ALMA MATER STUDIORUM –
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'ENERGIA ELETTRICA E DELL'INFORMAZIONE "GUGLIELMO MARCONI" DEI

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

TESI DI LAUREA

IN

ELABORAZIONE DEI SEGNALI NEI SISTEMI ELETTRONICI M

RICOSTRUZIONE DI SEGNALI ELETTROCARDIOGRAFICI SEGUENDO IL PARADIGMA DEL COMPRESSED SENSING ATTRAVERSO L'UTILIZZO DI RETI NEURALI PROFONDE

Relatore

Prof. MAURO MANGIA

Co-relatori

Prof. RICCARDO ROVATTI

Ing. ALEX MARCHIONI

Presentata da

ANDREA CARLONI

Prima Sessione di Laurea

Anno Accademico 2019 - 2020

A Giulia e ai miei genitori

Indice

Introduzione

In molti campi applicativi l'impiego di una coppia encoder-decoder classica, in cui l'encoder ha una complessità maggiore del decoder, risulta inadeguata e/o troppo complessa. Infatti, molte applicazioni richiedono un forte sbilanciamento in termini di costi di elaborazione, come per esempio l'insieme di metodi di compressione usati per preprocessare immagini, video o tracce audio. In simili casi è lecito supporre che il segnale originale sarà compresso una sola volta mentre la decodifica sarà eseguita tutte le volte in cui un utente accede al contenuto. Con il proliferarsi dei sistemi che rientrano nel mondo dell'Internet delle cose (Internet of Things) il paradigma presentato è rovesciato. In contesti come l'IoT si richiede al sistema di acquisizione di essere quanto più possibile poco invasivo, autonomo e persistente, il che si traduce in un sistema di codifica che deve essere semplice, poco oneroso dal punto di vista computazionale e tale da poter essere implementato su una vasta gamma di dispositivi. Da ciò nasce la necessità di sviluppare una coppia encoder - decoder in grado di semplificare quanto più possibile la componente di encoding consentendo di esplorare applicazioni a basso costo, a basso consumo e con ridotta potenza computazionale utili in applicazioni in cui l'encoder è difficilmente raggiungibile o si vuole per vincoli di progetto adottare una soluzione del genere. Si può, quindi, sviluppare la coppia utilizzando un paradigma come il Compressed Sensing e utilizzando una struttura più complessa per il

decoder che, seguendo il paradigma scelto, dovrà ricostruire il segnale raggiungendo un prefissato target di performance. Da questa esigenza si sviluppa il lavoro di tesi magistrale nel seguito descritto con l'auspicio di poter costruire un encoder semplice, che sia in grado di raggiungere buone performance e che accoppiato con un opportuno decoder possa ricostruire il segnale di partenza con una certa fedeltà imposta dal tipo di applicazione in cui il sistema sarà successivamente contestualizzato. Si valuteranno diverse soluzioni per quanto concerne la componente di encoding, modificando e/o eliminando gradi di libertà del sistema.

Il lavoro svolto è stato organizzato in capitoli, descrivendo prima lo stato dell'arte degli strumenti utilizzati, parlando poi della tipologia di segnale analizzato, descrivendo peculiarità e metodi di acquisizione, e concludendo con la descrizione dei risultati ottenuti nei vari test condotti al variare di gradi di libertà presenti o assenti e della tipologia di encoding. Si riportano in appendice alcuni concetti utili a comprendere meglio i punti focali del lavoro svolto.

Capitolo 1

Lo stato dell'arte

In questo capitolo saranno descritti con l'ausilio di articoli scientifici e figure i due strumenti cardine del lavoro svolto: il compressed sensing (CS) e le reti neurali profonde (DNN).

1.1 Il compressed sensing

Il compressed sensing è un nuovo paradigma di acquisizione e/o di campionamento di un segnale che si basa sulla possibilità di ricostruire un dato segnale utilizzando un numero di campioni inferiore al numero utilizzato in tecniche tradizionali. Questo paradigma si basa su due concetti principali che sono la *sparsità* e la *incoerenza*.

La sparsità è un concetto riferito al segnale che si sta analizzando ed esprime l'idea che alcuni segnali abbiano un numero limitato di rappresentazioni quando sono descritti in un'appropriata base sparsa.

L'incoerenza, invece, si riferisce al metodo di acquisizione del segnale e suggerisce come sia possibile descrivere la totalità delle informazioni del segnale partendo dall'idea che il segnale sia sparso in un'appropriata base. [1]

In particolar modo si suppone di considerare una coppia di basi ortonormali $(\Psi, \Phi) \in \mathbb{R}^n$ tale che Ψ sia utilizzata per ricavare i coefficienti della rappresentazione del segnale, mentre Φ sia utilizzata per rappresentare il segnale a partire dai suoi coefficienti. Si può definire la coerenza fra le due matrici come

$$\mu(\Psi, \Phi) = \sqrt{n} \cdot \max_{1 \leq k,j \leq n} |\langle \psi_k, \phi_j \rangle| \tag{1.1}$$

Sapendo che $\mu(\Psi, \Phi)$ varia fra 1 e $\sqrt{n}$, si ricava che il CS lavora in maniera corretta per bassi valori di coerenza (massima incoerenza). [7]

Risulta pertanto necessario approfondire la tematica relativa alla sparsità. In particolare, esistono in natura alcuni segnali, come quello elettrocardiografico o quello elettroencefalografico, che per loro definizione sono sparsi, scelta una base opportuna. Pertanto, proiettando il segnale di partenza sulla base ortonormale scelta (o più in generale su un frame stretto per aumentare la robustezza al rumore) è possibile ottenere un nuovo segnale con lo stesso contenuto informativo del segnale di partenza, ma descrivibile con un numero di campioni decisamente minore. Formalmente si suppone di avere un segnale x' suddiviso in finestre consecutive $x \in \mathbb{R}^n$,$x = (x_0, ..., x_{n-1})$ composte ciascuna da n campioni e si considera un frame $\{\psi_i\}_{i=0,...,n-1}$ contenuto in $\mathbb{R}^n$. Facendo ipotesi più stringenti rispetto alla definizione di frame è possibile giungere alla definizione di base ortonormale[1] che sarà d'ora in poi utilizzata. Scelta la base, si effettua la proiezione del segnale su questa e si ottengono i coefficienti $\theta_i = \langle x, \psi_i \rangle, i = 0, ..., n-1$ sfruttando il prodotto interno definito in $\mathbb{R}^n$. Questi coefficienti descrivono un segnale x κ-sparso se, posti $0 < p < 2$ e $R > 0$, vale

$$\|\theta\|_p \equiv \left(\sum_i \theta_i |^p \right)^{1/p} \leq R \tag{1.2}$$

[1] A partire da un generico frame appartenente a $\mathbb{R}^n$, si ottiene una base ortonormale se il frame è stretto con A=B=1 e ogni elemento del frame possiede norma unitaria

Si possono analizzare i due casi estremi. Quando $p < 1$ la sparsità richiesta cresce, mentre quando $p = 2$ la sparsità richiesta è nulla. Da questo segue che prendendo $\theta = (\theta_0, ..., \theta_{n-1})$ questo contiene al massimo $\kappa < n$ elementi non nulli [2] ed è possibile ricostruire x partendo dalla matrice ortonormale S (le cui colonne sono le basi ortonormali $\{\psi_i\}_{i=0,...,n-1}$utilizzate per determinare i θ_i) e da θ come $x = S\theta$. Generalizzando, la trattazione può essere elaborata anche utilizzando frames stretti. In questo caso i frames $\{\psi_i\}_{i=0,...,n-1}$ costituiscono le colonne di S, sono verificate le stesse relazioni appena citate e risulta verificata una relazione di tipo Parseval:

$$\|x\|_2 = \|\theta(x)\|_2 \tag{1.3}$$

Una relazione di questo tipo, verificabile con un procedimento analogo nel caso di matrice ortonormale, garantisce che l'energia del segnale sia uguale all'energia dei coefficienti, rendendo possibile la ricostruzione del segnale e la sua rappresentazione. [1][2]

Osservando che non occorre acquisire tutto il segnale nel dominio di partenza per poterlo descrivere, ma è sufficiente preservare i campioni più significativi nella base sparsa, il CS consente di acquisire e comprimere contemporaneamente il segnale sparso, limitando la banda richiesta per la trasmissione del segnale, lo spazio in memoria, il consumo energetico del canale di trasmissione e il tempo di acquisizione del segnale. Formalmente questo è possibile perché il CS è in grado di comprimere il segnale applicando un operatore lineare $\mathcal{L}_A : \mathbb{R}^n \mapsto \mathbb{R}^m$ che è funzione della matrice di encoding $A \in \mathbb{R}^{m \times n}$ scelta con $m < n$. Da questo segue la definizione di `compression ratio` come $CR = n/m$. Un'altra considerazione è legata al fatto che maggiore è la compressione che si applica al segnale (CR grande), minore è la dimensione del dato a valle dell'encoder e

[2]Dal punto di vista pratico, θ contiene κ elementi significativamente diversi da zero, mentre gli altri sono approssimativamente nulli. In questo caso il segnale è comprimibile.

minore sarà il valore di κ. Segue che quanto appena detto possa essere definito asintoticamente come $m = \mathcal{O}(\kappa log(n/\kappa))$.

Un esempio certamente significativo in questo senso è l'acquisizione e la trasmissione di immagini ottenute da una risonanza magnetica poiché le moderne tecniche diagnostiche impongono il salvataggio e la trasmissione una grande quantità di immagini. L'impiego di una tecnica di elaborazione dell'immagine tradizionale potrebbe risolvere il problema in questione utilizzando algoritmi di compressione con perdite che porterebbero alla privazione di alcuni dettagli a vantaggio della riduzione di occupazione di memoria. L'impiego del CS consente di ridurre il tempo necessario per il completamento dell'esame e analizzando le diverse tipologie di esami condotti si ottiene, ad esempio, un incremento di un fattore 2 nelle risonanze magnetiche effettuate sull'encefalo e un incremento di un fattore 12.5 nelle angiografie a risonanza magnetica. Occorre tuttavia precisare che i risultati descritti sono risultati limite, nel senso che descrivono il massimo speed up raggiungibile a fronte di una perdita della qualità dell'immagine che comunque rimane accettabile. [3] Analizzando diversi articoli in materia si verifica come una riduzione del tempo per completare l'esame di un fattore 12.5x sia raggiungibile, ma solo in alcune applicazioni. Osservando Fig. 1.1 si osserva come, fissato un gold-standard ed eseguite risonanze magnetiche su una gamma di pazienti, in molti casi l'impiego del CS risulta conveniente poiché nel 82% circa dei test effettuati la differenza fra il risultato ottenuto e il gold-standard è trascurabile o l'immagine acquisita è comunque accettabile. [3]

Osservati i vantaggi introdotti nell'encoder da questa tecnica, occorre anche tenere in considerazione gli svantaggi; in particolare, il problema più significativo da dover prendere in considerazione è l'aumento del costo computazionale per la decodifica del segnale. Infatti, per ricostruire il segnale occorre ricom-

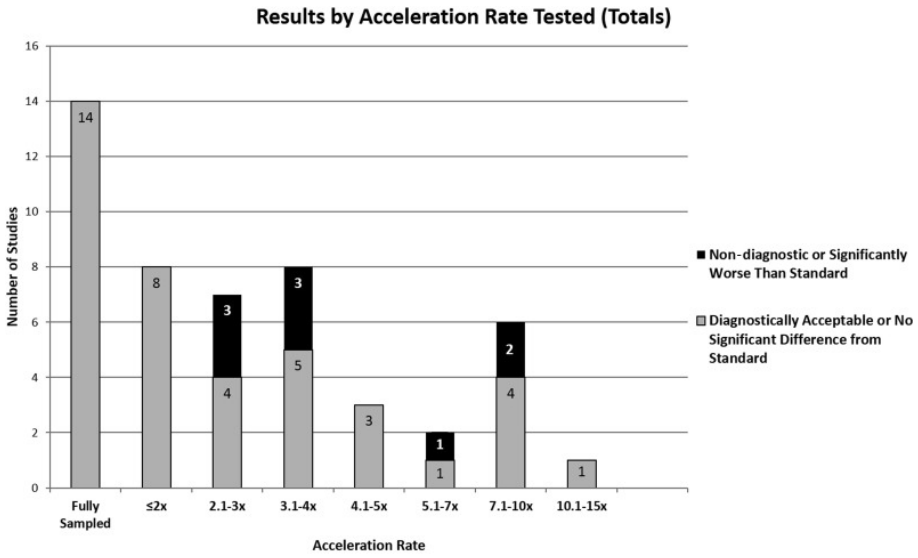


Figura 1.1: Fattore di riduzione nella durata del test nei vari studi condotti. Il colore grigio indica il numero di studi che ottengono immagini accettabili ad una risoluzione data, mentre il colore nero indica il numero di studi che ottengono immagini non accettabili.

porre il segnale sparso $x \in \mathbb{R}^n$ utilizzando un set di m elementi con $m < n$ che descrivono l'output dell'encoder utilizzando il CS. Formalmente occorre quindi definire un operatore $\mathcal{R}_A : \mathbb{R}^m \mapsto \mathbb{R}^n$ tale per cui, almeno idealmente, si possa definire $x = \mathcal{R}_A(\mathcal{L}_A(x))$ anche se poi dal punto di vista realizzativo non si potrà ottenere esattamente il segnale di partenza, ma una sua versione il più simile possibile. La necessità di risolvere un problema di questo tipo (detto anche problema NP-difficile) richiede l'impiego e la risoluzione di problemi di ottimizzazione detti Basis Pursuit nel caso di assenza di rumore. Il problema è definito come

$$min_x \|x\|_1 \ s.t. \ y = Ax \tag{1.4}$$

Il fatto che la soluzione al problema di decodifica sia possibile utilizzando un problema di ottimizzazione come quello appena citato permette di utilizzare nella pratica il CS, ma allo stesso tempo introduce un onere computazionale che nei sistemi a bassa complessità non è sostenibile. Per questo motivo, algoritmi alternativi come il Generalized Approximate Message Passing (GAMP) possono essere presi in considerazione. In aggiunta a questo insieme di algoritmi, solo recentemente si sono integrate le reti neurali profonde (DNN) all'interno di questo ambito. Sebbene queste soluzioni introducano grandi vantaggi in termini di riduzione del costo computazionale e incremento della qualità del segnale a valle della ricostruzione, occorre tenere a mente le problematiche introdotte da una soluzione di questo tipo (in particolar modo legate alla complessità del sistema finale e alla necessità di salvare il modello della DNN utilizzato). [1]

1.2 Le reti neurali profonde

Una rete neurale (NN) può essere definita come un sottoinsieme finito di elementi, chiamati neuroni e descritti dall'insieme $N = \{u_1, u_2, ..., u_k\}$, $k \in \mathbb{N}$

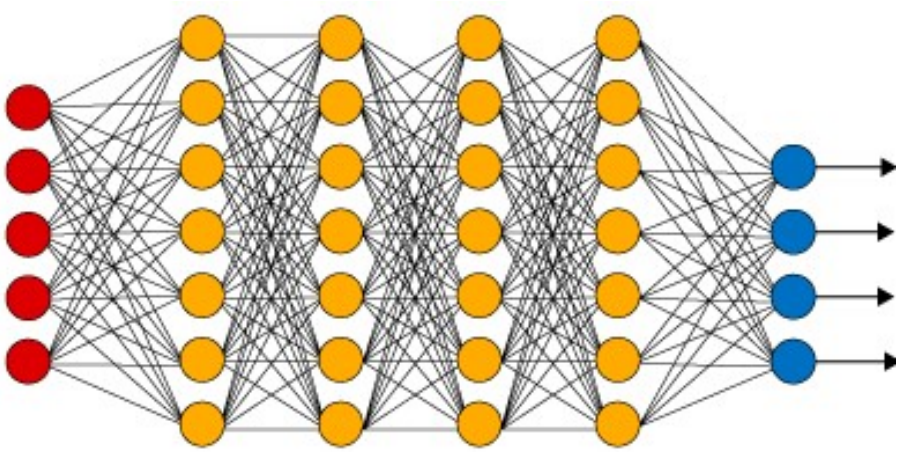


Figura 1.2: Struttura di una DNN a 6 livelli con un livello di ingresso (neuroni rossi), 4 layer profondi (neuroni gialli), un layer di uscita (neuroni blu) e descrizione delle interconnessioni fra i vari neuroni.

e un set finito di connessioni fra i vari nodi $H \subseteq N \times N$. L'idea principale nella definizione di una NN è la definizione di un set di parametri, detti pesi $\omega_i,\ i = (0, ..., n-1)$ e bias $b_i,\ i = (0, ..., n-1)$, che siano allenabili (una volta definiti possono eventualmente essere modificati) e a valori reali.[4] Compresa la struttura di una NN a 2 layer, si possono aggiungere un numero arbitrario (compatibilmente con la tipologia e la complessità dell'applicazione[3]) di layer fra il layer d'ingresso e quello di uscita costruendo una rete neurale profonda (DNN). Si consideri per generalità la DNN di Fig. 1.2. Si può descrivere il comportamento di un layer come:

$$h^k = f(b^k + W^k h^{k-1}) \tag{1.5}$$

[3]Se il modello definito possiede un numero di parametri troppo elevato rispetto alla complessità del problema si incorre in un problema di sovradattamento per cui il modello perde di generalità ed adattandosi alle caratteristiche del training set non è in grado di generalizzare l'apprendimento.

Dove h^k è il risultato calcolato al k-esimo livello (livello corrente) e W^k è la matrice dei pesi che permettono di determinare il risultato al k-esimo livello conoscendo l'output del livello precedente. [5] Si somma poi un bias b^k che assume valori reali e si applica una funzione non lineare, scelta fra una vasta gamma, in relazione al risultato desiderato e alla tipologia di rete che si vuole implementare. Alcuni esempi, le cui rappresentazioni grafiche sono riportate in Fig. 1.3, sono:

- Rectified Linear Unit (ReLU): implementa una funzione non lineare definita come la parte positiva del suo argomento:

$$ReLU = max(x, 0) \tag{1.6}$$

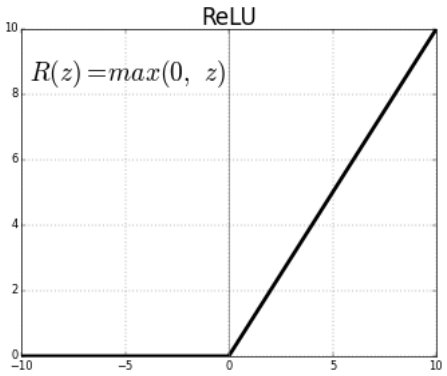
  Fra i principali punti di forza di questa funzione non lineare si evidenziano la semplicità computazionale, l'efficienza computazionale e l'invarianza di scala.

- Sigmoid: implementa una funzione non lineare definita come:

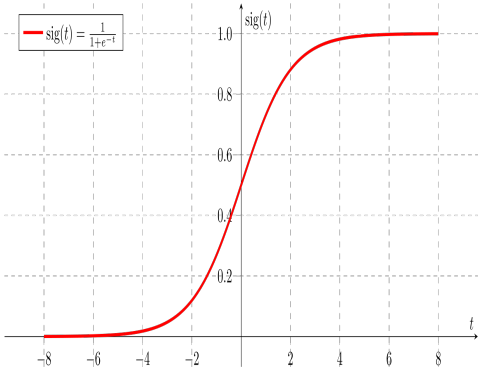
$$Sigmoid = \frac{1}{1 + exp(-x)} \tag{1.7}$$

  La funzione in questo caso ha un costo computazionale maggiore, ma grazie al suo andamento a "S" consente di mantenere in un range limitato fra 0 e 1 tutti i possibili valori che il neurone può assumere durante la fase di training.

Durante la fase di training occorre quindi determinare la combinazione migliore di pesi e bias che minimizzi l'errore commesso nel raggiungimento di un determinato risultato (ad esempio il riconoscimento di un numero). Affinché questo sia possibile, si introduce una funzione costo che deve essere minimizzata. Ponendo h il vettore dei risultati attesi e $\hat{h}$ il vettore dei risultati ottenuti



(a) Funzione di attivazione ReLU



(b) Funzione di attivazione Sigmoid

Figura 1.3: Esempi di funzioni di attivazione non lineari da applicare ad ogni neurone della rete.

con l'impiego della DNN si ottiene una funzione di costo $L(W, b)$

$$L(W, b) = \|h - \hat{h}\|_2 \tag{1.8}$$

da cui segue che si raggiunge il risultato migliore quando la funzione di costo descrive un minimo globale, mentre il risultato è ottimo quando la funzione di costo è nulla. Si suppone, per semplicità nella descrizione di quanto segue, che il vettore dei bias b sia il vettore nullo; allora per determinare la configurazione di W che rende minima la funzione di costo occorre risolvere un problema del tipo $\nabla L(W) = 0$. Appare immediato che presa una DNN con un numero elevato di neuroni, la difficoltà nella ricerca di un minimo globale cresca enormemente e per questo motivo occorre adottare un approccio iterativo combinando gli algoritmi di discesa stocastica del gradiente e la retropropagazione dell'errore.

Affinché la ricerca dei pesi che rendono minima la funzione di costo sia soluzione del problema di apprendimento si suppone che la funzione $L(W, b)$ sia continua e differenziabile e quindi si suppone di utilizzare una funzione Sigmoid invece che una ReLU. Per poter definire gli step di questo algoritmo si definisce un training set $\{(x_1, h_1), ..., (x_p, h_p)\}$ contenente p coppie (pattern

di ingresso e uscita), si suppone che inizialmente i pesi siano scelti casualmente e si calcola $L(W, b)$. Nota la funzione di costo, se ne calcola il gradiente

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_l} \right) \tag{1.9}$$

dove l indica il numero di pesi in quel determinato layer. A questo punto si aggiorna il valore del peso definendo l'incremento Δw_i come

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \; per \; i = 1, ..., l \tag{1.10}$$

con γ che definisce la costante di apprendimento, ovvero il parametro moltiplicativo che definisce la lunghezza dello step di ogni iterazione nella ricerca del minimo globale. [6] Ripetendo iterativamente gli ultimi due step si raggiunge una condizione di convergenza che può essere un minimo globale (caso ottimo), un minimo locale o un plateau (caso peggiore).

Terminato l'allenamento supervisionato della rete neurale occorre definirne le prestazioni. Si possono definire diverse performance, fra cui l'*accuracy* e la *loss*, di cui si discuterà più nel dettaglio discutendo alcuni aspetti essenziali durante la presentazione dell'architettura proposta nel capitolo 3.

Presentata brevemente la struttura e il funzionamento di una DNN resta da descrivere i principali vantaggi e svantaggi nell'impiego di una soluzione di questo tipo. I principali vantaggi sono:

- Elevate prestazioni quando il problema in esame è un problema non lineare con una grande dimensionalità. Non a caso fra le applicazioni più sviluppate si trova il riconoscimento di oggetti in un'immagine. Le performance molto interessanti sono raggiungibili grazie alla possibilità di suddividere un problema complesso in un problema più semplice analizzato in diversi livelli.

- Elevata flessibilità. Sapendo che la DNN è descrivibile come un modello matematico con una certa struttura, qualsiasi dato descrivibile per via

numerica può essere studiato con una DNN. Inoltre, la struttura della rete può essere modificata aggiungendo livelli o modificando il numero di neuroni di ogni layer.

- Capacità di risolvere efficacemente sia problemi classificativi che problemi di regressione.

- Il modello dopo essere stato allenato può essere memorizzato e riutilizzato senza essere riallenato o modificato. In questo senso la capacità predittiva a valle del training risulta piuttosto elevata.

I principali svantaggi sono:

- Presenza di under-fitting o over-fitting quando, rispettivamente, il modello è troppo semplice o troppo complesso se confrontato con il problema in esame. Nel primo caso non si è in grado di raggiungere una performance accettabile e la rete non è in grado di adempiere al compito per il quale è stata creata, mentre nel secondo caso la rete raggiunge altissime performance durante il training, ma perde di generalità e quando si applicano come input dati che non sono mai stati impiegati prima, le performance diminuiscono notevolmente. Per questo motivo, occorre conoscere a fondo il problema in esame prima di sviluppare l'architettura della DNN da utilizzare.

- Elevato costo computazionale e grande dispendio di tempo se il training è svolto con un'architettura single-core. In questo senso, si può ovviare il problema utilizzando architetture parallele come le Graphics Processing Units (GPU).

- Visione della DNN come una scatola nera. Sebbene si conosca il funzionamento della rete e gli step algoritmici seguiti durante la fase di

training e di predizione, non è possibile sapere in che maniera gli ingressi influiscano sul risultato finale.

Si conclude riportando un esempio di realizzazione di DNN per il riconoscimento di numeri scritti a mano. Per l'implementazione dell'esempio si è utilizzato il database `mnist`. Si tratta di un database composto da immagini 28x28 pixel con numeri scritti a mano. L'obiettivo è la realizzazione di una DNN che con una certa affidabilità (*accuracy*) sia in grado di riconoscere questi numeri scritti a mano. Per prima cosa occorre definire un training set e un test set (il primo utilizzato per l'allenamento della rete prevede di fornire a quest'ultima sia il pattern di ingresso che quello di uscita per implementare l'algoritmo di retropropagazione dell'errore, mentre il secondo è utilizzato per testare la rete e questa non conosce il pattern di uscita corretto, ma solo quello d'ingresso) suddividendo il data set. Dopodichè si costruisce il modello della DNN scegliendo il numero di layer profondi e il numero di neuroni in ciascuno di questi layer. Infine occorre definire alcuni parametri utili alla valutazione del training.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.compile(optimizer='adam',
```

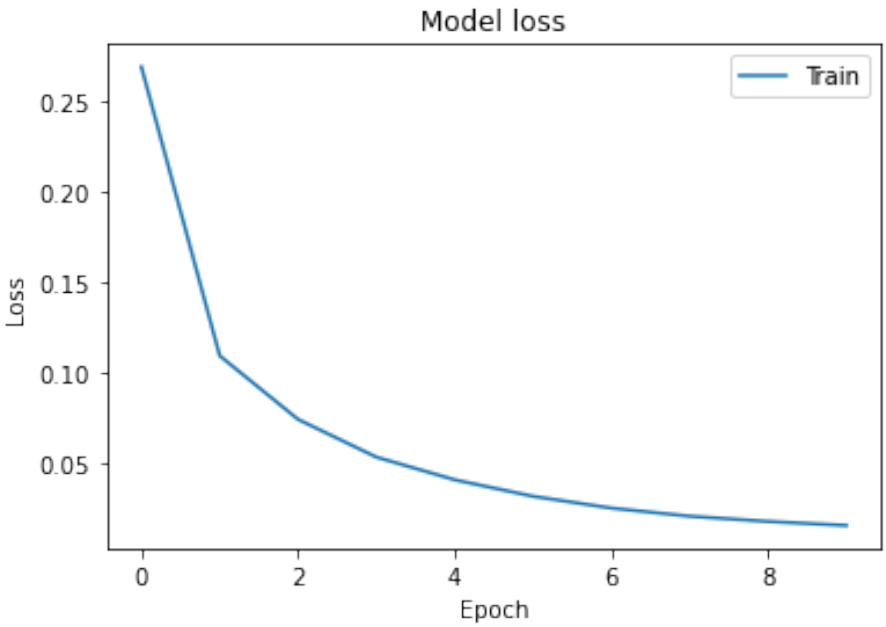


Figura 1.4: Evoluzione della curva di loss al variare delle epoche durante la fase di training della DNN.

```
14              loss='sparse_categorical_crossentropy',
15              metrics=['accuracy'])
16
17 model.fit(x_train, y_train, epochs=10)
```

Listato 1.1: Esempio di codice Python che implementi una Rete Neurale Profonda, la sua configurazione e il suo allenamento fissando funzione di attivazione e metriche osservate.

Nel listato 1.1 si riporta il codice utilizzato per descrivere questo semplice esempio. Eseguendo questo training si raggiunge un valore di accuracy intorno al 99.52% e un valore di loss intorno a 0.01. Al crescere del numero di epoche in cui si svolge il training l'accuracy cresce, mentre la loss diminuisce, ma occorre prestare attenzione al fenomeno dell'overfitting per cui se la rete ha troppi neuroni o è allenata per un numero di epoche troppo elevato, essa

perde di generalità e raggiunge performance molto elevate con il training set e performance scadenti con il test set. Infine, la curva di Fig. 1.2 descrive l'evoluzione della curva di loss con il passare delle epoche. Come già detto, si rimanda al capitolo 3 per una descrizione più approfondita delle DNN.

Capitolo 2

Segnali biomedici

In questo secondo capitolo si effettuerà una panoramica sul segnale elettrocardiografico (ECG) descrivendone la genesi e analizzandone la sparsità.

2.1 Genesi del segnale elettrocardiografico

Il segnale ECG è un segnale biomedico acquisibile attraverso l'impiego di una serie di elettrodi posizionati sul corpo del paziente che misurino l'attività elettrica del cuore. La posizione degli elettrodi sul corpo del paziente definisce una fissata forma d'onda poiché si può pensare sostanzialmente il segnale acquisito come la proiezione del campo elettrico su tali elettrodi. Si suppone di analizzare un segnale ECG utilizzando la regola del triangolo di Einthoven secondo cui si dispone di 10 elettrodi posizionati sul corpo del paziente, in grado di fornire 12 derivazioni complessive. Gli elettrodi sono distribuiti sul torace (6 elettrodi) e sugli arti (4 elettrodi) dell'individuo sottoposto all'esame, secondo una disposizione riportata in Fig. 2.1a. In particolare, i 4 elettrodi periferici (uno impiegato per la massa e tre per analizzare i potenziali d'azione) posizionati sugli arti, consentono di costruire un triangolo equilatero su cui

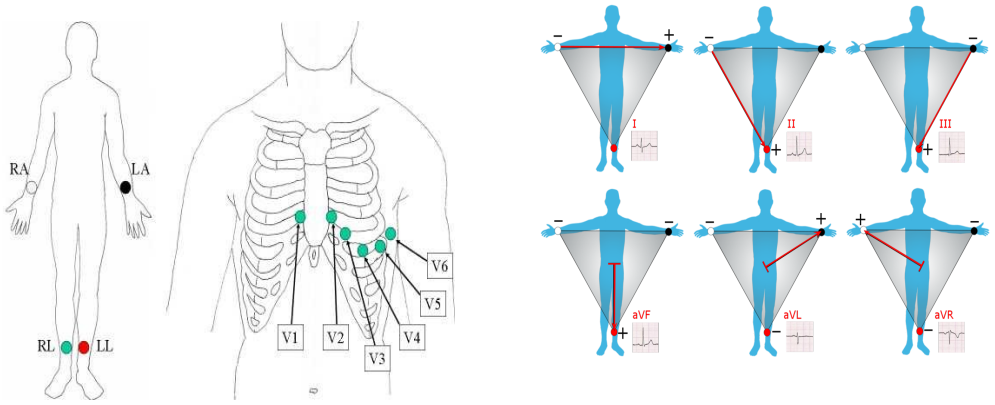
effettuare la proiezione del vettore cardiaco. Gli elettrodi periferici consentono la misurazione dei potenziali aVL, aVR, aVF (derivazioni unipolari degl'arti) definibili, utilizzando la notazione di Fig. 2.1a, dalle seguenti equazioni:

$$\begin{aligned} aVF &= LL - \frac{1}{2}(RA + LA) \\ aVR &= RA - \frac{1}{2}(LA + LL) \\ aVL &= LA - \frac{1}{2}(RA + LL) \end{aligned} \tag{2.1}$$

Le suddette derivazioni sono ottenibili a partire dal Triangolo di Einthoven utilizzando come direzioni di proiezione le bisettrici degli angoli del triangolo e derivano dagli stessi elettrodi di D_I,D_{II} e D_{III}. Le derivazioni aVR, aVF e aVL congiuntamente a D_I,D_{II} e D_{III} costituiscono la base del sistema di riferimento esassiale utilizzato per analizzare l'asse elettrico sul piano frontale (verticale). [10] Per analizzare l'asse elettrico sul piano trasversale (perpendicolare al primo trattato) si utilizzano le derivazioni precordiali di Wilson, utili anche ad effettuare misurazioni più dettagliate dell'attività cardiaca.

Il campo elettrico misurabile è dovuto alla variazione del potenziale d'azione che si propaga attraverso il cuore e grazie alla depolarizzazione e ripolarizzazione delle cellule nervose dello stesso si può ricavare il tracciato di Fig. 2.2. É utile ricordare che, essendo il segnale ECG ottenuto per proiezione del vettore cardiaco è assimilabile alla combinazione lineare delle ripolarizzazioni e delle depolarizzazioni delle varie cellule cardiache. Le fasi del ciclo cardiaco sono:

- Innesco del nodo senoatriale.
- Inizio dell'attività atriale e diffusione dello stimolo nervoso attraverso le superfici atriali.
- Raggiungimento del nodo atrioventricolare (AV). Il nodo AV funge da nodo di Peacemaker introducendo un ritardo nella propagazione del se-



(a) Posizione degli elettrodi centrali e periferici sul corpo del paziente.

(b) Potenziali misurabili con gli elettrodi periferici posizionati sugli arti del paziente.

Figura 2.1: Posizionamento degli elettrodi e segnali acquisibili dal corpo del paziente per la rilevazione del tracciato cardiaco.

gnale elettrico e garantendo il corretto afflusso di sangue fra gli atri e i ventricoli.

- L'impulso attraversa il fascio di His (fascio atrioventricolare comune) e raggiunge le fibre del Purkinje.
- Si completa la contrazione atriale e inizia la contrazione ventricolare. [8]

Facendo riferimento a Fig. 2.2 si possono identificare i tratti caratteristici del segnale ECG:

- Onda P: dovuta alla depolarizzazione degli atri.
- Segmento PQ: dovuto alla conduzione dell'impulso nervoso attraverso il nodo AV e il fascio AV.
- Onda Q: dovuta alla depolarizzazione del setto interventricolare.
- Onda R: dovuta alla depolarizzazione dell'apice del ventricolo sinistro.

Figura 2.2: Rappresentazione di un tracciato ECG fisiologico.

- Onda S: dovuta alla depolarizzazione delle regioni basale e posteriore del ventricolo sinistro.
- Segmento ST: dovuto alla ripolarizzazione ventricolare.
- Onda T: dovuto alla ripolarizzazione ventricolare. [8]

Per quanto riguarda l'aspetto ingegneristico, il segnale ECG può essere descritto nella tassonomia dei segnali biomedici come un segnale periodico e di conseguenza predicibile; una vasta gamma di algoritmi basati sulla costruzione o meno di un template posso essere impiegati per effettuare il riconoscimento di una forma d'onda (sia patologica che fisiologica). La costruzione del template può essere fatta dividendo il segnale in una fase di training in cui si costruisce il template e in una fase di testing dove il template è impiegato nella ricerca della forma d'onda. Inoltre, la componente di training può essere definita su una singola forma d'onda del segnale o su molteplici (aumentando la generalità del template, ma aumentando anche il costo computazionale per il calcolo dello stesso) e può essere ridefinito ciclicamente per migliorare l'affidabilità nel riconoscimento della forma d'onda. Gli algoritmi senza template,

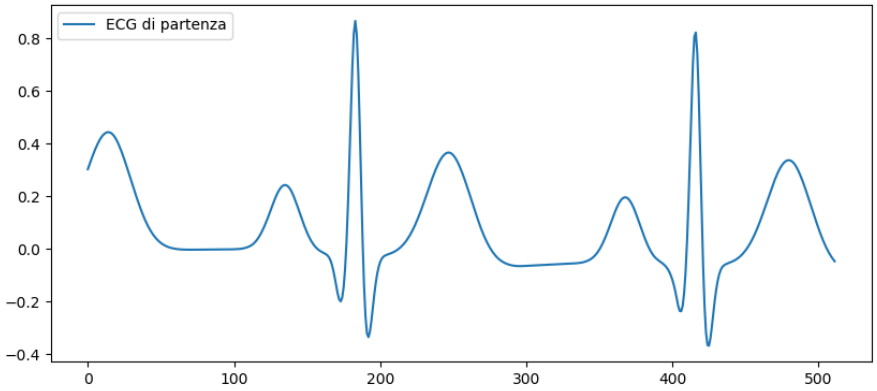


Figura 2.3: Rappresentazione di un segnale ECG nel dominio del tempo. Sull'asse dei tempi è riportata la lunghezza del segnale in campioni.

o euristici, invece, si basano sull'impiego di strumenti, come per esempio soglie e derivate, per poter riconoscere alcuni connotati specifici della forma d'onda in esame. Il segnale ECG, inoltre, ha un comportamento in frequenza prettamente passa-basso compreso fra qualche decimo di Hertz e le decine di Hertz che può essere descritto come la combinazione di due contributi. Un primo contributo legato al complesso QRS fra qualche Hertz e le decine di Hertz, con un'ampiezza significativa visto che il segnale nel dominio del tempo ha una durata ridotta e un'ampiezza elevata, e un secondo contributo legato alle onde P e T fra qualche decimo di Hertz e qualche Hertz, con un'ampiezza molto minore poiché queste componenti sono più piccole rispetto al complesso QRS. Per poter analizzare il comportamento in frequenza del segnale si è scelto di utilizzare il Metodo di Welch determinando la densità spettrale di potenza del segnale, partendo dal segnale ECG di Fig. 2.3. Si è utilizzato un segnale ECG di lunghezza pari a 512 campioni, si è posta una frequenza di campionamento fs pari a 256Hz e si è scelto di utilizzare una finestra di Hamming. Lo spettro risultante è quello di Fig. 2.4 dove si può osservare il comportamento passa

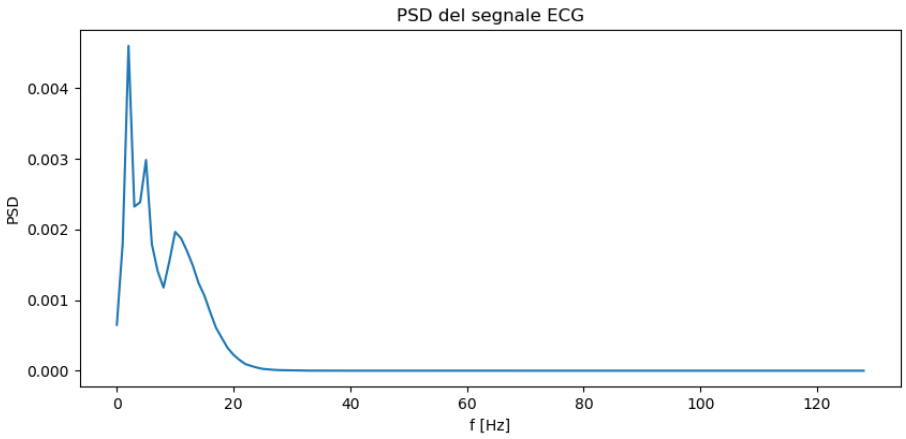


Figura 2.4: Rappresentazione della densità spettrale di potenza del segnale ECG.

basso descritto in precedenza utilizzando una rappresentazione dell'ordinata in scala lineare. Una descrizione più dettagliata del segnale ECG può essere fornita se si considera la sua rappresentazione nel piano tempo frequenza, utilizzando una Short Time Fourier Transform (STFT), indicata in Fig. 2.5, poichè si può definire in quali intervalli temporali sono presenti certe frequenze. Si considera per la realizzazione del risultato appena citato lo stesso segnale ECG di durata pari a 512 campioni utilizzato nell'analisi precedente, campionandolo con una frequenza di campionamento fs pari a 256Hz e utilizzando una finestra di Hamming per la suddivisione del piano tempo-frequenza. La rappresentazione colorata mostra sostanzialmente cinque zone con un contenuto informativo dominante rispetto al resto del piano tempo-frequenza. Due di queste, intorno a 0.50 e 1.50 secondi, (corrispondenti circa ai campioni 190 e 420) mostrano un comportamento passa banda con un'ampiezza piuttosto elevata in corrispondenza dei due complessi QRS, mentre gli altri tre contenuti informativi, intorno a 0, 1.0 e 2.0 secondi, descrivono le componenti delle onde T e P dei segnali ECG considerati nei 512 campioni. Noti i limiti della STFT

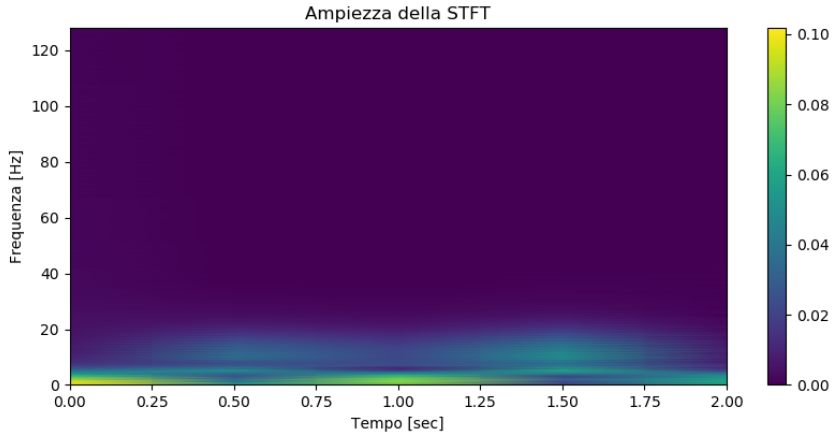
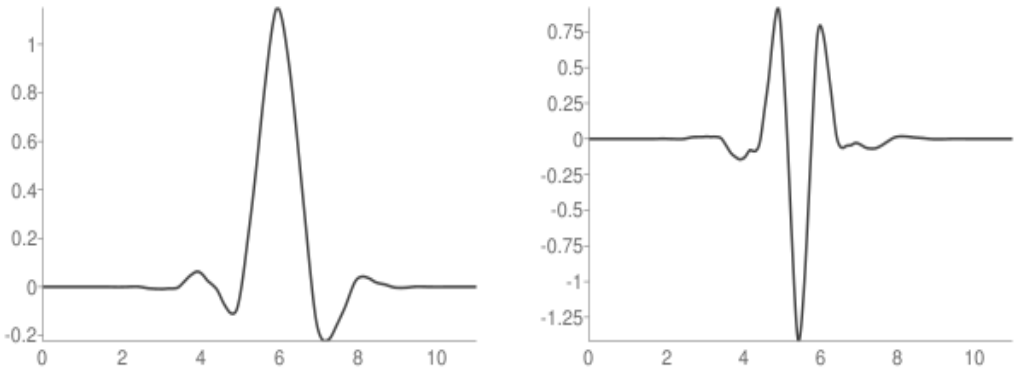


Figura 2.5: Rappresentazione nel piano tempo frequenza tramite STFT del segnale ECG.

in termini di risoluzione temporale e frequenziale, è interessante osservare come i due complessi QRS non siano localizzati in maniera precisa, ma si possa avere un'idea abbastanza corretta di dove si trovino evidenziando appunto i due contributi passa banda descritti dalla STFT, mentre per le onde T e P non si può dire assolutamente nulla sulla loro localizzazione poichè la porzione di piano tempo-frequenza che descrive questi contenuti non è in grado di separare sull'asse dei tempi i due contenuti. Nonostante le problematiche relative alla localizzazione dei risultati, si conferma comunque il trend descritto con il metodo di Welch relativamente al contenuto in frequenza.

Tuttavia, non è ragionevole esprimere il segnale ECG con una sua rappresentazione sparsificata utilizzando come base sparsa una base di esponenziali complessi poiché il numero κ di elementi non nulli preso un generico segnale potrebbe essere confrontabile con n, violando la teoria alla base del CS. Per poter descrivere meglio il segnale occorre identificare una base appropriata che sia conforme con le caratteristiche richieste dalla teoria del Compressed Sensing. In particolar modo, si farà sempre riferimento d'ora in poi all'impiego di



(a) Funzione di scala $\varphi[n]$ (b) Funzione wavelet $\psi[n]$

Figura 2.6: Funzioni di scala e wavelet per l'analisi multirisoluzione del segnale ECG.

una wavelet di tipo Symlet 6, un caso particolare delle wavelet di Daubechies che garantisce una regolarità maggiore rispetto al caso generale. Più nel dettaglio, si tratta di una famiglia di funzioni che consente di implementare una trasformata wavelet tempo-discreta e gode delle seguenti proprietà:

- ortogonalità;
- biortogonalità;
- quasi simmetria.

Essendo una trasformata wavelet permette di implementare un'analisi multirisoluzione discreta grazie all'impiego delle funzioni di scala ($\varphi[n]$) e wavelet ($\psi[n]$) riportate rispettivamente in Fig. 2.6a e 2.6b [9]. Inoltre, si osservi che, l'impiego di una trasformata wavelet discreta con le suddette caratteristiche soddisfa a pieno le ipotesi fatte per rendere applicabile la teoria sul Compressed Sensing.

2.2 Analisi della sparsità

Si vuole creare un Test Set, un Training Set e un Validation Set senza fissare a priori il valore di sparsità del segnale ECG considerato, ma determinandolo su ogni singola istanza di segnale in relazione ad una soglia di energia fissata. Così facendo la sparsità assume valori ottimi in maniera dipendente dal contenuto analizzato e non dipende da un valore fissato a priori che descriva correttamente una porzione statisticamente significativa dell'intero data set. La sezione è così organizzata: in una prima parte si mostrano i passi dell'algoritmo su una singola istanza di segnale ECG, poi si mostra l'algoritmo nella sua interezza ed infine si valutano i risultati ottenuti mostrando il rapporto κ/n al variare di n.

2.2.1 Descrizione dell'algoritmo su una singola istanza di segnale ECG

Il primo passo per la realizzazione dell'algoritmo è la lettura di due file contenenti segnali ECG a 60 e 80 battiti al minuto. I segnali sono stati letti, concatenati in un unico vettore di dimensione (512,200000) e mescolati in maniera casuale per evitare che le performance finali possano dipendere da un pattern con cui i due tipi di segnali sono mescolati. Si mostra poi in Fig.2.7 un'istanza del segnale contenuto nel data set di partenza che si sta considerando. Dopodiché si vuole effettuare un'operazione di data augmentation e per farlo si introducono due gradi di libertà:

- La dimensione del segmento da trattare, indicato con n;
- Il numero di segmenti da considerare all'interno di ogni istanza dell'ECG di partenza, indicato con $maxn$.

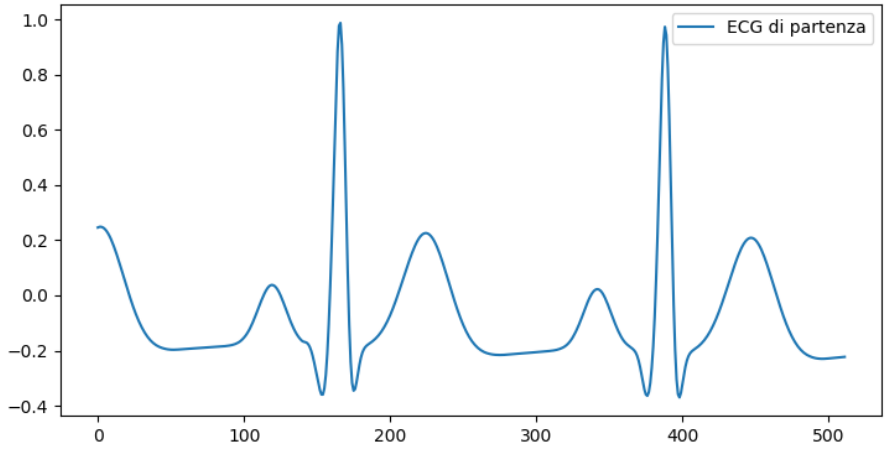


Figura 2.7: Singola istanza del segnale ECG di partenza.

Si ottiene, grazie all'impiego di *maxn*, una data augmentation di un fattore *maxn*. Si considerano d'ora in poi n pari a 128 e *maxn* pari a 100; quindi a valle di questo passo, per ogni istanza del segnale ECG di partenza, saranno estratti 100 segmenti di segnale ECG (scelti casualmente all'interno di ogni istanza) ognuno di lunghezza pari a 128. D'ora in poi si farà riferimento a questi *maxn* segnali ECG (di una singola istanza del cluster di partenza) che saranno utilizzati per completare l'algoritmo. Si osserva che aver considerato segnali ECG con diverse frequenze cardiache non altera la struttura dell'algoritmo implementato, ma consente di renderlo più generale mostrando come il valore di sparsità che descrive correttamente il segnale dipenda anche dalla frequenza. Da un punto di vista generale, mostrato questo algoritmo, si potrebbe generalizzare prendendo segnali ECG con frequenze cardiache tutte diverse e l'algoritmo sarebbe comunque performante.

In Fig 2.8 si mostra una rappresentazione grafica dei segnali a valle del primo passo dell'algoritmo, potendo descrivere l'operazione di data augmentation su una singola istanza come la scelta di una sequenza di n campioni contigui

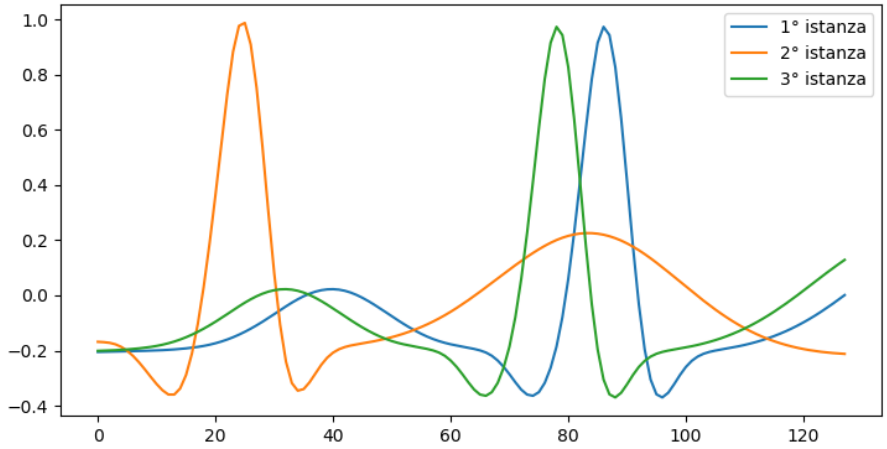


Figura 2.8: Step 1: Data augmentation di un fattore *maxn*. Rappresentazione di tre istanze a valle dell'operazione.

all'interno del segnale di partenza; formalmente:

$$x = ECG(k : k + n) \tag{2.2}$$

dove $ECG(\cdot) \in \mathbb{R}^N$ è il segnale ECG di partenza su cui si applica l'operazione di data augmentation e k è un numero casuale, estratto da una distribuzione gaussiana a valor medio nullo e varianza unitaria, tale che $k \in [0, N - n]$ con N lunghezza del segnale ECG considerato. L'operazione di data augmentation si ripete *maxn* volte sulla singola istanza di segnale ECG definendo un insieme I che contenga tutti i *maxn* elementi $x \in \mathbb{R}^n$ ottenuti allo step precedente. In questo caso non si osservano distinzioni fra i segnali poichè si rappresentano solamente 3 segnali di lunghezza pari a 128 campioni a partire da un segnale iniziale di 512 campioni.

Dopo aver manipolato il data set aumentandone la dimensionalità, occorre passare dal dominio del tempo al dominio dell'analisi tempo-frequenza. Il passaggio di dominio è effettuabile con un prodotto matriciale e si sceglie come

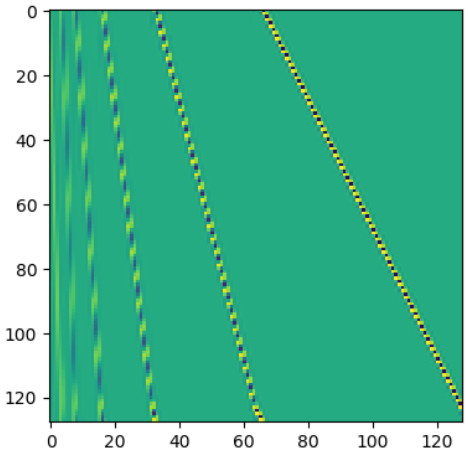


Figura 2.9: Rappresentazione grafica della trasformata wavelet Symlet 6.

trasformata Wavelet una Symlet 6. L'operazione da descrivere in questo caso è un prodotto matriciale fra le colonne della matrice di sparsità e il segnale ottenuto dallo step precedente. Per ogni segnale x descritto nello step precedente, definita la matrice di sparsità $D \in \mathbb{R}^{n \times n}$, si ricava la rappresentazione sparsa nel dominio tempo-frequenza, tramite l'operazione riportata in seguito:

$$\xi = D^{\top} \cdot x \ , \forall x \in I \tag{2.3}$$

con $\xi \in \mathbb{R}^n$. Le Fig. 2.9 e 2.10 mostrano rispettivamente una rappresentazione della Symlet 6 e il risultato ottenuto a valle del passaggio nel dominio tempo-frequenza (per le stesse 3 istanze del caso precedente).

Come si può osservare da Fig. 2.10, partendo da n pari a 128 campioni, sono necessari molti meno campioni per descrivere correttamente il segnale; infatti, molti di questi campioni sono nulli e sono relativamente pochi quelli che, in valore assoluto, contribuiscono significativamente all'energia del segnale. Da Fig. 2.10 si ottiene anche un riscontro pratico per quanto riguarda il concetto di sparsità. Il segnale ECG è un segnale per sua natura sparso, poiché, se

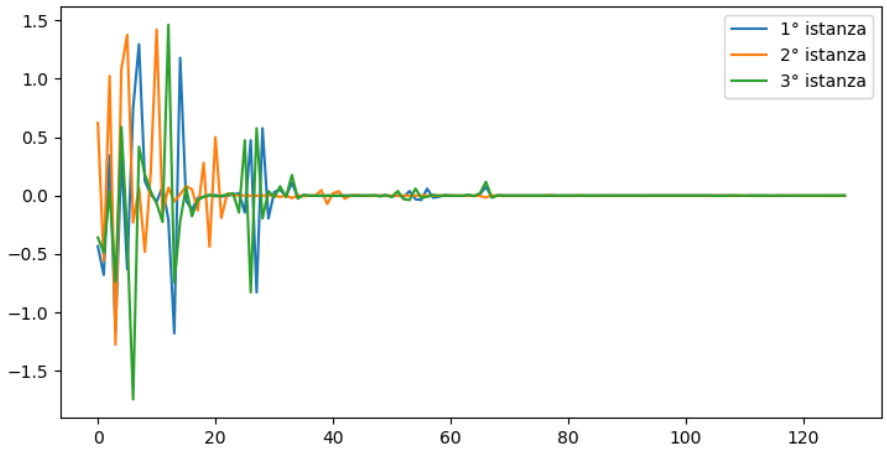


Figura 2.10: Step 2: Coefficienti della rappresentazione nel dominio trasformato dei segnali di partenza, descritti in Fig. 2.8.

rappresentato in un'opportuna base ha un numero κ di elementi non nulli molto minore di n (come si osserva nella sopracitata figura).

Sapendo che solo κ campioni della rappresentazione tempo-frequenza sono non nulli, occorre scegliere quale sia la percentuale di energia del segnale da preservare a valle di questo step per mantenere una ricostruzione accettabile. Fissato il valore di energia, si deve quindi determinare quali campioni della rappresentazione debbano essere preservati per ottenere una certa ricostruzione del segnale. Si introduce quindi un ulteriore grado di libertà definito dalla soglia di energia *ene_th*. In questo step si svolgono i seguenti passi:

- Si calcola l'energia di ogni istanza del segnale ECG;
- Si normalizzano le istanze per avere energia unitaria e combinando le due operazioni su ciascun segnale sparso si ottiene:

$$\xi_{norm}(i) = \frac{\xi(i) \cdot \xi(i)}{\sum_{j=0}^{n-1} \xi(j) \cdot \xi(j)}, \ \forall i \in [0, n-1] \tag{2.4}$$

con $\xi_{norm} \in \mathbb{R}^n$.

- Si definisce la somma cumulativa delle energie in ordine crescente, indicato con *vals*;

- Si definiscono gli indici associati alle componenti di segnale sommate ad ogni passo della somma cumulativa e si indica il vettore risultante con $inds \in \mathbb{R}^n$;

- Si definisce la sparsità κ per ogni istanza di segnale fissata *ene_th* analizzando la soglia di energia e gli elementi del vettore delle somme cumulative (operativamente parlando κ è definibile come $\kappa = \#[vals \leq ene\_th]$). Si osserva che i valori di κ sono compresi mediamente fra 7 e 16.

A questo punto si determinano quali campioni debbano essere preservati affinché si possa raggiungere la quota di energia fissata per determinare κ ($\kappa \ll n$ elementi non nulli). Definita la sparsità occorre creare un vettore $y \in \mathbb{R}^n$ (supporto del segnale sparsificato fissata l'energia *ene_th*) così descrivibile:

$$y(i) = \begin{cases} 1, & \text{se } i \in inds \\ 0, & \text{se } i \notin inds \end{cases} \tag{2.5}$$

La Fig. 2.11 mostra il risultato ottenuto sovrapponendo il segnale ottenuto al 2° step (rappresentazione tempo-frequenza del segnale di partenza, linea continua) e quello ottenuto ora (supporto del segnale sparsificato, punti) e si osserva come con una soglia pari a 0.99 servano relativamente pochi campioni per avere una ricostruzione accettabile. I campioni che si discostano poco dalla linea basale non sono per questo valore di energia presi in considerazione in quanto aggiungono un livello di dettaglio non richiesto dal livello di energia preso in esame. Dopo aver determinato il supporto del segnale sparsificato occorre preservare i campioni di ξ i cui indici coincidono con gli elementi non nulli del supporto, ottenendo così un vettore ξ con cardinalità κ ($\|\xi\|_0 = \kappa$).

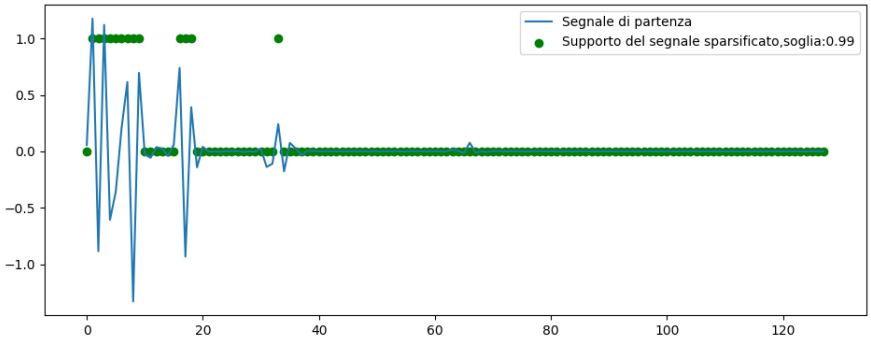


Figura 2.11: Step 4: definizione del supporto del segnale sparsificato, fissata l'energia che il segnale sparsificato descrive sovrapposto alla rappresentazione del segnale nella base sparsa.

Formalmente:

$$\xi(i) = \begin{cases} \xi(i), & \text{se } y(i) = 1 \\ 0, & \text{se } y(i) = 0 \end{cases} \tag{2.6}$$

Si conclude l'algoritmo effettuando l'antitrasformata wavelet, ritornando nel dominio del tempo e definendo un segnale sparsificato $\hat{x}$:

$$\hat{x} = D \cdot \xi \tag{2.7}$$

In Fig. 2.12, si rappresentano poi il segnale di partenza e il segnale sparsificato a valle dell'algoritmo. Aver fissato un valore di soglia di energia pari a 0.99 garantisce un ottimo risultato in termini di ricostruzione del segnale, fatto salvo per qualche campione diverso a causa della scelta della soglia di energia. L'aspetto certamente da osservare è, in linea con le considerazioni fatte nel Capitolo 1, legato alla conservazione del profilo del segnale che si sta analizzando. Il segnale ottenuto come risultato dell'algoritmo permette ancora di distinguere tutti i connotati caratteristici del segnale, come il complesso QRS, l'onda P e l'onda T (seppur solo parziale a causa della scelta di un numero

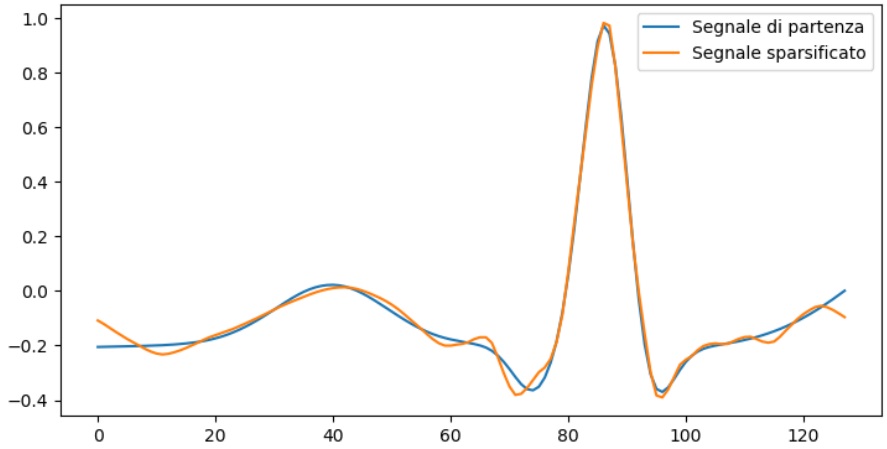


Figura 2.12: Step 5: Confronto fra il segnale sparsificato ricostruito a partire dai campioni che appartengono al supporto e il segnale di partenza.

limitato di campioni), ma utilizzando un numero κ di elementi molto minore di n.

2.2.2 Algoritmo per la valutazione della sparsità dato un dataset arbitrario

Gli step svolti singolarmente e descritti nella sezione precedente sono poi stati raccolti in una funzione che replichi il procedimento per ogni ECG presente nel data set di partenza (unione dei data set a 60 e 80 battiti al minuto e rimescolamento casuale degli elementi). In questo caso non sono stati prodotti grafici durante l'esecuzione della funzione essendo le dimensioni in gioco notevolmente alte e per lo stesso motivo i risultati ottenuti sono stati salvati in un file in memoria ogni 1000 iterazioni considerate, scrivendo ogni volta $1000 \cdot maxn$ elementi per ogni grandezza da salvare. Date le ingenti dimensioni si è preferito optare per un file in formato hdf5 salvando x (segnale di

partenza), *xhat* (segnale sparsificato) e *y* (supporto del segnale sparsificato) piuttosto che un file .mat. Infine, si sono esplorati due gradi di libertà (numero di campioni del dato da trattare e soglia di energia da considerare per la definizione del supporto) con i seguenti valori:

- n: 64, 128, 256
- th: 0.99, 0.999

Per i casi $n = 64$ e $n = 128$ si è utilizzato un $maxn$ pari a 100, mentre per il caso $n = 256$ si è considerato un $maxn$ pari a 20 per evitare la saturazione della memoria. L'algoritmo è stato poi descritto dal seguente pseudo-codice che descrive sommariamente tutti i passi necessari al raggiungimento del risultato voluto.

I dati in questo caso sono stati salvati prima in un unico file chiamato *DataSet* e identificato dai parametri n e th e solo alla fine dell'algoritmo è stato suddiviso in tre file diversi (con un numero di istante diverse a seconda del valore di n considerato), come descritto di seguito:

- Test Set con una dimensione corrispondente all'80% della dimensione totale del file *DataSet*. Per n pari a 64 e 128 si ha una dimensione pari 17×10^6, mentre per n pari a 256 si ha una dimensione pari a 3.2×10^6;
- Training Set con una dimensione corrispondente al 15% della dimensione totale del file *DataSet*. Per n pari a 64 e 128 si ha una dimensione pari 200×10^3, mentre per n pari a 256 si ha una dimensione pari a 600×10^3;
- Validation Set con una dimensione corrispondente al 5% della dimensione totale del file *DataSet*. Per n pari a 64 e 128 si ha una dimensione pari 100×10^3, mentre per n pari a 256 si ha una dimensione pari a 200×10^3.

Alla fine ognuno dei precedenti file conterrà: *x*, *xhat* e *y*.

Algorithm 1 Creazione del segnale sparsificato con quota di energia fissata

1: Definire x, $\hat{x}$ e y
2: **for** $iteration = 1, \ldots, q-1$ **do**
3: Selezionare $maxn$ x contigui nel data set di partenza
4: $\xi = D^{\top}x$
5: **for** $j = 1, \ldots, n-1$ **do**
6:

$$\xi_e(j) = \frac{\xi(j)\xi(j)}{\sum_l \xi(l)\xi(l)} \tag{2.8}$$

7: Disporre gli elementi di ξ_e in ordine decrescente
8: Determinare gli indici associati agli elementi riordinati
9: Definire κ
10: Determinare $\hat{\xi}$ conoscendo κ
11: $y(j) = sign(\hat{\xi})$
12: **end for**
13: $\hat{x} = D\hat{\xi}$
14: **end for**

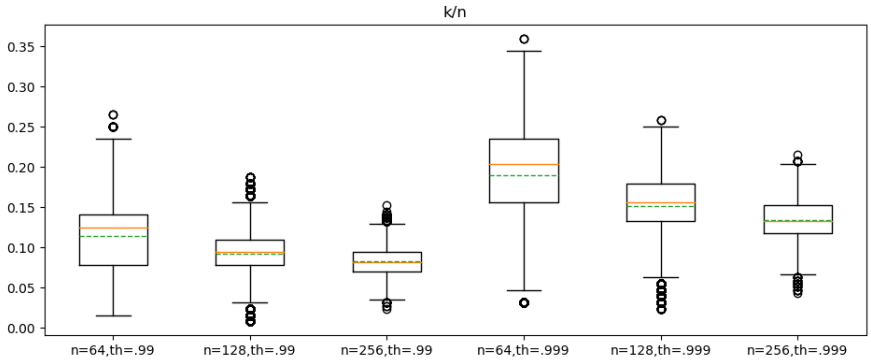


Figura 2.13: Valutazione della relazione che intercorre fra la sparsità κ del segnale e la dimensione dello stesso, indicata con n.

2.2.3 Valutazione della relazione fra sparsità e dimensione del dato

In questa ultima sotto-sezione si è provveduto alla valutazione della sparsità in tutte le casistiche determinate nel precedente algoritmo. Per limitare il numero di operazioni, l'occupazione di RAM e comunque ottenere un risultato significativo in termini statistici si è considerato il solo *ValidationSet*. Per ogni istanza ECG al variare di n si è determinato il valore di sparsità e si è ripetuto su tutte le combinazioni di n e th in esame. Dato che in precedenza non era stato salvato il valore di κ occorre determinarlo nuovamente e per farlo si è definita una funzione apposita che applichi sostanzialmente tutte le operazioni viste in precedenza al dato in ingresso e restituisca solamente il valore di κ ottenuto. Alla fine si mostra il risultato di sparsità normalizzato con n per tutti i casi descrivendo sostanzialmente la percentuale di campioni da mantenere per avere una descrizione con una certa energia fissata a priori.

I risultati relativi al confronto fra la sparsità e la dimensionalità del dato in esame sono presentati in Fig. 2.13 utilizzando una rappresentazione a boxplot.

label	mediana	valore medio	IQR
n=64, th=0.99	8	7.30	4
n=128, th=0.99	12	11.76	4
n=256, th=0.99	21	21.12	6
n=64, th=0.999	13	12.10	5
n=128, th=0.999	20	19.42	6
n=256, th=0.999	34	34.38	9

Tabella 2.1: Mediana, valor medio e range interquartile (IQR) al variare di n dimensione del dato e th energia normalizzata del segnale sparsificato.

Per ogni boxplot sono stati visualizzati:

- 25° e 75° percentile $\longrightarrow$ rettangolo nero
- Mediana $\longrightarrow$ linea arancione continua
- Valore medio $\longrightarrow$ linea tratteggiata verde
- Baffi o wiskers $\longrightarrow$ linea continua nera sotto e sopra al 25° e 75° percentile rispettivamente
- Outlier (elementi oltre $1.5 \cdot IQR$) $\longrightarrow$ cerchi neri

Analizzando i risultati di Fig. 2.13 graficamente normalizzati rispetto al numero n di campioni, fissata la soglia decisionale si osserva come un aumento della dimensione dell'istanza considerata riduca la mediana, il valore medio e il range interquartile (IQR). Il risultato è dovuto al fatto che se si considerano istanze con un n piccolo si potrebbe avere una variabilità di contenuto molto elevata (si potrebbe presentare un'istanza con un QRS completo, parziale o addirittura assente e non è noto a priori come siano distribuiti i complessi QRS

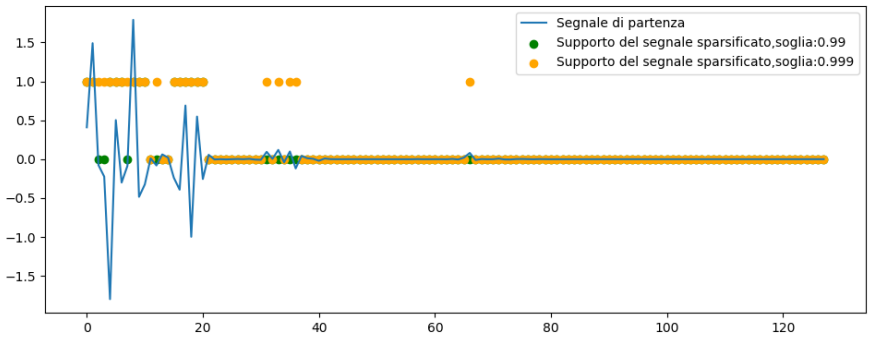


Figura 2.14: Rappresentazione del supporto del segnale sparsificato al variare dell'energia normalizzata del segnale sparsificato.

nelle varie istanze dato che la data augmentation è stata fatta prendendo le componenti in maniera casuale), mentre se si considera un'istanza con n=256 molto probabilmente queste conterranno un picco QRS. Essendo le istanze per n grande molto più simili (statisticamente parlando) fra loro questo riduce la sparsità media. Per un n generico si potrebbe, come accennato sopra, avere istanze con complessi QRS o istanze senza complessi QRS completi. Nel primo caso l'energia normalizzata del segnale sarà maggiore e quindi richiederà una sparsità maggiore (ergo richiederà un numero di campioni κ maggiore per descrivere la quota di energia th fissata), mentre nel secondo caso il valore di energia normalizzata sarà molto più basso del valore medio e quindi la sua sparsità scenderà notevolmente. In entrambi i casi potremmo quindi ottenere degli outlier nella rappresentazione del risultato, ovvero dei valori di κ che non sono significativi perché si discostano eccessivamente dal trend descritto per una fissata configurazione.

Fissata la dimensione, si può anche osservare come varia la sparsità al variare della soglia. Come è giusto aspettarsi, l'aumento della soglia da 0.990 a 0.999 implica un incremento del valore medio (come mostrato in Tab. 2.1)

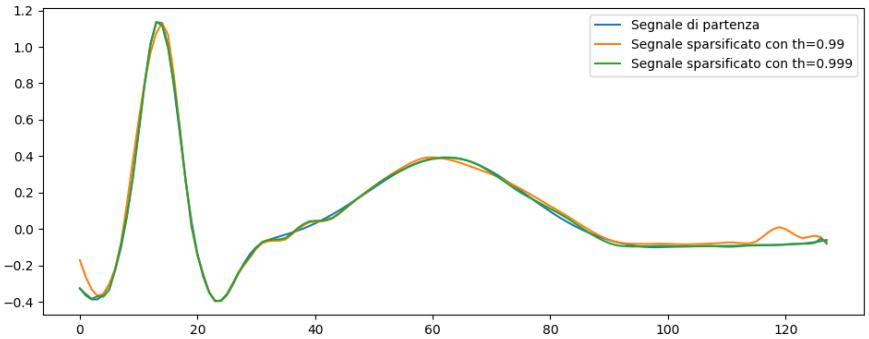


Figura 2.15: Confronto fra la ricostruzione del segnale sparsificato utilizzando lo 0.990 e lo 0.999 dell'energia del segnale non sparsificato.

e provoca un aumento della sparsità del 60% circa (risultato ottenuto come il rapporto fra la sparsità media per *th* pari a 0.99 e 0.999). L'aumento di sparsità può essere analizzato osservando la Fig. 2.14. In questo caso si è ripetuta l'analisi per due valori di soglia mostrando il supporto al segnale sparsificato risultante. Si osserva come nel passaggio da una soglia pari a 0.99 a una pari a 0.999 tutti i campioni impiegati nel primo caso sono utilizzati anche nel secondo (come giustamente ci si aspetterebbe), mentre sono relativamente pochi i campioni non utilizzati nel primo caso che sono aggiunti nel secondo. In questo caso specifico però, si sta trattando un'istanza scelta casualmente fra tutte quelle possibili e quindi si ha, rispetto alle considerazioni fatte per Fig. 2.13, un risultato che si discosta dal trend mostrato in precedenza. Si può poi ricostruire il segnale sparsificato a partire dal suo supporto e ottenere il risultato di Fig. 2.15. Come si può osservare, entrambe le ricostruzioni sono efficaci, nel senso che restituiscono un segnale ricostruito in cui sono riconoscibili correttamente tutti i connotati caratteristici. Le distinzioni messe in mostra sono talmente esigue che si potrebbe prediligere la scelta della soglia a 99% riducendo il numero κ di elementi non nulli considerati.

L'analisi dei risultati ottenuti può essere effettuata anche utilizzando la Tab. 2.1 dove i risultati sono assoluti e non normalizzati. Come ci si aspetta, quando la dimensionalità del problema cresce, aumentano tutti i valori in gioco (mediana, valore medio e IQR). L'aspetto sicuramente più interessante è legato al fatto che prendendo, ad esempio, un segnale descritto da 64 campioni, occorrono, mediamente, 7 campioni nella base sparsa per poter descrivere il 99% di energia del segnale di partenza, mentre ne servono appena 12 per descrivere il 99.9% di energia. Il risultato conferma, ancora una volta, i fondamenti teorici della teoria del Compressed Sensing. A questo si punto si possono esplorare due percorsi diversi:

- Fissare la sparsità κ ad un valore che, fissati n e th, descriva statisticamente tutti gli elementi del dataset (in questo caso riducendo i gradi di libertà del problema);
- Tenere la sparsità κ un grado di libertà del problema e calcolarla istanza per istanza sulla base del segnale considerato.

Nel capitolo successivo, saranno esplorate entrambe le strade e si valuteranno le performance nei due casi.

Capitolo 3

Sistema proposto

In questo capitolo si descrive l'architettura proposta, commentandone le caratteristiche salienti, descrivendo la struttura della DNN e le metodologie utilizzate per la sua definizione, il suo allenamento e i risultati ottenuti.

3.1 Presentazione dell'architettura

L'architettura presentata, dal punto di vista generale, è composta da una componente di compressione, l'encoder, e una di ricostruzione, il decoder, come mostrato in Fig. 3.1. Sfruttando l'architettura citata e il paradigma del Compressed Sensing, di cui si è parlato dal punto di vista teorico nel Cap. 1, l'encoder può essere descritto utilizzando una matrice di sensing $A \in \mathbb{R}^{m \times n}$ la quale implementi un operatore lineare $\mathcal{L}_A : \mathbb{R}^n \mapsto \mathbb{R}^m$, mentre il decoder è implementato da una rete $\mathcal{R}_A : \mathbb{R}^m \mapsto \mathbb{R}^n$ tale per cui idealmente si abbia $x = \mathcal{R}_A(\mathcal{L}_A(x))$, come mostrato in Fig. 3.2. Ovviamente, dato la limitata precisione che può essere messa in gioco durante il processo di encoding e decoding, il segnale che si ottiene alla fine del processo è solo un'approssimazione di quello di partenza e sarà pertanto necessario massimizzare la fedeltà con cui

il segnale è ricostruito [1]. Entrando più nello specifico, si vuole analizzare il comportamento dell'encoder e del decoder.

L'encoder è definito assumendo che i suoi ingressi siano independenti, identicamente distribuiti e gaussiani a valor medio nullo e varianza unitaria. Nel seguito saranno esplorate diverse soluzioni per implementare la matrice di sensing; la prima, più semplice, prevede l'impiego di una matrice antipodale (composta solamente da +1 e -1), riducendo il costo computazionale del sistema dato che i prodotti fra l'ingresso e la matrice si riducono ad un cambio di segno; la seconda soluzione, più generale, prevede l'impiego di una matrice di sensing a valori reali, aumentando il costo computazionale e le performance attese; la terza, prevede l'impiego di un parametro α utilizzato per ottenere un layer antipodale (con performance diverse da quello citato precedentemente).

Il decoder è costituito da una rete neurale profonda che restituisce un vettore di output $o \in [0,1]^n$ utilizzato per descrivere, tramite l'impiego di una soglia di cui si parlerà nel dettaglio successivamente, il supporto stimato $\hat{s}$ secondo la seguente regola: fissata la soglia o_{min},

- $\hat{s}_j = 1$ se $o_j \geq o_{min}$
- $\hat{s}_j = 0$ altrimenti.

Dopo aver determinato tutti gli elementi di $\hat{s}$, attraverso l'applicazione della pseudo-inversa di Moore-Penrose si ricava $\hat{\xi}_{|\hat{s}}$ dalla seguente equazione:

$$\hat{\xi}_{|\hat{s}} = (AS_{|\hat{s}})^{\dagger} y \tag{3.1}$$

dove A è la matrice di sensing che descrive l'encoder, S è la matrice che consente di descrivere il segnale di partenza x in un segnale sparso ξ ($x = S\xi$) e y è il segnale acquisito dal decoder. Nota la stima del supporto $\hat{s}$ e la stima del vettore sparso $\hat{\xi}_{|\hat{s}}$ condizionato dal supporto stimato, si effettuano le operazioni di ricostruzione del segnale, ottenendo $\hat{x}$, e il blocco di autovalutazione.

Figura 3.1: Struttura dell'architettura.

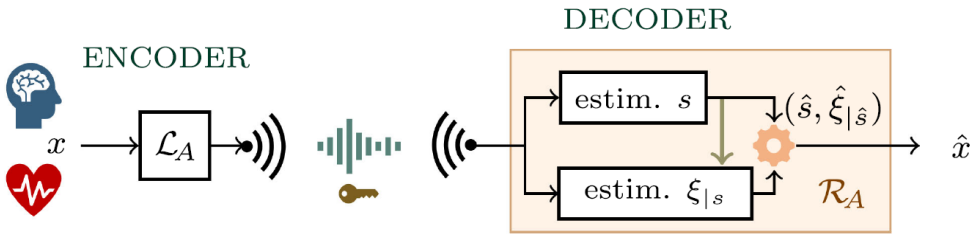


Figura 3.2: Struttura dell'architettura: descrizione semplificata dell'encoder e del decoder.

La prima operazione è semplicemente un prodotto matriciale fra la base ortonormale utilizzata per descrivere x a partire da ξ e il segnale ricostruito $\hat{\xi}_{|\hat{s}}$. L'autovalutazione consiste nel valutare se la differenza fra l'ipotetico ingresso al decoder (se si utilizzasse come segnale in ingresso all'encoder $\hat{x}$) $\hat{y}$ e il vero valore in ingresso al decoder y differiscono di un valore maggiore di $RMNR_{min}$ definendo se il risultato ottenuto nel processo di ricostruzione sia corretto oppure no entro una certa probabilità. Indicando con $\mathcal{N}_C$ il comportamento della DNN si può descrivere in Fig. 3.3 l'architettura proposta, durante la fase di training della rete neurale, nel caso in cui la matrice di sensing sia antipodale indicandola con $A^{\pm}$.

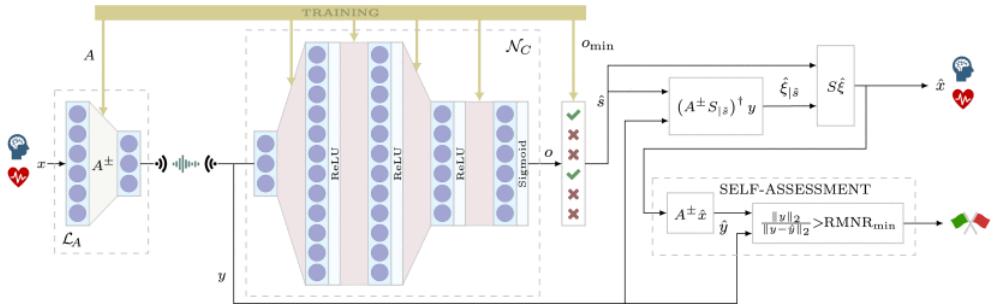


Figura 3.3: Struttura dell'architettura: implementazione algebrica dell'encoder utilizzando una matrice di sensing antipodale e del decoder tramite la DNN, la componente di ricostruzione e di self-assessment.

3.2 Training della Rete Neurale profonda - caso generale

Si vuole, a valle della descrizione dell'architettura, mostrare i passi seguiti per il training della rete neurale, per un caso specifico. Si fissano, pertanto, le dimensioni del segnale di partenza x, ponendo $n = 64$, e il valore di sparsità $\kappa = 16$; si osservi, analizzando i risultati ottenuti in Tab. 2.1 e Fig. 2.13, che scegliere un valore di κ pari a 16 significa fare un'ipotesi peggiorativa su alcune porzioni di segnale ottenendo come vantaggio la riduzione dei gradi di libertà del problema. In altre parole, molte porzioni di segnale prese in esame (sia che siano segnali elettrocardiografici sia che siano segnali elettroencefalografici) avranno un valore di sparsità inferiore (fissata l'energia descritta), ma assumere questo parametro variabile aggiungerebbe un grado di libertà al problema che al momento non si vuole investigare.

Si assume, per generalità, nella descrizione del training della rete neurale profonda del decoder, l'impiego di un layer a valori reali tale per cui il generico elemento della matrice A sia a valori reali; formalmente si pone $A_{j,k} \in \mathbb{R}$. Da Fig. 3.3 emerge come il processo di allenamento del sistema coinvolga anche

la componente di encoder e pertanto occorrerà definire una rete che contenga anche l'encoder.

Nella descrizione della rete neurale da implementare, si parte utilizzando le informazioni descritte e si continua a fare riferimento alla rete di Fig. 3.3 (assumendo la matrice di sensing a valori reali) fino ad ora per ottenere una struttura descritta nel Listato 3.1 e composta dalle seguenti componenti:

- Layer d'ingresso alla DNN, denominato `main_input` e suddiviso in cluster di n campioni ciascuno;
- Matrice di sensing A a valori reali, denominata `A`, descritta nel modello da un layer denso a m neuroni, una funzione di attivazione lineare e senza l'impiego di bias. La scelta della matrice di sensing (antipodale classica, a valori reali o antipodale con parametro di scaling) inciderà successivamente sulla descrizione di questo layer, rendendo in alcuni casi impossibile l'impiego del layer `Dense`;
- Tre layer profondi a valori reali, denominati `H0`, `H1` e `H2`, descritti nel modello da tre layer densi di dimensioni rispettivamente $2n$, $2n$ e n (dove n indica la dimensione del segnale di partenza x), con una funzione di attivazione ReLU e con l'impiego di bias;
- Un layer di output a valori reali, denominato `Out`, descritto nel modello da un layer denso di dimensione n, con funzione di attivazione Sigmoid e con l'impiego di bias.

```
# Modello DNN principale
    input_layer = Input(shape=(n,),name='main_input')
    model = Model(input_layer, dnnFloat(input_layer, n, m))

# Descrizione dell'architettura
```

```
6  def dnnFloat(input_0, n, m):
7      ### CS encoder
8      y_0 = Dense(units=m, activation='linear', use_bias=False, name='A')(
       input_0)
9      ### Level 0
10     out_0 = Dense(units=2*n, activation='relu', name='H0')(y_0)
11     ### Level 1
12     out_1 = Dense(units=2*n, activation='relu', name='H1')(out_0)
13     ### Level 2
14     out_2 = Dense(units=n, activation='relu', name='H2')(out_1)
15     ### Final output
16     out_nn = Dense(units=n, activation='sigmoid', name='Out')(out_2)
17     return out_nn
```

Listato 3.1: Modello della rete neurale, definito tramite una funzione ad hoc.

Si rimanda all' 3.6 per maggiori dettagli sulla sintassi relativa alla struttura della rete.

Durante il training si valuta la funzione di loss per determinare l'evoluzione del training della rete. Dal punto di vista teorico, una funzione di loss mappa gli eventi di una o più variabili in un numero reale che rappresenta il costo associato all'evento. Si vuole minimizzare la funzione di loss per poter raggiungere la più alta probabilità di corretta ricostruzione a valle della DNN sapendo che il cuore del training è l'algoritmo di backpropagation, il quale modifica il valore dei pesi e dei bias per minimizzare la funzione di costo.

Nota l'architettura della rete neurale, si può passare alla descrizione del processo di training vero e proprio. Per farlo, a partire da un Data Set in ingresso, si definiscono un Training Set (d'ora in poi definito dalla coppia [`X_train`, `Y_train`]) e un Test Set (d'ora in poi definito dalla coppia [`X_test`, `Y_test`]) sia per l'ingresso che per l'uscita della rete. Durante la fase di training si fornisce alla rete la coppia [`X_train`, `Y_train`] dove `X_train` definisce gli

ingressi della DNN mentre `Y_train` definisce le uscite attese; così facendo, l'algoritmo di backpropagation è in grado di minimizzare l'errore modificando il valore dei pesi e dei bias dei neuroni in relazione al valore della derivata della funzione costo. Durante la fase di test si fornisce alla DNN solamente `X_test` e l'uscita della rete si confronta con il valore atteso `Y_test` per determinare la affidabilità del modello.

Dopo aver descritto il modello della rete e la costruzione di un Training Set e di un Test Set, occorre definire gli strumenti necessari alla realizzazione del training. Per prima cosa occorre scegliere, qualora si abbia la disponibilità di scelta, se effettuare il training su una piattaforma single-core o multi-core. Le dimensioni della rete e il numero di epoche che si impiegano, nel caso in esame, per completare il training suggeriscono un training su una piattaforma multi-core come le Graphics Processing Unit (GPU). É quindi necessario provvedere alla scelta della GPU da impiegare e alla sua configurazione. Durante la fase di training si possono identificare due fasi ben distinte:

- Nella prima epoca si crea il modello della rete neurale, descritto nel Listato 3.1 e lo si allena. Al termine dell'epoca il modello è salvato in memoria e si prosegue alla seconda epoca;
- In tutte le altre epoche il modello è letto dalla memoria, allenato e poi risalvato in memoria.

A valle della lettura del modello si procede alla sua configurazione per il training dei pesi. Durante la fase di configurazione occorre definire i seguenti parametri:

- `optimizer`: il parametro consente di definire l'ottimizzatore da utilizzare per il training e nel caso specifico si è utilizzato il metodo della discesa stocastica del gradiente con momento nullo. Durante la fase di training i

pesi dei neuroni sono aggiornati utilizzando il learning rate e il gradiente calcolato precedentemente;

- `loss`: il parametro definisce la scelta della funzione di loss (o di costo) da utilizzare durante la fase di training. In questo caso si è scelto di utilizzare una funzione di loss custom `multiclass_loss_alpha`;

- `metrics`: l'ultimo parametro utilizzato è definito per specificare quali strumenti debbano essere utilizzati per definire le prestazioni della rete. Anche in questo caso sono stati utilizzati strumenti customizzati quali `sparse_accuracy`, `sparse_TP` e `sparse_TN`.

Infine si procede al training vero e proprio del modello utilizzando il metodo `model.fit()`. Anche in questo caso occorre definire una serie di parametri utili al corretto funzionamento del training. In particolare:

- `X_train, Y_train`: costituiscono il Training Set (il data set utilizzato per la parte di apprendimento della DNN);

- `epochs`: definisce il numero di epoche, ovvero il numero di volte in cui il processo di training del modello deve essere ripetuto prima di poter giungere ad un risultato finale;

- `batch_size`: definisce il numero di campioni che sono dati in ingresso alla DNN durante il training e tipicamente sono meno del numero di campioni che costituiscono il training set. Così facendo, il training set è suddiviso in cluster con una certa dimensionalità `batch_size` e ad ogni epoca sono tutti assegnati alla DNN;

- `validation_data`: definisce quali coppie ingresso-uscita della reta debbano essere impiegate per validare il processo di training. Per la fase

di test sono utilizzate coppie di dati che la rete non mai analizzato per prima per capire come questa risponda ad uno stimolo in ingresso ignoto;

- `callbacks`: definisce la possibilità di personalizzare il modello che si sta utilizzando descrivendo il comportamento della rete. Si è scelto per il caso specifico di adottare una serie di callbacks customizzate al fine di massimizzare i risultati per il problema in esame.

La singola epoca del training termina salvando il modello e svuotando la RAM della GPU dal modello appena allenato prima di rileggerlo nella successiva epoca. É opportuno decrivere più accuratamente l'impiego degli strumenti custom nell'analisi della rete. Si parte descrivendo la funzione di loss ed in particolare si è scelto di non utilizzare una delle funzioni built-in di tensorflow, ma piuttosto di crearne una nuova descritta dalla seguente equazione:

$$L(W,b) = media\{(1-\alpha)\cdot(1-y)\cdot log(1-\hat{y}) + \alpha\cdot y\cdot log\hat{y}\} \tag{3.2}$$

dove α è un parametro utilizzato per discriminare i valori pari a 0 e pari ad 1 nel supporto (porre $\alpha = 0.5$ significa semplicemente effettuare un'approssimazione negli elementi del supporto), y definisce il segnale atteso in uscita alla rete neurale, mentre $\hat{y}$ definisce il segnale ottenuto a valle dell'impiego della DNN. L'aspetto interessante è che in questo caso è possibile sbilanciare la scelta degli uni nel supporto del segnale in uscita semplicemente modificando il parametro α e osservando come questo modifica la prestazione della rete. Nella fase di configurazione della DNN sono state impiegate anche delle metriche custom quali `sparse_TP`, `sparse_TN` e `sparse_accuracy`. Le prime due metriche definiscono i risultati veri positivi (TP) e veri negativi (TN), rispettivamente. Nel primo caso si descrive il corretto riconoscimento di un uno nel supporto, mentre nel secondo caso si descrive il corretto riconoscimento di uno zero nel supporto del segnale considerato. Ovviamente, più TP e TN

sono elevati, più il training sta procedendo correttamente e la DNN è in grado di riconoscere efficacemente le corrette componenti del supporto. Conoscendo poi TP e TN, si possono calcolare le componenti di errore (i falsi positivi e i falsi negativi, rispettivamente FP e FN) per poter descrivere l'ultima metrica custom introdotta: la `sparse_accuracy`, definita come:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Fra le componenti custom utilizzate nel training della rete in questione rimangono da analizzare le callbacks; in particolare si è scelto di unire una serie di callbacks utili a controllare il training in una lista. Sono state utilizzate le seguenti callbacks:

- `TerminateOnNaN()`: consente di interrompere il training quando la funzione di loss assume un valore NaN (Not a Number);

- `ReduceLROnPlateau()`: consente di ridurre il passo di discesa del gradiente quando le metriche che si stanno valutando non migliorano più. In particolare, il learning rate è inizializzato ad un valore pari a 0.01 e quando per un fissato numero di epoche la metrica da monitorare non migliora, si riduce il learning rate di un certo fattore e si ottiene un nuovo learning rate. Formalmente:

  $$\text{nuovo learning rate} = \text{factor} \cdot \text{vecchio learning rate} \tag{3.4}$$

  Il processo si itera (nel caso in cui la metrica monitorata non migliori fino ad un valore minimo di learning rate). Sono pertanto, qualora si utilizzi tale callback, da fissare tutti i parametri descritti per ottenere un'evoluzione delle metriche adeguata;

- `ModelCheckpoint()`: consente di salvare il modello in memoria con una certa cadenza. Fissando i parametri si definisce il percorso del file ed altri parametri necessari al corretto funzionamento;

- `EarlyStopping()`: consente di fermare l'allenamento della DNN quando una metrica (sotto osservazione) non migliora ulteriormente trascorse un certo numero di epoche definibili da un parametro della callback;

- `TensorBoard()`: consente di scrivere un file che mostri durante l'allenamento, al termine di ogni epoca, l'evoluzione delle metriche che si stanno analizzando.

Terminata la presentazione degli elementi necessari all'allenamento della DNN, si procede al training vero e proprio. Sono pertanto fissati i seguenti vincoli:

- Sul segnale si pone $n = 64$, m variabile (è un grado di libertà del training per esplorare come il Compression Ratio incida sulla performance finale), $\kappa = 16$ e $\alpha = 0.5$;

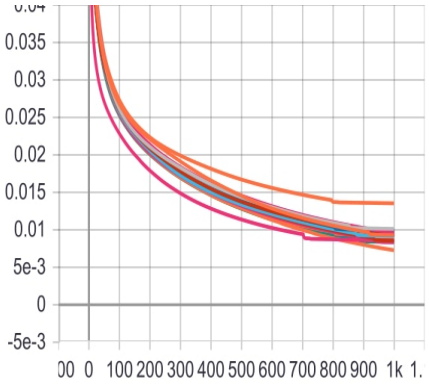
- Sul training si pone $epoch = 1000$, $batch\_size = 30$ e si sceglie l'impiego di una delle due GPU disponibili;

- Sulle callbacks si pone come variabile da valutare la performance di loss, il fattore per la modifica del learning rate pari a 0.2 e un'attesa di 20 epoche prima di modificare il learning rate.

3.2.1 Risultati del training

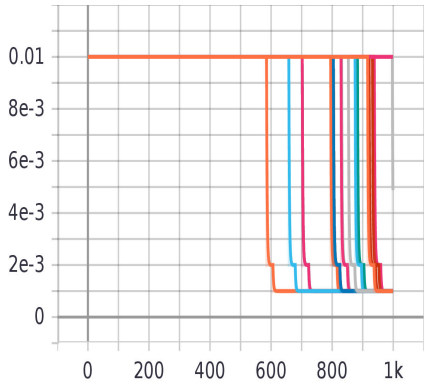
Si possono a questo punto valutare i risultati del training dal punto di vista dell'evoluzione della funzione di loss, del learning rate e dell'accuracy, al variare di m dimensione del dato a valle dell'encoder (utilizzo della matrice di sensing `A` a valori reali) come mostrato in Fig. 3.4 e riassunto in Tab. 3.1.

Analizzando la funzione di loss si osserva un trend discendente della performance, in accordo con la teoria descritta in precedenza. Ad ogni epoca l'algoritmo di retro-propagazione dell'errore modifica opportunamente i pesi e

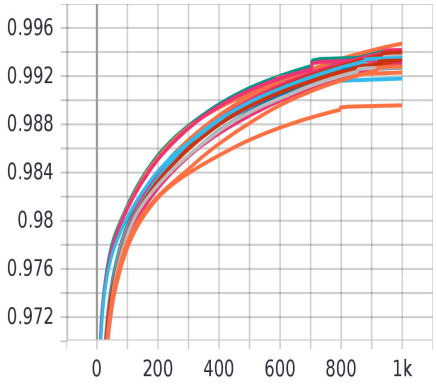


5

(a) Evoluzione della funzione di loss `multiclass_loss_alpha` durante la fase di training allo scorrere delle epoche.



(b) Evoluzione del parametro di learning rate durante la fase di training allo scorrere delle epoche.



(c) Evoluzione della metrica `sparse_accuracy` durante la fase di training allo scorrere delle epoche.

Figura 3.4: Risultati della fase di training di una rete neurale profonda con neuroni del primo livello a valori reali

m	max accuracy	min loss	learning rate
8	0.9896	0.01354	10^{-3} @ 836 epoch
12	0.9928	0.00928	10^{-3} @ 957 epoch
16	0.9931	0.00959	10^{-2} @ 999 epoch
20	0.9927	0.00989	10^{-3} @ 846 epoch
24	0.9940	0.00857	10^{-3} @ 973 epoch
28	0.9930	0.00973	10^{-3} @ 845 epoch
32	0.9934	0.00928	10^{-3} @ 845 epoch
36	0.9941	0.00837	10^{-3} @ 925 epoch
40	0.9936	0.00894	10^{-3} @ 872 epoch

Tabella 3.1: Risultati al variare di m, in termini di massima accuratezza, minimo valore della funzione di loss e epoca nella quale il learning rate.

la funzione di loss riduce i suoi valori. Per quanto noto dalla teoria presentata nel Cap. 1, la funzione di loss deve essere minimizzata e quando accade si è in presenza di un minimo locale o di un minimo globale della funzione. Appare chiaro, quindi, che all'avanzare delle epoche, se il training procede nella corretta direzione, la funzione di loss continua a diminuire, ma la riduzione rispetto all'epoca precedente è sempre meno marcata, poiché il gradiente calcolato dalla funzione di ottimizzazione (durante la fase di training si è imposto l'impiego di un ottimizzatore tipo discesa stocastica del gradiente) diventa sempre più piccolo avvicinandosi al minimo in questione (locale o globale che sia). Per ogni valore di m si associa una curva del grafico di Fig. 3.4a e nel momento in cui la rete non è più in grado di migliorare le proprie performance, tramite l'impiego della callback `ReduceLROnPlateau()` si riduce progressivamente il learning rate partendo da 0.01 ed utilizzando l'eq. 3.4 fino ad arrivare al mini-

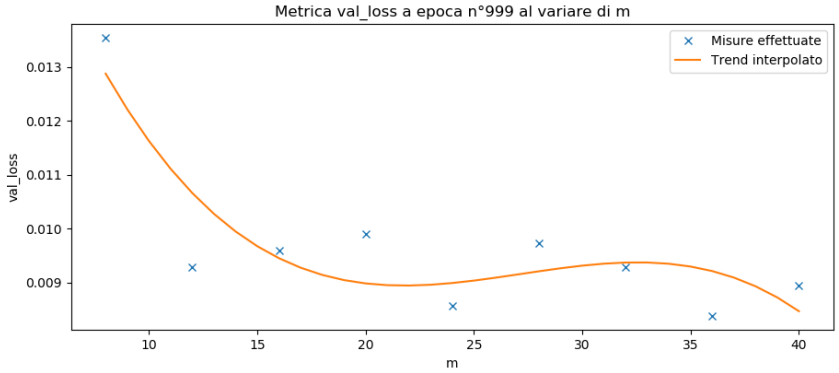


Figura 3.5: Valore della metrica `val_loss` alla fine del training (epoca n° 999) della rete neurale con primo layer a valori reali. Interpolazione `cubic spline` delle misure raccolte.

mo valore che il learning rate possa assumere, fissato a 0.001. Giunti al minimo valore assumibile, interviene una seconda callback `EarlyStopping()` che ferma il training della rete. Una volta terminato, si ripete lo stesso training variando il valore di m, esplorando così la performance al variare del Compression Ratio (CR) descritto nella Sez. 1.1. Analizzando poi la curva di Fig. 3.5 si osserva il trend ottenuto rappresentando i risultati al variare di m. Eccetto per il caso m=8, la cui funzione di loss raggiunge un valore nettamente maggiore rispetto agli altri casi, la metrica di loss si mantiene intorno a $9{\cdot}10^{-3}$ e si può dedurre, interpolando la curva, che qualsiasi altro valore di m considerato possa cadere nell'intorno di questo valore della funzione di loss con una certa approssimazione. Analizzando Fig. 3.4b circa la rappresentazione del parametro di learning rate al variare delle epoche si osserva che esistono valori di m per cui alla fine del training il passo di scorrimento è rimasto invariato a 0.01, mentre per la maggior parte dei casi in esame si ha il raggiungimento del minimo valore del parametro di learning rate e l'interruzione del training tramite l'impiego della callback (per le scelte fatte durante la fase di training si impone che passino

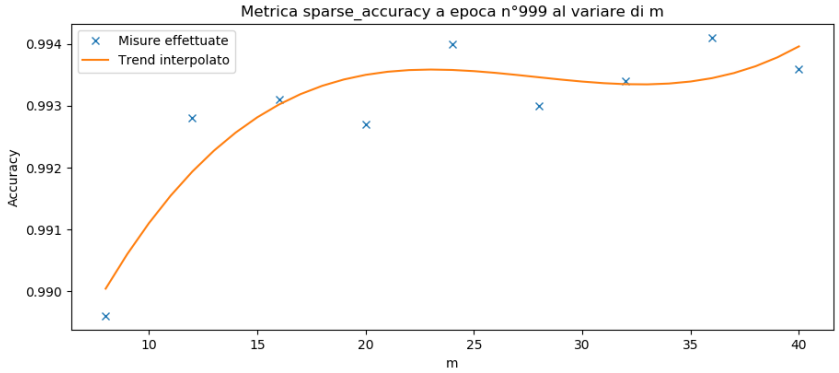


Figura 3.6: Valore della metrica `sparse_accuracy` alla fine del training (epoca n° 999) della rete neurale con primo layer a valori reali. Interpolazione `cubic spline` delle misure raccolte.

20 epoche senza miglioramento prima di ridurre il parametro di learning rate tramite l'apposita callback e dopo aver raggiunto il minimo valore del parametro di learning rate trascorrono altre 20 epoche prima che si attivi la callback per l'interruzione prematura del training della rete neurale). Per i casi in cui il parametro di learning rate non è stato ridotto potrebbe essere interessante proseguire il training per capire fino a che punto le metriche analizzate possano essere migliorate e se il costo, in termini computazionali e temporali, è sostenibile a fronte di un miglioramento delle metriche in esame non noto a priori. Si può, infine, analizzare la Fig. 3.4c relativa alla metrica `sparse_accuracy` allo scorrere delle epoche e al variare di m. Si osserva, in maniera duale rispetto alla funzione di loss, un incremento molto marcato nelle prime centinaia di epoche della performance, mentre nella seconda metà (ultime 500 epoche) il miglioramento è sempre meno marcato o addirittura assente. Il trend descritto è ciò che ci si aspetta nel momento in cui si valutano i risultati attesi di una rete neurale come quella presentata poiché nelle prime centinaia di epoche la discesa stocastica del gradiente consente di percorrere grandi step ad ogni

epoca (riducendo molto la funzione di loss e aumentando significativamente la metrica di accuracy), mentre successivamente gli step diventano sempre più piccoli o nulli (se l'algoritmo di retro-propagazione dell'errore non è in grado di migliorare ulteriormente i parametri del modello allenato). Nel primo caso la metrica di *sparse_accuracy* migliora, ma più lentamente, mentre nel secondo caso non si riscontra un aumento della metrica analizzata perchè il training è terminato. Dopo aver analizzato il trend al variare delle epoche, si può analizzare il valore di `sparse_accuracy` raggiunto alla fine del training al variare di m. Il risultato, rappresentato in Fig. 3.6, mostra in maniera duale il trend analizzato per la funzione di loss; in particolare, per $m = 8$ si ha una performance in termini di accuratezza nella ricerca del supporto inferiore rispetto agli altri valori di m, per i quali si ha un trend complessivamente positivo con valori compresi fra 99.3% e 99.4%. Anche in questo caso si può dedurre che, con una certa precisione, considerati altri valori di m tali da interpolare la presente curva, essi si allineeranno con il presente trend raggiungendo performance in termini di `sparse_accuracy` molto buone. Analizzando i dati ottenuti dal training e riportati, solo per alcuni valori di m in Tab. 3.1, si osservano per $m = 8$ e $m = 20$ i valori minimi di massima accuratezza. A questi risultati sono anche associati il massimo valore della funzione di loss e le prime riduzioni del fattore di learning rate. Rispetto alle distribuzioni mostrate in Fig. 3.4 si possono osservare due curve che si discostano dal trend poi analizzato in Fig. 3.5 e 3.6. La prima, per m pari a 8 è già stata commentata, mentre la seconda è per m pari a 34 dove si osserva il massimo valore di `sparse_accuracy`, pari a 99.47%, e il minimo valore della funzione di loss, pari a $7.28{\cdot}10^{-3}$; inoltre, per questo valore di m il parametro di learning rate non è stato ridotto fino al minimo valore, quindi la fase di training può ancora migliorare le performance per questa configurazione della rete neurale. Concludendo, il trend descritto

dai risultati mostrati è in linea con quanto ci si aspetta dal punto di vista teorico dato che al diminuire di m aumenta il Compression Ratio, come si osserva dalla definizione di CR. All'aumentare del CR si riduce la quantità di informazione inviata poiché la compressione riduce la dimensionalità del dato a valle dell'encoder e pertanto si degrada progressivamente il risultato finale. Ricordando che si è fissato n pari a 64, si osserva anche come ci sia un netto distacco fra la curva per m pari a 8 (CR=8) e gli altri risultati, segno che fino a m pari a 10 la performance è comunque accettabile e la quantità di informazione persa nella compressione eseguita dall'encoder non altera in maniera eccessiva il dato da ricostruire. Se, da un lato, prendere un valore di CR molto elevato consente di ridurre la quantità di informazione da inviare a discapito di risultati meno performanti, scegliere un CR molto piccolo aumenta la mole dei dati da gestire, ma consente di raggiungere prestazioni migliori. Si rende pertanto necessario scegliere un CR in funzione del problema da analizzare, valutando opportunamente vantaggi e svantaggi della scelta, eventualmente considerando trade-off fra le soluzioni limite proposte.

3.3 Analisi delle prestazioni al variare della matrice di sensing

Si vuole, dopo aver mostrato il training della DNN, utilizzarla per ricostruire il segnale valutando alcune metriche di seguito descritte al variare della matrice di encoding. Saranno considerate tre soluzioni distinte, come accennato in precedenza:

- Matrice di sensing antipodale;
- Matrice di sensing a valori reali;

- Matrice di sensing antipodale con parametro di scaling α.

Facendo riferimento alla struttura della DNN di Fig. 3.3, la possibilità di raggiungere una fedeltà nella ricostruzione accettabile, è vincolata alla scelta della soglia o_{min} utile a definire il supporto stimato $\hat{s}$. Si rende pertanto necessario definire delle metriche che consentano di raggiungere la miglior ricostruzione possibile, scegliendo accuratamente o_{min} e partendo dal presupposto che la compressione applicata all'encoder è un'operazione che provoca la perdita di informazione. Nel seguito saranno pertanto utilizzate le seguenti metriche:

- Average Reconstruction Signal-to-Noise Ratio (ARSNR):

$$ARSNR = \frac{1}{T}\sum_{t=0}^{T-1} RSNR^{(t)} \tag{3.5}$$

  dove il Reconstruction Signal-to-Noise Ratio (RSNR) si definisce come:

$$RSNR = \frac{\| x \|_2}{\| x - \hat{x} \|_2} dB \tag{3.6}$$

- Probability of Correct Reconstruction (PCR):

$$PCR(RSNR_{min}) = \frac{1}{T}\#\{t | RSNR^{(t)} \geq RSNR_{min}\} \tag{3.7}$$

  Denotando con # il numero di elementi nel set di dimensione T che soddisfano la condizione fra parentesi.

Il RSNR definisce la distanza, in senso geometrico, fra il segnale vero e quello ricostruito, calcolata in decibel (dB), l'ARSNR descrive una media dei valori di RSNR calcolati su un insieme T di istanze del segnale, mentre la PCR è utilizzata per determinare la probabilità che all'interno del set di dimensione T ci siano le ricostruzioni corrette del segnale.

Dopo aver definito le metriche necessarie alla valutazione dei risultati occorre procedere al calcolo della soglia o_{min} da applicare all'uscita della DNN

per definire il supporto stimato $\hat{s}$ sapendo che la DNN include nella sua struttura il CS encoder. Per la stima della soglia in esame occorre disporre del segnale in ingresso x e della relativa matrice di sparsità D, del modello della rete neurale ottenuto a valle di un training dello stesso e della figura di merito che deve essere ottimizzata. Sono poi parametri da utilizzare durante la ricerca della soglia anche il livello di rumore gaussiano bianco aggiunto al segnale, i punti iniziali dell'algoritmo e il numero di iterazioni da utilizzare. L'algoritmo partendo dal segnale x fornito in ingresso all'encoder, aggiunge un valore prefissato di rumore ν tale da soddisfare un vincolo sul rapporto segnale rumore (ottenendo un segnale rumoroso $x + \nu$ in ingresso), definisce l'uscita dell'encoder $y = \mathcal{L}_A(x + \nu) = A(x + \nu)$ e attraverso l'utilizzo del modello della DNN si genera la previsione dell'uscita della rete noto l'ingresso y applicato come:

$$o = \mathcal{N}_C(y) = (\mathcal{N}_C \circ \mathcal{L}_A)(x + \nu), \text{ con } \mathcal{N}_C : \mathbb{R}^m \longmapsto [0, 1]^n \tag{3.8}$$

dove $\mathcal{L}_A$ descrive la matrice di sensing $A^{\pm} \in [-1, +1]^{m \times n}$ se la matrice di encoding è antipodale, mentre nel caso reale vale $A \in \mathbb{R}^{m \times n}$. Occorre ora determinare il valore della soglia da applicare all'uscita della rete neurale per definire il supporto stimato $\hat{s}$. Per trovare tale costante si passa attraverso una serie di step, eseguiti per un numero fissato di volte, di seguito descritti:

1. Creazione di un vettore a scala logaritmica di soglie con un numero di elementi arbitrario e valori compresi fra 10^{-9} e 1. Terminata l'esecuzione si passa a 3;

2. Creazione di un vettore a scala lineare di soglie con estremi determinati runtime e un numero di elementi arbitrario (uguale a quello impiegato allo step 1). Terminata l'esecuzione si passa a 3;

3. Si stima il supporto $\hat{s}$ tramite una funzione definita ad hoc, si calcola il vettore di RSNR noti x, D e $\hat{s}$ al variare della dimensione di x e si valuta il valore della metrica da analizzare potendo scegliere fra due possibili: ARSNR e PCR. Nel primo caso occorrerà determinare il valore medio del vettore di RSNR ottenuto precedentemente, mentre nel secondo caso si dovrà determinare quanti elementi del vettore di RSNR raggiungono il minimo valore di RSNR ritenuto accettabile e si normalizza al numero di elementi del vettore per descrivere una probabilità. L'algoritmo termina poi con la ricerca del massimo valore di soglia per l'iterazione corrente e del massimo valore della metrica analizzata. Se le iterazioni sono terminate si prosegue nell'algoritmo, altrimenti si esegue 2.

Si osservi come nella ricerca della soglia ottima o_{min}, il calcolo del supporto step-by-step risulti fondamentale per poter massimizzare le metriche desiderate. Il risultato ottenuto dipende dalla quota di rumore introdotta nel segnale e dal numero di iterazioni che si applicano all'algoritmo. Nel primo caso si può quindi investigare l'effetto del rumore sulla ricostruzione del supporto; nel secondo caso una soglia più precisa implica migliori performance nella ricostruzione finale, ma richiede un costo computazionale più elevato, mentre una soglia meno precisa peggiora le performance sulla ricostruzione, ma riduce il costo computazionale rendendo necessaria la ricerca di un trade-off valido fra gli estremi proposti.

Dopo aver dimensionato correttamente il supporto si può procedere alla realizzazione della decodifica vera e propria. Anche in questo caso, come per la ricerca della soglia ottima con cui definire il supporto, si è definita una funzione ad hoc. Per poter ricostruire il segnale devono essere noti il segnale di partenza x e la sua matrice di sparsità D, il modello della DNN comprensivo del CS encoder $(\mathcal{N}_C, \mathcal{L}_A)$, la soglia ottima o_{min} con cui definire il supporto a

valle della predizione, il livello di rumore gaussiano bianco aggiunto al segnale e la soglia utilizzata per definire il supporto $\hat{s}$ a valle della sua stima. Sapendo che il modello di DNN impiegato si compone di un layer di encoder e della DNN vera e propria, occorre, per poter ricostruire il segnale, per prima cosa predirre il vettore o, uscita della DNN, utilizzato per definire il supporto stimato $\hat{s}$. Per raggiungere il risultato, si applica un rumore gaussiano bianco ν al segnale x in ingresso tale che la componente di segnale rispetto a quella di rumore soddisfi il vincolo fornito; dopodiché utilizzando l'eq. 3.8 si determina l'uscita della DNN. Nota l'uscita della DNN e conoscendo la soglia o_{min}, determinata precedentemente, da utilizzare per definire il supporto, si costruisce $\hat{s}$ come segue:

$$\hat{s}(i) = \begin{cases} 1, & \text{se } o(i) > o_{min} \\ 0, & \text{se } o(i) < o_{min} \end{cases}, \ \forall i \in [0, n] \tag{3.9}$$

Terminata la fase di predizione, occorre descrivere la fase di valutazione per giungere alla ricostruzione vera e propria. Per prima cosa, si stima la rappresentazione sparsa del segnale di partenza ξ conoscendo l'uscita del CS encoder, il supporto stimato $\hat{s}$ a valle della predizione e il prodotto fra la matrice di encoding e la base di sparsità $B{=}AD$. Noto ξ si ricostruisce il segnale di partenza $\hat{x}$ (che differirà da quello originale x a causa del rumore e della non idealità dell'oracolo), si definisce, conoscendo la matrice di encoder A e la rappresentazione sparsa del segnale di partenza ξ, l'uscita del CS encoder se fosse posto in ingresso all'encoder $\hat{x}$ invece di x e il supporto di ξ conoscendo la soglia che rende ottima la definizione del supporto. Si è quindi implementata la componente di self-assessment per valutare la corretta ricostruzione del segnale in maniera automatizzata. Occorre, infine, determinare il RSNR per valutare la correttezza nella ricostruzione del segnale di partenza e dato l'impiego della componente di self-assessment si introduce una terza metrica: il

Reconstruction Measurements-to-Noise Ratio (RMNR) definito come:

$$RMNR = \frac{\|y\|_2}{\|y - \hat{y}\|_2} \tag{3.10}$$

Il RMNR è molto importante perchè l'approccio di decoding proposto si basa sull'idea che il supporto sia stimato separatamente dalla rappresentazione sparsa del segnale di partenza ξ e quindi teoricamente è possibile ricostruire il segnale. Se nella fase di predizione l'oracolo completasse con successo la predizione, in assenza di rumore, si otterrebbe un supporto stimato $\hat{s} = s$, si avrebbe $y = AD_{|s}\xi_{|s}$ e allora $y \in span(AD_{|\hat{s}})$. Implica:

- $\hat{\xi}_{|\hat{s}} = \xi_{|s}$;
- Se $\hat{\xi}$ è sparso e attraverso un'opportuna base descrive x, allora $y = AD\hat{\xi}$.

Invece, se l'oracolo fallisce quanto appena detto non è più valido poiché $\hat{s} \neq s$ e poiché ξ è l'unica soluzione κ-sparsa del sistema $y = AD\xi$ non è possibile che $y \in span(AD_{|\hat{s}})$. In tal caso:

- $\hat{\xi}_{|\hat{s}} \neq \xi_{|s}$;
- Se $\hat{\xi}$ è sparso e attraverso un'opportuna base descrive x, allora $y \neq AD\hat{\xi}$.

Ora conoscendo $\hat{x}$ si può calcolare $\hat{y} = A\hat{x}$ e misurando la distanza fra il valore in ingresso al decoder e il valore ricalcolato a valle della predizione si definisce il RMNR [1]. Ne consegue che il risultato sia accettabile solamente quando il valore di RMNR trovato supera una certa soglia definita opportunamente e valutata sulla base dell'affidabilità che si deve raggiungere con il decoder in esame.

Alla fine dell'algoritmo si ottiene il valore di RSNR, RMNR, il supporto stimato $\hat{s}$ e la predizione della DNN o.

Per l'analisi delle performance sono fissati:

- La dimensione del dato di partenza, $n = 64$;
- Il valore di sparsità, $\kappa = 16$;
- Il parametro utilizzato per discriminare gli 0 e gli 1 nel supporto, $\alpha = 0.5$;
- Il modello della rete neurale utilizzata.

I risultati saranno valutati esplorando tre gradi di libertà: la tipologia della matrice di sensing, A, la dimensione del dato a valle della matrice di encoding, m, e la quota di rumore introdotta nella ricostruzione, valutando il caso noiseless e il caso con un valore fissato di ISNR. Per la ricerca dei risultati occorre fissare i parametri sopra definiti e procedere alla valutazione della soglia ottima massimizzando una delle metriche proposte al variare della dimensione del dato a valle dell'encoder m. Dopo aver determinato tale soglia si effettua la decodifica del segnale e si descrivono i risultati ottenuti al variare della matrice di encoding, del rumore introdotto e del livello di CR. Si effettuerà per prima cosa un confronto fra i risultati ottenuti con la matrice antipodale e con la matrice a valori reali, per poi introdurre successivamente la matrice antipodale con il parametro di scaling.

3.3.1 Risultati ottenuti

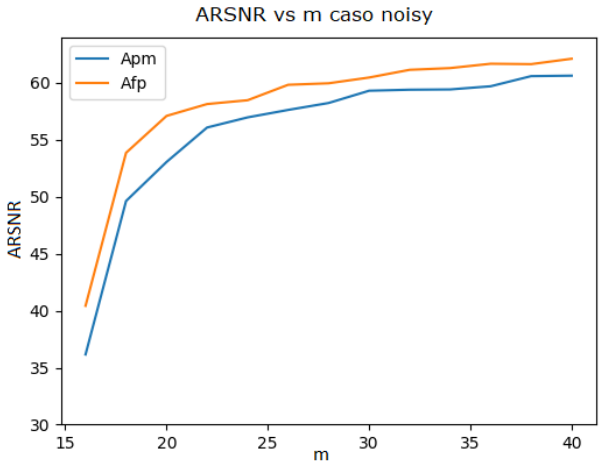
Si mostrano i risultati relativi alle metriche ARSNR e PCR-5dB al variare della matrice di encoding e del Compression Ratio (CR) ricordando che per l'analisi in esame si è fissato n pari a 64 e che il CR è definito come il rapporto n/m. L'analisi è stata condotta per valori di CR compresi fra 1.6 ($m = 40$) e 4 ($m = 16$), ma i risultati raccolti in Tab. 3.2 contengono solo alcune delle casistiche considerate. Analizzando i risultati di Tab. 3.2 si osserva un trend positivo di tutte le performance al diminuire del CR, inizialmente molto

			ARSNR		PCR-5dB	
		ISNR	60.0	∞	60.0	∞
α	CR	enc				
0.5	4.0	Apm	36.16	189.88	0.03	0.64
		Afp	40.43	204.76	0.15	0.69
	3.2	Apm	53.04	282.53	0.47	0.96
		Afp	57.10	283.42	0.78	0.97
	2.7	Apm	56.97	292.38	0.83	0.99
		Afp	58.48	290.85	0.86	0.99
	2.3	Apm	58.24	293.75	0.94	0.99
		Afp	59.97	291.86	0.95	0.99
	2.0	Apm	59.40	293.75	0.97	0.99
		Afp	61.15	293.32	0.98	0.99
	1.8	Apm	59.71	294.51	0.98	0.99
		Afp	61.68	294.16	0.99	0.99
	1.6	Apm	60.64	294.53	0.98	0.99
		Afp	62.12	294.51	0.99	0.99

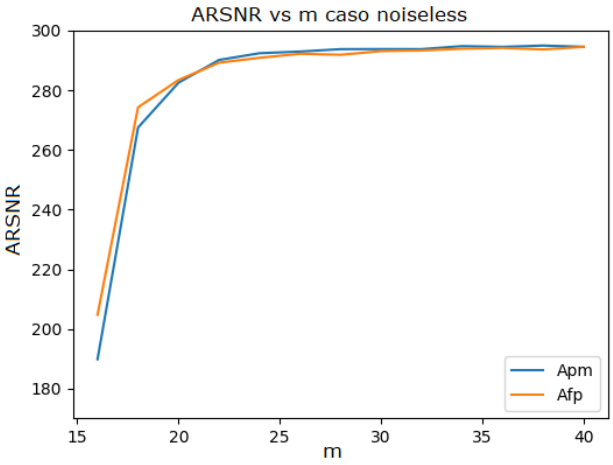
Tabella 3.2: Risultati ARSNR e PCR-5dB al variare del Compression Ratio e della matrice di encoding.

marcato (per valori di CR grandi) e successivamente più contenuto (per piccoli valori di CR). Analizzando Fig. 3.7a si osserva che l'ARSNR, nel caso in cui ci sia una componente di rumore tale da avere un SNR pari a 60 dB, valutato nel caso di impiego di un encoder a valori reali è sempre migliore del caso in cui si utilizzi un encoder antipodale, con un miglioramento medio di 2.5 dB circa. Si osserva, inoltre, un miglioramento più marcato per bassi valori di m (con valori inferiori a 20) dove il miglioramento è oltre i 4 dB, mentre per alti valori di m (con valori superiori a 28) il miglioramento si attesta fra 1.5 e 2 dB circa. Sapendo che m descrive la dimensione del dato a valle dell'encoder, il risultato ottenuto è ragionevole poiché se la matrice di encoder ha valori reali, la trasmissione di un dato codificato in un canale rumoroso risulta più robusta, riducendo la probabilità di commettere un errore nella ricostruzione implementata dal decoder. Osservando i due trend, si osserva inoltre che passando da $m = 16$ a $m = 18$ (da CR pari a 4 a CR pari a 3.6) il miglioramento è molto più marcato piuttosto che quando si passa da $m = 38$ a $m = 40$ (da CR pari a 1.7 a CR pari a 1.6). L'osservazione si può giustificare dicendo che nel primo caso la compressione porta alla perdita di una quantità di dati maggiore rispetto al secondo caso (che contiene circa il doppio dei dati rispetto al primo caso); ne consegue che, nel primo caso, ridurre il CR incrementando m implica un miglioramento apprezzabile delle performance che si attesta intorno a 16.8 dB sia per l'encoder a valori reali che per l'encoder antipodale mentre nel secondo caso il miglioramento per ambo gli encoder è inferiore ad 1 dB.

Nel caso, ideale, in cui il canale non presenti alcun tipo di rumore e la trasmissione sia effettuata considerando solamente le perdite introdotte dalla compressione, la performance, descritta in Fig. 3.7b, ottenuta con un encoder a valori reali è solo a tratti migliore in termini di valore di ARSNR rispetto al caso di un encoder antipodale. Il trend osservato in Fig. 3.7b è confermato dai



(a) Metrica ARSNR, ISNR = 60dB, sparsità fissata $\kappa = 16$, confronto matrice di encoding antipodale e a valori reali.



(b) Metrica ARSNR, ISNR = ∞, sparsità fissata $\kappa = 16$, confronto matrice di encoding antipodale e a valori reali.

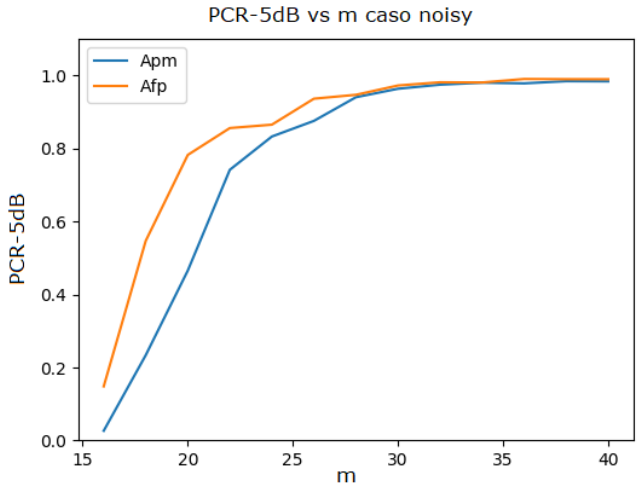
Figura 3.7: Metrica ARSNR al variare della dimensione del dato a valle dell'encoder, della tipologia di encoder, fissata la sparsità del segnale $\kappa = 16$.

dati riportati in Tab. 3.2 dove si osserva per $m = 16$ un miglioramento pari a 15 dB circa quando si utilizza un encoder a valori reali piuttosto che un encoder antipodale; per valori intermedi ($m = 24{,}28$) l'encoder antipodale risulta essere più performance rispetto all'encoder a valori reali di circa 1.6 - 1.9 dB, mentre per alti valori ($m = 36{,}40$) i due encoder sono pressoché equivalenti analizzando l'ARSNR. Si osserva poi come il trend nei due grafici sia sostanzialmente molto simile poiché si ha un forte incremento di performance nei primi valori e poi la curva tende a diventare sempre meno pendente per il caso di studio con rumore aggiunto, mentre diventa sostanzialmente piatta nel caso in cui non ci sia rumore. Si osserva poi che i valori di ARSNR sono profondamente diversi (circa di un fattore 5) nel caso noisy e noiseless come ci aspetterebbe. Si passa poi all'analisi della PCR-5dB.

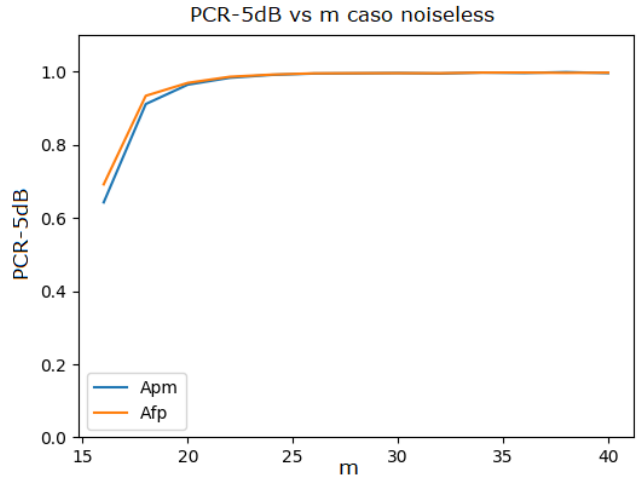
Per quanto riguarda la PCR-5dB (probabilità di ottenere una ricostruzione corretta del segnale considerando come minimo valore di RSNR accettabile per una corretta ricostruzione 5dB) si osserva che nel caso noisy, con SNR pari a 60 dB, descritto in Fig. 3.8a, la performance segue un trend inizialmente positivo e poi progressivamente sempre più piatto. Per bassi valori di m (nell'intorno di 16) la probabilità di ricostruire correttamente il segnale, fissato la soglia di RSNR da raggiungere, è piuttosto bassa, intorno al 5% se si impiega un encoder antipodale ed intorno al 15% se si utilizza l'encoder a valori reali. Analizzando casistiche intermedie ($m = 24$) si ottiene una PCR intorno all'85% per entrambi gli encoder, decisamente migliore rispetto al caso citato in precedenza per piccoli valori di m; infine, se si considerano elevati valori di m (piccoli valori di CR) le performance, non solo sono coincidenti, ma anche prossime all'unità ottenendo una elevata probabilità di ricostruire correttamente il segnale imposta come soglia di RSNR da superare 5dB. Significa che per bassi valori di m la ricostruzione è comunque portata a termine, ma la presenza di rumore dan-

neggia in maniera inevitabile il segnale $y = \mathcal{L}_A(x)$ a valle dell'encoder, mentre per alti valori di m la ricostruzione, salvo qualche raro caso, supera sempre la soglia di RSNR imposta, ma non sarà comunque perfetta perchè il segnale ha subito un processo di encoding con perdita di informazione. Analizzando ora le due curve nel complesso, si osserva una probabilità di corretta ricostruzione con soglia fissata a 5 dB migliore del 10% circa nel caso di encoder a valori reali piuttosto che con l'utilizzo di encoder antipodale quando $m = 16$ e del 30% quando $m = 20$; si deduce quindi che mentre per bassi valori di CR i due encoder siano prestanti circa nella stessa maniera, per elevati valori di CR (bassi valori di m) l'encoder a valori reali sia più affidabile in termini di PCR-5dB. Nel caso noiseless invece le curve sono sostanzialmente identiche dato che l'escursione media fra i due encoder è pressoché trascurabile. In questo caso la performance raggiunge una PCR-5dB del 65-70% per m pari a 16, ma già per $m = 20$ supera il 95% in entrambe le casistiche considerate. L'assenza di rumore nel canale di trasmissione garantisce, inoltre, il raggiungimento di una fedeltà nella ricostruzione piuttosto elevata.

Descrivendo la scelta di un encoder a valori reali con `floating`, la scelta di un encoder antipodale con `antipodale` e l'indifferenza nella scelta dell'encoder con `------`, in Tab. 3.3 si descrive al variare delle quattro metriche valutate e di tre range di m quale sia l'encoder con le prestazioni migliori. Come descritto in Fig. 3.7 e 3.8 l'impiego di un encoder a valori reali è sempre preferibile, dal punto di vista delle performance (ARSNR e PCR-5dB), per un'implementazione fisica del sistema, rispetto ad un encoder antipodale, in quanto non è possibile costruire un canale di trasmissione privo di rumore. Tuttavia, occorre valutare anche l'onere computazionale che nel caso di un encoder a valori reali è superiore rispetto all'impiego di un encoder antipodale. Significa che, a priori, non è possibile definire quale sia l'encoder più performante, ma si possono



(a) Metrica PCR-5dB, ISNR = 60dB, sparsità fissata $\kappa = 16$, confronto matrice di encoding antipodale e a valori reali.



(b) Metrica PCR-5dB, ISNR = ∞, sparsità fissata $\kappa = 16$, confronto matrice di encoding antipodale e a valori reali.

Figura 3.8: Metrica PCR-5dB al variare della dimensione del dato a valle dell'encoder, della tipologia di encoder, fissata la sparsità $\kappa = 16$.

	ARSNR 60 dB	ARSNR ∞	PCR-5dB 60 dB	PCR-5dB ∞
m = 16,20	floating	floating	floating	floating
m = 24,28	floating	antipodale	floating	——
m = 36,40	floating	antipodale	——	——

Tabella 3.3: Confronto finale fra le due tipologie di encoder per alcuni valori di m analizzando le metriche proposte.

definire due casistiche: se il dispositivo fisico utilizzato per l'implementazione dell'encoder dispone di una memoria e di una potenza di calcolo sufficiente ad immagazzinare la matrice di sensing e a svolgere i calcoli e la trasmissione del dato in un tempo ragionevole, allora la scelta di un encoder a valori reali può essere privilegiata rispetto alla scelta di un encoder antipodale. Qualora il trasmettitore, abbia limitate capacità di memorizzazione e di calcolo (si pensi ad esempio ad un trasmettitore low power che debba fornire uno stream di dati per un tempo elevato e non disponga di un modulo MAC apposito) allora può essere conveniente utilizzare un encoder antipodale per garantire lo streaming dei dati riducendo la quantità di operazioni da svolgere per unità di tempo e la performance finale in termini di ARSNR e di PCR-5dB.

3.3.2 Impiego di un encoder antipodale con parametro di scaling

Si vuole, dopo aver confrontato le performance ottenute con il layer antipodale e quello a valori reali, introdurre un secondo layer antipodale, che funga da encoder come i precedenti, i cui pesi siano moltiplicati per un coefficiente, detto parametro di scaling α, e successivamente manipolati da una funzio-

ne segno. Ancora una volta, sarà necessario definire una nuova rete neurale profonda, effettuare una fase di training al variare di m e utilizzare la DNN durante la fase di decodifica per valutare le performance del sistema dotato del nuovo encoder antipodale in termini di ARSNR e PCR-5dB. Si riporta in Appendice C i passaggi necessari a comprendere il dimensionamento del parametro di scaling. Facendo riferimento alla descrizione del layer antipodale di Appendice B, si può modificare il comportamento del layer senza alterarne la struttura, implementando le due seguenti operazioni:

$$\alpha = \frac{\sum_i |A_i|}{m \cdot n}$$

$$y = x \cdot \alpha \ \text{sign}(A)$$

Come per le altre casistiche analizzate in precedenza, occorre a valle della definizione della DNN, effettuare una fase di training della stessa e valutare i risultati in ricostruzione attraverso le metriche ARSNR e PCR-5dB. Si esegue la fase di training in accordo con quanto descritto nel Cap. 3.2 definendo la DNN con l'encoder antipodale con parametro di scaling α e si valutano i risultati. Dopo aver completato il training per due valori di m, pari a 16 e 28, si è provveduto all'implementazione della fase di decoding ricavando i risultati per le metriche ARSNR e PCR-5dB mostrati in Tab. 3.4. Analizzando i risultati, si osserva per $m = 16$ un valore di ARSNR più elevato quando si utilizza un encoder antipodale con parametro di scaling α rispetto al caso in cui l'encoder non impiega tale parametro, mentre per $m = 28$ il trend non è confermato rendendo l'encoder antipodale senza parametro di scaling più efficace. Il risultato ottenuto non è in linea con le aspettative per quanto concerne le metriche di ARSNR e PCR-5dB sia nel caso in cui ci sia rumore (fissando sempre per tutti i test effettuati un SNR pari a 60dB) che nel caso di assenza di rumore; pertanto sono stati condotti esperimenti volti a capire dove potesse essere il problema. Si è giunti alla conclusione che il layer antipodale con il parametro di scaling

			ARSNR		PCR-5dB	
		ISNR	60.0	∞	60.0	∞
α	CR	enc				
0.5	4.0	Apm	36.16	189.88	0.03	0.64
		Afp	40.43	204.76	0.15	0.69
		Apm scaling	37.98	190.89	0.04	0.64
	2.3	Apm	58.24	293.75	0.94	0.99
		Afp	59.97	291.86	0.95	0.99
		Apm scaling	57.99	293.48	0.91	0.99

Tabella 3.4: Risultati ARSNR e PCR-5dB al variare del Compression Ratio confrontando gli encoder antipodale, a valori reali e antipodale con parametro di scaling α.

α non fosse allenato durante il training dell'intera rete neurale, poiché la funzione di custom loss non presentava miglioramenti significativi (come invece ci si dovrebbe aspettare dal training di una rete neurale). Più nel dettaglio, si è giunti al risultato osservando che una DNN composta dal solo layer antipodale con parametro di scaling, aveva una funzione di custom loss che non presentava miglioramenti durante la fase di training. La problematica riscontrata è dovuta al fatto che la funzione segno applicata alla matrice elimina gli effetti del training e la matrice utilizzata a valle dell'algoritmo di retro-propagazione dell'errore non cambia mai. Possibili soluzioni, non esplorate durante il lavoro presentato, da vagliare potrebbero essere una differenziazione del parametro di learning rate fra l'encoder antipodale con parametro di scaling e il resto della rete oppure una differente inizializzazione del layer.

3.4 Training della Rete Neurale profonda con sparsità variabile

Dopo aver analizzato il comportamento del sistema variando la matrice di encoding e valutandone le performance si è provveduto alla modifica del sistema, rimuovendo un ulteriore vincolo. Rispetto ai risultati valutati in precedenza, d'ora in poi si assumerà la sparsità κ degli elementi del Training Set e del Testing Set variabile. Pertanto, a valle della rimozione del vincolo sulla sparsità, si rende necessaria l'introduzione di un nuovo parametro. Riprendendo quanto detto nel Cap. 2.2, la sparsità dei dati è stata determinata imponendo una determinata soglia di energia e ne consegue che sarà necessario tenere traccia della soglia utilizzata per analizzare un determinato set di segnali. L'impiego di un data set contenente segnali a sparsità variabile richiede la definizione e il training di una nuova rete neurale che evidentemente deve essere allenata sapendo a priori che la sparsità κ non è più fissata a 16 come nel precedente training, ma è variabile seguendo distribuzioni simili a quelle riportate in Fig. 2.13. L'analisi di sparsità condotta nel Cap. 2.2 è stata anche impiegata successivamente per la creazione del Training Set, del Test Set e del Validation Set da utilizzare durante il training della rete con segnali in ingresso a sparsità variabile; tuttavia le ingenti dimensioni del data set non consentono di impiegare l'intero Set, ma solo una sua porzione. Occorre quindi determinare le dimensioni del Training Set e del Test Set per il training della DNN associata:

- Dimensione del Training Set:

$$\#\text{ di elementi training} = \frac{\text{Limite memoria}}{2 \cdot 1.25 \cdot 8 \cdot n} \tag{3.11}$$

- Dimensione del Test Set:

$$\#\text{ di elementi test} = \frac{\#\text{ di elementi training}}{4} \tag{3.12}$$

dove il *Limite Memoria* è fissato a 5Gbit, n è la dimensione del dato in ingresso, fissata a 64 per il training in esame, 2 indica che si stanno considerando dati e label associati al data set, e la frammentazione fra Training Set e Test Set è pari a 80%-20% rispettivamente. Per quanto concerne la struttura della rete neurale, essa non è modificata nel numero di layer e di neuroni rispetto al training descritto nel dettaglio nel Cap. 3.2; pertanto non sarà descritto nuovamente il processo di training. Le callbacks utilizzate per il training proposto nel Cap. 3.2 non sono state modificate e non sono state alternate nemmeno le proprietà della fase di configurazione della rete e del training vero e proprio. Sono fissati i seguenti vincoli per la realizzazione della fase di training:

- Sul segnale si pone $n = 64$, m variabile (si esplorano $m = 20$, 24, 28 e 32), *soglia di energia per la sparsità* variabile (si esplorano $th = 0.99$ e 0.999) e $\alpha = 0.5$;

- Sul training si pone $epoch = 1000$, $batch\_size = 30$, si sceglie l'impiego di una delle due GPU disponibili e si fissano gli estremi del Training Set da importare per la fase di training $start = 0$ e $end = 4194304$ così da poter utilizzare tutta la memoria disponibile;

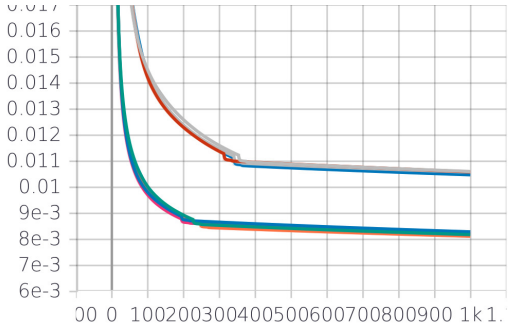
- Sulle callbacks si pone come variabile da valutare la performance di loss, il fattore per la modifica del learning rate pari a 0.2 e un'attesa di 20 epoche prima di modificare il learning rate.

Avendo a disposizione un data set composto dalla terna x, $\hat{x}$ e y, rispettivamente segnale di partenza, segnale di partenza sparsificato con energia fissata e segnale codificato secondo la legge $y = \mathcal{L}_A(x)$ si potrebbe scegliere su quale

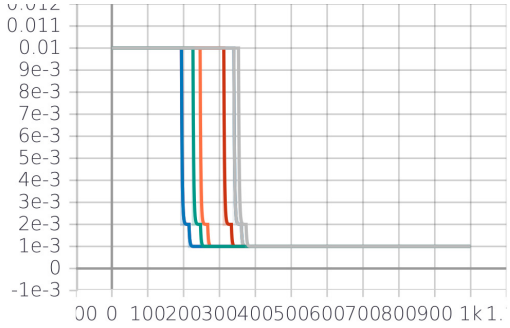
segnale in ingresso effettuare il training valutando fra il segnale originale e quello sparsificato. Si è scelto di utilizzare il segnale originale per effettuare il training della rete neurale in esame e per descrivere la fase di ricostruzione del segnale che porta alla descrizione delle metriche di ARSNR e PCR-5dB al variare del CR.

3.4.1 Risultati del training della DNN con κ variabile

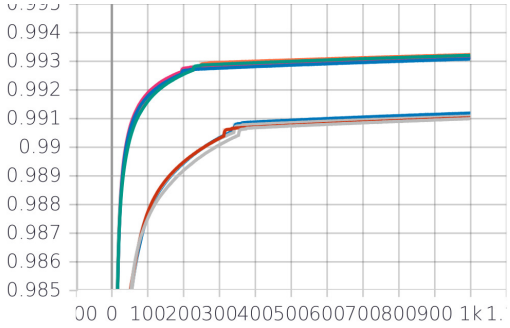
A valle del processo di training della DNN si valutano le performance di `multiclass_loss_alpha`, `learning_rate` e `sparse_accuracy`, descritte graficamente in Fig. 3.9 e numericamente in Tab. 3.5. Analizzando la funzione di loss si osserva un trend discendente della performance, in accordo con la teoria descritta in precedenza e similmente alla funzione di loss del training a sparsità fissata. Ad ogni epoca l'algoritmo di retro-propagazione dell'errore modifica opportunamente i pesi e la funzione di loss riduce i suoi valori. Per quanto noto dalla teoria presentata nel Cap. 1, la funzione di loss deve essere minimizzata e quando accade si è in presenza di un minimo locale o di un minimo globale della funzione. Appare chiaro, quindi, che all'avanzare delle epoche, se il training procede nella corretta direzione, la funzione di loss continua a diminuire, ma la riduzione rispetto all'epoca precedente è sempre meno marcata, poiché il gradiente calcolato dalla funzione di ottimizzazione (durante la fase di training si è imposto l'impiego di un ottimizzatore tipo discesa stocastica del gradiente, come nel training a sparsità fissata) diventa sempre più piccolo avvicinandosi al minimo in questione (locale o globale che sia). Per ogni valore di m e dell'energia del segnale sparsificato si associa una curva del grafico di Fig. 3.9a e nel momento in cui la rete non è più in grado di migliorare le proprie performance, tramite l'impiego della callback `ReduceLROnPlateau()` si riduce progressivamente il learning rate partendo da 0.01 ed utilizzando l'eq.



(a) Evoluzione della funzione di loss `multiclass_loss_alpha` con sparsità variabile del Training Set durante la fase di training.



(b) Evoluzione del parametro di learning rate con sparsità variabile del Training Set durante la fase di training.



(c) Evoluzione della metrica `sparse_accuracy` con sparsità variabile del Training Set durante la fase di training.

Figura 3.9: Risultati della fase di training di una rete neurale profonda con neuroni del primo livello a valori reali con sparsità variabile e energia del segnale sparsificato variabile.

3.4 fino ad arrivare al minimo valore che il learning rate possa assumere, fissato a 0.001, come nel caso a sparsità fissata. Giunti al minimo valore assumibile, interviene una seconda callback `EarlyStopping()` che ferma il training della rete. Analizzando Fig. 3.9a si osserva la presenza di due famiglie di curve distinte al variare dell'energia utilizzata per rappresentare il segnale sparsificato. La funzione di loss assume mediamente valori maggiori quando la soglia imposta è pari a 0.999 con valori compresi fra 0.0106 e 0.0105 circa e si osserva un trend discendente in termini di valori della funzione di loss passando da m pari a 20 a m pari a 32. Ne consegue che si possa trarre una prima conclusione che mostri un miglioramento della funzione di loss al diminuire del CR. Il risultato ottenuto è in linea con quanto ci si aspetterebbe dal punto di vista teorico; la presenza di una soglia fissata a 0.99 o a 0.999 modifica il valore di sparsità e quindi modifica la quantità di informazione preservata all'interno del segnale sparsificato. Il risultato è che, maggiore è la soglia di energia considerata, peggiori saranno le performance che si ottengono a valle della fase di training. Analizzando Fig. 3.9b circa la rappresentazione del parametro di learning rate al variare delle epoche si osserva che per tutti i valori di m si ha l'intervento della callback `ReduceLROnPlateau()` che riduce il parametro di learning rate, dopo 20 epoche senza miglioramenti di performance, da 0.01 a 0.001 con un passo di scorrimento fissato a 0.2; dopo essere giunti ad un parametro di learning rate pari a 0.001, in assenza di ulteriori miglioramenti per 20 epoche si ha l'interruzione del training, utilizzando l'apposita callback. Dai risultati riportati in Tab. 3.5, si osserva che anche analizzando il parametro di learning rate esiste una forbice di performance fra le DNN allenate con energia del segnale sparsificato fissata a 0.990 e le DNN allenate con energia fissata a 0.999. Infatti, nel primo caso la callback `ReduceLROnPlateau()` interviene sulle DNN ottenendo un parametro di learning rate fissato a 10^{-3} fra le epoche

m	energia	max accuracy	min loss	learning rate
20	0.990	0.9932	0.00818	10^{-3} @ 267 epoch
	0.999	0.9910	0.01061	10^{-3} @ 394 epoch
24	0.990	0.9931	0.00829	10^{-3} @ 235 epoch
	0.999	0.9910	0.01060	10^{-3} @ 353 epoch
28	0.990	0.9931	0.00823	10^{-3} @ 235 epoch
	0.999	0.9911	0.01058	10^{-3} @ 381 epoch
32	0.990	0.9932	0.00813	10^{-3} @ 287 epoch
	0.999	0.9912	0.01048	10^{-3} @ 381 epoch

Tabella 3.5: Risultati del training di una DNN con encoder a valori reali e data set con segnali a sparsità variabile, al variare di m e dell'energia del segnale sparsificato.

230 e 290, mentre nel secondo caso la callback interviene fissando il valore di learning rate a 10^{-3} fra le epoche 350 e 390 circa.

Si può, infine, analizzare la Fig. 3.9c relativa alla metrica `sparse_accuracy` allo scorrere delle epoche, al variare di m e dell'energia fissata per definire la sparsità. Si osserva, in maniera duale rispetto alla funzione di loss, un incremento molto marcato nelle prime centinaia di epoche della performance, mentre nella seconda metà (ultime 500 epoche) il miglioramento è sempre più debole o addirittura assente. Il trend descritto è ciò che ci si aspetta nel momento in cui si valutano i risultati attesi di una rete neurale come quella presentata dato che nelle prime centinaia di epoche la discesa stocastica del gradiente si muove utilizzando grandi step ad ogni epoca (riducendo molto la funzione di loss e aumentando significativamente la metrica di accuracy), mentre successivamente gli step diventano sempre più piccoli o nulli (se l'algoritmo

di retro-propagazione dell'errore non è in grado di migliorare ulteriormente i parametri del modello allenato) grazie all'impiego delle callback citate. Nel primo caso la metrica di *sparse_accuracy* migliora, ma più lentamente, mentre nel secondo caso non si riscontra un aumento della metrica analizzata perchè il training è terminato. Dopo aver analizzato il trend al variare delle epoche, si può analizzare il valore di `sparse_accuracy` raggiunto alla fine del training al variare di m. Supponendo di fissare l'energia utilizzata per descrivere la sparsità del segnale e valutando la `sparse_accuracy` al variare di m si ottengono valori pressoché identici. Infatti, considerando l'energia fissata a 0.99, al variare di m si ha una `sparse_accuracy` nell'ordine del 99.3%, mentre considerando l'energia fissata a 0.999, al variare di m si ha una `sparse_accuracy` nell'ordine del 99.1%. Osservando l'evoluzione della curva della funzione di loss di Fig. 3.9a la presenza di due famiglie di curve nella rappresentazione della `sparse_accuracy` è in linea con le aspettative. Effettuando un confronto fra i risultati ottenuti con il primo training proposto e quello appena descritto e facendo riferimento ai risultati raccolti nelle Tab. 3.1 e 3.5 si osserva che per il caso di energia fissata a 0.99 le performance sono circa coincidenti, in termini di `sparse_accuracy` e funzione di loss, mentre per l'energia fissata a 0.999 si ha un degrado delle performance. Per quanto concerne la prima osservazione, i due training non possono produrre lo stesso risultato in maniera esatta (a meno di un evento di casualità) poiché il Training Set è diverso e il punto della funzione di loss in cui si inizia il training è casuale. Per quanto riguarda la seconda osservazione, il degrado di performance è giustificabile sapendo che, più la soglia di energia considerata è alta, più è elevato il contenuto informativo e quindi diventa più difficile per la DNN ricostruire con una certa affidabilità il segnale.

Concludendo, il trend descritto dai risultati mostrati è in linea con quanto

ci si aspetta dal punto di vista teorico dato che al diminuire di m aumenta il Compression Ratio, come si osserva dalla definizione di CR. All'aumentare del CR si riduce la quantità di informazione, fissata l'energia utilizzata per descrivere il segnale sparsificato, inviata poiché la compressione riduce la dimensionalità del dato a valle dell'encoder e pertanto si degrada progressivamente il risultato finale. Ricordando che si è fissato n pari a 64, si osserva anche come ci sia un netto distacco fra le curve per energia pari a 0.990 e 0.999, mentre al variare di m non si osservano apprezzabili differenze. Se, da un lato, prendere un valore di CR molto elevato consente di ridurre la quantità di informazione da inviare a discapito di risultati meno performanti, scegliere un CR molto piccolo aumenta la mole dei dati da gestire, ma consente di raggiungere prestazioni migliori. Si rende pertanto necessario scegliere un CR in funzione del problema da analizzare, valutando opportunamente vantaggi e svantaggi della scelta, eventualmente considerando trade-off fra le soluzioni limite proposte.

3.4.2 Analisi delle prestazioni al variare dell'energia utilizzata per descrivere il segnale sparsificato

Dopo aver mostrato i risultati della fase di training della DNN dati in ingresso segnali con sparsità variabile, si vuole utilizzare la DNN per ricostruire il segnale y codificato al variare delle sue dimensioni e della percentuale di energia conservata nel segnale sparsificato. Saranno pertanto considerate le combinazioni di m ed *energia* e saranno valutate le metriche ARSNR e PCR-5dB utilizzando con le modifiche necessarie gli strumenti presentati nel Cap. 3.3. Infatti, l'idea attorno cui ruota la decodifica di y non è modificata e pertanto occorre ripetere esattamente le stesse operazioni utilizzando gli strumenti già presentati.

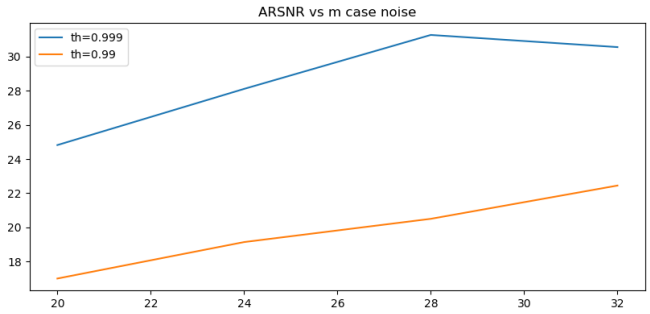
Si mostrano, quindi, i risultati relativi alle metriche ARSNR e PCR-5dB

al variare del Compression Ratio (CR) e dell'energia conservata nel segnale sparsificato (sapendo che la sparsità è variabile), ricordando che per lo studio condotto in questa fase si è posto n pari a 64 e sapendo che le ricostruzioni sono effettuate per CR compresi fra 3.2 e 2.0 e per valori di energia pari a 0.990 e 0.999. L'idea alla base è l'impiego di una soluzione che si ponga come obiettivo la sparsificazione del segnale e la ricostruzione del segnale che non è stato sparsificato. Segue che quando si considerano segnali con una quota di energia fissata a 0.99, il massimo RSNR raggiungibile è 20dB nel caso ideale in cui la componente di decoding riesca ad individuare correttamente tutte le componenti del segnale da ricostruire. Nel caso in cui si consideri un'energia del segnale sparsificato pari a 0.999 allora analogamente il massimo RSNR raggiungibile non supera 30dB quando il decoder è ideale[1]. La troncatura prodotta dalla sparsificazione del segnale introduce un degrado nel segnale stesso tale da rendere trascurabile la componente di rumore gaussiano bianco introdotto nel sistema. Partendo da ciò, si giustificano risultati circa uguali quando si analizza l'ARSNR nel caso noiseless e nel caso con rumore additivo introdotto nel segnale. Sebbene si sia valutata anche la metrica di PCR-5dB, si è deciso di non mostrare tali risultati poiché sono privi di consistenza. Infatti, nel momento in cui si osservano le metriche di ARSNR si osserva che non è possibile avvicinarsi ad un valore di ARSNR pari a 60dB. Ne consegue che non ha nessun senso parlare di PCR a 55dB dato che tale risultato non è raggiungibile. Sono quindi solamente rappresentati i risultati in termini di ARSNR al variare di energia del segnale sparsificato e del CR. Si abbandona la scelta di ragionare analizzando l'energia del segnale per introdurre problematiche legate alla ridefinizione del supporto che sono trattate nel seguito.

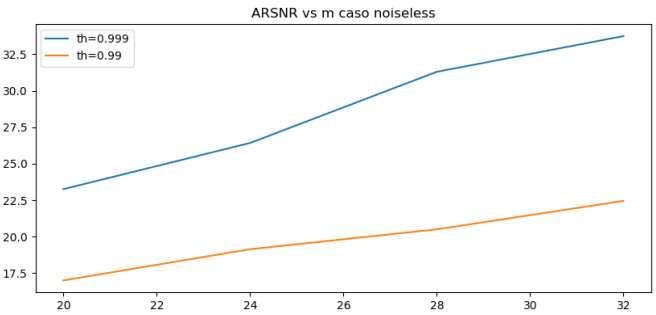
[1]Ciò è dovuto al fatto che il segnale che consideriamo durante l'analisi non è il segnale di partenza dato che è stato sparsificato, quindi una componente di dettaglio del segnale è persa a priori nel processo di sparsificazione.

			ARSNR	
		ISNR	60.0	∞
α	CR	energia		
0.5	3.2	0.990	17.00	17.00
		0.999	24.82	23.25
	2.7	0.990	19.14	19.14
		0.999	28.11	26.42
	2.3	0.990	20.50	20.50
		0.999	31.27	31.29
	2.0	0.990	22.45	22.45
		0.999	30.56	33.74

Tabella 3.6: Risultati ARSNR al variare del Compression Ratio con sparsità κ variabile congiuntamente nel caso rumoroso e in assenza di rumore.



(a) Metrica ARSNR, ISNR = 60dB, sparsità variabile funzione dell'energia del segnale sparsificato.



(b) Metrica ARSNR, ISNR = ∞, sparsità variabile funzione dell'energia del segnale sparsificato.

Figura 3.10: Metrica ARSNR al variare della dimensione del dato a valle dell'encoder, con sparsità variabile fissati due valori energetici del segnale sparsificato.

3.5 Metodi di realizzazione del supporto del segnale sparsificato

Per l'ultima parte del lavoro svolto, si è voluto esplorare una generalizzazione del segnale sparso. Infatti, nel momento in cui si misura un segnale fisicamente presente e, attraverso una opportuna base sparsa, se ne valuta la sua rappresentazione sparsa $\xi = D^\top x$ non si ottiene un segnale con κ contributi non nulli e $n - \kappa$ nulli, ma piuttosto ci saranno κ campioni con un'ampiezza elevata e $n - \kappa$ con un'ampiezza piccola o trascurabile, rendendo il segnale non più sparso, ma comprimibile [12]. Analizzando un segnale del genere, non è possibile descrivere il supporto del segnale x utilizzando una tecnica classica poiché il segnale sparsificato contiene, dal punto di vista generale, n elementi non nulli. Il segnale acquisito, quindi, può essere rappresentato come:

$$x = D_{|z}\xi_{|z} + x_d \tag{3.13}$$

dove $D_{|z}$ è la matrice di sparsità che contiene la base utilizzata per descrivere la rappresentazione sparsa del segnale che preserva solo le colonne associati agli elementi non nulli del supporto, $\xi_{|z}$ è la rappresentazione del segnale sparsificato nell'opportuna base sparsa che contiene elementi non nulli in corrispondenza degli elementi non nulli del supporto e x_d contiene gli elementi del segnale trascurabili [12]. Diventa di fondamentale importanza, quindi, definire un supporto z in grado di descrivere il massimo RSNR durante la ricostruzione del segnale. La scelta del numero e delle posizioni degli uni del supporto è vincolata alle due seguenti tematiche:

- Aumentare il numero di uni nel supporto riduce gli errori introdotti da x_d;

- Ridurre il numero di uni nel supporto riduce l'errore commesso durante la ricostruzione.

Si vuole, quindi, determinare il supporto z^* che definisca il trade-off ottimo fra le due tematiche appena descritte, risolvendo il seguente problema di ottimizzazione:

$$z^* = \arg \max_{z \in [0,1]^n} \frac{\|x\|_2}{\|x - D_{|z}\hat{\xi}_{|z}\|_2} \tag{3.14}$$

Tuttavia, il problema principale è legato al fatto che, per determinare il supporto z che renda massimo il RSNR, occorre calcolare tale metrica per tutte le possibili configurazioni del supporto z. L'operazione, per valori di n pari a 128, ad esempio, è insostenibile, per via dell'elevato costo computazionale richiesto dato che fissato $n = 128$, occorrerebbe valutare approssimativamente $3.4 \cdot 10^{38}$ vettori binari. Si valutano pertanto due approcci distinti descritti nel seguito, denominati primo metodo e secondo metodo.

3.5.1 Approccio greedy diretto

Il primo metodo proposto si basa sulla definizione di un algoritmo greedy che riduca il numero di passi da compiere per definire il supporto, passando da 2^n a n. Esso, descritto nell'Algoritmo 2, si basa sulla ricerca attraverso la base sparsa, del segnale rappresentato in tale base, per poi farne il valore assoluto e ordinarne gli elementi in ordine decrescente. Occorre poi, in maniera ciclica, introdurre nel supporto un elemento non nullo in corrispondenza dell'indice associato al massimo valore di $|\xi(i)|$ non ancora considerato. Ad ogni elemento aggiunto nel supporto si definisce il segnale sparso $\hat{\xi}$ nella base sparsa ristretta agli elementi non nulli del supporto $D_{|z}$, si ricostruisce il segnale e si calcola il RSNR. Sono fissati per l'analisi dell'algoritmo la dimensione del segnale di partenza, pari a $n = 128$, la dimensione del dato a valle dell'encoder, pari a

Algorithm 2 Step per l'implementazione del primo metodo nella ricerca del supporto ottimo

1: $y = Ax$
2: $\xi = D^\top x$
3: $|\xi|$
4: Ordinare in maniera decrescente $|\xi|$
5: **for** $i = 0, \ldots, n-1$ **do**
6: Posizionare un uno nel supporto z seguendo gli indici descritti dall'ordinamento di $|\xi|$
7: $B^\dagger_{|z} = (AD_{|z})^\dagger$
8: $\hat{\xi} = B^\dagger_{|z} y$
9: $\hat{x} = D_{|z}\hat{\xi}$
10: $\text{RSNR}(x, \hat{x}) = 20 \log_{10} \frac{\|x\|_2}{\|x-\hat{x}\|_2}$
11: **end for**

$m = 40$, e l'energia del segnale sparsificato, pari a 0.99. Si è poi scelto di utilizzare una matrice di encoding antipodale basata sull'impiego della Rakeness descrivendo un approccio Compressed Sensing basato su Rakeness (RakCS). L'approccio consente di modellare il segnale in ingresso assumendo che esso faccia parte di un processo stocastico la cui matrice di autocorrelazione è definita come:

$$\mathcal{A} = \mathbf{E}[aa^\top] = \frac{1}{2}\frac{n\mathcal{X}}{\text{tr}(\mathcal{X})} + \frac{1}{2}I_n \tag{3.15}$$

dove I_n è una matrice identità quadrata e $\mathcal{X} = \mathbf{E}[xx^\top]$ è la matrice di autocorrelazione del processo stocastico che genera gli ingressi. Analizzando il risultato ottenuto in Fig. 3.11 si osserva l'evoluzione del RSNR al variare della cardinalità del supporto z. In particolare si osservano due trend distinti: per bassi valori di cardinalità, si ha un trend positivo e all'aumentare della cardinalità, il RSNR cresce molto velocemente, per poi raggiungere una sorta

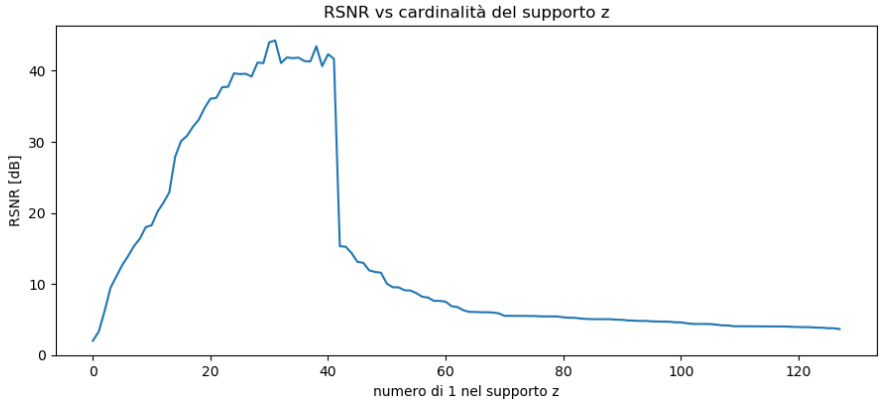


Figura 3.11: Evoluzione della metrica RSNR al variare della cardinalità del supporto, fissato il criterio di scelta degli elementi non nulli dello stesso, primo metodo.

di plateau fra $\|z\|_0 = 30$ e 40. Intorno a $\|z\|_0 = 40$ si ha un decadimento drastico del valore di RSNR poiché quando la cardinalità raggiunge tale valore, la matrice di cui occorre fare la pseudoinversa per ricostruire il segnale, descritto con $\hat{x}$, torna ad essere una matrice quadrata stesa con infinite controimmagini, rendendo poco performance la ricostruzione. Oltre a $\|z\|_0 = 40$ si ha sostanzialmente un progressivo decadimento della metrica valutata, indice del fatto che includere troppi elementi nel supporto porti ad un aumento degli errori commessi durante la ricostruzione. Numericamente parlando, il massimo valore di RSNR raggiunto è pari a 44.25dB con una cardinalità del supporto pari a 31, mentre il decadimento improvviso della metrica RSNR si ha con una cardinalità pari a 42, passando da un RSNR pari a 41.69dB ad un RSNR intorno a 15.32dB. I risultati ottenuti sono in linea con i risultati attesi poiché avendo considerato un SNR del segnale di partenza pari a 50dB, è ragionevole non raggiungere esattamente 50dB di RSNR visti gli errori e le approssimazioni introdotte nella fase di ricostruzione. Per quanto concerne invece il trend descritto, il risultato è in linea con quanto ci si aspetti sulla

base dei ragionamenti condotti in precedenza.

3.5.2 Approccio greedy a due livelli

Il secondo metodo proposto si basa sulla definizione di un algoritmo che attraverso un approccio iterativo, differente dal primo, possa comunque costruire il supporto che massimizzi il RSNR. Differentemente dal primo approccio, con il metodo proposto ora, fissata una configurazione del supporto risultato dello step (i-1)-esimo, si considerano, in maniera ciclica, tutti gli indici a cui è associato uno zero nel supporto considerato, si assume ad ogni iterazione che un elemento diverso fra quelli da considerare sia aggiunto al supporto, si calcola l'RSNR per ogni supporto considerato allo step i-esimo (cardinalità del supporto pari ad i) e si mantiene il supporto a cardinalità i che rende massimo l'incremento di RSNR rispetto al supporto con cardinalità i-1. Ne consegue che il modello che si implementa ora sia più lento e più oneroso in termini computazionali dovendo al i-esimo step calcolare n-i ricostruzioni, scegliere la più performante e determinare nuovamente la ricostruzione ottima che definirà il nuovo supporto del segnale, ottenendo un costo computazionale proporzionale a $n!$. Si può riassumere i passi dell'algoritmo utilizzando lo pseudo-codice definito in Algoritmo 3, dove fra righe 2 e 9 si procede alla valutazione del supporto con cardinalità incrementata di uno rispetto allo step precedente scegliendo il supporto che garantisce il massimo RSNR in ricostruzione, mentre fra 10 e 15 si definisce il supporto ottimo per la cardinalità i corrente. Le operazioni sono poi ripetute assumendo una cardinalità crescente fra 0 e n-1. Sono fissati per l'analisi dell'algoritmo la dimensione del segnale di partenza, pari a $n = 128$, la dimensione del dato a valle dell'encoder, pari a $m = 40$, e l'energia del segnale sparsificato, pari a 0.99. Si è poi scelto di utilizzare una matrice di encoding antipodale basata sull'impiego della Rakeness descrivendo anche

per il secondo metodo un approccio RakCS. Si valutano poi i risultati ottenuti mostrati in Fig. 3.12.

Algorithm 3 Step per l'implementazione del secondo metodo nella ricerca del supporto ottimo

for $i = 0, \ldots, n-1$ **do**
  for $j = 0, \ldots, n-1$ **do**
    $z_{test}[z == 1] = 1$
    $D_{test}[:, z == 1] = 1$
    $B^{\dagger}_{test|z} = (AD_{test|z})^{\dagger}$
    $\hat{\xi}_{test} = B^{\dagger}_{test|z} y$
    $\hat{x}_{test} = D_{test|z}\hat{\xi}$
    $\mathrm{RSNR}(x, \hat{x}_{test}) = 20 \log_{10} \frac{\|x\|_2}{\|x - \hat{x}_{test}\|_2}$
  end for
  idx = argmax(RSNR)
  z[idx] = 1
  $B^{\dagger}_{|z} = (AD_{|z}[:, \mathrm{idx}])^{\dagger}$
  $\hat{\xi} = B^{\dagger}_{|z} y$
  $\hat{x} = D_{|z}\hat{\xi}$
  $\mathrm{RSNR}(x, \hat{x}) = 20 \log_{10} \frac{\|x\|_2}{\|x - \hat{x}\|_2}$
end for

Come per il primo metodo proposto e valutato, anche per il secondo metodo si evince un'evoluzione del RSNR al variare della cardinalità del supporto z composta da 2 trend distinti: per bassi valori di cardinalità, si ha un trend positivo e all'aumentare della cardinalità, il RSNR cresce, ma in maniera meno pronunciata rispetto al primo metodo proposto. Intorno ad una cardinalità circa pari a 40 si ha un decadimento del valore di RSNR meno pronunciato rispetto al primo metodo, ma comunque tale da motivare un peggioramento

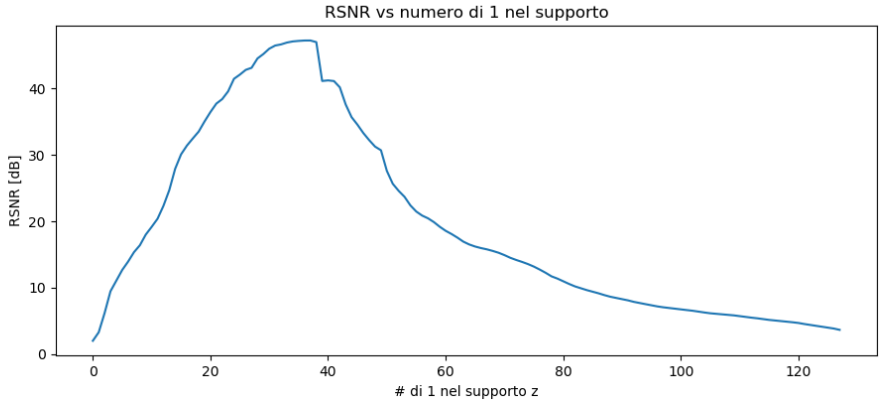


Figura 3.12: Evoluzione della metrica RSNR al variare della cardinalità del supporto, fissato il criterio di scelta degli elementi non nulli dello stesso, secondo metodo.

della performance. Il degrado di risultati è dovuto, come per il primo metodo, al fatto che la matrice di cui si effettua la pseudoinversa per ricostruire il segnale, definito da $\hat{x}$, diventa quadrata stesa e presenta infinite controimmagini. La ricostruzione tuttavia, per valori di cardinalità maggiori di 40, è migliore rispetto al primo metodo e il risultato potrebbe essere dovuto alla scelta differente degli elementi da inserire progressivamente nel supporto. Numericamente parlando, il massimo valore di RSNR raggiunto è pari a 46.06 dB con una cardinalità del supporto pari a 38, mentre si osserva un decadimento brusco del massimo RSNR intorno a 40 passando da 46.06dB a 40.72dB. L'aspetto interessante da evidenziare è legato al fatto che nel primo metodo il massimo valore di RSNR era raggiunto precedentemente al crollo di prestazione, mentre nel secondo metodo il massimo valore di RSNR si raggiunge al valore di cardinalità antecedente il decadimento di prestazione. La minor pendenza della fase discendente si può attribuire al diverso metodo di costruzione del supporto dato che si introduce il contributo che massimizza il RSNR piuttosto che il massimo valore assoluto disponibile nel segnale sparso. I risultati ottenuti

mostrano un miglioramento per quanto concerne il massimo valore di RSNR raggiungibile, circa 2dB maggiore a discapito di un aumento della cardinalità circa del 5%. Come nel primo metodo, non si può ambire al raggiungimento di un RSNR pari a 50dB dato che la fase di encoding produce la perdita di informazione e il segnale, a causa della sua natura compressa e non sparsa, introduce componenti trascurabili e maggiormente sensibili al rumore.

3.6 Confronto fra i due metodi in termini statistici

Si è infine realizzata una statistica su un numero fissato a priori di segnali per valutare il comportamento dei due metodi. Visto l'onere computazionale diverso fra le due metodologie proposte, per il primo metodo si è scelto un data set composto da 10^4 segnali con sparsità arbitraria, mentre per il secondo metodo si è scelto un data set composto da 2000 segnali. La statistica è comunque descrivibile anche con 2000 elementi e si evita una simulazione troppo dispendiosa. Sono stati ricavati durante l'analisi il massimo valore di RSNR e l'indice associato al massimo valore di RSNR, ripercorrendo per tutti i segnali del data set considerato gli step analizzati nelle sezioni precedenti, si sono sovrapposti i risultati e si è valutato il diverso comportamento fra le due strategie adottate. Analizzando Fig. 3.13 si osserva che il primo metodo, descritto dall'istogramma blu, ha una varianza minore, ma una media maggiore rispetto al secondo metodo, descritto dall'istogramma arancione. Nel primo caso si ha un valore medio di massimo RSNR pari a 41.57dB, mentre nel secondo caso il valore medio di massimo RSNR si aggira intorno a 38.07dB. Si osserva inoltre, per il secondo metodo una componente non trascurabile di risultati per una sparsità intorno a 20, mentre per il primo metodo non si rilevano, intorno

a 20, risultati considerevoli. Per quanto riguarda invece il trend descritto in Fig. 3.14 rappresentante l'indice associato al massimo valore di RSNR si ha una varianza simile fra i metodi, mentre il valore medio per il primo metodo è circa 33, mentre per il secondo metodo circa 34. In tal caso si può osservare una coda sostanzialmente sovrapponibile fra i metodi per κ che tende a 0, mentre per il secondo metodo si osservano alcuni risultati per valori di sparsità oltre 40. L'aspetto contrastante è la diversa metodologia per raggiungere il risultato finale; entrambi sono approcci algoritmici di tipo greedy, ma utilizzano approcci completamente differenti dato che il secondo metodo non conosce il segnale senza rumore poiché utilizza sempre il segnale a cui è stato aggiunto rumore gaussiano bianco (quindi il metodo proposto potrebbe trovare un'implementazione pratica date le ipotesi sul segnale di partenza) e non ha alcuna informazione in merito alla tipologia di rumore introdotta, mentre il primo metodo conosce il segnale senza rumore. Inoltre, il supporto ottimo non può essere determinato se non in maniera combinatoria, ma non è operativamente fattibile perché dovrebbero essere valutati tutti i possibili supporti di un segnale di lunghezza pari a 128. Occorre considerare pertanto, una serie di elementi che potrebbero fornire una componente buona per il calcolo del risultato finale. Nel primo metodo si può considerare un segnale con un elevato contenuto informativo ed un elevato rumore, mentre nel secondo metodo si potrebbe considerare una direzione dello spazio conoscendone l'incremento di RSNR prodotto per poter determinare le componenti che massimizzino la performance. Si consideri ora il grafico di Fig. 3.15 rappresentante la distribuzione delle coppie massimo valore di RSNR e indice associato confrontando i due metodi proposti. Il grafico è stato realizzato utilizzando un grado di trasparenza medio per poter mettere in risalto la concentrazione dei risultati oltre che la loro distribuzione. Analizzando i risultati del primo metodo si

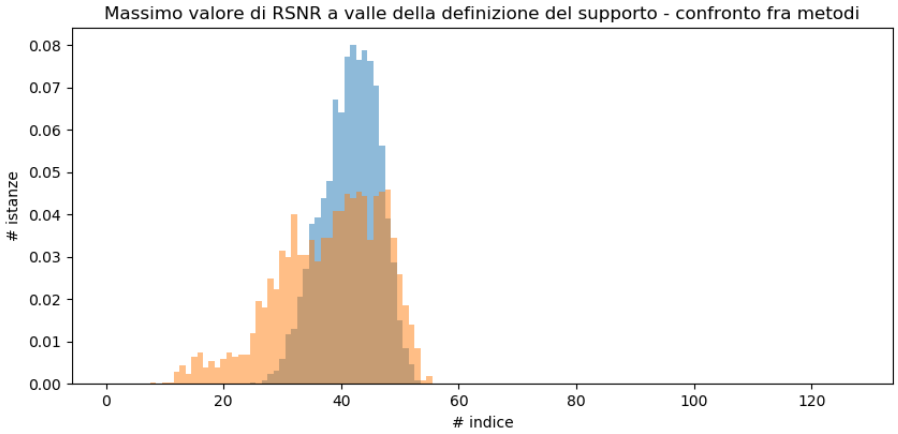


Figura 3.13: Rappresentazione della distribuzione dei massimi valori di RSNR nel data set considerato. Rappresentazione delle istanze in percentuale.

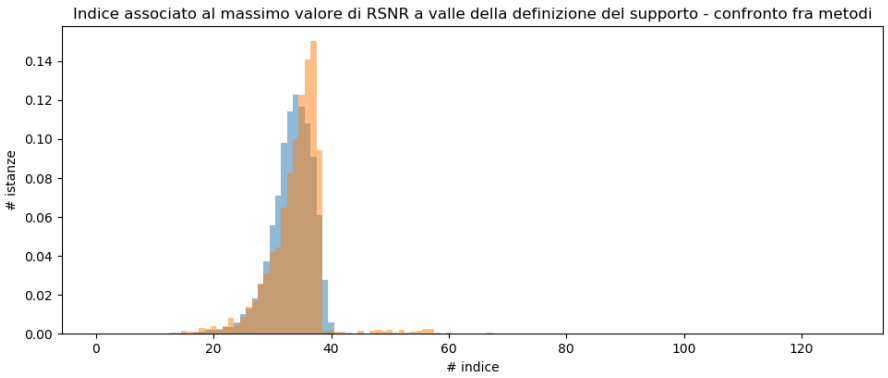


Figura 3.14: Rappresentazione della distribuzione degli indici associati al massimo valore di RSNR nel data set considerato. Rappresentazione delle istanze in percentuale.

osservano elevati valore di RSNR quando la cardinalità del segnale è ridotta; all'aumentare della cardinalità, la variabilità del massimo RSNR aumenta e si ha un crollo drastico delle performance quando la cardinalità supera 40 poiché la matrice diventa stesa e possiede infine controimmagini (non ho certezza che quella trovata sia la corretta). Se si considerano i risultati del secondo metodo si osserva per bassi valori di cardinalità una performance piuttosto scadente (intorno a 10-20dB), mentre all'aumentare della cardinalità, anche per il secondo metodo, si ha un aumento della variabilità. Nel caso in esame, tuttavia, per valori oltre 40 si trovano prestazioni in linea (dal punto di vista numerico) con i risultati ottenuti per bassi valori di cardinalità. I due metodi, sebbene sfruttino diverse informazioni, producono una distribuzione più o meno simile dato che la maggioranza delle istanze raccolte si trova fra 30 e 40 dB (con le opportune considerazioni già descritte in precedenza). Un aspetto certamente rilevante è la presenza di una coda della distribuzione intorno a 50dB per bassi valori di cardinalità del supporto nel primo metodo. Il risultato potrebbe essere dovuto ad una serie di segnali più semplici da ricostruire, come ad esempio onde P e T, che possono essere ricostruite con un numero esiguo di contributi del segnale sparsificato ξ. Un insieme di segnali del genere permette utilizzando il primo metodo di raggiungere sin da subito elevate performance in termini di RSNR per la struttura del metodo utilizzato, mentre per l'altro metodo ciò non è possibile perché il secondo metodo sfrutta gli incrementi di RSNR per definire quali elementi includere nel supporto. In Fig. 3.16 si valuta la rappresentazione della sparsità confrontando i risultati del primo e del secondo metodo. Si vorrebbe ottenere, dal punto di vista ottimo, una bisettrice del grafico per mettere in luce una dualità nei due metodi; tuttavia, il risultato non è propriamente così, ma piuttosto una bolla nel piano. Mediamente si può comunque osservare che quando il primo metodo fornisce una bassa sparsità,

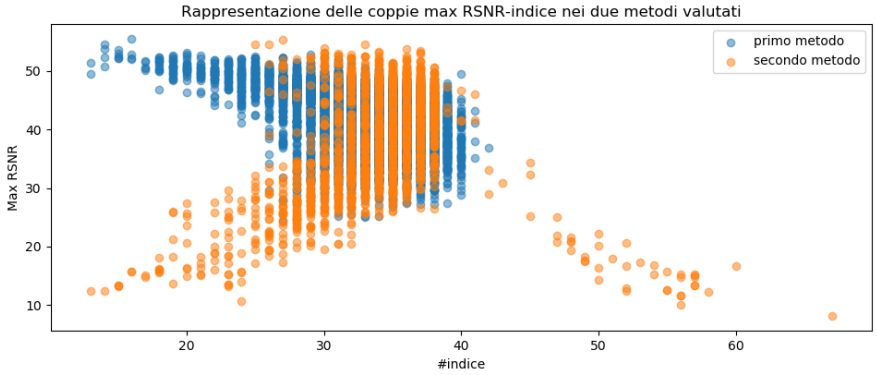


Figura 3.15: Rappresentazione tramite scatter plot delle coppie massimo valore di RSNR e indice nei due metodi valutati.

anche il secondo metodo fornisce valori bassi di sparsità. Il fatto che si ottenga una palla e non una bisettrice implica che i due metodi applicano strumenti totalmente diversi. Infine e in maniera duale, si osserva Fig. 3.17. Analizzando la distribuzione dei valori di RSNR confrontando i due metodi si possono fare alcune considerazioni. Per prima cosa si osserva un trend simile ad una bisettrice, quindi quando un metodo produce una certa performance, anche l'altro metodo, con una certa affidabilità, produce una performance simile. Secondo, analizzando alti valori di RSNR si osserva una variabilità minore, mentre per valori di RSNR fra 30 e 40dB si ha una variabilità maggiore in linea con la maggiore varianza offerta dal secondo metodo. Si può quindi dedurre che per alti valori di RSNR il comportamento sia simile, mentre per medi valori di RSNR il comportamento si differenzia.

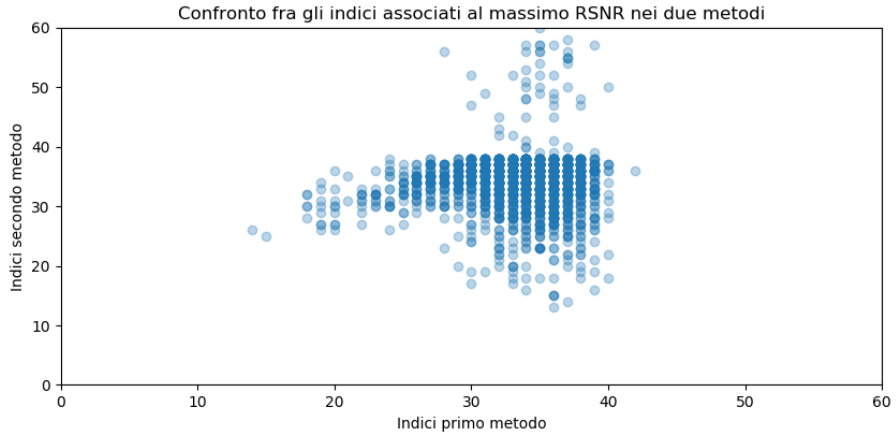


Figura 3.16: Rappresentazione tramite scatter plot della relazione fra gli indici associati al massimo RSNR nel primo e nel secondo metodo.

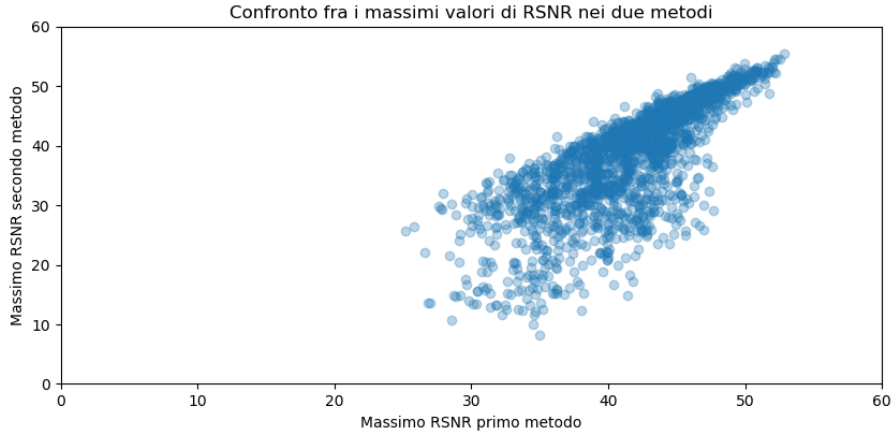


Figura 3.17: Rappresentazione tramite scatter plot della relazione fra i massimi valori RSNR nel primo e nel secondo metodo.

Conclusioni

Durante il lavoro di tesi sono stati appresi i principi cardine del Compressed Sensing e le fondamenta del deep learning per poter implementare una coppia encoder - decoder con encoder a bassa complessità computazionale in accordo con il principio del compressed sensing e sfruttando le competenze in ambito deep learning per implementare una rete neurale profonda come parte integrante del decoder. Sono state valutate diverse tipologie di encoder come la matrice di sensing a valori reali, antipodale e antipodale con parametro di scaling, osservando un delta di performance, in termini di ARSNR e PCR-5dB, e costo computazionale fra le varie soluzioni. Si ricordi che implementare una matrice di encoding antipodale non richiede l'utilizzo di operazioni di moltiplicazione poiché sarà sufficiente effettuare un'inversione di segno prima di applicare l'operatore di somma. Per quanto riguarda il delta di ARSNR si parla di circa 2.5dB nel caso rumoroso e di circa 12dB per alti valori di CR, mentre circa 2dB per bassi valori di CR nel caso noiseless. Analizzando la PCR-5dB invece il delta è contenuto fra il 5% e il 10% nei casi considerati. Dopo aver analizzato un data set di segnali ECG dal punto di vista della sparsità, variando la lunghezza del segnale e l'energia descritta, si è poi passati all'analisi del sistema rimuovendo il vincolo di sparsità fissata, ridefinendo parti dello stesso e osservando una notevole variazione delle performance rispetto ai precedenti (a tal punto che non più ha senso parlare di PCR-5dB); infatti, le performance

raggiunte a parità di encoder sono circa dimezzate raggiungendo al più circa 30dB di ARSNR nei casi migliori. Si sono infine esplorati due metodi distinti per la costruzione del supporto, ma senza l'impiego di reti neurali. In tal caso si osservano punti comuni e aspetti divergenti fra i due metodi proposti che comunque consentono di costruire un supporto del segnale di partenza in grado di raggiungere performance in linea con le aspettative, tuttavia per alcune casistiche erano attese performance più convincenti.

Futuri scenari in merito all'applicazione potrebbero essere la realizzazione fisica delle componenti progettate durante il lavoro di tesi, uno sviluppo più dettagliato dei metodi proposti per la ricerca del supporto valutando l'evoluzione del RSNR.

Appendice A

Il linguaggio di programmazione utilizzato per la realizzazione di tutte l'attività svolte è Python, un linguaggio ad oggetti ad alto livello orientato alla programmazione ad oggetti. In questo contesto, si inserisce Tensorflow, una piattaforma open-source per il machine learning utilizzata per la realizzazione del modello della DNN descritto nel Listato 3.1 di Cap. 3. Dal punto di vista prettamente pratico, Tensorflow è utilizzabile semplicemente importando una libreria all'inizio dello script che si sta valutando e consente di realizzare, fra le varie opportunità che offre, una DNN comunque complessa, semplicemente combinando layer con diverse caratteristiche. Volendo quindi descrivere più nel dettaglio la sintassi impiegata nella descrizione delle DNN si inizia dalla struttura della rete. Esistono, infatti, diversi modi per poter descrivere una DNN utilizzando Tensorflow: si può definire un modello utilizzando l'oggetto `Sequential` oppure si possono concatenare diversi livelli utilizzando l'oggetto `Model`. In entrambi i casi occorre poi definire tutti i livelli che descrivono la DNN in esame per completare la descrizione del modello. Durante la realizzazione di questo training si è preferito utilizzare la seconda tecnica per la descrizione della DNN in quanto risulta più semplice ed intuitivo la rappresentazione della rete. Si è poi scelto di descrivere i vari elementi utilizzando una serie di layer `Dense` di cui si riporta la sintassi nel Listato 3.2.

```
tf.keras.layers.Dense(
```

```
    units, activation=None, use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros', kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Listato 3.2: Struttura Layer Dense

Nonostante la vasta gamma di attributi che si possono utilizzare per definire il layer di una DNN utilizzando la sintassi del Listato 3.2, sono stati utilizzati i seguenti attributi:

- `units`: intero positivo, definisce la dimensionalità del layer che si sta definendo;
- `activation`: stringa, definisce la funzione di attivazione che si può utilizzare per determinare l'output del layer. Sono già state descritte nella Sezione 1.2 alcune delle soluzioni adottabili, mentre se non si specifica alcuna funzione di attivazione si utilizza di default la funzione lineare $a(x) = x$.
- `use_bias`: valore booleano, definisce la presenza o meno dell'elemento di bias all'interno di ogni neurone.
- `name`: stringa, definisce il nome di ciascun layer.

Si definisce fra parentesi tonde, fuori dalla funzione `Dense`, il livello da cui sono presi i neuroni per il calcolo del livello corrente, definendo così le connessioni

fra i vari layer della DNN. L'output di ogni layer è poi determinato in accordo con la teoria sulle DNN descritta nella Sezione 1.2.

Appendice B

Nella realizzazione della DNN impiegata per studiare l'architettura proposta si è inserito all'interno del modello anche la matrice dell'encoder. Quando si impiega la matrice di sensing a valori reali è sufficiente definire un layer Denso, descritto in Appendice A con la scelta opportuna dei parametri, mentre quando si sceglie di studiare il sistema con una matrice di sensing antipodale occorre definire appositamente il layer della DNN che descriva l'encoder. Il layer è definito da una classe contenente metodi per la definizione del layer, l'inizializzazione e la definizione delle proprietà, descritto nel Listato 3.3. Il layer contiene dei pesi allenabili ai quali si applica la funzione segno a valle del training in accordo con la definizione dello stesso. L'aspetto interessante per quanto riguarda l'impiego di una matrice di encoding antipodale è legata al costo computazionale dato che il processo di codifica del segnale non richiede, in questo caso, un prodotto matriciale, ma solamente un cambio di segno. Infatti, considerando una matrice di sensing A a valori reali il costo computazionale richiesto è di *mn* MUL e *m(n-1)* ADD, mentre considerando una matrice di sensing antipodale sono richieste solamente *m(n-1)* ADD. Dal punto di vista dell'occupazione di memoria, assumendo che entrambe le matrici siano memorizzate con la stessa precisione, non si evidenziano differenze, dato che entrambe sono matrici a $m \cdot n$ elementi.

```
class Antipodal(Layer):
    def __init__(self, units, **kwargs):
        self.units = units
        super(Antipodal, self).__init__(**kwargs)

    def build(self, input_shape):
        # Create a trainable weight variable for this layer.
        self.kernel = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.units),
                                      initializer='uniform',#random_normal
                                      trainable=True)
        self.type_l = 'Antipodal'
        super(Antipodal, self).build(input_shape)  # Be sure to call this
    at the end

    def call(self, x):
        return tf.keras.backend.dot(x,tf.keras.backend.sign(self.kernel))

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.units)

    def get_config(self):
        config = {
            'units': self.units,
        }
        base_config = super(Antipodal, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

Listato 3.3: Custom Layer

Appendice C

Si vuole mostrare i passi svolti per determinare il valore ottimo del parametro di scaling α da utilizzare nella definizione dell'encoder antipodale con parametro di scaling utilizzato nel Cap. 3.3.2. Si parte sapendo che il valore ottimo del parametro di scaling α è determinabile come soluzione del seguente problema di ottimizzazione:

$$\alpha^*, A^{\pm *} = \arg \min_{\alpha, A^{\pm}} L(A^{\pm}, \alpha) \tag{3.16}$$

dove $L(A^{\pm}, \alpha)$ è la funzione costo, $A^{\pm}$ è una matrice binaria composta da +1 e -1 definiti come $A^{\pm} = sign(A)$ con $A = \mathbb{R}^{m \times n}$, mentre α è il parametro di scaling che deve essere ottimizzato. In precedenza, la funzione costo è sempre stata indicata come una funzione dei pesi e dei bias $L(W, b)$, ma dato che il problema richiede di determinare la matrice antipodale e il parametro di scaling ottimi è ragionevole descrivere la funzione costo in relazione ai parametri ignoti. La funzione costo $L(A^{\pm}, \alpha)$ è formalmente definita come:

$$L(A^{\pm}, \alpha) = \|A - \alpha A^{\pm}\|^2 \tag{3.17}$$

Da cui si ottiene:

$$L(A^{\pm}, \alpha) = \alpha^2 \ (A^{\pm})^T A^{\pm} - 2\alpha A^T A^{\pm} + A^T A \tag{3.18}$$

Sapendo che $A^{\pm} \in [-1, +1]^{m \times n}$, allora si ottiene $(A^{\pm})^T A^{\pm} = m \cdot n$ e tale prodotto è una costante essendo n e m le dimensioni dell'encoder, costanti e

note a priori. Da questo segue anche che $A^T A$ è costante perché A è nota a valle della fase di training. Si ottiene:

$$L(A^{\pm}, \alpha) = -2\alpha A^T A^{\pm} + \text{cost.} \tag{3.19}$$

Allora il valore ottimo di $A^{\pm}$ è determinato dalla soluzione del seguente problema di ottimizzazione:

$$(A^{\pm})^* = \arg \min_{A^{\pm}} (-A^T A^{\pm}) \text{ s.t. } A^{\pm} \in [-1, +1]^{mxn} \tag{3.20}$$

Dal precedente problema di ottimizzazione si estrae il valore ottimo di $A^{\pm}$ sapendo come la matrice è definita, ottenendo:

$$(A^{\pm})^* = \text{sign}(A) \tag{3.21}$$

Infine, il valore ottimo di α è determinato a valle della valutazione del gradiente della funzione costo rispetto ad α e posto uguale a zero:

$$\begin{aligned} \frac{\partial L(A^{\pm}, \alpha)}{\partial \alpha} &= 2\alpha A^{\pm T} A^{\pm} - 2A^T A^{\pm} = 0 \\ \alpha^* &= \frac{1}{m \cdot n} A^T (A^{\pm})^* \end{aligned} \tag{3.22}$$

Ottenendo:

$$\alpha^* = \frac{1}{m \cdot n} A^T \text{sign}(A) = \frac{1}{m \cdot n} \sum_i |A_i| = \frac{1}{m \cdot n} \|A\|_{\ell 1} \tag{3.23}$$

Si può concludere quindi che la stima migliore della matrice a valori reali si può ottenere facendo il segno degli elementi della matrice, mentre il valore ottimo del parametro di scaling α è la media dei valori assoluti degli elementi della matrice [11].

Bibliografia

[1] M. Mangia, L. Prono, A. Marchioni, F. Pareschi, R. Rovatti, G. Setti, *Deep Neural Oracles for Short-window Optimized Compressed Sensing of Biosignals*, IEEE Transactions on Biomedical Circuits and Systems, vol. 14, no. 3, 2020.

[2] D. L. Donoho, *Compressed Sensing*, IEEE Transactions on Information Theory, vol. 52, no. 4, 2006.

[3] O. Jaspan, R. Fleysher, M. L. Lipton, *Compressed Sensing MRI: a review of the clinical literature*, British Journal of Radiology, vol. 88, no. 1056, 2015

[4] J. Schmidhuber, *Deep Learning in Neural Networks: An Overview*, Neural Networks, vol. 61, 2015

[5] Y. Bengio, *Learning Deep Architectures for AI*, Foundations and TrendsR in Machine Learning, vol. 2, no. 1, 2009

[6] Raúl Rojas, *The backpropagation algorithm of Neural Networks - A Systematic Introduction*

[7] E.J. Candès, M.B. Wakin, *An Introduction To Compressive Sampling*, IEEE Signal Processing Magazine, vol .21, no. 2, 2008

[8] F.R. Martini, M. J. Timmons, R. B. Tallisch, *Anatonomia Umana*, 2012

[9] I. Daubechies, *Ten Lectures of Wavelets*, SIAM, 1992

[10] A.C. Guyton, J.E. Hall, *Fisiologia Medica*, 12ª ed., Elsevier, 2012

[11] B. Sun, H. Feng, K. Chen and X. Zhu, *A Deep Learning Framework of Quantized Compressed Sensing for Wireless Neural Recording*, vol. 4, 2016

[12] L. Prono, M. Mangia, A. Marchioni, F. Pareschi, R. Rovatti, G. Setti, *Deep Neural Oracle with Support Identification in the Compressed Domain*, sottomesso a IEEE Journal on Emerging and Selected Topics in Circuits and Systems