del training di LeNet-5 sul dataset *Fashion MNIST*.

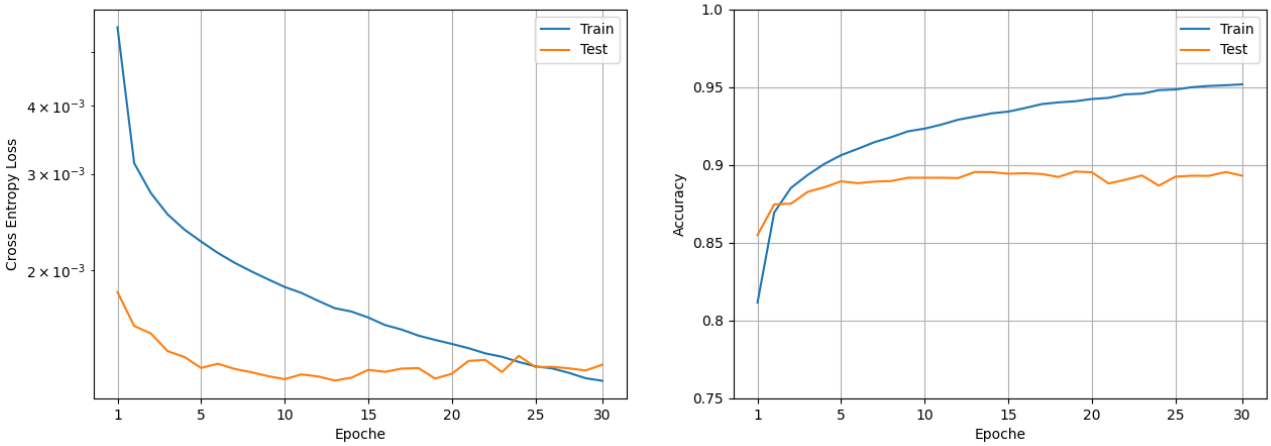

Figura 4.14: Risultati del training di PFNet-RBF sul dataset *Fashion MNIST*.

Capitolo 5

Conclusioni

In conclusione, la ricerca condotta in questo elaborato ha evidenziato l'importanza e l'applicabilità del calcolo tensoriale e dell'analisi tensoriale nell'elaborazione e nell'analisi dei dati multimodali. Grazie alle opere di eminenti studiosi come Kruskal, Harshman, Carroll, Chang e Tucker, sono stati sviluppati metodi di decomposizione tensoriale, in particolare la PARAFAC, che permettono di sintetizzare le caratteristiche dei dati e di estrarne le correlazioni più significative.
Le applicazioni presentate nell'elaborato hanno dimostrato che a partire da un approccio più canonico, basato sulla decomposizione PARAFAC, è possibile costruire modelli e metodi in grado di preservare la struttura multidimensionale dei dati multilineari senza dover ricorrere a vettorizzazioni o reshape. Inoltre, la PARAFAC si è rivelata un metodo robusto per la rappresentazione dei dati rispetto a sistemi di riferimento di dimensione inferiore, indipendentemente dalla numerosità delle osservazioni. Questa caratteristica consente di ridurre non solo la dimensionalità dei dati, ma anche i tempi di apprendimento degli algoritmi di machine learning.
L'inclusione della PARAFAC nelle pipeline e negli algoritmi del deep learning, come le reti neurali artificiali, può contribuire a migliorarne l'interpretabilità e semplificarne le architetture, in un contesto in cui sempre più spesso vengono utilizzati modelli complessi e blackbox per ottenere solo piccoli miglioramenti delle performance.
Nel corso di questa ricerca, è stata presentata una descrizione approfondita della PARAFAC dal punto di vista matematico e implementativo, evidenziandone le proprietà attraverso diverse applicazioni, incluso l'utilizzo in architetture neurali denominate più generalmente PFNet. Nonostante PFNet-RBF fosse un modello relativamente semplice, i risultati ottenuti dimostrano come l'analisi tensoriale possa fornire benefici significativi quando si affronta il problema della *curse of dimensionality*.
In conclusione, questa ricerca ha contribuito ad arricchire la conoscenza sull'analisi tensoriale e ha dimostrato le sue potenzialità nella risoluzione di sfide complesse legate all'elaborazione e all'analisi dei dati multidimensionali. Si spera che i risultati ottenuti in questo lavoro possano stimolare ulteriori ricerche nel campo dei big data e dell'analisi tensoriale, aprendo nuove strade per l'applicazione di questi metodi in settori più estesi e complessi.

Bibliografia

1. Aravindan Vijayaraghavanz Aditya Bhaskara, Moses Charikary. Uniqueness of tensor decompositions with applications to polynomial identifiability. 2013.

2. Richmond Alake. Understanding and implementing lenet-5 cnn architecture (deep learning). *Towards Data Science*, 2020. URL https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342.

3. Monica De Angelis and Enrico Mazziotti. Breve manuale di calcolo tensoriale per le applicazioni.

4. Stefano Berrone, Francesco Della Santa, and Ulderico Fugacci. Computational linear algebra - materiale didattico, 2022.

5. Barbara Caputo. Machine learning and deep learning - materiale didattico, 2022.

6. Andrzej Cichoki. Era of big data processing: A new approach via tensor networks and tensor decompositions. 2014.

7. P. Comon, X. Luciani, and André L. F. de Almeida. Tensor decompositions, alternating least squares and other tales. *Journal of Chemometrics*, 2009. URL https://hal.science/hal-00410057.

8. Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press.

9. Gérard Favier and André L. F. de Almeida. Overview of constrained parafac models. *EURASIP Journal on Advances in Signal Processing*, 2014. URL https://asp-eurasipjournals.springeropen.com/articles/10.1186/1687-6180-2014-142.

10. Richard A. Harshman. *Foundation of the PARAFAC Procedure: Models and Conditions for an "Explanatory" Multi-Modal Factor Analysis*. UCLA.

11. Richard A. Harshman and Margaret E. Lundy. Parafac: Parallel factor analysis. 1994. URL https://psychology.uwo.ca/faculty/harshman/csda94.pdf.

12. Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015. URL http://arxiv.org/abs/1502.03167v3.

13. Tamara G. Kolda and Brett W. Bader. *Tensor Decompositions and Applications*. Society for Industrial and Applied Mathematics.

14. Anh Huy Phan, Petr Tichavský, and Andrzej Cichocki. Candecomp/parafac decomposition of high-order tensors through tensor reshaping. 2012. URL http://arxiv.org/abs/1211.3796v1.

15. Sandra Pieraccini. Numerical optimization for large scale problems and stochastic optimization - materiale didattico, 2021.

16. Bro Rasmus. Parafac. tutorial and applications. *Chemometrics and intelligent laboratory systems*.

17. John A. Rhodes. A concise proof of kruskal's theorem on tensor decomposition. URL http://arxiv.org/abs/0901.1796v1.

18. Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning.

19. Gilbert Strang. *Linear Algebra and its Applications*. Elsevier, a.

20. Gilbert Strang. *Linear Algebra and Learning from Data*. Wellesley - Cambridge Press, b.

21. Mingxing Tan and Quoc V.Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2020. URL https://arxiv.org/pdf/1905.11946.pdf.