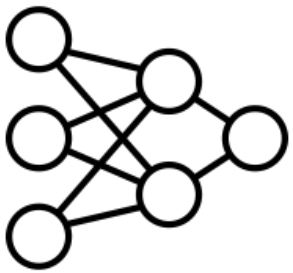
Classificazione Cellule Ematiche

Alessandro P.
Gaia M.
Antonio S.

Unipa

Settembre 2024



Indice

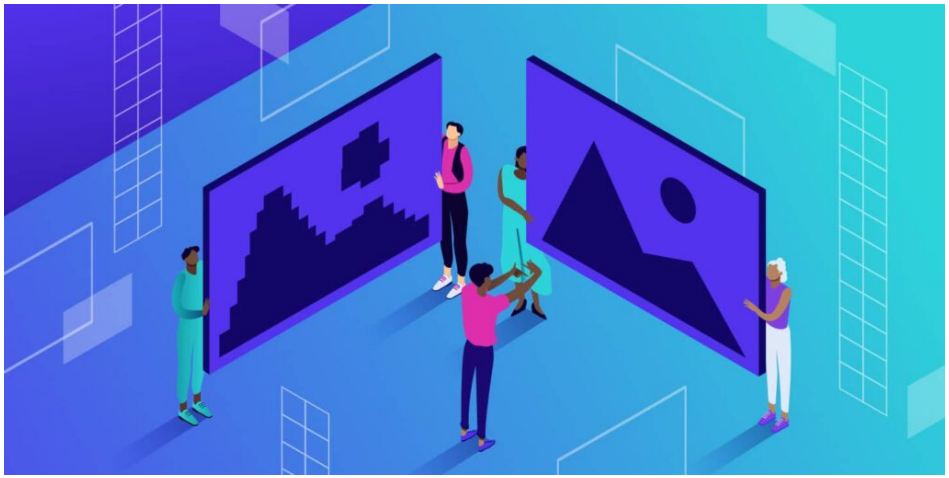
Introduzione

In ambito medico, l'avvento dell'intelligenza artificiale ha consentito di migliorare e velocizzare processi di analisi e diagnosi mediche. Una delle applicazioni più promettenti è la classificazione automatizzata delle cellule ematiche. Posta in input un'istantanea, a un'intelligenza artificiale, scattata su un campione di sangue è possibile valutare la morfologia, le percentuali numeriche e lo stadio di maturazione della popolazione di globuli rossi, leucociti e piastrine.

Nello specifico ci occuperemo di descrivere come una rete neurale opera catalogando le diverse tipologie di leucociti, sani o tumorali, su un dataset di immagini raccolte da campioni di sangue.

Metodi per la Compressione delle Immagini

Per poter gestire una grande mole di immagini (quindi una grande quantità di dati rappresentati sottoforma di matrici), bisogna ridurre le dimensioni di queste ultime al fine di ridurre la complessità temporale e spaziale per l'elaborazione. I metodi che utilizzeremo per applicare la compressione sono la **fattorizzazione QR** e **SVD**. Una volta compresse possiamo fornire l'input alla rete neurale per stimare la classificazione.



Fattorizzazione QR

La fattorizzazione QR è una tecnica di decomposizione di una matrice quadrata o rettangolare. Essa scompone una matrice A in un prodotto di due matrici: una matrice ortogonale Q e una matrice triangolare superiore R.
Consideriamo una matrice A di dimensioni $m \times n$, dove $m \geq n$. La decomposizione QR di A è data da:

$$A = QR$$

dove:

- Q è una matrice **ortogonale** $m \times m$, tale che $Q^T Q = I$, dove I è la matrice identità.
- R è una matrice **triangolare superiore** $m \times n$.

Fattorizzazione QR

La fattorizzazione QR è utilizzata in varie applicazioni, come la risoluzione di sistemi di equazioni lineari, la regressione lineare, il calcolo degli autovalori di una matrice e per la compressione delle immagini, che è ciò in cui ci soffermeremo in questo progetto.

Fattorizzazione QR

Per ottenere questa decomposizione utilizziamo le **matrici elementari di Householder** nella fattorizzazione QR per ridurre una matrice a una forma triangolare superiore. La matrice elementare di Householder è una matrice ortogonale che, quando moltiplicata per un vettore, lo **trasforma** a una forma in cui tutte le componenti, ad eccezione della prima, vengono annullate. Questa proprietà viene sfruttata per annullare gli elementi sotto una certa posizione nella matrice, ottenendo così una matrice triangolare superiore. Queste trasformazioni permettono di ridurre la correlazione tra i pixel e quindi eliminare la ridondanza spaziale, contribuendo così alla compressione dell'immagine.

Fattorizzazione QR

Applicando l'operazione a una colonna della matrice, gli elementi sotto la diagonale principale vengono annullati, e questo processo viene ripetuto iterativamente fino a quando la matrice non viene trasformata in forma triangolare superiore. Le matrici elementari di Householder sono più efficienti e stabili numericamente rispetto alle matrici elementari di Givens, per questo motivo abbiamo preferito utilizzarle per la decomposizione QR all'interno del nostro progetto.

Fattorizzazione QR

La **matrice elementare di Householder** è la matrice di ordine n:

$$H = I - 2ww^T$$

dove w è un vettore unitario $||w||_2 = 1$.
Proprietà:

- **Simmetria**: Si verifica che $H^T = (I - 2ww^T)^T = I - 2(w^T)^T w^T = I - 2ww^T = H$, dimostrando che H è simmetrica.
- **Ortogonalità**: La moltiplicazione di H per la sua trasposta porta a $H^T H = (I - 2ww^T)^T(I - 2ww^T) = (I - 2ww^T)(I - 2ww^T) = (I - 4ww^T + 4ww^T) = I$.

Fattorizzazione QR

- **Involutorietà**: H è involutoria poiché $H^2 = (I - 2ww^T)(I - 2ww^T) = I - 4ww^T + 4ww^T ww^T = I$, quindi $H^2 = I$, il che implica che $H = H^{-1}$, ovvero che coincide con la sua inversa.

Funzionamento: la trasformazione su un vettore x viene eseguita utilizzando l'equazione $Hx = -\sigma e_1$, dove σ è uno scalare e e_1 è il primo vettore della base canonica di $\mathbb{R}^n$. Il valore di σ è uguale alla norma del vettore x, cioè $\sigma = ||x||_2$.

Fattorizzazione QR

Partendo dall'equazione $Hx = -\sigma e_1$, dove e_1 è un vettore di dimensioni $n \times 1$, vogliamo calcolare il prodotto matrice-vettore Hx con se stesso. Questo è rappresentato come $(Hx)^T(Hx)$.

Utilizziamo l'espressione per Hx data dall'equazione, e otteniamo:

$$(Hx)^T(Hx) = (-\sigma e_1)^T(-\sigma e_1)$$

Poiché H è una matrice ortogonale, la norma al quadrato di x rimane invariata, quindi $\|x\|^2 = \sigma^2$.

Espandendo l'espressione Hx, otteniamo:

$$Hx = x - 2ww^T x$$

Fattorizzazione QR

Sostituendo l'espressione ottenuta per Hx nell'equazione $(Hx)^T(Hx)$, otteniamo una nuova espressione che coinvolge w, la cui scelta influenzerà la trasformazione Hx.

Risolvendo l'equazione ottenuta per w, otteniamo il valore di w che consente la trasformazione $Hx = -\sigma e_1$.

Costruzione del vettore v: Per soddisfare l'equazione $Hx = -\sigma e_1$, costruiamo un vettore v che ha le stesse componenti di x, eccetto la prima componente, alla quale viene aggiunto σ o $-\sigma$, a seconda del segno di x_1.

Calcolo di Hx: Utilizzando l'espressione $Hx = x - 2ww^T x$ calcoliamo Hx come:

$$Hx = x - 2w(w^T x)$$

Fattorizzazione QR

Complessità lineare: La trasformazione di un vettore tramite la matrice elementare di Householder ha una complessità computazionale lineare. Questo perché coinvolge solo operazioni di somma e moltiplicazione scalare, il che riduce gli errori numerici e mantiene una complessità computazionale ragionevole.

Fattorizzazione QR

Per la **fattorizzazione QR di una matrice A**, ricordiamo che è possibile scrivere una qualunque matrice A come:

$$A = [a_1\ a_2\ ...\ a_n]$$

dove $a_1 = [a_{11}, a_{21}, ..., a_{n1}]^T$, $a_2 = [a_{12}, a_{22}, ..., a_{n2}]^T$, ... ,
$a_n = [a_{1n}, a_{2n}, ..., a_{nn}]^T$, ovvero a_1 è la prima colonna della matrice A e così dicendo per le altre.
Perciò, sfruttando le matrici elementari di Householder precedentemente definite, è possibile calcolare H_1 tale che:

$$H_1 a_1 = -\sigma_1 e_1, \qquad con \qquad \sigma_1 = ||a_1||_2$$

Fattorizzazione QR

Applicando a sinistra di A tale matrice, si ottiene una seconda matrice:

$$A^{(2)} = H_1 A^{(1)} = [a_1^{(2)}, a_2^{(2)}, ..., a_n^{(2)}]$$

in cui la prima colonna è trasformata in una forma triangolare. H_1 annulla le componenti del primo vettore colonna al di sotto della prima componente, trasformando $A^{(1)}$ in $A^{(2)}$.

$$a_1^{(2)} = \sigma_1 e_1$$

Di conseguenza la matrice appena ottenuta ha la struttura:

$$A^{(2)} = \begin{bmatrix} -\sigma_1 & a^T \\ 0 & \hat{A}^{(2)} \end{bmatrix}$$

$a = [a_{21}, ..., a_{n1}]^T$

Fattorizzazione QR

Reiterando il procedimento per la sottomatrice $\hat{A}^{(2)}$ si otterrà una matrice elementare $\hat{H}$ di dimensione minore. Tale matrice sarà orlata mediante le righe e le colonne di una matrice identità per ottenerne una del tipo:

$$H^{(2)} = \begin{bmatrix} 1 & 0^T \\ 0 & \hat{H}^{(2)} \end{bmatrix}$$

La matrice H_2 viene applicata alla matrice trasformata da H_1, lasciando inalterate la prima riga e la prima colonna e modificando il resto della matrice.

Fattorizzazione QR

Quindi al *k-esimo* passo la matrice risulta:

$$H^{(i)} = \begin{bmatrix} I^{(k-1,k-1)} & \Omega^{(k-1,n-k+1)} \\ \Omega^{(n-k+1,k-1)} & \hat{H}^{(n-k+1,n-k+1)} \end{bmatrix}$$

Al termine dell'algoritmo si ha che l'ultima matrice ottenuta è evidentemente una matrice triangolare superiore R:

$$R = A^{(n)} = H_{n-1}A^{(n-1)} = \ldots = H_{n-1}H_{n-2}\ldots H_1 A$$

Fattorizzazione QR

Poichè $Q = H_1 H_2 \dots H_{n-1}$ è ortogonale in quanto prodotto di matrici ortogonali, la relazione appena scritta implica:

$$A = QR$$

Nel caso in cui la matrice A sia quadrata l'algoritmo si arresta dopo $n-1$ passi; se invece la matrice è rettangolare, l'algoritmo si arresta al passo n, poichè tale ulteriore passo è necessario per azzerare gli elementi dell'ultima colonna di A.

Codice MATLAB - Fattorizzazione QR (Parte 1)

```
function [Q, R] = fattorizzazioneQR(A)
    [m, n] = size(A);
    % Inizializzazione delle matrici
    Q = eye(m);
    R = A;
    for i = 1 : n
        % Calcolo Householder
        x = R(i:m, i);
        e = zeros(length(x), 1);
        e(1) = 1; % Imposto ad 1 il primo elemento
        v = sign(x(1)) * norm(x) * e + x;
        % il segno del primo elemento di x * norma aggiungendo x al
        % risultato, v punta in direzione opposta di v ma con la
            stessa
```

Codice MATLAB - Fattorizzazione QR (Parte 2)

```
        % norma
        v = v / norm(v);
        % Aggiorna la sottomatrice di R
        R(i:m, i:n) = R(i:m, i:n) - 2 * v * (v' * R(i:m, i:n));
        % Aggiorna la sottomatrice di Q
        Q(i:m, :) = Q(i:m, :) - 2 * v * (v' * Q(i:m, :));
    end
end
```

Fattorizzazione QR

Una volta ottenute le matrici Q e R, il sistema lineare $Ax = b$ può essere risolto in due passaggi: Si risolve il sistema $Qy = b$ per ottenere y che equivale a trasformare il vettore b: $y = Q^T b$. Si risolve il sistema $Rx = y$ per ottenere x, che è la soluzione del sistema originale $Ax = b$.
La **complessità computazionale** della fattorizzazione QR è $O(2n^3/3)$.
Tuttavia, la fattorizzazione QR è stabile ed è sempre possibile garantendo una soluzione accurata al sistema lineare.

Fattorizzazione QR

Interpretazione Geometrica della Riflessione

La moltiplicazione di una matrice di riflessione per un vettore è equivalente a riflettere tale vettore attraverso l'iperpiano perpendicolare a $\mathbf{w}$.

- **Matrice di riflessione:** Una matrice di riflessione H è costruita in modo tale da riflettere un vettore attraverso l'iperpiano in questione.
- **Iperpiano Π:** Un iperpiano Π in $\mathbb{R}^n$ è un sottospazio affine di dimensione $n-1$. Se $\mathbf{w}$ è un vettore in $\mathbb{R}^n$, l'iperpiano Π ortogonale a $\mathbf{w}$ contiene tutti i vettori che sono ortogonali a $\mathbf{w}$.
- **Vettore riflesso:** Se $\mathbf{x}$ è un vettore in $\mathbb{R}^n$, e H è una matrice di riflessione rispetto al vettore $\mathbf{w}$, allora il vettore riflesso $\mathbf{y} = H\mathbf{x}$.

Fattorizzazione QR

La riflessione del vettore $\mathbf{x}$ attraverso l'iperpiano Π ortogonale a $\mathbf{w}$ è data dall'operazione:

$$\mathbf{y} = H\mathbf{x} = (I - 2\mathbf{w}\mathbf{w}^T)\mathbf{x}$$

Dove:

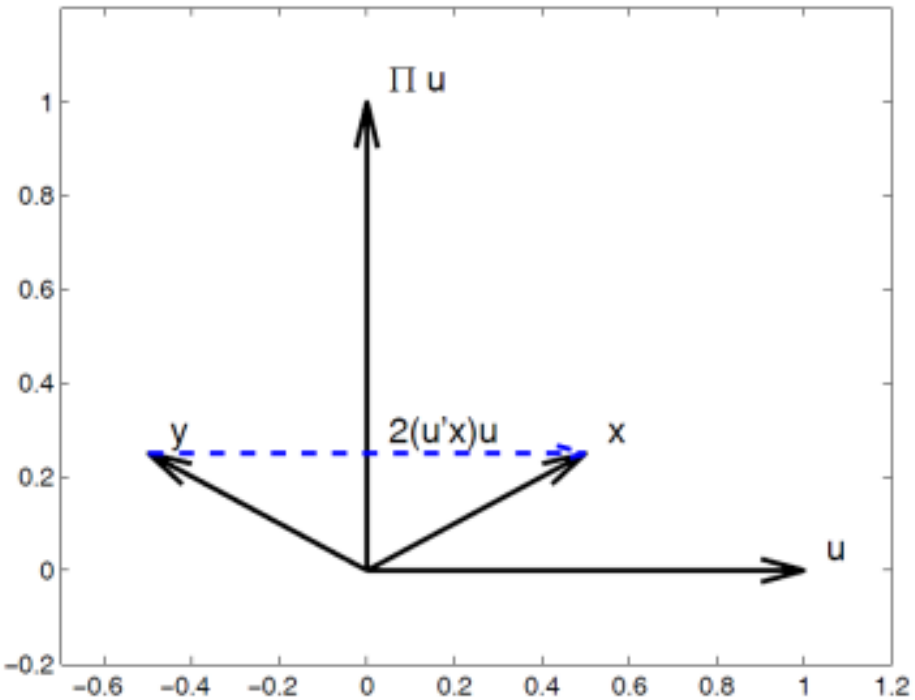
- I è la matrice identità.
- $\mathbf{w}^T\mathbf{x}$ è il prodotto scalare tra $\mathbf{w}$ e $\mathbf{x}$, che rappresenta la proiezione di $\mathbf{x}$ nella direzione di $\mathbf{u}$.
- $2(\mathbf{w}^T\mathbf{x})\mathbf{w}$ è il vettore che rappresenta il doppio della proiezione di $\mathbf{x}$ su $\mathbf{w}$.

Il vettore riflesso $\mathbf{y}$ è ottenuto sottraendo da $\mathbf{x}$ il doppio della componente di $\mathbf{x}$ nella direzione di $\mathbf{w}$. Questo è il significato dell'espressione:

$$\mathbf{y} = \mathbf{x} - 2(\mathbf{w}^T\mathbf{x})\mathbf{w}$$

Fattorizzazione QR

La differenza $\mathbf{x} - \mathbf{y}$ è uguale a $2(\mathbf{w}^T\mathbf{x})\mathbf{w}$. Visivamente, si può pensare alla riflessione di $\mathbf{x}$ rispetto a $\mathbf{w}$ come la costruzione di un parallelogramma, dove uno dei lati è il vettore $\mathbf{w}$, e $\mathbf{y}$ è il punto opposto a $\mathbf{x}$ rispetto al lato $\mathbf{w}$. Interpretazione grafica in $\mathbb{R}^2$ con $u = e_1$ e x:



Fattorizzazione QR

Per comprimere un'immagine utilizzando la fattorizzazione QR, l'immagine può essere rappresentata come una matrice, in cui ogni colonna rappresenta un vettore di pixel. Applicando la fattorizzazione QR a questa matrice, otteniamo Q, che contiene informazioni sulle trasformazioni ortogonali applicate all'immagine originale. Queste trasformazioni possono ruotare, riflettere o traslare l'immagine in modo tale da ridurre la correlazione tra i pixel e quindi eliminare la ridondanza spaziale, essendo inoltre Q, una matrice ortogonale, essa conserva tutte le informazioni dell'immagine originale senza perdita di dati.

Fattorizzazione QR

La matrice triangolare superiore R calcolata fornisce dettagli sulle variazioni di luminosità e contrasto dell'immagine. Una volta ottenute queste due matrici è possibile utilizzarle per rappresentare l'immagine in modo più efficiente. Ad esempio, è possibile comprimere l'immagine memorizzando solo le prime k colonne di Q e le prime k righe di R, riducendo così lo spazio di archiviazione richiesto per memorizzare l'immagine. Quindi otteniamo una rappresentazione più compatta dell'immagine originale,ma allo stesso tempo manteniamo tutte le informazioni rilevanti dell'immagine.

Singular Value Decomposition

La decomposizione ai valori singolari (SVD) è una tecnica di analisi matriciale, utilizzata nelle compressioni delle immagini. Questa fattorizzazione scompone una matrice rettangolare in tre componenti principali. Data una matrice $A \in \mathbb{R}^{m \times n}$

$$A = U \Sigma V^T$$

$$(\Sigma)_{ij} = \begin{cases} \sigma_i, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

1. **Matrice U**: Una matrice ortogonale che contiene i "vettori singolari sinistri", $U \in \mathbb{R}^{m \times m}$.
2. **Matrice Σ (Sigma)**: Una matrice diagonale (non necessariamente quadrata) contenente i "valori singolari", ordinati in modo decrescente, $\Sigma \in \mathbb{R}^{m \times n}$.
3. **Matrice V^T**: La trasposta della matrice ortogonale contenente i "vettori singolari destri", $V \in \mathbb{R}^{n \times n}$.

Singular Value Decomposition

Dove $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, $p = \min\{m, n\}$, quindi gli elementi della matrice diagonale Σ, sono ordinati in senso decrescente e sono definiti come $\sigma_i = \sqrt{\lambda_i}$, dove λ_i sono gli autovalori della matrice $B = A^T A$.

I valori non nulli della matrice Σ sono i valori singolari.
I vettori colonna u_i della matrice $\mathbf{U}$ sono chiamati vettori singolari sinistri. Questi vettori sono associati ai valori singolari e rappresentano la trasformazione degli spazi degli ingressi.
I vettori v_i della matrice $\mathbf{V}$ sono chiamati vettori singolari destri. Essi sono associati ai valori singolari e rappresentano la trasformazione degli spazi delle uscite.

Posso riscrivere la decomposizione per $m \neq n$:

$$A = \sum_{i=1}^{p} u_i \sigma_i v_i^T$$

Singular Value Decomposition

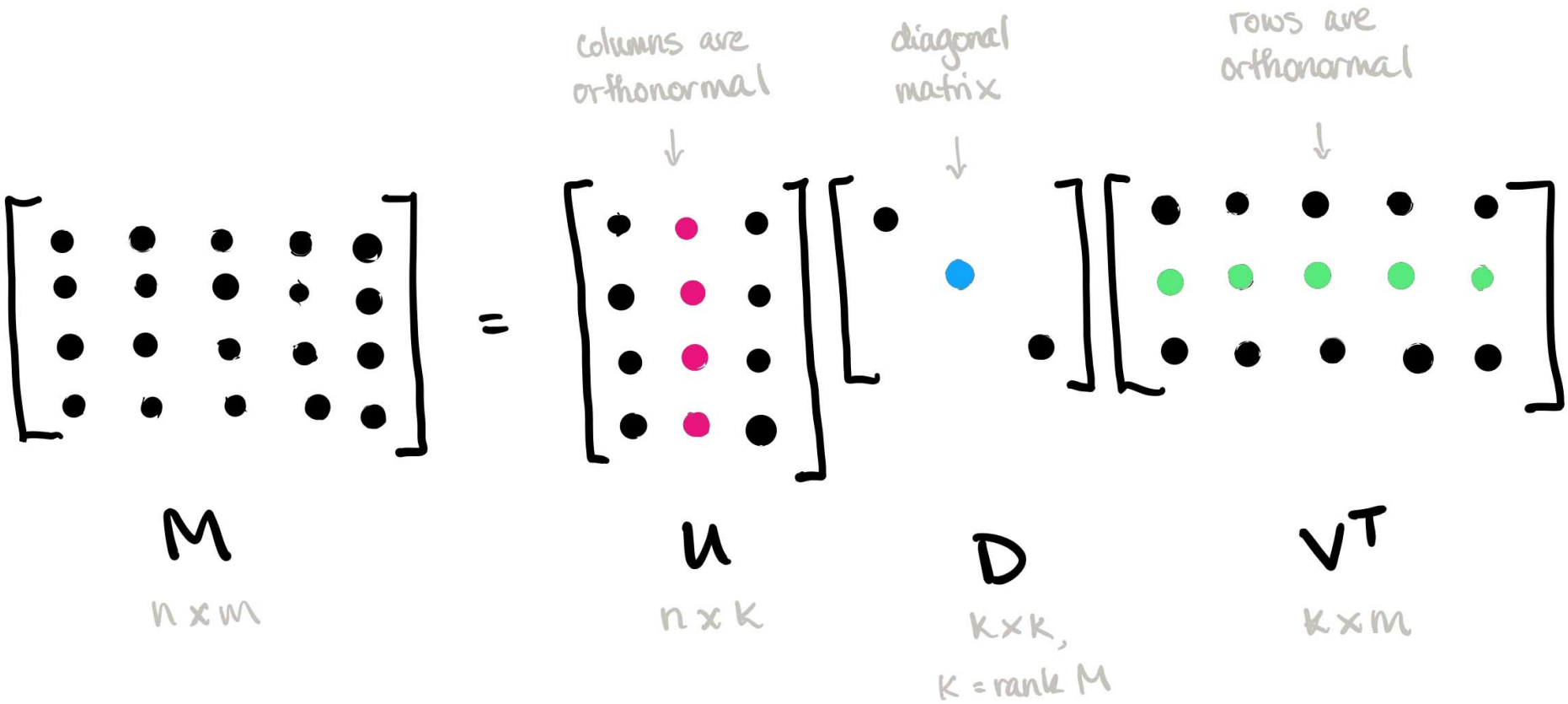


Figure: SVD divide A in tre componenti

Singular Value Decomposition

La decomposizione ai valori singolari può essere utilizzata per calcolare gli autovalori e gli autovettori di una matrice quadrata, anche se non è simmetrica. Questo è particolarmente utile quando la matrice non ha una struttura particolare come la simmetria, perché la SVD funziona su qualsiasi tipo di matrice.
Consideriamo l'equazione $A^T A$:

$$A^T A = (U\Sigma V^T)^T (V\Sigma^T U^T) = (V\Sigma^T U^T)(U\Sigma V^T) = V\Sigma\Sigma^T V^T$$

Questa equazione ci mostra che il prodotto $A^T A$ è uguale al prodotto di V, Σ e Σ^T, dove Σ^T è la trasposta della matrice diagonale Σ. Questo implica che le entrate diagonali della matrice quadrata $\Sigma^T\Sigma$, che sono il quadrato dei valori singolari, sono gli autovalori della matrice $A^T A$.
Inoltre, considerando il prodotto AA^T, otteniamo:

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T)(V\Sigma^T U^T) = U\Sigma\Sigma^T U^T$$

Singular Value Decomposition

Questo ci mostra che il prodotto AA^T è uguale al prodotto di U, Σ e Σ^T, dove Σ^T è ancora la trasposta della matrice diagonale Σ. Pertanto, le entrate diagonali della matrice quadrata $\Sigma\Sigma^T$ sono gli autovalori della matrice AA^T.
Poiché il rango della matrice A è r, solo i primi r autovalori di A^TA e AA^T sono diversi da zero. Questo è un risultato importante che ci dice che possiamo concentrarci solo sui primi r autovalori e autovettori più significativi durante il calcolo della SVD.

Singular Value Decomposition

Teorema 1

Questo teorema riguarda la relazione tra i valori singolari e il rango di una matrice. Se per qualche r tale che $1 \leq r < p$ abbiamo

$$\sigma_1 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$$

allora

- Il rango di A è r.
- $A = \sum_{i=1}^{r} u_i \sigma_i v_i^T$

Questo significa che tutte le altre $p - r$ dimensioni della matrice A sono combinazioni lineari delle prime r.

Singular Value Decomposition

Teorema 2: Approssimazione lower rank

Sia $A \in \mathbb{R}^{m \times n}$ una matrice il cui rango è $\text{rank}(A) = r$. Se per un valore intero fisso $k < r$ definiamo

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T \quad (2)$$

Si definisce una matrice A_k come la somma dei primi k termini della decomposizione ai valori singolari (SVD) di A. Questo significa che stiamo considerando solo i primi k valori singolari e i loro vettori associati per approssimare A.
Definiamo

$$B = \{B \in \mathbb{R}^{m \times n} : \text{rank}(B) = k\}$$

B contiene tutte le matrici che hanno un rango esattamente uguale a k.

Singular Value Decomposition

Teorema 2: Approssimazione lower rank

allora

$$\min_{B \in B} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

Si cerca la matrice B in B che minimizza la norma spettrale della differenza tra A e B, cioè $\|A - B\|_2$.
La migliore approssimazione di A (considerando la norma spettrale) di rango k è A_k, cioè la matrice che contiene i primi k valori singolari della decomposizione ai valori singolari di A, è utile perché ridurre il rango di una matrice può semplificare la sua rappresentazione e rendere più efficienti molte operazioni su di essa.

Singular Value Decomposition

L'approssimazione di rango inferiore è una tecnica comune utilizzata nella compressione delle immagini per ridurre lo spazio di archiviazione richiesto per memorizzarle, mantenendo al contempo una qualità accettabile dell'immagine ricostruita. La sua efficacia deriva dalla capacità di rappresentare l'immagine con un numero inferiore di parametri, mantenendo al contempo le caratteristiche essenziali dell'immagine originale.

SVD e Compressione delle Immagini: Nella compressione delle immagini, l'uso della decomposizione ai valori singolari (SVD) è fondamentale. L'SVD scompone una matrice (nel nostro caso, la matrice delle intensità di grigio dell'immagine) in tre componenti: due matrici unitarie (U e V) e una matrice diagonale dei valori singolari (Σ).

Singular Value Decomposition

Rappresentazione di Rango Inferiore: L'idea principale dell'approssimazione di rango inferiore è quella di approssimare la matrice Σ con una versione "tagliata" che contiene solo i primi k valori singolari più significativi. Questo significa che, anziché memorizzare l'intera matrice delle intensità di grigio, si memorizza solo una porzione dei valori singolari più importanti.

Scelta di k: La scelta del parametro k è cruciale nella compressione delle immagini. Un valore troppo basso di k potrebbe comportare una perdita significativa di informazioni e una qualità dell'immagine finale scarsa. Al contrario, un valore troppo alto di k potrebbe non portare a un risparmio di spazio significativo. È importante trovare un compromesso tra la dimensione del file compresso e la qualità dell'immagine ricostruita.

Singular Value Decomposition

Visualizzazione e Ricostruzione: Dopo aver approssimato la matrice delle intensità di grigio con una versione di rango inferiore, è possibile ricostruire un'approssimazione dell'immagine originale utilizzando le matrici U_k, Σ_k e V_k. Questa immagine approssimata può essere visualizzata utilizzando strumenti come il comando `image` in Matlab.

Singular Value Decomposition

Interpretazione Geometrica della SVD:
Quando si parla dei vettori e_i^n ed e_i^m, ci si riferisce ai vettori della base canonica negli spazi $\mathbb{R}^n$ e $\mathbb{R}^m$ rispettivamente.
La relazione

$$Av_i = U\Sigma v_i = U\Sigma e_i^n = \sigma_i u_i$$

mostra che, quando si applica la matrice A a un vettore singolare destro v_i (una colonna di V), si ottiene un multiplo del corrispondente vettore singolare sinistro u_i (una colonna di U), moltiplicato per il valore singolare corrispondente σ_i. In altre parole, A "mappa" il vettore v_i in $\mathbb{R}^n$ in una direzione specifica u_i in $\mathbb{R}^m$, scalando la lunghezza di questo vettore per σ_i. Le colonne delle matrici U e V formano delle basi ortonormali nei rispettivi spazi $\mathbb{R}^m$ e $\mathbb{R}^n$.

Singular Value Decomposition

Ciò significa che possiamo pensare alla SVD come a un modo di trovare nuove coordinate nei due spazi, tali che la trasformazione A diventa particolarmente semplice. In queste nuove coordinate, A scala i vettori lungo gli assi (i vettori delle basi ortonormali) senza mescolare le loro direzioni.
In altre parole, applicare A a un vettore di $\mathbb{R}^n$ corrisponde a ridimensionarlo (secondo σ_i) e a reindirizzarlo lungo un vettore di $\mathbb{R}^m$.

Singular Value Decomposition

Una trasformazione lineare A può essere vista come una combinazione di tre semplici operazioni:

1. Rotazione del vettore x (tramite V),
2. Ridimensionamento lungo direzioni specifiche (tramite Σ),
3. Un'altra rotazione del risultato (tramite U).

Questo ci dice che per ogni trasformazione lineare tra $\mathbb{R}^n$ e $\mathbb{R}^m$, possiamo trovare delle basi ortonormali (formate dai vettori singolari) tali che A agisca in modo molto semplice: ogni vettore nella base di $\mathbb{R}^n$ viene trasformato in un multiplo del corrispondente vettore nella base di $\mathbb{R}^m$. Rispetto a queste basi, A è rappresentata dalla matrice Σ, che è diagonale.

Singular Value Decomposition

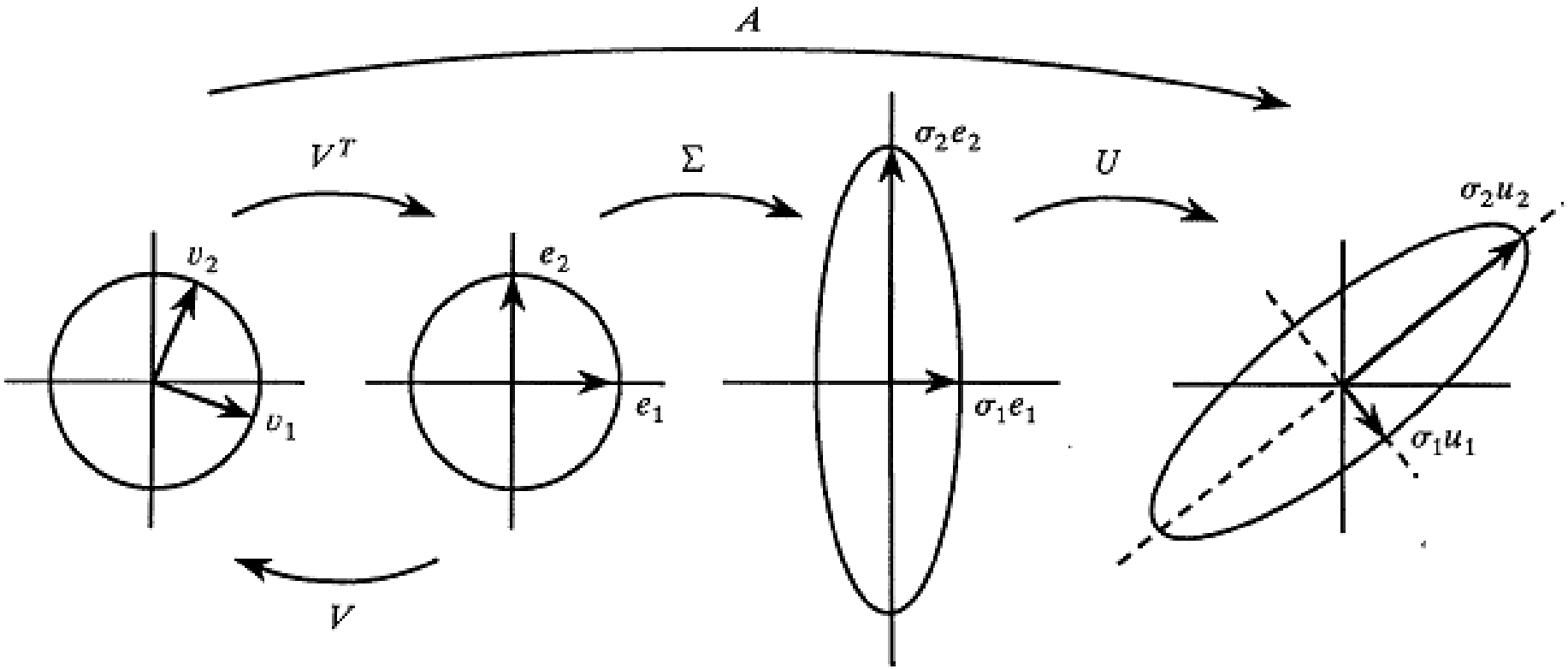


Figure: Rotazione, ridimensionamento, rotazione del SVD

Singular Value Decomposition

Calcolo della decomposizione ai valori singolari (SVD) di una matrice A.
Calcolo di $B = A^T A$:
Si inizia calcolando la matrice B, data dal prodotto della trasposta di A con A stessa:

$$B = A^T A$$

B è una matrice quadrata simmetrica di dimensione $n \times n$, se A è di dimensione $m \times n$.

Singular Value Decomposiotion

Diagonalizzazione di B: Il passo successivo è diagonalizzare B, trovando una matrice ortogonale V (le cui colonne sono i vettori propri di B) e una matrice diagonale Λ, tale che:

$$B = V\Lambda V^T$$

Qui, $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, dove λ_i sono gli autovalori di B, ordinati in modo decrescente.

Calcolo di Σ: La matrice Σ nella SVD è diagonale e contiene i valori singolari σ_i di A. Si ottiene prendendo la radice quadrata di ogni autovalore λ_i di B:

$$\Sigma = \text{diag}(\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_r})$$

Qui, r è il rango di A, ovvero il numero di valori singolari non nulli.

Singular Value Decomposition

Calcolo delle Colonne di U: Una volta ottenuti Σ e V, si può calcolare la matrice U utilizzando la relazione:

$$U\Sigma = AV$$

Le colonne di U si ottengono quindi come:

$$u_i = \frac{Av_i}{\sigma_i}, \quad \text{per } i = 1, \ldots, r$$

Dove v_i sono le colonne di V e σ_i sono i valori singolari corrispondenti. Le colonne $u_{r+1}, \ldots, u_m$ si ottengono completando U in modo che sia una matrice ortogonale, ovvero formando una base ortonormale di $\mathbb{R}^m$.

Singular Value Decomposition

Accuratezza Numerica: Una delle principali limitazioni di questo metodo è legata alla precisione numerica. Quando si calcola $A^T A$, si possono amplificare gli errori di arrotondamento, specialmente se A è mal condizionata. Questo può portare a risultati inaccurati.
Vicinanza dei Valori Singolari: Se i valori singolari sono molto vicini tra loro, potrebbe essere difficile distinguerli correttamente con questo metodo, aumentando la possibilità di errore.

Codice MATLAB - SVD (Parte 1)

```
function A_k = func_ourSVD(A, k)
    [m,n] = size(A);

    % Decomposizione SVD
    if m >= n
        % Ricavo degli autovettori e autovalori di A * A' (m x m)
        [U, lambda] = eig(A * A');
        % Ordinamento degli autovalori
        [lambda, indici] = sort(diag(lambda), "descend");
        % Ordinamento delle colonne rispetto agli autovalori
        U = U(:, indici);
        % Trovo i valori singolari
        Sigma = sqrt(diag(lambda));
        % Seleziono le prime k righe e le prime k colonne
        Sigma_k = Sigma(1:k, 1:k);
        % Normalizzazione degli autovettori di U
        for i = 1:m
            U(:, i) = U(:, i) / norm(U(:, i));
        end
```

Codice MATLAB - SVD (Parte 2)

```
        % Selezione delle prime k colonne di U
        U_k = U(:, 1:k);
        % Calcolo V
        V_k = A' * U_k / Sigma_k;
        % Normalizzazione degli autovettori di V
        for i = 1:k
            V_k(:, i) = V_k(:, i) / norm(V_k(:, i));
        end
    else
        % Ricavo degli autovettori e autovalori di A' * A (n x n)
        [V, lambda] = eig(A' * A);
        % Ordinamento degli autovalori
        [lambda, indici] = sort(diag(lambda), "descend");
        % Ordinamento delle colonne rispetto agli autovalori
        V = V(:, indici);
        % Trovo i valori singolari
        Sigma = sqrt(diag(lambda));
        % Seleziono le prime k righe e le prime k colonne
        Sigma_k = Sigma(1:k, 1:k);
        % Normalizzazione degli autovettori di V
        for i = 1:n
            V(:, i) = V(:, i) / norm(V(:, i));
```

Codice MATLAB - SVD (Parte 3)

```
        end
        % Selezione delle prime k colonne di U
        V_k = V(:, 1:k);
        % Calcolo U
        U_k = A * V_k / Sigma_k;
        % Normalizzazione degli autovettori di U
        for i = 1:k
            U_k(:, i) = U_k(:, i) / norm(U_k(:, i));
        end
    end
    A_k = round(U_k * Sigma_k * V_k');
end
```

Confronto tra QR e SVD

Come abbiamo potuto vedere, i metodi operano con due approcci diversi:

- **fattorizzazione QR**: scinde la matrice in un prodotto tra una matrice ortogonale e una triangolare superiore.
- **Decomposizione SVD**: scinde la matrice in un prodotto di composizione tra la matrice **U** dei vettori singolari sinistri, la matrice diagonale Σ, dei valori singolari e la matrice **V** dei vettori singolari destri.

Confronto tra QR e SVD

Quello più indicato per i nostri scopi è proprio SVD, poiché a differenza di QR, è in grado di prendere in considerazione soltanto le prime k **componenti principali a varianza maggiore**. Questo comporta una ridotta perdita di informazione sui dati iniziali, ma che consente di mantenere soltanto i dati salienti escludendo quelli ridondanti e poco rilevanti.

Il metodo QR è possibile adottarlo tuttavia la compressione stimata non prende in considerazione le componenti a maggior varianza andando a risultare poco congeniale per l'addestramento o collaudo di una rete di classificazione.

Risultati ottenuti con SVD

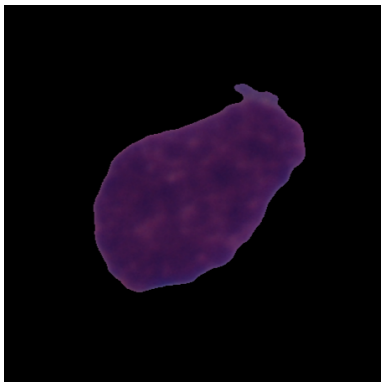


Figure: Cellula Malata 1

Figure: Cellula Malata 2

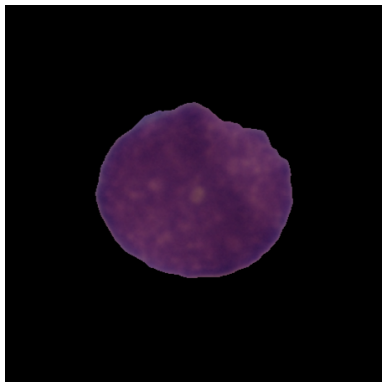


Figure: Cellula Sana 1

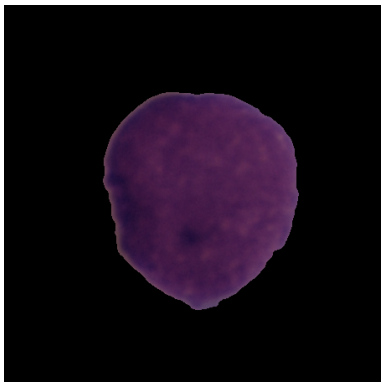
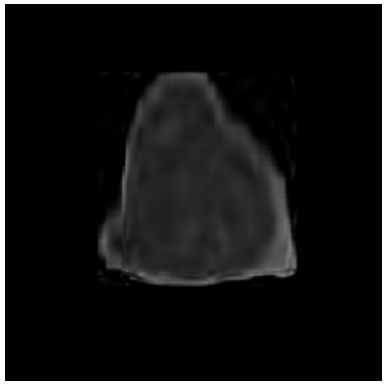


Figure: Cellula Sana 2



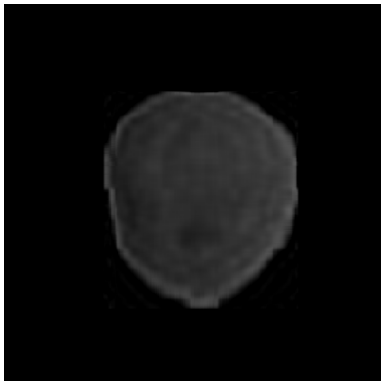


Figure: Applicazione SVD in scala di grigi

Rapporto di Compressione

Le immagini ottenute sopra sono il risultato della conversione in scala di grigi delle immagine da dare in input alla rete mantenendo soltanto le 25 componenti più significative.

Il risultato ottenuto è il seguente:

```
Dimensione totale in MB delle immagini originali: 6486.73 MB
Dimensione totale in MB delle immagini compresse: 171.93 MB
Rapporto di compressione: 97.35 %
```

Figure: Compression Ratio

Introduzione alle Reti Neurali

Le **reti neurali** sono modelli computazionali ispirati al funzionamento del cervello umano. Sono progettate per riconoscere pattern e sono ampiamente utilizzate in vari campi come:

- **Visione artificiale**: riconoscimento di immagini e oggetti.
- **Elaborazione del linguaggio naturale**: traduzione automatica, riconoscimento vocale.
- **Previsioni e analisi dei dati**: previsione di trend, analisi predittiva.
- **Gioco e robotica**: apprendimento di strategie e controllo di movimenti.

Le reti neurali sono composte da strati di nodi interconnessi, detti **neuroni**, che elaborano e trasmettono informazioni.

Struttura di una Rete Neurale a Un Solo Strato

Una **rete neurale a un solo strato** (perceptron) è composta da:

- **Strato di Input**: riceve i dati iniziali.
- **Strato di Output**: produce la previsione o classificazione.

Matematicamente, il funzionamento di una rete neurale può essere descritto come un **prodotto matrice-vettore**. Dato un vettore di input $\mathbf{x}$, i pesi $\mathbf{W}$ e i bias $\mathbf{b}$, l'output $\mathbf{y}$ è calcolato come:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

dove σ è una funzione di attivazione applicata elemento per elemento.

Reti Neurali come Composizione di Funzioni

Le reti neurali profonde, composte da più strati, possono essere interpretate come una **composizione di funzioni**. Ogni strato esegue una trasformazione lineare seguita da una non linearità:

$$\mathbf{y} = \sigma_L(\mathbf{W}_L(\sigma_{L-1}(\mathbf{W}_{L-1}(\cdots \sigma_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\cdots) + \mathbf{b}_{L-1}) + \mathbf{b}_L)$$

dove L è il numero di strati, $\mathbf{W}_i$ sono le matrici dei pesi e $\mathbf{b}_i$ i bias per ciascun strato i.

Questa struttura permette alla rete neurale di apprendere rappresentazioni gerarchiche dei dati, con strati successivi che catturano caratteristiche sempre più astratte.

La Rete Neurale come Approssimatore Universale

La **rete neurale** è un **approssimatore universale**, il che significa che può approssimare qualsiasi funzione continua su uno spazio compatto con una precisione arbitraria. Questa proprietà è formalmente dimostrata dal **Teorema dell'Approssimazione Universale**.

In pratica, ciò implica che, con un numero sufficiente di neuroni e una corretta scelta della funzione di attivazione, una rete neurale può apprendere e generalizzare modelli estremamente complessi, rendendola uno strumento versatile per molte applicazioni.

Addestramento della Rete Neurale tramite Discesa del Gradiente Coniugato

Il **metodo del gradiente coniugato** è un algoritmo utilizzato per risolvere sistemi lineari del tipo $A\mathbf{x} = \mathbf{b}$, dove A è una matrice simmetrica e definita positiva. Questo metodo è molto efficace perché permette di trovare la soluzione esatta del sistema lineare in un numero di iterazioni pari al massimo alla dimensione del sistema.
Il problema che si cerca di risolvere è il seguente: dati una matrice A simmetrica e definita positiva e un vettore noto $\mathbf{b}$, si vuole trovare il vettore $\mathbf{x}$ tale che:

$$A\mathbf{x} = \mathbf{b}$$

Addestramento della Rete Neurale tramite Discesa del Gradiente Coniugato

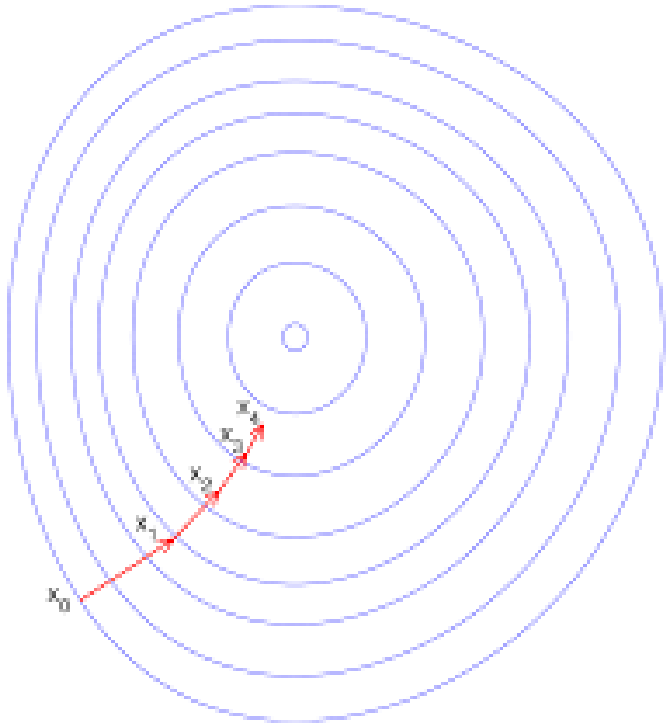


Figure: Discesa del gradiente

Addestramento della Rete Neurale tramite Discesa del Gradiente Coniugato

La matrice A induce un prodotto scalare particolare, chiamato prodotto scalare A-coniugato, definito come:

$$\langle \mathbf{u}, \mathbf{v} \rangle_A := \mathbf{u}^T A \mathbf{v}$$

Due vettori $\mathbf{u}$ e $\mathbf{v}$ sono detti A-coniugati se:

$$\langle \mathbf{u}, \mathbf{v} \rangle_A = 0$$

Questo concetto è fondamentale nel metodo del gradiente coniugato.
La soluzione $\mathbf{x}$ del sistema lineare può essere vista come il punto di minimo della forma quadratica associata:

$$Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b}$$

Addestramento della Rete Neurale tramite Discesa del Gradiente Coniugato

Minimizzare $Q(\mathbf{x})$ è equivalente a risolvere il sistema $A\mathbf{x} = \mathbf{b}$, poiché il gradiente di $Q(\mathbf{x})$ rispetto a $\mathbf{x}$ è dato da:

$$\nabla Q(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$$

E quindi, il minimo di $Q(\mathbf{x})$ si ottiene quando $\nabla Q(\mathbf{x}) = 0$, che equivale a risolvere $A\mathbf{x} = \mathbf{b}$.
A differenza del metodo del gradiente classico, in cui ad ogni passo la direzione di discesa è semplicemente l'opposto del gradiente, nel metodo del gradiente coniugato la direzione di discesa $\mathbf{p}_k$ viene scelta in modo tale che essa sia A-coniugata rispetto alle direzioni di discesa precedenti.
Questo significa che:

$$\langle \mathbf{p}_j, \mathbf{p}_k \rangle_A = 0 \quad \text{per ogni } j < k$$

Addestramento della Rete Neurale tramite Discesa del Gradiente Coniugato

Questa scelta delle direzioni A-coniugate garantisce che il metodo convergerà esattamente alla soluzione in al massimo n iterazioni, dove n è la dimensione del sistema.

Il vantaggio principale del metodo del gradiente coniugato rispetto al metodo del gradiente classico è che, grazie alla scelta delle direzioni A-coniugate, esso converge molto più rapidamente. Infatti, in condizioni ideali, il metodo converge in esattamente n iterazioni, dove n è la dimensione del sistema.
Questo metodo è particolarmente utile quando A è una matrice di grandi dimensioni, ma sparsa (cioè con molti elementi zero), poiché richiede meno memoria e operazioni rispetto ai metodi diretti come la decomposizione di Cholesky.

Pseudocodice - Metodo del Gradiente Coniugato

```
function x = gradiente_coniugato(A, b, x0, tol)
    % A     la matrice del sistema
    % b     il vettore del termine noto
    % x0     il vettore iniziale
    % tol     la tolleranza per la convergenza

    r0 = b - A * x0             % Calcolo del residuo iniziale
    p0 = r0                     % Direzione iniziale
    k = 0                       % Iterazione iniziale

    while ||r0|| > tol          % Ciclo finch  il residuo non
        sufficientemente piccolo
        alpha_k = (p0' * r0) / (p0' * A * p0)    % Calcolo di
            alpha_k
        x = x0 + alpha_k * p0                    % Aggiorna x_k+1
        r1 = b - A * x                           % Calcolo del nuovo
            residuo
        if ||r1|| <= tol                         % Controllo del
            criterio di convergenza
            return x                             % Se il residuo
                piccolo, uscire
        end
```

Pseudocodice - Metodo del Gradiente Coniugato

```
        beta_k = (p0' * A * r1) / (p0' * A * p0) % Calcolo di
            beta_k
        p1 = r1 + beta_k * p0                    % Aggiorna la
            direzione p_k+1
        p0 = p1                                  % Aggiorna la
            direzione
        r0 = r1                                  % Aggiorna il
            residuo
        x0 = x                                   % Aggiorna la
            soluzione
        k = k + 1                                % Incrementa l'
            indice k
    end
end
```

Discesa del Gradiente Classica

La **discesa del gradiente** è un algoritmo di ottimizzazione utilizzato per minimizzare la funzione di perdita $L(\mathbf{w})$ di una rete neurale.

Passaggi principali:

1. **Calcolo del Gradiente**: Si calcola $\nabla L(\mathbf{w})$, il gradiente della funzione di perdita rispetto ai pesi $\mathbf{w}$.
2. **Aggiornamento dei Pesi**: I pesi vengono aggiornati nella direzione opposta al gradiente:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla L(\mathbf{w}_k)$$

dove η è il tasso di apprendimento.

Gradiente Coniugato con Variante Fletcher–Reeves

Il **metodo del gradiente coniugato** migliora la velocità di convergenza rispetto alla discesa del gradiente classica.

Iterazioni principali:

1. **Calcolo della direzione di discesa più ripida**:

$$\Delta \mathbf{x}_n = -\nabla f(\mathbf{x}_n)$$

   dove $\nabla f(\mathbf{x}_n)$ è il gradiente della funzione di perdita f nel punto $\mathbf{x}_n$.

2. **Calcolo del parametro** β_n:

$$\beta_n = \frac{\nabla f(\mathbf{x}_n)^T \nabla f(\mathbf{x}_n)}{\nabla f(\mathbf{x}_{n-1})^T \nabla f(\mathbf{x}_{n-1})}$$

3. **Aggiornamento della direzione coniugata**:

$$\mathbf{s}_n = \Delta \mathbf{x}_n + \beta_n \mathbf{s}_{n-1}$$

   dove $\mathbf{s}_0 = \Delta \mathbf{x}_0$ è la direzione iniziale.

Gradiente Coniugato: Ricerca in Linea (Line Search)

4. **Ricerca in linea (line search)**:
   - **Calcolo del passo ottimale** α_n:

$$\alpha_n = \arg\min_{\alpha} f(\mathbf{x}_n + \alpha \mathbf{s}_n)$$

   - **Aggiornamento della posizione** $\mathbf{x}_{n+1}$:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{s}_n$$

Gradiente Coniugato con Variante Fletcher–Reeves

La **ricerca in linea** viene utilizzata per migliorare la velocità di convergenza dei metodi di ottimizzazione, inclusi i metodi basati su gradiente come il gradiente coniugato con variante Fletcher-Reeves. In un semplice metodo del gradiente con passo fisso, si può incorrere in diversi problemi:

- Se il passo è **troppo piccolo**, l'algoritmo potrebbe convergere troppo lentamente, richiedendo molte iterazioni per raggiungere una soluzione accettabile.
- Se il passo è **troppo grande**, l'algoritmo potrebbe oscillare attorno alla soluzione senza convergere, o addirittura divergere.

Gradiente Coniugato con Variante Fletcher–Reeves

Per superare questi problemi, la **ricerca in linea** ottimizza il valore di α_n ad ogni iterazione, trovando il miglior passo possibile lungo la direzione corrente di discesa s_n. Questo ottimizza la riduzione della funzione obiettivo in ogni iterazione, garantendo una discesa più rapida verso il minimo globale.
Ottimizzando α_n, si riduce il numero di iterazioni necessarie per raggiungere la soluzione ottimale, migliorando quindi l'efficienza complessiva dell'algoritmo. Nel **metodo del gradiente Fletcher-Reeves**, dopo aver calcolato la direzione di discesa coniugata s_n, si utilizza la ricerca in linea per ottimizzare il passo α_n. Questo permette di determinare il miglior spostamento lungo la direzione s_n che minimizza la funzione obiettivo $f(x)$ lungo quella direzione. Questa ottimizzazione iterativa garantisce che il metodo sia più efficiente, migliorando la velocità di convergenza verso la soluzione ottimale.

Confronto: Discesa del Gradiente vs. Gradiente Coniugato

Discesa del Gradiente Classica:

- **Pro**: Semplicità e adattabilità.
- **Contro**: Convergenza lenta, rischia di fermarsi in minimi locali.

Gradiente Coniugato (Fletcher–Reeves):

- **Pro**: Convergenza più rapida, particolarmente efficace per grandi reti.
- **Contro**: Complessità di implementazione, sensibilità a errori numerici.

Backpropagation

Backpropagation è un algoritmo per calcolare i gradienti in modo efficiente, utilizzando la regola della catena per propagare l'errore dallo strato di output agli strati precedenti.

Vantaggi:

- Riduce il costo computazionale del calcolo dei gradienti.
- È essenziale per addestrare reti neurali profonde.

Limiti:

- Può essere lento a convergere, specialmente in problemi complessi.
- Richiede un'attenta selezione del tasso di apprendimento η.

Backpropagation: Discesa del Gradiente Classica

La **backpropagation** è un algoritmo fondamentale per l'addestramento delle reti neurali, utilizzato per calcolare efficientemente i gradienti della funzione di perdita rispetto ai pesi della rete.

Passaggi principali:

1. **Propagazione in avanti (forward pass)**:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = \sigma(\mathbf{z}^{(l)})$$

   Dove $\mathbf{z}^{(l)}$ è l'input del layer l, $\mathbf{a}^{(l)}$ è l'output attivato, e σ è la funzione di attivazione.

2. **Calcolo dell'errore al livello di output**:

$$\delta^{(L)} = \nabla_a L \odot \sigma'(\mathbf{z}^{(L)})$$

Backpropagation: Discesa del Gradiente Classica

Dove $\delta^{(L)}$ è l'errore al livello di output, L è l'indice dell'ultimo layer, L è la funzione di perdita, e $\odot$ denota il prodotto elemento per elemento (Hadamard product).

3. **Propagazione dell'errore all'indietro (backward pass)**:

$$\delta^{(l)} = (\mathbf{W}^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(\mathbf{z}^{(l)})$$

Dove $\delta^{(l)}$ è l'errore al livello l, propagato dagli strati successivi.

4. **Aggiornamento dei pesi e dei bias**:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \delta^{(l)} (\mathbf{a}^{(l-1)})^T$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta \delta^{(l)}$$

Dove η è il tasso di apprendimento.

Esempio di Backpropagation

Consideriamo una semplice rete neurale con un singolo neurone, utilizzando la funzione di attivazione sigmoide $\sigma(x) = \frac{1}{1+e^{-x}}$ e la funzione di perdita MSE (Mean Squared Error).

Passo 1: Propagazione in avanti:

$$z = w \cdot x + b$$

$$a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Passo 2: Calcolo dell'errore:

$$\delta = (a - y) \cdot \sigma'(z)$$

Dove y è il valore target.

Passo 3: Aggiornamento dei pesi:

$$w \leftarrow w - \eta \cdot \delta \cdot x$$

$$b \leftarrow b - \eta \cdot \delta$$

Risultati del nostro Modello

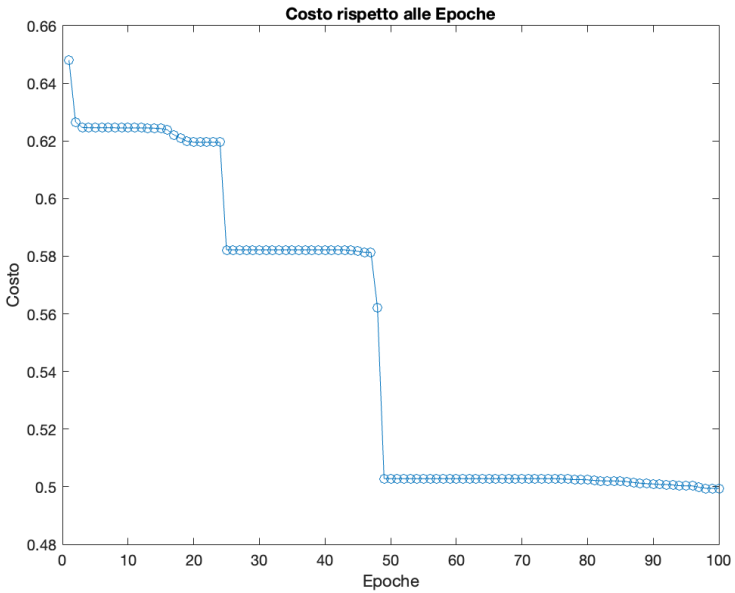


Figure: Perdita in funzione delle epoche

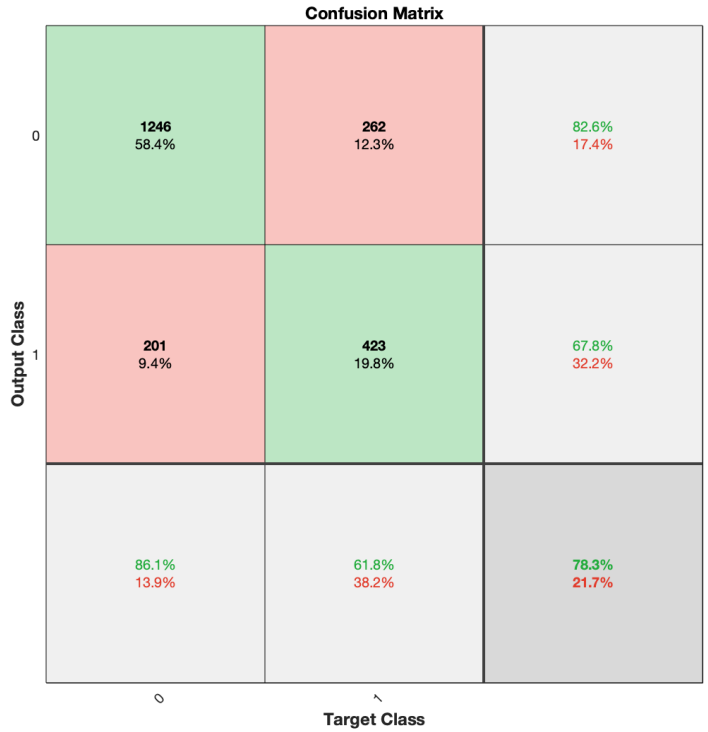


Figure: Matrice di Confusione

Risultati SGD implementato in Matlab

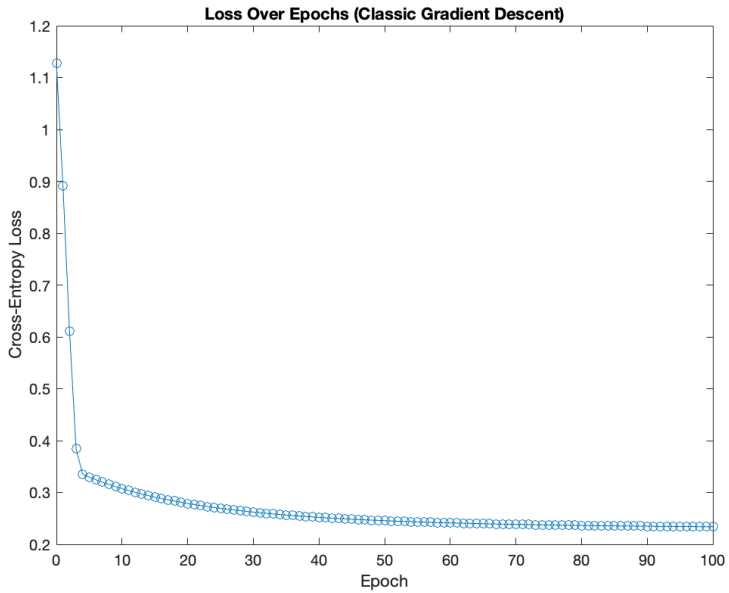


Figure: Perdita in funzione delle epoche

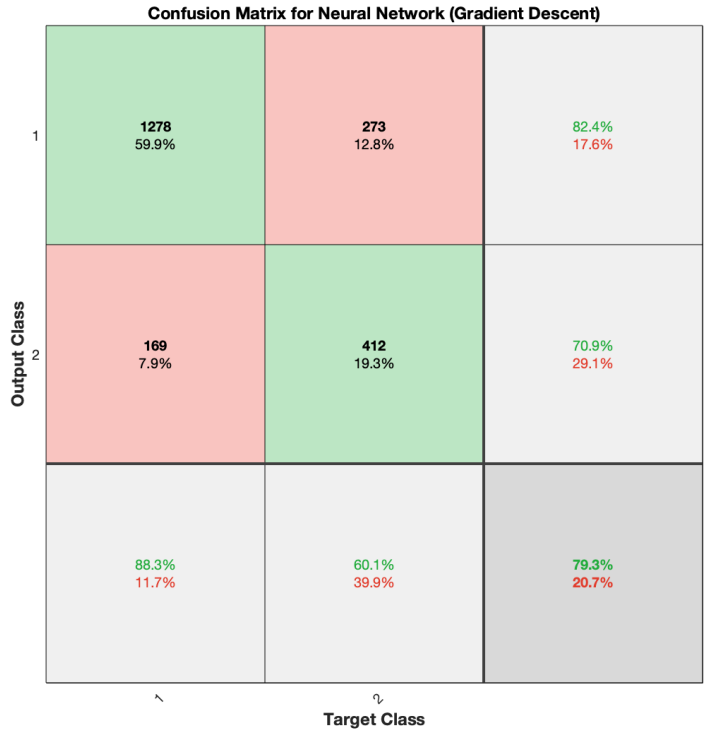


Figure: Matrice di Confusione

Risultati Gradiente Coniugato implementato in Matlab

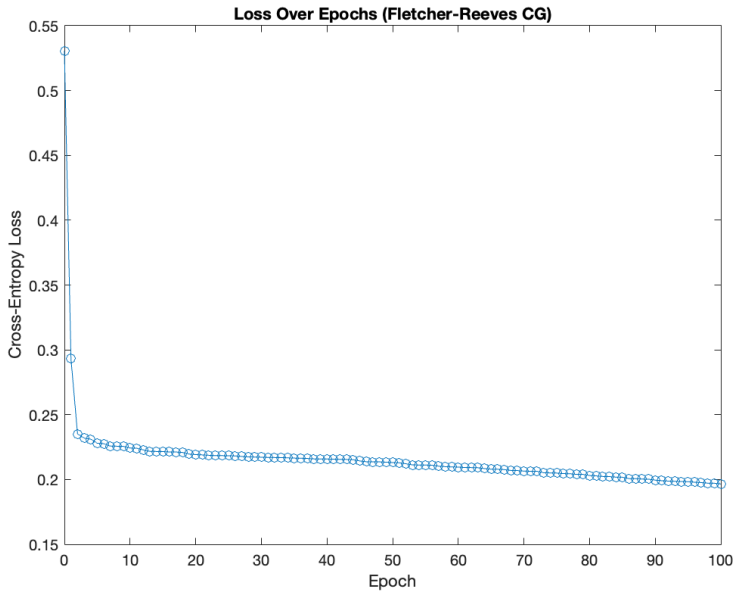


Figure: Perdita in funzione delle epoche

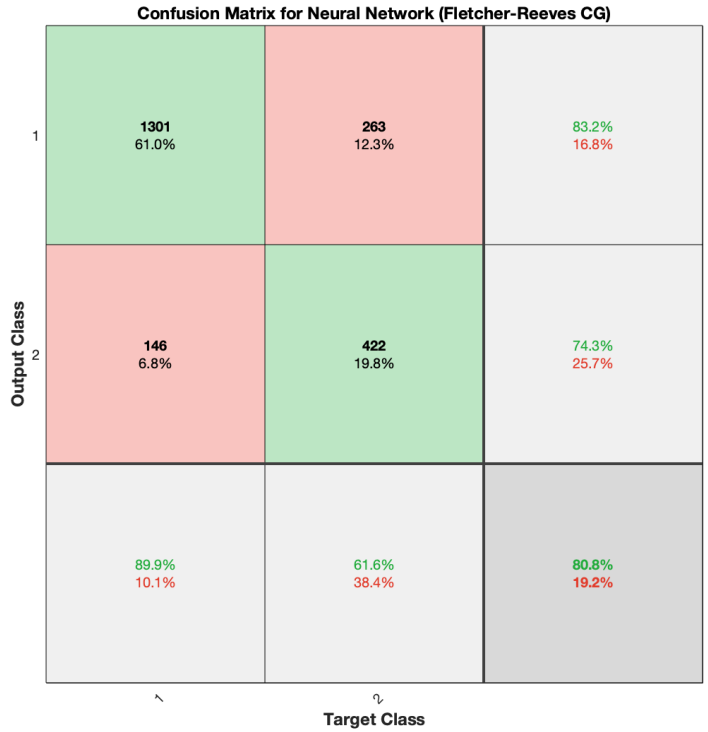


Figure: Matrice di Confusione

Sitografia

- Slide della materia
- Dataset cellule ematiche
- Documentazione Matlab per costruzione della rete neurale
- Decomposizione QR
- Trasformazione di Householder
- Fattorizzazione QR teoria ed esempi applicativi
- Interpretazione geometrica della fattorizzazione QR

Sitografia

- Decomposizione ai valori singolari
- Calcolo SVD
- Metodo del gradiente coniugato
- Metodo del gradiente coniugato non lineare
- Teorema dell'approssimazione universale
- Reti neurali artificiali
- Dataset sulla leucemia