Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienza Informatiche

STUDIO E SPERIMENTAZIONE DI CHARACTER-LEVEL CONVOLUTIONAL NEURAL NETWORKS PER LA SENTIMENT CLASSIFICATION

Relatore:
Chiar.mo Prof.
Gianluca Moro

Presentata da:
Luca Agatensi

Correlatori:
Ing. ***Andrea Pagliarani***
Ing. ***Roberto Pasolini***

Sessione III
Anno Accademico 2016-2017

A Maurizio, Sabrina, Martina e Federica

Introduzione

L'obiettivo di questa tesi progettuale è quello di utilizzare una particolare tipologia di reti neurali, definite Char-CNN ovvero Character-Level Convolutional Neural Networks, per classificare e testare una serie di recensioni di prodotti relativi a differenti contesti commerciali e con differenti linguaggi, in modo tale che il sistema sia poi in grado di riconoscere la classe delle recensioni future.
Gli esperimenti sono stati sviluppati sia In-Domain che Cross-Domain, cioè significa che il sistema addestrato è stato valutato sia con recensioni appartenenti allo stesso dominio utilizzato per l'addestramento, sia invece con recensioni appartenenti ad un altro dominio.
L'utilità di esperimenti di tipo Cross-Domain è relativa al fatto che preclassificare su un nuovo dominio sia un compito particolarmente oneroso che deve essere effettuata da un uomo.
La tesi per cui è incentrata su analizzare l'efficacia di questa tecnica di Deep Learning e confrontare poi i risultati ottenuti con altre tecniche.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Deep e Transfer Learning

1.1 Introduzione

Transfer e Deep Learning sono due aspetti fondamentali del Machine Learning e più in generale dell'Artificial Intelligence. Definire il Deep Learning come disciplina è stato per molti una sfida in quanto ha cambiato forma lentamente negli ultimi dieci anni. Una definizione utile, come specificano Adam Gibson e Josh Patterson[17], è che il Deep Learning si occupi di "reti neurali con più di due strati". L'aspetto problematico di questa definizione è che fa sembrare il Deep Learning allo stesso modo in cui si presentava negli anni '80.

Si ritiene che le reti neurali moderne debbano trascendere architetturalmente dallo stile delle prime reti neurali. In generale il Deep Learning si può definire come parte di una più grande famiglia costituita dal Machine Learning e più in alto dell'Artificial Intelligence[18].

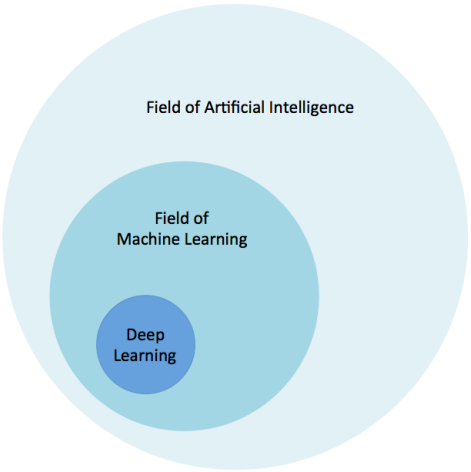

Figura 1.1: Relazione Deep Learning, Machine Learning e Artificial Intelligence

1.2 Artificial Intelligence e Machine Learning

Parlare di Big Data è ormai diventata una normale consuetudine e i termini Artificial Intelligence e Machine Learning sono molto frequenti ed è facile percepirli come la medesima cosa.

1.2.1 Definizioni

Come scritto da Bernard Marr in un articolo pubblicato su Forbes nel Dicembre 2016[19], l'Artificial Intelligence può essere definita come un concetto più ampio di macchine che sono in grado di svolgere compiti(tasks) in un modo che il nostro mondo possa riconoscerle come "intelligenti"; il Machine Learning invece può definirsi come applicazione corrente di AI basata sull'idea che bisogna fornire alle macchine i dati e che da quelli poi vengano addestrate in modo tale che riconoscano determinati elementi senza che siano esplicitamente programmate.

1.2.2 Inizi

L'Artificial Intelligence come idea di base è presente da molto tempo, infatti i primi computer europei erano stati concepiti come "macchine logiche" che riproducessero le basi aritmetiche e mnemoniche presenti nel cervello umano.
Col progredire della tecnologia e della conoscenza della nostra mente è cambiato il concetto di fondo dell'AI: piuttosto che calcoli sempre più complessi, il lavoro nel campo dell'AI si è concentrato sulla imitazione dei processi decisionali umani e svolgendo compiti in modi sempre più umani.
I sistemi disegnati per l'AI sono spesso classificati in due gruppi:

- Applicati - Sistemi molto più comuni, come ad esempio sistemi di trading o di movimento autonomo dei veicoli.
- Generali - Sistemi che possono occuparsi di ogni tipo di task (compito), meno comuni, ma con molto più potenziale; è l'area in cui si è sviluppato il Machine Learning.

1.2.3 L'ascesa del Machine Learning

Due importanti scoperte hanno portato alla nascita del Machine Learning come veicolo che sta portando avanti l'AI con la velocità che ha attualmente.

- Una di queste fu la realizzazione, accreditata da Arthur Samuel nel 1959, che, piuttosto che insegnare ai computer tutto ciò di cui hanno bisogno per conoscere il mondo e come svolgere i tasks, potrebbe essere possibile insegnarli a imparare da soli.
- La seconda, più recente, fu l'emergere di internet e l'enorme incremento di dati generati, immagazzinati e resi disponibili per l'analisi.

Una volta preso atto di queste innovazioni si pensò per cui che piuttosto che insegnare a computer e macchine come fare tutto, sarebbe stato molto più efficiente codificarle per pensare come esseri umani per poi collegarle a internet.

1.2.4 Tipologie di Apprendimento

Si possono definire due grandi classi di metodi di apprendimento:

- Apprendimento Supervisionato - l'algoritmo di Machine Learning usa un Dataset etichettato per dedurre l'esito desiderato. Questo però occupa notevole tempo e spazio, perché i dati necessitano di essere etichettati a mano.

- Apprendimento Non Supervisionato - non ci sono risposte predefinite o corrispondenti. L'obiettivo è ricercare i pattern nascosti tra i dati. Di solito viene utilizzato per clustering o task associativi, come raggruppare clienti per comportamento.

Mentre l'apprendimento supervisionato può essere utile, spesso dobbiamo ricorrere a un apprendimento non supervisionato. Il Deep Learning si è dimostrato un'efficace tecnica senza supervisione.

1.2.5 Reti Neurali

Le Reti Neurali sono state la chiave per permettere all'uomo di insegnare ai computer a pensare e a capire il mondo come noi lo intendiamo.
Una Rete Neurale è un sistema disegnato per classificare informazioni come il cervello umano. Come ad esempio riconoscere una immagine e classificare gli elementi che contiene.
Essenzialmente lavora su un sistema probabilistico in base ai dati forniti per addestrarla così da essere in grado di fare predizioni con un certo grado di accuratezza. Le applicazioni di Machine Learning possono variare dal riconoscere e valutare recensioni, a comprendere la melodia di un brano musicale, fino al riconoscimento di figure all'interno di immagini. L'idea di fondo di tutto ciò rimane il fatto di poter comunicare e interagire con le macchine come se interagissimo con altri esseri umani: il Natural Language Processing (NLP) è una branca dell'AI in grande sviluppo che si occupa proprio di capire la comunicazione tra esseri umani sia scritta che parlata.

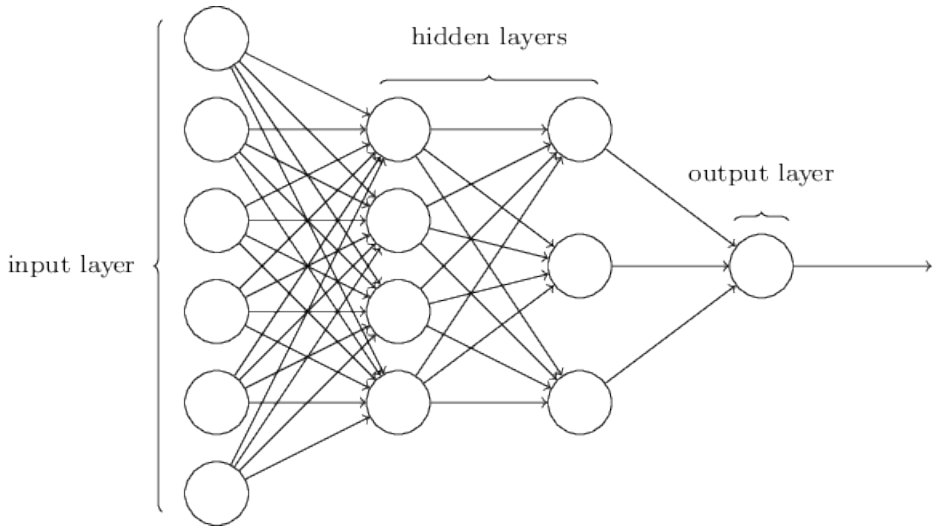

Figura 1.2: Rappresentazione di una Rete Neurale

Questi argomenti verranno poi approfonditi nel dettaglio nei capitoli successivi.

1.2.6 Confronto con il Deep Learning

Il Machine Learning prende alcune delle idee centrali dell'AI e si concentra su quelle per risolvere problemi reali attraverso le reti neurali: il Deep Learning si concentra ancor più strettamente su un sottoinsieme di tecniche e tools di ML e le applica per risolvere problemi che richiedono "pensiero" umano o artificiale.

1.3 Come funziona il Deep Learning?

Essenzialmente il Deep Learning consiste nell'alimentare un sistema informatico con numerosi dati, che può utilizzare per prendere decisioni su altri dati. Questi dati vengono alimentati attraverso reti neurali, come avviene per il Machine Learning. Queste reti sono costruzioni logiche che chiedono una serie di domande vero / falso, o estraggono un valore numerico, di ogni bit di dati che li attraversano e classificano in base alle risposte ricevute.

Il lavoro del DL è incentrato sullo sviluppo di queste reti definite anche Reti Neurali Deep. Queste reti possono essere applicate su un qualsiasi tipo di dato - segnali macchina, audio, video, parole scritte o parlate.
Fondamentale è l'addestramento di queste reti alle quali quindi devono essere messi a disposizione grossi quantitativi di dati[20].

1.4 Perché il Deep Learning è importante?

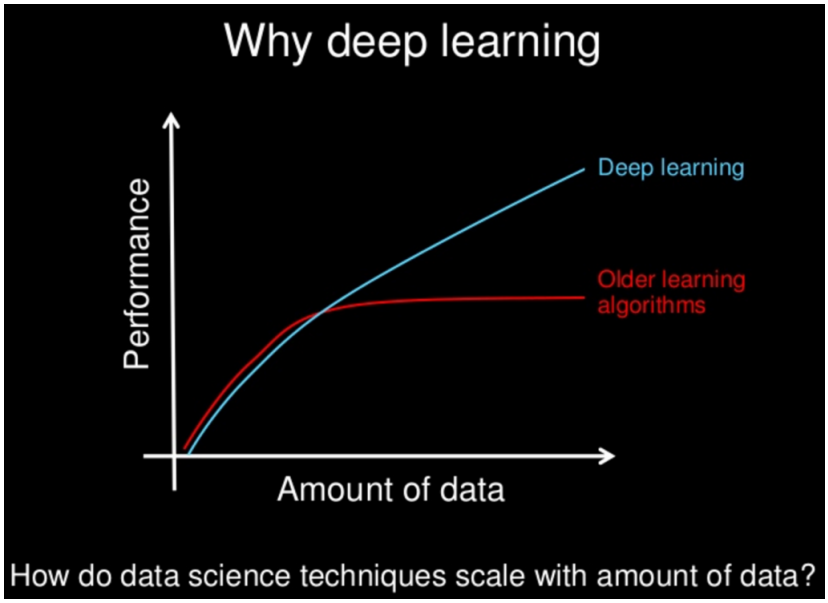

Figura 1.3: Grafico delle performance Deep Learning

I computer hanno a lungo avuto tecniche per riconoscere figure ed elementi all'interno delle immagini. I risultati non erano sempre ottimi. La Computer Vision è stata uno dei principali beneficiari del Deep Learning. Facebook ha avuto un grande successo nell'identificazione di volti all'interno di fotografie.
Il riconoscimento vocale è un altro ambito in cui il Deep Learning ha avuto un grandissimo impatto. Anche Google ne fa uso per gestire l'energia nei data centers aziendali[21].

1.5 Transfer Learning

Il Transfer Learning è un problema di ricerca nell'ambito del Machine Learning che si concentra sulla memorizzazione delle conoscenze acquisite durante la risoluzione di un problema e l'applicazione delle stesse a problemi differenti, ma correlati[22].
La necessità di utilizzare il Transfer Learning avviene nel momento in cui ci sia una fornitura limitata di dati per l'addestramento; Ciò potrebbe essere dovuto al fatto che i dati siano rari oppure costosi da raccogliere o da etichettare oppure inaccessibili. Con la sempre maggiore presenza di grandi ammassi di dati, l'opzione del Transfer Learning è diventata sempre più gettonata.

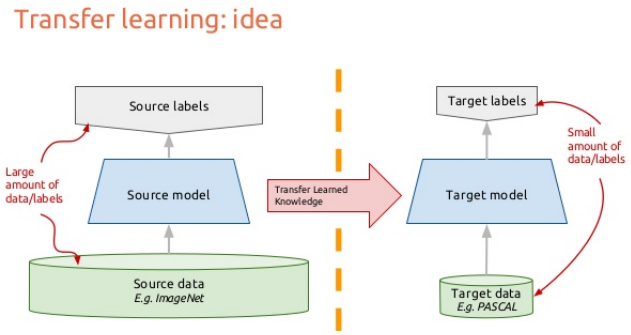

Figura 1.4: Rappresentazione del Transfer Learning

1.5.1 Definizione

Per definire in modo adatto il Transfer Learning si può utilizzare la definizione redatta da Pan[23] nel 2010.

Un dominio $\mathcal{D}$ si può dividere in due parti: uno spazio funzionale $\mathcal{X}$ e una distribuzione marginale $P(X)$, dove $X = x_1, \cdots, x_n \in \mathcal{X}$. Sempre per un dominio $\mathcal{D}$, un task $\mathcal{T}$ è definito da due parti: uno spazio delle etichette $\mathcal{Y}$ e una funzione predittiva $f(.)$ che viene appreso dal vettore delle funzioni e dalle coppie di etichette $\{x_i, y_i\}$ dove $x_i \in \mathcal{X}$ e $y_i \in \mathcal{Y}$.

Dalle definnizioni date in precedenza si può dedurre che $\mathcal{D} = \{\mathcal{X}, P(X)\}$ e $\mathcal{T} = \{\mathcal{Y}, f(.)\}$.

Ora, dato un dominio di origine $\mathcal{D}_S = \{(x_{S1}, y_{S1}), ..., (x_{Sn}, y_{Sn})\}$, dove $x_{Si} \in \mathcal{X}_S$ è l'i-esima istanza in $\mathcal{D}_S$ e y_{Si} è l'i-esima etichetta relativa all'istanza x_{Si}.
Analogamente $\mathcal{D}_T = \{(x_{T1}, y_{T1}), ..., (x_{Tn}, y_{Tn})\}$ è il dominio target dove $x_{Ti} \in \mathcal{X}_T$ è l'i-esimo istanza in $\mathcal{D}_T$ e y_{Ti} è l'i-esima etichetta relativa all'istanza x_{Ti}. Inoltre, il task sorgente è indicato come $\mathcal{T}_S$ e il task target con $\mathcal{T}_T$, mentre la funzione sorgente predittiva con $f_s(.)$ e la funzione target predittiva con $f_T(.)$ Ora che il Transfer Learning è stato definito, possiamo visualizzarlo come un processo per migliorare la funzione target $f_T(.)$ usando le informazioni relative da $\mathcal{D}_S$ e $\mathcal{T}_S$.

1.5.2 Utilizzi

Il Transfer Learning, in alternativa al Deep Learning, è diventato sempre più utilizzato tra chi possiede risorse computazionali o di dati limitati: attraverso questo, si può utilizzare un modello pre-addestrato su un dataset grande e accessibile, in modo tale da trovare layers il cui output possiede features riusabili. Usando questi outputs come inputs per addestrare un rete più piccola che richiede un numero minore di parametri. Questa rete dovrà solamente conoscere la relazione tra i pattern ricavati dai modelli pre-addestrati e il problema specifico da risolvere.

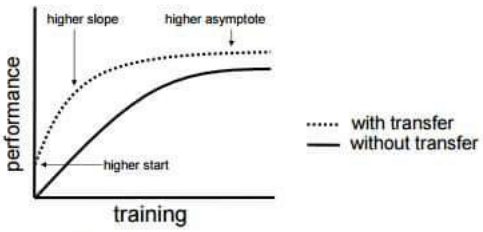

Fig. 2. Three ways in which transfer might improve learning.

Figura 1.5: Come il Transfer Learning migliora l'apprendimento

Un altro grande vantaggio del Transfer Learning è il grado di generalizzazione del modello. I modelli più grandi tendono a sovrascrivere i dati e non lavorano così bene[24].

Capitolo 2

Reti Neurali - Stato dell'arte

2.1 Introduzione

Le reti neurali sono un insieme di algoritmi, modellati ispirandosi al cervello umano, che sono progettati per riconoscere i pattern. Le reti interpretano i dati sensoriali attraverso una sorta di percezione, etichettatura o raggruppamento di input grezzi. I modelli che riconoscono sono numerici, contenuti nei vettori, dentro i quali devono essere tradotti tutti i dati reali, siano essi immagini, suoni o testo.
Le reti neurali ci aiutano a raggruppare e classificare i dati non etichettati secondo le similarità tra gli input di esempio e li classificano secondo il dataset etichettato su cui sono state addestrate.

2.2 Storia

2.2.1 Inizi

Nel 1943, neurofisiologo Warren McCulloch e matematico Walter Pitts scrissero un articolo su come i neuroni potrebbero funzionare. Per descrivere come potrebbero funzionare i neuroni nel cervello, hanno modellato una semplice rete neurale utilizzando circuiti elettrici[25].

Nel 1949 Donald Hebb scrive "*The Organization of Behavior*"[26], un lavoro che ha sottolineato il fatto che i percorsi neurali vengono rafforzati ogni volta che vengano usati, un concetto fondamentalmente essenziale per i modi in cui gli esseri umani imparano. Se due nervi trasmettono dati allo stesso tempo, ha sostenuto, la connessione tra di loro è accentuata.

2.2.2 Anni '50

Poiché i computer sono diventati più avanzati negli anni '50, è stato finalmente possibile simulare una ipotetica rete neurale. Il primo passo verso ciò è stato fatto da Nathanial Rochester dai laboratori di ricerca IBM. Purtroppo per lui, il primo tentativo di farlo è fallito.

Nel 1959 Bernard Widrow e Marcian Hoff di Stanford svilupparono modelli chiamati "ADALINE" e "MADALINE". I nomi derivano dal loro utilizzo di Multiple ADAptive LINear Elements. ADALINE è stato sviluppato per riconoscere i pattern binari in modo che, se, ad esempio, stava leggendo bit streaming da una linea telefonica, avrebbe potuto prevedere il bit successivo. MADALINE è stata la prima rete neurale applicata a un problema reale, usando un filtro adattativo che elimina echi sulle linee telefoniche. Anche se il sistema è molto antico, esso è ancora in uso nei sistemi di controllo di traffico aereo.

Nonostante il successivo successo della rete neurale, l'architettura classica di von Neumann si è presa la scena informatica e la ricerca neurale è stata per un po' accantonata. Lo stesso John von Neumann, però, suggerì l'imitazione delle funzioni neurali utilizzando relè telegrafici o tubi a vuoto.

Nello stesso periodo, è stato redatto un documento che suggerì che non sarebbe stato possibile passare da una rete con un singolo layer a una rete multi-layer. Inoltre, molti ricercatori nel campo dell'informatica, stavano utilizzando una funzione di apprendimento fondamentalmente difettosa perché non era diversificabile attraverso tutta la linea. Di conseguenza, la ricerca e il finanziamento sono scesi drasticamente.

2.2.3 Anni '70

Nel 1972, Kohonen e Anderson svilupparono una rete simile indipendentemente l'uno dall'altro. Entrambi utilizzarono matrici matematiche per descrivere le loro idee, ma non capirono immediatamente che ciò che stavano sviluppando era la creazione di una serie di circuiti analogici ADALINE. I neuroni avrebbero dovuto attivare una serie di outputs invece di una sola.

La prima rete multistrato è stata sviluppata nel 1975, una rete non supervisionata.

2.2.4 Dagli anni '80 ad ora

Nel 1986, con le reti neurali a più strati diventate più comuni, il problema era come estendere la regola Widrow-Hoff a più livelli. Tre gruppi indipendenti di ricercatori, tra cui David Rumelhart, ex membro del dipartimento di psicologia di Stanford, hanno presentato idee simili, in riferimento a reti che sono ora chiamate back-propagation networks perché distribuiscono errori di riconoscimento dei pattern in tutta la rete. Le reti ibride utilizzano solo due livelli, queste reti di propagazione ne utilizzano molti.

Ora, le reti neurali vengono utilizzate in diverse applicazioni. L'idea alla base delle reti neurali è che se funziona in natura, deve essere in grado di funzionare anche per i computer. Il futuro delle reti neurali, però, è nello sviluppo dell'hardware. Proprio come le macchine avanzate di scacchi come Deep Blue, le reti neurali veloci ed efficienti dipendono dall'hardware specificato per il suo eventuale utilizzo.

2.3 L'ispirazione dalla biologia

Il cervello è composto principalmente da circa 10 miliardi di neuroni, ciascuno collegato a circa 10 mila altri neuroni. Ogni neurone è composto dal corpo neuronale (soma) e da linee di connessione con altri neuroni che sono i canali di ingresso e uscita (dendriti e assoni) che li collegano. Ogni neurone

riceve, nei propri dendriti, input elettrochimici da altri neuroni. Se la somma di questi inputs elettrici è sufficientemente potente, il neurone si attiva e trasmette un segnale elettrochimico lungo l'assone che arriva agli altri neuroni i quali dendriti sono collegati ad uno qualsiasi dei terminali dell'assone. Questi neuroni a cui arriva il segnale, possono a loro volta, ritrasmetterlo se possiede la giusta potenza. è importante notare che un neurone si accende

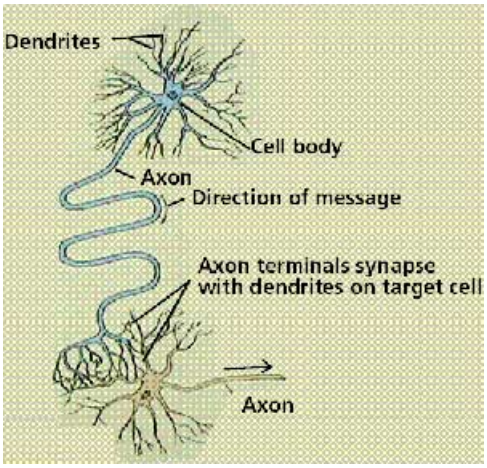

Figura 2.1: Rappresentazione di un neurone biologico

solo se il segnale totale ricevuto al corpo cellulare supera un certo livello. Così, il nostro intero cervello è composto da questi neuroni trasmittenti interconnessi tra loro. Da un numero molto elevato di unità di elaborazione estremamente semplici (ognuna eseguendo una somma ponderata dei suoi ingressi e poi sparando un segnale binario se l'ingresso totale supera un certo livello) il cervello riesce a svolgere compiti estremamente complessi.

Questo è il modello su cui si basano reti neurali artificiali. Finora, le reti neurali artificiali non sono nemmeno prossime a modellare la complessità del cervello, ma hanno dimostrato di essere buone per problemi che sono semplici per un essere umano, ma difficili per un computer tradizionale, come il riconoscimento delle immagini e le previsioni basate su conoscenza passata.

2.4 Il neurone artificiale

Le reti neurali artificiali, dette ANN (Artificial Neural Networks), sono costituite da singole unità, cioè neuroni artificiali che sono il modello matematico del neurone biologico. Mentre per i neuroni biologici i segnali elettrici sono ricevuti attraverso i dendriti dagli assoni di altri neuroni, in quelli artificiali questi segnali sono rappresentati da valori numerici. Nelle sinapsi tra il dendrite e gli assoni, i segnali elettrici sono modulati in varie quantità, questo viene modulato nei neuroni artificiali moltiplicando ogni valore in input in un valore chiamato peso (*weight*).

Per modellare il fatto che un neurone trasmetta un segnale elettrico solo al superamento di una certa soglia, viene calcolata la somma ponderata di tutti gli input per rappresentare la forza totale dei segnali in input e applicando una funzione di step sulla somma per determinare l'output; questo output poi sarà l'input per altri neuroni come per le reti biologiche.

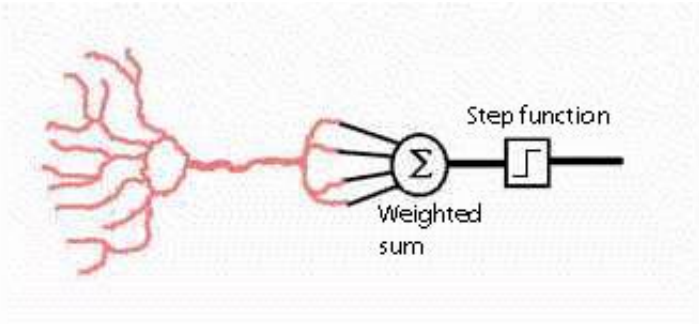

Figura 2.2: Rappresentazione di un neurone artificiale

Si possono definire due tipi di vettori:

- Input Vector - Vettore che rappresenta ogni singolo valore ricevuto in input da un neurone artificiale.

- Weight Vector - Vettore che rappresenta tutti i pesi di un neurone artificiale.

2.4.1 Funzionamento

Come menzionato sopra, un neurone artificiale calcola la somma ponderata dei valori in input. Per semplicità, assumiamo esserci due valori in input, x e y per un certo neurone P, con i pesi per x y rispettivamente A B e la cui somma ponderata è rappresentata come: $Ax + By$.

Poiché il neurone artificiale invia in output un valore che non sia zero solo se la somma ponderata supera la soglia C, si può scrivere l'output del neurone come segue:

$$P = \begin{cases} Ax + By > C \rightarrow 1 \\ Ax + By \leq C \rightarrow 0 \end{cases}$$

Le equazioni $Ax + By > C$ e $Ax + By < C$ sono le due regioni sul piano xy separate dalla linea $Ax + By + C = 0$. Se consideriamo l'input (x,y) come punto su un piano allora il neurone ci dice a quale regione del piano il punto appartiene. Tali regioni, in quanto separate da una singola linea, sono chiamate regioni linearmente separabili.

Questo concetto è utile perché risulta che alcune funzioni logiche come gli operatori booleani AND, OR e NOT sono linearmente separabili, cioè possono essere eseguiti usando un singolo neurone. Possiamo illustrare perché sono linearmente separabili, come illustrato in figure

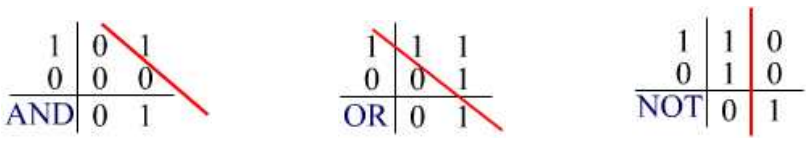

Figura 2.3: Grafici che mostrano le regioni linearmente separabili

Nei grafici di cui sopra, i due assi sono gli inputs che possono assumere il valore di 0 o 1 e i numeri sul grafico sono l'output previsto per un determinato input. Utilizzando un Weight Vector appropriato per ogni caso, un singolo neurone artificiale può eseguire tutte queste funzioni.

Tuttavia, non tutti gli operatori logici sono linearmente separabili. Ad esempio, l'operatore XOR non è linearmente separabile e non può essere

raggiunto da un singolo neurone. Tuttavia questo problema potrebbe essere superato utilizzando più di un neurone organizzato in reti *feed-forward*.

1	1	0
0	0	1
XOR	0	1

Figura 2.4: Grafico dello XOR

2.4.2 Reti Feed-Forward

Le Reti di tipo Feed-Forward hanno le seguenti caratteristiche:

- I neuroni artificiali sono organizzati in layers, con il primo layer che si occupa degli input e l'ultimo degli output. I layers centrali non sono connessi con l'esterno e quindi sono chiamati *Hidden Layers* (Strati nascosti).
- Ogni neurone artificiale in un layer è connesso con ogni neurone dello strato successivo. Quindi l'informazione è costantemente "Spinta in avanti" da un layer all'altro, da qui il nome delle reti.
- Non c'è nessuna connessione tra i neuroni dello stesso layer.

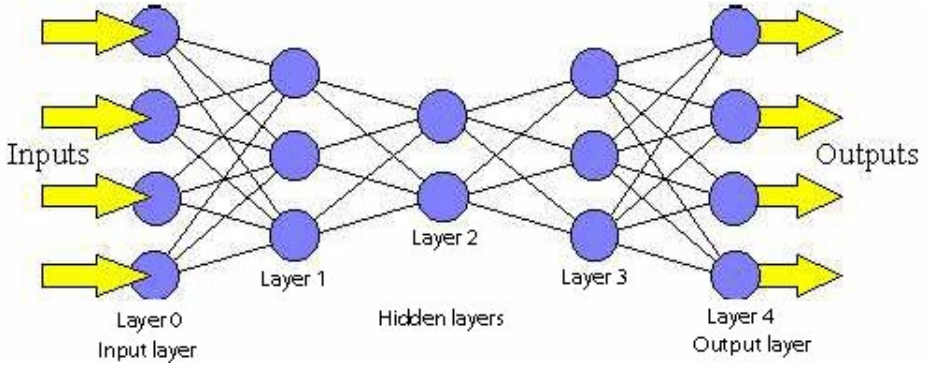

Figura 2.5: Una Rete Feed-Forward

Un singolo neurone artificiale può per cui classificare punti appartenenti a due regioni linearmente separabili. Considerando la rete in Figura 2.6, vediamo cosa succede se i due punti appartengono a regioni che non sono linearmente separabili Con un input (x,y) fornito alla rete attraverso l'input layer; con

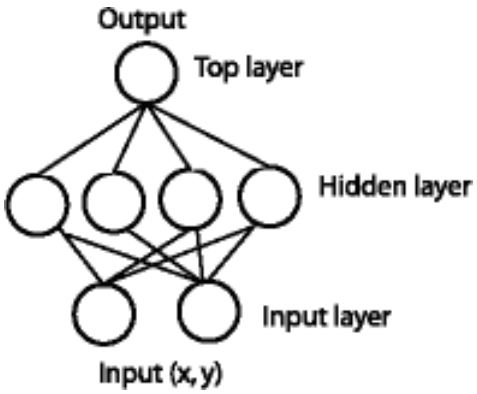

Figura 2.6: Una rete Feed-Forward con un Hidden Layer

quattro neuroni indipendenti appartenenti all'hidden layer, il punto è classificato in quattro coppie di regioni linearmente separabili, ognuna delle quali ha una unica linea separatrice.

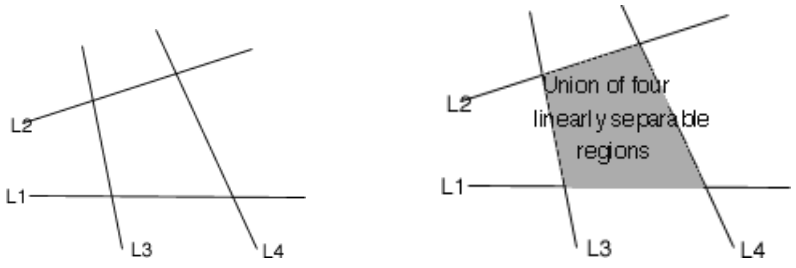

Figura 2.7: Le 4 linee che dividono il piano e la loro intersezione

Il neurone superiore esegue operazioni logiche sulle output degli hidden layers in modo che l'intera rete sia in grado di classificare i punti di input in 2 regioni che potrebbero non essere linearmente separabili. Ad esempio, utilizzando l'operatore AND su queste quattro uscite, si ottiene l'intersezione delle 4 regioni che costituiscono la regione centrale.

Modificando il numero di nodi nell'hidden layer, il numero di livelli e il numero di nodi di input e output, è possibile classificare punti in dimensione arbitraria in un numero arbitrario di gruppi. Di conseguenza, le reti feed-forward sono comunemente utilizzate per la classificazione.

2.4.3 Backpropagation - Addestramento di reti Feed-Forward

L'addestramento di reti Feed-Forward appartiene alla tipologia di addestramento detto supervisionato, dove la coppia di input e output sono inserite nella rete per molti cicli, cosicché la rete "impari" le relazioni tra input e output.

Viene fornito alla rete un certo numero di campioni addestranti, che consistono in un Input Vector i e il suo output o. La regola base per scegliere il numero di nodi di output dipende dal numero di regioni che si formano; è consigliabile usare una notazione unica per rappresentare le differenti regioni, cioè, ad esempio, per ogni output solo un nodo può avere valore 1; quindi il numero nodi output corrisponde al numero di regioni differenti - 1.

Nell'addestramento di backpropagation, ogni volta che un Input Vector della campionatura viene presentato, l'Output Vector o è comparato al valore desiderato d. Il confronto viene fatto calcolando il quadrato della differenza dei due valori

$$Err = (d - o)^2 \tag{2.1}$$

Il valore di Errore (Err) comunica quanto distante sia il valore desiderato da un particolare input. L'obiettivo della Backpropagation è la minimizzazione della somma degli Errori all'interno della campionatura. In modo tale che la rete si comporti nel modo desiderato.

$$\min \sum Err = (d - o)^2 \tag{2.2}$$

Possiamo esprimere Err in termini di Input Vector(i), Weight Vector(w) e della funzione di soglia delle percezioni. Usando una funzione continua,

invece che una funzione di step, come una funzione di soglia, si può esprimere il gradiente di Err in riferimento a w in termini di w e i.

Stando al fatto che diminuendo il valore di w nella direzione in cui conduce il gradiente per la diminuzione più rapida in Err, si aggiorna il Weight Vector ogni volta che un campione viene presentando utilizzando la formula:

$$w_{new} = w_{old} - n\frac{\delta Err}{\delta w} \tag{2.3}$$

dove n è il learning rate (tasso di addestramento); usando l'algoritmo i Weight Vectors sono modificati un po' ogni volta che un campione viene presentato. Quando tutti gli input di addestramento sono inviati alla rete a turno per molti cicli, la somma di Err diminuisce gradualmente fino a raggiungere un valore minimo, che è l'obiettivo della funzione.

2.4.4 Reti Ricorrenti

Le reti neurali, come già definito precedentemente, possono essere rappresentate come grafi (con nodi e archi pesati), nei quali non ci sono cicli, come nel caso delle reti *Feed-Forward* oppure dove sono presenti cicli, che è il caso delle *Reti Ricorrenti*.

L'idea di fondo delle RNN è di utilizzare le informazioni sequenzialmente: in una rete neurale tradizionale si assume che tutti gli inputs (e outputs) siano indipendenti gli uni dagli altri, ma per alcuni tasks non è un approccio corretto. Infatti se ad esempio si desidera predire la parola successiva all'interno di una frase è importante sapere quale parola sia stata elaborata in precedenza. Le RNN sono chiamate "ricorrenti" perché eseguono lo stesso task per ogni elemento della sequenza in esame, con l'output che dipende dalla computazione precedente.

Si può pensare alle RNN come se avessero una "memoria" che catturi l'informazione su quello che è stato già calcolato. Teoricamente, le RNN possono utilizzare informazioni in sequenze arbitrariamente lunghe, ma in pratica sono limitate a guardare indietro solo di qualche step. Il grafo in figura 2.8 mostra una RNN che viene dispiegata in una rete completa. Dispiegarla si-

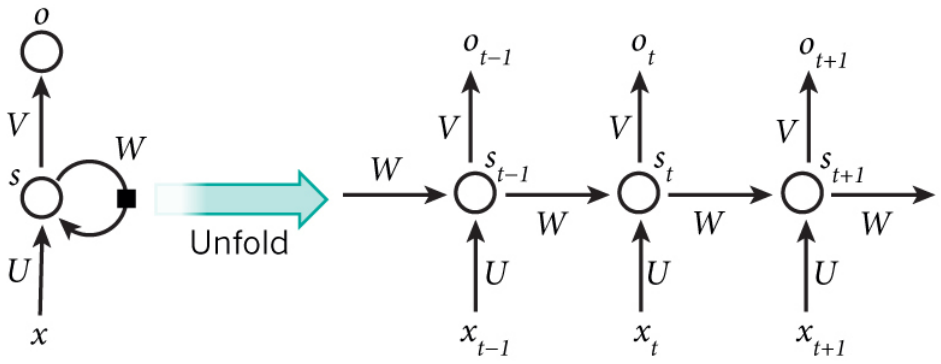

Figura 2.8: Rete RNN dispiegata, con parametri U,V e W

gnifica semplicemente di riscrivere la rete nella sua sequenza completa. Le formule che regolano il calcolo in una RNN sono le seguenti:

- x_t è l'input allo step t.
- s_t è lo stato nascosto allo step t. è la "memoria" della rete; s_t è calcolato in base allo stato nascosto precedente e l'input allo step corrente corrisponde: $s_t = f(U_{x_t} + W_{s_{t-1}})$. La funzione f solitamente non è lineare. $s - 1$, è tipicamente inizializzato a zero, perché è richiesto per calcolare il primo stato nascosto.
- o_t è lo step di output. Corrisponde a $o_t = softmax(V_{s_t})$.

Si può pensare allo stato nascosto s_t come la memoria della rete, il quale cattura l'informazione su cosa sia successo negli step precedenti. L'output o_t è calcolato solo sulla base della memoria allo step t.

A differenza di una rete neurale tradizionale, che utilizza diversi parametri ad ogni livello, una RNN condivide gli stessi parametri (U, V, W sopra) in tutti gli step. Questo riflette il fatto che si stia eseguendo lo stesso task ad ogni step, solo con diversi input. Ciò riduce notevolmente il numero totale di parametri che la rete deve imparare.

Il grafo in Figura 2.8 produce output in ogni step, ma a seconda del task potrebbe non essere necessario. Ad esempio, quando si esegue Sentiment

Analysis di una frase, l'importante è l'output finale, non dopo ogni parola. Allo stesso modo, potrebbe non avere bisogno di input a ogni step. La caratteristica principale di un RNN è il suo stato nascosto, che cattura alcune informazioni su una sequenza.

Le RNN hanno avuto notevole successo nella risoluzione dei tasks di NLP (Natural Language Process); la più utilizzata RNN è LSTM (Long-Short Term Memory). Alcuni esempi di utilizzo di RNN sono: modellazione del linguaggio e generazione automatica di testo, traduzione automatica di testo e riconoscimento vocale e di immagini.

2.5 Addestramento di una rete neurale

L'interesse principale delle reti neurali è per la possibilità di apprendimento e per l'addestramento. Dato uno specifico task da risolvere e una classe di funzioni $\boldsymbol{F}$, addestrare la rete significa usare un set di osservazioni per trovare $f^* \in \mathbf{F}$ che risolve il task in modo ottimale.

Ciò comporta la definizione di una funzione di costo $C : F \to \mathbb{R}$, cosicché la soluzione ottima f^* è $C(f^*) \leq C(f) \forall f \in F$, quindi nessuna soluzione costa meno di quella ottima.

La funzione di costo è un concetto importante nell'apprendimento, in quanto misura quanto lontano sia una particolare soluzione da una ottimale per il task da risolvere. L'algoritmo di apprendimento cerca nello spazio delle soluzioni per trovare una funzione che ha il minor costo possibile.

Per applicazioni dove la soluzione dipende dai dati, il costo deve necessariamente essere una funzione delle osservazioni, altrimenti il modello non è in relazione con i dati; ciò è spesso definito come una statistica a cui possono essere apportate solo approssimazioni. Ad esempio, considerando il problema di trovare il modello f, che minimizzi $C = E[(f(x) - y)^2]$, per una coppia (x,y) pescata dall'insieme $\mathcal{D}$. In una situazione pratica avremmo solamente

N campioni da $\mathcal{D}$ e quindi si dovrà minimizzare

$$\widehat{C} = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2 \tag{2.4}$$

Per cui il costo è minimizzato su un subset dei dati invece che su tutta la distribuzione; è necessario avere un modello che definisca in quale ambiente opera una rete neurale, questi modelli si possono definire come Paradigmi di apprendimento che distingue le regole e gli algoritmi per il corretto apprendimento.

2.5.1 Paradigmi di Apprendimento

Si possono distinguere 3 tipologie di paradigmi di apprendimento, come già visto anche in ambito di Machine Learning: Apprendimento Supervisionato, Non Supervisionato e Ibrido.

Apprendimento Supervisionato

Tecnica di apprendimento automatico che mira a istruire un sistema informatico in modo da consentirgli di risolvere dei compiti in maniera autonoma sulla base di una serie di esempi ideali, costituiti da coppie di input e di output desiderati, che gli vengono inizialmente forniti. L'obiettivo di un sistema basato sull'apprendimento supervisionato è quello di produrre un'ipotesi induttiva ossia una funzione in grado di produrre i risultati forniti durante la fase di esempio e in grado di avvicinarsi a dei risultati desiderati per tutti gli esempi non forniti. Una sua variante è il *Reinforcement Learning*.

Apprendimento Non Supervisionato

L'apprendimento non supervisionato è una tecnica di apprendimento automatico che consiste nel fornire al sistema informatico una serie di input che egli riclassificherà ed organizzerà sulla base di caratteristiche comuni per cercare di effettuare ragionamenti e previsioni sugli input successivi. Al contrario dell'apprendimento supervisionato, durante l'apprendimento vengono

forniti al sistema solo esempi non annotati, in quanto le classi non sono note a priori ma devono essere apprese automaticamente.

Apprendimento Ibrido

Paradigma di apprendimento che comporta una soluzione ibrida delle tue tecniche analizzate precedentemente. Una parte è determinata da apprendimento non supervisionato e il resto da apprendimento non supervisionato.

2.6 Tipologie di Reti Neurali

Di seguito verranno analizzate le più conosciute tipologie di reti neurali e il loro stato dell'arte.

2.6.1 Long-Short Term Memory

L'apprendimento per memorizzare informazioni su intervalli di tempo prolungati attraverso backpropagation ricorrente richiede molto tempo, soprattutto a causa di un Backflow Error insufficiente. Hochreiter e Schmidhuber analizzano il problema e offrono una soluzione, efficiente e basata sul gradiente, quale la rete neurale di tipo Long-Short Term Memory (LSMT)[27].

Troncando il gradiente laddove è possibile farlo, LSTM può imparare a colmare il minimo ritardo di tempo, in eccesso di 1000 time step, applicando un flusso costante di errore all'interno di unità speciali (Gate Units). Le Gate Units imparano ad aprire e chiudere l'accesso al flusso di errore costante.

LSTM è una rete neurale ricorrente, locale nello spazio e nel tempo, ed ha una complessità computazionale per step e weight pari a $O(1)$; è inoltre in grado di risolvere tasks complessi, artificiale e con lunghi ritardi, che non erano mai stati risolti prima da una rete ricorrente. Inizialmente, le reti ricorrenti, ad esempio le Short-Term Memory, utilizzavano le connessioni di feedback per immagazzinare informazioni degli input più recenti; ciò è

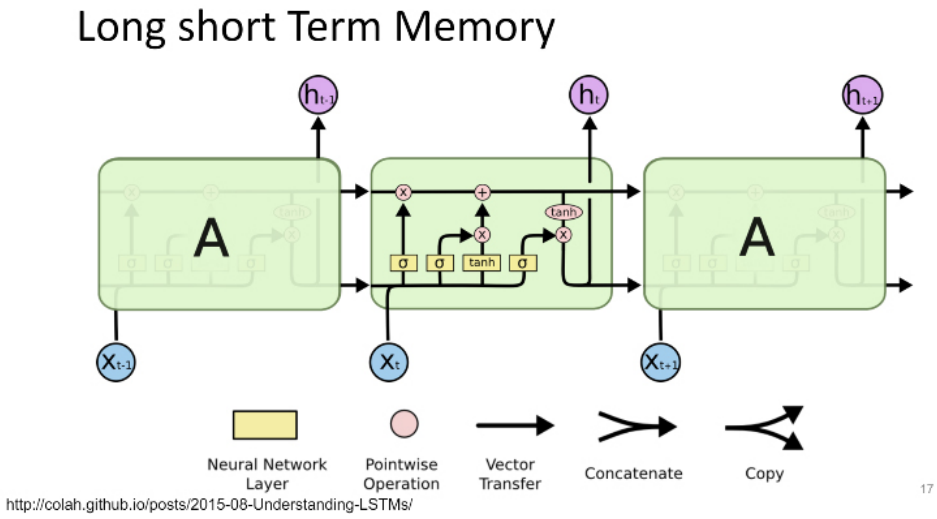

Figura 2.9: Rappresentazione di una LSTM

potenzialmente significativo per molte applicazioni come il riconoscimento vocale, ma se lo scarto di tempo tra l'input e il segnale successivo non è più minimo, allora questo tipo di rete non funziona correttamente e in tempi brevi; in alternativa a ciò, le LSTM, cambiando i valori dei pesi lentamente, riescono a ovviare a queste problematiche. L'espressione Long-Short Term si riferisce al fatto che LSTM è un modello di Short-Term applicato a un lungo periodo di tempo.

Un blocco LSTM è composto da quattro componenti principali: una cella, un input gate, un output gate e una forget gate. La cella è la responsabile per "ricordare" informazioni ogni intervallo di tempo arbitrario.

Ognuno dei tre gate può essere considerata come un neurone artificiale "convenzionale", come in una rete neurale multi-layer (o feedforward): cioè calcolano un'attivazione (utilizzando una funzione di attivazione) di una somma ponderata. Intuitivamente, possono essere considerati come regolatori del flusso di valori che attraversa le connessioni del LSTM; da qui la denotazione "gate". Ci sono connessioni tra questi gate e la cella. Alcune delle connessioni sono ricorrenti, alcune non lo sono. LSTM è adatto per classificare,

processare o predire una serie di dati temporali di dimensione sconosciuta.

2.6.2 Neural Turing Machines

Le reti neurali ricorrenti (RNN) si distinguono da altri metodi di Machine Learning per la loro capacità di apprendere e di eseguire complicate trasformazioni dei dati su lunghi periodi di tempo. Inoltre, è noto che le RNN sono Turing-Complete (Siegelmann e Sontag, 1995), e quindi hanno la capacità di simulare processi arbitrari, se correttamente cablate. Quindi vengono arricchite le capacità delle reti ricorrenti standard per semplificare la soluzione di algoritmi. Questo arricchimento è dovuto principalmente a una grande memoria indirizzabile, quindi, analogamente alla macchina di Turing a stati finiti con un nastro di memoria infinito, definiamo il nostro dispositivo una "Neural Turing Machine" (NTM). A differenza di una macchina Turing, un NTM è un computer diversificabile che può essere addestrato per la discesa del gradiente, rendendo questo meccanismo pratico per i programmi di apprendimento[28]. Le NTM contengono due componenti di base: un

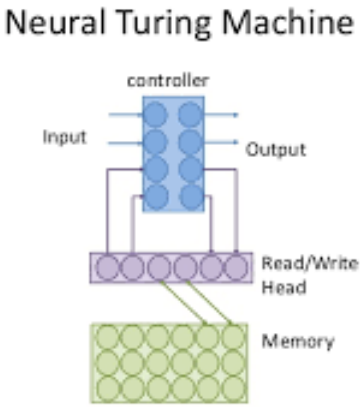

Figura 2.10: Rappresentazione di una NTM

controller e una memoria. Il controller si occupa di interagire con il mondo esterno attraverso Input e Output Vectors. A differenza di reti neurali

tradizionali, la NTM interagisce con con una matrice di memoria usando operazioni di Lettura/Scrittura, per analogia con la macchina di Turing le reti di output che parametrizzano queste operazioni vengono chiamate "heads".

Il controller è la parte centrale della NTM e può essere di due tipologie:

- Ricorrente - come per le LSTM, il controller ricorrente ha una propria memoria interna che può essere complementare alla matrice di memoria.
- Feed-Forward - un controller feed-forward può imitare quello ricorrente leggendo e scrivendo la stessa locazione in memoria ad ogni step. Inoltre, spesso conferiscono maggiore trasparenza al funzionamento della rete perché il modello di lettura e scrittura sulla matrice di memoria è solitamente più facile da interpretare rispetto allo stato interno di un RNN.

Lettura

Sia M_t il contenuto della matrice di memoria NxM al tempo t, dove N è il numero di locazioni di memoria e M è la dimensione del vettore per ogni locazione. Sia w_t un vettore dei pesi sulle N locazioni emesse da una head di lettura al tempo t. Fino a quando tutti i pesi sono normalizzati, gli N elementi $w_t(i)$ di w_t seguono i seguenti vincoli:

$$\sum_i w_t(i) = 1, \quad 0 \leq w_t(i) \leq 1, \forall i. \tag{2.5}$$

La lunghezza M del read vector r_t restituito dalla lettura, è definito come una combinazione convessa dei vettori-riga $M_t(i)$ in memoria:

$$r_t \longleftarrow \sum_i w_t(i) M_t(i) \tag{2.6}$$

Scrittura

Prendendo ispirazione dagli input e forget gates nelle LSTM, la scrittura si decompone in due parti: una eliminazione seguita da una aggiunta.

Dato un peso w_t emesso da una write head al tempo t, insieme con un vettore di cancellazione e_t con M elementi tutti nell'intervallo (0,1), i vettori memoria $M_{t-1}(i)$ dallo step precedente sono modificati come segue:

$$\tilde{M}_t(i) \longleftarrow M_{t-1}(i)[1 - w_t(i)e_t] \tag{2.7}$$

dove 1 è il vettore-riga tra tutti. Di conseguenza, gli elementi di una posizione di memoria vengono azzerati solo se entrambi i pesi nella posizione e l'elemento di cancellazione sono uno; se sia i pesi o la cancellazione sono zero, la memoria rimane invariata. Quando sono presenti più write heads, le cancellazioni possono essere eseguite in qualsiasi ordine, Poiché la moltiplicazione è commutativa.

Ogni write heads produce anche un vettore somma a_t di lunghezza M, che viene aggiunto alla memoria dopo che è stata eseguita la fase di cancellazione:

$$M_t(i) \longleftarrow \tilde{M}_t(i)a_t \tag{2.8}$$

Ancora una volta, l'ordine con cui viene effettuata l'aggiunta è irrilevante. Le operazioni di cancellazione e somma di tutte le write heads, combinate, producono un risultato finale al tempo t.

2.6.3 Neural Network with Dynamic External Memory

L'ultimo modello delle reti neurali che verrà analizzato è chiamato Differentiable Neural Computer (DNC), che significa che la rete neurale può leggere e scrivere da una matrice di memoria esterna analoga alla RAM di un computer[31]. Come un computer convenzionale, può utilizzare la memoria per rappresentare e manipolare strutture dati complesse, ma come le reti neurali, può imparare a fare ciò dai dati.

Una DNC è una rete neurale accoppiata a una matrice di memoria esterna; il comportamento della rete è indipendente dalla dimensione della memoria. Se la memoria esterna può essere pensata come una RAM, il controller, che si

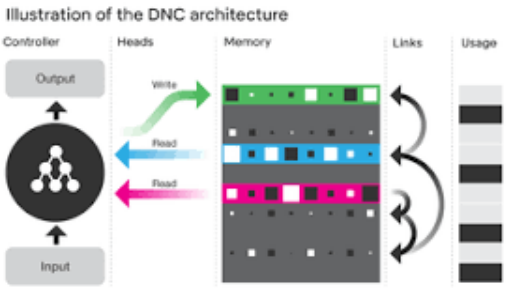

Figura 2.11: Rappresentazione di una DNC

occupa di definire le operazioni da eseguire, è una CPU differenziabile le cui operazioni sono imparate con la discesa del gradiente. La Neural Turing Machine si può definire come una rete con struttura simile ma con accesso alla memoria molto più limitato. Mentre i computer tradizionali usano indirizzi univoci per accedere ai contenuti in memoria, le DNC utilizzano un meccanismo differenziabile per definire la distribuzione di dati sopra le N righe nella $N \times W$ matrice di memoria M. Questa distribuzione, che possiamo chiamare ponderazione, rappresenta il grado in cui ogni locazione è coinvolta in operazioni di lettura/scrittura. Il Read Vector r restituito da una lettura pesata w^r sulla memoria M è una somma ponderata sulle locazioni di memoria: $r = \sum_{i=1}^{N} M[i, .]w^r[i]$, dove il punto denota tutti gli $j = 1, \cdots W$. Similmente, l'operazione di scrittura usa una scrittura pesata w^w prima per eliminare, con l'Erase Vector e, poi aggiungendo un Write Vector $v : M[i, j] \longleftarrow M[i, j]$ $(1 - w^w[i]e[j]) + w^w[i]v[j]$. Le unità funzionale che si occupano di lettura e scrittura sono chiamate Read and Write Heads.

In conclusione, Le DNC vengono utilizzate nello stesso ambito delle NTM e DMN, ma hanno ottenuto risultati più vantaggiosi in quanto utilizzano meglio e in modo più efficiente la memoria esterna.

2.6.4 Dynamic Memory Networks

La maggior parte dei tasks in NLP possono essere riassunti in problemi di risposta alle domande su input linguistici(QA - Question Answering). In

questo ambito si possono per cui introdurre le Dynamic Memory Networks (DMN), un'architettura di rete neurale che elabora le sequenze di input e domande, crea memorie temporanee e genera risposte pertinenti. Le domande innescano un processo iterativo che consente al modello di concentrare la sua attenzione sugli input e sul risultato delle iterazioni precedenti. Il DMN può essere addestrato in modalità end-to-end e ha ottenuto risultati importanti su diversi tipi di tasks e set di dati: come per il QA, sentiment classification e riconoscimento vocale[29].

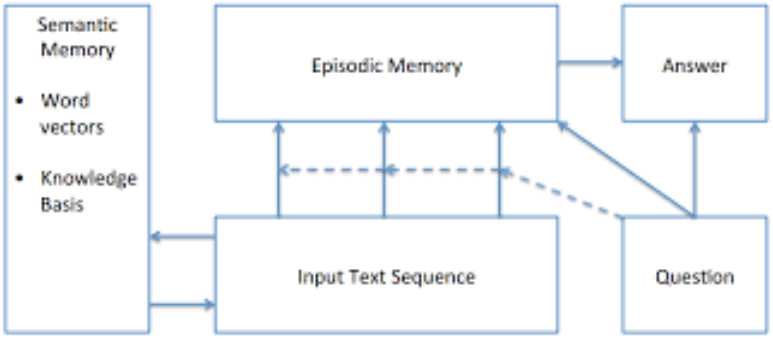

Figura 2.12: Rappresentazione ad alto livello di una DMN

Ecco una panoramica dei moduli che compongono le DMN:

- Input Module - Codifica gli input testuali grezzi dal task alla rappresentazione vettoriale.
- Question Module - Codifica le domande nel task in una rappresentazione vettoriale.
- Episodic Memory Module - Data una collezione di rappresentazioni di input, sceglie su quale parte dell'input concentrare il meccanismo. Poi produce un vettore di memoria che tiene conto della domanda e della memoria precedente. Ogni iterazione fornisce al modulo informazioni in più sull'input: in altre parole, il modulo ha la capacità di recuperare nuove informazioni, nella forma delle rappresentazioni di input, cui si pensava essere irrilevanti nelle iterazioni precedenti.

- Answer Module - Genera una risposta dal vettore di memoria finale generato dall'Episodic Module.

Il modello DMN è un'architettura potenzialmente generale per varietà di applicazioni NLP, compresa la classificazione, il QA e la modellazione di sequenze. Una singola architettura è un primo passo verso un unico modello congiunto per problemi multipli di NLP. Il DMN è addestrato in modalità end-to-end con una, seppur complessa, funzione oggettiva. Un lavoro futuro potrà prevedere l'esplorazione di modi per scalare il modello con più inputs, che potrebbe essere effettuato eseguendo un sistema di recupero di informazioni per filtrare gli ingressi più rilevanti prima di utilizzare la DMN, o utilizzando un modulo gerarchico.

2.6.5 Convolutional Neural Networks

Le ConvNets sono una particolare tipologia di reti neurali che hanno trovato, negli ultimi anni, risultati molto buoni negli ambiti quali la sentiment analysis, computer vision e riconscimento di immagini. Recentemente però le CNN sono state utilizzate anche per problemi di NLP, con risultati interessanti[16].

Per comprendere al meglio il concetto di Convolution per una immagine, si può utilizzare un esempio classico di una finestra scorrevole applicata ad una matrice (come si può vedere in Figura 2.13) La matrice sulla sinistra di colore verde rappresenta l'immagine in esame in bianco e nero. Ogni casella corrisponde a un pixel che può assumere valori quali 0(nero) o 1(bianco), se si utilizza una scala di grigi i valori possono andare da 0 a 255. La matrice di colore giallo rappresenta la finestra scorrevole chiamata *kernel, filter o feature detector*. In questo caso il *filter* è 3x3, per ottenere la convolution rappresentata dalla matrice di destra si fa scorrere la finestra su tutti gli elementi della matrice e l'operazione da compiere consiste nel moltiplicare ogni elemento della finestra con la parte di matrice originale presa in esame e poi sommare, i valori ottenuti.

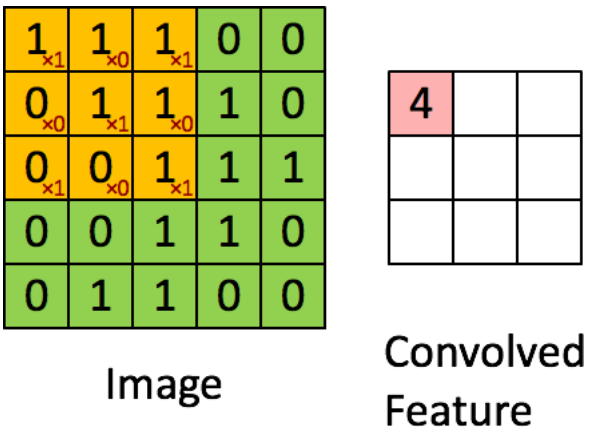

Figura 2.13: Schema di una Convolution

Dopo aver introdotto la convolution, si possono definire le CNN che altro non sono che diversi layers di convolutions con funzioni di attivazione non-lineari (ad esempio ReLU o tanh) applicate ai risultati dopo ogni convolution.

In una CNN, a differenza delle tradizionali reti feed-forward in cui ogni neurone di un layer è collegato a ciascun neurone del layer successivo, su ogni layer viene applicata la convolution per ottenere l'output. Ogni layer applica un filtro e in seguito poi i risultati vengono combinati. Durante la fase di addestramento la CNN automaticamente impara i valori dei suoi filtri, basati sul task in esame.

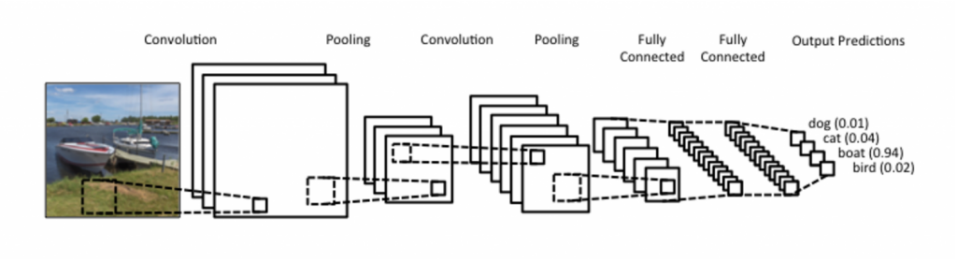

Figura 2.14: Layers di una CNN

Ci sono due aspetti importanti durante il processo di calcolo:

- Location Invariance - Grazie al fatto che il filtro è scorrevole, non importa dove si trovi l'oggetto di interesse all'interno dell'immagine, o se

sia ruotato o di differenti dimensioni.

- Compositionality - Ogni filtro compone una patch locale di features di basso livello in una rappresentazione di alto livello.

In NLP, gli inputs della rete neurale sono frasi o documenti rappresentati sotto forma di matrice: Ogni riga della matrice tipicamente corrisponde a una parola, ma potrebbe essere anche un carattere. Per rappresentare le frasi queste devono essere codificate con rappresentazioni quali *word2vec, GloVe* oppure *One-Hot*.

Questa volta i filtri scorrono sopra le righe della matrice, in questo modo la larghezza dei filtri è come quella della matrice in input, mentre l'altezza può variare fino a 2-5 parole ogni volta.

La Location Invariance e la Compositionality sono proprietà che però non possono valere per le frasi, perché ad esempio i pixel vicini spesso sono correlati semanticamente, cosa che non vale sempre per le frasi. Ciò può far sembrare che le CNN non siano adatte per NLP, ma, trascurando i modelli che non sono adatti, esse possono essere performanti anche in relazione al fatto che sono una tipologia di rete estremamente veloce.

Ci sono alcuni iper-parametri fondamentali quali la tipologia di convolution: *wide* utilizza una politica di inserire uno zero per riempire gli spazi vuoti nel caso in cui venga applicato il filtro a un elemento della matrice che si trova nel bordo, in questo modo si può applicare il filtro a ogni elemento della matrice , ma si ottiene un output di dimensione uguale o maggiore della matrice oppure la politica opposta *narrow* in cui non si usa questo riempimento.

Un secondo parametro importante è la *stride* ovvero di quanto il filtro deve scorrere.

L'ultimo aspetto importante delle CNN sono i Pooling Layers applicati a seguito dei Convolutional. Essi estraggono dall'input un sottoinsieme, l'operazione di pooling più comune è applicare una operazione di *max* al risultato del filtro. Non è necessario applicare il pooling a tutta la matrice, ma anche a solo una finestra. Viene effettuata questa operazione in particolare

perché fornisce una matrice di output fissata, che è tipicamente richiesto per la classificazione. Oltretutto riduce la dimensione dell'output ma mantiene le informazioni salienti.

L'architettura e il funzionamento di CNN verranno discussi anche nel prossimo capitolo in modo dettagliato, sia più a livello generale, sia in una particolare architettura di ConvNets chiamata Character-Level Convolutional Neural Network.

2.6.6 Dynamic Convolutional Neural Networks

Le Dynamic Convolutional Neural Networks (DCNN) sono una architettura di rete neurale che viene utilizzata per la modellazione semantica delle frasi[30]. La rete utilizza Dynamic k-Max Pooling, una operazione di pooling globale sopra le sequenze lineari. La rete gestisce l'input di frasi di lunghezza variabile e crea un grafo delle funzioni sopra la frase che è in grado di catturare esplicitamente relazioni brevi e lunghe tra parole. La rete non fa affidamento su un albero di parse ed è facilmente applicabile a qualsiasi lingua.

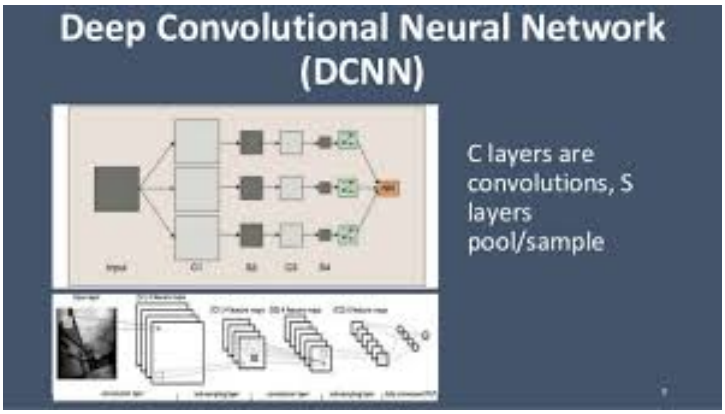

Figura 2.15: Rappresentazione di una DCNN

L'obiettivo della modellazione delle frasi è quello di analizzare e rappresentare la semantica del contenuto di una frasi per uno scopo quale la classificazione o la generazione delle frasi. Il sentence modelling è un problema

chiave di molti task quali la sentiment analysis, machine translation, image retrieval ecc... : le DCNN sono un modello di rete neurale che si propongono per risolvere questi tipi di tasks.

In una DCNN le frasi vengono modellate usando una architettura di tipo convolutional che alterna grandi convolutional layers a pooling layers dinamici. Nella rete la larghezza della mappa delle features in layers intermedi dipende dalla lunghezza della frase in input.

Data una frase in input, per ottenere il primo layer di una DCNN, si prende per ogni parola della frase la corrispettiva incorporazione $w_i \in \mathbb{R}^d$ e viene costruita la matrice della frase $s \in \mathbb{R}^{d \times s}$. I valori di w_i sono parametri che vengono ottimizzati durante il training. Il convolutional layer nella rete è ottenuto convolvendo una matrice dei pesi(weights) $m \in \mathbb{R}^{d \times s}$ con la matrice di attivazione del layer sottostante. Ad esempio, il secondo layer è ottenuto applicando una convolution alla frase della matrice s stessa. La dimensione d e il filtro di larghezza m sono iper-parametri della rete. La matrice risultante ha dimensioni $d \times (s + m - 1)$.

Ecco la descrizione dell'operazione di pooling che si susseguono nella costruzione di una DCNN: dato un valore k e una sequenza $p \in \mathbb{R}^p$ di lunghezza $p \geq k$ k-max pooling seleziona la sottosequenza p^k_{max} del più alto k valore di p. L'ordine dei valori in p^k_{max} corrisponde all'ordine originale in p.

L'operazione di k-max pooling rende possibile di mettere insieme le k features più attive in p che può essere un numero di posizioni diverse; ciò preserva l'ordine delle features, ma non è sensibile alla specifica posizione. L'operatore di k-max pooling è applicato alla rete dopo il convolutional layer; questo garantisce che l'input ai fully-connected layers sia indipendente dalla lunghezza della sequenza in input.

Invece una operazione di tipo Dynamic k-max Pooling 'e una operazione di tipo k-max dove si lascia che k sia una funzione della lunghezza della frase e della profondità della rete, e si può semplificare come:

$$k_l = \max(k_{top}, \lfloor \frac{L - l}{L} s \rfloor) \tag{2.9}$$

dove l è il numero del convolutional layer corrente a cui è applicato il pooling e L è il numero totale di convolutional layers nella rete; k_{top} è il parametro di pooling fissato tra i top convolutional layers.

Dopo l'applicazione di k-max pooling al risultato della convolution, una polarizzazione $b \in \mathbb{R}^d$ e una funzione non-lineare g è applicata ai componenti della matrice di pooling. C'è un singolo valore di polarizzazione per ogni riga della matrice.

Se ignoriamo temporaneamente il pooling layer si potrebbe indicare come si calcola ogni dimensione-d delle colonne a nella matrice risultante dopo le operazioni precedenti. Definita M come la matrice delle diagonali:

$$M = [diag(m_{:,1}, \cdots, diag(m_{:,m}))] \tag{2.10}$$

dove m sono i pesi dei filtri d della convolution. Dopo un paio di convolutional e non-linear layer, ogni colonna a nella matrice è ottenuta come segue, per qualche indice j:

$$a = g\left(M \begin{bmatrix} w_j \\ \vdots \\ w_{j+m-1} \end{bmatrix} + b\right) \tag{2.11}$$

In questo modo, applicando le operazioni di convolution, dynamic k-max pooling e non-linear function alla matrice sequenza in input, si ottiene la *feature map*. Infine, per terminare la descrizione della DCNN, si applica alla mappa una operazione definita *folding* che non introduce parametri aggiuntivi, ma si occupa di dimezzare la dimensione delle rappresentazioni.

In conclusione, le DCNN hanno ottenuto risultati molto rilevanti, anche oltre il 90% di accuratezza, negli ambiti già citati come la sentiment analysis.

Capitolo 3

Character Level - Convolutional Neural Networks

3.1 Introduzione

Le ConvNets, ovvero le Convolutional Neural Networks [1][2], sono una particolare tipologia di reti neurali particolarmente utili per estrarre informazioni da segnali grezzi in ambiti che possono variare dalle applicazioni per la visione artificiale al riconoscimento vocale e altri. In particolare le Time-Delay Neural Networks (TDNN), utilizzate nelle prime ricerche di Deep Learning, sono essenzialmente ConvNets che modellano dati sequenziali [3][4].

L'applicazione delle ConvNets per la classificazione testuale è un argomento largamente trattato in letteratura, per il quale è stato dimostrato che possono essere applicate per insiemi di parole distribuiti [5][6] o discreti [7], senza conoscenza alcuna della struttura semantica o sintattica del linguaggio. Questo approccio è stato provato essere competitivo nei confronti dei metodi tradizionali[11].

Per questa tesi sono state utilizzate CNN a livello di caratteri per la classificazione di frasi, per eseguire ciò sono stati utilizzati dataset di Amazon contenenti recensioni relative a quattro categorie differenti di prodotti:

Libri[12], Elettronica[13], Vestiario[14] e Film[15].

3.2 Design delle Character-Level CNN

Le character-level CNN hanno un design modulare dove i gradienti sono ottenuti attraverso la back-propagation[8], cioè diminuire il gradiente linearmente o esponenzialmente a ogni epoca di training, per rendere l'ottimizzazione più performante.
Il componente principale è il temporal convolutional module, che semplicemente calcola una convolution 1-D. Supponendo di avere una funzione di input discreta $g(x) \in [1, l] \rightarrow \mathbb{R}$ e una funzione kernel discreta $f(x) \in [1, k] \rightarrow \mathbb{R}$. La convolution $h(x) \in [1, \lfloor (l-k)/d \rfloor + 1] \rightarrow \mathbb{R}$ tra $f(x)$ e $g(x)$ con stride d è definita come

$$h(y) = \sum_{x=1}^{k} f(x) \cdot g(y \cdot d - x + c), \tag{3.1}$$

dove c = k - d + 1 è un offset costante. Come le tradizionali ConvNets, il module è parametrizzato da una serie di funzioni kernel $f_{ij}(x)$ (i = 1,2,...,m and j = 1,2,...,n) che vengono chiamati *weights* (pesi), su una serie di inputs $g_i(x)$ e outputs $h_i(y)$. Ogni inputs (o outputs) g_i (o h_j) vengono chiamati *features* (caratteristiche), per m (o n) input si intendono la dimensione di ogni feature. L'output $h_i(j)$ è ottenuto da una somma su i delle convolution tra $g_i(x)$ e $f_{ij}(x)$.

3.2.1 Modulo Chiave

Un modulo chiave che può aiutare ad addestrare modelli è quello denominato Temporal Max-Pooling, che si tratta della versione 1-D del modulo Max-Pooling usato per la computer vision [9]. Data una funzione di input discreta $g(x) \in [1, l] \rightarrow \mathbb{R}$, la funzione Max-Pooling $h(x) \in [1, \lfloor (l-k)/d \rfloor + 1] \rightarrow \mathbb{R}$ di $g(x)$ è definita come

$$h(y) = \max_{x=1}^{k} g(y \cdot d - x + c), \tag{3.2}$$

dove $c = k$ - d + 1 è un offset costante. Questo modulo Pooling ci permette di addestrare la rete andando oltre sei layers (strati) di profondità, dove tutte le altre reti neurali falliscono.

3.2.2 Layers

Il modello prevede nove layers di profondità di cui sei sono di tipo convolutional mentre le altre tre sono di tipo fully-connected.

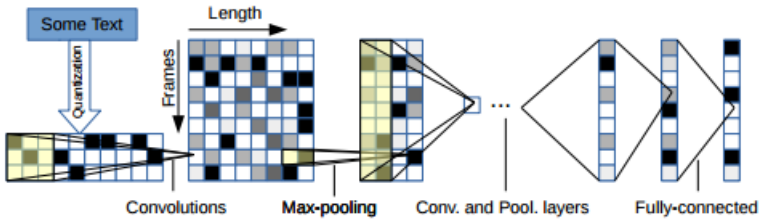

Figura 3.1: Illustrazione del modello

3.3 Ambiti di utilizzo delle ConvNets

Le Character-Level ConvNets sono solamente delle tipologie di ConvNets, le quali sono una categoria di reti neurali risultate particolarmente efficaci in ambiti quali il riconoscimento di immagini oltre che alla classificazione testuale. In particolare hanno avuto successo nell'identificazione dei volti, oggetti e segnali stradali in campi come la visione artificiale e pilota automatico nelle automobili.

3.3.1 Classificazione di Immagini

Le ConvNets per il riconoscimento di immagini si differenziano in modo sensibile da una regolare rete neurale in quanto i layers hanno neuroni sistemati in tre dimensioni: larghezza, altezza e profondità. Ogni layer trasforma

il volume 3-D in input in un volume 3-D di output per l'attivazione dei neuroni.

Una ConvNet è essenzialmente una sequenza di layers[10], ognuno dei quali

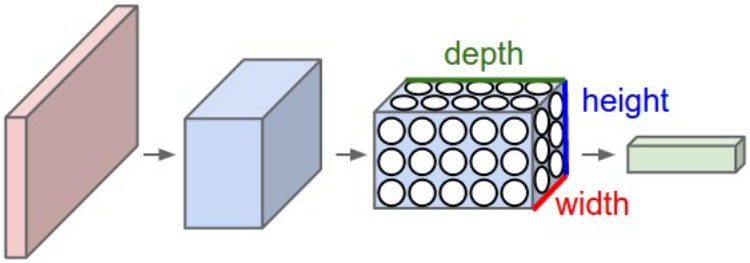

Figura 3.2: CNN per riconoscimento di Immagini

trasforma trasforma un volume di input in un altro attraverso una funzione differenziale, più in particolare, una architettura ConvNet per l'elaborazione di immagini è costituita da tre tipi di layers principali: Convolutional, Pooling e Fully-Connected Layer. Un esempio di questo tipo di architettura può essere una semplice ConvNet per una classificazione CIFAR-10 (un problema classico di Machine Learning che consiste nel classificare immagini RGB 32x32 in 10 categorie) del tipo [INPUT-CONV-RELU-POOL-FC]:

- INPUT[32x32x3] prende i valori grezzi dei pixel nell'immagine delle dimensioni specificate e colorate R,G,B.
- CONV layer che calcola l'output di neuroni che sono connessi a regioni locali nell'input. Il risultato è un volume [32x32x12] se i filtri applicati sono dodici.
- RELU layer che applica una funzione di attivazione elementare, come $max(0, x)$ con soglia a zero. Questo lascia inalterato le dimensioni del volume.
- POOL layer che esegue una operazione di discesa lungo le dimensioni spaziali (larghezza, altezza), il risultato è ad esempio un volume [16x16x12].

- FC layer che calcola la classe di appartenenza dell'immagine, arrivando a un volume di [1x1x10], dove ognuno dei dieci numeri corrisponde a una differente classe e quindi categoria del CIFAR-10.

In questo modo la ConvNet trasforma l'immagine originale, strato per strato, dai pixel originali alle classi finali. Da notare che alcuni layer possiedono parametri in input.
In particolare, i layers CONV/FC eseguono trasformazioni che sono una funzione non solo delle attivazioni nel volume di input, ma anche dei parametri (peso e inclinazione dei neuroni); dall'altra parte i layers RELU/POOL implementano una funzione già prefissata.
I parametri nei layers CONV/FC saranno perciò addestrati con una discesa del gradiente così che la classe risultante sia più vicini alle etichette iniziali associate alle immagini in fase di addestramento.

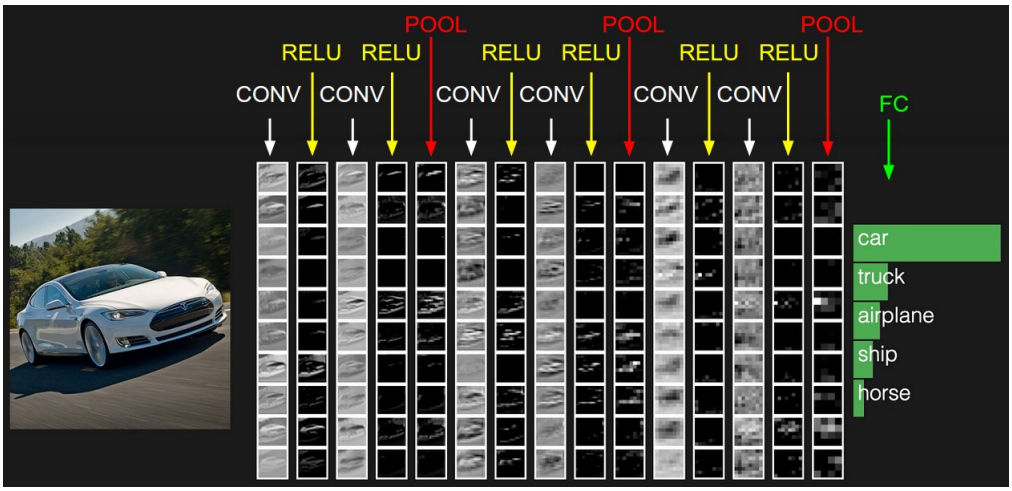

Figura 3.3: Esempio di utilizzo di ConvNet per CIFAR-10

3.3.2 Classificazione Testuale

Come già discusso in precedenza, le ConvNets possono essere utilizzate per la classificazione testuale a livello di carattere oppure anche a livello di frase: ogni frase necessita di essere trasformata in una matrice immagine dove ogni carattere opportunamente codificato è equivalente a un pixel nell'immagine. Anche per la classificazione testuale vengono utilizzati una serie

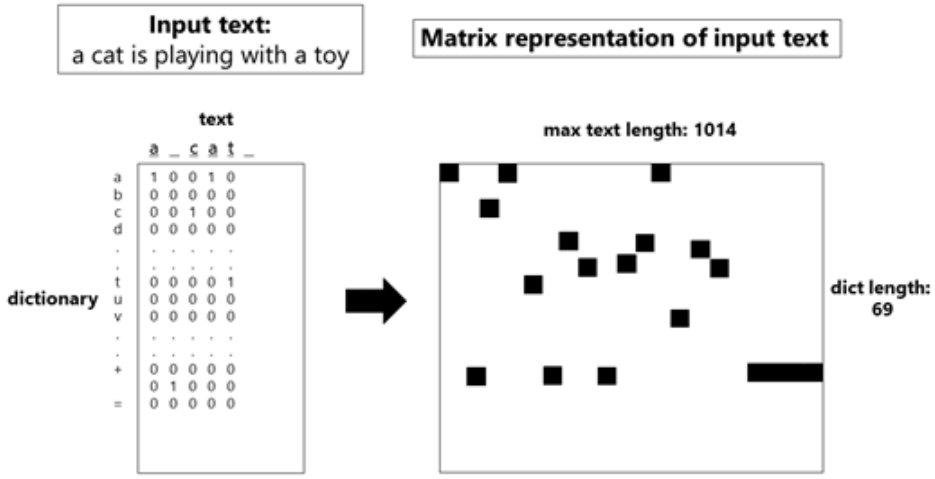

Figura 3.4: Schema della codifica dei caratteri

di layers che, partendo dal testo opportunamente codificato con tecniche di NLP (generalmente per questo tipo di reti neurali si utilizza una codifica One-Hot), producono l'output desiderato e addestrano la rete neurale.

La figura sottostante mostra la serie di layers di cui è composta la rete neurale di cui poi verrà fornita una descrizione più dettagliata.

Embedding Layer

Il primo layer è quello definito Embedding Layer che accetta in input un vettore bidimensionale contenente la sequenza codificata. Generalmente questo layer è utilizzato per diminuire la dimensione del tensore in input, con l'ausilio anche di una operazione di zero-padding, per arrivare a una dimensione fissata. L'output di questo layer può essere trattato come una

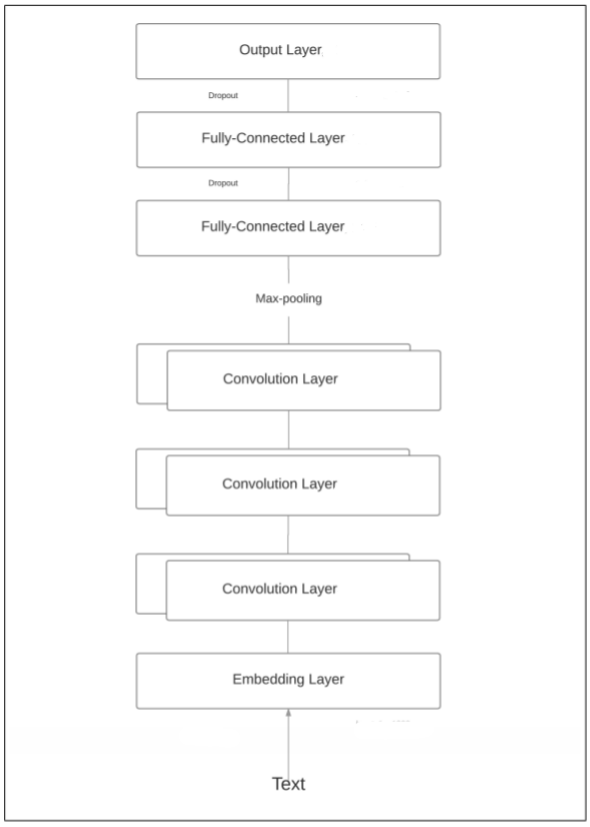

Figura 3.5: Layers di una Char-CNN

immagine bidimensionale, convertendo poi il carattere in input in un vettore mono dimensionale, la rete è in grado di estrarre la feature dal kernel convoluzionale.

Convolutional Layer

Nel modello descritto per il progetto il numero di layer convoluzionali sono sei mentre i successivi sono tre. L'operazione di convoluzione, come quella descritta in precedenza, è largamente utilizzata per il processamento di immagini e di segnali. Per questo tipo di operazione ci sono due segnali utilizzati che sono il vettore del testo e il kernel; alla fine dell'operazione ce ne sarà un terzo che è l'output, che è definita anche convoluzione 1-D. L'output

fornito da questo layer fornisce una rappresentazione gerarchica del testo in input.

A seguire si trova un Max-Pooling Layer che è in grado di selezionare le features più importanti dall'output della convoluzione 1-D.

Fully-Connected Layer

Il Layer definito Fully-Connected è conosciuto anche come Dense Layer. A questo punto vengono combinate tutte le features selezionate dal Max-Pooling Layer. Come menzionato, il Max-Pooling Layer seleziona le feature da ogni convolutional kernel. Il Fully-Connected Layer si occupa della maggior parte dell'assemblaggio e poi costruisce una rappresentazione gerarchica per l'output layer.

Quest'ultimo layer utilizza una funzione di attivazione non lineare chiamata *softmax*, e qui ci sono un numero di neuroni pari al numero delle classi.

Capitolo 4

Codifiche del Linguaggio

4.1 Introduzione

Con il termine NLP, Natural Language Processing, si intendono i differenti processi che vengono applicati ai linguaggi naturali, per renderli codificabili e adatti ad essere utilizzati per reti neurali. Di seguito verranno analizzati gli approcci di maggior successo in ambito di Machine Learning.

4.2 Definizione

Il Natural Language Processing è il processo di trattamento automatico di informazioni, da parte di un sistema informatico, scritte o parlate in lingua naturale.

Questo processo è reso particolarmente difficile e complesso a causa delle caratteristiche intrinseche di ambiguità del linguaggio umano. Per questo motivo il processo di elaborazione viene suddiviso in fasi diverse, tuttavia simili a quelle che si possono incontrare nel processo di elaborazione di un linguaggio di programmazione:

- Analisi Lessicale - scomposizione di parole in token.

- Analisi Grammaticale - associazione delle parti del discorso a ciascuna parola nel testo.

- Analisi Sintattica - arrangiamento dei token in una struttura sintattica (ad albero: parse tree).

- Analisi Semantica - assegnazione di un significato alla struttura sintattica e, di conseguenza, all'espressione linguistica.

4.3 Word Embedding

Con l'espressione Word Embedding si intende un insieme di tecniche di modellazione del linguaggio e di feature learning in NLP che si occupano mappare frasi e parole in vettori di numeri reali. Questa trasformazione è necessaria perché le reti neurali prendono in input solamente vettori di valori continui.

L'aspetto centrale però è che la rappresentazione vettoriale deve preservare la similarità di contesto delle parole; così facendo le parole che normalmente sono affiancate nel testo oppure hanno significato simile sono prossime nello spazio vettoriale creato. Qui di seguito verranno elencate le tecniche più comuni in Machine Learning.

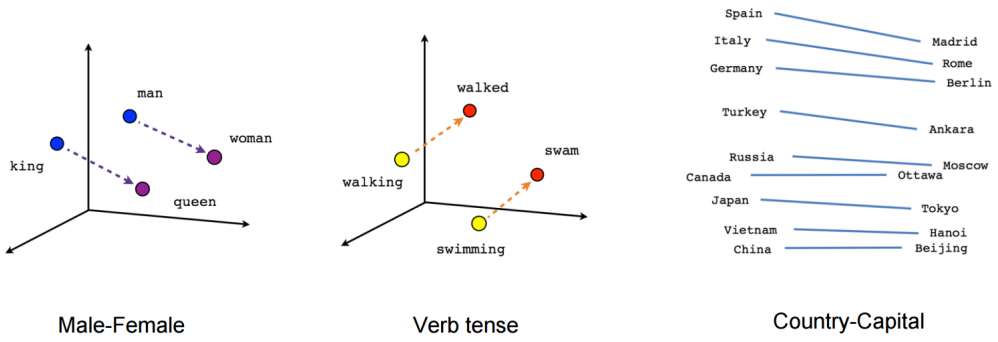

Figura 4.1: Esempio di spazio vettoriale

4.4 Bag of Words

Bag of Words è un modello di rappresentazione delle parole comunemente utilizzato per la classificazione di documenti, la frequenza di ognuno è utilizzata come feature per l'addestramento.

Questo tipo di rappresentazione vede il testo come una sequenza di parole le quali sono indicative del suo contenuto. Si considera un multiset in quanto una parola può comparire più volte; in genere si calcola la frequenza di ogni termine all'interno del testo e da qui viene creato un vettore dove ogni posizione rappresenta una parola e il numero all'interno la sua frequenza.

Spesso però la frequenza non è indicativa dell'importanza di una parola all'interno di un documento (come nel caso degli articoli) per cui è necessario "normalizzare" utilizzando una certa funzione: la più famosa è la tf-idf (term frequency-inverse document frequency) che determina come assegnare valori (pesi) a ciascun termine in ciascun vettore.

- tf - è il fattore locale, che pesa la rilevanza di ciascun termine in ciascuno dei singoli documenti.

- idf - è il fattore globale, che pesa l'importanza del termine nell'intera collezione di documenti.

$$tf.idf(t,d) = \log\left(f(t,d)\right) \cdot \log\left(\frac{|D|}{|d \in D : t \in d|}\right) \tag{4.1}$$

La funzione cresce proporzionalmente maggiore è la frequenza di un termine nel documento ed è inversamente proporzionale alla frequenza del termine nel documento.

In conclusione, Bag-Of-Words viene utilizzata spesso con l'ausilio di altre tecniche (tokenizzazione, lemming, stemming, tf-idf ...) per ottenere rappresentazioni vettoriali delle parole più veritieri possibili.

4.5 Codifica One-Hot

La codifica One-Hot viene utilizzata in algoritmi di Machine Learning per rappresentare le parole in forma vettoriale. La particolarità deriva dal fatto che questi vettori possiedano n elementi di cui tutti posti a valore 0 tranne uno di valore 1; per cui ogni parola è codificata univocamente dalle altre.

Questa codifica è particolarmente utilizzata nei moderni algoritmi di Machine Learning, in particolare in quelli di classificazione e regressione; alcuni, come il Random Forest[32], invece non sono adatti a questa codifica.

4.6 Modello N-Gram

Il modello n-gram per il testo è usato per processare task in NLP e data mining. Essenzialmente il modello si occupa di suddividere il testo in subset di parole definito da una finestra fissata e scorrendola, in genere, avanti una parola per volta (in scenari più avanzati per N parole); per cui la finestra può essere di due parole per volta (bigram), tre parole(trigram) oppure più parole (n-gram). In una frase sono presenti:

$$Ngrams_K = X - (N - 1) \tag{4.2}$$

dove X è il numero delle parole nella frase K.

Il modello N-Gram è un modello probabilistico utilizzato per prevedere l'elemento successivo in una sottosequenza (n-1). A seconda della finestra scelta, gli elementi possono essere fonemi, sillabe, lettere, parole o coppie di parole.

4.7 Codifica Word2vec

Word2vec è un modello predittivo particolarmente computazionalmente efficiente per la rappresentazione vettoriale di testo non elaborato. Word2vec prende in input un testo corposo e produce uno spazio vettoriale, tipicamente di qualche centinaia di dimensioni, dove a ogni parola del testo è assegnato un

corrispondente vettore nello spazio, univoco. I vettori vicini corrispondono a parole contestualmente simili.

Questo modello è stato creato da un team di ricercatori di Google con a capo Tomas Mikolov[33]. Word2vec può utilizzare due modelli di architettura per produrre una rappresentazione vettoriale:

- Continuos Bag-Of-Word (CBOW) - Il modello predice la parola corrente da una finestra di parole di contesto circostanti. L'ordine di queste ultime non influisce sulla predizione.

- Skip-Gram - Il modello utilizza la parola corrente per predire la finestra di parole di contesto circostanti. L'architettura skip-gram pesa le parole di contesto vicine più gravemente di più parole di contesto distanti[34].

4.8 Rappresentazione GloVe

GloVe è un algoritmo di apprendimento non supervisionato per ottenere una rappresentazione vettoriale delle parole. Il Training viene eseguito su statistiche aggregate globali di co-occorrenza di parole da un testo, e le rappresentazioni risultanti mostrano interessanti sottostrutture lineari dello spazio vettoriale[35].

L'obiettivo del training è di apprendere i vettori di parole cosicché il prodotto dei punti sia uguale al logaritmo della probabilità di co-occorrenza delle parole. A causa del fatto che il logaritmo di un rapporto è uguale alla differenza dei logaritmi, questo obiettivo associa i rapporti delle probabilità di co-occorrenza delle parole con le differenze vettoriali nello spazio vettoriale delle parole. Poiché questi rapporti possono codificare una qualche forma di significato, questa informazione viene codificata anche come differenze vettoriali. Per questo motivo, i vettori di parole risultanti funzionano molto bene su compiti di analogia delle parole.

La similarità coseno tra due vettori di parole fornisce un metodo efficiente per misurare la distanza semantica e linguistica tra due parole. Questo

metodo è chiamato Nearest Neighbors e produce un singolo scalare che quantifica la relazione tra due parole. Questa semplicità può essere problematica poiché due parole mostrano quasi sempre relazioni più complesse di quelle che possono essere catturate da un singolo numero.

Per catturare in modo quantitativo la sfumatura necessaria per distinguere, ad esempio, l'uomo dalla donna, è necessario che un modello associ più di un singolo numero alla coppia di parole. Un candidato naturale e semplice per una serie allargata di numeri discriminanti è la differenza vettoriale tra i due vettori di parole. GloVe è progettato in modo che tali differenze vettoriali catturino il più possibile il significato specificato dalla giustapposizione di due parole.

4.9 Position Encoding

Il Position Encoding è una particolare codifica delle parole in relazione alla loro posizione all'interno della frase[36]. Ciò prende la forma: $m_i = \sum_j l_j \cdot Ax_{ij}$, dove . è una moltiplicazione tra elementi e l_j è una colonna vettore con la struttura $l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$ (assumendo una indicizzazione di base 1), dove J è il numero di parole nella frase e d è la dimensione dell'embedding. Questa rappresentazione significa che l'ordine delle parole ora influisce su m_i. La stessa rappresentazione è utilizzata per domande, memoria di input e memoria di output.

Capitolo 5

TensorFlow

5.1 Che cos'è un Framework?

I Framework sono infrastrutture generali che si pongono tra il sistema operativo e il software che permettono a chi programma di semplificare e risparmiare il codice; alla base di un Framework c'è un linguaggio di programmazione e una serie di librerie che mettono a disposizione tutte le diverse funzionalità sotto forma di classi e metodi; implementando le classi astratte quindi si istanzia il relativo Framework. Per quanto riguarda le reti neurali sono diversi i Framework utilizzabili, in questo caso di studio, riguardante le CNN (Convolutional Neural Network), è stato utilizzato TensorFlow.

5.2 Che cos'è TensorFlow?

TensorFlow è una libreria software open source rivolta alla computazione numerica che utilizza grafi di tipo DataFlow. Il progetto è disponibile sulla piattaforma GitHub ed è stato originariamente sviluppato da ricercatori e ingegneri che lavorano nel team Google Brain nell'ambito dell'organizzazione di ricerca di Google Machine Intelligence per effettuare apprendimento automatico e Deep Learning con reti neurali, ma il sistema è abbastanza generale

per essere applicabile in una vasta gamma di altri domini. Il linguaggio di programmazione è Python.

5.3 Funzionamento

Come definito precedentemente, TensorFlow utilizza grafi di tipo DataFlow, dove i nodi del grafo rappresentano operazioni matematiche, mentre gli archi rappresentano array di dati multidimensionali detti tensori che permettono ai nodi di essere collegati: i tensori sono la struttura dati principale di questo tipo di framework. Questo tipo di architettura flessibile permette di sviluppare computazioni su singole o multiple CPUs o GPUs. TensorFlow mette a disposizione diversi tipi di API: quelle di più basso livello (TensorFlow Core) e alcune di più alto livello che sono costruite sulla base di quelle Core, un esempio di API di alto livello è quello delle librerie denominate Estimator che facilitano la configurazione, l'istruzione e la valutazione dei vari modelli.

5.3.1 I Tensori

La struttura dati principale è il tensore, che consiste in un set di valori primitivi inseriti all'interno di un array di qualsiasi dimensione: si definisce il rank di un tensore come il numero delle sue dimensioni (Figura 3.1).

```
3 # a rank 0 tensor; this is a scalar with shape []
[1., 2., 3.] # a rank 1 tensor; this is a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

Figura 5.1: Rappresentazione dei Tensori

L'oggetto di riferimento per il tensore è il tf.Tensor. Questo oggetto rappresenta una computazione parziale che eventualmente può produrre un valore; i programmi in TensorFlow inizialmente costruiscono un grafo partendo da questi oggetti, specificando come ogni tensore viene calcolato in base agli

altri tensori disponibili e quindi eseguendo parti di questo grafo per ottenere i risultati desiderati.

Proprietà dei tensori:

- tipo di dato (ad esempio int32, float32, string)
- forma (numero degli elementi in ogni singola dimensione)

Ogni elemento di un tensore ha lo stesso tipo di dato che è sempre conosciuto, mentre per quanto riguarda la forma, essa può essere solo parzialmente conosciuta. Se determinati input sono di forme completamente conosciute allora i tensori prodotti dalle diverse operazioni hanno forme note, in casi particolari è possibile trovare la forma dei tensori solamente al momento dell'esecuzione del programma.
Sono presenti tipologie di tensori particolari, i principali sono:

- tf.Variable
- tf.Constant
- tf.PlaceHolder
- tf.SparseTensor

Fatta eccezione per tf.Variable, i valori dei tensori, relativamente a una singola esecuzione, sono immutabili e quindi possiedono un singolo valore, tuttavia se si valuta lo stesso tensore più volte, esso può assumere valori differenti.
Come definito precedentemente, il rank di un tensore corrisponde al numero delle sue dimensioni, ogni rank però corrisponde a una diversa entità matematica.

```
#Rank 0 --> Scalare
variableRank0 = tf.Variable("Elephant", tf.string)

#Rank 1 --> Vettore
```

```
variableRank1 = tf.Variable([2, 3, 5, 7, 11], tf.int32)

#Rank 2 --> Matrice
variableRank2 = tf.Variable([[7],[11]], tf.int16)

#Rank 3 --> 3-Tensor
variableRank3 = tf.Variable([[4], [9], [16], [25]], tf.int32)

#Rank n --> n-Tensor
variableRankN = tf.zeros([10, 299, 299, 3]) # batch x height x
    width x color

#Ottenere il Rank di un Tensore
r = tf.rank(variableRank2)
```

Per quanto riguarda la forma di un tensore, TensorFlow la intuisce automaticamente durante la costruzione del grafo se i Ranks sono conosciuti. Le forme sono rappresentate in Python attraverso liste/tuple di interi oppure con l'oggetto tf.TensorShape.

```
#Ottenere la forma di un Tensore
shape=tf.shape(my_matrix)[1]

#Cambiare la forma di un Tensore
rank_three_tensor = tf.ones([3, 4, 5])
matrix = tf.reshape(rank_three_tensor, [6, 10])
```

Una volta costruito il grafo è possibile valutare un determinato Tensore, cioè scoprire il valore assegnato ad esso: la funzione eval() è il modo più semplice per effettuare questa operazione, tuttavia, nel caso in cui il valore del Tensore sia assegnato in modo dinamico (come per la tipologia di Tensore tf.PlaceHolder), è necessario fornire prima un valore.

```
#Valutazione di un Tensore di tipo tf.Constant
```

```
constant = tf.constant([1, 2, 3])
tensor = constant * constant
print tensor.eval()

#Valutazione di un Tensore di tipo tf.PlaceHolder
p = tf.placeholder(tf.float32)
t = p + 1.0
t.eval() #Il Tensore non ha un valore, quindi la valutazione
    fallisce
t.eval(feed_dict={p:2.0})

#Stampa di un Tensore
tensor=tf.Variable("Elephant", tf.string)
print tensor
tensor=tf.Print(tensor, [tensor])
```

5.3.2 TensorFlow Core

Attraverso l'importazione di Tensorflow all'interno del proprio programma viene messo a disposizione di Python l'accesso a tutte le classi, metodi e simboli delle API denominate Tensorflow Core.

```
#Importazione di Tensorflow
import tensorflow as tf
```

Un Grafo Computazionale è la struttura principale alla quale fanno riferimento i programmi che sfruttano Tensorflow come framework, esso rappresenta, attraverso i tensori, la rete neurale a cui si fa riferimento; perciò nei programmi che utilizzano Tensorflow Core devono essere presenti due sezioni discrete e distinte:

1. Costruzione del Grafo Computazionale
2. Esecuzione del Grafo Computazionale

5.3.3 Costruzione del Grafo Computazionale

Per costruire un Grafo è necessario definire nodi e tensori nei modi già citati precedentemente, attraverso i quali poi è possibile eseguire varie operazioni che portano alla sua costruzione. Generalmente il modello deve poter prendere input arbitrari e per fare in modo che sia addestrabile deve essere possibile modificare il Grafo per avere nuovi output dagli stessi input. La struttura dati tf.Variable permette di aggiungere al grafo parametri addestrabili.

```
#Creo due variabili e un placeholder e ne definisco il tipo
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)

#Creazione di un modello lineare
linear_model = W * x + b
```

Per inizializzare tensori e variabili bisogna creare un oggetto Session, che incapsula lo stato e il controllo di Tensorflow runtime, e al suo interno eseguire il Grafo.

```
#Creazione dell'oggetto Session
sess = tf.Session()
#Inizializzazione delle variabili
init = tf.global_variables_initializer()
#Esecuzione della sessione
sess.run(init)
```

In questo modo si crea un modello ma per determinare la sua efficienza è necessario valutarlo su un training set e utilizzare anche una funzione di loss, la quale misura quanto sia distante il modello dai dati forniti.

```
#Placeholder in cui andranno inseriti i dati di training
y = tf.placeholder(tf.float32)
```

```
#Quadrato della distanza tra il valore nel modello e i dati di
    training
squared_deltas = tf.square(linear_model - y)
#Funzione classica di loss
loss = tf.reduce_sum(squared_deltas)
```

5.3.4 Esecuzione del Grafo Computazionale

Tensorflow mette a disposizione degli Optimizers che si occupano di minimizzare la funzione di loss cambiando gradualmente ogni variabile, ecco un elenco:

- tf.train.Optimizer
- tf.train.GradientDescentOptimizer
- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

Per quanto riguarda invece le differenti funzioni di loss è possibile evidenziare le più utilizzate:

- tf.reduceSum

- tf.contrib.losses.meanSquaredError
- tf.contrib.losses.softmaxCrossEntropy
- tf.contrib.losses.sparseSoftmaxCrossEntropy

Ecco come viene realizzato un ciclo di training

```
# Dati di training
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# Ciclo di training
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(1000):
  sess.run(train, {x: x_train, y: y_train})
# Valutazione dell'accuratezza
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y:
    y_train})
```

5.4 TensorBoard

TensorBoard è lo strumento di Tensorflow attraverso il quale è possibile visualizzare il grafo creato, tracciare la sua esecuzione e mostrare dati aggiuntivi. Per rendere il proprio programma visualizzabile da TensorBoard è necessario serializzare i dati: per fare ciò bisogna annotare quelli che si desiderano visualizzare attraverso i metodi messi a disposizione dalla classe tf.summary e inoltre marcare le variabili interessate attraverso il metodo tf.nameScope. In questo modo il programma salva all'interno di una cartella tutti i dati salvati nel momento dell'esecuzione che sono poi utilizzabili da TensorBoard una volta eseguito.

```
#Comando per eseguire TensorBoard indicando l'apposita cartella
    dove sono stati salvati i dati durante l'esecuzione
tensorboard --logdir=path/to/log-directory
```

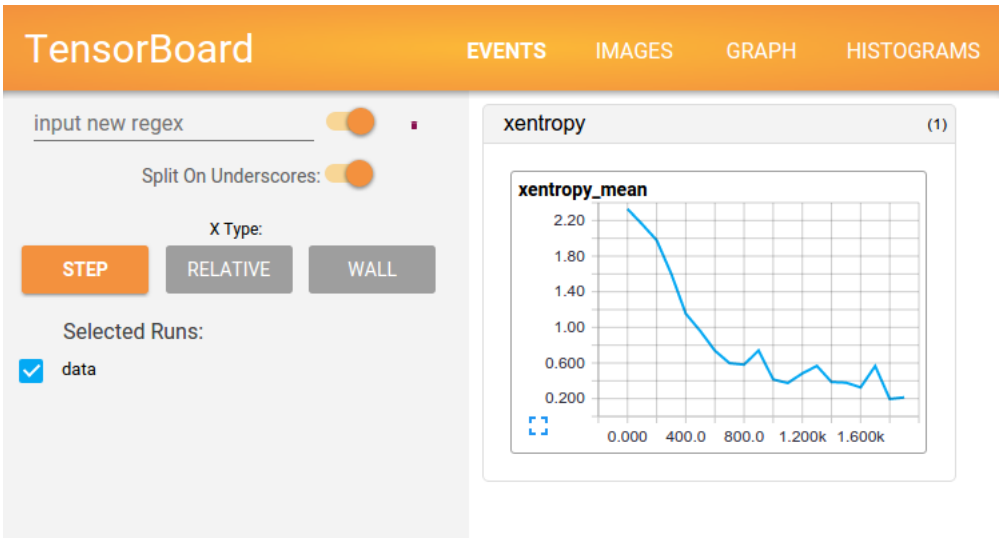

Figura 5.2: Esempio di schermata di TensorBoard

Capitolo 6

Schema Progettuale

6.1 Introduzione

Conoscere l'opinione e il grado di soddisfazione di un cliente o di un determinato gruppo di persone è un fattore sempre più importante e fondamentale al giorno d'oggi. Il problema, nel corso degli anni, si è spostato prima dal riuscire a reperire il maggior numero di dati poi classificarli e etichettarli, data la loro sovrabbondanza. Infatti, grazie a social networks e siti web sempre più incentrati sull'interattività con le persone, la quantità di dati disponibili e reperibili si è moltiplicata nel corso degli anni, ma la maggior parte di queste informazioni sono non etichettate, cioè non possiedono una valutazione numerica, quindi direttamente interpretabile da un calcolatore, perciò il costo per etichettarle in termini di tempo e persone è molto oneroso.

La Sentiment Analisys e la Sentiment Classification sono due discipline introdotte per sopperire ai costi di etichettatura dei dati: esse si occupano di estrapolare sistematicamente la polarità di opinioni fornite da un gruppo di persone, in particolare con una applicazione di tipo *Cross-Domain*, in cui la conoscenza estrapolata su un dominio di partenza (recensioni di Libri, Elettronica ecc ...), viene applicata a un dominio differente da quello iniziale.

La differenza *In/Cross-Domain* sta proprio nel differenziare il dominio su cui si desidera fare Sentiment Classification da quello da cui si è estrapolata la

conoscenza. Infatti il Cross-Domain è il vero vantaggio di questa disciplina, poiché se si possiedono un certo numero di recensioni etichettate in un certo dominio, il vantaggio non è tanto testarle sul dominio stesso, ma estrapolare da esso la conoscenza e utilizzarla su recensioni non ancora etichettate di un dominio diverso, anche se possiede un linguaggio più eterogeneo.

Perciò l'obiettivo di questa tesi è quello di applicare tecniche di Transfer Learning per eseguire compiti di Sentiment Classification In e Cross Domain, in particolare valutare come si comporta una specifica rete neurale (Character-Level Convolutional Neural Network), facendo particolare attenzione alle codifiche per la rappresentazione testuale utilizzate (Word2vec e One-Hot).

6.2 Prerequisiti

Per capire al meglio le procedure utilizzate in questa tesi è utile avere una conoscenza di base di alcuni argomenti:

1. Conoscenza del Deep e Transfer Learning su scenari In/Cross-Domain. (Cap. 1)

2. Conoscenza dell'architettura e del funzionamento delle reti neurali in generale. (Cap. 2)

3. Conoscenza della rete neurale presa in esame, Character-Level CNN. (Cap. 3)

4. Conoscenza delle principali tecniche di elaborazione e rappresentazione vettoriale del testo. (Cap. 4)

5. Conoscenza del Framework Tensorflow utilizzato per gli esperimenti. (Cap. 5)

6. Conoscenza approfondita del linguaggio di programmazione Python.

6.3 Caso di Studio

Il caso di studio preso in esame in questa tesi utilizza una serie di datasets che contengono recensioni di prodotti commerciali. I Test sono stati eseguiti su una serie di quattro domini principali, che fanno parte di una collezione di recensioni di Amazon messe a disposizione dall'università di Stanford[38].

- Libri (L)
- Elettronica (E)
- Film e Serie TV (M)
- Abbigliamento, Scarpe e Gioielleria (C)

Ogni recensione possiede numerosi campi che la identificano, ma quelli interessati per il caso di studio sono essenzialmente due:

- reviewText - Campo contente la recensione vera e propria in formato testuale.
- overall - Campo contenente una valutazione, in formato numerico (da 1 a 5), in riferimento alla recensione testuale.

Una polarità negativa viene associata alle recensioni con *overall* pari a 1 o 2, mentre polarità positiva a quelle con *overall* pari a 4 o 5; le recensioni pari a 3 vengono ignorate in quanto considerate difficili da classificare.

6.4 Organizzazione degli Esperimenti

L'obiettivo degli esperimenti è quello di massimizzare l'accuratezza della funzione di predizione relativa al dominio target usando esempi che provengono da domini differenti dallo stesso. Questi datasets contengono dati reali, i quali contengono recensioni principalmente sbilanciate verso una polarità positiva (circa 80%), per cui, per gli esperimenti, è necessario bilanciare le

recensioni positive e negative (50%) sia per i datasets di training sia per quelli di testing.

Per dimostrare l'efficienza della rete neurale in esame, essa viene addestrata e testata con datasets di tre diverse dimensioni (2.000-20.000-200.000 unità) con un rapporto tra recensioni per il training, validation e test equivalente a 80%-10%-10%.

I parametri di configurazione della rete sono stati ottenuti attraverso sperimentazioni in modo tale da migliorare l'efficienza, l'accuratezza e i tempi di addestramento.

Si possono definire due differenti tipologie di testing già citate in precedenza:

- In-Domain - Dominio sorgente e dominio target sono gli stessi utilizzati per addestramento, validazione e testing.

- Cross-Domain - Dominio sorgente per l'addestramento e validazione è differente dal dominio target per testing.

La prima serie di test viene effettuata In-Domain, per cui l'algoritmo viene testato su un unico dominio ed è possibile capire quanto la dimensione del dataset influisca sull'accuratezza degli esperimenti.

La seconda serie di test viene effettuata Cross-Domain, in cui l'algoritmo utilizza ciò che ha imparato durante l'addestramento per un testing su un dominio differente, ciò risalta la capacità di generalizzazione dell'algoritmo stesso.

La rete neurale utilizzata per gli esperimenti è una Character-Level Convolutional Neural Network (già descritta nel Cap. 3), tipologia di rete molto utilizzata in altri ambiti, ma ancora poco esplorata per nell'ambito della classificazione testuale, i cui risultati poi verranno confrontati con altre tipologie di rete neurali.

6.4.1 Manipolazione dei Datasets

Nel Capitolo 4, è stato già definito che il testo presente nelle recensioni deve essere prima processato per essere poi utilizzato come input della rete e sono state mostrate le principali tecniche di NLP. Per questo progetto è stato scelta una codifica di tipo One-Hot, preferita a una codifica di tipo Word2Vec a seguito di varie sperimentazioni, più predisposta e adatta come input delle Char-CNN.

Come già definito precedentemente, è opportuno suddividere i datasets secondo la suddivisione predefinita rispettando il rapporto 80%-10%-10%. I processi di training e validazione si svolgono su un numero di 25 epoche (parametro modificabile). Per epoca si intende un processo di addestramento sullo stesso dataset in cui spesso i dati vengono rimescolati. Per addestramento si intende il processo tramite il quale viene aggiornata la rete la quale cerca di conformare i propri pesi e la propria funzione di predizione dell'output all'output desiderato. La validazione serve a monitorare l'addestramento e a testarlo su elementi al di fuori del dataset. Il testing è invece il processo tramite il quale si verifica l'accuratezza dell'addestramento tramite un dataset(anche dello stesso dominio) che non è stato però utilizzato per l'addestramento, in questo modo viene simulato una possibile applicazione in un contesto reale.

6.4.2 Configurazione degli Iperparametri

La configurazione dei parametri di una rete neurale è un punto chiave, in quanto sono determinanti per ottenere buoni risultati. L'addestramento richiede una serie di iperparametri:

- Learning Rate - Valore che definisce la velocità di apprendimento della rete. Utilizzando la backpropagation si parte da un learning rate prefissato fino a farlo convergere a un valore finale, diminuendolo a ogni epoca di training. Mantenere un learning rate troppo alto non permetterebbe all'addestramento di convergere e quindi oscillerebbe solamente

attorno a un minimo locale, al contrario un valore troppo basso può portare a tempi di convergenza troppo lunghi, per questi motivi si utilizza la backpropagation. Per gli esperimenti i valori tra cui oscilla il learning rate sono $1e-3(0.001)$ e $1e-4(0.0001)$.

- Dropout - Valore che indica la percentuale di neuroni che la rete azzererà, quindi non verranno aggiornati, in modo casuale a ogni epoca. Attraverso ciò la rete viene regolarizzata e così i neuroni imparano in modo indipendente gli uni dagli altri. A seguito di varie sperimentazioni, il valore di dropout è impostato a 0.7.

- Epoche - Numero di epoche utilizzate per l'addestramento, cioè numero di volte in cui viene ripetuto il processo di addestramento per migliorare l'accuratezza. All'inizio di ogni epoca gli elementi dei datasets vengono rimescolati e al termine di ogni epoca viene effettuata la fase di testing. Il numero di epoche utilizzate per l'esperimento è 25.

- Classi - Valore che rappresenta il numero delle classi su cui si basa la classificazione. In questo caso 2 (identifica la polarità positiva o negativa) oppure 5 (identifica i valori da 1 a 5 di ogni valutazione delle recensioni).

- Istanze - Numero delle istanze utilizzate per il training e per il testing.

- Batch - Valore che identifica la dimensione del batch che influisce molto nell'addestramento: infatti un valore di batch piccolo permette alla rete di aggiornarsi più di frequente,ma con tempi maggiori di addestramento, viceversa con un valore più grande la rete si aggiorna con meno frequenza ma con tempi più brevi. Tale valore è importante ed è stato ottenuto empiricamente ed equivale a 128 in questi esperimenti.

6.4.3 Suddivisione dei Dataset

Inizialmente, come ricordato in precedenza, i dataset vengono manipolati creando gruppi di istanze che mantengano il rapporto training-validation-

test 80%-10%-10% le cui istanze vengono prelevate in modo random; per l'addestramento a 2 classi il numero di istanze all'interno di ogni gruppo devono essere suddivise equamente non tenendo conto delle valutazioni pari a 3 perché ambigue.

6.4.4 Creazione della Rete Neurale

Successivamente alla suddivisione dei datasets, viene creata la Char-CNN descritta nel Capitolo 4. Questo processo di costruzione viene effettuato tramite le funzionalità messe a disposizione dal framework Tensorflow e inizializzata attraverso gli iperparametri.

6.4.5 Compilazione del Modello

A seguito della creazione del modello viene scelta la funzione di ottimizzazione adatta. Il modello viene per cui compilato, con una funzione di *loss* (che esprime l'errore della predizione) e con un optimizer che si occupa di ottimizzare tale funzione, quindi minimizzare l'errore. Tensorflow mette a disposizione sia la funzione loss *tf.nn.softmax_cross_entropy_with_logits* e sia l'optimizer *tf.train.AdamOptimizer*[39]. L'optimizer è stato selezionato da una numerosa lista, che è visitabile al Capitolo 5, i quale è particolarmente adatto a datasets di grandi dimensioni e che permette una computazione efficiente.

6.4.6 Addestramento della Rete

A seguito della fase di compilazione viene effettuato il training della rete durante il quale vengono stampati i valori dell'accuratezza relativi a ogni iterazione e il numero di ogni epoca. Al termine di ogni epoca viene stampata la media ponderata di accuratezza e loss.

6.4.7 Testing della Rete

Il modello addestrato viene poi testato al termine di ogni epoca di training e i risultati riguardanti la media ponderata e i valori migliori di testing di accuratezza vengono stampati.

6.5 Implementazione della Rete Neurale

L'implementazione della Char-CNN è basata sul progetto GitHub di Mohammed Jabreel del 2016[40], ed è stato creato attraverso Tensorflow e in codice Python. Il lavoro di sviluppo si è concentrato principalmente nel modificare le classi in cui viene processato l'input e per adattarle a ricevere le recensioni del caso di studio in esame.

6.5.1 Struttura del Progetto

Il progetto è organizzato su 4 file:

- CharCnn.py - File che contiene l'implementazione della rete neurale in esame.
- Configuration.py - File di configurazione attraverso il quale è possibile modificare gli iperparametri.
- ReviewReader.py - File dove sono presenti le funzioni per la gestione e manipolazione dei datasets.
- TrainingSentimentClassification.py - File dal quale vengono lanciati gli esperimenti che contiene tutta la logica dell'addestramento e del testing.

6.5.2 Librerie

Qui di seguito vengono elencate le librerie che sono incluse nel progetto e che sono necessarie per il suo funzionamento:

- Tensorflow, Framework utilizzato per l'implementazione del progetto.
- Numpy[37] - Libreria estensione di Python che aggiunge funzionalità per la manipolazione di vettori e matrici multidimensionali.
- Os - Modulo per manipolare processi e navigare nei percorsi dei file.
- Sys - Libreria che fornisce l'accesso a variabili che interagiscono direttamente con l'interprete Python.
- Datetime - Libreria per manipolazione di date.
- Time - Libreria che espone le funzioni della libreria in C per manipolare tempo e date.
- Json - Libreria per manipolare file in formato Json.
- Random - Libreria che genera numeri pseudo-casuali.
- Gensim - Libreria per la modellazione ed elaborazione del linguaggio naturale.
- Math - Libreria per utilizzare le funzioni matematiche.

Capitolo 7

Esperimenti

7.1 Introduzione

In questo capitolo verranno presentati i risultati ottenuti negli esperimenti di tipo In-Domain e Cross-Domain, sia con classificazione binaria (quindi con polarità positiva o negativa), sia con classificazione a 5 classi, detta Fine-Graned. Gli esperimenti sono stati condotti utilizzando una rete neurale di tipologia Character-Level Convolutional Neural Network, descritta nel Capitolo 3 e implementata nel Capitolo 6; la codifica utilizzata per la rappresentazione vettoriale del testo e quella One-Hot.

I risultati ottenuti verranno poi confrontati con altre tipologie di reti neurali, LSTM e DMN+ che hanno già effettuato sperimentazione su datasets Amazon.

In un primo momento verrà analizzato il comportamento della rete in relazione a test In-Domain su recensioni Amazon e di seguito poi verranno analizzati i risultati Cross-Domain, quindi in un classico scenario di adattamento del dominio.

7.2 Risultati In-Domain

Qui di seguito verranno mostrati i risultati delle sperimentazioni In-Domain con polarità binaria o Fine-Graned. per ognuno dei datasets verrà mostrata l'accuratezza di testing e infine una media tra tutti.

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	Fine-Graned	Binary	Fine-Graned	Binary	Fine-Graned	Binary
	In-Domain					
M → M	24.58	63.33	20.25	68.50	52.00	63.84
E → E	35.59	64.00	35.20	66.00	40.25	72.00
C → C	25.80	64.33	42.65	69.00	22.56	64.00
B → B	33.23	71.33	28.80	68.50	35.48	72.33
Average	**29.80**	**65.74**	**31.73**	**67.50**	**37.25**	**67.75**

Tabella 7.1: Risultati che fanno riferimento a esperimenti con reti neurali Char-CNN In-Domain su datasets contenenti recensioni Amazon.

7.2.1 Char-CNN Classificazione Binaria

Osservando i risultati nella tabella soprastante si può notare come la grandezza dei datasets non influisca eccessivamente nell'accuratezza che si stima attorno al 70%.

7.2.2 Char-CNN Classificazione Fine-Graned

Analizzando i risultati di un approccio Fine-Graned si può notare un prevedibile calo delle prestazioni che comunque non rimangono ottime, anche qui non si riscontra una differenza sostanziale al variare della dimensione delle istanze nei datasets.

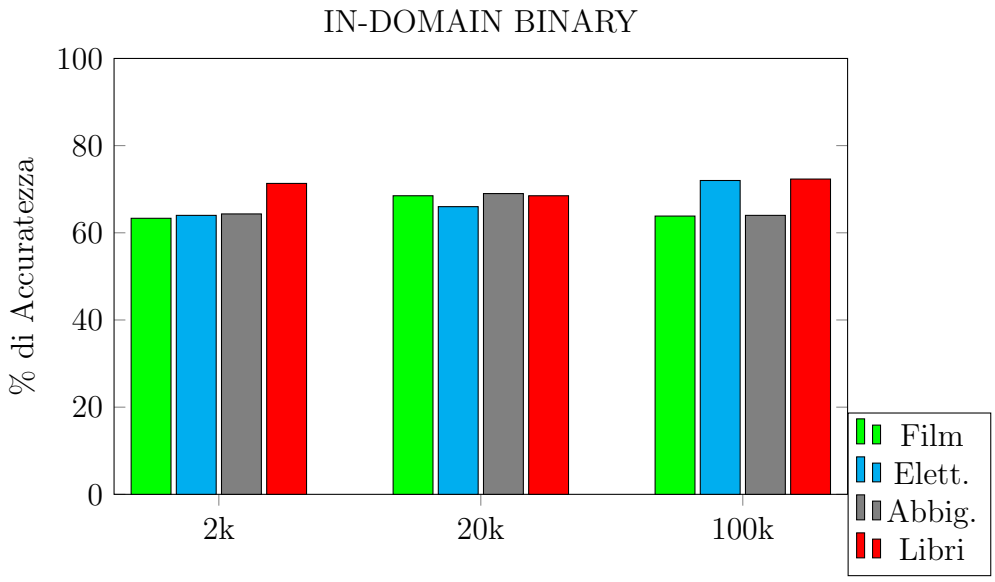

Figura 7.1: Grafico che mostra l'accuratezza per ogni categoria con esperimenti In-Domain Binary.

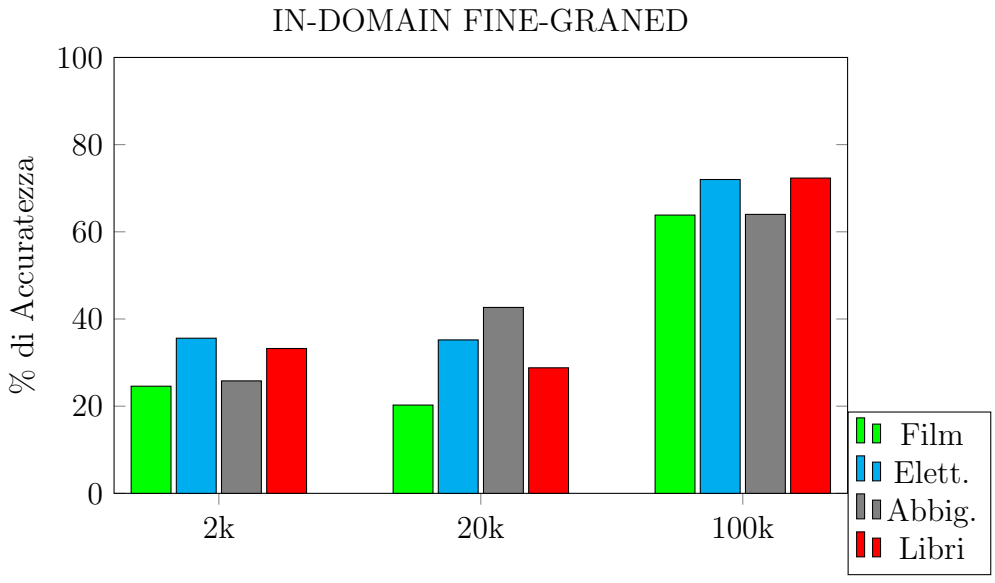

Figura 7.2: Grafico che mostra l'accuratezza per ogni categoria con esperimenti In-Domain Fine-Graned.

7.3 Risultati Cross-Domain

Qui di seguito verranno mostrati i risultati della seconda tipologia di esperimenti che riguardano il Cross-Domain, quindi il dominio su cui è addestrata la rete è differente da quello su cui è testata, sia con polarità binaria o Fine-Graned. per ognuno dei datasets verrà mostrata l'accuratezza di testing e infine una media tra tutti.

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	Fine-Graned	Binary	Fine-Graned	Binary	Fine-Graned	Binary
	In-Domain					
M → E	33.58	67.55	31.26	76.50	42.11	68.11
M → B	25.01	60.24	36.89	73.12	40.29	70.00
M → C	25.59	72.78	29.20	69.89	39.65	67.56
E → B	25.74	66.07	37.20	65.28	33.69	67.23
E → C	29.96	71.25	35.19	71.09	39.40	66.89
E → M	28.13	62.34	34.78	68.00	40.05	68.75
C → M	22.99	70.33	29.96	69.00	23.65	66.25
C → E	24.55	63.78	43.65	70.47	25.13	72.79
C → B	24.35	69.25	23.01	70.93	22.96	67.32
B → E	25.23	71.78	28.80	63.19	29.32	63.22
B → M	28.49	76.21	39.11	65.36	26.50	68.05
B → C	28.65	72.10	30.55	63.45	30.84	64.66
Average	**26.33**	**68.25**	**32.83**	**68.50**	**32.25**	**67.16**

Tabella 7.2: Risultati che fanno riferimento a esperimenti con reti neurali Char-CNN Cross-Domain su datasets contenenti recensioni Amazon. Il dominio a sinistra della freccia è quello sorgente e quello a destra è quello target

7.3.1 Char-CNN Classificazione Binaria

Si può osservare come il cambiamento di dominio in fase di testing non abbia diminuito, in alcuni casi anche migliorato, le prestazioni che rimangono con una accuratezza attorno al 70%.

7.3.2 Char-CNN Classificazione Fine-Graned

Anche qui, come per i test In-Domain, vi è un calo notevole delle prestazioni quando si passa ad una classificazione a più classi. I grafici sottostanti evidenziano le minime differenze di prestazione tra esperimenti Binary, mentre invece per i Fine-Graned c'è più differenza al variare del numero di istanze di training.

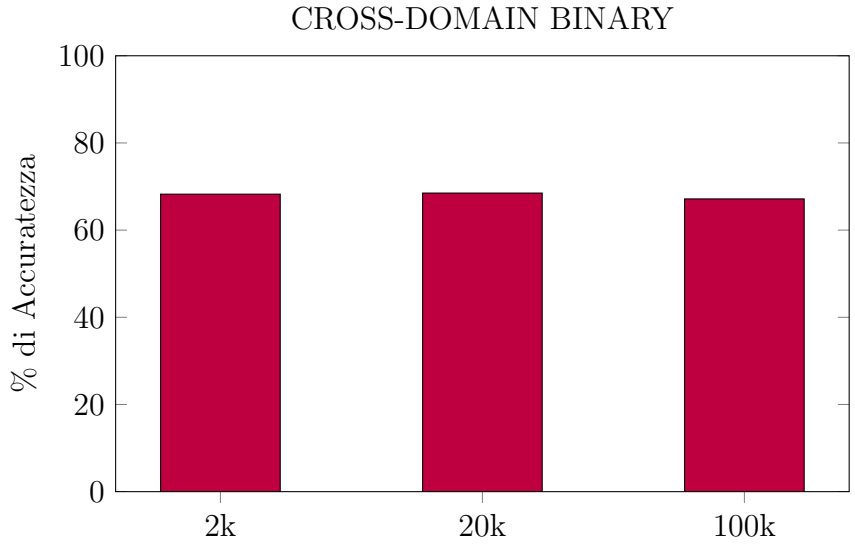

Figura 7.3: Grafico che mostra l'accuratezza per ogni categoria con esperimenti Cross-Domain Binary.

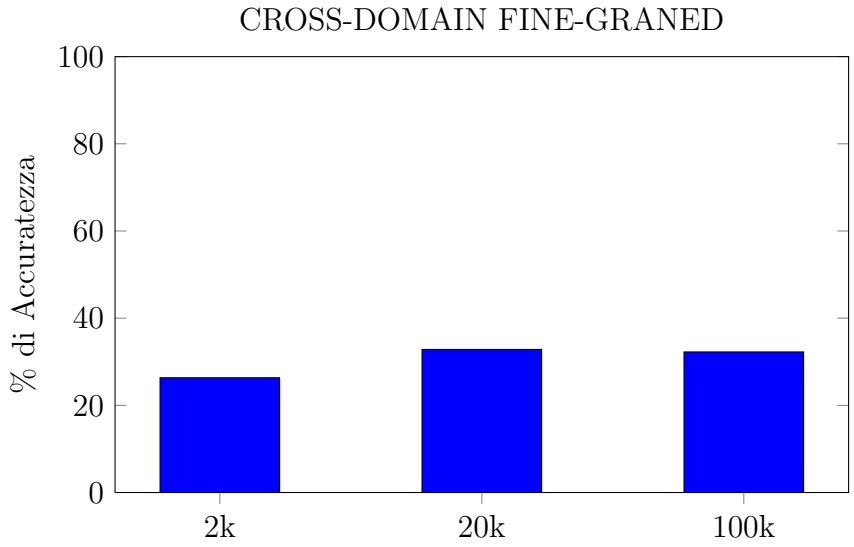

Figura 7.4: Grafico che mostra l'accuratezza per ogni categoria con esperimenti Cross-Domain Fine-Graned.)

7.4 Stanford Sentiment Treebank

Lo Stanford Sentiment Treebank è un noto dataset fornito dall'università di Stanford (Socher Et Al,2013)[46], in cui sono presenti frasi etichettate da 0 a 1, da cui si ottiene una classificazione a 5 classi, e una suddivisione delle stesse in datasets di training-validation-testing già preimpostata.

La sperimentazione eseguita su questo dataset è stata per cui eseguita con la Char-CNN con una classificazione Fine-Graned e codifica One-Hot come gli esperimenti eseguiti in precedenza.

	Test Accuracy (%)
Domain	Fine-Graned
	Stanford Sentiment Treebank
SST	**26.33**

Tabella 7.3: Risultato di una media ponderata di tre diverse sperimentazioni eseguite sul SST.

Per avere una maggiore sicurezza sull'accuratezza dei dati forniti, la sperimentazione è stata eseguita tre volte sullo stesso dataset. Dai risultati si può evincere che non ci sia molta differenza tra lo Stanford Sentiment Treebank e dai datasets di Amazon con una classificazione Fine-Graned, la cui accuratezza si attesta vicina al 30%.

7.5 Confronto con altre Reti Neurali

In questa sezione verranno confrontati i risultati ottenuti con reti neurali differenti dalle Char-CNN, DMN+ e LSTM, analizzate e studiate da altri due studenti, rispettivamente Nicola Piscaglia e Federico Pucci; gli esperimenti sono sempre effettuati sui dataset di Amazon per esperimenti In e Cross Domain con classificazione binaria.

7.5.1 Risultati In-Domain

La tabella sottostante mostra i risultati ottenuti con esperimenti In-Domain sempre sugli stessi dataset di Amazon: come si può facilmente notare la differenza in termini di accuratezza tra Char-CNN e DMN+ e LSTM è abbastanza marcata, in particolare con le reti DMN+ che sono risultate essere le più performanti anche su datasets di piccole dimensioni.

	Test Accuracy (%)								
	2k			20k			100k		
Domain(s)	DMN+	LSTM	C-CNN	DMN+	LSTM	C-CNN	DMN+	LSTM	C-CNN
	In-Domain								
M → M	90.75	76.75	63.33	87.85	78.92	68.50	88.65	79.50	63.84
E → E	89.50	80.75	64.00	91.35	82.50	66.00	92.36	85.24	72.00
C → C	86.00	82.25	64.33	90.35	85.00	69.00	90.04	84.40	64.00
Average	**88.75**	**79.92**	**63.88**	**89.85**	**82.14**	**67.16**	**90.35**	**83.05**	**66.61**

Tabella 7.4: Risultati di esperimenti In-Domain che confrontano tre reti neurali differenti, DMN+, LSTM e Char-CNN.

7.5.2 Risultati Cross-Domain

Una differenza di risultati ancor più marcata si può evidenziare anche per gli esperimenti Cross-Domain dove le reti DMN+ ottengono risultati molto superiori rispetto alle Char-CNN e alla LSTM sopratutto su datasets di piccole e medie dimensioni. In datasets di grandi dimensioni invece l'accuratezza delle Char-CNN è inferiore del 27%, differenza che non è così marcata con le LSTM con dimensioni inferiori di datasets.

	Test Accuracy (%)								
	2k			20k			100k		
Domain(s)	DMN+	LSTM	C-CNN	DMN+	LSTM	C-CNN	DMN+	LSTM	C-CNN
	Cross-Domain								
M → E	97.00	76.20	67.55	91.20	80.29	76.50	97.50	94.29	68.11
M → C	89.75	75.45	72.78	92.40	81.33	69.89	95.00	95.84	67.56
E → M	90.00	76.84	62.34	95.00	78.20	68.00	87.50	95.82	68.75
E → C	95.00	81.12	71.25	92.00	84.85	71.09	95.00	96.65	66.89
C → M	97.00	70.78	70.33	96.00	74.80	69.50	95.00	89.90	66.25
C → E	93.00	80.30	63.78	93.00	84.04	70.47	97.50	94.53	72.79
Average	**93.63**	**76.78**	**68.01**	**94.00**	**80.54**	**70.91**	**94.58**	**94.51**	**68.39**

Tabella 7.5: Risultati di esperimenti Cross-Domain che confrontano tre reti neurali differenti, DMN+, LSTM e Char-CNN.

7.6 Analisi dei Tempi

In questa ultima sezione verranno analizzati i tempi impiegati per l'addestramento delle reti neurali con l'obiettivo di avere una visione d'insieme esaustiva e completa.

L'addestramento della rete neurale Char-CNN è avvenuto principalmente su una GPU Nvidia Titan X con 32gb di RAM e qualche test è stato provato anche su un computer Quad Core con 8gb di RAM. I risultati verranno poi confrontati con le altre tipologie di reti neurali che hanno utilizzato tecnologie analoghe.

I valori elencati nella tabella sottostante fanno riferimento ad una approssimazione nell'ordine dei 30 minuti, in quanto un medesimo test sullo stesso dataset potrebbe variare al massimo di qualche minuto, per cui si predilige evidenziare le differenze in termini di tempi di calcolo, al variare delle dimensioni dei datasets.

	Training Time (Hours)				
Domain Size	C-CNN(CPU)	C-CNN(GPU)	LSTM(CPU)	DMN+(CPU)	DMN+(GPU)
2k	0.4	0.3	0.5	1.07	0.65
20k	5.2	4.3	3	4.19	2.01
100k	23.5	22.1	16	229.5	22.95

Tabella 7.6: Confronto dei tempi di calcolo su tre diverse dimensioni dei datasets e su tre reti neurali differenti.

Questa tabella evidenzia come, nel caso delle Char-CNN, non ci sia una differenza marcata tra l'utilizzo di CPU e GPU in termini di tempi di calcolo, per quanto riguarda il confronto con le altre due reti in esame, le Char-CNN possiedono tempi di calcolo più veloci su di piccole dimensioni ma sono leggermente più lente su medie e grandi dimensioni dove le LSTM si sono rivelati le più veloci. Il risultato ottenuto su dataset di grandi dimensioni con le DMN+ utilizzate su una CPU è solo stimato a partire dal risultato ottenuto sulla GPU moltiplicato per 10.

Capitolo 8

Conclusioni e sviluppi futuri

In questa tesi sono stati presentati una serie di esperimenti basati sull'utilizzo di una rete neurale chiamata Character-Level Convolutional Neural Network, che si attesta essere una particolare tipologia delle più classiche Convolutional Neural Networks, addestrata con l'ausilio di datasets con recensioni di prodotti Amazon con differenti classificazioni e sia In-Domain che Cross-Domain.

La codifica del testo utilizzata è stata quella One-Hot in quanto è risultata essere più efficiente rispetto a quella Word2Vec testata empiricamente. I risultati In-Domain mostrano come questa particolare tipologia di rete non risenta troppo del variare di dimensioni del dataset utilizzato per l'addestramento, infatti l'accuratezza risultante si attesta attorno al 70% per tutte e tre le casistiche con una classificazione binaria; con una classificazione di tipo Fine-Graned si evidenzia come questa rete neurale non sia molto efficiente in quanto l'accuratezza è attorno al 35%. L'utilizzo di un differente dataset come lo Stanford Sentiment Treebank ha portato risultati simili agli esperimenti Fine-Graned in quanto anch'esso classificato su 5 classi.

Negli esperimenti Cross-Domain c'è stato un leggero miglioramento delle prestazioni, sia con classificazione binaria e sia Fine-Graned, anche se non sostanziali.

Gli esperimenti hanno evidenziato come questa tipologia di rete sia più

adatta a una classificazione di tipo binario, anche se il confronto con le reti DMN+ e LSTM sia un abbastanza impari in quanto le prestazioni raggiunte da loro sono in certi casi superiori del 20%.

Le Char-CNNs sono reti neurali ancora poco esplorate in letteratura, per cui, in futuro, si consiglia di tentare approcci anche con differenti codifiche oltre a quella One-Hot o Word2Vec per provare a migliorare le prestazioni anche con classificazione Fine-Graned che ancora è carente e cercare di migliorare anche la classificazione binaria in quanto anche un altro esperimento[47] ha attestato una accuratezza simile.

Bibliografia

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation, Winter 1989.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, November 1998.

[3] L. Bottou, F. Fogelman Soulie, P. Blanchet, and J. Lienard. Experiments with time delay networks and dynamic time warping for speaker independent isolated digit recognition. In Proceedings of EuroSpeech 89, volume 2, pages 537–540, Paris, France, 1989.

[4] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. Acoustics, Speech and Signal Processing, IEEE Transactions on, 1989.

[5] C. dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 69–78, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[6] Y. Kim. Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natu-

ral Language Processing (EMNLP), pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[7] R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. CoRR, abs/1412.1058, 2014.

[8] D. Rumelhart, G. Hintont, and R. Williams. Learning representations by back-propagating errors. Nature, 1986.

[9] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce. Learning mid-level features for recognition. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pages 2559–2566. IEEE, 2010.

[10] Convolutional Neural Network Architectures: from LeNet to ResNet - Lana Lazebnik

[11] "Character-level Convolutional Networks for Text Classification", Xiang Zhang, Junbo Zhao, Yann LeCun, 4 Aprile 2016

[12] Recensioni Amazon Libri

[13] Recensioni Amazon Elettronica

[14] Recensioni Amazon Vestiario

[15] Recensioni Amazon Film

[16] "Understanding Convolutional Neural Networks for NLP", Denny Britz, Novembre 2015

[17] Deep Learning by Adam Gibson, Josh Patterson

[18] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85–117

[19] "What Is The Difference Between Artificial Intelligence And Machine Learning?", Bernard Marr, Forbes, 6 Dicembre 2016

[20] "What Is The Difference Between Deep Learning, Machine Learning and AI?", Bernard Marr, Forbes, 6 Dicembre 2016

[21] "An Introduction to Deep Learning", Algoritmia Web site, Charlie Crawford, November 2016

[22] West, Jeremy; Ventura, Dan; Warnick, Sean (2007). "Spring Research Presentation: A Theoretical Foundation for Inductive Transfer". Brigham Young University, College of Physical and Mathematical Sciences. Archived from the original on 2007-08-01. Retrieved 2007-08-05

[23] Pan SJ, Yang Q. A survey on transfer learning. IEEE Trans Knowl Data Eng. 2010;22(10):1345–59.

[24] "NanoNets : How to use Deep Learning when you have Limited Data", 30 Gennaio 2016, Sarthak Jain

[25] "A Logical Calculus of the Ideas immanent in nervous activity", McCulloch and Pittis, 1943

[26] " The Organization of Behavior", Donald Hebb, 1949

[27] "Long-Short Term Memory", Sepp Hochreiter, Jurgen Schmidhuber 1997

[28] "Neural Turing Machines", Alex Graves, Greg Wayne, Ivo Danihelka, 10 Dicembre 2014

[29] "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing", Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, Richard Socher, 2016

[30] "A Convolutional Neural Network for Modelling Sentences", Nal Kalchbrenner Edward Grefenstette, Phil Blunsom, Department of Computer Science, University of Oxford, 8 Aprile 2014

[31] “Hybrid computing using a neural network with dynamic external memory”, Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia,Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, Demis Hassabis

[32] “How the Random Forest Algorithm works in Machine Learning”, May 22, 2017, Saimadhu Polamuri

[33] “Efficient Estimation of Word Representations in Vector Space”, Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, 7 Settembre 2013

[34] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S.; Dean, Jeff (2013). Distributed representations of words and phrases and their compositionality.

[35] “GloVe: Global Vectors for Word Representation”,Jeffrey Pennington, Richard Socher, Christopher D. Manning, 2014

[36] “End-To-End Memory Networks”, Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus, 24 Novembre 2015

[37] Libreria Numpy

[38] Datasets Amazon dall’università di Stanford

[39] “Adam: A Method for Stochastic Optimization”, Diederik P. Kingma, Jimmy Ba, 22 Dicembre 2014

[40] Implementazione di una Char-CNN

[41] Eclipse IDE

[42] PyDev - Eclipse Plug-In

[43] Python

[44] Installazione Tensorflow

[45] TensorBoard

[46] "Parsing With Compositional Vector", Richard Socher and Alex Perelygin and Jean Wu and Jason Chuang and Christopher Manning and Andrew Ng and Christopher Potts

[47] "Character-Level Text Classification: CNN", Puya Sharif, 17 Ott 2017

Ringraziamenti

Desidero innanzitutto ringraziare il prof. Gianluca Moro e i suoi collaboratori Andrea Pagliarani e Roberto Pasolini per la professionalità, attenzione, disponibilità e pazienza che mi hanno dimostrato durante tutto il percorso di tesi.

Desidero poi ringraziare tutta la mia famiglia, dai miei genitori a mia sorella, ai miei nonni e ai miei zii, che ogni giorno mi ha sopportato e supportato.

Desidero ringraziare la mia fidanzata Federica che più di tutti ha saputo capirmi e sopportarmi.

Per ultimi ma non meno importanti ringrazio tutti i miei amici, quelli storici, quelli che hanno condiviso con me il percorso dell'università e quelli che ogni giorno condividono con me la passione per la pallacanestro.

Appendice A

Guida agli Esperimenti

In questa appendice verrà riportata la guida per poter eseguire gli esperimenti in completa autonomia, è necessario seguire passo passo il procedimento di installazione dei package, di download dei datasets e di avviamento degli esperimenti. Per poter avviare gli esperimenti si può utilizzare una semplice riga di comando da qualsiasi sistema operativo, oppure si possono utilizzare IDE come Eclipse[41] o Visual Studio con l'ausilio di opportuni plug-in (nel caso di Eclipse si tratta di PyDev)[42].

A.1 Installazioni di base

Per poter eseguire gli esperimenti è necessario installare alcuni componenti di base:

- Python 3.x - Interprete per il codice degli esperimenti installabile da riga di comando o dal sito web[43].

- Tensorflow - Framework per la rete neurale, installabile da riga di comando o dal sito web[44]. Nel Capitolo 5 e sul sito sono presenti guide esaustive per testare la corretta installazione.

A.2 Installazione delle librerie

Nel Capitolo 6, sottosezione 6.5.2, sono elencate una serie di librerie necessarie per il corretto funzionamento del sistema, per cui è necessario assicurarsi che esse siano installate all'interno del proprio computer prima di poter avviare gli esperimenti; altrimenti, dopo aver installato Python, versione 3 o maggiore, si può inserire il comando sotto riportato per procedere con le installazioni.

```
pip3 <nome libreria>
```

A.3 Download dei Datasets

Effettuare il download dei datasets di Amazon disponibili online[38] e salvare i file in un percorso predefinito.

A.4 Configurazione dei parametri

Recarsi nella directory in cui è stato inserito il progetto ed aprire con un editor di testo o all'interno di un IDE il file *Configuration.py*:

1. Per prima cosa modificare le variabili *data_train* e *data_test* inserendo il percorso relativo ai file precedentemente scaricati che si intendono utilizzare rispettivamente per training e testing (in un sistema operativo Windows separare i nomi delle directories con un doppio backslash).

2. la variabile *stanford* è booleana e se impostata a True permette di eseguire esperimenti sullo Stanford Sentiment Treebank il cui percorso deve essere specificato nell'apposita variabile *data_stanford*, ricordando che la variabile *no_of_classes* deve essere impostata al valore 5, se invece impostata a False permette di eseguire le altre tipologie di esperimenti.

3. La variabile *domain* indica la tipologia di esperimento da effettuare e può assumere due differenti valori: *'In'* per esperimenti In-Domain per cui il dataset in esame è solamente quello nella variabile *data_train*, *'Cross'* per esperimenti Cross-Domain in cui vengono usati entrambi i datasets.

4. La variabile *no_of_classes* indica il numero di classi da utilizzare per la classificazione e può assumere due valori: 2 per indicare una classificazione binaria e 5 per una classificazione Fine-Grained o con SST.

5. Le variabili *train_number* e *test_number* possono essere modificate per inserire il numero di istanze per il training e per il testing a piacimento.

6. Infine sono presenti una serie di valori già preimpostati, come Learning Rate, Dropout e numero di Epoche, che possono essere modificati ma già ottimizzati tramite sperimentazione. Una spiegazione di questi valori è fornita nel Capitolo 6.

A.5 Lanciare Addestramento e Testing

Una volta modificato il file di configurazione e inserita la logica desiderata è possibile lanciare gli esperimenti con il comando sottostante preselezionandosi sulla directory in cui è presente il progetto.

```
python3 TrainingSentimentClassification.py
```

L'output prodotto evidenzia l'epoca corrente, l'iterazione e la media ponderata dell'accuratezza; alla fine di ogni epoca di training viene effettuato il testing e viene mostrato in output. Al termine della sperimentazione l'output evidenzia il tempo trascorso, la media di train e test e i valori migliori, al termine di ogni epoca, di train e test.

A.6 Visualizzare la Rete Neurale su TensorBoard

Durante l'addestramento viene creata una directory denominata *runs* in cui vengono salvati gli esperimenti lanciati; è possibile visualizzare questi esperimenti da *localhost* attraverso lo strumento messo a disposizione da Tensorflow, cioè TensorBoard. Per prima cosa è necessario verificare la corretta installazione di TensorBoard da riga di comando seguendo la guida dal sito ufficiale[45], poi bisogna eseguire il seguente comando inserendo tra i doppi apici la directory dove si trova la sotto cartella *summaries* che si riferisce all'esperimento lanciato.

```
tensorboard --logdir="percorso/a/directory/runs/summaries"
```

In seguito non chiudere il processo lanciato da riga di comando e aprire un browser e digitare il codice sottostante nella barra degli indirizzi

```
localhost:6006
```

per visualizzare uno schema grafico dell'esperimento.