Università degli Studi di Bari - "Aldo Moro"

DIPARTIMENTO DI INFORMATICA
Corso di Laurea Magistrale in Informatica

Intelligenza Computazionale

Previsione della severità in soggetti affetti da disturbi respiratori del sonno attraverso Convolutional Neural Networks

Relatori:
Prof.ssa Anna Maria Fanelli
Prof. Corrado Mencar

Laureando:
Christopher Piemonte
Matricola 659723

Anno Accademico 2016-2017

Indice

Capitolo 1

Abstract

La funzione respiratoria è influenzata da molteplici sistemi di controllo, diretti a regolare i parametri respiratori in funzione delle diverse esigenze della vita vegetativa e di relazione. Particolare interesse è rappresentato dalla modalità di adattamento correlata al ritmo sonno-veglia. Per contro il sonno è una condizione fisiologica relativamente sfavorevole per la respirazione e possono portare una patologia respiratoria o ad una anomalia anatomica delle prime vie aeree, fino a sfociare in disturbi respiratori del sonno di diversa natura e gravità.

Il lavoro svolto si propone l'obiettivo di utilizzare le Reti Neurali per produrre regole di classificazione, con le quali diagnosticare la severità del Disturbo Respiratorio del Sonno attraverso rilevazioni effettuate su pazienti. Con questo proposito è stato applicato il processo di Knowledge Discovery ai dati forniti dal Dipartimento di Scienze Mediche e Chirurgiche dell'Università degli Studi di Foggia e su queste sono state effettuate diverse sperimentazioni utilizzando modelli di Reti Neurali Fully-Connected e Convolutional.

Capitolo 2

Reti Neurali

Introdotte negli anni ‘50, ispirate all’architettura del cervello umano e promosse come approssimatori universali, le reti neurali hanno goduto di popolarità relativamente breve, in quanto sono state presto accantonate in seguito alla pubblicazione di ”An introduction to computational geometry” (Minsky, 1969), dove é stata dimostrata l’impossibilità di risolvere problemi non caratterizzati da separabilità lineare in reti a due strati. Fu solo intorno agli anni ‘80 che furono poste le basi matematiche per l’addestramento di reti neurali multi-strato da (Werbos, 1974) e (Hopfield, 82), attraverso l‘algoritmo di backpropagation. Negli ultimi anni, grazie al notevole miglioramento delle risorse computazionali, sono tornate oggetto di nuovo entusiasmo.

2.1 Modello matematico

Il neurone è l’elemento basilare del sistema nervoso, tanto che può essere considerato l’unità di calcolo primaria alla base della mente umana. In figura è mostrata la rappresentazione biologica di un neurone ed il modello matematico ad esso ispirato: ogni neurone riceve in input il segnale dai suoi dendriti e, una volta elaborato, produce un segnale di output lungo il suo unico as-

sone, che una volta diramatosi lo collega ai neuroni successivi attraverso le sinapsi.

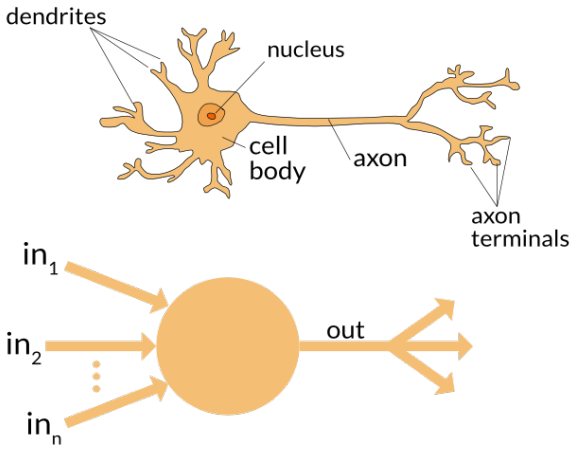

Figura 2.1: Neurone biologico e modello matematico.

Questa attività biologica può essere rappresentata da un modello matematico nel quale i segnali che viaggiano attraverso gli assoni interagiscono con i dendriti dei neuroni con i quali sono collegati. La modifica delle sinapsi (i pesi w), rappresenta l'apprendimento. Nel modello base i dendriti portano il segnale al corpo della cellula, dove sono sommati: se il risultato di questa somma supera una certa soglia, il neurone attiva l'impulso attraverso l'assone. Tale somma viene attivata da una funzione di attivazione. In altre parole, ogni neurone esegue un prodotto vettoriale tra i suoi input e il suo set di pesi, somma un termine di distorsione (bias) e infine applica una funzione di attivazione non lineare. La funzione di attivazione deve necessariamente essere non lineare, in caso contrario la rete neurale si ridurrebbe ad un modello lineare generalizzato.

In figura 2.2 è rappresentata una semplice rete neurale con 4 predittori x_j , un singolo strato nascosto composto da 5 neuroni calcolato: $x^{(1)} = \sigma(Wx^{(0)} + b)$ dove σ è la funzione di attivazione, W è il set di pesi, $x^{(0)}$ è il vettore di input e b è il bias. L'output è data da una singola unità $y = \sigma(Wx^{(1)} + b)$.

L'idea di base è quella che ogni neurone apprenda una semplice funzione binaria. Lo strato finale è caratterizzato anch'esso dalla presenza di un vet-

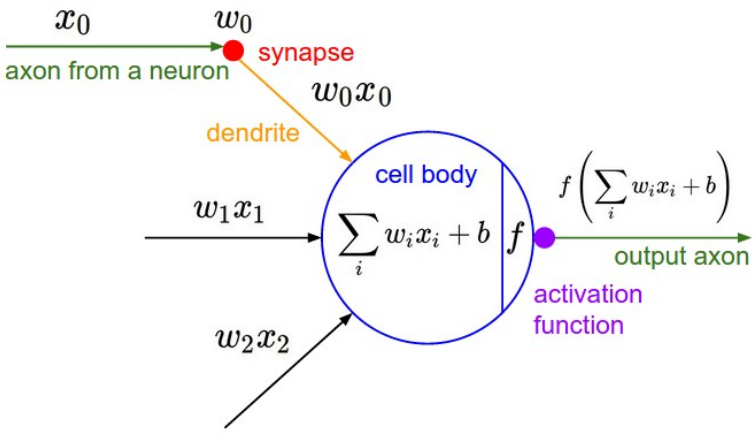

Figura 2.2: Modello di Neurone.

tore di pesi e di una funzione di output, che generalmente corrisponde alla funzione identità in problemi di regressione, o alla softmax in problemi di classificazione binaria. Cambiando il numero degli strati e dei neuroni, le reti neurali possono essere considerate come approssimatori universali di funzioni (Hornik, 1989).

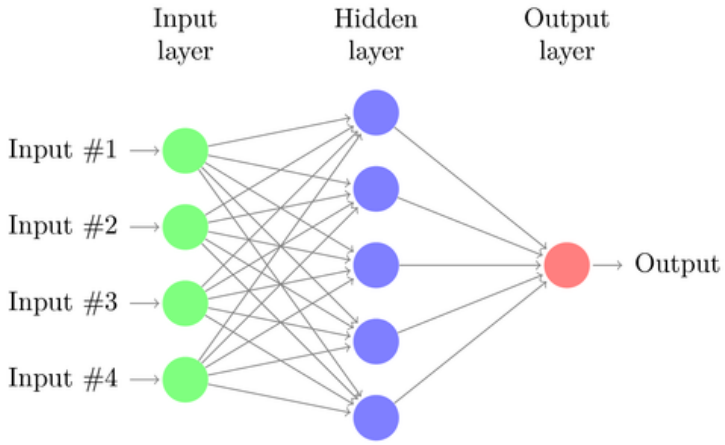

Figura 2.3: Rete con uno strato hidden.

Il termine Deep Learning prende nome dalla profondità che caratterizza le reti neurali, dove con profondità ci si riferisce al numero di strati nascosti presenti. Le reti deep vengono infatti chiamate anche MLP, ovvero MultiLayer Perceptron (Goodfellow, 2016).

2.1.1 Funzione di attivazione

Lo scopo della funzione di attivazione è quello di aggiungere non-linearità nella rete, permettendo di modellare output che non variano linearmente al variare dei predittori. Questo significa che l'output non può essere rappresentato da una combinazione lineare dell'input. Senza non-linearità, anche aggiungendo diversi strati nascosti in una rete, questi risulterebbero equivalenti ad uno strato solo (single-layer Perceptron).

Sigmoid

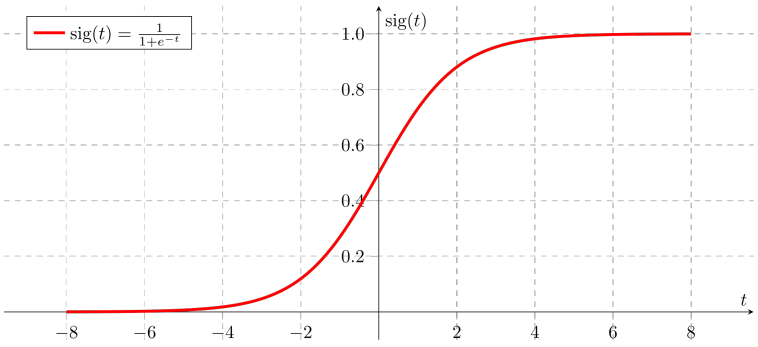

Figura 2.4: Funzione di attivazione Sigmoidea.

Inizialmente la più utilizzata, la funzione sigmoide è stata progressivamente accantonata negli ultimi anni per via di alcune problematiche che comporta a livello pratico.

La prima e più importante è quella relativa alla dissolvenza del gradiente in seguito alla saturazione dei neuroni, ossia quei neuroni che presentano valori di output agli estremi del codominio della funzione di attivazione, in questo caso $(0, 1)$. Tale saturazione diventa problematica durante la fase di Back Propagation, in quanto il gradiente locale assume valori prossimi allo zero, conducendo così all'annullamento del gradiente globale (Glorot - Bengio, 2010). In pratica si ha un flusso utile del gradiente solo per valori di input che rimangono all'interno di una zona di sicurezza, cioè nei dintorni dello zero.

Il secondo problema deriva invece dal fatto che gli output della funzione sigmoide non sono centrati intorno allo zero (LeCun, 98). Si consideri il caso in cui tutti gli input di un nodo sono positivi. L'aggiornamento dei pesi entranti in un determinato neurone avviene in modo proporzionale all'errore in quel nodo (uno scalare) ed il vettore di input. Se tutte le componenti del vettore in input sono positive, tutti gli aggiornamenti dei pesi che arrivano in quel neurone avranno lo stesso segno (il segno dell'errore) e i pesi quindi saranno o tutti incrementati o tutti diminuiti. Se il vettore di pesi in considerazione deve cambiare direzione, questo può avvenire solamente dopo diverse iterazioni di aggiornamento (zigzagando), rallentando così l'apprendimento. Questo si comporta inoltre come una fonte di bias sistematico per i neuroni nello strato successivo della rete. Per aggirare questo problema è stata introdotta come funzione di attivazione la Tangente Iperbolica in quanto funzione dispari. L'apprendimento con Backpropagation con funzioni dispari porta ad una convergenza più veloce rispetto ad un processo con funzioni non simmetriche (Haykin, 2004).

Il terzo e ultimo difetto della funzione sigmoide è che l'operazione esponenziale al denominatore è molto costosa dal punto dal punto di vista computazionale, soprattutto rispetto alle alternative che verranno presentate di seguito.

Tangente Iperbolica

Il problema degli output non centrati sullo zero (Haykin, 2004) della sigmoide può essere risolto ricorrendo all'utilizzo della tangente iperbolica, la quale presenta codominio $(-1, 1)$ centrato sull'origine degli assi. Tuttavia, rimane il problema della saturazione dei neuroni, anzi viene addirittura accentuato, dal momento che la zona di sicurezza risulta ancora più ristretta.

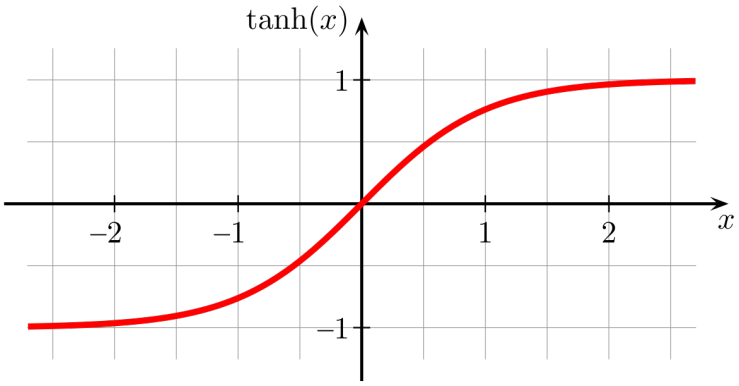

Figura 2.5: Funzione di attivazione Tangente Iperbolica.

Rectified Linear Unit

La Rectified Linear Unit (ReLU) è diventata popolare negli ultimi anni per via dell'incremento prestazionale che offre nel processo di convergenza: velocizza infatti di circa 6 volte la discesa del gradiente rispetto alle alternative viste finora. Questo risultato è da attribuire in larga parte al fatto che la ReLU risolve il problema della dissolvenza del gradiente (Glorot - Bengio, 2010), non andando a saturare i neuroni. Durante la fase di Back Propagation infatti, se il gradiente calcolato fino a quel punto è positivo questo viene semplicemente lasciato passare, perchè la derivata locale per il quale viene moltiplicato è pari ad uno. Eventuali problemi sorgono invece quando il gradiente accumulato ha segno negativo, in quanto questo viene azzerato (le derivata locale è nulla lungo tutto il semiasse negativo) con la conseguenza che i pesi non vengono aggiornati. Fortunatamente questo problema può essere alleviato attraverso l'utilizzo di un algoritmo SGD (batch size maggiori di 1) (Ioffe, 2015): considerando più dati alla volta c'è infatti la speranza che non tutti gli input del batch provochino l'azzeramento del gradiente, tenendo così in vita il processo di apprendimento del neurone. Al contrario, se per ogni osservazione la ReLU riceve valori negativi, allora il neurone "muore", e non c'è speranza che i pesi vengano aggiornati. Valori elevati del learning rate amplificano questo problema, dal momento che cambiamenti più consistenti dei pesi si traducono in una maggiore probabilità che questi affondino

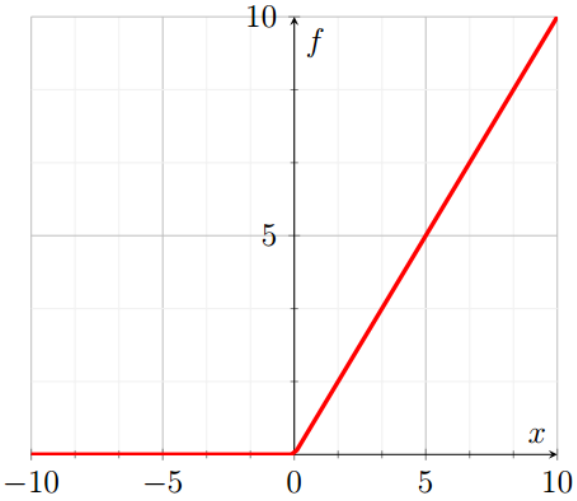

Figura 2.6: Funzione di attivazione Rectified Linear Unit.

nella "zona morta".

2.2 Apprendimento

2.2.1 Backpropagation

Stimare una rete neurale non risulta un compito semplice, trattandosi come abbiamo visto di una complessa funzione gerarchica $f(x; W)$ del vettore di input x e della collezione di pesi W del vettore di input x. Innanzitutto è necessario scegliere opportunamente le funzioni di attivazione in modo che tale funzione risulti differenziabile. Si tratta quindi di risolvere una funzione di perdita $L[y, f(x)]$.

Le funzioni di perdita sono generalmente convesse in f, ma non negli elementi di W, con la conseguenza che ci troviamo a dover risolvere un problema di minimizzazione su una superficie che presenta una grande quantità di minimi locali. Una soluzione è quella di effettuare più stime dello stesso modello con differenti inizializzazioni dei parametri, per scegliere infine quello che risulta essere migliore. Una procedura di questo tipo può però richiedere molto

tempo, che spesso non è disponibile, pertanto generalmente ci si tende ad accontentare di buoni minimi locali.

I principali metodi utilizzati sono basati sulla tecnica della discesa del gradiente, la cui implementazione in questo contesto prende il nome di Back-Propagation (Werbos, 1974), in virtù del fatto che lo scarto registrato in corrispondenza di un certo dato viene fatto propagare all'indietro nella rete per ottenere le formule di aggiornamento dei coefficienti. Dal momento che $f(x; W)$ è definita come una composizione di funzioni a partire dai valori di input della rete, gli elementi di W sono disponibili solo in successione (quella degli strati) e pertanto la differenziazione del gradiente seguirà la regola della catena (chain rule).

Data una generica osservazione (x, y) l'algoritmo di back-propagation prevede di effettuare un primo passo in avanti lungo l'intera rete (feed-forward step) e salvare le attivazioni che si creano ad ogni nodo $x_l^{(k)}$ di ogni strato l, incluso quello di output. La responsabilità di ogni nodo nella previsione del vero valore di y viene quindi misurata attraverso il calcolo di un termine d'errore $\delta_l^{(k)}$. Nel caso delle attivazioni terminali $x_l^{(k)}$ il calcolo di tali errori è semplice: coincide infatti con i residui o loro trasformazioni, in base a come viene definita la funzione di perdita. Per le attivazioni degli strati intermedi $\delta_l^{(k)}$ viene calcolato invece come somma pesata dei termini d'errore dei nodi che utilizzano $x_l^{(k)}$ come input.

Una volta calcolato il gradiente è possibile procedere con l'aggiornamento del sistema dei pesi. Il calcolo del gradiente ci fornisce la direzione lungo la quale la superficie da minimizzare è più ripida, ma nessuna informazione circa la lunghezza del passo che dovremmo compiere. Tale quantità viene denominata learning-rate e costituisce l'iperparametro più importante da regolare in una rete neurale: valori alti portano ad una convergenza più veloce ma sono rischiosi dal momento che potrebbero saltare il minimo ottimale o fluttuare intorno ad esso, mentre valori bassi sono responsabili di una convergenza lenta e possono comportare il blocco dell'algoritmo in un minimo locale non ottimale.

2.2.2 Stochastic Gradient Descent

Esistono alcune varianti dell'algoritmo di discesa del gradiente che differiscono nella quantità di dati utilizzati nel calcolo del gradiente prima di effettuare l'aggiornamento dei parametri. Quella classica utilizza ad ogni iterazione l'intero dataset. Tuttavia, può spesso risultare più efficiente processare piccole quantità di dati (batch) per volta, generalmente campionate in maniera casuale, da qui il nome Stochastic Gradient Descent (Bottou, 2010). Tale scelta è obbligata quando le dimensioni del dataset sono tali da non poter essere caricato in memoria. Valori estremi del batch come n oppure 1 possono causare problemi rispettivamente di calcoli ridondanti (il gradiente viene ricalcolato sempre su osservazioni simili prima dell'aggiornamento) e di fluttuazioni della funzione da minimizzare, a causa di aggiornamenti troppo frequenti e variabili, essendo basati sulla singola osservazione. Di conseguenza si tendono a scegliere valori intermedi. Indipendentemente dal numero di iterazioni e dalla dimensione del batch prescelta, ogni volta che tutte le osservazioni del dataset vengono utilizzate per il calcolo del gradiente si dice che viene completata un'epoca.

2.2.3 Ottimizzazione discesa del Gradiente

Ci sono alcune modifiche che possono essere apportate all'algoritmo di discesa del gradiente per migliorarne le performance, legate soprattutto al learning rate. Possono sorgere complicazioni invece quando si tratta di minimizzare funzione non convesse, con il rischio di rimanere intrappolati in minimi locali non ottimali. Di seguito verranno presentati alcuni algoritmi di ottimizzazione che mirano alla risoluzione di questo tipo di problematiche.

- **Momentum**: SGD presenta alcune difficoltà in prossimità di aree dove la superficie presenta una curvatura molto più accentuata in una direzione rispetto all'altra (Sutton, 1986). In questo scenario infatti l'algoritmo tende ad oscillare lungo il versante più ripido, rallentando così la convergenza verso il minimo locale ottimale. Momentum (Qian,

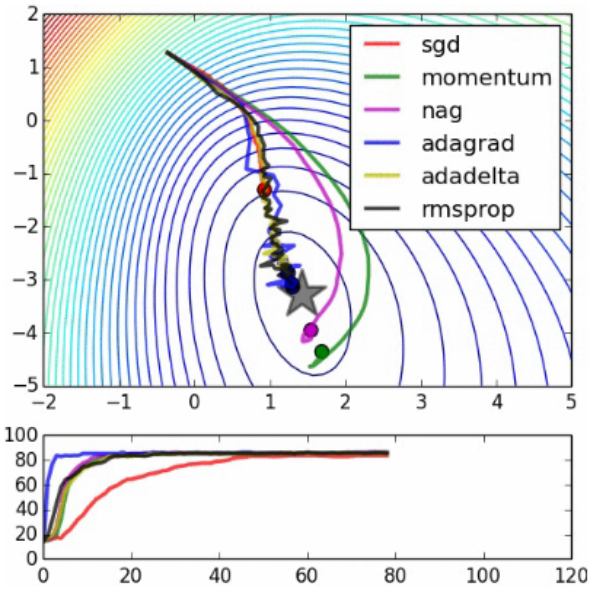

Figura 2.7: Confronto tra alcuni metodi di ottimizzazione - fonte.

1999) è un metodo che aiuta ad accelerare la discesa del gradiente verso la direzione corretta, smorzando l'effetto indotto dalle oscillazioni. Tale risultato si ottiene aggiungendo al termine di aggiornamento corrente una frazione γ del vettore di aggiornamento precedente.

- **AdaGrad**: Momentum permette di adattare i nostri aggiornamenti in relazione alla superficie da minimizzare velocizzando così la convergenza, ma non fa nessuna distinzione circa l'importanza dei parametri, trattandoli tutti allo stesso modo. Adagrad (Duchi, 2011) è un algoritmo di ottimizzazione nato proprio con questo scopo: adattare il learning rate ai parametri, permettendo così di effettuare aggiornamenti più consistenti in corrispondenza dei parametri relativi alle features meno frequenti, e viceversa. In particolare, nella sua regola di aggiornamento, Adagrad utilizza un α differente per ogni parametro ad ogni iterazione, modificando quest'ultimo sulla base dei gradienti passati che sono stati calcolati per lo specifico parametro. Uno dei più grandi benefici di Adagrad consiste nell'eliminare la necessità di settare manualmente il learning rate: è necessario impostare solamente il

valore iniziale. D'altra parte, il suo punto debole è invece l'accumulo eccessivo del quadrato dei gradienti al denominatore, che comporta un'esagerata e progressiva riduzione del learning rate, il quale tende a diventare infinitamente piccolo, al punto tale che l'algoritmo non è più in grado di acquisire nuova informazione, arrestando così il processo di convergenza.

2.3 Convolutional Neural Networks

Le Convolutional Neural Networks sono ispirate dall'organizzazione della corteccia visiva animale (Hubel, 1968) (Fukushima, 1980) ed il nome deriva dall'utilizzo di un'operazione matematica lineare chiamata convoluzione. Di conseguenza, una rete neurale classica che implementa operazioni di convoluzione in almeno uno dei suoi strati, viene definita Convolutional. Gli strati composti da operazioni di convoluzione prendono il nome di Convolutional Layers, ma non sono gli unici strati che compongono una CNN: la tipica architettura prevede infatti l'alternarsi di Convolutional Layers, Pooling Layers e Fully Connected Layers (Krizhevsky, 2012).

Come visto nella sezione precedente, le Reti Neurali tradizionali ricevono in input un singolo vettore, e lo trasformano attraverso una serie di strati nascosti, dove ogni neurone è connesso ad ogni singolo neurone sia dello strato precedente che di quello successivo (ovvero "Fully-Connected") e funziona quindi in maniera completamente indipendente, dal momento che non vi è alcuna condivisione delle connessioni con i nodi circostanti. Nel caso l'input sia costituito da immagini di dimensioni ridotte, ad esempio $32 \times 32 \times 3$ (32 altezza, 32 larghezza, 3 canali colore), un singolo neurone connesso in questa maniera comporterebbe un numero totale di $32 \times 32 \times 3 = 3072$ pesi, una quantità abbastanza grande ma ancora trattabile. Le cose si complicano però quando le dimensioni si fanno importanti: salire ad appena 256 pixel per lato comporterebbe un carico di $256 \times 256 \times 3 = 196608$ pesi per singolo

neurone, ovvero quasi 2 milioni di parametri per una semplice rete con un singolo strato nascosto da dieci neuroni.

L'architettura Fully-Connected risulta perciò troppo esosa in questo contesto, comportando una quantità enorme di parametri che condurrebbe velocemente a casi di sovradattamento. Inoltre, considerazione ancor più limitante, un'architettura di questo tipo fatica a cogliere la struttura di correlazione tipica dei dati a griglia. Le Convolutional Neural Networks prendono invece vantaggio dall'assunzione che gli input hanno proprio una struttura di questo tipo (LeCun, 1998), e questo permette loro la costruzione di un'architettura su misura attraverso la formalizzazione di tre fondamentali proprietà: l'interazione sparsa (*sparse interaction*), l'invarianza rispetto a traslazioni (*translation invariance*), e la condivisione dei parametri (*weight sharing*) (Zhou, 2015). Il risultato è una rete più efficace e allo stesso tempo parsimoniosa in termini di parametri. L'assunzione di weight sharing è ragionevole dal momento che individuare un determinato pattern è importante in qualsiasi posizione dell'input, e di conseguenza non c'è la necessità di imparare a localizzare la stessa caratteristica in tutte le possibili porzioni dell'input.

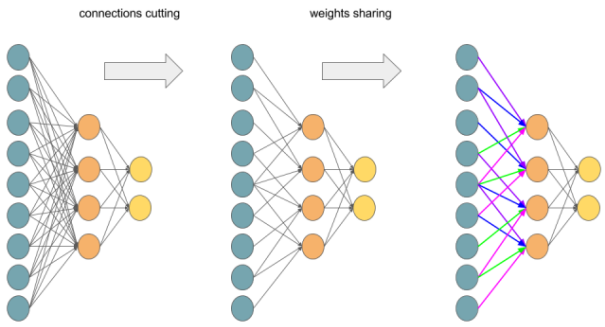

Figura 2.8: Weight sharing.

2.3.1 Operazione di Convoluzione

In problemi discreti l'operazione di convoluzione non è altro che la somma degli elementi del prodotto di Hadamard fra un set di parametri (che prende

il nome di filtro) e una porzione dell'input di pari dimensioni. L'operazione di convoluzione viene quindi ripetuta spostando il filtro lungo tutta la superficie dell'input, sia in altezza che in larghezza. Questo produce quella che viene chiamata mappa di attivazione, la quale costituisce di fatto il primo strato nascosto della rete. In altre parole, ragionando dalla prospettiva opposta, la mappa di attivazione è formata da neuroni connessi localmente allo strato di input attraverso i parametri del filtro che li ha generati. L'estensione spaziale di questa connettività, che coincide con la dimensione del filtro, costituisce un iperparametro della rete e viene chiamata anche campo recettivo del neurone, o receptive field. Questa rappresenta la prima delle tre proprietà fondamentali delle Convolutional Neural Networks, ossia la sparse interaction.

É importante notare che la singola operazione di convoluzione produce sempre uno scalare, indipendentemente da quali siano le dimensioni del volume di input, sia esso bidimensionale o tridimensionale. Di conseguenza, una volta che il filtro viene fatto convolvere lungo tutta la superficie dell'input, si ottiene sempre una mappa di attivazioni bidimensionale.

Finora abbiamo sempre visto la convoluzione di un singolo filtro ma generalmente un Convolutional Layer è formato da un set di filtri molto più numeroso, diciamo n, tutti con la medesima estensione spaziale. Durante la fase feed-forward ogni filtro viene fatto convolvere lungo la larghezza e l'altezza del volume di input, e pertanto vengono prodotte n mappe di attivazioni bidimensionali, le quali forniscono ognuna la risposta del relativo filtro in ogni posizione spaziale. La loro concatenazione lungo la terza dimensione produce l'output del Convolutional Layer. É facile notare che gli indici dei pesi che definiscono il singolo neurone non dipendono da (i, j), ovvero dalla sua posizione nella mappa di attivazione, ma solamente dal filtro generatore f. In altre parole, questo significa che i neuroni appartenenti alla stessa mappa di attivazione condividono sempre lo stesso set di pesi, ossia quelli del filtro che li ha generati. Questo definisce la seconda proprietà fondamentale delle Convolutional Neural Networks, ossia il weight sharing.

Disposizione finale e numero di neuroni che compongono il volume di ouput

del Convolutional layer sono controllati da tre iperparametri: profondità, stride, e padding.

- **Profondità**: corrisponde al numero di filtri n che compongono lo strato, ognuno in cerca di caratteristiche differenti nel volume di input.
- **Stride**: specifica il numero di pixel di cui si vuole traslare il filtro ad ogni spostamento. Valori più alti muovono il filtro con salti maggiori, e pertanto viene generato un output di dimensioni minori.
- **Padding**: a volte può risultare conveniente aggiungere un bordo di zeri al volume di input, in modo così da controllare le dimensioni dell'output ed evitare incongruenze durante le operazioni. Spesso utilizzato per far combaciare la dimensione dell'input con quella dell'output.

2.3.2 Pooling

Nell'architettura di una CNN è pratica comune inserire fra due o più convolutional layers uno strato di Pooling (Ciresan, 2011), la cui funzione è quella di ridurre progressivamente la dimensione spaziale degli input (larghezza e altezza), in modo da diminuire numero di parametri e carico computazionale, e di conseguenza controllare anche il sovradattamento. Il Pooling Layer opera indipendentemente su ogni mappa di attivazioni applicando un filtro di dimensione $F * F$ che esegue una determinata operazione deterministica (tipicamente il massimo o la media), e pertanto non comporta la presenza di pesi. Anche qui, il calcolo del volume finale dipende, oltre che dalle dimensioni di input, dai due iperparametri richiesti, ossia stride ed estensione spaziale del filtro.

Capitolo 3

Knowledge Discovery

Il Knowledge Discovery è un processo non banale di identificazione di pattern nei dati che siano validi, nuovi, potenzialmente utili e comprensibili. Per dati si intende un insieme di fatti, per pattern si intende un'espressione in un qualche linguaggio che descrive un sottoinsieme dei dati o un modello degli stessi. Estrarre un pattern quindi vuol dire trovare un modello dei dati o una descrizione di questi ad alto livello. Il termine processo è utilizzato perché il Knowledge Discovery richiede una serie di passi per essere attuato. I pattern estratti dovranno essere nuovi, validi con un certo grado di certezza, potenzialmente utili per i compiti per cui saranno usati e comprensibili per l'utente.

3.1 Task

Il lavoro svolto si propone l'obiettivo di utilizzare le Reti Neurali per produrre regole di classificazione, con le quali diagnosticare la severità del Disturbo Respiratorio del Sonno attraverso rilevazioni effettuate su pazienti. Con questo proposito è stato applicato il processo di Knowledge Discovery ai dati forniti dal Dipartimento di Scienze Mediche e Chirurgiche dell'Università de-

gli Studi di Foggia e su queste sono state effettuate diverse sperimentazioni utilizzando modelli di Reti Neurali Fully-Connected e Convolutional.

La funzione respiratoria è influenzata da molteplici sistemi di controllo, diretti a regolare i parametri respiratori in funzione delle diverse esigenze della vita vegetativa e di relazione. Particolare interesse è rappresentato dalla modalità di adattamento correlata al ritmo sonno-veglia. Durante la veglia la normale funzione respiratoria è assicurata dal controllo e dalla possibilità di intervento del sistema nervoso centrale e la pervietà delle vie aeree superiori è normalmente assicurata da un adeguato tono delle relative strutture muscolari. Per contro il sonno è una condizione fisiologica relativamente sfavorevole per la respirazione.

Quando le modificazioni della funzione proprie del sonno si sovrappongono ad una patologia respiratoria o ad una anomalia anatomica delle prime vie aeree, possono prodursi disturbi respiratori del sonno di diversa natura e gravità. Ai disturbi respiratori del sonno (**DRS**) viene oggi fornita sempre maggiore attenzione nell'ambito delle malattie dell'apparato respiratorio.

Le conseguenze dei disturbi respiratori del sonno vanno dai disturbi dell'umore al pericolo di vita, dovute al fatto che l'ossigenazione del sangue viene ad essere temporaneamente compromessa spesso in modo serio, e poiché il fenomeno può ripetersi decine di volte ogni notte, questo si traduce in uno stato di "stress" reiterato per gli organi più sensibili allo stato di ossigenazione ematica: cuore e cervello. Così per una malattia che origina dal sonno, presto le conseguenze diventano visibili e si palesano durante la veglia. E' infatti comune che la sindrome delle apnee ostruttive nel sonno, uno dei più comuni DRS, si accompagni ad ipertensione e aritmie.

Inoltre i pazienti con questa sindrome hanno un rischio maggiore rispetto alla popolazione normale di avere episodi di infarto cardiaco o di ictus cerebrale. A causa della eccessiva sonnolenza diurna, che può arrivare a compromettere le normali occupazioni di vita sociale e lavorativa, i pazienti con il disturbo hanno inoltre un elevatissimo rischio di incorrere in incidenti automobilistici, infatti quasi il 20% di tutte le lesioni gravi da incidenti

d'auto nella popolazione generale sono associati proprio alla sonnolenza di chi guida.

3.2 Data Understanding

Il dataset in esame è composto da 313 campioni, ciascuno dei quali è composto da 21 feature, di individui affetti da disturbi respiratori del sonno con la relativa severità del disturbo diagnosticato. Molti attributi presentano una quantità notevole di valori mancanti. I dati ricevuti sono privi di qualunque informazione che permetta l'individuazione dei pazienti coinvolti nella raccolta dei dati.

Feature	**Missing values**
AHI	0
aa	0
sex	0
BMI	17
ESS	66
SMOKE	0
Mallampati	84
Tipo di Russamento	68
Frequenza della stanchezza	60
Frequenza dei risvegli	64
Frequenza Apnee	65
Frequenza Addormentamento alla guida	70
IPERTENSIONE.	0
CARDIOPATIA	0
DIABETE	0
paO2	12
paCO2	12
FVC%	35
FEV1%	35
Tiff	35

In seguito viene riportata la distribuzione dei vari attributi.

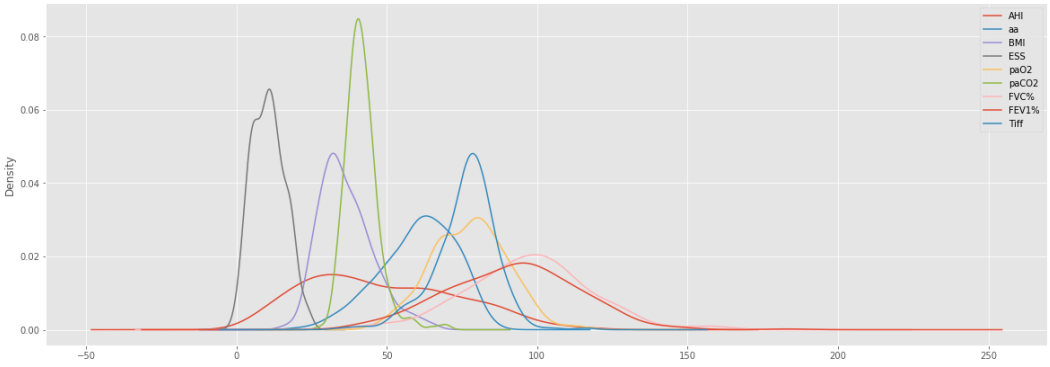

Figura 3.1: Distribuzione.

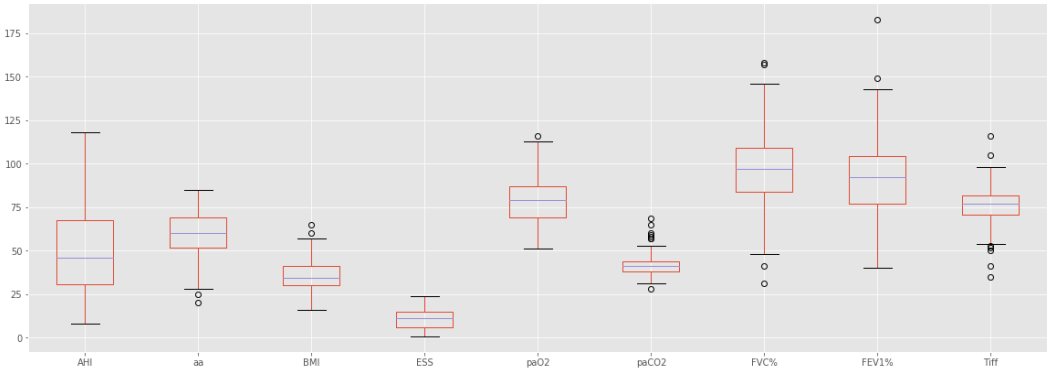

Figura 3.2: Boxplot.

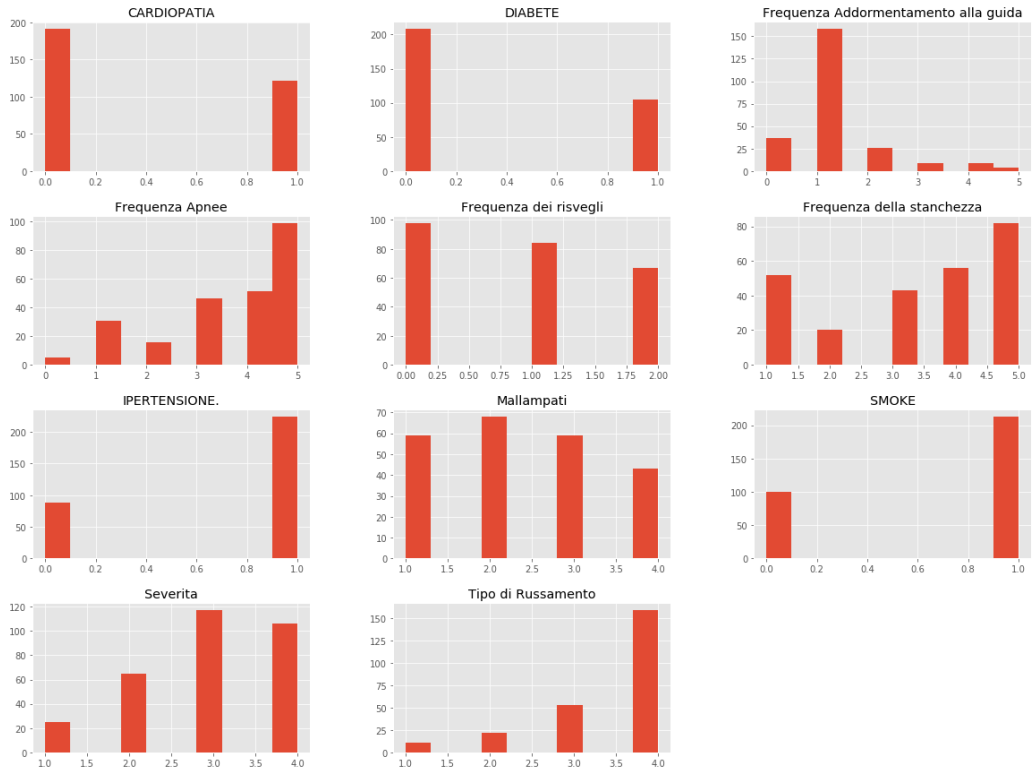

Figura 3.3: Distribuzione delle feature.

3.3 Preprocessing

In questa fase del processo di Knowledge Discovery è necessario stimare la bontà dei dati a disposizione, in modo da fornire dati in formato idoneo per la fase successiva di training, tenendo sempre a mente gli obiettivi prefissati all'inizio del processo.

3.3.1 Missing Values

Per la presenza di un alto numero di dati mancanti e la bassa cardinalità di campioni nel dataset, sono state utilizzate tre diverse tecniche per la loro gestione, descritte in seguito.

Rimozione valori mancanti

Nel primo caso sono state semplicemente rimosse tutte le osservazioni contenenti valori mancanti, portando il numero di osservazioni da 313 a 168.

Sostituzione valori mancanti

Nel secondo caso sono stati sostituiti i valori mancanti negli attributi **BMI**, **ESS**, **paO2**, **paCO2**, **FVC%**, **FEV1%**, **Tiff**, con la media, mentre quelli negli attributi **Mallampati**, **Tipo di Russamento**, **Frequenza della stanchezza**, **Frequenza dei risvegli**, **Frequenza Apnee**, **Frequenza Addormentamento alla guida** sono stati sostituiti con la moda.

Predizione valori mancanti

Nell'ultimo caso i valori mancanti sono stati stimati, usando come predittori solo le feature che non hanno presentato valori mancanti, ovvero **Severita**, **AHI**, **aa**, **sex**, **SMOKE**, **IPERTENSIONE.**, **CARDIOPATIA**, **DIABETE**.

Anche in questo caso è stata fatta distinzione tra le feature discrete e quelle continue. Per le prime, i valori mancanti sono stati stimati attraverso un classificatore *Random Forest* con 10 alberi aventi profondità massima pari a 5. Per la stima dei valori continui, invece, sono stati utilizzati metodi di regressione *Support Vector Regression* usando un kernel **RBF** con **gamma** pari a 0.1 e **c** pari a $1e3$.

3.3.2 Normalizzazione

La normalizzazione è una fase necessaria in quanto le varie feature presentano differenti scale di grandezza. Inoltre portando i vettori ad avere norma unitaria, algoritmi basati sulla discesa del gradiente arrivano prima a convergenza, in quanto i passi da fare ad ogni iterazione verso il minimo saranno

influenzati dalla scala di grandezza delle varie feature. L'ottimo locale potrebbe trovarsi a poca distanza lungo un asse ma lontano in un altro asse, questo porterebbe ad effettuare passi troppo piccoli, rapportati alla distanza sull'asse più vicino al minimo, ed a convergere più lentamente.

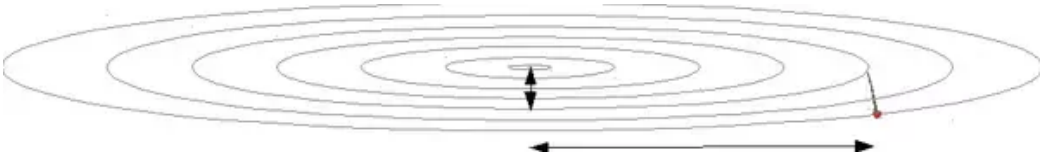

Figura 3.4: Diverse scale di grandezza.

3.3.3 Sampling

Il dataset è stato suddiviso in strati, uno per ogni valore dell'attributo target Severita, e da questi sono state prelevate osservazioni in numero tale da rendere uguale il numero di osservazioni in ogni strato. Questo ha comunque portato ad una riduzione significativa del dataset a 100 osservazioni, in quanto la classe meno ricorrente presenta 25 istanze.

3.4 Training

Le reti neurali sono state addestrate utilizzando Python 3.5.4 e Tensorflow 1.3.0 come libreria, tutte le dipendenze sono specificate nel file requirements.txt. Le configurazioni degli iperparametri con i quali sono stati ricavati i risultati saranno mostrati nella sezione succcessiva di Evaluation. Di seguito saranno riportate le scelte effettuate per l'architettura delle reti fully-connected e convolutional, per la funzione costo e per l'algoritmo di ottimizzazione, comuni in tutte le configurazioni.

3.4.1 Cross Entropy

Come funzione costo è stato scelta la funzione Cross Entropy. Questa viene maggiormente utilizzata in problemi di classificazione rispetto alla *MSE*. L'obiettivo è quello di minimizzare il numero di esempi classificati incorrettamente attraverso l'aumento esponenziale dell errore all'allontanamento dell'output al valore desiderato, comportandosi meglio rispetto alla *MSE* per funzioni non lineari.

3.4.2 Ottimizzatore AdaGrad

AdaGrad (Adaptive Gradient algorithm) è una versione modificata dello Stochastic Gradient Descent che presenta un learning rate diverso per ogni parametro. Intuitivamente il learning rate viene aumentato per i parametri che risultano rari e diminuito per quelli meno rari. Questa strategia spesso migliora le performance, arrivando prima a convergenza quando i parametri più sparsi sono quelli più informativi. Rimane necessario inizializzare il valore del learning rate, ma questo poi viene adattato automaticamente ai singoli parametri.

3.4.3 Layer di Convolution

Data la natura monodimensionale degli esempi di input ed alla non rilevanza dell'ordine delle feature, è stato utilizzato un solo strato di convoluzione. Inoltre sono stati utilizzati solamente kernel di lunghezza 19 (lunghezza di tutto il vettore di input) o di lunghezza 1. Questa decisione è stata presa perchè i gli attributi, come detto, non presentano una dipendenza spaziale gli uni dagli altri.

3.5 Evaluation

I valori degli iperparametri sono riportati nelle tabelle, dove per ogni configurazione è stato ricavato il valore di cross validation.

3.5.1 Cross Validation

La k-fold cross-validation consiste nella suddivisione del dataset totale in k parti di uguale numerosità e, ad ogni passo, la k-esima parte del dataset viene utilizzata come test set, mentre la restante parte costituisce il training set. Così, per ognuna delle k parti si allena il modello, evitando quindi problemi di overfitting, ma anche di campionamento asimmetrico del training dataset, tipico della suddivisione del dataset in due sole parti. In altre parole, si suddivide il campione osservato in gruppi di egual numerosità, si esclude iterativamente un gruppo alla volta e lo si cerca di predire con i gruppi non esclusi. Ciò al fine di verificare la bontà del modello di predizione utilizzato. In questo caso la divisione del dataset in 10 parti è avvenuta in modo stratificato, ovvero mantenendo costanti le proporzioni dei valori della classe target in tutti i fold.

CV-SCORE

- Il modello è addestrato utilizzando come training set **k-1** fold del dataset originale
- Le performance vengono misurate utilizzando l'unico fold rimanente per calcolare l'accuratezza del modello
- Si ripete il tutto **k** volte, utilizzando come test set ogni volta un fold diverso
- Ottenuti **k** valori di accuratezza, il CV-SCORE è ottenuto facendo la media di questi

Le metriche riportate in seguito sono relative al dataset con i valori mancanti sostituiti con la media o con la moda, il quale ha restituito valori complessivamente migliori, e con le feature normalizzate. La normalizzazione tuttavia non ha inciso significativamente sulle performance, fattore dovuto principalmente all'utilizzo di AdaGrad come algoritmo di discesa del gradiente. Il campionamento stratificato non è stato utilizzato in quanto diminuiva di molto le osservazioni già in numero ridotto, suggerendo un approccio al problema del bilanciamento dei dati diverso (SMOTE).

Metriche Rete Dense

input	hidden	output	learning rate	batch size	CV-SCORE
19	-	4	0.002	10	0.358233
19	-	4	0.002	100	0.388014
19	-	4	0.002	250	0.392992
19	-	4	0.003	10	0.359602
19	-	4	0.003	100	0.377474
19	-	4	0.003	250	0.389962
19	10	4	0.002	10	0.403434
19	10	4	0.002	100	0.391685
19	10	4	0.002	250	0.389383
19	10	4	0.003	10	0.360670
19	10	4	0.003	100	0.411738
19	10	4	0.003	250	0.407496
19	7	4	0.002	10	0.373822
19	7	4	0.002	100	0.391685
19	7	4	0.002	250	0.375758
19	7	4	0.003	10	0.377889
19	7	4	0.003	100	0.378532
19	7	4	0.003	250	0.399153
19	4	4	0.002	10	0.393285
19	4	4	0.002	100	0.381344
19	4	4	0.002	250	0.417503
19	4	4	0.003	10	0.414121
19	4	4	0.003	100	0.339783
19	4	4	0.003	250	0.387305

Metriche Rete Convolutional

in	hid	out	kernel size	n kernels	l rate	batch	CV-score
19	-	4	19	1	0.002	10	0.392510
19	-	4	19	1	0.002	100	0.398892
19	-	4	19	1	0.002	250	0.359583
19	-	4	19	1	0.003	10	0.425758
19	-	4	19	1	0.003	100	0.418889
19	-	4	19	1	0.003	250	0.393757
19	-	4	19	3	0.002	10	0.382701
19	-	4	19	3	0.002	100	0.373830
19	-	4	19	3	0.002	250	0.377777
19	-	4	19	3	0.003	10	0.386726
19	-	4	19	3	0.003	100	0.385651
19	-	4	19	3	0.003	250	0.379414
19	-	4	1	1	0.002	10	0.390052
19	-	4	1	1	0.002	100	0.419314
19	-	4	1	1	0.002	250	0.384627
19	-	4	1	1	0.003	10	0.353845
19	-	4	1	1	0.003	100	0.390787
19	-	4	1	1	0.003	250	0.371332
19	-	4	1	3	0.002	10	0.400795
19	-	4	1	3	0.002	100	0.404001
19	-	4	1	3	0.002	250	0.387097
19	-	4	1	3	0.003	10	0.376941
19	-	4	1	3	0.003	100	0.393925
19	-	4	1	3	0.003	250	0.344179
19	10	4	19	1	0.002	10	0.335917
19	10	4	19	1	0.002	100	0.390531
19	10	4	19	1	0.002	250	0.365299
19	10	4	19	1	0.003	10	0.326312
19	10	4	19	1	0.003	100	0.403116
19	10	4	19	1	0.003	250	0.335135

in	hid	out	kernel size	n kernels	l rate	batch	CV-score
19	10	4	19	3	0.002	10	0.363266
19	10	4	19	3	0.002	100	0.376012
19	10	4	19	3	0.002	250	0.372095
19	10	4	19	3	0.003	10	0.376223
19	10	4	19	3	0.003	100	0.345624
19	10	4	19	3	0.003	250	0.355506
19	10	4	1	1	0.002	10	0.348512
19	10	4	1	1	0.002	100	0.364155
19	10	4	1	1	0.002	250	0.389568
19	10	4	1	1	0.003	10	0.357784
19	10	4	1	1	0.003	100	0.354275
19	10	4	1	1	0.003	250	0.364439
19	10	4	1	3	0.002	10	0.357547
19	10	4	1	3	0.002	100	0.339492
19	10	4	1	3	0.002	250	0.364294
19	10	4	1	3	0.003	10	0.346856
19	10	4	1	3	0.003	100	0.350228
19	10	4	1	3	0.003	250	0.338084

Capitolo 4

Conclusione

Il lavoro svolto si era riproposto l'obiettivo di utilizzare le Reti Neurali per produrre regole di classificazione, con le quali si potesse diagnosticare la severità del Disturbo Respiratorio del Sonno di cui soffre un paziente. Con questo proposito è stato applicato il processo di Knowledge Discovery ai dati forniti dal Dipartimento di Scienze Mediche e Chirurgiche dell'Università degli Studi di Foggia e su questi sono state effettuate diverse sperimentazioni utilizzando modelli di Reti Neurali Fully-Connected e Convolutional.

Dai risultati non è emerso nessun aumento significativo nelle performance di classificazione rispetto ai metodi tradizionali utilizzati in altri lavori. I risultati hanno comunque evidenziato che le Convolutional Neural Network possono rivelarsi utili anche in presenza di dati strutturati, così come nel loro più frequente utilizzo nella elaborazione del linguaggio naturale o delle immagini.

Bibliografia

Bottou Leon - *Large-Scale Machine Learning with Stochastic Gradient Descent* 2010

Ciresan Dan, Meier Ueli, Masci Jonathan, Gambardella Luca, Schmidhuber Jurgen - *Flexible, High Performance Convolutional Neural Networks for Image Classification* 2011

Duchi John, Hazan Elad, Singer Yoram - *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization* 2011

Fukushima Kunihiko - *Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position* 1980

Glorot Xavier, Bengio Yoshua - *Understanding the difficulty of training deep feedforward neural networks* 2010

Goodfellow Ian, Bengio Yoshua, Courville Aaron - *Deep Learning* 2016

Haykin Simon - *Neural Networks and Learning Machines* 2004

Hopfield John - *Neural networks and physical systems with emergent collective computational abilities* 1982

Hornik Kurt, Stinchcombe Maxwell, White Halbert - *Multilayer feedforward networks are universal approximators* 1989

Hubel David, Wiesel Torsten - *Receptive fields and functional architecture of monkey striate cortex* 1968

Ioffe Sergey, Szegedy Christian - *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift* 2015

Krizhevsky Alex, Sutskever Ilya, Hinton Geoffrey - *ImageNet Classification with Deep Convolutional Neural Networks* 2012

LeCun Yann, Bottou Leo, Orr Genevieve, Muller Klaus-Robert - *Efficient BackProp* 1998

Minsky L. Marvin, Papert A. Seymour - *An introduction to computational geometry* 1969

Ning Qian - *On the momentum term in gradient descent learning algorithms* 1999

Werbos Paul - *Beyond regression: New tools for prediction and analysis in the behavioral sciences* 1974

Zhou Bolei, Khosla Aditya, Lapedriza Agata, Oliva Aude, Torralba Antonio - *Object detectors emerge in deep scene CNNs* 2015