ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Ingegneria ed Architettura
Corso di Laurea in **Ingegneria Informatica**
Tesi Magistrale in **Intelligenza Artificiale**

Previsione del declino funzionale tramite l'utilizzo di Reti Neurali Ricorrenti

Candidato:
Daniele Fongo

Relatore:
Chiar.mo Prof.
Federico Chesani

Correlatore:
Ing. Luca Cattelani

Sessione II

Anno Accademico 2016-2017

Indice

Introduzione

PreventIT è un progetto europeo nato con il fine di prevenire il declino funzionale e l'insorgere di disabilità in persone prossime all'anzianità. Questo viene fatto da una parte promuovendo un invecchiamento salutare attraverso l'uso di dispositivi tecnologici come smartphone e smartwatch che incoraggino l'attività fisica, dall'altra parte effettuando degli screening continui, osservazioni o controlli periodici per analizzare i rischi del declino ed individuare le persone più in pericolo al fine di intervenire il prima possibile.

A partire dagli stessi dati utilizzati all'interno del progetto europeo sopra citato, lo scopo della seguente tesi è quella di sviluppare un tool parallelo basato sulle Reti Neurali Artificiali in grado di automatizzare tale analisi del rischio, offrendo una buona previsione del possibile declino funzionale futuro a partire da una serie di osservazioni sulle persone. In particolare, l'interesse scientifico del progetto è nel constatare quale sia il modello di rete neurale che meglio riesce a predire una classe di rischio partendo da uno scenario complesso in cui le osservazioni risultano eterogenee poiché estrapolate da multipli dataset differenti ed in cui la probabilità che avvenga un declino funzionale reale risulta inferiore alla probabilità che non vi sia alcun peggioramento fisico.

Nell'ambito del progetto è stato confrontato l'utilizzo di una Long Short-Term Memory (LSTM), ovvero una rete ricorrente in grado di elaborare sequenze di osservazioni per predire una probabilità di declino, con alcune reti feedforward a diversi livelli di profondità in grado di fornire previsioni a partire da singole osservazioni. In tutti i casi le reti sono state addestrate al fine di fornire una classificazione probabilistica del rischio di

declino funzionale a partire da due diversi dataset di riferimento. L'attenzione maggiore è quindi nel confronto tra le performance di reti ricorrenti e reti feedforward nei casi applicativi reali. Data la natura del problema, la scelta delle migliori architetture per ogni rete, così come il confronto tra le reti stesse, è stata effettuata sulla base di metriche per la classificazione, le quali offrono una visione sintetica della qualità dei risultati prodotti. Si è infine implementato uno Stacked Denoising Autoencoder per ottenere una rappresentazione omogenea delle osservazioni indipendente dal dataset di provenienza; in tal caso si è verificato se vi fossero margini di miglioramento nelle predizioni offerte dalle feedforward e dalla Long Short-Term Memory.

Il primo capitolo tratterà la classificazione al fine di comprendere i problemi di previsione, definendo anche come tali previsioni possano essere valutate. In particolare verrà definito come è possibile trovare la migliore architettura di un classificatore e come possa esserne verificata la performance. A tal proposito verranno indicate alcune metriche di valutazione per il caso di studio, partendo dalle più semplici a quelle che forniscono maggiori dettagli sulla qualità della classificazione, come l'AUCROC ed il Brier-score. Si mostreranno inoltre alcuni possibili accorgimenti che sono stati poi considerati per gestire lo sbilanciamento della classe di interesse.

Nel secondo capitolo si descriveranno le Reti Neurali Artificiali, partendo dal loro modello e da come esse apprendono a risolvere problemi, giungendo alle loro possibili ottimizzazioni e alla definizione delle tipologie di reti implementate all'interno del progetto di tesi. Nel dettaglio verranno mostrate alcune strategie allo stato dell'arte adoperate per migliorare nel complesso l'apprendimento e le performance delle reti, altre ancora per gestire alcuni problemi specifici insiti nel caso di studio, come la carenza di dati e la grande dimensionalità delle osservazioni.

Nel terzo capitolo si spiegherà nel dettaglio la problematica e le scelte implementative effettuate per sviluppare il tool. In particolare si mostrerà inizialmente come il declino funzionale futuro sia stato definito e come i dati siano stati trattati e rappresentati. In un secondo momento si indicheranno le scelte progettuali per la modellazione delle reti neurali di interesse. Infine verrà descritto come siano state scelte le migliori archi-

tetture di ogni singola rete e come le performance di queste ultime siano state confrontate.

Nel quarto capitolo si mostreranno i risultati sperimentali del progetto, indicando innanzitutto quali siano state le migliori architetture trovate per ogni rete, procedendo poi ad una analisi dei risultati forniti dalle metriche di riferimento al fine di comprendere quale tipo di approccio risulti migliore per il dominio in questione.

Capitolo 1

Classificazione

In statistica e nell'informatica, la classificazione è nota come il problema di identificare le categorie o *classi* di appartenza dei dati, basandosi su un insieme di *osservazioni* la cui categoria risulta già nota. Una possibile classificazione è ad esempio quella di assegnare "spam" oppure "no-spam" a nuove email considerando lo storico delle email segnalate come spam. Le classi possono essere dei numeri, dei valori categorici come "gatto" e "cane", oppure ancora dei valori ordinali come "sufficiente" e "buono". Le osservazioni sono invece definite da una serie di proprietà, note come *attributi*: nell'ambito delle reti neurali o del machine learning in generale, le osservazioni divengono dei vettori di features.

Quando la classe da predire è solamente una, si parla di *classificazione binaria* e il risultato del classificatore indica appartenenza oppure non appartenenza alla classe. Quando invece le classi sono più di una, se ogni osservazione ha una sola classe, si parla di *classificazione multi-class*, se invece ogni dato può avere più classi, si parla di *classificazione multi-label*. Nell'ambito della seguente tesi l'ultimo tipo di classificazione non verrà discusso. La classificazione può essere principalmente di due tipi: *crisp*, in cui si assegna una sola classe fra quelle possibili, oppure *probabilistica*, in cui il classificatore assegna all'oggetto da classificare una probabilità di appartenenza a ciascuna delle classi possibili; quest'ultima risulta concettualmente più valida in quanto permette di fornire una stima dell'incertezza del classificatore stesso oltre che della sua correttezza [1].

Solitamente nella classificazione probabilistica il valore atteso è 1 quando vi è appartenenza e 0 quando non vi è appartenenza, per cui la probabilità di appartenza viene indicata da un valore tra 0 ed 1, dove quest'ultimo rappresenta il 100%.

1.1 Training-set e Test-set

Un classificatore addestrato su un certo insieme di osservazioni potrebbe rappresentare un modello che derivi anche dalle specificità e dalle caratteristiche irrilevanti dei dati (rumore), divenendo poco generale, ovvero poco preciso per le nuove osservazioni; questo problema è noto come *overfitting*. Dall'altra parte il modello del classificatore potrebbe risultare così semplice da non riuscire ad eseguire un adeguato pattern recognition che gli garantisca dei buoni risultati di classificazione; in tal caso si parla di *underfitting* [2]. Mentre quest'ultimo può essere risolto tramite un aumento della complessità del classificatore, l'overfitting è un problema altamente indesiderabile poiché non garantisce una classificazione affidabile nei casi applicativi. Per tale ragione, per stimare la accuratezza del classificatore nelle nuove osservazioni, parte dei dati non viene utilizzato durante l'addestramento ma adoperato come verifica del modello appreso. Ciò viene fatto dividendo l'insieme dei dati in due set, il *training-set* per l'addestramento e il *test-set* per la convalida. importante che la loro intersezione risulti vuota, così da garantire la validità della verifica stessa.

La selezione dei due set può avvenire in diversi modi. Il metodo più semplice è l'*holdout* [3], in cui viene selezionata casualmente una frazione α del dataset che verrà usata come training-set, mentre il restante $1 - \alpha$ andrà a far parte del test-set. La principale limitazione di questo approccio consiste nel fatto che, se le classi sono fortemente sbilanciate, potrebbero non essere ugualmente rappresentate in training-set e test-set, per cui la valutazione potrebbe non essere corretta. L'alternativa è quella di eseguire un holdout stratificato, in cui la proporzione delle classi è la medesima nei due insiemi. Tuttavia l'holdout, sia stratificato che non, limita nel training e nella valutazione la quantità di osservazioni disponibili. Per usare l'intero dataset ed effettuare una valutazione più significativa del classificatore si

può usare una *k-fold cross-validation* [3], in cui si partiziona il training set in k sottoinsiemi ed iterativamente 1 di essi viene usato come test e gli altri $k-1$ come training; in un successivo momento si calcola la correttezza del classificatore applicando una media sulle metriche ottenute dai k training differenti. Maggiore è il valore di k, più la stima del reale errore risulta accurata; tuttavia questo allunga i tempi di addestramento e valutazione. Il valore empirico di k solitamente usato è 10.

1.2 Ottimizzazione degli iperparametri

Al fine di scegliere il migliore classificatore tra diverse possibili implementazioni usare il test-set come riferimento è sufficiente. Poiché ogni classificatore può essere calibrato a priori impostando un certo numero di *iperparametri* che ne definiscono l'architettura, la complessità ed altre caratteristiche, usare il test-set per scegliere tali iperparametri non è sufficiente perché il modello che ne deriverebbe risulterebbe costruito ad-hoc per il test-set stesso, il cui scopo invece è quello di verificare il grado di generalizzazione dei classificatori. Per tale ragione solitamente il training-set è diviso ulteriormente in due sottoinsiemi, il training-set in senso stretto ed il *validation-set*, che funge da set di test per la scelta dei migliori iperparametri del classificatore [2]. Le possibili divisioni tra training-set e validation-set sono analoghe a quelle già viste.

Lo scopo dell'ottimizzazione degli iperparametri è quindi quello di ottenere un modello che minimizzi l'errore sul validation-set, il quale rappresenta poi una stima dell'errore sul test-set; in questo modo si può ottenere un modello che riesca da una parte a rappresentare i dati di training (no underfitting) e dall'altra ad essere sufficientemente generale (no overfitting).

1.3 Valutazione

La valutazione del classificatore viene effettuata usando diverse possibili *metriche*, in base al tipo di classificazione. Nel seguente progetto saranno considerate solamente le metriche per la classificazione binaria.

Partendo dal caso crisp, sia la classe attesa che la classe predetta possono assumere due possibili valori: appartenenza e non appartenenza. Per tale ragione i risultati per ogni osservazione possono essere divisi in quattro categorie:

- *Veri positivi* o true positive (TP): predetta correttamente l'appartenenza alla classe.
- *Veri negativi* o true negative (TN): predetta correttamente la non appartenenza alla classe.
- *Falsi positivi* o false positive (FP): predetta erroneamente l'appartenenza alla classe.
- *Falsi negativi* o false negative (FN): predetta erroneamente la non appartenenza alla classe.

I conteggi di queste quattro categorie vengono solitamente poi rappresentati attraverso una *matrice di confusione binaria* come quella mostrata in tabella 1.1.

		Classe predetta	
		Appartenenza	Non appartenenza
Classe attesa	Appartenenza	TP	FN
	Non appartenenza	FP	TN

Figura 1.1: Matrice di confusione binaria.

A partire da quest'ultima possono venire definite diverse metriche di valutazione per la classificazione binaria che tengono conto di tali conteggi in modi differenti. Quando la classificazione è probabilistica, è possibile imporre una soglia o *threshold* che divida l'appartenenza dalla non appartenenza alla classe attesa: per fare un esempio, se la soglia è imposta a 0.7, allora 0.8 corrisponde ad appartenenza, mentre 0.6 a non appartenenza. Scegliere la giusta threshold risulta difficile e dipendente dal caso applicativo, inoltre ad ogni threshold il rapporto tra le quattro categorie sopra

descritte cambia all'interno della matrice di confusione, cambiando anche i risultati delle metriche basate su essa. Tuttavia, dato che la classificazione probabilistica fornisce dettagli sull'incertezza della classificazione oltre che sulla correttezza, è preferibile utilizzare delle metriche che tengano conto delle distribuzioni di probabilità. Nella seguente tesi le metriche considerate sono quattro: l'Accuracy, la Statistica Kappa che richiedono una classificazione crisp (e quindi una discretizzazione degli output), il Brier-Score e l'AUCROC che invece considerano le distribuzioni di probabilità, indicando anche l'incertezza della classificazione.

1.3.1 Accuracy

La metrica più semplice presa in considerazione è l'*Accuracy*, o accuratezza. Essa è una misura di concordanza data dalla semplice equazione:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.1}$$

con TP, TN, FP ed FN rappresentanti le quantità delle quattro categorie riscontrate nella matrice di confusione. Questa misura potrebbe portare ad una interpretazione errata della qualità della classificazioni quando le osservazioni appartengono a classi fortemente sbilanciate verso i veri positivi o i veri negativi. Per fare un esempio, se il classificatore deve riconoscere l'occorrenza di una malattia particolarmente rara, che si riscontra nell'1% delle persone, se esso fornisse sempre come predizione la non-malattia, si avrebbe comunque un'accuratezza del 99%. Tuttavia risulta facile da calcolare e può dare una visione rapida della correttezza di un classificatore.

1.3.2 Statistica Kappa

La *Statistica Kappa* o Kappa di Cohen [4] è una metrica per il calcolo della correttezza più robusta rispetto all'Accuracy. Infatti, supponendo che il 50% delle osservazioni appartenga alla classe, classificare tramite un lancio di moneta porterebbe ad una Accuracy del 50%, poiché il 25% dei casi finirebbe nei veri positivi ed il 25% nei veri negativi. La Statistica

Kappa tiene conto di questa casualità, rappresentando piuttosto il miglioramento dato dal classificatore C di interesse rispetto ad un classificatore probabilistico P in cui la proporzione delle predizioni (appartenenza, non-appartenenza) sia uguale a quella prodotta da C. Per calcolare la Kappa di Cohen si devono quindi prima determinare sulla matrice di confusione MC di C le somme marginali delle righe e delle colonne, poi si deve costruire una matrice di confusione MP per P che mantenga le stesse proporzioni, infine valutare l'incremento di casi concordanti (diagonale), considerando come limite superiore la massima concordanza possibile (numero totale di osservazioni). Questo viene fatto attraverso la seguente equazione:

$$Kappa = \frac{\sum_{i=1}^{2} MC_{ii} - \sum_{i=1}^{2} MP_{ii}}{N - \sum_{i=1}^{2} MP_{ii}} \tag{1.2}$$

dove N è il numero totale di osservazioni. Il codominio è $[-1; 1]$: quando $Kappa < 0$, C si comporta peggio di P, mentre più si avvicina ad 1 più il classificatore C è accurato. Si considera una discreta correttezza se $0.4 < Kappa \leq 0.6$, buona se $0.6 < Kappa \leq 0.8$ ed ottima se $Kappa > 0.8$ [5].

1.3.3 Brier-Score

Il *Brier-Score* [6] è una metrica utilizzata per le classificazioni probabilistiche che permette di misurare l'incertezza di un classificatore; esso è indicato come la distanza media quadrata tra la probabilità predetta ed il valore atteso (1 = appartenenza) e può essere applicato solo al caso binario e al caso multi-class poiché richiede che le classi attese siano mutuamente esclusive. Viene descritto dall'equazione:

$$BS = \frac{1}{N} \sum_{i=1}^{N} (f_i - o_i)^2 \tag{1.3}$$

dove N è il numero osservazioni su cui viene calcolato il Brier-Score, f_i la probabilità predetta e o_i il valore atteso. Essendo una metrica rappresentativa di una distanza, minore è il suo valore, maggiore è la qualità del classificatore. In particolare il suo dominio è $[0, 1]$, dove il valore è 0 indica nessuna incertezza, 0.25 indica che le probabilità prodotte si trovano

mediamente alla stessa distanza dal valore atteso (es: 1) e dal suo opposto (es: 0), mentre valori superiori indicano che le predizioni sono più lontane dal valore atteso rispetto al suo opposto.

1.3.4 AUCROC

L'*AUCROC* è una particolare metrica utilizzata per le classificazioni probabilistiche binarie ma estendibile comunque anche al caso multi-class. Essa è basata sulla *curva ROC* [7] ed indica la probabilità che, estraendo casualmente un elemento appartenente alla classe, il classificatore produca una probabilità di appartenenza più alta rispetto a quella che si avrebbe estraendo un elemento non appartenente alla classe. In altre parole, questa metrica indica quanto le distribuzioni dei veri positivi e dei veri negativi date dal classificatore siano disgiunte e quindi si abbia una buona classificazione (pochi falsi positivi o negativi). Poiché al variare della threshold il numero di occorrenze sulla matrice di confusione cambia, se le due distribuzioni non sono sovrapposte, esisterà un particolare valore di threshold per il quale la classificazione risulta esatta.

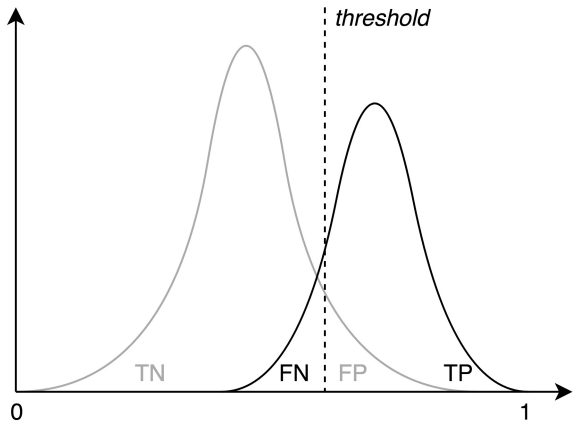

Figura 1.2: Distribuzioni dei veri negativi e dei veri positivi.

Come si può vedere in figura 1.2, nei casi reali le due distribuzioni risultano spesso sovrapposte ed è possibile determinare, al variare del valore della threshold (lungo l'asse X della figura), la percentuale di veri positivi rispetto al totale dei positivi (True Positive Rate) e la percentuale di

falsi positivi sul totale dei negativi (False Positive Rate) sulla matrice di confusione binaria. Tracciando per ogni threshold queste due percentuali in un grafico come quello mostrato in figura 1.3, si ottiene la curva ROC: più le distribuzioni sono disgiunte, più la curva tende verso l'alto e più il classificatore risulta buono. L'AUCROC è rappresentata dall'area sotto tale curva, per cui il suo dominio è [0,1]: un AUCROC pari a 0.5 indica che le due distribuzioni hanno la stessa media e sono sovrapposte, mentre un valore prossimo ad 1 indica una buona separazione delle distribuzioni e dunque un'ottima classificazione.

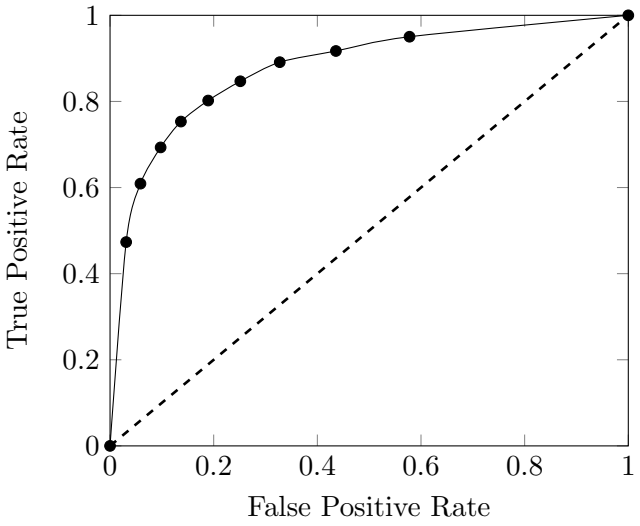

Figura 1.3: Curva ROC.

1.4 Classi sbilanciate

Spesso accade che le classi non siano equamente rappresentate nel dataset, per cui un classificatore potrebbe essere orientato a riconoscere meglio le classi dominanti, divenendo poco generale per quelle minoritarie. Questa problematica può essere gestita in diversi modi:

- *Undersampling*: si sottocampionano le classi maggioritarie. Questo non risulta possibile in tutte quelle situazioni in cui l'insieme di training-set è piccolo, poiché questo tipo di operazione riduce ul-

teriormente la quantità di dati usufruibile per l'addestramento del classificatore [8].

- *Oversampling*: si sovracampionano le classi minoritarie. Quando le classi minoritarie presentano pochi elementi si rischia di avere un numero eccessivo di duplicati e quindi il classificatore può impararne le specificità [8].

- *Rumore* e *generazione di dati*: si aggiunge del piccolo rumore alle osservazioni in modo da generare nuovi dati che siano distribuiti in un intorno delle osservazioni originali; un'alternativa è quella di sintetizzare nuovi dati a partire dalle classi minoritarie [9]. In questo modo si risolve il problema presente nell'oversampling. Si possono usare questi approcci anche quando i dati di training risultano pochi.

- *Cost-sensitive learning*: si penalizza l'algoritmo di apprendimento al fine di aumentare il costo di errata classificazione per le classi che risultano meno frequenti del dataset [8]. Questo può essere fatto ad esempio tramite l'uso di un vettore di costo in cui per ogni classe si ha un valore numerico inversamente proporzionale alla sua frequenza nel dataset. Il cost-sensitive learning può essere usato anche in tutti quei casi in cui l'errata classificazione risulta più rilevante per certe classi rispetto che per altre. Per fare un esempio, in ambito medico predire erroneamente una malattia (falso positivo) è meno problematico che predire una malattia che invece non si presenterà (falso negativo).

Capitolo 2

Reti Neurali Artificiali

Le reti neurali artificiali (Artificial Neural Network) rappresentano un modello matematico per la risoluzione di problemi fortemente ispirato alle reti neuronali.

Le reti neuronali sono reti o circuiti complessi di neuroni interconnessi tra loro che svolgono una particolare funzione fisiologica. Tali circuiti, negli organismi dotati di sistema nervoso, hanno un ruolo fondamentale nel riconoscere ed elaborare gli stimoli esterni, reagendo ad essi tramite l'innesco di una serie di impulsi nervosi che provocano mutazioni alla struttura del reticolo neuronale stesso; è grazie a tale riconfigurazione che la rete si adatta agli stimoli e memorizza.

La rete neurale artificiale (d'ora in poi, semplicemente NN) impara dunque ad approssimare una funzione di risoluzione di un determinato problema, come ad esempio classificare delle immagini, tramite una progressiva modifica delle connessioni della rete al fine di migliorare l'elaborazione dell'input (stimoli) e massimizzare la precisione della funzione che si intende imparare. Nel caso della classificazione delle immagini, ad esempio, le foto sono semplicemente dei dati in ingresso, mentre la categoria del soggetto rappresentato nelle stesse diviene l'output che la rete deve imparare a riconoscere; in questo caso la NN dovrà cercare di minimizzare l'errore di classificazione.

Le NN, per come sono state ideate e modellate, possono quindi essere considerate come delle "blackbox" in grado di apprendere qualsiasi fun-

zione (lineare o non lineare) sfruttando esclusivamente l'uso di un numero considerevole di stimoli o dati di riferimento [10]. Esse risultano pertanto utili in tutte quelle applicazioni in cui le informazioni da elaborare sono estremamente complesse (si veda ad esempio lo speech-recognition) e/o la definizione della funzione risolutrice non sia possibile attraverso l'analisi del problema. Inoltre, grazie alla loro struttura basata su connessioni (connessionismo), le NN possono essere eseguite parallelamente su più processori, su GPU oppure su architetture distribuite di computer per accelerare l'addestramento.

D'altro canto, essendo la conoscenza appresa per esempi, è necessario fornire alle NN un insieme di stimoli o dati estremamente numeroso e rappresentativo del problema: ad esempio una rete neurale artificiale non potrà mai imparare a distinguere i gatti se le foto fornite ad essa non contengono sufficienti immagini di gatti. Tale conoscenza risulta per di più sub-simbolica in quanto rappresentata dalle dinamiche inter-neuronali e dalla robustezza o debolezza delle connessioni, e quindi solo parzialmente estrapolabile ed interpretabile [11]. Inoltre, nella realtà implementativa, per ottimizzare l'apprendimento ed i risultati, le NN hanno bisogno di una messa a punto degli iperparametri estremamente difficile: spesso la scelta degli iperparametri migliori sta nell'esperienza del programmatore o nell'uso di strumenti che eseguono ricerche esaustive sul dominio degli iperparametri, con conseguente esplosione del tempo di addestramento.

Nonostante le limitazioni delle NN, alcune delle quali indicate precedentemente, esse rappresentano uno strumento "universale" per la risoluzione di problemi ampiamente usato al giorno d'oggi.

2.1 Il modello

L'unità elementare della rete neuronale, il neurone biologico, è costituito da tre elementi fondamentali mostrati in figura 2.1:

- Il dendrite: il canale d'entrata che riceve i segnali ottenuti da altri neuroni mediante le sinapsi; esso attenua o inibisce i segnali in base ad un "peso" rappresentato dalla potenza delle connessioni sinaptiche.

- Il soma: il corpo centrale del neurone; esso riceve dai suoi numerosi dendriti una somma di segnali pesati e, se tale valore supera una determinata soglia di attivazione, il neurone produce un segnale in uscita.

- L'assone: il canale di uscita estremamente ramificato del neurone; esso propaga il segnale verso altri neuroni.

Il segnale viene inviato tramite impulsi discreti, dunque il superamento della soglia di attivazione dipende dalla frequenza con cui tali impulsi vengono ricevuti dal neurone, e quindi dalla velocità con cui questo accumula potenziale elettrico. Gli impulsi sono di intensità costante e ciò che caratterizza la comunicazione neuronale è la frequenza con cui questi vengono generati.

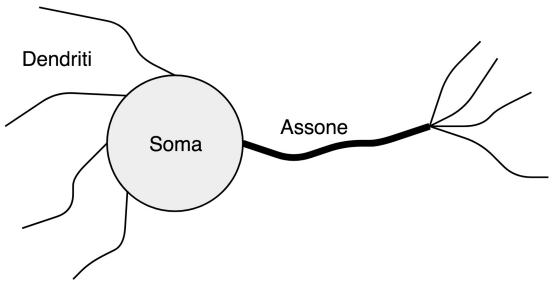

Figura 2.1: Schema di un neurone biologico.

La struttura del *neurone artificiale* è stata modellata sul neurone biologico, ma il segnale in uscita dipende dall'intensità degli ingressi e non dalla loro frequenza; sono comunque stati introdotti dei modelli alternativi di neurone artificiale, chiamati *spiking neurons*, che incorporano il concetto di tempo e di potenziale elettrico nel loro modello operativo [12].

Il neurone artificiale riceve in ingresso un vettore di *input* numerici moltiplicati scalarmente per un vettore di *pesi*. Uno dei pesi viene chiamato *bias* ed è collegato ad un input fittizio di valore 1. Dopodiché il neurone applica a tale somma pesata una determinata funzione di attivazione

(lineare o non lineare), o *activation-function*, producendo 1 singolo *output*. La formula matematica che riassume tale funzionamento è la seguente:

$$y = activation(\sum_{i=1}^{N} w_i x_i + b) \tag{2.1}$$

dove y è l'output, *activation* è la funzione d'attivazione, N è il numero di input, x_i e w_i sono l'input i-esimo ed il suo corrispondente peso, e b è il bias. Per comprendere il significato dei pesi e del bias, si immagini di avere una funzione d'attivazione che restituisca esclusivamente i valori 0 o 1 (ad esempio una binary-step); poiché l'input è N-dimensionale, l'insieme dei pesi genera un fascio di iperpiani, tutti passanti per l'origine, che dividono gli input che producono l'uscita 0 da quelli che producono l'uscita 1; il bias serve ad aggiungere un'intercetta all'iperpiano in modo da ampliare l'insieme dei problemi risolubili. L'apprendimento di un neurone è quindi rappresentato da un ribilanciamento dei pesi e bias al fine di approssimare correttamente la funzione desiderata.

L'output potrà poi costituire un input di uno o più eventuali neuroni a valle oppure essere il risultato (o parte di esso) della funzione che la rete neurale dovrà imparare. Per una trattazione più dettagliata dell'apprendimento si rimanda alla sezione 2.2.

2.1.1 Topologie

Il singolo neurone artificiale rappresenta la topologia più semplice che si possa avere. Grazie alla semplicità dell'interfaccia input/output del neurone, esso può essere poi collegato con altri neuroni per costituire topologie più complesse, le quali permettono di risolvere problemi di difficoltà superiore. Tipicamente i neuroni vengono organizzati in livelli, o *layer*, in modo che gli output di un determinato livello divengano gli input del livello successivo; in questo caso ogni layer presenta una matrice di pesi ed un vettore di bias. I layer più comunemente usati sono i *fully-connected*, in cui ogni neurone è collegato con tutti i neuroni a monte e/o a valle. La propagazione del segnale tra i layer è solitamente *sincrona* per cui, prima di computare gli ingressi del layer $k+1$, è necessario il layer k abbia pro-

dotto delle uscite. Nonostante nel tempo si siano definite reti sempre più articolate, esse ricadono tendenzialmente in due topologie principali:

Feedforward Network : essa presenta un *input-layer* per la ricezione dei dati, un *output-layer* costituito da un numero di neuroni pari al numero di valori in uscita che si desiderano, ed un numero generico (anche zero) di layer nascosti, o *hidden-layer*, ognuno di dimensione arbitraria. Il nome deriva dal fatto che il segnale inserito nella rete (feed) viene propagato in avanti (forward) tra un layer e l'altro.

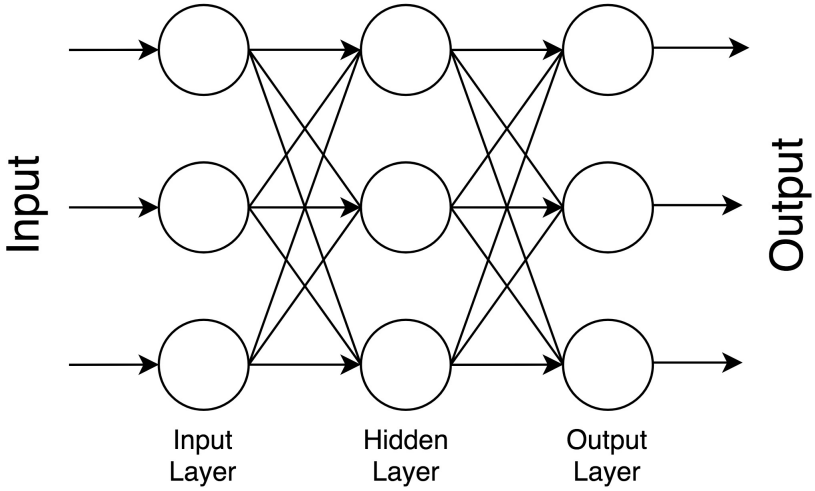

Figura 2.2: Esempio di una feedforward Network.

Recurrent Network : essa presenta, rispetto alle feedforward network, delle connessioni di feedback. Tali connessioni sono generalmente verso i neuroni dello stesso livello, ma è comunque possibile averne verso layer precedenti. Questa topologia, introducendo dei cicli all'interno della rete, complica il flusso delle informazioni, infatti si richiede di considerare il comportamento della rete anche in relazione alle connessioni di feedback: una connessione ciclica causerebbe un "loop infinito" nella propagazione del segnale, per cui risulta necessario interrompere quest'ultima dopo N cicli o *timestep*. Tuttavia risulta interessante notare che è possibile, per ogni timestep, inserire un nuovo input, per cui l'output diviene l'elaborazione di una intera

serie temporale lunga N. Per tale ragione le recurrent network, o *reti ricorrenti*, vengono usate per l'analisi di sequenze come ad esempio frasi di un linguaggio naturale, audio etc.

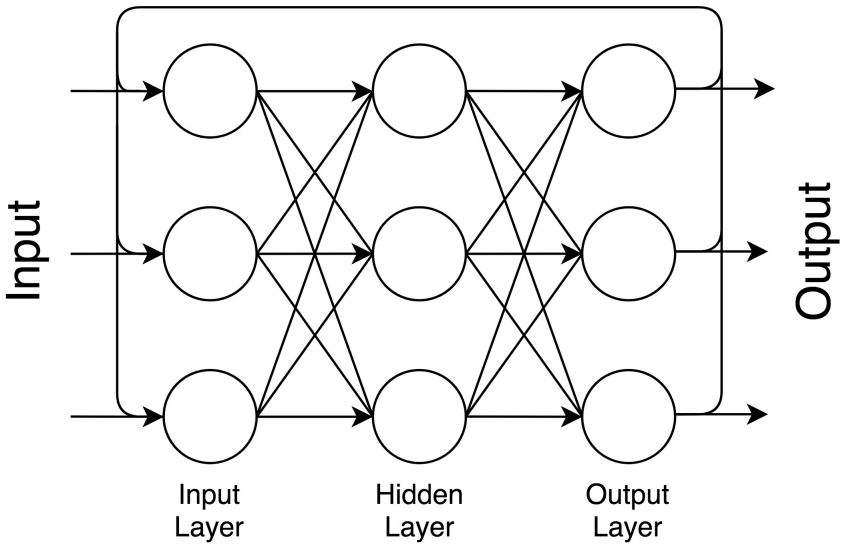

Figura 2.3: Esempio di una recurrent Network.

Generalmente, quando sono presenti molteplici hidden-layer o si ha una composizione a cascata di Reti Neurali, si parla di Reti Neurali Deep, o *Deep Neural Network* (DNN). Il motivo dietro la nascita delle DNN risiede nel fatto che, nonostante il *Teorema dell'Approsimazione Universale* affermi che è possibile approssimare qualsiasi funzione con una rete feedforward con un singolo hidden-layer, il numero dei suoi neuroni crescerebbe esponenzialmente con la difficoltà del problema [10]; usando invece una DNN è possibile ottenere medesimi risultati aggiungendo più hidden-layer di dimensione ridotta: ogni layer infatti elabora ed estrae feature degli output del layer precedente, offrendo un maggiore grado di astrazione. Tuttavia l'addestramento delle DNN risulta estremamente più difficile dal punto di vista pratico a causa di molte problematiche correlate alla profondità e alla complessità stessa della rete.

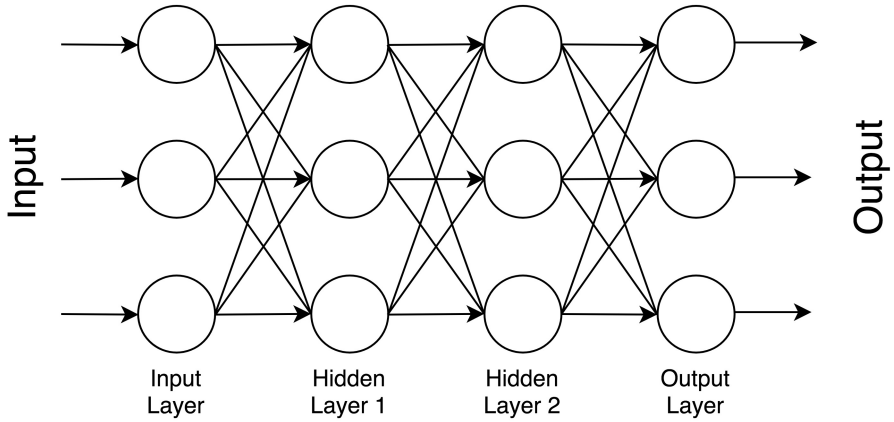

Figura 2.4: Esempio di una Deep Neural Network.

2.2 Apprendimento

Una NN può essere addestrata per approssimare qualsiasi funzione, perciò i possibili task che essa può svolgere sono molti, ma sono comunque divisibili in diverse categorie. Alcune categorie d'interesse per il progetto sono:

- *classificazione*, per categorizzare i dati.
- *denoising*, per riprodurre dati puliti a partire da dati corrotti.
- *estrazione di feature*, per ridurre la dimensionalità e/o ottenere una rappresentazione sub-simbolica omogenea dei dati a partire da osservazioni eterogenee.

In base al task da svolgere viene scelta la topologia di rete più adatta ed il tipo di addestramento. Nell'ambito della seguente tesi i tipi di apprendimento rilevanti sono quello supervisionato e quello non supervisionato:

Apprendimento supervisionato : viene scelto quando si conosce il *valore atteso* dei dati impiegati per l'addestramento. In questo apprendimento la rete deve dunque imparare a inferire la relazione che lega gli ingressi con i valori attesi, cercando di minimizzare l'errore di previsione. Come si vedrà nella sezione 2.2.1, la minimizzazione

dell'errore avviene tramite l'applicazione di un algoritmo noto come *gradient-descent*. L'obiettivo finale è quello di addestrare la rete al fine di dotarla di una sufficiente capacità di generalizzazione che le permetta, basandosi sugli esempi con cui è stata addestrata, di prevedere correttamente il valore atteso di ingressi laddove non è noto. Per tali ragioni l'apprendimento supervisionato viene usato ad esempio per risolvere problemi di classificazione.

Apprendimento non supervisionato : viene scelto quando i dati impiegati per l'apprendimento non hanno un valore atteso. Lo scopo di questo approccio è quello di inferire una funzione che riesca a descrivere la struttura interna o estrarre le caratteristiche interessanti di un insieme di dati in ingresso. Questo tipo di apprendimento permette di estrarre feature e/o ignorare dati ridondanti e questo lo rende adatto anche per l'addestramento di reti per la compressione delle informazioni. Alcune NN per l'apprendimento non supervisionato, come gli Autoencoders (si veda sezione 2.4.2), presentano dei valori attesi e quindi possono essere comunque addestrate utilizzando l'algoritmo del gradient-descent.

Nella seguente tesi le NN adoperate sono state tutte addestrate utilizzando l'algoritmo del gradient-descent.

2.2.1 Gradient-descent e backpropagation

All'interno di una NN le molteplici connessioni tra i neuroni assumono la forma di un grafo orientato pesato, dove i nodi corrispondono ai singoli neuroni, gli archi alle connessioni pesate tra i neuroni e la direzione degli archi al verso di propagazione del segnale. L'elaborazione di un dato in ingresso corrisponde quindi alla trasmissione attraverso tale grafo di un flusso di informazioni che viene di volta in volta alterato dai pesi e dalle funzioni di attivazione dei neuroni. Il risultato di tale propagazione viene dunque rappresentato dai nodi del grafo che non presentano ulteriori neuroni a valle. Una rappresentazione matematica più interessante e funzionale del grafo appena descritto è quella definita da un *grafo*

computazionale, in cui ogni nodo non rappresenta più un neurone ma una singola operazione matematica. In figura 2.5 si può osservare un esempio di semplice grafo computazionale.

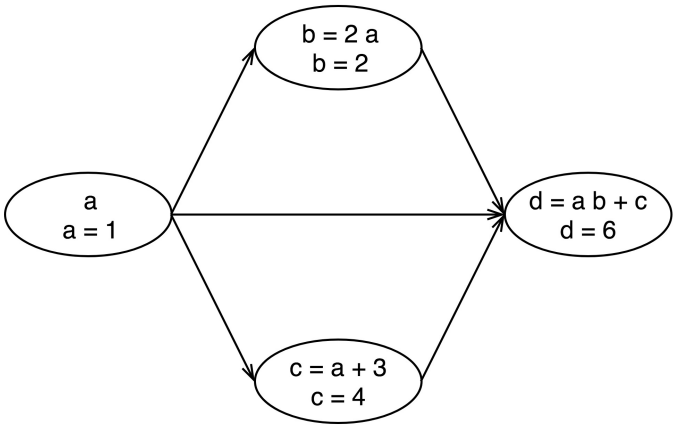

Figura 2.5: Schema di un semplice grafo computazionale.

Poiché lo scopo della rete è quello di minimizzare l'errore di previsione dell'output rispetto al valore atteso dei dati, è necessario calcolare tale errore e procedere a ritroso lungo il grafo orientato per calibrare i pesi in base a quanto essi hanno inciso sull'output errato.

Il principio alla base della minimizzazione dell'errore nelle NN con apprendimento supervisionato è la discesa del gradiente, o *gradient-descent*. Questa tecnica si basa sul concetto che per una funzione $f(w)$ la direzione di massima discesa verso un minimo locale o globale in un determinato punto w corrisponde a quella determinata dall'opposto del suo gradiente (vettore delle derivate parziali) $p = -\nabla f(w)$ [13]. Per minimizzare la funzione, si parte quindi da una w casuale e si procede poi iterativamente, calcolando ad ogni step $f(w)$ ed aggiornando $w = w - \Delta$, con Δ definito come

$$\Delta = -\eta \cdot p = \eta \cdot \nabla f(w) \tag{2.2}$$

dove η rappresenta un tasso di apprendimento, o *learning-rate*. La variazione di w verrà d'ora in poi indicata con Δ. Il gradient-descent presenta

dunque due fasi essenziali che si alternano: la prima, in cui si calcola il valore in uscita della funzione, la seconda in cui si calibra il valore di w al fine di minimizzare la funzione. L'algoritmo termina se una *stopping-rule* viene verificata, per esempio se il minimo viene trovato o se il valore della funzione non decresce per N iterazioni consecutive. In figura 2.6 si possono osservare i vari passaggi dell'algoritmo per una semplice funzione.

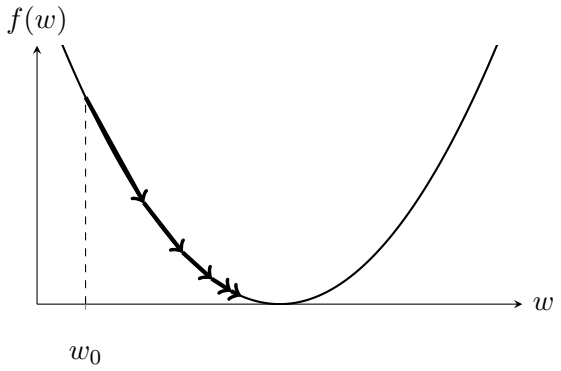

Figura 2.6: Discesa del gradiente in una funzione.

Singolo neurone : nel caso di un singolo neurone, $f(w)$ rappresenta una funzione di perdita, o *loss-function*, che computa l'errore di previsione del singolo output rispetto al valore atteso, mentre w costituisce l'insieme dei suoi pesi $(w_1, ..., w_n)$, dove n è il numero di input. La funzione $f(w)$ risulta pertanto composta:

$$f(w) = loss(target, activation(sum(w, input))) \tag{2.3}$$

dove *loss* è la loss-function, *activation* è la funzione d'attivazione, *target* è il valore atteso e *sum* rappresenta la somma pesata tra gli *input* ed i pesi w, i quali sono la vera incognita della funzione. Per il calcolo della derivata della loss-function rispetto ad un particolare peso w_i, è sufficiente seguire la *chain-rule*, una regola di derivazione matematica con la quale è possibile calcolare in modo semplice la derivata composta di due funzioni differenziabili:

$$\frac{\delta loss}{\delta w_i} = \frac{\delta loss}{\delta activation} \cdot \frac{\delta activation}{\delta weighted\_sum} \cdot \frac{\delta weighted\_sum}{\delta w_i} \tag{2.4}$$

È necessario quindi che sia la funzione di attivazione che la funzione di perdita siano continue e differenziabili. Si ha in tal caso che

$$\Delta = \eta \cdot [\frac{\delta loss}{\delta w_1}, ..., \frac{\delta loss}{\delta w_n}] \tag{2.5}$$

Il calcolo a ritroso, a partire dalla funzione di perdita, dei contributi dei pesi sull'errore prende nome di *back-propagation*.

Rete Neurale : nel caso di una rete neurale più o meno complessa è sufficiente, a partire dal grafo computazionale, procedere a ritroso per calcolare tramite la chain-rule le derivate di ogni nodo del grafo rispetto a quelli che lo seguono; ad esempio, usando il semplice grafo rappresentato in figura 2.5, si ottiene il grafo in figura 2.7. Dopodiché si procede con il medesimo algoritmo di minimizzazione dell'errore.

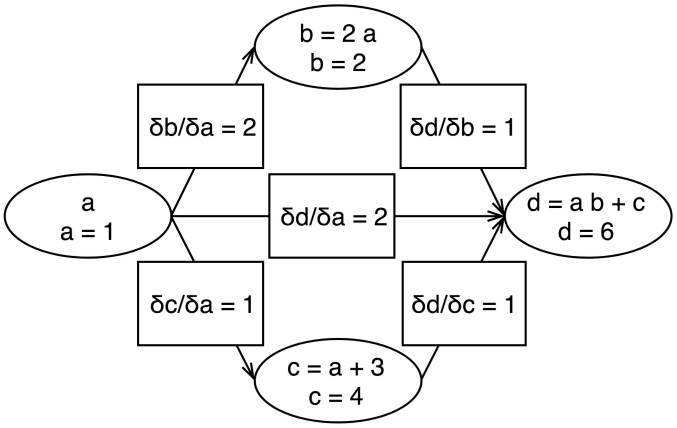

Figura 2.7: Applicazione delle derivate parziali sul grafo computazionale.

L'algoritmo gradient-descent standard, o *batch gradient-descent*, applica le modifiche dei pesi usando l'errore medio sull'output sfruttando tutto l'insieme di dati di input [13]; ogni passo del procedimento iterativo a due fasi corrisponde ad una *epoca*. Tuttavia il gradient-descent non è l'unico ottimizzatore della funzione di perdita; inoltre è possibile lavorare con un sottoinsieme dei dati per volta invece che usare tutti i dati prima di

procedere all'aggiornamento dei pesi; per una trattazione approfondita si rimanda alla sezione 2.3.2.

2.2.2 Input ed output

Le reti neurali artificiali, per come sono progettate, approssimano delle funzioni matematiche, pertanto i dati in ingresso alla NN, sia di input che di output, devono poter essere rappresentati tramite dei vettori numerici. Nell'ambito del progetto di tesi i dati utilizzati presentano semantiche diverse e sono descritti da attributi numerici, categorici, ordinali e binari. Per tale motivo essi necessitano di essere trasformati coerentemente per poter essere utilizzati all'interno delle reti.

Attributi numerici : essi assumono valori numerici sia discreti (es: interi) che continui (es: reali). Questi attributi possono essere codificati in diverse maniere. Si può lasciare l'attributo così com'è, ma risulta un approccio naive poiché alcuni attributi, che possono essere rappresentati su scale differenti, finiscono per avere un impatto maggiore o minore sull'apprendimento della rete: uno stipendio annuale, la cui media si aggira intorno alle decine di migliaia di euro, risulta più significativo rispetto all'età, la cui media è circa 45 anni. Inoltre, come si vedrà nella sezione 2.2.3, input numerici troppo grandi portano a possibili saturazioni di alcune funzioni d'attivazione. Un'alternativa è la *normalizzazione*, in cui l'attributo viene appunto normalizzato in modo che qualsiasi valore appartenga al range $[0; 1]$. In tal modo gli attributi rappresentati su scale differenti hanno la medesima importanza nella rete. Inoltre è meno probabile che vi siano saturazioni nelle activation-function. Per evitare fraintendimenti, questa non corrisponde alla normalizzazione statistica.

Attributi categorici : essi assumono valori che, come dice il nome stesso, rappresentano categorie. Ad esempio un attributo categorico può essere il sesso, la città natale o l'orientamento politico. Data la loro natura nominale, non possono essere convertiti direttamente in numeri, usando per esempio la codifica ASCII, poiché la distanza tra

tali valori non sarebbe rappresentativa della reale differenza tra essi. Un semplice ed efficace modo per codificare tali attributi è quello di rappresentare ogni possibile valore distinto come un *vettore one-hot* in cui un solo elemento vale 1, mentre tutti gli altri valgono 0. Per esempio, il sesso femminile verrebbe codificato in $[1, 0]$ e quello maschile in $[0, 1]$.

Attributi ordinali : essi sono particolari attributi categorici che incorporano un concetto di ordine, per cui sono facilmente trasformabili in valori numerici: un esempio può essere la votazione scolastica "insufficiente", "sufficiente", "buono" etc. In questi casi è possibile usare un solo valore numerico in quanto i vettori one-hot non rappresentano il concetto di distanza, fattore che potrebbe essere rilevante per gli attributi ordinali.

Attributi binari : essi rappresentano il caso limite dei dati categorici o dei dati numerici discreti, in cui sono possibili solo due valori. Ciò permette di usare un solo numero per rappresentare le due categorie oppure di utilizzare un vettore one-hot di due elementi.

Ogni tipo di attributo può inoltre non assumere alcun valore definito. In tal caso, è possibile usare un termine o una flag-word che indichi un dato mancante o non valido. Sfruttando sempre l'esempio del sesso, si hanno quindi in realtà 3 possibilità: "maschio", "femmina", "non-specificato". Per i categorici esso può essere trattato come un ulteriore categoria oppure può essere definito dal vettore $[0, ..., 0]$, mentre per i numerici si può aggiungere un ulteriore attributo binario che indichi la (in)validità del dato stesso.

2.2.3 Funzioni di attivazione

Ogni layer di una NN è solitamente definito da un numero N di neuroni con le medesime funzioni d'attivazione. Queste ultime, come anticipato nella sezione 2.2.1 devono essere continue e differenziabili affinché si possa applicare il procedimento di backpropagation. Ogni activation-function ha delle caratteristiche peculiari che la rendono più adatta a certi domini

applicativi piuttosto che ad altri. Per tale motivo, durante la definizione delle reti del progetto, le funzioni di attivazione degli output-layer sono state scelte a priori in base all'obiettivo prefissato, mentre le activation-function degli hidden-layer sono state selezionate tra un pool di possibili funzioni d'attivazione tramite una ottimizzazione degli iperparametri (si veda sezione 2.3.6).

Nella trattazione seguente verranno presentate alcune funzioni di attivazione d'interesse in cui y rappresenterà l'output di un neurone, mentre s la somma pesata $\sum_{i=0}^{n} w_i x_i + b$ in ingresso, con b bias, x_i i-esimo input e w_i il peso corrispondente.

Sigmoide

La *sigmoide* è una funzione non-lineare ampiamente usata nelle NN ed è definita da:

$$y = \frac{1}{1 + e^{-s}} \tag{2.6}$$

A piccole variazioni dell'input s corrispondono simili variazioni in uscita solo quando l'input si trova approssimativamente intorno al range $[-3, 3]$, dal momento che la funzione sigmoide schiaccia asintoticamente il resto degli input verso valori in output pari 0 o ad 1, come si può osservare in figura 2.8. Ciò implica che se l'input s si trova fuori da $[-3, 3]$ la funzione satura, causando un gradiente prossimo allo 0 e portando quindi ad una possibile stasi dell'apprendimento. Inoltre, non essendo centrata sull'origine, gli output di un layer con sigmoide sono sempre positivi e quindi tutti i pesi di un neurone a valle, durante la backpropagation, tendono a crescere o decrescere tutti insieme.

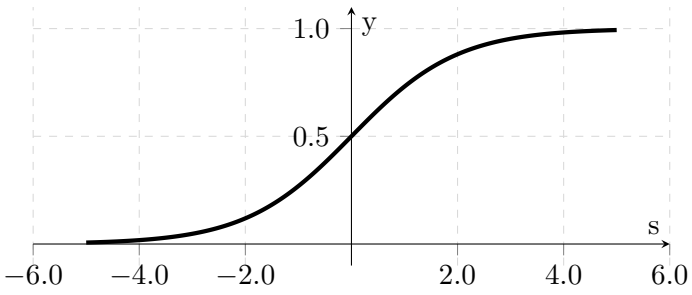

Figura 2.8: Funzione sigmoide.

La sigmoide rappresenta una versione continua di una *binary-step*, una funzione che assume valore 0 per $s < 0$ ed 1 per $s \geq 0$ e che può essere usata per classificazioni binarie crisp. L'output di una sigmoide può dunque essere interpretato come una probabilità di appartenere ad una determinata classe: il valore 0 rappresenta una probabilità dello 0%, mentre 1 una probabilità del 100%. Non essendo l'output di un neurone con sigmoide vincolato dai neuroni dello stesso layer, il vettore di output rappresenta un insieme di probabilità indipendenti tra loro. Per tali ragioni questa funzione d'attivazione viene solitamente usata nell'ambito delle classificazioni probabilistiche multi-label o binarie (nel caso di singolo neurone in uscita).

Tangente iperbolica

La *tangente iperbolica* è una funzione non-lineare definita dall'equazione $y = 2 \cdot sigmoid(2s) - 1$ e rappresenta una versione scalata della sigmoide, con codominio $(-1; 1)$. Essa è spesso preferibile rispetto alla sigmoide sia perché è centrata sull'origine, sia perché la sua derivata è maggiore e questo garantisce una convergenza verso il minimo della funzione di perdita più rapida [14].

Softmax

La *softmax* è una particolare sigmoide definita dall'equazione:

$$y_i = \frac{e^{s_i}}{\sum e^{s_i}} \tag{2.7}$$

con y_i e s_i rappresentanti rispettivamente l'output e la somma pesata dell'i-esimo neurone del layer. Questa particolare funzione è un tipo di sigmoide che vincola i neuroni di un layer normalizzandone il vettore di output in modo che $0 \leq y_i \leq 1 \forall i$ e che $\sum y_i = 1$.

Il vettore di output può essere dunque visto come una distribuzione discreta di probabilità. Grazie a questa particolare caratteristica essa può venire utilizzata per classificare dei dati quando le classi sono indipendenti e si vuol ottenere una probabilità che un ingresso appartenga ad una delle possibili classi, rappresentate come vettori one-hot (classificazione multi-class).

ReLu, Softplus e Leaky-ReLu

La *ReLu*, nota anche come funzione rampa, è funzione non differenziabile nell'intero dominio e definita dall'equazione $y = max(0, s)$, per cui a somme pesate negative viene associato il valore 0. Ciò implica che un layer che usa questa attivazione garantisce che solo alcuni neuroni a valle vengano attivati (quando $s > 0$), rendendo la rete "sparsa" ed alleggerendo la computazione. Quest'ultima risulta inoltre efficiente in quanto l'output di un neurone con ReLu può essere prodotto imponendo una soglia a 0 sopra la quale l'output è semplicemente la somma pesata ricevuta in ingresso. Tuttavia il gradiente nella regione $s \leq 0$ è esattamente 0, il che implica che per i neuroni che finiscono in questa regione i pesi non verranno modificati. Questo problema è noto come *dying ReLu*. Due possibili soluzioni alternative sono la *Leaky-ReLu*, dove nella regione $s \leq 0$ si ha $y = \alpha s$ con α scalare positivo prossimo allo 0, e la *Softplus*, una ReLu differenziabile ma computazionalmente meno efficiente.

2.2.4 Errore e funzioni di perdita

L'errore di previsione in base al quale vengono computati i gradienti durante la backpropagation viene definito *loss* ed è un termine generalmente non negativo computato dalla loss-function sulle uscite dell'output-layer nel rispetto dei valori attesi. La loss è solitamente calcolata come valore medio degli errori computati per l'insieme di input usati per una iterazio-

ne del gradient-descent. Poiché la loss-function rappresenta la funzione da minimizzare, il valore della loss durante l'applicazione del gradient-descent tende a decrescere; un possibile andamento è quello mostrato in figura 2.9.

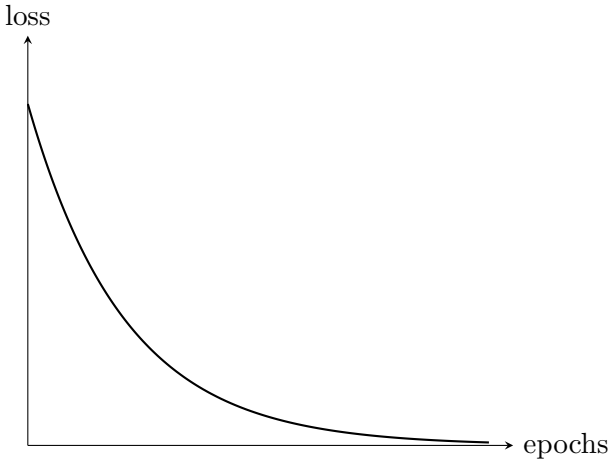

Figura 2.9: Andamento del valore di loss.

Analogamente a quanto detto per le funzioni di attivazione, anche la scelta della loss-function dipende dal caso applicativo e deve essere differenziabile per poter essere usata nella backpropagation. La scelta della funzione di perdita risulta essenziale in quanto da essa dipende l'aggiornamento dei pesi dell'intera rete; per tale ragione essa deve poter rappresentare in modo significativo la differenza tra l'output ed il valore atteso in base al tipo e al dominio di quest'ultimo. Le due funzioni di perdita adoperate nella presente tesi sono l'RMSE e la cross-entropy.

RMSE

L'*RMSE* è una misura usata per indicare la deviazione standard delle differenze tra i valori predetti da uno stimatore, in questo caso la NN, ed i valori attesi. La formula per il calcolo è

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(p_i - y_i)^2}{N}} \tag{2.8}$$

con $(p_i - y_i)$ il valore residuale (o differenza) dell'output predetto p_i rispetto al valore atteso y_i corrispondente. Nelle NN ogni p_i corrisponde

all'uscita di uno dei neuroni dell'output-layer. Data la sua natura, questo tipo di misura risulta molto sensibile ad output elevati, inoltre è poco adatta in caso di classificazione multi-class: facendo un esempio, se l'output calcolato è $[0, 0, ..., 0, 1]$, mentre la classe attesa è $[1, 0, ..., 0, 0]$, nonostante la predizione sia sbagliata, l'RMSE risulterà estremamente basso a causa dei valori residuali degli 0 centrali che influenzano la media.

Cross-entropy

La *cross-entropy* è una misura usata in statistica per valutare quanto una distribuzione probabilistica P approssima bene una distribuzione nota Y: nel caso delle NN, P può essere rappresentato dall'insieme degli output di un layer con attivazione softmax, mentre Y può essere il vettore one-hot della classe attesa. Infatti, in entrambi casi, la somma delle componenti delle due distribuzioni risulta pari ad 1.

Questa misura si fonda sul concetto di entropia $H(Y)$, intesa come il costo in bit per codificare un'informazione Y: se quest'ultima può assumere ad esempio 4 possibili valori ed ognuno ha una probabilità del 25%, i bit per codificare Y saranno 2, se invece solo uno dei valori ha il 100% di probabilità, è possibile non codificare Y in quanto l'informazione risulta ovvia. L'entropia da sola non è sufficiente in quanto non tiene conto di quanto la distribuzione P si avvicina o si allontana dalla distribuzione di riferimento Y, per cui l'ulteriore fattore tenuto in conto in questa misura è l'entropia relativa $H(Y|P)$, che indica invece quanto P diverge da Y. La formula per il calcolo è

$$CROSS\_ENTROPY = H(Y) + H(Y|P) \tag{2.9}$$

Nel caso delle NN con attivazione softmax, in cui il vettore di output rappresenta una distribuzione discreta di probabilità, la formula è semplicemente

$$CROSS\_ENTROPY = -\sum_{i=1}^{N} y_i \cdot \log_2(p_i) \tag{2.10}$$

con N numero di classi, $p_i \in P$ vettore di output e $y_i \in Y$ classe one-hot attesa.

2.3 Ottimizzazione dell'apprendimento

Le scelte progettuali sull'architettura della rete risultano essenziali al fine di garantire che la rete apprenda bene e sia in grado di approssimare correttamente la funzione che si intende imparare. In questa sezione si parlerà dei principali problemi delle reti neurali ed in seguito verranno descritte alcune tecniche prese in considerazione nel caso di studio per migliorare la rete ed il suo training. Si vedrà che tali soluzioni spesso portano a considerare degli iperparametri aggiuntivi, la cui scelta determina l'esito dell'apprendimento.

Convergenza : l'algoritmo del gradient-descent standard, con un appropriato learning rate, garantisce che se la superficie degli errori generata dalla loss-function nel rispetto di tutti i pesi della rete è convessa vi sia una convergenza verso un minimo globale [13]. Infatti in una funzione convessa ogni minimo locale è un minimo globale. Nella realtà, anche quando la superficie degli errori è convessa, potrebbero esistere punti in cui essa è tendenzialmente pianeggiante, per cui i gradienti risultano prossimi a 0 e la convergenza risulta estremamente lenta. Questo può succedere per esempio a fronte di funzioni d'attivazione che saturano, come una sigmoide. Inoltre, se la superficie degli errori non è convessa, essa può presentare *punti critici* in cui il gradiente è esattamente 0 come selle, plateau e minimi locali. I due fattori che incidono maggiormente sulla convergenza sono il learning rate ed il gradiente calcolato.

Partendo dal learning-rate, se esso è troppo basso la convergenza è lenta ed è molto probabile che l'algoritmo si fermi in uno dei punti critici. Se il learning rate è troppo alto, invece, può portare all'*overshooting*, ovvero all'oscillazione intorno ad un minimo, oppure ad una divergenza. Questi due ultimi casi sono indicati in figura 2.10.

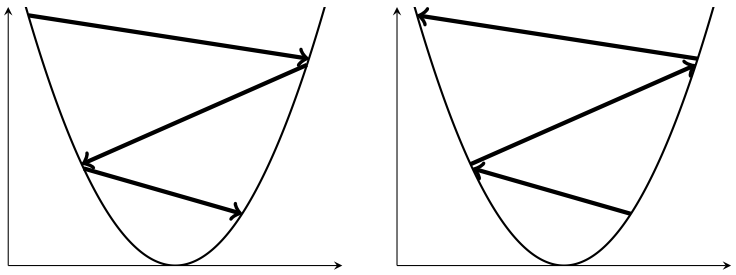

Figura 2.10: Overshooting e divergenza.

Un discorso analogo può essere fatto per il valore del gradiente. Esso però dipende da molteplici fattori, essendo calcolato su una catena di derivazione che parte dalla loss-function, passando per le varie activation-function e per i pesi. La chain-rule è moltiplicativa, quindi se si hanno gradienti molto bassi, dovuti per esempio a sigmoidi saturate, si può avere il cosiddetto *vanishing gradient*. Analogamente, se i termini della chain-rule risultano superiori ad 1, questo può portare a gradienti alti, causando l'*exploding gradient*. Questi due fenomeni possono portare a overshooting, divergenze o a stalli dell'apprendimento e risultano ovviamente più evidenti nelle DNN.

Generalizzazione : uno dei punti critici per le reti neurali è la capacità di generalizzazione, ovvero la capacità di ottenere buoni risultati sia per dati che appartengono al dataset con cui essa è stata addestrata, sia per quelli esterni ad esso. Una rete non sufficientemente complessa o con pochi neuroni può non riuscire ad approssimare la funzione desiderata ed avere un errore alto anche sul training-set (underfitting); solitamente questo è osservabile attraverso un prematuro raggiungimento di un asintoto nella curva della loss. Una rete troppo complessa e/o addestrata per troppe epoche invece potrebbe imparare a riconoscere anche i dettagli irrilevanti dei dati del training-set e quindi divenire poco precisa per nuovi dati (overfitting). In figura 2.11 si può osservare come, al variare del numero di epoche di addestramento, l'errore sul training-set continui a decrescere, mentre l'errore sul test-set scenda per poi risalire; questo accade perché la rete sta imparando a mappare esattamente i dati

di training sui propri output.

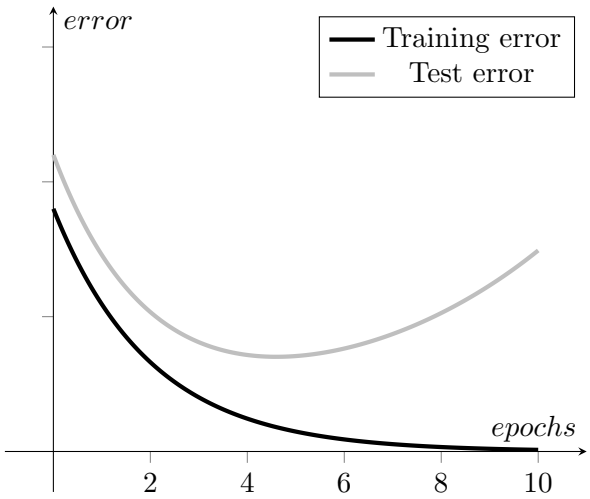

Figura 2.11: Errore nel training-set e nel test-set.

Un ulteriore problema rilevante per la capacità generalizzante è dovuto alla qualità degli ingressi e alla loro *dimensionalità*. Se molti degli elementi del vettore di input risultano irrilevanti per l'output desiderato, la rete potrebbe usare buona parte delle proprie risorse per rappresentare porzioni irrilevanti del dominio d'ingresso: ad esempio, per classificare il benessere di una persona, dati come la carta d'identità o il numero civico risulterebbero inutili e la rete potrebbe cercare di inferire una funzione che dipenda anche da essi. Inoltre, anche quando la rete riesca a focalizzarsi sulle porzioni rilevanti del dominio d'input, il numero di dati necessari per discernere ciò che è importante da ciò che non lo è crescerebbe esageratamente [15]. Risulta noto infatti che il numero di dati necessario per il training della rete cresce esponenzialmente con la dimensione del vettore di input; questo problema è noto come la *maledizione della dimensionalità* [16].

2.3.1 Learning rate decay

Un possibile approccio per migliorare la convergenza di una NN consiste nel partire da un learning-rate sufficientemente alto ma che non porti a

divergenze, decrescendolo poi nel tempo; in tal modo la funzione di perdita all'inizio del training scende rapidamente lungo la superficie degli errori, rallentando progressivamente la velocità in modo che, in prossimità di un minimo, si eviti l'overshooting [17]. Nel progetto si sono considerate due possibili strategie di riduzione del tasso di apprendimento:

- *exponential decay*: segue la curva

$$lr_{epoch} = lr_0 \cdot \alpha^{epoch} \tag{2.11}$$

  dove lr_0 è il learning-rate iniziale, *epoch* è l'epoca e α è un coefficiente di decadimento.

- *fractional decay*: seque la curva

$$lr_{epoch} = \frac{lr_0}{1 + epoch} \tag{2.12}$$

  dove lr_0 è il learning-rate iniziale e *epoch* è l'epoca.

2.3.2 Mini-batch

Nell'algoritmo del gradient-descent descritto in 2.2.1 viene usato il cosiddetto approccio *batched*: l'errore di previsione viene calcolato ad ogni epoca come media delle loss derivanti dall'insieme totale dei dati di training. Essenzialmente si effettua la propagazione per tutti gli elementi del dataset e l'aggiornamento dei pesi viene fatto con un singolo update. Nel progetto non è stato utilizzato un approccio batched in quanto presenta delle limitazioni, che verranno esposte nel seguito. Nell'approccio batched la loss media è indicativa dell'intero dataset, per cui l'applicazione del gradient-descent standard può portare ad una lenta convergenza, specialmente se l'insieme di training è molto grande; quest'ultimo potrebbe per di più non entrare in memoria. Inoltre, in presenza ad esempio di minimi locali e di selle nella superficie degli errori, è facile che l'algoritmo si fermi in questi punti critici [13]. L'idea dunque è quella di computare l'output ed la loss su *mini-batch* di dati selezionati casualmente dal training-set: questo gradiente risulta rumoroso poiché più sensibile a casi particolari (outlier), facilitando la discesa del gradiente portando potenzialmente ad evitare

o saltare i punti critici; esistono inoltre alcune tecniche (non utilizzate in questo progetto) in cui si aggiunge casualmente del rumore dopo aver computato il gradiente in modo da permettere la discesa quando vengono raggiunti alcuni punti critici come le selle [18]. Tuttavia se viene raggiunto un minimo, il gradiente risulta comunque mediamente molto piccolo indipendentemente dall'applicazione di mini-batch o gradiente rumoroso, portando quindi progressivamente verso l'arresto dell'apprendimento. Vi sono comunque altri ottimizzatori del gradiente, come il Momentum ed Adam (descritti nel seguito), in grado di gestire queste situazioni. Data la natura stocastica di questo approccio, la curva di discesa della loss presenta molte oscillazioni ed un andamento come quello mostrato in figura 2.12, dove *iterations* indica l'iterazione; per ottenere una convergenza più stabile, è possibile tuttavia decrescere il learning-rate nel tempo in modo che, raggiunto un minimo, si evitino overshooting o divergenze [17].

Figura 2.12: Andamento della loss in un approccio mini-batch.

In questo approccio, detto *mini-batched*, i mini-batch sono costituiti da un numero di elementi che si aggira tra i 32 ed i 256 e un'epoca corrisponde a $\frac{N}{mini\_batch\_size}$ iterazioni, con N numero di dati nel training-set e $mini\_batch\_size$ dimensione del mini-batch. In particolare, la combinazione di gradient-descent come ottimizzatore e approccio mini-batched viene nominata *stochastic gradient-descent* o SGD.

Ottimizzatori alternativi

L'ottimizzatore alternativo utilizzato è Adam, ma per comprendere meglio come esso funziona, verrà prima descritto un altro ottimizzatore, il Momentum.

Il *Momentum* rappresenta un'estensione dell'SGD in cui si accelera la discesa del gradiente nella direzione rilevante riducendo le oscillazioni nelle direzioni irrilevanti. Questo viene fatto prendendo in considerazione il momento, inteso qui come la derivata seconda della loss-function rispetto ai pesi. Ciò viene fatto aggiungendo al Δ_t nell'istante t una frazione γ del Δ_{t-1}, con γ detto *momentum term* e solitamente con valore 0.9.

$$\Delta_t = \eta \nabla f(w) + \gamma \cdot \Delta_{t-1} \tag{2.13}$$

In tal modo se Δ_{t-1} e Δ_t risultano nella stessa direzione, la modifica risulta più significativa rispetto al caso in cui sono discordanti o in direzioni diverse, come si osserva in figura 2.13.

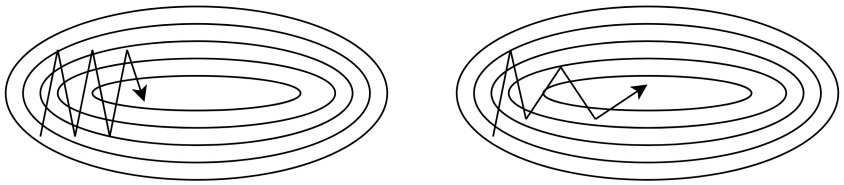

Figura 2.13: SGD senza momentum e con momentum [13].

Grazie all'accelerazione data da questo ottimizzatore, è più facile superare punti critici quali minimi locali e selle. Inoltre il numero di aggiornamenti non necessari si riduce, portando a meno oscillazioni e a convergenze più veloci [13].

Adam [19] è una tecnica adattiva in cui il Δ viene calcolato stimando il momento primo ed il momento secondo del gradiente per ogni peso, rispettivamente come media e varianza non centrata dei gradienti precedentemente applicati ad esso:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \end{aligned} \tag{2.14}$$

dove g_t rappresenta il gradiente attuale, β_1 e β_2 dei valori scalari prossimi ad 1, mentre m_t e v_t sono rispettivamente la media e la varianza dei gradienti. Poiché quest'ultimi sono inizializzati a 0, i loro valori tendono a rimanere molto piccoli, specialmente nelle prime fasi del training; pertanto, i due momenti corretti utilizzati nell'aggiornamento dei pesi sono i seguenti:

$$\begin{aligned} \widehat{m_t} &= \frac{m_t}{1-\beta_1} \\ \widehat{v_t} &= \frac{v_t}{1-\beta_2} \end{aligned} \tag{2.15}$$

Il Δ_t viene poi calcolato nel seguente modo:

$$\Delta_t = \frac{\eta}{\sqrt{\widehat{v_t}} + \epsilon} \cdot \widehat{m_t} \tag{2.16}$$

con ϵ scalare positivo molto piccolo per evitare divisioni per 0 [13].

2.3.3 Inizializzazione dei pesi

L'inizializzazione dei pesi è un fattore che incide molto sulla convergenza della rete. Se i pesi vengono inizializzati a valori troppo grandi, le somme pesate in ingresso ai neuroni potrebbero provocare saturazioni delle activation-function. Dall'altra parte, se i pesi vengono inizializzati a valori troppo piccoli, gli output dei neuroni tenderanno ad essere prossimi allo 0. In entrambi i casi si avrà nella backpropagation un vanishing gradient. Inoltre i pesi non possono essere inizializzati tutti al medesimo valore (ad esempio 0) in quanto la rete non convergerà, fatta eccezione per i casi banali. Infatti in questo caso gli output di ogni unità nello stesso layer saranno gli stessi e quindi anche i contributi calcolati mediante la backpropagation avranno il medesimo valore: questo porterebbe ad una modifica costante distribuita su tutta la rete.

Un modo usuale di inizializzare i pesi è quello di usare una distribuzione normale con media 0 o una distribuzione uniforme centrata sullo 0. Questo garantisce da una parte che ci siano sia pesi positivi che negativi, dall'altra che siano casuali e quindi vengano applicati aggiornamenti

sufficientemente diversificati. Tale peculiarità deriva dal concetto fisico di *simmetry breaking* ovvero un fenomeno per cui, tramite fluttuazioni infinitesimali indipendenti per ogni variabile, uno stato inizialmente caotico ma mediamente simmetrico viene modificato fino al superamento di un determinato "punto critico" e quindi alla conseguente rottura della simmetria iniziale e al raggiungimento di un equilibrio stabile. Tuttavia queste inizializzazioni non tengono conto del numero di input di un neurone, il quale ne influenza l'ampiezza dell'ingresso e quindi è correlato con i problemi sulla dimensione dei pesi sopra descritti.

L'inizializzazione utilizzata nella tesi è quella proposta da Xavier Glorot e Yoshua Bengio e nota come *Xavier initialization* [20]; essa tiene conto delle dimensioni dei layer in modo da bilanciare le due distribuzioni e risolvere i vanishing gradient nella backpropagation. Nello specifico si è utilizzata l'inizializzazione uniforme data dalla seguente equazione:

$$w = uniform\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}}\right] \tag{2.17}$$

dove w è il peso da inizializzare, n_{in} il numero di input layer e n_{out} la dimensione del layer in questione. In presenza di layer di dimensione molto grande, questa inizializzazione porta a pesi eccessivamente piccoli e quindi alle conseguenze già descritte in precedenza.

2.3.4 Regolarizzazione

Le tecniche di regolarizzazione hanno lo scopo di aumentare la capacità di generalizzazione di una NN riducendo l'overfitting e quindi la differenza tra l'errore nel training-set e nel test-set. Nel progetto i dati di training sono relativamente pochi, quindi si è fatto uso di forme di *Data Augmentation* per generare più osservazioni. Inoltre, per evitare che le reti imparassero troppo i dettagli dell'insieme di training, si è considerato l'uso di un *Early stop*. Altre tecniche di regolarizzazione come il *Dropout* sono state prese in considerazione per lo sviluppo di un'ulteriore rete, nota come Stacked Denoising Autoencoder (si veda sezione 2.4.2), per l'ottenimento di vettori numerici per le reti che rappresentassero in

modo omogeneo dati provenienti da fonti eterogenee. Infatti i dati di training per il caso di studio provengono da due dataset nello stesso dominio ma rappresentati in modo differente. Queste tre forme di regolarizzazione vengono descritte nel seguito.

Early stop : come si è osservato precedentemente, mentre l'errore sul training-set continua a scendere con il numero di epoche di training, l'errore sul test-set inizialmente scende per poi ricominciare a salire. Ciò significa che la rete si sta specializzando a mappare esattamente i dati di training nell'output. L'idea dell'*early-stop* consiste nell'usare un validation-set su cui calcolare periodicamente la loss durante l'addestramento in modo da interrompere quest'ultimo quando la loss riprende a salire [21]. È possibile comunque che la loss sull'insieme di validazione oscilli nelle varie epoche, raggiungendo in un secondo momento una loss inferiore a quella precedentemente trovata. Per tale ragione, se si interrompe il training appena la loss risulta maggiore di quella computata all'epoca precedente, la rete potrebbe non raggiungere la massima generalizzazione. Un'eventuale alternativa può consistere nell'eseguire un intero ciclo di training e a posteriori trovare l'epoca con il più basso errore nel validation-set.

Data augmentation : per ridurre l'overfit la rete ha bisogno innanzitutto di molti dati di training. A volte però quest'ultimi sono limitati, per cui è possibile sintetizzare nuovi dati tramite ad esempio interpolazione oppure trasformazioni: per esempio, nella classificazione delle immagini, è possibile applicare semplici modifiche come rotazioni o spostamenti. La generazione di nuovi dati risulta semplificata nell'ambito della classificazione, anche se è necessario ricordare che le trasformazioni sui dati devono essere sufficientemente sensate da non cambiare la classe stessa: se una variazione dell'input causa un cambiamento della categoria di appartenenza, questo può portare ad un degradamento della capacità di generalizzazione. Una tecnica di data augmentation di interesse è il *rumore*, in cui si aggiunge un valore casuale ad ogni elemento del vettore di input della rete neurale durante il training [15]. Questa tecnica può migliorare molto i

risultati se l'intensità del rumore è finemente calibrata [22]. Infatti, se esso è sufficientemente piccolo, i nuovi dati generati ricadranno in un intorno del dato originale, mentre l'output rimarrà essenzialmente lo stesso. Per tali ragioni solitamente si applica un rumore gaussiano con media 0 e varianza molto piccola.

Dropout : l'idea del *dropout* è quella di rimuovere casualmente e temporaneamente alcune connessioni nella rete durante ogni elaborazione di un dato durante la fase di training in modo che i neuroni siano meno sensibili ai pesi dei nodi a monte, rendendo il modello più robusto [23]. Quando un input arriva ad un determinato neurone, c'è una probabilità p che il neurone propaghi un output ed una probabilità $1-p$ che il suo output sia 0. Nel secondo caso il valore di uscita del nodo rappresenterà un input nullo per i neuroni a valle, i quali saranno obbligati a lavorare con una differente configurazione di dati in ingresso. Si tratta essenzialmente di una forma estrema di *bagging*: ogni diversa configurazione (connessioni ridotte casualmente) della rete funge da regolarizzatore per le altre [24]. È possibile imporre il dropout su qualsiasi layer, compreso l'input-layer. In figura 2.14 si può osservare una rete senza dropout e con dropout.

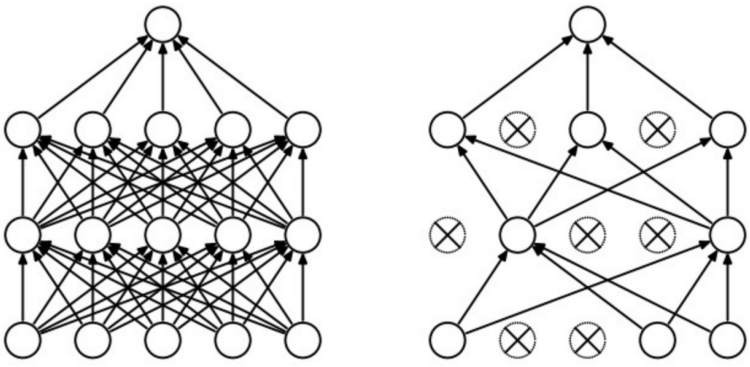

Figura 2.14: Rete senza dropout e con dropout [23].

L'applicazione del dropout rende inoltre la propagazione sparsa e più facilmente computabile per motivi analoghi a quelli descritti per le funzioni d'attivazione ReLu. La probabilità p diviene un ulteriore

iperparametro da configurare e da valutare durante la scelta del modello.

2.3.5 Riduzione della dimensionalità

Come anticipato, la dimensionalità dello spazio degli input risulta un fattore limitante per l'addestramento della NN. Tuttavia è possibile, specialmente se si hanno pochi dati di training, ridurre la dimensione del vettore di input tramite oppurtune tecniche di selezione degli attributi o *feature-extraction*. Nel progetto i dati di riferimento sono relativamente pochi ed hanno una grande dimensionalità per cui è stato necessario introdurre delle tecniche di riduzione della dimensionalità. Tramite quest'ultime è possibile di fatto rimuovere attributi inutili oppure attributi ridondanti semplificando il dominio d'ingresso, facilitando il training ed alleggerendo il modello della NN. Infatti, con un vettore di ingresso limitato, il numero di neuroni necessari ad ogni layer risulta inferiore. La selezione degli attributi può essere fatta ad esempio tramite degli Autoencoder, il cui scopo è quello di estrarre feature interessanti dai dati e darne una rappresentazione sub-simbolica di dimensione inferiore. Nell'ambito della classificazione, la feature-extraction risulta facilitata in quanto è possibile applicare delle tecniche che tengano conto della relazione dei vari attributi con la classe attesa al fine di selezionare quelli che risultano statisticamente più rilevanti.

2.3.6 Ottimizzazione degli iperparametri

Nella definizione di un modello di rete neurale artificiale vengono introdotti molti iperparametri, come ad esempio la dimensione di ogni singolo layer, il tipo di funzioni di attivazione, l'ottimizzatore del gradiente o il tipo di learning-rate decay. Al fine di scegliere gli iperparametri migliori si utilizza solitamente un validation-set e si eseguono diversi possibili addestramenti al fine di trovare la combinazione di iperparametri che massimizzi il grado di generalizzazione della rete. Per tale ragione è applicabile in tutti e soli quei casi in cui vi sono delle metriche di valutazione che possano dare una stima della performance della rete. Tuttavia per reti non

supervisionate addestrate a comprimere dati classificabili, come l'Autoencoder usato nella seguente tesi, è possibile ottimizzarne gli iperparametri cercando di massimizzare il grado di generalizzazione dei classificatori a valle. Una tecnica esaustiva per la definizione del migliore modello è la *grid-search* in cui, per ogni iperparametro, viene definito un set di possibili valori discreti che esso può assumere, mentre la ricerca del miglior insieme di iperparametri viene effettuata esplorandone tutte le possibili combinazioni. Per tale ragione la complessità della grid-search è esponenziale ed è necessario selezionare dei set finiti di valori ragionevoli per ogni iperparametro: se gli iperparametri sono n ed hanno m possibili valori ciascuno, il numero di training e di valutazioni da fare, se non si usa una k-fold cross-validation, è pari a m^n.

2.4 Neural Network utilizzate

2.4.1 Reti ricorrenti

La rete ricorrente, o RNN, è una particolare famiglia di NN in cui vi sono alcune connessioni di feedback, ovvero dei cicli o loop all'interno della struttura della rete. Come è stato anticipato, la presenza di cicli permette di analizzare sequenze temporali. Infatti, dal punto di vista del grafo computazionale, è possibile eseguire il cosiddetto *unfolding* della struttura come quello mostrato in figura 2.15 in modo da ottenere una versione "feedforward" della rete di arbitraria lunghezza che dipenda da una sequenza di ingressi: mentre i pesi ed i bias del blocco S sono condivisi, ogni output h_t dipende dall'elaborazione da parte della rete di tutti gli input x_i con $i \in [0; t]$. Il numero di blocchi della versione unfolded dipende essenzialmente dalla lunghezza della sequenza che si intende analizzare.

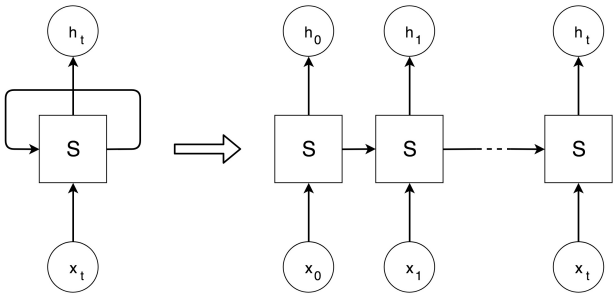

Figura 2.15: Unfolding di una rete ricorrente [25].

Ciò che distingue la RNN da una feedforward è dunque la condivisione di uno *stato* (pesi e bias) tra gli elementi della sequenza, per cui ciò che viene memorizzato all'interno della rete rappresenta un pattern che lega temporalmente gli elementi delle serie che la RNN analizza.

L'esempio mostrato presenta un output per ogni input, ma nella realtà è possibile mascherare parte degli input o parte degli output in modo da ottenere diverse combinazioni, alcune mostrate in figura 2.16. Ad esempio è possibile usare una many-to-one per classificare una sequenza di dati con un singolo output, oppure usare un one-to-many per etichettare a partire da un'immagine l'insieme di soggetti presenti.

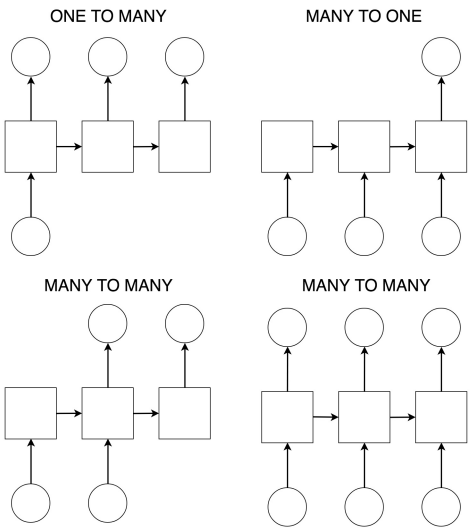

Figura 2.16: Alcune possibili combinazioni in una RNN.

La RNN più semplice è la *Vanilla RNN* e presenta un singolo layer che riceve in ingresso una concatenazione dell'input x_t e l'output h_{t-1} prodotto dalla rete all'istante $t-1$.

L'apprendimento della rete non è molto diverso da quello mostrato in sezione 2.2.1; la differenza è dovuta essenzialmente alla presenza di pesi comuni tra i vari istanti temporali per cui, una volta ottenuta la struttura unfolded, la modifica dell'unico stato viene fatto nel rispetto di tutti gli output: il Δ di aggiornamento è quindi computato come una somma di gradienti [26]. La backpropagation nelle RNN soffre chiaramente di vanishing o exploding gradients in quanto la struttura unfolded è essenzialmente una DNN; questo implica che le RNN non sono in grado di apprendere dipendenze temporali molto estese [27].

Long Short-Term Memory

La *Long Short-Term Memory* [28], o più semplicemente LSTM, rappresenta una particolare rete ricorrente proposta da Hochreiter e Schmidhuber al fine di risolvere il problema del vanishing ed exploding gradient che comprometteva l'efficacia delle RNN. Il principio alla base delle LSTM è la *memory cell* o cella di memoria, la quale mantiene lo stato c al di fuori del normale flusso della rete ricorrente. Lo stato infatti presenta una connessione diretta con se stesso: lo stato c_t è semplicemente una somma tra il precedente stato c_{t-1} e l'elaborazione dell'input attuale "modulati" tramite delle maschere in modo che solo parte di essi venga ricordato. Poiché la funzione d'attivazione per l'aggiornamento dello stato è di fatto l'identità, la derivata sarà 1 e quindi il gradiente nella backpropagation non svanirà ne esploderà, ma rimarrà costante attraverso tutti gli istanti temporali della rete unfolded.

Per la modulazione dei dati, l'implementazione della LSTM prevede la presenza di layer detti *gate*, i quali applicano alla concatenazione dell'input x_t e dell'output h_{t-1} un'attivazione sigmoide, ottenendo un vettore in uscita i cui elementi appartengono al range $(0; 1)$. Tale vettore può dunque essere usato come maschera da moltiplicare element-wise (elemento per elemento) ai vettori interessati. Usare una sigmoide invece che una

semplice binary-step risulta più efficace sia perché è differenziabile, sia perché essa indica una quantità del dato da ricordare invece di offrire un'uscita dicotomica che indichi solamente se ricordare o non ricordare. I gate sono 3:

- *forget gate* (f): viene usato per mascherare lo stato c_{t-1} in modo che solo parte dello stato venga considerato allo step t.

- *input gate* (i): viene usato per mascherare quali nuove informazioni verranno immagazzinate nello stato. Quest'ultime sono elaborate da un'ulteriore layer, solitamente con tangente iperbolica come attivazione, per garantire che la somma che aggiorna lo stato coinvolga anche valori negativi e quindi sottrazioni.

- *output gate* (o): viene usato per mascherare l'output della rete. Quest'ultimo è dato da una semplice applicazione di una tangente iperbolica allo stato corrente.

In figura 2.17 si può osservare la struttura della LSTM, dove le operazioni "x" e "+" indicano prodotti e somme element-wise, i blocchi rettangolari sono dei layer, *tanh* indica una tangente iperbolica, x l'input, h l'output e c lo stato.

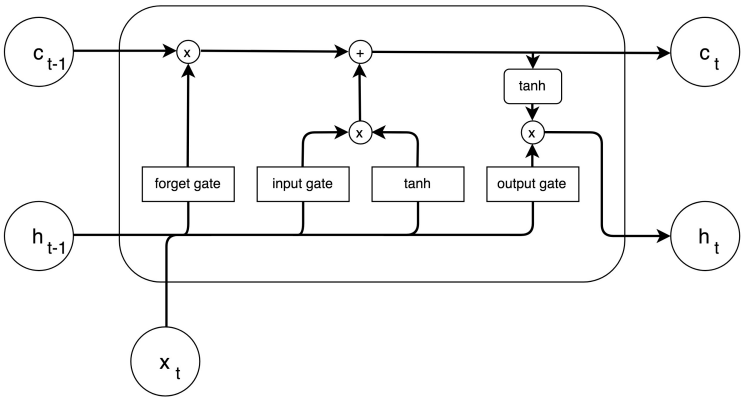

Figura 2.17: Schema di una LSTM [25].

2.4.2 Autoencoders

L'*Autoencoder* [29] rappresenta una particolare famiglia di reti neurali artificiali feedforward usate per l'apprendimento non supervisionato. Lo scopo di un Autoencoder è quello di inferire una funzione che riesca ad estrarre le caratteristiche interessanti di un insieme di dati in ingresso al fine di ridurne la dimensione.

A tal fine l'Autoencoder viene addestrato a copiare il proprio ingresso x in uscita previa riduzione della dimensione tramite un layer centrale con un numero di neuroni inferiore, come si può vedere in figura 2.18. Grazie a tale struttura a clessidra, la rete deve imparare a ricostruire il dato originale basandosi su un'informazione ridotta, per cui è vincolata a dare la priorità agli input e alle feature che rappresentano maggiormente il dato originale.

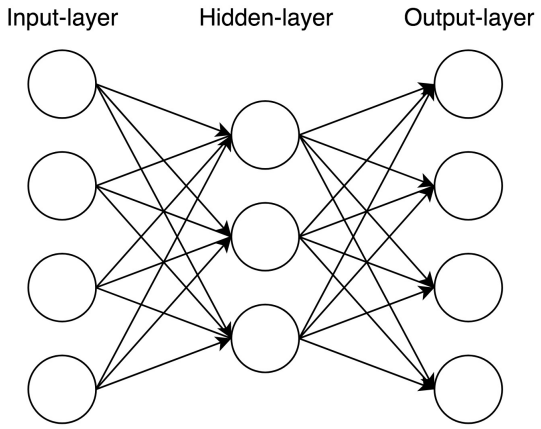

Figura 2.18: Schema di un Autoencoder.

L'hidden-layer funge dunque da *encoding-layer* (codifica) ed il suo output, detto *code*, equivale ad una rappresentazione sub-simbolica ridotta del dato originale; l'output-layer ricopre invece il ruolo di *decoding-layer* (decodifica) e la sua uscita viene poi confrontata con l'input per il calcolo della funzione di perdita durante l'addestramento della rete. Per tale ragione, nonostante si usi un apprendimento non supervisionato, è possibile applicare il gradient-descent e la backpropagation esattamente come avviene nel caso supervisionato. Una peculiarità dell'Autoencoder è che i

pesi, esclusi i bias, che si hanno nell'hidden-layer e nell'output-layer sono gli stessi, solamente trasposti per mantenere la compatibilità dimensionale; i bias invece sono diversi in fase di codifica ed in fase di decodifica. Una volta addestrata la rete, è possibile usare l'encoding-layer come estrattore di feature, mentre il decoding-layer come generatore di nuovi dati. Risulta infatti possibile utilizzare quest'ultimo per fare data-augmentation quando il dataset usato per addestrare la rete è limitato.

È possibile inoltre rendere l'Autoencoder più robusto applicando alcune forme di regolarizzazione come il dropout ed il rumore. Esiste una particolare categoria di Autoencoders, detta *Denoising Autoencoder* [30], in cui si cerca di ricostruire l'input originale x a partire dall'input corrotto $\tilde{x}$; per ottenere quest'ultimo si può ad esempio applicare un rumore gaussiano. In questo caso, durante il training, l'output della rete viene confrontato con l'input originale x e non con quello corrotto. In fase di codifica invece non si applica alcun rumore.

Stacked Autoencoder

Lo *Stacked Autoencoder* è una DNN e rappresenta la versione stacked (impilata) di un numero variabile di Autoencoder. Ogni Autoencoder aggiunto all'architettura di rete viene innestato tra l'encoding-layer ed il decoding-layer più interni.

Dunque l'output dell'encoding-layer di ogni Autoencoder k rappresenta l'input dell'Autoencoder $k + 1$ mentre l'uscita del decoding-layer di $k+1$ diviene l'input del decoding-layer di k. Analogamente a quanto detto precedentemente, ogni Autoencoder della struttura stacked presenta la stessa matrice di pesi in encoding e decoding e bias differenti. Un esempio dell'architettura stacked appena descritta è rappresentato in in figura 2.19.

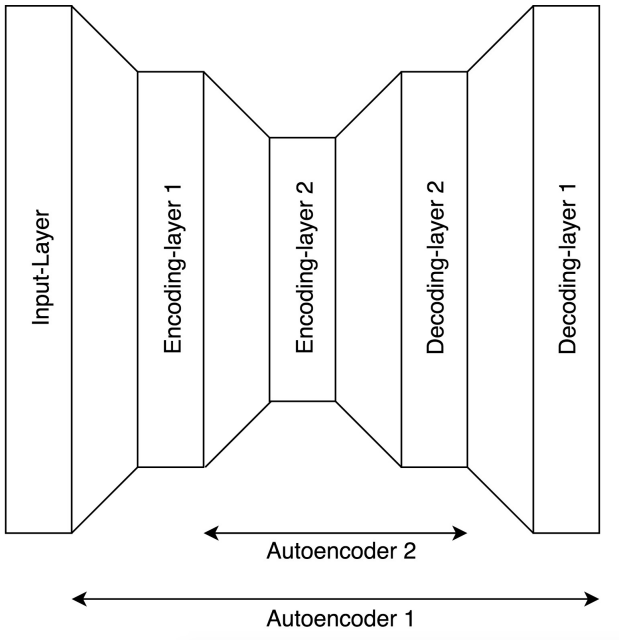

Figura 2.19: Schema di uno Stacked Autoencoder.

L'apprendimento dello Stacked Autoencoder è diviso in due fasi: il pre-training e il fine-tuning. Nella prima fase, partendo dall'esterno, si addestrano sequenzialmente in modo greedy gli Autoencoder: una volta addestrato l'Autoencoder k, si usano i dati codificati da esso come input per il pre-training dell'Autoencoder $k+1$; ovviamente gli input dell'Autoencoder 0 sono i dati originali. Il pre-training, addestrando singolarmente i vari Autoencoder, che non sono DNN, garantisce che non vi siano problemi di vanishing o exploding gradients; inoltre, si è dimostrato che ciò porta ad una migliore calibrazione iniziale dei pesi e ad una migliore generalizzazione [31]. Nella seconda fase si addestra l'intera rete stacked al fine di calibrare il comportamento della rete nel suo complesso.

Come per gli Autoencoder, anche in questo caso esiste una versione dello Stacked Autoencoder nota come *Stacked Denoising Autoencoder* (SDAE), in cui ogni Autoencoder può avere la sua forma di rumore.

Capitolo 3

Implementazione

PreventIT [32, 33, 34], come anticipato nell'introduzione, è un progetto europeo nato per la prevenzione del declino funzionale nelle persone prossime alla pensione o che hanno smesso di lavorare di recente, facilitando un sano invecchiamento. Tale obiettivo viene perseguito da una parte attraverso screening e monitoraggi continui, ad esempio tramite dei dispositivi mHealth, dall'altra promuovendo dell'attività fisica ed uno stile di vita regolare. Risulta infatti noto che un approccio preventivo basato su un'analisi dei rischi ed un tempestivo intervento può aiutare a ridurre se non prevenire la comparsa di alcune forme di perdita dell'autonomia e di disabilità [35].

Per rilevare quali siano i rischi o le cause più o meno dirette del declino funzionale e quindi individuare i soggetti più a rischio e che necessitano maggiormente di interventi geriatrici, PreventIT ha sfruttato la raccolta ed analisi di pazienti di differenti istituti di ricerca ed ospedalieri. Questi dati coinvolgono diverse aree della vita di una persona: vi sono sia resoconti di referti medici o test fisici/cognitivi, sia questionari riguardanti le sfere personali come la famiglia, il lavoro, la pensione etc. Poiché questi dati provengono da diversi istituti, il formato ed il tipo di informazioni risulta spesso eterogeneo; inoltre i report dei questionari o gli stessi referti sono talvolta mal strutturati, presentano dati mancanti o dati fortemente influenzati dalle risposte del paziente a determinate domande. Per tali ragioni l'analisi dei rischi risulta un lavoro particolarmente difficoltoso senza

l'aiuto di esperti del settore.

Un ulteriore fattore rilevante in ambito medico è la presenza di dipendenze temporali potenzialmente molto lunghe tra gli eventi clinici e la manifestazione di malattie o disabilità fisico-mentali. Per tale ragione spesso è necessario analizzare lo storico di un paziente per avere una stima più corretta del potenziale declino funzionale. A questo scopo gli istituti eseguono una raccolta periodica di dati sui soggetti.

Nonostante questo progetto di tesi non faccia parte di PreventIT, l'idea è quella di poter usufruire delle medesime risorse per poter sviluppare uno strumento in grado di eseguire un'analisi del rischio di declino funzionale che fornisca, senza l'intervento di esperti e indipendentemente dal dataset di provenienza delle osservazioni, un'unica informazione aggregata di rischio. A tale fine sono stati utilizzati due studi longitudinali europei di riferimento, l'*English Longitudinal Study of Ageing* o ELSA [36], e *The Irish Longitudinal Study on Ageing* o TILDA [37, 38, 39]. Entrambi gli istituti collezionano ogni 2 anni informazioni dettagliate delle persone anziane che partecipano ai relativi studi.

Data la natura del problema, si è optato per effettuare una classificazione probabilistica del declino funzionale mediante una Long Short-Term Memory a partire da un'analisi temporale dei controlli dei due studi longitudinali. Per validare il modello e dimostrare la presenza di possibili cause scatenanti anteriori, si è confrontato il grado di generalità raggiunto dall'LSTM rispetto a due reti feedforward, una delle quali deep, che cercano di prevedere il declino di funzionale di una persona a partire da una singola osservazione, senza tenere conto dell'intera evoluzione storica.

Data la disomogeneità dei dati forniti dai due studi longitudinali, le reti sono state addestrate fornendo degli ingressi che dessero una informazione della provenienza. A tal proposito si è pensato inoltre di modellare uno Stacked Denoising Autoencoder, a cura di Lorenzo Donati [40], per estrarre feature comuni ai due dataset al fine di ottenere una rappresentazione omogenea delle osservazioni che potesse essere usata come dato di ingresso per le reti classificatrici. Si è poi verificato se tale omogenizzazione dei dati migliorasse le performance delle NN.

3.1 Declino funzionale

Per perseguire lo scopo ultimo della seguente tesi, ovvero predire il declino funzionale di un paziente, è stato necessario innanzitutto definire quale fattore potesse indicare nel modo più appropriato tale declino. Per tale ragione si è inizialmente analizzata la documentazione dei 2 studi longitudinali di riferimento al fine di individuare gli attributi comuni che potessero indicare lo stato di salute di una determinata persona. Nonostante le forti disomogeneità nella rappresentazione delle informazioni dei pazienti, sono state rilevate alcune metriche comuni relative alla condizione fisica. Queste metriche, 22 in totale, sono facilmente comparabili tra i due dataset e coinvolgono le cosiddette ADL e IADL, le quali sono talvolta utilizzate da professionisti della salute come metro di misurazione della condizione funzionale.

Le ADL, ovvero *Activities of Daily Living* [41], rappresentano attività di base riguardanti la cura personale, per cui l'incapacità di una persona nello svolgere una o più di queste diviene indicativo di una disabilità. Le ADL includono attività nelle seguenti categorie:

- Mobilità
- Lavarsi
- Vestirsi
- Nutrirsi in modo autosufficiente
- Igiene e cura personale
- Igiene legato alla toilette

Le IADL, ovvero *Instrumental Activities of Daily Living* [42, 43], non riguardano necessariamente la funzionalità fondamentale, ma facilitano e rendono possibile la vita di una persona all'interno della comunità. Le IADL includono attività nelle seguenti categorie:

- Lavori domestici
- Preparazione dei pasti

- Assunzione dei farmaci prescritti
- Gestione dei soldi
- Spese inerenti a cibo e vestiario
- Utilizzo del telefono e di altre forme di comunicazione
- Trasporto all'interno della comunità

Le metriche ADL e IADL, all'interno dei due dataset, sono state rappresentate come abilità/disabilità, per cui è stato possibile unificarle in un unico dato aggregato che indicasse un livello di disabilità funzionale: maggiore è il suo valore, maggiore è il grado di disabilità. Nell'ambito del progetto non è stato considerato questo valore aggregato come classe da predire, sia perché la sua distribuzione all'interno dei due dataset è fortemente sbilanciata, come si può osservare in figura 3.1, sia perché non incapsula un concetto di variazione dello stato di salute.

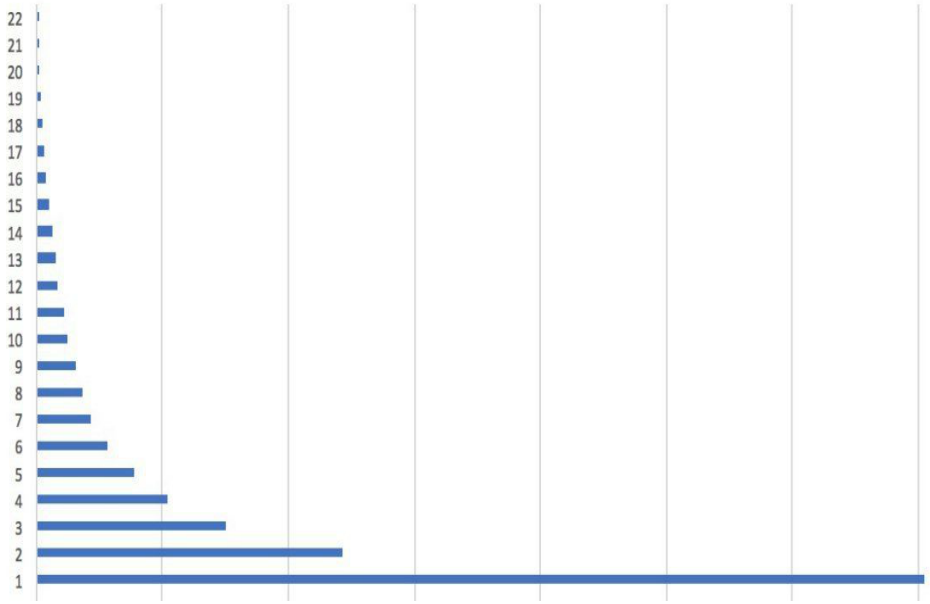

Figura 3.1: Distribuzione del valore aggregato ADL+IADL.

Per tale ragione si è optato piuttosto per la predizione dell'incremento e del non-incremento del valore aggregato ADL+IADL, la cui distribuzione risulta inoltre più equilibrata: il rapporto tra un incremento ed un

non-incremento in questo caso è infatti circa 1 : 2. È importante ricordare comunque che attualmente non vi è ancora un accordo, tra le varie scuole di pensiero, su cosa sia il declino funzionale e da cosa sia veramente rappresentato; ad ogni modo, nell'attuale caso di studio, si è optato di usare l'incremento / non-incremento delle ADL e IADL come indicatore della variazione dello stato di abilità funzionale di una persona.

3.2 Preprocessing dei dati

Una volta individuata e scelta la classe da predire, si è proceduto al preprocessing dei dati al fine di ottenere una rappresentazione adatta da usare come input per le reti neurali. Come anticipato, i due studi longitudinali prevedono la raccolta di informazioni sulle persone periodicamente anche al fine di analizzarne peggioramenti o miglioramenti nello stato di salute; per tale ragione, all'interno dei dataset, i dati di ogni persona vengono descritti attraverso una sequenza di *wave*, ovvero da una serie temporale di osservazioni. La scelta è stata quella di rappresentare ogni osservazione attraverso un vettore di input che fosse per metà rappresentativo delle informazioni di un dataset e per metà dell'altro, con l'aggiunta di opportuni ulteriori input che indicassero alle reti quale tipo di dato stessero elaborando; per ulteriori dettagli si rimanda alla sezione 3.2.3.

Innanzitutto i 2 studi longitudinali presentano delle disparità nel numero e nel tipo di informazioni fornite; infatti ELSA presenta per ogni paziente 570 attributi, mentre in TILDA il numero di attributi sale a 1680. Questi attributi inoltre non sono facilmente associabili 1 ad 1 tra i due studi sia a causa della loro disparità numerica, sia perché la documentazione può essere di difficile comprensione. Nonostante ciò sono stati rilevati alcuni attributi riguardanti la salute fisica comuni e che potevano essere rappresentati alla medesima maniera per entrambi i dataset. Un ulteriore problema relativo alla dimensione dei dataset è dato dal fatto che in ELSA vi sono quasi 110mila record divisi in 6 wave, mentre in TILDA si hanno solamente 15mila record divisi in 2 sole wave. Per quanto riguarda la rappresentazione dei dati all'interno dei due dataset, i punti da sottolineare sono essenzialmente i seguenti:

- *Dati misti*: durante l'analisi dei dataset è stata riscontrata la presenza, in TILDA, di molteplici attributi misti, in cui è possibile avere sia valori numerici che valori nominali. Un esempio notevole è stato il primo attributo considerato, ovvero il "self reported health", in cui vi erano sia i valori $[1, 2, 3, 4, 5]$, sia ["excellent", "very good", "good", "fair", "poor"].

- *Dati aggregati*: per questioni legate alla privacy, molti attributi sono stati rimossi dai dataset. Altri invece presentano valori aggregati qualora i valori reali diano informazioni sulle singole persone. Per fare un esempio poche persone hanno meno di 50 anni, per cui un'indicazione esatta dell'età minerebbe alla privacy di tali persone, oltre ad essere statisticamente poco rilevante. Per tali ragioni, subset di valori condivisi da poche persone sono stati aggregati; nel caso dell'esempio precedente, l'età per le persone sotto i cinquanta anni è stata indicata come "less than 50".

- *Wave mancanti*: all'interno di ogni studio longitudinale è possibile che alcune persone non siano presenti in tutte le wave. Vi sono infatti persone che hanno cominciato a partecipare dopo l'avvio degli studi, altre che sono assenti in alcune wave centrali, altre ancora che hanno cessato di essere presenti poiché uscite dal progetto per diverse possibili ragioni. Questo è un fattore molto importante da tenere in conto quando si vuol prevedere un rischio di declino funzionale futuro. Ad ogni modo tale problema è poco evidente in TILDA, avendo essa solamente 2 wave.

- *Dati invalidi* e *dati mancanti*: ultima ma non meno importante è la presenza di dati invalidi o mancanti. Quando una persona si rifiuta o non può rispondere ad una domanda oppure semplicemente non è presente in una wave, l'attributo corrispondente assume uno dei possibili valori che indicano un dato invalido. Infatti, in entrambi i dataset i dati invalidi sono rappresentabili da codici differenti in base al caso riscontrato.

Per tali ragioni si è proceduto ad una iniziale pulizia dei dati, per poi estrarre statisticamente un sottoinsieme rilevante di attributi al fine di ridurre il numero di nodi in ingresso alle reti e bilanciare le disparità dimensionali; infine si è scelta la rappresentazione da usare per i vettori di input e per le classi attese delle reti.

3.2.1 Pulizia dei dati

Una volta individuate le criticità nella rappresentazione delle informazioni all'interno dei dataset originali, si è proceduto ad una pulizia dei dati prima di poter eseguire delle analisi statistiche per estrarre un numero di attributi interessanti (si veda sezione 3.2.2). Innanzitutto i dati non validi, sia numerici che nominali, sono stati convertiti in un unico valore rappresentativo che si potesse distinguere da tutti gli altri valori, ovvero "unknown". In secondo luogo sono state applicate delle trasformazioni sui dati misti e sui dati aggregati al fine di renderli facilmente analizzabili durante l'estrazione degli attributi. Per esempio dati che indicavano dei range come "50-100" sono stati sostituiti dal loro valore medio.

3.2.2 Riduzione della dimensionalità

A causa della maledizione della dimensionalità, il numero di dati di training necessari per addestrare una rete neurale cresce esponenzialmente con la dimensione del vettore di input; dall'altra parte l'insieme degli esempi forniti dai due studi longitudinali risulta estremamente limitato: combinando i due dataset insieme, il numero dei record è circa 120mila, mentre gli attributi sono quasi 2000. Per tale ragione si è scelto di applicare una feature-extraction per ridurre la dimensionalità e scegliere gli attributi maggiormente significativi per la classe attesa per ognuno dei due studi longitudinali.

Lo sviluppo del progetto è stato incrementale, nel senso che si è partiti dal considerare pochi attributi, quelli riguardanti lo stato fisico delle persone e che erano comuni ad entrambi i dataset; in questo modo si è potuta verificare prima di tutto la funzionalità delle implementazioni. In un secondo momento si è proceduto ad un progressivo ampliamento del

numero di attributi in ingresso alle reti attraverso un algoritmo noto come mRMR, o *minimum Redundancy Maximum Relevance*. Alla termine del progetto il numero di attributi considerati era 80 per ognuno dei due dataset, più i singoli attributi di ADL e IADL. Quest'ultimi sono stati ragionevolmente aggiunti in quanto è plausibile che un peggioramento dello stato funzionale di una persona possa dipendere anche dalla presenza a priori di alcune disabilità.

mRMR

L'algoritmo mRMR [44] rappresenta un algoritmo greedy di selezione statistica degli attributi applicabile nel caso delle classificazioni. L'idea è quella di identificare un'insieme di attributi che siano maggiormente rilevanti (*relevance*) per la classe attesa e che siano contemporaneamente meno ridondanti (*redundancy*) tra di loro. Ciò significa che il subset scelto sarà il più significativo per la classe da predire e non vi saranno al suo interno attributi che contengano medesime informazioni, poiché scelti per essere reciprocamente molto differenti. L'algoritmo si divide in due fasi: la prima in cui il subset viene inizializzato con l'attributo maggiormente rilevante, la seconda in cui iterativamente viene scelto tra tutti gli attributi rimanenti quello più rilevante e contemporaneamente meno ridondante con gli attributi già inseriti nel subset; in questo secondo caso lo score è basato su un trade-off tra le due misure, calcolato tramite una sottrazione tra la relevance e la redundancy.

Nell'ambito del progetto gli attributi sono stati inizialmente divisi in numerici (comprensivi degli ordinali) e categorici attraverso uno script di parsing ed analisi automatica della documentazione fornitaci. Tale script individua le parole chiave che si riferiscono ai nomi degli attributi all'interno della documentazione e ne identifica il tipo (numerico o categorico). Il motivo dietro a tale separazione della tipologia degli attributi risiede nel fatto che le misure utilizzate per calcolare rilevanza e ridondanza sono differenti nel caso numerico e nel caso categorico. Attraverso l'algoritmo sono state prodotte quattro liste ordinate, 2 per ELSA e 2 per TILDA, degli attributi in base allo score ottenuto attraverso l'algoritmo. In questo

modo l'aggiunta progressiva degli attributi, effettuata tramite una estrazione top-down dalle 4 liste, ha permesso una rappresentazione bilanciata delle tipologie degli attributi e dei dataset di provenienza all'interno del vettore di input delle reti; per ulteriori dettagli sulla rappresentazione del vettore di input si rimanda alla sezione 3.2.3.

Nel caso numerico la classe di riferimento rappresenta un intero e la misura adottata è la *correlazione*: l'attributo maggiormente rilevante per la classe è quello con il più alto coefficiente di correlazione, mentre il meno ridondante è quello la cui somma dei coefficienti di correlazione con il subset di attributi già considerati è la più bassa. L'indice di correlazione viene definito dalla seguente equazione:

$$Correlazione(X,Y) = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \overline{x})^2 \sum_{i=1}^{N}(y_i - \overline{y})^2}} \tag{3.1}$$

con N numero totale dei record, x_i e y_i i valori dell'i-esimo record per gli attributi X ed Y rispettivamente, mentre $\overline{x}$ e $\overline{y}$ ne rappresentano le medie.

Per i categorici è stata invece usata, in modo analogo, la *mutua informazione*. Essa rappresenta la quantità di informazione ottenuta riguardo ad una variabile X a partire dalla conoscenza della variabile Y; per tale ragione è fortemente collegata al concetto di entropia. La mutua informazione per due variabili discrete, come gli attributi categorici, è definita dalla seguente equazione:

$$MI(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) log \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{3.2}$$

con x e y uno dei possibili valori che le variabili X ed Y possono assumere, $p(x)$ e $p(y)$ le probabilità che X ed Y assumano tali valori, mentre $p(x,y)$ è la probabilità congiunta, ovvero la probabilità che contemporaneamente X ed Y assumano i due valori x ed y.

È importante ricordare che durante l'algoritmo mRMR i valori invalidi non sono né stati presi in considerazione, né usati per determinare le medie, né per il calcolo delle probabilità congiunte.

3.2.3 Rappresentazione dei dati

In seguito alla pulizia e alla selezione degli attributi, si è scelto come rappresentare il vettore di input delle reti. In particolare gli attributi categorici e binari sono stati descritti attraverso dei vettori one-hot in cui il caso $[0, ..., 0]$ indica un dato invalido, mentre gli attributi numerici e ordinali sono stati normalizzati nel range $[0; 1]$ ed affiancati da un ulteriore attributo binario che indica con 1 la validità e con 0 la non validità del dato stesso; in quest'ultimo caso anche il valore numerico vale 0. Dopodiché si è definito il *vettore di input* in modo che potesse rappresentare una concatenazione "mutuamente esclusiva" degli attributi di ELSA e di TILDA, fatta eccezione per i 30 attributi in comune e per gli attributi indicanti le ADL e IADL: all'interno del vettore di input, per ogni record appartenente ad uno dei due dataset, i nodi che corrispondono ad attributi dell'altro dataset hanno valore 0. Tali scelte sono state fatte per alcune ragioni:

- Omogeneità: tutti gli elementi dell'input vector sono rappresentati nel range $[0; 1]$. Inoltre in questo modo si riduce la probabilità che alcune funzioni di attivazione come la sigmoide saturino.

- Computabilità: la presenza di un vettore di input sparso può portare ad alleggerimento dell'elaborazione a causa del minor numero di pesi modificati contemporaneamente.

- Dati invalidi: ogni dato non valido viene rappresentato attraverso una serie di elementi a 0; ciò è semanticamente coerente con la rappresentazione del vettore di input utilizzata per distinguere i dati dei due dataset.

- Missing wave: la mancanza di dati per una wave viene rappresentata tramite un vettore $[0, ..., 0]$.

- Applicazione dello SDAE: come si vedrà nell'implementazione dello SDAE in sezione 3.3.8, l'input-layer del primo Autoencoder presenta un dropout e la rappresentazione del vettore di input scelta

garantisce che l'applicazione della regolarizzazione in questione sia coerente.

Poiché ogni vettore di input, costituito da 321 nodi, corrisponde ad una particolare wave di una persona, se quest'ultima è presente anche alla wave successiva, allora è possibile associare al vettore una classe attesa valida che assume valore 1 se vi sarà incremento del valore aggregato ADL+IADL, 0 altrimenti; se invece la persona non è presente alla wave successiva, nessuna predizione può essere fatta, per cui viene associata al vettore una classe attesa invalida. La classe invalida ha principalmente lo scopo di discriminare quali siano i vettori di input validi su cui computare loss e metriche; infatti, mentre è possibile per una rete feedforward considerare singole wave con classe futura valida, l'LSTM elabora intere sequenze. Si è inoltre deciso, qualora una persona salti alcune wave centrali, di associare all'ultimo vettore delle missing wave una classe attesa corrispondente ad un eventuale incremento o non incremento rispetto all'ultima wave valida; il motivo dietro a questa scelta implementativa risiede nel fatto che l'LSTM, incorporando la capacità di analizzare serie temporali, potrebbe imparare a prevedere un peggioramento nel tempo anche quando la persona non risulta presente.

La *classe attesa* è stata rappresentata tramite un vettore one-hot in cui [1,0] indica un non-incremento, [0,1] un incremento e [0,0] la non validità. I motivi sono essenzialmente due: da una parte risulta necessario poter rappresentare la classe invalida per poter modulare la loss nell'LSTM, dall'altra parte perché il codice risulta in questo modo facilmente estendibile. Si potrebbe infatti pensare in futuro di applicare un binning (lineare o logaritmico) sul dato aggregato ADL+IADL in modo da risolvere parzialmente le disparità tra le classi ed eseguire poi una classificazione multi-class con classi attese rappresentate come vettori one-hot.

3.3 Ambiente di sviluppo ed implementazione

Una volta eseguita un'analisi approfondita del dominio dei dati e scelta la migliore rappresentazione di quest'ultimi, si sono implementati i modelli

delle Reti Neurali Artificiali. Si è scelto di usare Python come linguaggio di programmazione ed in particolare *Tensorflow* [45] come libreria di supporto per la modellazione di sistemi intelligenti; per il calcolo di alcune metriche, in particolare del Brier-score e dell'AUCROC, si è fatto uso della libreria *Scikit-learn* [46].

Nell'ambito del progetto sono stati poi definiti i 3 tipi di reti neurali: una feedforward non deep (1 hidden-layer), una feedforward deep e la Long Short-Term Memory. Mentre le prime due eseguono una classificazione probabilistica a partire da una singola osservazione, l'ultima applica un *sequence-labeling* per restituire una probabilità di declino funzionale per ogni wave dell'intera sequenza temporale di osservazioni per una persona. In tal modo la predizione per ogni osservazione è determinata, oltre che dalla wave attuale, anche dallo storico dei controlli precedenti. Una volta definiti i modelli, le migliori reti sono state poi selezionate tramite un'ottimizzazione degli iperparametri su un validation-set cercando di massimizzare le metriche Accuracy, Statistica Kappa, AUCROC e minimizzando il Brier-score. Le performance delle migliori reti sono state poi valutate su un test-set e confrontate tra loro mediante le stesse 4 metriche. L'utilizzo delle due reti feedforward ha il duplice scopo di verificare da una parte la necessità di DNN per risolvere il problema, dall'altra la validità del modello ricorrente dell'LSTM. Infatti, se le metriche risultano particolarmente buone ma sono molto simili per l'LSTM e per le feedforward, potrebbe non essere necessario un approccio temporale oppure, nel caso limite, potrebbe bastare una semplice feedforward non deep. Infine si è implementato uno Stacked Denoising Autoencoder per ottenere una rappresentazione omogenea dei dati provenienti dai due dataset. Tale rappresentazione è stata poi utilizzata come input per addestrare le stesse tipologie di reti classificatrici utilizzate precedentemente, al fine di misurare eventuali margini di miglioramento nella classificazione finale.

3.3.1 Tensorflow

Tensorflow è una libreria open source per il machine learning sviluppata dal Google Brain Team. Essa fornisce moduli testati ed ottimizzati

per lo sviluppo di reti neurali ed altri sistemi intelligenti in moltissimi linguaggi di programmazione, tra cui Python; inoltre supporta i principali sistemi operativi a 64bit, oltre che Android. Un punto di forza di questa libreria consiste nell'offrire API a diversi livelli di profondità, permettendo al programmatore sia di usufruire di modelli già ben assemblati, sia di poter lavorare a basso livello per definire i propri modelli o calibrarne di esistenti.

In Tensorflow i modelli vengono definiti attraverso la costruzione di un grafo computazionale: ogni nodo accetta uno o più tensori in ingresso, producendo un tensore in output. Per ottenere l'uscita dei nodi, è necessario eseguire il grafo computazionale all'interno di una *sessione*. I tipi di nodi possono essere i seguenti:

- *Constant*: sono i valori costanti del modello.
- *Variable*: sono i pesi addestrabili della rete; mantengono il proprio valore tra le varie sessioni e sono condivisibili nel grafo computazionale.
- *Placeholder*: sono delle variabili metasintattiche che non assumono alcun valore fino a che non vengono inizializzate tramite dati esterni, dunque usati per passare alla rete input e valori attesi, oltre che possibili ulteriori strutture di supporto.
- *Operation*: sono delle operazioni eseguibili sui nodi precedentemente definiti.

La loss-function ad esempio è un nodo al penultimo step del grafo computazionale, seguita da un ottimizzatore che riceve in ingresso il tensore della loss ed esegue una backpropagation sulle variabili.

Un'ulteriore feature interessante di Tensorflow è la presenza della *Tensorboard*, un server locale che permette di osservare la struttura del grafo computazionale a diversi livelli di dettaglio ed ottenere grafici temporali sugli output di qualsiasi nodo valutato durante una sessione. Ciò permette di ottenere facilmente gli andamenti della loss oppure dei pesi e dei gradienti applicati ad essi. Le seguenti figure presenti in questo capitolo sono state tutte estratte dalla Tensorboard.

3.3.2 Architetture delle reti

Le quattro reti utilizzate all'interno del progetto sono state progettate from-scratch, con qualche eccezione. In particolare le funzioni d'attivazione, gli ottimizzatori del gradiente, l'inizializzatore dei pesi, la funzione di perdita cross-entropy e la cella LSTM sono quelle già implementate in Tensorflow, in quanto risultano più stabili ed efficienti [47]. Ogni rete è stata implementata usando lo stesso pattern:

- *Init*: nell'inizializzazione della rete viene definito il modello, passando ad esso tutti gli iperparametri. Qui si definisce la struttura della rete ed il grafo computazionale, partendo dagli input e giungendo fino agli output, alla loss-function e all'ottimizzatore.

- *Train*: riceve in ingresso i dati di training e di validation e procede con l'addestramento della rete usando un early-stop. In particolare si avvia una sessione di Tensorflow, si eseguono N epoche di training, ognuna con selezioni casuali di mini-batch, fino a che non si verifica una condizione di terminazione in base alla loss del validation-set. Restituisce in uscita le metriche ottenute sul validation-set.

- *Test*: riceve in ingresso i dati di test, avvia una sessione di Tensorflow e produce gli output corrispondenti. Restituisce le metriche ottenute sul test-set.

Per quanto riguarda i modelli, le reti sono state definite seguendo le tipologie precedentemente descritte, considerando dei layer di tipo fully-connected. Ogni modello è totalmente parametrico e gli iperparametri sono semplici stringhe (es: "ReLu" per una funzione d'attivazione), numeri (es: 0.001 per il learning-rate) oppure vettori (es: [200,100] dimensioni degli hidden-layer). In questo modo la fase di ottimizzazione degli iperparametri risulta facilitata ed è possibile usare delle semplici liste Python di riferimento per definire i pool di iperparametri interessanti.

Nelle due figure esposte in seguito vengono mostrati i modelli semplificati delle reti classificatrici utilizzate, dove *optimizer* è l'ottimizzatore del gradiente, *loss* è la funzione di perdita, *metric* un nodo utiliz-

zato esclusivamente in fase di debug per osservare, durante il training, l'andamento dell'Accuracy, *input* e *target* il vettore di input e la classe attesa, *cost_vector* il vettore di costo per il cost-sensitive learning e *seq-length* il vettore delle lunghezze delle sequenze ricevute dall'LSTM. L'output delle reti è rappresentato dal/dai tensori in uscita dei blocchi *LSTM* e *Feedforward*. I dettagli verranno discussi più nello specifico nelle successive sezioni.

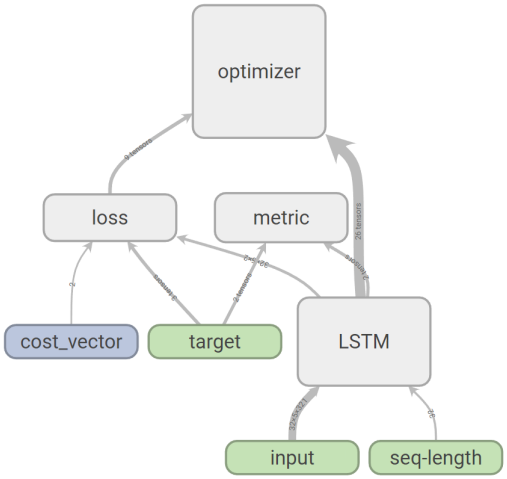

Figura 3.2: Schema della LSTM.

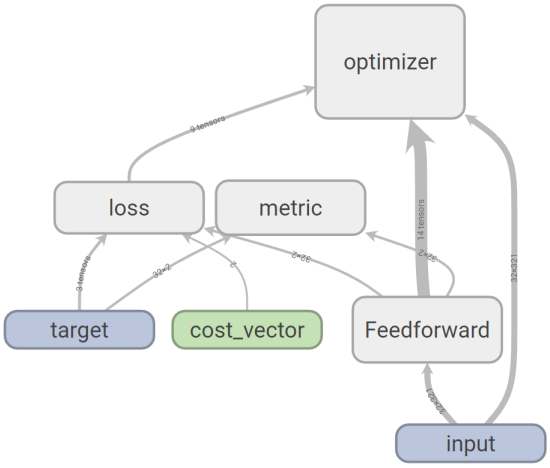

Figura 3.3: Schema di una feedforward.

3.3.3 Selezione dei dati

Per la selezione dei dati durante la fase di training si è optato per l'utilizzo di un mini-batch stocastico. Il numero di elementi per ogni mini-batch risulta un iperparametro da valutare in fase di ottimizzazione del modello. Risulta importante notare che, mentre per le reti feedforward il mini-batch è costituito da un numero N di singole osservazioni, per l'LSTM il mini-batch è costituito da N serie di osservazioni per di più di lunghezza variabile; per tale ragione, mentre nel primo caso gli input vengono selezionati dal sottoinsieme di vettori con classe futura valida, la Long Short-Term Memory necessita di un meccanismo interno per discriminare quali vettori di ogni serie temporale risultino interessanti e quale sia la lunghezza delle sequenze. Dato che i mini-batch devono essere passati come tensori all'LSTM, si sono omogeneizzate le sequenze usando la massima lunghezza possibile, ovvero 5. Infatti, poiché il numero di wave maggiore è 6 ma si deve predire la classe futura, la lunghezza massima scende a 5; per motivi analoghi, in TILDA (2 sole wave) le sequenze sono costituite in realtà da una singola osservazione e quindi l'addestramento per esse risulta paragonabile a quello che si ha per le feedforward.

Per omogeneizzare le sequenze si sono rimosse tutte le missing wave iniziali e si è eseguito un padding con vettori vuoti (per gli input) e classi invalide (per le classi attese) fino a raggiungere la lunghezza massima di 5 timestep. In questo modo tutte le sequenze partono dall'indice 0 ed è possibile passare all'LSTM un vettore della stessa lunghezza del mini-batch come descrittore delle lunghezze effettive; l'LSTM può così eseguire un unfolding dinamico. La classe invalida è stata poi usata per regolare le funzioni di perdita e le metriche.

In figura 3.4 si possono osservare gli ingressi dell'LSTM per un batch di 32 sequenze, dove *input* e *target* sono rispettivamente i vettori di input e le classi attese, mentre *seq-length* il vettore delle lunghezze; il numero 321 indica la dimensione del vettore di input.

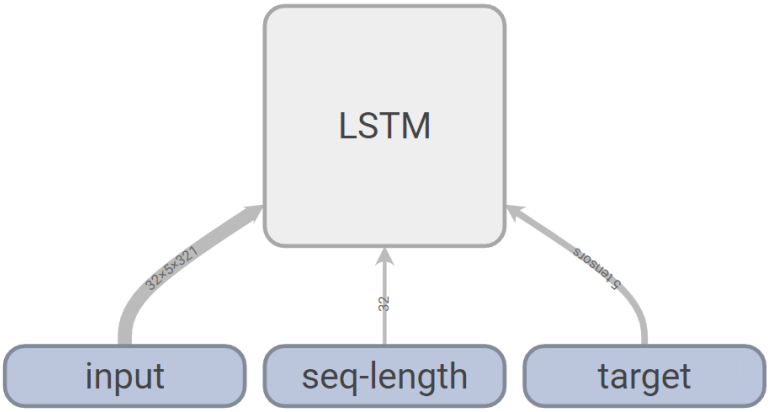

Figura 3.4: Input e vettore delle lunghezze per l'LSTM.

3.3.4 Output e funzioni di attivazione

Le reti neurali artificiali implementate eseguono una classificazione probabilistica del rischio di declino funzionale. La funzione d'attivazione dell'output-layer per tutte le reti classificatrici è stata implementata come softmax per facilitare l'interpretazione probabilistica dell'output.

Per gestire le classi invalide e per rendere il codice estendibile, è stato scelto di usare due neuroni in uscita che rappresentassero una distribuzione di probabilità discreta; questo non limita né le funzionalità della rete né l'uso delle metriche per la valutazione, poiché è possibile utilizzare il valore del solo neurone che corrisponde all'output positivo (declino funzionale) allo stesso modo in cui si utilizzerebbe il valore di un output-layer con un singolo neurone attivato tramite la funzione sigmoide.

Entrando nel dettaglio, mentre le due reti Feeforward presentano semplicemente un output-layer con 2 neuroni, l'output di una LSTM ha la stessa dimensione del suo stato, per cui è stata necessaria l'aggiunta di un ulteriore layer di proiezione con attivazione softmax per la restituzione della classe. In questo modo lo stato non è limitato ad essere descritto da 2 singoli neuroni, che potrebbero non essere sufficienti a cogliere le dinamiche temporali. Il layer di proiezione presenta anch'esso dei pesi e viene addestrato insieme al resto dei gate dell'LSTM.

Come si può osservare in figura 3.5, dato un mini-batch di 32 sequenze, se lo stato dell'LSTM è rappresentato da 50 neuroni, il *projection* rappre-

senta il layer di proiezione che converte l'insieme degli output dell'LSTM in un insieme di *output* validi e confrontabili con le classi attese.

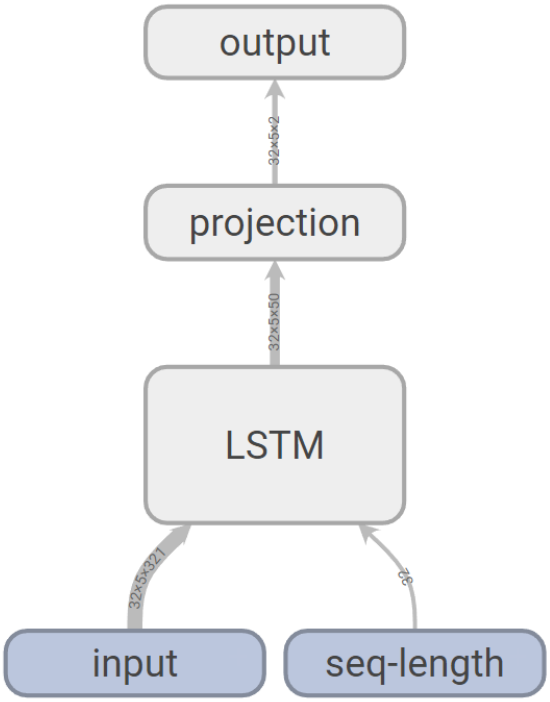

Figura 3.5: Layer di proiezione per l'LSTM.

Per quanto riguarda le reti feedforward, il numero di hidden-layer risulta un iperparametro, così come la dimensione e la funzione di attivazione di ciascun hidden-layer. In figura 3.6 si può osservare la struttura della feedforward deep, dove il mini-batch è costituito questa volta da 32 vettori di input, i *layer-1* (200 neuroni) e *layer-2* (100 neuroni) rappresentano gli hidden-layer della rete, mentre il *layer-3* l'output-layer con funzione di attivazione softmax.

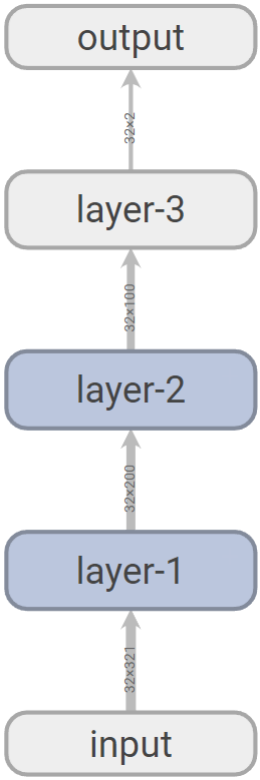

Figura 3.6: Layer della feedforward deep.

3.3.5 Funzione di perdita e cost-sensitive learning

Per la valutazione dell'errore, si è usata una loss-function di tipo cross-entropy; la scelta di questa funzione è intrinsecamente legata al tipo di valore atteso definito durante la progettazione. Poiché le due classi attese sono sbilanciate ed il numero di casi in cui non vi è declino è il doppio rispetto al numero di casi in cui la persona presenta un peggioramento dello stato funzionale, si è deciso di penalizzare l'algoritmo di apprendimento usando un cost-sensitive learning. Senza questo approccio la rete si polarizzerebbe verso la classe dominante. Conseguentemente, se l'output probabilistico venisse trasformato in crisp imponendo una soglia esattamente al 50% (che equivale a scegliere il neurone dell'output-layer con il valore maggiore), la classificazione fornirebbe molti falsi negativi per il rischio di declino funzionale. Un falso negativo potrebbe portare ad ignorare situazioni in cui è richiesto un intervento essenziale e tempestivo sul

paziente, mentre un falso positivo porterebbe eventualmente solo a prendere delle precauzioni non strettamente necessarie. In questo contesto un falso negativo potrebbe quindi essere più grave di un falso positivo.

Un ulteriore motivo dietro all'utilizzo del cost-sensitive learning risiede nel fatto che non è possibile effettuare un oversampling o un undersampling delle osservazioni poiché l'LSTM elabora delle sequenze temporali e in questo caso non si può dividere la serie senza che la rete perda di significato. Per i motivi sopra citati, sia le reti feedforward che l'LSTM sono state addestrate mediante lo stesso apprendimento cost-sensitive; in questo modo la loro accuratezza ed il loro grado di generalizzazione può essere comparato in fase di valutazione.

Implementativamente si è introdotto un vettore di costo di errata classificazione i cui elementi assumono un valore inversamente proporzionale alla frequenza delle due classi nel dataset: la loss-function viene poi moltiplicata, in base alla classe attesa di ogni singola wave, per il corrispettivo elemento del vettore di costo [48]. In questo modo la superficie degli errori cambia ed i minimi della loss-function risultano differenti da quelli che si avrebbero senza questo tipo di approccio che, come si è detto, porterebbe ad una polarizzazione verso la prima classe (non-declino).

Come si può osservare in figura 3.7, l'output prodotto dall'LSTM viene passato alla funzione di perdita *loss* insieme alle classi attese, indicate da *target*, e ad un vettore di costo *cost_vector*. Per quanto riguarda la presenza di classi invalide all'interno delle sequenze, l'errore è stato calcolato come media, all'interno di ogni batch, di tutte e sole le wave con classe valida. Infatti, grazie all'utilizzo di $[0, 0]$ come classe invalida, la cross-entropy fornisce in questi casi sempre il valore 0, per cui per ottenere la loss è sufficiente computare la somma delle cross-entropy e dividerne il valore per il numero di classi valide

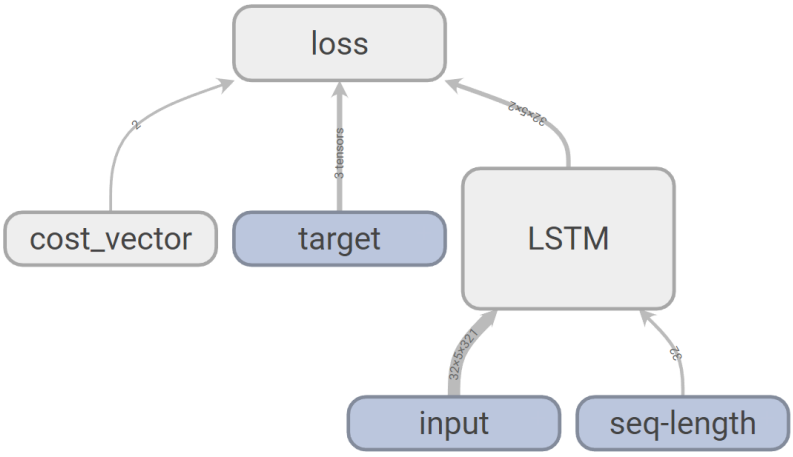

Figura 3.7: Cost-sensitive learning per l'LSTM.

Il calcolo dell'errore risulta analogo per le feedforward, ma non è necessario modulare la loss-function in base alle classi invalide in quanto esse non sono presenti nei mini-batch elaborati da queste reti. In figura 3.8 si osserva uno schema per la feedforward simile a quello visto per l'LSTM: la *loss* rappresenta la funzione di perdita, *input* e *target* mini-batch di singoli vettori di input e le classi attese corrispondenti, mentre il *cost_vector* è il medesimo vettore di costo.

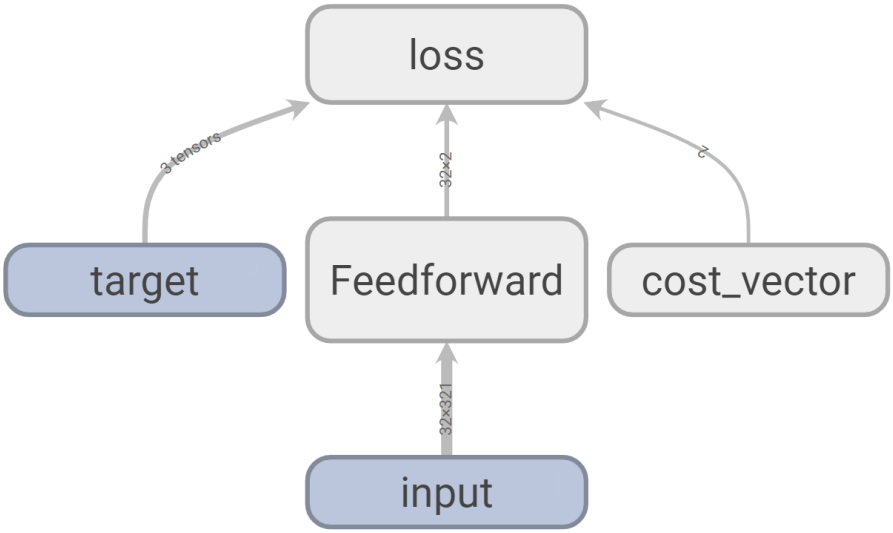

Figura 3.8: Cost-sensitive learning per il feedforward.

3.3.6 Inizializzazione dei pesi

I pesi delle reti sono stati inizializzati usando una Xavier initialization. Il problema principale di Xavier risulta la dimensione dei vettori di input a causa della formula di calcolo dei pesi che li porta ad assumere valori molto bassi in presenza di input dimensionalmente elevati, tuttavia, grazie ad una riduzione della dimensionalità applicata tramite mRMR, i vettori di input presentano un numero di nodi sufficientemente piccolo da permettere l'applicazione efficace questo tipo di inizializzazione.

3.3.7 Regolarizzazioni

Le regolarizzazioni prese in considerazione per le reti classificatrici sono essenzialmente due, l'early stop ed il data augmentation. L'early stop risulta una tecnica particolarmente efficace per ridurre l'overfitting; anche applicata da sola garantisce che la rete, se è sufficientemente complessa, abbia un buon grado di generalizzazione. Dato che la loss nel validation-set oscilla molto durante il training, si è deciso di non interrompere quest'ultimo appena la loss risale, ma di usare un parametro di look-ahead al fine di sospendere l'addestramento solo se per un numero N di epoche consecutive non vi è alcun miglioramento. Ogni volta che la loss sul validation-set risulta inferiore alla precedente miglior loss trovata, il conteggio delle epoche viene resettato. In questo modo, oltre a diminuire l'overfitting, si riducono i tempi di training e si gestiscono le naturali oscillazioni della funzione di perdita sul validation-set [21].

Tuttavia l'uso di un validation-set, che sia usato per un early stop oppure per l'ottimizzazione degli iperparametri, riduce ulteriormente l'insieme di osservazioni usufruibili. Poiché nel caso di studio l'insieme dei dati di riferimento non risulta particolarmente grande, si è fatto uso di data augmentation per generare nuove osservazioni valide. In particolare si è applicato del rumore gaussiano con media 0 e deviazione standard pari a 0.05 ai vettori di input prima dell'inserimento nel grafo computazionale. Tuttavia la presenza o non presenza del rumore è stata considerata come un ulteriore iperparametro da valutare in fase di ottimizzazione.

3.3.8 Stacked Denoising Autoencoder

Un ulteriore approccio di riduzione della dimensionalità e/o omogenizzazione dei dati affrontato è stato quello di usare uno Stacked Denoising Autoencoder per ottenere dei vettori di ingresso sintetici per le reti classificatrici che fossero rappresentativi delle feature rilevanti di qualsiasi record, indipendentemente dal dataset di provenienza. Questo approccio ricade nel campo del cosiddetto *domain adaptation*, una branca del machine learning e del transfer learning il cui scopo è quello di apprendere, a partire da una distribuzione X di riferimento, un modello che performi bene per una diversa distribuzione Y semanticamente simile ad X.

Nel caso di studio lo Stacked Denoising Autoencoder è stato modellato sulla scia dello studio di Xavier Glorot, Antoine Bordes e Yoshua Bengio sul domain adaptation nel campo del sentiment-analysis [49]. Lo Stacked Denoising Autoencoder utilizzato presenta un numero variabile di Autoencoder innestati con diverse forme di regolarizzazione in fase di training. Il primo Autoencoder presenta un dropout a differenti intensità sull'input-layer (iperparametro) ed il suo decoding-layer è costituito da una funzione d'attivazione sigmoide, mentre la funzione di perdita corrispondente è la cross-entropy su sigmoide. Quest'ultima è molto simile alla cross-entropy descritta in 2.2.4, ma considera ogni uscita dell'output-layer come una distribuzione di probabilità indipendente; l'implementazione usata è quella offerta da Tensorflow. Questi iperparametri sono stati fissati per questioni progettuali, infatti il vettore di input è costituito da elementi nel range [0;1], inoltre l'applicazione del dropout risulta consistente con la rappresentazione del vettore di input, in cui i dati non validi e mancanti vengono indicati con un numero variabile di nodi a 0. Gli iperparametri dell'encoding-layer sono stati scelti invece tra un pool di possibili valori. Per tali ragioni il code del primo Autoencoder non è limitato ad essere rappresentato da valori nel range [0;1] e le possibili loss-function degli eventuali Autoencoder innestati sono state scelte tra cross-entropy e RMSE. Per quanto riguarda i successivi Autoencoder, ognuno di essi riceve in ingresso i dati codificati dai precedenti Autoencoder ed alterati da un rumore gaussiano solo in fase di pre-training. Durante il finetuning solo

il primo encoding-layer presenta la corrispettiva forma di regolarizzazione scelta.

Poiché l'apprendimento dello Stacked Denoising Autoencoder è non supervisionato, per poter insegnare ad esso a rappresentare equamente le osservazioni per le due classi attese si è effettuato un oversampling in modo che la frequenza delle due classi fosse la medesima. Questo non risulta limitativo in quanto le forme di regolarizzazione usate dalla rete stessa risultano sufficienti a mitigare i problemi dell'oversampling. L'undersampling invece non è stato preso in considerazione poiché l'insieme di dati è già sufficientemente piccolo. Risulta inoltre importante sottolineare che lo Stacked Denoising Autoencoder è stato addestrato a rappresentare solo i vettori di input la cui classe attesa risultava valida.

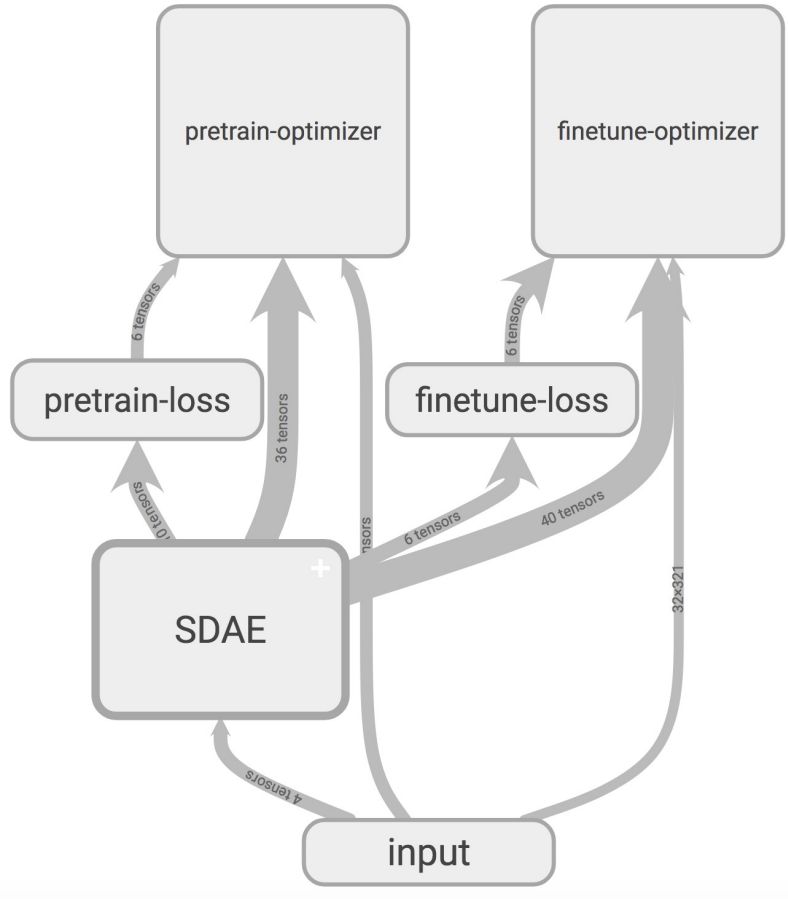

Figura 3.9: Schema dello Stacked Denoising Autoencoder.

In figura 3.9 viene mostrato lo schema dello Stacked Denoising Autoencoder, dove *input* è formato da un mini-batch di 32 vettori di input, *pretrain-loss* e *pretrain-optimizer* sono la loss-function e l'ottimizzatore del gradiente per i singoli Autoencoder durante il pretraining, mentre *finetune-loss* e *finetune-optimizer* sono la funzione di perdita e l'ottimizzatore applicati invece all'intera rete nel finetuning.

3.3.9 Ottimizzazione degli iperparametri e valutazione

Alcuni iperparametri per i vari modelli delle reti classificatrici sono stati fissati in fase di modellazione sia per i vincoli progettuali, sia per ridurre il costo computazionale della fase di ottimizzazione degli iperparametri stessi; alcuni di questi sono le attivazioni degli output-layer e l'inizializzazione dei pesi. Il restante set di iperparametri è stato scelto applicando una grid-search su un pool di possibili valori discreti:

- Dimensione dello stato dell'LSTM e dell'hidden-layer della feedforward non deep.
- Profondità e forma del feedforward deep: oltre a considerare il numero di hidden-layer, si sono esplorate sia forme rettangolari (medesima dimensione nei vari layer) che forme triangolari (dimensione del layer k maggiore della dimensione del layer $k+1$).
- Funzioni di attivazione degli hidden-layer delle feedforward: le possibili attivazioni considerate sono quelle esposte in sezione 2.2.3.
- Ottimizzatore del gradiente: le due opzioni vagliate sono state Adam e gradient-descent standard.
- Dimensione del mini-batch.
- Learning-rate iniziale.
- Learning-rate decay: le due opzioni sono l'exponential ed il fractional, mentre non è stato considerato il caso privo di decadimento poiché esso è richiesto per garantire la correttezza dell'approccio mini-batch durante l'apprendimento.

- Rumore: presenza o non presenza di un rumore gaussiano.

Il training-set, il validation-set ed il test-set sono stati selezionati tramite un semplice holdout sulle serie temporali, con una proporzione pari a 70%, 20% e 10% rispettivamente. Ciò è stato fatto selezionando tali percentuali separatamente da ELSA e TILDA ed unendo poi i set al fine di rappresentare equamente le serie temporali dei due studi longitudinali di riferimento all'interno dei tre insiemi. Quest'ultimi sono rimasti costanti per tutto il processo di training, validation e test successivo: in questo modo sia l'ottimizzazione degli iperparametri per ogni modello sia le valutazioni del grado di generalizzazione risultano comparabili. Si è optato per l'holdout per semplici vincoli temporali, ma è possibile prevedere, vista la limitatezza dei dataset utilizzati, l'uso futuro di un k-fold cross-validation per ottenere una stima migliore delle performance delle reti.

Nella fase di ottimizzazione sono state selezionate, per ognuna delle tre reti, le 4 migliori configurazioni di iperparametri in relazione alle quattro metriche utilizzate nell'ambito del progetto: Accuracy, Statistica Kappa, Brier-Score e AUCROC. In particolare, al termine di ogni fase di training, i metodi restituiscono le metriche ottenute dal validation-set con la minima loss e queste vengono confrontate con le migliori metriche fin'ora trovate; se una di esse risulta migliore delle precedenti, allora per essa viene memorizzato/aggiornato il set di iperparametri. Come è stato anticipato, le metriche utilizzano solo il neurone che descrive la probabilità di appartenenza alla classe di declino funzionale, per cui per ottenere una classificazione crisp ed applicare le prime due metriche citate si è imposta una threshold al 50%, sopra la quale si predice declino. Data la presenza di classi attese invalide, prima di calcolare le metriche sono state selezionate tutte e sole le coppie output - classe attesa valida.

Nella fase di test ogni rete è stata valutata sul test-set per le quattro metriche di riferimento alla medesima maniera, usando per ognuna di esse il corrispettivo set di iperparametri scelto. Queste metriche sono state poi analizzate ortogonalmente per osservare se vi erano sostanziali differenze di performance tra le reti per ogni metrica utilizzata.

Dopodiché si è verificato se l'uso dei vettori di ingresso omogenei e

indipendenti dal dataset prodotti dallo Stacked Denoising Autoencoder migliorassero il training ed il grado di generalizzazione delle reti trovate. Per fare ciò, si sono determinati per ogni rete classificatrice i set di iperparametri dello Stacked Denoising Autoencoder a monte che ne massimizzassero le performance. Anche in questo caso si è effettuata una grid-search su un pool di possibili valori discreti. Dopodiché si sono confrontate le metriche pre-SDAE e post-SDAE.

Capitolo 4

Risultati Sperimentali

Nel seguente capitolo verranno presentati nel dettaglio i risultati sperimentali, partendo innanzitutto col definire quali sono stati i migliori iperparametri scelti dalla grid-search per ogni rete classificatrice in relazione alle quattro metriche utilizzate. Dopodiché si mostreranno i risultati ottenuti sul test-set per ogni migliore configurazione trovata, mettendo a confronto le varie NN ed analizzando eventuali differenze di performance. Infine si osserverà se l'applicazione di uno Stacked Denoising Autoencoder per l'omogenizzazione dei dati migliori i risultati ottenuti dalle reti classificatrici.

4.1 Ottimizzazione degli iperparametri

Ogni rete è stata scelta utilizzando un pool ristretto di iperparametri per vincoli temporali. I migliori iperparametri ottenuti per ogni rete classificatrice sono in quasi tutti i casi i medesimi per tutte le metriche. L'unica eccezione risulta l'AUCROC per l'LSTM, la quale prevede esattamente gli stessi iperparametri delle altre metriche tranne la dimensione del mini-batch, che è in questo caso 32 (contro i 128 delle altre). Per tali ragioni le migliori configurazioni non sono 12 ma solamente 4: ogni rete ha un unico set di iperparametri, tranne l'LSTM che ne ha 2. Nelle liste mostrate nel seguito verranno descritti i pool di iperparametri utilizzati, evidenziando in grassetto quali sono risultati i migliori.

LSTM: per la Long Short-Term Memory gli iperparametri sono i seguenti:

- Ottimizzatore: gradiend-descent, **Adam**.
- Learning rate: 0.1, 0.01, **0.001**.
- Learning rate decay: **exponential** ($\alpha = 0.99$), fractional.
- Rumore: nessun rumore, **gaussiano** ($stddev = 0.05$).
- Dimensione dello stato: 50, 100, **200**, 300.
- Dimensione del mini-batch: **32** (AUCROC), 64, **128**, 256.

In tabella 4.1 viene presentata per ogni metrica, il valore ottenuto sul validation-set utilizzando la corrispettiva migliore combinazione.

Tabella 4.1: Metriche sul validation-set per l'LSTM.

Metric	Value
Accuracy	0.808
Statistica Kappa	0.566
Brier-score	0.140
AUCROC	0.873

Feedforward deep: per il feedforward deep gli iperparametri sono i seguenti:

- Ottimizzatore: gradiend-descent, **Adam**.
- Funzioni d'attivazione negli hidden-layer: sigmoide, **tangente iperbolica**, softplus.
- Learning rate: 0.1, 0.01, **0.001**.
- Learning rate decay: **exponential** ($\alpha = 0.99$), fractional.
- Rumore: nessun rumore, **gaussiano** ($stddev = 0.05$).

- Dimensione e forma: [100, 50], [200, 100], [150, 150], [300, 300], [200, 100, 50], [300, 200, 100], [150, 150, 150], **[300, 300, 300]**. Essi rappresentano vettori la cui lunghezza indica il numero di hidden-layer ed i valori le singole dimensioni.
- Dimensione del mini-batch: 32, 64, **128**, 256.

In tabella 4.2 viene presentata per ogni metrica, il valore ottenuto sul validation-set utilizzando la corrispettiva migliore combinazione.

Tabella 4.2: Metriche sul validation-set per la feedforward deep.

Metric	Value
Accuracy	0.825
Statistica Kappa	0.603
Brier-score	0.130
AUCROC	0.879

Feedforward non deep: per il feedforward non deep gli iperparametri sono i seguenti:

- Ottimizzatore: gradiend-descent, **Adam**.
- Funzioni d'attivazione nell'hidden-layer: sigmoide, **tangente iperbolica**, softplus.
- Learning rate: 0.1, **0.01**, 0.001.
- Learning rate decay: **exponential** ($\alpha = 0.99$), fractional.
- Rumore: nessun rumore, **gaussiano** ($stddev = 0.05$).
- Dimensione dell'hidden-layer: 100, 200, 300, **400**, 500, 600, 700, 800.
- Dimensione del mini-batch: 32, 64, **128**, 256.

In tabella 4.9 viene presentata per ogni metrica, il valore ottenuto sul validation-set utilizzando la corrispettiva migliore combinazione.

Tabella 4.3: Metriche sul validation-set per la feedforward non deep.

Metric	Value
Accuracy	0.470
Statistica Kappa	0.129
Brier-score	0.298
AUCROC	0.702

4.1.1 Considerazioni sugli iperparametri

Si può notare che, fatta eccezione per le dimensioni dei layer, per il learning rate del feedforward non deep e la dimensione del mini-batch per l'AUCROC dell'LSTM, il resto dei iperparametri risulta il medesimo per tutte e tre le reti. Si ritiene pertanto necessario introdurre qualche considerazione personale sugli iperparametri ottenuti.

La tangente iperbolica è risultata sempre migliore di softplus e sigmoide. Una motivazione potrebbe risiedere nel tipo di inizializzazione usata. Infatti, la Xavier initialization adoperata è quella basata sul paper originale [20], nella quale si definisce il range per una tangente iperbolica; tuttavia per altre funzioni di attivazione potrebbe essere necessario introdurre un fattore di scala aggiuntivo [50]. Inoltre la tangente iperbolica, rispetto alla sigmoide, presenta dei vantaggi dovuti al fatto che è centrata sull'origine ed ha una derivata leggermente superiore [14].

Per quanto riguarda l'ottimizzatore, nonostante l'approccio mini-batch garantisca dei gradienti rumorosi, la superficie degli errori potrebbe presentare molti punti critici in cui l'algoritmo con gradient-descent può comunque fermarsi. Inoltre possono essere presenti zone tendenzialmente pianeggianti dovute a possibili saturazioni delle funzioni d'attivazione, in cui il gradiente è prossimo a 0 e la convergenza risulta estremamente lenta senza una certa accelerazione. Infatti, partendo dall'LSTM, essa è costituita esclusivamente da sigmoidi e tangenti iperboliche, mentre le feedforward, secondo le considerazioni precedenti, risultano migliori con la tangente iperbolica; in tutti i casi siamo in presenza di funzioni che potrebbero saturare. L'applicazione di Adam potrebbe dunque riuscire

a gestire in modo migliore la discesa del gradiente in zone con gradienti prossimi allo 0 o che valgono esattamente 0.

Il learning rate scelto, fatta eccezione per la feedforward non deep, è lo stesso consigliato da Diederik P. Kingma e Jimmy Lei Ba nel paper in cui descrivono l'algoritmo Adam [19], quindi è probabile che learning rate più alti siano eccessivi per questo ottimizzatore, che risulta il migliore tra i due analizzati.

Il decadimento del tasso di apprendimento di tipo fractional risulta probabilmente troppo greedy: esso riduce molto velocemente il learning-rate ma le reti potrebbero aver bisogno di molte più epoche per poter giungere ad un minimo. L'exponential utilizzato, invece, risulta particolarmente lento e quindi potrebbe migliorare effettivamente la convergenza dell'approccio mini-batch, grazie anche all'utilizzo combinato con Adam.

Per quanto riguarda il rumore gaussiano, applicarne uno con deviazione standard pari a 0.05 ha effettivamente migliorato la qualità dei risultati di tutte le reti rispetto alle corrispettive versioni senza rumore, riducendo quindi l'overfitting. L'intensità scelta evidentemente non altera eccessivamente i dati, generando piuttosto nuove osservazioni valide con la medesima classe attesa, fondamentale in uno scenario come quello di questo progetto in cui gli elementi sono relativamente pochi.

La dimensione del mini-batch, pari a 128, può essere il giusto trade-off che meglio rappresenta la varianza interna dei dati dei due dataset di riferimento.

4.2 Confronto tra i classificatori

Una volta ottenute le migliori configurazioni, si sono calcolate le performance delle reti sul test-set, confrontandole poi al fine di comprendere quale sia l'approccio più corretto a questo tipo di problema. Come già anticipato, l'idea è infatti quella di valutare da una parte la performance dell'LSTM sul test-set, dall'altra verificarne il modello confrontandola con una feedforward deep e con una non deep. Nel seguito vengono mostrate, per ogni rete, le metriche ottenute sul test-set usando il corrispettivo miglior set di iperparametri.

Partendo dall'LSTM, i risultati risultano particolarmente interessanti:

Tabella 4.4: Metriche sul test-set per l'LSTM.

Metric	Value
Accuracy	0.813
Statistica Kappa	0.578
Brier-score	0.137
AUCROC	0.881

L'Accuracy mostra che l'81% delle previsioni risultano corrette, anche se la Statistica Kappa non si trova ancora nel range dei valori buoni ($[0.6; 0.8]$). Ad ogni modo l'AUCROC, che rappresenta una stima dell'ordinamento delle osservazioni rispetto all'evento da predire, è prossima al range ottimo ($[0.9; 1]$), indicando dunque una buona classificazione. Si ricorda che questa AUCROC è in realtà quella ottenuta da una configurazione leggermente differente; tuttavia usando lo stesso set di iperparametri delle altre metriche si ottiene un AUCROC solo di poco inferiore.

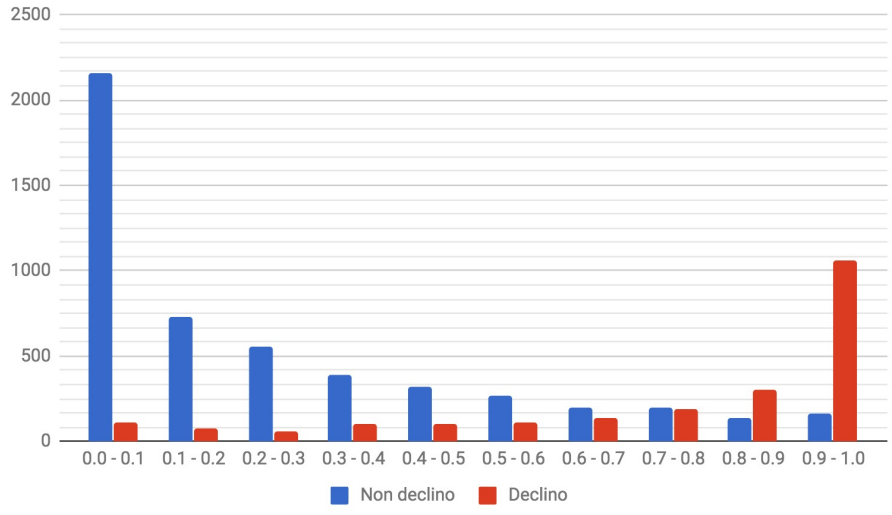

Figura 4.1: Distribuzioni degli output prodotti dalla Long Short-Term Memory per i due valori attesi.

Come si può notare dal grafico 4.1, le distribuzioni degli output prodotti dalla rete per i due valori attesi (non declino, declino) presentano il picco ai due estremi corrispondenti del dominio del neurone d'interesse. Comunque una buona porzione delle aree generate risulta in comune e le due distribuzioni scendono molto lentamente, portando alla presenza di output opposti a quelli dei valori attesi. Questo spiega il Brier-score ottenuto, il quale indica una discreta incertezza rispetto ai valori attesi.

Per quanto riguarda la feedforward deep, si sono ottenute delle metriche sul test-set leggermente migliori a quelle ottenute dalla Long Short-Term Memory:

Tabella 4.5: Metriche sul test-set per la feedforward deep.

Metric	Value
Accuracy	0.830
Statistica Kappa	0.614
Brier-score	0.129
AUCROC	0.883

L'Accuracy qui raggiunge l'83% (+2%) e la Statistica Kappa si trova nel range dei valori buoni, passando da 0.578 dell'LSTM ad un 0.614. Per quanto riguarda le altre due metriche, l'AUCROC risulta essenzialmente la stessa ottenuta dalla LSTM ed il Brier-score indica una minore incertezza.

Come si nota nel grafico 4.2, mentre la distribuzione del declino risulta molto simile a quella osservata per l'LSTM, la distribuzione del non declino presenta un picco più alto sullo 0. Intuitivamente questo potrebbe portare ad un aumento dell'Accuracy e della Statistica Kappa, unitamente ad una diminuzione del Brier-score.

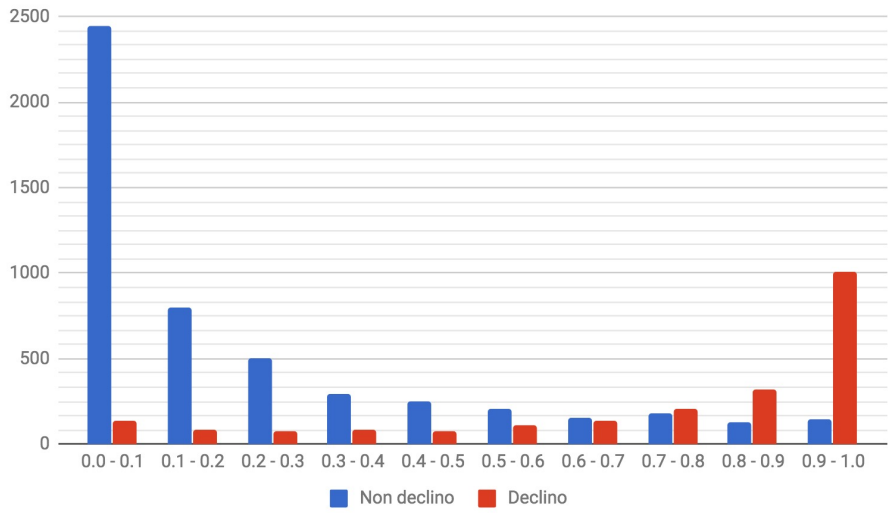

Figura 4.2: Distribuzioni degli output prodotti dalla feedforward deep per i due valori attesi.

Infine la feedforward non deep ha fornito dei risultati non promettenti. Nella tabella seguente vengono presentati i risultati ottenuti dalla rete sulle metriche di riferimento:

Tabella 4.6: Metriche sul test-set per la feedforward non deep.

Metric	Value
Accuracy	0.469
Statistica Kappa	0.127
Brier-score	0.301
AUCROC	0.698

L'Accuracy risulta in questo caso addirittura inferiore al 50% e la Statistica Kappa indica solo un leggero miglioramento rispetto al classificatore probabilistico. Il Brier-score inoltre mostra che mediamente le predizioni sono più lontane dal valore atteso rispetto al suo opposto.

Come si osserva in figura 4.3, la distribuzione per la classe declino è oltre la threshold imposta al 50%, quindi potrebbe sembrare che classifichi bene il caso di maggiore rilevanza. Il problema sta nel fatto che l'altra

distribuzione è molto sovrapposta alla prima, portando anche per i casi in cui non vi è un declino una buona probabilità di predizione del declino funzionale. Questo classificatore dunque non fornisce alcuna reale informazione sulla probabilità di declino.

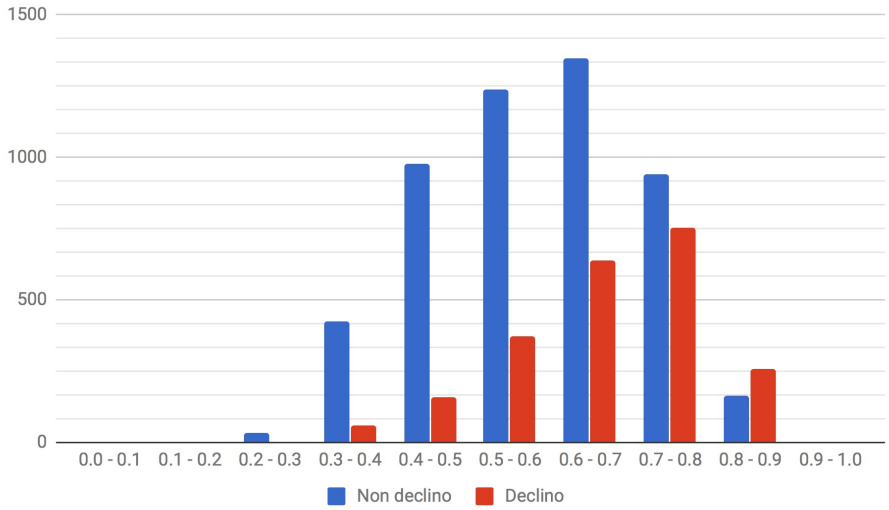

Figura 4.3: Distribuzioni degli output prodotti dalla feedforward non deep per i due valori attesi.

Le metriche ottenute sul test-set sono particolarmente simili a quelle ottenute nel validation-set e nella maggioranza delle volte risultano leggermente migliori. Il validation-set potrebbe essere stato selezionato, a causa dell'holdout, in modo approssimativo (classi non rappresentate alla stessa maniera) e per tale ragione il test-set può teoricamente ottenere risultati migliori.

Nel seguito vengono mostrati, per ogni metrica, gli istogrammi dei valori ottenuti dalle varie NN del progetto, per effettuare un confronto grafico.

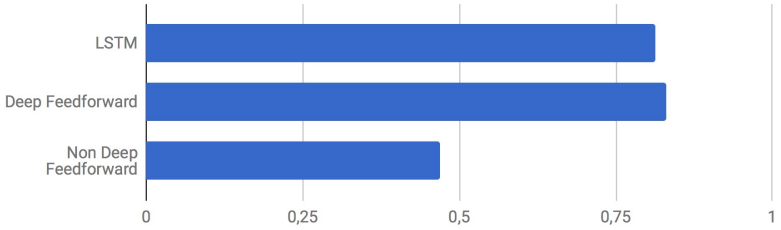

Figura 4.4: Accuracy nelle varie reti.

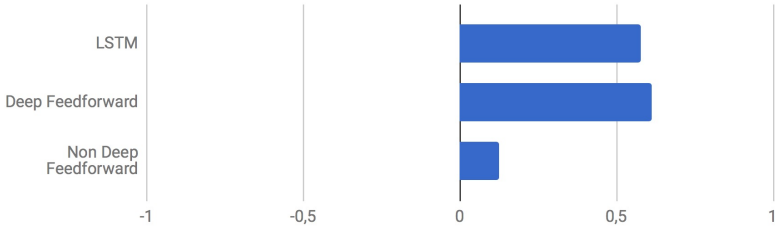

Figura 4.5: Statistica Kappa nelle varie reti.

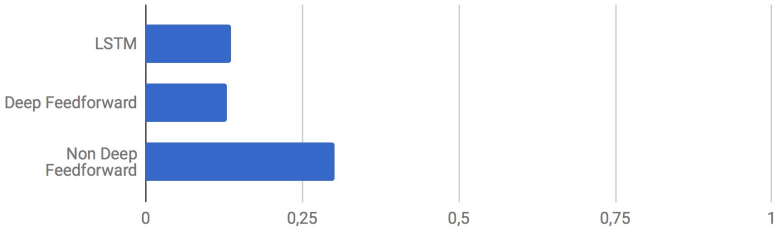

Figura 4.6: Brier-score nelle varie reti.

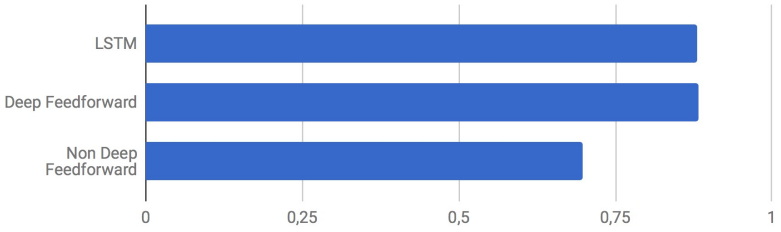

Figura 4.7: AUCROC nelle varie reti.

4.2.1 Considerazioni sui classificatori

Si può osservare dall'analisi comparata delle metriche, che la feedforward non deep non riesce ad eseguire una adeguata classificazione se confrontata con le altre due reti, le quali invece risultano particolarmente simili e performanti in termini di correttezza e incertezza della classificazione. Per l'LSTM e la feedforward deep si hanno infatti AUCROC pari a 0.881 e 0.883 rispettivamente, mentre il Brier-score vale 0.137 per la prima e 0.129 per la seconda. La feedforward risulta più precisa dell'LSTM nella classificazione del non declino, tuttavia la previsione di maggior rilievo è quella data dal declino, in cui entrambe le reti si comportano alla medesima maniera.

I risultati forniti indicano quindi che l'uso di una Long Short-Term Memory potrebbe non essere necessario per la previsione del declino funzionale e che le singole osservazioni sui pazienti siano sufficienti per ottenere una buona stima della probabilità di declino funzionale. Dall'altra parte è possibile invece che gli attributi utilizzati, essendo essi selezionati solo in base all'immediato declino funzionale (si veda sezione 3.2.2), non contengano sufficienti informazioni riguardo un eventuale declino a distanza di diverse wave. Per questo motivo gli attributi dei dataset utilizzati per addestrare la Long Short-Term Memory potrebbero non permettere alla rete di trovare dei pattern temporali che ne migliorino le performance rispetto alla feedforward deep.

4.3 Stacked Denoising Autoencoder

L'ultimo passo progettuale è stato quello di verificare se fosse possibile incrementare le performance delle tre NN utilizzate attraverso l'uso di dati d'ingresso ottenuti come codifica di uno Stacked Denoising Autoencoder. Come anticipato, le migliori configurazioni, che dovevano essere 12, sono solamente 4; per semplicità si è deciso di usare per l'LSTM solo la configurazione con mini-batch di dimensione 128. I migliori iperparametri dello Stacked Denoising Autoencoder per le tre reti presentano molti elementi in comune, simili a quelli ottenuti per le reti classificatrici:

- Ottimizzatore: Adam.
- Funzioni d'attivazione: tangente iperbolica.
- Learning rate: 0.001.
- Learning rate decay: exponential.
- Dimensione del mini-batch: 128.

Altro iperparametro comune tra le configurazioni degli Stacked Denoising Autoencoder interessante è la dimensione del code layer: nonostante lo SDAE presenti per l'LSTM un singolo Autoencoder e per le feedforward 3 Autoencoder innestati, il code layer finale risulta in tutti i casi pari a 300, leggermente inferiore del vettore di input originale (321 elementi).

I risultati ottenuti tramite l'applicazione dello Stacked Denoising Autoencoder sono i seguenti, uno per rete classificatrice:

Tabella 4.7: Metriche sul test-set per l'LSTM post SDAE.

Metric	Value
Accuracy	0.880
Statistica Kappa	0.720
Brier-score	0.096
AUCROC	0.908

Tabella 4.8: Metriche sul test-set per la feedforward deep post SDAE.

Metric	Value
Accuracy	0.866
Statistica Kappa	0.686
Brier-score	0.106
AUCROC	0.904

Tabella 4.9: Metriche sul test-set per la feedforward non deep post SDAE.

Metric	Value
Accuracy	0.791
Statistica Kappa	0.531
Brier-score	0.161
AUCROC	0.840

L'applicazione dello Stacked Denoising Autoencoder per ottenere una rappresentazione omogenea dei dati ha migliorato effettivamente le performance delle reti. In particolare, nell'LSTM e nella feedforward deep si osservano AUCROC nel range ottimo; esse infatti hanno un valore superiore a 0.9. Anche il Brier-score risulta particolarmente buono; esso infatti in entrambi casi è stato ridotto di molto, raggiungendo il valore 0.096 per l'LSTM e 0.106 per la feedforward deep. Negli istogrammi seguenti vengono mostrati i risultati ottenuti dalle tre reti classificatrici, per ogni metrica, usando il vettore di input originale (pre-SDAE) ed usando il vettore omogeneizzato (post-SDAE).

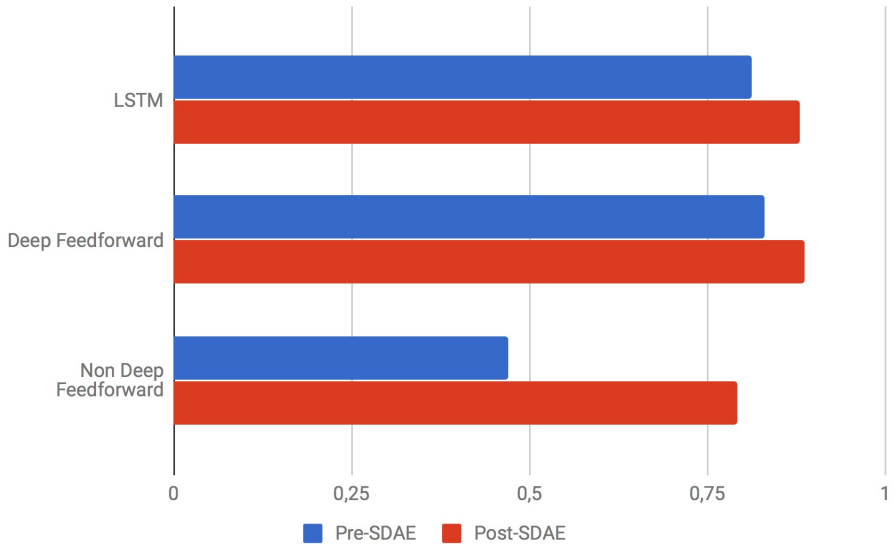

Figura 4.8: Accuracy delle reti prima e dopo lo SDAE.

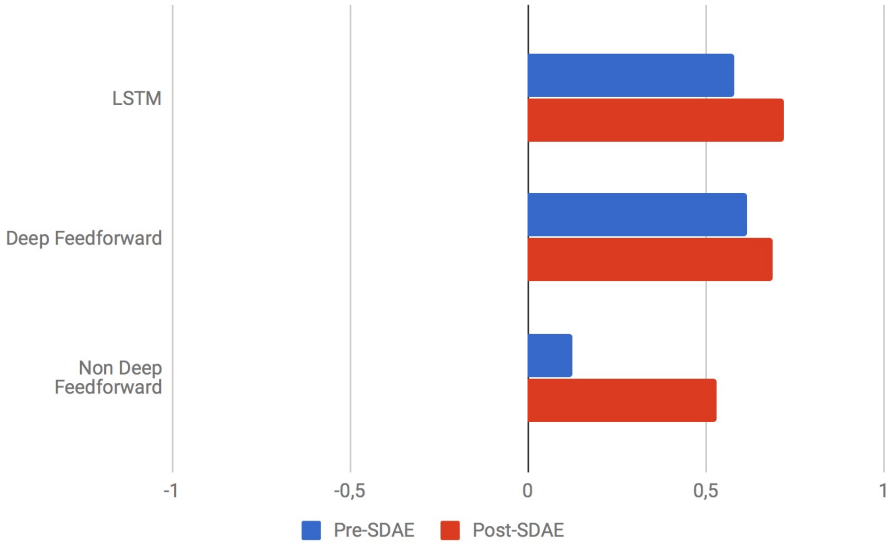

Figura 4.9: Statistica Kappa delle reti prima e dopo lo SDAE.

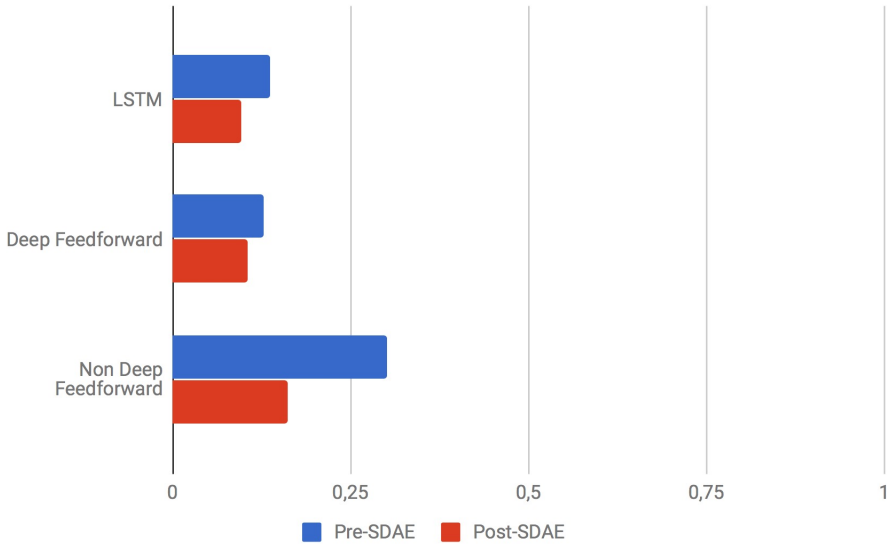

Figura 4.10: Brier-score delle reti prima e dopo lo SDAE.

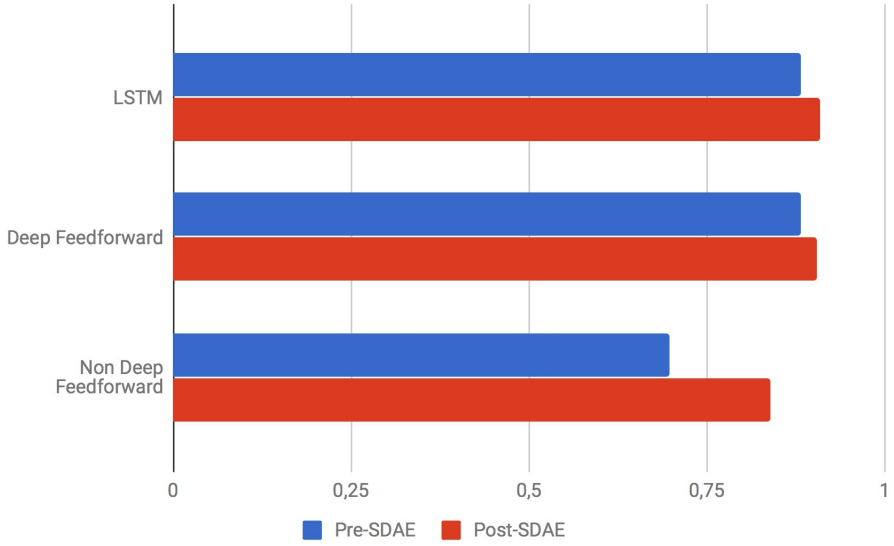

Figura 4.11: AUCROC delle reti prima e dopo lo SDAE.

Si può notare che la rete che è migliorata maggiormente è la feedforward non deep. Il motivo potrebbe intuitivamente risiedere nel semplice fatto che la concatenazione di SDAE e feedforward non deep genera una DNN, addestrata tra l'altro in maniera greedy. Inoltre si osserva che, a differenza del caso precedente, l'LSTM con i dati codificati ottiene risultati leggermente migliori del feedforward deep.

Capitolo 5

Conclusioni

I risultati ottenuti dall'applicazione di Reti Neurali Artificiali per la previsione del declino funzionale sono stati particolarmente interessanti. La Long Short-Term Memory e la feedforward deep hanno fornito un AUCROC di 0.881 e 0.883 in casi applicativi reali, dimostrando di essere in grado di ottenere ottime previsioni di declino funzionale a partire da uno scenario in cui le osservazioni risultano fortemente eterogenee. Inoltre, l'applicazione di forme di omogenizzazione dei dati e riduzione della dimensionalità come quella data dallo Stacked Denoising Autoencoder utilizzato nel seguente progetto, garantisce un miglioramento di queste reti, portando ad AUCROC pari a 0.908 (+0.027) e 0.904 (+0.021) rispettivamente, riducendo l'incertezza della classificazione. È stato inoltre osservato che l'approccio non deep utilizzato non riesce a cogliere la complessità dello spazio dei dati utilizzati ed offre dunque risultati limitati.

Tuttavia si è riscontrato che le performance della LSTM e della feedforward deep risultano molto simili. Ciò può significare da una parte che potrebbe non essere necessario l'utilizzo di una rete ricorrente e che la previsione del declino funzionale possa essere eseguita a partire da singole osservazioni, dall'altra parte che gli attributi utilizzati per addestrare le reti, essendo stati selezionati solo in funzione di un immediato declino funzionale, potrebbero non fornire sufficienti informazioni all'LSTM per collegare cause ed effetti molto distanti nel tempo.

Possibili sviluppi futuri potrebbero quindi prevedere un ampliamento

del numero di wave e del numero di attributi utilizzati per la rappresentazione delle osservazioni nelle Reti Neurali Artificiali. Si potrebbe inoltre sfruttare l'architettura per predire una classe di stato funzionale invece di predirne la variazione. Infine, per ottenere parametri migliori e valutazioni più precise potrebbe essere interessante prendere in considerazione l'uso di una k-fold cross-validation sia in fase di ottimizzazione che in fase di valutazione.

Riconoscimenti

The data relative to ELSA were made available through the United Kingdom Data Archive. ELSA was developed by a team of researchers based at the NatCen Social Research, University College London and the Institute for Fiscal Studies. The data were collected by NatCen Social Research. The funding is provided by the National Institute of Aging in the United States, and a consortium of United Kingdom government departments coordinated by the Office for National Statistics. The developers and funders of ELSA and the Archive do not bear any responsibility for the analyses or interpretations presented here.

TILDA is an interinstitutional initiative led by Trinity College Dublin. TILDA data have been co-funded by the Government of Ireland through the Office of the Minister for Health and Children, by Atlantic Philanthropies, and by Irish Life; have been collected under the Statistics Act, 1993, of the Central Statistics Office. The project has been designed and implemented by the TILDA study team, Department of Health and Children. Copyright and all other intellectual property rights relating to the data are vested in TILDA. Ethical approval for each wave of data collection is granted by the Trinity College Research Ethics Committee. TILDA data is accessible for free from the following sites: Irish Social Science Data Archive at University College Dublin http://www.ucd.ie/issda/data/tilda/; Interuniversity Consortium for Political and Social Research at the University of Michigan (`http://www.icpsr.umich.edu/icpsrweb/ICPSR/studies/34315`).

Elenco delle figure

Elenco delle tabelle

Bibliografia

1. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009.

2. Rojas Raúl. The bias-variance dilemma. Feb 2015.

3. Z Reitermanov. Data splitting. pages 31–36, 01 2010.

4. Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

5. J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1), 1977.

6. G. W. Brier. Verification of Forecasts Expressed in Terms of Probability. *Monthly Weather Review*, 78:1, 1950.

7. Rajeev Kumar and Abhaya Indrayan. Receiver operating characteristic (roc) curve for medical researchers. *Indian Pediatrics*, 48(4):277–287, Apr 2011.

8. Xinjian Guo, Yilong Yin, Cailing Dong, Gongping Yang, and Guangtong Zhou. On the class imbalance problem. In *Proceedings of the 2008 Fourth International Conference on Natural Computation - Volume 04*, ICNC '08, pages 192–201, Washington, DC, USA, 2008. IEEE Computer Society.

9. Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

10. G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

11. Troy D. Kelley. Symbolic and sub-symbolic representations in computational models of human cognition: What can be learned from biology? *Theory & Psychology*, 13(6):847–860, 2003.

12. Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997.

13. Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

14. Yann LeCun, Ido Kanter, and Sara A. Solla. Second order properties of error surfaces: Learning time and generalization. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 918–924. Morgan-Kaufmann, 1991.

15. Warren S. Sarle. comp.ai.neural-nets faq. `http://www.faqs.org/faqs/ai-faq/neural-nets/`, 2002.

16. M. Verleysen, D. Francois, G. Simon, and V. Wertz. *On the effects of dimensionality on data analysis with neural networks*, pages 105–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

17. Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

18. Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points - online stochastic gradient for tensor decomposition. *CoRR*, abs/1503.02101, 2015.

19. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

20. Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International*

Conference on Artificial Intelligence and Statistics, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

21. Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

22. Jianping Hua, James Lowey, Zixiang Xiong, and Edward R. Dougherty. Noise-injected neural networks show promise for use on small-sample expression data. *BMC Bioinformatics*, 7(1):274, May 2006.

23. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

24. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

25. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

26. Michael C. Mozer. Backpropagation. chapter A Focused Backpropagation Algorithm for Temporal Pattern Recognition, pages 137–169. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

27. Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994.

28. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

29. Geoffrey E Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In J. D. Cowan, G. Te-

sauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 3–10. Morgan-Kaufmann, 1994.

30. Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

31. Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

32. White paper of the project results in preventit (deliverable 8.2), 2017. Early risk detection and prevention in ageing people by self-administered ICT-supported assessment and a behavioural change intervention delivered by use of smartphones and smartwatches.

33. Clemens Becker Chris Todd Kristin Taraldsen Mirjam Pijnappels Kamiar Aminian Jorunn L. Helbostad, Beatrix Vereijken and Sabato Mellone. Mobile health applications to promote active and healthy ageing, 2016.

34. Lindy Clemson Elisabeth Boulton Helen Hawley-Hague Clemens Becker Michael Schwenk Michaela Weber, Nacera Belala. Feasibility and effectiveness of intervention programmes integrating functional exercise into daily life of older adults: A systematic review, 2017.

35. R Hébert. Functional decline in old age. *CMAJ: Canadian Medical Association Journal*, 157(8):1037–1045, 10 1997.

36. James Nazroo James Banks, G.D. Batty and Andrew Steptoe. *The dynamics of ageing: Evidence from the English Longitudinal Study of Ageing 2002-15 (Wave 7)*. The Institute for Fiscal Studies, 2016.

37. Rose Anne Kenny. The irish longitudinal study on ageing (tilda), 2009-2011, Jul 2014.

38. Brendan J. Whelan and George M. Savva. Design and methodology of the irish longitudinal study on ageing. *Journal of the American Geriatrics Society*, 61:S265–S268, 2013.

39. Caoimhe Hannigan, Robert F. Coen, Brian A. Lawlor, Ian H. Robertson, and Sabina Brennan. The neil memory research unit: psychosocial, biological, physiological and lifestyle factors associated with healthy ageing: study protocol. *BMC Psychol*, 3(1):20, Jun 2015. 79[PII].

40. Donati Lorenzo. Domain adaptation through deep neural networks for health informatics, 2017.

41. Linda S. Noelker and Richard Browdie. Sidney katz, md: A new paradigm for chronic illness and long-term care. *The Gerontologist*, 54(1):13–20, 2014.

42. Harrington M. Pass L. Bookman, A. and E. Reisner. *Family Caregiver Handbook*. MIT, 2007.

43. Cynthia Williams. *Healthy Aging & Assessing Older Adults*. 2011.

44. J. R. Berrendero and Alma-Delia Cuevas. The mrmr variable selection method: a comparative study for functional data. 2015.

45. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/`, 2015. Software available from tensorflow.org.

46. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

47. Tensorflow - api r1.4. `https://www.tensorflow.org/api_docs/python/`.

48. Matjaz Kukar and Igor Kononenko. Cost-sensitive learning with neural networks. In *ECAI*, 1998.

49. Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 513–520, USA, 2011. Omnipress.

50. Siddharth Krishna Kumar. On weight initialization in deep neural networks. *CoRR*, abs/1704.08863, 2017.

Ringraziamenti

Un ringraziamento va ai miei *genitori*. Grazie per aver avuto in ogni momento fede nelle mie potenzialità, permettendomi di proseguire gli studi fino a questo fondamentale traguardo.

Un ringraziamento va a *Francesca*. Grazie per l'affetto che mi dimostri, per i piccoli gesti che mi fanno capire quanto tu tenga a me. Ricorda che, nonostante la mia introversione e timidezza, ti voglio un bene dell'anima.

Un ringraziamento va a *Giulia* che, nonostante tutto, è sempre stata al mio fianco. È stato un momento difficile per te ma non hai mai smesso di credere in me e mi hai costantemente sostenuto in questo importante percorso. Ti ringrazio, sei una persona forte ed unica. Ti amo.

Un ringraziamento va a *Lorenzo* che mi ha fatto appassionare ancor di più all'informatica e con cui ho condiviso tutti i miei studi fin dalle superiori. Siamo diversi, io sono intuitivo, tu sei metodico e questo binomio perfetto ci ha portati lontano.

Un ringraziamento va a tutti i miei *amici*, vicini e lontani. Grazie di ricordarmi costantemente che sono una persona valida e che le mie sono sempre le solite paure.

Grazie di cuore a tutti voi.