UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di laurea triennale in ingegneria biomedica

IMPLEMENTAZIONE DI UNA RECURRENT NEURAL NETWORK (RNN) PER LA CLASSIFICAZIONE DI SOGGETTI SANI E AFFETTI DA ALZHEIMER TRAMITE L'ELABORAZIONE DI SEGNALI ELETTROENCEFALICI (EEG)

IMPLEMENTATION OF A RECURRENT NEURAL NETWORK FOR THE CLASSIFICATION OF ALZHEIMER'S DISEASE FROM EEG SIGNAL

Relatore: Chiar.mo
Prof. Claudio Turchetti

Correlatore: Chiar.ma
Prof.ssa Laura Falaschetti

Tesi di laurea di*:*
Daniele Pio Germano

Anno Accademico
2020-2021

Tesi dedicata a Nonno Donato, che da lassù veglia su di me, e a tutti coloro che mi sono stati vicini e hanno creduto in me in questo percorso di studi.

Indice

Lista di Figure

Lista di Tabelle

Introduzione

Negli ultimi anni sono aumentati gli ambiti disciplinari che hanno mostrato un grande interesse nei confronti dell'Intelligenza Artificiale, dovuto alle molteplici funzionalità che essa propone e alla maturità tecnologica che attualmente ha raggiunto.

L'Intelligenza Artificiale (AI) si definisce come una branca dell'informatica che copre vari aspetti dell' intelligenza umana [1]. Include le competenze legate al ragionamento ed alla risoluzione di problemi di diversa natura, proponendo lo sviluppo di componenti software dotati di capacità cognitive tipiche dell'essere umano. Questi si basano sull'utilizzo di tecniche di apprendimento che consentono di elaborare e comprendere informazioni presenti in un contesto di dati.

Uno dei tanti settori che si è aperto a questo approccio è quello medico, permettendo di fornire al personale sanitario un supporto alle diagnosi (CAD Computer-aided diagnosis) e strumenti che possano intervenire sui singoli soggetti, tramite misure adatte ad esigenze specifiche. Attraverso l'utilizzo di strumenti di misura, vengono prelevati i segnali di interesse e convertiti in set di dati, utilizzati per addestrare i modelli di intelligenza artificiale che permetteranno l'identificazione di eventuali patologie.

Il seguente lavoro espone l'analisi e la conversione in dataset di segnali elettroencefalici (EEG), prelevati utilizzando opportuni elettrodi posti sullo scalpo dei vari soggetti presi in esame.

L'obiettivo della presente tesi è quello di analizzare alcune tecniche di Deep Learning, che verranno successivamente approfondite, in modo da poter individuare l'approccio migliore per la soluzione di un problema di classificazione comprendente soggetti sani e malati di Alzheimer.

La tesi è strutturata nel seguente modo:

- Descrizione delle modalità di utilizzo del Deep Learning impiegate per la classificazione e delle tecniche di feature extraction.
- analisi dei campioni provenienti dai pazienti , dapprima in termini di forme d'onda graficate nel tempo e poi in figura matriciale.
- conversione dei file di dati, contenenti le informazioni elettroencefaliche dei soggetti, in formati più semplici da manipolare dai software.
- applicazione di algoritmi di pre-processing sul dataset per il filtraggio dai valori ridondanti.
- progettazione e realizzazione della rete neurale ricorrente (LSTM).
- valutazione dei risultati ottenuti e ottimizzazione degli hyperparameters.

CAPITOLO 1

MACHINE LEARNING E DEEP LEARNING

L'Intelligenza Artificiale non possiede una definizione univoca. Le interpretazioni posso variare in base al punto di vista con cui la si vede: da una parte ci si può concentrare sui processi interni che comportano il ragionamento in un algoritmo, da un'altra sul comportamento esterno dei sistemi dato dai risultati ottenuti nella predizione di una condizione, ruolo che, prima dell'avvento di queste tecniche, veniva affidata agli esseri umani. Una definizione comune a tutte le interpretazioni è quella che la vede come una sorta di "misura di efficacia" della somiglianza dell'algoritmo al comportamento umano, attraverso tecniche di apprendimento che permettono ad un sistema intelligente di evolvere e quindi di modificare il proprio stato in modo completamente autonomo, anche grazie alle conoscenze acquisite in precedenza.

L'intelligenza artificiale si suddivide in due sottocampi, ovvero il Machine Learning e il Deep Learning, caratterizzati da diversi modelli di apprendimento che si differenziano per la modalità di costruzione dell'algoritmo.

Il Machine Learning [2], in italiano apprendimento automatico, racchiude tutti quei sistemi capaci di imparare, di adattarsi ed ottimizzarsi nel loro ambiente di lavoro, in modo da svolgere un determinato compito in maniera automatica, senza che siano precedentemente programmati dall'uomo.

Gli algoritmi di Machine Learning, a differenza dei classici codici di programmazione in cui si dice all'elaboratore cosa fare attraverso modelli matematici predeterminati, utilizzano metodi matematico-computazionali in grado di catturare le informazioni più importanti, per poi generare una nuova conoscenza basata sull'esperienza e costruendo in modo intuitivo un modello capace di effettuare delle predizioni anche su compiti che non ha mai affrontato prima.

Il modello si adatta al problema e migliora le sue prestazioni a seguito di un aumento progressivo dei dati da cui può apprendere, quindi basterà fornirgli un set di dati e sarà lui stesso a definire una logica propria che gli permetta di svolgere il compito richiesto.

1.1 Obiettivi del Machine Learning

Il fine ultimo di utilizzo del Machine Learning riguarda la risoluzione di tre grandi problemi, tra cui il clustering (raggruppamento), regressione e classificazione.

Nello specifico :

- nella regressione [3], l'output desiderato è costituito da una o più variabili continue e si ottiene attraverso l'utilizzo di valori di input ben definiti nel loro dominio. Si usano soprattutto funzioni lineari dai parametri regolabili, che permettano di scoprire la relazione causa-effetto tra le variabili, in modo da prevedere l'andamento di una variabile sulla base degli input forniti. Il modello più semplice di regressione è quello lineare, in cui il numero di variabili indipendenti è uno (x) e sono collegate alle variabili dipendenti (y) attraverso una funzione lineare del tipo $y = a0 + a1*x$. Il lavoro svolto dall'algoritmo è quello di trovare i valori di a0 e a1 migliori che permettano di avere una previsione corretta.
- Il clustering {I} permette il raggruppamento dei dati in gruppi (cluster) nello spazio delle loro funzionalità, ovvero possono essere mappati ed etichettati sulla base di informazioni comuni per singolo cluster. I dati in questo caso non presentano etichette e l'algoritmo è di tipo non supervisionato.

- La classificazione {II} è una tecnica di machine learning che consiste nel catalogare i dati presi in esame in opportune classi, rappresentanti il sistema output (uscita) visto sotto forma di label (etichette). Per la risoluzione di tale problema l'obiettivo è quello di realizzare un modello predittivo, partendo dal set di dati acquisiti in precedenza, così da eseguire successivamente una previsione su nuovi dati forniti in seguito. Questa modellazione si basa sull'approssimazione di una funzione *f* sulla base di variabili in ingresso *X* e variabili *Y* in uscita secondo lo schema $Y = f(X)$, comportamento simile ad una funzione di trasferimento.

  La suddetta funzione è una legge che lega i dati tra loro, che viene trovata e valutata dalla macchina, in modo che sia quanto più possibile affidabile nel lavoro di previsione. Dal punto di vista della modellazione, la classificazione richiede un set di dati di training con molti esempi di input e output da cui imparare, in modo che apprenda il modo migliore per mappare i dati in ingresso in etichette di classe specifiche. Se le classi da predire sono due, si parlerà di classificazione binaria.

In generale, attraverso la rilevazione dei cosiddetti parametri stimatori quali:

- True positive (TP), True negative (TN)
- False positive (FP), False negative (FN)

l'elaboratore fornisce la misura di *accuracy* {III} (accuratezza), ovvero la metrica percentuale fondamentale che rappresenta la fedeltà del modello rispetto al risultato richiesto. Più questo valore è alto, migliore sarà la performance dell'algoritmo e la predizione di risultati corretti.

L'accuratezza è definita come il rapporto tra i dati classificati correttamente e quelli totali, ovvero $Accuracy = \frac{TP + TN}{FP+FN+TP+TN}$.

Per la creazione di un buon classificatore, oltre all'accuratezza, c'è da considerare anche la rapidità con cui viene processato il risultato, corrispondente al *training time*, e la complessità del modello, che inficia sul tempo di produzione dell'esito.

1.2 - Valutazione del modello

Un altro fattore fondamentale per la realizzazione di un classificatore è la sua buona capacità di generalizzazione del modello. Infatti, bisogna essere certi che questo effettui delle previsioni corrette su dati che non ha mai visto in precedenza. Da qui la necessità di considerare tecniche di valutazione che utilizzino un test set per valutare le prestazioni del modello, e che non presentino dati già impiegati nella creazione dello stesso, al fine di evitare che si verifichi un fenomeno indesiderato, chiamato overfitting {XVI}. Questo si presenta nel momento in cui il modello si ricorda il set di allenamento e inizia

a predire sempre l'etichetta corretta in ogni punto dell'allenamento.

In questo progetto si sono utilizzati due metodi: uno non convenzionale, attraverso il quale verrà effettuato uno splitting (divisione) del dataset per soggetti (e non in percentuale) in training set e test set, in modo tale da non avere uno split con dati di un singolo paziente in entrambi i set, l'altro attraverso la tecnica chiamata K- Fold Cross Validation, descritta in seguito.

1.3 - Modalità di apprendimento

Le informazioni che la macchina assimila nel tempo vengono utilizzate per la creazione del modello, ovvero vengono definite delle regole che saranno utili nelle elaborazioni future. Infatti, maggiore sarà l'esperienza sui dati, più esso diverrà efficiente nella risoluzione del problema.

Esistono 3 macro-metodologie di apprendimento {IV}, ossia:

- Apprendimento di rinforzo, utilizzato se il feedback della validità delle scelte è disponibile con un certo ritardo.
- Apprendimento non supervisionato, utilizzato soprattutto nel clustering, in cui l'assimilazione delle conoscenze avviene per mezzo dell'esperienza. In questo caso non vengono forniti le etichette dall'utenza esterna, in modo tale che gli ingressi vengano classificati sulla base di caratteristiche comuni tra i dati.

- Apprendimento supervisionato, utilizzato per regressione e classificazione. Consiste nel fornire al modello un insieme di dati raggruppati in un train set, responsabile dell'allenamento dell'elaboratore. Al suo interno questo set conterrà sia i valori di input, che i valori di output , ovvero le label. Da qui si deduce che l'algoritmo, dopo aver analizzato il set , conosca a priori la risposta e sia in grado di creare un modello capace di predire il risultato incognito di dati futuri, contenuti in un test set.

1.4 - Deep learning

Per anni le tecniche di machine learning convenzionali erano limitate nelle loro capacità di elaborare i dati nella loro forma grezza, richiedendo una grande esperienza nel dominio dei dati per la progettazione di un estrattore di funzionalità, che gli permettesse di trasformare i dati grezzi in una rappresentazione interna adatta, da cui il sistema potesse rilevare e classificare l'input nel modello.

Quell'insieme di metodi che permette ad una macchina di essere alimentata con dati grezzi e di scoprire in modo autonomo le rappresentazioni necessarie per la classificazione viene chiamato Representation learning, in italiano

Apprendimento delle Rappresentazioni.

Su questo concetto si basano i metodi di Deep Learning [4], in italiano allenamento profondo, ottenuti componendo moduli non lineari a cascata che, partendo dagli input grezzi, trasformano la rappresentazione in un livello sempre più alto ed astratto, portando il modello ad imparare funzioni molto complesse. Ciascun livello utilizza come input l'uscita del livello precedente, che estrae e trasforma i dati per adattarli al livello successivo.

Per il task di classificazione , l'organizzazione gerarchica in livelli più alti di rappresentazione permette di ottenere l'amplificazione delle informazioni più rilevanti da un set di dati, sopprimendo quelle irrilevanti.

Quindi l'allenamento profondo può essere definito come una classe di algoritmi di apprendimento automatico, dove possono essere usati livelli nascosti di una rete neurale insieme a formule di logica proporzionali (AND, OR, NOT…)

1.5 Reti neurali

Concettualmente una rete neurale nasce dal tentativo di replicare, attraverso un modello matematico, il funzionamento e la capacità di apprendimento del cervello umano. Può essere vista come un modello di Machine Learning con comportamento simile ad altri, ma costruita in maniera più complessa.

Difatti è possibile impiegare le stesse metriche che si utilizzano in altri modelli.

Dalla sua prima interpretazione, con la creazione del modello percettrone {V}(unità fondamentale analoga al neurone), si è arrivati via via a sviluppare modelli sempre più complessi ed efficienti.

Il percettrone, che si serve di una legge matematica per emulare il comportamento delle sinapsi, è costituito da :

- valori di uscita booleani (0 o 1) ;
- valori di ingressi X_j e pesi W_j reali positivi o negativi ;
- tre elementi, ovvero: ingressi, somma degli ingressi moltiplicati per i loro pesi e infine la funzione soglia;

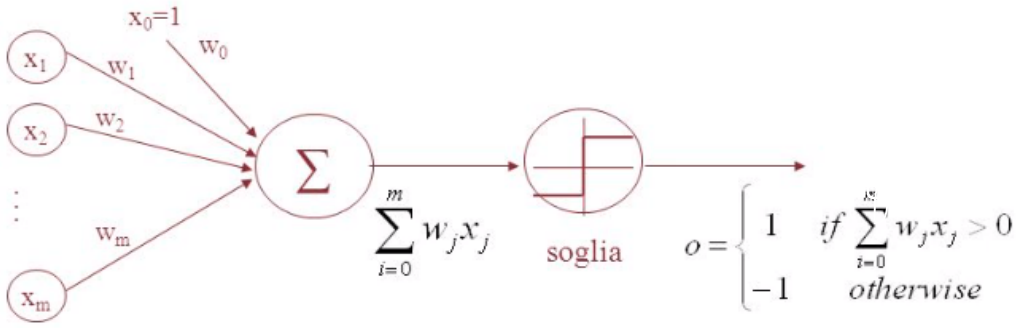

Fig.1.1 Modello di Percettrone

L'apprendimento consiste nel determinare i giusti pesi (che sono casuali) e nel selezionare la miglior funzione di soglia. Quest'ultima, preferibilmente non

lineare, permette l'attivazione del percettrone quando vengono forniti ad esso i giusti valori di segnali di input. Il problema di classificazione è quindi ridotto alla determinazione dell'insieme dei pesi migliore, con lo scopo di minimizzare gli errori di classificazione ed ottimizzare la funzione O in figura.

Il Single Layer Feedforward Network {VI}, ossia una rete neurale a singolo strato con propagazione frontale senza cicli, fu uno dei primi modelli di rete contenente diversi percettroni. Ognuno di questi neuroni elabora un'uscita come combinazione lineare degli ingressi, processata attraverso una funzione non lineare, chiamata funzione di attivazione. Tutti insieme posti in parallelo, i percettroni formano così uno stato detto layer , capace di elaborare le informazioni in maniera più profonda.

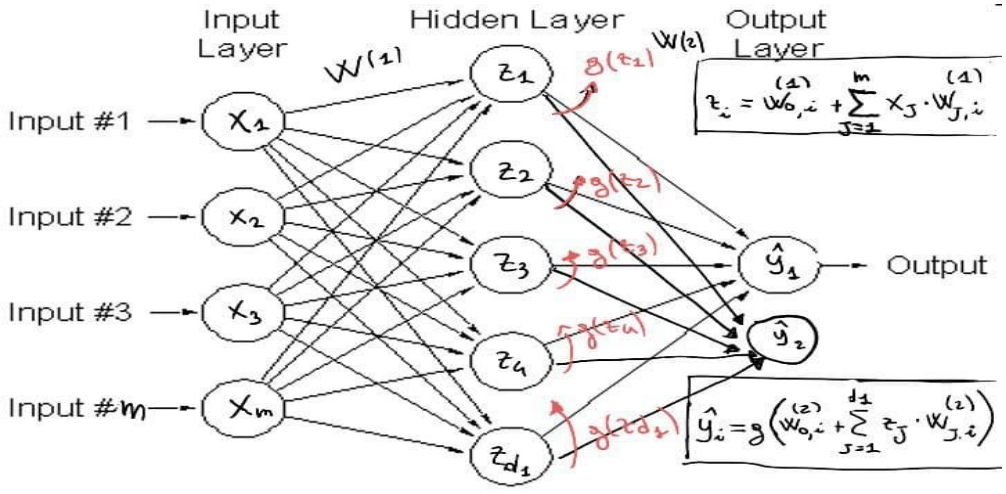

Fig. 1.2 Modello di Single Layer Feedforward Network

La struttura di questa rete comprende, oltre ad un layer di input e uno di output, un ulteriore strato anche detto nascosto, che rappresenta il vero e proprio strato di elaborazione degli ingressi. Inoltre, si osserva che tutti i livelli di input sono collegati a ogni unità di attivazione del livello successivo. Questa configurazione di livelli è detta fully connect (completamente connessi), la quale permette ad ogni ingresso di condizionare ogni percettrone e quindi influenzare in modo uguale la decisione della rete rispetto ad altri ingressi. A livello computazionale, queste due implementazioni risultano essere abbastanza dispendiose, a causa degli elevati numeri di parametri da valutare nella fase di addestramento della rete, ma, nonostante ciò, le donano una notevole efficienza e flessibilità.

1.6 - Reti Neurali Ricorrenti

Un'importante limitazione delle reti di tipo feedforward è il non riuscire a prendere buone decisioni in quei set di dati contenenti informazioni che si prolungano nel tempo. In questi casi è utile considerare una tipologia di rete neurale che si avvicini il più possibile al funzionamento del cervello umano, ovvero che sia capace di simulare le connessioni che procedono "all'indietro".

Le reti neurali ricorrenti {VII} (RNN) riescono a processare meglio le informazioni temporali, rispetto altre tipologie di reti, grazie all'utilizzo di

un'informazione aggiuntiva presente nel percettrone, ovvero lo stato, creando una rappresentazione ciclica dello stesso. Questa condizione consente di tenere traccia delle informazioni passate per poter valutare in modo corretto i dati correnti, attraverso l'utilizzo di una funzione f_w non lineare (gradino, sigmoide o tangente iperbolica) che processa una combinazione di ingressi e gli stati precedenti:

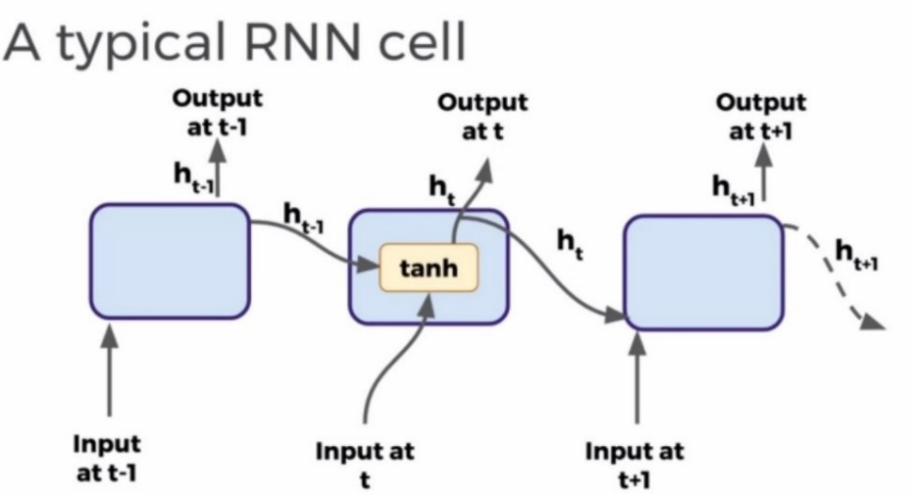

Fig. 1.3 Schema di una tipica Rete Neurale Ricorrente con $h_t = f_w\,(\,h_{t-1}\,, x_t\,)$, con h_t stato corrente, h_{t-1} stato precedente e x_t vettore di input al tempo t

Questo schema rappresenta il layer più semplice presente nella libreria Keras di Tensorflow chiamato SimpleRNN, che comprende un percettrone rappresentato nei diversi istanti di tempo, passato, presente e futuro.

Questa tipologia di configurazione di apprendimento presenta alcuni problemi, tra cui:

- aggiornamento dei pesi anche sulla linea temporale, oltre che sulla classica linea feedforward caratteristica di altri tipi di modelli. Ciò provoca un aumento di difficoltà nella valutazione dell'errore nella discesa del gradiente, quindi risulterà più complicata la fase di allenamento;
- non fa distinzione , all'interno dei dati, tra informazioni più o meno rilevanti. Questo problema può confondere lo stato futuro, a causa di una memorizzazione di caratteristiche non utili alla previsione;
- nel caso di algoritmi di apprendimento basati sul gradient backpropagation, cioè quando nella fase di apprendimento i gradienti dell'errore si propagano all'indietro fino al layer iniziale , si possono creare due condizioni dette vanishing/exploding gradient {VIII}. Nella prima, i pesi convergono a zero e la rete , nella fase di predizione, ricorda solo gli eventi più recenti, mentre nella seconda il valore dei pesi diverge a infinito e non si avrà memoria passata, con valori di gradiente indefiniti.

Un'architettura di rete più complessa che riesce a bypassare questi inconvenienti, appartenete alla stessa famiglia delle reti ricorrenti, è la cosiddetta Long Short – Term Memory (LSTM) [5] tradotto dall'inglese in memoria a breve termine. Essa utilizza il concetto di gate per migliorare l'elaborazione dei dati durante la propagazione del gradiente a ritroso e il calcolo dello stato, permettendo così di memorizzare l'informazione per lunghi periodi.

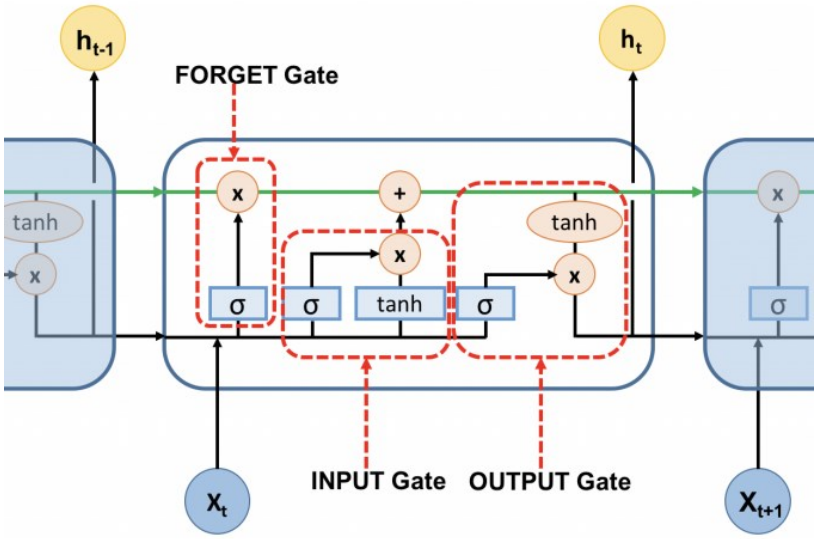

Fig. 1.4 Schema logico di una Rete LSTM con: INPUT, FORGET, OUTPUT gate e stato di cella; funzioni di attivazione tanh e sigmoide σ.

Questo layer è composto da quattro elementi :

- stato di cella, informazione che può essere modificata dai gate(linea verde) ;
- forget gate, l'informazione passa attraverso la funzione sigmoide, con valori di uscita compresi tra 0, che corrisponde a "dimentica l'informazione" , o 1 , "mantienila";
- input gate, ovvero la struttura che aggiorna lo stato di cella, comprende anch'essa una funzione sigmoide, ma posta in combinazione con la funzione tanh. Quest'ultima, cambia i valori di ingresso in un range tra -1 e 1, che verranno moltiplicati poi con l'output della funzione sigmoide , che deciderà quali saranno le informazioni da mantenere dall'output tanh;
- output gate, elemento che decide quale deve essere il prossimo stato nascosto, ricordando gli input precedenti alle celle successive. Lo stato nascosto precedente e l'input corrente passano in una funzione sigmoide, il cui output, corrispondente allo stato di cella modificato, passa per la funzione tanh. Tale output viene moltiplicato con l'output sigmoideo, così da scegliere quale informazione dovrà contenere lo stato nascosto. Infine, il nuovo stato nascosto e quello di cella vengono portati al passaggio di tempo successivo.

1.7 - Feature extraction: RPCA (Robust Principal component analysis)

Una problematica spesso presente nei grandi dataset è l'esistenza di componenti ridonanti, ovvero valori uguali chiamati outliners, che rappresentano una vasta porzione della totalità dei dati e che possono influenzare negativamente il risultato finale delle sperimentazioni, falsandolo. Un caso molto comune di questi tipi di dataset è la presenza nei segnali di componenti rappresentanti la taratura dello strumento. Questi vengono visualizzati come onde quadre ad inizio e fine del segnale. Non esiste un significato matematico del termine outliers, ma si è pensato di trattare questo problema come una corruzione additiva, in cui i valori errati si verificano più raramente rispetto ai valori reali.

Date queste considerazioni, si è deciso di creata una modifica della procedura statistica di Analisi delle Componenti Principali (PCA), chiamata Robust principal component analysis (RPCA)[6][7]. Essa mira a recuperare dai dati originali D una matrice di basso rango L, costituita dalle componenti principali, assieme ad una matrice S detta matrice sparsa, il tutto legato dalla relazione $D = S + L$.

Questa scomposizione viene ottenuta con il metodo PCP (Principal Component Pursuit Metod) in cui $\| L \|* + \lambda \| S \|$ dove

$\| L \|*$ rappresenta la nuclear norm di L (somma dei valori singolari) $\| L \|* = traccia(\sqrt{L^* * L}) = \sum_{i=1}^{\min\{m,n\}} \sigma_i (L)$ con L^* che denota la trasposta coniugata di L , matrice di dimensione m x n.

Questa tecnica, basata sull'algoritmo di discesa del gradiente, offre una garanzia teorica di recupero della matrice L, migliorando accuratezza e tasso di convergenza, preservando le informazioni statiche più importante presenti nei dati originali.

1.8 K - Fold Cross Validation

La precisione di test fornisce una stima elevata di varianza, poiché il cambiamento delle osservazioni che si trovano nel test set può modificare significativamente la precisione dei test nel modello. La tecnica chiamata *k-Fold Cross Validation* {XI} assicura l'imparzialità delle divisioni del set di dati in training e test set.

In questa tecnica il set di dati viene diviso in K partizioni di uguali dimensioni, chiamate pieghe, dove le K-1 finestre vengono utilizzate per l'allenamento, mentre l'unica partizione rimanente viene associata alla fase di test del

modello. Il processo viene ripetuto K volte, cambiando ad ogni iterazione la finestra di test utilizzata. Una volta ottenuti i valori di accuratezza per ogni iterazione, si effettua una loro media. Sebbene questa tecnica produca stime migliori, inevitabilmente aumenterà i tempi di esecuzione dell'algoritmo, visto che il processo di training sullo stesso modello viene effettuato K volte.

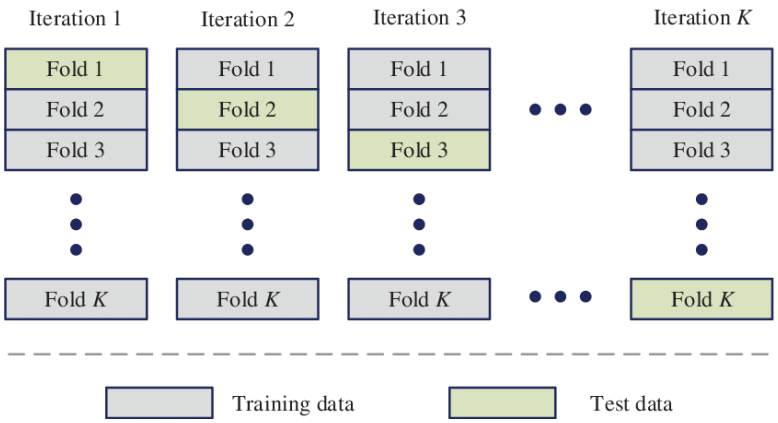

Fig. 1.5 Schematizzazione della tecnica di K-Fold Cross Validation.

CAPITOLO 2

Dataset

Nel presente progetto sono stati impiegati dei segnali elettroencefalografici (EEG)[11], ovvero dei segnali biomedici che rappresentano una misura nel tempo dei biopotenziali, prodotti dalle correnti generate dai neuroni e dalle loro interconnessioni attraverso le sinapsi. Queste correnti creano dei campi magnetici ed elettrici, che si propagano attraverso il tessuto connettivale fino ad arrivare al cuoio capelluto. Tali correnti sono captate da elettrodi in Ag - AgCl su cui viene applicato un gel conduttore, che favorisce la diminuzione d'impedenza, migliorando la trasmissione del segnale. L'acquisizione della misura è di tipo bipolare, ovvero viene prelevato in coppia un potenziale dalla zona d'interesse e uno da una zona di riferimento, che coincide con uno dei lobi delle orecchie.

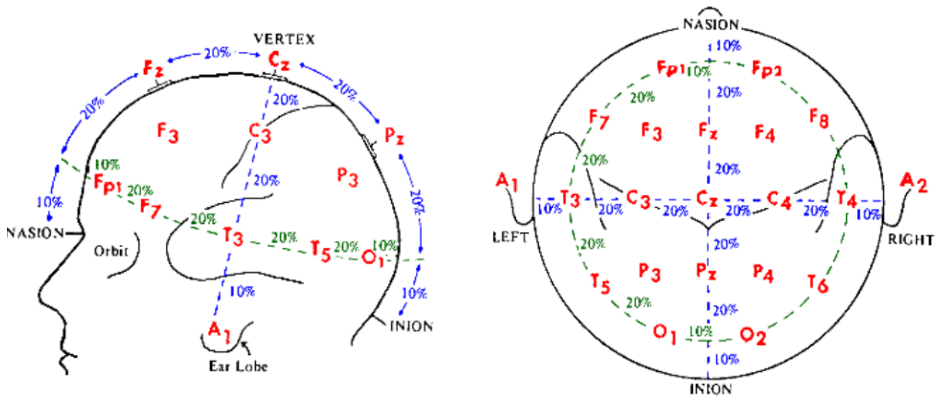

Fig. 2.1 Schema a 21 elettrodi sul piano sagitale (sinistra) e trasverso (destra)

Da questi segnali ci si aspettano valori abbastanza piccoli, dato che a riposo il potenziale d'azione di un neurone sappiamo essere corrispondente a circa -70 mV (milliVolt), sino ad aumentare a un valore cosiddetto di soglia, dove lo stimolo genera un potenziale d'azione, responsabile della trasmissione dell'impulso elettrico attraverso il neurone e le sinapsi.

La frequenza del segnale è centrata in una finestra compresa tra i 0.5 – 100 Hz, a sua volta suddivisa in cinque sottobande, corrispondenti ad un'attività celebrale distinta. Alle basse frequenze (alte ampiezze) si associa uno stato di rilassamento , alle alte frequenze(minore ampiezza) invece uno stato di attività.

Le onde più importanti si distinguono in :

- alfa , che identificano uno stato di rilassamento;
- gamma e beta, riscontrate nei casi in cui la mente è concentrata ed attiva;
- delta, presenti in uno stato patologico;
- teta, caratteristiche nella fase di sonno profondo.

2.1 - Analisi dati e Conversione File in formato mat

I segnali EEG, trattati in questo lavoro, riguardano 4 soggetti sani e 7 pazienti affetti dalla malattia neurodegenerativa denominata morbo di Alzheimer, per un totale di 11 pazienti. Questi segnali sono stati acquisiti in forma anonima, mediante un numero variabile di elettrodi ad una frequenza di 128 Hz , ovvero un campione ogni 7,8 millisecondi, ottenendo così dalle 21 alle 23 tracce registrate sottoforma di dati grezzi in formato file del tipo EDF (European Data Format), suddivisi in due cartelle denominate in questo modo :

soggetti malati → AD (Alzheimer disease) ;

soggetti sani → HD (Healthy subject) .

Per un'analisi qualitativa accurata, è stato utilizzato un software specifico per la visualizzazione dei tracciati aventi questo tipo di estensione file, chiamato Edfbrowser, un toolbox gratuito ed open source per il tracciamento di segnali biomedici nel tempo.

È stato così possibile raffigurare, in un'unica finestra di lavoro, tutte le tracce di ogni paziente della durata media di 24 minuti, in cui sono state adattate la scala temporale a tutto l'intervallo di registrazione e l'ampiezza alla dimensione della finestra.

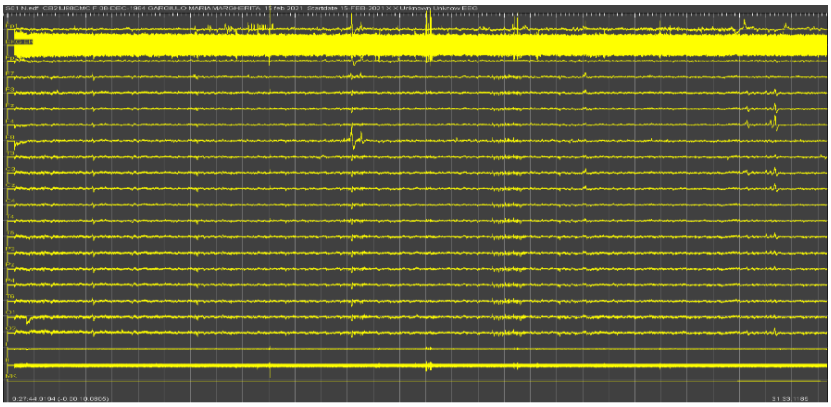

Fig. 2.2 Tracciato EEG paziente sano ↑

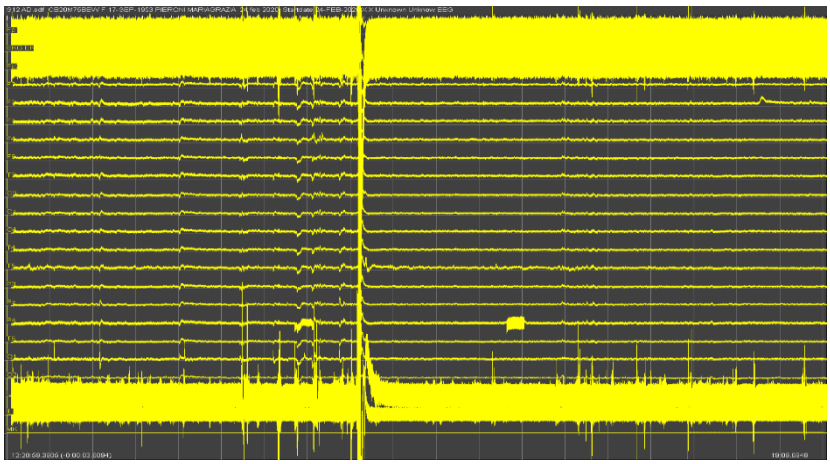

Fig. 2.3 Tracciato EEG di un paziente con il morbo di Alzheimer ↑

Per l'agevolazione del processo di data mining e per le operazioni matriciali si è scelto di convertire i file dal formato EDF a tipo MAT, estensione file utilizzata dal software di calcolo scientifico MATLAB. Questo software verrà utilizzato anche per la creazione della matrice di dati data_matrix e per l'applicazione della tecnica di feature extraction RPCA.

2.2 - Data Matrix

A causa della differente dimensione delle matrici degli undici pazienti in termini di colonne, dovuto all'elevata variabilità dei tempi di registrazione dei segnali, non è stato possibile eseguire una concatenazione tra le stesse in maniera immediata. Per ovviare a questo inconveniente ed avere una migliore uniformità dei dati , è stato necessario determinare un valore minimo comune a tutte le matrici, in modo tale da troncarle tutte allo stesso valore ed avere un numero equo di colonne.

Un'altra operazione svolta durante la fase di concatenazione delle matrici è stata quella di aggiungere due colonne ad ogni matrice, così da poter identificare la classe di appartenenza del soggetto (HS o AD) ed il numero di paziente. Questo passaggio fondamentale è stato implementato perché, costruendo una rete neurale con allenamento supervisionato, nella fase di train il modello deve conoscere a priori la classe, visto che il task da svolgere è l'identificazione della label.

Nello specifico, le label per la classe di appartenenza del soggetto sono state definite da:

- 0 → Label per la classe AD
- 1 → Label per la classe HS

Mentre la label per identificare il paziente è stata utilizzata nel codice per lo split tra train e test set, considerando in serie pazienti malati e sani. La matrice così creata avrà dimensioni 251x139264 , dove 251 rappresenta il valore corrispondente a tutte le tracce dei 11 pazienti (righe) e 139264 al numero di campioni (colonne).

2.3 - Applicazione RPCA

Dall'analisi dei dati ci si è resi conto che una buona parte del segnale di ogni paziente presenta valori ridondanti, caratterizzati dalla sezione di testa e coda del segnale. Questi valori sono eterogenei, cioè differiscono tra ogni soggetto, perciò è risultato necessario utilizzare un procedimento automatico, e non manuale, per estrapolare dalla matrice solo le informazioni rilevanti.

La tecnica utilizzata e precedentemente sopracitata è la Robust PCA, che procede all'esclusione automatizzata degli outliers, permettendo quindi di ricavare dei risultati che siano inequivocabili.

Nel codice l'argomento di input è la matrice "data_matrix", che verrà divisa in matrice di train e di test, a cui verrà applicata la RPCA ed entrambe e successivamente verranno salvate in formato .mat. Lo split è stato eseguito prendendo per il train set 8 pazienti (dal terzo al decimo) e per il test set i restanti tre pazienti (1, 2, 11).

A livello di codice, la funzione Robust PCA è composta da alcuni argomenti {IX}{X}, che in ordine rappresentano :

- Train_mat_rpca_noLabel, ovvero la matrice di train privata delle label, che rappresenta la matrice D, cioè una matrice rettangolare di dimensioni N x M che verrà decomposta in una componente a basso rango (L) e una di tipo sparso (S) → D= L + S;
- Lambda, attribuibile al parametro del problema convesso $\| L \|* + \lambda \| S \|$, di default corrisponde a $1/(\sqrt{(max(N,M))}$
- μ (mu), parametro riguardante il moltiplicatore di Lagrange aumentato, di default μ=10 x lambda;
- tol = 1e^ (-9), corrisponde al valore di tolleranza dell'errore in fase di ricostruzione, di default corrisponde a 1e^(-6);
- max_iter = 2500, ossia il numero massimo di iterazioni, di default sono 1000, parametro che aumenta di molto il tempo computazionale dell'algoritmo.

Il fine ultimo dell'applicazione dell'algoritmo RPCA è quindi quello di ridurre al più possibile il rango della matrice L risultante. Per fare ciò sono state effettuate diverse prove, variando gli argomenti μ, tol e max_iter. Alla fine, si è passati da un rango iniziale di 177 della matrice di train a un rango di 95 della matrice L, mentre per quella di test si è attivati da un rango di 66 a 35,

impostando i valori sopra citati.

Infine, dopo la creazione di una variabile chiamata LRPCA, contente i valori di L uniti alle label della matrice di train e test originale, si è salvato il tutto in un file .mat che costituirà la nuova matrice di train, rinominata in "train_L_10-lambda_1e-9_iteration_2500_RPCA" per denotare quali parametri sono stati usati, stessa cosa per la matrice di test rinominata "test_L_10-lambda_1e-9_iteration_2500_RPCA".

2.4 - Conversione file in csv

Per la realizzazione della rete neurale si è scelto di utilizzare il linguaggio di programmazione Python, essendo questo molto versatile, flessibile e ricco di librerie, sia per la manipolazione di dati che per la gestione delle reti neurali. Inoltre, per l'elevato costo computazionale dei processi di classificazione, si è optato per un lavoro su server. A tal proposito, si è deciso di effettuare un'ulteriore conversione dei file in una estensione opportuna, che potesse semplificare i processi di analisi e modellazione dei dati. Si è scelto il formato CSV (comma – separeted value), ovvero un particolare formato testo dove i valori sono separati da virgole. Per la stesura del codice si è optato per l'utilizzo di Jupyter Notebook, un'applicazione web open source, utilizzata per creare documenti contenenti codice live, grafici, testo ed altro, in grado di eseguire la presentazione interattiva di progetti di data science.

Capitolo 3

Progettazione Rete neurale ricorrente

3.1 - Architettura della rete

Dopo aver convertito il dataset con cui addestrare la rete, si è proceduto alla progettazione della sua architettura, cercando di costruire un modello capace di riconoscere e distinguere un paziente affetto da Alzheimer da uno sano. Il cuore dell'architettura della seguente RNN consiste nell'utilizzo di due layer LSTM, i quali prelevano come input una finestra di campioni e la interpretano come una sequenza di valori nel tempo, attraverso la propagazione dello stato interno. Inizialmente i dati vengono normalizzati attraverso l'uso di un layer di Batch Normalization [9], il quale mantiene in output la media a 0 e la deviazione standard vicina ad 1, per poi passare ad un layer di dropout, che in ogni epoca disattiva un numero predefinito di neuroni casuali (in questo caso la metà del totale), in modo da migliorare la predizione. I valori in uscita rappresentano l'input del layer LSTM.

Una volta processati i dati dalle LSTM essi vengono nuovamente normalizzati dai layer di Batch Normalization e Dropout, infine, vengono elaborati da un layer di Fully Connect, il quale restituisce in uscita, attraverso una funzione di

attivazione Softmax, i possibili valori di classe.

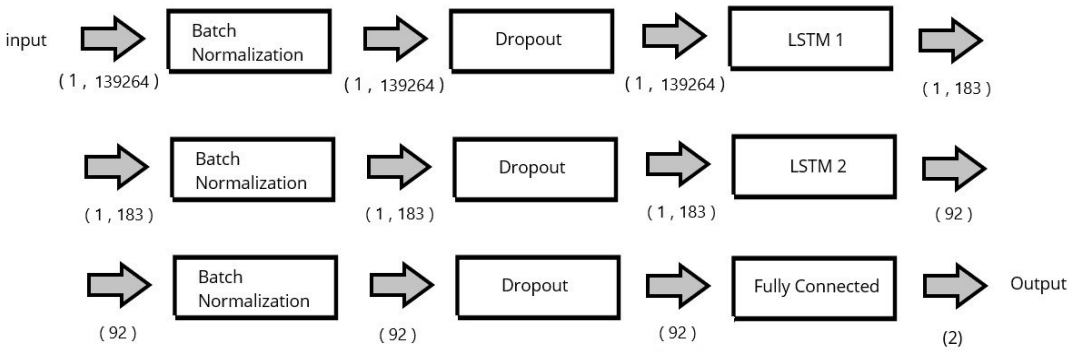

Fig. 3.1 Schema di rete utilizzato nella seguente Tesi.

La figura mostra come il modello gerarchico della rete sia capace di ridurre la dimensionalità del problema, portandola da una matrice tridimensionale a un valore unidimensionale. I valori 183 e 92 si riferiscono al numero di neuroni utilizzati nel primo e nel secondo layer LSTM.

3.2 – Implementazione ed addestramento

Per l'implementazione della rete, lavorando in ambiente Python, si è deciso di utilizzare le librerie Keras e Tensorflow , create appositamente per semplificare la costruzione di reti neurali. Inoltre, vista la complessità dei calcoli che un normale elaboratore è chiamato a svolgere, per ottenere una migliore potenza di calcolo si è sfruttato l'ambiente di programmazione cloud Google Colab, il

quale permette di scrivere codice sorgente su un foglio di testo su browser ed eseguirlo direttamente su server Google.

Per la fase di allenamento si è usato il file rinominato "train_L_10-lambda_1e-9_iteration_2500_RPCA.csv" , contenente la matrice di train di dimensione 183 x139264, mentre per il test set il file di nome "test_L_10-lambda_1e-9_iteration_2500_RPCA.csv" con matrice di dimensione 68x139264 .

Un problema che si è presentato durante la creazione della rete è stato quello della dimensionalità delle matrici supportata dal layer di input di Keras {XIX}. Infatti, questo lavora con matrici a tre dimensioni della forma :

[Sample(righe), Time steps, Features(colonne)]

dove l'argomento time steps, ovvero le osservazioni passate per caratteristica, viene impostato al valore 1 a causa della mancata finestratura del segnale.
Quindi, dopo questa trasformazione, le matrici avranno la seguente dimensione :

- train set → 183 x 1 x 139264
- test set → 68 x 1 x 139264

Nei layer LSTM, utilizzando come funzione di attivazione la tangente iperbolica, si è dovuti procedere verso un ridimensionamento dei dati, ovvero

una trasformazione che converte tutti i dati presenti nel dataset in valori compresi tra -1 e 1, corrispondente al valore minimo e massimo del set.

Il codice finale è composto da diversi blocchi, con le relative attività :

- caricamento delle librerie di Python utili ;
- rilevamento e caricamento dei file di estensione .csv in modo automatico ;
- prelievo dalle matrici di train e test dei valori di input (x) e delle label (y);
- normalizzazione dei dati di input ;
- cambio di dimensione delle matrici x e y ;
- individuazione numero di label e rimescolamento dei dati, per migliorare l'accuratezza nella fase di allenamento;
- creazione del modello e compilazione della rete ;

3.3 Ottimizzazione del modello

Come già esposto nel presente elaborato, lo scopo principale del progetto è stato quello di trovare un modello di rete neurale adatto al fine di permettere la classificazione dei segnali provenienti da pazienti sani e affetti da morbo di Alzheimer.

Per riuscire ad arrivare alla definizione di un buon modello, si è tenuto conto di due metriche fondamentali utilizzate in questo tipo di problemi: l'accuratezza e l'errore. La prima, come già precedentemente discusso, rappresenta il rapporto fra valori predetti in modo corretto e il numero totale dei campioni, e il suo valore deve essere il più alto possibile. La seconda rappresenta lo scostamento del valore predetto da quello reale, quindi questo valore deve necessariamente essere molto basso.

Per permettere un'ottimizzazione delle prestazioni del modello della rete, rispetto le metriche e tempistiche, sono stati utilizzati vari stratagemmi {XVIII} che vanno ad agire su diversi fattori, anche se questi non sempre portano al risultato desiderato, quindi si è proceduto per tentativi.

Di seguito vengono illustrati i metodi utilizzati per l'ottimizzazione del modello, che vanno a lavorare:

sui *dati* → le performance aumentano all'aumentare del numero di dati, quindi un dataset corposo potrebbe aumentarne l'accuratezza della rete. Questo deve essere opportunamente ridimensionato, normalizzato {XV} (da 0 a 1) o standardizzato, in base ai limiti imposti dalla funzione di attivazione. Nel presente caso è stata utilizzata la tangente iperbolica, con ridimensionamento da -1 a 1. Un'ulteriore trasformazione utilizzata sul set di dati è stata quella di

apportare un loro rimescolamento, sulla base dello scambio di indice delle righe, in modo tale che la rete non segua degli esempi impilati gli uni agli altri molto simili tra loro.

Dal momento che le reti neurali eseguono automaticamente l'apprendimento delle funzionalità più importanti, se gli viene esposta meglio la struttura del problema saranno in grado di risolverlo più rapidamente. Per questo motivo si è proceduti alla preelaborazione dei dati, utilizzando la RPCA, che ha proceduto all'eliminazione dei valori ridondanti.

sull'*algoritmo* → un fattore importante da considerare, che aiuta a migliorare l'accuratezza, ma che inficia sul tempo di elaborazione, è il tasso di apprendimento {XVII}. Infatti, è necessario un compromesso, se si vogliono ottenere buoni valori di accuratezza a discapito del tempo di esecuzione nella fase di training. Da qui sono stati valutati tre tassi di apprendimento, ossia 0.1, 0.01 e 0.001, tutti provati attraverso l'ottimizzatore Adam [10]. Dato che valori troppo piccoli possono aumentare in modo considerevole il tempo di allenamento della rete, dai tassi di apprendimento esaminati è emerso che il valore che ha contribuito ad aumentare il livello di accuratezza avendo anche un discreto tempo di esecuzione corrisponde allo 0.01.

Per quanto riguarda la topologia della rete, si è tentato di migliorare il modello variando in modo differente il numero di neuroni nello strato nascosto, ma

dato che il risultato non sembrava cambiare di molto, si è optato per l'utilizzo di una rete con un numero di neuroni pari a 183 nel primo layer LSTM e 92 nel secondo. Inoltre, l'assunzione di più livelli LSTM ha consentito una maggiore opportunità di ricomposizione gerarchica delle funzionalità astratte apprese dai dati, riducendo così ad ogni passaggio di layer LSTM la dimensionalità del problema.

Una soluzione risultata fondamentale è rappresentata dalla definizione di due componenti: la dimensione del batch e il numero di epoche. Il primo fattore rappresenta la frequenza con cui la rete aggiorna i pesi, mentre il secondo equivale all'intero set di dati di training esposto alla rete.

Per la batch size si è provato dapprima a utilizzare un valore uguale a quella dei dati di training, per poi man mano ridurlo, fino ad arrivare a 8192 , equivalente a 139264/17 , che ha portato a buoni risultati in termini di tempo di elaborazione di ogni epoca. Per quanto riguarda il numero di epoche, è stato evidenziato che un valore troppo elevato non portava a significativi cambiamenti, quindi nell'ultima prova effettuata si è optato per l'utilizzo di 70 epoche. Per arrivare a questo valore si è utilizzato un modulo di Tensorflow, chiamato EarlyStopping che, data come metrica di confronto l'accuratezza, ferma automaticamente l'allenamento della rete quando rileva che questa non cambia dopo un certo numero di epoche.

CAPITOLO 4

Risultati sperimentali

Per valutare l'andamento delle metriche ad ogni prova, si è utilizzato un modulo chiamato Tensorboard, che ad ogni esecuzione di codice, costruisce dei grafici dove sulle ascisse sono presenti i numeri di epoche, mentre, sulle ordinate i valori di accuratezza e di loss. Inoltre, il modulo summary stampa a schermo il modello della rete, mentre il modulo fit restituisce i valori delle metriche e il tempo di esecuzione di ogni epoca. I valori non sono sempre identicamente uguali, a causa della natura randomica dei pesi. Basti pensare che in molte prove l'accuratezza dell'allenamento nella prima epoca si aggirava in un range del 40 - 55%. Inoltre l'andamento crescente dei valori di accuratezza e decrescente dei valori di perdita sono rimasti invariati.

Come detto precedentemente nella sezione riguardante l'ottimizzazione del modello, non tutti gli stratagemmi applicati hanno permesso di modificare in modo considerevole l'accuratezza nella fase di test del modello.

Sono state effettuate due tipi di prove, entrambe utilizzanti come input per la rete le matrici di train e test processate attraverso la tecnica di feature

extraction RPCA. La prima con solo la rete, la seconda applicando la tecnica di K – Fold Cross Validation.

4.1 Prova con solo la rete LSTM

La tecnica di RPCA ha permesso alla rete di identificare più velocemente le informazioni rilevanti. Si è notato come la differenza tra l'utilizzo o meno della tecnica non incideva sull'accuratezza del modello, ma riduceva il tempo di allenamento di ogni epoca, diminuendo così il tempo di esecuzione dell'intero codice.

	Accuratezza Media train	Perdita Media train	Accuratezza test	Perdita test	Tempo di Allenamento
No RPCA	**94,97 % ± 6,25 %**	**0,11**	**64,71%**	**1,38**	**15 min 29 s**
RPCA	**94,72 % ± 7,88 %**	**0,13**	**64,71%**	**1,27**	**10 min 58 s**

Nella seguente tabella vengono riportati:

- i tempi di esecuzione della porzione di codice inerente al modulo fit della rete, addetto alla fase di allenamento;

- Accuratezza e perdita media nella fase di train, visto che ad ogni epoca viene fornito un valore di accuratezza e di perdita. Vengono fornite le deviazioni standard perché, impostando la rete con un numero di 70 epoche, non tutti i valori sono stati uguali ad ogni epoca.
- Accuratezza e perdita nella fase di test.

I valori riportati in tabella si riferiscono all'ultima esecuzione effettuata prima della stesura di questa tesi. Infatti, a causa della natura randomica dei pesi e al rimescolamento dei dati, le metriche saranno diverse, a meno che non si utilizzi un seed, ovvero un numero casuale che identifica la stessa procedura anche in esecuzioni diverse.

Dai grafici ricavati da Tensorboard, è stato effettuato un processo di smooting (levigazione) attraverso l'utilizzo della media mobile esponenziale, che ha permesso di appiattire le fluttuazioni a breve termine per evidenziare cicli a lungo termine. Data una serie di numeri e una finestra di dimensione fissa del sottoinsieme, il primo elemento della media mobile si ottiene prendendo la media del sottoinsieme iniziale, successivamente il sottoinsieme si sposta in avanti, escludendo il primo numero della serie e includendo il valore successivo. Nei seguenti grafici la linea più chiara rappresenta l'andamento reale, quella più scura indica lo smoothing dei valori (di 0.9)

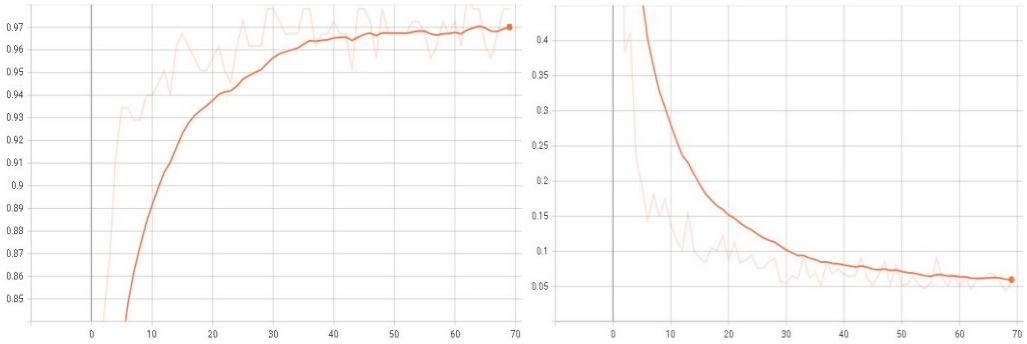

Fig. 4.1 Andamento accuratezza (sinistra) e perdita (destra) di allenamento in funzione del numero di epoche, senza Rpca

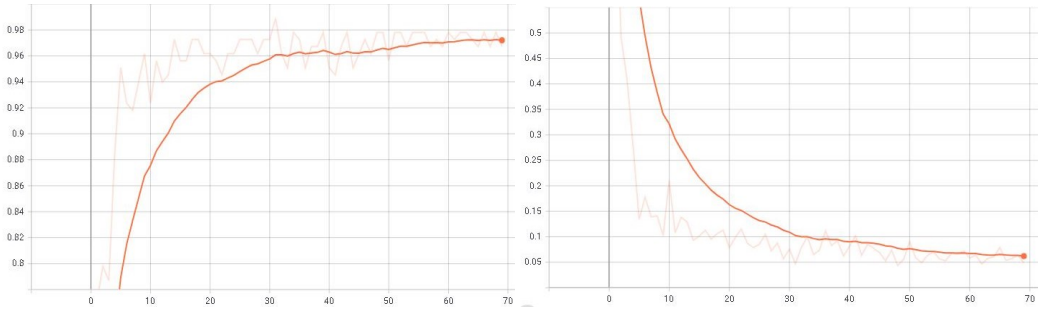

Fig. 4.2 Andamento accuratezza (sinistra) e perdita (destra) di allenamento in funzione del numero di epoche, con Rpca

Per quanto riguarda la fase di test, con l'utilizzo del metodo evaluate, sono state caricate le matrici di input (x_test) e di label (y_test), valutando il test della rete con un batch size di 9256 campioni.

4.2 Prova con K – Fold Cross Validation

Questa tecnica è stata implementata per aumentare l'accuratezza nel test e per avere un'imparzialità sulla valutazione del modello. Si è mantenuta la stessa architettura di rete, cambiando soltanto il numero di epoche per iterazione, passando a 100 ed impostando diciassette pieghe.

In questa prova non si è potuto utilizzare il modulo Tensorboard, ma si è riusciti comunque a graficare, per ogni iterazione, i classici grafici di accuratezza e perdita.

	Accuratezza Media train	Perdita Media train	Accuratezza test	Perdita test	Tempo di Allenamento
K Fold Cross Validation	**76,07 % ±2,3 %**	**0,49**	**65,77 % ±11,53%**	**0,59**	**2 h 39 min**

In questo caso l'accuratezza e perdita nella fase di training sono il risultato di due medie: la prima sulle metriche ottenute in ogni epoca per tutte le iterazioni, la seconda invece si ricava dalla media dei valori di accuratezza e perdita di tutte le iterazioni. Dalla tabella si nota come, rispetto al primo tipo di

prova, il tempo di allenamento è aumentato notevolmente, a causa della valutazione delle 17 pieghe. Inoltre si nota come nella fase di test non si è ottenuto l'aumento del risultato sperato.

In entrambe i tipi di prove si è ipotizzato che il problema fosse legato alla topologia di rete, a quel punto sono stati effettuati dei cambiamenti al numero di neuroni presenti in un singolo layer, alla struttura della rete, alle funzioni d'attivazione nei layer LSTM e all'ottimizzatore. Poi si è provato a variare il batch_size e il tasso di apprendimento, ma in tutti i cambiamenti effettuati l'accuratezza non varia.

Date tali evidenze si ritiene che il problema sia direttamente associato al dataset, alla sua effettiva divisione in training/test set e alle tecniche di pre-processing dei dati.

Due possibili soluzioni possono essere:

- l'utilizzo di dataset più grandi, in modo da avere anche un test set di maggiori dimensioni e soprattutto con un maggior range di casistiche differenti;
- l'utilizzo di tecniche di pre-processing dei dati [8] sulla base di informazioni spettrali del segnale, attraverso il filtraggio con filtro passa-basso, tecniche di finestramento, utilizzando la Trasformata Veloce di Fourier o il sottocampionamento.

CAPITOLO 5

Conclusioni

L'idea iniziale del progetto di tesi proponeva la determinazione delle prestazioni di tecniche di Machine Learning per la classificazione di soggetti affetti da patologie neurodegenerative, attraverso l'utilizzo di segnali biomedici di diversa natura.

L'unica patologia che si è riusciti a prendere in considerazione è stata il morbo di Alzheimer. Dopo aver ottenuto i dati in forma "grezza", è stata effettuata una conversione in formato .mat, per poi concatenare le matrici in un unico dataset, diviso successivamente in matrici di allenamento e test. Dopodiché, attraverso l'algoritmo di RPCA, si è provveduto ad eliminare i valori ridondanti, che avrebbero potuto contaminare la componente di dati rilevanti per il problema, determinando interpretazioni fuorvianti per l'intero dataset. In seguito ad un'ulteriore conversione dei file in formato csv, si è proceduti alla costruzione dell'architettura della rete neurale ricorrente e alla sua valutazione. Successivamente è stata applicata la tecnica di K-Fold Cross Validation.

Al fronte delle varie sperimentazioni, si evidenzia come la modellazione di una rete neurale, seppur molto affidabile ed efficiente, non è una procedura

standardizzabile ed applicabile a tutti i problemi. Il risultato finale è infatti soggetto a molte modifiche di strutture di rete e parametri, che le permettono di predire in modo corretto o meno i valori utili. Tutto questo mette in evidenza come un giusto metodo di validazione del modello sia necessario per ridurre al minimo, in prossimità di una diagnosi precoce, l'errore di valutazione del modello.

L'implementazione di una rete neurale per questo problema di classificazione può considerarsi come un'indagine esplorativa, a causa della scarsa presenza in letteratura scientifica, di materiale simile per quanto riguarda l'implementazione di reti neurali su segnali EEG di pazienti affetti dal morbo di Alzheimer.

Concludendo, nonostante la parziale completezza del progetto finale, è possibile affermare che sono state raggiunte le competenze e le abilità informatiche in ambito di elaborazione dati e machine learning, obiettivi principali del progetto di tesi.

Bibliografia

[1] Van der Velde, Frank. *Where ArtificialIntelligence and Neuroscience Meet:The Search for Grounded Architec-tures of Cognition*. Adv Artif Intell.vol. 2010, Article ID 918062.

[2] T. M. Mitchell, *The Discipline of Machine Learning*, Pittsburgh, PA 15213, USA: School of Computer Science Carnegie Mellon University, July 2006.

[3] Bishop, C. M. , *Machine Learning and Pattern Recognition,* Clarendon Press, pp. 137, 2006

[4] LeCun, Y., Bengio, Y. & Hinton, G. *Deep learning*. Natura 521, 436-444 (2015). https://doi.org/10.1038/nature14539

[5] Sepp Hochreiter e Jürgen Schmidhuber. *Long Short-Term Memory. In: Neural Computation.* 9.8, pp. 1735–1780(nov. 1997), issn: 0899-7667.

doi: 10.1162/neco.1997.9.8.1735.

[6] Samaneh Yazdani, Jamshid Shanbehzadeh, Mohammad Taghi Manzuri Shalmani, "RPCA: *A Novel Preprocessing Method for PCA", Advances in Artificial Intelligence*, vol. 2012, Article

ID 484595, 7 pages, 2012. https://doi.org/10.1155/2012/484595

[7] E. J. Cand`es, X. Li, Y. Ma e J. Wright, «*ROBUST PRINCIPAL COMPONENT ANALYSIS?*,» arXiv:0912.3599 ,December 17, 2009.

[8] Yannick Roy, *Deep learning-based electroencephalography analysis: a systematic review*et, *J. Neural Eng.* **16** 051001,2019

[9] Sergey Ioffe e Christian Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. In: arXiv preprint arXiv:1502.03167 (2015)

[10] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. In: arXiv e-prints, arXiv:1412.6980 (dic. 2014), arXiv:1412.6980. arXiv: 1412.6980 [cs.LG].

[11] Francesco P. Branca, *Fondamenti di ingegneria clinica*, Vol 1 Springer, 2000

SITOGRAFIA

I. https://machinelearningmastery.com/clustering-algorithms-with-python/

II. https://machinelearningmastery.com/types-of-classification-in-machine-learning/

III. https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/

IV. https://it.myservername.com/types-machine-learning

V. https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d

VI. http://computationalsciencewithsuman.blogspot.com/p/single-layer-and-multilayer-feed.html

VII. A. Soleimany, «6.S191 - Recurrent Neural Network,» MiT, 2020. [Online]. Available: https://www.youtube.com/watch?v=SEnXr6v2ifU&t=1848s.

VIII. https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11#:~:text=In%20a%20network%20of%20n,the%20problem%20of%20exploding%20gradient%20.

IX. https://cran.r-project.org/web/packages/rpca/rpca.pdf

X. https://github.com/dlaptev/RobustPCA

XI. https://www.machinecurve.com/index.php/2020/02/18/how-to-use-k-fold-cross-validation-with-keras/

XII. https://stackoverflow.com/

XIII. https://www.tensorflow.org/

XIV. https://keras.io/

XV. https://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/

XVI. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

XVII. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

XVIII. https://machinelearningmastery.com/improve-deep-learning-performance/

XIX. https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/