ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

Corso di Laurea in Ingegneria e Scienze Informatiche

Deep Learning per la Sentiment Analysis Cross-Domain: Studio e Sperimentazione con Dynamic Memory Networks

Relatore:
Chiar.mo Prof.
Gianluca Moro

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

Presentata da:
Nicola Piscaglia

Sessione II
Anno Accademico 2016-2017

Dedico questa mia tesi ed il percorso triennale, tanto impegnativo quanto edificante, che si conclude, alle persone che mi supportano sempre e da cui ho più imparato,

Antonio, Daniela e Linda

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

L'argomento principale della tesi è la progettazione di un classificatore di recensioni relative a prodotti commerciali, caratterizzate da linguaggi e contesti differenti, utilizzando lo stato dell'arte in ambito Deep Learning, rappresentato dalle reti neurali artificiali Dynamic Memory Networks Plus, e tecniche di Transfer Learning in compiti di in-domain e cross-domain sentiment classification.
Quest'ultima tecnica consiste nel valutare pareri positivi o negativi di un certo dominio, detto target, utilizzando la conoscenza estratta da un dominio sorgente eterogeneo nel linguaggio. La motivazione delle tecniche cross-domain è il superamento della costosa preclassificazione di numerosi esempi altrimenti operata da un esperto umano.
Soluzioni altamente scalabili sono state largamente applicate solamente in compiti di classificazione in-domain, utilizzando documenti appartenenti allo stesso dominio, nonostante l'area di ricerca del Transfer Learning sia di grande interesse in termini economici da parte di numerose aziende.
Questa tesi mira ad analizzare l'efficacia di applicazione di tecniche di Deep Learning prevalentemente tramite Dynamic Memory Networks Plus modellando i problemi di classificazione come compiti di Question Answering, e negli esperimenti In-Domain anche attraverso l'uso di Convolutional Neural Networks per la Sentiment Classification.
I risultati ottenuti verranno confrontati con esperimenti di classificazione analoghi eseguiti attraverso l'utilizzo di reti Dynamic Memory Network originali e Long-Short-Term-Memory per dimostrare come l'applicazione delle architetture CNN e DMN+ migliori effettivamente i risultati precedentemente ottenuti in questo ambito.

Capitolo 1

Deep Learning e Transfer Learning

1.1 Machine Learning

Il Machine Learning è la branca della scienza informatica che fornisce ai computer la capacità di imparare senza essere esplicitamente programmati. Si evolve dallo studio del pattern recognition e della teoria dell'apprendimento computazionale nel campo dell'intelligenza artificiale.
Il Machine Learning esplora lo studio e la costruzione di algoritmi che possono imparare e fare previsioni sui dati - tali algoritmi superano le istruzioni di programma strettamente statiche, predisponendo invece predizioni o decisioni, attraverso la costruzione di modello di conoscenza dal campionamento dell'input.
L'apprendimento delle macchine viene impiegato in una serie di compiti di calcolo in cui la progettazione e la programmazione di algoritmi espliciti con buone prestazioni è difficile o impraticabile; Le applicazioni di esempio includono il filtraggio tramite posta elettronica, la rilevazione di intrusi di rete o gli insider maliziosi intenzionati a violare i dati, riconoscimento ottico dei caratteri (OCR), classificazione e computer vision.

1.2 Discipline correlate

Il Machine Learning è strettamente legato alle seguenti discipline:

- Statistica computazionale: si concentra anche sulla previsione attraverso l'utilizzo di computer;
- Ottimizzazione matematica: fornisce metodi, teorie e domini applicativi al settore;
- Data Mining: si concentra maggiormente sull'analisi dei dati esplorativi e si profila come apprendimento senza supervisione.

L'apprendimento automatico può anche essere non supervisionato, può imparare e stabilire i profili comportamentali di base di varie entità e poi utilizzarli per trovare anomalie significative.
Nell'ambito dell'analisi dei dati, il machine learning è un metodo utilizzato per ideare complessi modelli e algoritmi che si prestano alla previsione; In ambito commerciale, questo è noto come analisi predittiva. Questi modelli analitici permettono ai ricercatori, agli scienziati di dati, agli ingegneri e agli analisti di produrre decisioni e risultati affidabili e ripetibili e scoprire informazioni nascoste all'occhio attraverso l'apprendimento delle relazioni e delle tendenze dei dati.

1.3 Deep Learning

1.3.1 Definizione

Il Deep Learning è un'area del Machine Learning che si basa sull'apprendimento dei dati "in profondità" su più livelli. È caratterizzato dalla creazione di un modello di apprendimento automatico a più strati, nel quale i livelli più profondi prendono in input le uscite dei livelli precedenti, trasformandoli e astraendoli sempre di più. Questa intuizione sui livelli di apprendimento dà il nome all'intero ambito (apprendimento in profondità) e si ispira al modo in cui il cervello dei mammiferi processa le informazioni ed impara, rispondendo agli stimoli esterni.
Ogni livello di apprendimento corrisponde, in questo ipotetico parallelo, ad una delle diverse aree che compongono la corteccia cerebrale. Ad esempio, la corteccia visiva, che è preposta al riconoscimento delle immagini, mostra una sequenza di settori, posti in gerarchia. Ciascuno di questi settori riceve una rappresentazione in ingresso, per mezzo dei segnali di flusso che lo collegano agli altri settori. Ogni livello di questa gerarchia rappresenta un diverso livello di astrazione, con le caratteristiche più astratte definite in termini di quelle del livello inferiore.
Nel momento in cui il cervello riceve in ingresso delle immagini, le elabora tramite diverse fasi, ad esempio il rilevamento dei bordi, la percezione delle forme (da quelle primitive a quelle gradualmente sempre più complesse). Si parla per questo di rappresentazione gerarchica dell'immagine a livello di astrazione crescente. Così come il cervello apprende per tentativi e attiva nuovi neuroni apprendendo dall'esperienza, anche nelle architetture preposte al Deep Learning, gli stadi di estrazione si modificano in base alle informazioni ricevute in ingresso.
Lo sviluppo del Deep Learning è avvenuto conseguentemente e contestualmente allo studio delle intelligenze artificiali, ed in particolar modo delle reti neurali. Dopo gli inizi negli anni '50, 'e soprattutto negli anni '80 che questo ambito si sviluppa, per mano di Geoff Hin-

ton e dei suoi collaboratori di Machine Learning. In quegli anni però, la tecnologia informatica non era sufficientemente sviluppata per consentire un reale miglioramento in questa direzione, per questo abbiamo dovuto aspettare fino ai nostri giorni per vedere, grazie alla disponibilità di dati ed alla potenza di calcolo, progressi ancora più significativi.

1.3.2 Motivazioni

Fortemente legato al Machine Learning è il tema del pattern recognition, ovvero della ricerca di schemi e regolarità nei dati che permettono ad un sistema, dopo una fase di addestramento, di classificare ogni input che gli viene presentato, solitamente sotto forma di un vettore di features, in una particolare classe di un insieme di classi possibili.
Nell' ambito del pattern recognition, uno dei principali ostacoli riguarda l'elevata dimensionalità dei dati. Ciò significa che, al crescere del numero delle features disponibili in input, il numero degli esempi di training necessari al sistema per l' addestramento cresce esponenzialmente. Ovvero, fissato il numero di esempi disponibili, la capacità predittiva del sistema diminuisce al crescere del numero di features, con un conseguente peggioramento delle prestazioni.
Questo problema, detto "curse of dimensionality", ha fatto si che l'approccio tipicamente usato per sistemi di pattern recognition consista nella divisione del sistema in due moduli : un modulo di feature extraction , che pre-elabora i dati in input in al fine di ridurne la dimensionalità , seguito da un classificatore , che prende in input i dati di dimensionalità ridotta dalla feature extraction e su di essi viene addestrato.
Il principale problema di un tale approccio risiede nel fatto che l' accuratezza e la precisione delle predizioni sono fortemente influenzate dall' abilità di colui che realizza il modulo di feature extraction, che richiede inoltre un forte sforzo ingegneristico ed una notevole conoscenza del dominio di applicazione del sistema.
Il Deep Learning è uno degli approcci di Machine Learning che permette principalmente il superamento di questo problema, attraverso metodi e algoritmi di rappresentazione dei dati in maniera gerarchica, su vari livelli di astrazione. L'aspetto chiave del Deep Learning sta nel fatto che questi livelli di rappresentazione delle features, ad un grado di astrazione sempre crescente, non sono progettati e realizzati direttamente dall'uomo, ma sono appresi automaticamente dai dati utilizzando particolari algoritmi di apprendimento : i concetti ai livelli più alti sono definiti a partire da quelli ai livelli più bassi tramite una serie di trasformazioni non lineari.

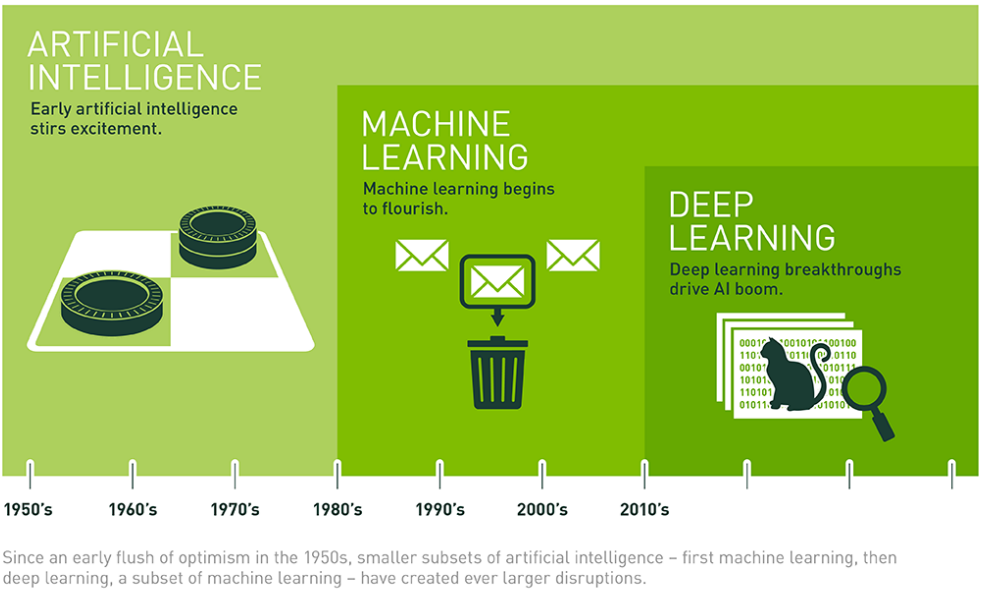


Figura 1.1: Evoluzione temporale dell'AI

1.4 Modalità di apprendimento

Le attività di apprendimento automatico sono tipicamente classificate in tre categorie:

- Apprendimento supervisionato: vengono presentati degli esempi in input e l'output desiderato. L'obiettivo è quello di insegnare alla rete una regola generale su un certo dominio di dati in modo che successivamente riesca a generalizzare

- Apprendimento non supervisionato: non vengono fornite etichette all'algoritmo di apprendimento, lasciando che trovi da solo la struttura nel suo input. L'apprendimento non supervisionato può costituire un obiettivo in sè (scoprire modelli nascosti nei dati) o un mezzo per una precisa finalità (apprendimento delle funzionalità).

- Renforcement learning: un applicativo interagisce con un ambiente dinamico in cui deve svolgere un determinato compito (ad esempio guidare un veicolo o giocare una partita contro un avversario). Al programma è fornito un feedback in termini di ricompense e punizioni mentre si naviga sul suo spazio problema.

Tra l'apprendimento supervisionato e non sorvegliato si trova l'apprendimento semi-supervisionato, in cui il "trainer" fornisce un insieme di informazioni incomplete i.e. un dataset con alcuni (spesso molti) risultati mancanti.

1.5 Transfer Learning

1.5.1 Introduzione

Il Transfer Learning è una tecnica di Machine Learning, dove il modello di conoscenza costruito durante l'addestramento in un tipo di problema viene adattato e applicato al fine di effettuare previsioni su un altro problema di dominio simile.
Nel Deep Learning, i primi strati sono addestrati per identificare le caratteristiche del problema. In questo modo, durante l'applicazione del Transfer Learning, è possibile rimuovere gli ultimi strati della rete addestrata e riqualificarla con livelli nuovi adatti al dominio di destinazione.
Il Transfer Learning si rileva davvero utile quando non si possiedono sufficienti dati etichettati nello scenario da analizzare con cui addestrare il modello di conoscenza, in questo caso infatti il tradizionale paradigma di apprendimento supervisionato perde di efficacia.

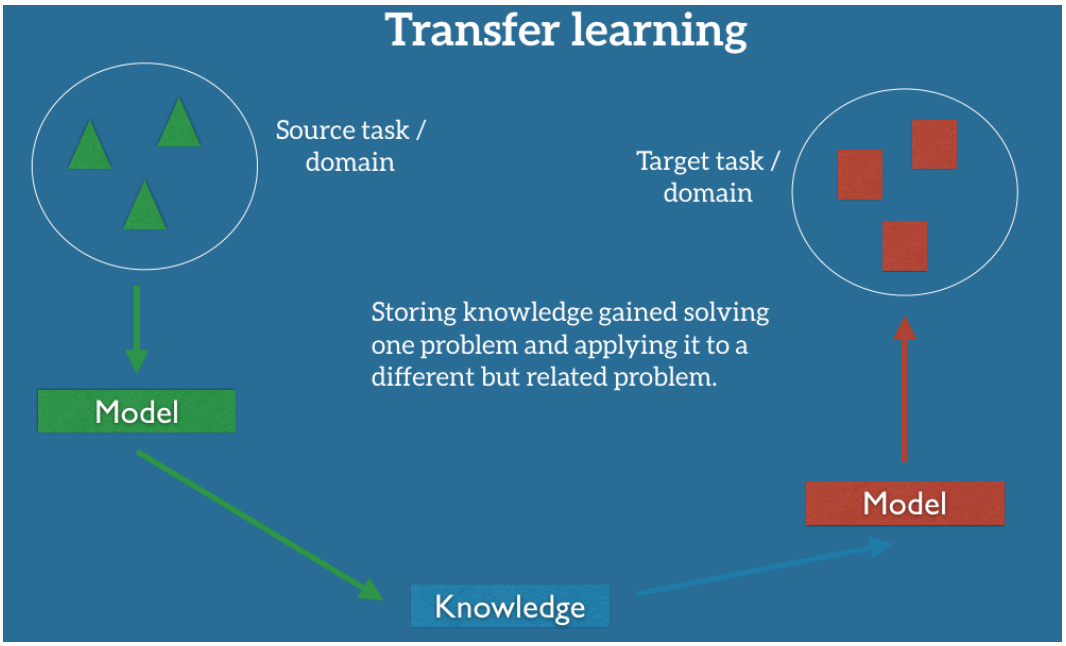


Figura 1.2: Modello base del Transfer Learning

1.5.2 Motivazioni

Nel contesto attuale, l'applicazione di tecniche di machine learning è caratterizzata da due fattori:

- Da una parte, la capacità di addestrare modelli sempre più accurati e precisi, con la conseguente diffusione di tali tecnologie su larga scala a milioni di utenti;
- Guardando l'altra faccia della medaglia, tali modelli si sono dimostrati poco efficienti se non supportati da una gran quantità di dati utili all'addestramento. Sicuramen-

te bisogna considerare che i grandi datasets sono spesso proprietari o estremamente costosi.

Ciò implica che utilizzare in un particolare scenario un modello addestrato su un contesto differente, porta inevitabilmente a significativi cali di performance, se non persino all'inefficacia totale del sistema. Questo perchè, applicando il modello addestrato al mondo reale, è normale imbattersi in condizioni e combinazioni differenti che non si erano mai visti. Il Transfer Learning diventerà un elemento chiave per il successo di tali tecnologie nel settore industriale, grazie alla sua capacità di estendere l'utilizzo del Machine Learning oltre ai compiti ed ai domini di cui si è colmi di informazioni.

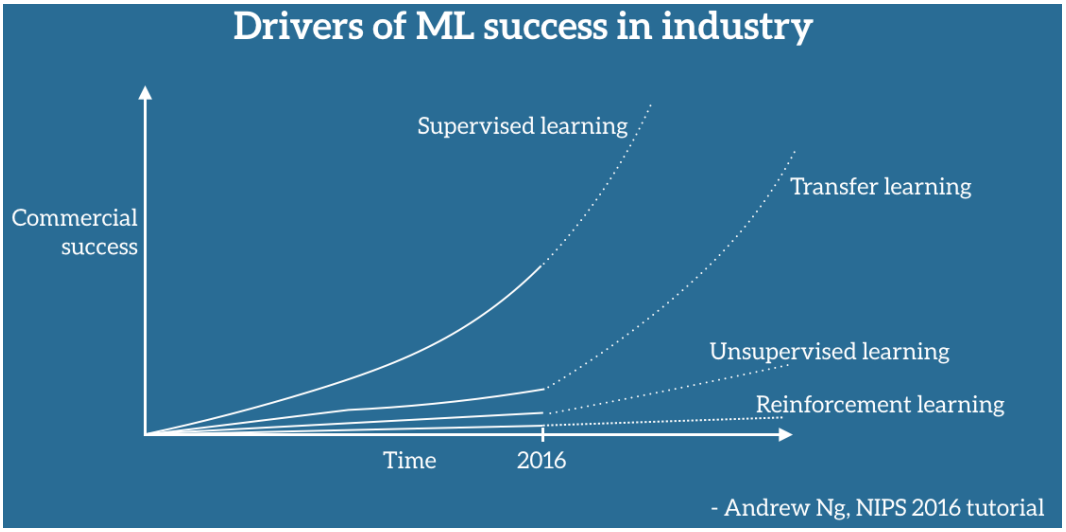


Figura 1.3: Fonti del successo industriale del Machine Learning secondo Andrew Ng, chief scientist at Baidu e professore di Stanford, fonte: conferenza NIPS 2016 (Neural Information Processing Systems) - top two conference in Machine Learning (insieme a ICML)

1.5.3 Definizione

Per la seguente definizione seguiamo lo studio condotto da Pan e Yang (2010), [28]:
Il Transfer Learning coinvolge i concetti di dominio e di compito. Un dominio $\mathcal{D}$ è definito da due parti, uno spazio funzionale $\mathcal{X}$ e una distribuzione marginale di probabilità $P(X)$, dove $X = x_1, \cdots, x_n \in \mathcal{X}$.
Successivamente analizziamo un esempio in cui l'applicazione del Machine Learning ha il compito di identificare moduli difettosi di un software.
Ogni metrica del software è considerata una feature dove x_i è l'i-esimo vettore delle features corrispondente all'i-esimo modulo software, in cui n è il numero di vettori features in X, $\mathcal{X}$ è invece lo spazio di tutti possibili vettori features di cui X è un particolare esempio.

Dato un dominio $\mathcal{D}$, un task $\mathcal{T}$ è definito da due parti: uno spazio delle etichette $\mathcal{Y}$ ed una funzione di predizione $f(.)$, distribuzione di probabilità condizionale $P(Y|X)$, la quale è appresa dalle coppie di esempi provienti dagli spazi feature ed etichetta $\{x_i, y_i\}$ dove $x_i \in X$ e $y_i \in \mathcal{Y}$. Riferendoci all'esempio precedente, $\mathcal{Y}$ è l'insieme di tutte le etichette, il quale conterrà i valori *true* e *false* mentre $f(x_i)$ è la funzione che predice l'etichetta y_i di un dato modulo x_i da classificare.
Dalle spiegazioni precedenti deriva che $\mathcal{D} = \{\mathcal{X}, P(X)\}$ e $\mathcal{T} = \{\mathcal{Y}, f(.)\}$. Dato un dominio di origine $\mathcal{D}_S = \{(x_{S1}, y_{S1}), ..., (x_{Sn}, y_{Sn})\}$, dove $x_{Si} \in \mathcal{X}_S$ è l'i-esimo esempio in $\mathcal{D}_S$ e y_{Si} è l'i-esima etichetta relativa all'esempio x_{Si}.
Analogamento $\mathcal{D}_T = \{(x_{T1}, y_{T1}), ..., (x_{Tn}, y_{Tn})\}$ è il dominio su cui verrà effettuato il trasferimento dell'apprendimento, dove $x_{Ti} \in \mathcal{X}_T$ è l'i-esimo esempio in $\mathcal{D}_T$ e y_{Ti} è l'i-esima etichetta relativa all'esempio x_{Ti}. Oltre a ciò, $\mathcal{T}_S$ denota il task svolto sul dominio sorgente dalla funzione di predizione $f_S(.)$ mentre $\mathcal{T}_T$ denota il task svolto sul dominio di trasferimento dalla funzione di predizione $f_T(.)$.
Si ottiene così la definizione formale di Transfer Learning:
Dato un dominio sorgente $\mathcal{D}_S$ ed un corrispondente task $\mathcal{T}_S$ ed un dominio target $\mathcal{D}_T$ con un corrispondente task $\mathcal{T}_T$, con Transfer Learning si intende la tecnica di miglioramento della funzione di predizione $f_T(.)$ usando le informazioni e gli esempi appartenenti al dominio $\mathcal{D}_S$ ed al compito $\mathcal{T}_S$, dove $\mathcal{D}_S \neq \mathcal{D}_T$ o $\mathcal{T}_S \neq \mathcal{T}_T$.
Tale definizione di Transfer Learning su un unico dominio sorgente appena descritto può essere esteso anche all'utilizzo di molteplici domini sorgente.
Poichè sia il dominio $\mathcal{D}$ che l'attività $\mathcal{T}$ sono definiti come tuple, queste disuguaglianze danno luogo a quattro scenari di Transfer Learning, discussi in seguito.

1.5.4 Scenari

Seguendo la definizione precedente, dati:

- Domini sorgente e destinazione, $\mathcal{D}_S$ e $\mathcal{D}_T$ dove $\mathcal{D} = \{\mathcal{X}, P(X)\}$
- Task sorgente e destinazione, $\mathcal{T}_S$ e $\mathcal{T}_T$ dove $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$

le condizioni di sorgente e target possono variare in quattro principali condizioni, che illustreremo tramite un esempio che concerne la classificazione dei documenti:

- $\mathcal{X}_S \neq \mathcal{X}_T$. Lo spazio del dominio sorgente e destinazione sono differenti, e.g. i documenti sono scritti in due linguaggi differenti. Nel contesto della NLP ci si riferisce solitamente ad adattamento del linguaggio;
- $P(X_S) \neq P(X_T)$. La probabilità di massa marginale del dominio sorgente e destinazione è differente, e.g. i documenti discutono di argomenti differenti. Questo scenario è conosciuto come adattamento del dominio;

- $\mathcal{Y}_S \neq \mathcal{Y}_T$. I task devono svolgere classificazioni differenti, e.g. i documenti vengono applicati a funzioni differenti. In pratica, questo scenario si verifica generalmente insieme allo scenario seguente, poichè è raro che due compiti differenti, con funzioni differenti, abbiano la stessa probabilità di distribuzione condizionale;

- $P(Y_S|X_S) \neq P(Y_T|X_T)$. La probabilità di distribuzione condizionale del task sorgente e destinazione sono diversi, e.g. i documenti sono sbilanciati riguardo le loro classi. Questo è uno scenario molto comune nella pratica, poichè i dati non sono creati ad-hoc ma provengono da contesti reali.

1.6 Applicazioni

Per quanto riguarda gli ambiti di applicazione, il Deep Learning trova ampio spazio nello sviluppo di sistemi di riconoscimento vocale e del parlato, nella ricerca di pattern e soprattutto nel text processing e nel riconoscimento di immagini, grazie alle sue caratteristiche di apprendimento per livelli, che gli consentono di concentrarsi, passo dopo passo, sulle varie aree di un'immagine da processare e classificare.

Capitolo 2

Reti neurali artificiali

2.1 Definizione

La risoluzione dei problemi cognitivi all'interno del cervello umano è affidata a reti (o circuiti) neurali, composte solitamente da svariate centinaia di cellule neuronali. Queste reti, la cui estensione e grandezza varia a seconda del compito richiesto, possono coinvolgere anche diverse aree cerebrali e il loro sviluppo e formazione è stato fondamentale nel processo di evoluzione della specie umana.
Quando ci si riferisce alle reti neurali artificiali, si intendono sistemi di elaborazione dell'informazione il cui scopo è di simulare il funzionamento delle reti biologiche all'interno di un sistema informatico.
Le reti neurali artificiali possono essere considerate come un'ampia rete informatica composta da diverse decine di entità di elaborazione che svolgono lo stesso ruolo che i neuroni svolgono all'interno delle reti biologiche. Ognuno di questi nodi artificiali è collegato agli altri nodi della rete attraverso una fitta rete di interconnessioni, le quali permettono anche alla rete stessa di comunicare con il mondo esterno. Lo scopo finale di una rete così articolata è quello di acquisire informazioni dal mondo esterno, elaborarle e restituire un risultato sotto forma di impulso.
Le reti neurali artificiali sono quindi spesso utilizzate nel campo della programmazione delle intelligenze artificiali per affrontare e tentare di risolvere determinate categorie di problemi. Nonostante la loro utilità, ovviamente non si può dire che questi reti siano intelligenti: l'elemento di intelligenza nell'intero processo è inserito dallo sviluppatore che deve analizzare il problema e costruire un'applicazione che utilizzi le reti neurali con la configurazione appropriata.

2.2 Analogia biologica

Il neurone è una cellula come qualsiasi altra, con una membrana ed un nucleo centrale, ma si differenzia notevolmente sia anatomicamente sia fisiologicamente. Il neurone "tipico"

(garden-variety neuron) presenta da un lato una protuberanza piuttosto lunga simile ad un filo chiamata assone e dall'altro lato una serie di fili ramificati più corti, spinosi e aguzzi chiamati dendriti.

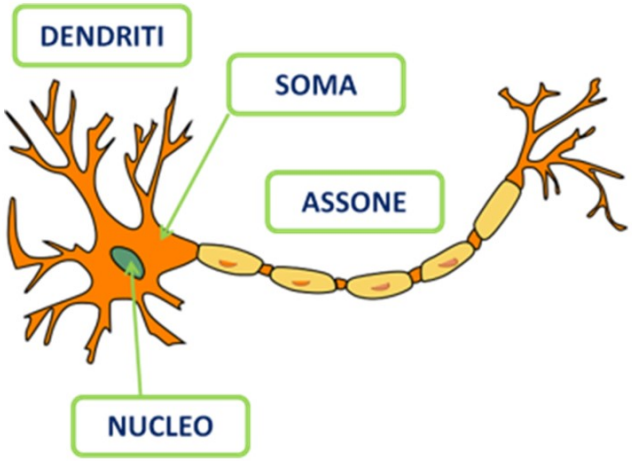


Figura 2.1: Modello di un neurone biologico

Ciascun neurone riceve segnali attraverso i suoi dendriti, li elabora nel corpo cellulare o soma e poi lancia un segnale tramite l'assone al neurone successivo. Il neurone si eccita mandando un impulso elettrico lungo l'assone. L'assone di un neurone non è direttamente collegato ai dendriti di altri neuroni: il punto in cui il segnale viene trasmesso da una cellula ad un'altra è un piccolo spazio denominato fessura sinaptica.

Una sinapsi è formata da una protuberanza sull'assone chiamata "bottone" o "nodo sinaptico" che sporge a forma di fungo e che si incastra con una prominenza a forma di spina dorsale sulla superficie del dendrite.

L'area tra il bottone e la superficie dendritica postsinaptica è chiamata fessura sinaptica ed è attraverso di essa che il neurone eccitato trasmette il segnale. Il segnale viene trasmesso non da una connessione elettrica diretta tra il bottone e la superficie dendritica, ma dall'emissione di una piccola quantità di fluidi chiamati neurotrasmettitori.

Quando il segnale elettrico si muove dal corpo cellulare lungo l'assone fino alla fine del bottone, questi provoca l'emissione di fluidi neurotrasmettitori nella fessura sinaptica. Questi ultimi entrano in contatto con dei recettori posti sul lato dendritico post-sinaptico.

Ciò causa l'apertura dei canali e gli ioni - atomi e gruppi di atomi carichi elettricamente - entrano ed escono dal lato dendritico, alterando così la carica elettrica del dendrite. Lo schema è dunque il seguente: vi è un segnale elettrico sul lato dell'assone seguito da una

trasmissione chimica nella fessura sinaptica, seguito da un segnale elettrico sul lato del dendrite.
La cellula riceve un'intera serie di segnali di suoi dendriti, li somma all'interno del suo corpo cellulare e, sulla base della somma, aggiusta la frequenza delle scariche da inviare alla cellula successiva.
I neuroni ricevono sia segnali eccitatori - ovvero segnali che tendono ad aumentare la loro frequenza di scarica - che segnali inibitori - quelli cioè che tendono a diminuire la loro frequenza di scarica.
Anche se ogni neurone riceve sia segnali eccitatori che inibitori, esso emette poi un solo tipo di segnale.
Per quanto si sa, salvo poche eccezioni un neurone è di tipo sia eccitatorio che in inibitorio.
La struttura e il funzionamento dei neuroni costituiscono l'intero fondamento causale della nostra vita cosciente.

2.3 Storia

L'ampia varietà di modelli non può prescindere dal costituente di base, il neurone artificiale proposto da W.S.McCulloch e W.Pitts in un famoso lavoro del 1943, il quale schematizza un combinatore lineare a soglia, con dati binari multipli in entrata e un singolo dato binario in uscita: un numero opportuno di tali elementi, connessi in modo da formare una rete, è in grado di calcolare semplici funzioni booleane.
Nel 1958, F.Rosenblatt introduce il primo schema di rete neurale, detto "perceptron" (percettrone), precursore delle attuali reti neurali, per il riconoscimento e la classificazione di forme, allo scopo di fornire un'interpretazione dell'organizzazione generale dei sistemi biologici [1]. Il modello probabilistico di Rosenblatt è quindi mirato all'analisi, in forma matematica, di funzioni quali l'immagazzinamento delle informazioni, e della loro influenza sul riconoscimento dei patterns; esso costituisce un progresso decisivo rispetto al modello binario di McCulloch e Pitts, perchè i suoi pesi sinaptici sono variabili e quindi il percettrone è in grado di apprendere.
L'opera di Rosenblatt stimola una quantità di studi e ricerche, e suscita un vivo interesse e notevoli aspettative nella comunità scientifica, destinate tuttavia ad essere notevolmente ridimensionate allorchè nel 1969 Marvin Minsky e Seymour A.Papert mostrano i limiti operativi delle semplici reti a due strati basate sui percettroni, e dimostrano l'impossibilità di risolvere per questa via molte classi di problemi, ossia tutti quelli non caratterizzati da separabilità lineare delle soluzioni: questo tipo di rete neurale non è abbastanza potente, infatti non è in grado di calcolare neanche la funzione or esclusivo (XOR). [3] Di conseguenza, a causa di queste limitazioni, ad un periodo di euforia per i primi risultati della "cibernetica" (come veniva chiamata negli anni '60), segue un periodo di diffidenza durante il quale tutte

le ricerche in questo campo non ricevono più alcun finanziamento dal governo degli Stati Uniti d'America e le ricerche sulle reti tendono, di fatto, a ristagnare per oltre un decennio, e l'entusiasmo iniziale risulta fortemente ridimensionato.

Il contesto matematico per addestrare le reti MLP (Multi-Layers Perceptron, ossia percettrone multistrato) fu stabilito dal matematico americano Paul Werbos nella sua tesi di dottorato del 1974. Uno dei metodi più noti ed efficaci per l'addestramento di tale classe di reti neurali è il cosiddetto "algoritmo di retropropagazione dell'errore" (error backpropagation), proposto nel 1986 da David E.Rumelhart, G. Hinton e R. J. Williams, il quale modifica sistematicamente i pesi delle connessioni tra i nodi, così che la risposta della rete si avvicini sempre di più a quella desiderata [4]. L'algoritmo di backpropagation (BP) è una tecnica d'apprendimento tramite esempi, costituente una generalizzazione dell'algoritmo d'apprendimento per il percettrone sviluppato da Rosenblatt [2] nei primi anni '60.

2.4 Applicazioni

Le ANN vengono utilizzati in molte discipline scientifiche, in cui modelli analitico-matematici non sono disponibili oppure presentano una complessità eccessiva che li rende inutilizzabili. In questi casi, le ANN possono estrarre un modello empirico a partire dai dati sperimentali. In un certo senso possono essere considerati degli interpolatori: sulla base di una serie di informazioni sul comportamento di un certo sistema (informazioni spesso incomplete e/o in parte scorrette in quanto affette da errori sperimentali), una rete neurale artificiale può essere "allenata" su questi dati al fine di poter prevedere il risultato di nuovi esperimenti.

2.5 Esempi

Esempi classici di utilizzo degli ANN comprendono il pattern recognition / classification: questo compito consiste nell'assegnare un pattern in entrata (ad esempio un segnale acustico o un'immagine) ad una o più classi predefinite. Le applicazioni pratiche sono innumerevoli: classificazione dei segnali bioelettrici, riconoscimento della voce, riconoscimento di un volto in un'immagine, ecc. L'applicazione classica è il Data Mining, cioè l'analisi minuziosa di banche dati enormi (es. dati sui clienti di una compagnia di assicurazioni) per trovare relazioni nascoste e correlazioni interessanti. Altri due esempi significativi sono la forecasting market prediction e le previsioni metereologiche: in questi due casi è necessario poter prevedere il comportamento futuro di un sistema conoscendo le osservazioni compiute nel passato.

2.6 Caratteristiche

2.6.1 Insieme di apprendimento

L'insieme di apprendimento contiene l'informazione su ciò che la rete neurale dovrebbe fare. La rete neurale, infatti, dovrebbe acquisire esperienza dagli esempi facenti parte dell'insieme di apprendimento. A questo proposito, quindi, la rete neurale può essere considerala come un meccanismo di generazione di un modello di conoscenza basato sull'imitazione della struttura del cervello umano. Questo processo va rapportato all'ambito dell'intelligenza artificiale, dove l'imitazione del cervello umano avviene a livello di macrostruttura. II problema di valutare se l'insieme di apprendimento contenga tutta l'informazione necessaria al compilo che deve essere realizzato non è stato ancora interamente risolto ed è inoltre collegato a quello della riduzione della ridondanza dei dati.

2.6.2 Algoritmo di apprendimento

L'algoritmo di apprendimento serve a elaborare ed inserire l'informazione contenuta nell'insieme di apprendimento nell'architettura della rete neurale, fissandone i parametri. In particolare l'apprendimento si articola in una sequenza di operazioni iterative, che nel generico passo prevedono l'azione dell'ambiente esterno e la modifica convergente dei parametri della rete mediante un algoritmo opportuno. Anche in questo caso non è completamente risolto il problema di valutare l'efficienza dell'algoritmo circa la capacità di estrarre la suddetta informazione. Sono disponibili soltanto risultati parziali.

2.6.3 Architettura

La struttura della rete neurale con i parametri che mette a disposizione è per sua natura adatta a realizzare compiti specifici. Quindi, assegnato un certo compito, occorre utilizzare un'architettura che ben si presti al tipo di problema per risolverlo. Uno dei problemi posti dal tipo di architettura è l'eccessiva dimensionalità e ridondanza dei dati. Sono stati sviluppati diversi tipi di architetture di reti neurali e nelle sezioni successive verranno trattati di modelli di maggior successo.

2.7 Modello generale

Tra i componenti principali del Modello generale di rete neurale troviamo:

- un insieme di unità elaborative;
- uno stato di attivazione;
- una funzione di output per ogni unità;

- le connessioni fra le unità;
- una regola di propagazione per propagare i valori; di output attraverso la rete di connessioni;
- una funzione di attivazione per combinare gli input con il valore di attivazione corrente e produrre un nuovo livello di attivazione;
- una regola di apprendimento per modificare le connessioni.

2.7.1 Unità elaborative

Le unità elaborative sono semplici e uniformi e l'attività di una unità consiste nel ricevere un input da un insieme di unità connesse e calcolare un valore di output da inviare ad un altro insieme di unità connesse in output. Il sistema è intrisecamente parallelo poichè molte unità possono effettuare la loro computazione in modo concomitante generalmente si distinguono tre tipi di unità:

- unità di input;
- unità di output;
- unità interne (Hidden).

2.7.2 Stato di attivazione

Ogni unità è caratterizzata da un valore di attivazione al tempo t : $a_i(t)$ Modelli diversi fanno assunzioni diverse circa i valori di attivazione che le unità possono assumere alternativamente valori:

- discreti;
- binari (attivo/inattivo) es.: 0,1, +1,-1;
- in un insieme ristretto es.: -1,0,+1, 1,2, ... 9;
- continui;
- limitati es.: [0,1];
- illimitati $[-\infty, +\infty]$.

2.7.3 Funzione di output

Le unità interagiscono trasmettendo segnali ai loro vicini (unita connesse). La forza dei loro segnali, e quindi il grado di influenza esercitata sul vicini dipende dal loro grado di attivazione. Ogni unità ui ha associata una funzione di output $f_i(a_i(t))$ che trasforma il valore corrente di attivazione $a_i(t)$ nel segnale in output

$$o_i(t) : o_i(t) = f_i(a_i(t)) \tag{2.1}$$

In alcuni modelli il valore di output è uguale a quello di attivazione: $f(x) = x$. In altri modelli f è una funzione a soglia, cioè un'unità non influenza le altre se la sua attivazione non supera un certo valore, in altri ancora f è una funzione stocastica del valore di attivazione.

2.7.4 Connessioni fra unità

Le unità sono connesse in una struttura a rete (digrafo). È definita una matrice di connessioni W in cui ogni connessione w_ji è caratterizzata da:

- unità di partenza (j)
- unità di arrivo (i)
- forza della connessione (numero reale):
  - $w_{ji} > 0$ connessione eccitatoria
  - $w_{ji} < 0$ connessione inibitoria
  - $w_{ji} = 0$ nessuna connessione

L'insieme delle connessioni costituisce la conoscenza della rete neurale e determina la sua risposta agli input. La memoria risiede nelle connessioni, le quali non sono programmate ma apprese automaticamente. Vi possono essere diversi tipi di connessione (a, b, ...), ognuno dei quali rappresentato da una matrice di connessione ($w_a, w_b, ...$).

2.7.5 Regola di propagazione

È la modalità con cui gli output delle unità $o(t)$ sono propagati attraverso le connessioni della rete per produrre i nuovi input. Se vi sono più tipi di connessioni diverse (es.: eccitatorie, inibitorie, ...) la propagazione avviene indipendentemente per ogni tipo di connessione. Denominiamo $net_{ai}(t)$ l'input di tipo a dell'unità -iesima.

2.7.6 Funzione di attivazione

È una funzione, F, che prende il valore corrente di attivazione $a(t)$ e il vettore net_k di input all'unità (k = tipo di connessione) e produce un nuovo valore di attivazione.

$$a(t+1) = F(a(t), net_1(t), net_2(t), ...) \tag{2.2}$$

In genere F è:

- una funzione a soglia;
- una funzione quasi-lineare (es: sigmoide);
- una funzione stocastica.

2.7.7 Regola di apprendimento

Le reti neurali non sono programmate da un esperto umano ma si autodefiniscono mediante apprendimento automatico. L'apprendimento in una RN consiste nella modifica delle connessioni effettuata mediante una regola di apprendimento. Vi possono essere tre tipi di modifiche:

1. la creazione di nuove connessioni;
2. la perdita di connessioni esistenti;
3. la modifica della forza di connessioni esistenti.

I casi 1 e 2 possono essere visti come casi particolari del caso 3.
La regola di apprendimento verrà approfondita in maniera esauriente durante la trattazione dell'algoritmo di Backpropagation.

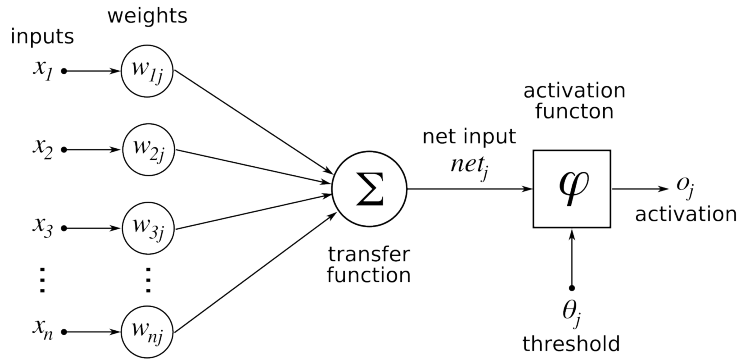


Figura 2.2: Modello computazionale di una rete neurale artificiale

2.8 Funzionamento in sintesi

I nodi che compongono una rete neurale artificiale sono suddivisi in tre macro-categorie. Abbiamo i nodi appartenenti alla categoria delle unità di ingresso (Input); i nodi appartenenti alle unità di uscita (Output); infine i nodi delle unità nascoste (Hidden). Ognuna di queste unità svolge un compito molto semplice: attivarsi nel caso in cui la quantità totale di segnale che riceve (sia da un'altra unità sia dal mondo esterno) supera una certa soglia di attivazione. In questo caso emette a sua volta un segnale attraverso dei canali di comunicazione fino a raggiungere le altre unità cui è connessa.
La funzione di trasferimento del segnale nella rete non è programmata ma è ottenuta attraverso un processo di apprendimento basato su dati empirici. Questo processo può essere supervisionato, non supervisionato o per rinforzo. Nel primo caso la rete utilizza un insieme di dati di addestramento grazie ai quali riesce a inferire i legami che legano questi dati e sviluppare un modello "generale". Questo modello verrà successivamente utilizzato per risolvere problemi dello stesso tipo.
Nel caso del processo di apprendimento non supervisionato, il sistema fa riferimento ad algoritmi che tentano di raggruppare i dati di ingresso per tipologia, individuando cluster rappresentativi dei dati stessi facendo uso tipicamente di metodi topologici o probabilistici.
Nel processo per rinforzo un algoritmo si prefigge di individuare un modus operandi a partire da un processo di osservazione dell'ambiente esterno. In questo processo è l'ambiente stesso a guidare l'algoritmo nel processo di apprendimento.

2.9 Reti feedforward

Le reti neurali si basano principalmente sulla simulazione di neuroni artificiali opportunamente collegati.
I suddetti neuroni ricevono in ingresso degli stimoli e li elaborano. L'elaborazione può essere anche molto sofisticata ma in un caso semplice si può pensare che i singoli ingressi vengano moltiplicati per un opportuno valore detto peso, il risultato delle moltiplicazioni viene sommato e se la somma supera una certa soglia il neurone si attiva attivando la sua uscita.
Il peso indica l'efficacia sinaptica della linea di ingresso e serve a quantificarne l'importanza, un ingresso molto importante avrà un peso elevato, mentre un ingresso poco utile all'elaborazione avrà un peso inferiore. Si può pensare che se due neuroni comunicano fra loro utilizzando maggiormente alcune connessioni allora tali connessioni avranno un peso maggiore, fino a che non si creeranno delle connessioni tra l'ingresso e l'uscita della rete che sfruttano "percorsi preferenziali". Tuttavia è sbagliato pensare che la rete finisca col produr-

re un unico percorso di connessione: tutte le combinazioni infatti avranno un certo peso, e quindi contribuiscono al collegamento ingresso/uscita.
Le equazioni 2.3 e 2.4 rappresentano la formulazione matematica formale di quanto descritto precedentemente. Ogni livello avrà una propria variabile di indice: k per i nodi di uscita, j (e h) per i nodi nascosti e i per i nodi di input.
In una rete di avanzamento, il vettore di input x, viene propagato attraverso un layer di pesi V. θ è detto bias e può essere pensato come un settaggio della funzione di attivazione (per es. quando $f(x) = x$), o come un livello base di attivazione per l'output del neurone (per es. quando $f(x) = sign(y)$ o $f(x) = y\Theta(y)$ dove Θ è la funzione di Heaviside).
In quest'ultima situazione, il valore $-b$ rappresenta un valore di soglia che la somma pesata degli input deve superare affinché il dispositivo sia attivo (cioè che l'output sia positivo).

$$y_j(t) = f(net_j(t)) \tag{2.3}$$

$$net_j(t) = \sum_i^n x_i(t)v_{ji} + \theta_j \tag{2.4}$$

n è il numero di ingressi, θ_j è un bias, e f è una funzione di output (di qualsiasi differenziabile tipo).
L'uscita della rete è determinata dallo stato e da un insieme di pesi di output W:

$$y_k(t) = g(net_k(t)) \tag{2.5}$$

$$net_k(t) = \sum_j^m y_j(t)w_{kj} + \theta_k \tag{2.6}$$

dove g è una funzione di output (possibilmente uguale a f).
I singoli neuroni vengono collegati alla schiera di neuroni successivi, in modo da formare una rete di neuroni. Normalmente una rete è formata da tre strati.
Nel primo abbiamo gli ingressi (I), questo strato si preoccupa di trattare gli ingressi in modo da adeguarli alle richieste dei neuroni. Se i segnali in ingresso sono già trattati può anche non esserci.
Il secondo strato è quello nascosto (H, hidden), si preoccupa dell'elaborazione vera e propria e può essere composto anche da più colonne di neuroni.
Il terzo strato è quello di uscita (O) e si preoccupa di raccogliere i risultati ed adattarli alle richieste del blocco successivo della rete neurale. Queste reti possono essere anche molto complesse e coinvolgere migliaia di neuroni e decine di migliaia di connessioni.
Per costruire la struttura di una rete neurale multistrato si possono inserire N strati hidden; vi sono però alcune dimostrazioni che mostrano che con 1 o 2 strati di hidden si ottiene una

stessa efficace generalizzazione da una rete rispetto a quella con più strati hidden. L'efficacia di generalizzare di una rete neurale multistrato dipende ovviamente dall'addestramento che ha ricevuto e dal fatto di essere riuscita o meno ad entrare in un minimo locale buono.

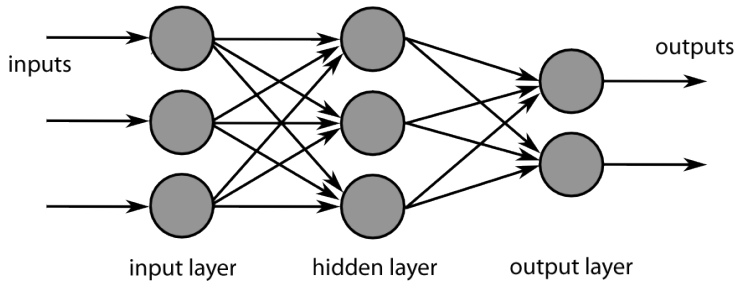


Figura 2.3: Modello di base di una rete neurale feedforward

2.10 Reti ricorrenti

Una rete neurale ricorrente[8] (RNN) è una classe di rete neurale artificiale in cui i collegamenti tra unità formano un ciclo diretto. Questa architettura consente alla RN di avere comportamenti temporali dinamici. A differenza delle reti neurali senza stato, le RNN possono utilizzare la propria memoria interna per elaborare sequenze arbitrarie di input. Ciò li rende applicabili a attività come il riconoscimento della scrittura a mano o il riconoscimento vocale non segmentato. In una semplice rete ricorrente, il vettore di input è similmente propagato attraverso un layer di pesi, ma anche combinato con la precedente attivazione di stato attraverso un ulteriore layer di pesi ricorrente U:

$$y_j(t) = f(net_j(t)) \tag{2.7}$$

$$net_j(t) = \sum_i^m x_i(t)v_{ji} + \sum_h^m y_h(t-1)u_{jh}) + \theta_j \tag{2.8}$$

dove m è il numero dei nodi "di stato".
L'uscita della rete è determinata dallo stato e da un insieme di pesi di output W in modo equivalente a quanto descritto nella sezione precedente sulle reti neurali feedforward.

2.10.1 Breve storia

Le reti neurali ricorrenti sono state sviluppate negli anni '80. Tra le prime architetture ricorrenti sono state sviluppate le reti Hopfield, inventate da John Hopfield nel 1982, e nel 1993 un sistema neurale compressore ha risolto un compito "molto profondo" che richiedeva più di 1000 livelli successivi in una RNN distribuita nel tempo.

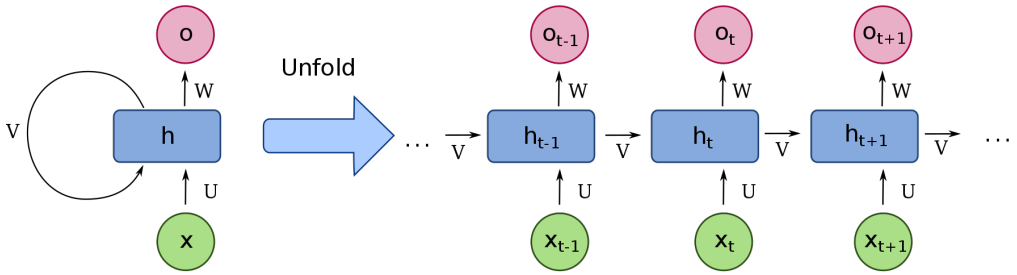


Figura 2.4: Modello di una Recurrent Neural Network

Nel 1997, Hochreiter e Schmidhuber inventano le Long-Short-Term-Memory[11] che raggiungono il record di precisione in più domini di applicazioni.
Intorno al 2007, le LSTM hanno iniziato a rivoluzionare il riconoscimento vocale, superando i modelli tradizionali in alcune applicazioni vocali. Nel 2009, le LSTM addestrate dalla CTC sono state il primo modello di RNN a vincere i concorsi di pattern-recognition, nel riconoscimento della scrittura a mano collegata. Nel 2014, il gigante cinese di ricerca Baidu ha utilizzato i RNN addestrati da CTC per superare il benchmark di riconoscimento vocale Switchboard Hub5'00, senza utilizzare metodi tradizionali di elaborazione vocale.
Le reti LSTM hanno inoltre migliorato il riconoscimento vocale, la sintesi text-to-speech, anche per Google Android. Nel 2015, le prestazioni del riconoscimento vocale di Google sono aumentate del 49% tramite le LSTM addestrato da CTC, utilizzato dalla ricerca vocale di Google.
Le LSTM hanno superato i record nella traduzione automatica, la modellazione linguistica e l'elaborazione della lingua multilingue.
Inoltre le LSTM combinate con reti neurali convolutive (CNNs) hanno migliorato la didascalia automatica delle immagini.

2.11 Principali architetture

2.11.1 Fully Recurrent

Le RNN di base sono una rete di nodi simili al neurone, ognuno con una connessione diretta (unidirezionale) ad ogni altro nodo. Ogni nodo (neurone) ha un valore di attivazione reale dipendente dal tempo. Ogni connessione (sinapsi) ha un peso valorizzato da un numero reale. I nodi sono: in ingresso (che ricevono dati dall'esterno della rete), nodi di output (forniscono i risultati) o nodi nascosti (che modificano i dati in elaborazione dall'ingresso all'uscita).

Nell'apprendimento supervisionato con configurazioni temporali discrete, sequenze di vettori di input validi in valore arrivano ai nodi di ingresso, un vettore alla volta. In un determinato periodo di tempo, ogni unità di non-input calcola la sua attivazione corrente (risultato) come funzione non lineare della somma ponderata delle attivazioni di tutte le unità che si collegano ad essa. Le attivazioni di target specificate dal supervisore possono essere fornite per alcune unità di output in determinati passaggi temporali. Ad esempio, se la sequenza di input è un segnale vocale corrispondente a una cifra vocale, l'output finale di destinazione alla fine della sequenza può essere un'etichetta che classifica la cifra.
Nelle configurazioni di apprendimento per rinforzo, una funzione "ricompensa" è occasionalmente utilizzata per valutare le prestazioni del RNN, che influenza il suo flusso di ingresso attraverso unità di uscita collegate ad attuatori che influenzano l'ambiente. Questa configurazione potrebbe essere usata ad esempio per giocare una partita in cui il progresso viene misurato con il numero di punti vinti.
Ogni sequenza produce un errore come somma delle deviazioni di tutti i segnali delle attivazioni corrispondenti calcolate dalla rete. Per un training set di numerose sequenze, l'errore totale è la somma degli errori di tutte le sequenze individuali.

2.11.2 Recursive

Una rete neurale ricorsiva è una sorta di rete neurale creata applicando lo stesso insieme di pesi ricorsivamente su una data struttura, per produrre una predizione sulle strutture di input a grandezza variabile o scalare su di esso, visitando il percorso in ordine topologico. Le reti ricorsive hanno avuto successo, ad esempio, nell'apprendimento di sequenze e strutture ad albero nel Natural Language Processing, principalmente su rappresentazioni continue e frasi basate sull'integrazione di parole.
Le RNN ricorsive sono state introdotte per imparare rappresentazioni distribuite di struttura, come termini logici.
Modelli e quadri generali sono stati sviluppati in ulteriori opere[14] fin dagli anni '90 tra cui le Recursive Neural Tensor Networks[13] (Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng & Christopher Potts, 2013) applicate al dataset Stanford Sentiment Treebank.

2.11.3 Long short-term memory

La Long Short-Term Memory (Hochreiter & Schmidhuber, 1997)[11] è un'architettura di rete neurale artificiale ricorrente (RNN), che consente ai dati di fluire sia in avanti che indietro all'interno della rete. La LSTM è universale nel senso che, date delle unità di rete sufficienti, permette calcolare tutti i risultati alla portata di un computer convenzionale, a condizione che abbia configurata una corretta matrice di pesi, che può essere considerata come il

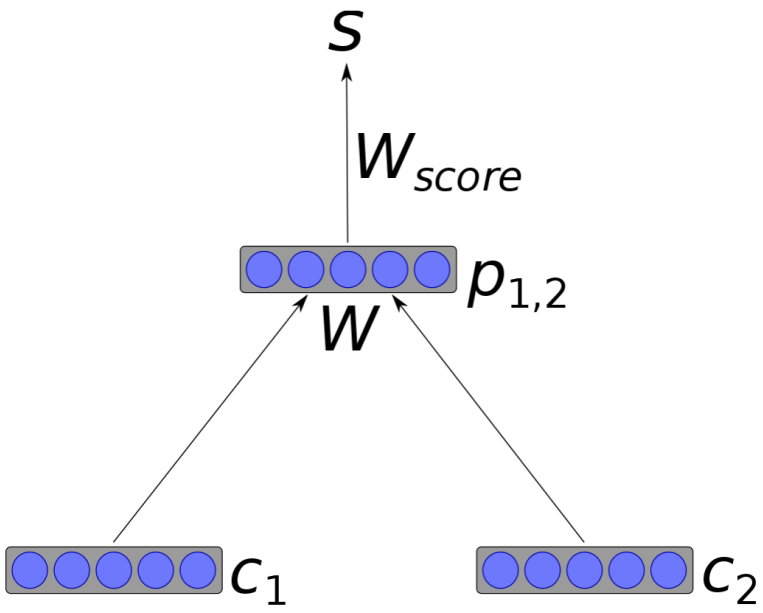


Figura 2.5: Recursive Neural Network

suo programma. Una LSTM è adatta ad imparare a classificare, elaborare e predire la serie temporale dati dei ritardi di tempo variabili compresi tra più eventi. L'insensibilità relativa alla lunghezza delle fessure offre un vantaggio alle LSTM sugli altri tipi di RNN, sui modelli di Markov e altri metodi di apprendimento sequenziale in numerose applicazioni[10]. Una rete LSTM contiene unità LSTM in aggiunta o in sostituzione ad altre unità elaborative convenzionali. Un'unità LSTM eccelle a ricordare i valori sia per periodi di tempo lunghi sia per periodi brevi. Il motivo principale di questo punto di forza delle LSTM è che non utilizzano alcuna funzione di attivazione all'interno dei suoi componenti ricorrenti. Pertanto, il valore memorizzato non viene periodicamente trascinato verso lo 0 e il gradiente non tende a svanire nel tempo, quando viene addestrato con l'algoritmo di backpropagation.
Le unità LSTM vengono spesso implementate in "blocchi" contenenti diverse unità. Questo design è tipico delle reti neurali e facilita le implementazioni con hardware parallelo. Nelle equazioni sottostanti, ogni variabile in corsivo minuscolo rappresenta un vettore con una lunghezza pari al numero di unità LSTM nel blocco.
Valori iniziali: $c_0 = 0$ e $h_0 = 0$. L'operatore $\circ$ rappresenta il prodotto di Hadamard.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{2.9}$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{2.10}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{2.11}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{2.12}$$

$$h_t = o_t \circ o_h(c_t) \tag{2.13}$$

Variabili:

- x_t: vettore di input;
- h_T: vettore di output;
- c_t: vettore di stato della cella;
- W, U e b: matrici dei pesi e vettore dei bias;
- f_t, i_t e o_t: vettori dei gate
  - f_t: vettore del gate Forget;
  - i_t: vettore del gate Input;
  - o_t: vettore del gate di output.

Funzioni di attivazione tradizionali

- σ_g: funzione sigmoidea;
- σ_c: tangente iperbolica;
- σ_h: tangente iperbolica.

I blocchi LSTM contengono 3 o 4 gate che controllano il flusso di informazioni. Queste porte vengono implementate usando la funzione logistica per calcolare un valore compreso tra 0 e 1. La moltiplicazione viene applicata con questo valore per consentire o negare parzialmente le informazioni che entrano o escono dalla memoria. Ad esempio, una porta "input" controlla la misura in cui un nuovo valore scorre nella memoria. Una porta "forget" controlla la misura in cui un valore rimane in memoria. Una porta "output" controlla la misura in cui viene utilizzato il valore nella memoria per calcolare l'attivazione di uscita del blocco. (In alcune implementazioni, le porte di ingresso e di "forget" vengono fuse in un unico gate. La motivazione per combinare i due tipi di gate è che il momento in cui dimenticare un valore coincide al momento in cui viene reso disponibile un valore nuovo da memorizzare.)
I pesi in un blocco LSTM (W e U) vengono utilizzati per dirigere il funzionamento delle porte. Questi pesi sono utilizzati quando i valori che entrano nel blocco (compreso il vettore di input x_t e l'output precedente al passo h_{t-1}) e in ciascuna delle porte. In questo modo,

il blocco LSTM determina come mantenere la propria memoria in funzione di tali valori e l'addestramento dei suoi pesi insegna al blocco la funzione che minimizza l'errore tra input e output desiderato.

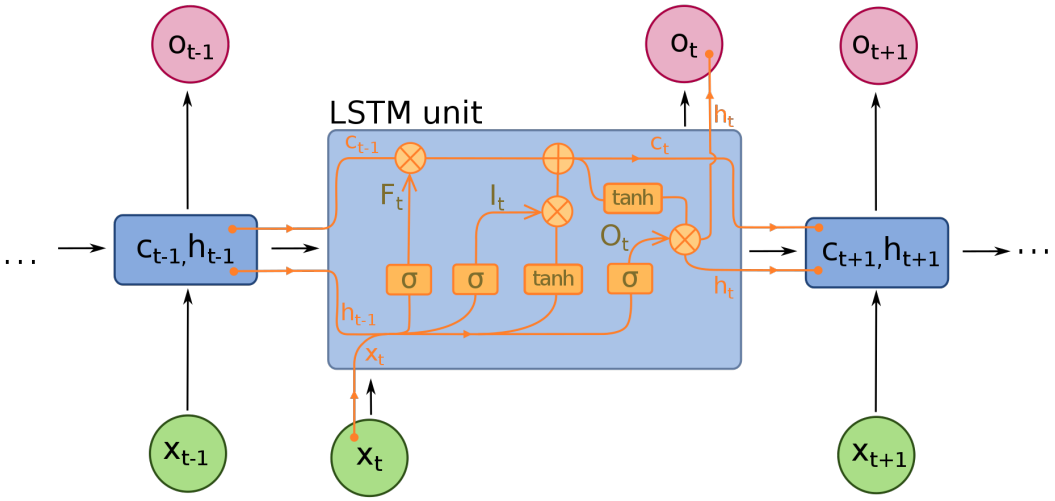
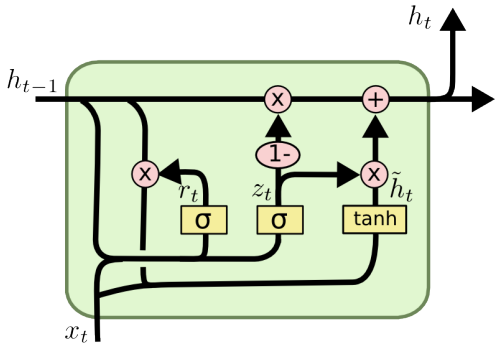


Figura 2.6: Modello tradizionale LSTM

2.11.4 Gated recurrent unit

Le Gated recurrent unit (GRUs) sono un meccanismo di gating in reti neurali ricorrenti, introdotte nel 2014. La loro performance sulla modellazione polifonica e la modellazione del segnale vocale è stata simile a quella delle LSTM. Hanno meno parametri rispetto a LSTM, poichè mancano di una porta di uscita. Combina i gate Forget e Input in una singolo "Update Gate". Inoltre unisce lo stato della cella e lo stato nascosto e apporta alcune altre modifiche. Il modello risultante è più semplice dei modelli standard LSTM e sta diventando sempre più popolare.



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

Figura 2.7: Modello con equazioni di una Gated Recurrent Unit

2.11.5 Bi-directional

Le RNN bidirezionali utilizzano una sequenza finita per predire o etichettare ogni elemento della sequenza in base ai contesti passati e futuri dell'elemento. Ciò avviene concatenando le uscite di due RNN, una che tratta la sequenza da sinistra a destra, l'altra da destra a sinistra. Le uscite combinate sono le previsioni dei segnali target forniti dal supervisore. Questa tecnica è risultata particolarmente utile quando è combinata con le LSTM.

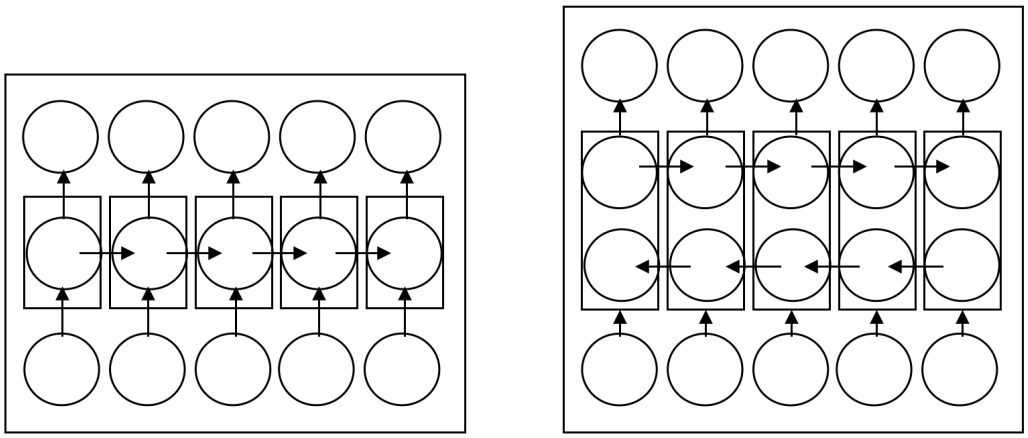


Figura 2.8: Un-directional vs. Bi-directional Neural Network

2.11.6 Convolutional Neural Networks

Le reti neurali convoluzionali (ConvNets o CNN) sono una categoria di reti neurali che si sono dimostrate molto efficaci in settori quali il riconoscimento e la classificazione delle immagini. I ConvNets sono riusciti a identificare facce, oggetti e segni di traffico, oltre a supportare la visione in robot e autovetture. Ultimamente, i ConvNets sono stati efficaci in diverse attività di elaborazione del linguaggio naturale[12] (come la classificazione di frasi[15] in Figura 2.14). ConvNets, quindi, sono uno strumento importante per la maggior parte dei praticanti del Machine Learning oggi.
LeNet è stata una delle prime reti neuronali convoluzionali che hanno contribuito all'evoluzione del campo del Deep Learning. L'architettura LeNet è stata utilizzata principalmente per attività di riconoscimento dei caratteri come la lettura di codici postali, cifre, ecc.
La rete neurale convoluzionale in Figura 2.9 è simile in architettura al LeNet originale e classifica un'immagine di input in quattro categorie: cane, gatto, barca o uccello (il LeNet originale è stato utilizzato principalmente per attività di riconoscimento dei caratteri). Come evidente dalla figura precedente, quando riceve un'immagine barca come input, la rete assegna correttamente la più alta probabilità per la barca (0.94) tra tutte e quattro le categorie. La somma di tutte le probabilità nel livello di output deve essere 1.
Le ConvNet si basano su 4 principali operazioni mostrate in Figura 2.9:

1. Convoluzione
2. Funzione di attivazione non lineare (ReLU)
3. Pooling o Sub Sampling
4. Classificazione (Fully Connected Layer)

Queste operazioni sono i blocchi fondamentali di ogni CNN, per questo ne forniremo una spiegazione in modo intuitivo al fine di comprendere il funzionamento di questo tipo di reti neurali.

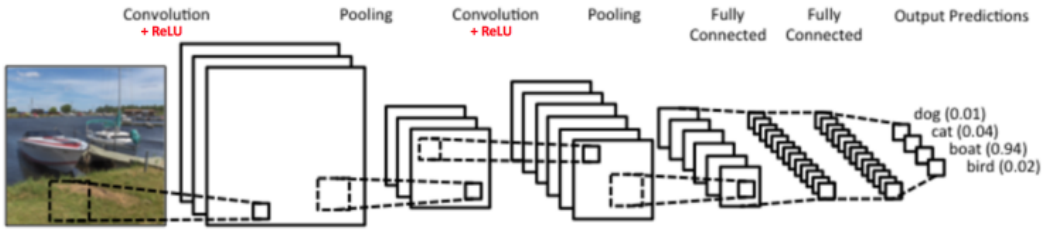


Figura 2.9: Una semplice Convolutional Neural Network

Convolution Step Le ConvNets devono il loro nome all'operatore "convolution". Lo scopo principale di Convolution in caso di una ConvNet è quello di estrarre le features dall'immagine di input. La convoluzione conserva la relazione spaziale tra i pixel imparando le funzioni dell'immagine usando piccoli quadrati di dati di input. Non ci addentreremo nei dettagli matematici della Convoluzione, ma cercheremo di capire come funziona sulle immagini.

Ogni immagine può essere considerata come una matrice di valori di pixel: per esempio si consideri un'immagine 5 x 5 i cui valori di pixel sono solo 0 e 1 (si noti che per un'immagine in scala di grigi, i valori dei pixel vanno da 0 a 255, la matrice verde sotto è un caso speciale dove i valori dei pixel sono solo 0 e 1):

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Inoltre, si consideri anche la matrice 3 x 3 seguente:

1	0	1
0	1	0
1	0	1

Dunque, la convoluzione dell'immagine 5 x 5 e della matrice 3 x 3 può essere calcolata come mostrato dal primo passo dell'operazione nella Figura 2.10:

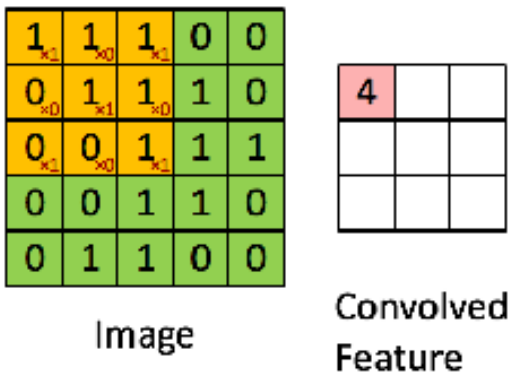


Figura 2.10: L'operatore di convoluzione. La matrice di output è chiamata Convolved Feature o Feature Map

La matrice arancione viene traslata sull'immagine originale (verde) di un pixel (chiamato anche "stride") e per ogni posizione, viene calcolata la moltiplicazione per elemento (tra

le due matrici) ed i prodotti vengono sommati per ottenere un l'intero finale che forma un singolo elemento della matrice di output (rosa). Si noti che la matrice 3 x 3 viene applicata solo ad un parte dell'input ad ogni "stride". Nella terminologia delle CNN, la matrice 3 x 3 è chiamata "filtro" o "nucleo" o "feature detector" e la matrice costruita applicando in successione il filtro sull'immagine e calcolando il prodotto scalare è chiamata "Convolved Feature" o "Activation Map" o "Feature Map". È importante notare che i filtri fungono da feature detectors dell'immagine in input originale.
È evidente che valori diversi della matrice di filtro produrranno diverse Feature Maps per la stessa immagine di input. È importante notare che l'operatore di convoluzione cattura le dipendenze locali nell'immagine originali: in pratica, una CNN impara i valori di questi filtri da sé durante il processo di training (sebbene sia necessario specificare parametri come il numero di filtri, la dimensione dei filtri e l'architettura della rete prima dell'addestramento). Più filtri abbiamo, più caratteristiche dell'immagine vengono estratte e meglio la nostra rete riconosce immagini mai viste prima.
La dimensione della mappa delle funzionalità (Feature Map) è controllata da tre parametri che dobbiamo decidere prima della fase di convoluzione:

- Profondità: corrisponde al numero di filtri utilizzati per l'operazione di convoluzione.

- Stride: è il numero di pixel con cui scorriamo la nostra matrice di filtro sulla matrice di input. Quando il passo è 1, spostiamo i filtri un pixel alla volta. Quando il passo è 2, i filtri salgono due pixel alla volta mentre li scorriamo. Avere un passo più grande produrrà mappe di funzionalità più piccole.

- Zero-padding: a volte è conveniente applicare la matrice di input con zeri intorno al bordo, in modo da poter applicare il filtro a elementi ai bordi della nostra matrice di immagini di input. Una caratteristica utile del zero padding è che ci permette di controllare la dimensione delle mappe delle funzioni. L'aggiunta del zero-padding è anche chiamata wide convolution, e non utilizzando lo zero-padding può essere pensata come una convoluzione ridotta.

Introducing Non Linearity (ReLU) Una operazione aggiuntiva denominata ReLU è stata utilizzata dopo ogni operazione di Convoluzione nella Figura 2.9. ReLU rappresenta l'unità lineare rettificata ed è un'operazione non lineare. Il suo output è dato da:
ReLU è un'operazione applicata elemento per elemento (pixel per pixel in questo caso) e sostituisce tutti i valori dei pixel negativi nella mappa delle funzioni con il valore zero. Lo scopo di ReLU è quello di introdurre la non linearità nel nostro ConvNet poiché la maggior parte dei dati del mondo reale che vorremmo che la nostra ConvNet imparasse sarebbe non

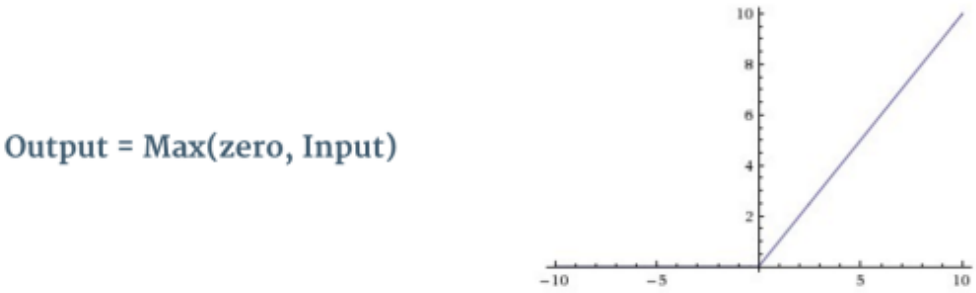


Figura 2.11: L'operatore ReLU

lineare (la Convoluzione è un'operazione lineare così come la moltiplicazione e somma di matrici, quindi si introduce una funzione non lineare come ReLU).
Anche altre funzioni non lineari come la tangente iperbolica o la sigmoide possono essere utilizzate anziché ReLU, ma ReLU si è dimostrata più efficace nella maggior parte delle situazioni.

Pooling Step Lo Spatial Pooling (anche chiamato subsampling or downsampling) riduce la dimensionalità di ogni mappa delle funzionalità ma mantiene le informazioni più importanti. Il pooling nello spazio può essere di diversi tipi: Max, Media, Somma, ecc.
Nel caso di Max Pooling, si definisce un intorno nello spazio (ad esempio una finestra 2x2) e si prende l'elemento più grande dalla mappa funzionale rettificata all'interno di quella finestra. Invece di prendere l'elemento più grande possiamo anche prendere la media (Average Pooling) o la somma di tutti gli elementi in quella finestra. Nella pratica, Max Pooling ha dimostrato di funzionare meglio.
La Figura 2.12 mostra un esempio di operazione Max Pooling su una Rectified Feature map (ottenuta applicando la convoluzione + l'operazione ReLU) usando una finestra 2x2.
La funzione di Pooling consiste nel ridurre progressivamente la dimensione della rappresentazione dell' input rendendola più piccola e gestibile. Questa operazione riduce il numero di parametri e calcoli della rete, controllando quindi l'overfitting. Rende la rete invariante a piccole trasformazioni e distorsioni nell'immagine di input (una piccola distorsione dell'ingresso non cambierà l'output di Pooling - poiché prendiamo il valore massimo/medio in una finestra locale). Il Pooling quindi ci aiuta ad arrivare ad una rappresentazione quasi invariante della nostra immagine: in questo modo possiamo rilevare gli oggetti in un'immagine indipendentemente da dove si trovano.

Fully Connected Layer Il livello Fully Connected è un tradizionale Percettrone Multi Layer che utilizza una funzione di attivazione softmax nel livello di output (altri classificatori come SVM possono essere usati, ma useremo Softmax). Il termine "Fully Connected"

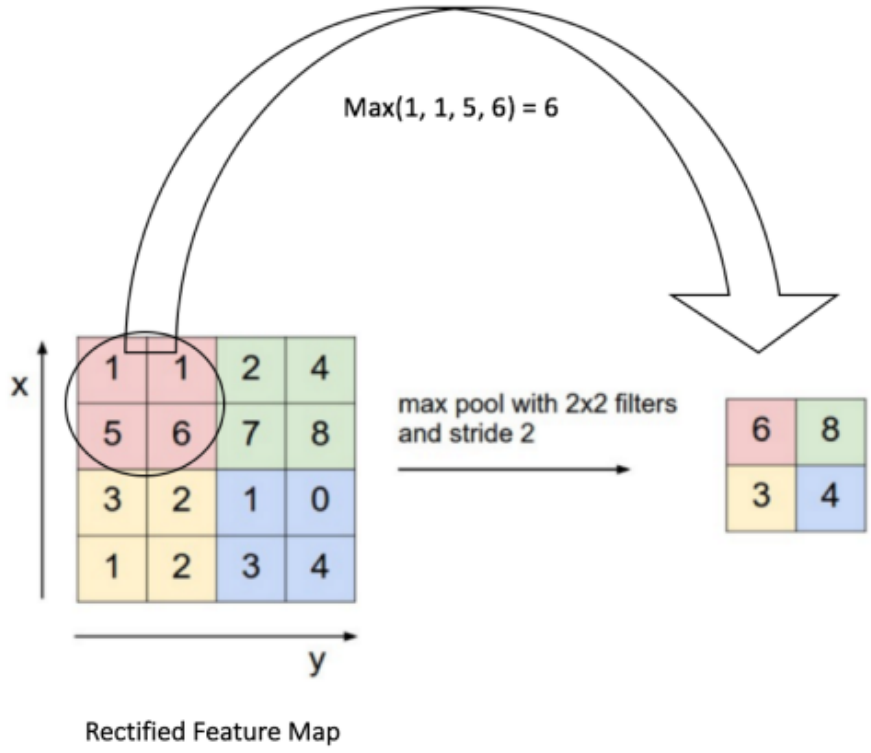


Figura 2.12: Max Pooling

implica che ogni neurone nello strato precedente sia collegato ad ogni neurone sul livello successivo.
L'output dei livelli convoluzionali e di pooling rappresentano le caratteristiche di alto livello dell'immagine di input. Lo scopo del livello Fully Connected è di utilizzare queste funzionalità per classificare l'immagine di input in varie classi in base al set di dati di addestramento. Ad esempio, l'attività di classificazione dell'immagine che abbiamo impostato dispone di quattro possibili uscite come mostrato nella Figura 2.13 (si noti che la Figura 2.13 non mostra le connessioni tra i nodi del livello Fully Connected).

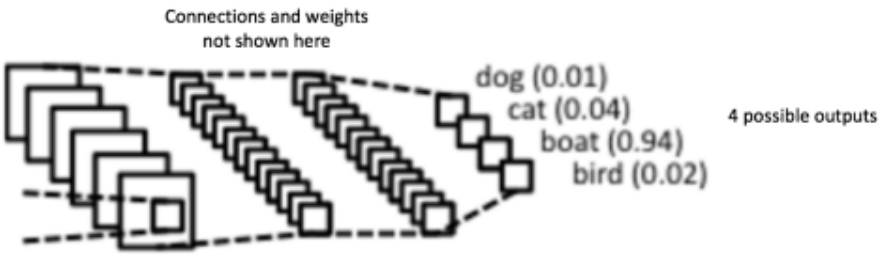


Figura 2.13: Livello Fully Connected - ogni nodo è connesso ad ogni altro nodo nel livello adiacente

Oltre alla classificazione, aggiungere un livello completamente collegato è anche un modo (di solito) economico per imparare combinazioni non lineari di queste funzionalità.

La maggior parte delle funzionalità degli strati convoluzionali e di pooling può essere utile per l'attività di classificazione, ma le combinazioni di queste funzionalità potrebbero essere ancora migliori.

La somma delle probabilità di uscita dal livello Fully Connected è 1. Ciò è garantito utilizzando Softmax come funzione di attivazione nel livello di uscita. La funzione Softmax prende un vettore di punteggi arbitrari di valori reali e lo trasforma in un vettore di valori tra zero e uno la cui somma è uno.

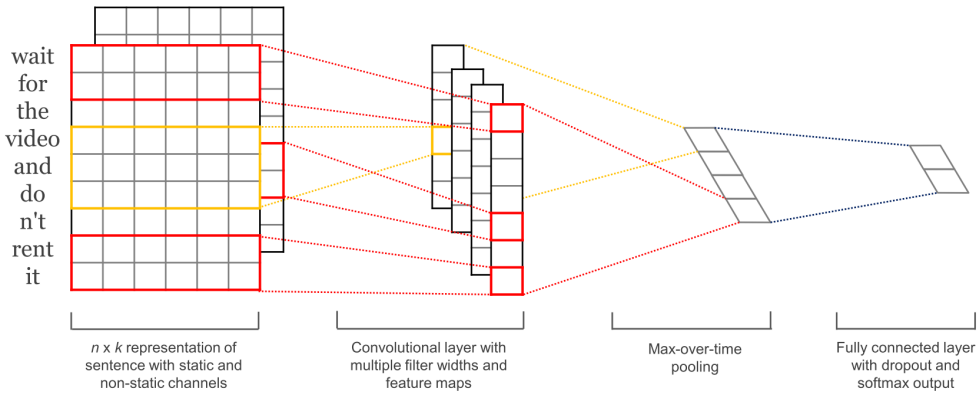


Figura 2.14: Convolutional Neural Network per la Sentiment Classification

2.11.7 Neural Turing machines

Le Neural Turing Machines (Alex Graves, Greg Wayne & Ivo Danihelka, 2014)[23] combinano le capacità di matching dei modelli delle reti neurali con la potenza algoritmica dei computer programmabili. Una NTM ha un controller della rete neurale accoppiato a risorse di memoria esterne, che interagisce con i meccanismi di focus dell'attenzione. I meccanismi di interazione con la memoria sono differenziabili, permettendo di ottimizzare il sistema usando la discesa di gradiente. Una NTM con un controller di rete LSTM può inferire semplici algoritmi quali la copia, l'ordinamento dagli esempi di input e di output.
I Differential Neural Computers rappresentano un'evoluzione delle Neural Turing Machines. Sono dotati meccanismi di attenzione che controllano dove la memoria è attiva ed in generale raggiungono prestazioni migliori.

2.11.8 Memory Networks

L'architettura delle Memory Networks (Jason Weston, Sumit Chopra & Antoine Bordes, 2015)[24] combina componenti inferenziali a componenti di memoria a lungo termine.

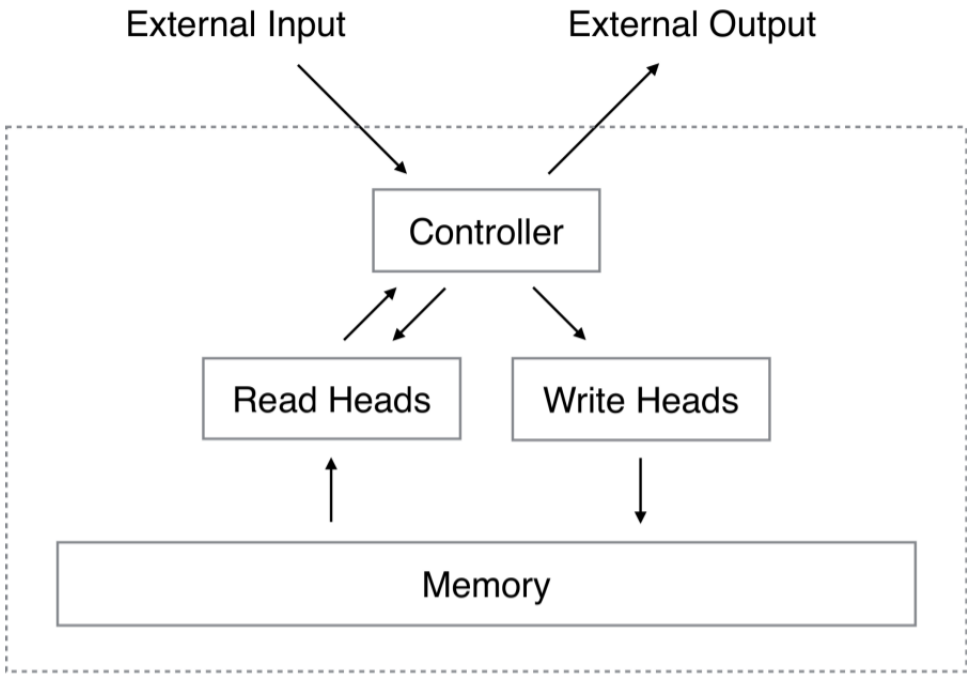


Figura 2.15: Neural Turing Machine

Queste reti sono in grado di imparare ad usare entrambi i tipi di compenenti in modo congiunto. La memoria a lungo termine può essere letta e scritta, al fine di effettuare predizioni successivamente.
Questo modello è stato sperimentato nel contesto del question answering (QA) dove la memoria a lungo termine funge efficacemente da conoscenza di base, e l'output è rappresentato da una risposta testuale.
Le Memory Networks sono state valutate su una serie numerosa di compiti di QA ed anche su alcuni compiti più complessi per simulare la loro applicazione nel mondo reale.
La trattazione "Memory Networks" di Jason Weston, Sumit Chopra & Antoine Bordes mostra la padronanza di queste reti nel ragionamento concatenando più frasi di supporto alla risposta a domande che richiedono la comprensione del significato dei verbi.
L'idea centrale di questo modello di reti è la combinazione di strategie di apprendimento di successo nella letteratura del Machine Learning a un componente di memoria che può essere letto e scritto.
Il modello è addestrato per imparare ad utilizzare efficacemente il componente di memoria.

2.11.9 End-To-End Memory Networks

È una forma di Memory Network ma a differenza del modello citato in precedenza, è addestrato end-to-end e quindi richiede meno supervisione durante l'addestramento, rendendo l'architettura più generale ed applicabile in contesti reali. La flessibilità del modello permette di essere applicato a diversi compiti come il question answering e la modellazione del linguaggio. Nel lavoro di Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston & Rob Fergus (End-To-End Memory Networks, 2015)[25] vengono riportate performance comparabili alle reti RNN e LSTM sui dataset Penn TreeBank e Text8. In entrambi i dataset è stato mostrato che il concetto chiave di passi computazionali multipli ha migliorato i risultati. In questa trattazione viene mostrato come una rete neurale con una memoria esplicita e un meccanismo di attenzione ricorrente per leggere la memoria possano essere addestrati via backpropagation su diversi compiti dal QA alla modellazione del linguaggio.

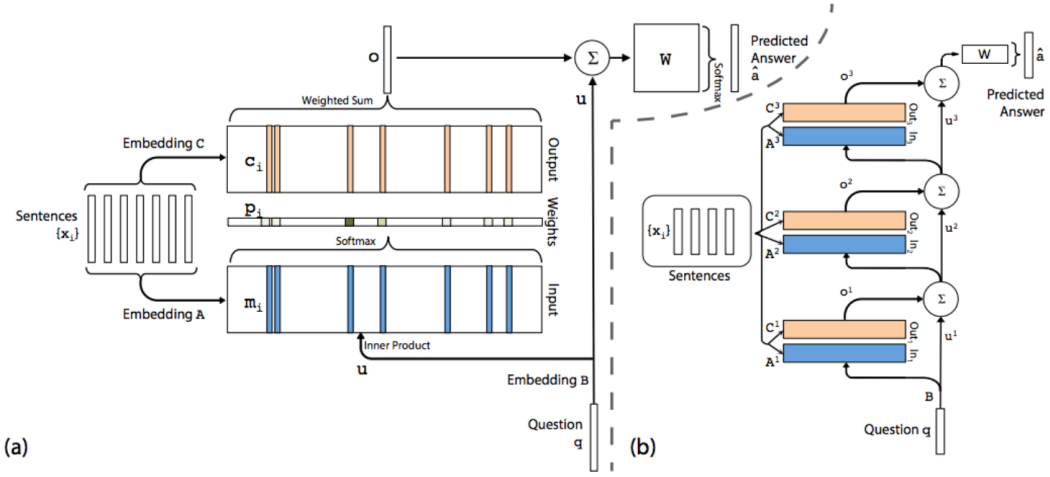


Figura 2.16: Architettura di una Memory Network End-To-End

2.11.10 Differentiable Neural Computers

I DNC (Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu & Demis Hassabis, 2016)[26] consistono in una rete neurale che può leggere e scrivere da una matrice di memoria esterna, analogamente alla memoria ad accesso casuale in un computer convenzionale.
Come quest'ultimo, i DNC possono usare la propria memoria per rappresentare e manipolare strutture dati complesse, ma come una rete neurale, sono in grado di imparare a fare questo dai dati. Quando sono addestrati tramite apprendimento supervisionato, un DNC può

rispondere con successo a domande sintetiche atte ad emulare problemi di ragionamento ed inferenza nel linguaggio naturale.
È stato mostrato che questo modello può imparare ad eseguire algoritmi come la ricerca del percorso più breve tra punti specificati e l'inferenza di collegamenti mancanti in grafi generati in modo casuale, e successivamente riescono a generalizzare questi compiti a grafi specifici come la rete dei trasporti ed alberi genealogici.
I risultati ottenuti in letteratura con questo modello di rete neurale mostrano la capacità dei DNC di risolvere compiti complessi e strutturati per i quali le reti neurali senza memoria esterna di lettura/scrittura non sono efficaci. Alla base del DNC vi è quindi l'Intelligenza Artificiale (controller) addestrata ad interpretare la memoria, e responsabile quindi della lettura della memoria, della produzione dell'interpretazione, ma anche della scrittura in memoria, in quanto la nuova interpretazione dei dati deve venir memorizzata.
La memoria è una serie di locazioni, ognuna che può memorizzare un vettore di informazioni. Il controller decide quando scrivere e dove scrivere nella memoria, può scegliere di scrivere in un nuovo spazio di memoria o sopra ad uno già utilizzato, aggiornando quindi le informazioni. Se la memoria è piena, il controller può decidere quali informazioni eliminare e quindi quali locazioni liberare. Ogni locazione di memoria rappresenta quindi un "ricordo" e come tale è connessa ad altre locazioni, come i ricordi sono connessi l'uno all'altro.
Il controller si occupa poi di leggere i dati che memorizza, può scorrerli seguendo le connessioni, seguendo l'ordine temporale con cui sono state memorizzate (sia in avanti che indietro). La struttura stessa delle informazioni memorizzate finora, può suggerire al controller come memorizzare nuove informazioni, in modo che vengano inserite con una logica (come l'uomo, il Differentiable Neural Computer impara a memorizzare, riconoscendo quando ha sbagliato a memorizzare e quando ha memorizzato bene).

2.12 Addestramento

2.12.1 L'algoritmo di Backpropagation

L'algoritmo di backpropagation[9][18] è un algoritmo di apprendimento delle reti neurali. L'algoritmo confronta il valore in uscita del sistema con il valore obiettivo desiderato. Sulla base della differenza così calcolata (i.e. l'errore della rete), l'algoritmo modifica i pesi sinaptici della rete neurale, facendo convergere progressivamente il set dei valori di uscita verso quelli desiderati. Ad esempio, data una rete neurale con un solo nodo N e una sola entrata X, il sistema ha l'obiettivo di raggiungere un determinato livello di uscita Y_D. Al termine di ogni ciclo, l'algoritmo confronta il risultato ottenuto Y con quello desiderato Y_D e calcola un errore $Y_D - Y$ (e). La retroazione (feedback) consente all'algoritmo

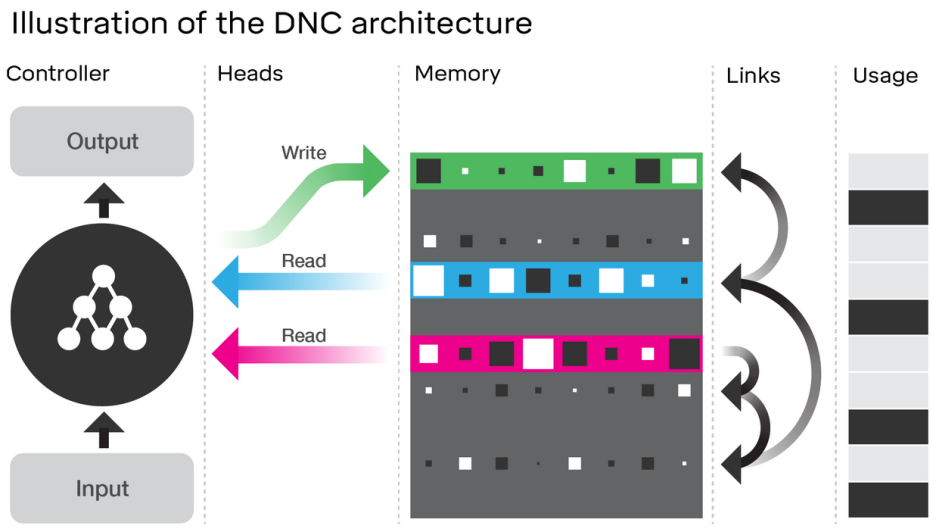


Figura 2.17: Architettura di un DNC

di utilizzare l'errore per aggiustare il peso sinaptico p_t ed eseguire una nuova iterazione (back-propagation o propagazione all'indietro).
L'aggiustamento del peso sinaptico è determinato dall'errore e da due parametri del sistema: il tasso di apprendimento e il momento. Entrambi i parametri possono variare da zero a uno (0-1). Il tasso di apprendimento influenza la velocità del processo di apprendimento. Quanto più è elevato il tasso di apprendimento, vicino al valore massimo uno, tanto più veloce è il processo di apprendimento, in quanto sono più grandi le variazioni da apportare sul peso sinaptico. Tuttavia, un tasso di apprendimento molto alto aumenta anche il rischio delle variazione erratiche (oscillazioni del risultato) e, quindi, l'instabilità del sistema. Per ridurre le oscillazioni e favorire la convergenza dei risultati, nella formula di aggiustamento del peso sinaptico si utilizza un secondo parametro (α), detto momento, anch'esso compreso tra zero e uno (0-1).

2.12.2 Descrizione del processo di training

Ogni neurone è composto da due unità. La prima unità aggiunge i prodotti di coefficienti di peso e i segnali di ingresso. La seconda unità realizza una funzione non lineare, chiamata funzione di attivazione del neurone. Il segnale e è il segnale di uscita della prima unità, e $y = f(e)$ è il segnale di uscita dell'elemento non lineare. Il segnale y è anche il segnale di uscita del neurone.
Per insegnare alla rete neurale abbiamo bisogno di un training set. Quest'ultimo è costituito da segnali di ingresso (x_1 e x_2) assegnati al corrispondente target (output desiderato) z. L'addestramento della rete è un processo iterativo. In ogni iterazione i coefficienti di peso dei nodi vengono modificati utilizzando nuovi dati provenienti dal training set. La modifica viene calcolata utilizzando l'algoritmo descritto di seguito: ogni passo dell'addestramento

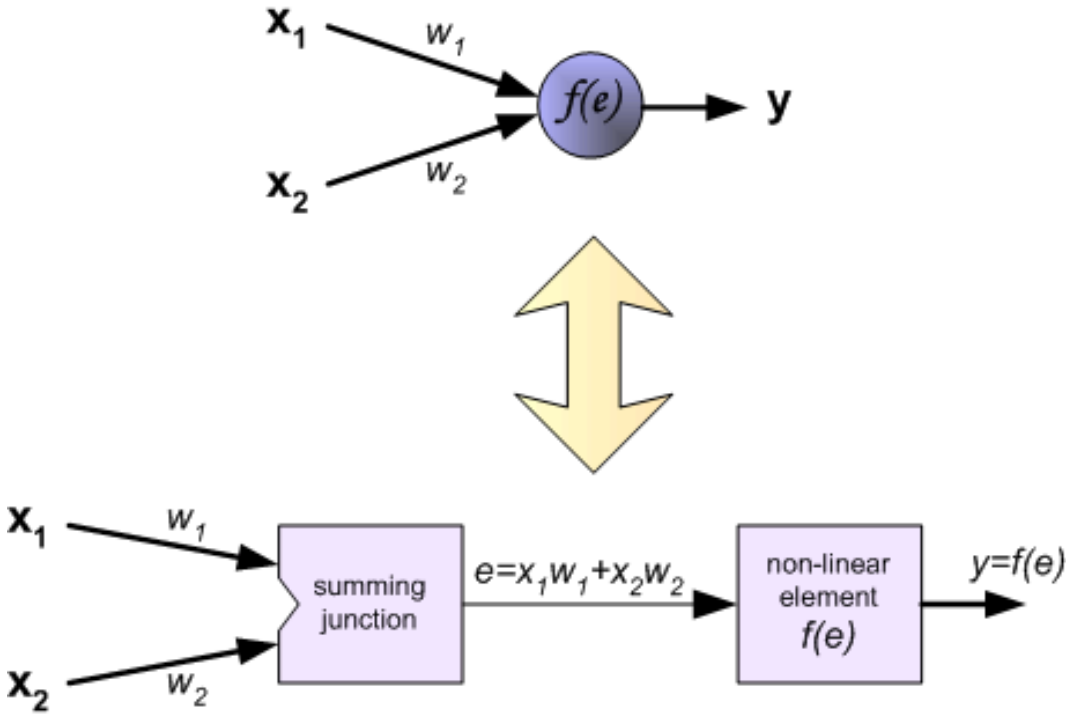


Figura 2.18: Composizione di un neurone artificiale

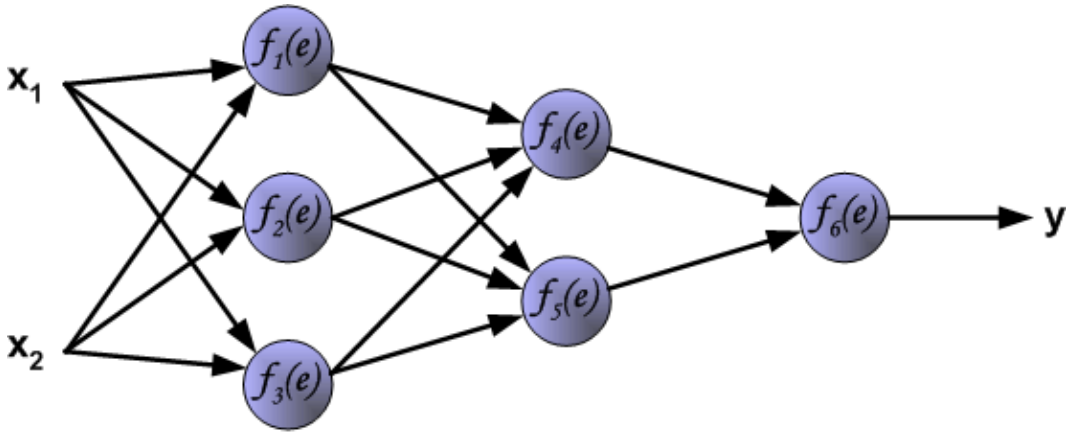


Figura 2.19: Rete neurale di riferimento per la spiegazione dell'addestramento

inizia con forzare la propagazione di entrambi i segnali di ingresso dal training set. Dopo questa fase possiamo determinare i valori dei segnali di uscita per ciascun neurone in ciascun livello di rete. Le immagini seguenti illustrano come il segnale si sta propagando attraverso la rete, i simboli $w_{(xm)n}$ rappresentano i pesi delle connessioni tra l'ingresso di rete x_m e il neurone n nel livello di input. Simboli y_n rappresentano il segnale di uscita del neurone n.

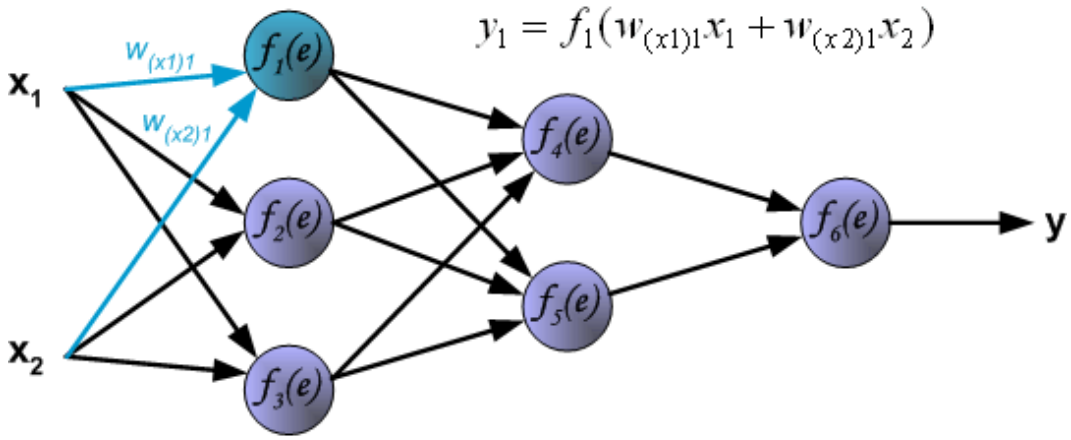


Figura 2.20: Primo step dell'addestramento della rete neurale

I simboli w_{mn} rappresentano i pesi delle connessioni tra l'output del neurone m e l'input del neurone n nello strato successivo.

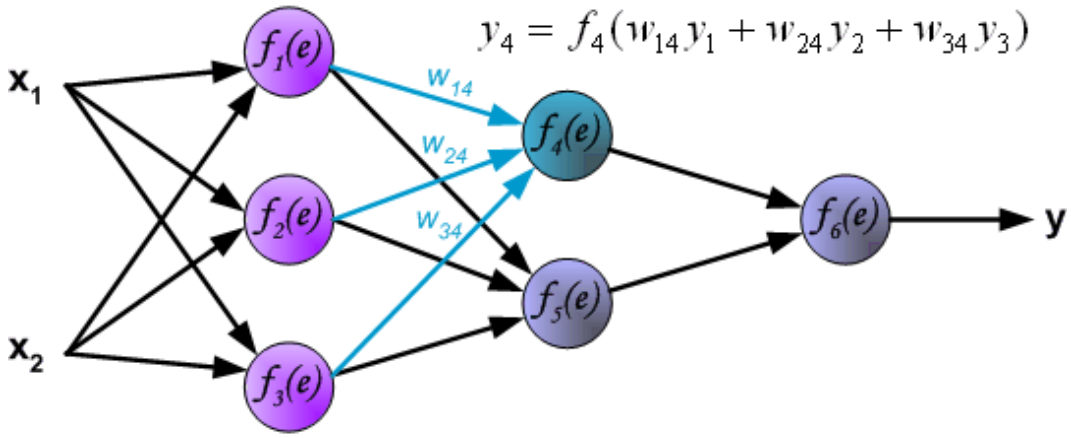


Figura 2.21: Propagazione dei segnali attraverso lo strato nascosto

Nel prossimo step dell'algoritmo, il segnale y della rete è confrontato con l'output desiderato, il quale è fornito del training set. La differenza tra i due valori è chiamata errore δ del neurone dello strato di output.
È impossibile calcolare direttamente l'errore del segnale per i neuroni interni, perchè l'output di questi neuroni è sconosciuto.

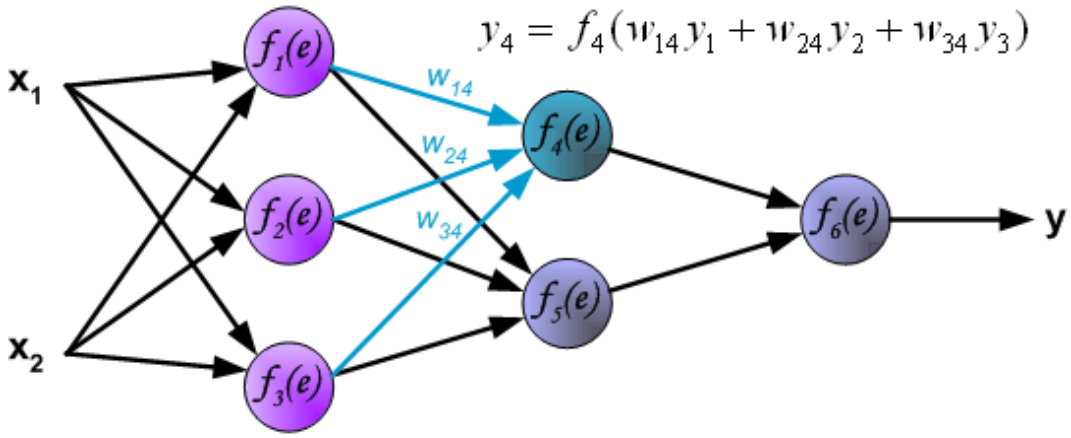


Figura 2.22: Propagazione dei segnali attraverso lo strato di output

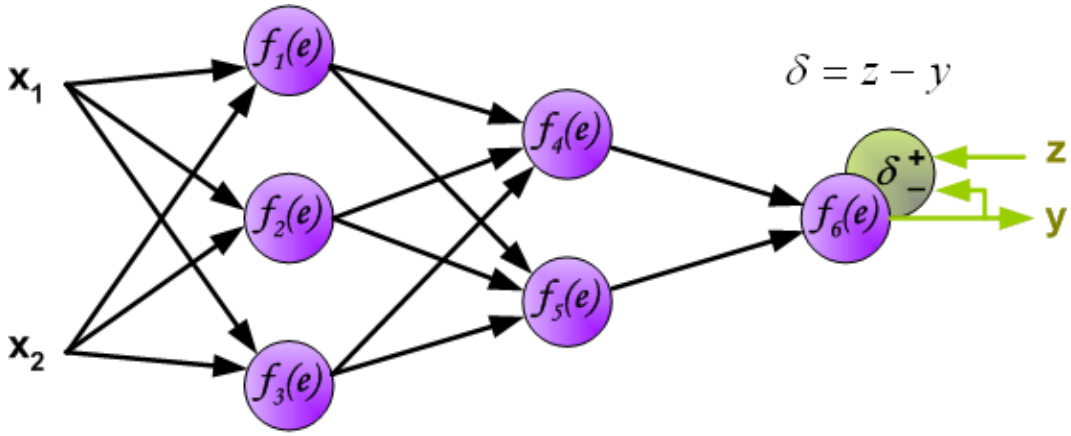


Figura 2.23: Il segnale di output y della rete è confrontato con il valore in output desiderato

Per molti anni il metodo per addestrare reti multilivello è rimasto sconosciuto. Solo a metà degli anni 80 l'algoritmo di backpropagation è stato elaborato. L'idea consiste nel propagare l'errore δ del segnale (calcolato ad ogni singolo step) indietro a tutti i neuroni, il cui output è stato l'input del neurone in questione.

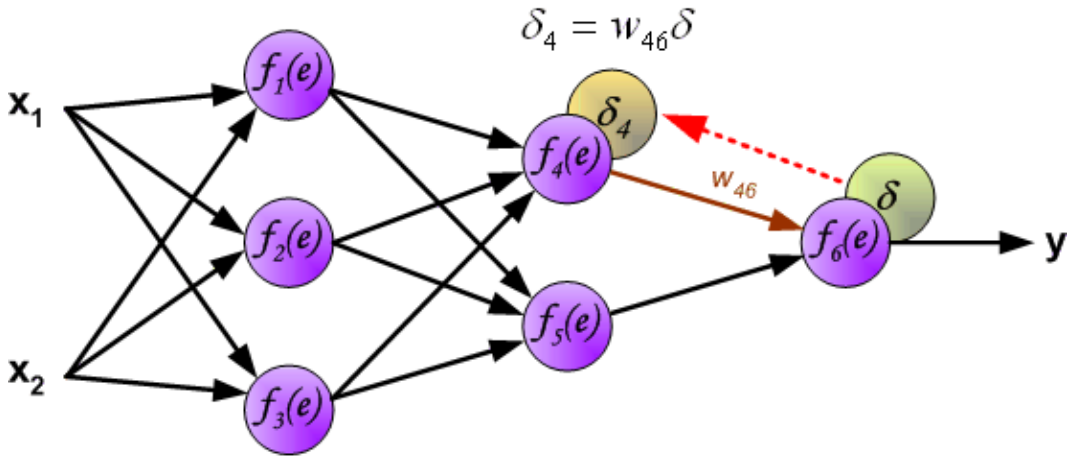


Figura 2.24: Retropropagazione dell'errore δ del segnale

I coefficienti dei pesi w_{mn} usati per propagare gli errori indietro sono uguali a quelli utilizzati durante il calcolo dei valori di uscita. Viene modificata soltanto la direzione del flusso di dati (i segnali vengono propagati dall'uscita agli ingressi uno dopo l'altro). Questa tecnica viene utilizzata per tutti i livelli di rete. Se gli errori propagati provengono da più neuroni allora vengono sommati.

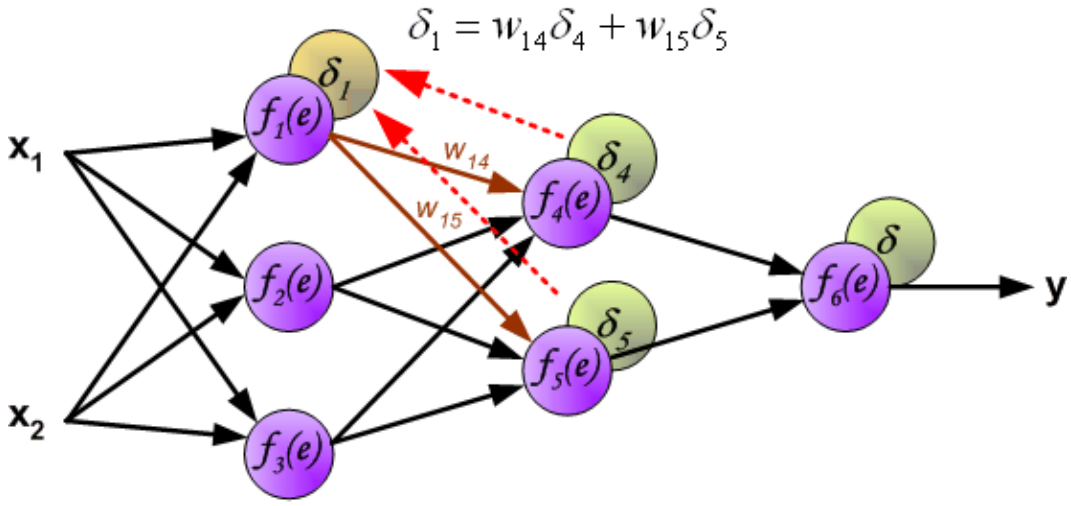


Figura 2.25: Gli errori propagati e provenienti da più neuroni vengono sommati

Quando viene calcolato l'errore del segnale per ogni neurone, i coefficienti dei pesi di ciascun nodo di ingresso del neurone possono essere modificati.

Nelle formule sottostanti $df(e)/de$ rappresenta la derivata della funzione di attivazione dei neuroni (di cui i pesi sono modificati).

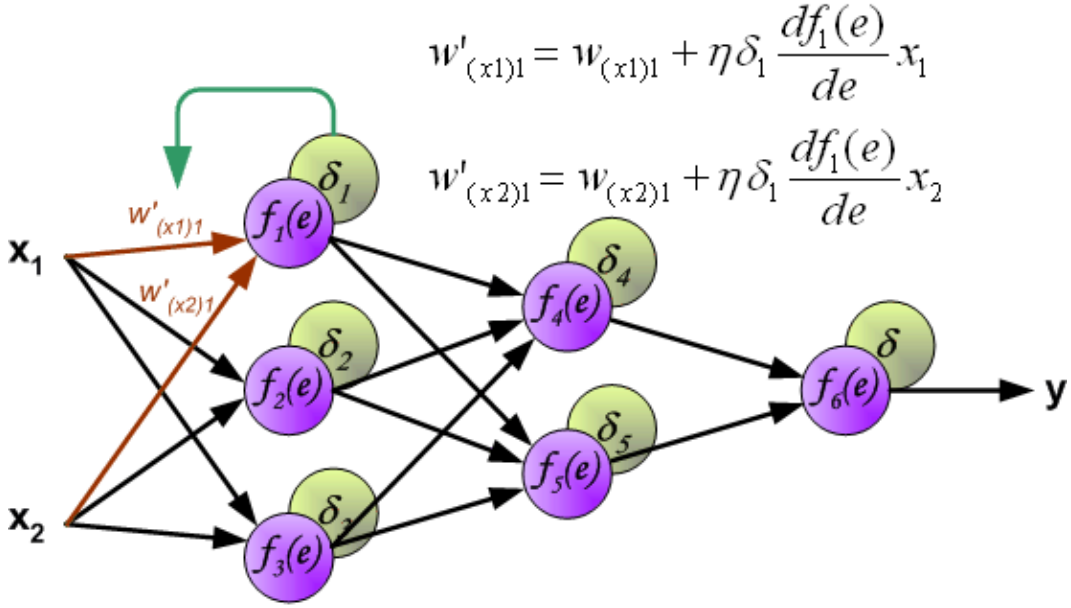


Figura 2.26: Aggiornamento dei pesi

Il coefficiente η influenza la velocità di addestramento della rete. Ci sono alcune tecniche per configurare questo parametro. Il primo metodo è quello di avviare il processo di insegnamento con un grande valore del parametro e mentre vengono stabiliti i coefficienti dei pesi, il parametro viene gradualmente diminuito. Il secondo metodo, più complicato, lancia l'addestramento con un valore basso del parametro. Durante il processo di training il parametro viene aumentato quando l'addestramento è avanzato e poi diminuito nuovamente nella fase finale.

2.12.3 Complessità dell'algoritmo

La complessità dell'algoritmo di backpropagation cresce con la dimensione della rete neurale e nell'algoritmo originale, su un singolo processore, è pari a $O(W^3)$; dove W è il numero dei pesi della rete. La complessità è giustificata dal numero di passaggi richiesti per aggiornare i pesi delle connessioni del grafo. Il primo passo è il calcolo dell'errore propagando il segnale in input fino all'uscita della rete. Il secondo passo è la retropropagazione dell'errore alle connessioni con peso più basso. Il passo finale è l'aggiornamento di ogni peso. Quando la rete neurale è composta da pochi neuroni si verifica il problema dell'underfitting e il processo di apprendimento della rete potrebbe essere inefficace. Viceversa, quando la rete è composta da molti neuroni si verifica il problema dell'overfitting facendo diventare più difficoltoso il processo di generalizzazione.

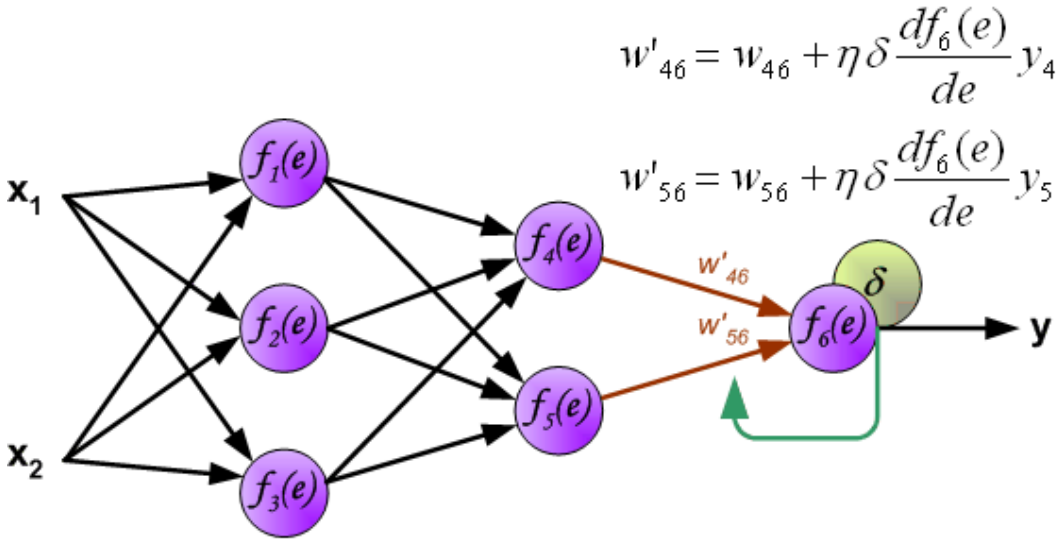


Figura 2.27: Passo finale di aggiornamento pesi e dell'algoritmo di Backpropagation

Capitolo 3

Dynamic Memory Networks

3.1 Introduzione

3.1.1 Definizione

Una Dynamic Memory Network (DMN) è un'architettura di rete neurale ottimizzata per problemi di Question Answering (QA). Dato un insieme di addestramento di sequenze di input e di domande, le DMN sono in grado di generare memorizzare episodi ed usarli per generare risposte rilevanti.
Nonostante i modelli classici Encoder-Decoder (Seq2Seq) possano risolvere i problemi di QA, la loro prestazione è limitata dalla piccola dimensione della loro "memoria". Quest'ultima è codificata dagli stati e pesi nello strato nascosto e riflette le informazioni passate tra codificatore e decodificatore. Questa limitazione diventa particolarmente evidente quando si tratta di sequenze molto lunghe di dati, come possiamo trovare in fonti come libri o video, dove i fatti salienti potrebbero verificarsi a lungo in più contesti molto diversi.
Si ottiene un miglioramento per questo problema memorizzando più stati nascosti e quindi utilizzando una strategia chiamata "meccanismo di attenzione" che si occupa di scegliere tra di essi. Ciò consente alla rete di fare riferimento alla sequenza di input, invece di forzarlo per codificare tutte le informazioni in un vettore a lunghezza fissa come Seq2Seq. Nelle sezioni successive verrà spiegato con maggiore precisione come il meccanismo di attenzione risulti utile per migliorare le prestazioni nel QA.
DMN raffina il meccanismo di attenzione, in modo che le domande inneschino un processo di attenzione iterativo che consenta al modello di focalizzare la propria attenzione sulla base degli input e sul risultato delle iterazioni precedenti.

3.1.2 Motivazioni delle DMNs

La maggioranza dei compiti nel Natural Language Processing può essere ricondotta ai problemi di Question Answering (QA). Le Dynamic Memory Networks (DMNs) rappresentano un architetura di reti neurali che processo le sequenze in input e le domande, memo-

rizzano episodi, e generano risposte rilevanti. Le DMN+ possono essere interamente addestrate e ottengono risultati che raggiungono lo stato dell'arte su diversi tipi di attività e set di dati: QA sul dataset bAbI di Facebook, text classification per la sentiment analysis sul dataset Stanford Sentiment Treebank) e la modellazione di sequenze per la codifica part-of-speech (WSJ-PTB). Il training per questi compiti diversi si basano esclusivamente su rappresentazioni vettoriali addestrate e sulle triplette input-domanda-risposta.

3.2 Architettura

In questa sezione viene fornita una panoramica sui moduli che compongono una Dynamic Memory Network. Nella sezione successiva verrà poi approfondito e formalizzato il funzionamento di ciascun modulo. Nella Figura 3.1 viene mostrato ad alto livello l'architettura di base di una DMN.

Modulo di memoria semantica Il modulo di memoria semantica (analogamente a una base di conoscenza) è costituito da vettori GloVe pre-addestrati che vengono utilizzati per creare sequenze di word embeddings dalle frasi di input. Questi vettori agiranno come input del modello.

Modulo di input Il modulo di input codifica gli insiemi di testo non elaborati estratti dal compito selezionato in rappresentazioni vettoriali distribuite. In questa trattazione, ci si concentra sui problemi legati al testo anche se, come verrà citato successivamente, l'input potrebbe anche essere una sequenza di immagini. Nel nostro caso, l'input può essere ad esempio: una frase, una lunga storia, una recensione di un film, un articolo di giornale o di Wikipedia.

Modulo di domanda Analogamente al modulo di input, il modulo di domanda codifica la domanda del compito scelto in una distribuzione rappresentazione vettoriale. Ad esempio, nel caso del QA, la domanda può essere una frase come "Dove si trovava Marco prima?". La rappresentazione viene fornita al modulo di memoria episodica, e costituisce la base, o lo stato iniziale, sul quale il predetto modulo itera.

Modulo di memoria episodica Data una raccolta di rappresentazioni in input, il modulo di memoria episodico sceglie su quali parti dell'input concentrarsi attraversi il meccanismo di attenzione. Quindi produce una rappresentazione vettoriale della "memoria" acquisita tenendo conto della domanda e della precedente memoria salvata. Ogni iterazione fornisce al modulo nuove informazioni rilevanti sull'input. In altre parole, il modulo ha la capacità

di recuperare nuove informazioni, in forma di rappresentazione dell'input, che si pensava fossero irrilevanti nelle iterazioni precedenti.

Modulo di risposta Alla fine del processo, questo modulo genera una risposta appropriata sulla base del vettore finale di memoria recuperato dal modulo precedente.

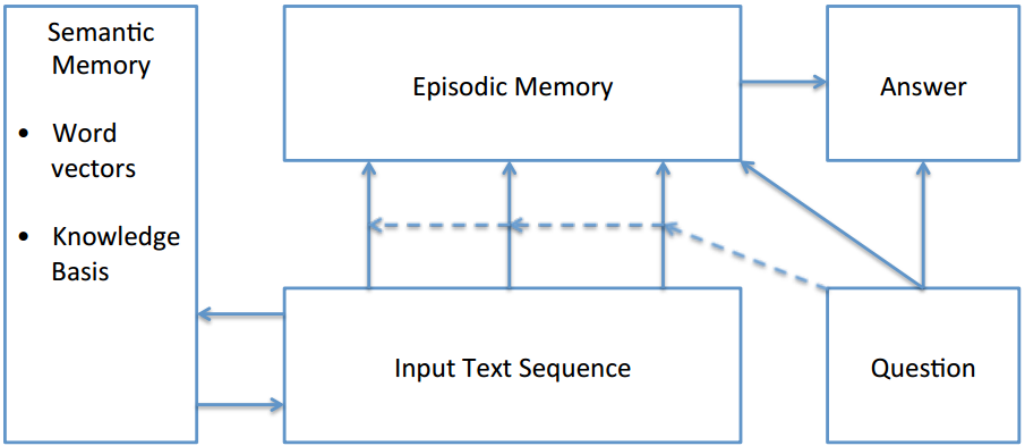


Figura 3.1: Architettura ad alto livello di una DMN

3.3 Specifiche dei moduli

Di seguito viene approfondito e formalizzato il funzionamento dei quattro principali moduli elencati nella sezione precedente, fornendo le intuizioni alla base della relativa formulazione.

3.3.1 Modulo di input

Nei problemi di Natural Language Processing, l'input è costituito da una sequenza di T_I parole $w_1, ..., w_{T_I}$. Un modo di codificare la sequenza in input è attraverso una Recurrent Neural Network alla quale vengono passati i word embeddings. Ad ogni step temporale t, la rete aggiorna il suo stato nascosto $h_t = RNN(L[w_t], h_{t-1})$, dove L è la matrice di embedding e w_t è l'indice della t-esima parola della sequenza di input. Nei casi in cui la sequenza di input sia una singola frase, il modulo di input emette gli stati nascosti della rete ricorrente.
Nei casi in cui la sequenza di input è un elenco di frasi, le frasi vengono concatenate in una lunga lista di parole Token, inserendo dopo ogni frase un token di fine-frase. Gli stati nascosti per ciascuno dei token di fine frase sono quindi le rappresentazioni finali del modulo di input. Nelle sezioni successive, si indica l'output del modulo di input come sequenza di T_C

rappresentazioni di fatti c, per cui c_t indica l'elemento t-esimo nella sequenza di output del modulo di input.
Si noti che nel caso in cui l'input sia una sola frase, $T_C = T_I$, i.e. che il numero di rappresentazioni in output è uguale al numero di parole nella frase. Nel caso in cui l'input sia un elenco di frasi, T_C è uguale al numero di frasi.

Scelta della rete ricorrente Negli esperimenti, viene utilizzata una Gate Recurrent Unit (GRU) (Cho et al., 2014a; Chung Et al., 2014). Nel famoso paper "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing" (Salesforce Inc., CA USA)[27] viene anche discussa la scelta di utilizzare le LSTM (Hochreiter e Schmidhuber, 1997), ma gli stessi autori hanno ottenuto risultati simili alle GRUs con lo svantaggio di una computazione più costosa. Sia le LSTM sia le GRUs performano molto meglio delle RNN standard grazie al vantaggio principale di avere dei gates che permettono al modello di soffrire meno del problema della scomparsa del gradiente (Hochreiter e Schmidhuber, 1997).

3.3.2 Modulo di domanda

Analogamente alla sequenza di input, anche la domanda è comunemente fornita come sequenza di parole nel Natural Language Processing. Come nel modulo di input, la domanda è codificata attraverso una RNN. Data una domanda di T_Q parole, gli stati nascosti per il question encoder al momento T è dato da $q_t = GRU(L[w_t^Q], q_{t-1})$, L rappresenta la matrice di word embeddings come nella sezione precedente e w_t^Q rappresenta l'indice di parola della t-esima parola nella domanda. La matrice di word embeddings è condivisa tra il modulo di ingresso e il modulo di domanda. A differenza del modulo di ingresso, il modulo di domanda produce come uscita lo stato nascosto finale dell'encoder di rete ricorrente: $q = q_{T_Q}$.

3.3.3 Modulo di memoria episodica

Nella sua forma generale, il modulo di memoria episodica è composto da una memoria interna, un meccanismo di attenzione e una rete neurale ricorrente per aggiornare la sua memoria. Durante ogni iterazione, il meccanismo di attenzione analizza le rappresentazioni dei fatti c usando una funzione di gating (descritta successivamente) mentre consida la rappresentazione della domanda q e la memoria precedente m^{i-1} per produrre un episodio e^i. L'episodio è poi usato, insieme alla memoria pregressa m^{i-1}, per aggiornare la memoria episodica $m^i = GRU(e^i, m^{i-1})$. Lo stato iniziale di questa GRU è inizializzato con il vettore della domanda: $m^0 = q$. Per alcuni compiti il modulo di memoria episodica performa meglio analizzando l'input a passi multipli. Dopo T_M passi, la memoria finale m^{T_M} sarà fornita al modulo di risposta.

Episodi multipli La natura iterativa di questo modulo permette di analizzare diversi input ad ogni passo. Permette anche di realizzare una specie di inferenza transitiva, dal momento che il primo passo potrebbe non fornire le informazioni richieste.

Meccanismo di attenzione Ad ogni passo i, il meccanismo prende come input un fatto candidato c_t, la memoria precedente m^{i-1}, e la domanda q per computare un gate:
$g_t^i = G(c_t, m^{i-1}, q)$.
La funzione di punteggio G prende in input un set di feature $z(c, m, q)$ e produce un punteggio scalare. Senza entrare nei formalismi matematici possiamo dire che un vettore di feature cattura una serie di similarità tra i vettori di input, memoria e domanda. La funzione G è una semplice rete neurale feed-forward a due strati:

$$G(c, m, q) = \sigma(W^2 \tanh(W^1 z(c, m, q) + b^1) + b^2). \tag{3.1}$$

Alcuni dataset come il bAbI di Facebook, specificano quali fatti sono importanti per una certa domanda. In questi casi, il meccanismo di attenzione della funzione G può essere addestrato in modo supervisionato con una funzione di costo cross-entropy.

Meccanismo di aggiornamento della memoria Per calcolare l'episodio per il passo i, viene impiegata una rete GRU modificata in base alla sequenza di input $c_1, ..., c_{T_C}$, pesata dai gates g^i. Il vettore episodico che è fornito al modulo di risposta è lo stato finale della rete GRU. L'equazione per aggiornare gli stati nascosti della rete GRU al tempo t e l'equazione per calcolare l'episodio, sono rispettivamente:

$$h_t^i = g_t^i GRU(c_t, h_{t-1}^i) + (1 - g_t^i) h_{t-1}^i \tag{3.2}$$

$$e^i = h_{T_C}^i \tag{3.3}$$

Criterio di fine iterazione Il modulo di memoria episodica ha anche un segnale per fermare le iterazioni sull'input. Per fare questo, viene aggiunta una rappresentazione speciale, ad indicare la fine dei passi, all'input e viene fermato il processo iterativo di attenzione se questa rappresentazione è scelta dalla funzione di gate. Per i dataset senza supervisione esplicita, viene fissato un numero massimo di iterazioni. L'intero modulo è differenziabile.

3.3.4 Modulo di risposta

Il modulo di risposta genera una risposta dato un vettore. A seconda del tipo di compito, il modulo di risposta è attivato o alla terminazione della memoria episodica o ad ogni step temporale. Viene utilizzata un'altra rete GRU il cui stato iniziale è impostato alla memoria

finale $a_0 = m^{T_M}$. Ad ogni step temporale, riceve in input la domanda q, l'ultimo stato nascosto a_{t-1} e l'output previsto y_{t-1}.

$$y_t = softmax(W^{(a)} a_t) \tag{3.4}$$

$$a_t = GRU([y_{t-1}, q], a_{t-1}) \tag{3.5}$$

Nell'equazione 3.5 vengono concatenati i vettori dell'ultima parola generata e della domanda, come input ad ogni step temporale. L'output è addestrato tramite la funzione di errore cross-entropy e la classificazione della corretta sequenza aggiunta ad uno speciale token di fine frase.

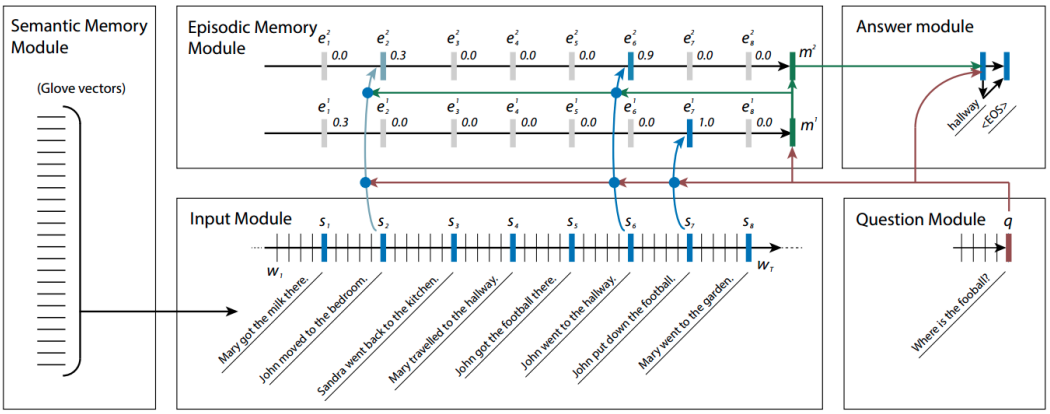


Figura 3.2: Architettura dettagliata di una DMN

3.4 Addestramento

L'addestramento si può considerare un problema di classificazione supervisionata volto a ridurre al minimo l'errore (rappresentato dalla funzione cross-entropy) della sequenza di risposta. Per i dataset con supervisione dei gate, come bAbI, aggiungiamo l'errore cross-entropy dei gates al costo complessivo. Poichè tutti i moduli comunicano attraverso rappresentazioni vettoriali e vari tipi di reti neurali deep differenziabili, l'intero modello delle DMN può essere addestrato via backpropagation e discesa del gradiente.

3.5 DMN Plus

3.5.1 Motivazioni e differenze

Le architetture di reti neurali con memoria e i meccanismi di attenzione mostrano una certa capacità di ragionamento necessaria per rispondere a domande. L'architettura DMN ha ottenuto un'elevata precisione su una varietà di compiti linguistici. Tuttavia, non è stato mostrato che l'architettura raggiungesse risultati brillanti nel Question Answering quando i fatti di supporto non sono etichettati durante l'addestramento o che potesse essere applicata ad altre modalità di input come le immagini. Sulla base di un'analisi delle DMN, nel paper "Dynamic Memory Networks for Visual and Textual Question Answering" (Caiming Xiong, Stephen Merity, Richard Socher di MetaMind, Palo Alto, CA USA)[20] vengono proposti diversi miglioramenti al modulo di memoria ed input. Insieme a questi cambiamenti viene introdotto un nuovo modulo di input per le immagini che permette al sistema di poter rispondere a domande riguardanti figure. Il modello DMN+ migliora lo stato dell'arte sia nel dataset Visual Answering sia nel dataset text-answering bAbI-10k senza supervisione dei fatti di supporto.
Nella sezione successiva vengono analizzati i componenti innovativi delle DMN+, in particolare il modulo di input e di memoria, i quali permettono di migliorare la risposta alle domande. Il nuovo modulo di input utilizza un encoder a due livelli con un lettore di frase e un livello di fusione dell'input che permette all'informazione di navigare in entrambe le direzioni tra le frasi. Per quanto concerne la memoria, viene proposta una modifica alle unità ricorrenti GRU (Chung et al., 2014). La nuova formulazione delle GRU incorpora gates di attenzione che vengono calcolati utilizzando la conoscenza globale sui fatti.
A differenza del modello originale, le DMN+ non richiedono che i fatti di supporto (cioè i fatti che sono rilevanti per rispondere a una domanda particolare) siano etichettati durante l'addestramento. Il modello impara a selezionare i fatti importanti da un set più grande.
Inoltre viene introdotto un nuovo modulo di input per rappresentare le immagini (non approfondito nella sezione successiva). Questo modulo è compatibile con il resto dell'architettura DMN e il suo output viene fornito al modulo di memoria. Nel lavoro precedentemente citato gli autori mostrano anche che le modifiche nel modulo di memoria che migliorano la risposta testuale, portano a migliorare anche la risposta ad una domanda visuale.

3.5.2 Componenti caratteristici

Successivamente vengono analizzati e paragonate diverse scelte di progettazione dei componenti cruciali: la rappresentazione dell'input, il meccanismo di attenzione e l'aggiornamento della memoria. Viene così ottenuto il modello DMN + finale che raggiunge la massima precisione sul dataset bAbI-10k senza supervisione e sul dataset VQA (Antol et al.,

2015). Parecchie scelte di progettazione sono motivate dall'intuizione e dai miglioramenti dell'accuratezza ottenuta sul relativo dataset.

Modulo di input per il Text QA Nel modello di DMN proposto da Kumar et al. (2015), una singola rete GRU viene utilizzata per elaborare tutte le parole della storia, estraendo le rappresentazione delle frasi mediante la memorizzazione degli stati nascosti prodotti alla fine dei marker della frase. La GRU fornisce anche un componente temporale che permette ad una frase di conoscere il contenuto delle frasi precedenti. Mentre questo modulo di ingresso funzionava bene per bAbI-1k con supporto dei fatti, come riportato da Kumar et al. (2015), non ha performato bene sul dataset bAbI-10k senza supervisione dei fatti. Vi sono due motivi principali per questa disparità di performance, aggravata dalla rimozione del supporto dei fatti. In primo luogo, la rete GRU consente solo alle frasi di formare il contesto in base alle frasi precedenti, ma non a quelle seguenti. Ciò impedisce la propagazione delle informazioni dalle frasi future. In secondo luogo, le frasi di supporto possono essere troppo lontane l'une dalle altre per consentire loro di interagire attraverso il livello della rete GRU.

Livello di fusione dell'input Nel modello DMN+, viene proposto di sostituire la singola rete GRU con due diversi componenti.
Il primo componente è un lettore di frasi, responsabile solamente di codificare le parole in un embedding di frase.
Il secondo componente è un livello di fusione dell'input che permette le interazioni tra frasi. Per questo livello viene utilizzato la GRU bidirezionale per permettere all'informazione di essere utilizzata sia dalle frasi passate sia dalle frasi future.
Dal momento che i gradienti non devono essere necessariamente propagati tra le frasi, il livello di fusione permette alle frasi distanti di supporto, di avere una maggiore interazione.
La Figura 3.3 illustra il modulo di input (nella parte sinistra dell'immagine) formato dal Sentence reader e dall'Input Fusion Layer.
Ogni codifica di una frase f_i è l'output di uno schema che riceve in input i token di parola $[w_1^i, ..., w_{M_i}^i]$, dove M_i è la lunghezza della frase. Il lettore di frasi può essere basato su una varietà di schemi di codifica.
Nel paper di riferimento viene scelta la codifica posizionale di Sukhbaatar et al. (2015) ma sono state considerate anche le reti GRUs e LSTMs. Queste due ultime due soluzioni però, richiedono più risorse computazionali e sono inclini al problema dell'overfitting se non sono impiegati procedure di supporto, come la ricostruzione della frase originale.
Il livello di fusione dell'input riceve in input i fatti e permette uno scambio di informazioni

tra questi utilizzando una rete GRU bidirezionale.

$$\overrightarrow{f_i} = GRU_{fwd}(f_i, \overrightarrow{f_{i-1}}) \tag{3.6}$$

$$\overleftarrow{f_i} = GRU_{bwd}(f_i, \overleftarrow{f_{i+1}}) \tag{3.7}$$

$$\overleftrightarrow{f_i} = \overleftarrow{f_i} + \overrightarrow{f_i} \tag{3.8}$$

dove f_i è il fatto in input allo step i. $\overrightarrow{f_i}$ è lo stato nascosto della GRU forward allo step i, e $\overleftarrow{f_i}$ è lo stato nascosto della GRU backward allo step i.
Questo meccanismo permette alle informazioni di contesto di valorizzare $\overleftrightarrow{f_i}$ in base ai fatti futuri e passati.

Modulo di memoria episodica IL modulo di memoria episodica recupera informazioni dai fatti in input, $\overleftrightarrow{F} = [\overleftrightarrow{f_1}, ..., \overleftrightarrow{f_N}]$ focalizzando l'attenzione su una sottoparte di questi fatti. L'attenzione è implementata associando un valore scalare, il gate di attenzione g_i^t, ad ogni fatto $\overleftrightarrow{f_i}$ al passo t. Il gate è calcolato sulla base delle interazioni tra il fatto ed entrambe le rappresentazioni della domanda e dello stato della memoria episodica.

$$z_i^t = [\overleftrightarrow{f_i} \circ q; \overleftrightarrow{f_i} \circ m^{t-1}; |\overleftrightarrow{f_i} - q|; |\overleftrightarrow{f_i} - m^{t-1}|] \tag{3.9}$$

$$Z_i^t = W^{(2)} \tanh(W^{(1)} z_i^t + b^{(1)}) + b^{(2)} \tag{3.10}$$

$$g_i^t = \frac{\exp(Z_i^t)}{\sum_{k=1}^{M_i} \exp(Z_k^t)} \tag{3.11}$$

dove $\overleftrightarrow{f_i}$ è l'i-esimo fatto, m^{t-1} è il precedente episodio, q è la domanda originale, $\circ$ è la moltiplicazione per elemento, $|\cdot|$ è il valore assoluto, e ; rappresenta la concatenazione dei vettori.
Le DMN implementate da Kumar et. al (2015) utilizzano un set di interazioni più complesso all'interno di z. Quest'ultimo contiene termini aggiuntivi che dopo un'analisi iniziale sono stati considerati non necessari da Xiong, Merity e Socher (2016).

Meccanismo di attenzione Una volta calcolato il gate di attenzione g_i^t, viene utilizzato un meccanismo di attenzione per estrarre un vettore di contesto c^t basato sul corrente focus. Gli autori del paper di riferimento hanno posto l'attenzione su due tipi di focus: Soft Attention e Attention Based GRU. Approfondiremo il secondo tipo in quanto migliora le performance ed è stato scelto nel modello delle DMN+.

Attention based GRU Al contrario del meccanismo di Soft Attention, l'Attention Based GRU è sensibile sia alla posizione sia all'ordine dei fatti in input $\overleftrightarrow{F}$.
Una RNN sarebbe vantaggiosa in questa situazione ma non potrebbe utilizzare il gate di attenzione calcolato nell'equazione 3.11. Viene quindi proposta una modifica all'architettura della rete GRU incorporando informazione dal meccanismo di attenzione.
Il gate di aggiornamento z_t nella prima equazione della Figura 2.7 decide quanta informazione mantenere per ogni dimensione dello stato nascosto e quanta dovrebbe essere aggiornata con l'input elaborato x_i nel corrente step temporale.
Dal momento che z_t è calcolato usando solo l'input corrente e lo stato nascosto dei precedenti step, non tiene in considerazione nessun tipo di conoscenza derivante dalla domanda o da un episodio precedente.
Sostituendo al gate di aggiornamento z_t nella rete GRU (Figura 2.7) con l'output del gate di attenzione g_i^t (Equazione 3.11) nella prima equazione in Figura 2.7, la rete GRU può ora usare il gate di attenzione per aggiornare il suo stato interno.

$$h_t = (1 - g_i^t) * h_{t-1} + g_i^t * \tilde{h}_i \tag{3.12}$$

Un importante considerazione è che g_i^t è uno scalare, generato usando un funzione di attivazione softmax, al contrario del vettore $z_t \in \mathbb{R}^{n_H}$, generato usando una funzione di attivazione sigmoide.
Questo ci permette di visualizzare facilmente come i gate di attenzione si attivano in base all'input nel visual QA. Anche se non è stato provato, si potrebbe sostituire la funzione softmax con la sigmoide nell'equazione 3.11 in modo che $g_i^t \in \mathbb{R}^{n_H}$.
Per produrre il vettore di contesto c^t usato per aggiornare lo stato della memoria episodica m^t, viene utilizzato lo stato nascosto finale di una GRU attention based.

Aggiornamento della memoria episodica Dopo ogni passo del meccanismo di attenzione, vogliamo aggiornare la memoria episodica m^{t-1} con il vettore c^t appena costruito, producendo m^t. Nelle DMN, è usata per questo scopo una rete GRU con lo stato nascosto iniziale impostato al vettore domanda q. La memoria episodica al passo t è calcolata come:

$$m^t = GRU(c^t, m^{t-1}) \tag{3.13}$$

Il lavoro di Subhbaatar et al. (2015) suggerisce però, che usare pesi diversi per ogni passo della memoria episodica può essere vantaggioso. Quando il modello contiene solo un set di pesi per tutti i passi episodici, viene definito un "modello legato". Seguendo il componente di aggiornamento della memoria di Sukhbaatar et al. (2015) e Peng et al. (2015) gli autori del paper di riferimento hanno sperimentato l'uso di un later ReLU per l'aggiornamento

della memoria, calcolando il nuovo stato della memoria episodica come:

$$m^t = ReLU(W^t[m^{t-1}; c^t; q] + b) \tag{3.14}$$

dove ; è l'operatore di concatenazione, $W^t \in \mathbb{R}^{n_H \times n_H}$, $b \in \mathbb{R}^{n_H}$, e n_H è la dimensione dello stato nascosto. La sconnessione dei pesi e l'uso della funzione ReLU per la formulazione dell'aggiornamento della memoria migliora la precisione di un ulteriore 0.5% come mostrato nel paper di Xiong, Merity e Socher (2016). L'output finale della rete della memoria è passato al modulo di risposta come nel modello di DMN originale.

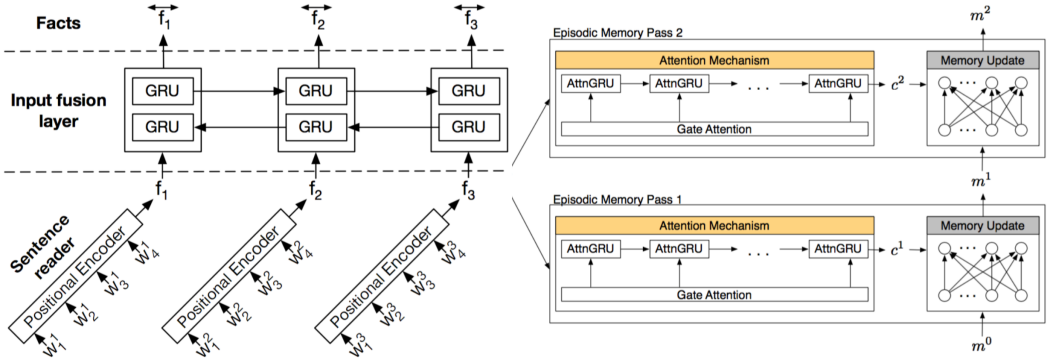


Figura 3.3: Architettura dettagliata del modulo Input ed Episode di una DMN Plus

Capitolo 4

Natural Language Processing

L'obiettivo di questo capitolo è di fornire un'introduzione al NLP per contestualizzare il tema della Sentiment Analysis e di trattarne gli approcci di maggior successo in ambito Machine Learning. Viene fornita innanzitutto una definizione e breve storia del NLP per poi focalizzarsi sui modelli di riferimento tra cui il Position Encoding e GloVe[19] che hanno fornito i risultati migliori in letteratura e verranno utilizzati negli esperimenti oggetto di questo lavoro.

4.1 Definizione

Il nome Natural Language Processing si riferisce al trattamento informatico (computer processing) del linguaggio naturale, per qualsiasi scopo, indipendente dal livello di approfondimento dell'analisi. Per linguaggio naturale si intende la lingua che usiamo nella vita di tutti i giorni, come l'Inglese, il Russo, il Giapponese, il Cinese, ed è sinonimo di linguaggio umano, principalmente per poterlo distinguere dal linguaggio formale, incluso il linguaggio dei computer. Così com'è, il linguaggio naturale è la forma di comunicazione umana più naturale e più comune, e non solo nella sua versione parlata, anche quella scritta sta crescendo esponenzialmente negli ultimi anni, da quando il mobile internet è in fermento con i nuovi social media. Rispetto al linguaggio formale, il linguaggio naturale è molto più complesso, contiene spesso sottointesi e ambiguità, il che lo rende molto difficile da elaborare.
La Natural Language Processing (d'ora in poi NLP) è un campo che quindi coinvolge, tra le altre, l'Intelligenza Artificiale, l'informatica e la linguistica.
Con NLP non solo si intende l'abilità di capire il linguaggio, ma anche di saperlo elaborare, e quindi produrlo.
La NLP include i seguenti campi:

- Speech Recognition: la conversione del parlato in testo;
- Sentiment Analysis: l'analisi del significato dato dal relatore alla frase;

- Syntatic Analysis: lo studio della struttura della frase;
- Pragmatics: quanto il significato della frase dipende da come essa viene utilizzata quotidianamente.

L'uomo tradizionalmente "parla" ai computer tramite un linguaggio ben definito e strutturato, il quale è caratterizzato da codici ai quali corrisponde un chiaro e preciso comando. Nell'atto di programmare è l'umano che adatta il suo linguaggio a quello naturale delle macchine.
Nella NLP ciò che si cerca di fare è insegnare alle macchine ad adattarsi al nostro lingauggio naturale.
La comunicazione che avviene tra macchina e umano è quindi completamente diversa da quella che avviene tra persone.
Pur ipotizzando di poter dotare un computer di un immenso dizionario e di tutte le regole grammaticali, l'elaborazione del linguaggio non è affatto semplice. Le parole non solo devono essere corrette dal punto di vista grammaticale ma devono anche essere associate al loro vero significato, tenendo in considerazione anche il contesto in cui la frase si trova.
I programmi di NLP si basano sul Machine Learning e permettono di segmentare frasi o solo parti di esse, di analizzare attentamente un testo, di identificare il significato di espressioni particolari, distinguere tra espressioni diverse tra loro, tradurre da una lingua ad un'altra e viceversa e molto altro ancora.

4.2 NLP e ML

Un termine che viene spesso usato in parallelo a NLP è Machine Learning (ML, Apprendimento Automatico). Strettamente parlando, machine learning e NLP sono concetti di livello completamente diverso, il primo si riferisce a un tipo di approccio, mentre il secondo rappresenta un'area tematica.
Nonostante questo, a causa della natura di "panacea" del machine learning, unita al fatto che il ML ha dominato la scena del NLP (specialmente nel mondo accademico) degli ultimi anni, a volte viene dimenticata o semplicemente ignorata l'esistenza degli altri approcci di NLP, ossia le regole linguistiche.
In realtà, naturalmente, il machine learning va ben oltre l'ambito del NLP. Gli algoritmi di apprendimento automatico usati per diversi casi di elaborazione del linguaggio possono essere ugualmente usati per risolvere altri problemi di Artificial Intelligence (AI, Intelligenza Artificiale), come ad esempio l'analisi del mercato azionario, le previsioni meteo, il rilevamento di frodi delle carte di credito, la visione artificiale, la classificazione della sequenza del DNA, e addirittura la diagnosi medica.

4.3 Breve storia

La storia della NLP inizia negli anni 40' subito dopo la guerra. A partire dagli anni 50' fino a metà anni 60' gli investimenti in traduzione automatica, ma più in generale in IA, furono molto elevati.

Nel 54', dopo anni di ricerca, si era in grado di tradurre oltre 60 frasi dal russo all'inglese (il bisogno in questo caso derivava anche da un contesto sociopolitico e militare di quegli anni).

Agli inizi dell'era dei computer i progressi erano certamente più lenti di quelli che seguirono nei decenni successivi. Un notevole stop provenne dall'istituzione dell'ALPAC (comitato sull'elaborazione automatica del linguaggio) che bocciò la traduzione automatica nel 66', tagliando i fondi ad essa destinati.

A partire dagli anni 90', con la diffusione di Internet tra il grande pubblico, nacque nuovamente un forte bisogno della NLP in seguito all'estendersi del fenomeno di internet e dell'internazionalizzazione. In particolare si sentiva l'esigenza di estrazione e catalogazione delle informazioni prodotte dalla rete al fine di renderle diffusamente disponibili.

Ad oggi un vasto numero di programmi NLP viene stato sviluppato sia da centri di ricerca universitari che da imprese private. Programmi in grado di gestire con sempre più accuratezza conversazioni sui più svariati argomenti.

4.4 Word embedding

Word embedding è il nome collettivo di un insieme di tecniche di modellazione linguistica e di feature learning in ambito NLP, dove le parole o frasi sono mappate dal vocabolario a vettori di numeri reali.
Concettualmente, questo meccanismo proietta un vettore da uno spazio N-dimensionale, con N numero di parole considerate, ad uno spazio M dimensionale, tipicamente con $M \ll N$.
Tra i metodi per generare questo mapping troviamo le reti neurali, la riduzione della dimensionalità basata su matrice di co-occorenze, modelli probabilistici, e rappresentazioni esplicite in base al contesto nel quale le parole appaiono.
Gli embedding di parola e di frase, quando usati come rappresentazione dell'input, hanno mostrato di poter migliorare le performance nei compiti di NLP come l'analisi sintattica e la sentiment analysis.

4.5 Modelli per il ML

4.5.1 One-hot encoding

La codifica One-hot trasforma le caratteristiche categoriche in un formato che funziona bene con gli algoritmi di classificazione e di regressione: per questo tale rappresentazione ben si presta alla maggior parte degli algoritmi di machine learning (altri invece come Random Forest [7], gestiscono in maniera automatica tale rappresentazione per cui non risulta utile la One-hot encoding).
Questo tipo di codifica utilizza una matrice di M vettori colonna $1 \times N$ per rappresentare la presenza di ogni parola di un vocabolario in un testo in input.
Viene detta codifica one-hot poichè il vettore colonna rappresentante la parola viene costruito con N elementi, ognuno relativo ad certo testo e soltanto le celle nelle righe corrispondenti al testo in cui appare la parola assumono valore 1, mentre gli altri sono azzerati.
Il processo di one-hot encoding, è implementato nelle più moderne librerie di Machine Learning.

4.5.2 Modello N-gram

Per n-gram si intende una sottosequenza di n elementi di una data sequenza. A seconda dell'applicazione gli elementi possono essere fonemi, sillabe, lettere, parole o coppie di parole. Gli n-grams appartengono tipicamente ad un contesto come un corpus di testo o da un discorso. Un n-gramma di lunghezza 1 è detto "unigram", uno di lunghezza 2 è detto "bigram", di lunghezza 3 "trigram" e dalla quarta lunghezza in poi, vengono chiamati "n-gram".
Un modello n-gram rappresenta sequenze, in particolare di linguaggi naturali, usando le proprietà statistiche degli n-gram.
Questa idea si può ricondurre ad un esperimento di Claude Shannon nella teoria dell'informazione. Shannon si fece la domanda: data una sequenza di lettere, qual è la probabilità di ogni lettera di essere la prossima? Più formalmente, un modello n-gram predice x_i basandosi su $x_{i-(n-1)}, ..., x_{n-1}$ dove n è la lunghezza delle sequenze passate.
In termini probabilistici, si indica come $P(x_{i-(n-1)}, ..., x_{n-1})$.
Quando usato per la modellazione del linguaggio, si assume che gli eventi siano indipendenti, in tal modo ogni parola dipende solo dalle ultime $n - 1$ parole.
Questa assunzione è importante perchè semplifica enormemente il problema della stima del modello linguistico dai dati. In un semplice modello di linguaggio n-gram, la probabilità di una parola, condizionata da un certo numero di parole precedenti (una parola in un modello bigram, due parole in un modello trigram, ecc.) può essere descritta come una distribuzione categorica (spesso chiamata "distribuzione multinomiale").

I modelli n-gram sono ampiamente utilizzati in probabilità, teoria della comunicazione, NLP e biologia computazionale.

4.5.3 Bag of words

Il modello bag-of-words è una rappresentazione del testo, usata nel Natural Language Processing ed Information Retrieval. In questo modello, un testo è rappresentato come un multiset (chiamato "bag" ovvero borsa in inglese) composto dalle sue parole, trascurando la grammatica e l'ordine delle parole ma mantenendo la loro molteplicità.
La rappresentazione bag-of-words ha diverse applicazioni di successo, ad esempio filtraggio tramite e-mail, ed è stata usata anche in computer vision.
Nella pratica, è utilizzato comunemente nella classificazione di documenti dove l'occorrenza di ogni parola è una caratteristica utile all'addestramento del classificatore.
Dopo aver trasformato il testo in una "bag-of-words", è possibile calcolarne diverse misure caratterizzanti di cui la più comune è per l'appunto il numero di volte che il termine appare nel testo.
Concettualmente, possiamo considerare il modello bag-of-words come un caso particolare del modello n-gram, dove $n = 1$. Tuttavia, le frequenze dei termini non sono necessariamente la migliore rappresentazione del testo. Le parole comuni come "il", "a", "a" sono quasi sempre i termini con la massima frequenza nel testo. Quindi, avere un conteggio elevato non significa necessariamente che la parola corrispondente sia più importante. Per affrontare questo problema, uno dei modi più diffusi per "normalizzare" le frequenze di termine è quello di pesare un termine in base alla frequenza inversa nei documenti. Inoltre, per lo scopo specifico della classificazione, sono state sviluppate alternative supervisionate che tengano conto della classe dell'etichetta di un documento.
Per alcuni problemi, viene utilizzata una ponderazione binaria (presenza/assenza o 1/0) al posto delle frequenze.

4.5.4 Word2vec

Word2vec è un insieme di modelli usati per produrre word embeddings.
Questi modelli sono costituiti da reti neurali poco profonde di due livelli che sono addestrate per ricostruire i contesti linguistici delle parole.
Word2vec riceve come input un grande corpo di testo e produce uno spazio vettoriale, tipicamente di diverse centinaia di dimensioni, nel quale ogni parola è associata ad un corrispondente vettore nello spazio.
I vettori delle parole sono posizionati nello spazio vettoriale in modo che le parole che condividono il contesto nel corpo del testo, siano vicine l'una dall'altra nello spazio. Word2vec

è stato creato da un team di ricercatori guidati da Tomas Mikolov nella famosa azienda Google[17]. L'algoritmo è stato successivamente analizzato e spiegato da altri ricercatori. Word2vec può utilizzare alternativamente due modelli architetturali per produrre una rappresentazione distruibuita delle parole: continuous bag-of-words (CBOW) o continuous skip-gram.
Nell'architettura continous bag-of-words, il modello predice la parola corrente da una finestra di parole di contesto circostanti. L'ordine delle parole di contesto non influisce sulla previsione (assunzione del modello bag-of-words).
Nell'architettura continuous skip-gram, il modello usa la parola corrente per predire la finestra di parole di contesto circostanti.
L'architettura skip-gram attribuisce maggiore importanza alle parole di contesto vicine rispetto a quelle più distanti. Secondo le note degli autori, il modello CBOW è computazionalmente più veloce dello skip-gram ma quest'ultimo performa meglio con parole poco frequenti.

4.5.5 GloVe

GloVe (Jeffrey Pennington, Richard Socher & Christopher D. Manning, 2014)[19] è un algoritmo di apprendimento non supervisionato per ottenere rappresentazione vettoriali di parole. L'addestramento è eseguito su statistiche aggregate globali di co-occorenza delle parole estratte da un corpo di testo. Le risultanti rappresentazioni mostrano sottostrutture lineari interessanti nello spazio vettoriale delle parole.

La distanza euclidea (o similarità coseno) tra due vettori di parole fornisce un metodo efficace per misurare la similarità linguistica o semantica delle parole corrispondenti.

A volte, le parole più vicine secondo questa metrica si rivelano parole ricercate ma pertinenti che si trovano al di fuori di un vocabolario umano medio.

Le metriche di somiglianza utilizzate per le valutazioni delle parole più simili producono un singolo scalare che quantifica la loro correlazione. Questa semplicità può essere problematica poiché due parole quasi sempre presentano relazioni più intricate di quelle che possono essere catturate da un singolo numero. Ad esempio, l'uomo può essere considerato simile alla donna in quanto entrambe le parole descrivono gli esseri umani; D'altra parte, le due parole sono spesso considerate come opposti, poiché evidenziano un asse primario lungo il quale gli esseri umani si differenziano l'uno dall'altro.

Per catturare in modo quantitativo la sfumatura necessaria per distinguere l'uomo dalla donna, è necessario che un modello associasse più di un singolo numero alla coppia di parole. Un candidato naturale e semplice per un insieme ingrandito di numeri discriminanti è la differenza vettoriale tra i due vettori di parole. GloVe è progettato affinché tali differenze di

vettori catturino quanto più possibile il significato specificato dalla giustapposizione di due parole.

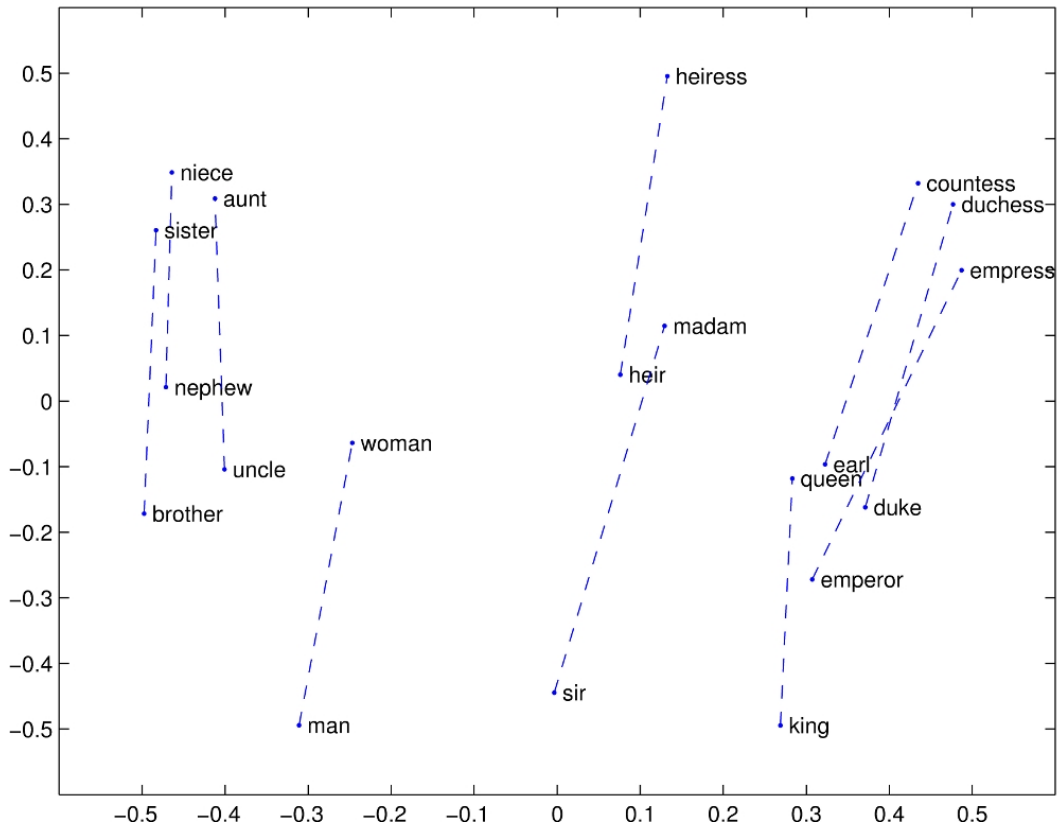


Figura 4.1: Rappresentazione del testo GloVe

Il concetto sottostante che distingue l'uomo dalla donna, i.e. il sesso, può essere specificato equivalentemente da varie coppie di parole, come il re e la regina o fratello e sorella. Per affermare matematicamente questa osservazione, possiamo aspettarci che le differenze di vettore uomo - donna, regina - regina e fratellaster - siano tutte ugualmente uguali. Questa proprietà e altri modelli interessanti possono essere osservati nella Figura 4.1.
Il modello GloVe viene addestrato sulle voci non nulle di una matrice di co-occorrenza globale parola-parola, che elabora quanto spesso le parole coincidono l'una con l'altra in un determinato corpus di testo. La popolazione di questa matrice richiede un singolo passaggio attraverso l'intero corpus per raccogliere le statistiche. Per i corpora di grandi dimensioni, questo passaggio può essere computazionatamente costoso, ma è un costo da affrontare in anticipo una volta sola. Le successive iterazioni di training sono molto più veloci perché il numero di voci non nulle della matrice è in genere molto più piccolo del numero totale di parole nel corpus.

4.5.6 Position Encoding

La Position Encoding, proposta nel noto paper End-To-End Memory Networks (Sukhbaatar, Szlam, Weston, Fergus, 2015), codifica la posizione delle parole all'interno della frase. Il vettore in memoria può essere rappresentato come: $m_i = \sum_j l_j \cdot Ax_{ij}$, dove $\cdot$ è la moltiplicazione elemento per elemento. l_j è un vettore colonna avente la struttura $l_{kj} = (1 - j/J) - (k/d)(1 - 2j/J)$ (assumendo una indicizzazione a base 1), con J che rappresenta il numero di parole nella frase, A la matrice di embedding, x_{ij} la matrice di input e d la dimensione dell'embedding. La rappresentazione della frase, definita in questo modo, comporta che m_i sia influenzato dall'ordine delle parole. La stessa rappresentazione è usata per le domande, gli input e output della memoria. Nel lavoro precedentemente citato viene mostrato come questo tipo di codifica migliori le prestazioni rispetto alla codifica Bag-Of-Words e sarà utilizzata nel progetto in quanto caratteristica delle reti DMN+.

Capitolo 5

La libreria TensorFlow

In questo capitolo viene definita la libreria Tensorflow[30] utilizzata nel progetto descritto nei capitoli successivi. Si riportano in particolare i principi di base utili per iniziare ad utilizzare la libreria e per analizzare l'implementazione del progetto sviluppato in questo lavoro.

5.1 Definizione

TensorFlow è una libreria software open source per vari compiti di Machine Learning e sviluppata da Google per soddisfare le loro esigenze di sistemi in grado di costruire e addestrare reti neurali per rilevare e decifrare modelli e correlazioni, analogamente all'apprendimento e al ragionamento degli esseri umani. Attualmente viene utilizzato sia per la ricerca sia per la produzione nei prodotti Google, sostituendo spesso il ruolo del suo predecessore closed-source, DistBelief. TensorFlow è stato originariamente sviluppato dal team Google Brain per uso interno di Google prima di essere rilasciato sotto la licenza open source di Apache 2.0 il 9 novembre 2015.

5.2 Breve storia

5.2.1 DistBelief

A partire dal 2011, Google Brain ha sviluppato DistBelief come un sistema di Machine Learning proprietario basato su reti neurali tipiche del Deep Learning. Il suo uso è cresciuto rapidamente nelle aziende Alphabet sia in ambito di ricerca sia in applicazioni commerciali. Google ha assegnato più computer scientists, tra cui Jeff Dean, per semplificare e rifattorizzare la base del codice di DistBelief in una libreria di applicazioni più veloce e più robusta, diventata TensorFlow. Nel 2009, il team, guidato da Geoffrey Hinton, aveva implementato in modo generalizzato la backpropagation e altri miglioramenti che consentivano la gene-

razione di reti neurali con un'accuratezza sostanzialmente più elevata, ad esempio con una riduzione del 25% di errori nel riconoscimento vocale.

5.2.2 Tensorflow

TensorFlow è il sistema di apprendimento automatico di seconda generazione di Google Brain, pubblicato come software open source il 9 novembre 2015. La versione 1.0.0 è stata rilasciata l'11 febbraio 2017. Mentre l'implementazione di riferimento viene eseguita su singoli dispositivi, TensorFlow può eseguire più CPU e GPU (con estensioni opzionali CUDA per il calcolo generale delle unità di elaborazione grafica). TensorFlow è disponibile nelle piattaforme di calcolo di 64-bit Linux, macOS, Windows e mobile, tra cui Android e iOS. I calcoli di TensorFlow sono espressi come grafi di stato. Il nome TensorFlow deriva dalle operazioni che tali reti neurali eseguono su matrici di dati multidimensionali. Questi array multidimensionali sono definiti "tensori". Nel giugno 2016, Jeff Dean di Google ha dichiarato che 1.500 repository su GitHub hanno citato TensorFlow, di cui solo 5 erano da Google.

5.2.3 Tensor processing unit (TPU)

Nel maggio 2016 Google ha annunciato la sua unità di elaborazione basata sul tensore (TPU), un ASIC personalizzato costruito appositamente per l'apprendimento macchina e su misura per TensorFlow. Il TPU è un acceleratore programmabile AI progettato per fornire un elevato throughput di aritmetica a bassa precisione (ad esempio, a 8 bit) e orientato verso l'utilizzo o l'esecuzione di modelli piuttosto che l'addestramento. Google ha annunciato di aver eseguito TPU all'interno dei propri data center per più di un anno e li ha notato che forniscono prestazioni più ottimizzate per watt nei compiti di Machine Learning. Nel maggio 2017 Google ha annunciato la seconda generazione della TPU, nonchè la disponibilità di TPU in Google Compute Engine. Le TPU di seconda generazione offrono fino a 180 teraflops di prestazione e, se organizzati in cluster di 64 TPU, possono fornire fino a 11,5 petaflops.

5.2.4 Tensorflow Lite

Nel Maggio 2017 Google ha annunciato uno stack software specifico per lo sviluppo Android, chiamato TensorFlow Lite, supportato dalla versione Android O.

5.3 Requisiti

Per approcciarsi all'utilizzo di TensorFlow, occorre conoscere:

- la programmazione in Python;
- le strutture dati array;
- i principi base del Machine Learning. In ogni caso, anche se si conosce poco o niente del Machine Learning, i principi di TensorFlow sono utili per l'apprendimento iniziale di questa materia.

TensorFlow fornisce diverse APIs. Le API di più basso livello, TensorFlow Core, forniscono un controllo sulla programmazione completo. TensorFlow Core è consigliato ai ricercatori di Machine Learning e agli sviluppatori che cercano un controllo fine sul proprio modello. Le APIs di più alto livello sono costruite sulla base di TensorFlow Core. Quest'ultime sono tipicamente più facili da imparare e da usare rispetto a TensorFlow Core. Inoltre, le APIs di alto livello facilitano i compiti più ripetitivi e frequenti tra gli utenti. Una API di alto livello come tf.contrib.learn facilita la gestione di dataset, stime, addestramento ed inferenza.

5.4 Importare la libreria

L'istruzione canonica di import per i programmi TensorFlow è la seguente:

```
import tensorflow as tf
```

Questa importazione fornisce a Python l'accesso a tutte le classi, metodi e simboli di TensorFlow. La maggior parte della documentazione ufficiale assume che questa istruzione sia stata eseguita.

5.5 Principi di base

5.5.1 I Tensori

L'unità centrale dei dati in TensorFlow è il tensore. Un tensore consiste in un insieme di valori primitivi dalla forma di array di un numero qualsiasi di dimensioni. Il rango di un tensore è il numero delle sue dimensioni. Di seguito si riportano alcuni esempi di tensori:

```
# a rank 0 tensor; this is a scalar with shape []
3
```

```
# a rank 1 tensor; this is a vector with shape [3]
[1. ,2., 3.]

# a rank 2 tensor; a matrix with shape [2, 3]
[[1., 2., 3.], [4., 5., 6.]]

# a rank 3 tensor with shape [2, 1, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]]
```

5.5.2 Il grafo computazionale

I programmi TensorFlow Core consistono in due parti:

1. costruire il grafo computazionale
2. eseguire il grafo computazionale

Un grafo computazionale è una serie di operazioni di TensorFlow disposte in un grafo di nodi. Costruiamo ora un semplice grafo computazionale. Ogni nodo prende zero o più tensori in input e produce in output un tensore. Un tipo di nodo è una costante. Tutte le costanti di TensorFlow non ricevono nessun input ed emettono in output un valore che memorizzano internamente. Possiamo creare due tensori a virgola mobile, node1 e node2 nella maniera seguente:

```
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)
```

La chiamata a funzione print finale produce il seguente output:

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
```

Si può notare che stampando a video i nodi, questi non emettono i valori 3.0 e 4.0 come ci si potrebbe aspettare. I rispettivi valori 3.0 e 4.0 saranno prodotti quando i predetti nodi saranno valutati.

5.5.3 Sessione

Per valutare realmente i nodi, dobbiamo eseguire il grafo computazione all'interno di una sessione. Una sessiona incapsula il controllo e stato a runtime di TensorFlow. Il codice seguente crea un oggetto Session e poi invoca il suo metodo "run" sufficiente ad eseguire il grafo computazionale per valutare node1 e node2.

Eseguendo il grafo computazionale in una sessione come in seguito:

```
sess = tf.Session()
print(sess.run([node1, node2]))
```

vedremo a video i valori attesi 3.0 e 4.0

```
[3.0, 4.0]
```

È possibile costruire computazione più complicate combinando i nodi tensori con le operazioni (anch'esse dei nodi). Per esempio, possiamo sommare i nostri due nodi di costanti e produrre un nuovo grafo come di seguito:

```
node3 = tf.add(node1, node2)
print("node3: ", node3)
print("sess.run(node3): ", sess.run(node3))
```

Le ultime due chiamate a print producono:

```
node3:  Tensor("Add:0", shape=(), dtype=float32)
sess.run(node3):  7.0
```

5.5.4 TensorBoard

TensorFlow fornisce uno strumento di utility chiamato TensorBoard che può visualizzare l'immagine di un grafo computazionale. Di seguito si riporta l'immagine del nostro grafo visualizzato da TensorBoard:

Figura 5.1: Operazione di somma in un grafo TensorFlow

5.5.5 Placeholders

Il grafo descritto in precedenza non è particolarmente interessante perchè produce sempre un risultato costante. Un grafo può essere parametrizzato per ricevere input esterni, conosciuti come placeholders. Un placeholder può essere paragonato ad una promessa di fornire un valore successivamente.

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b  # + provides a shortcut for tf.add(a, b)
```

Le precedenti tre linee assomigliano ad una funzione o lambda nel quale definiamo due parametri di input (a e b) e poi un operatore su di essi. Possiamo valutare questo grafo con input multipli usando il parametro feed_dict per specificare i tensori che forniscono i valori concreti a questi placeholders.

```
print(sess.run(adder_node, {a: 3, b:4.5}))
print(sess.run(adder_node, {a: [1,3], b: [2, 4]}))
```

che restituisce il seguente output:

```
7.5
[ 3.  7.]
```

In TensorBoard, il grafo verrà visualizzato come nella figura sottostante:

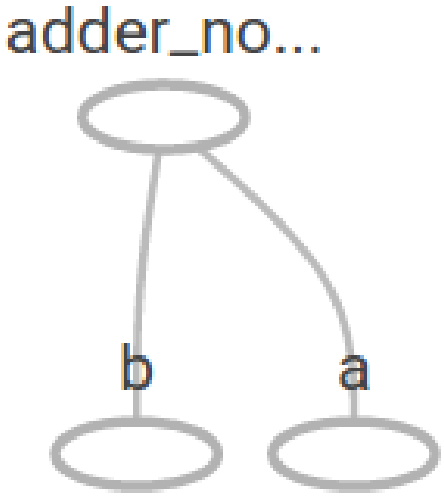


Figura 5.2: Nodo dell'operazione di somma tra due placeholders in un grafo TensorFlow

Possiamo rendere il grafo computazionale più complesso aggiungendo un'altra operazione. Per esempio:

```
add_and_triple = adder_node * 3.
print(sess.run(add_and_triple, {a: 3, b:4.5}))
```

che produce l'output:

```
22.5
```

Il precedente grafo computazionale verrebbe visualizzato in TensorBoard come:

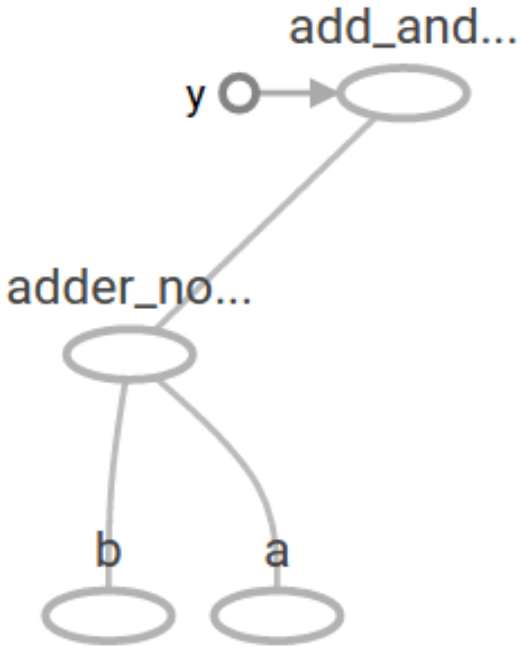


Figura 5.3: Nodo somma tra due placeholders e moltiplicazione del risultato parziale in un grafo TensorFlow

5.5.6 Variabili

Nel Machine Learning vogliamo tipicamente un modello con un numero arbitrario di input, come quello descritto in precedenza. Per rendere il modello addestrabile, abbiamo bisogno di poter modificare il grafo per ottenere nuovi output con gli stessi input. Le variabili ci permettono di aggiungere parametri addestrabili al grafo. Sono inizializzati specificando un tipo ed un valore iniziale:

```
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
x = tf.placeholder(tf.float32)
linear_model = W * x + b
```

Le costanti sono inizializzate quando chiami tf.costant, e il loro valore non può cambiare. Al contrario, le variabili non sono inizializzate quando si chiama tf.Variable. Per inizializzare tutte le variabili in un programma TensorFlow, bisogna chiamare esplicitamente un'operazione speciale come descritto di seguito:

```
init = tf.global_variables_initializer()
```

```
sess.run(init)
```

È importante considerare che init è un controllo del sottografo TensorFlow che inizializza tutte le variabili globali. Finchè non chiamiamo sess.run, le variabili non saranno inizializzate.
Dal momento che x è un placeholder, possiamo valutare linear_model per diversi valori di x simultaneamente nel seguente modo:

```
print(sess.run(linear_model,{x:[1,2,3,4]}))
```

che produce il seguente output:

```
[ 0.          0.30000001  0.60000002  0.90000004]
```

5.5.7 Training

Abbiamo creato un modello, ma non sappiamo quanto preciso sia. Per valutare il modello sui dati di training, abbiamo bisogno di un placeholder y per fornire i valori desiderati, e poi di definire un funzione di perdita.
Una funzione di perdita misura quanto il modello di scosta dai dati forniti desiderati. Useremo un modello di perdita standard per la regressione lineare, il quale somma i quadrati dei delta tra il modello corrente e i dati forniti. L'istruzione "linear_model - y" crea un vettore dove ogni elemento è il delta dell'errore degli esempio corrispondente. Chiamiamo tf.square per elevare al quadrato l'errore. Successivamente, sommiamo tutti gli errori quadratici per creare un singolo scalare che astrae l'errore di tutti gli esempi usando tf.reduce_sum:

```
y = tf.placeholder(tf.float32)
squared_deltas = tf.square(linear_model - y)
loss = tf.reduce_sum(squared_deltas)
print(sess.run(loss, {x:[1,2,3,4], y:[0,-1,-2,-3]}))
```

che produce il valore di perdita:

```
23.66
```

Possiamo migliorare manualmente il modello riassegnando i valori di W e b ai valori ottimali di -1 e 1. Una variabile è inizializzata al valore fornito alla funzione tf.Variable ma può essere cambiata usando operazioni come tf.assing. Per esempio, W=-1 e b=1 sono i parametri ottimali per il nostro modello. Possiamo cambiare W e b di conseguenza:

```
fixW = tf.assign(W, [-1.])
fixb = tf.assign(b, [1.])
sess.run([fixW, fixb])
print(sess.run(loss, {x:[1,2,3,4], y:[0,-1,-2,-3]}))
```

La chiamata finale a print mostra che la perdita ora è azzerata:

```
0.0
```

In questo caso abbiamo impostato i valori ottimali di W e b conoscendo la funzione di regressione lineare, ma l'intero punto chiave del Machine Learning è quello di trovare i parametri del modello automaticamente.
TensorFlow fornisce degli ottimizzatori che lentamente cambiano ogni variabile al fine di minimizzare la funzione di perdita. L'ottimizzatore più semplice è la discesa del gradiente. Quest'ultimo modifica ogni variabile in base alla derivata della perdita rispetto alla relativa variabile. In generale, calcolare le derivate manualmente è tedioso e soggetto ad errori. Per questo, lasciamo che TensorFlow calcoli automaticamente le derivate fornendogli solamente una descrizione del modello usando la funzione tf.gradients. Per semplicità, gli ottimizzatori tipicamente eseguono questo lavoro per noi. Per esempio:

```
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

sess.run(init) # reset values to incorrect defaults.
for i in range(1000):
sess.run(train, {x:[1,2,3,4], y:[0,-1,-2,-3]})


print(sess.run([W, b]))
```

che porta ai seguenti parametri finali del modello:

```
[array([-0.9999969], dtype=float32), array([ 0.99999082],
 dtype=float32)]
```

Quest'ultimo rappresenta veramente un esempio di Machine Learning. Sebbene costruire questa semplice regressione lineare non richieda molto codice TensorFlow Core, modelli e metodi più complicati richiedono più codice. Per questo TensorFlow fornisce astrazioni di più alto livello per pattern, strutture e funzionalità comuni.

Programma completo Il modello completo ed addestrabile della regressione lineare è mostrato di seguito:

```
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
```

```
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)
# loss = sum of the squares
loss = tf.reduce_sum(tf.square(linear_model - y))
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
# training data
x_train = [1,2,3,4]
y_train = [0,-1,-2,-3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
sess.run(train, {x:x_train, y:y_train})


# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss],
 {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

Quando viene eseguito produce:

```
W: [-0.9999969] b: [ 0.99999082] loss: 5.69997e-11
```

La perdita è un numero molto vicino allo zero. Se si esegue questo programma la perdita non sarà esattamente la stessa, poichè il modello è inizializzato con valori casuali.

Questo programma più complesso può essere visualizzato in TensorBoard come segue:

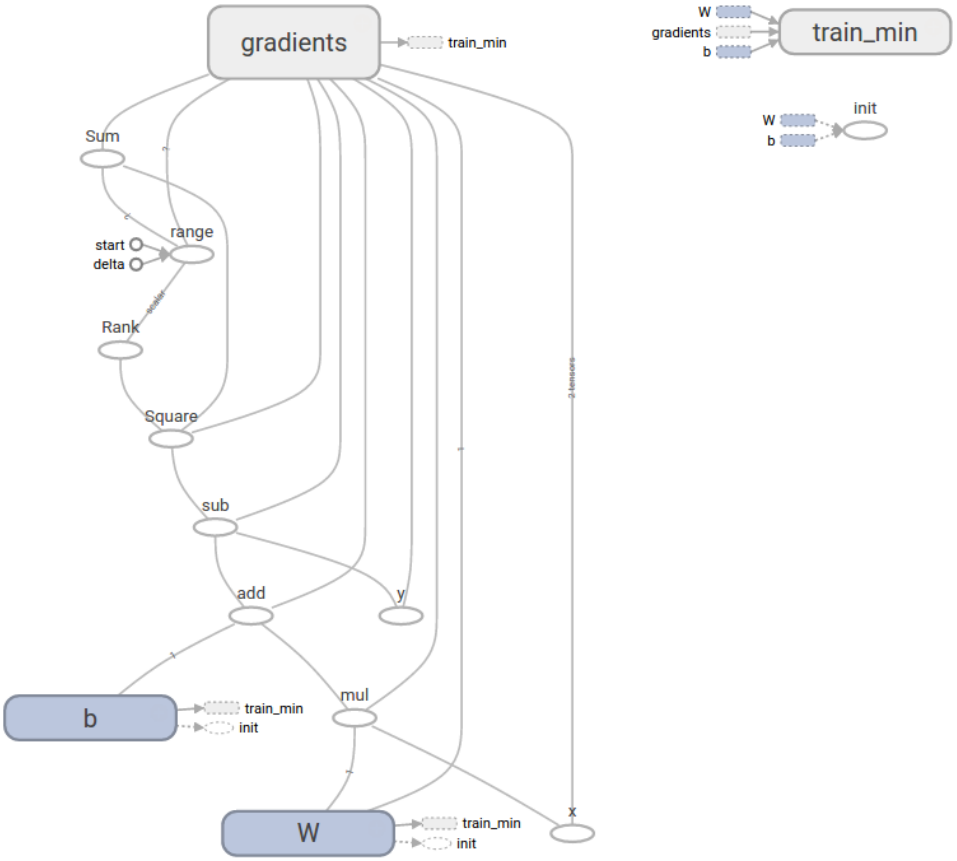


Figura 5.4: Grafo TensorFlow della regressione lineare

5.5.8 tf.contrib.learn

tf.contrib.learn una libreria TensorFlow di alto livello che semplifica i meccanismi del Machine Learning come:

- esecuzione delle iterazioni di training
- esecuzione delle iterazioni di valutazione
- gestione dei datasets
- gestione della fornitura dei dati

Questa libreria mette a disposizione molti modelli comuni.

Utilizzo di base Di seguito si riporta il programma TensorFlow della regressione lineare, codificato con tf.contrib.learn. Si noti quanto il codice diventi più semplice.

```
import tensorflow as tf
# NumPy viene spesso usato per caricare, manipolare e
# preprocessare i dati.
import numpy as np

# Si dichiara una lista di features. Abbiamo feature
# valorizzate solamente con numeri reali. Ci sono molti
# altri tipi di colonne che sono pi\‘u complicate e utili.
features = [tf.contrib.layers
        .real_valued_column("x", dimension=1)]

# Un estimator rappresenta l'interfaccia per invocare
# l'addestramento (fitting) e la valutazione (inference).
# Ci sono molti tipi predefiniti come la regressione lineare,
# la regressione logistica, la classificazione lineare,
# la classificazione logistica e molti classificatori composti
# da reti neurali e regressori.
# Il codice seguente fornisce un estimator che implementa la
# regressione lineare
estimator = tf.contrib.learn
        .LinearRegressor(feature_columns=features)

# TensorFlow fornisce molti metodi di aiuto per leggere
# ed impostare datasets.
# Qui usiamo due datasets: uno per il traning e uno per la
# valutazione. Dobbiamo specificare alla funzione quanti batch
# di dati (num_epochs) vogliamo e quanto deve essere grande
# ogni batch.
x_train = np.array([1., 2., 3., 4.])
y_train = np.array([0., -1., -2., -3.])
x_eval = np.array([2., 5., 8., 1.])
y_eval = np.array([-1.01, -4.1, -7, 0.])
input_fn = tf.contrib.learn.io.numpy_input_fn({"x":x_train},
        y_train, batch_size=4, num_epochs=1000)

eval_input_fn = tf.contrib.learn.io.numpy_input_fn(
{"x":x_eval}, y_eval, batch_size=4, num_epochs=1000)
```

```
# Lanciamo 1000 step di addestramento invocando il metodo
# fit e passando il traning set.
estimator.fit(input_fn=input_fn, steps=1000)

# Qui valutiamo l'efficacia del nostro modello.
train_loss = estimator.evaluate(input_fn=input_fn)
eval_loss = estimator.evaluate(input_fn=eval_input_fn)
print("train loss: %r"% train_loss)
print("evaluation loss: %r"% eval_loss)
```

In esecuzione produce l'output:

```
train loss: {'global_step': 1000, 'loss': 4.3049088e-08}
evaluation loss: {'global_step': 1000, 'loss': 0.0025487561}
```

Da notare come i dati di validazione abbiano una perdita più alta, ma ancora vicina allo zero. Questo ci conferma che l'addestramento del modello è corretto.

Capitolo 6

Specifiche del progetto

Lo scopo di questo progetto risulta essere l'applicazione di tecniche di Transfer Learning su scenari di adattamento del dominio in compiti di sentiment classification cross-domain. Vogliamo valutare come e con quale entità l'utilizzo di algoritmi di Deep Learning quali reti neurali di tipo Dynamic Memory Networks impatti nella risoluzione del compito precedentemente delineato, utilizzando in particolare le codifiche per la rappresentazione del testo GloVe e Position Enconding che si sono dimostrate più efficaci ed efficienti nell'addestramento.
Verranno inoltre effettuati alcuni esperimenti analoghi utilizzando le Convolutional Neural Networks in quanto in letteratura[12][15][16] si sono dimostrate efficaci anche in compiti di Sentence Classification.
Negli esperimenti sul dataset Stanford Sentiment Treebank verranno inizializzati i word embeddings caricando i vettori addestrati GloVe forniti dalla stessa università di Stanford ed, in seconda battuta, i vettori Google News addestrati secondo il modello Word2Vec.
In questo capitolo vengono presentate analisi e progettazione del sistema che si è realizzato, nel capitolo successivo invece vengono analizzati i risultati ottenuti e confrontati con alcuni risultati derivati dall'utilizzo di reti LSTM utilizzate in compiti di classificazione analoghi.

6.1 Introduzione

Nella realtà dell'ultimo decennio, con l'espansione di internet e dei social networks, quali Facebook e Twitter, si è creata sovrabbondanza di dati non strutturati ed etichettati il cui valore potenziale 'e realmente alto. Riuscire a derivare l'opinione delle persone relative a prodotti, servizi, brands ecc..., è un processo di sentiment classification di fondamentale importanza nella pianificazione di strategie operative e di business. Una soluzione per l'utilizzo di queste potenziali informazioni si basa sul lavoro di persone nella classificazione e elaborazione dei dati non etichettati. Processo che, oltre che costoso, si rivela impraticabile per grandi volumi di dati.

A tal proposito la cross-domain sentiment classification è stata introdotta per risolvere queste limitazioni. Ad esempio, volendo estrapolare l'opinione delle persone su determinati prodotti, quali telefonia ed elettronica, è possibile utilizzare la conoscenza estrapolata da un modello addestrato a riconoscere la polarità di prodotti differenti quali ad esempio libri.
Lo scopo della classificazione cross-domain 'e quello di utilizzare la conoscenza appresa nel contesto del primo dominio ed applicarla alla comprensione di esempi appartenenti al secondo dominio.
La classificazione cross-domain, a differenza della classificazione in-domain, richiede generalmente una fase intermedia detta di Transfer Learning, così che la conoscenza costruita sul modello sorgente possa effettivamente essere adattata al dominio target.
Questa fase è fondamentale anche nel caso in cui si cambi solamente la categoria dei prodotti, poichè il modello apprende dei concetti tipici e appartenenti al dominio di addestramento che non riguardano il dominio target.

6.2 Prerequisiti

Per una piena comprensione della realizzazione del progetto è necessario conoscere i seguenti argomenti:

- Tecniche di Transfer Learning su scenari di adattamento del dominio (Cap. 1);
- Conoscenza dell'architettura generale e del funzionamento delle reti neurali (Cap. 2);
- Funzionamento generale e struttura delle reti DMN+ e CNN (Cap. 3);
- Rappresentazione del testo Bag-Of-Words, Word2Vec, GloVe e Position Encoding (Cap. 4);
- Buona conoscenza del linguaggio Python;
- Conoscenza della libreria TensorFlow (Cap. 5).

6.3 Analisi del problema

Nel caso di studio analizzato in questo progetto, abbiamo a disposizione due importanti datasets con recensioni di prodotti commerciali. Più precisamente, il primo di questi consiste in una collezione di recensioni di Amazon[33], messi a disposizione dall'università di Stanford, su cui sono stati eseguiti i test su 3 domini principali, di cui in seguito indicheremo le abbreviazioni:

- Film e Serie TV (M)

- Elettronica (E)
- Abbigliamento Scarpe e Gioielleria (J)
- Libri (B)

Ogni recensione è composta da numerosi campi che la identificano. Nel progetto ci si riferisce a due campi in particolare:

- reviewText, è il campo contenente la recensione in formato testo. Il contenuto e la dimensione della stessa variano in maniera casuale;
- overall, è il valore da 1 a 5, in termini di punteggio, che l'utente assegna a tale prodotto in seguito all'acquisto.

Associamo alle recensioni con overall 1 e 2 un'etichetta negativa, analogamente 4 e 5 avranno etichetta positiva, mentre ignoriamo nella classificazione binaria, le recensioni con overall 3 poichè ritenute neutre e più difficili da classificare. Il secondo dataset utilizzato per la sentiment classification è il noto Stanford Sentiment Treebank fornito dalla predetta università. In questo dataset vengono fornite frasi etichettate da valori compresi da 0 a 1 ed una divisione dei dati di train/validation/test predefinita.

6.4 Design degli esperimenti

Lo scopo sarà quello di massimizzare l'accuratezza della funzione di predizione relativa allo specifico dominio target utilizzando esempi provenienti da domini sorgente di inerenti dallo stesso.
Verranno presentati risultati inerenti sia alla classificazione binaria (positivo/negativo) sia alla classificazione a 5 classi (fine-grained) entrambe eseguite come problema di QA per mostrare se questa tecnica abbia un impatto positivo sulla risoluzione del problema.
Essendo dati provenienti dal contesto reale, notiamo che vi è uno sbilanciamento relativo alle classi, in particolare le recensioni positive rappresentano una percentuale media dell'80% rispetto al totale.
Per evitare casi in cui i risultati siano influenzati da porzioni di dataset particolarmente uniformi (e quindi più facili da classificare) sono stati eseguiti più run su diversi esempi del dataset per ogni singolo esperimento per poi mediarne le precisioni ottenute. Sono stati lanciati 3 run negli esperimenti da 2000 esempi e 2 in quelli da 20000, mentre con 100000 esempi è stato effettuato un singolo run, in quanto l'uniformità della distribuzione delle classi è attenuata dall'aumento del numero di esempi.
Al fine di dimostrare l'efficienza della tecnica sviluppata su domini di diversa grandezza, la rete è stata addestrata e testata con datasets di tre diverse dimensioni, i.e. 2k-20k-100k,

preservando sempre il rapporto dimensionale training-validation-testing, i.e. 80%-10%-10%.
Il rapporto dimensionale ed il bilanciamento dei dataset non sono stati applicati agli esperimenti sul dataset Stanford Sentiment Treebank in quanto considerati predefiniti. In quest'ultimo tipo di esperimento, sono state utilizzate 125882 istanze in addestramento, 825 in validazione e 1749 nel test.
I parametri di configurazione della rete, sono stati studiati in maniera empirica, iniziando ad utilizzare quelli ritenuti più utili in compiti di sentiment classification dalla relativa letteratura.
Nella prossima sezione presentiamo più tipologie di test differenti al fine di mettere in evidenza le caratteristiche ed i pregi delle tecnologie confrontate. Prima però definiamo il significato dei termini che utilizzeremo in seguito:

- Test in-domain: il dominio sorgente utilizzato per l'addestramento è identico al dominio target utilizzato per validazione e testing;
- Test cross-domain: il dominio sorgente utilizzato per l'addestramento è differente dal dominio target utilizzatto per validazione e testing.

A tal proposito, nel testing cross-domain utilizzeremo un approccio di Transfer Learning per adattare il dominio di testing al dominio sorgente, in maniera tale da migliorare le performance relative all'accuratezza di predizione.
La prima serie di test viene condotta in-domain, ovvero gli algoritmi sono testati su di un set di recensioni provenienti dallo stesso dominio utilizzato per l'addestramento.
Tale test ci permette di analizzare quanto la dimensione del dataset di training influisca sull'accuratezza registrata.
Il secondo tipo di esperimento si basa su dei test cross-domain. In questo scenario l'algoritmo utilizza la conoscenza di generalizzazione che ha appreso imparando sul proprio dominio applicandola ad un dominio differente.
Tale testing mette in risalto l'abilità di generalizzazione dell'algoritmo e la sua stabilità al variare della dimensione del testing set.
L'ultima serie di esperimenti consiste nell'applicazione della tecnica del Transfer Learning, in uno scenario di adattamento del dominio, i.e. la categoria di appartenenza delle recensioni, e nell'analisi dei miglioramenti che tale tecnica riesce a conseguire.
Per eseguire questo test prendiamo un modello già addestrato su un dominio X con l'utilizzo di 2k, 20k e 100k esempi etichettati. Lo sottoponiamo ad un secondo addestramento con un numero limitato di esempi appartenenti ad un secondo dominio Y (nel nostro caso 200 in training e 50 in validation). Al termine dell'addestramento testiamo il modello finale rispettivamente con 200, 2000 e 100000 elementi provenienti dal secondo dominio Y.

Nella scelta della tipologia di rete neurale da utilizzare si è optato per una rete Dynamic Memory Networks (DMN, capitolo 3), modellando il problema di classificazione come uno di Question Answering, per verificare come questo tipo di reti ottimizzato per il QA, riescano ad effettuare predizioni accurate anche in compiti di sentiment classification, confrontando i risultati con i precedenti analoghi esperimenti eseguiti con reti LSTM.

6.4.1 Gestione dei dataset

Per addestrare una rete neurale in processi di sentiment classification è necessario processare tutte le parole nelle recensioni considerate al fine di ottenere una loro rappresentazione in un formato utilizzabile in tali contesti di applicazione. Come spiegato nel capitolo 4 sulla NLP, vi sono numerose tecniche di rappresentazione del testo più o meno efficaci. L'architettura DMN+ utilizza la rappresentazione Position Encoding tipica di questo modello, mentre negli esperimenti con Convolutional Neural Networks viene utilizzata la codifica Bag-of-Words. L'applicativo delle reti DMN+ mette a disposizione anche la possibilità di precaricare i word embeddings GloVe (forniti dall'università di Stanford) o Google News addestrati.
Nel processo di addestramento è opportuno suddividere il dataset in porzioni dedicate ai relativi procedimenti. Suddividiamo dunque il dataset creato utilizzando un rapporto 80%-10%-10%, rispettivamente per addestramento, validazione e testing della rete. I processi di training e validazione si svolgono su un numero massimo di epoche (256 nel nostro caso ma configurabile dagli iperparametri) specificando un criterio di fine anticipata.
Il criterio di fine anticipata dell'addestramento sarà configurabile come iperparametro e rappresenta il numero di epoche nelle quali la validation loss non è diminuita rispetto a quella più bassa ottenuta complessivamente nell'addestramento.
Con epoca si intende l'iterazione del processo di addestramento sullo stesso dataset, talvolta mescolando gli elementi al suo interno al fine di randomizzare la sequenza di input passata alla rete. L'addestramento è il processo tramite il quale la rete aggiorna la sua conformazione al fine di adattare i propri pesi e la propria funzione di predizione dell'output all'output desiderato, relativo agli esempi utilizzati in questo processo.
La validazione è utile al fine di monitorare il processo di addestramento e testarlo su elementi fuori dal dataset di addestramento. Infine con testing si intende validare l'accuratezza di predizione della rete utilizzando un dataset mai sfruttato all'interno del processo di addestramento ed utile alla sua comprensione.
Tale suddivisione è importante poichè in questo modo potremo testare la rete neurale con degli elementi su cui non è mai stata addestrata, simulando quindi una probabile applicazione in un contesto reale.
Dopo aver suddiviso il dataset nelle relative porzioni utilizzate per training-validation-

testing, rispettivamente (80-10-10)%, l'intero corpus in lingua inglese di ogni documento verrà preprocessato e suddiviso in frasi. È questo il momento in cui scegliere il tipo di rappresentazione del testo da applicare: infatti le frasi verranno poi converite in vettori a seconda della tipologia di codifica scelta.

6.4.2 Configurazione iperparametri

I risultati di una rete neurale in compiti di sentiment classification variano notevolmente al variare degli iperparametri che ne regolano il funzionamento e svolgono un ruolo cruciale all'interno dell'addestramento. Di seguito si riportano gli iperparametri ritenuti più importanti ed influenti sul processo di addestramento:

- Numero classi: come prima cosa è opportuno definire il numero di classi su cui si basa la classificazione, nel nostro caso 2 o 5, poichè viene utilizzata per predire la polarità di una recensione;

- Numero esempi: definiamo inoltre il numero di esempi su cui verrà addestrata la rete, nel nostro caso variabile poichè condurremo esperimenti con datasets di dimensioni differenti al fine di mettere in mostra come tale scelta influisca nella capacità di generalizzazione e predizione della rete;

- Numero massimo di epoche: definiamo inoltre il numero massimo di epoche di addestramento, le quali simboleggiano il numero massimo di volte in cui il processo di addestramento verrà iterato al fine di migliorare le qualità della rete. Alla fine di ogni epoca, gli elementi all'interno del dataset verranno mescolati, in maniera tale da sottoporre sequenze sempre differenti, ed evitare che la rete si adegui a strutture non desiderate;

- Dimensione del batch: tale valore influisce molto nei termini relativi all'addestramento: infatti tenendo un valore di batch molto piccolo la rete aggiornerà i propri pesi più di frequente causando una dilatazione dei tempi di addestramento, viceversa tenendo un batch con dimensione elevata potrebbe causare peggioramenti nel processo di predizione a favore di tempi di addestramento molto più corti. Tale trade-off va valutato accuratamente e impostato in maniera empirica a seguito di numerosi test, tenendo conto del limite oltre il quale la rete non migliorerà più la propria accuratezza. Nel nostro caso abbiamo notato come nella configurazione ideale il batch della nostra rete fosse composto da 10 elementi, tale valore ha sicuramente impattato sui tempi di addestramento, i quali hanno registrato in alcuni casi periodi superiori alle 24 ore, a beneficio di una migliore accuratezza come evidenziato dai risultati;

- Fine anticipata: rappresenta il criterio di fine anticipata dell'addestramento. L'addestramento finisce, se non è stato raggiunto il numero massimo di epoche, quando la funzione di perdita riferita al procedimento di validazione non migliora da un certo numero di epoche specificato dal parametro in questione. Per i nostri esperimenti il valore 5 si è rivelato ragionevole poichè un valore molto alto potrebbe far crescere in modo esponenziale i tempi di addestramento;

- Dropout: è un metodo di regolarizzazione della rete neurale che consiste nel impostare a zero le attivazioni di un certo numero di neuroni in modo casuale. Con il Dropout, i pesi dei neuroni impostati attraverso la backpropagation sono meno influenzati dai pesi degli altri nodi ed imparano in modo indipendente dagli altri neuroni. Il valore di questo parametro rappresenta la percentuale dei neuroni i cui pesi non verranno aggiornati (in quanto il gradiente che scorre sarà effettivamente azzerato). Un valore tipico è 0,5 ma nei nostri esperimenti abbiamo utilizzato il valore 0,9 come suggerito nel paper Dynamic Memory Networks for Question Answering (Arushi Raghuvanshi & Patrick Chase, 2016)[21] dal quale sono stati presi i parametri iniziali per i nostri esperimenti. Questo valore ha sicuramente dato buoni risultati, anche se ha allungato i tempi di addestramento.

- Learning Rate: definisce la velocità di apprendimento della rete. Tale valore va scelto con cura e spesso non viene deciso a tavolino ma calcolato empiricamente. Se si utilizzasse un learning rate troppo alto il processo di addestramento non convergerebbe mai poichè la funzione tenderebbe ad oscillare attorno al punto di minimo senza mai avvicinarsi, viceversa scegliendo un valore troppo piccolo potrebbe portare a tempi di convergenza troppo lunghi. Nel nostro caso abbiamo usato un Learning Rate di 0,001.

6.4.3 Creazione della rete

Creiamo il nostro modello, utilizzando le funzioni messe a disposizione dalla libreria TensorFlow. Costruiamo su di esso la struttura della rete aggiungendo layers e attivazioni in sequenza come spiegato nel capitolo 5. In questo momento ci occupiamo inoltre di definire i parametri relativi alle funzioni di attivazione e ai valori di dropout sulle connessioni, al fine di prevenire overfitting sulla rete.

6.4.4 Compilazione del modello

Dopo aver creato il nostro modello dobbiamo preoccuparci di scegliere la funzione di ottimizzazione adatta. Compiliamo dunque il modello, specificandone la metrica, la funzione di loss (che esprime l'errore della nostra predizione) e l'optimizer che si incarica di mini-

mizzare l'errore, addestrando la rete. Si è optato per l'utilizzo di una funzione di loss di tipo sparse-softmax-cross-entropy nel progetto delle DMN+ e softmax-cross-entropy-with-logits nel caso delle reti CNN, le quali rappresentano due casi speciali di cross-entropy che misurano la probabilità di errore in compiti di classificazione discreta nei quali le classi sono mutuamente esclusive (ogni valore rappresenta esattamente una classe). TensorFlow mette a disposizione numerosi optimizer già funzionanti, nel nostro caso è stato scelto Adam (Adaptive Moment estimation)[22], il quale risulta essere computazionalmente efficiente, con poca esigenza di memoria ed adatto per problemi con dataset molto grandi.

6.4.5 Addestramento della rete

Dopo la fase di compilazione segue la fase di addestramento in cui è opportuno impostare i parametri definiti precedentemente. Durante l'addestramento verranno stampati il valore della funzione di loss, gli step totali ed eseguiti, il numero dell'epoca corrente. Alla fine di ogni epoca vengono visualizzati il traning/validation loss e la traning/validation accuracy al fine di monitorare il processo di addestramento.

6.4.6 Salvataggio e Testing della rete

Il modello addestrato verrà salvato alla fine di ogni epoca di addestramento che ha migliorato la validation loss o dopo un numero di step di training eseguiti. Questo ci permetterà di fermare a piacere il processo di training, in base ai valori di perdita stampati, avendo salvato il modello che può essere ricaricato successivamente per il testing o per riprendere l'addestramento dal punto in cui si è interrotto. Il framework TensorFlow permette, infatti, di salvare la configurazione strutturale della rete, oltre ai parametri aggiornati. Allo stesso modo, in fase di testing è possibile recuperare i pesi relativi ad un modello precedentemente salvato e pronto per essere utilizzato direttamente in fase di testing.

6.5 Implementazione

6.5.1 Funzionalità sviluppate

DMN+

L'implementazione del progetto sulle DMN+ è stata realizzata interamente in Python utilizzando principalmente la libreria TensorFlow Core poichè l'architettura DMN+ non è ancora implementata in API di alto livello o framework come Keras[31]. Il lavoro di programmazione si è basato sull'implementazione delle Dynamic Memory Networks Plus di Alex Barron reperibile su GitHub. L'applicazione citata contiene un modulo di input specifico per task eseguiti sul dataset di Facebook bAbI, per questo i primi sviluppi si sono incentrati sulla modifica adattiva per lanciare gli esperimenti sui dataset da noi utilizzati. Nonostante le

numerose modifiche al modulo di input è stata mantenuta la compatibilità con i task bAbI in modo che l'applicazione sia utilizzabile con più tipi di dataset. Successivamente sono stati aggiunte funzionalità ed iperparametri per la gestione dei dataset, come la possibilità di specificare:

- percentuali di divisione dei dati in training/validation/testing;
- numero delle classi;
- numero complessivo di esempi da utilizzare;
- percentuale di recensioni negative/positive da usare in traning;
- offset di sampling del dataset che specifica quale parte di dataset campionare per reperire gli esempi;

Inoltre sono stati risolti dei problemi riguardanti il caricamento dei vettori GloVe ed una volta verificato il funzionamento in-domain dell'applicazione, sono state aggiunte le funzionalità di test cross-domain e training con fine-tuning (attivabili direttamente da linea di comando).

CNN

L'implementazione delle CNN per la Sentiment Classification utilizzata è codificata in Python ed è stata adattata al nostro caso dal progetto di Denny Britz reperibile anch'esso sulla piattaforma GitHub. Il lavoro di sviluppo si è concentrato principalmente sulle modifiche alle classi di processamento dell'input per adattare questa implementazione al caso di classificazione di recensioni Amazon.

6.5.2 Librerie utilizzate

Prima di analizzare i dettagli implementativi del progetto forniamo un quadro delle librerie utilizzate all'interno del progetto, utili al suo studio e sviluppo:

- TensorFlow, libreria software open source per l'apprendimento automatico già analizzata nelle sezioni precedenti;
- NumPy[29], estensione open source del linguaggio di programmazione Python, che aggiunge supporto per vettori e matrici multidimensionali e di grandi dimensioni e con funzioni matematiche di alto livello con cui operare;
- Argparse, libreria completa per l'elaborazione degli argomenti;
- Os, modulo contenente molte funzioni utili per manipolare file, processi, percorsi di file e directory;

- Csv, modulo utile per lavorare con dati esportati in file di testo da fogli elettronici e database;
- Datetime, modulo che include le funzioni e le classi per analizzare, formattare e compiere operazioni aritmetiche sulle date;
- Sys, modulo che fornisce l'accesso ad alcune variabili usate o mantenute dall'interprete Python, e a funzioni che interagiscono fortemente con l'interprete stesso;
- Time, modulo che espone le funzioni della libreria C per manipolare date e tempo;
- Json, modulo che fornisce una API per la conversione in memoria di oggetti Python verso una rappresentazione serializzata nota come "JavaScript Object Notation";
- Pathlib, modulo che offre classi rappresentanti percorsi su file system con semantiche appropriate a diversi sistemi operativi;
- Re, modulo che implementa la gestione efficiente delle espressioni regolari in stile Perl all'interno di Python.

6.5.3 Struttura del progetto

DMN+

L'implementazione delle DMN+ si sviluppa principalmente su cinque file Python:

- data_input: in questo modulo troviamo la gestione dei dataset e funzioni per il text processing come la creazione dei word embeddings;
- attention_gru_cell: modulo che implementa la cella della Gated Recurrent Unit utilizzata nelle DMN+ per il meccanismo di attenzione;
- dmn_plus: implementazione dell'architettura delle DMN+ descritta nel capitolo 3. Contiene gli iperparametri utilizzati durante testing e training;
- dmn_train: esegue l'addestramento della rete;
- dmn_test: esegue il test della rete.

Nelle directory di progetto troviamo invece le cartelle contenenti i file di dataset, i sommari di training ed i pesi salvati.

CNN

L'implementazione delle CNN è organizzata su quattro file Python:

- data_helpers: contiene la gestione dei dataset ed il processing dell'input;
- text_cnn: rappresenta l'implemenetazione del modello di CNN per la Sentiment Classification;
- train: contiene il codice per lanciare l'addestramento;
- eval: permette di testare la rete al termine dell'addestramento.

Nella directory di progetto si trovano inoltre, le cartelle contenenti i dataset ed i salvataggi della rete effettuati durante gli addestramenti.

Capitolo 7

Esperimenti

7.1 Introduzione

In questo capitolo esaminiamo gli esperimenti relativi all'applicazione di reti Dynamic Memory Network Plus e Convolutional Neural Networks a compiti di sentiment classification (modellati come problemi di Question Answering nel caso delle reti DMN+).
Vengono mostrati i risultati sia della classificazione binaria sia della classificazione fine-grained (su 5 classi).
In prima battuta analizzeremo come queste reti si comportano In-Domain su recensioni Amazon e successivamente analizzeremo i risultati derivanti dalla tecnica del Transfer Learning nel contesto della sentiment classification cross-domain applicando le DMN+, in un tipico scenario di adattamento del dominio.
Mostriamo come l'impiego del Transfer Learning riesca a migliorare in maniera efficiente e veloce l'accuratezza di predizione in scenari differenti da quelli di addestramento.
Applicheremo le DMN+ anche nella sentiment classification sul noto dataset Stanford Sentiment Treebank, il quale fornisce una suddivisione dei dati predefinita, emulando l'analogo esperimento realizzato con reti DMN e citato nel paper Ask Me Anything: Dynamic Memory Networks for Natural Language Processing (Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus & Richard Socher, 2016)[27] per verificare come quest'ultima versione di DMN migliori i risultati precedentemente ottenuti.
Lo stesso esperimento sarà riproposto variando il tipo di vettori caricati all'inizializzazione della rete: in particolare verrà mostrato anche il risultato ottenuto inizializzando gli embeddings con i vettori Google News per verificare se migliorano le prestazioni registrate con i vettori GloVe.
Verrà inoltre mostrato il confronto dei risultati degli esperimenti sul dataset Amazon tra quelli effettuati con reti DMN+ nel nostro progetto e quelli ottenuti dall'applicazione di reti LSTM.

Infine, viene riportata un'analisi dei tempi di addestramento, confrontando le architetture DMN+ e LSTM e, sempre a livello temporale, l'addestramento eseguito con una CPU Quad Core rispetto alla GPU Nvidia Titan X.

7.2 Amazon Reviews

7.2.1 Risultati In-Domain

I risultati In-Domain su recensioni Amazon si sviluppano su 3 serie di risultati corrispondenti a diverse dimensioni del set di esempi utilizzato in training/validation/test. Ciascuna serie riporta le accuratezze registrate nel rispettivo dominio abbreviato dalla corrispondente lettera secondo la legenda riportata nella sezione precedente.

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	Fine-Grained	Binary	Fine-Grained	Binary	Fine-Grained	Binary
DMN+ IN-DOMAIN						
M → M	73.00	90.75	59.30	87.85	60,86	88.65
E → E	69.00	89.50	77.85	91.35	65.14	92.36
J → J	65.50	86.00	69.60	90.35	62.99	90.04
B → B	62.00	87.00	44.29	76.20	62.25	89.33
average	**67.38**	**88.31**	**62.80**	**86.44**	**62.81**	**90.10**

Tabella 7.1: Risultati relativi alla classificazione In-Domain eseguita sul dataset di recensioni Amazon applicando reti DMN+. Le precisioni riportate negli esperimenti con 2k, 20k, 100k esempi sono state ottenute eseguendo rispettivamente 3, 2, 1 run.

DMN+ Binary classification

Osservando le accuratezze riscontrate in Figura 7.1, è possibile notare facilmente come la dimensione del dataset non influenzi in maniera importante la precisione della rete, la quale ottiene accuratezze vicine al 90% anche nel caso di dataset contenente 2000 esempi.

DMN+ Fine-grained classification

Nel caso di classificazione fine-grained è stato registrato, come prevedibile, un calo delle precisioni intorno al 15%, ma anche in questo caso, si può notare dalla Tabella 7.1 e Figura 7.2 che la dimensione del dataset non varia in modo consistente l'accuratezza della rete. Nel caso di addestramento con 80000 esempi, infatti, viene riscontrata una media addirittura inferiore a quella dei casi con 1600 e 16000 esempi in training.

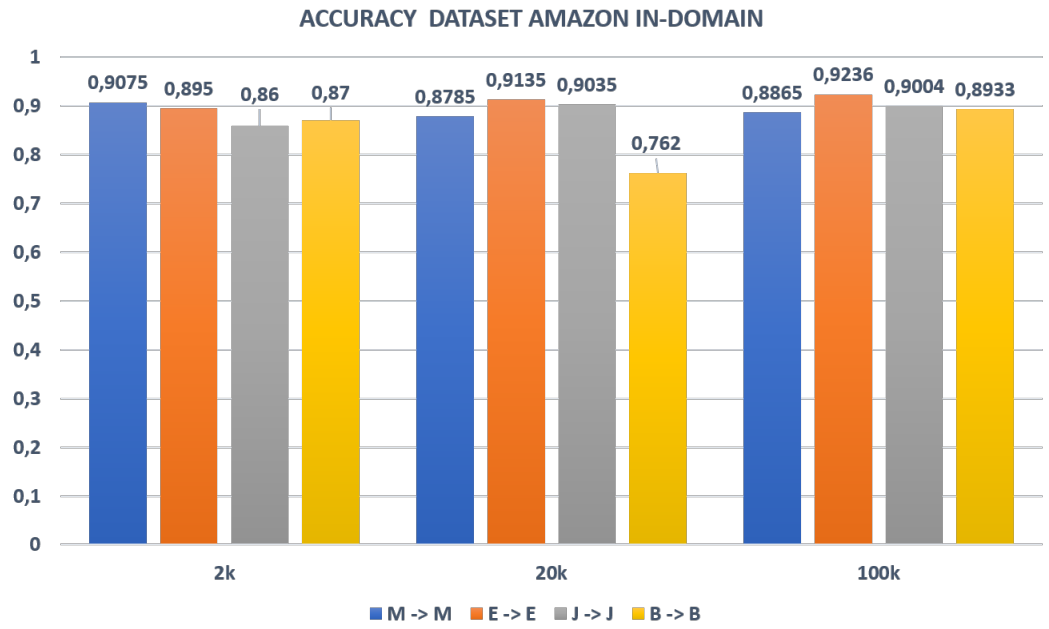


Figura 7.1: Risultati classificazione binaria In-Domain Dataset Amazon applicando DMNs+

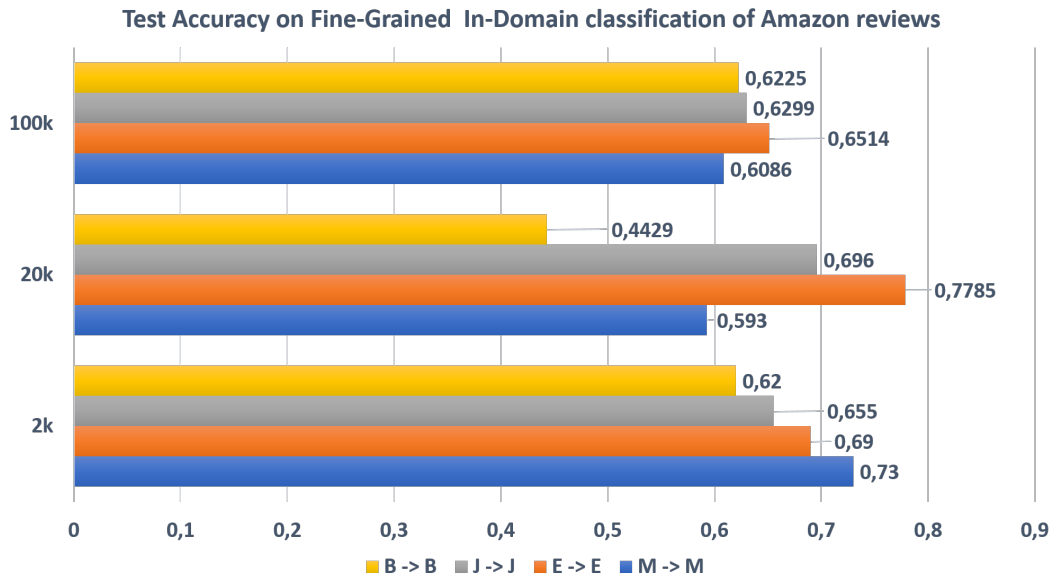


Figura 7.2: Risultati classificazione fine-grained In-Domain Dataset Amazon applicando DMNs+

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	Binary	Fine-Grained	Binary	Fine-Grained	Binary	Fine-Grained
CNN IN-DOMAIN						
M → M	100.00	89.50	99.00	90.35	99.05	87.50
E → E	98.50	90.00	99.25	88.55	98.41	88.93
J → J	100.00	89.50	99.55	91.05	99.12	90.06
B → B	99.50	90.50	99.50	91.70	100.00	88.70
average	**99.50**	**89.88**	**99.33**	**90.41**	**98.86**	**88.83**

Tabella 7.2: Risultati relativi alla classificazione In-Domain eseguita sul dataset di recensioni Amazon applicando reti CNN. Le precisioni riportate negli esperimenti sono state ottenute eseguendo un singolo run.

CNN Binary classification

Nella classificazione binaria le CNN si dimostrano davvero precise con un valore di accuratezza minimo del 98,41% come è possibile visualizzare nella Tabella 7.2. I risultati ottenuti in compiti di classificazione binaria In-Domain superano nella totalità dei casi le precisioni delle reti DMN+ e dimostrano come questo tipo di reti neurali riescano a performare in maniera ottima anche con input testuale. Va puntualizzato però, che in questi esperimenti è stato eseguito un solo run ed è possibile che il dataset usato sia più semplice da classificare.

CNN Fine-grained classification

Dalla Tabella 7.2 si evinche che nella classificazione fine-grained le reti CNN mantengono precisioni davvero elevate con un valore minimo del 87,50%.
Anche in questo caso le reti CNN migliorano i risultati ottenuti attraverso le DMN+ in compiti analoghi di classificazione fine-grained, ma bisogna sempre considerare che le precisioni sono riferite ad una singola esecuzione dell'esperimento nel quale potrebbe essere stato reperito un porzione di esempi particolarmente uniforme.

7.2.2 Risultati Cross-Domain

In questi esperimenti viene utilizzata la conoscenza estratta da un dominio sorgente per classificare istanze di un differente dominio destinazione. In primo luogo sono stati provati gli esperimenti senza applicazione della tecnica di Transfer Learning, i.e. è stato eseguito il test sulle istanze del dominio target utilizzando il modello salvato del dominio sorgente. Questo tipo di test però, non ha fornito risultati significativi in termini di precisione mostrando come la rete non sia sufficientemente adattabile ad un contesto diverso da quello

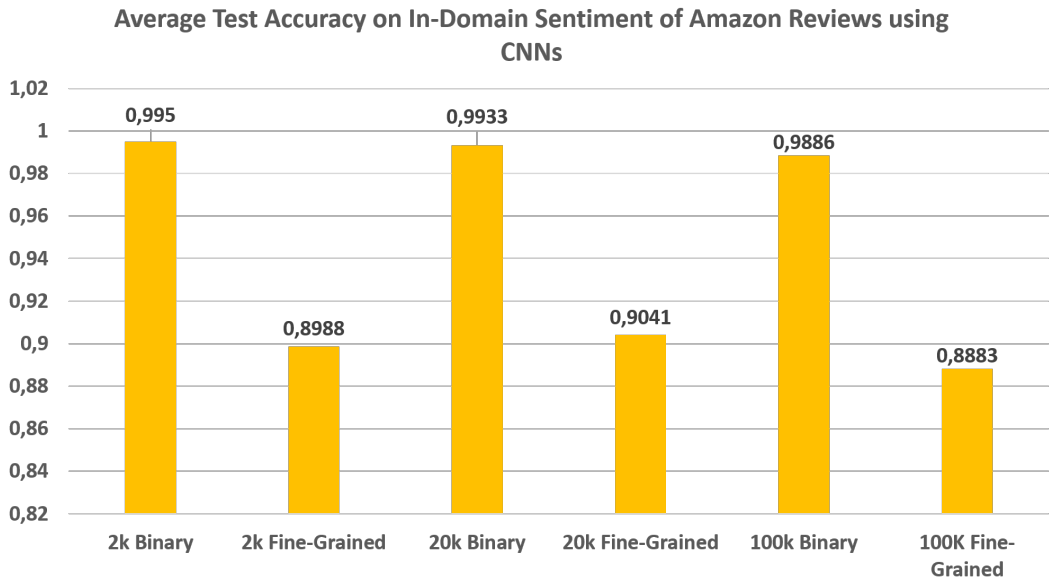


Figura 7.3: Media dei risultati ottenuti nella classificazione fine-grained In-Domain di recensioni Amazon applicando CNNs

di addestramento. A seguito di questi risultati è stata eseguita una seconda serie di esperimenti cross-domain applicando la tecnica di Transfer Learning del fine-tuning. Di seguito vengono mostrati i risultati relativi a quest'ultimo tipo di esperimenti mostrando come il fine-tuning migliora significativamente i risultati precedentemente riportati.

Binary classification

La Figura 7.4 mostra le accuratezze riscontrate dai test su un differente dominio rispetto a quello di addestramento, dopo aver eseguito un addestramento di fine-tuning con 200 istanze in training e 50 in validazione. La scelta di eseguire il fine-tuning con pochi elementi simula uno scenario reale dove i dati devono essere etichettati da un esperto umano. È possibile notare come già con un fine-tuning di 200 elementi la rete sia in grado di adattarsi ad un contesto differente, ed inoltre si riscontra un miglioramento delle precisioni rispetto ai risultati In-Domain di classificazione binaria con 2000 e 20000 istanze.

Fine-grained classification

Anche nel caso di classificazione fine-grained l'applicazione della tecnica di Transfer Learning ha portato ad un miglioramento dei risultati come riportato in Tabella 7.3 e in Figura 7.5. La precisione migliore della rete è stata raggiunta negli scenari con M come dominio target ed un modello sorgente addestrato su 2000 istanze, dove il valore di accuratezza si attesta sul 84.50%.

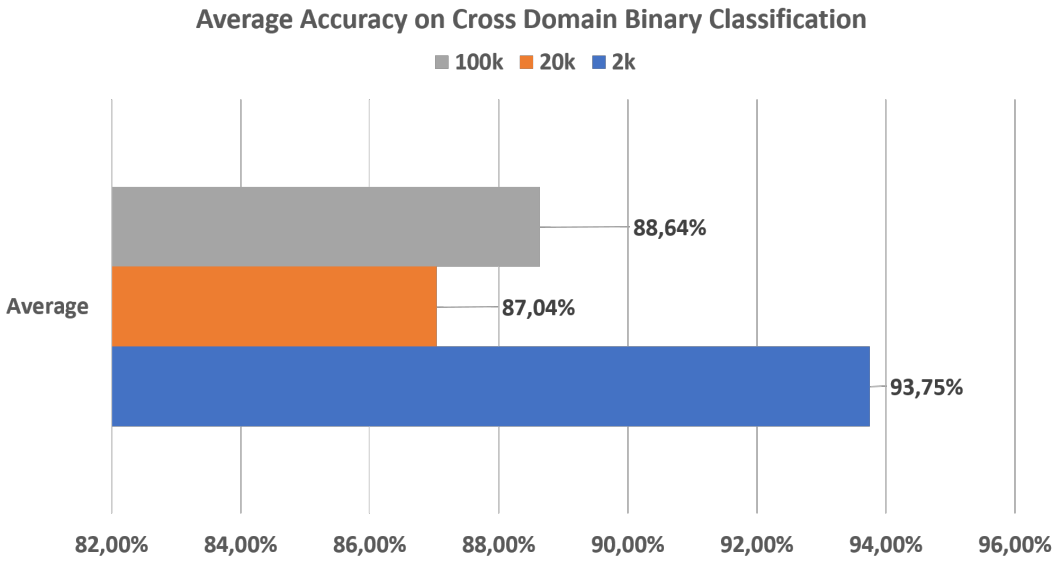


Figura 7.4: Media dei risultati ottenuti nella classificazione binaria Cross-Domain di recensioni Amazon con Fine-Tuning di 200 esempi applicando reti DMNs+

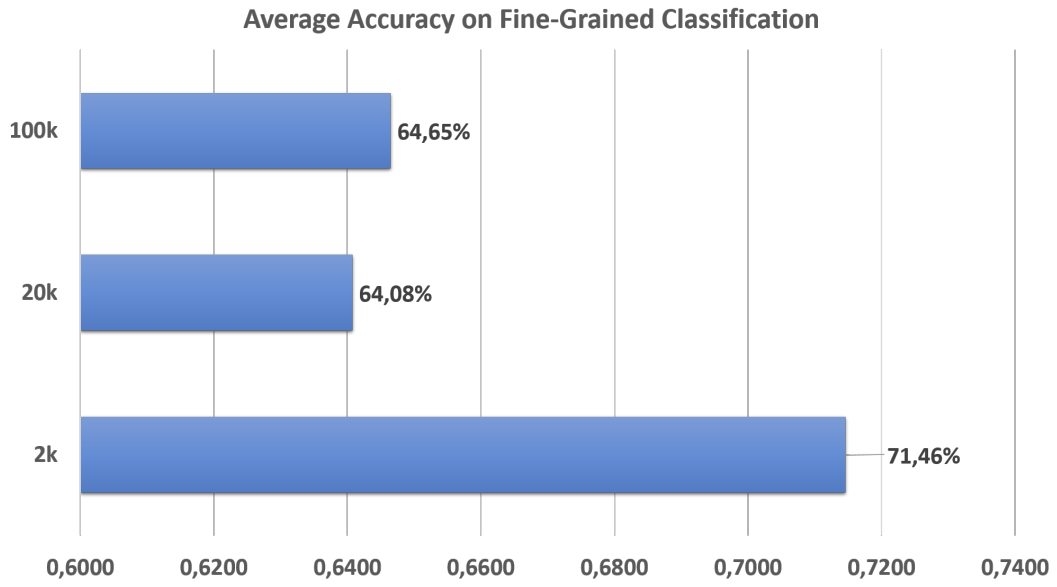


Figura 7.5: Media dei risultati ottenuti nella classificazione Fine-Grained Cross-Domain di recensioni Amazon con Fine-Tuning di 200 esempi applicando reti DMNs+

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	Fine-Grained	Binary	Fine-Grained	Binary	Fine-Grained	Binary
TRANSFER LEARNING						
M → E	70.00	95.50	67.40	86.90	65.86	89.06
M → J	66.50	94.50	57.15	81.40	66.28	86.64
M → B	74.00	92.00	67.40	92.00	62.25	89.33
E → M	84.50	98.00	64.35	87.85	63.22	89.54
E → J	39.00	87.00	57.15	81.40	66.28	86.64
E → B	74.00	92.00	67.40	92.00	62.25	89.33
J → M	84.50	98.00	64.35	87.85	66.22	89.54
J → E	70.00	95.50	67.40	86.90	65.86	89.06
J → B	74.00	92.00	67.40	92.00	62.20	89.33
B → M	84.50	98.00	64.35	87.85	63.22	89.54
B → E	70.00	95.50	67.40	86.90	65.86	89.06
B → J	66.50	87.00	57.15	81.40	66.28	86.64
average	**71.46**	**93.75**	**64.08**	**87.40**	**64.65**	**88.64**

Tabella 7.3: Risultati relativi alla classificazione Cross-Domain eseguita sul dataset di recensioni Amazon applicando la tecnica del Transfer Learning e reti DMNs+. Le precisioni riportate negli esperimenti sono state ottenute eseguendo un singolo run.

7.3 Stanford Sentiment Treebank

Il Stanford Sentiment Treebank (SST) (Socher et al., 2013)[32] è un noto dataset per la sentiment classification. Fornisce frasi, etichette valorizzate da 0 a 1 e una divisione dei dati tra training/validation/testing set. Presentiamo i risultati (Tabella 7.4, Figura 7.6 e 7.7) in due formati:

- predizione fine-grained, dove tutte le frasi complete del testing set devono essere classificate come horrible, negative, neutral, positive o wonderful;
- predizione binaria, dove tutte le frasi non neutrali del testing set devono essere classificate come positive o negative.

Per addestrare il modello, sono state usate tutte le frasi complete e un 50% delle frasi con etichetta durante ogni epoca. Nella validazione sono state utilizzate solamente le frasi complete.
Nella classificazione binaria, le frasi neutrali sono state rimosse dal dataset. La Tabella 7.4 mostra i risultati ottenuti comparati a quelli dello stesso esperimento eseguito tramite reti DMN originali nel paper Ask Me Anything: Dynamic Memory Networks for Natural

Language Processing (Ankit Kumar, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus & Richard Socher, 2016)[27].
Vengono anche riportate le precisioni dell'esperimento eseguito con reti DMN+ caricando i vettori di Google News per verificare l'efficacia di quest'ultimi rispetto ai vettori GloVe.
I risultati finali mostrano che i vettori Google News performano praticamente allo stesso modo nella classificazione binaria rispetto ai vettori GloVe, mentre nella classificazione fine-grained la precisione registrata con i vettori GloVe supera nettamente quella ottenuta caricando i vettori Google News.
La rete DMN+ (GloVe) raggiunge una precisione che rappresenta lo stato dell'arte sia nella classificazione binaria sia nella classificazione fine-grained.

Network Type (Trained Vectors loaded)	Test Accuracy (%)	
	Fine-Grained	Binary
STANFORD SENTIMENT TREEBANK		
DMN+ (GloVe)	72.19	98.47
DMN+ (Google News)	63.54	97.93
DMN (GloVe)	52.10	88.60

Tabella 7.4: Risultati relativi alla classificazione binaria e fine-grained eseguita sul dataset Stanford Sentiment Treebank. Le precisioni riportate negli esperimenti sono state ottenute eseguendo un singolo run.

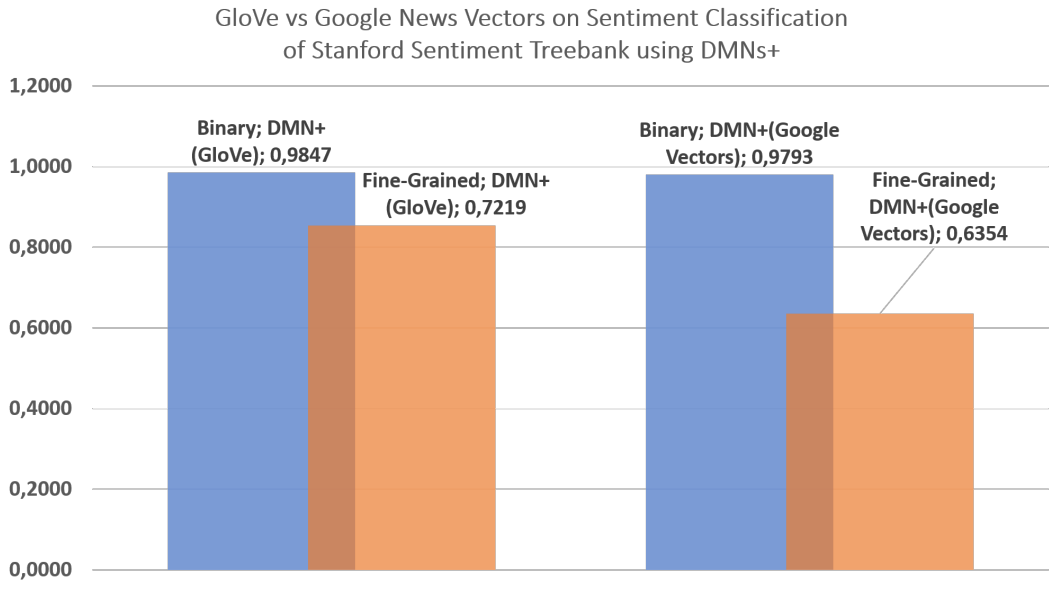


Figura 7.6: Confronto dei risultati ottenuti nella Sentiment Classification sullo Stanford Sentiment Treebank utilizzando DMN+ inizializzate con vettori GloVe e Google News

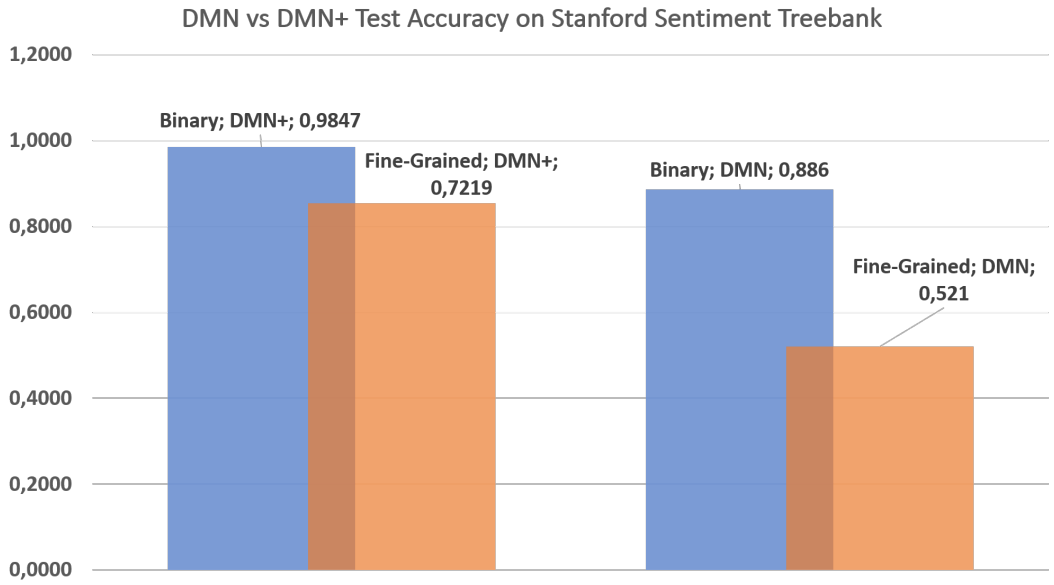


Figura 7.7: Confronto dei risultati ottenuti nella Sentiment Classification sullo Stanford Sentiment Treebank utilizzando DMN e DMN+ inizializzate entrambe con vettori GloVe

7.4 DMN+ vs. LSTM

In questa sezione vengono mostrati i risultati degli esperimenti su recensioni Amazon ottenuti applicando reti DMN+ e LSTM. Le accuratezze delle LSTM sono riportate da esperimenti precedenti a questo lavoro, effettuati da Federico Pucci.

7.4.1 In-Domain

Il grafico in Figura 7.8 e la Tabella 7.5 mostra come le DMN+ ottengono in media risultati superiori alle LSTM nella classificazione binaria di recensioni. In particolare la differenza di accuratezza si presenta negli esperimenti con dataset da 2k e 20k esempi, confermando ancora una volta come l'architettura DMN+ riesca a fornire buone accuratezze anche con dataset di piccola-media dimensione.

7.4.2 Cross-Domain

Nei risultati cross-domain le DMN+ si attestano superiori con uno scarto del 17,97% nel caso di dataset da 2k esempi e del 4,84% nel caso di 20k esempi. Anche negli esperimenti cross-domain lo scarto tra le due architetture è evidente quando si utilizzano dataset di piccole-medie dimensioni, mentre nell'esperimento da 100k esempi prevalgono le reti LSTM (GRU) con una differenza di precisione del 6,1%.

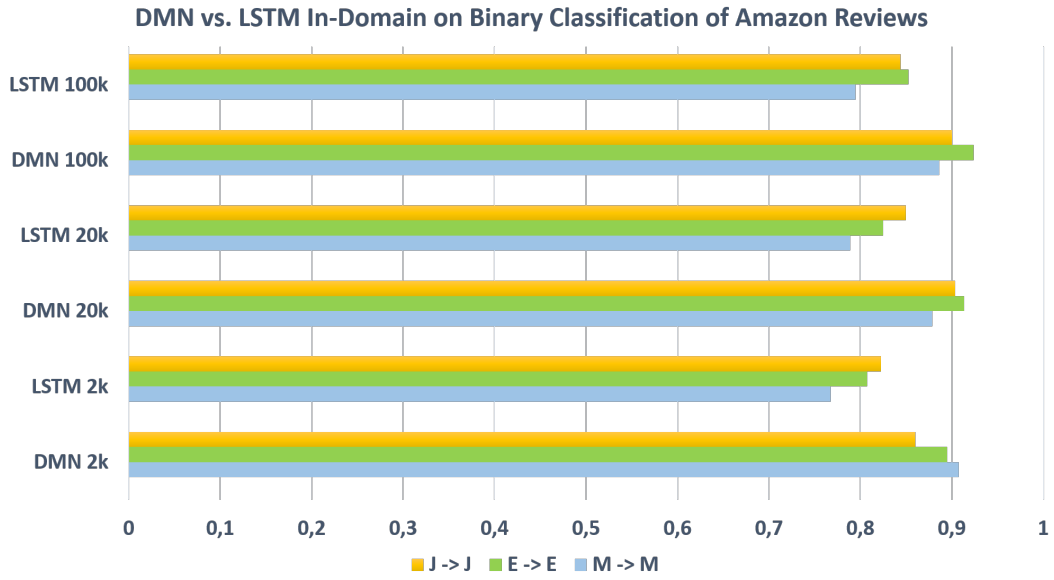


Figura 7.8: DMN+ vs. LSTM In-Domain

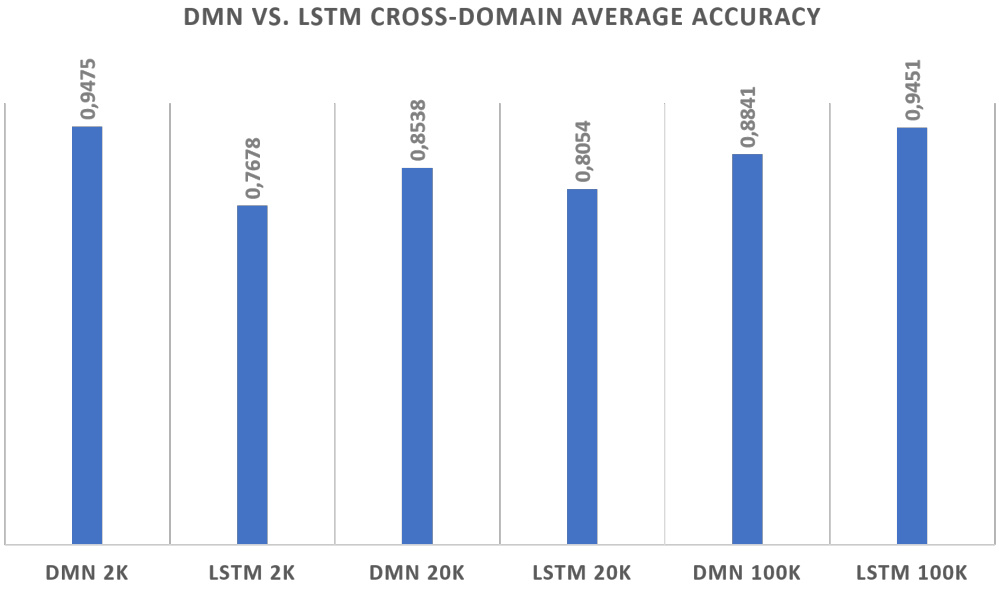


Figura 7.9: DMN+ vs. LSTM Cross-Domain

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	DMN+	LSTM	DMN+	LSTM	DMN+	LSTM
DMN+ VS. LSTM IN-DOMAIN						
M → M	90.75	76.75	87.85	78.92	88.65	79.50
E → E	89.50	80.75	91.35	82.50	92.36	85.24
J → J	86.00	82.25	90.35	85.00	90.04	84.40
average	**88.75**	**79.92**	**89.85**	**82.14**	**90.35**	**83.05**

Tabella 7.5: Risultati relativi alla classificazione In-Domain eseguita sul dataset di recensioni Amazon applicando reti DMN+ e LSTM

	Test Accuracy (%)					
	2k		20k		100k	
Domain(s)	DMN+	LSTM	DMN+	LSTM	DMN+	LSTM
DMN+ VS. LSTM CROSS-DOMAIN						
M → E	95.50	76.20	86.90	80.29	89.06	94.29
M → J	94.50	75.45	81.40	81.33	86.64	95.84
E → M	98.00	76.84	87.85	78.20	89.54	95.82
E → J	87.00	81.12	81.40	84.85	86.64	96.65
J → M	98.00	70.78	87.85	74.80	89.54	89.90
J → E	95.50	80.30	86.90	84.04	89.06	94.53
average	**94.75**	**76.78**	**85.38**	**80.54**	**88.41**	**94.51**

Tabella 7.6: Risultati relativi alla classificazione Cross-Domain eseguita sul dataset di recensioni Amazon applicando reti DMN+ e LSTM

7.5 Analisi dei tempi

Al fine di fornire un quadro complessivo dei tempi necessari all'addestramento delle reti riportiamo una tabella riassuntiva dei valori.

Nelle sottosezioni successive vengono riportati grafici per confrontare, in primo luogo, le architetture di reti neurali e in seguito, l'incidenza sui tempi di addestramento dell'utilizzo di una GPU.

L'addestramento della rete 'e avvenuto su due computer in concorrenza entrambi di proprietà dell'Università. Il primo computer utilizzato dispone di un processore AMD Quad Core e 16GB di memoria RAM, mentre il secondo elaboratore è composto da i5 di sesta generazione, GPU Nvidia Titan X (utilizzata per il calcolo durante l'addestramento) e 32GB di

memoria RAM.
I valori elencati sono approssimati sull'ordine dei 30 minuti poich'e lo stesso test, con la stessa configurazione, sullo stesso dataset porterebbe sicuramente a variazioni di alcuni minuti. Anche le variazioni relative al dominio risultano minime, e per evitare ridondanza di dati viene riportata una media dei tempi di addestramento senza distinzione in base al dominio di applicazione ma in base alla dimensione del dataset di training, del tipo di reti neurali e architettura hardware utilizzata per il calcolo. [1]

	Training Time (Hours)		
Domain Size	LSTM (CPU)	DMN+ (CPU)	DMN+ (GPU)
1.6k	0.5	1.07	0.65
16k	3	4.19	2.01
80k	16	229.5[1]	22.95

Tabella 7.7: Tempo utilizzato (in ore) nei processi d'addestramento delle reti neurali. I valori sono classificati in relazione al tipo di reti neurali (Long-Short-Term-Memory o Dynamic Memory Network Plus), all'architettura hardware utilizzata per il calcolo ed alla dimensione del dataset di training.

7.5.1 Tempi In-Domain

La Figura 7.10 mostra i tempi di addestramento della rete DMN+ in compiti di classificazione In-Domain su recensioni Amazon. L'andamento, al crescere della dimensione del dataset, è esponenziale ed infatti è stato necessario l'utilizzo della GPU Nvidia Titan X per avere risultati in tempi rapidi con dataset da 100000 esempi.

7.5.2 DMN+ vs. LSTM

Dall'analisi del grafico a linee in Figura 7.11 possiamo confrontare i tempi di addestramento delle architetture di reti neurali DMN+ e LSTM. Si evince che in generale le LSTM si dimostrano più veloci da addestrare anche se per dataset di piccole-medie dimensioni le due architetture richiedono sostanzialmente lo stesso tempo di addestramento mentre la differenza cresce in maniera importante nel caso di traning con 100000 esempi. In questo confronto bisogna considerare la differenza tra le due architetture del criterio di fine addestramento e del valore di dropout applicato nelle due serie di esperimenti (0.5 per le LSTM vs. 0.9 nelle DMN+).

[1] Il valore annotato rappresenta una stima ottenuta moltiplicando per 10 il tempo di addestramento della rete DMN+ eseguito su 80000 esempi con GPU Titan X.

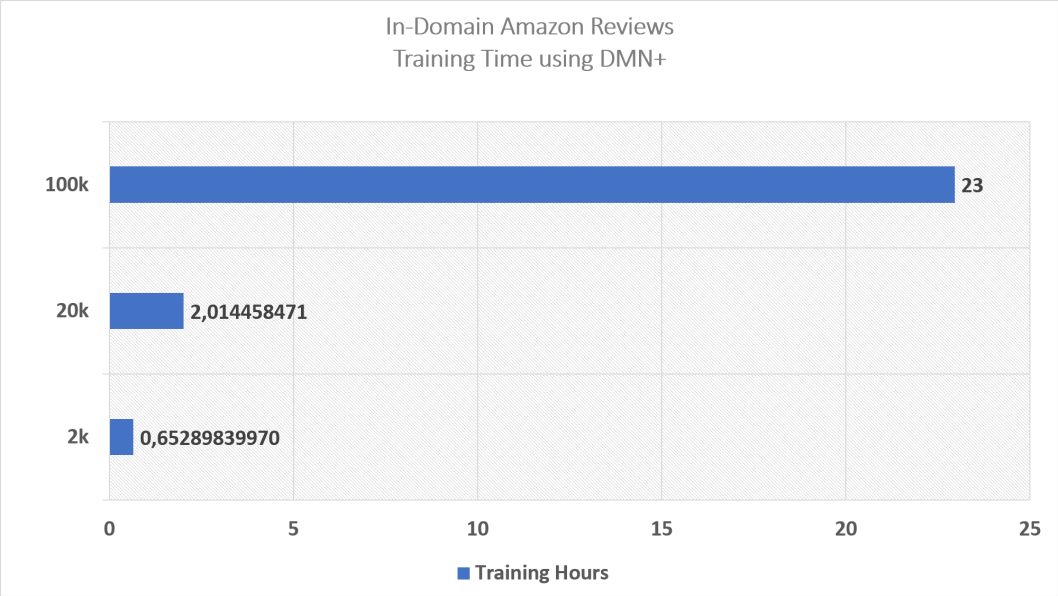


Figura 7.10: Tempi di addestramento In-Domain DMN+ su recensioni Amazon

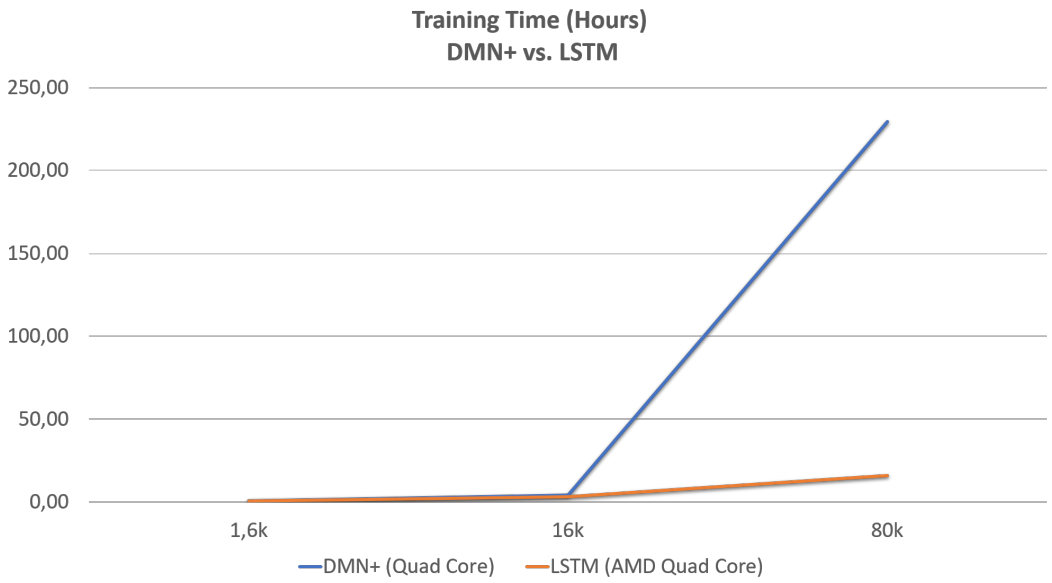


Figura 7.11: Tempi di addestramento DMN+ vs. LSTM

7.5.3 CPU vs. GPU

Il grafico in Figura 7.12 mostra una media dei tempi di addestramento eseguiti su 1600, 16000 e 80000 esempi, della rete DMN+ ottenuti con una CPU Quad Core e una GPU Nvidia Titan X. La differenza sostanziale di prestazioni si rileva nell'addestramento con 80000 esempi anche se il valore rilevato per la CPU Quad Core rappresenta una stima (ottenuta moltiplicando per un fattore 10 il tempo registrato per 80k esempi della GPU Titan X) in quanto gli esperimenti con dataset da 100000 esempi sono stati effettuati solamente sulla macchina con GPU.

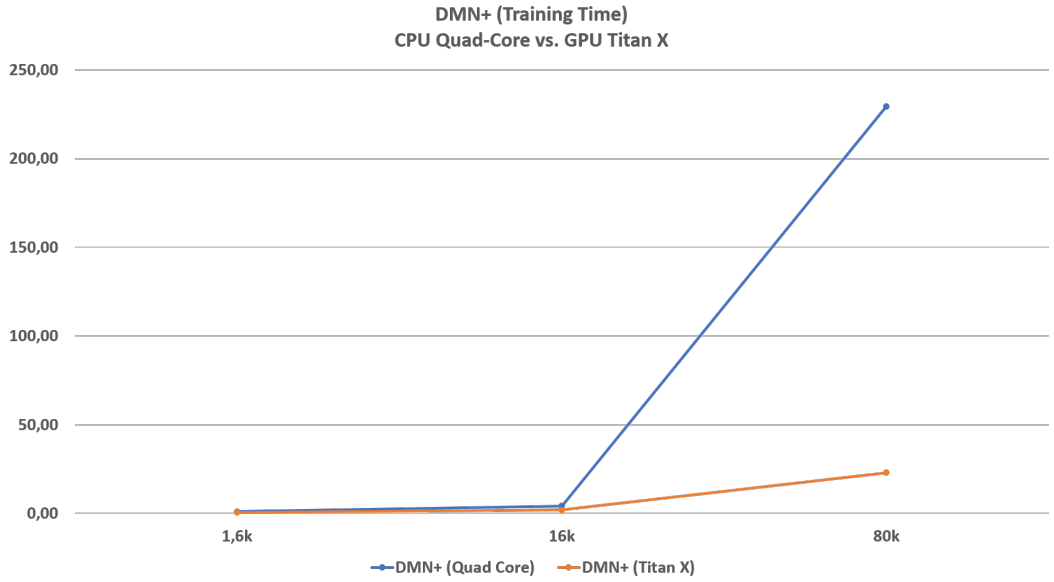


Figura 7.12: Tempi di addestramento CPU Quad Core vs. GPU Nvidia Titan X

Capitolo 8

Conclusioni e sviluppi futuri

8.1 Sintesi dei risultati

In questo lavoro abbiamo applicato le Dynamic Memory Networks, in particolare nella versione Plus, a compiti di Sentiment Classification, sia In-Domain sia Cross-Domain, modellandoli come problemi di Question Answering ed utilizzando la codifica del testo GloVe e Position Encoding.
La precisione della rete ottenuta In-Domain nella classificazione binaria di recensioni Amazon si attesta intorno al 90% mentre nella classificazione fine-grained è stata raggiunta un'accuratezza media del 67%. Le reti DMN+ si sono dimostrate efficaci anche con dataset da 2000 esempi, riscontrando una lieve differenza con le precisioni di esperimenti con dataset da 20000 e 100000 esempi.
La classificazione Cross-Domain sul dataset Amazon senza l'utilizzo di una tecnica di Transfer Learning esplicita non ha avuto successo, tuttavia applicando una tecnica di Transfer Learning esplicita su scenario di adattamento del dominio le reti DMN+ hanno raggiunto accuratezze che superano il 90% in diversi casi di classificazione binaria e il 60% nella quasi totalità degli esperimenti di classificazione fine-grained.
Nell'applicazione della tecnica di Transfer Learning abbiamo utilizzato solamente 200 recensioni in training e 50 in validazione. Ogni test è stato eseguito su un numero di istanze pari agli al numero di esempi utilizzati nel testing del rispettivo dominio sorgente.
Sebbene le precisioni registrate nella classificazione binaria siano ottime, i risultati più importanti sono stati ottenuti nella classificazione Fine-Grained, che si è dimostrata un task difficile in letteratura, nella quale sono state ottenute accuratezze inferiori al 60%.
I risultati ottenuti dalla classificazione di recensioni Amazon sono stati confrontati con esperimenti analoghi eseguiti tramite reti LSTM ed in generale le reti DMN+ hanno registrato precisioni maggiori In-Domain di circa 7 punti percentuali, mentre nel caso dello scenario Cross-Domain le differenze di accuratezza si attestano sul 17.97% e 4.84% rispettivamente nei dataset di piccole-medie dimensioni. Nel caso di classificazione Cross-Domain,

invece, su dataset da 100k esempi l'architettura LSTM (GRU) ha ottenuto, in media, una maggior precisione (6.1%).
Abbiamo ottenuto ottimi risultati anche nella classificazione binaria (98.47%) e fine-grained (72.19%) di esempi provenienti dal noto dataset Stanford Sentiment Treebank, superando le precisioni ottenute in precedenti esperimenti analoghi eseguiti con reti DMN originali. Inoltre, è stato effettuato un confronto tra la precisione ottenuta inizializzando la rete DMN+ con vettori Google News e GloVe. Gli esperimenti con vettori GloVe hanno registrato precisioni più alte sia nella classificazione binaria sia fine-grained con una differenza di prestazione sostanziale nell'ultimo caso.
Nella classificazione In-Domain di recensioni Amazon sono state applicate anche Convolutional Neural Networks per la Sentence Classification per confrontare le performance di questa architettura con quelle delle reti LSTM e DMN+ precedentemente utilizzate. Le accuratezze registrate dalle reti CNN hanno registrato un valore medio del 99% nella classificazione binaria e del 89% nel caso di classificazione fine-grained. I risultati ottenuti dall'applicazione di Convolutional Neural Networks In-Domain su recensioni Amazon hanno superato di circa 9 punti percentuali nella classificazione binaria e 22 punti percentuali nella classificazione fine-grained, le precisioni registrate dalle reti DMN+. Nel confronto tra i risultati delle DMNs e CNNs bisogna considerare che è stato eseguito un solo run per esperimenti eseguiti con quest'ultimo tipo di reti. In questi esperimenti quindi, potrebbe essere stata selezionata una porzione uniforme (e quindi semplice da classificare) del dataset completo. Questo fatto potrebbe giustificare le elevate accuratezze registrate e sarà quindi interessante sperimentare ulteriormente questo tipo di reti neurali.

8.2 Sviluppi futuri

Sviluppi futuri del progetto presentato potranno riguardare:

- ottimizzazione del consumo di memoria usando i generatori Python, in quanto l'implementazione della rete DMN+ mantiene il dataset in memoria RAM richiedendo molto spazio (si potrà memorizzare in RAM solamente il batch corrente durante l'addestramento);
- miglioramento della fase di preprocessing utilizzando una libreria esterna;
- valutazione dell'impatto della dimensione del traning set nel caso di fine-tuning;
- applicazione delle CNN in compiti di cross-domain classification e sul dataset Stanford Sentiment Treebank, visti gli ottimi risultati ottenuti In-Domain da questo tipo di reti neurali.

Nei futuri esperimenti si potranno tentare approcci differenti cercando di migliorare la capacità di predizione cross-domain analizzando anche i lavori prodotti recentemente in letteratura ([5], [6]). A livello di ricerca, sarà interessante rivolgere l'attenzione verso tecniche di Transfer Learning basate sulle relazioni semantiche (e non sui word embeddings, considerando che le codifiche Word2Vec e GloVe riescono già parzialmente a catturarne) e l'analisi morfologica del testo utilizzando il noto database lessicale WordNet ed analizzando i recenti lavori prodotti in letteratura, per migliorare la capacità di predizione delle reti neurali.

Bibliografia

[1] Rosenblatt, F. (1958) The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.

[2] Rosenblatt, F. Principles of Neurodynamics (Spartan, Washington, DC, 1961).

[3] Minsky, M. L. & Papert, S. Perceptrons (MIT, Cambridge, 1969).

[4] Learning representations by back-propagating errors; Rumelhart, Hinton & Williams, 1986

[5] Cross-Domain Sentiment Classification via Spectral Feature Alignment

[6] Cross-Domain Sentiment Classification using a Sentiment Sensitive Thesaurus

[7] Breiman (2001) Random Forests

[8] Mikolov et al. (2010), Recurrent neural network based language model

[9] Bodèn (2011), A guide to recurrent neural networks and backpropagation

[10] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):15-166.

[11] Hochreiter, Schmidhuber (1997) Long Short-Term Memory

[12] Xiang Zhang, Yann LeCun (2016), Text Understanding from Scratch.

[13] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng and Christopher Potts, EM NLP (2013) Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

[14] Aditya Timmaraju, Vikesh Khanna, Sentiment Analysis on Movie Reviews using Recursive and Recurrent Neural Network Architectures.

[15] Convolutional Neural Networks for Sentence Classification, Yoon Kim (2014)

[16] A Convolutional Neural Network for Modelling Sentences, Nal Kalchbrenner, Edward Grefenstette & Phil Blunsom (2014)

[17] Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov, Kai Chen, Greg Corrado & Jeffrey Dean (2013)

[18] Principles of training multi-layer neural network using backpropagation

[19] GloVe: Global Vectors for Word Representation, Jeffrey Pennington, Richard Socher & Christopher D. Manning (2014)

[20] Dynamic Memory Networks for Visual and Textual Question Answering, Caiming Xiong, Stephen Merity & Richard Socher (2016)

[21] Dynamic Memory Networks for Question Answering, Arushi Raghuvanshi & Patrick Chase

[22] Adam: A Method for Stochastic Optimization, Diederik P. Kingma, Jimmy Ba (2017)

[23] Neural Turing Machines, Alex Graves, Greg Wayne & Ivo Danihelka (2014)

[24] Memory Networks, JasonWeston, Sumit Chopra & Antoine Bordes (2015)

[25] End-To-End Memory Networks, Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston & Rob Fergus (2015)

[26] Hybrid computing using a neural network with dynamic external memory, Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu & Demis Hassabis (2016)

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

[28] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345-1359.

[29] NumPy

[30] TensorFlow

[31] Keras

[32] Stanford Sentiment Treebank

[33] Collection of datasets from Amazon, grouped by category.

Ringraziamenti

Voglio ringraziare innanzitutto il prof. Gianluca Moro per avermi avvicinato ad un campo affascinante e destinato a futuri sviluppi come quello del Deep Learning, per la fiducia dimostratami e per la sua meticolosità.
Ringrazio inoltre i suoi collaboratori: Andrea Pagliarani e Roberto Pasolini, per la grande disponibilità e pazienza con la quale mi hanno seguito nello sviluppo della tesi.
Il ringraziamento più grande va alla mia famiglia, per essersi sempre spesa per me in ogni situazione ed in particolare per la mia istruzione. Spero che il conseguimento di questa laurea ripaghi, almeno in parte, la pazienza, l'impegno e la fiducia profusi.
Ci tengo a ringraziare anche Alessandra e Gabriele che mi hanno supportato senza tirarsi mai indietro all'inizio di questo percorso.
Il ringraziamento finale va agli amici di una vita ed ai miei compagni di facoltà Mattia, Federico, Martina, Alessandro e Lorenzo per avermi accompagnato con entusiasmo e spirito di collaborazione in questi anni.

Guida agli esperimenti

Di seguito si riportano le istruzioni per avviare gli esperimenti descritti in precedenza distinguendo tra processo di training e testing. Tutti i comandi elencati successivamente, richiedono il posizionamento via terminale all'interno della cartella di progetto corrispondente al tipo di rete neurale da utilizzare.

.1 Training DMN+

Per avviare un addestramento è sufficiente avviare il file Python dmn_train.py con gli opportuni flags. Nelle sezioni successive verranno spiegati la logica dei flag ed i comandi principali utilizzati nel progetto.

.1.1 Lanciare un nuovo addestramento

Per lanciare un addestramento, senza partire da un modello salvato in precedenza, occorre impartire il comando:

```
python dmn_train.py -b xx
```

dove il flag -b indica il numero del task su cui effettuare l'addestramento. I numeri dei task sono salvati all'interno del file review_map.json la cui struttura è formata dal una label numerica equivalente al numero del task e dal corrispondente nome come valore associato. Di default verranno utilizzati 2000 esempi tra training/validation/testing.

.1.2 Variare il numero di esempi utilizzati

Per variare la dimensione del set di esempi da utilizzare, è sufficiente specificare il numero complessivo di istanze da campionare dal dataset originale. Il numero specificato al flag -size rappresenta la dimensione totale di esempi utilizzati in training/validation/testing.

```
python dmn_train.py -b xx -size xx
```

.1.3 Effettuare Fine-Tuning su un modello salvato

Per lanciare un addestramento di Fine-Tuning su un modello precedentemente salvato occorre lanciare il seguente comando:

```
python dmn_train.py -b xx -fn xx
```

dove il valore del flag -b rappresenta l'indice del task precedentemente eseguito di cui caricare i pesi (source), e il valore del flag -fn corrisponde all'indice del task che utilizza il dataset con cui si vuole fare fine-tuning (target). Di default vengono usate 2000 istanze complessive per effettuare il fine-tuning (sempre considerando complessivamente training/-validation/testing). Prima di eseguire il comando è necessario verificare che nella directory 'data/en-sent' siano presenti i file txt contenenti gli esempi (e.g. reviews_Books_10_train.txt o reviews_Books_10_test.txt) del dominio source e i corrispondenti pesi (e.g. task23.weights.data-00000-of-00001, task23.weights.index, task23.weights.meta) nella directory 'weights'.

.1.4 Variare dimensione del Fine-Tuning

Per utilizzare un numero arbitrario di istanze per il fine-tuning è sufficiente specificarlo con il flag -fns come mostrato di seguito:

```
python dmn_train.py -b xx -fn xx -fns xx
```

.1.5 Riprendere un addestramento interrotto

Per riprendere un addestramento dall'ultimo salvataggio effettuato è possibile utilizzare il flag restore nel seguente comando:

```
python dmn_train.py -b xx -restore
```

.1.6 Eliminare i file di training/validation/testing al termine del processo

Per eliminare i file con estensione txt, contenenti gli esempi utilizzati nell'addestramento e nel testing, è possibile aggiungere il flag -clean al comando di train:

```
python dmn_train.py -b xx -clean
```

se l'argomento -fn è specificato, l'applicativo eliminerà anche i file del dominio target utilizzato per il fine-tuning.

.1.7 Specificare il numero di esecuzioni dell'addestramento

Per eseguire più volte l'addestramento della rete, occorre specificarne il numero con il flag –n:

```
python dmn_train.py -b xx -n xx
```

.1.8 Specificare il valore di perdita L2

Per specificare la costante di perdita L2 è possibile utilizzare il flag -l:

```
python dmn_train.py -b xx -l xx
```

.1.9 Abilitare la supervisione

Utilizzando il flag -s la rete utilizzerà i fatti di supporto etichettati durante l'addestramento:

```
python dmn_train.py -b xx -s
```

.2 Training CNN

Per effettuare l'addestramento della rete CNN è necessario eseguire il comando:

```
python train.py
```

a cui possono essere applicati i seguenti argomenti opzionali:

- -h, –help mostra il messaggio di aiuto con i flag opzionali disponibili e chiude il processo;
- –embedding_dim EMBEDDING_DIM Dimensione degli embedding dei caratteri (default: 128);
- –filter_sizes FILTER_SIZES Dimensioni dei filtri separate dalla virgola (default: '3,4,5');
- –num_filters NUM_FILTERS Numero dei filtri per ogni dimensione di filtro (default: 128)
- –l2_reg_lambda L2_REG_LAMBDA regolarizzazione L2 lambda (default: 0.0);
- –dropout_keep_prob DROPOUT_KEEP_PROB Dropout keep probability (default: 0.5);
- –batch_size BATCH_SIZE Dimensione del Batch (default: 64);
- –num_epochs NUM_EPOCHS numero di epoche di addestramento (default: 100);

- –evaluate_every EVALUATE_EVERY Valuta il modello sul validation set dopo questo numero di step (default: 100);
- –checkpoint_every CHECKPOINT_EVERY Salva il modello dopo questo numero di step (default: 100);
- –allow_soft_placement ALLOW_SOFT_PLACEMENT Abilita il soft device placement;
- –noallow_soft_placement;
- –log_device_placement LOG_DEVICE_PLACEMENT Log placement delle operazioni sulla macchina;
- –nolog_device_placement;
- –data_size Specifica il numero di esempi da utilizzare complessivamente tra training/-validation/testing.
- –positive_data_file Specifica il file da cui reperire gli esempi di polarità positiva;
- –negative_data_file Specifica il file da cui reperire gli esempi di polarità negativa;

.3 Testing DMN+

Per avviare il test della rete neurale è sufficiente avviare il file Python dmn_test.py con gli opportuni flags. Nelle sezioni successive verranno spiegati la logica dei flag ed i comandi principali utilizzati nel progetto.

.3.1 In-Domain

Per effettuare un test In-Domain su un modello precedentemente addestrato occorre lanciare il comando:

```
python dmn_test.py -b xx
```

dove il flag -b corrisponde all'indice del task lanciato in precedenza per l'addestramento della rete. Per conoscere gli indici dei task è possibile fare riferimento sempre al file review_map.json

.3.2 Cross-Domain

È possibile testare un modello salvato su un diverso dominio rispetto quello di addestramento, lanciando il seguente comando:

```
python dmn_test.py -b xx -d xx
```

dove il valore del flag -d corrisponde all'indice del task target sul quale testare il modello salvato dopo l'addestramento sul task specificato dal flag -b (source). Di default l'applicativo recupera 2000 istanze del dominio target per il test cross-domain.
Come per l'addestramento di fine-tuning, è necessario verificare che nella directory 'data/en-sent' siano presenti i file txt contenenti gli esempi (e.g. reviews_Books_10_train.txt o reviews_Books_10_test.txt) del dominio source e i corrispondenti pesi (e.g. task25.weights.data-00000-of-00001, task25.weights.index, task25.weights.meta) nella directory 'weights'.

.3.3 Variare numero di esempi nei test Cross-Domain

Per variare il numero di istanze del dominio target è sufficiente specificarlo con il flag size nel seguente modo:

```
python dmn_test.py -b xx -d xx -size xx
```

.3.4 Eliminare i file di training/validation/testing set al termine del processo

Per eliminare i file con estensione txt, contenenti gli esempi utilizzati nell'addestramento e nel testing, è possibile aggiungere il flag -clean al comando di test:

```
python dmn_test.py -b xx -clean
```

se l'argomento -d è specificato, l'applicativo eliminerà anche i file del dominio target utilizzato per il testing.

.4 Testing CNN

L'avvio del test sulle reti CNN avviene lanciando il comando:

```
python eval.py --checkpoint_dir="./runs/xxx/checkpoints/"
```

dove i caratteri ignoti rappresentano il numero dell'ultimo salvataggio della rete effettuato. Può essere visualizzato al termine dell'addestramento oppure nella cartella runs all'interno del progetto. È possibile anche specificare il flag –eval_train che permette di testare la rete sugli esempi del traning set. Per testare la rete su un determinato file, è sufficiente specificarne il percorso con i flag positive_data_file e negative_data_file come nel caso dell'addestramento.