Alma Mater Studiorum · Università di Bologna

Campus di Cesena
Scuola di Scienze
CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

DIFFERENTIABLE NEURAL COMPUTER: STUDIO E SPERIMENTAZIONE NELLA CROSS-DOMAIN SENTIMENT CLASSIFICATION

Tesi in: Programmazione di Applicazioni Data Intensive

Relatore:
Prof. GIANLUCA MORO
Correlatori:
DOTT. ING. ANDREA PAGLIARANI
DOTT. ING. ROBERTO PASOLINI

Presentata da:
DIEGO PERGOLINI

ANNO ACCADEMICO 2016-2017
II SESSIONE

ABSTRACT

Keywords: Sentiment Analysis, Machine Learning, Deep Learning, In-Domain, Transfer Learning, Big Data, Language Heterogeneity,Cross-Domain, Differentiable Neural Computer, DNC.

La straordinaria diffusione di piattaforme online attraverso le quali le persone possono esprimere opinioni,come blog, social network,forum, e la conseguente enormità di testo non strutturato generato, ha fatto emergere sempre più interesse verso la *Sentiment Classification*. Essa, proponendosi di carpire la connotazione, positiva o negativa, di un testo, può assumere un'evidente rilevanza commerciale. Il problema delle soluzioni che sono state proposte in questo ambito è che molto spesso si tratta di tecniche che agiscono con efficacia solo in-domain , cioè addestrando e testando un modello di classificazione sullo stesso dominio. I casi reali,però, sono costituiti da scenari *cross-domain*, proprio perché spesso non sono disponibili dati categorizzati per eseguire un addestramento sul dominio target. Questo lavoro si propone di utilizzare un approccio moderno come quello rappresentato dalle DNC, un nuovo tipo di reti neurali dotate di una memoria esterna, unito ad una codifica semantica del testo come i word embeddings, allo scopo di svolgere al meglio classificazioni cross-domain, sia a 2 che a 5 classi. I risultati ottenuti dimostrano la bontà di questo approccio, visto il raggiungimento di ottime accuratezze nell'ambito cross-domain e il superamento di alcuni dei risultati ottenuti in letteratura rappresentanti lo stato dell'arte. Le DNC hanno mostrato di essere in grado capaci di generalizzare, dimostrandosi un modello di classificazione efficace anche se testato su domini diversi da quello di addestramento e facendo segnare un ottimo grado di accuratezza sia applicando tecniche di transfer learning esplicite che non.

INTRODUZIONE

Le opinioni guidano il nostro vivere quotidiano,le nostre relazioni,la nostra società. Le opinioni assumono qualsiasi forma, ma al giorno d'oggi vi è un mezzo prevalente per manifestarle, e questo mezzo è rappresentato proprio dalla tecnologia. Grazie ad essa abbiamo un supporto pressoché infinito per esprimere opinioni, Facebook,Twitter,Trip Advisor, IMDB,Yelp sono solo alcuni della mole di piattaforme che ci consentono di commentare qualsiasi cosa.
Ma questa immensa mole di dati non strutturati, può essere utilizzata in qualche modo? Non sarebbe utile ad un azienda, ad esempio, cogliere il gradimento che sta ottenendo un determinato prodotto? Beh, ovviamente la risposta è si, e tutti i più grandi gruppi commerciali si stanno muovendo per poter disporre di mezzi adeguati a questo compito, non è, infatti,di certo concepibile affrontare l'infinita quantità di testi a disposizione con un approccio manuale. A questo scopo, è nata la Sentiment Analysis, una branca di ricerca interna alla sterminata area del Natural Language Processing,essa può essere vista come il trattamento computazionale di opinioni e sentimenti in un testo libero.
Ma con quale tecnica affrontare questo problema? Nel corso degli anni ne sono state proposte molte, inizialmente nell'ambito del text mining, ma il contributo maggiore, recentemente, è stato fornito dal Deep Learning. Purtroppo però, molti degli approcci proposti si sono rivelati efficaci solo nel caso di classificazioni in-domain, cioè addestrando ed applicando il modello creato su documenti appartenenti allo stesso dominio. Per coloro che non conoscono il Deep Learning, è importate precisare che queste tecniche hanno bisogno di molti esempi per funzionare, esempi che nella stragrande maggioranza dei casi reali, non sono disponibili. Infatti non tutte le fonti da cui possiamo carpire opinioni sono state concepite a questo scopo, rendendo, perciò, di fatto inutili approcci funzionali solo in-domain.
Proprio per superare queste limitazioni, nascono le soluzioni cross-domain, le quali si pongono l'obbiettivo di creare un modello a partire da dati categorizzati (ad esempio recensioni Amazon,in cui insieme al testo è presente uno score) di un certo dominio per poi utilizzarlo in domini diversi. Per assolvere questo compito bisogna però disporre di una tecnologia adeguata, che riesca a cogliere quali sono le caratteristiche contraddistintive di un opinione positiva o negativa,per esempio.
Abbiamo detto poco fa che le reti neurali hanno dato un incisivo contributo alla risoluzione di questa classi di problemi, ma quale tipologia di rete è abbastanza complessa da cogliere le innumerevoli sfumature del linguaggio umano?
Si dice spesso che i ricordi siano il primo passo verso la coscienza, come

si può, altrimenti, imparare dai propri sbagli, se non li si ricordano? Trovo questa riflessione particolarmente adatta ad introdurre la tipologia di rete neurale che è stata utilizzata in questa tesi per svolgere il task preposto, essa è detta Differentiable Neural Computer (DNC), ed è una innovativa rete neurale dotata di una memoria ed avanzati meccanismi di interazione con essa, di ispirazione biologica ed informatica. Le DNC,presentate nello scorso ottobre e di cui il codice è stato rilasciato ad aprile 2017, hanno da subito catturato l'interesse dell'intero panorama scientifico, grazie ad i loro avanzati meccanismi, ispirati sia al funzionamento delle Turing Machine, sia al cervello umano ed alla loro presunta capacità di astrarre algoritmi.
Questa tesi si propone quindi di utilizzare nell'ambito della sentiment classification sia 2 che a 5 classi, in domain ma sopratutto cross-domain, le DNC, al fine di verificare la loro efficacia in un task complesso come quello presentato. Verranno inoltre utilizzati vari dataset di benchmark al fine di confrontare i nostri risultati con quelli ottenuti in letteratura e capire se questo approccio può essere qualitativamente migliore.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

ACRONIMI

DNC Differentiable Neural Computer

SST Stanfor Sentiment Treebank

MC Markov Chain

NB Naive Bayes

PV Paragraph Vector

NTM Neural Turing Machine

RNN Recurrent Neural Network

CNN Convolutional Neural Network

LSTM Long Short Term Memory

W2V Word to Vector

CBOW Continuous Bag of Word

NLP Natural Language Processing

1

LE RETI NEURALI

Obiettivo di questo capitolo è di presentare l'attuale stato dell'arte riguardo le reti neurali ed intelligenza artificiale, ciò verrà fatto introducendo per prima cosa le fonti di ispirazioni biologiche dietro al loro sviluppo. Verranno poi passati in rassegna gli step che hanno portato ai più moderni modelli di Deep Neural Network,partendo dal neurone artificiale di McCulloch and Pitts [23], per arrivare alle dirompenti LSTM [17] e terminare con le DNC (Differentiable Neural Computers), l'oggetto di studio di questa testi. Verranno inoltre enucleati i vari concetti cardine del funzionamento di queste tecnologie, come discesa del gradiente,funzioni di costo e tuning degli iperparametri.

1.1 INTRODUZIONE

Con l'avvento dei computer, l'uomo ha potuto disporre di uno straordinario strumento di calcolo ed un incredibile supporto di memorizzazione, motivo per cui, nei task in cui queste qualità sono necessarie e preminenti, il computer sovrasta le capacità umane. Basti pensare a compiti quali l'ordinamento di un elenco in ordine alfabetico,il calcolo di medie su grandi quantità di recensioni e tante altre mansioni di cui l'affidamento ai computer ha cambiato la nostra società,per sempre. In generale, quindi, la macchina svolge perfettamente quei compiti facilmente descrivibili tramite una procedura logico/matematica, ciò che invece è più difficile per un calcolatore,è quello che a noi riesce invece con facilità e naturalezza. Riconoscere una persona,un oggetto o un animale in mezzo a tanti altri/e, capire sentimenti quali ironia, disprezzo, amore o compiacimento è praticamente impossibile utilizzando i normali paradigmi di programmazione. Proprio per colmare questo gap che divide l'intelligenza umana e i modelli computazionali classici, ci si è ispirati al nostro cervello ed al suo funzionamento biologico, dapprima imitando la logica dei neuroni fino ad arrivare ai giorni nostri in cui vengono imitatà le modalità di funzionamento di intere aree cerebrali, come nel caso delle DNC che riproducono(suppur con le dovute ed ovvie differenze) il funzionamento dell'ippocampo.

1.2 L'EVOLUZIONE DELLE RETI NEURALI

1.2.1 *Il neurone biologico*

Per apprezzare al meglio l'idea alla base del primo neurone artificiale è necessario conoscere il funzionamento di quello biologico, saranno così lampanti i parallelismi fra le due strutture
Definizione:

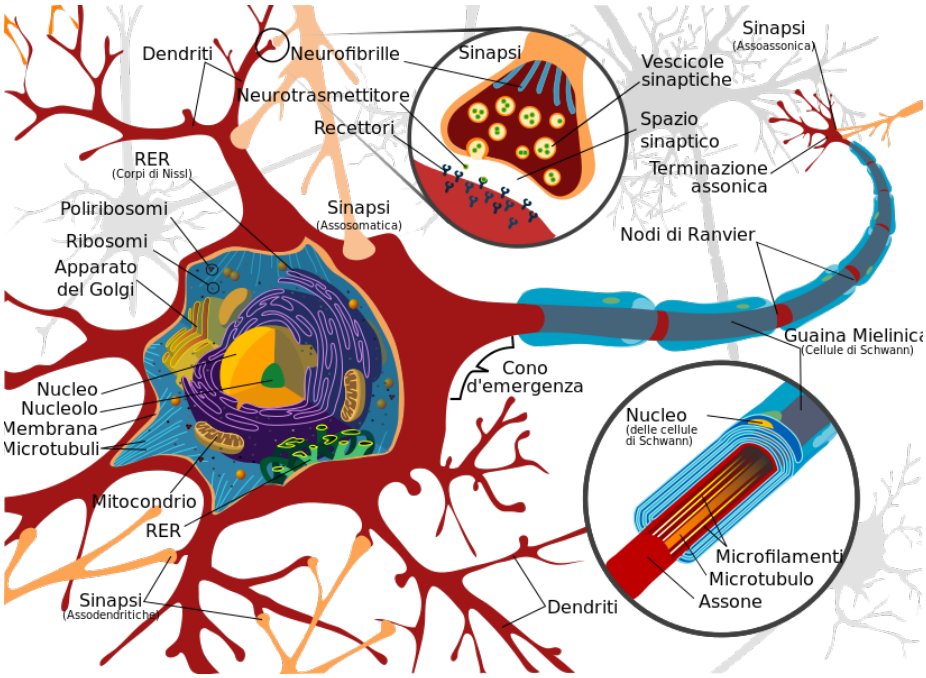


Figura 1.: Diagramma completo di una cellula neuronale. Da [34]

" *Il Neurone è un'unità funzionale del sistema nervoso, cellula altamente specializzata per ricevere, elaborare e trasmettere le informazioni ad altri n. o a cellule effettrici (per es., muscolari o ghiandolari) attraverso segnali elettrici e chimici* " [1]
Esso è composto da 4 principali componenti:
Il corpo cellulare: detto anche soma,è il centro metabolico del neurone,contiene il nucleo e l'apparato biosintetico per la produzione dei costituenti di membrana, degli enzimi e di altre sostanze chimiche di cui la cellula necessita per svolgere le sue mansioni.
I dendriti: sono fibre minori che si ramificano a partire dal corpo cellulare del neurone che tendono,tal volta,a suddividersi più volte formando intorno al corpo cellulare dando vita una struttura molto ramificata.Le ramificazioni dendritiche rappresentano le strutture di ricezione dei segnali in arrivo da altre cellule nervose che comunicano i loro messaggi in zone specializzate.
L'assone: è la fibra principale che parte dal soma,ha origine nel cono di emergenza e si allontana da esso per portare ad altri neuroni (non per forza adiacenti) l'output.

Le sinapsi: sono strutture altamente specializzate che consentono la comunicazione delle cellule del tessuto nervoso sia tra loro sia con altre cellule (muscolari, sensoriali o ghiandole endocrine).

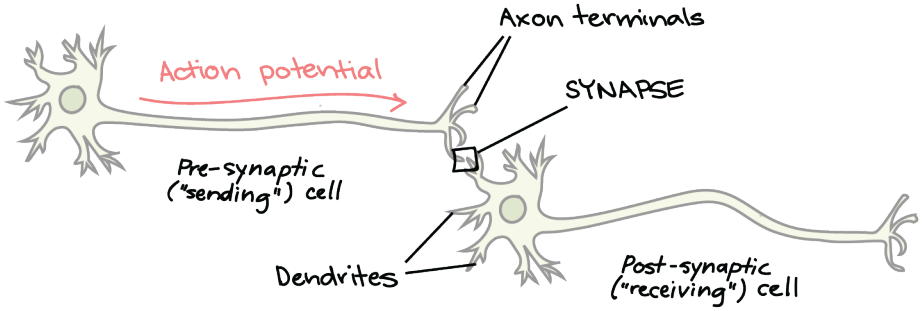


Figura 2.: Schema di una sinapsi. Da [18]

Nella comunicazione fra neuroni hanno un ruolo centrale gli ioni,essi infatti, con il loro ingresso nelle sinapsi dei dendriti, determinano la formazione di una differenza di potenziale tra il corpo del neurone e l'esterno. Quando questo potenziale supera una certa soglia si produce un' impulso: il neurone propaga un breve segnale elettrico detto potenziale d'azione lungo il proprio assone: questo potenziale determina il rilascio di ioni dalle sinapsi dell'assone. La variabilità dell'efficacia di questo meccanismo di collegamento tra neuroni sta alla base del processo di apprendimento e di memoria,prendendo il nome di plasticità sinaptica,certi percorsi tra neuroni saranno,a seconda dell'efficacia dei collegamenti, più o meno favoriti.

1.2.2 *Il Neurone Artificiale di Pitts-McCulloch*

Il primo modello di neurone artificiale è stato quello proposto da McCulloch e Pitts [23] nel 1943,esso era composto da elementi a soglia, connessi tra loro da connessioni ad efficacia sinaptica variabile (i pesi).

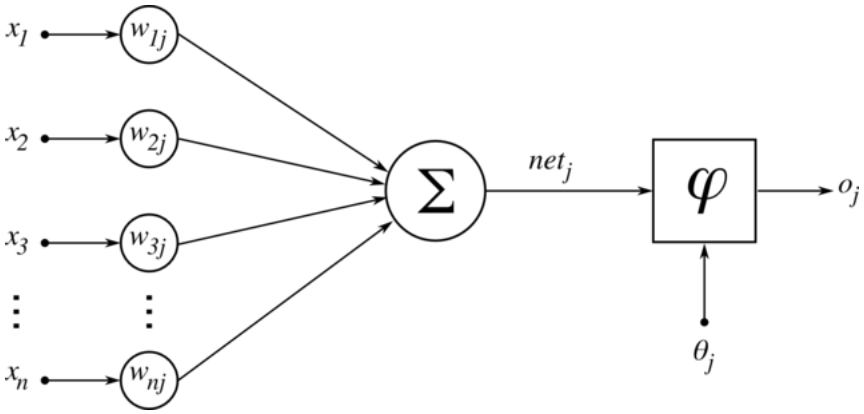


Figura 3.: Schema del neurone artificiale di Pitt. Da [18]

La sua logica di funzionamento prende ispirazione dal neurone biologico, infatti gli input $x_1, x_2, ..., x_n$ modellano gli ingressi legati ad assoni di neuroni differenti, i pesi $w_{1j}, w_{2j}, ..., w_{nj}$ simboleggiano il comportamento delle sinapsi ed, a seconda che siano positivi o negativi, l'azione eccitatoria o inibitoria della comuniciazione. La funzione di soglia φ invece imita il comportamento del corpo cellulare, che produce uno stimolo solo quando viene superata una certa soglia di eccitazione. Matematicamente gli output di un neurone artificiale sono quindi descritti come:

$$o_j = \theta(\sum_{j=1}^{n} w_j x_j - T)$$

E' evidente da questa formula che il neurone produrrà output solo se la sommatoria degli input pesati supererà il valore soglia T,un numero opportuno di tali elementi, connessi in modo da formare una rete, è in grado di calcolare semplici funzioni booleane. Il limite maggiore di questo modello è il fatto che pesi e valori di soglia sono determinabili solo a priori,non esistendo quindi un meccanismo adattivo per svolgere le mansioni desiderate.

1.2.3 *Il Percettrone di Rosenblatt*

Un evoluzione del modello di Pitt,è quello proposto nel 1958 dallo psicologo americano Frank Rosenblatt [30].Il percettrone è composto un ingresso, un'uscita e una regola di apprendimento basata sulla minimizzazione dell'errore (error back-propagation). Questa intuizione è alla base di tutti i modelli successivi di reti neurali,seppur con rilevanti modifiche. L'apprendimento viene ottenuto per retroazione adattando i pesi numerici con lo scopo di minimizzare la differenza tra l'uscita prodotta e quella desiderata.Tale idea fece crescere un grande interesse dietro questa branca dell'informatica, interesse che scemò bruscamente quando nel 1969 Marvin Minsky e Seymour Papert dimostrarono che il percettrone non era in grado di apprendere la funzione XOR,infatti esso riesce a risolvere solo funzioni linearmente separabili. Sebbene questo problema poteva essere risolto utilizzando reti multi-strato, la crescente complessità computazionale dell'addestramento rendeva impraticabile questa strada, così che il vero passo significativo nella rinascita della ricerca in questo campo fu fatto solo nel 1986 grazie al lavoro di Rumelhart [31], che si propone di generalizzare la procedura di apprendimento del percettrone, e descrivere semplice schema per implementare un metodo di discesa del gradiente allo scopo di individuare pesi che riducano al minimo l'errore del sistema.

1.2.4 *Deep Learning*

Il Deep Learning rappresenta un moderno approccio al machine learning, in cui vengono utilizzati diversi strati che contribuiscono all'elaborazione desiderata. Ciascuno livello utilizza l'output dello strato precedente ed invia il proprio output al layer successivo, questo meccanismo contribuisce a rappresentare gerarchicamente l'informazione, infatti con l'aumento della profondità, aumenta anche il livello d'astrazione. Ciò che le differenzia maggiormente dalle reti classiche è l'applicazione di funzioni altamente non lineari tra un livello e l'altro, e non semplici combinazioni lineari degli input, capaci, quindi, di catturare dipendenze più complesse. Ogni livello, perciò, tende ad assumere una sua funzione specifica, diversa da quella del livello precedente, capacità di cui invece le tecniche classiche non disponevano. L'apprendimento viene quindi compiuto partendo da dati grezzi ed astraendo mano a mano una informazione di alto livello. Un esempio molto calzante è proprio quello della sentiment analysis: supponiamo che il primo strato della rete prenda in input le parole codificate di una recensione, esse verranno poi processate dagli strati successivi che individueranno eventuali sotto-frasi e le relative polarità. A più alto livello verrà infine individuata la polarità risultante che deriva da quella delle sottofrasi individuate negli strati inferiori. Questo tipo di reti imparano per esempi, riadattandosi al fine di creare l'output che più si avvicini a quello desiderato. Il tipo di apprendimento può essere sia supervisionato che non,inoltre, la combinazione di una fase iniziale in cui vengono presentati dati alla rete senza supervisione ad una fase finale in cui invece vengono presentate informazioni categorizzate,può aiutare a ridurre sensibilmente la quantità di dati con label, la cui disponibilità è molto spesso limitata.Per contro, questo approccio richiede reti molto vaste e quindi computazionalmente costose,non è un caso, infatti, che solo con l'avvento di hardware dedicati e prestazionali questa branca di ricerca abbia conosciuto un grande sviluppo.

1.3 COME FUNZIONA UNA RETE NEURALE?

1.3.1 *I nodi della rete*

Qualsiasi rete neurale è composta da nodi, anche detti neuroni artificiali, ciascuno di essi ha collegato in input N nodi e produce in output ad M nodi un valore descritto dalla funzione $y_m = f_m(x)$

É utile a questo punto focalizzarsi sul come matematicamente un neurone artificiale reagisca ad un determinato input. L'output di un qualsiasi nodo è descritto come:

$$y_j = \sigma(\sum_i w_{ij} \cdot x_j + b_j)$$

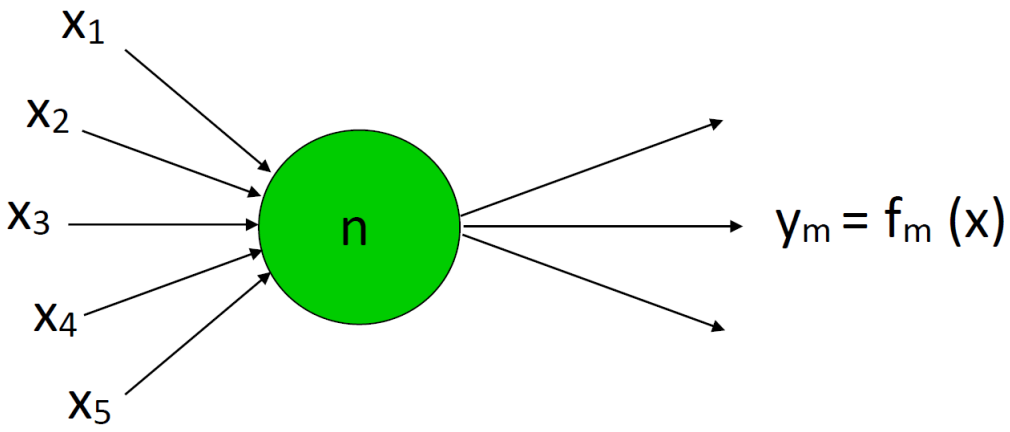


Figura 4.: Nodo n di una rete neurale

oppure:

$$y = \sigma(W \cdot x + b)$$

Come evidenziato nella figura 5 possiamo distinguere tre componenti fondamentali nella formula appena enunciata:

- **Somma Pesata degli Input** $(W \cdot x)$: i pesi (W) determinano l'efficacia della connessione fra i due neuroni e quindi in che misura considerare quel determinato input. Torna evidente il parallelismo biologico con le connessione dendritiche delle sinapsi, che favorisco o meno la produzione di un certo output.
- **Bias**(b): detto anche offset è un ulteriore peso, serve per regolare il punto di lavoro del neurone stesso
- **Funzione di attivazione** (σ): viene applicata all'input pesato e determina la reazione del neurone allo stimolo,ne esistono di diversi tipi e li vedremo in dettaglio.

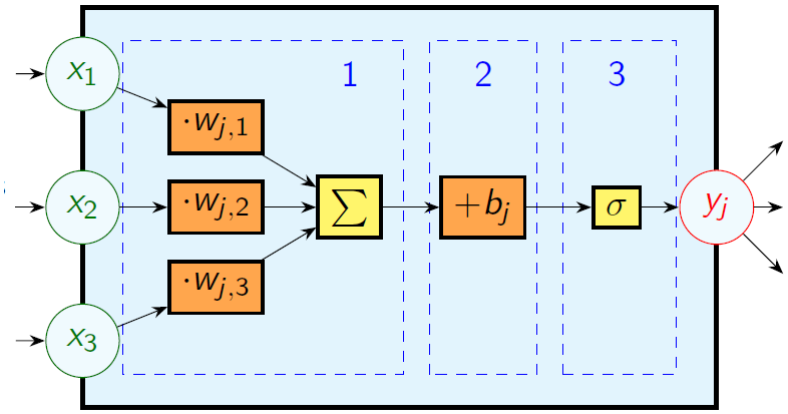


Figura 5.: Schema interno di un neurone artificiale

1.3.2 *Funzioni di attivazione*

Esse giocano un ruolo fondamentale nelle Neural Network,essendo responsabili dell'attivazione o meno di un neurone,ne esistono di diversi tipi, anche in base al ruolo che determinati neuroni devo ricoprire nella rete e al task da svolgere.La più semplice è sicuramente la funzione scalino (Figura 6a),come quella utilizzata nel neurone di Pitt [23], che imita proprio il funzionamento del suo corrispettivo biologico. Questo tipo di funzioni non viene però più utilizzato da tempo. Quasi tutte le reti neurali moderne però utilizzano funzioni che abbiano due caratteristiche fondamentali:

- **Non linearità:** è necessaria per eseguire una mappatura più precisa possibile dell'input, inoltre strati multipli con funzioni lineari sarebbero inutili.Nell'applicare la discesa a gradiente infatti si noterebbe,che per una funzione lineare, la derivata rispetto a x è una costante.Ciò significa che il gradiente non ha alcuna relazione con X. É evidente come questo sia un comportamento da evitare.

- **Continuità e differenziabilità:** queste caratteristiche sono necessarie per la back-propagation, infatti per trovare i minimi la funzione deve essere differenziabile o perlomeno in soli pochi punti di essa può essere non differenziabile

1.3.2.1 *La sigmoide*

La funzione Sigmoide è tra le più utilizzate, al momento, nel campo delle reti neurali. E' una versione "smussata"della funzione gradino,è non lineare, consente inoltre di avere una buona variazione dell'ouput prodotto anche con piccole modifiche dei dati di input, permettendo di descrivere al meglio certi comportamenti, i valori prodotti saranno comunque sempre compresi tra 0 ed 1. Il suo comportamento è descritto dalla formula:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Il problema di questo approccio è che come evidenziato dal grafico in figura 6b, agli estremi della funzione i valori di Y tendono a variare molto meno ai cambiamenti in X, ciò significa che in quella regione il gradiente sarà molto piccolo, l'apprendimento sarà quindi molto lento o addirittura inesistente,questo problema prende il nome **Vanishing gradient problem**.

1.3.2.2 *La tangente iperbolica*

Questa funzione è molto simile alla sigmoide appena discussa,anch'essa è non lineare ed è quindi possibile comporre reti multistrato con neuroni che ne fanno uso.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 = 2 \cdot sigmoid(2x) - 1$$

Gli output prodotti possono variare nell'intervallo $(-1, 1)$, la vera differenza risiede nel fatto è che il suo gradiente è più forte rispetto alla sigmoide, la discesa a gradiente è quindi più ripida, anch'essa non è pero esente dal Vanishing gradient problem (Figura 6c).

1.3.2.3 *Rectified Linear Unit (ReLU)*

Venne introdotta da Hahnloser et al nel 2000 [15] con forte ispirazione biologica e molti aspetti innovativi,dal 2015 sono la famiglia di funzioni maggiormente utilizzate in ambito deep learning [21]. Innanzitutto è di natura non lineare e differenziabile, a parte nel punto 0,è in grado quindi di approssimare qualsiasi tipo di funzione come combinazione di ReLu. Un enorme vantaggio che offre rispetto ad altri tipi di funzione è la sparsità di attivazione, infatti, in una rete inizializzata casualmente solo il 50% dei neuroni viene attivato. É descritta come:

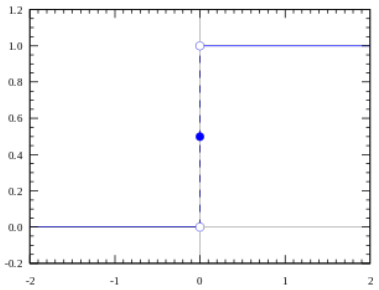
$$f(x) = max(0, x)$$

A causa delle Y prodotte per X negativo(Figura 6d), il gradiente può andare verso 0. Per le attivazioni in quella regione, il gradiente sarà 0 per cui i pesi non saranno regolati durante la discesa. Ciò significa che quei neuroni che entrano in questo stato smetteranno di rispondere alle variazioni,questi neuroni si dicono "morti".Esistono però versioni differenti della ReLu atte a mitigare questo problema.

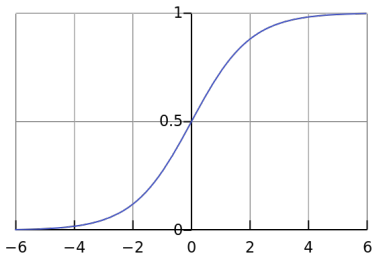
1.3.3 *Strati di una rete*

Per aumentare la capacità espressiva i nodi sono disposti in strati,seppur all'inizio l'utilizzo di un grande numero di strati era limitato, la disponibilità sempre maggiore di supporti di elaborazione efficaci ha portato ad un aumento incredibile della stratificazione delle reti neurali in ambito Deep Learning garantendo un ottimo grado di astrazione delle stesse.
Possiamo quindi distinguere tre tipi di layer:

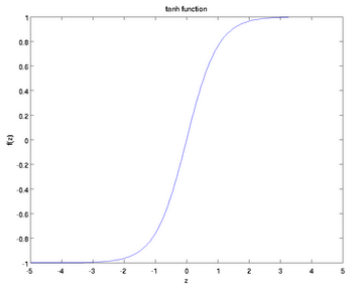
- **Input Layer:** Sono nodi adibiti esclusivamente a fornire i dati di input alla rete.
- **Hidden Layer:** Sono tutti quegli strati intermedi che elaborano gli input astraendo le feature a livelli sempre più alti in modo da svolgere il task richiesto.



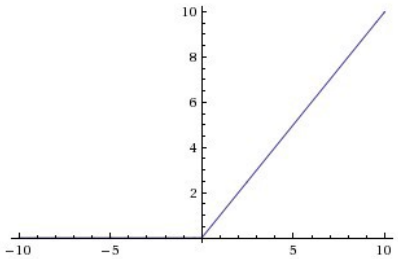
(a) *A gradino*



(b) *Sigmoide*



(c) *Tangente iperbolica*



(d) *ReLu*

Figura 6.: Varie tipologie di funzioni di attivazione

- **Output Layer:** Sono lo strato finale della rete, forniscono gli output prodotti.

I collegamenti fra nodi di layers diversi possono essere di vario tipo a secondo della rete utilizzata, vedremo successivamente degli esempi.

1.3.4 *Addestramento di una rete*

Una volta che è stata decisa l'architettura della rete,il numero di nodi e strati di cui comporla e quali funzioni di attivazione utilizzare si può procedere all'addestramento vero e proprio, in cui la rete cerca di imparare a svolgere al meglio il task preposto. Possiamo distinguere tre principali tipologie di addestramento:

- **Addestramento supervisionato**: In un addestramento di questo tipo vengono forniti sia gli input che gli output desiderati. La rete quindi elabora gli ingressi e confronta le uscite prodotte rispetto alle quelle attese. L'obbiettivo è quello di minimizzare la differenza fra i due valori,questo discostamento viene valutato sulla base di una funzione di costo(errore quadratico medio oppure cross-entropy),bisognerà quindi diminuire al massimo questo costo,utilizzando la discesa a gradiente,ciò vede la realizzazione nell'algoritmo di back-propagation dell'errore,di cui verrà discusso tra poco. Gli eventuali errori vengono quindi propagati indietro attraverso la sistema, provocando l'aggiustamento dei pesi che

influiscono sul funzionamento della rete. Questo processo viene applicato iterativamente su insieme di dati detto "training set". Lo stesso training set può essere ripresentato più volte alla rete, magari cambiando l'ordine di presentazione dei dati, l'addestramento quindi, avviene in più epoche. Non sempre però il training set presentato può portare ad un grado di efficacia della rete soddisfacente, ciò può essere dovuto a limiti intrinsechi della rete ma anche da un volume troppo basso di dati di input o da una varietà inadeguata di dati. Può capitare, infatti, che la rete si adatti bene al training set, ma invece si comporti meno bene con i dati scelti per testarla (test set), in questo caso la rete non è riuscita ad astrarre sufficientemente il comportamento desiderato, ma si è semplicemente adattata alle label presentate. Per esempio, addestrare la rete per troppe epoche potrebbe portare ad over-fitting, cioè la situazione appena descritta, è sempre bene, quindi, valutare con attenzione le scelte che riguardano il come addestrare la rete, in quanto ogni fattore concorre poi al raggiungimento, o meno, di risultati soddisfacenti.

- **Addestramento non supervisionato**: Nell'addestramento senza supervisione, alla rete vengono forniti solo gli input,senza dati completi di label. Il sistema stesso deve quindi decidere quali funzioni utilizzerà per raggruppare i dati di input. Questo è spesso definito come auto-organizzazione o adattamento. Il più comune metodo di apprendimento non supervisionato è il clustering per trovare feature nascoste o raggruppare dati. I cluster sono modellati utilizzando come misura di somiglianza,metriche come la distanza euclidea. I metodi di apprendimento non supervisionati vengono utilizzati nella bioinformatica per l'analisi delle sequenze e il clustering genetico,nell'attività di data mining,nel natural language processing,e nell'ambito della visione artificiale atta al riconoscimento di oggetti.

- **Apprendimento per rinforzo (Reinforcement Learning)** :è un tipo di apprendimento influenzato dalla psicologia comportamentale. Riguarda il come gli agenti software debbano agire in un ambiente in modo da massimizzare una certa ricompensa cumulativa. Alla rete non viene detto cosa fare, ma semplicemente viene detto di massimizzare il punteggio ottenuto, comportamenti conformi all'obbiettivo verranno premiati, mentre quelli inadeguati verranno penalizzati. I risultati ottenuti dalle reti di questo tipo negli ultimi anni, sono straordinari, basti pensare alle dirompenti DQN(Deep Q-Network) che utilizzano questo principio e si sono dimostrate capaci di imparare a giocare a 50 diversi giochi per la console Atari 2600 solo guardando la variazione del punteggio in relazione alle proprie mosse. [26]

1.3.5 *Funzioni di costo*

Come introdotto nella sezione precedente relativa all'addestramento di una rete neurale,per valutare la bontà del processo di apprendimento, è necessario avere una metrica di giudizio, cioè una funzione di costo, con cui computare quanto sia grande la differenza fra gli output prodotti e quelli attesi per poi andare a regolare la rete minimizzando questo valore. Per farlo, tuttavia, dobbiamo scegliere una funzione di costo che rispetti alcuni vincoli,il più importante di questi è quello di utilizzare una differenziabile rispetto a tutti i possibili output.Ciò è necessario per il corretto funzionamento della discesa a gradiente.Storicamente le funzioni di costo più utilizzate sono state:

- **Errore quadratico medio (MSE)** : indica la discrepanza quadratica media fra i valori dei dati osservati ed i valori dei dati stimati. È descritto dalla formula:

$$MSE = \frac{1}{2N}\sum_{1}^{N}\|g_i - y_i\|^2$$

  dove g_i indica l'output atteso e y_i quello prodotto. Questo tipo di funzione risente di un particolare problema,cioè porta ad un addestramento molto lento in caso di output marcatamente errati, contrariamente a quello che ci si può aspettare infatti, essa porta a computare gradienti molto piccoli nei casi di errori estremi, rendendo quindi difficile un ulteriore addestramento

- **Cross entropy** : funzione di costo che nasce proprio per ovviare ai problemi riscontrati nel MSE,è descritta come:

$$= -\frac{1}{n}\sum_{x}[g \ln y + (1-g)\ln(1-y)]$$

  dove g_i indica l'output atteso e y_i quello prodotto. Due proprietà in particolare rendono questa funzione di costo molto efficace,infatti in primo luogo, il valore prodotto non è mai negativo. Cioè è evidente dal fatto che tutti i singoli termini della somma nella formula sono negativi, poiché entrambi i logaritmi sono di numeri nell'intervallo 0,1 e vi è un segno meno davanti alla sommatoria . In secondo luogo, se l'output effettivo del neurone è vicino all'output desiderato allora la cross-entropy sarà prossima a zero,mentre tanto più i due valori si discosteranno e tanto la rete apprenderà più velocemente, in quanto la curva del gradiente è molto ripida,permettendo una discesa più efficace.

1.3.6 *Discesa del gradiente e Backpropagation*

Discesa a gradiente

Nelle precedenti sezioni sono state introdotte le modalità di addestra-

mento di una rete neurale, che si basa sul principio cardine di minimizzazione dell'errore, cioè quel valore calcolato dalla funzione di costo che indica lo scarto fra l'output prodotto e quello atteso. Per addestrare la rete vanno quindi regolati i parametri della rete (pesi e bias) al fine di ottenere il costo più basso possibile, matematicamente parlando, dobbiamo cercare il minimo della funzione. Un esempio molto semplificato di ciò è in figura 7 , nel grafico viene infatti quali valori produce la funzione di costo applicata agli output di una rete con i pesi W_1 e W_2 . Abbiamo quindi realizzato che per migliorare il comportamento

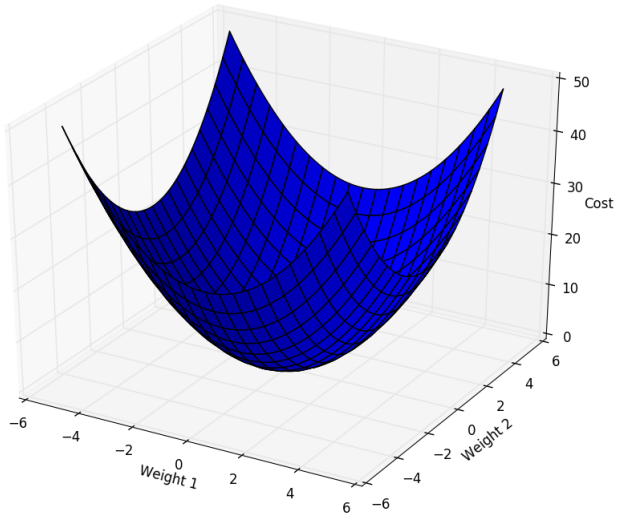


Figura 7.: Grafico dell'andamento della funzione di costo al variare dei pesi.Nell'asse z è rappresentato il valore assunto dalla funzione,negli assi x e y i pesi

della rete dobbiamo trovare la giusta combinazione di pesi che ci porti ad incorrere in un punto di minimo,il problema non è però per nulla banale, infatti una rete neurale non possiede soltanto due pesi come nell'esempio giocattolo appena descritto, ma anche migliaia di parametri e quindi se volessimo rappresentare il comportamento della funzione di costo in un grafico, migliaia di assi. Di primo acchito potremmo pensare di risolvere la questione ponendo la derivata della funzione di costo a zero e risolvere l'equazione, questa strada non è però percorribile, innanzitutto, infatti, non abbiamo una equazione che descriva semplicemente la funzione di costo, quindi individuarla e calcolarne le derivate è tutt'altro che scontato. In secondo luogo la funzione in gioco è multidimensionale,come già enunciato, vi è una dimensione per ogni peso, ciò significa che trovare i punti in cui tutte le derivate sono a zero è altrettanto complicato. Infine possono esistere molti punti di minimo e di massimo, trovare il migliore sarebbe computazionalmente costoso. Esiste però una classe di algoritmi che posseggono tutte le caratteristiche per svolgere al meglio questo compiti,sono detti algoritmi di ottimizzazione iterativa, in quanto progrediscono gradualmente

verso la soluzione ottimale.
Il più comune ed utilizzato è la **discesa a gradiente**, essa si propone di seguire la direzione del gradiente calcolando le derivate parziali della funzione di costo rispetto ai pesi,cercando di minimizzare il costo ad ogni batch aggiustando i pesi stessi. Questo processo è descritto da:

$$W = W - \eta \frac{\delta C}{\delta W}$$

Tanto più ci si avvicinerà al minimo e tanto più il termine derivato diventerà piccolo, avvicinandosi sempre di più allo zero, ed assumendo quindi il miglior valore possibile,il discorso appena fatto vale ovviamente per ciascuno dei pesi della rete. Per η si intende un iperparametro detto **learning rate**, che ci dà la misura di quanto velocemente la rete debba apprendere. Scegliere con oculatezza questo parametro è di fondamentale importanza, con un learning rate troppo alto, infatti, la rete potrebbe aggiustarsi in modo troppo brusco facendo variare il gradiente troppo rapidamente, non riuscendo quindi ad individuare un punto di minimo. Al contrario, se questo parametro assume un valore troppo basso la rete potrebbe non convergere mai, o con una velocità troppo bassa. Per scegliere questo numero non esiste una formula adatta a tutte le situazioni, dipende molto dal tipo di task a cui si vuole addestrare la rete,dal tipo e dalla quantità di dati di training,dal tipo di rete utilizzata... Pratica comune è però diminuire il learning rate durante l'arco dell'addestramento, in quanto si vuole mentre all'inizio del training si è più lontani dal punto di minimo,essendo quindi necessari "salti" più ampi per trovare il punto,col passare del tempo le variazioni veramente necessarie sono minori.

Backpropagation

La Backpropagation è un metodo utilizzato nelle reti neurali per calcolare il contributo all'errore totale da parte di ciascun neurone dopo ogni batch di training e regolare,quindi, di conseguenza i giusti pesi al fine di minimizzare lo scostamento tra l'output prodotto e quello atteso,è quindi un metodo di ottimizzazione della discesa a gradiente appena descritta. La parola retro-propagazione è autoesplicativa sul come ciò venga fatto,l'errore viene appunto calcolato all'uscita della rete e distribuito indietro attraverso tutti i livelli della stessa,viene quindi utilizzata la regola di derivazione a catena per calcolare il gradiente per ogni layer. L'algoritmo che descrive questo metodo è il seguente:

- Sia N una rete neurale con e connessioni m input ed n output
- Con $x_0, x_1, ..., x_m$ denoteremo i vettori in R^m detti input.Con $y_0, y_1, ..., y_n$ denoteremo i vettori in R^n detti output e Con $w_0, w_1, ..., w_e$ denoteremo i vettori in R^e detti pesi.
- La rete neurale corrisponde ad una funzione $y = f_N(w, x)$ che posto un peso w e dato un input x produce un output y

- L'ottimizzazione ha luogo prendendo in input una sequenza di esempi di training $(x_1, y_1), ..., (x_p, y_p)$ e producendo una sequenza di pesi $w_0, w_1, ..., w_p$ partendo da un w_0 inizializzato casualmente

- Questi pesi vengono calcolati a loro volta, prima viene calcolato w_i utilizzando soltanto (x_i, y_i, w_{i-1}) per ogni $i = 1, ..., p$ L'output dell'algoritmo è ora w_p fornendoci ora una nuova funzione $y = f_N(w_p, x)$. Consideriamo ora il caso in cui i = 1

- Il calcolo di w_1 a partire da (x_1, y_1, w_0) viene compiuto considerando un peso variabile w ed applicando la discesa a gradiente sulla funzione $w \mapsto E(f_n(w, x_1), y_1)$ per trovare un minimo locale partendo da $w = w_0$

1.4 TIPOLOGIE DI RETI NEURALI

Nel campo delle reti neurali possiamo distinguere due macro famiglie di reti neurali:

- **Reti Feed-forward**: sono reti in cui le connessioni tra le neuroni non costituiscono cicli,è stata la prima e più semplice tipologia di rete neurale.In essa le informazioni si muovono in una sola direzione, in avanti, dai nodi di ingresso, passando per i nodi nascosti (se presenti) per poi terminare nei nodi di uscita.

- **Reti ricorrenti**: sono in cui i collegamenti tra neuroni formano dei cicli diretti. Ciò consente di processare informazioni ottenute in passi temporali diversi,modellando così comportamenti dinamici. A differenza delle reti neurali feed-forward, le RNN possono utilizzare il propria stato interno per elaborare sequenze arbitrarie di input.

Ma ora vediamole in dettaglio.

1.4.1 *Feed-forward Networks*

Come già anticipato sono la prima tipologia di reti che è emersa, dapprima con le reti a singolo strato e poi con quelle multi-layer. In questo tipo di reti sono consentite solo connessioni verso lo strato successivo,sono quindi acicliche,il loro comportamento è paragonabile ad una funzione combinatoria e sono perciò incapaci di mantenere uno stato interno.

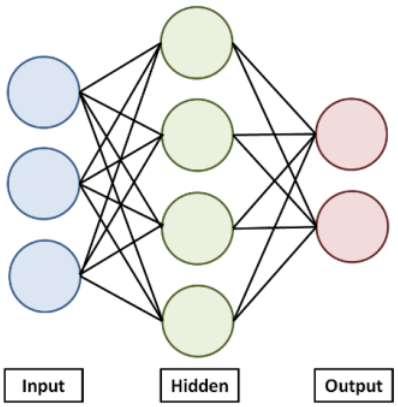


Figura 8.: Schema di una rete feed-forward a tre strati

Un grande limite delle Feed-forward network è che a causa della loro dimensione prefissata gli input ed gli output devono avere a loro volta una dimensione fissa, nell'affrontare problemi reali, però ,ci si imbatte molto spesso con dati che non hanno dimensione fissa, basti pensare ad una immagine, la cosa si fa ancora più limitante se si vogliono trattare dei testi. In generale, queste reti non sono adatte alla gestione di sequenze, infatti i loro output dipendono solo dall'attuale input, caratteristica che mal si concilia con le dipendenze temporali che si possono riscontrare, ad esempio, nel linguaggio umano.

1.4.2 *Convolutional Neural Networks*

Fanno parte della famiglia delle reti feed-forward ma i nodi tra gli strati inferiori non sono connessi in modo denso,ciò per consentire il riconoscimento di pattern, determinati nell'addestramento della rete. Dato in input un array multidimensionale (spesso 2D), la rete cerca determinati pattern sull'intero array tramite operazioni di convoluzione.Le reti convoluzionali sono state ispirate dal modello di connettività dall'organizzazione della corteccia visiva animale,infatti i singoli neuroni corticali rispondono agli stimoli solo in una regione limitata del campo visivo noto come campo ricettivo. I campi ricettivi di diversi neuroni si sovrappongono parzialmente in modo tale da coprire l'intero campo visivo.

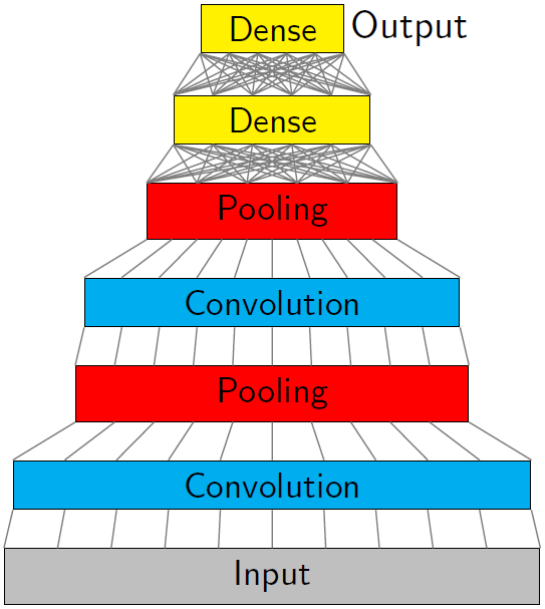


Figura 9.: Rappresentazione degli strati di una Convulutional Neural Network

Le CNN utilizzano i campi ricettivi come strati, sfruttando la struttura spaziale degli input,ogni nodo elabora una regione specifica dello strato sottostante. Tipicamente esistono diversi tipi di layers:

- **Input Layer**: lo strato adibito alla raccolta degli input,la sua dimensione varia in base all'ambito in cui viene applicato.
- **Convolution Layer**: sono composti da nodi che coprono le regioni sovrapposte dello strato sottostante, i pesi sono condivisi, ma ogni nodo li applica alla propria regione.I pesi definiscono le feature che si stanno ricercando nel dato processato,per ogni caratteristica ricercata vi è un array di neuroni adibita al rilevamento.
- **Pooling Layer**: esegue un'aggregazione delle informazioni del volume di input,riducendo ulteriormente la dimensione dei dati,l'obbiettivo è quello di conferire invarianza rispetto a trasformazioni dell'input non significative mantenendo al contempo la capacità di individuazione dei pattern ricercati.
- **Dense Layer**: posto dopo diversi strati convoluzionali e di pooling, il ragionamento ad alto livello nella rete neurale avviene tramite questi strati in cui i neuroni sono collegati a tutte le attivazioni nel livello precedente.

Storicamente vengono utilizzate per lo più nella classificazione di immagini e video, uno dei primi esempi di tale utilizzo fu compiuto attraverso la pioneristica LeNet-5 [20] il cui scopo era quello di riconoscere numeri

scritti a mano. Non va però sottovalutato l'applicazione anche in altri campi, ne è prova evidente gli ottimi risultati ottenuti nella Sentiment Analysis a livello di caratteri utilizzando le CNN [35].

1.4.3 *Recurrent Neural Networks*

La comprensione umana è basata sulla creazione di un contesto con cui interpretare la realtà, i nostri pensieri, i nostri ricordi sono persistenti e contribuiscono a quello che può essere considerato come lo "stato interno" della propria mente. Le reti di tipo Feed-Forward che abbiamo appena considerato non seguono questa logica,non hanno uno stato interno dipendente dalla sequenza di input presentata, è evidente come ciò sia estremamente limitante in quegli ambiti in cui è richiesta un ragionamento contestuale, come ad esempio nel riconoscimento del linguaggio umano. Le RNN risolvono questo problema, possedendo collegamenti che possono formare cicli, dando quindi la possibilità di modellare comportamenti più complessi. L'output della rete è quindi, ora, dipendente sia dall'input ma anche dallo stato in cui si trova.
Possiamo pensare alle RNN come copie multiple della stessa rete,con ognuna di esse che trasmette un messaggio alla successiva. La figura 10 rende bene l'idea di questo comportamento "a catena", che la rende particolarmente adatta a gestire dati sequenziali.

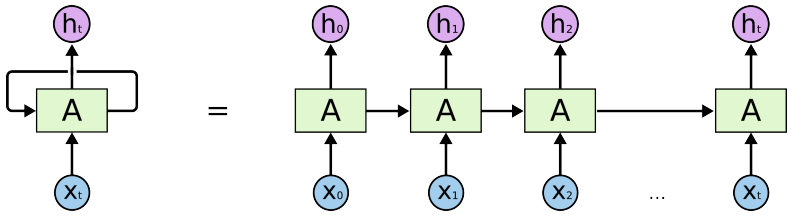


Figura 10.: Una rete neurale ricorrente "srotolata" nel tempo

Talvolta, però assolvere certi task, la rete dovrebbe cogliere delle dipendenze a lungo termine tra elementi (possono essere frame non sequenziali in un video,come parole in un testo), teoricamente le RNN sono capaci di gestire certe dipendenze, in pratica però, è stata riscontrata una certa difficoltà nel farlo, come evidenziato da Bengio et Al. nel 1994 [2], questo perchè il loro stato interno cambia troppo rapidamente in funzione dell'input. Inoltre durante la back-propagation il gradiente della funzione di costo potrebbe assumere valori inutilizzabili dopo molti passi temporali(Vanishing Gradient e Exploding Gradient). L'esempio più semplice di RNN sono le reti di Elman, introdotte nel 1990 [9].

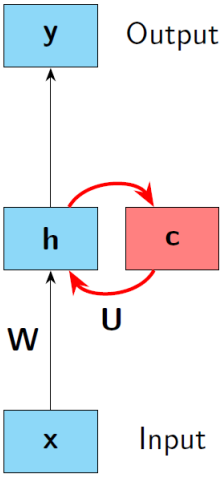


Figura 11.: Rappresentazione di una rete di Elman

Come visibile nella figura 11 la rete è composta da tre strati, la particolarità di questa architettura, però, risiede nelle unità contestuali (c, in rosso nella figura) collegate allo strato nascosto, in esse infatti, vengono propagati gli output ad ogni passo, permettendo quindi di mantenere lo stato precedentemente acquisito. Lo stato attuale è descritto come:

$$h_t = \sigma(W \cdot x_t + U \cdot h_{t-1} + b)$$

Durante l'addestramento essa può essere considerata come una rete multi-strato avente un layer nascosto per ogni passo temporale.

1.4.4 *Long Short-Term Memory Networks*

Sono una speciale tipologia di RNN,sviluppate proprio per risolvere le problematiche riscontrare nelle RNN normali,sono quindi capaci di gestire dipendenze a lungo termine, memorizzando informazioni per un lungo periodo. Vengono introdotte nella loro forma base nel 1997 da Hochreiter & Schmidhuber [17] e sono al giorno d'oggi utilizzate con successo in una larga varietà di task.
Ogni rete ricorrente può essere vista come la ripetizione a catena di un certo nodo, la differenza tra le LSTM e le RNN standard sta nella composizione del nodo ripetuto, infatti, invece di avere un singolo layer, il nodo è composto da 4 strati interagenti tra loro.

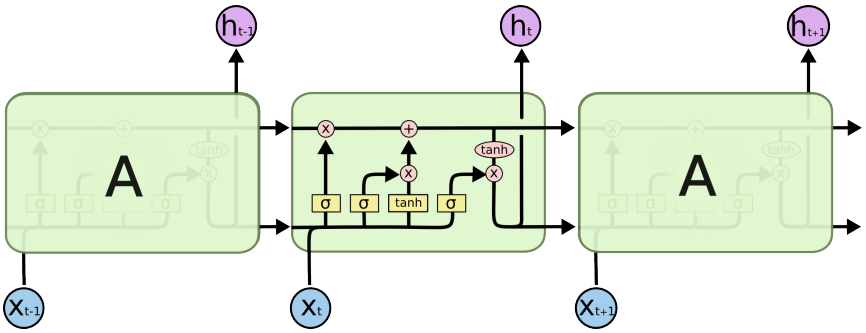


Figura 12.: Rappresentazione di una LSTM in diversi time step

L'idea di base dietro le LSTM è quella di evitare che l'input influenzi direttamente lo stato della rete,ma scegliendo selettivamente quali (ed in che misura) nuove informazione considerare per aggiornare il proprio stato e produrre un output. Esse hanno l'abilità, attraverso particolari strutture chiamate gates, di rimuovere o aggiungere informazioni all'interno dello stato del nodo, basandosi su pesi che indicano quanto "memorizzare" e quanto "dimenticare". Vediamo quali sono questi gates:

- **Forget Gate layer**: questo strato è adibito a decidere cosa rimuovere dalla memoria, producendo un vettore $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- **Inpute Gate layer**: questo strato è adibito a decidere valori nella memoria andremo a modificare, producendo un vettore $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- **Tanh layer**: questo strato è adibito alla creazione di un vettore contenente i valori candidati per aggiornare la memoria,descritto da $\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C)$
- **Output gate layer**: è lo strato adibito alla produzione dell'output,utilizzando lo stato ottenuto in questo passo temporale, $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$, viene quindi computato un vettore che indica quale parto dello stato della cella verrà prodotto in output,cioè $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$. Infine l'output prodotto è il seguente: $h_t = o_t * \tanh(C_t)$

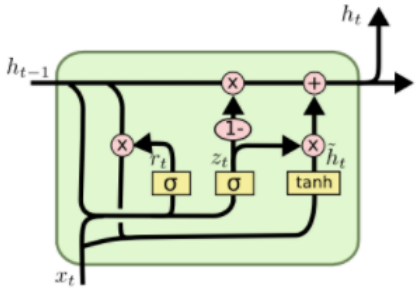
1.4.5 *Approcci recenti*

Verranno ora passate rapidamente in rassegna le più recenti e promettenti tipologie di reti neurali.

1.4.5.1 *Gated Recurrent Neural Networks*

'E una variante delle LSTM, sono state introdotte nel 2014 da Cho et Al [5], in esse i Forget ed Input gate vengono accorpati in un unico

componente detto Update gate.Inoltre le GRU non hanno bisogno di utilizzare una unita di memoria per controllare il flusso di informazioni (come invece accade nelle LSTM),essa fa direttamente uso di tutti gli strati nascosti. Grazie a questi accorgimenti,la rete risulta avere meno parametri da addestrare,rendendo il training più veloce senza però perdere in accuratezza.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figura 13.: Rappresentazione di una cella GRU con relative formule di produzione dell'output

1.4.5.2 *Dynamic Memory Networks*

É un tipo di rete neurale ottimizzata per problemi di question-answering,dato un certo set di training e delle domande, la rete è capace di formare una sorta di memoria episodica con cui generare delle risposte rilevanti. Le DMN memorizzano al loro stati nascosti multipli ed usano un'accurata strategia per scegliere uno di essi, questa strategia è detta Attention Mechanism, proprio perché individua su quali elementi concentrarsi.Sono state introdotte nel 2016 da Kumar et Al [19].

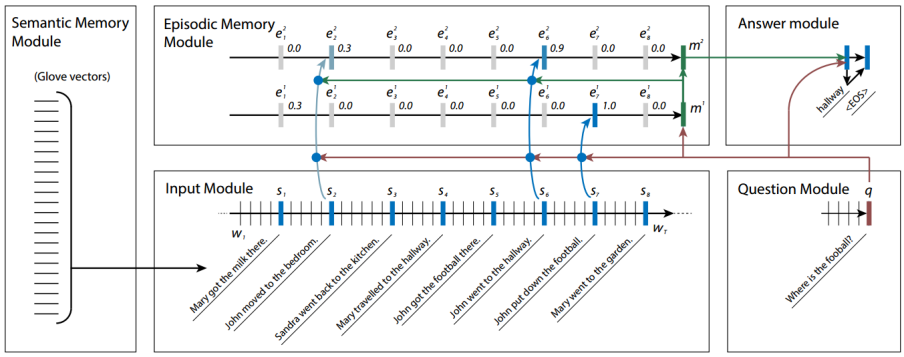


Figura 14.: Schema di una Dynamic Memory Networks

Come evidenziato in Figura 15 è composta da 5 moduli: Semantic Memory Module,Input Module,Question Module,Episodic Memory Module ed infine Answer module.

1.4.5.3 *Neural Turing Machine*

Introdotta nel 2014 da Graves et Al [13],una NTM è costituita da una rete neurale (feed-forward o RNN) che interagisce con una memoria indirizzabile esterna per mezzo di "testine" di lettura e scrittura. Gli indirizzi di memoria a cui accedere in lettura e/o scrittura sono controllati dalla rete secondo un meccanismo che agevola pattern comuni di accesso alla memoria, come il reperimento di informazioni simili a un input dato e lo scorrimento sequenziale. Si sono dimostrate molto più efficaci delle LSTM nell'apprendimento di semplici algoritmi (ripetizione o copia di una sequenza,per esempio).

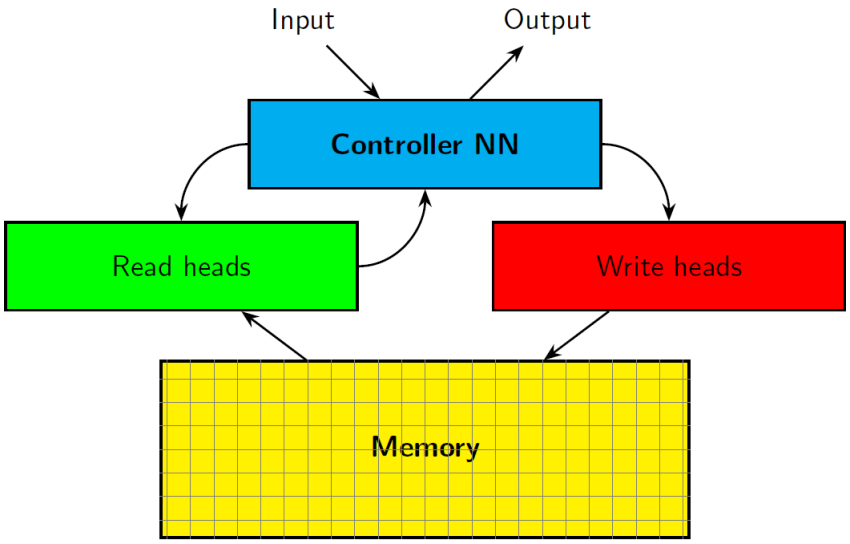


Figura 15.: Schema di una Neural Turing Machine

Sono costituite da:

- **Controller**, che interagisce sia con l'esterno sia con la memoria
- **Testine di lettura e scrittura**, mediante le quali il controller interagisce con la memoria, scrivendo o leggendo dati in base alla necessità.
- **Memoria** costituita da una matrice di dimensione MxN dove M è il numero di locazione di memoria ed N è la dimensione dei vettori memorizzati in ogni cella.

2

DIFFERENTIABLE NEURAL COMPUTER

2.1 INTRODUZIONE

Molteplici sono i risultati ottenuti dalle reti neurali in campi dov'è richiesto di riconoscere pattern, che si tratti di determinati oggetti in un'immagine, di un numero scritto di pugno umano o del riconoscimento di un viso, ma nella costruzione di una rete che riesca a rappresentare strutture di dati complesse e mantenere dati anche per lunghi periodi, si è ancora all'inizio del percorso. Le DNC [14] si propongono come una struttura flessibile e dinamica atta a risolvere questo problema, grazie all'utilizzo combinato di una rete neurale ed una memoria indirizzabile.Il loro nome ha origine dal fatto che possono apprendere per esempi(e sono differenziabili) come le reti neurali ma sono capaci anche di memorizzare strutture dati complesse come i computer. Ciò che rende veramente innovative queste reti, non è tanto l'utilizzo di una memoria esterna, la quale era già stata introdotta con altre tecnologie, ma il come queste reti utilizzino questa memoria. In generale quando parliamo di utilizzo della memoria in ambito informatico, siamo abituati a pensare in termini di strutture dati e variabili il cui utilizzo è stato determinato a priori, le DNC invece sono capaci di organizzare strutture dati in modo totalmente automatico, in modo da assolvere i propri compiti al meglio. La rete, quindi, riesce autonomamente a gestire al meglio la propria memoria imparando a farlo con un apprendimento per esempi, in cui le viene presentato un input e mostrato il corrispondente output che ci aspettiamo produca. Tutto il sistema è infatti differenziabile,permettendo quindi alla rete di organizzare la propria memoria con l'obbiettivo di minimizzare il gradiente. Questo tipo di reti hanno dimostrato di saper svolgere egregiamente task altamente complessi come trovare il percorso più corto da una stazione ad un altra di una metropolitana,rispondere a domande relative ad alberi genealogici, ma come anche apprendere semplici algoritmi. Come vedremo in dettaglio, la memoria esterna delle DNC è espandibile,sono quindi Turing-complete, infatti, i comportamenti appressi sono indipendenti da dove sono memorizzati effettivamente i dati.

2.2 ARCHITETTURA

Essendo una espansione delle NTM, condivide con l'architettura di base, è quindi composta da un Controller che interagisce con l'esterno e con la memoria esterna attraverso delle testine(di scrittura e lettura), ci si riferisce alle componenti che si relazionano con la memoria con il termine "testine" per analogia con la macchina di Turing, a cui sono ispirate queste reti. Vengono inoltre utilizzate delle strutture dati atte a mantenere informazioni sullo stato di utilizzo delle celle di memoria ed all'ordine con cui sono state scritte.

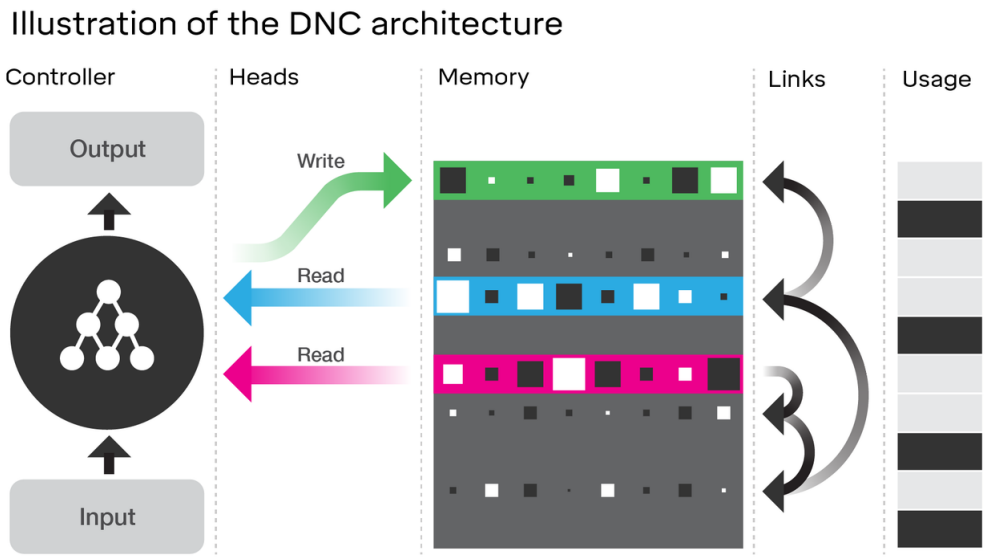


Figura 16.: Illustrazione dell'architettura di una DNC. La rete neurale che fa da controller riceve un input dall'esterno e basandosi su di esso, interagisce con la memoria utilizzando le testine di lettura e di scrittura. Per aiutare il controller a navigare la memoria,le DNC memorizza dei 'Temporal Link' per tenere traccia dell'ordine in cui le celle sono state scritte e mantenendo le informazioni relative all'utilizzo di ogni cella('Usage').Da [14]

2.2.1 *Il controller*

Si tratta di una rete neurale multi strato, adibita ad accogliere gli input,alla gestione della memoria esterna ed alla produzione di output. Per ogni passo temporale, esso decide cosa leggere o eliminare dalla memoria,ma anche cosa scrivere, come vedremo in seguito il meccanismo di scrittura è particolarmente importante e distintivo rispetto al altri tipi di rete. Possono essere utilizzati, alternativamente, due tipologie di reti:

- **Recurrent neural network** : di solito viene utilizzata una LSTM, che grazie al suo stato interno può agire in complementarietà con la memoria esterna, se volessimo fare un parallelo con la CPU di un computer(nel nostro caso il controller) diremmo che le attivazioni nascoste della rete ricorrente sono come i registri del processore. Queste caratteristiche permettono al controller di poter aggregare informazioni che provengo da più time step diversi e cosa molto importante per i task più complessi, utilizzare contemporaneamente più testine.

- **Feed-Forward network**: anche se non si tratta di rete ricorrente, possono mimarne il comportamento utilizzando la memoria esterna, leggendo e scrivendo sulla stessa locazione di memoria in ogni passo temporale. Hanno il vantaggio di utilizzare la memoria in modo più trasparente, in quanto il loro funzionamento è più semplice da interpretare raffrontato allo stato interno di una RNN. Il problema di utilizzare reti di questo tipo per il controller è che il numero di testine concorrentemente utilizzate impone un collo di bottiglia sul tipo di computazione che le DNC possono compiere, a differenza delle RNN che invece non soffrono di questa limitazione, essendo capaci di memorizzare il vettore di lettura proveniente dagli stati precedenti.

2.2.2 *Le testine*

Come già anticipato, sono le componenti responsabili della lettura e scrittura di informazioni in memoria, si differenziano in:

- **Testine di lettura** (Read Heads): rappresentate in figura 16 in blu ed in rosa, possono essere una o,come in questo caso, più, che leggono contemporaneamente una parte della memoria. Le testine possono scegliere di utilizzare alternativamente fra tre diversi gate che simboleggiano diverse modalità di lettura. Può quindi decidere di leggere in modalità **Content Lookup** ('C') facendo uso di una chiave di lettura, oppure leggere nelle locazione di memoria muovendosi all'indietro(**Backwards**, 'B') o in avanti (**Forwards**, 'F') rispetto all'ordine in cui sono state scritte. La modalità scelta va ad influenzare i pesi di lettura,definiscono le distribuzioni di attenzione riguardanti le varie locazioni di memoria, infatti il vettore di lettura è definito come:

$$r = \sum_{i=1}^{N} M[i, \cdot\,] w^r[i]$$

  dove M è la memoria esterna, N è il numero di locazioni ,W la dimensione delle locazioni, w_r la distribuzione di attenzione e $'\cdot'$ tutti i $j = 1, ..., W$

- **Testine di scrittura** (Write Heads): rappresentate in figura 16 in verde,anch'esse possono essere più di una. Definiscono un erase vector ed un write vector allo scopo, rispettivamente, di scegliere cosa eliminare e cosa scrivere in memoria. Inoltre, un vettore che fa da chiave di scrittura viene utilizzato per trovare le locazioni precedentemente scritte e che la rete desidera modificare, grazie a questo meccanismo la scrittura viene compiuta in modo da agevolare la formazione di pattern comuni. L'operazione di scrittura è quindi descritta nel seguente modo:

$$M[i,j] \leftarrow M[i,j](1 - w^W[i]e[j]) + w^W[i]v[j]$$

ove M è la memoria esterna,W la dimensione delle locazioni, w_r i pesi di scrittura,e è il vettore di cancellazione e v il vettore di scrittura.

2.2.3 *La memoria*

Si tratta essenzialmente di una matrice a valori reali di dimensione $N \times W$, dove N è il numero di locazioni e W la dimensione del vettore memorizzato in ogni locazione. La cosa interessante ed innovativa,anche rispetto alle NTM, è il fatto che la memoria sia ridimensionabile senza dover procedere ad un nuove addestramento,rendendola di fatto una macchina touring-completa, capace di scalare in base alla dimensione del problema. Nel paper di presentazione delle DNC viene infatti mostrato come una rete con 256 locazioni di memoria, pre-addestrata a trovare percorsi in grafi, riesca a modulare il proprio comportamento in base alla memoria a disposizione. L'obbiettivo di questo esperimento era che sebbene all'aumentare delle triplette del grafo(source label,edge label, destination label) è richiesta una dimensione della memoria più elevata,la rete riesca a migliorare il proprio comportamento senza essere addestrata nuovamente,ma soltanto espandendo la memoria.

2.3 INTERAZIONE TRA TESTINE E MEMORIA

Come è già stato introdotto le testine utilizzano tre diversi meccanismi per "capire" su quali locazioni di memoria concentrarsi, al fine di organizzare le informazioni nel modo più significativo possibile, dobbiamo, infatti, ricordare sempre che la memoria esterna di una DNC non è un mero supporto di stoccaggio di dati.

2.3.1 *Content attention*

In questo meccanismo il controller, forma un vettore-chiave che viene comparato al contenuto di ogni locazione di memoria,basandosi sulla similarità tra i dati(calcolata in questo caso,con la similarità coseno). Il

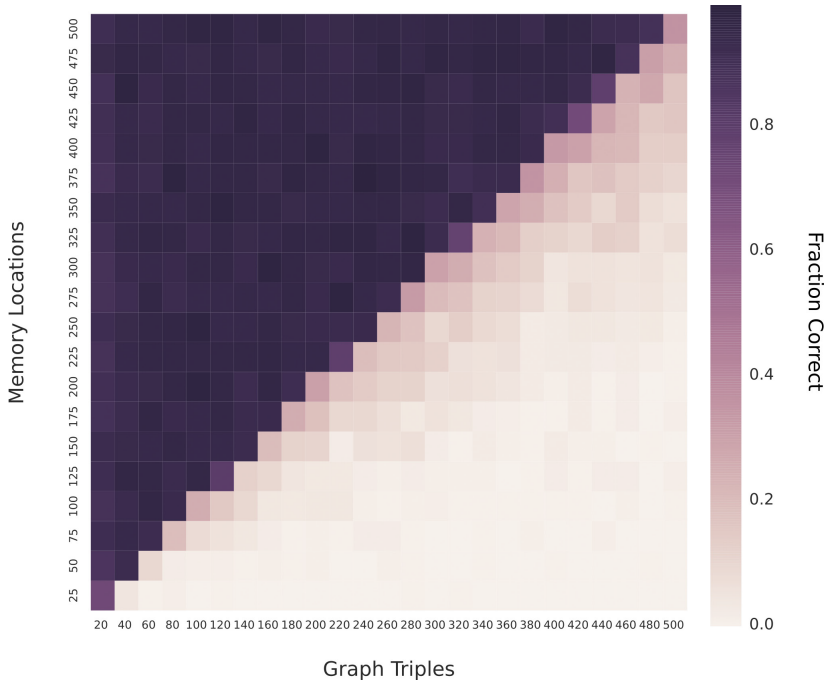


Figura 17.: Grafico della variazione di accuratezza di una rete pre-addestrata su un task di attraversamento di grafi. Da [14]

grado di somiglianza ottenuto determina quindi un peso che può essere utilizzato dalle testine di lettura dando luogo ad associative recall [1] , na anche da quelle di scrittura per modificare un vettore pre-esistente in memoria. É importante precisare che anche se la similarità tra la locazione e key-vector è parziale, è comunque possibile concentrare l'attenzione su quella cella. Grazie a questo meccanismo,infatti, è possibile leggere delle locazione di memoria che contengono informazioni importanti ma che non sono presenti nella chiave. In conclusione, quindi, basarsi sul contenuto delle celle è molto utile per navigare strutture dati associative, in quanto il contenuto di un indirizzo può includere dei riferimenti ad un altro associato.

2.3.2 *Temporal order attention*

Preservare l'ordine temporale è condizione necessaria in molti task e sembra avere anche un ruolo fondamentale nel processo cognitivo umano, le DNC si propongono quindi, attraverso questo meccanismo, di navigare tra i dati della memoria nell'ordine in cui sono stati scritti(o aggiornati). Per fare ciò utilizza una matrice(L) temporale dei collegamenti,di dimensione $N \times N$, in pratica $L[i, j]$ ha valore vicino ad 1 se i è stata la locazione di memoria scritta dopo j, altrimenti ha valore vici-

1 Meccanismo attraverso il cui si riesce a recuperare un'informazione a partire da solo una piccola parte di essa

no 0. Per ogni peso w l'operazione Lw sposta il focus in avanti verso le locazione di memoria scritte dopo quelle indicate in w, viceversa $L^T w$ sposta il focus all'indietro. Tutto ciò permette di recuperare sequenze nell'ordine in cui sono state scritte, anche se ciò è avvenuto in passi temporali non adiacenti.

2.3.3 *Memory allocation attention*

Questo terzo meccanismo è utilizzato nelle procedure di scrittura in memoria,a differenza delle NTM che potevano "allocare" in memoria solo secondo blocchi contigui,dando origine a problemi di gestioni della stessa,le DNC invece dispongono di una lista differenziabile che tiene traccia dell'utilizzo di ogni locazione di memoria, rappresentandolo con un numero compreso tra 0 ed 1 ed un peso corrispondente che segnala alla testina di scrittura in quali celle scrivere, in quanto libere. Se una cella viene scritta, il suo valore di utilizzo nel vettore u_t viene incrementato,mentre ad ogni lettura, viene diminuito, dando quindi la possibilità al controller di poter riallocare celle di memoria il cui contenuto non è più utile. Questo meccanismo ha un altra importante implicazione,cioè che i suoi principi sono avulsi dalla dimensione della memoria(come già introdotto nel capitolo 2.2.3) e visibile nella figura 17, la rete può venire addestrata utilizzando un certo quantitativo di memoria e poi essere utilizzata con uno slot di memoria diverso.

2.4 MECCANISMI DI FUNZIONAMENTO

In questa sezione verranno analizzati in dettaglio i principi di funzionamento delle singole componenti delle DNC, corredati di formule matematiche.

2.4.1 *Controller network*

In ogni instante di temp t la rete neurale che fa da Controller (N) riceve un vettore di input $x_t \in \mathbb{R}^{\mathbb{X}}$ dal dataset e produce un vettore di output $y_t \in \mathbb{R}^{\mathbb{Y}}$, che può rappresentare sia una distribuzione predittiva per un vettore target $z_t \in \mathbb{R}^{\mathbb{Y}}$ nel caso di apprendimento supervisionato, o una distribuzione di azioni nel caso di apprendimento per rinforzo. Inoltre, il controller riceve un insieme di R (con R numero di testine di lettura) vettori di lettura $r^1_{t-1}, ..., r^R_{t-1}$ dalla matrice di memoria $M_{t-1} \in \mathbb{R}^{\mathbb{N}\times\mathbb{W}}$ dello step temporale precedente attraverso le testine di lettura. A questo punto emette un vettore ξ_t il quale definisce l'interazione con la memoria nel time step corrente. Per convenienza nella notazione verranno concatenati i vettori di input e quelli di lettura in un unico vettore $\chi_t = [x_t; r^1_{t-1}; ...; r^R_{t-1}]$. Come già discusso nella sezione 2.2.1 può essere utilizzata come Controller sia una rete Feed-Forward che una

ricorrente,ma nell'implementazione utilizzata si fa uso di una LSTM [12] descritta dalle seguenti equazioni:

$$
\begin{aligned}
i_t^l &= \sigma(W_i^l[\chi_t; h_{t-1}^l; h_t^{l-1}] + b_i^l) \\
f_t^l &= \sigma(W_f^l[\chi_t; h_{t-1}^l; h_t^{l-1}] + b_f^l) \\
s_t^l &= f_t^l s_{t-1}^l + i_t^l \tanh(W_s^l[\chi_t; h_{t-1}^l; h_t^{l-1}] + b_s^l) \\
o_t^l &= \sigma(W_o^l[\chi_t; h_{t-1}^l; h_t^{l-1}] + b_o^l) \\
h_t^l &= o_t^l \tanh(s_t^l)
\end{aligned}
$$

dove l è l'indice del layer, $\sigma = 1/(1+\exp(-x))$ è la funzione sigmoide. $h_t^l, i_t^l, f_t^l, s_t^l, o_t^l$ sono rispettivamente i vettori di attivazione degli hidden,input,forget,state e output gate del layer l al tempo t. Con $h_t^0 = 0$ per ogni t e $h_0^l = s_0^l = 0$ per ogni l. Il termine W indica la matrice dei pesi addestrabili(per esempio,W_i^l è la matrice dei pesi che vanno negli input gate del layer l) e i termini b sono i bias addestrabili.
Ad ogni passo temporale, il controller produce un vettore di output v_t ed un vettore di interfacciamento $\xi_t \in \mathbb{R}^{(W\times R)+3W+5R+3}$ definiti come:

$$
\begin{aligned}
v_t &= W_y[h_t^1, ..., h_t^L] \\
\xi_t &= W_\xi[h_t^1, ..., h_t^L]
\end{aligned}
$$

Essendo il tipo di controller utilizzato ricorrente, i suoi output sono in funzione di tutti gli input presentati fino al momento corrente, possiamo perciò incapsulare questa operazione del controller come:

$$(v_t, \xi_t = \mathrm{N}[\chi_1; ...; \chi_t]; \theta)$$

dove θ sono l'insieme dei pesi addestrabili della rete. Infine il vettore di output y_t viene formato aggiungendo v_t al vettore ottenuto moltiplicando la concatenazione dei vettori di lettura e la matrice(di dimensione $RW \times Y$) dei pesi W_r :

$$y_t = v_t + W_r[r_t^1, ..., r_t^R]$$

Questo accorgimento permette alle DNC di condizionare il proprio output basandosi sulla memoria appena letta.

2.4.2 *Parametri di interazione*

Il vettore ξ_t prima di essere utilizzato per regolare l'interazione con la memoria viene suddiviso in questo modo:

$$\xi_t = [k_t^{r,1}; ...; k_t^{r,R}; \hat{\beta}_t^{r,1}; ...; \hat{\beta}_t^{r,R}; k_t^w; \hat{\beta}_t^W; \hat{e}_t; v_t; \hat{f}_t^1; ...; \hat{f}_t^R; \hat{g}_t^a; \hat{g}_t^w; \hat{\pi}_t^1; ...; \hat{\pi}_t^R]$$

Con : $k_t^{r,i}$ read key $i(1 \le i \le R)$, $\beta_t^{r,i}$ read strenght i ,β_t^W write strenght, e_t erase vector, v_t write vector, f_t^i free gate i, g_t^a allocation gate, g_t^w write gate, π_t^i read mode i.
I singoli componenti vengono quindi processati con varie funzioni per accertare che appartengano al dominio corretto:

- La funzione sigmoide viene utilizzata per vincolare i valori nell'intervallo $[0, 1]$
- La funzione 'oneplus' viene utilizzata per vincolare i valori nell'intervallo $[1, \infty]$, dove $oneplus(x) = 1 + \log(1 + e_x)$
- La funzione softmax viene utilizza per vincolare i valori nello spazio $S_n = \{\alpha \in \mathbb{R}^N : \alpha_i \inf[0, 1], \sum_{i=1}^{N} \alpha_i = 1\}$

Dopo aver processato i singoli componenti si ottengono i seguenti scalari e vettori:

- R read keys $\{k_t^{r,i} \in \mathbb{R}^W, (1 \leq i \leq R)\}$
- R read strenght $\{\beta_t^{r,i} = oneplus(\hat{\beta}_t^{r,i}) \in [1, \infty]; (1 \leq i \leq R)\}$
- La write key $\{k_t^q \in \mathbb{R}^W$
- La write strenght $\beta_t^W = oneplus(\beta_t^W) \in [1, \infty]$
- L'erase vector $e_t = \sigma(\hat{a}_t) \in [0, 1]^W$
- Il write vector $v_t \in \mathbb{R}^W$
- R free gates $\{f_t^i = \sigma(\hat{f}_t^i) \in [0, 1]; 1 \leq i \leq R\}$
- L'allocation gate $g_t^a = \sigma(\hat{g}_t^a) \in [0, 1]$
- Il write gate $g_t^w = \sigma(\hat{g}_t^w) \in [0, 1]$
- Le R read modes $\pi_t^i = softmax(\hat{\pi}_t^i) \in S_3; 1 \leq i \leq R$

2.4.3 *Leggere e scrivere in memoria*

La scelta delle locazioni su cui leggere o scrivere dipende da dei vettori di numeri non negativi la cui somma fa al massimo 1,detti "weightings". L'insieme completo dei vettori ammissibili su tutte le locazioni N è definito come :

$$\Delta_N = \{\alpha \in \mathbb{R}^N : \alpha_i \inf[0, 1], \sum_{i=1}^{N} \alpha_i \leq 1\}$$

Per le operazioni di lettura vengono utilizzati R read weightings per calcolare la media pesata del contenuto delle locazioni di memoria,possiamo quindi definire i vettori di lettura $r_t^1, ..., r_t^R$ (read vector) come

$$r_t^i = M_t^T w_t^{r,i}$$

Come abbiamo già visto nella sezione precedente i vettori di lettura vengono concatenati agli input, dando così accesso al contenuto della memoria.
Le operazioni di scrittura invece vengono mediate attraverso un solo

write weighting $w_t^W \in \Delta_N$ utilizzato insieme ad un vettore di cancellazione $e_t \in [0,1]^W$ (erase vector) ed un vettore di scrittura $v_t \in \mathbb{R}^W$, entrambi prodotti dal controller. La memoria viene quindi aggiornata come segue:

$$M_t = M_{t-1} \circ (E - w_t^w e_t^T) + w_t^W v_t^T$$

dove $\circ$ denota una moltiplicazione per elementi e E è una matrice $N \times W$ di uni.

2.4.4 *Indirizzamento della memoria*

Come già esplicato nelle precedenti sezioni, il sistema usa una combinazione di indirizzamento content-based e allocazione dinamica per determinare in quali locazioni di memoria scrivere,ed il content-base addressing insieme ad una matrice dei collegamenti temporali per determinare quali parti della memoria leggere. Vediamoli in dettaglio:

2.4.4.1 *Content-base addressing*

Ogni operazione di content lookup effettuata sulla memoria $Min\mathbb{R}^{N\times W}$ è descritta dalla funzione:

$$C(M,k,\beta)[i] = \frac{exp(\mathcal{D}(k, M[i,\cdot])\beta}{\sum_j exp(\mathcal{D}(k, M[j,\cdot])\beta}$$

dove $k \in \mathbb{R}_W$ è una chiave di lookup,$\beta \in [1,\infty]$ è uno scalare che rappresenta la forza della chiave(per valori di β grandi ci si concentrerà sui contenuti più simili,mentre per valori piccoli ci si concentrerà su più locazioni di memoria) e $\mathcal{D}$) è la similarità coseno,dove $\mathcal{D}(u,v) = \frac{u\cdot v}{|u||v|}$ I pesi $C(M,k,\beta)$ definiscono una probabilità di distribuzione normalizzata sulle locazione di memoria

2.4.4.2 *Dynamic memory allocation*

Al fine di permetter al controller di liberare ed allocare memoria a seconda delle necessità, le DNC contengono una linked list su cui è indicata quali locazioni di memoria sono disponibili, questa lista è detta memory usage vector ed è indicabile come $u_t \in [0,1]^N$ al tempo t. Prima di scrivere nella memoria,quindi, il controller produce un insieme di free gates f_t^i,uno per ogni testina di lettura che determinano se le locazioni lette recentemente possono essere liberate. Il vettore $\psi_t \in [0,1]^N$ rappresenta quante informazioni mantenere per ogni cella,ed è definito come:

$$\psi_t = \prod_{i=1}^{R} (1 - f_t^i w_{t-1}^{r,i}) \tag{1}$$

Mentre l'usage vector è definito come:

$$u_t = (u_{t-1} + w_{t-1}^w - u_{t-1} \circ w_{t-1}^w) \circ \psi_t$$

Intuitivamente, le celle sono utilizzate se vengono mantenute dai free gates(cioè $\psi_t[i] \approx 1$). Ogni scrittura in una locazione di memoria fa aumentare il proprio valore di utilizzo fino ad un massimo di 1,il quale valore,può essere diminuito fino a 0 usando i free gates. Una volta calcolato il vettore u_t, la free list $\phi_t \in \mathbb{Z}^N$ viene definita ordinando gli indici delle locazioni di memoria in ordine crescente di utilizzo($\phi_t[1]$ è quindi la meno utilizzata). Gli allocation weighting $a_t \in \Delta_N$, che vengono utilizzati per stabilire in quali nuove locazioni scrivere sono quindi:

$$a_t[\phi_t[j]] = (1 - u_t[\phi_t[j]]) \prod_{i=1}^{j-1} u_t[\phi_t[i]]$$

Se tutte le locazioni hanno utilizzo pari ad 1, allora $a_t = 0$ e non potrà essere allocata memoria senza prima liberarla.

2.4.4.3 *Write Weighting*

Il controller può decidere di scrivere in locazioni appena allocate,oppure in locazione individuate per contenuto ma anche di non scrivere nulla. Prima di tutto viene costruito un write content weighting $c_t^w = \mathcal{C}(M_{t-1}, k_t^W, \beta_t^W)$ utilizzato una write key k_t^w ed una write strenght β_t^w. Il risultato viene quindi interpolato con i pesi di allocazione (allocation weighting) a_t definiti nell'equazione 1 per determinare i write weighting $w_t^W \in \Delta_N$:

$$w_t^W = g_t^w[g_t^a a_t + (1 - g_t^a)c_t^w] \tag{2}$$

dove $g_t^a \in [0, 1]$ è l'allocation gate responsabile dell'interpolazione e $g_t^w \in [0, 1]$ è il write gate, se esso è a 0 non viene scritto niente in memoria, questo meccanismo può essere quindi utilizzato per proteggere la memoria da modifiche non necessarie.

2.4.4.4 *Temporal memory linkage*

Il metodo di allocazione descritto finora non conserva alcuna informazione sull'ordine in cui le locazione vengono scritte. Però, come già discusso, è molte situazione è fondamentale conservare questa informazione,proprio per questo viene utilizzata una temporal link matrix $L_t \in [0, 1]^{N \times N}$ per tenere traccia delle sequenza di celle modificate(figura 16 d).
Il valore $L_t[i, j]$ rappresenta di fatto se la locazione i è stata scritta,o meno, dopo la locazione j ed ogni ogni riga e colonna di L_t definisce i pesi riferiti alle celle: $L_t[i, \cdot] \in \Delta_N$ e $L_t[\cdot, j] \in \Delta_N$ per ogni i, j, t. Per definire L_t è richiesto un "precedence weighting" $p_t \in \Delta_N$, dove gli elementi $p_t[i]$ rappresentano se la locazione i era l'ultima scritta. Viene definita ricorrentemente dalla relazione:

$$p_0 = 0$$
$$p_t = (1 - \sum_i w_t^W[i])p_{t-1} + w_t^W$$

dove w_t^W è il "write weighting" definito nell'equazione 2. Ogni volta che una locazione di memoria viene modificata, la "link matrix" viene aggiornata per rimuovere i vecchi link da e verso quella cella ed ovviamente vengono aggiunti i nuovi link . Questa logica è basata sulla seguente definizione ricorrente:

$$L_0[i,j] = 0 \forall i,j$$
$$L_t[i,i] = 0 \forall i$$
$$L_t[i,j] = (1 - w_t^W[i] - w_t^W[j])L_{t-1}[i,j] + w_t^W[i]p_{t-1[j]}$$

I link verso se stessi sono esclusi, le righe e le colonne di L_t rappresentano i pesi dei link temporali che partono o escono da una determinata cella. Dato la matrice L_t i pesi $b_t^i \in \Delta_N$ verso l'indietro ed i pesi verso l'avanti $f_t^i \in \Delta_N$ per la testina di lettura i sono definiti come:

$$f_t^i = L_t w_{t-1}^{r,i} b_t^i = L_t^T w_{t-1}^{r,i}$$

dove $w_{t-1}^{r,i}$ è l'iesimo read weighting relativo allo step precedente.

2.4.4.5 *Sparse link matrix*

Dato un certo valore K prefissato, vengono prima calcolati gli sparse vectors $\hat{w}_t^w$ e $\hat{p}_{t-1}$ ordinando w_t^W e p_{t-1} , ponendo a zero i K valori più alti e dividendo i rimanenti K per la loro somma in modo da assicurarsi che la loro somma faccia 1. La sparse link matrix può essere quindi aggiornata con :

$$\hat{L}_t[i,j] = (1 - \hat{w}_t^W[i] - \hat{w}_t^W[j])\hat{L}_{t-1}[i,j] + \hat{w}_t^W[i]\hat{p}_{t-1[j]}$$

per poi porre a zero tutti gli elementi di $\hat{L}_t$ più piccoli di $1/K$, a questo punto possono essere aggiornati anche i forward e bakcward weighting come segue:

$$f_t^i = \hat{L}_t w_{t-1}^{r,i} b_t^i = \hat{L}_t^T w_{t-1}^{r,i}$$

2.4.4.6 *Read weighting*

Ogni testina di lettura produce un content weighting $c_t^{r,i} \in \Delta_N$ utilizzando una read key $k_t^{r,i} \in \mathbb{R}^W$:

$$c_t^{r,i} = \mathcal{C}(M_t, k_t^{r,i}, \beta_t^{r,i})$$

Ogni testina riceve anche un read mode vector(che indica quali delle tre modalità di lettura utilizzare), che interpola i backward weighting b_t^i, i forward weighting f_t^i ed i content read weighting $c_t^{r,i}$ per determinare i read weighting $w_t^{r,i} \in \mathcal{S}_3$:

$$w_t^{r,i} = \pi_t^i[1]b_t^i + \pi_t^i[2]c_t^{r,i} + \pi_t^i[3]f_t^i$$

Se prevale $\pi_t^i[3]$ come read mode allora le testine percorreranno le locazione di memoria nell'ordine in cui sono state scritte,se prevale $\pi_t^i[2]$ verranno percorse in ordine inverso ed invece se prevale $\pi_t^i[1]$ i pesi verranno regolati in base al meccanismo di content-lookup già enucleato.

2.5 CONFRONTO CON LE NEURAL TURING MACHINE

Come già anticipato le DNC derivano dalle Neural Turing Machine [13],esse condividono l'architettura di base, composta da controller che interagiscono con la memoria attraverso testine, ciò che però le differenzia , è il meccanismo attraverso il quale esse si interfacciano con la memoria. Nelle NTM , il content based addressing veniva utilizzato in combinazione con il location based addressing per permettere alla rete di scorrere le locazioni di memoria attraverso i loro indici. Ciò permetteva alla rete di memorizzare e recuperare sequenze in blocchi contigui della memoria, dando però luogo ad alcuni inconvenienti. Per prima cosa le NTM non hanno nessun meccanismo che si accerti che blocchi di memoria allocata non interferiscano o si sovrappongano. Questo problema non colpisce invece le DNCs, la quale non ha questo problema in quanto tiene conto dell'utilizzo di tutte le locazioni di memoria e non necessita di memorizzare le informazioni in blocchi contigui. In secondo luogo, le NTM non offrono un meccanismo di svuotamento delle locazioni di memoria che sono già state scritte e quindi, non vi è modo per riusare queste locazioni in caso di lunghe sequenze da processare. Nelle DNC il problema è invece risolto utilizzando i free-gates per la deallocazione. Infine, nelle NTM , le informazioni sequenziali vengono mantenute solo fino a quando essa continua ad iterare lungo locazioni consecutive, infatti, appena una testina di scrittura "salta" in una parte diversa della memoria(utilizzando in content based addressing) diventa impossibile recuperare quali parti siano state scritte prima e dopo del salto. La temporal link matrix delle DNC si propone invece proprio lo scopo di risolvere questo problema, in quanto tiene traccia dell'ordine secondo il quale sono state compiute le scritture.

3

NATURAL LANGUAGE PROCESSING

3.1 INTRODUZIONE

Il linguaggio umano è quanto ci differenzia dal resto delle specie viventi, non esiste, infatti, nessun altro tipo di comunicazione fra essere dotate di tale livello di espressività e complessità. È subito lampante quanto una prerogativa così esclusiva dell'uomo, sia difficilmente riproducibile da delle macchine non senzienti. Il Natural Language Processing è quella branca dell'informatica che si propone di ridurre il gap comunicativo tra esseri umani e computer, al fine di rendere sempre più efficace la risoluzione di quei task in cui è richiesta una comprensione più o meno profonda del linguaggio naturale. Questa ampia disciplina raccoglie vari area di ricerca, in alcune di esse ci si sofferma sull'analisi sintattica del testo,in altre su quella semantica, in altre ancora sul riconoscimento semantico della lingua parlata. Cogliere le infinite sfaccettature della lingua non è affatto semplice, basti pensare ai significati multipli che può assumere un solo termine,oppure sentimenti come l'ironia, dove anche il tono con cui certe parole vengono pronunciate può andare a influire sul significato della frase. Le applicazioni di questa branca nel mondo reali sono molteplici, tra le più significative possiamo elencare:

- Riassumere una porzione di testo in modo da estrarre le nozioni più importanti ed ignorare quelle meno significative.
- Creare un bot automatico con il quale le persone possono relazionarsi in svariati ambiti.
- Generare automaticamente l'elenco delle parole chiave riferiti ad un certo testo
- Identificare il tipo di entità incluse in un testo che si sta analizzando, per determinare se una parola rappresenta un nome di persona, un luogo o anche un'organizzazione.
- Riconoscere le opinioni riferite ad un determinato argomento utilizzando la Sentiment Analysis
- Ridurre le parole alla propria radice morfologica o alla propria forma base

L'avvento dei big data, con conseguente enorme disponibilità di testi, unita al sempre maggiore sviluppo nell'ambito delle reti neurali, ha portato alla creazione di modelli linguistici sempre più performanti, che hanno sostituito in larga parte i vecchi sistemi a regole. Approfondiremo ora le varie tecniche di rappresentazione del testo.

3.2 RAPPRESENTAZIONE DEL TESTO

Nella stragrande maggioranza dei casi per adempiere a task in cui è presente il linguaggio naturale bisogna prima codificare questo in valori numeri reali, con i quali poi la rete può lavorare per raggiungere lo scopo prefissato. Esistono due macro-famiglie di approcci per rappresentare il testo,dette **Local Represention** e **Continuous representation**,vediamole ora in dettaglio presentando per ciascuna anche le tecniche più significative.

3.2.1 *Local representation*

In essa ogni concetto(che sia una parola,un n-gram o qualsiasi altra cosa) viene codificato utilizzando una sola dimensione, dando luogo ad una rappresentazione facile da comprendere,da codificare ma anche sa imparare per una rete neurale. Il problema di questa famiglia di tecniche è che non scalano adeguatamente all'aumentare della quantità dei concetti da rappresentare, ed inoltre danno luogo a modelli molto inefficienti quando si prendono in esame dati che hanno una struttura complessa e molte relazioni intrinseche. Fanno parte di questa famiglia le seguenti tecniche:

3.2.1.1 *One hot encoding*

In questo approccio ogni elemento del vettore corrisponde ad un unica e precisa parola del dizionario preso in considerazione. Poniamo ad esempio che la dimensione del vocabolario sia $\boldsymbol{N}$, l'$\boldsymbol{i-esima}$ parola del dizionario sarà rappresentata con un vettore di N elementi in cui nella $i-esima$ posizione ci sarà un 1 mentre in tutte le altre vi saranno degli zero. Quindi ad ogni posizione del vettore corrisponde una parola nel dizionario. Questo approccio, che è comunque ragionevole e funziona bene in molti casi, è pero poco efficiente in quanto a utilizzo di spazio in memoria e inoltre non fornisce nessun significato semantico alla rappresentazione,gli elementi sono infatti tutti ugualmente diversi tra loro, il che può risultare limitante se si è alla ricerca di pattern comuni nell'analizzare un testo.

3.2.1.2 *Bag of words*

Ogni documento può essere visto come un insieme di parole, le parole che compaiono ovviamente sono indicative del suo contenuto, la tec-

nica Bag of Words si basa su questo principio, raccogliendo dal testo tutto l'insieme delle parole("la borsa",bag) e contando le occorrenze di ciascuna di esse, in modo da fornire un'indicazione di massima sul contenuto. Questa tecnica ignora gli errori grammaticali e l'ordine in cui compaiono le parole, ma preserva il numero di occorrenze, rivelandosi quindi utile per classificare dei documenti in base al loro contenuto . Com'è facilmente intuibile non tutti i termini che compaiono spesso in un documento sono rilevanti e significati per la sua classificazione, per evitare, quindi, di dare eccessiva rilevanza a queste parole, viene utilizzata una funzione di peso **tf-idf**, grazie alla quale è possibile misurare l'importanza di un termine rispetto alla frequenza in cui compare in quel documento ed in una intera collezione di documenti. L'obbiettivo è quello di dare maggiore peso a quei termini che compaiono raramente all'interno di tutta la collezione di documenti ma che in quello preso in considerazione al momento sono più presenti. Il limite maggiore di questa tecnica è il fatto che ignori totalmente sia il contesto che l'ordine con cui le parole compaiono, cosa che può essere molto problematica in quanto nel linguaggio l'ordine dei termini ed il contesto in cui compaiono possono far variare completamente il significato di una frase. Inoltre i sinonimi vengono effettivamente trattati come termini diversi, dando luogo, potenzialmente, a grosse imprecisioni nella classificazione, così come che nella codifica.Pensiamo ad esempio alle frasi "Buy used cars" and "Purchase old automobiles", che hanno esattamente lo stesso significato ma verrebbero codificate con due vettore ortogonali tra loro.

3.2.1.3 *N-Gram*

Un **n-gram** è una sequenza contigua di $\boldsymbol{n}$ elementi provenienti da una certa parte di testo o di discorso. Questi elementi possono essere fonemi,sillabe, parole o anche coppie di basi a seconda del caso d'uso e delle necessità. Un n-gram model è un modello linguistico probabilistico per prevedere il prossimo elemento in una data sequenza nella forma di una catena di Markov di ordine $(n-1)$. Gli n-gram sono quindi essenzialmente un'insieme di elementi che occorrono insieme in una data finestra. Tra i problemi che affliggono gli n-grams vi è sicuramente l'assenza di una esplicita rappresentazione di dipendenze a lungo termine, il quale li rende non troppo adatti a creare un modello di rappresentazione del linguaggio naturale.

3.2.2 *Continuous representation*

In contrasto con le local representation, queste tecniche rappresentano ogni concetto utilizzando molte dimensioni, ognuna di queste dimensioni simboleggia un'aspetto di quel termine, ciò è di fondamentale importanze nel catturare le relazione fra i vari elementi presi in considerazione. Infatti queste tecniche permettono di includere già nella codifica dei termini le relazione semantiche fra di essi, idealmente ogni

dimensione utilizzata per la rappresentazione è identificativa di una particolare feature. Da questo aspetto è lampante che il numero di dimensioni che si è scelto di utilizzare va ponderato rispetto anche dal numeri e dalla composizione dei concetti che si vogliono rappresentare, se ad esempio, si sceglie un numero molto elevato di dimensioni per rappresentare una quantità abbastanza ristretta di concetti, si produrrà un modello poco accurato con cui poi sarà difficile lavorare, viceversa utilizzare poche dimensioni per codificare tanti concetti potrebbe far perdere la capacità di discernere fra elementi diversi. Tra queste tecniche è sicuramente importante citare la Latent Semantic Analysis,la Latent Dirichlet Allocation e la famiglia delle Distributed Representations,di cui fa parte Word2Vec, uno dei modelli più utilizzati nel ambito delle reti neurali, il quale verrà trattato nelle prossime sezioni.

3.2.2.1 *Latent Semantic Analysis*

La LSA è una tecnica che analizza le relazioni fra un insieme di documenti e dei termini contenuti in esse, producendo un set di concetti legati ad essi.Essa traspone i documenti in un nuovo spazio dove le dimensioni corrispondono idealmente a concetti semantici,ciascuno dato da un autovettore nello spazio originale.Questa tecnica si basa sull'ipotesi che parole con significato simile compariranno in porzioni di testo simili. Il suo funzionamento si basa sulla costruzione di una matrice, contenente il numero di occorrenze delle parole in un paragrafo (le righe rappresentano le singole parole e le colonne i vari paragrafi), a partire da una porzione di testo più vasta utilizzando la singolar value decomposition(SVD) al fine ridurre il numero di righe ma preservando la similarità tra colonne. A questo punto la matrice costruita viene utilizzata per valutare la similarità fra parole (calcolata sulla base della similarità coseno di righe), valori molto vicini a 1 rappresentano parole molto simili, mentre valori vicino allo zero simboleggiano parole molto dissimili.

3.2.2.2 *Latent Dirichlet allocation*

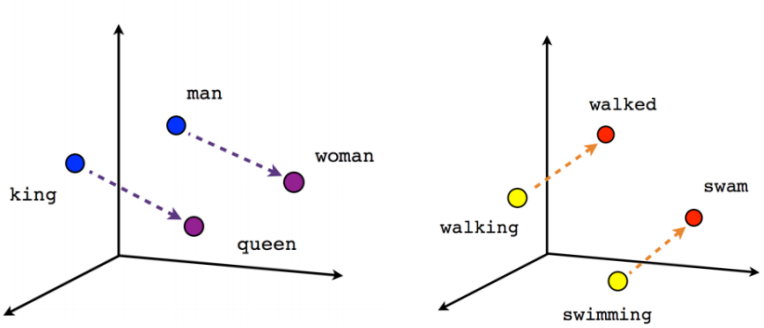
E' un modello di analisi del linguaggio naturale che permette di comprendere il significato semantico del testo analizzando la somiglianza tra la distribuzione dei termini del documento con quella di un argomento specifico (topic) o di un'entità. Il modello LDA è stato ideato da David Blei, Andrew Ng e Michael Jordan nel 2003 [4]. Più di recente, la Latent Dirichlet allocation ha conquistato una certa notorietà anche nell'ottimizzazione SEO semantica come possibile fattore di ranking del search engine Google.

3.3 WORD EMBEDDINGS

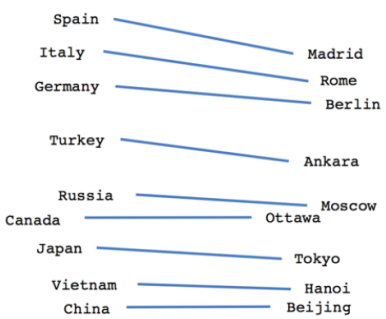
Per Word Embeddings si intendono un insieme di tecniche di modellazione nell'ambito della NLP in cui le parole di un dizionario vengono mappate in un vettore di numeri reali, in pratica una funzione $W \rightarrow \mathbb{R}^n$, con W l'insieme delle parole del dizionario ed n numero di dimensioni che si vogliono utilizzare per la rappresentazione. Come già discusso, nell'ambito delle reti neurali è richiesto l'utilizzo di input a valori reali, non sono infatti capaci di lavorare direttamente con stringhe di testo, tecniche come i word embeddings adempiono a questa esigenza ma fornendo anche due importanti vantaggi correlati:

- **Riduzione della dimensionalità** - Approcci come Bag of Words (sezione 3.2.1.1) o il one hot encoding (sezione 3.2.1.2) richiedono vettori di dimensioni pari a quella del vocabolario da rappresentare, mentre i word embeddings permettono di creare una rappresentazione vettoriale con molte meno dimensioni.

- **Similarità intrinseca nella codifica** - I vettori che rappresentano le parole preservano la similarità tra le stesse, infatti, termini simili si trovano vicini nello spazio vettoriale originato.



(a) *Maschile-Femminile* (b) *Tempi verbali*



(c) *Capitali*

Figura 18.: Vari esempi di rappresentazione nello spazio vettoriale

Un interessante caratteristica dei word vector è il fatto che possono essere manipolati matematicamente per ricavare parole presenti in determinati punti dello spazio vettoriale. Prendiamo come esempio i tre casi in figura 18.Come è evidente in figura 18a si può conoscere quale sia il femminile di Re(King) sottraendo a uomo (**Man**) il vettore Re(**King**) e aggiungendogli il vettore Donna(**Woman**), ottenendo così la parola Regina(**Queen**). Possiamo fare lo stesso ragionamento per i tempi verbali (Figura 18b) oppure le capitali degli stati (Figura 18c).

3.3.1 *Le origini*

Bengio et al. coniano il termine word embeddings nel 2003 [3], proponendo anche una architettura di rete ad hoc per la loro creazione. Il loro classic neural language model consisteva in rete feed-forward con un solo strato nascosto la quale si proponeva lo scopo di prevedere la prossima parola in una sequenza, addestrandola tramite stochastic gradient descent e backpropagation. Come mostrato in figura 19, questa architettura è strutturata in tre layers:

- **Embedding Layer**: è lo strato che genera i word embeddings moltiplicando un vettore indice con una matrice di word embedding.
- **Intermediate Layer(s)**: uno o più strati che producono una rappresentazione intermedia degli input.
- **Softmax Layer**: l'ultimo strato, il quale produce la distribuzione di probabilità sulle parole facenti parte del dizionario.

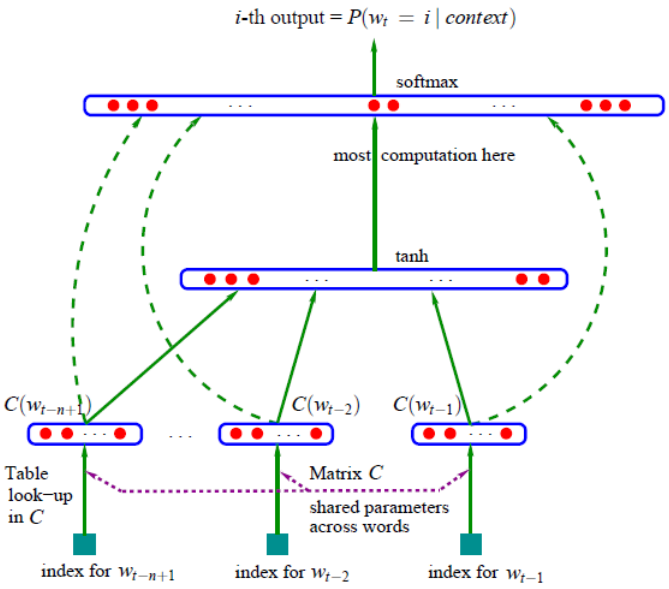


Figura 19.: Schema dell'architettura del classic neural language model

Questo modello, seppure migliorato sensibilmente negli anni a venire, ha costituito una base di partenza per tutte le tecniche proposte successivamente.
Nel 2008 furono poi Collobert e Weston [6] a mostrare le potenzialità legate all'utilizzo di un modello pre-addestrato di word embeddings in vari task, proponendo nello stesso paper anche un architettura che ha ispirato i più moderni approcci.

3.3.1.1 *Word2Vec*

Il successo e la popolarità dei word embedding è dovuto in larga parte dalla classe di modelli creata da Mikolov et al. nel 2013, detta Word2Vec. In un primo paper [24] Mikolov propose due tipologie di architetture,CBOW e Skip-gram (le quali vedremo in dettaglio nelle prossime sezioni),computazionalmente più efficaci delle precedenti architetture.Mentre in un secondo paper [25] sempre del 2013 i modelli proposti vengono migliorati con l'aggiunta di nuove strategie al fine di aumentare velocità di training ed accuratezza. Le reti di questa classe, preso in input un corpus di testi non categorizzati, produce per ogni parola un vettore che codifica intrinsecamente le proprie informazioni semantiche. Questi vettori sono molto utili per due principali ragioni:

- Risulta possibile misurare la similarità semantica tra due parole calcolando la similarità coseno tra i due corrispondenti word vectors.
- I word vectors possono essere utilizzati come feature in vari task supervisionati del NLP, come la document classification,la named entity recognition e la sentiment analysis. Le informazioni semantiche contenute in questi vettori hanno, infatti, dimostrato di incrementare sensibilmente l'accuratezza nella risoluzione di questi task.

3.3.2 *Continuous Bag of Words*

Mentre i language model classici sono in grado di basarsi solo sulle parole precedenti per fare delle previsioni, sulla base delle quali, vengono poi valutati, il modello CBOW non soffre di questa restrizione. Infatti vengono data una parola w_t che si vuole predire, vengono considerate sia le n parole precedenti a w_t sia le n successive (Figura 20) La funzione obbiettivo utilizzata dalle CBOW è la seguente:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-n}, ..., w_{t-1}, w_{t+1}, ..., w_{t+n})$$

dove θ è l'insieme dei parametri del modello, T è l'insieme dei training corpus e n è la dimensione della finestra.
Potremmo quindi riassumere questo approccio con la seguente frase: **Predire la parola a partire dal suo contesto**.

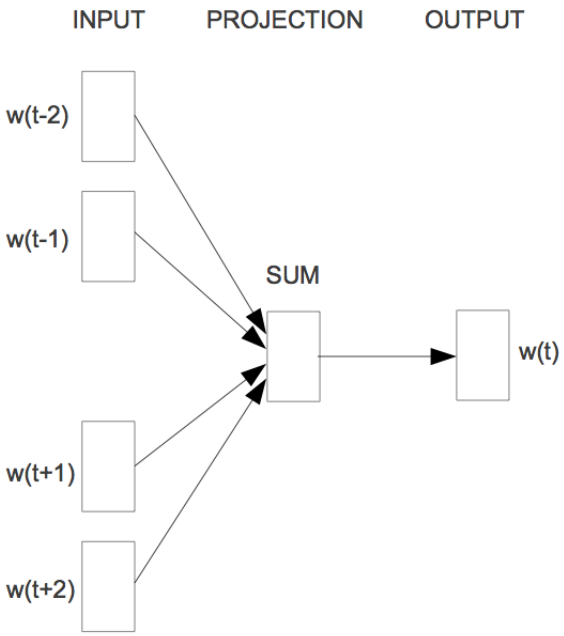


Figura 20.: Schema di funzionamento di CBOW

3.3.3 *Skip Gram*

Mentre CBOW usa i termini precedenti e successivi ad una data parola, **skip-gram** utilizza una parola per predire le n precedenti e successive (Figura 22).

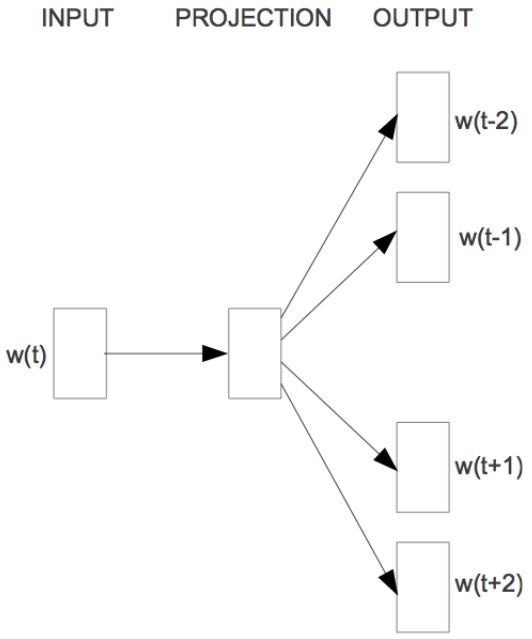


Figura 21.: Schema di funzionamento di Skip-Gram

La funzione obbiettivo degli skip-gram, considerando le n parole precedenti e successive alla parola target w_t è la seguente:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leq j\leq n,\neq 0} \log p(w_t + j|w_t)$$

Potremmo invece riassumere l'approccio degli skip-gram come: **Predire il contesto a partire da una parola**.
In definitiva, gli Skip-gram lavorano bene con piccole quantità di dati, rappresentando al meglio anche parole di rara frequenza, potendo creare una buona quantità di istanze di training anche con disponibilità limitate di dati. Mentre i CBOW sono molto più veloci da addestrare e danno una rappresentazione migliore per le parole frequenti, ma ovviamente richiedono una maggior quantità di dati per creare il modello.

3.3.4 *Glove*

Si tratta di un altro approccio alla costruzione di word embedding ideato da Pennington,Socher e Manning nel 2014 [29]. Si differenzia dagli Skip-gram in quando si tratta di un metodo "count-based" e non predittivo, basato esplicitamente sul grado di co-occorrenza delle parole, gli autori ritengono infatti, che sia proprio questo che fornisca le informazioni semantiche nella rappresentazione. Glove quindi basa il suo funzionamento sul cercare la rappresentazione più dimensionalmente ridotta possibile che possa però preservare la maggior parte della varianza dei dati con molteplici dimensionalità. La funzione obbiettivo è quindi:

$$J = \sum_{i,j=1}^{V} f(X_{i,j})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{i,j})^2$$

dove w_i e b_i sono i word vector ed i bias corrispondenti alla parola i. $\tilde{w}_j$ e $\tilde{b}_j$ sono i word vector ed i bias della parola j. $X_{i,j}$ è il numero di volte che la parola i compare nel contesto della parola j , f è una funzione che attribuisce pesi minori alle co-occorrenze rare ed a quelle frequenti.
I risultati ottenuti con GloVe sono molto simili a quelli raggiunti da Word2Vec, pur essendo Glove più facilmente parallelizzabile, rendendo possibile l'addestramento su quantità di dati ancora maggiori in tempi ragionevoli.

3.4 DOCUMENT EMBEDDINGS

Lo scopo principale dei Document Embeddings (detti anche Paragrapgh Vector) è associare, a dei documenti, delle label,può essere quindi visto come una estensione di W2V, proposta sempre nel 2014 da Mikolov [25]. Il nome paragraph vector enfatizza proprio la capacità di

questo metodo di essere applicato a porzioni di testo di lunghezza variabile, a partire da una frase per arrivare fino ad un intero documento. In questo modello vengono concatenati i paragraph vector con molti word vector provenienti dal un paragrafo e viene predetta la parola successiva a partire dal contesto. Sia i word vector che i paragraph vector vengono addestrati via stochastic gradient descent e backpropagation, i primi però,sono condivisi in tutto il modello mentre i paraghraph vector sono unici nell'insieme di tutti i paragrafi. Vengono proposti due approcci alternativi per addestrare i PV:

- Distributed Memory Model of Paragraph Vectors (PV-DM).
- Distributed Bag of Words version of Paragraph Vector (PV-DBOW)

3.4.1 *Distributed Memory Model*

In questo approccio ogni paragrafo viene mappato in un singolo vettore, rappresentato da una colonna in una matrice D,anche ogni parola viene mappata in singolo vettore,ma rappresentato da una colonna nella matrice W . A questo punto i PV e WV vengono concatenati(o ne viene calcolata la media) per predire la prossima parola in un determinato contesto

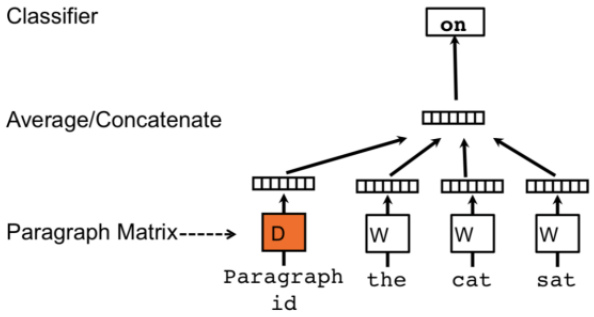


Figura 22.: Schema di funzionamento di PV-DM. Il vettore relativo al paragrafo unito al contesto di tre parole considerato, viene utilizzato per predire la terza parola. Il paragraph vector rappresenta quindi le informazioni mancanti, agendo come una memoria in cui è contenuto l'argomento del paragrafo.

3.4.2 *Distributed Bag of Words*

In contrasto con PV-DM , questo metodo ignora il contesto in input ma forza il modello a predire parole scelte casualmente nel paragrafo. In pratica, quindi, in ogni iterazione viene scelta casualmente una finestra testuale, viene selezionata una parola a caso di questa finestra dando poi origini ad un task di classificazione a partire da un dato paragraph vector. Questo approccio permette la memorizzazione di una minor quantità di dati.

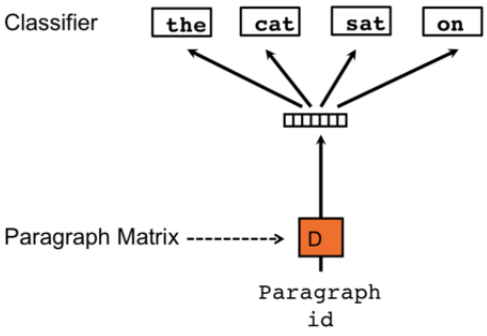


Figura 23.: Schema di funzionamento di PV-DBOW. Il paragraph vector viene utilizzato per predire le parole in una finestra testuale ristretta

In definitiva,secondo il lavoro di Mikolov et al. [25], PV-DM ottiene generalmente risultati migliori di PV-DBOW, ma la pratica è consigliata è quella di combinare insieme i due approcci, in quanto ha dimostrato di ottenere i risultati migliori.

4

CROSS-DOMAIN SENTIMENT CLASSIFICATION & TRANSFER LEARNING

4.1 SENTIMENT ANALYSIS

"*Ciò che turba gli uomini non sono le cose, ma le opinioni che essi hanno delle cose*"[10]
Già Epitteto , più di 2000 anni fa aveva già espresso quanto importanti siano le opinioni per l'essere umano. Essi ci guidano in ogni scelta che compiamo, ma assumono ancora più importanza nell'acquisto o nell'usufrutto di beni e servizi. Tanto più nella nostra era, in cui le opinioni pervadono la società attraverso i più disparati mezzi, possiamo trovare recensioni su riviste specializzate,sulle pagine dei prodotti negli E-Store, sui social network. L'incredibile supporto fornito dalla tecnologia ha fatto crescere sempre maggiore interesse sul campo dell'**Opinion Mining** o anche detta **Sentiment Analysis**, questa branca dell'informatica si propone, infatti, di automatizzare la valutazione dei sentimenti espressi in un testo scritto. Risulta senz'altro facile comprendere i motivi per i quali è interessante carpire automaticamente questi aspetti di un testo. Fornire un prodotto o un servizio che si adatti al meglio alle esigenze dei consumer non può che essere l'obbiettivo principe di qualsiasi azienda o ente. La necessità della sentiment analysis sorge prepotentemente vista l'enorme quantità di dati non categorizzati fornita dai social network, in questi contesti non vi sono, infatti, metriche di valutazione precise, come possono essere in numero di stelline su Amazon o i like e dislike su Netflix. Facciamo un esempio pratico: una importante emittente televisiva produce una serie Tv, ovviamente vi sarà una corrispondente pagina dedicata sui vari Social Network, dopo la messa in onda una grande quantità di telespettatori andrà a commentare ciò che ha visto. A questo punto l'emittente, per evitare di fornire un servizio non apprezzato,e per cui rimettere denaro, vuole capire cosa pensino gli ascoltatori, ed ecco che entra in gioco la Sentiment Analysis, come sarebbe possibile, altrimenti per uno o più esseri umani catalogare una mole così vasta di commenti prodotta sui social?(Basti pensare che solo su Facebook, su di un singolo post legato al finale di stagione di una famosa serie statunitense possiamo trovare più di 10 mila commenti). Utilizzare queste tecniche può quindi fare la differenza tra un prodotto di successo ed uno fallimentare.

4.1.1 *Sentiment Classification*

Uno degli ambiti più studiati in letteratara [28] nell'ambito della Sentiment Analysis è la **Sentiment Classification**, in esso confluiscono sia la classificazione binaria sia quella multi-classe. L'obbiettivo preposto è quello di attribuire un valore di polarità ad una porzione di testo più o meno vasta,i campi di applicazione infatti possono variare da interi documenti a singoli tweet (soltanto 140 caratteri). Nella binary sentiment classification il testo viene valutato cercando di capire se possiede una connotazione negativa o positiva, tal volta però, può essere interessante attribuire un ranking più espressivo del semplice positivo/negativo, potendo così scegliere fra più classi (ad esempio 5, come i rating su Amazon). Ovviamente la difficoltà del problema tende ad aumentare con il numero di classi, come mostrato da Pang e Lee [27] , non è semplice nemmeno per un essere umano determinare quale sia il grado di gradimento espresso da un certo testo. La comprensione di un opinione è però un problema enormemente complesso a prescindere dalla metrica con cui valutarla, basti pensare alle infinite sfaccettature del linguaggio naturale, l'ironia, l'ordine delle parole, le negazioni, sono tutte problematiche che vanno affrontate per risolvere al meglio questi task. Recentemente, il Deep Learning ha dato un forte impulso a questo campo di ricerca, grazie alla sua intrinseca capacità di modellare le feature di un testo. Nel 2013 Socher et al.[32] dimostrano come le RNN siano in grado di riconoscere e valutare correttamente anche frasi strutturare in maniera complessa, come quelle in cui compaiono negazioni.

4.2 IN-DOMAIN SENTIMENT CLASSIFICATION

I task di sentiment classification tipicamente rispecchiano un approccio in-domain,cioè il modello viene addestrato utilizzando documenti pre-categorizzati di un certo dominio per venire poi testato su altri documenti appartenenti allo stesso dominio. Supponiamo che vengano utilizzati dei dataset di prodotti Amazon, questi articoli sono divisi in categorie (libri,film,elettronica di consumo,vestiti), un modello addestrato con questo approccio viene istruito utilizzando, ad esempio, un training set composto da n recensioni della categoria 'Film' per poi essere testato su altre m reviews sempre appartenenti al dominio 'Film'. I moderni approcci riescono a raggiungere risultati molto buoni sia nella classificazione a due, che a $Z > 2$ classi , ciò è spiegato anche dal fatto che per ogni dominio vi sono alcuni termine, alcuni pattern che hanno una esplicita connotazione. Prendiamo ancora in considerazione il dominio 'Film', aggettivi come 'eccitante','sorprendente' hanno un valore nettamente positivo e li troveremo in molte recensioni nelle quali è espresso un giudizio positivo. Al contrario aggettivi come 'noioso' o 'scontato', dalla spiccata connotazione negativa, saranno molto ricor-

renti in recensioni negative. È quindi evidente che per ogni dominio ci siano certi termini o certe sequenze di termini che danno un'orientamento chiaro ed evidente alla recensione(in questo caso parliamo di recensioni,ma lo stesso discorso vale esattamente anche per qualsiasi altro tipo di testo), ma domini diversi hanno in comune gli stessi pattern per il riconoscimento delle opinioni? Non esattamente...

4.3 CROSS-DOMAIN SENTIMENT CLASSIFICATION

Ii casi in cui i domini di training e di applicazione coincidano sono soltanto una parte dei casi reali, molto spesso, infatti, vi è la necessità di applicare la sentiment classification a domini diversi da quello con cui è stato addestrato il modello. Ciò può accadere per vari motivi, ad esempio potremmo disporre di una grande quantità di dati provvisti di label appartenenti ad un certa categoria di prodotti, ma avere la necessità di scovare la polarità di recensioni di un dominio del quale non si dispongono dati utili all'addestramento. Altro caso molto comune, soprattuto nell'era dei Social Network, è l'enorme quantità di testo libero composta da commenti, post, tweet e quant'altro . Come è possibile quindi riuscire ad applicare la sentiment classification anche a questi ambiti?
La risposta sta proprio nella **Cross-Domain Sentiment Classification**, visto il volume di dati richiesto per addestrare un modello di conoscenza, non sarebbe praticabile la strada di far categorizzare questi testi da un umano. In pratica con questo approccio ci si propone di costruire un modello di conoscenza a partire da un certo dominio, per poi testarlo ed applicarlo su un dominio diverso. Riprendiamo l'esempio della sezione 4.2, un modello viene addestrato sulla base di una certa quantità di recensioni della categoria 'Film', una volta terminata questa fase vogliamo riutilizzare la conoscenza appresa applicandola su un dominio come quello dei 'Libri'. Bisogna però necessariamente considerare l'eterogeneità dei domini di applicazione, se per esempio, il dominio 'Film' e quello relativo ai 'Libri', probabilmente condividono la connotazione di molti aggettivi e pattern (eccitante e sorprendente hanno connotazione positiva in entrambi così come noioso e scontato sono aggettivi negativi in ciascuno di essi); altri domini potrebbero essere molto più distanti come significato e come quantità di termini ricorrenti in entrambi. Alcuni aggettivi potrebbero addirittura assumere connotati opposti in domini diversi, prendiamo per esempio, il caso di un film con una trama imprevedibile , il quale è sicuramente un film valutato positivamente; ora consideriamo una lavastoviglie con un funzionamento imprevedibile, il quale è sicuramente il corpo di una recensione negativa sul prodotto in questione, è evidente come ciò complichi enormemente la cross domain sentiment classification.

4.4 TRANSFER LEARNING

Proprio allo scopo di creare modelli che pur essendo addestrati su un certo dominio possano comportarsi correttamente su un dominio diverso da quello di origini, nasce il **transfer learning**(TL). Il suo obbiettivo è quello di riutilizzare la conoscenza acquisita durante l'addestramento di un modello con istanze di un certo dominio sorgente(source domain) e riutilizzarla in un dominio diverso(target domain).

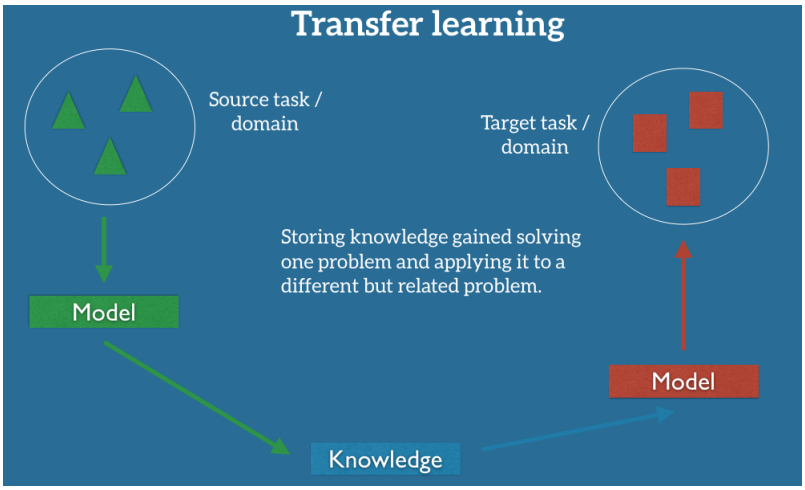


Figura 24.: Schema di applicazione del transfer learning

Si vuole quindi cercare di ricavare più conoscenza possibile da un certo dominio, per poi applicarlo in un diverso,ma seppur correlato, dominio, potendo dar vita a modelli che altrimenti non sarebbero nemmeno esistiti(in caso di mancanza di dati) oppure che sarebbero stati molto meno accurati. Nell'ambito della sentiment classification una fase di transfer learning può essere inclusa implicitamente con l'utilizzo di word vector, come già mostrato nel capitolo precedente, questa codifica fornisce una rappresentazione semantica delle parole. Questo è di fondamentale importanza in quanto la rete potrebbe essere stata addestrata su certe termini propri di un determinato dominio, una volta applicata ad un dominio diverso potrebbe quindi dover elaborare termini che non hai mai visto, come si comporterà in questi casi? Se si avesse utilizzato una codifica senza alcun significato semantico intrinseco, probabilmente la rete classificherà in modo errato la recensione in questione, con l'utilizzo di word embeddings, invece, parole dal contesto simile, si trovano vicine nello spazio di rappresentazione, ecco quindi che la rete può gestire al meglio anche termini che magari non gli sono già stati presentati, ma sono correlati ad alcuni che ha già elaborato.Nella sentiment classification, per aumentare la qualità del modello di conoscenza astratto e diminuire quindi il gap di accuratezza in cui si può incorrere passando da un dominio all'altro, è senz'altro una buona pratica quella di addestrare il modello su dati il più eterogenei

possibile fra loro, elevando così l'astrazione, che non è più concentrata su un singolo dominio ma su più domini.
Gli ambiti di applicazione del TL sono pressoché infiniti, tra le i più interessanti possiamo senz'altro considerare tutti quelli che concernono l'apprendimento da simulazioni, questi approcci permettono di risparmiare tempo e risorse in molti processi produttivi, basti pensare allo sviluppo di auto dalla guida automatica o alla costruzione di robot. In questo verso va l'interessante,e per certi versi inquietante, lavoro di Heess et al. [16], in cui viene utilizzato l'apprendimento per rinforzo al fine di insegnare ad un robot a camminare, senza però fornirlo di alcuna indicazione sul come farlo, è lampante come ciò possa dare un notevole incremento alla prototipazione della robotica, permettendo di risparmiare tempo, e soprattuto risorse.Nell'ambito del deep-learning un modo attraverso il quale si adempie a questo scopo è il riconoscimento da parte dei primi strati delle rete delle feature caratteristiche del problema generale,per cui durante la fase di transfer learning vengono eliminati gli ultimi strati(che sono invece legati al problema più specifico relativo al dominio sorgente) e sostituiti con nuovi strati addestrabili con pochissime istanze del dominio target.

5

PROGETTO E IMPLEMENTAZIONE

5.1 SCOPO DEL PROGETTO

Come già discusso nel capitolo precedente, la sentiment classification è un campo di ricerca molto in voga al momento, riuscire a rilevare le opinioni delle persone su determinati prodotti, servizi o argomenti è infatti un'attività che può rivelarsi molto redditizia. La sempre crescente mole di dati non etichettati rappresentata dai contenuti condivisi nei social network e più in generale nel Web, richiede tecniche sempre più efficienti e sempre più generali, capaci di scalare sia per numero di dati che per tipologia. Nel Capitolo 3 è stata introdotta estensivamente una nuovissima tipologia di reti neurali, le DNC (Differentiable Neural Commputer) , esse si propongono come una tecnologia capace di comportarsi egregiamente in disparati task, ma di fatto ne le DNC, ne le NTM(da cui derivano), sono state mai utilizzati in un task complesso come la Sentiment Classification In e Cross Domain.
Questa tesi si propone proprio di adattare le DNC a questi task, utilizzando dataset di riferimento, in modo da potersi confrontare con efficacia ad i risultati ottenuti in letteratura. Per la rappresentazione del testo verrà inoltre mostrata l'efficacia di tecniche come Word2Vec, di cui già discusso nella sezione 3.3.1.1.
Ci si propone, inoltre, di utilizzare tecniche di fine-tuning, allo scopo di migliorare la classificazione cross-domain.

5.2 COMPONENTI DEL PROGETTO

Passeremo ora in rassegna le macro-componenti del progetto, riservando però la descrizione del codice specifico per la sentiment classification alle prossime sezioni.

5.2.1 *Ambiente di sviluppo*

Per la creazione e l'utilizzo di questo progetto si è deciso di utilizzare il linguaggio di programmazione Python. Esso ci offre la possibilità di usufruire di svariate librerie, nello specifico sono state utilizzate:

- TensorFlow : è una libreria software open source per il calcolo

numerico che utilizza grafici di flusso di dati. I nodi del grafico rappresentano operazioni matematiche, mentre gli archi del grafo rappresentano gli array di dati multidimensionali (tensori) comunicati tra di essi. L'architettura flessibile consente di distribuire il calcolo a una o più CPU o GPU in un desktop, un server o un dispositivo mobile con una singola API.

- Gensim: è una libreria specificamente progettata per gestire grandi raccolte di testo, utilizzando lo streaming dei dati e efficienti algoritmi incrementali, che la differenzia dalla maggior parte degli altri pacchetti software scientifici che si occupano solo di batch e di elaborazione in memoria. Nel progetto è stata utilizzata per far uso dei word embeddings.

- Numpy: è una libreria che aggiunge il supporto per matrici e grandi array multidimensionali, insieme ad una vasta collezione di funzioni matematiche di alto livello per operare su questi array.

- Sonnet: è una libreria costruita basandosi su TensorFlow per la costruzione di reti neurali complesse.

Per l'installazione e la configurazione delle librerie sopra citate si rimanda all'appendice A.1

5.2.2 *Word Embeddings*

Per la codifica delle recensioni è stato utilizzato un modello Word2Vec addestrato sulle Google-News del 2012 e del 2013, un corpus di testi composto da 100 miliardi di parole. Il modello contiene le rappresentazioni 300-dimensionali di circa 3 milioni di parole. Il motivo per cui si è deciso di utilizzare dei word embeddings già addestrati va ricercato nel significativo apporto che forniscono modelli creati a partire da testi tra di loro più eterogenei possibile. Nelle fasi iniziali del progetto, infatti, si era proceduto ad addestrare un modello basandosi su circa un milione di recensioni Amazon, che pur sembrando tante, sono comunque un numero molto ristretto rispetto al corpus delle Google News, oltre ad essere piuttosto costoso (computazionalmente parlando), quel approccio faceva segnare risultati molto peggiori . Passando al modello fornito da Google(istruzioni per il download in appendice A), si è ottenuto un aumento di più del 15% in accuratezza.
Durante il progetto, però, è nata la necessità di provare a vedere come si comportasse la rete se alle recensioni in input venissero prima applicate delle tecniche di pre-processing (come stemming,lemmatization,postagging), ciò richiedeva per forza di cose dei modelli W2V ad hoc. Sono stati quindi creati modificando parzialmente la procedura descritta all'indirizzo `https://github.com/tmikolov/word2vec` , al fine di applicare quelle tecniche anche al corpus delle Google-News.

5.2.3 *Implementazione DNC*

Come anticipato, uno degli obbiettivi principali del progetto era quello di testare le potenzialità delle DNC in task più complessi, come la sentiment classification. Si è quindi utilizzata l'implementazione fornitaci da DeepMind, basata sul già citato articolo di Graves et al [14].

5.2.3.1 *Descrizione dell'architettura DNC*

Le Differentiable Neural Computer sono reti neurali ricorrenti, quindi ad ogni time step possiede uno stato composto dal contenuto attuale della memoria (ed altre informazioni ausiliarie come l'utilizzo di essa) e mappa gli input al tempo t sugli output al tempo t. Essa è implementata come una collezione di moduli RNNCore, permettendoci di inter-cambiare vari moduli con agevolezza. Essenzialmente sono tre i componenti principali:

- Il modulo *access*, dove la maggior parte della logica delle DNC si sviluppa, in quanto è dove la memoria viene scritta e letta. Ad ogni timestep un vettore viene passato in input al modulo dal controller, producendo un vettore che riflette il contenuto della memoria. Usa due ulteriori moduli *RNNCore*, cioè *TemporalLinkage*, il quale tiene traccia dell'ordine di scrittura delle locazioni di memoria e *Freeness* che invece è adibito a mantenere traccia delle locazioni scritte e non immediatamente liberate, entrambi i moduli sono definiti nel file addressing.py.

- Il modulo *Controller* gestisce l'accesso alla memoria, è costituito da una LSTM, e produce degli output che sono il risultato della concatenazione degli input con il contenuto della memoria allo step precedente.

- Il modulo *DNC* wrappa i moduli *access* e *controller* e forma l'RNNCore di base di tutta l'architettura, è definito nel file dnc.py.

In generale, la classe *DNC* può essere utilizzata come una normale cella RNN di TensorFlow e usata in task sequenziali attraverso la funzione tf.nn.dynamic_rnn.

5.3 PREPARAZIONE DEGLI INPUT

In questa sezione verranno presentate tutte le problematiche relative alla conversione delle recensioni in un formato adeguato per presentarle alla rete. Come già detto in varie sezioni, è necessario dapprima il testo in vettori di valori reali e poi organizzarli in batch e time-step.

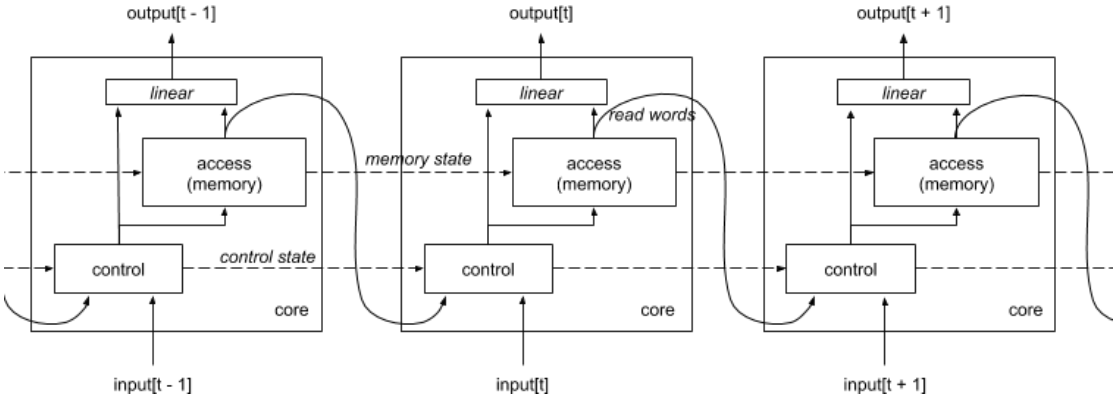


Figura 25.: Schema di funzionamento delle DNC

5.3.1 *Scelta delle recensioni*

5.3.1.1 *Classificazione a 2 classi*

Per poter addestrare e testare la rete dobbiamo prima scegliere con quale recensioni farlo, prendendo solo quelle che ci facciano rispettare i vincoli imposti dagli esperimenti. Questi vincoli ci impongono di bilanciare opportunamente le polarità fra le due classi e di considerare solo le recensioni composte da massimo 150 parole.Bisogna poi considerare solo le recensioni con score diverso da 3,come già spiegato. Per fare ciò ho implementato un'apposita funzione che permette di specificare il numero di recensioni positive e negative desiderate in fase di training e testing, fornendo inoltre, la possibilità di scegliere con una modalità random le varie recensioni. La funzione ritorna quindi due liste di tuple, rispettivamente lista delle recensioni di training e di test. Le tuple contenute nella lista contengono i testi delle recensioni e gli score.

```
def
    getBalancedReviewsAndOverall(numPositiveTraining,numNegativeTraining,numPositiveTesting,numNeg
    = False,seed=9):
    listT = []
    listTe = []
    in_file = open(fileName, "r")
    posT = 0
    negT = 0
    posTe = 0
    negTe = 0
    ranT = random.Random(seed)
    ranTe = random.Random(seed)
    while posT < numPositiveTraining or negT <
        numNegativeTraining or posTe < numPositiveTesting or
        negTe < numNegativeTesting:
        raw = in_file.readline()
        jsonLine = json.loads(raw)
        if True:
            review = jsonLine['reviewText']
```

```
num_words =
    len(gensim.utils.simple_preprocess(review))
if num_words<max_lenght and num_words >=0:
    if randomized:
        if random.randint(0, 1) == 1:
            if jsonLine['overall'] < 3.0:
                if negT < numNegativeTraining:
                    current = (jsonLine['reviewText'],
                        jsonLine['overall'])
                    listT.append(current)
                    negT += 1
                elif negTe < numNegativeTesting:
                        current =
                            (jsonLine['reviewText'],
                            jsonLine['overall'])
                        listTe.append(current)
                        negTe += 1
            else:
                if jsonLine['overall']> 3.0:
                    if posT < numPositiveTraining:
                        current =
                            (jsonLine['reviewText'],
                            jsonLine['overall'])
                        listT.append(current)
                        posT += 1
                    elif posTe <numPositiveTesting:
                        current =
                            (jsonLine['reviewText'],
                            jsonLine['overall'])
                        listTe.append(current)
                        posTe += 1
    else:
        if jsonLine['overall'] < 3.0:
            if negT < numNegativeTraining:
                current = (jsonLine['reviewText'],
                    jsonLine['overall'])
                listT.append(current)
                negT += 1
            elif negTe < numNegativeTesting:
                current = (jsonLine['reviewText'],
                    jsonLine['overall'])
                listTe.append(current)
                negTe += 1
        else:
            if jsonLine['overall'] > 3.0:
                if posT < numPositiveTraining:
                    current = (jsonLine['reviewText'],
                        jsonLine['overall'])
                    listT.append(current)
                    posT += 1
                elif posTe < numPositiveTesting:
```

```
                                current = (jsonLine['reviewText'],
                                    jsonLine['overall'])
                                listTe.append(current)
                                posTe += 1

    ranT.shuffle(listT)
    ranTe.shuffle(listTe)
    in_file.close()
    return listT,listTe
```

5.3.1.2 *Classificazione a 5 classi*

Nel caso della classificazione a 5 classi, usato negli esperimenti comprensivi di titolo per quanto riguarda le recensioni Amazon, mostreremo una funzione simile a quella della sezione precedente, ma con gli accorgimenti necessari a bilanciare egualmente le recensioni nelle 5 classi ed includendo il titolo nel testo da passare alla rete:

```
def getBalancedReviewsAndOverall(trainingInstances,
    testingInstances,
                                fileName, max_lenght,
                                    randomized=False, seed=9):
    listT = []
    listTe = []
    in_file = open(fileName, "r")
    numTrain = 0
    numTest = 0
    countTrain = [0, 0, 0, 0, 0]
    countTest = [0, 0, 0, 0, 0]
    i = 0
    ranT = random.Random(seed)
    ranTe = random.Random(seed)
    while i < trainingInstances + testingInstances:
        raw = in_file.readline()
        if raw == "":
        jsonLine = json.loads(raw)
        review = jsonLine['reviewText']
        title = jsonLine['summary']
        index = int(jsonLine['overall']) - 1
        num_words = len(gensim.utils.simple_preprocess(review))
            + len(gensim.utils.simple_preprocess(title))
        if num_words <= max_lenght:
            if randomized:
                if random.Random(seed).randint(0, 1) == 1:
                    if countTrain[index] < (trainingInstances/5):
                        countTrain[index]+=1
                        current = ((title + review),
                            jsonLine['overall'])
                        listT.append(current)
                        numTrain += 1
                        i+=1
                    elif countTest[index] < (testingInstances/5):
```

```
                    countTest[index]+=1
                    current = ((title + review),
                        jsonLine['overall'])
                    listTe.append(current)
                    numTest += 1
                    i+=1
            else:
                    if countTrain[index] < (trainingInstances/5):
                    countTrain[index]+=1
                    current = ((title + review),
                        jsonLine['overall'])
                    listT.append(current)
                    numTrain += 1
                    i+=1
                    elif countTest[index] < (testingInstances/5):
                    countTest[index]+=1
                    current = ((title + review),
                        jsonLine['overall'])
                    listTe.append(current)
                    numTest += 1
                    i+=1
    ranT.shuffle(listT)
    ranTe.shuffle(listTe)
    in_file.close()
    return listT, listTe
```

5.3.1.3 *In domain*

Se si desidera effettuare un test in domain l'implementazione creata nel progetto utilizza la funzione *getBalancedReviewsAndOverall* per ottenere testi e score delle recensioni che si desiderano passare alla rete, ottenendo una tupla di liste(training e test) e passandola alla funzione *balancedGetDataset()* la quale fornisce un generatore attraverso il quale ottenere mano a mano i vari batch di training e testing.

5.3.1.4 *Cross domain*

Se invece si desidera effettuare un test cross-domain, è necessario selezionare le recensioni da due file diversi(source e target dataset), per fare questo è stata implementata una funzione che 'wrappa' la già introdotta *getBalancedReviewsAndOverall*:

```
def getCrossDomainReviewsAndOverall(numTraining,numTesting,
fileTraining,fileTesting,max_lenght,randomized= False,seed=9):
    listT,_ =
        getBalancedReviewsAndOverall(numTraining,numTraining,0,0,
                            fileTraining,max_lenght,randomized,seed)
```

5.3.2 *Codifica Word2Vec*

Come abbiamo più volte sottolineato, di fondamentale importanza è utilizzare i word embeddings per codificare le recensioni, posto ciò che è stato considerato nella sezione 5.2.2 bisogna ora utilizzare il modello pre addestrato per mappare in valori reali le varie parole.
Per fare questo bisogna innanzitutto caricare il modello desiderato con la seguente istruzione:

```
loadedModel =
    gensim.models.KeyedVectors.load_word2vec_format(modelName,binary=True)
```

Esso verrà quindi utilizzato da una apposita funzione che preso un testo ed il già citato modello, produrra in output la lista di parole che compongono quel testo, codificata con W2V.

```
def getW2vRepresentation(text,loadedModel):
    countDeleted = 0
    toReturn = []
    for element in gensim.utils.simple_preprocess(text):
        try:
            toReturn.append(loadedModel[element].tolist())
        except KeyError:
            countDeleted+=1
    return toReturn
```

5.3.3 *Riduzione in vettori multidimensionali*

Come vedremo nelle sezioni relative all'implementazione del grafo computazionale relativo al task da svolgere, la cella DNC per permettere il completo unrolling degli input fa utilizzo della funzione fornita TensorFlow , tf.nn.dynamic_rnn . Per utilizzare questa funzione, bisogna presentare gli input un formato ben preciso, cioè un tensore di forma [batch_size, max_time, ...] . In breve, visto che nelle prossime sezioni verranno approfonditi, il batch_size e max_time sono rispettivamente il numero di recensioni che vengono passate contemporaneamente alla rete e il numero massimo di step temporali con cui fare unrolling della rete. Nel nostro caso, le recensioni vengono divise per parole , ed ognuna di esse viene presentata in un passo temporale, la rete, in questo modo, può cogliere la sequenzialità del testo.
Per addestrare e testare la rete, però, non bastano solo gli input, ma servono anche delle label per confrontare i risultati prodotti con quelli attesi e delle maschere per valutare correttamente gli errori della rete durante l'addestramento.

5.3.3.1 *Divisione in batch*

La divisione in batch viene effettuata nella funzione *balancedGetDataset()*, ad essa vengono passati come parametri i training e testing set

(ricavati nelle modalità descritte nella sezione precedente),il percorso al modello W2V da utilizzare ed i vari flag riguardanti i parametri scelti per il task. Di questi flag è importante citare, visto il loro utilizzo in questa funzione, quelli che indicano proprio la dimensione dei batch da costruire ed il numero massimo di step temporali. Ecco le parti di codice relative al batching:

```
batched_input = []
batched_label = []
batched_mask = []
i = 0
num_batch = 0
batch_completed = False
if len(allLines[0])>0:
    while i < (len(allLines[0])+1):
        if not batch_completed:
            #
            #Qui ci sarebbero le parti relative alla
                costruzione dei vettori di input,label e mask
            #le omettiamo per brevita
            #
            else:
                reshaped = np.reshape(batched_input,
                    newshape=[batch_size, max_lenght,
                    wordSize])
                labels =
                    np.reshape(batched_label,newshape=[batch_size,max_lenght,2])
                masks =
                    np.reshape(batched_mask,newshape=[batch_size,max_lenght])
                toAppend = (reshaped,labels,masks)

                num_batch+=1
                if (len(allLines[0])//batch_size) ==
                    num_batch:
                    batch_completed = True
                batched_input = []
                batched_label = []
                batched_mask = []
                yield toAppend
```

5.3.3.2 *Costruzione tensori di input*

Come già introdotto gli input devono avere la forma [batch_size, max_time, ...] , per arrivare a questo formato partendo da del semplice testo bisogna seguire tre step:

- Codifica W2V :

```
data =
    getW2vRepresentation(allLines[1][i][0],
    loadedModel)
```

- Aggiunta di padding per arrivare alla lunghezza max_time(150):

```
if len(data)>0:
    padded = np.lib.pad(data,
        ((max_lenght-len(data),0),(0,0)),
        'constant', constant_values=(0))
else:
    padded = np.zeros(shape=[max_lenght,
        wordSize])
batched_input.append(padded)
```

- Reshaping una volta raggiunta la dimensione del batch prefissata:

```
reshaped = np.reshape(batched_input,
    newshape=[batch_size, max_lenght,
    wordSize])
```

5.3.3.3 *Costruzione tensori di label*

Le label sono fondamentali per confrontare l'output prodotto dalla rete e quello desiderato. Nel caso di task di classificazione, come il nostro, si utilizza una codifica di tipo One-Hot, cioè un vettore lungo N con N numero di classi, dove vi è un 1 in corrispondenza della classe che si vuole rappresentare e nel resto delle posizioni degli zero. Nel caso della classificazione a due classi è stato posto che la prima posizione sia identificativa della classe positiva e la seconda di quella negativa. In generale quindi una label viene prodotta per ogni time-step di ogni batch:

```
index = int(allLines[0][i][1]) - 1
for j in range(0, max_lenght):
    np.put(label[j], index, 1.0)
batched_label.append(label)
```

5.3.3.4 *Costruzione tensori di maschere*

La maschera è un vettore di dimensione batch_size $\times$ max_lenght , viene utilizzata durante il calcolo dell'errore relativo ai singoli batch, permettendoci di considerare gli output prodotti dalla rete solo negli unrolling corrispondenti a delle parole della recensione e non considerando quelli prodotti in corrispondenza di padding. La maschera applicata fornisce peso via via crescente mano a mano che si procede verso le ultime parole della recensione. Viene formata così:

```
mask = np.zeros(max_lenght)
np.put(mask, np.arange(max_lenght -
    len(data), max_lenght), 1.0)
batched_mask.append(mask)
```

```
masks = np.reshape(batched_mask,
    newshape=[batch_size, max_lenght])
```

5.4 APPLICAZIONE DELLE DNC ALLA SENTIMENT-CLASSIFICATION

In questa sezione verrà mostrato come le DNC sono state utilizzate per svolgere il task oggetto di studio di questa tesi. È importante, però,prima di iniziare a descrivere la composizione dell'architettura creta,introdurre brevemente il funzionamento del framework Tensorflow.

5.4.1 *Funzionamento di TensorFlow*

Per capire meglio le logiche alla base della costruzione del modello di rete neurale adibita alla sentiment-classification, è consigliabile conoscere alcune basilari,ma indispensabili, nozioni sulle dinamiche interne di TensorFlow.
Come intuibile dal nome, l'unità centrale di questo framework è il **tensore**, cioè un insieme di valori primitivi organizzati dentro un array di n dimensioni.
Il funzionamento di TensorFlow può essere essenzialmente decomposto in due fasi:

- Costruzione del grafo computazionale
- Esecuzione del grafo computazionale

Un **grafo computazionale** consiste in una serie di operazione TensorFlow organizzate in un grafo a nodi. Ognuno di questi nodi prende in input zero o più tensori e ne produce uno in input. Un nodo esprime quindi un operazione,anche costante, che viene utilizzata ogni volta che viene eseguito il grafo. L'esecuzione del grafo e quindi la valutazione degli output prodotti, avviene durante una **session** , la parte runtime di TensorFlow. Mostriamo ora un banale esempio del suo utilizzo,creando un grafo che somma due numeri:

- Creazione del grafo(Figura 26):

  ```
  a = tf.placeholder(tf.float32)
  b = tf.placeholder(tf.float32)
  adder_node = a + b
  ```

  I placeholder sono una sorta di variabili a cui può essere attribuito un valore a tempo di esecuzione.

- Esecuzione del grafo:

  ```
  sess = tf.Session()
  print(sess.run(adder_node, {a: 4.5, b: 4.5}))
  ```

Verrà quindi stampato in output il valore 9.

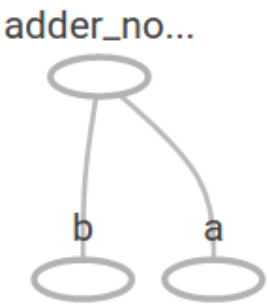


Figura 26.: Grafo computazionale per la somma di due numeri

5.4.2 *Creazione della rete*

La rete utilizzata è composta essenzialmente da una sola cella DNC, replicata però nel tempo grazie alla funzione tf.nn.dynamic_rnn() fornita da TensorFlow. Essa ci permette di srotolare nel tempo gli input, per essere più chiari e precisi facciamo l'esempio proprio del nostro caso di studio. Una recensione è composta da diverse parole, ognuna delle quali contribuisce alla polarità totale del testo, il linguaggio umano ha però una natura sequenziale, è importante quindi, che l'ordine con cui vengano presentate le parole alla rete sia quello corretto. Nel nostro caso le recensioni sono composte al massimo da 150 parole, ciò significa che quando applichiamo l'unrolling della rete la cella di base(la DNC) viene replicata in 150 passi temporali, in ognuno dei quali viene passata una nuova parola, ma anche lo stato della cella proveniente dallo step precedente, al termine degli step vieni quindi valutato l'output prodotto e retropropagato l'errore in modo da addestrare la rete. In definitiva, vale questa considerazione, fatta nell'articolo Deep Learning del 2015 da LeCunn et al.: [21] Le RNN, una volta srotolate nel tempo, possono essere viste come reti feed-forward molto profonde in cui ogni strato condivide gli stessi pesi.
Questo comportamento è descritto dalla funzione *run_model(input_sequence, output_size)* la quale prendendo come parametri gli input da elaborare e la dimensione dell'output da produrre, ritorna la sequenza di output ricavata. Nella prima parte della funzione viene creata ed inizializzata la cella DNC, secondo dei parametri esplicitati nel file di configurazione:

```
access_config = {
    "memory_size": FLAGS.memory_size,
    "word_size": FLAGS.word_size,
    "num_reads": FLAGS.num_read_heads,
    "num_writes": FLAGS.num_write_heads,
}
```

```
controller_config = {
    "hidden_size": FLAGS.hidden_size,
}
clip_value = FLAGS.clip_value
#Creo la cella dnc
dnc_core = dnc.DNC(access_config, controller_config,
    output_size, clip_value)
initial_state = dnc_core.initial_state(FLAGS.batch_size)
```

Mentre nella seconda parte viene eseguito l'unrolling della rete, sulla base della cella DNC appena creata e sugli input da analizzare:

```
output_sequence, _ = tf.nn.dynamic_rnn(
    cell=dnc_core,
    inputs=input_sequence,
    time_major=False,
    initial_state=initial_state)

return output_sequence
```

L'output della funzione tf.nn.dynamic_rnn è rappresentato da una coppia (outputs, state) dove outputs sarà un tensore di forma [batch_size, max_time, cell.output_size] e state un tensore di forma [batch_size, cell.state_size]. La funzione run_model() verrà utilizzata in uno specifico nodo del grafo, prendendo come input il contenuto di un PlaceHolder, riempito durante i cicli di addestramento e test con i dati forniti dall'apposito generatore (Sezione 5.3.3.1).

```
x = tf.placeholder(tf.float32, [FLAGS.batch_size, max_lenght,
    dictSize])

output_logits = run_model(x, FLAGS.num_classes)
```

5.4.3 *Calcolo degli errori sull'output*

Passo fondamentale nell'addestramento di qualsiasi rete neurale è calcolare quale sia la differenza tra l'output prodotto dalla rete e quello atteso, in modo da andare ad agire sui vari pesi in modo da minimizzare questa discrepanza. È quindi evidente che bisogna scegliere un approccio per fare questo calcolo che rifletta il più possibile il reale errore della rete, in modo da agire più efficacemente possibile nell'aggiustamento dei pesi attuato dall'ottimizzatore. Come spiegato nella sezioni precedente l'output ci viene fornito come un tensore di forma [batch_size, max_time, cell.output_size] , allo stesso modo, le label con cui confrontarsi vengono fornite tramite un tensore della stessa forma. A questo punto per calcolare l'errore, dobbiamo utilizzare una formula che evidenzi la differenza tra i due tensori. Perfettamente adatta a questo scopo è la funzione di costo denominata cross entropy(vedi sezione 1.3.5) unita però alla funzione softmax, fornita da TensorFlow

con il metodo *tf.nn.softmax_cross_entropy_with_logits*, il quale misura l'errore probabilistico in una classificazione discreta dove le classi sono mutualmente esclusive.

```
cross = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
    logits=output_logits)
```

Lo scopo di applicare anche la funzione softmax agli input è quello di determinare la probabilità,che secondo la rete,ogni recensione ha di appartenere ad una classe piuttosto che ad un altra. Per esempio,dovendo svolgere una classificazione a due classe, la rete elabora una recensione e produce in output per ognuna di esse un vettore di lunghezza due, dentro il quale ci sono due valori reali che esprimono la scelta fatta dalle rete, applicando softmax, questi valori vengono utilizzati per calcolare la probabilità con cui la rete ha previsto l'appartenenza alle classi. Poniamo il caso di una recensione che la rete ha deciso di reputare positiva, nel vettore prodotto,dopo aver applicato softmax, ci sarà un valore X_1 compreso tra 0 e 1.0 nella posizione 0 e un valore $X_2 < X_1$ nella posizione 1 . Il risultato di questa operazione è quindi un tensore di forma [batch_size,max_time] il quale contiene tutte le discrepanze tra gli output prodotti in ogni timestep di ogni batch e le label corrispondenti. Tra tutti gli errori calcolati ce ne sono però molti corrispondenti a timestep in cui la rete non ha elaborato parole di una recensione, ma solo il padding adibito a dare la corretta forma alla matrice di input, è quindi evidente che gli output prodotti in quei timestep non vadano considerati, a questo proposito è stata utilizzata una maschera (la cui formazione è descritta nella sezione 5.3.3.4). Molto spesso, viene utilizzata una maschera che considera solo l'output prodotto alla fine della sequenza introdotta, il che ha perfettamente senso in quanto effettivamente la polarità di una frase va considerata al termine della stessa. Però, nella fase di calcolo di addestramento e di minimizzazione quindi, della cross_entropy, si è rivelata,almeno empiricamente, una scelta vincente , quella di attribuire peso via via maggiore agli output prodotti dalla rete in corrispondenza di parole sempre più vicine alla fine della frase. Il concetto alla base di questo meccanismo è che seppur la polarità di una recensione vada valutata alla fine della sequenza di parole, ognuna di queste contribuisce alla connotazione della frase, e il loro contributo vada considerato in misura maggiore se sono verso la fine della recensione. Empiricamente, nei test, è stato verificato che questo approccio ha portato ad incrementare l'accuratezza di alcuni punti percentuali.
A prescindere dal tipo di maschera utilizzato, bisogna comunque calcolare gli errori nella classificazione per ogni batch:

```
#Calcolo l'errore relativo ai singoli batch, applicando una
    maschera che considera gli output prodotti dalla rete
#solo negli unrolling corrispondenti a delle parole della
    recensione e non considerando quelli prodotti in
```

```
    corrispondenza
#di padding. La maschera applicata fornisce peso via via
    crescente mano a mano che si procede verso le ultime
    parole
#della recensione.
batch_error = tf.reduce_sum(cross * mask, 1)
```

A questo punto viene calcolata la media degli errori compiuti tra tutte le recensioni del batch, questo valore sarà poi quello da minimizzare grazie all'azione dell'ottimizzatore.

```
#Media degli errori dei singoli batch
total_error = tf.reduce_mean(batch_error)
```

5.4.4 *Ottimizzatore*

Il vero responsabile dell'addestramento della rete è l'ottimizzatore, il quale computa il gradiente della funzione di costo e lo applica alle variabili al fine di migliorare la classificazione. In questo caso è stato utilizzato RMSProp [33], che si è dimostrata essere un ottimo algoritmo, perfettamente adatto ad i nostri scopi. Per istanziare l'ottimizzatore bisogna per prima cosa definire il valore del learning rate, nel nostro caso, avendo deciso di farlo variare nel corso delle epoche, è contenuto in un placeholder.

```
learning_rate = tf.placeholder(tf.float32)

optimizer = tf.train.RMSPropOptimizer(
    learning_rate, epsilon=FLAGS.optimizer_epsilon)
```

Ed altrettanto necessariamente, vanno indicate quali sono le variabili su cui deve agire RMSProp ed il valore da minimizzare.

```
trainable_variables = tf.trainable_variables()
grads, _ = tf.clip_by_global_norm(
    tf.gradients(total_error, trainable_variables),
        FLAGS.max_grad_norm)
```

A questo punto, per ogni passo di addestramento verrà applicato l'ottimizzatore alle variabili precedentemente specificate:

```
#Passo da eseguire per addestrare la rete
train_step = optimizer.apply_gradients(
    zip(grads, trainable_variables), global_step=global_step)
```

Quindi durante l'esecuzione della sessione,ogni qual volta viene richiesta la computazione del nodo train_step, verrà eseguita tutta la parte di grafo relativa all'addestramento.

5.4.5 *Addestramento*

Parte fondamentale del progetto è quella relativa all'addestramento della rete, in cui le recensioni vengono presentate a gruppi di *batch_size* elementi, ottenuti da una apposita funzione generatrice.
L'addestramento viene ripetuto per più epoche, in ognuna delle quali vengono presentate tutte le recensioni che si sono scelte all'inizio con le apposite funzioni, cambiando però, in ogni epoca, l'ordine con cui vengono presentate alla rete. Inoltre, in ogni epoca il learning rate iniziale può essere diminuito progressivamente fino a raggiungere un valore specificabile nella configurazione:

```
delta = (FLAGS.learning_rate-FLAGS.final_learning_rate)/
                        (FLAGS.num_epochs-1)


newLearningRate = FLAGS.learning_rate - delta * (epochs)
```

Come già spiegato, per istruire la rete bisogna eseguire un train_step , che però, per svolgere le sue mansioni ha bisogno che tutti i placeholder relativi ad input,label e maschere siano riempiti:

```
_, act_accuracy, entropy = sess.run([train_step, accuracy,
    total_error],
                                        {x: (datasetTrain[0]),
                                         y_: (datasetTrain[1]),
                                         mask:
                                             (datasetTrain[2]),
                                         learning_rate:
                                             newLearningRate})
```

In questo passaggio è stata anche ricavata l'accuratezza con la quale la rete ha classificato le varie recensioni, esse viene calcolata confrontando l'output prodotto all'ultimo parola di ogni recensione del batch e le corrispondenti label. Come già discusso, la rete fornisce una distribuzione di probabilità sulle due classi attraverso la funzione softmax, bisognerà quindi vedere in quale indice del vettore emesso c'è la probabilità maggiore,per poi verificare se corrisponde effettivamente alla label. Il numero di recensioni del batch classificate correttamente in raffronto al totale di esse, ci darà l'accuratezza:

```
#Si ricava la polarita' che la rete ha indicato all'ultima
    parola di ogni recensione,
prediction = tf.arg_max(output_logits[:,max_lenght-1], 1)
#Si ricava la polarita' indicata dalle label
expected = tf.arg_max(y_[:,max_lenght-1], 1)
#Si ricava quante predizioni della polarita' sono state fatte
    correttamente
correct_prediction = tf.equal(prediction, expected)
#Accuratezza ottenuta in questo batch
accuracy = tf.reduce_mean(tf.cast(correct_prediction,
    tf.float32))
```

Al termine di ogni epoca di addestramento viene quindi effettuata una fase di test.

5.4.6 *Test*

Nella fase di test, compiuta al termine di ogni epoca, vengono passate alla rete recensioni che non ha mai visto prima e di cui deve prevedere autonomamente la polarità, nella pratica dell'implementazione, perciò, non viene eseguito il classico passo train_step , ma soltanto la parte di grafo computazionale che è responsabile della produzione degli output e del calcolo dell'accuratezza sugli stessi. Le modalità di ottenimento dei tensori utilizzati ed il calcolo dell'accuratezza sono le stesse definite nei capitoli precedenti. Al termine di ogni sessione di test viene riportata quale accuratezza sia stata raggiunta in media su tutte le istanze di test.

5.4.7 *Salvataggio del modello*

Al fine di poter riutilizzare una rete già addestrata, è stata fornita la possibilità di salvare su file lo stato interno della stessa. Nel file di configurazione viene scelto ogni quanti iterazioni di training salvare lo stato, tipicamente viene scelto un valore che coincide con il numero di passi in una epoca. Nella pratica si è utilizzata una utility fornita da TensorFlow detta CheckpointSaverHook:

```
#Oggetto per salvare lo stato della rete.
saver = tf.train.Saver()

#Impostazione dei parametri relativi al salvataggio dello
    stato della rete.
if FLAGS.checkpoint_interval > 0:
  hooks = [
      tf.train.CheckpointSaverHook(
          checkpoint_dir=FLAGS.checkpoint_dir,
          save_steps=FLAGS.checkpoint_interval,
          saver=saver)
  ]
else:
```

5.5 CONFIGURAZIONE DEI PARAMETRI

Il progetto sviluppato è stato organizzato e ideato al fine di incapsulare tutti i parametri dell'esperimento in un unico file di configurazione, questo sia per favorirne l'utilizzo da parte di terzi che non conoscono l'implementazione e sia per lanciare molteplici esperimenti specificando per ognuno quale file di configurazione utilizzare. La lista completa dei parametri, unita alle istruzione per il lancio di un esperimento, è

disponibile in Appendice B. In questa sezione passeremo in rassegna i parametri più importanti e se ne discuteranno gli impatti sui risultati.

5.5.1 *Batch size*

Il numero di recensioni utilizzate in un passo di training è il valore che prende nome di **batch size**. Sceglierlo correttamente è fondamentale per addestrare un buon modello di classificazione, batch troppo piccoli portano la rete ad andare in over-fitting, una condizione per la quale il modello che la rete ha costruito è troppo legato alle istanze di training che gli sono state presentate. Al contrario, scegliere un batch size troppo grande potrebbe portare ad un tempo di convergenza troppo elevato o addirittura non riuscire mai a creare un modello performante visto le troppe recensioni presentate, non riuscendo quindi a cogliere uno schema di risoluzione appropriato. D'altro canto, la scelta del batch size influisce molto anche sull'effettivo tempo di calcolo richiesto, visto che rappresenta la misura di ogni quanto vengonoa aggiornati i pesi della rete, un'operazione sicuramente molto costosa. Aggiornamenti frequenti portano a training lenti,mentre batch size grandi danno luogo ad addestramenti più veloci, ovviamente si intende il tempo impiegato per completare un epoca,non il tempo per produrre risultati ottimali.
È quindi importante trovare un giusto trade-off tra questi aspetti, ed essendo questa scelta fortemente influenzata sia dal task che si deve svolgere e sia dalle caratteristiche dei dati utilizzati, è sempre bene anteporre una fase in cui si cercano i valori migliori alla fase di test vera e propria . Nel nostro caso, il valore con il quale si sono ottenuti risultati migliori è diverso per i dataset Amazon e lo Stanford Sentiment Treebank, infatti, per il primo tipo di dataset si è utilizzato un batch size di 60, mentre per il secondo i risultati migliori sono stati ottenuti con un batch size di 300. Tale differenza va probabilmente ricercata nella differente natura delle recensioni contenute nei due dataset, nel caso di Amazon, ci troviamo di fronte a testi lunghi mediamente 100 parole, mentre nel Treebank la lunghezza massima è di 50. Ipotizziamo che essendo le recensioni del secondo tipo, composte da meno parole, la rete ha bisogno di più esempi per poter generalizzare un modello efficace.

5.5.2 *Learning rate*

La misura di quanto aggiornare i pesi in ogni iterazione è data dal **learning rate**. Scegliere con oculatezza questo parametro è di fondamentale importanza, con un learning rate troppo alto, infatti, la rete potrebbe aggiustarsi in modo troppo brusco facendo variare il gradiente troppo rapidamente, non riuscendo quindi ad individuare un punto di minimo. Al contrario, se questo parametro assume un valore troppo basso la rete potrebbe non convergere mai, o con una velocità troppo

bassa. Per scegliere questo numero non esiste una formula adatta a tutte le situazioni, dipende molto dal tipo di task a cui si vuole addestrare la rete,dal tipo e dalla quantità di dati di training,dal tipo di rete utilizzata... Pratica comune è però diminuire il learning rate durante l'arco dell'addestramento, in quanto si vuole mentre all'inizio del training si è più lontani dal punto di minimo,essendo quindi necessari "salti" più ampi per trovare il punto,col passare del tempo le variazioni veramente necessarie sono minori. Nel nostro caso si è scelto, nella maggior parte dei casi, di utilizzare un learning rate abbastanza alto alla prima epoca (10^{-2}) per poi ridurlo progressivamente, tipicamente fino a 10^{-3} . Nel caso di training con un numero di recensioni elevatissimo (più di 3 milioni), il learning rate utilizzato è stato sensibilmente più basso, $(5 \cdot 10^{-4})$, vista la mole di dati disponibili all'addestramento è giusto che la rete cambi poco per volta i pesi.

5.5.3 *Numero epoche*

Ogni volta che l'intero training set viene fatto elaborare dalla rete, si è compiuta una **epoca** di addestramento, in ognuna delle quali, i dati vengono presentati in ordine diverso al fine di contribuire a creare un modello il più efficace possibile. Pratica comune è compiere più epoche di training, ed al termine di ognuna, eseguire la fase di test, questo permette di osservare l'evoluzione del modello previsionale nel corso delle epoche. In ogni esperimento per le prime n epoche osserveremo un miglioramento dei risultati in fase di test,ottenendo un valore di accuratezza Xn ma da una certa epoca $i > n$ in poi noteremo dei risultati peggiori. Questo significa che la rete è andata in over-fitting, in quanto la continua presentazione degli stessi input(anche se in ordine diverso) la ha portata a creare un modello meno generale. È perciò importante scegliere un numero non troppo basso di epoche,in quanto non si può conoscere a priori in quale epoca la rete farà segnare il miglior risultato. Empiricamente, nel nostro task e con la nostra implementazione, si è notato che il numero di epoche dopo le quali la rete va in over-fitting, è inversamente proporzionale al numero di iterazioni per epoca, partendo dalle circa 13 epoche necessarie ad ottenere il risultato migliore con 1600 recensioni di addestramento, passando alle 5-6 per 16000 recensioni e per concludere con le 2 epoche necessarie con 3 milioni e 600 mila recensioni.

5.5.4 *Parametri specifici DNC*

Essendo l'architettura delle DNC implementata da DeepMind molto flessibile, è possibile impostare vari parametri ad essi relativa. Vediamoli:

- Hidden size : è il numero di strati nascosti nel controller LSTM. Tipicamente è stato utilizzato il valore 128, come negli esempi

forniti da DeepMind. Negli esperimenti più grandi è stato però necessario aumentarlo fino a 256 per ottenere risultati migliori.

- Memory size : è il numero di locazioni di memoria da utilizzare nella DNC. Per la gran parte degli esperimenti la memoria è stata composta di 64 celle.
- Word size : è la dimensione delle singole locazioni di memoria, il valore che si è mostrato ottenere i risultati migliori è stato 64, valori più grandi hanno peggiorato le performance della rete.
- Write heads : è il numero di testine di scrittura. Ne è stata sempre utilizzata una.
- Read heads : è il numero di testine di lettura. Solitamente ne vengono utilizzate 4, in alcuni esperimenti ha portato beneficio utilizzare solo una.

6

ESPERIMENTI ED ANALISI

In questo capitolo verranno presentati vari esperimenti svolti applicando le DNC ad un task complesso come la sentiment classification, sia in domain che cross domain sui dataset Amazon ed anche sullo Stanfor Sentiment Treebank. I vari test mostreranno l'efficacia di questa tipologia di rete neurale messa a confronto con i risultati ottenuti in letteratura, tal volta superando anche lo stato dell'arte relativo ad alcune configurazioni.

6.1 SETUP

6.1.1 *Dataset*

Nel progetto sviluppato sono stati utilizzate due tipologie di dataset contenenti recensioni, entrambi forniti dall'università di Stanford [22].

6.1.1.1 *Amazon Dataset*

Questa tipologia di dataset comprende svariate recensioni di prodotti Amazon, divise per categorie. Sono essenzialmente file JSON composti da milioni di righe, ognuna corrispondente ad una recensione, essendo dati reali, contengono tutta una serie di informazioni caratteristiche dei prodotti Amazon, di tutti questi campi abbiamo però preso in considerazione soltanto:

- **reviewText** : è il campo che contiene il testo della recensione, la lunghezza può variare da 2 parole a 1000 , ma statisticamente la media è di 100 parole.
- **summary** : è il titolo della recensione, spesso molto indicativo sulla polarità della stessa. È stato utilizzato solo in alcuni esperimenti.
- **overall** : è il campo in cui è espresso il punteggio attribuito dall'utente al determinato prodotto. I valori assumibili sono 1,2,3,4,5 .

Di tutte le categorie messe a disposizione abbiamo scelto di utilizzare le 4 più corpose in termini di quantità di recensioni:

- Books (B)- 19.3 GB
- Electronics (E) - 5.3 GB
- Movies and TV (M) - 3.9 GB
- Clothing,Shoes and Jewelry (J) - 2.9 GB

Le istruzioni per scaricare questi dataset sono in Appendice A.2.
Per la sentiment classification a due classi in domain e cross domain sono stati utilizzate tutte e 4 le categorie, considerando, però , le recensioni con score 1 e 2 come di polarità negativa, mentre quelle con 4 e 5 positiva, si è scelto per cui , di non tenere in considerazione le recensioni con valutazione 3 (neutrale) perché di fatto più complesse e non utili nell'ambito della classificazione a due classi.
Per gli esperimenti in cui si è considerato anche il titolo delle recensioni, al fine di poter effettuare un confronto con le stesse condizioni espresse nel paper di LeCunn e Zhang [35], si sono dovuti comporre due dataset speciali. Un esperimento prevedeva l'utilizzo di 3 milioni e 600 mila recensioni per la fase di training e 400 mila per la fase di test, equamente distribuite fra le due polarità (esperimento a 2 classi), mentre l'altro prevedeva l'uso di 3 milioni di recensioni per la fase di training e 650 mila per la fase di test, equamente distribuite tra le cinque classi (esperimento a 5 classi).
La composizione dei dataset per le due prove è stata quindi la seguente:

- Esperimento a 2 classi: si è creato un dataset apposito,composto da 4 milioni di recensioni, bilanciando equamente le polarità e componendolo nel seguente modo: 1 milione e 120 mila dal dataset B, 880 mila da M, 1 milione da J ed un milione da E. Il motivo per cui sono presenti più recensioni della categoria Books e meno di Movies va ricercato nel fatto che in 'M' non vi erano 500 mila recensioni di polarità negativa.Quindi per mantenere equilibrata la proporzione tra le due polarità si è deciso di prendere 440 mila recensioni positive ed altrettante negative da Movies e le 60 mila mancanti per ciascuna polarità sono state prese da Book, che oltre ad essere il dataset più grande è soprattutto quello che semanticamente si avvicina di più a Movies.

- Esperimento a 5 classi: è stato creato un dataset apposito composto da 3 milioni e 650 mila recensioni,bilanciando equamente la distribuzione nelle 5 classi (730 mila review per classe) e componendolo prendendo 1.216.667 recensioni dai dataset B,J,E. In questo caso non sono state prese in considerazioni recensioni proveniente dal dataset Movies in quanto non vi erano sufficienti valutazioni di punteggio 2 per mantenere equilibrata la distribuzione fra classi e categorie. Vista la generalità dell'esperimento, si è ritenuto che non comprendere le recensioni della categoria Movies non avrebbe inficiato sulla espressività del modello addestrato.

Per le prove in domain e cross domain sui dataset Amazon sono state utilizzate tre diverse partizione tra insieme di training e di test, preservando però sempre la proporzione tra i due gruppi (80%-20%) sul totale dei dati presi in considerazione:

- Configurazione 1600-400 - Small
- Configurazione 16000-4000 - Medium
- Configurazione 80000-20000 - Large

In ognuna di queste configurazioni è sempre presente una equa distribuzione fra recensioni positive e negative.

6.1.1.2 *Stanford Sentiment Treebank*

Per confrontarsi con i risultati ottenuti da Socher et al [32] si è dovuto far uso di questo particolare dataset, che contiene nella sua forma originale 11,855 frasi singole estratte da recensioni di film. Il corpus reso disponibile non era però utilizzabile direttamente in quanto la tecnica utilizzata nel loro paper si differenzia dalla nostra, è stato perciò necessario fare alcune join tra file diversi al fine di ricomporre l'esatta configurazione. Il dataset è diviso in tre parti:

- Training set, composto da 8544 recensioni.
- Validation set, composto da 1101 recensioni.
- Test set, composto da 2210 recensioni.

I dataset pronti all'uso sono scaricabili seguendo le istruzioni in Appendice A.2

6.1.2 *Configurazione esperimenti*

Si precisa che ogni risultato riportato è il valore medio di tre esperimenti eseguiti con la stessa identica configurazione, cambiando però, ogni volta il seed per la generazione di numeri random. Questo seed agisce, infatti, sulla scelta delle recensioni da utilizzare per addestramento e test, permettendo di ottenere un risultato più indicativo sull'effettiva capacità di classificazione della rete. Non bisogna trascurare che non tutte le recensioni sono di egual difficoltà, il che potrebbe portare a risultati anche molto diversi tra un run ed un altro, sopratutto quando si utilizzano poche recensioni di test. Proprio alla luce di questa considerazione, negli esperimenti relativi al Very Large Dataset non si sono eseguiti più run in quanto l'enorme mole di recensioni di test(400 mila e 650 mila) permette già di ottenere risultati che non risentono di grosse variazioni causate dalla variabilità delle recensioni.

6.1.2.1 *Parametri della rete*

Verranno elencate le varie configurazioni di parametri propri della rete neurale utilizzati negli esperimenti

- **In-Domain e Cross-Domain ,dataset Small,Medium e Large:** Hidden Size:128 - Memory Size: 64 - Word Size: 64 - Read Heads : 4 - Write Heads : 1

- **Classificazione a 2 e 5 classi,Dataset Very Large (3600k-400k) :** Hidden Size:256 - Memory Size: 64 - Word Size: 64 - Read Heads : 1 - Write Heads : 1 - Learning Rate iniziale: 5×10^{-4}, Learning Rate finale: 5×10^{-4}

- **Classificazione a 2 classi, Treebank:** Hidden Size:256 - Memory Size: 128 - Word Size: 128 - Read Heads : 8 - Write Heads : 2 - Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

- **Classificazione a 5 classi, Treebank:** Hidden Size:128 - Memory Size: 64 - Word Size: 64 - Read Heads : 4 - Write Heads : 1 - Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

6.1.2.2 *Parametri di addestramento*

- **In-Domain e Cross-Domain ,dataset Small**: Numero epoche: 15 - Batch size: 60 - Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

- **In-Domain e Cross-Domain ,dataset Medium**: Numero epoche: 10 - Batch size: 60 - Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

- **In-Domain e Cross-Domain ,dataset Large**: Numero epoche: 5 - Batch size: 60 - Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

- **Classificazione a 2 e 5 classi,Dataset Very Large (3600k-400k) :** Numero epoche: 15 - Batch size: 60Learning Rate iniziale: 5×10^{-4}, Learning Rate finale: 5×10^{-4}

- **Classificazione a 2 classi , Treebank:** Numero epoche: 15 - Batch size: 30 -Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

- **Classificazione a 5 , Treebank:** Numero epoche: 15 - Batch size: 300 - Learning Rate iniziale: 10^{-2}, Learning Rate finale: 10^{-3}

6.2 RISULTATI IN-DOMAIN

Per prima cosa sono stati compiuti gli esperimenti In-Domain relativi ai dataset Amazon (B,M,E,J), i risultati ottenuti sono stati confrontati con quelli prodotti con vari approcci sperimentati da Domeniconi et al. [8]. Nello specifico le tecniche a confronto sono : Naive Bayes (NB), Paragraph Vector (PV) e Markov Chain(MC) nella versione descritta in (Domeniconi et al. [7]).
I risultati nella tabella 1 mostrato le accuratezze ottenute nella classificazione a 2 classi sui domini Books (B), Movies (M), Electronics (E) e Clothing- Shoes-Jewelry (J). Come si evince dai dati riportati, nella configurazione più piccola, con 1600 recensioni di training , le DNC$(81,78\%)$ in media si comportano molto meglio di Naive Bayes$(74,94\%)$ e dei Paragraph Vector$(75,44\%)$, mentre ottengono una accuratezza leggermente inferiori rispetto alle Markov Chain $(83,61\%)$. Questi dati sono perfettamente interpretabili nell'ottica che essendo PV e DNC tecniche di Deep Learning hanno bisogno di una grande quantità di dati per funzionare al meglio, infatti come si può notare nella figura 27 , entrambi gli approcci migliorano sensibilmente i propri risultati nei dataset più grandi. Le DNC però si dimostrano un approccio sensibilmente migliore rispetto a tutti gli altri, in quanto ottengono una accuratezza media del 90.08% già con 16000 recensioni di training, superando di gran lunga sia le MC, che ottengono comunque buoni risultati, ed i PV. Nel set più grande, infine,le DNC migliorano ancora la propria accuratezza media, facendo segnare un $91,24\%$, risultando ancora come miglior tecnica fra quelle presentate. È importante notare la curva descritta dall'andamento della accuratezza nelle varie configurazioni, infatti si evince che l'incremento maggiore è in corrispondenza del passaggio da 1600 a 16000 recensioni di training, portando ad un aumento del $8,3\%$, mentre nel passaggio da 16 mila a 80 mila, l'incremento è solo del $1,16\%$, dimostrando che questo tipo di reti riescono ad apprendere piuttosto in fretta anche su task complessi come la sentiment classification

6.3 RISULTATI CROSS-DOMAIN

Sono stati poi effettuati gli esperimenti Cross-Domain relativi ai dataset Amazon (B,M,E,J) in tutte le combinazioni possibili tra dataset $X \rightarrow Y, X \neq Y$. I risultati ottenuti sono stati confrontati con quelli prodotti con vari approcci sperimentati da Domeniconi et al. [8]. Nello specifico le tecniche a confronto sono : Naive Bayes (NB), Paragraph Vector (PV) e Markov Chain(MC) nella versione descritta in (Domeniconi et al. [7]).
L'obbiettivo principale di queste prove era verificare se le DNC fossero capaci di generalizzare un modello di classificazione indipendente dal dominio di applicazione, senza alcun meccanismo di transfer learning

	1.6k-0.4k				16k-4k				80k-20k			
Domain	MC	NB	PV	**DNC**	MC	NB	PV	**DNC**	MC	NB	PV	**DNC**
M → M	79,25%	78,25%	67,25%	84,28%	81,90%	63,30%	75,40%	90,47%	83,84%	66,36%	84,74%	91,20%
J → J	91,23%	74,50%	79,75%	80,48%	82,43%	70,10%	74,87%	90,24%	80,23%	71,30%	84,11%	93,20%
E → E	92,00%	79,50%	79,25%	78,57%	80,72%	69,53%	80,15%	90,07%	84.41%	76,41%	85,61%	91,98%
B → B	71,97%	67,50%	75,50%	85,71%	83,76%	72,23%	80,08%	89,55%	86,98%	73,81%	85,25%	88,59%
Average	**83,61%**	**74,94%**	**75,44%**	**81,78%**	**82,20%**	**68,79%**	**77,63%**	**90,08%**	**83,87%**	**71,97%**	**84,93%**	**91,24%**

Tabella 1.: Risultati ottenuti nella sentiment classification a 2 classi con Naive Bayes (NB), Paragraph Vector (PV), Markov Chain(MC) e DNC. Nk-Mk indicano che l'esperimento è stato svolto utilizzando N * 1000 istanze di training e M * 1000 instanze di test.$X \rightarrow Y$ significa che il modello è stato addestrato sul dominio X e testato sul dominio Y

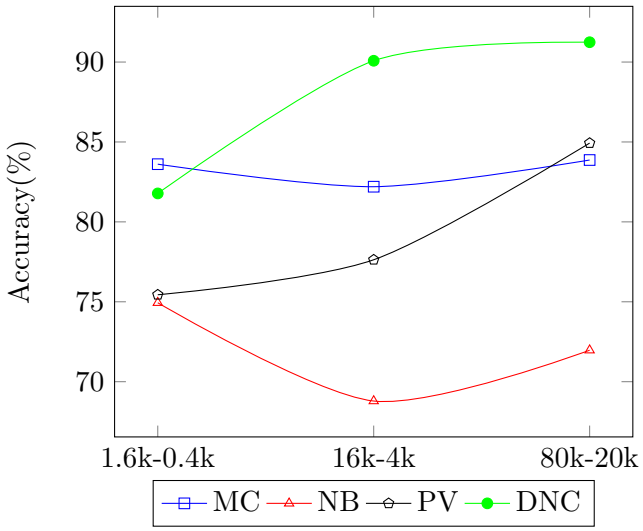


Figura 27.: Andamento dell'accuratezza nelle tre configurazioni di dataset su un task di sentiment classification a 2 classi. Nk-Mk indicano che l'esperimento è stato svolto utilizzando N * 1000 istanze di training e M * 1000 instanze di test.

esplicito in una prima fase, e con una fase di fine tuning secondariamente. I risultati ottenuti sono stati valutati non solo in termini di accuratezza, ma anche di due metriche particolari per sperimentazioni cross-domain definite in (Glorot et al., 2011), cioè:

- **Transfer Loss**, definito come $t(S,T) = e(S,T) - e_b(T,T)$, dove S è il dominio sorgente,T quello destinazione ed e è l'errore nella classificazione
- **Transfer Ratio**, definito come $Q = \frac{1}{n} \sum_{(S,T),S \neq T} \frac{e(S,T)}{e_b(T,T)}$ con n numero di coppie (S,T) con $S \neq T$

I risultati ottenuti sono riportati nella tabella 4 ,messi a confronto con quelli ottenuti dalle altre tecniche. A differenza del caso in-domain le DNC risultano essere anche nella configurazione di training più piccola, la tecnica che fa registrare i migliori risultati $(82,79\%)$, contro il $71,49\%$ di Naive Bayes, il $73,25\%$ dei Paragraph Vector e il $71,49\%$ delle Markov Chain. Il che è sorprendente se si considera il fatto che non è stato applicato nessun meccanismo di transfer learning esplicito, l'ottimo risultato ottenuto può essere spiegato dall'utilizzo della codifica Word2Vec per rappresentare le parole nelle recensioni, essa, come già argomentato nei precedenti capitoli, fornisce una codifica semantica, grazie alla quale parole correlate sono vicine nello spazio di rappresentazione. Probabilmente, quindi, pur dovendo la rete elaborare recensioni di domini diversi, che talvolta possono essere anche intrinsecamente molto lontani (basti pensare a libri(B) ed apparecchi elettronici (E), riesce a costruire un modello di astrazione molto valido, dando luogo a risultati ottimi. Questa tesi è avvalorata dal fatto che anche i risultati

ottenuti con i Paragraph Vector sono migliori di quelli delle Markov Chain, che invece negli esperimenti in domain si era rivelata una tecnica più efficace, confermando che la rappresentazione vettoriale non supervisionata è un'ottima base per affrontare task cross-domain. Anche in questo caso le DNC migliorano i propri risultati all'aumentare dei dati a disposizione per il training ottenendo accuratezze veramente alte, sopratutto se comparate alle altre tecniche (Figura 28).

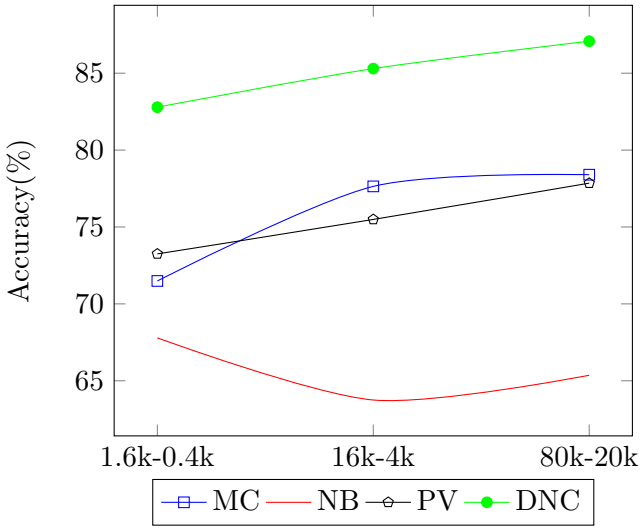


Figura 28.: Andamento dell'accuratezza media nelle tre configurazioni di dataset su un task di sentiment classification cross domain a 2 classi. Nk-Mk indicano che l'esperimento è stato svolto utilizzando N * 1000 istanze di training e M * 1000 instanze di test.

6.3.1 *Transfer Loss e Transfer Ratio*

Per poter valutare al meglio la capacità della tecnica analizzata di generalizzare un modello adatto anche a domini diversi da quello con cui è stato compiuto l'addestramento, può essere utile esprimere i risultati in termini di Transfer Loss e Transfer Ratio (definiti all'inizio di questa sezione). Queste due metriche ci danno la misura in cui la rete è riuscita a trasferire conoscenza da un dominio all'altro, è infatti importante considerare quale sia la differenza tra utilizzare su un dominio X un modello addestrato sullo stesso X oppure su un diverso dominio Y. Come evidenziato dalla tabella 2, addestrando la rete con sole 1600 recensioni, il modello risultante è molto generale, in alcuni casi, addirittura, si ottengono risultati migliori che nelle sperimentazioni in domain. Il valore $-1,05\%$ sta proprio a significare che in media si è ottenuto una diminuzione degli errori di classificazione del $1,05\%$. Nelle configurazione di dataset più grandi, invece, la percentuale di Transfer Loss è più rilevante, ciò è probabilmente dovuto al fatto che

la rete, venendo addestrata con un numero rilevante di recensioni di un dominio, si adatta in misura maggiore ad esso. In ogni caso, la perdita si attesta intorno al $4,76\%$ addestrando la rete con 16000 recensioni e al $4,16\%$ con 80000, mostrando la bontà e la scalabilità dell'approccio scelto.

	Transfer Loss		
Domain(s)	1.6k-0.4k	16k-4k	80k-20k
$M \rightarrow J$	$-0,95\%$	$6,74\%$	$7,40\%$
$M \rightarrow E$	$5,48\%$	$3,24\%$	$5,20\%$
$M \rightarrow B$	$-2,62\%$	$1,19\%$	$-1,35\%$
$B \rightarrow J$	$-4,28\%$	$9,00\%$	$8,24\%$
$B \rightarrow E$	$-0,23\%$	$3,50\%$	$8,58\%$
$B \rightarrow M$	$-0,01\%$	$3,78\%$	$1,45\%$
$E \rightarrow J$	$-7,14\%$	$3,22\%$	$4,45\%$
$E \rightarrow M$	$-3,33\%$	$8,28\%$	$3,38\%$
$E \rightarrow B$	$6,19\%$	$3,24\%$	$4,30\%$
$J \rightarrow M$	$0,47\%$	$7,09\%$	$3,99\%$
$J \rightarrow E$	$-2,14\%$	$2,66\%$	$1,43\%$
$J \rightarrow B$	$2,14\%$	$5,14\%$	$2,85\%$
Average	$-1,05\%$	$4,76\%$	$4,16\%$

Tabella 2.: Transfer loss ottenuti nella sentiment classification cross-domain a 2 classi le DNC. Nk-Mk indicano che l'esperimento è stato svolto utilizzando N * 1000 istanze di training e M * 1000 instanze di test.$X \rightarrow Y$ significa che il modello è stato addestrato sul dominio X e testato sul dominio Y

Una metrica forse più indicativa, visto la sua minori sensibilità a grandi variazioni degli errori in-domain , è il **Transfer Ratio**. Le considerazioni già fatte valgono anche per questi dati (Tabella 3), infatti possiamo notare che questo valore aumenta proporzionalmente con la dimensione del dataset, e quindi dell'accuratezza in-domain. Essendo il risultato del rapporto tra l'errore compiuto cross-domain e quello in domain, è bene che questo valore sia il più basso possibile, ovviamente però , va anche considerata l'effettiva accuratezza ottenuta, nel nostro caso, molto alta.

	Transfer Ratio		
Domain(s)	1.6k-0.4k	16k-4k	80k-20k
$* \rightarrow M$	$0,939\%$	$1,670\%$	$1,334\%$
$* \rightarrow J$	$0,789\%$	$1,648\%$	$1,985\%$
$* \rightarrow E$	$1,048\%$	$1,316\%$	$1,632\%$
$* \rightarrow B$	$1,134\%$	$1,304\%$	$1,169\%$
Average	$0,977\%$	$1,485\%$	$1,53\%$

Tabella 3.: Transfer ratio ottenuti nella sentiment classification cross-domain a 2 classi le DNC. Nk-Mk indicano che l'esperimento è stato svolto utilizzando N * 1000 istanze di training e M * 1000 instanze di test.$* \rightarrow Y$ simboleggia la media dei risultati ottenuti da modelli addestrati sui domini $X \in B, J, M, E, X \neq Y$ e testati sul dominio Y

6.3.2 *Fine Tuning*

Al fine di migliorare i risultati ottenuti nella classificazione cross-domain si è scelto di utilizzare anche una tecnica di transfer learning esplicita come il fine tuning.
Il **fine tuning** consiste nel ri-addestrare parzialmente una rete, precedentemente istruita con un gran numero di recensioni appartenenti ad un certo dominio X,con una mole abbastanza contenuta di recensioni del dominio target Y.Questo moderno approccio ben si adatta a scenari di utilizzo del mondo reale, poniamo,ad esempio, che si disponga di un gran numero di recensioni già dotate di label di un certo dominio X e poche (o zero)recensioni appartenenti ad un altro dominio Y, ma si vuole comunque creare un modello per classificare recensioni di Y. Secondo questo approccio verrebbe prima addestrala la rete su X e poi raffinato il suo comportamento con un numero molto limitato(tanto da render praticabile anche la scelta di catalogarle da un umano) di recensioni di Y.
Gli esperimenti condotti avevano proprio il fine di verificare il grado di efficacia di questa soluzione. Sono stati perciò presi in esame tutti i modelli salvati precedentemente per ogni dominio B,J,E,M ottenuti con un training di 80000 recensioni, riaddestrandoli con altre 1000 o 5000 del dominio target e testando quindi il modello risultante su 20000 recensioni del dominio target.
I risultati riportati nella tabella 5 mostrano l'incremento di accuratezza ottenuto con i due approcci di fine tuning. Seppur non sono stati ottenuti incrementi drastici nell'accuratezza, i dati dimostrano comunque l'effettiva utilità di questa pratica, nell'ordine di 1 punto percentuale nel caso di ri-addestramento con 5000 istanze.

Domain(s)	Accuracy Previous	1k	5k
$M \rightarrow J$	85,80%	87,79%	87,89%
$M \rightarrow E$	86,78%	85,12%	87,41%
$M \rightarrow B$	89,94%	89,24%	88,43%
$B \rightarrow J$	84,96%	86,11%	85,53%
$B \rightarrow E$	83,40%	84,86%	88,81%
$B \rightarrow M$	89,75%	89,55%	90,13%
$E \rightarrow J$	88,75%	89,46%	90,85%
$E \rightarrow M$	87,82%	87,10%	87,01%
$E \rightarrow B$	84,29%	86,41%	88,15%
$J \rightarrow M$	87,21%	86,46%	86,71%
$J \rightarrow E$	90,55%	89,93%	90,00%
$J \rightarrow B$	85,74%	85,45%	85,90%
Average	87,08%	87,29%	88,07%

Tabella 5.: Confronto tra risultati ottenuti nella sentiment classification cross-domain a 2 classi senza fine tuning(previous),con fine tuning di 1000 istanze e con fine tuning di 5000 istanze.$X \rightarrow Y$ significa che il modello è stato addestrato principalmente sul dominio X e testato sul dominio Y

L'efficacia del fine tuning è però lampante se confrontata con l'addestramento ex-novo di una rete con le sole istanze che sono state invece utilizzare per riaddestrare un modello costruito su un altro dominio.I test effettuati con la rete addestrata ex-novo (baseline) sono in pratica esperimenti in domain a 1000 e 5000 istanze di training.
In tabella 6 sono riportati i risultati di questo confronto, ed è lampante come il fine tuning sia efficace rispetto ad una classificazione in domain con solo 1000 o 5000 istanze del dominio target.

Domain(s)	1k Baseline	1k Fine Tuning	5k Baseline	5k Fine Tuning
$* \rightarrow M$	73,14%	87,70%	81,35%	87,95%
$* \rightarrow J$	73,43%	87,79%	82,05%	88,09%
$* \rightarrow E$	69,60%	86,64%	80,99%	88,74%
$* \rightarrow B$	75,51%	87,03%	81,10%	87,49%
Average	73,42%	87,29%	81,37%	88,07%

Tabella 6.: Confronto tra risultati ottenuti addestrando una rete ex novo con 1000 o 5000 istanze e con 1000 o 5000 istanze per il fine tuning di una rete già addestrata su un diverso dominio .$* \rightarrow Y$ simboleggia la media dei risultati ottenuti da modelli addestrati sui domini $X \in B, J, M, E, X \neq Y$ e testati sul dominio Y

6.4 LARGE AMAZON DATASET

Visti gli ottimi risultati ottenuti negli esperimenti precedenti, dimostrando, quindi la bontà delle DNC nella sentiment classification, si è ritenuto interessante procedere con ulteriori esperimenti atti a scoprire se esse si comportino meglio di altre tecniche ritenute lo stato dell'arte nello svolgimento di questo task. Il paper di [35], 2015 è ritenuto lo stato dell'arte nella sentiment classification su dataset Amazon. Al fine di replicare gli esperimenti condotti in esso, si è proceduto ad addestrare la rete con 3 milioni e 600 mila recensioni (con ognuna delle polarità equi-presente) per la classificazione a 2 classi e con 3 milioni per la classificazione a 5 classi, testandole rispettivamente con 400 mila e 650 mila recensioni. I dettagli sulla composizione dei dataset utilizzati possono essere trovati alla sezione 6.1.1.1, è però importante ricordare che le recensioni utilizzate provengono da tutte e 4 le categorie a disposizione, rendendo particolarmente significativo questo esperimento. Inoltre, come anche nel paper citato, è stato utilizzato anche il campo relativo al titolo delle recensioni, preponendolo al corpo di essa. I risultati sono stati ottenuti utilizzando le configurazione riportate nella sezione 6.1.2.

Classes	**DNC**	Conv. Th. [35]
2	$\mathbf{95,51}\%$	$95,07\%$
5	$\mathbf{61,45}\%$	$59,70\%$

Tabella 7.: Risultati ottenuti nel Very Large dataset da DNC e dal paper [35] con delle Small Character-level Convolutional Networks(nel caso a 5 classi) e Large Character-level Convolutional Networks (nel caso a 2 classi)

Come evidenziato nella tabella 7 la DNC supera lo stato dell'arte (ottenuto con una CNN) di uno $0,44\%$ nel caso a 2 classi e di un $1,75\%$ in quello a 5. Il risultato ottenuto è veramente incoraggiante se si considera che le recensioni provengono da 4 domini diversi, perciò il modello risultante è capace di riuscire a determinare la polarità di istanze anche molto diverse tra loro.

6.5 STANFORD SENTIMENT TREEBANK

Per completare l'opera di confronto con gli approcci famosi in letteratura si è deciso di utilizzare anche lo Stanford Sentiment Treebank, maggiori dettagli su di esso possono essere trovati alla sezione 6.1.1.2, mentre per quanto riguarda le configurazioni utilizzate, si veda la sezione 6.1.2. Per confrontare i risultati ottenuti si sono presi in considerazione gli esperimenti condotti nel paper di Socher et al.,2013, il quale non rappresenta più lo stato dell'arte ma rimane comunque un buon

articolo con cui raffrontarsi.
Nella tabella 8 sono riportati i risultati ottenuti dalla DNC e dal miglior metodo esposto nel paper [32], cioè le Recursive Neural Tensor Network (RNTN).Per gli esperimenti a due classi vengono usate 6920 recensioni per il training e 1821 per il test,venendo scartate le review con score uguale a 3.Mentre per la classificazione a 5 classi ne vengono utilizzate 8544 per il training e 2210 per il test.
Si evince che, mentre le RNTN ottengono un risultato migliore, seppure di poco, nella classificazione a 2 classi, le DNC superano di $1,08\%$ le RNTN in quella a 5 classi. Bisogna tener conto, però, anche del numero abbastanza limitato di istanze di training in entrambi gli esperimenti, visto che le DNC sembrano necessitare di un cospicuo numero di dati per funzionare al meglio.Questi dati sono, in ogni caso, un ulteriore conferma di quanto le DNC siano performanti in vari task e dataset, bisogna infatti considerare la differente natura delle recensioni facenti parte dello Stanford Sentiment Treebank rispetto a quelle provenienti da Amazon. Le prime sono tipicamente molto più corte, ed ovviamente non è sempre facile determinare la polarità di recensioni con pochi termini. Mentre le RNTN costituiscono

Classes	**DNC**	RNTN [32]
2	$85,22\%$	$\mathbf{85,40}\%$
5	$\mathbf{46,78}\%$	$45,70\%$

Tabella 8.: Risultati ottenuti nello Stanford Sentiment Treebank DNC e dalla tecnica RNTN (Socher et al.)

6.6 ANALISI DEI TEMPI

Per fornire un quadro completo relativo agli esperimenti sostenuti è bene fornire anche le tempistiche con cui si sono svolti gli addestramenti, da sempre infatti, il tempo impiegato per addestrare reti neurali di tipo Deep non è trascurabile. Il tempo impiegato varia molto in base all'hardware che si ha a disposizione, in questo caso gli esperimenti sono stati svolti su 5 differenti macchine vista la mole di prove da effettuare. Riporteremo i tempi impiegati dalle 2 macchine più performanti,una dotata di una scheda grafica Nvidia GTX 1050 TI e 16 GB di Ram, l'altra equipaggiata di una Nvidia TITAN XP e 32 GB di Ram, per eseguire un epoca di addestramento.

Dataset	Epoche	GTX 1050 TI	TITAN XP
1.6k-0.4k	15	3 Minuti	NA
16k-4k	10	12 Minuti	NA
80k-20k	5	1 Ora	NA
3600k-400k	3	52 Ore	26 Ore
3000k-650k	3	NA	24 Ore

Tabella 9.: Tabella riassuntiva dei tempi necessari all'esecuzione di un'epoca di addestramento con i vari dataset. Con NA si intende che quella tipologia di esperimenti non sono stati svolti su quella macchina

Domain	1.6k-0.4k MC	NB	PV	**DNC**	16k-4k MC	NB	PV	**DNC**	80k-20k MC	NB	PV	**DNC**
M → J	81,95%	63,25%	82,25%	81,43%	74,86%	62,03%	73,45%	83,50%	78,58%	67,26%	76,96%	85,80%
M → E	68,18%	65,25%	71,75%	73,09%	77,10%	66,35%	75,32%	90,24%	72,87%	63,94%	74,79%	86,78%
M → B	67,75%	65,25%	74,45%	88,33%	86,05%	76,55%	85,55%	88,36%	83,811%	69,05%	85,11%	89,94%
B → J	79,70%	63,75%	73,25%	84,76%	71,83%	53,38%	70,60%	81,24%	75,99%	54,92%	74,87%	84,96%
B → E	69,29%	67,50%	70,75%	78,80%	71,22%	55,85%	67,27%	89,55%	74,05%	58,77%	73,24%	83,40%
B → M	70,85%	70,50%	66,75%	84,29%	61,15%	61,15%	80,25%	86,69%	79,01%	61,99%	81,97%	89,75%
E → J	74,25%	72,00%	82,75%	87,62%	80,49%	63,23%	79,47%	87,02%	81,91%	73,09%	80,80%	88,75%
E → M	56,75%	70,75%	71,50%	87,61%	76,20%	64,43%	76,17%	82,19%	77,15%	66,06%	76,86%	87,82%
E → B	54,00%	64,50%	74,00%	79,52%	80,10%	65,43%	78,80%	86,31%	79,19%	66,15%	76,87%	84,29%
J → M	81,25%	72,50%	74,25%	83,81%	74,30%	63,25%	70,77%	83,38%	77,93%	66,27%	76,07%	87,21%
J → E	80,60%	75,75%	76,50%	80,71%	79,76%	68,90%	78,55%	87,41%	81,79%	70,88%	80,08%	90,55%
J → B	75,25%	62,50%	66,25%	83,57%	80,55%	64,48%	69,62%	84,41%	78,55%	65,88%	76,53%	85,74%
Average	**71,49%**	**67,79%**	**73,25%**	**82,79%**	**77,64%**	**63,75%**	**75,49%**	**85,30%**	**78,40%**	**65,36%**	**77,85%**	**87,08%**

Tabella 4.: Risultati ottenuti nella sentiment classification cross-domain a 2 classi con Naive Bayes (NB), Paragraph Vector (PV), Markov Chain(MC) e DNC. Nk-Mk indicano che l'esperimento è stato svolto utilizzando N * 1000 istanze di training e M * 1000 instanze di test.$X \rightarrow Y$ significa che il modello è stato addestrato sul dominio X e testato sul dominio Y

7

CONCLUSIONI

L'obbiettivo di questa tesi era di testare una tecnica recentissima come le DNC, la quale gode di grande interesse nel panorama scientifico , viste le sue innovative caratteristiche, in un task complesso come la Sentiment Classification In-Domain e Cross-Domain. Per adempiere al meglio questo task, una rete neurale deve riuscire a generalizzare,producendo un modello capace di distinguere le feature di una recensione che ne contrassegnano la sua connotazione positiva o negativa, ed in taluni casi, addirittura indicare quale è il grado di soddisfazione dell'utente in una scala da 1 a 5. Il tipo di rete analizzata è risultata essere perfettamente in grado di svolgere questi compiti, sia quando chiamata a classificare recensioni dello stesso dominio per cui era stata addestrata(In-Domain), sia quando utilizzata per classificare istanze domini diversi(Cross-Domain) da quello di training. Vista l'eterogeneità dei domini scelti(basti pensare,per esempio, a quanto possano essere diverse delle recensioni che hanno come oggetto di valutazione un elettrodomestico da quelle relative ad un libro), il risultato delle prove cross-domain, sopratutto di quelle senza una fase di transfer learning esplicita, è indicativo della bontà di questa tecnica, già dimostratasi efficace in diversi altri task [14].
Le DNC si sono dimostrate migliori delle altre tecniche prese in considerazione in quasi tutti gli esperimenti fatti. Nello specifico, per le prove in-domain (tabella 1),hanno fatto registrare accuratezze migliori di tutte le altre tecniche sia con l'utilizzo di 16000 recensioni che con 80000, mentre utilizzando soltanto 1600 recensioni sono seconde solo alle Markov Chain.
Ma è nell'ambito del cross-domain che sono stati ottenuti i risultati migliori, mostrando quanto sia efficace l'utilizzo combinato delle DNC e della codifica Word2Vector, la quale come già dimostrato in altri lavori, contribuisce fortemente a creare modelli adatti a domini eterogenei. I risultati (tabella 4) ottenuti con la tecnica in questione sono migliori in tutte e tre le configurazioni di dataset, raggiungendo addirittura un'accuratezza media del $87,08\%$ con un addestramento di 80000 recensioni, con un relativo transfer loss del $4,16\%$ e transfer ratio di $1,53\%$. L'utilizzo di una tecnica di transfer learning esplicita come il fine tuning ha permesso di migliorare,seppur di poco, i già ottimi risultati ottenuti cross domain

Altri risultati notevoli sono stati quelli ottenuti utilizzando una enorme mole di recensioni, al fine di potersi confrontare con il paper che secondo molti rappresenta lo stato dell'arte della sentiment classification applicata ai dataset Amazon (Zhang et al.,2015). Le prove condotte hanno portato ad ottenere risultati migliori rispetto a quelli del paper, con un $95,51\%$ nelle prove a 2 classi e $61,45\%$ in quelle a 5. I numeri ottenuti in queste prove sono particolarmente rilevanti, in quanto la mole di recensione utilizzate,(400.000 e 650.000), unita alla loro eterogeneità,dà luogo ad un test altamente probante, riducendo al minimo la possibilità di incorrere in un test composto da recensioni più semplici,al quale solo esposti i test composti da poche istanze.
Ad attestare la generalità delle DNC vi sono anche i risultati(tabella 8) ottenuti nell'altro dataset di benchmark, profondamente diverso dal dataset Amazon, nella quale si sono raggiunte accuratezze paragonabili a quelle ottenute da altre tecniche in letteratura.
Possiamo, alla luce degli esperimenti compiuti, asserire che i Differentiable Neural Computers possono svolgere egregiamente un task complesso come la Sentiment Classification,sia In-Domain che Cross-Domain, indipendentemente dalla struttura delle recensioni utilizzate.

BIBLIOGRAFIA

[1] Dizionario di medicina,neurone. URL `http://www.treccani.it/enciclopedia/neurone_res-f6daa336-9b53-11e1-9b2f-d5ce3506d72e_%28Dizionario-di-Medicina%29/`.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan): 993–1022, 2003.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[7] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Markov chain based method for in-domain and cross-domain sentiment classification. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K), 2015 7th International Joint Conference on*, volume 1, pages 127–137. IEEE, 2015.

[8] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. On deep learning in cross-domain sentiment classification. In *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2017.

[9] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14 (2):179–211, 1990.

[10] Epictetus, Giacomo Leopardi, and Claudio Moreschini. *Il manuale di Epitteto*, volume 9. Salerno editrice, 1990.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[13] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[14] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[15] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000.

[16] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[18] Khan Academy. Synapse,human biology. URL `https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/the-synapse`.

[19] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387, 2016.

[20] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[22] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[23] Pitts Walter McCulloch Warren S. A logical calculus of the ideas immanent in nervous activity. 1943.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

[27] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

[28] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[29] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[30] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[32] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[33] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[34] Wikipedia, the free encyclopedia. Neurone. URL `https://it.wikipedia.org/wiki/Neurone#/media/File:Complete_neuron_cell_diagram_it.svg`.

[35] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

RINGRAZIAMENTI

In primo luogo ci tengo a ringraziare il prof. Gianluca Moro, che mi ha dato la possibilità di svolgere una tesi su argomenti così interessanti e stimolanti, fornendomi sempre fondamentali spunti di riflessione.
Ringrazio profondamente anche i miei correlatori Andrea Pagliarani e Roberto Pasolini, che con le loro capacità, mi hanno sempre guidato e consigliato nell'affrontare temi e tecnologie così complessi.
Ringrazio infinitamente mia famiglia tutta che mi ha sempre supportato ed ha reso possibile questo traguardo.
Ed infine grazie a tutte le persone speciali che mi stanno accanto e mi hanno accompagnato fin qui.

A

OPERAZIONI PRELIMINARI

In questo capitolo verranno mostrati i passi necessari ad installare tutti i software e le librerie necessarie all'esecuzione del codice sviluppato per questa tesi. Il codice è scaricabile all'indirizzo `https://drive.google.com/open?id=0B_c80gg7c8oIOVdQUWowcjJQZFU`. Durante la guida si assume che si disponga già di Python e si utilizzi un sistema Linux, nello specifico le procedure proposte sono state testate con Ubuntu Mate 16.04.
Scaricare, inoltre, il modello Word2Vec che è stato utilizzato, a questo indirizzo: `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit`

A.1 INSTALLAZIONE LIBRERIE

A.1.1 *Installazione TensorFlow*

La seguente procedura si riferisce all'installazione della versione di TensorFlow senza accelerazione grafica. Verrano proposte due alternative modalità di installazione.

A.1.1.1 *Con PIP*

- Se pip non è gia presente o è presente in una versione minore di 8.1 (verificare con il comando pip3 -V o pip -V) eseguire il seguente comando:

  ```
  sudo apt-get install python-pip python-dev
  ```

- Installare TensorFlow con uno dei seguenti comandi a seconda della versione di Python che si è deciso di utilizzare:

  ```
  sudo pip3 install tensorflow
  ```

  oppure:

  ```
  sudo pip install tensorflow
  ```

- Per determinare se l'installazione è andata a buon fine digitare dall'interprete interattivo di Python (avviabile digitando python o python3 nel prompt dei comandi) le seguente istruzioni:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

Se tutto è andato a buon fine comparirà la scritta "Hello, TensorFlow!", in caso contrario seguire le istruzioni per i problemi più comuni: `https://www.tensorflow.org/install/install_linux#CommonInstallationProblems`

A.1.1.2 *Con Virtual-Env*

- Se non sono già presenti, installare pip e virtualenv, tramite il comando:

```
sudo apt-get install python-pip python-dev
    python-virtualenv
```

- Creare un ambiente virtuale tramite il seguente comando, è possibile sostituire ~/tensorflow con qualsiasi altro nome, questa sarà la cartella dove verrà creato l'ambiente virtuale con tensorflow:

```
python3 -m venv -system-syte-packages ~/tensorflow
```

- Bisogna ora attivare l'ambiente virtualenv appena creato, utilizzando il seguente comando(se si è utilizzata una cartella diversa da ~/tensorflow , sostituitelo con il nome della directory scelta):

```
source ~/tensorflow/bin/activate
```

A questo punto la source del prompt dei comandi dovrebbe cambiare in:

```
(tensorflow)$
```

- A questo punto installare TensorFlow con uno dei seguenti comandi a seconda della versione di Python che si è deciso di utilizzare:

```
pip3 install --upgrade tensorflow
```

oppure:

```
pip install --upgrade tensorflow
```

- Per determinare se l'installazione è andata a buon fine digitare dall'interprete interattivo di Python (avviabile digitando python o python3 nel prompt dei comandi) le seguente istruzioni:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

  Se tutto è andato a buon fine comparirà la scritta "Hello, TensorFlow!", in caso contrario seguire le istruzioni per i problemi più comuni: `https://www.tensorflow.org/install/install_linux#CommonInstallationProblems`

- Una volta terminata l'installazione, ogni qual volta si vuole utilizzare TensorFlow, bisognerà avviare il virtualenv associato:

```
source ~/tensorflow/bin/activate
```

- Per disattivare il virtual-env digitare:

```
deactivate
```

A.1.2 *Installazione librerie esterne*

- Salvare le seguenti righe di testo in un file chiamato 'requirement.txt':

```
nltk==3.2.3
gensim==2.2.0
numpy==1.11.0
```

- Installare le librerie python indicate nel file requirements.txt con il comando:

```
sudo pip install -r requirements.txt
```

A.1.3 *Installazione materiale per libreria DNC*

A.1.3.1 *Installazione Bazel*

- Aggiungere l'URI di distribuzione di Bazel come sorgente di pacchetti:

```
echo "deb [arch=amd64]
    http://storage.googleapis.com/bazel-apt
    stable jdk1.8" | sudo tee
    /etc/apt/sources.list.d/bazel.list
```

```
curl https://bazel.build/bazel-release.pub.gpg |
    sudo apt-key add -
```

- Installare Bazel tramite il seguente comando:

```
sudo apt-get update && sudo apt-get install bazel
```

- Aggiornare Bazel:

```
sudo apt-get upgrade bazel
```

A.1.3.2 *Installazione Sonnet*

- Se si utilizza virtualEnv, attivarlo:

```
source ~/tensorflow/bin/activate
```

- Clonare il repository di Sonnet:

```
git clone --recursive
    https://github.com/deepmind/sonnet
```

- Eseguire la configurazione, lasciare le impostazione di default a meno di esigenze particolari:

```
cd sonnet/tensorflow
./configure
cd ../
```

- Ora lanciare lo script di installazione per creare un file wheel in una directory temporanea:

```
mkdir /tmp/sonnet

sudo bazel build --config=opt :install
    --genrule_strategy=standalone
    --spawn_strategy=standalone
    --copt="-D_GLIBCXX_USE_CXX11_ABI=0"

./bazel-bin/install /tmp/sonnet
```

- Installare attraverso pip utilizzando il file appena creato:

```
sudo pip install /tmp/sonnet/*.whle
```

- A questo punto uscire dalla cartella Sonnet e per determinare se l'installazione è andata a buon fine digitare dall'interprete interattivo di Python (avviabile digitando python o python3 nel prompt dei comandi) le seguente istruzioni:

```
import sonnet as snt
import tensorflow as tf
snt.resampler(tf.constant([0.]),
    tf.constant([0.]))
```

L'output dovrebbe essere questo:

```
<tf.Tensor 'resampler/Resampler:0' shape=(1,)
    dtype=float32>
```

A.2 DOWNLOAD DEI DATASET

A.2.0.1 *Dataset Amazon*

I dataset utilizzati negli esperimenti relativi alle recensioni Amazon sono scaricabili ad i seguenti indirizzi:

- Books (B): `http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Books.json.gz`
- Electronics (E): `http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Electronics.json.gz`
- Clothing,Shoes and Jewelry (J): `http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Clothing_Shoes_and_Jewelry.json.gz`
- Movies and TV (M): `http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Movies_and_TV.json.gz`

A.2.0.2 *Stanford Sentiment Treebank*

Lo Stanford Sentiment Treebank è un dataset particolare, dove ogni recensione è organizzata sotto forma di albero dove ogni nodo è composto da una porzione di essa. Inoltre le valutazione sono espressi con punteggi da 0 ad 1. Si rende necessario quindi applicare alcuni accorgimenti al fine di poter utilizzare questo dataset al codice precedentemente creato. I dataset direttamente utilizzabili, da me composti, sono disponibili ai seguenti indirizzi:

- Dataset di training: `https://drive.google.com/file/d/0B_c80gg7c8oIOTF2OXdLUm55VGM/view?usp=sharing`
- Dataset di testing: `https://drive.google.com/open?id=0B_c80gg7c8oISkJsbUlMMkhOSUE`

B

AVVIO DI UN ESPERIMENTO

In questa sezione dell'appendice verranno descritte le procedure necessarie per lanciare un esperimento. Si presume siano stati già compiuti tutti i passi descritti nell'appendice A.

B.1 PROCEDURE GENERALI

Se non lo si è già fatto scompattare l'archivio SentimentClassificationWithDNC, al suo interno troveremo vari file, quelli che sono utili a lanciare i vari esperimenti sono:

- SentimentClassificationCrossDomainOfficial.py
- SentimentClassificationOfficial.py
- SentimentClassificationStanford.py
- SentimentClassificationWithTitleOfficial.py

La sintassi per lanciare un esperimento è la seguente:

```
python nome_file.py --configuration file_configurazione.json
```

A seconda dell'esperimento che si vuole lanciare si sostituisca il suo nome a nome_file.py. Il parametro —configuration serve invece per indicare al programma quale file di configurazione usare per quel determinato esperimento. Vediamo un esempio di file di configurazione, come quello fornito nel materiale:

```
{
   "hidden_size": "256",
   "memory_size": "32",
   "word_size": "64",
   "num_write_heads": "1",
   "num_read_heads": "1",
   "clip_value": "10",
   "max_grad_norm": "10",
   "batch_size":"60",
   "learning_rate": "1e-3",
   "final_learning_rate": "1e-3",
   "optimizer_epsilon": "1e-10",
   "num_training_iterations": "1620",
```

```
    "num_testing_iterations": "420",
    "num_epochs": "8",
    "report_interval": "10",
    "checkpoint_dir" :
        "/home/diego/sentiment-classification/mega/mixed256",
    "checkpoint_interval": "-1",
    "word_dimension" : "300",
    "max_lenght" : "150",
    "dataset" :
        "/media/diego/Volume/Reviews/reviews_Electronics.json",
    "datasetDest" :
        "/media/diego/Volume/Reviews/reviews_Electronics.json",
    "w2v_model" :
        "/media/diego/Volume/GoogleNews-vectors-negative300.bin",
    "random" : "True",
    "seed" : "19",
    "num_classes": "2"
}
```

Vediamo quindi in dettaglio cosa stanno a significare questi valori.

- hidden_size : è il numero di strati nascosti nel controller LSTM.
- memory_size : è il numero di locazioni di memoria da utilizzare nella DNC.
- word_size : è la dimensione delle singole locazioni di memoria.
- num_write_heads : è il numero di testine di scrittura.
- num_read_heads : è il numero di testine di lettura.
- clip_value: indica che gli output della rete potranno assumere valori compresi tra -clip_value e + clip_value.
- batch_size : è la dimensione del batch che si desidera usare.
- learning_rate: è il valore di learning rate iniziale
- final_learning_rate : è il valore finale del learning rate, il programma è fatto in modo cheil learning rate scali dal valore iniziale a quello finale nel corso delle epoche. Ovviamente se non si desidera variare il learning rate basta inserire lo stesso valore per quello iniziale e quello finale.
- num_training_operation: è il numero di recensioni che si vogliono utilizzare per la fase di training.
- num_testing_iterations: è il numero di recensioni che si vogliono utilizzare per la fase di testing.
- num_epoch: è il numero di epoche di cui si vuole comporre l'esperimento

- report_interval: indica ogni quanti batch riportare le informazioni relative all'addestramento o all test
- checkpoint_dir: è la cartella dove salvare o dove è contenuto,lo stato della rete.
- checkpoint_interval : indica ogni quanti batch salvare lo stato della rete, se posto a -1 significa che non verrà salvato alcuno stato.
- word_dimension : è la dimensionalità dei word embeddings, lasciarla a 300 se si utilizza il modello Word2Vec fornito.
- max_lenght : indica qual'è il numero massimo di parole consentito in una recensione per essere presa in esame.
- dataset : è il percorso al file dove si trova il dataset da considerare
- dataset_test : è il percorso al file dove si trova il dataset target in un esperimento cross-domain
- w2v_model : è il percorso al file dove sono contenuti i word embeddings pre-addestrati.
- random : può assumere i valori True e False, se posto a True significa che ogni recensione che viene scorsa nel dataset ha una probabilità su due di essere utilizzata.
- seed: è il seme di generazione per le operazioni che utilizzano la funzione random
- num_classes : è il numero di classi che si vogliono considerare, può assumere il valore 5 o 2.

B.2 ESPERIMENTI IN-DOMAIN

B.2.0.1 *Senza considerare il titolo delle recensioni*

Per lanciare un esperimento in-domain occorre :

- Specificare il dataset che si desidera utilizzare, aggiornando il campo 'dataset' del file di configurazione
- Aggiornare tutti quei campi del file di configurazione che si ritengano vadano cambiati.
- Lanciare l'esperimento (poniamo che il file di configurazione sia il file configuration.json):

```
python SentimentClassificationOfficial.py
    --configuration configuration.json
```

B.2.0.2 *Considerando il titolo delle recensioni*

Per lanciare un esperimento in-domain considerando il titolo delle recensioni occorre :

- Specificare il dataset che si desidera utilizzare, aggiornando il campo 'dataset' del file di configurazione
- Aggiornare tutti quei campi del file di configurazione che si ritengano vadano cambiati.
- Lanciare l'esperimento (poniamo che il file di configurazione sia il file configuration.json):

```
python SentimentClassificationWithTitleOfficial.py
    --configuration configuration.json
```

B.3 ESPERIMENTI CROSS-DOMAIN

Per lanciare un esperimento cross-domain occorre :

- Specificare i dataset che si desiderano utilizzare, aggiornando il campo 'dataset' (source domain) e 'dataset_dest' (target domain) del file di configurazione.
- Aggiornare tutti quei campi del file di configurazione che si ritengano vadano cambiati.
- Lanciare l'esperimento (poniamo che il file di configurazione sia il file configuration.json):

```
python SentimentClassificationCrossDomainOfficial.py
    --configuration configuration.json
```

B.4 ESPERIMENTI DATASET STANFORD

Per lanciare un esperimento cross-domain occorre :

- Inserire nel campo 'dataset' il percorso al file StanfordSentencesNTest.json e nel campo dataset_dest il percorso al file StanfordSentencesNTest.json
- Aggiornare tutti quei campi del file di configurazione che si ritengano vadano cambiati.
- Lanciare l'esperimento (poniamo che il file di configurazione sia il file configuration.json):

```
python SentimentClassificationStanford.py
    --configuration configuration.json
```