

UNIVERSITÀ DEGLI STUDI DI UDINE

---

Dipartimento di Scienze Matematiche, Informatiche e Fisiche  
Corso di Laurea Magistrale in Informatica

Tesi di Laurea

**ANALISI E ANNOTAZIONE  
AUTOMATICA DI TESTI BASATA SU  
RETI NEURALI RICORRENTI**

Relatore:  
Prof. CARLO TASSO

Laureanda:  
ELISA ANTOLLI

Correlatori:  
Prof. GIUSEPPE SERRA  
Prof. MARCO BASALDELLA

---

ANNO ACCADEMICO 2016-2017



Alla mia famiglia, il mio più grande sostegno; in particolare a mio padre Aldo, a mia madre Luzia e a coloro con i quali condivido la cosiddetta *cognatio spiritualis*: zia Liliana e zio Paolo.

## Ringraziamenti

Finalmente il giorno è arrivato: è stato un periodo di profondo apprendimento, non solo a livello scientifico, ma anche personale.

Vorrei ringraziare tutte le persone che mi hanno sostenuto e aiutato durante questo periodo, ma innanzitutto voglio ringraziare Dio: in diversi momenti del mio percorso accademico mi sono sentita debole ed impotente; quando tutto sembrava impossibile e troppo difficile, Lui è stato il mio sostegno.

Desidero inoltre ricordare tutti coloro che mi hanno guidato nella stesura della tesi, con suggerimenti, critiche ed osservazioni: il Prof. Tasso, relatore di questa tesi di laurea, il dott. Basaldella per l'ausilio nell'elaborazione e specialmente il correlatore Prof. Serra, che oltre all'aiuto fornитomi, è sempre stato disponibile, ha creduto nel mio potenziale e mi ha incoraggiato a dare il meglio di me stessa. A loro pongo i miei più cordiali ringraziamenti.

Proseguo con i miei amici e colleghi che mi hanno appoggiato, incoraggiato e speso parte del proprio tempo per aiutarmi. Vorrei infine ringraziare le persone a me più care: gli amici più vicini e tutta la mia famiglia, quindi i miei genitori, sorella, nipote, cugini, nonna, zii e in special modo zia Lili, so come a lei non piacerebbe essere messa in evidenza ma devo farlo perché sicuramente lei sa più di me su questa tesi. Per concludere, tutti coloro che sanno di avere un posto speciale e insostituibile nel mio cuore.

*"Ohana means family. Family means nobody gets left behind or forgotten."*

Un sentito grazie a tutti!

# Indice

<b>Elenco delle figure</b>	<b>iv</b>
<b>1 Introduzione</b>	<b>2</b>
<b>2 Obiettivi, fondamenti e stato dell'arte</b>	<b>8</b>
2.1 Obiettivi . . . . .	8
2.2 Natural Language Processing . . . . .	9
2.3 Machine Learning . . . . .	10
2.3.1 Deep Learning . . . . .	13
2.4 Estrazione di parole e frasi chiave . . . . .	13
2.5 Related works . . . . .	15
2.6 Epilogo . . . . .	19
<b>3 Reti neurali</b>	<b>20</b>
3.1 Le reti neurali artificiali . . . . .	20
3.1.1 Paradigmi di apprendimento . . . . .	21
3.1.2 Training, validation e test . . . . .	24
3.1.3 L'architettura della rete . . . . .	25
3.2 Il Perceptron . . . . .	27
3.3 L'MLP ed il Backpropagation . . . . .	28
3.4 Epilogo . . . . .	31
<b>4 Approccio proposto</b>	<b>32</b>
4.1 La strutturazione . . . . .	32
4.1.1 Word Embeddings . . . . .	36
4.1.2 Funzioni di attivazione . . . . .	37
4.1.3 Funzioni di costo . . . . .	37
4.1.4 Algoritmo di ottimizzazione . . . . .	38
4.1.5 Altri iperparametri . . . . .	38
4.2 Epilogo . . . . .	39

<b>5 Pre-processing dei dati</b>	40
5.1 L'elaborazione del linguaggio naturale . . . . .	40
5.1.1 Tokenization . . . . .	40
5.2 L'elaborazione dei dati di input . . . . .	43
5.3 Epilogo . . . . .	44
<b>6 Word Embeddings</b>	45
6.1 Obiettivo . . . . .	46
6.2 Approcci . . . . .	47
6.2.1 Continuous Bag of Words . . . . .	48
6.2.2 Skip-gram . . . . .	51
6.3 Gli ambiti di applicazione Word2Vec . . . . .	59
6.4 Epilogo . . . . .	60
<b>7 Bidirectional LSTM</b>	61
7.1 Le Reti Neurali Ricorrenti . . . . .	61
7.2 Long-short term memory . . . . .	63
7.3 Bidirectional Long Short-term memory . . . . .	68
7.4 Epilogo . . . . .	70
<b>8 Strumenti</b>	71
8.1 L'impostazione dell'ambiente . . . . .	71
8.2 Epilogo . . . . .	73
<b>9 Risultati sperimentali</b>	74
9.1 Metriche di valutazione . . . . .	74
9.2 I Dataset utilizzati . . . . .	76
9.3 Impostazione dei parametri . . . . .	77
9.4 Risultati comparativi . . . . .	81
9.5 Epilogo . . . . .	82
<b>10 Conclusioni e sviluppi futuri</b>	83
<b>Riferimenti bibliografici</b>	85

# Elenco delle figure

3.1 A basic Artificial Neural Network . . . . .	21
3.2 Activation functions . . . . .	28
3.3 Architettura MLP - propagation . . . . .	29
3.4 Architettura MLP - backpropagation . . . . .	30
4.1 Esempio dell'approccio . . . . .	35
4.2 Modello proposto . . . . .	36
6.1 Le similarità . . . . .	46
6.2 I due approcci Word2Vec . . . . .	48
6.3 L'approccio CBOW . . . . .	50
6.4 Lookup table . . . . .	51
6.5 L'approccio Skip-gram . . . . .	53
6.6 Input pairs . . . . .	53
6.7 Output layer . . . . .	54
6.8 Output layer - Softmax . . . . .	55
6.9 Hierarchical Softmax . . . . .	57
7.1 Rete Neurale Ricorrente . . . . .	62
7.2 Esempio derivativa . . . . .	64
7.3 Schema - gates . . . . .	66
7.4 LSTM . . . . .	66
7.5 BLSTM . . . . .	69

# Abstract

Automatic Keyphrase Extraction is the task of extracting a set of significant and topical phrases from a text document which summarizes its contents. Extracting high quality and reliable keyphrases can benefit various natural language processing applications: text summarization, document categorization, text clustering, information retrieval, etc. The Automatic Keyphrase Extraction problem is tackled with different supervised and unsupervised techniques, strongly rely on domain specific knowledge and sophisticated features: statistical (e.g. number of appearances), positional (e.g. the position of the first occurrence), linguistic (e.g. part-of-speech), etc. However, due to the wide variety of lexical, linguistic and semantic characteristics, it is extremely hard to design hand-crafted features that can deal with all of these aspects. Recently, Deep Learning techniques, which do not require feature engineering, have shown an impressive result in Natural Language Processing. To Best of our knowledge, few attempts have been made to address keyphrase extraction task. Newly, through Deep Learning, Neural Networks architectures that contain a large number of hidden layers, has been successfully applied in many fields such as medical, economic and energy. New recent advancements in this research field are changing our way of living. Think of how life becomes increasingly easier with the progress of text translation functions, voice recognition systems, video and images recognition: all of these applications are now based on Deep Learning. The most interesting ability of a neural network is the capacity of learning from data. Nowadays, the age of the Big Data, Neural Networks have received a lot of attention, because they can effectively use massive amounts of data to extract high-level and complex abstractions as data representation through a hierarchical learning process.

In this thesis, we investigate the usage of Neural Network Architectures. In particular, the proposed solution, is based on a Bidirectional Long Short-Term Memory Recurrent Neural Network that can detect the main topics on the input documents without the need of defining new hand-crafted features. First, we extract a word embedding representation for each word, which maps it into a semantic vector representation. Then, we fed into a Bidirectional Recurrent Neural Network with LSTM units, which it can effectively deal with the variable lengths of sentences and it can analyze word features and their context (e.g. the distant relation between words). Finally, we perform some tests on datasets using different parameters and techniques. Experimental evaluations on well-known and public databases are performed. To test the importance of word embedding, we perform experiments with the pre trained Word2Vec and Stanford's GloVe Embeddings using different feature dimensionality. Finally, we compare the proposed solution against other recent competitive methods confirming that the proposed technique is effective and achieves the best performance without requiring hand-crafted features.

# Capitolo 1

## Introduzione

Le frasi chiave, o “*keyphrases*” dall’inglese, sono espressioni che “catturano gli argomenti principali discussi in un dato documento” [81]; composte tipicamente da un minimo di una ad un massimo di cinque parole, esse appaiono all’interno di un documento e possono essere utilizzate per riassumere brevemente il suo contenuto.

Gli algoritmi di estrazione automatica di frasi chiave permettono di individuare un insieme di frasi significative e descrittive a partire da un documento di testo, il quale genererà come risultato il suo contenuto riepilogato. Estrarre frasi chiave di buona qualità può facilitare il procedimento delle diverse applicazioni che sfruttano l’area del *Natural Language Processing*, come il *text summarization* [88], il *document categorization* [26], il *text clustering* [75], l’*information retrieval*, ecc. Processare e analizzare le informazioni che i diversi *social network* producono ogni giorno, ad esempio, può essere un compito utile in svariati ambiti, di natura economica o sociale, poiché si tratta di informazioni utilizzabili in diversi modi, come nella generazione di pubblicità direzionale o nella creazione di un’interazione personalizzata. Considerando che i dati disponibili attualmente risultano sovrabbondanti e privi di strutturazione, secondo [73], essere in possesso di un metodo per analizzare i dati significa avere l’equivalente moderno di un microscopio, con la differenza che: invece di mostrare ciò che prima appariva invisibili agli occhi, ora siamo in grado di raggruppare le informazioni al fine di dar loro un significato preciso e una chiara visualizzazione.

L’analisi dei dati viene eseguita in modo più scorrevole se essi vengono categorizzati, per esempio attraverso l’associazione del dato stesso a termini che indicano il suo principale contenuto. Gli articoli scientifici, a titolo di esempio, quando associati a frasi chiave, facilitano l’analisi del profilo della

conferenza annuale della rivista nella quale gli articoli sono stati pubblicati. L'associazione di frasi chiave permette inoltre la ricerca efficiente di articoli di interesse, indicando l'idea principale dell'articolo e permettendo così di allineare facilmente l'interesse del ricercatore con quanto proposto nel documento.

Risulta evidente come l'assegnazione manuale di frasi chiave a documenti di testo sia in alcuni casi impraticabile, considerando soprattutto il grande volume di dati disponibili attualmente. A causa di questa impossibilità, viene a crearsi una grande richiesta di approcci che siano capaci di estrarre automaticamente termini adeguati, ovvero frasi chiave significative che verranno poi associate ai rispettivi documenti. Eventuali problemi nel campo dell'estrazione automatica di frasi chiave vengono affrontati mediante tecniche differenti, basate sulle conoscenze specifiche del dominio o su altre caratteristiche [72]. Tuttavia, a causa dell'ampia varietà delle caratteristiche lessicali, linguistiche e semantiche, appare estremamente difficile progettare sistemi che siano idonei ad affrontare tutti questi aspetti, tenendo conto di tutte le possibili variabili appena menzionate. Per venire incontro alle richieste di maggior indipendenza del dominio e delle proprietà di carattere linguistico, le tecniche di *Deep learning*, che non richiedono la “*feature engineering*”, hanno ricevuto particolare attenzione.

Il *Deep learning* è un'area dell'intelligenza artificiale preposta all'elaborazione di grandi quantità di dati in modo efficace: con gli avanzamenti delle tecnologie *hardware*, infatti siamo oggi in grado di elaborare sistemi per l'analisi di testo, immagini e audio, capaci di restituirci dati molto più significativi rispetto al passato, soprattutto grazie alla possibilità, ora esistente, di poter trattare una maggior quantità di dati. In merito all'avanzamento tecnologico appena menzionato, l'ottenimento di dati meno soggettivi diventa un risultato possibile; i valori restituiti da sistemi basati sulle tecniche di *Deep learning* risultano più ricchi di aspetti che gli permettono di essere estremamente precisi. Tale accuratezza si lega prevalentemente al fatto che adesso è possibile racchiudere una maggiore quantità di caratteristiche in un singolo dato. Per raggiungere tale precisione, le tecniche sopramenzionate si basano sull'uso di un insieme di algoritmi per la creazione di modelli matematici, come ad esempio le reti neurali artificiali.

Le reti neurali artificiali sono in grado di imparare per mezzo di esempi, dunque in un modo simile al processo di apprendimento umano. Esse sono capaci di riconoscere eventuali somiglianze tra dati diversi, quando in possesso di campioni rappresentativi e distintivi di tali similitudini. Più precisamente,

una rete neurale è un processore che funziona in modo parallelo e distribuito, dotato di diverse unità di elaborazione che hanno la capacità di memorizzare determinate conoscenze e renderle disponibili per l'uso [25]. In modo analogo al cervello (da cui deriva la sua denominazione), la conoscenza della rete viene acquisita dal processo di apprendimento, dove l'intensità delle connessioni tra le varie unità di elaborazione (chiamate neuroni) sono utilizzate per salvare le conoscenze acquisite dalla rete. Solitamente le reti neurali artificiali sono modelli architetturali nei quali i passaggi per l'elaborazione dell'informazione sono suddivisi in uno o più livelli, tra i quali i più comuni sono quelli di input e output. Possono inoltre essere presenti altri livelli denominati "livelli nascosti": essi elaborano altre caratteristiche della rete prima di restituire una risposta effettiva al problema per cui la rete è stata impiegata [70].

Il principale aspetto differenziale delle tecniche di *Deep learning* è perciò centrato sull'uso di modelli matematici per i quali l'apprendimento è basato su molteplici livelli, ossia sull'uso di più di un livello nascosto. Secondo [45], tali livelli sono responsabili dell'elaborazione di caratteristiche specifiche non progettate da ingegneri umani, poiché impiegati nell'apprendimento di caratteristiche distinte, appartenenti al dominio di applicazione della rete. In questo modo, la struttura del modello perde una parte di specificità legata al dominio, ovvero all'ambiente di applicazione, ottenendo come conseguenza una maggiore naturalizzazione del processo. Il metodo di apprendimento di un modello basato su *Deep learning* diviene dunque insito nella capacità umana di individuare oggetti e caratteristiche spontaneamente. Infatti, l'estrazione di caratteristiche in un modello basato su *Deep learning* è implicita e il suo apprendimento diviene orientato alla comprensione da parte del computer. Ciò significa che, tale tecnica consente al computer di costruire concetti complessi per mezzo di altri più semplici, poiché la rappresentazione dei dati viene elaborata in diversi livelli di astrazione. Quindi, se lo scopo di un modello fosse, ad esempio, quello di riconoscere una macchina in un'immagine, un modello basato su *Deep learning* costruirebbe la rappresentazione di tale oggetto attraverso la combinazione di diversi livelli nascosti, dove ciascun livello rappresenterebbe un livello di astrazione dell'immagine: dapprima si riconoscerebbero i pixel, poi i contorni, successivamente le parti specifiche dell'oggetto ricercato, in questo caso di una macchina, ecc. [21].

Recentemente, le architetture di reti neurali artificiali che si basano su tali tecniche sono state applicate con esito positivo in diversi ambiti, quali ad esempio quello medico, economico ed accademico [84], [3], [29]. I recenti avanzamenti in questo campo di ricerca hanno profondamente modificato il nostro

modo di vivere: basti pensare a come la quotidianità sia diventata più semplice grazie anche ai progressi delle funzioni di traduzione automatica di testi, di riconoscimento vocale e visuale: la maggior parte delle applicazioni costruite per offrire tali funzionalità sono basate proprio su *Deep learning*. Al giorno d'oggi, nell'era del *Big Data*, le reti neurali hanno ricevuto diversi riconoscimenti, soprattutto perché sono effettivamente capaci di analizzare o utilizzare enormi quantità di dati per estrarre informazioni complesse e ad alto livello semantico. Un esempio di ciò è dato dalla rappresentazione di dati attraverso un processo di apprendimento gerarchico: difatti, l'abilità più interessante di una rete neurale è proprio la capacità di imparare attraverso i dati.

In questa tesi è stata investigata l'utilità delle reti neurali artificiali nell'ambito dell'estrazione di frasi chiave, poiché risultano scarsi in letteratura i risultati raggiunti in merito a tale problematica. In particolare, la soluzione qui proposta, diversamente della maggior parte degli approcci presentati, è basata sulla *Bidirectional Long Short-term memory Recurrent Neural Network* (BLSTM RNN) [71]: essa può rilevare i principali argomenti presenti nei documenti dati come input senza la necessità di definire modelli specifici del dominio di interesse, oltre a tener conto di diversi altri aspetti rilevanti, per quanto concerne l'ambito di ricerca proposto in questa tesi, come il contesto in cui si trova una parola o un termine cospicuo in un documento.

In particolare un intero testo è fornito come input ad un modello, il quale sarà capace di impiegarlo allo scopo di estrarre parole e frasi chiave che descrivano correttamente il suo contenuto. Riuscire a processare un intero documento, tenendo conto dell'informazione sia passata che futura, appare molto importante per poter racchiudere il senso fondamentale di un discorso. Possedere un'informazione contestuale è infatti rilevante quando il proposito della ricerca riguarda l'estrazione di un contenuto a partire da un qualsiasi testo. Per questo motivo si è scelto di far uso delle reti neurali ricorrenti BLSTM, che effettivamente possono memorizzare le informazioni contestuali.

Una rete neurale ricorrente costituisce un tipo di rete neurale artificiale che elabora le informazioni in modo sequenziale, prendendo pertanto come input le previsioni passate, quelle cioè, fatte in un periodo di tempo precedente, combinandole con l'input attuale. Si dice perciò che il processo di apprendimento delle suddette reti venga fatto per mezzo delle informazioni imparate precedentemente [32], [57]. Tuttavia, la principale problematica nell'impiego delle reti neurali ricorrenti sta nel fatto che esse non sono in grado di imparare una lunga dipendenza tra valori attuali e precedenti. Per affrontare tale limi-

---

tazione, Hochreiter e Schmidhuber [32] hanno introdotto le *Long Short-term memory Recurrent Neural Network* (LSTM RNN), reti progettate per gestire lunghi tempi di dipendenza. Con il passare degli anni, l'LSTM è stata perfezionata e altre versioni – più semplici o soffisicate – della medesima sono state proposte, come la *Gated Recurrent Unit* (GRU) [34] e la sopramenzionata BLSTM. Reti di questo tipo sono capaci di lavorare con input di lunghezza variabile, come ad esempio le frasi di un testo. Per questo motivo, vengono spesso utilizzate nella traduzione automatica di un testo, insieme a numerose altre funzioni nell'ambito del *Natural language processing*.

Oltre all'uso delle BLSTM, questa tesi esplora una tecnica chiamata “*word embeddings*”, che ci consente di rappresentare le parole attraverso spazi vettoriali. Tale tipo di rappresentazione consiste nel catturare il significato di una parola e trasformarlo in un vettore di caratteristiche, ovvero in una rappresentazione distribuita. In questo modo risulta possibile calcolare, ad esempio, la similarità tra due parole attraverso delle operazioni relativamente semplici, come la somma o la sottrazione (cfr. capitolo 6).

Nell'ambito del *Natural Language Processing*, esiste un acceso dibattito su quali siano i metodi migliori e più moderni per l'elaborazione e la predisposizione dei dati. Possedere un metodo plausibile per la rappresentazione delle parole di un testo è considerato una buona pratica per chi intende svolgere una ricerca in quest'area specifica. Difatti, secondo [43] “una rappresentazione che cattura con accuratezza come le parole vengono usate nel loro contesto naturale catturerà molto meglio il loro significato”. Uno degli obiettivi di questa tesi è appunto quello di comprendere se le prestazioni della rete neurale miglioreranno mediante l'utilizzo delle rappresentazioni che comprendono meglio il significato della parola, ovvero se la performance del sistema trarrà giovamento dall'uso degli “*word embeddings*”. La base di tale tecnica si fonda sull'ipotesi distribuzionale di Harris [28], la quale afferma che “la differenza di significato è correlata alla differenza di distribuzione”. Ciò che il *word embedding* propone, infatti, è nient'altro che la rappresentazione delle parole attraverso vettori distribuiti, dove ogni concetto viene rappresentato per più di una caratteristica.

Nello specifico, la tesi descrive nel dettaglio lo studio presentato in [4], includendo per di più alcuni miglioramenti e riportando ulteriori risultati. Fondamentalmente, i passaggi eseguiti al fine di poter concretizzare il modello sono stati i seguenti: dapprima sono state create le rappresentazioni delle parole attraverso gli “*word embedding*”; esse sono state poi date in input ad una *Bidirectional Recurrent Neural Network*, in grado di processare lun-

ghi documenti di testo e analizzare le caratteristiche di ogni parola nel suo contesto di riferimento (ad es. il rapporto tra le parole); infine sono stati eseguiti alcuni esperimenti utilizzando svariate configurazioni della rete neurale proposta, oltre all'uso di diversi dataset per la valutazione dei risultati. Per testare l'importanza degli “*word embedding*”, sono stati inoltre compiuti alcuni esperimenti con i dati pre-addestrati “*Word2Vec pre-trained vectors*” [58] e *Stanford's GloVe Embeddings*” [63], usando diverse configurazioni della loro caratteristica dimensionale. La soluzione proposta è stata confrontata con altri metodi conosciuti, confermando come la tecnica proposta sia efficace e capace di raggiungere le migliori performance trovate sullo stato dell'arte (fino all'inizio della stesura della tesi).

Nel secondo capitolo saranno esposti gli obiettivi della tesi, un breve quadro teorico sui macro argomenti e sullo stato dell'arte. Nel terzo capitolo saranno poi introdotte le reti neurali, i modelli didattici *Perceptron* e *Perceptron a Multi livelli*, le principali caratteristiche dell'impostazione di una rete neurale, ovvero la configurazione di parametri (ad es. numero di livelli, neuroni, ecc.), dati di training, dati di test, di valutazione, ecc. Nel quarto capitolo sarà presentato l'approccio proposto per il raggiungimento degli obiettivi introdotti dal secondo capitolo, ovvero l'architettura proposta, l'impostazione del problema e le metodologie utilizzate. Nel quinto capitolo saranno riportate le tecniche usate per il *pre-processing* dei dati, quali siano stati i principali problemi riscontrati e le problematiche intercorse nell'uso di differenti dataset. Nel sesto capitolo sarà riportata la ricerca realizzata su “*word embeddings*”: l'obiettivo, gli approcci e alcuni esempi d'uso. Il settimo capitolo tratterà le *Bidirectional Long short-term memory Recurrent Neural Network*, introducendo la rete neurale ricorrente seguita dalle motivazioni per la scelta della rete *Long short-term memory*, la sua descrizione e l'approccio bidirezionale. Nell'ottavo e nono capitolo saranno infine riportati gli strumenti utilizzati e i diversi risultati sperimentalmente ottenuti attraverso l'impostazione di differenti parametri della rete. Nella parte finale verranno poi inserite le conclusioni e i possibili sviluppi futuri.

# Capitolo 2

## Obiettivi, fondamenti e stato dell'arte

In questo capitolo saranno descritti gli obiettivi della tesi e come si intende raggiungerli. Saranno elencati gli argomenti che necessari da conoscere affinché sia possibile avere una maggiore cognizione del testo; per una miglior comprensione del contenuto dei prossimi capitoli, verrà infine presentato un breve quadro teorico e lo stato dell'arte degli ultimi anni.

### 2.1 Obiettivi

Al fine di raggiungere la performance dei sistemi presentati in letteratura, questa tesi propone l'uso di un modello matematico architettonale chiamato rete neurale di tipo ricorrente. In particolare, per tale obiettivo si farà ricorso ad una *Bidirectional Long Short-Term Memory Recurrent Neural Network* (BLSTM RNN).

Partendo dallo studio dello stato dell'arte sui metodi che permettono l'estrazione della conoscenza di documenti e dall'approfondimento del modello “*Word2Vec*”, è nata l'idea di strutturare un'architettura di rete neurale che seguisse uno dei modelli proposti in [42]. Il modello appena riferito si basa sull'utilizzo di una BLSTM per compiere la funzione *Named Entity Recognition* in un testo.

La maggior parte dei sistemi attuali basati su reti neurali impiegano la rappresentazione testuale “*word embedding*”, poichè essa ha mostrato risultati promettenti in altri ambiti del *Natural Language Processing* (a titolo di esem-

pio [42], [92] e [55]).

L'intento principale della tesi può essere riassunto in quattro punti:

- Approfondire, usare e testare la rappresentazione testuale “*word embedding*”;
- Progettare una struttura architetturale semplice basata sulle reti neurali ricorrenti;
- Analizzare i vantaggi e le limitazioni dell'uso di nuovi strumenti per la creazione di reti neurali;
- Testare il modello utilizzando differenti impostazioni su diversi dataset, e confrontare i risultati ottenuti con lo stato dell'arte.

## 2.2 Natural Language Processing

Il *Natural Language Processing* (NLP) studia le tecniche computazionali per risolvere problemi legati al linguaggio umano, scritto o parlato. Programmi e/o procedure per il riconoscimento vocale, come il *Google Cloud Speech*, rappresentano esempi di processing della lingua parlata. Tra le diverse problematiche ancora irrisolte del NLP, sono presenti la comprensione e l'estrazione del significato del linguaggio. Innumerevoli sono oggi le applicazioni del NLP, partendo dall'estrazione di informazioni e dalla correzione automatica alla traduzione simultanea e al riconoscimento vocale. Molte di queste applicazioni si basano solitamente sulle tecniche di *Machine Learning*: metodologia che consente lo sviluppo di modelli analitici senza l'intervento umano attraverso l'analisi dei dati. Nel momento in cui i modelli basati su tali metodologie entrano in contatto con nuovi dati, sono in grado di imparare da essi, adattandosi in modo che i risultati e le decisioni ottenuti possano essere riprodotti con fedeltà.

Le tecniche di *information extraction* sono, ad esempio, applicazioni del NLP molto studiate nell'ambito della ricerca informatica. In quest'area vengono esplorate le problematiche di estrazione automatica di parole e frasi chiave, di riconoscimento di entità (locali, organizzazioni o persone) ecc., le quali forniscono un potente mezzo per il filtraggio di contenuti e l'organizzazione testuale. Numerosi sono gli utilizzi di frasi chiave, termini raccolti da un insieme di dati: suddivisione di documenti d'interesse per l'ambito accademico, scelta di dati rilevanti per la medicina basata sull'evidenza, categorizzazione di dati, analisi di marketing ecc.

## 2.3 Machine Learning

Contrariamente a quanto si potrebbe pensare, il termine *Machine Learning* (ML) esiste già da tanti anni; fu Arthur Samuel, un ricercatore famoso per il suo “*checker-playing program*”, il pioniere di molte tecniche di *machine learning* e *game-playing* [69]. Negli ultimi anni l’interesse per l’ML è aumentato esponenzialmente.

I fattori che hanno contribuito a tale incremento sono stati gli stessi che hanno reso popolari l’analisi di *data mining* e bayesiana: la crescita della varietà e del volume di dati disponibili, l’elaborazione computazionale più potente e poco costosa, l’archiviazione dei dati resa più accessibile, ecc. Tali progressi tecnologici hanno consentito un ulteriore sviluppo del ML e la creazione di nuove tecnologie sul campo, grazie anche al fatto che ora risulta possibile produrre rapidamente ed automaticamente modelli in grado di analizzare dati di dimensioni maggiori e molto più complessi, fornendo comunque dei risultati immediati e accurati. Essi sono delle previsioni in grado di condurre a decisioni e azioni intelligenti più efficaci, senza la necessità di un intervento umano.

Un modo per poter creare azioni intelligenti in tempo reale è rappresentato dallo sviluppo di modelli automatizzati. Con l’enorme velocità di crescita e l’evoluzione del volume di dati, risultano necessari flussi di modellizzazione di rapida evoluzione che siano in grado accompagnarli, ottenibili attraverso le tecniche di ML.

L’apprendimento automatico è particolarmente presente nei giorni nostri più di quanto si possa pensare. Permea la tecnologia che ci circonda, come ad esempio i consigli per gli annunci online in tempo reale come, *Amazon* o contenuti come *Netflix*, il filtro antispam nelle caselle e-mail, il rilevamento di frodi, il riconoscimento di immagini e modelli, i risultati di ricerca web (*Google*), le previsioni di guasti alle apparecchiature, i nuovi modelli di prezzo, le analisi ed il rilevamento di sentimenti nei testi, il punteggio di credito, i sondaggi di opinione sui *social network* e persino le auto che si guidano autonomamente sviluppate da *Google*.

Esistono diversi metodi per l’apprendimento nel ML: sebbene la maggior parte prenda il nome di apprendimento supervisionato, esistono però anche altre tipologie, come l’apprendimento non supervisionato, semi-supervisionato e per rinforzo. Di seguito, sarà riportata una spiegazione più dettagliata per ciascuno di essi:

- Apprendimento supervisionato: gli algoritmi utilizzati in tale contesto vengono addestrati usando esempi etichettati, per cui ad ogni input viene associato un output noto. Per esempio: una serie di apparecchi televisivi può avere tra i suoi dati informativi un’etichetta, nella quale i dispositivi difettosi sono etichettati con una ‘‘F’’ che rappresenta “fallimento” mentre i dispositivi in buone condizioni di funzionamento sono etichettati con una ‘‘B’’. Dunque, l’algoritmo impara confrontando l’output effettivo con l’output desiderato, per rilevare eventuali errori e successivamente modificare il modello in base alla dimensione dell’errore. Questo metodo ML è ampiamente utilizzato in applicazioni nelle quali esistono dati storici che rendono possibile la previsione di probabili eventi futuri. Può essere usato ad esempio per prevedere la probabilità e, di conseguenza, rilevare quali transazioni con carte di credito possano risultare fraudolente o quale cliente di una determinata società possa presentare un reclamo.
- Apprendimento non supervisionato: questo metodo ML viene utilizzato per i dati privi di etichette. In questi casi, il sistema non sa quale sia la “risposta giusta” e spetta quindi all’algoritmo identificare una struttura dei dati posta in input. Lo scopo dell’apprendimento non supervisionato è dunque quello di esplorare i dati e rilevare alcune strutture presenti in essi. Un esempio d’uso di questa tipologia di apprendimento è dato dalle operazioni di marketing, dove si identificano i clienti con caratteristiche comuni e che possono quindi essere trattati in modo simile, oppure nel rilevamento delle caratteristiche principali che dividono i segmenti di clienti. Tale metodologia viene utilizzata anche nella raccomandazione di articoli, nella suddivisione di un testo in argomenti principali e nell’identificazione di dati divergenti.
- Apprendimento semi-supervisionato: questo metodo viene applicato negli stessi casi dell’apprendimento supervisionato, con la differenza che qui vengono utilizzati dati etichettati e non contrassegnati nella sua formazione. Più precisamente, viene impiegata una piccola quantità di dati etichettati e una grande quantità di dati non etichettati poiché questi ultimi richiedono meno sforzi per esser ottenuti, risultando di conseguenza più economici. Questo tipo di ML viene usato principalmente in metodi quali la regressione, la previsione e classificazione. L’apprendimento semi-supervisionato si rivela molto utile quando il processo di formazione completamente etichettato viene reso impossibile a causa di un costo elevato associato all’etichettatura. Un esempio di impiego di tale metodo

di apprendimento è quello del riconoscimento facciale, il quale identifica o verifica l'identità di una persona.

- Apprendimento per rinforzo: esso è un metodo di apprendimento automatico ampiamente utilizzato nei giochi, nella robotica e nella navigazione. Usando l'apprendimento per rinforzo, l'algoritmo impara per tentativi ed errori quali azioni generano maggiori ricompense. Tale apprendimento consta di tre principali agenti: l'agente stesso, colui che sta imparando e prenderà poi le decisioni; l'ambiente, ovvero tutto ciò con cui l'agente interagisce; e le azioni, che l'agente può svolgere. Lo scopo di questo metodo è far sì che l'agente alla fine decida di eseguire le azioni che massimizzeranno la ricompensa prevista in un determinato periodo di tempo. Questo obiettivo verrà raggiunto molto più rapidamente dall'agente se esso persegua una buona politica; si può dunque concludere che lo scopo di tale metodo ML sia quello di imparare la migliore politica possibile.

In breve, l'ML viene impiegato per replicare automaticamente modelli e conoscenze note, che verranno poi applicate ad altri dati ed infine utilizzate, fornendo risultati rilevanti ed utilizzandoli infine per prendere decisioni e compiere azioni. Grazie all'aumento della potenza dei computer, molte tecniche, che prima risultavano di difficile applicazione, si sono ora evolute, divenendo combinabili con il *Machine Learning*. Un esempio di tale processo di evoluzione è dato dal fatto che, grazie a questa maggiore potenza di calcolo, risulta oggi possibile ottimizzare il tempo e l'utilizzo delle reti neurali a più livelli (cfr. cap [3.1](#)).

Le reti neurali artificiali sono sistemi computazionali paralleli costituiti da unità di elaborazione semplice (denominate neuroni); esse sono collegate tra loro in un modo specifico nell'intento di realizzare un determinato compito. I neuroni artificiali sono fondamentalmente modelli matematici che elaborano le informazioni ricevute e ponderate da pesi sinaptici, fornendo una risposta al problema proposto [\[51\]](#). Solitamente una rete neurale artificiale ha due caratteristiche elementari, che devono essere progettate nel dettaglio: la sua architettura e l'algoritmo di apprendimento. A differenza dei soliti computer pre-programmati, una rete neurale viene addestrata per mezzo di esempi che vengono forniti come input. Posto che la conoscenza del problema risiede nel dato stesso, la principale funzione dell'algoritmo di apprendimento è quella di generalizzare tali dati, memorizzando la conoscenza acquisita attraverso l'aggiustamento di alcuni parametri della rete, i pesi. La natura del problema definisce come la rete dovrà essere modellata e quali saranno gli algoritmi di

apprendimento più adeguati [66]. Normalmente i passaggi per l’elaborazione dell’informazione nelle reti neurali sono suddivisi tra uno o più livelli, tra i quali i più comuni sono quelli di input e output. Possono inoltre essere presenti altri livelli intermedi, chiamati “livelli nascosti”: le reti dotate di più livelli nascosti sono denominate reti neurali profonde; i modelli matematici basati su tale caratteristica si possono definire appartenenti al “*Deep learning*” (tecnica che verrà approfondita in seguito).

È proprio l’aumento della capacità dei computer che consente ora una rapida elaborazione di più livelli di reti neurali per l’apprendimento automatico. L’ML tradizionale si basa dunque su reti poco profonde, composte solitamente da un livello di input e uno di output, e contenenti al massimo uno o due livelli nascosti. Quando si configura una rete che fa uso di più di tre livelli intermedi tra il livello di input e quello output, allora essa si qualifica come “apprendimento profondo”, o *Deep learning* dall’inglese.

### 2.3.1 Deep Learning

Il cosiddetto *Deep learning* fa uso del miglioramento della potenza computazionale, abbinato ai diversi tipi di modelli matematici, per elaborare dati complessi e in grande quantità. I risultati restituiti da sistemi basati su tale tecnica risultano più ricchi di aspetti e di conseguenza più precisi. Tale accuratezza è raggiungibile poiché attualmente è possibile racchiudere una maggiore quantità di caratteristiche in un singolo dato. Il *Deep learning* ha riscosso molto successo nel riconoscimento di immagini, nel riconoscimento vocale e in quello testuale. I ricercatori dell’ambito dell’intelligenza artificiale si stanno oggi dedicando a un’altra sfida: riuscire a replicare tale successo nel riconoscimento di modelli in situazioni più complesse e delicate come ad esempio le diagnosi mediche, le traduzioni automatiche di lingue e molte altre attività quotidiane e professionali. Per quanto riguarda lo scopo di questa tesi, il *Deep learning* sarà usato nell’ambito dell’estrazione di frasi chiave provenienti da documenti di testo.

## 2.4 Estrazione di parole e frasi chiave

Come accennato precedentemente, l’estrazione di informazioni è una pratica largamente analizzata in informatico. Per quanto concerne la ricerca qui proposta, saranno trattati in particolar modo gli argomenti inerenti all’estrazione di parole chiave (“*keywords*” dall’inglese) e frasi chiave (“*keyphrases*” dall’inglese) di testi scientifici. I termini “*keyphrases*” e frasi chiave saranno

considerati interscambiabili nel proseguo dell'elaborato.

L'estrazione di parole e frasi chiave viene applicata in diversi *task*, come ad esempio:

- Nell'organizzazione in base al contenuto (terminologie, ontologie, ecc.), molto importante nel settore medico;
- Nell'analisi di grandi quantità di dati;
- Nella ricerca di informazioni relative all'interesse dell'utente, utile soprattutto nel settore del *marketing* e della pubblicità.

Soltanamente una parola descrive un oggetto, un soggetto, un aspetto, ecc. Sia parole che frasi possono essere considerate *termini*. La principale differenza tra parole e frasi chiave è data dalla specificità: l'uso di parole singole può generare ambiguità, visto che molte volte queste risultano essere troppo generiche e quindi non sufficientemente specifiche; l'uso di frasi chiave può ridurre l'ambiguità ad esempio tramite l'utilizzo di termini composti, come “*hard disk*” piuttosto che il solo “*hard*” o “*disk*”, il che chiaramente muta il significato di ciò che vogliamo esprimere. L'uso di entrambi termini combinati è dunque necessario, non escludendo l'uno o l'altro.

Secondo [72], possono essere diversi gli approcci per l'estrazione di parole e frasi chiave; i principali presenti in letteratura sono quattro:

1. l'approccio basato su regole linguistiche;
2. l'approccio statistico;
3. l'approccio basato su tecniche di *Machine Learning*;
4. l'approccio basato sulle specificità del dominio.

Il primo approccio richiede la definizione manuale di regole e caratteristiche linguistiche, ovvero l'inserimento di informazioni inerenti al linguaggio preso in analisi. Grazie a tali connotazioni, l'approccio risulta più preciso ma meno veloce, poiché esso richiede più tempo per l'analisi lessicale, sintattica e l'analisi del discorso. Il secondo approccio fa spesso uso di un corpus<sup>1</sup> linguistico e

---

<sup>1</sup> Un corpus per training è una collezione di testi contenenti informazioni linguistiche valutate manualmente. Tali informazioni sono spesso usate nel *Machine learning* per la creazione di modelli statistici.

di caratteristiche statistiche che derivano dal medesimo. Tali peculiarità vengono raffinate nel processo di analisi dei documenti e di addestramento della rete. Solitamente tale approccio viene impiegato nei metodi supervisionati, poiché esso segue un *pattern* specifico, informato a priori. Alcuni esempi sono il “tf\*idf”, la “distance” e la “supervised keyphraseness” (per ulteriori approfondimenti consultare [2]). Questo metodo può risultare meno preciso, ma solitamente più veloce. Il terzo approccio fa ricorso alle tecniche di *Machine Learning* per estrarre frasi chiave attraverso l’apprendimento delle caratteristiche specifiche di ciascun documento fornito come input. Il modello dei dati viene creato in base agli esempi forniti e poi utilizzato per ricercare nuove frasi chiave in altri documenti. Il quarto e ultimo metodo si basa, infine, sulla conoscenza specifica del dominio di applicazione e su informazioni esterne, impiegate per ereditare una struttura che verrà poi usata per migliorare il modello (ad es. ontologia, thesaurus ecc.).

Per quanto riguarda invece la generazione automatica di parole o frasi chiave, esistono due metodi principali:

- Assegnazione di parole e frasi chiave: l’insieme delle possibili parole e frasi chiave risulta delimitato da un vocabolario predefinito di parole, spesso difficile/costoso da elaborare. Solitamente si trova un piccolo insieme di termini in grado di descrivere un singolo documento.
- Estrazione di parole e frasi chiave: le parole più significative presenti nel documento sono selezionate, ma tale procedura non dipende da un vocabolario definito a priori. Le parole estratte sono presenti nel documento stesso.

Gli studi in quest’area sono genericamente suddivisi tra metodi per l’estrazione di parole e frasi chiave in modo statistico, supervisionato, non supervisionato o semi-supervisionato. Nella sezione 3.1 saranno approfondite le informazioni riguardanti il modello matematico denominato Rete Neurale Artificiale – spesso impiegato per la classificazione di dati in modo supervisionato o non-supervisionato.

## 2.5 Related works

In questa sezione saranno introdotti progetti in comune con ciò che viene proposto in questa tesi: sarà dunque presentata una breve revisione dello stato dell’arte degli ultimi anni nell’ambito dell’estrazione di frasi chiave. In riferimento a quanto descritto nei paragrafi precedenti, saranno approfonditi

alcuni progetti incentrati sugli approcci descritti in [72] per l'estrazione e la generazione automatica di frasi chiave.

In [52], a titolo di esempio, è stato ideato un approccio per l'estrazione di frasi chiave basato sull'utilizzo di caratteristiche particolari appartenenti alla specificità del dominio di applicazione. Tutto ciò viene attuato per mezzo di un grafo fondato sul rapporto semantico tra termini e sull'analisi dei loro aspetti quantitativi, oltre alla valutazione della coincidenza (o *co-occurrence* dall'inglese) e dell'arricchimento di tale rappresentazione con informazioni estratte dal *Wordnet*<sup>2</sup>. Secondo gli autori, l'uso di *Wordnet* non comporta grandi miglioramenti: in effetti, i termini tecnici estranei a questo database lessicale non vengono estratti. L'utilità di tale conoscenza esterna si è mostrata dunque fortemente dipendente dal contesto d'uso, ovvero dal suo dominio di applicazione.

Un altro esempio di progetto che mira a catturare relazioni semantiche tra parole è quello descritto in [86], dove gli autori propongono l'estrazione di frasi chiave mediante un approccio denominato “*pattern mining*”. Il *pattern* per le frasi chiave viene scoperto specificamente per ogni singolo documento attraverso il loro algoritmo, il quale compie un'analisi del contesto per mezzo di *wildcards (gap constraints)*. Tale algoritmo fa inoltre uso di metodi supervisionati per l'apprendimento e per la costruzione di un modello per l'estrazione di frasi chiave.

In [92] l'estrazione di parole e frasi chiave viene compiuta attraverso l'impiego di una rete neurale ricorrente. Per questo progetto, è stato utilizzato un dataset di dati estratti dal *social network Twitter*: il modello proposto è costituito da due livelli nei quali il primo distingue le parole chiave ed il secondo le frasi chiave. Tra i punti di maggior rilievo riportati in tale progetto viene evidenziata la combinazione tra gli obiettivi “*keywork ranking*” e “*keyphrase generation*”. Viene inoltre dimostrato come la dimensione del vocabolario per l'uso di “*word embeddings*” (cfr. cap. [6]) debba essere proporzionale al dataset utilizzato, pena l'invalidità dell'estrazione.

Un altro studio recente basato sulle reti neurali artificiali per la predizione di frasi chiave è quello proposto dal [55]: il modello presentato si basa sull'uso di un framework “*encoder-decoder*”, capace di generare frasi chiave presenti nei testi, ma anche parole e frasi chiave assenti, chiamati da loro “*absent keyphrases*”.

---

<sup>2</sup>Wordnet: “*WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.*”

*ses*”. L’uso di una rete neurale ricorrente, insieme all’*encoder-decoder* basato sulla posizione dell’informazione, permette di ritrovare parti importanti del documento preso in analisi. È stato inoltre eseguito un confronto con altri sei modelli *baseline* impiegando cinque dataset diversi. Tra i dataset usati, viene evidenziato il dataset Kp20k: insieme di dati che verrà utilizzato in questa tesi per compiere alcuni esperimenti[27], [54].

Diversi altri ricercatori hanno fatto uso delle tecniche “*n-gram*” e “*noun phrase*” per l’acquisto e l’identificazione di potenziali frasi chiave candidate. Esse sono capaci di individuare frasi che corrispondono ad un certo pattern di “*Part Of Speech*” ([33], [44], [50]). Fondamentalmente, per *n-gram* in un testo si intende l’insieme di parole che compaiono una a fianco all’altra dato un determinato parametro quantitativo: se  $n= 2$ , ad esempio, allora l’insieme di n-gram (chiamato bi-gram in questo caso) in un dato testo sarà composto dai termini che contengono sempre due parole adiacenti. Per quanto riguarda il “*noun phrase*”, invece, si tratta dell’identificazione di frasi che impiegano lo stesso ruolo dei sostantivi in un testo, ovvero termini composti che includono un nome (entità, persona, luogo) o pronome e una o più parole che li modificano, li separano, ecc., come ad esempio “*ingegneria del software*”.

Un’altra importante ricerca che fa uso delle tecniche supervisionate per l’estrazione di frasi chiave è stata compiuta da [5]. In questo studio è stato realizzato un confronto tra diversi algoritmi, verificando il suo range di miglioramento attraverso l’utilizzo del contesto della parola o frase chiave in un documento. Supponendo di avere la seguente frase: “*John Doe è un ingegnere informatico. Lui è molto bravo*”, nel momento in cui l’entità *John Doe* viene trovata, non si sa ancora se essa risulterà un elemento importante nel testo. Proseguendo nella lettura, però, si incontra il termine “*ingegnere informatico*” e il pronome “*Lui*”, che chiaramente sottolineano la sua importanza nel contesto. “*Lui*” e “*ingegnere informatico*” sono termini che vengono chiamati anafore e la tecnica per trovare tale informazione contestuale è chiamata *anaphora resolution*.

Altra proposta per l'estrazione di frasi chiave è quella descritta in [41], in cui gli autori affermano l'esistenza di una grande predisposizione all'identificazione di frasi chiave candidate con una *n-gram* relativamente bassa. Essi hanno dunque proposto una tecnica non supervisionata, basata su grafi per il filtraggio di *n-gram*. Tale tecnica fa uso di una strategia di espansione semantica attraverso il *Dbpedia*<sup>3</sup>.

In [4], invece, gli autori sfruttano l'estrazione di parole e frasi chiave come metodo per facilitare la creazione di una nuova ontologia per il dominio legale e semi-legale. Esso ha come obiettivo popolare la creazione di una tassonomia per l'ambito legale, come quello delle *privacy policies*.

Inoltre, per quanto riguarda l'estrazione di frasi chiave per mezzo di approcci supervisionati, [78] descrive un esperimento in cui l'uso di dataset contrassegnati da un unico annotatore (persona che segna quali sono le frasi chiave in un testo) può influenzare negativamente la valutazione delle frasi chiave estratte in modo automatico. I ricercatori di tale esperimento hanno quindi proposto l'assegnazione di etichette da molteplici annotatori e la riponderazione delle stesse, assegnate da singoli annotatori.

Altre ricerche si sono concentrate poi nell'estrazione automatica di contenuti, in particolare quella di frasi chiave, ricorrendo ad impostazioni multilingue, come in [20] e [7].

Infine, studi di terze parti possono essere rintracciati analizzando i risultati di SemEval (*Semantic Evaluation*), una competizione annuale alla quale partecipano vari gruppi di ricerca; alla fine di tale gara, viene fornito un sommario storico su diversi compiti, alcuni dei quali collegati, ad esempio, alla estrazione di frasi chiave. Il più recente è *SemEval-2017 Task 10*, [2]

---

<sup>3</sup>Dbpedia: progetto nato nel 2007 il cui obiettivo è l'estrazione di informazioni strutturate da *Wikipedia* e la loro pubblicazione in formato RDF (*Resource Description Framework*)

## 2.6 Epilogo

In questo capitolo sono stati introdotti i principali obiettivi della tesi, l'ambito e lo stato dell'arte dell'argomento trattato. Tale elaborato consiste fondamentalmente in un progetto sperimentale in cui viene proposta una soluzione per il problema dell'estrazione automatica di frasi chiave all'interno di documenti di testo di varia natura. Alla fine del capitolo il lettore avrà appreso i principali argomenti concernenti l'area del *Natural Language Processing*, *Machine learning*, *Deep learning* ed i metodi più utilizzati per l'estrazione automatica di parole e frasi chiave esistenti in letteratura.

# Capitolo 3

## Reti neurali

In questo capitolo saranno descritte le reti neurali artificiali ed elencate le principali tipologie esistenti. Saranno inoltre raffigurati più dettagliatamente i modelli Perceptron e Perceptron a molteplici livelli in quanto semplici e di facile comprensione.

### 3.1 Le reti neurali artificiali

Le reti neurali artificiali (RNA) sono modelli matematici ispirati dal sistema cerebrale biologico introdotti nel 1943 da Warren McCulloch e Walter Pitts [53]: il cervello umano possiede una grande quantità di neuroni; questi eseguono il *processing* dell'informazione in maniera parallela e distribuita, permettendoci di eseguire compiti complessi in modo efficiente, come la capacità di comprendere un discorso, la percezione visiva, ecc. Un neurone biologico è composto da un corpo cellulare (chiamato soma), dalle ramificazioni che ricevono le informazioni in entrata (dendrite) e da una prolunga che trasmette i segni del corpo cellulare (assone). Le estremità di un neurone sono collegate con le ramificazioni (dendrite) di altri neuroni attraverso il bottone sinaptico (*synapse*), formando così grandi reti. Il bottone sinaptico è una regione di comunicazione tra neuroni dove l'impulso nervoso passa da un neurone all'altro per mezzo di neurotrasmettitori.

Il modello iniziale di McCulloch e Pitts è composto da un insieme di  $n$  valori di input: essi vengono moltiplicati per un determinato peso  $e$ , in seguito, i risultati ottenuti vengono sommati e comparati con una soglia di confronto.

Attraverso algoritmi di apprendimento, una rete neurale artificiale simula una rete neurale biologica: si acquisisce e si utilizza la conoscenza estratta

dall'esperienza. In una RNA l'elemento base che compie il processing delle informazioni viene chiamato neurone artificiale; la computazione di una RNA viene perciò eseguita mediante una rete di neuroni connessi tra di loro. La conoscenza è poi codificata per mezzo dei pesi delle connessioni presenti tra i neuroni; gli algoritmi di apprendimento hanno dunque l'incarico di aggiornare i valori dei pesi d'accordo con un determinato problema. Tali pesi sono computati per mezzo di una funzione matematica che determina l'attivazione del neurone, la quale verrà poi tradotta come output della rete (cfr. fig. 3.1).

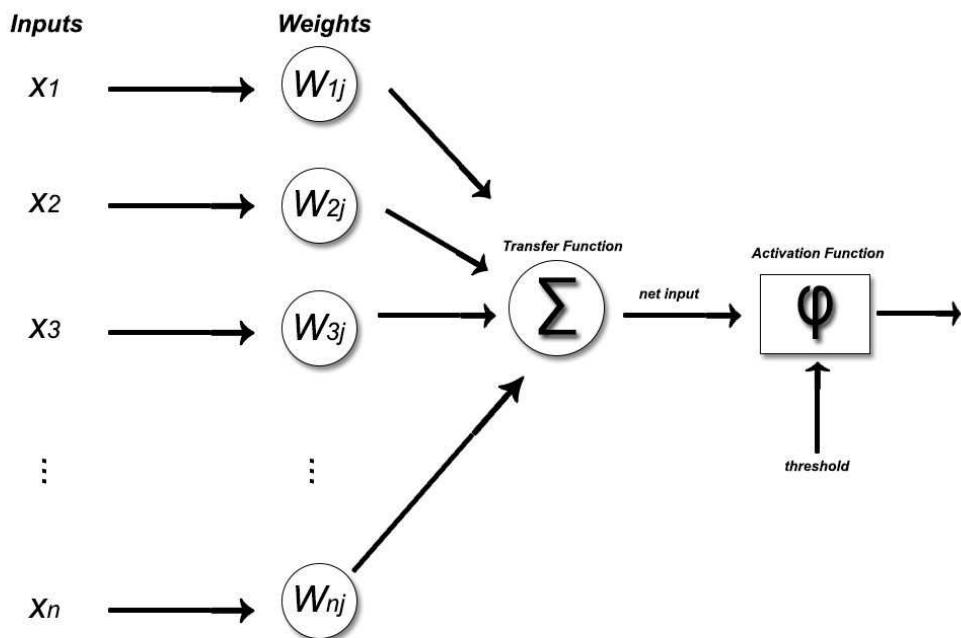


Figura 3.1: A basic Artificial Neural Network

### 3.1.1 Paradigmi di apprendimento

La capacità di “imparare” associata ad una RNA è una delle qualità più importanti di queste strutture. Le reti neurali sono in grado di adattarsi, in base a regole predefinite e al loro ambiente, modificando le loro prestazioni nel tempo. Per tali ragioni, viene denominato “apprendere” il processo che adatta il comportamento della rete portando da un miglioramento delle pre-

stazioni. Nel contesto delle RNA, l'apprendimento o il *training* corrisponde al processo di adeguamento dei parametri della rete attraverso un meccanismo di presentazione degli stimoli ambientali, noti come input. L'obiettivo dell'apprendimento di una RNA è quello di ottenere un modello implicito del sistema in esame, aggiornando i parametri della rete.

In particolare i parametri vengono adattati durante il processo di training in una RNA semplice e tradizionale, corrispondono unicamente ai pesi della rete (i pesi sinaptici). Esistono però alcuni parametri che non vengono adeguati durante il training della rete; si tratta di elementi predefiniti, come ad esempio l'intera struttura di rete, ovvero i tipi di neuroni, la loro quantità, le funzioni di attivazione, ecc.

Data una RNA, sia  $w_t$  il peso associato ad un determinato neurone al tempo  $t$ ; allora  $(t+1) = w_t + \delta w_t$ , dove  $\delta$  rappresenta un aggiustamento che verrà applicato al peso  $w_t$  nel tempo  $t$ , generando come conseguenza un nuovo valore per  $w$ . Per poter calcolare il valore di  $\delta$ , esistono diversi modi, poiché il tipo di apprendimento che determina la tecnica di aggiustamento dei pesi.

Gli algoritmi di apprendimento sono diversi; alcuni esempi di base sono: il *backpropagation*, la regola Delta [30], la regola di Hebb [82], ecc. Ciò che definisce invece il tipo di una RNA sono le modalità utilizzate per risolvere la tipologia di problema, soprattutto, il modo in cui i dati d'input influiscono nell'apprendimento della rete.

Ne consegue che la domanda da porsi nel momento in cui si appresta a strutturare la rete con la quale si intende lavorare sia la seguente: qual è il ruolo dell'algoritmo di apprendimento della rete? Gran parte degli algoritmi supervisionati cercano di ridurre al minimo il valore di una funzione di costo basata sul calcolo dell'errore, ossia, dello scostamento tra valori attesi e valori ottenuti.

Esiste un'ampia varietà di funzioni per le reti neurali, incluse quelle per propositi ingegneristici, quali ad esempio il riconoscimento di *pattern*, previsione e comprensione di dati [25]. L'architettura della rete cambia in base alla natura del problema che la rete è impiegata a risolvere. Solitamente le reti neurali sono adoperate per la risoluzione di problemi di classificazione e rappresentazione di classi, posto un insieme di dati.

La classificazione dei modelli di reti neurali può essere elaborata in relazione

alle unità di rete o alla struttura della rete. Le possibili classificazioni legate alle unità di reti sono:

- Basate sul Perceptron (ad es.: *Multi-Layer Perceptron*);
- Basate sul Prototipo (ad es.: *Radial Basis Function*).

Per quanto riguarda la classificazione in base alla struttura della rete, esistono tipo diversi:

- Reti con un unico livello;
- Reti con molteplici livelli (*Deep*);
- Reti di tipo uni-direzionale (*Feed-Forward*);
- Reti di tipo ricorrente (*Feed-Back*);
- Reti con struttura statica;
- Reti con struttura dinamica.

Le classificazioni delle reti possono essere eseguite anche in base all'apprendimento (come già menzionato precedentemente nella sezione 2.3), i modelli più comuni sono quattro:

- Apprendimento supervisionato;
- Apprendimento non supervisionato;
- Apprendimento semi-supervisionato;
- Apprendimento per rinforzo (*reinforcement learning*);

I passaggi per l'apprendimento vengono solitamente suddivisi tra fase di *training* e fase di *test*. Risulta pertanto necessario effettuare una raccolta dei dati relativi al problema che verrà trattato e la relativa separazione di questi in un insieme di dati di *train* e di test. In questa fase, è necessaria molta cautela nell'analisi del problema al fine di evitare ambiguità ed errori nei dati. È poi importante garantire che i dati raccolti siano significativi e coprano ampiamente il dominio del problema, non solo le operazioni più comuni o di routine, ma anche quelle che si incontrano nei limiti del dominio e le sue possibili eccezioni. Come già accennato in precedenza, la separazione dei dati viene effettuata generalmente in un set di *train*, che verrà impiegato nella formazione della rete, e in un set di dati di test, che sarà responsabile della verifica delle

prestazioni dell'RNA in situazioni reali di utilizzo. In alcuni casi, è possibile applicare un'ulteriore suddivisione all'interno del set di *training* che forma un cosiddetto set di validazione: esso può essere usato per analizzare l'efficienza della rete in casi generici durante il *training* e può essere adottato come criterio per interrompere l'addestramento. Solitamente l'insieme di validazione è grande circa il 20% del set di *train* e, in alcuni casi, quest'ultimo è diviso tra l'80% e il 20%, dove la porzione più grande viene usata per l'addestramento e la più piccola per la validazione della rete [18].

Una volta che questi insiemi sono stati determinati, essi vengono collocati in modo casuale al fine di impedire l'insorgere di tendenze associate all'ordine di inserimento dei dati. In alcuni casi, potrebbe essere necessario "pre-elaborare" questi dati; tale processo può avvenire tramite la normalizzazione e le conversioni di formato in modo da rendere i dati più adatti all'uso in quella specifica RNA [76], [40].

Dopo la raccolta e la separazione dei dati, è necessario definire una configurazione di rete. Essa viene effettuata attraverso la selezione del paradigma neurale più appropriato, la determinazione della tipologia di rete che verrà utilizzata (numero di livelli, numero di unità in ogni livello, ecc.) e infine la scelta dei parametri di sviluppo dell'algoritmo di *training* (le funzioni di attivazione, di costo, ecc.). Tutto ciò determina le prestazioni del sistema risultante. Per eseguire tali passaggi esistono diverse tecniche; solitamente la scelta finale viene compiuta in modo empirico basandosi sulla tipologia del problema da risolvere. A causa di questi aspetti, la definizione di configurazione di rete viene considerata un'operazione difficile e richiede una grande esperienza e competenza dei suoi progettisti.

### 3.1.2 Training, validation e test

Nella fase di addestramento (o *training*) della rete, guidata dall'algoritmo selezionato, verranno regolati i pesi delle connessioni. In questo stadio è fondamentale tener conto di alcuni fattori, come l'inizializzazione della rete, la modalità di addestramento ed il tempo impiegato dallo stesso. Una corretta scelta dei valori iniziali dei pesi consente una riduzione del tempo di *training*. In generale, questi valori sono numeri casuali distribuiti uniformemente su un dato intervallo. Una scelta sbagliata di tali valori può provocare una convergenza a minimi locali. Per risolvere questo inconveniente, i ricercatori Nguyen e Widrow hanno scoperto una funzione particolare che può essere utilizzata per determinare valori iniziali più idonei, attraverso una semplice scelta dei

numeri in modo del tutto casuale [60].

Per quanto riguarda la modalità di *training*, si usa spesso scegliere la quantità standard di esempi per ciascun passo di addestramento. Tale scelta viene operata a causa della minore memorizzazione dei dati e della minore suscettibilità verso problemi dei minimi locali, grazie alla natura stocastica della ricerca che l'algoritmo persegue. Con l'uso della modalità *batch*, i valori  $\delta$  di aggiustamento vengono accumulati e la stima del vettore del gradiente risulta migliore, offrendo così maggiore stabilità all'addestramento.

Definire quale modalità sia la più efficiente dipenderà dalla natura del problema in questione. Quanto al tempo di *training*, esso può essere influenzato da fattori differenti, anche se nella maggior parte dei casi è interessante definire un criterio di interruzione dello stesso. Il criterio di fermata dell'algoritmo *backpropagation*, ad esempio, utilizza un numero massimo di cicli, sebbene dovrebbe essere considerata anche la percentuale di errore medio per ciclo e la capacità di generalizzazione della rete in questione. È possibile che in un dato momento, la generalizzazione inizi a degenerare, causando così un problema detto *over-training*, che si verifica quando la rete si specializza nel set di dati usato per il *training*, perdendo quindi la sua capacità di generalizzazione. Il momento giusto per interrompere il *training* si presenta dunque quando la rete mostra una buona capacità di generalizzazione e, allo stesso tempo, la percentuale di errore risulta abbastanza limitata, inferiore ad un errore ammissibile.

Dopo il *training*, si arriva alla fase di test. In questo passaggio, il set di test selezionato viene utilizzato per determinare le prestazioni della rete con dati mai visti prima. Le prestazioni della rete valutate in questa fase rappresentano un buon indicatore dell'effettiva efficienza della rete. È inoltre importante considerare altri test, come l'analisi del comportamento della rete attraverso l'utilizzo di input speciali e la valutazione dei pesi attuali della rete.

### 3.1.3 L'architettura della rete

La quantità di neuroni nel livello d'input della rete viene generalmente definita in base al problema che verrà affrontato. Tuttavia, la quantità di neuroni nei livelli intermedi e di output dipende da scelte strutturali, a livello di progettazione, dalla forma in cui si vuole predisporre l'architettura, dal tipo di caratteristiche che si vogliono valutare e così via. La rete inoltre, possiede un sorta di peso speciale chiamato "Bias", il cui valore è solitamente 1 o -1: considerando che una rete neurale cerca di adattare una funzione per risolvere un certo problema, il Bias altro non è che l'elemento che indica verso quale

direzione dobbiamo “spostare” i nostri valori. Pertanto, esso ha l’effetto di aumentare o diminuire l’input “netto” della funzione in base al suo valore (positivo o negativo).

Per quanto concerne la rete Perceptron a molteplici livelli (approfondita ulteriormente), l’aumento della quantità di neuroni nel livello nascosto intensifica anche la capacità di mappatura non lineare della rete, ovvero la quantità di caratteristiche che si desidera tenere in conto. Ciò nonostante, occorre prestare molta attenzione affinchè il modello non sovrapponga alcuni dati; se si utilizza un modello troppo complesso su un problema semplice, la rete sarà soggetta a *overfitting* [77], [90]. Potrebbe anche accadere il contrario, nel caso in cui una rete con pochi neuroni nel livello nascosto non sia in grado di riprodurre la classificazione desiderata, causando quello che viene chiamato *underfitting* [90]. Quest’ultimo evento potrebbe anche essere causato da un’interruzione anticipata del training dei dati, ossia, quando la quantità di epoche<sup>1</sup> non risulta sufficiente. Anche la normalizzazione dei dati di input è un processo di grande importanza per la prevenzione di questo tipo di problemi: le caratteristiche delle funzioni Sigmoid quando sono in presenza di saturazione rappresentano un esempio in grado di illustrare propriamente l’importanza della normalizzazione dei dati. Risulta quindi interessante avere i valori degli attributi degli input contenuti in un intervallo predefinito; si potrebbe ad esempio dire che i valori degli attributi saranno collocati nell’intervallo  $[0; 1]$  o  $[-1; 1]$ .

Per quanto riguarda poi l’inizializzazione dei valori nei vettori di pesi e il Bias, appare oramai evidente come la qualità e l’efficienza dell’apprendimento supervisionato nelle RNA, soprattutto quelle a molteplici livelli, dipendano molto dalla strutturazione dell’architettura della rete, dalla funzione di attivazione dei neuroni, dalla regola di apprendimento e, infine, dai valori iniziali dei vettori dei pesi. I valori ottimali dei pesi sono sconosciuti a priori, in quanto dipendono principalmente dall’insieme dei dati di *training* e dalla natura della soluzione [79].

Se si suppone che l’architettura di una rete ipotetica che si vuole progettare sia già stata definita, ossia che le funzioni di attivazione e le relative regole di apprendimento siano già state correttamente determinate, allora l’unico fattore dal quale dipende un buon processo di *training* è rappresentato dalla corretta definizione del set iniziale di pesi della rete. Il che vuol dire avere un insieme che guida la fase di *training* verso una soluzione soddisfacente, al di là dei minimi locali e dei problemi di instabilità numerica. Pertanto, la matrice di pesi

---

<sup>1</sup>Chiamasi *epoca* ciascun ciclo di apprendimento.

iniziale da usare nella fase di addestramento di reti a molteplici livelli esercita una grande influenza sulla velocità dell'apprendimento.

Una scelta iniziale non corretta per i valori dei pesi può condurre il processo di *training* ad una soluzione non soddisfacente (anche se il processo di ottimizzazione è riuscito), oppure ad altri problemi che potrebbero altrimenti essere evitati.

## 3.2 Il Perceptron

Il modello di rete denominato Perceptron fu proposto da Rosenblatt nel 1958 [67]. Inizialmente si trattava di un modello basato sull'apprendimento supervisionato, composto da un unico neurone che compie la somma ponderata delle entrate producendo un valore che viene poi valutato per mezzo di una funzione di attivazione di tipo lineare.

Una funzione di attivazione definisce l'output di un neurone in termini del “potenziale di attivazione”, ovvero è responsabile del formato che si ottiene come output. Alcuni esempi di funzioni di attivazione sono *Hard limit*, *Piecewise linear*, *Sigmoid* (cfr. fig. 3.2).

Un Perceptron riceve un insieme di valori di entrata  $\hat{x}$  ed il valore desiderato  $y$  per il dato insieme di entrata. I valori di entrata vengono moltiplicati per un vettore di pesi  $\hat{w}$  (i valori di  $\hat{w}$  non devono essere nulli e vengono generati aleatoriamente) e poi sommati. La somma ottenuta ( $s$ ) è passata come parametro ad una funzione di attivazione  $\phi$ :

$$\phi(s) = \begin{cases} +1, & \text{se } s \geq 0 \\ -1, & \text{se } s < 0 \end{cases} \quad (3.1)$$

L'algoritmo per l'apprendimento di una rete di tipo Perceptron si basa sulla valutazione del risultato della funzione di attivazione, confrontandolo con il valore atteso  $y$ . In questo caso, l'attivazione consiste in una funzione per la classificazione di tipo lineare. Dunque, se la rete classifica come 1 quando dovrebbe invece classificare come -1, allora i pesi vengono diminuiti proporzionalmente al valore di entrata; se invece avviene il contrario, i pesi devono essere aumentati di forma proporzionale a  $\hat{x}$ , perciò:

$$w_{i+1} = w_i + c(y_i - o_i)x_i \quad (3.2)$$

Dove  $o_i$  è l'output ottenuto e  $c$  è una costante di correzione.

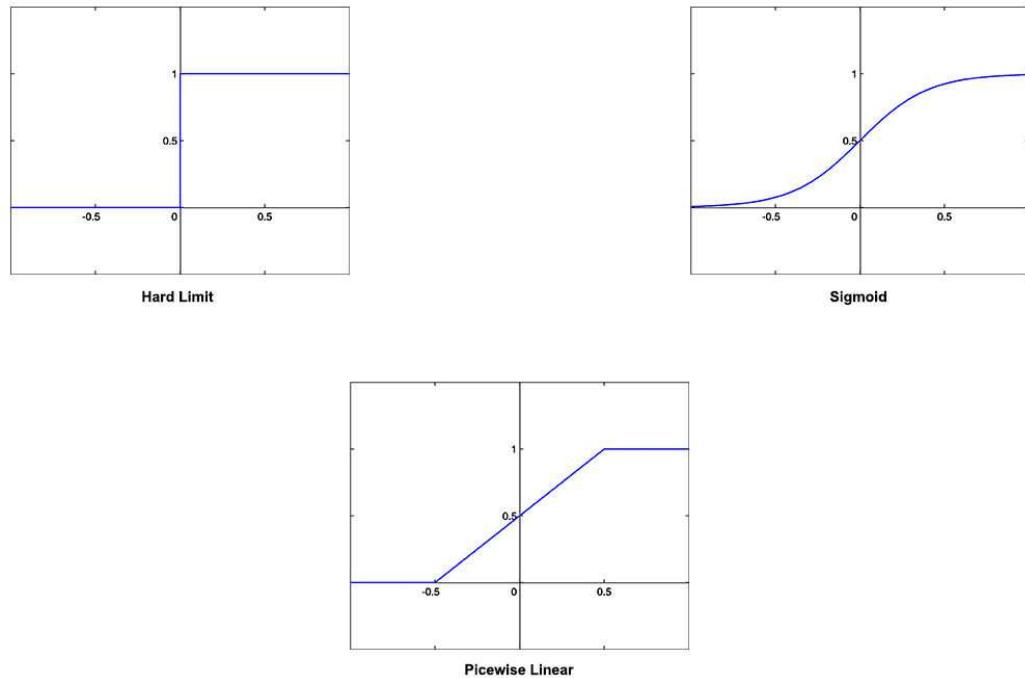


Figura 3.2: Activation functions

### 3.3 L'MLP ed il Backpropagation

Il Perceptron con molteplici livelli viene utilizzato spesso in quanto si tratta di un classificatore di tipo universale. Nel caso di risoluzione di problemi di tipo non lineare, non è possibile ricorrere ad un modello Perceptron elementare. L'architettura di un MLP viene definita come segue: il livello di input (in giallo), il livello nascosto (in azzurro) ed il livello di output (in verde) (cfr. figura 3.3). La quantità di neuroni muta invece in base al tipo di problema, risultando dunque adattabile a seconda del modello di riferimento.

Sostanzialmente esistono due fasi per il processing di questo tipo di rete: la fase di propagazione e quella di adattamento. Nella fase di propagazione, il flusso della rete va nella direzione del livello di output: si parla pertanto della cosiddetta fase *forward*.

La fase di adattamento avviene invece quando risulta necessario tornare in direzione del livello di input in modo da aggiustare i pesi della rete. Tale aggiustamento di pesi viene compiuto verosimilmente dall'algoritmo di apprendimento di un Perceptron, dove per ogni neurone si calcola l'output e la differenza tra il valore atteso e quello ottenuto; suddetta differenza viene poi usata per correggere i pesi attraverso la propagazione all'indietro (*backpropagation*).

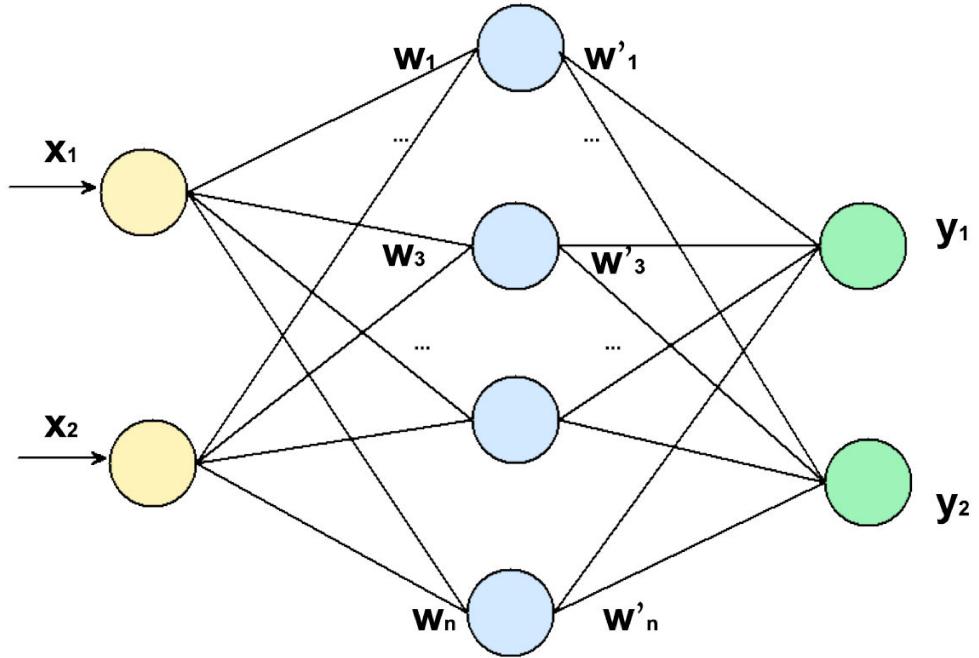


Figura 3.3: Architettura MLP - propagation

*gation).* L’idea base dell’algoritmo di propagazione all’indietro è relativamente semplice: essendo richiesto che tutte le funzioni usate nella rete non possiedano valori privi di risultato, che siano cioè derivabili in tutti i punti, allora ne consegue che attraverso quello che chiamiamo “gradiente” risulta possibile trovare quei valori che massimizzano il risultato della funzione di calcolo dell’errore, ossia  $valore_{atteso} - valore_{ottenuto}$  (ad es.: *mean square error*). Chiaramente, tale comportamento non corrisponde a ciò che vogliamo ottenere, poiché l’idea è proprio quella di minimizzare l’errore e non di massimizzarlo; per questo motivo, il gradiente viene sottratto ai pesi. Il gradiente non è nient’altro che il calcolo della derivata parziale della funzione di calcolo dell’errore in rapporto con il peso che vogliamo aggiornare. Lo pseudo-algoritmo per l’esecuzione della propagazione all’indietro è riportato in [3.1]:

Algorithm 3.1: Backpropagation

```

1 >> Inizializzare i pesi con valori casuali;
2 >> Fornire i valori di input e propagarli attraverso
   la rete, in direzione dell'output: x[1..n]
3 >> Calcolare gli errori: e[1..n]
4 >> Calcolare i gradienti locali dei neuroni del
   livello di output: delta_output[1..n]
5 >> Aggiornare i pesi per il livello di output:
6   for i=1; i++; i <= n-1:
7     w_output[i+1] = w_output[i] + costante_e *
                     delta_output[i] * x[i]
8 >> Calcolare i gradienti locali dei neuroni del
   livello nascosto: delta_hidden[i]
9 >> Aggiornare i pesi per il livello nascosto:
10  for i=1; i++; i <= n-1:
11    w_hidden[i+1] = w_hidden[i] + costante_e *
                     delta_hidden[i] * x[i]
12 >> Ripetere i passi 2 a 9 per ogni epoca
13 >> Ripetere il passo 8 fino a quando l'errore <= all'
   errore desiderato.

```

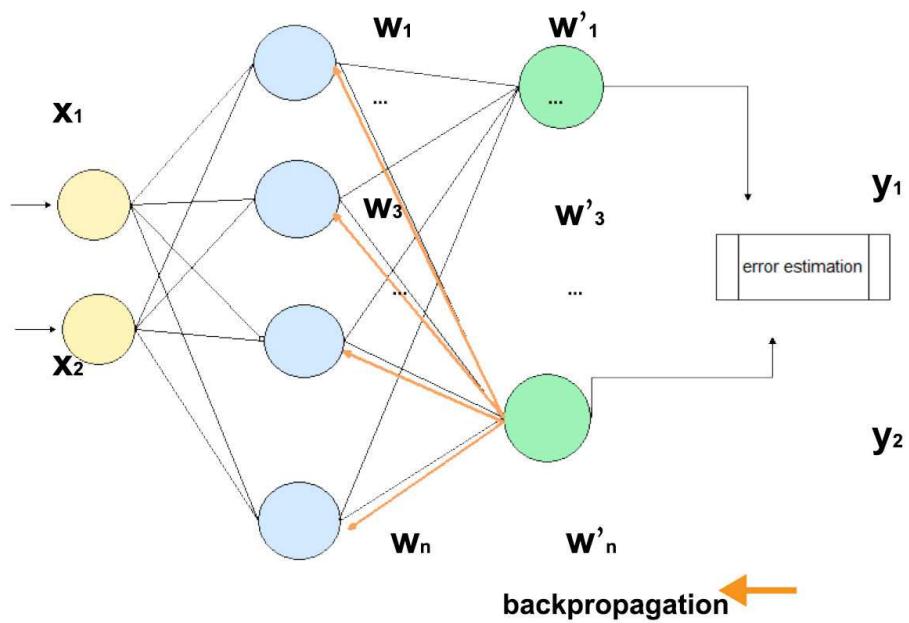


Figura 3.4: Architettura MLP - backpropagation

### 3.4 Epilogo

Questo capitolo ha trattato i principali argomenti riguardanti le reti neurali artificiali (RNA), presentando la definizione di una RNA, i suoi paradigmi di apprendimento e la sua classificazione in base alla struttura o ad altre caratteristiche. Sono state inoltre presentate le diverse fasi di elaborazione dei dati in una RNA e le variabili che compongono una qualsiasi scelta strutturale. Due famosi modelli infine, il Perceptron e l'MLP, sono stati brevemente descritti con l'obiettivo di dimostrare alcuni esempi di rete e di scelte architetturali.

# Capitolo 4

## Approccio proposto

Gli algoritmi di estrazione automatica di frasi chiave permettono di estrarre un insieme di frasi significative e descrittive a partire da un documento di testo, il quale avrà come risultato il suo contenuto riepilogato. Al fine di raggiungere le *performance* dei sistemi presentati in letteratura, questa tesi propone l'utilizzo di un modello matematico architetturale denominato rete neurale ricorrente, in particolare una *Bidirectional Long Short-Term Memory Recurrent Neural Network* (BLSTM RNN). In questo capitolo verrà presentato l'approccio indicato per la strutturazione di tale modello.

### 4.1 La strutturazione

La strutturazione di una rete neurale artificiale che verrà poi impiegata per risolvere un problema ben definito non può essere considerata un compito banale: essendo le reti neurali sono molto flessibili e capaci di analizzare diverse combinazioni di input, è necessario provare differenti modalità di elaborazione al fine di ottenere dei buoni risultati.

In corrispondenza della definizione di topologia di rete (vedi cap. 3), per la strutturazione del modello proposto, si è scelto di utilizzare la rete di tipo ricorrente (cfr. cap. 7), in grado di processare informazioni di tipo sequenziale. Le reti neurali ricorrenti sono capaci di eseguire lo stesso compito per ogni elemento di una sequenza, il quale può appartenere ad una qualsiasi tipologia di dato (immagini, frames, testo, ecc.). Il calcolo dell'output per questo tipo di rete dipende dai calcoli effettuati in precedenza; per tali ragioni, si usa spesso dire che tale tipologia di rete possieda una sorta di “memoria”.

La principale motivazione nell'uso di un modello di rete neurale di tipo ricorrente è ricollegabile al tentativo di creare una rete capace di realizzare compiti complessi, che il cervello umano compie normalmente con facilità, in particolare l'estrazione di parole o frasi chiave significative di un testo. Si tratta, pertanto, di strutturare un processore capace di estrarre conoscenza e di renderla disponibile in modo pratico. Il grande vantaggio nell'utilizzo di reti di questo tipo risiede principalmente nella loro capacità di risolvere problemi senza la necessità della definizione di un insieme di regole determinate a priori (come succede nella maggior parte dei modelli statistici classici) o di un modello esplicito. La scelta della più struttura corretta rende possibile quindi l'elaborazione di soluzioni per problemi per i quali sarebbe difficile creare un modello adeguato senza modificare interamente la sua struttura in presenza di cambiamenti del dominio o dell'ambiente.

La modellazione o strutturazione di reti neurali artificiali può essere sintetizzata nei seguenti quattro passi:

1. Scegliere i dati di esempi che verranno utilizzati per l'addestramento della rete, in modo che il dominio che vogliamo lavorare sia ben rappresentato. Va notato come gli esempi consistano in coppie di tipo (input, output desiderato);
2. Scegliere i dati di esempi che verranno usati per la validazione (valutazione della capacità di generalizzazione del modello);
3. Scegliere l'architettura appropriata;
4. Scegliere l'algoritmo per l'addestramento della rete;

Il problema dell'estrazione automatica di **parole chiave** da un insieme di dati testuali può essere facilmente risolto poiché l'obiettivo di un modello strutturato per questo fine è quello di imparare a classificare i dati in solo due classi (classificazione binaria): 0 o 1. Tali strutturazioni, tuttavia, non prevedono il compito di estrarre **frasi chiave** dai documenti testuali, cioè non sono in grado di determinare quando due parole chiave, l'una accanto all'altra, rappresentano un unico termine.

Seguendo quindi i passaggi sopraelencati e al fine di configurare la struttura di partenza, si è optato l'utilizzo di un dataset ben noto nell'ambito dell'estrazione di frasi chiave: Inspec 2003 [33] (cfr. cap. 9). L'approccio scelto per risolvere il problema è stato quello supervisionato: a ciascun dato di ingresso nel set di *train* è stato perciò necessario associare un dato di output che

rappresentava la risposta al problema, ovvero un'etichetta. Dunque, ad ogni parola “ $w$ ” appartenente al testo, viene associata una delle seguenti etichette:

- N\_KP: rappresenta le parole nel testo che non sono parole-chiave;
- B\_KP: rappresenta l'inizio di una frase chiave;
- I\_KP: rappresenta l'interno di una frase chiave.

È stato scelto di dare una rappresentazione numerica ad ogni etichetta, ovvero:

- N\_KP = 0;
- B\_KP = 1;
- I\_KP = 2.

Pertanto la risoluzione di tale problema prevede la classificazione dello stesso a multi-classi.

Per risolvere il problema dell'estrazione di **parole e frasi chiave** dai documenti di testo, sono stati attuati i seguenti passaggi: per prima cosa, per ogni documento si divide il testo in un insieme di frasi; ogni frase appartenente a tale insieme viene a sua volta suddivisa in un gruppo di parole (tale processo verrà spiegato nel dettaglio nel capitolo 5). Ogni parola così ottenuta viene poi sostituita dalla sua rappresentazione in uno spazio vettoriale (cfr. capitolo 6) e ad ogni rappresentazione viene associata un'etichetta: N\_KP = 0, B\_KP = 1 o I\_KP = 2; tali rappresentazioni descrivono i valori desiderati per ogni parola nel set di dati, cioè per il set di training, validazione e di test. Vengono assegnati, infine, i valori di input alle unità della rete *Bidirectional Long short-term memory* (BLSTM), ovvero si assegnano alle unità BLSTM le rappresentazioni delle parole in un spazi vettoriali (*Word Embeddings*); vengono inoltre forniti i valori desiderati, associati ad ogni input. Si collega poi il livello contenente le unità BLSTM con quello di output Softmax, che comprende tre neuroni per ogni singola parola dell'input (cfr. Figura 4.1).

È importante notare come il livello di output di Softmax restituisca un valore per ogni neurone e come ciascun neurone rappresenti la probabilità di un'etichetta sommata ad uno (Softmax viene approfondita nella sezione 6.2.2). Detto questo, è opportuno sottolineare come i valori “ $y$ ” desiderati siano raffigurati mediante una rappresentazione “one-hot”; ad esempio, data l'etichetta N\_KP, la sua rappresentazione one-hot sarà [1, 0, 0], data l'etichetta B\_KP,

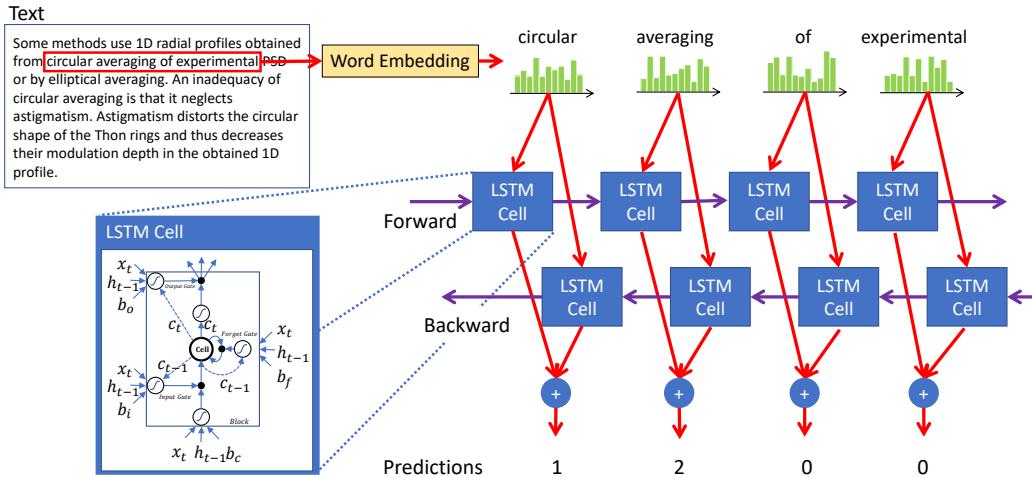


Figura 4.1: Esempio dell'approccio

si ottiene  $[0, 1, 0]$  e finalmente dato  $\text{I\_KP}$ , si ha  $[0, 0, 1]$ .

In questo modo, la rete è in grado di apprendere correttamente attraverso gli esempi, classificandoli adeguatamente come appartenenti ad una delle tre possibili classi (vedi figura 4.2). Nell'esempio precedente è possibile notare un aspetto molto comune dei problemi di classificazione multi-classe: uno sbilanciamento del numero di esempi per ogni classe, nel quale la classe N\_KP appare la più favorita. Riscontrare questo tipo di squilibrio nei dati è abbastanza comune, soprattutto quando si tratta dell'estrazione di frasi chiave. Quando leggiamo un testo e dobbiamo evidenziare le frasi chiave che vengono incontrate all'interno del medesimo (come ad esempio l'atto di sottolineare un testo), infatti, è normale considerare frasi o parole chiave che sono sparse, cioè, separate nel testo e di solito non presenti in grande quantità. Per affrontare questo problema, si è optato per adottare la seguente soluzione: per ogni esempio di *train* e per ciascun valore “ $y$ ” desiderato associato all’ input, viene assegnato peso diverso per ogni classe. Ciò assegnerà una certa importanza a quelle classi che contengono un numero minore di esempi, ottenendo così un migliore equilibrio tra i dati. Ad esempio, se prendiamo come riferimento una parola appartenente alla classe N\_KP, questa avrà importanza uguale a “1”, mentre un esempio della classe B\_KP, avrà importanza uguale a “50”; questo approccio aumenta le stime di probabilità per le classi meno favorite nella matrice di probabilità. Se si saltasse tale passaggio, il sistema probabilmente classificherebbe quasi tutte le parole come appartenenti alla classe N\_KP.

Per esempio, sia la seguente frase “*We train a neural network using Keras*” e siano le seguenti frasi chiave “*neural network*” e “*Keras*”; la rete è in grado

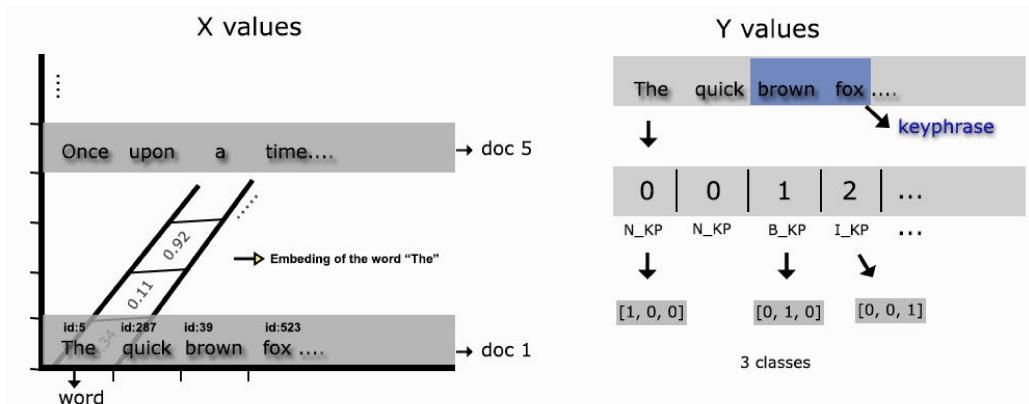


Figura 4.2: Modello proposto

di classificare ciascuna parola come segue:

$$\begin{aligned}
 &We/N\_KP/[1,0,0] \quad train/N\_KP/[1,0,0] \quad a/N\_KP/[1,0,0] \\
 &neural/B\_KP/[0,1,0] \quad network/I\_KP/[0,0,1] \quad using/N\_KP/[1,0,0] \\
 &\quad Keras/B\_KP/[0,1,0].
 \end{aligned}$$

Come affermato in precedenza, ciascun neurone restituisce un valore che rappresenta la probabilità della classe sommata ad uno; perciò, in un secondo momento, tale valore verrà valutato e classificato effettivamente come una parola appartenente ad una delle tre classi.

Per la creazione della rete si è scelto di ricorrere al linguaggio Python grazie alla sua facile integrazione con strumenti che supportano le tecniche di *Machine Learning* basate su *Deep Learning*, come ad esempio **Keras**.

### 4.1.1 Word Embeddings

Il livello di input del modello qui proposto calcola la rappresentazione di ogni parola in spazi vettoriali (cfr. capitolo 6). Secondo [58], è possibile ottenere una migliore rappresentazione semantica delle parole quando il modello viene addestrato su un corpus di training di dimensioni sufficientemente grandi. Sfortunatamente, la maggior parte dei set di dati trovati in letteratura (appropriati per svolgere il compito di estrarre frasi chiave) sono relativamente piccoli, il che provoca una grande difficoltà nell'atto di estrarre informazioni semantiche dalle parole rappresentate dagli spazi vettoriali. Per risolvere questo problema, si è scelto di adottare il modello *Stanford's GloVe Embeddings*

[63]: un modello pre-addestrato su oltre sei miliardi di parole estratte da *Wikipedia* e da altri testi trovati sul Web.

Il modello *GloVe* è disponibile in diversi formati, che variano in base al numero di *features* con cui esso è stato addestrato; sono disponibili modelli addestrati con 50, 100, 200 o 300 *features*. Si è scelto inoltre di testare l'efficienza del modello mediante l'uso del modello pre-addestrato *Word2Vec pre-trained vectors* [58]. I risultati ottenuti per ciascun modello e formato sono stati confrontati e presentati nel capitolo 9.

### 4.1.2 Funzioni di attivazione

Il livello più interno dell'architettura proposta formula i passaggi di una *Bidirectional Long short-term memory Recurrent Neural Network* (BLSTM). Sono stati qui utilizzati due tipi di funzioni di attivazione della rete: la funzione hard Sigmoid (vedi eq. 4.1), usata come funzione di attivazione al passo ricorrente, e la funzione Tangente iperbolica (vedi eq. 4.2) come funzione di attivazione del livello. Tali funzioni sono state scelte in base agli aspetti descrittivi di un BLSTM (spiegato in maggior dettaglio nel capitolo 7). In particolare, la funzione hard Sigmoid costituisce un'approssimazione della funzione Sigmoid [24].

$$\text{sigmoid}_{\text{hard}}(x) = \begin{cases} 0, & x < -2.5 \\ 1, & x > 2.5 \\ 0.2x + 0.5 & -2.5 \leq x \leq 2.5 \end{cases} \quad (4.1)$$

$$\tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}} \quad (4.2)$$

### 4.1.3 Funzioni di costo

La funzione di costo scelta per essere ridotta al minimo durante la fase di training della rete è la cosiddetta *Categorical Cross Entropy* (eq. 4.3). La *Categorical Cross-entropy loss* viene solitamente impiegata quando si desidera conseguire un'interpretazione probabilistica dello score ottenuto con l'addestramento della rete. Sudetta funzione, infatti, misura la dissomiglianza tra la risposta ottenuta e la risposta desiderata, producendo così un classificatore multi-classe in grado di prevedere quale classe si adatti meglio alla risposta tra tutte le altre disponibili, oltre a prevedere la distribuzione probabilistica sulle possibili classi. Ricorrendo alla *Categorical Cross-entropy loss*, si presume che l'output sia stato prodotto utilizzando una trasformazione Softmax (il che si adatta perfettamente al nostro problema) [18]. Tale funzione di costo, dunque, calcola l'entropia incrociata tra la corretta distribuzione

di probabilità “p”, codificata in una rappresentazione *one-hot*, e la distribuzione delle probabilità dell’output “o” della rete neurale generata dalla funzione Softmax.

$$\text{CrossEntropy}(p, o) = - \sum_x p(x) \log o(x) \quad (4.3)$$

#### 4.1.4 Algoritmo di ottimizzazione

L’aggiornamento dei parametri della rete dev’essere eseguito in modo da ridurne il costo. Generalmente l’algoritmo *Gradient Descent* viene utilizzato in combinazione con il metodo *backpropagation* ma, in alcuni casi, soprattutto dove risulta necessario utilizzare l’ottimizzazione in milioni di istanze, l’uso del *Gradient Descent* diviene proibitivo, poiché questo algoritmo calcola il gradiente per tutti le singole istanze. Tale difficoltà emerge soprattutto quando, per il calcolo di una singola epoca risulta è necessario caricare tutte le istanze in una memoria limitata, un processo spesso impraticabile. Per risolvere questo problema sono state proposte alcune alternative:

- SGD [10];
- Momentum [68];
- Adagrad [16];
- RMSProp [80];
- Adam [36].

In questa tesi si è scelto di far uso dell’algoritmo *Root Mean Square Propagation* (RMSProp) [80], che, secondo la documentazione di Keras, risulta il più adatto all’uso delle reti neurali ricorrenti. RMSProp calcola le medie dei gradienti più recenti per ciascun parametro e le utilizza per modificare individualmente il tasso di apprendimento (“*learning rate*” in inglese) prima di applicare i gradienti, pertanto risulta un metodo in cui il tasso di apprendimento viene adattato in base a ciascun parametro.

#### 4.1.5 Altri iperparametri

Alcuni iperparametri, come il numero di unità BLSTM e il numero di epoche, sono stati scelti empiricamente attraverso l’esecuzione di diversi test. Alcuni test con valori differenti per questi iperparametri sono riportati nel capitolo 9, ma la scelta finale più adatta al dataset Inspec consta di 184 unità nel livello BLSTM e 14 Epoche. La dimensione del *Batch* di training (numero di esempi da utilizzare in ciascuna iterazione di training) dovrebbe essere definita tenendo conto dei vincoli di memoria, così come degli algoritmi di ottimizzazione utilizzati. *Batch* di piccole dimensioni rendono difficile ridurre al minimo i costi, mentre *Batch* di grandi dimensioni possono interferire nella velocità di convergenza. Pertanto, sulla base di

esempi di altri modelli Deep learning, si è scelto il valore di 32 esempi per *Batch*. Per evitare problemi come l'*overfitting*, è stato poi utilizzato un dropout di 0.5 (50%) tra il livello BLSTM ed il livello di output della rete. Il dropout è una tecnica proposta in [31] per la minimizzazione dell'overfitting che, nella fase di training, regola arbitrariamente l'attivazione dei neuroni, causando la riduzione della complessità delle funzioni utilizzate e, di conseguenza, aumentando l'obliquità delle medesime. Le scelte dei valori attinenti ai parametri degli altri dataset utilizzati per effettuare i test sulla struttura proposta vengono riportati nel capitolo 9.

## 4.2 Epilogo

In questo capitolo sono stati descritti i passaggi compiuti per la strutturazione del modello proposto in questa tesi. La parte iniziale espone i passi usati per la strutturazione della rete, seguiti dalla descrizione del modello e dall'insieme dei dati impiegati per i test di adattamento degli iperparametri. Le altre sezioni sono dedicate ad una narrazione più dettagliata delle scelte strutturali fatte e delle motivazioni per ciascuna di esse.

# Capitolo 5

## Pre-processing dei dati

Il pre-processing dei dati è considerato un compito richiedente una grande quantità di conoscenza del dominio in cui viene fornita l'indagine. Generalmente, il *pre-processing* dei dati è un processo semi-automatico, legato cioè alla capacità di un umano di identificare i problemi presenti nei dati e di trattarli correttamente. In questo capitolo verrà descritto nel dettaglio il processo di elaborazione dei dati di input della rete, le principali difficoltà incontrate e quale metodologia produce risultati migliori, posto il problema di partenza.

### 5.1 L'elaborazione del linguaggio naturale

Di solito le fasi dell'elaborazione del linguaggio naturale, o *Natural language processing* (NLP) in inglese, riflettono sulle distinzioni linguistiche tracciate tra sintassi, semantica e pragmatica. Tali diversificazioni rappresentano un buon punto di partenza quando si considera l'elaborazione di testi in linguaggio naturale. In generale, le tecniche di NLP mirano a risolvere problemi che richiedono l'elaborazione computazionale di uno o più linguaggi naturali. Per creare un modello linguistico e consentire al computer di comprenderlo, è necessario eseguire una serie di filtri (*pre-processing*) del testo (audio, ecc.), lasciando solo ciò che costituisce un'informazione rilevante; in questo modo, i dati diventano meno scarsi e più adeguati alla risoluzione del problema in esame.

#### 5.1.1 Tokenization

La tokenizzazione (o *tokenization* in inglese) è un'operazione che viene spesso utilizzata per lo scopo della normalizzazione del testo. Con normalizzazione del testo si intende l'esecuzione di una serie di procedure in grado di modificarlo, come ad esempio la trasformazione di lettere maiuscole in minuscole, la rimozione di caratteri speciali, la rimozione delle *tag* e molte altre funzioni. La normalizzazione del testo si ottiene poi per mezzo della segmentazione delle parole, ovvero attraverso

la suddivisione del testo in sequenze di caratteri. Tale ripartizione viene eseguita tramite la localizzazione del limite di ogni parola, cioè, tramite l'indicazione di dove essa inizia e termina, ad esempio spazi, virgole, punti e così via. Il risultato di tale segmentazione viene spesso chiamato “token” [E]. Per quanto riguarda le lingue in cui la delimitazione è indicata da qualche segno di punteggiatura o spazio bianco (diversamente ad esempio dal cinese e tailandese), è necessario fare attenzione alle ambiguità esistenti nell’uso di tali segni di punteggiatura, poiché lo stesso segno può svolgere funzioni diverse all’interno della stessa frase, come nel caso delle abbreviazioni in un testo: “Dr.”, “Mr.”, “Ms.”, ecc. Il punto finale (“.”), in questo caso, potrebbe quindi essere utilizzato sia per indicare il limite di una frase, ovvero dove finisce, sia per segnalare un’abbreviazione.

#### Lexical Tokenization

La tokenizzazione lessicale configura ogni parola come un token nel testo, identificandola anche se questa viene trovata senza uno spazio tra il segno di punteggiatura e la fine della stessa. Per illustrare tale operazione, si riporta il seguente esempio:

“Giovanni è un ragazzo felice.”  
[“Giovanni”, “è”, “un”, “ragazzo”, “felice”, “.”]

#### Sentence Tokenization

La tokenizzazione in frasi, invece, identifica e contrassegna le frasi in un testo:

“Prima frase. Seconda frase . Terza frase!”  
[“Prima frase.”, “Seconda frase .”, “Terza frase!”]

Per quanto riguarda il compito di estrazione automatica di frasi chiave, la divisione di un testo in frasi, e successivamente in token, risulta un passaggio molto importante per il *pre-processing* del testo poiché i segni di punteggiatura indicano dove termina una frase e ne inizia un’altra. Se tali segni di punteggiatura fossero completamente esclusi da un testo relativamente lungo, un sistema automatico impiegato per l’estrazione di frasi chiave potrebbe effettivamente realizzare delle classificazioni errate, soprattutto se non riceve informazioni aggiuntive sulle caratteristiche linguistiche delle parole (o se semplicemente non ne tiene conto). Di seguito viene riportato un esempio di un errore di estrazione di frase chiave da un testo, nel quale vengono esclusi i segni di punteggiatura. Supponiamo di aver il seguente testo:

“She was looking for something new. York is a big city, there must be something!”

Supponiamo quindi di impiegare un sistema di estrazione automatica di frasi chiave di documenti di testo, che escluda la punteggiatura. Per la frase sopra

riportata, “*new york*” potrebbe rappresentare una frase chiave estratta da questo ipotetico sistema. E’ chiaro che “*new york*” non costituisca una frase chiave in grado di rappresentare il testo, dal momento che esso non parla di *New York* in nessun momento. Se il testo viene infatti filtrato eseguendo la rimozione di tutti i segni di punteggiatura, il risultato ottenuto sarebbe ambiguo:

“she was looking for something new york is a big city there must be something”

Inizialmente, il modello proposto in questa tesi faceva ricorso alle funzioni fornite da **Keras** per la pre-elaborazione dei dati [13]. Esse costituiscono una buona alternativa per alcune tipologie di problema, soprattutto quelle più semplici, ma non sono indicate quando il problema richiede una strutturazione più dettagliata, come ad esempio la suddivisione di un testo prima in frasi e poi in parole. **Keras** risulta quindi un’alternativa interessante ma, al contempo, limitata.

**Keras** fornisce una funzione che divide il testo in sequenze di parole separate da un carattere opzionale (il cui valore predefinito è uno spazio). Tramite la stessa funzione è possibile scegliere come filtrare la punteggiatura del testo, come ad esempio “!#\$%&()\*”. È anche possibile scegliere di convertire tutto il testo in “*lowercase*” o meno: nel caso delle attività come il *Named entity recognition*, quest’ultima opzione non è consigliabile, poiché la classificazione delle parole del documento tiene in conto il testo a livello di caratteri (le maiuscole, ecc.) e non solo a livello di parole.

Senza necessariamente reinventare la ruota, ovvero senza dover adattare le funzioni di **Keras**, si è scelto di utilizzare uno strumento ben noto nel campo dell’elaborazione del linguaggio naturale, chiamato **NLTK** (*Natural Language Toolkit*). L’NLTK è un set di moduli **Python** che fornisce varie operazioni per l’elaborazione dei dati, oltre a diversi corpus di esempio, alcuni tutorial e così via [9]. In questa tesi, è stato confrontato l’uso del tokenizzatore **Keras** con quello NLTK: a causa dei punti evidenziati sopra, il tokenizzatore NLTK ha mostrato risultati nettamente migliori.

Esistono però anche altre operazioni offerte da NLTK che coprono l’ambito del pre-processing di dati, quali il *tagging* e lo *stemming*. Di seguito verranno presentate altre opzioni aventi come obiettivo la normalizzazione del testo nel NLP. Non tutte sono state sfruttate per lo scopo di questa tesi; in effetti, è stato possibile dimostrare come il modello qui proposto raggiunga i risultati dello stato dell’arte mediante tecniche di *Deep Learning* senza ricorrere a tecniche di filtraggio (pre o post processing), come ad esempio il filtraggio tramite *part-of-speech*, la rimozione di *stop-words* e di cifre numeriche.

- Rimozione di Stop-words: un’operazione molto usata nella pre-elaborazione di testi, dove parole molto frequenti nel tesò, come “la”, “di”, “del”, “della”, “che”, ecc., vengono filtrate. Il più delle volte, queste parole non rappresentano informazioni rilevanti per la costruzione del modello;

- Rimozione dei numeri: operazione che rimuove le cifre di un testo o le sostituisce con una *flag* (ad es. “DIGIT”);
- *Spell Checkers*: usati per gestire set di dati che contengono errori di ortografia, abbreviazioni e vocabolario informale;
- *Stemming* e *Lemmatization*: il processo di *stemmatization* riduce una parola al suo radicale. La parola “gatto” sarebbe ridotta a “gatt”, così come “gatta” e “gattino”. Il processo di *lematization* invece, riduce la parola al suo lemma, ossia la sua forma maschile singolare, ad esempio “gatto”, “gatta”, “gattini” sarebbero tutte parole rappresentate dal motto “gatto”; nel caso dei verbi, è la forma all’infinito ad essere il rappresentante: ad esempio il verbo “avere” è il rappresentante di tutte le sue varie forme “avevamo”, “avevo”, “aveva”, ecc;
- *Pos-tagging*: un *tagger* è responsabile del processo di definizione della classe grammaticale delle parole secondo le funzioni sintattiche, ad esempio “**sostantivo plurale**”, “**avverbio**”, ecc.

## 5.2 L'elaborazione dei dati di input

Per prima cosa si pone l’accento sulla scelta di usare l’NLTK solamente per lo scopo di suddividere il testo in frasi e successivamente in parole (*tokens*). I test, inoltre, sono stati eseguiti utilizzando dataset diversi che, a causa della grande differenza nella loro strutturazione (come il modo in cui le frasi chiave sono fornite, il formato dei file, ecc.), trattano i dati separatamente. Una volta ottenuti i dati pre-elaborati, inizia il processo di strutturazione delle frasi chiave desiderate attraverso l’assegnazione di etichette ad ogni parola (vedi capitolo 4). L’elaborazione del testo viene quindi svolta per ciascun insieme di dati: il dataset di *training*, *validation* e *test*.

Come primo passo, il testo viene suddiviso in frasi e, solo successivamente in *tokens*. Una volta eseguita tale operazione per ogni set di dati, viene creato un vocabolario attraverso l’uso di operazioni basate sulla classe *Dictionary* di Keras, dove la frequenza delle parole viene contata. Il vocabolario risultante rappresenta l’elenco di tutte le parole uniche trovate nei testi, in cui a ciascuna parola (o *token*) viene associata una frequenza che rappresenta il numero di volte che essa è stata individuata (cfr. Tabella 5.1). Volendo, è possibile scegliere un valore “x” in modo empirico, che servirà a limitare la quantità di parole impiegate per costruire il modello; si può scegliere, ad esempio, di usare solamente le “x” parole più frequenti nel vocabolario.

Tabella 5.1: Vocabolario

Word	Count
the	3387
and	2843
a	1287
...	
family	18
cold	7
honey	1

Una volta fatto questo, il vocabolario risultante sarà quello in cui ogni parola possiede un indice e una frequenza. Tale vocabolario sarà poi utilizzato per mappare ogni rappresentazione `word embedding` del modello pre-addestrato *GloVe* (o *Word2Vec*) con l'indice della parola nel vocabolario, potendo così sostituire ogni parola appartenente ad ogni testo con la sua rappresentazione vettoriale (vedi capitolo 4, figura 4.2). Il passo successivo sarà quello di fornire l'input elaborato al modello RNA proposto.

### 5.3 Epilogo

In questo capitolo sono state descritte le difficoltà incontrate nello *step* di elaborazione dei dati e le scelte compiute per affrontarle. In particolare, sono state introdotte le principali differenze tra lo strumento per la pre-elaborazione di dati NLTK [9] e le funzioni fornite da Keras [13]; la scelta fatta è stata motivata e la procedura iniziale per avviare il modello è stata brevemente descritta.

# Capitolo 6

## Word Embeddings

Esistono metodi differenti per rappresentare le parole; i due principali sono: la rappresentazione esplicita e la rappresentazione distribuita [46]. Per quanto riguarda quest'ultima tipologia, vi sono diverse metodologie che compiono questa funzione: *word embeddings*, ad esempio, è un metodo per la rappresentazione di parole basato sull'ipotesi distribuzionale di Harris [28], la quale afferma come parole in contesti simili abbiano probabilmente significati affini. Negli ultimi anni, sono stati introdotti da [59] gli spazi vettoriali continui. Essi esprimono le parole per mezzo di una rappresentazione basata sui *word embedding*, in un modo diverso dei classici modelli “*n-gram*”, considerando che questi ultimi lavorano a livelli di unità distinte. Secondo tale interpretazione, è possibile verificare la similitudine tra vettori di parole generati attraverso modelli matematici ricorsivi, che compiono operazioni lineari relativamente semplici.

Il metodo proposto da [59] per costruire rappresentazioni di parole tramite spazi vettoriali viene chiamato *Word2Vec*; esso consiste in un modello matematico volto all'apprendimento, ovvero una rete neurale artificiale (RNA) che elabora testi al fine di produrre “*word embeddings*”. *Word2Vec* riceve come input un corpus<sup>1</sup> testuale e produce come output un insieme di vettori di caratteristiche per le parole contenute nel corpus, dunque, per il suo vocabolario. Il modello *Word2Vec* è una RNA simile ad un *autoencoder* in cui si codificano le parole in vettori e dove l'obiettivo è quello di adattare i pesi correttamente per ricostruire ciò che si fornisce come input alla rete. Perciò, nel *Word2Vec*, i vettori dei pesi vengono adattati durante il *training* della rete a seconda del modello con l'intento di approssimare il più possibile le parole ai loro veri contesti o viceversa. In questo modo è possibile rappresentare i significati, caratteristiche semantiche e sintattiche delle parole in termini numerici

---

<sup>1</sup>Si ricorda che: un corpus è una collezione di testi contenente informazioni linguistiche valutate manualmente. Queste sono spesso usate nel *Machine learning* per la creazione di modelli statistici.

o più precisamente in termini probabilistici, ovvero la chance di coincidenza (*co-occurrence* dall'inglese)<sup>2</sup>.

## 6.1 Obiettivo

L'obiettivo di *Word2Vec* è quello di raggruppare i vettori di parole simili in uno spazio vettoriale che ci consenta di ricercare corrispondenze matematiche. Attraverso tale processo è possibile stabilire associazioni tra parole, ad esempio: "uomo sta per ragazzo ben come donna sta per ragazza" (ragazzo - uomo + donna = ragazza); è dunque possibile accettare l'esistenza di relazioni semantiche e sintattiche tra parole. Queste similitudini sono conosciute come regolarità linguistiche e non si limitano ad un'unica lingua; attraverso esse, è possibile catturare diversi tipi di rapporti. Le rappresentazioni distribuite vengono raffigurate come vettori che possiedono tante dimensioni: ogni dimensione potrebbe essere immaginata come una caratteristica (dall'inglese *feature*). Se venisse aggiunta un'etichetta per ogni dimensione in un vettore ipotetico posizionato a fianco degli altri vettori, sarebbe possibile visualizzare più chiaramente come la rappresentazione distribuita delle parole ci permetta di trovare similitudini e relazioni tra le stesse (cfr. figura 6.1) [14].

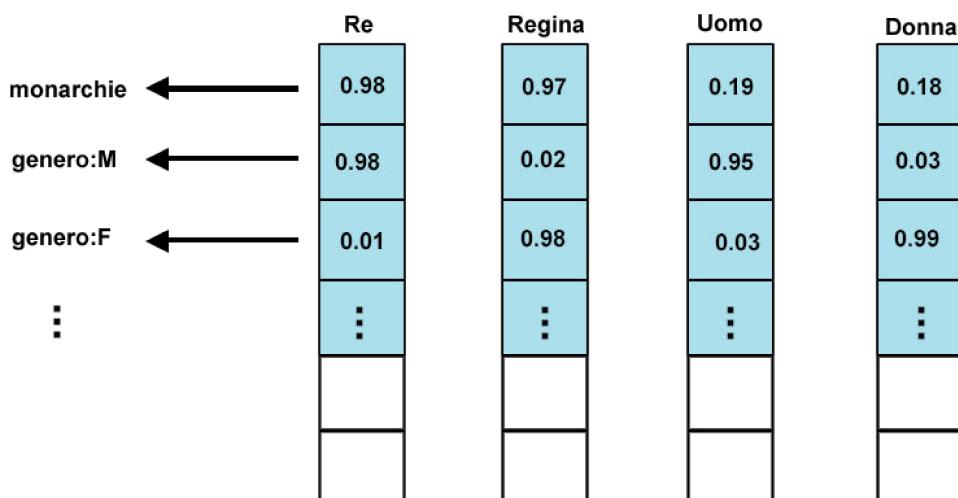


Figura 6.1: Le similarità

---

<sup>2</sup>Co-occurrence è un termine linguistico utilizzato per specificare la probabilità che due termini di un corpus compaiano l'uno a fianco all'altro in un certo ordine. Indica dunque la prossimità semantica.

Come già accennato precedentemente, la descrizione delle parole in maniera distribuita ci dà la possibilità di computare operazioni matematiche semplici allo scopo di trovare relazioni e similitudini: una volta ottenuti i risultati, è possibile visualizzarli utilizzando mezzi adeguati (funzioni, *library*, ecc) che ci consentano di “interpretare”, o trasformare, le centinaia di dimensioni in uno spazio 2D; fatto ciò, risulta relativamente semplice computare operazioni sopra tali vettori utilizzando ad esempio la “*cosine distance*”.

## 6.2 Approcci

Secondo [56], due sono gli approcci principali per la progettazione dei modelli *Word2Vec* (cfr. figura 6.2):

1. Continuous bag of words (CBOW)
2. Skip-gram

Entrambi gli approcci consistono in una RNA con un livello nascosto che utilizza l’algoritmo di *backpropagation* per l’aggiornamento dei parametri ad ogni nuovo esempio osservato. La principale differenza tra gli approcci CBOW e Skip-gram è data da ciò che ognuno di essi riceve come input e restituisce come output. I due approcci ricevono un intero corpus testuale e realizzano alcune impostazioni sui dati, al fine di poterli utilizzare come input alla rete. Data la seguente frase come esempio:

The quick brown fox jumps over the lazy dog

Descrivendo in maniera astratta il compito di *Word2Vec*, data la parola  $x = \text{fox}$  e dato l’obiettivo di trovare le parole dotate di un maggior livello di similitudine contestuale a  $x$ , il modello calcola la probabilità che la parola  $y$  (ad es.:  $y = \text{brown}$ ) compaia nei contesti della parola  $x$ . Ciò viene ripetuto per ogni parola appartenente al vocabolario. Dunque, **CBOW** è l’approccio che racchiuderà tutti gli  $y$  come input, cercando di attribuire una certa probabilità per la parola  $x$  in relazione agli  $y$ . **Skip-gram** invece, opera in modo inverso: data la parola  $x$ , esso calcola le probabilità per tutte le altre parole  $y$ . Mikolov et al. [56] affermano come l’approccio più usato sia Skip-gram poiché esso si è dimostrato più accurato.

Dall’esempio precedente, se chiamiamo “target” la parola  $x$ , è possibile affermare come l’approccio CBOW utilizzi il contesto per prevedere la parola target, a differenza di Skip-gram che ricorre invece alla parola target per prevedere il contesto. Posto che i modelli altro non sono che reti neurali artificiali semplici, articolate in tre livelli (il livello d’input, il livello di proiezione e di output), ciò che accade durante il *processing* della rete è proprio l’aggiornamento dei componenti della medesima (che verranno presentati in seguito), quando il vettore assegnato ad una parola non può ancora prevederne il contesto, o viceversa. Aggiornando i componenti della rete, ad ogni interazione i vettori delle parole simili in base al contesto vengono spostati sempre più vicini, l’uno all’altro.

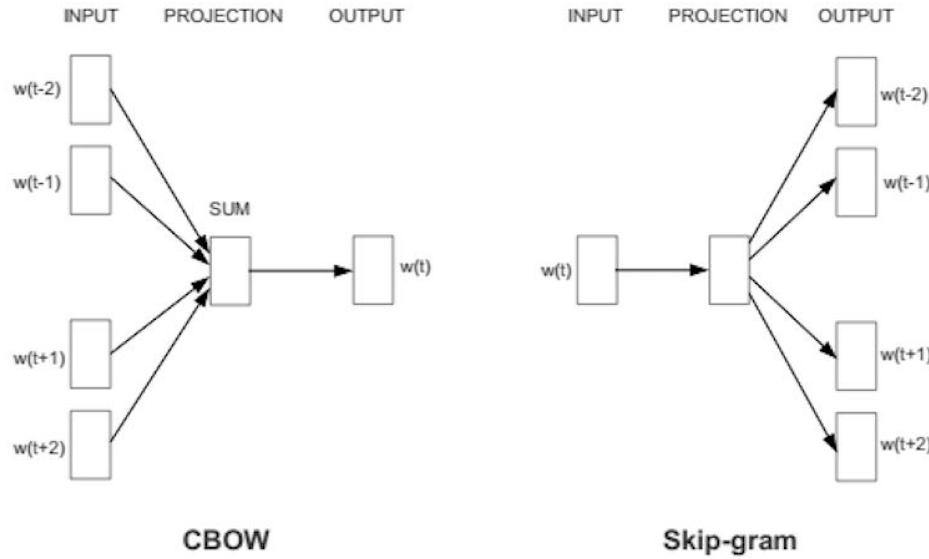


Figura 6.2: I due approcci Word2Vec

### 6.2.1 Continuous Bag of Words

Come già menzionato in precedenza, nell'approccio CBOW la rete neurale possiede tre livelli: il livello dell'input, quello dell'output ed il livello nascosto. Come input, CBOW usa il contesto riferito alla parola target nella frase in analisi. Il contesto di una parola viene selezionato in accordo con le impostazioni di quello che viene denominato “finestra contestuale”: dato un valore  $l$  che rappresenta la lunghezza della finestra e data una parola target  $x$ , il contesto sarà dato dalle  $l$  parole più a destra e dalle  $l$  parole più a sinistra di  $x$ . Dunque, dalla frase data come esempio precedentemente e ponendo una finestra contestuale di lunghezza  $l = 1$  si ottiene:

The quick brown fox jumps over the lazy dog

In cui le parole in verde sono il contesto, mentre la parola evidenziata in giallo rappresenta la parola target. Nella Tabella 6.1 viene invece indicato il vocabolario per la frase esempio.

Tabella 6.1: Vocabolario

brown	dog	fox	jumps	lazy	over	quick	the
-------	-----	-----	-------	------	------	-------	-----

Ogni parola viene rappresentata da un vettore *1-to-N (one-hot)*: quindi, in presenza di un vocabolario con otto parole differenti (come nell'esempio), ognuna di esse verrà rappresentata da un vettore di lunghezza uguale a otto, dove solamente in

una posizione verrà posto il valore “1”, quella che appunto rappresenta la posizione della parola in questione; in tutte le altre posizioni verrà indicato il valore “0”. Tale rappresentazione può essere ricavata semplicemente attraverso l’indicizzazione delle parole, processo mediante il quale ogni parola viene rappresentata da un numero costituisce anche il suo indice nel vettore del vocabolario, ad esempio “1- *brown*, 2- *dog*, 3- *fox*”, ecc. Solitamente, il vocabolario è ordinato in accordo con la frequenza con la quale le parole compaiono nel corpus; ciò vuol dire che se la parola  $x$  compare più frequentemente rispetto alla parola  $y$ , allora  $x$  verrà trovata prima della parola  $y$  nel vocabolario.

Come output si vuole invece ottenere la parola target, che viene rappresentata anche attraverso un vettore; l’obiettivo è dunque quello di ottenere un vettore che si somigli il più possibile a quello della parola target, massimizzando la probabilità condizionale di osservare l’output desiderato dato un determinato input. Risulta quindi necessario fornire come input alla rete ciò che ci si aspetta implicitamente. Essendo *Word2Vec* una RNA simile ad un *autoencoder*, i valori forniti come input ci permetteranno in un secondo momento di calcolare l’errore di previsione dell’output e, attraverso l’algoritmo *backpropagation*, di aggiustare i pesi. Non ci si aspetta però che l’output sia simile ad un “valore atteso” poiché tale l’approccio non è supervisionato: l’output è un vettore con le rappresentazioni di probabilità condizionate, in cui ogni indice rappresenta una parola e dove ciascuna posizione è riempita con la probabilità di rintracciare tale parola dato un determinato contesto (o vice-versa nel caso di Skip-gram 6.2.2). Per questo motivo, nell’esempio precedente, si intende massimizzare le probabilità per la parola “*fox*” in seguito all’osservazione dei suoi contesti. Questo viene attuato attraverso l’aggiornamento dei componenti nella fase di *training* della rete. Nella Figura 6.3 viene illustrato l’approccio CBOW.

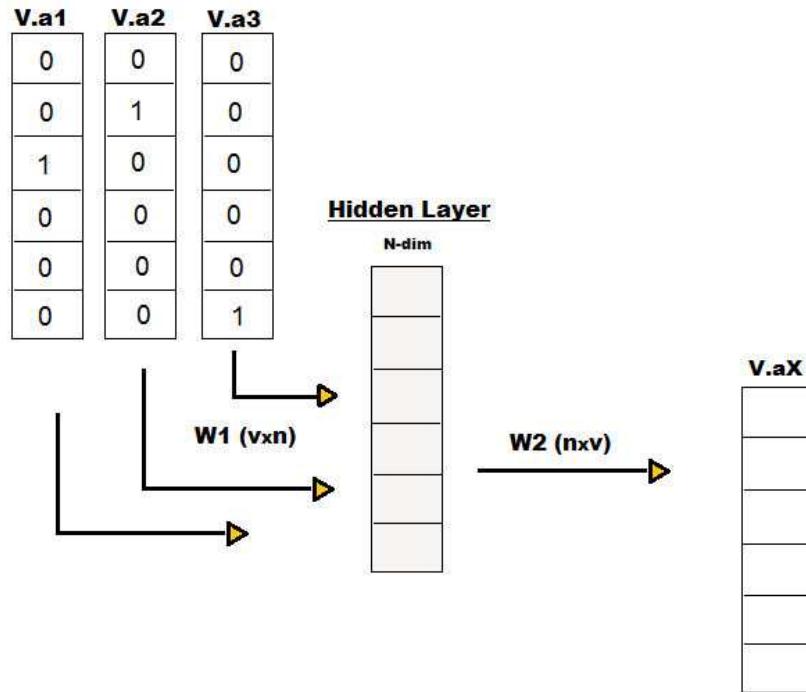


Figura 6.3: L'approccio CBOW

Avendo  $k$  vettori d'input, ossia i  $k$  indici che raffigurano le parole input, la funzione di attivazione per il livello nascosto equivale semplicemente alla somma delle linee di  $W1$  che sono equivalenti agli indici delle parole nel vocabolario e alla divisione di tale somma per  $k$  (il che suggerisce che il passaggio dall'input al livello nascosto attraverso la matrice  $W1$  corrisponda ad una tabella di *lookup*, cfr. figura 6.4). In questo modo si ottiene la media; la funzione di attivazione risulta dunque banalmente lineare. La seconda matrice dei pesi ( $W2$ ) può invece essere utilizzata, ad esempio, per calcolare lo score di ogni parola nel vocabolario.

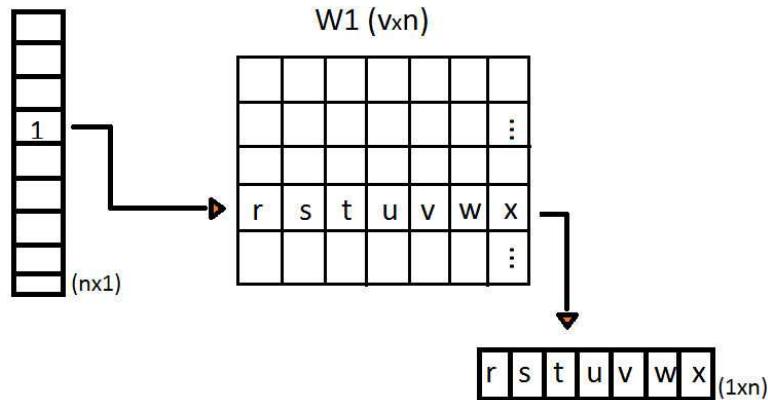


Figura 6.4: Lookup table

### 6.2.2 Skip-gram

L'approccio Skip-gram è, in poche parole, il contrario dell'approccio CBOW. Si riceve come input un unico vettore *1-to-N*, ossia l'indice della parola nel vocabolario, che rappresenta la parola target. Come output, ci si aspetta di ottenere vettori in cui le parole che rappresentano il contesto della parola target abbiano una maggiore probabilità, presupponendo di ottenere il contesto “giusto” data una determinata parola target. L'obiettivo, dunque, diviene quello di minimizzare l'errore di previsione sommata per tutte le parole nel livello di output, il che significa massimizzare la classificazione di una parola in base ad altre presenti nella stessa frase (viene usato un *log-linear classifier* per prevedere le parole ad un certo *range*, precedenti e successive rispetto alla parola target) [56].

Secondo [56], quanto più distante è il range contestuale considerato, minore sarà il rapporto con la parola target, spiegando così la loro affermazione: “più lunga è la finestra contestuale, più accurato sarà il risultato (ma anche più costoso)”.

Se fossero attribuiti dei pesi ad ogni parola vicina alla parola target in un range  $= l$ , tanto per la destra quanto per la sinistra (con  $l =$  lunghezza della finestra), allora i pesi più bassi sarebbero attribuiti alle parole più distanti dalla parola target. Le parole molto frequenti, trovate spesso insieme in contesti differenti, vengono inoltre considerate parole il cui significato è di minor importanza (ad esempio quelle classificate spesso come *stop words*: “the”, “and”, “with”, ecc.). Esse verranno utilizzate nell'approccio *Negative-sampling*, che sarà descritto nella sezione 6.2.2.

In [58] viene attribuito all'approccio Skip-gram l'obiettivo di massimizzare la media della probabilità logaritmica dei contesti in base alle parole fornite come input, ossia le parole target. Data  $\{w_1, w_2, w_3, \dots, w_T\}$  una sequenza di parole, la massima media della probabilità logaritmica viene tradotta come:

$$\frac{1}{T} \sum_{j=0}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j}|w_t) \quad (6.1)$$

Dove  $-c, c$  sono i limiti della finestra contestuale,  $j$  rappresenta l'indice del contesto che viene spostato nel range della finestra contestuale,  $t$  è l'indice della parola target e infine  $T$  è la lunghezza della sequenza presa in analisi.

Il punto chiave della formula 6.1 gira attorno al calcolo della probabilità condizionata  $p(w_{t+j}|w_t)$ : per fare ciò sono stati proposti alcuni metodi, come la funzione *Softmax* semplice, l'*Hierarchical Softmax* e il *Negative Sampling* [58], [19].

In accordo con [58], aggiornare i pesi per ogni contesto nel livello di output risulta eccessivamente costoso in ottica computazionale. In effetti, il costo del modello dipende delle dimensioni del vocabolario, che possono essere anche molto elevate. Per venire incontro a tale problematica, gli autori hanno realizzato un confronto tra i metodi sopramenzionati, che verranno brevemente descritti in questa tesi per scopi didattici.

Nella figura 6.5, viene illustrato il modello Skip-gram, composto da tre livelli distinti, il livello d'input, che riceve la parola target, il livello nascosto, che realizza la proiezione ed il livello di output, che restituisce le rappresentazioni di probabilità condizionate in uno spazio vettoriale per i contesti della parola data come input.

Se si osserva la formula 6.1 tradotta in termini computazionali, per ogni parola target esiste un'iterazione che va a calcolare la probabilità esistente per ogni contesto. È bene ricordare come in realtà i dati iniziali non siano nient'altro che un corpus testuale, con il quale si lavora per la costruzione di ciò che serve come input al modello. Risulta innanzitutto necessario ottenere il vocabolario attraverso l'analisi del corpus; il passo successivo è quello di selezionare una lunghezza per il *batch* con cui si intende lavorare ed infine, suddividere le coppie di parole vs. contesto. Dato il solito esempio:

The quick brown fox jumps over the lazy dog

Ciò che si vuole ottenere sono le coppie  $(fox, brow)$ ,  $(fox, jumps)$ , sempre se la finestra contestuale ha lunghezza  $l = 1$ , il che vuol dire che si sta analizzando una sola parola come contesto destro ed una sola come contesto sinistro (vedi figura 6.6). La prima sommatoria della formula 6.1 significa che, data la frase input sopra

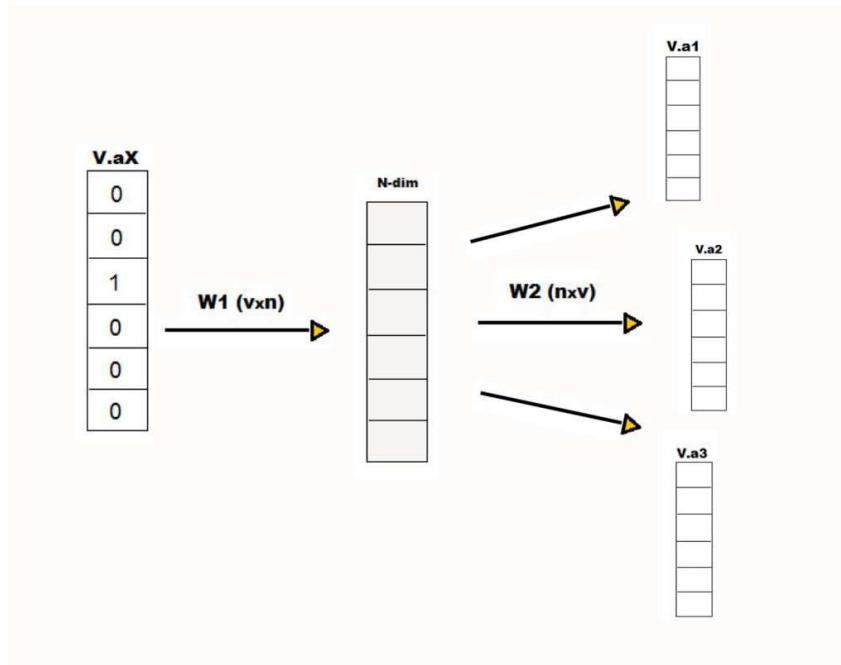


Figura 6.5: L'approccio Skip-gram

definita, ogni parola andrà ad occupare il posto di parola target per ogni iterazione durante l'elaborazione dell'input della rete, il che presuppone come, al termine del ciclo sia possibile ottenere un insieme di coppie (*target, context*).

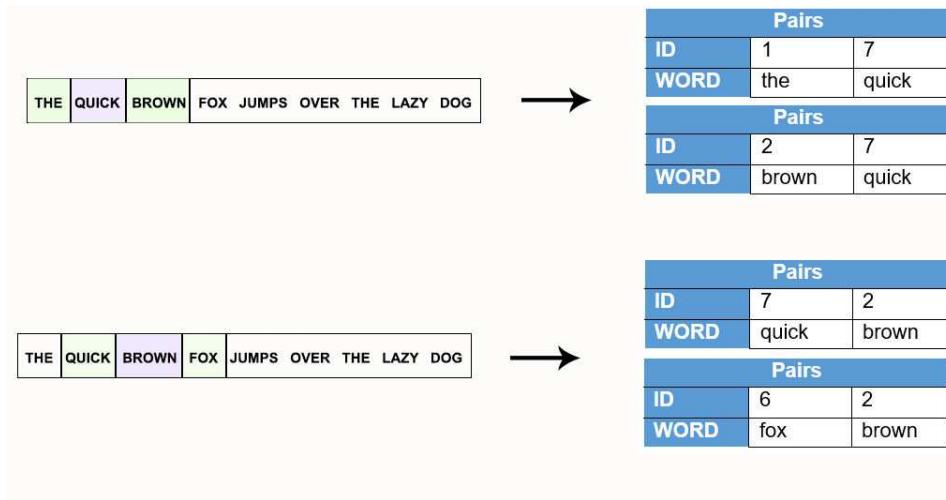


Figura 6.6: Input pairs

E' facile osservare che il costo nel computare la probabilità della formula 6.1 sia,

di fatto, proporzionale alla lunghezza del vocabolario [58], poiché per ogni coppia (*target, context*) appartenente alla j-ésima parola target ( $wt_j$ ), l'ultimo livello del modello deve computare il vettore finale contenente la distribuzione di probabilità voluta; in base all'approccio utilizzato, è dunque necessario effettuare tale passaggio per ogni neurone computando la loro rispettiva probabilità (cfr. figura 6.7) – si ricorda che la quantità di neuroni presenti nel livello di output è equivalente alla quantità di parole nel vocabolario.

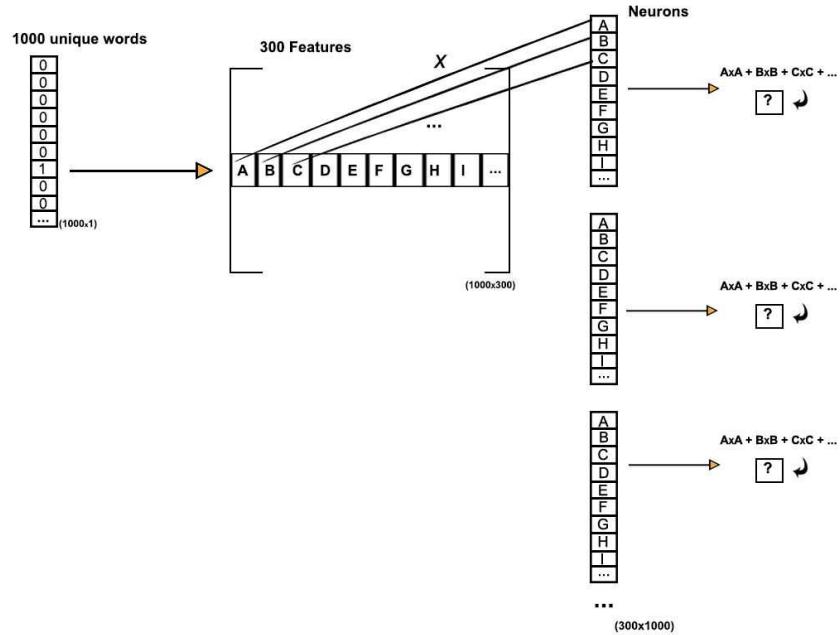


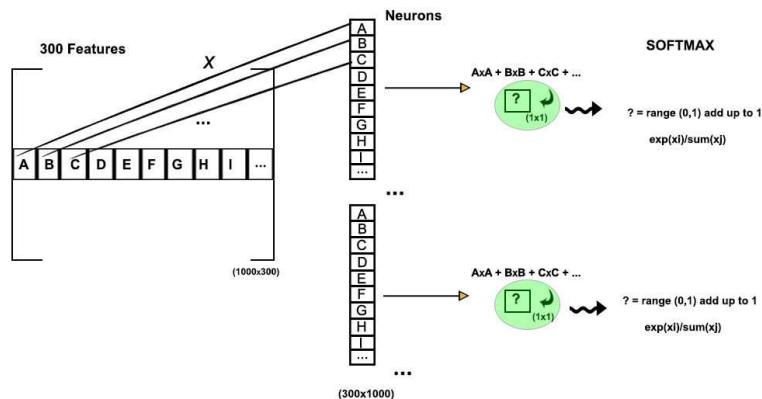
Figura 6.7: Output layer

## Softmax

Funzioni del tipo *Softmax* (SM) consentono di interpretare l'output della RNA come possibili probabilità per una variabile categorica in analisi. In questo modo, la funzione SM viene vista come una generalizzazione della funzione logistica, in cui l'obiettivo è quello di “appiattire” un vettore di  $N$  dimensioni di valori  $\in \mathbb{R}$  in un vettore  $N$ -dimensionale con valori  $\in (0,1)$ . Tali valori sono poi trasformati in “percentuali” (per tale ragione, in inglese si usa dire “*add up to 1*”). In un modo del tutto intuitivo, l'SM è la versione “soft” della funzione di massimizzazione, poiché tale funzione opera l'analisi dei valori di un vettore  $V$  di lunghezza  $N$  e distribuisce valori proporzionalmente rappresentativi in un altro vettore  $V'$  di lunghezza  $N$ ; ad esempio: il valore più alto di  $V$  viene rappresentato in  $V'$  come l'indice che prende il pezzo proporzionalmente maggiore tra tutti quelli distribuiti ad ogni indice del vettore  $V'$ . Sia  $x_i$  un input fornito ad una RNA tale che  $i \in [1, \dots, c]$ , dove  $c$  rappresenta il numero di categorie classificatorie. Il risultato di *Softmax* per un determinato neurone di output ( $o_i$ ) viene rappresentato come segue:

$$o_i = \frac{\exp(x_i)}{\sum_{j=1}^c \exp(x_j)} \quad (6.2)$$

Nel caso del modello *Word2Vec*, la probabilità restituita da Softmax viene calcolata per ogni neurone appartenente al livello di output, come mostrato dalla figura 6.8.



$*|*|*|*|*|*|*|*|*|*|*|*|*|*|*|*|*|*$

$$* = p(w_o | w_i)$$

Figura 6.8: Output layer - Softmax

Come affermato in precedenza, un classificatore *Softmax* è una generalizzazione della forma binaria della Regressione Logistica [65]. La funzione di costo (tradotta in inglese come *loss function*) ha infatti come obiettivo quello di minimizzare la probabilità logaritmica negativa di selezionare la classe corretta dato un determinato input, ne deriva che:

$$L_i = -\ln P(Y = y_i | X = x_i) \quad (6.3)$$

La probabilità nell'equazione 6.3 può essere interpretata nella seguente forma:

$$P(Y = y_i | X = x_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \quad (6.4)$$

Dove  $s = f(x_i, W)$  rappresenta la funzione di “scoring”, data come il prodotto scalare semplice tra l'input  $x$  e la matrice dei pesi  $W$ . L'esponenziale e la normalizzazione attraverso la somma degli esponenti costituiscono ciò che viene chiamato funzione *Softmax*.

In alcuni casi, come nell'uso della funzione SM in reti neurali artificiali, è necessario tener conto della derivazione della funzione; per tali ragioni si calcola solitamente la derivata parziale del  $i$ -ésimo output SM in rapporto con il  $j$ -ésimo valore di input:

$$\frac{\delta S_{O_i}}{\delta y_j} \quad (6.5)$$

Dove  $S_{O_i}$  corrisponde all' $i$ -ésimo output SM. Giacché SM mappa  $\mathbb{R}^N \rightarrow \mathbb{R}^N$ , la derivazione viene computata sopra la matrice di Jacobian, dove per ogni singolo elemento della matrice viene eseguito il seguente calcolo [19]:

$$\frac{\delta \left( \frac{e^{y_i}}{\sum_{k=1}^N e^{y_k}} \right)}{\delta y_j} \quad (6.6)$$

## Hierarchical Softmax

Attraverso l'uso di *Hierarchical Softmax* (HS), è possibile limitare il numero di vettori di output che saranno computati per il modello Skip-gram. L'HS fa uso di un albero binario per selezionare le parole che avranno i vettori aggiornati nel livello di output; dunque, invece di aggiustare i pesi per ogni neurone nell'ultimo livello, si impiega un albero binario, come quello illustrato nella figura 6.9. Le foglie dell'albero rappresentano le parole del vocabolario e per ogni parola, ossia per ogni foglia, esiste un unico cammino che parte dalla radice; esso viene usato per stimare la probabilità condizionata di individuare la parola rappresentata dalla foglia, dato un determinato input. La probabilità per ogni parola sarà quindi proporzionale alla lunghezza del cammino radice-foglia. Secondo [58], il vantaggio nell'uso dell'HS sta proprio nel costo computazionale, per il quale invece di valutare  $N$  parole per ottenere la

distribuzione di probabilità al livello di output ( $N$  = lunghezza del vocabolario), si valutano solamente  $\log_2(N)$ .

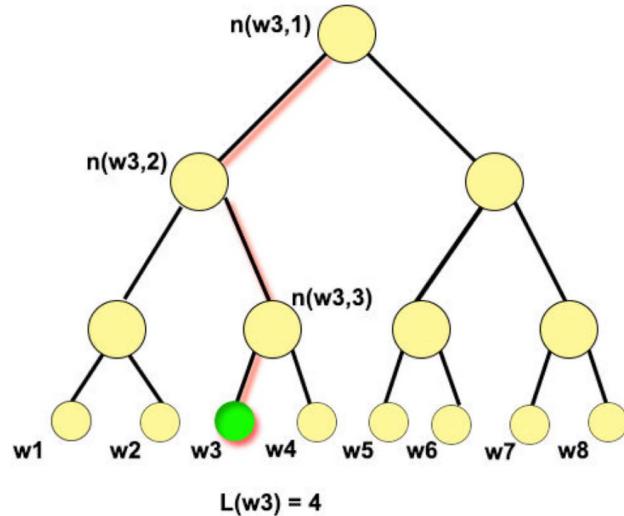


Figura 6.9: Hierarchical Softmax

Nell'immagine,  $n(w, j)$  rappresenta il  $j$ -ésimo nodo nel cammino che parte dalla radice fino alla parola (foglia)  $w$ , mentre  $L(w) = x$  corrisponde alla lunghezza del cammino che parte dalla radice fino alla foglia.

### Negative sampling

In [58] è stato proposto un nuovo metodo per l'aggiornamento dei pesi al livello di output, denominato *Negative Sampling* (NS). L'idea di NS è quella di aggiornare unicamente un campione di vettori dei pesi nel livello di output: l'output desiderato deve appartenere al campione e gli elementi rimasti possono essere scelti in base alla loro lontananza dal valore atteso; tale distanza dev'essere sufficientemente grande affinché essi vengano selezionati. In questo modo viene creata una distribuzione probabilistica, gli autori si basano, infatti, su *Noise Contrastive Estimation* (NCE), ovvero sull'affermazione che un buon modello dovrebbe essere capace di distinguere l'output atteso da ciò che viene denominato “rumore”, quei valori sicuramente non aspettati e quindi considerati esempi “negativi”. NS costituisce pertanto un approccio che lavora su un obiettivo diverso rispetto agli altri: è quello di distinguere la parola attesa dal cosiddetto “rumore”.

Secondo [58], viene aggiornato un valore aleatorio di pesi appartenenti al livello di output che può variare tra 2 e 5 per grandi dataset e tra 5 e 20 per piccoli dataset.

Supponendo essere in possesso di un dataset dove esistono circa 10.000 parole, per una data parola input in una certa iterazione si vuole scegliere un numero aleatorio di altre parole tra quelle dell'intero vocabolario, per cui i loro pesi verranno aggiornati. Tali parole vengono scelte nel seguente modo: si supponga di avere la coppia  $(w, c)$ , dove la parola “ $w$ ” è l'input e “ $c$ ” è un suo contesto; l'obiettivo è quello massimizzare la probabilità che l'output osservato appartenga al contesto della parola analizzata, perciò tali esempi negativi vengono scelti usando l’ “*unigram distribution*”. Per tale distribuzione viene considerata la frequenza delle parole nel corpus: più essa è alta, maggiore sarà la sua probabilità di essere scelta come esempio negativo. Ciò deriva dalla supposizione che parole frequenti in diversi contesti non corrispondano alla descrizione dei significati delle stesse. Nell'equazione 6.7 viene dimostrato il calcolo della probabilità che servirà come base per la scelta del campione di esempi negativi.

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^n (f(w_j)^{\frac{3}{4}})} \quad (6.7)$$

In [19] vengono evidenziati i seguenti punti:

- $f(w_i)$  è la frequenza con la quale la parola  $w_i$  compare nel corpus mentre  $\frac{3}{4}$  rappresenta una scelta empirica compiuta dagli autori di [58];
- $f(w_i)^{\frac{3}{4}}$  è quindi il peso attribuito alla parola  $w_i$ ;

- Ogni contesto costituisce anche una parola target così come ogni parola target è anche un contesto.
- L'implementazione dell'*unigram distribution* nel linguaggio di programmazione C viene attuata per mezzo della creazione di un grande vettore, in cui l'inserimento della parola  $w_i$  viene fatto  $P(w_i) \times N$ , dove  $N$  è la lunghezza del vettore; in questo modo, scegliendo un indice aleatorio del vettore, le parole che compaiono più volte hanno maggiore probabilità di essere selezionate.

Una volta completato tale passaggio, viene calcolata la funzione obiettivo, ossia la massimizzazione della probabilità che la parola osservata non sia un “rumore” (per ulteriori approfondimenti cfr. [19]).

## 6.3 Gli ambiti di applicazione Word2Vec

In questa sezione verranno riportati alcuni progetti di applicazione di *Word2Vec* rintracciati in letteratura, che impiegano la tecnica *word embeddings*.

*Word2Vec* può essere usato per costruire dizionari specifici per un determinato dominio, come nel caso di [87]: gli autori hanno scaricato circa 2 milioni di “*Weibos*” (un social network simile a *Twitter* molto diffuso in Cina) e hanno sviluppato un sistema che applica *Word2Vec* su un dataset di emozioni già noto, al fine di trovare ulteriori sentimenti associati a quelli già esistenti. Per ogni possibile coppia di parole, è stata calcolata la distanza del coseno dai suoi vettori. Per esempio, data una parola  $w$ , sono state selezionate le 40 parole più vicine semanticamente ad essa. Un altro lavoro analogo è stato il [17], in cui il dominio del problema cambia: un sistema che fa uso di *Word2Vec* ha realizzato un confronto di termini appartenenti ad un’ontologia di geni per trovare quelli che sono più vicini a certe proteine; essendo i termini in questo caso degli insiemi di parole, gli autori hanno deciso di utilizzare la *Hausdorff distance* per misurare la distanza tra due insiemi.

In [74] è stato studiato l'impatto di *Word2Vec* sull'ambito del *Named Entity Recognition*, riscontrando come la principale motivazione nell'impiego di tale tecnica sia fornita dall'aggiunta di ulteriori informazioni allo scopo di classificazione. Il complemento di informazione ha rappresentato anche la motivazione per [61], dove lo scopo di *Word2Vec* era quello di sostituire la funzione di un'ontologia, trovando il collegamento tra le parole chiave vere e altre simili, in relazione con esse.

*Word2Vec* può inoltre essere utilizzato in altri ambiti di ricerca, non solamente testuali ma anche, ad esempio, visuali: in [38], ad esempio, gli autori hanno usato *Word2Vec* per catturare le relazioni semantiche implicite in un “mondo visuale”.

Infine, un’interessante ricerca è stata quella presentata in [48], dove gli autori hanno proposto alcuni miglioramenti ai modelli Skip-Gram e CBOW, aventi come

obiettivo quello di adattare *Word2Vec* anche alle funzioni basate sulla sintassi delle parole, come il *Part-of-Speech Tagging* e il *Dependency parsing*.

## 6.4 Epilogo

In questo capitolo sono stati introdotti i *word embedding*. In particolare è stato approfondito il metodo *Word2Vec*, esponendo i suoi principali obiettivi, fondamenti ed i relativi approcci per la costruzione del modello: *CBOW* e *Skip-gram*. Essi sono stati confrontati e approfonditi, descrivendo nel dettaglio soprattutto l'approccio *Skip-gram*. Per concludere, sono stati analizzati i diversi ambiti di applicazione di tale metodologia.

# Capitolo 7

## Bidirectional LSTM

Le *Bidirectional LSTM* sono specializzazioni delle reti LSTM, che a loro volta consistono in un'architettura specifica per le reti neurali ricorrenti (RNN). Le LSTM sono progettate per l'elaborazione di dati sequenziali e per la gestione delle loro dipendenze a lungo termine, in modo più accurato rispetto alle convenzionali RNN. Per certi versi, l'uso delle RNN tradizionali può rappresentare la scelta più adatta ma per altri, sarebbe più consigliato ricorrere alle LSTM. In questo capitolo saranno introdotte le reti neurali ricorrenti, le loro variazioni e le motivazioni legate alla scelta della struttura più adeguata ad ogni tipo di problema.

### 7.1 Le Reti Neurali Ricorrenti

Le reti neurali ricorrenti (RNN) sono una tipologia di RNA che prende come input non solo il dato in ingresso attuale, ovvero l'informazione corrente, ma anche l'entrata ricevuta in un tempo precedente. Questo tipo di RNA include cicli nel *processing* dell'informazione, permettendo al modello di memorizzare i dati già presi in analisi nel momento in cui nuovi input sono elaborati (cfr. figura 7.1).

Per poter calcolare l'output in un determinato momento è dunque necessario possedere anche l'informazione dell'input precedente; la decisione al momento  $t$  dipenderà quindi dal momento  $t - 1$  mentre la decisione al momento  $t + 1$  dipenderà dal momento  $t$ .

Comunemente si afferma come le reti neurali ricorrenti siano dotate di una sorta di “memoria”. A partire da un insieme di entrate, infatti, il dato sequenziale viene mantenuto attraverso uno stato nascosto, in grado di processare l'informazione tramite una funzione che calcola l'output.

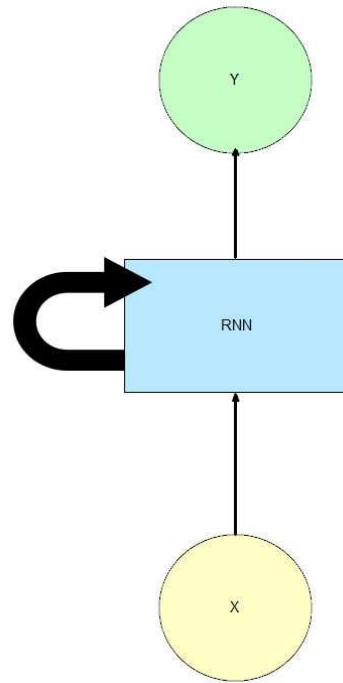


Figura 7.1: Rete Neurale Ricorrente

Dato quindi un vettore di informazioni di entrata  $\hat{x}$ , una sequenza di  $\hat{x}$  viene calcolata applicando la funzione di ricorrenza (riportata in seguito) per ogni passo temporale:

$$\textcolor{red}{h}_t = \textcolor{blue}{f}_W(\textcolor{brown}{h}_{t-1}, \hat{x}_t) \quad (7.1)$$

Tale funzione di ricorrenza  $\textcolor{blue}{f}_W$  utilizza lo stesso insieme di parametri per passo temporale, così come gli stessi pesi  $W$  vengono usati per processare le informazioni di entrata per ogni passo temporale;  $h$  ad indicare lo stato nascosto, che nella versione di base (“vanilla”) delle reti neurali ricorrenti costituisce un unico stato,  $\textcolor{red}{h}_t$  rappresenta lo stato attuale e  $\textcolor{brown}{h}_{t-1}$  lo stato precedente.

Questo tipo di rete è indicato per la risoluzione di problemi di natura sequenziale, dove gli input ricevuti nello step precedente devono essere presi in considerazione al fine di computare la risposta desiderata (ad es.: le serie temporali). Per questa ragione i modelli matematici architettonici basati su (*deep learning*) sono spesso incentrati sulle RNN.

Un esempio di applicazione delle RNN è dato da un sistema per il *processing* del linguaggio naturale in cui lettere o parole rappresentano i dati d’input della rete.

Quando si intende considerare le parole, le lettere un elemento importanti per il contesto; quando invece, esse appaiono isolate, non forniscono un grande utilità. In questo ambito, ad esempio, l'uso di RNN risulta indicato poiché in grado di supportare il comportamento temporale.

Le RNN non sono caratterizzate come un'unica classe bensì come un insieme di topologie di reti che vengono applicate a problemi diversi, siano essi della stessa natura o di natura simile; i tipi più comuni sono:

- Hopfield: non adeguate per problemi temporali ma comunque di natura ricorrente [47], [83];
- Semplice: ad esempio Elman e Jordan [37], [91];
- LSTM: fa uso della memoria a breve e lungo termine per l'apprendimento profondo [32];
- GRU: una variazione dell'LSTM che impiega un numero minore di risorse [34].

Le RNA tradizionali possiedono algoritmi specifici per la fase di *training*, che gli consentono di mantenere lo storico delle informazioni sequenziali o temporali. Una tecnica popolare per l'addestramento della rete è la *backpropagation through time* (BPTT) [85]: gli aggiornamenti dei pesi della rete vengono compiuti attraverso la somma di tutti gli aggiustamenti degli errori accumulati per ciascun elemento della sequenza. Questa tecnica può causare la scomparsa o “l'esplosione” dei pesi, qualora vi siano grandi sequenze di entrata. Questi problemi vengono denominati *vanishing problem* e *exploding problem* e fanno parte delle principali motivazioni alla base dell'utilizzo delle LSTM; il primo problema verrà descritto ulteriormente con maggiori dettagli.

## 7.2 Long-short term memory

L'RNN *Long-short term memory* (LSTM) è stata introdotta da Hochreiter nel 1997 [32] allo scopo di affrontare i problemi incontrati nel momento in cui si fa uso di metodi per l'apprendimento basati sul gradiente e sulla propagazione all'indietro attraverso il tempo. Con l'uso delle LSTM è possibile ridurre i tempi di apprendimento e avere un flusso di errore costante, grazie all'introduzione di unità speciali contenenti caratteristiche aggiuntive, risultando così più adeguate all'elaborazione di lunghe sequenze di dati rispetto al modello architettonale convenzionale RNN.

### Il Vanishing Problem

Il *Vanishing Problem* è un tipo di problema che può verificarsi quando si fa uso di reti neurali basate sull'apprendimento dei gradienti dei parametri della rete – ad

es. la propagazione all'indietro – assieme al *deep learning* (il *deep learning* è un tipo di apprendimento per il quale l'architettura di rete solitamente possiede più di un livello nascosto; per ulteriori approfondimenti vedi la sezione 2.3). L'obiettivo dell'uso di metodi basati sull'apprendimento dei gradienti è quello di comprendere come piccoli cambiamenti nei valori dei parametri della rete possano influenzare il suo output, ossia verso quale direzione è necessario “spostarsi”, attuando una piccola alterazione nei parametri (cfr. fig 7.2). Nel caso in cui il cambiamento del parametro causi una modifica relativamente piccola nell'output, la rete semplicemente non ha imparato il parametro.

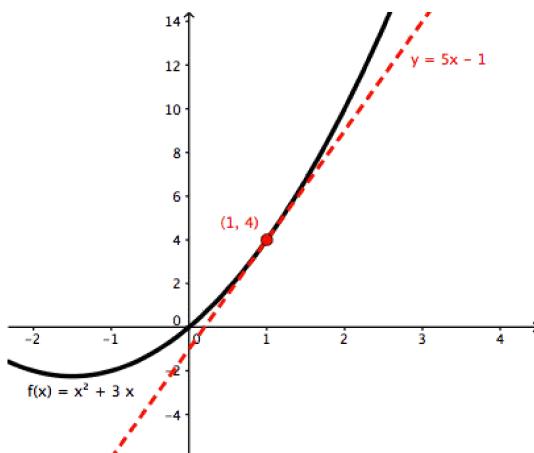


Figura 7.2: Esempio derivativa

Quando si trova di fronte ad una rete con diversi livelli nascosti, per i quali è necessario calcolare i gradienti dell'output in rapporto con i parametri della rete, essi cominciano a diventare molto piccoli; se si decide di applicare una cospicua alterazione nei valori dei parametri dei livelli precedenti, l'output non si modificherà di molto, generando un comportamento non desiderato. Tale fenomeno si chiama appunto *Vanishing problem*. Solitamente esso può essere causato dalla scelta delle funzioni di attivazione, poiché le più comuni, come ad esempio la Sigmoid o la Tangente iperbolica, “schiacciano” i valori a loro passati, producendo un output appartenente ad un piccolo range compreso tra  $[0, 1]$ . Per questo motivo, ampie regioni dello spazio dell'input vengono mappate in un range stretto, provocando l'effetto descritto in precedenza, della non effettiva varianza dell'output in risposta a un grande cambiamento dell'input. È facile notare come il problema divenga maggiore ad ogni livello aggiunto; in una rete neurale di tipo ricorrente è perciò comune avere questo tipo di problema quando si deve far fronte ad input che esigono la memorizzazione di lunghi periodi di tempo (ovvero tanti *time-step*).

Per affrontare questo tipo di problema si può utilizzare l'LSTM. La principale differenza tra le reti neurali ricorrenti convenzionali e le LSTM è che l'RNN memo-

rizza tutti i parametri dei *time-step* precedenti quando “attraversa” la rete, portando con sé tutto in avanti in un passaggio. L’LSTM, invece, riesce a selezionare ciò che, secondo determinati controlli, appare sufficientemente rilevante per essere portato in avanti attraverso la rete. Questo viene fatto per mezzo dell’aggiunta di “porte” (*gates*) che controllano ciò che dev’essere dimenticato, aggiornato ed aggiunto in un nuovo parametro usato per la rete: la cellula di memoria. In una RNN convenzionale (o *Vanilla RNN* in termini tecnici) – come descritto in precedenza – sono previsti tre passaggi principali, compiuti attraverso i livelli della rete: *input-hidden*, *hidden-hidden*, *hidden-output*, in cui *hidden-hidden* introduce il concetto di tempo. L’LSTM invece, controlla ogni passaggio rispettivamente con le porte *input*, *forget* e *output*: tramite alcune funzioni che verranno descritte successivamente, la rete controlla ciò che viene rimosso, aggiornato o aggiunto alla cellula di memoria, valutando e verificando le modalità di passaggio dell’informazione.

In 7.3 e 7.4 vengono riportati alcuni schemi visivi volti a facilitare l’interpretazione dei passaggi di una LSTM (schemi basati su [62]). Vengono introdotte quindi nuove porte, insieme al concetto di cellula di memoria, che prevede il salvataggio delle informazioni ritenute opportune, l’aggiornamento di ciò che è necessario e la cancellazione di tutto quello che non risulta interessante. L’output finale è basato sulla cellula di memoria. Le porte (*gates*) sono in pratica dei livelli con una funzione di attivazione, in base alla quale l’output descrive quanto di ogni componente dovrebbe essere portato in avanti. Formalmente, le LSTM si basano sulle seguenti funzioni:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (7.2)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (7.3)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (7.4)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (7.5)$$

$$h_t = o_t \tanh(c_t) \quad (7.6)$$

Dove  $x_t$ ,  $h_t$  e  $c_t$  denotano rispettivamente le sequenze dei vettori di input, input precedente (*hidden*) e cellula di memoria;  $W$  indica la matrice dei pesi (ad esempio,  $W_{xi}$  è la matrice dei pesi che collega il livello di input con il livello di *input gate*),  $b$  rappresenta il vettore bias,  $\sigma$  è la funzione sigmoide logistica e  $i$ ,  $f$  ed  $o$  rappresentano rispettivamente l’*input gate*, il *forget gate* e l’*output gate* [49].

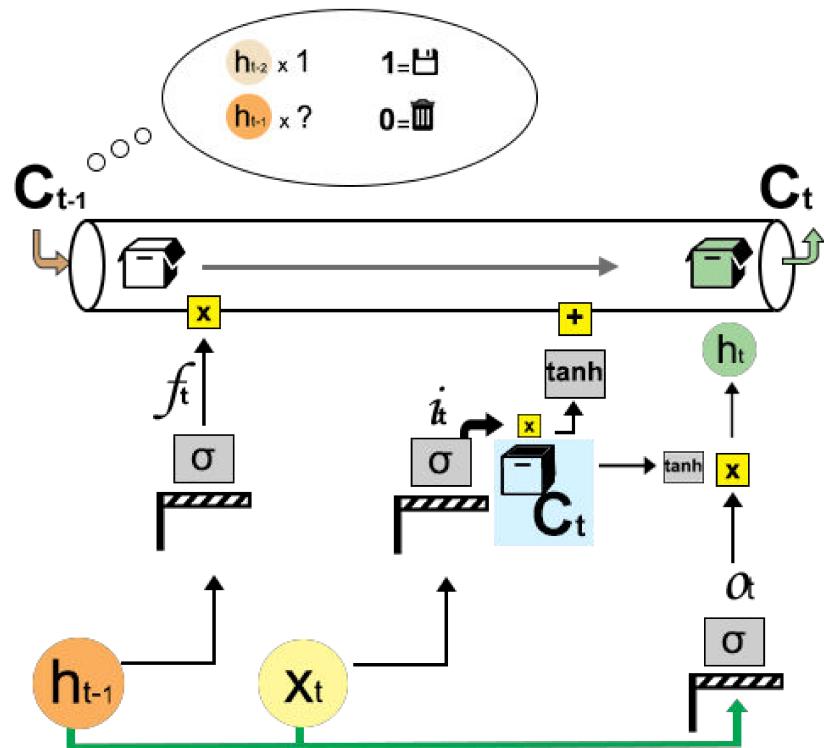


Figura 7.3: Schema - gates

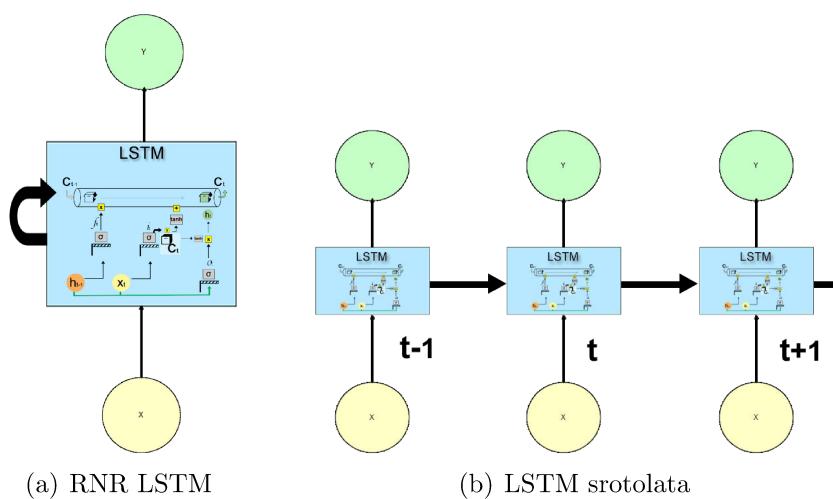


Figura 7.4: LSTM

In [7.1](#) viene riportato lo pseudo-algoritmo (*step-by-step*) del passaggio *forward* di una LSTM.

Algorithm 7.1: Long-short term memory

```

1 >> Forget gate layer: controlla l'input  $x(t)$ , i valori
2     del livello nascosto precedente  $h(t-1)$  e restituisce
3     un valore tra 0 e 1 per ogni valore in  $C$ .
4
5 >> Si decidono i dati che saranno memorizzati:
6     1. Input gate layer: decide i valori che saranno
7         aggiornati:
8
9     2. Tanh layer:  $C'(t)$  = crea i nuovi valori
10        candidati:
11
12    > $C'(t) = \tanh(W(c) \cdot [h(t-1), x(t)] + b(c))$ 
13
14 >> Aggiorna la vecchia cellula di memoria:
15     1. Moltiplico  $C(t-1)$  per  $f(t)$ : il che significa
16         dimenticare cio' che ho deciso di dimenticare.
17     2. Add " $i(t) * C'(t)$ ": il che significa
18         aggiungere la quantita' scelta delle nuove
19         informazioni (update):
20              $C(t) = f(t) * C(t-1) + i(t) * C'(t)$ 
21
22 >> Output gate layer:
23     1. Decido quali valori della cellula portero'
24         come risultato:
25
26     > $o(t) = \text{sig}(W(o) \cdot [h(t-1), x(t)] + b(o))$ 
27
28     2. Calcolo la tanh della cellula di memoria e
29         moltiplico per l'output:
30
31     > $h(t) = o(t) * \tanh(C(t)).$ 
```

### 7.3 Bidirectional Long Short-term memory

Le *Bidirectional Recurrent Neural Network* (BRNN) sono state introdotte da Schuster e Paliwal nel 1997; l’idea chiave di tale architettura è quella di suddividere l’unico stato di una rete neurale ricorrente in due parti: lo stato in avanti (*forward*) e quello all’indietro (*backward*) [71]. Nel 2005 sono state presentate le *Bidirectional Long Short-term memory Recurrent Neural Network* (BLSTM): un compromesso tra le BRNN e le LSTM. Le BLSTM aggiungono alle LSTM due nuovi livelli indipendenti, aventi come obiettivo quello di accumulare informazione contestuale, sia del contesto passato che del contesto futuro (da ciò deriva il nome “bidirezionale”) [22].

Nell’ambito dell’extrazione automatica di frasi chiave da documenti testuali, un intero testo viene solitamente fornito al sistema come input e, dopo una serie di modifiche (chiamate “*pre-processing*”), i dati sono impiegati allo scopo di estrarre parole e frasi chiave. È perciò possibile affermare come, per quanto riguarda l’uso di RNA nell’ambito soprammenzionato, la rete abbia accesso all’informazione sia passata che futura di un determinato istante del tempo; dato il tempo  $t$ , è dunque possibile avere accesso sia alle informazioni dell’input del tempo  $t - 1$  che del tempo  $t + 1$ ,  $t + 2$ , ecc.

Come già accennato precedentemente, le reti neurali ricorrenti LSTM sono capaci di valutare quali informazioni mantenere nella cellula di memoria dato un determinato tempo  $t$ . Tale informazione viene poi trasferita al passo successivo, che a sua volta valuterà l’importanza di mantenere o meno l’informazione. Quello che occorre nel *processing* di una LSTM è dunque la valutazione dell’informazione passata come criterio di uscita, ossia la valutazione dei dati precedenti per generare l’output.

Essere in possesso dell’informazione contestuale è difatti molto importante quando il proposito della ricerca riguarda l’extrazione automatica di frasi chiave a partire da un qualsiasi testo; quando si fa riferimento al contesto, non si pensa solamente a quello passato ma anche a quello futuro, di pari importanza. Infatti se l’obiettivo di un sistema qualsiasi fosse quello di indovinare quale sarebbe la prossima parola in un testo in base alla parola precedente, in alcuni casi conoscere unicamente il contesto passato non restituirebbe la risposta esatta a questo tipo di richiesta. A titolo di esempio, se l’input fosse un testo dove si fa presente la seguente frase:

“Io parlo molto bene la lingua ? perché ho vissuto due anni a Londra”

Quando arriva il momento di indovinare quale parola viene dopo “lingua”, per poter ottenere una risposta precisa ed esatta è necessario conoscere anche il contesto futuro, in questo caso il luogo in cui il soggetto della frase ha vissuto, ovvero la parola “Londra” al fine di poter dedurre inglese piuttosto che italiano o portoghese, ad esempio. In [5], viene sottolineata l’importanza del contesto futuro in un testo affermando il valore di riconoscere le anafore presenti. Sia  $F$  la seguente frase:

$F$  = “John Doe is a lawyer; he likes fast cars”

Una volta trovato “John Doe” nella frase, non è ancora possibile determinare se esso rappresenti o meno un’entità importante; il pronome “he”, presente nella seconda parte della frase, sottolinea chiaramente l’importanza di “John Doe in  $F$ , una volta che la sua rilevanza viene accentuata nel contesto. Le tecniche per trovare suddette informazioni contestuali vengono chiamate *anaphora resolution*.

Al fine di utilizzare il contesto futuro, si è scelto di adottare le BLSTM. Attraverso l’uso di tale architettura è infatti possibile impiegare entrambi i contesti, passato e futuro, di una specifica parola. Essa consiste di due livelli nascosti e separati, dove il primo calcola la sequenza di entrata dal tempo precedente in avanti  $\vec{h}_t$ , mentre il secondo calcola la sequenza all’indietro  $\overleftarrow{h}_t$ ; i risultati ottenuti vengono infine combinati per generare l’output  $y_t$ . Siano gli stati  $h$  due blocchi LSTM, allora una BLSTM viene implementata per mezzo delle seguenti funzioni:

$$\vec{h}_t = H(W_{x\vec{h}} \vec{x}_t + W_{\vec{h}\vec{h}} \vec{h}_{t-1} + b_{\vec{h}}) \quad (7.7)$$

$$\overleftarrow{h}_t = H(W_{x\overleftarrow{h}} \overleftarrow{x}_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}}) \quad (7.8)$$

$$y_t = W_{\overrightarrow{h}y} \vec{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y \quad (7.9)$$

Dove  $x_t$  e  $y_t$  denotano rispettivamente le sequenze dei vettori di input e output;  $W$  indica la matrice dei pesi,  $b$  rappresenta il vettore bias e  $\vec{h}_t$  e  $\overleftarrow{h}_t$  sono già stati definiti in precedenza. Nella figura 7.5 viene illustrato il modello BLSTM.

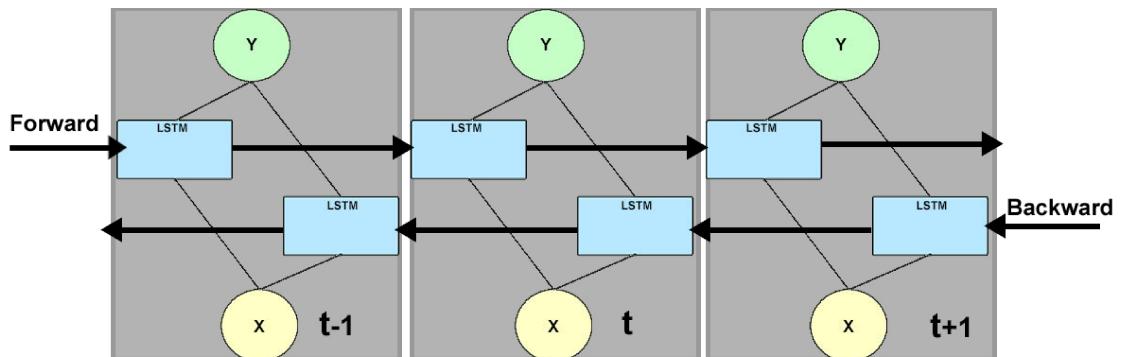


Figura 7.5: BLSTM

## 7.4 Epilogo

In questo capitolo sono state descritte le reti neurali ricorrenti (RNN), le *Long short-term memory* (LSTM) e le *Bidirectional Long short-term memory* (BLSTM). La prima parte del capitolo presenta le caratteristiche principali delle RNN e spiega per quale motivo il loro utilizzo risulti interessante per l'ambito dell'estrazione automatica di parole e frasi chiave. La scelta delle LSTM è stata poi giustificata, attraverso soprattutto la descrizione del *Vanishing problem*: noto come problema frequente in architetture a più livelli che impiegano l'algoritmo *backpropagation through time* per la ponderazione e l'aggiornamento dei pesi. Infine, sono state introdotte le BLSTM: specializzazione delle LSTM capaci di utilizzare sia il contesto passato che quello futuro nell'elaborazione di dati di tipo sequenziale.

# Capitolo 8

## Strumenti

In questo capitolo sono stati brevemente descritti gli strumenti impiegati per la parte pratica svolta in questa tesi. Alcuni esempi sono riportati solamente a titolo esplicativo; sono state inoltre elencate le principali tecniche, gli strumenti di supporto e le tecnologie hardware usate.

### 8.1 L'impostazione dell'ambiente

Al fine di implementare l'approccio proposto in questa tesi, si è scelto di fare ricorso ad'API<sup>1</sup> **Keras** per le RNA. Implementato in **Python**, **Keras** è un'API elaborata per supportare il rapido sviluppo e facilitare il processo di *testing*; esso consente di ridurre il tempo dedicato all'implementazione, permettendo al ricercatore di prestare maggiore attenzione alla progettazione dell'architettura della rete. Grazie alla sua facilità d'uso, **Keras** è adatto non solo ai principianti nell'area di *Machine Learning* ma anche a tutti coloro che necessitano di supporto per lo sviluppo di tecniche di *Deep learning*, in quanto supporta le reti neurali ricorrenti e convoluzionali [13]. Tra le diverse librerie che appoggiano lo sviluppo del *Deep learning*, le principali sono: **TensorFlow**, **Theano**, **Torch** e **Caffe**. **Keras** funziona come frontend in **TensorFlow** o **Theano**.

In questa tesi si utilizzerà **Theano** come backend per **Keras**, una *library* di computazione simbolica per il *Machine Learning* gestita dal gruppo *Montréal University* [8]. Essa ottimizza le espressioni matematiche e le compila per il codice nativo, consentendo anche l'esecuzione attraverso la GPU. Grazie alla facile integrazione con **Numpy**, il sovraccarico di calcoli numerici appare ridotto poiché esso lavora con il tipo di dato “array multidimensionale”, che copre diverse funzioni per l'indicizzazione,

---

<sup>1</sup>Application Programming Interface: “insieme di funzioni e procedure che consentono la creazione di applicazioni che accedono alle funzionalità o ai dati di un sistema operativo, un'applicazione o un altro servizio. Si tratta di un insieme di metodi per la comunicazione tra sistemi e/o componenti diversi.”

il ridimensionamento e la realizzazione di calcoli elementari in maniera semplice e rapida. Tali operazioni vengono eseguite, ad esempio, in un intero array e non per ogni singolo valore di input.

La definizione del modello di dati con **Keras** risulta relativamente semplice: i modelli creati con la suddetta *library* sono definiti come una sequenza di livelli, in grado di facilitare la creazione del modello. Basta semplicemente inserire un livello alla volta fino a quando non si è soddisfatti della topologia della rete. È innanzitutto necessario assicurarsi che il livello di input possieda il numero corretto di entrate e che l'ultimo livello segua il ragionamento logico dell'architettura, consentendo al modello di rispondere correttamente in base alla strutturazione del problema. **Keras** rende possibile, attraverso i passaggi degli argomenti in ogni livello, la definizione del valore dei vari iperparametri utilizzati nella costruzione delle RNA (vedi sezione 3.1). La definizione di tale modello può essere sviluppata come segue:

Algorithm 8.1: Sequential model Keras

---

```

1 >> model = Sequential()
2     ...
3 >> model.add(Dense(3, activation='softmax'))

```

---

La compilazione del modello utilizza invece una delle *library* di *backend* selezionate, **Theano** o **TensorFlow**. È proprio il backend che sceglie automaticamente il modo migliore per rappresentare la rete e fare previsioni. In questa fase vengono passati come argomento le funzioni di costo, l'*optimizer* scelto e le metriche utilizzate per la valutazione al modello sequenziale di **Keras**. Per tali ragioni è necessario possedere una conoscenza preliminare dei concetti definiti in questa tesi, poiché per la corretta strutturazione della rete risulta importante conoscere nel dettaglio il problema da risolvere prima di definire tale architettura.

Algorithm 8.2: Compiling model Keras

---

```

1 >> model.compile(loss='categorical_crossentropy',
2                     optimizer=optimizer,
3                     metrics=['accuracy'],
4                     sample_weight_mode='temporal')

```

---

È inoltre possibile regolare il modello in base al numero di iterazioni (o epoch) e al numero di istanze che verranno valutate per iterazioni. Questi valori possono essere regolati empiricamente o definiti basandosi sul problema (vedi codice 8.3). È proprio in questo momento che si definiscono i dati di training e si sceglie quale porzione o quali dati verranno impiegati per la valutazione (cfr. codice 8.4). L'analisi effettiva del modello viene invece effettuata indicando quali saranno i dati di test, essendo possibile tramite questi eseguire la previsione del modello (vedi codice 8.5).

Algorithm 8.3: Fitting model Keras

---

```

1 >> model.fit(x.train, y.train,
2                 validation_split = 0.2,
3                 epochs=EPOCHS,
4                 batch_size=BATCH_SIZE)

```

---

Algorithm 8.4: Evaluating model Keras

---

```

1 >> model.evaluate(x.test, y.test, verbose=0)

```

---

Algorithm 8.5: Predicting model Keras

---

```

1 >> model.predict(x=x.test, batch_size=32, verbose=0)

```

---

Una volta ottenuti i valori di predizione del modello, si è scelto di fare uso di un modulo ben definito del framework ‘‘Distiller’’ [6] al fine di eseguire le valutazioni delle metriche **Precision**, **Recall**, **F-Measure**, **F1@5** e **F1@10** (cfr. capitolo 9).

In sintesi, per raggiungere gli obiettivi di questa tesi, è stato implementato un modello architetturale di rete neurale artificiale utilizzando il linguaggio **Python**, inizialmente nella versione 2.7.13 e successivamente adattando il codice alla versione 3.5.2 di **Python**; si è fatto ricorso alla library **Keras** con il backend **Theano**, al toolkit **NLTK** per l’attività di pre-elaborazione (vedi capitolo 5) ed al modulo ‘‘**metrics**’’ del framework **Distiller** per la valutazione dei dati di previsione. I test sono stati inoltre eseguiti su una GPU **GeForce 940M** e su una GPU **GeForce GTX Titan X Pascal**.

## 8.2 Epilogo

In questo capitolo sono stati descritti gli strumenti *software* e *hardware* utilizzati per la costruzione del modello proposto: sono stati riportati alcuni esempi di utilizzo dell’API e messe in evidenza le principali caratteristiche dell’impostazione dell’ambiente di sviluppo.

# Capitolo 9

## Risultati sperimentali

In questo capitolo verranno presentati i risultati ottenuti dai test effettuati sul modello proposto e la tipologia di dataset sui quali essi sono stati eseguiti; verranno elencati i parametri scelti per ciascuna configurazione di test, confrontando i loro risultati. Verrà poi presentata una breve sintesi delle metriche utilizzate per la valutazione dei test e indicata quale configurazione risulti più adeguata per ciascun dataset, confrontandole con i risultati ottenuti da altri modelli ben noti in letteratura.

### 9.1 Metriche di valutazione

Le metriche di valutazione sono strumenti volti ad esaminare la qualità dei risultati di un modello in relazione allo scopo dell'attività per la quale esso è stato proposto. Tali metriche sono funzioni matematiche utili nella valutazione dell'errore e del grado di correttezza dell'architettura in analisi. Esistono diversi tipi di metriche e analisi di qualità, alcune più semplici, altre più complesse o maggiormente specifiche e così via. Per compiere la scelta di una metrica di valutazione, tuttavia, risulta interessante prendere in considerazione alcuni fattori, come ad esempio l'applicazione pratica del modello. Per le attività di classificazione si cerca di prevedere a quale categoria appartenga ciascun campione. Le metriche utilizzate per questo scopo usano parametri facilmente adattabili. In seguito verranno elencate le metriche più usate nell'ambito dell'estrazione di frasi chiave e quelle impiegate per la valutazione dei risultati ottenuti in questo elaborato [64], [89]:

- Precisione: metrica che calcola quali esempi classificati come corretti risultino effettivamente tali; in termini matematici corrisponde al numero di esempi classificati correttamente diviso per tale numero più la quantità di esempi classificati erroneamente (cfr. eq. 9.1).

$$Precision = \frac{RA_c}{RA_c + WR_c} \quad (9.1)$$

Dove “RA” sta per risposte giuste (dall’inglese “*right answers*”), “WR” identifica le risposte sbagliate (dall’inglese “*wrong answers*”), mentre “ $c$ ” sta per “appartenenti alla classe  $c$ ” ;

- Recall: metrica che misura il numero di esempi che il sistema ha valutato erroneamente, ossia il numero di esempi classificati correttamente diviso per il totale di esempi appartenenti alla classe in analisi. Tale metrica non tiene però conto di tutte le classi esistenti; da ciò ne deriva che, se tutti gli esempi fossero classificati come appartenenti ad una singola classe, si otterrebbe allora il massimo Recall per quella classe specifica (cfr. eq. 9.2).

$$\text{Recall} = \frac{RA_c}{TS_c} \quad (9.2)$$

Dove “RA” sta per risposte giuste, “TS” sta per “tutti gli esempi” (dall’inglese “*total samples*”) mentre “ $c$ ” sta per “appartenenti alla classe  $c$ ”;

- Accuracy: metrica semplice, attraverso la quale si divide il numero di risultati corretti per il numero totale di esempi. Tale metrica viene solitamente utilizzata per la valutazione di modelli nei i quali l’insieme di dati contiene la stessa proporzione di esempi per ogni classe. Nel caso di classi sproporzionate, l’*accuracy* provoca una falsa impressione di buone prestazioni (cfr. eq. 9.3).

$$\text{Accuracy} = \frac{RA_c}{TS_{dt}} \quad (9.3)$$

Dove “RA” sta per risposte giuste, “TS” sta per “tutti gli esempi”, “ $c$ ” sta per “appartenenti alla classe  $c$ ” e “ $dt$ ” sta per “dataset”;

- F-measure: metrica armonica tra Recall e Precision; essa indica la qualità generale del modello e funziona bene su dati appartenenti a classi sproporzionate. Tale metrica misura l’efficienza del sistema tenendo conto di tutte le classi: è infatti necessario che il risultato positivo per entrambe le classi aumenti in modo che anche il valore della metrica stessa cresca. Essa consente di assegnare pesi diversi a ciascuna classe; quando il valore del peso è uguale a 1, la metrica è denominata F1 (cfr. eq. 9.4).

$$F1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (9.4)$$

- F1@ $k$ : può essere a volte interessante calcolare la qualità del modello basandosi su un insieme di valori classificati, invece di calcolarne la qualità in base ad una lista disordinata. Per questo motivo, è possibile far ricorso ad una porzione di  $k$  documenti rilevanti per le metriche Precision e Recall, calcolando quindi la metrica armonica tra Recall@ $k$  e Precision@ $k$ . Tale calcolo viene chiamato F1@ $k$ ;

- Average Precision: metrica che, in sostanza, consiste in una media del punteggio della Precision, calcolato successivamente alla restituzione di ogni documento classificato come rilevante. (cfr. eq. 9.5).

$$AveragePrecision = \frac{\sum_r Precision@r}{R} \quad (9.5)$$

Dove “ $r$ ” è la posizione del documento rilevante e “ $R$ ” è invece il numero totale di documenti rilevanti.

- Mean Average Precision (MAP): questa metrica rappresenta un'estensione dell'*Average Precision* nella quale viene calcolata la media di tutte le *Average Precision* al fine di ottenere la così cosiddetta MAP.

## 9.2 I Dataset utilizzati

Come già menzionato in precedenza nel capitolo 4, uno dei passi fondamentali per la strutturazione di una rete neurale artificiale è proprio quello di “scegliere i dati di esempi che verranno utilizzati per l’addestramento della stessa, in modo che il dominio di lavoro sia ben rappresentato”. È inoltre indispensabile scegliere propriamente i dati di esempi che verranno in seguito impiegati per la fase di test del modello, quella in cui verrà valutata l’effettiva capacità di generalizzazione dello stesso.

Per la strutturazione del modello architettonale proposto in questo elaborato si è optato per l’utilizzo del dataset **Inspec** 2003 [33]. Le scelte strutturali sono state replicate ed applicate ad altri insiemi di dati, cambiando soltanto il valore di alcuni parametri e non la struttura in sé. I dataset scelti per la disposizione del modello e per l’esecuzione degli esperimenti saranno descritti nell’elenco seguente:

- **Inspec** 2003: una raccolta di dati in lingua inglese composta da 2.000 abstract e dalle loro corrispondenti frasi chiave. Si tratta di articoli scientifici, riguardanti in particolare discipline di *Computers and Control* e *Information Technology*. Ogni documento è associato a due tipi di etichette: i set di frasi chiave che contengono termini controllati, presenti nel Database **Inspec**, e i set di frasi chiave che contengono invece termini non sottoposti a controllo, che possono quindi non essere presenti nel Database. Sia per l’insieme degli item controllati che per quello degli item non controllati, le frasi chiave possono essere presenti o meno nel documento. Si è scelto di utilizzare il set di elementi sottoposti a controllo per eseguire gli esperimenti, poiché, secondo Hulth [33], circa il 76,2% delle frasi chiave appartenenti a tale set sono presenti nei documenti, rispetto al 18,1% dell’insieme degli elementi controllati [33];
- **Krapivin** 2009: un set di dati proposto da Krapivin [39] e composto da 2.304 articoli scientifici pubblicati da ACM, i cui contenuti sono strettamente

legati alle discipline informatiche. Ad ogni documento ne è associato un altro, contenente le frasi chiave; gli esperimenti sono stati svolti utilizzando solo gli abstract di ogni documento. Non è stato specificato esattamente come suddividere il set di dati; si è quindi deciso di impiegare 400 documenti per il test, 400 per la validazione e il resto per il training;

- **Kp20k:** set di dati costruito da Meng, all'interno del ogni item contiene titolo, abstract e frasi chiave di circa 567.830 articoli scientifici dell'area informatica. Tale set di dati è suddiviso in 527.830 documenti per l'addestramento, 20.000 per la validazione e 20.000 per il test. In [55], sono stati impiegati soltanto 20.000 documenti per addestrare le *baseline* dei modelli usati nel confronto di risultati; si è quindi optato per l'addestramento della rete utilizzando la stessa quantità di dati.

Si è deciso di impiegare i dataset **Krapivin 2009** e **Inspec** poiché si tratta di tipologie di set di dati noti e dalla dimensione relativamente estesa se paragonate, ad esempio, al dataset **SemEval 2010**. Esso è stato usato nella competizione **SemEval 2010** e contiene solo 144 documenti per il training, 40 per la validazione e 100 per il test [35], rispetto ai 2.000 documenti totali di **Inspec** e ai 2.304 di **Krapivin 2009**. Si è inoltre deciso di non utilizzare il dataset impiegato nella competizione **SemEval 2017** [2] poiché, sebbene fosse presente la *subtask B* della competizione – elemento specifico per compiti di estrazione di frasi chiave – essa comprende molte altre caratteristiche, come la divisione di frasi chiave in categorie e l'estrazione di relazioni tra i termini, obiettivi che esulano dagli ambiti di questa tesi.

### 9.3 Impostazione dei parametri

I test sono stati effettuati sul modello presentato nel capitolo 4 e sulle GPU **GeForce 940M** e **GeForce GTX Titan X Pascal**; è stato modificato il valore di più attributi al fine di ottimizzare i risultati e verificare l'importanza di determinati parametri per la rete, come nel caso degli *embedding*.

Il modello proposto è di natura sequenziale e la sua composizione risulta articolata nella seguente strutturazione: il livello di input riceve i documenti mappati dai *word embedding*; esso è collegato ad una **BLSTM** contenente la funzione di attivazione Tangente iperbolica e l'attivazione dei livelli ricorrenti tramite la funzione Hard Sigmoid; il livello di output è composto da tre neuroni la cui l'attivazione viene eseguita dalla funzione Softmax; tra la **BLSTM** e il livello di output è stato inserito un Dropout di 0.5 (50%) per evitare l'*overfitting*; si è preferito il valore di 32 esempi per *Batch*; l'*optimizer* scelto è stato il RMSprop con il *learning rate*= 0.0005, mentre la scelta della funzione di costo è ricaduta sulla *categorical crossentropy* (cfr. cap. 4). Le ulteriori variabili come la quantità di epoche, il valore del *batch*, il numero di unità **BLSTM**, gli *embedding* ecc. possono essere soggette a modifiche.

I test sono stati eseguiti mantenendo la configurazione predefinita dei livelli principali, ossia i livelli principali, l'*optimizer*, le funzioni di attivazione e la funzione di costo, modificando esclusivamente il valore dei seguenti parametri:

- numero di *feature* utilizzate per la costruzione degli *embedding*;
- numero di unità nel livello BLSTM;
- numero di livelli aggiuntivi;
- valore passato al livello di *dropout*.

Il numero di unità nel livello BLSTM è un valore scelto empiricamente. Per ciascun dataset sono stati eseguiti svariati test, i quali hanno dimostrato come il valore più adatto vari tra le 100 e le 200 unità BLSTM; valori più alti risultano non idonei a causa della difficoltà che la rete riscontra nel processo di apprendimento. Per il dataset *Inspec* si è optato per 184 unità BLSTM, mentre per il dataset *Krapivin* e *Kp20k*, 151 e 155 unità, rispettivamente.

I risultati ottenuti per le metriche Precision, Recal e F1-score sono riportati nella tabella 9.1. Viene inoltre segnalato il modello pre-addestrato *word embedding*, per il quale la combinazione “units + embedding” ha prodotto i risultati più adatti per ciascun test.

Unità BLSTM						
Dataset	Embedding	Units BLSTM	Precision	Recall	F1-score	
Inspec	Glove-200	184	0.380	0.642	0.477	
Krapivin	Glove-200	151	0.243	0.481	0.323	
Kp20k	Word2Vec-300	155	0.136	0.453	0.210	

Tabella 9.1: BLSTM Units

La quantità di unità più idonea a ciascun dataset, nonostante la natura empirica, potrebbe essere relazionata alla dimensione dei documenti appartenenti all’insieme di dati. Tale valore è collegato alla quantità di informazioni che la rete analizza e conserva per ogni spazio di tempo.

Come già accennato nel capitolo 4, l’approccio presentato in questa tesi esegue i test sui modelli pre-addestrati *Stanford’s GloVe Embeddings* [63] e *Word2Vec pretrained vectors* [58]. Il primo propone quattro alternative, che differiscono in base alla quantità di caratteristiche applicate alla rappresentazione delle parole in spazi vettoriali: 50, 100, 200 o 300 dimensioni per un vocabolario di 400 mila parole. Il secondo contiene invece vettori di 300 dimensioni per circa tre milioni di parole e termini.

Per verificare quale sia il modello più adatto e la quantità più idonea di caratteristiche da utilizzare nel livello *embeddings* sono stati eseguiti dei test, ricorrendo ai modelli pre-addestrati *GloVe* e *Word2Vec* per ciascun dataset (cfr. Tabelle 9.2, 9.3 e 9.4).

INSPEC							
Embedding	Precision	Recall	F1-score	MAP	F1@5	F1@10	Epochs
GloVe-50	0.297	0.637	0.405	0.336	0.271	0.333	14
GloVe-100	0.346	<b>0.653</b>	0.453	0.373	0.301	0.378	14
<b>GloVe-200</b>	<b>0.380</b>	<u>0.642</u>	<b>0.477</b>	<b>0.390</b>	<b>0.320</b>	<b>0.404</b>	14
GloVe-300	0.359	0.639	0.460	0.376	0.311	0.382	14
Word2Vec-300	0.290	0.593	0.390	0.302	0.245	0.310	28

Tabella 9.2: Inspec Embeddings

KRAPIVIN							
Embedding	Precision	Recall	F1-score	MAP	F1@5	F1@10	Epochs
GloVe-50	0.086	0.456	0.144	0.339	0.272	0.199	8
GloVe-100	0.091	0.460	0.152	0.344	0.276	0.206	7
<b>GloVe-200</b>	<b>0.243</b>	<b>0.481</b>	<b>0.323</b>	<b>0.392</b>	<b>0.345</b>	<b>0.276</b>	18
GloVe-300	0.214	0.478	0.295	0.383	0.330	0.264	18
Word2Vec-300	0.205	0.458	0.284	0.374	0.324	0.252	18

Tabella 9.3: Krapivin Embeddings

Kp20k							
Embedding	Precision	Recall	F1-score	MAP	F1@5	F1@10	Epochs
GloVe-50	0.127	0.437	0.196	0.263	0.247	0.218	6
GloVe-100	0.123	0.459	0.193	0.265	0.247	0.218	6
GloVe-200	0.125	<b>0.460</b>	0.197	0.267	0.248	0.220	7
GloVe-300	0.129	0.448	0.200	0.257	0.236	0.215	7
Word2Vec-300	<b>0.136</b>	0.453	<b>0.210</b>	<b>0.270</b>	<b>0.253</b>	<b>0.227</b>	14

Tabella 9.4: Kp20k Embeddings

Per quanto riguarda i dataset **Inspec** e **Krapivin**, risulta evidente come la quantità di caratteristiche più adatta all'utilizzo nel livello *embeddings* sia pari a 200. Viene inoltre dimostrato come la dimensione del vocabolario usato per la costruzione dei “*word embedding*” debba essere proporzionale al dataset impiegato. Dataset minori hanno una maggior probabilità di avere vocabolari più ristretti; di conseguenza, modelli pre-addestrati su una quantità minore di termini risultano più adatti a tale tipologia di dataset e viceversa [92].

### Ulteriori Test: KRAPIVIN E INSPEC

In seguito verranno riportati alcuni test supplementari effettuati unicamente sui dataset **Inspec** e **Krapivin**. Si è scelto di non applicare tali esperimenti al dataset **Kp20k**, a causa della discrepanza tra le moli di dati contenute in ciascun insieme di dati e dell'elevata quantità di tempo richiesta per effettuare ogni test.

Secondo [77] e tramite l'osservazione dell'impiego del *dropout* in altri studi, si è riscontrato che il valore passato come parametro al livello di *dropout* dovrebbe essere compreso tra 0.25 e 0.50. Per tali ragioni, sono stati realizzati dei test utilizzando questi due valori soglia per i dataset **Inspec** e **Krapivin** (cfr. tabella 9.5).

INSPEC e Krapivin					
Dataset	Dropout	Precision	Recall	F1-score	Epochs
Inspec	0.25	0.219	0.541	0.312	14
Inspec	0.50	<b>0.380</b>	<b>0.642</b>	<b>0.477</b>	14
Krapivin	0.25	0.139	0.472	0.214	14
Krapivin	0.50	<b>0.243</b>	<b>0.481</b>	<b>0.323</b>	18

Tabella 9.5: Inspec e Krapivin - Dropout

Per entrambi i dataset il valore 0.50 (50%), passato al livello di *dropout*, risulta pertinente alle scelte strutturali del modello proposto.

Si è inoltre tentato di far uso di un livello nascosto che fosse completamente connesso (**Time Distributed Dense**) con il livello **BLSTM** e quello di output **Softmax**, contenendo la funzione di attivazione (*Rectified Linear* (ReLU)). Sono stati inseriti due livelli di *dropout*: uno tra il livello **BLSTM** e quello **Time Distributed Dense** ed altro tra quest'ultimo ed il livello di output. Per i livelli di *dropout* si è scelto il valore 0.25. Il livello aggiuntivo contiene inoltre la stessa quantità di unità presente nel livello **BLSTM**, ossia 184 per il dataset **Inspec** e 151 per **Krapivin** (cfr. tabella 9.6).

Come è possibile osservare nella tabella 9.6, l'impiego di un livello supplementare, completamente connesso, tra i livelli **BLSTM** e di output, non ha prodotto dei miglioramenti apprezzabili.

INSPEC e Krapivin				
Dataset	Precision	Recall	F1-score	Epochs
Inspec	0.346	0.599	0.438	14
Krapivin	0.098	0.460	0.161	14

Tabella 9.6: Inspec e Krapivin livello aggiuntivo

### Risultati idonei

Si evince che i risultati più adatti per ciascun dataset sono stati ottenuti mediante le seguenti configurazioni:

Impostazioni						
Dataset	Embeddings	BLSTM Units	Dropout	Add. Layer	Epoche	
Inspec	Glove-200	184	0.50	no	14	
Krapivin	Glove-200	151	0.50	no	18	
Kp20k	Word2Vec-300	155	0.50	no	14	

Tabella 9.7: Impostazioni più adeguate per ogni dataset

Il numero di epoche è stato definito sulla base della regola di arresto anticipato di **Keras**, valutando per ciascun modello il numero esatto di volte necessario affinché la funzione di costo non diminuisca per due epoche consecutive.

## 9.4 Risultati comparativi

Di seguito verranno presentate alcune tabelle comparative, all'interno delle quali saranno eseguiti dei confronti tra l'approccio proposto in questa tesi ed altri sistemi competitivi, basati su tecniche di *Machine learning* supervisionate e non.

Per quanto concerne il dataset **Inspec**, è stato eseguito un confronto tra differenti approcci ed il metodo proposto in questo elaborato: i primi tre sistemi mostrati nella tabella 9.8 impiegano tecniche diverse per la generazione di frasi chiave candidate: *n-gram*, *Noun Phrase chunking* e *patterns* [33]; il quarto sistema, chiamato *TopicRank*, corrisponde invece ad un sistema per l'estrazione di frasi chiave incentrato sull'uso di grafi per la rappresentazione topica del documento [11].

Metodo	Precision	Recall	F1-score
<b>BLSTM AKE</b>	<b>0.380</b>	<b>0.642</b>	<b>0.477</b>
n-grams with tag	0.252	<u>0.517</u>	<u>0.339</u>
NP Chunking with tag	0.297	0.372	0.330
Pattern with tag	0.217	0.399	0.281
TopicRank	<u>0.348</u>	0.404	0.352

Tabella 9.8: Confronto usando Inspec: F1-score, Recall e Precision

Nella tabella 9.9 sono invece riportati i risultati comparativi ottenuti ricorrendo ai valori delle metriche F1@5 e F1@10, confrontando l'approccio proposto in questa tesi con il modello presentato in [55] per i dataset **Inspec**, **Krapivin** e **Kp20k**:

Metodo	Inspec		Krapivin		Kp20k	
	F1@5	F1@10	F1@5	F1@10	F1@5	F1@10
<b>BLSTM AKE</b>	<b>0.320</b>	<b>0.404</b>	<b>0.345</b>	<b>0.276</b>	0.253	0.227
Deep KP Extraction	0.278	0.342	0.311	0.266	<b>0.333</b>	<b>0.262</b>

Tabella 9.9: Confronto usando Inspec: F1@5 e F1@10

Si conclude che, per quanto concerne i dataset **Inspec** e **Krapivin**, il modello proposto in questa tesi restituisce dei risultati di fatto interessanti, di gran lunga migliori di quelli risultati riscontrati in letteratura. Sarebbe tuttavia interessante studiare ulteriormente la configurazione della rete, a patto che sia possibile raggiungere risultati simili per altri insiemi di dati, specialmente per quanto riguarda il dataset **Kp20k**, per il quale risulterebbe senza dubbio necessario, soprattutto a causa dell'elevato tempo impiegato per l'esecuzione della rete.

## 9.5 Epilogo

In questo capitolo sono stati descritti i vari test eseguiti sulla struttura proposta nel presente elaborato. Sono state definite le metriche di valutazione utilizzate e la tipologia di dataset impiegati nell'analisi della validità del modello indicato. Sono state messe a confronto le impostazioni di differenti parametri, concludendo il capitolo con una comparazione dei risultati ottenuti dalla struttura proposta con quelli di altri modelli ben noti in letteratura.

# Capitolo 10

## Conclusioni e sviluppi futuri

In questa tesi è stato presentato un approccio basato sul *Deep Learning* per l'estrazione di frasi chiave da documenti di testo. Esso impiega il modello di rete neurale *Bidirectional Long short-term memory*. Per confermare l'importanza della rappresentazione delle parole in spazi vettoriali, si è analizzato l'approccio usando cinque tipi di modelli pre-addestrati *word embedding*, che utilizzano rispettivamente 50, 100, 200, e 300 caratteristiche distinte. Si è inoltre esaminato il comportamento della rete ricorrendo a differenti impostazioni dei suoi parametri, come ad esempio la quantità di unità BLSTM.

Si è dimostrato come l'approccio proposto sia efficiente e capace di raggiungere gli stessi risultati ottenuti da metodi riconosciuti e noti in letteratura, per quanto riguarda l'estrazione di frasi chiave di documenti di testo. È importante sottolineare che il modello proposto in questo elaborato non richiede l'uso di caratteristiche aggiuntive determinate a priori, come ad esempio il *Part Of Speech tagging* o il filtraggio di *stop words*. La rete risulta infatti capace di imparare autonomamente quali siano le caratteristiche di una frase chiave, ossia come un termine dovrebbe essere composto (ad es.: *patterns “noun + preposition + noun”*, ecc.); a tale scopo, basta solamente specificare quali siano i dati di input e, per quanto riguarda la fase di training, quali sono le etichette per ciascun dato.

Il modello proposto è stato poi confrontato con altri approcci attraverso l'analisi di metriche standard di valutazione. Per quanto concerne la valutazione dell'architettura sui dataset **INSPEC** e **Krapivin**, si è verificato che, in confronto agli altri approcci impiegati sugli stessi dataset, si ottengono effettivamente risultati migliori. Appare inoltre ormai noto come l'impiego di diverse rappresentazioni per le parole del vocabolario dei differenti dataset influisca sul risultato.

Degna di nota è inoltre la capacità del modello proposto di affrontare l'ampia varietà di caratteristiche lessicali, linguistiche e semantiche presenti in diversi documenti di testo, in quanto, come già accennato, non è necessario determinare a priori

---

caratteristiche specifiche, quali ad esempio quelle di natura linguistica.

Per quanto concerne l'impiego della rappresentazione distribuita dei termini appartenenti agli insiemi di dati utilizzati, è possibile concludere che tale pratica non implica direttamente il miglioramento delle prestazioni, sebbene essa venga utilizzata in modo evidente per tale scopo. L'aspetto rilevante da considerare è soprattutto quello dell'identificazione di similitudini all'interno dell'insieme analizzato. Gran parte dei metodi usati per l'estensione di dati non risulta in grado di distinguere i concetti di similarità e di rapporto tra parole; possedere l'informazione riguardante i termini in rapporto non significa necessariamente che essi avranno significati simili. L'impiego delle rappresentazioni distribuite diviene dunque un elemento di considerevole importanza per quanto riguarda l'estrazione automatica di frasi chiave da documenti di testo, specialmente nel caso in cui lo scopo sia quello di estrarre termini rilevanti non presenti in esso. Un'interessante prospettiva di continuità della ricerca presentata in questo elaborato potrebbe essere quella di collegare effettivamente i termini trovati in ciascun documento con altri termini simili; una volta che ogni parola risulti associata al suo rispettivo *embedding*, la connessione dipenderebbe unicamente dall'utilizzo o dalla costruzione di uno strumento capace di calcolare tale correlazione e di generalizzare i possibili casi.

Si conclude infine che il *Deep Learning* rappresenta un'area dimostratasi di grande interesse per ricercatori provenienti da diversi campi di studio. In merito a questo fattore, sarebbe auspicabile proseguire lo studio eseguendo ulteriori valutazioni su altri dataset, compresi quelli impostati in altre lingue. Si potrebbe inoltre considerare l'impiego di altre tecniche, applicate nella costruzione di modelli matematici basati sul *Deep Learning*, come ad esempio l'utilizzo di altri livelli aggiuntivi, come quelli del tipo *dilated convolutions* [12] e *attentive* [23].

# Riferimenti bibliografici

- [1] D. A. Audich, R. Dara, and B. Nonnemecke. Extracting keyword and keyphrase from online privacy policies. In *Digital Information Management (ICDIM), 2016 Eleventh International Conference on*, pages 127–132. IEEE, 2016.
- [2] I. Augenstein, M. Das, S. Riedel, L. Vikraman, and A. McCallum. Semeval 2017 task 10: Scienceie-extracting keyphrases and relations from scientific publications. *arXiv preprint arXiv:1704.02853*, 2017.
- [3] Y. Bar, I. Diamant, L. Wolf, and H. Greenspan. Deep learning with non-medical training used for chest pathology identification. In *Proc. SPIE*, volume 9414, page 94140V, 2015.
- [4] M. Basaldella, E. Antolli, G. Serra, and C. Tasso. Bidirectional lstm recurrent neural network for keyphrase extraction. In *Italian Research Conference on Digital Libraries*, pages 180–187. Springer, 2018.
- [5] M. Basaldella, G. Chiaradia, and C. Tasso. Evaluating anaphora and coreference resolution to improve automatic keyphrase extraction. In *COLING*, pages 804–814, 2016.
- [6] M. Basaldella, D. De Nart, and C. Tasso. Introducing distiller: A unifying framework for knowledge extraction. In *IT@ LIA@ AI\* IA*, 2015.
- [7] M. Basaldella, M. Helmy, E. Antolli, M. H. Popescu, G. Serra, and C. Tasso. Exploiting and evaluating a supervised, multilanguage keyphrase extraction pipeline for under-resourced languages. In *RANLP*, pages 4–6, 2017.
- [8] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.
- [9] S. Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.

- [10] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.
- [11] A. Bougouin, F. Boudin, and B. Daille. Topicrank: Graph-based topic ranking for keyphrase extraction. In *International Joint Conference on Natural Language Processing (IJCNLP)*, pages 543–551, 2013.
- [12] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.
- [13] F. Chollet et al. Keras. <https://github.com/fchollet/keras>, 2015.
- [14] A. Colyer. The amazing power of word vectors. <https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/>, 2016. consultato il 26-05-2017 alle 08:43:00.
- [15] R. Dale, H. Moisl, and H. Somers. *Handbook of natural language processing*. CRC Press, 2000.
- [16] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [17] D. Duong, E. Eskin, and J. Li. A novel word2vec based tool to estimate semantic similarity of genes by using gene ontology terms. *bioRxiv*, page 103648, 2017.
- [18] Y. Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.
- [19] Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.
- [20] P. Golik, Z. Tüske, R. Schlüter, and H. Ney. Multilingual features based keyword search for very low-resource languages. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.
- [21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [22] A. Graves and J. Schmidhuber. Framewise phoneme classification with bi-directional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.
- [23] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

- [24] C. Gulcehre, M. Moczulski, M. Denil, and Y. Bengio. Noisy activation functions. In *International Conference on Machine Learning*, pages 3059–3068, 2016.
- [25] N. Gupta. Artificial neural network. *Network and Complex Systems*, 3(1):24–28, 2013.
- [26] C. Gutwin, G. Paynter, I. Witten, C. Nevill-Manning, and E. Frank. Improving browsing in digital libraries with keyphrase indexes. *Decision Support Systems*, 27(1):81–104, 1999.
- [27] S. Han, D. He, J. Jiang, and Z. Yue. Supporting exploratory people search: a study of factor transparency and user control. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 449–458. ACM, 2013.
- [28] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.
- [29] J. Heaton, N. Polson, and J. H. Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.
- [30] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.
- [31] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.
- [32] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [33] A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 216–223. Association for Computational Linguistics, 2003.
- [34] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.
- [35] S. N. Kim, O. Medelyan, M.-Y. Kan, and T. Baldwin. Semeval-2010 task 5: Automatic keyphrase extraction from scientific articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 21–26. Association for Computational Linguistics, 2010.
- [36] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

- [37] T. Koskela, M. Lehtokangas, J. Saarinen, and K. Kaski. Time series prediction with multilayer perceptron, fir and elman neural networks. In *Proceedings of the World Congress on Neural Networks*, pages 491–496. INNS Press San Diego, USA, 1996.
- [38] S. Kottur, R. Vedantam, J. M. Moura, and D. Parikh. Visual word2vec (vis-w2v): Learning visually grounded word embeddings using abstract scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4985–4994, 2016.
- [39] M. Krapivin, A. Autaeu, and M. Marchese. Large dataset for keyphrases extraction. Technical report, University of Trento, 2009.
- [40] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238, 1995.
- [41] N. Kumar, K. Srinathan, and V. Varma. A graph-based unsupervised n-gram filtration technique for automatic keyphrase extraction. *International Journal of Data Mining, Modelling and Management*, 8(2):124–143, 2016.
- [42] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.
- [43] T. K. Landauer and S. T. Dumais. A solution to plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.
- [44] T. T. N. Le, M. Le Nguyen, and A. Shimazu. Unsupervised keyphrase extraction: Introducing new kinds of words to keyphrases. In *Australasian Joint Conference on Artificial Intelligence*, pages 665–671. Springer, 2016.
- [45] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [46] O. Levy and Y. Goldberg. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the eighteenth conference on computational natural language learning*, pages 171–180, 2014.
- [47] X. Liao and J. Wang. Global dissipativity of continuous-time recurrent neural networks with time delay. *Physical Review E*, 68(1):016118, 2003.
- [48] W. Ling, C. Dyer, A. W. Black, and I. Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304, 2015.

- [49] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.
- [50] Z. Liu, W. Huang, Y. Zheng, and M. Sun. Automatic keyphrase extraction via topic decomposition. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 366–376. Association for Computational Linguistics, 2010.
- [51] M. L. Marques da Silva Binoti, D. H. Breda Binoti, and H. Garcia Leite. Aplicação de redes neurais artificiais para estimação da altura de povoamentos equiâneos de eucalipto. *Revista Árvore*, 37(4), 2013.
- [52] J. Martinez-Romo, L. Araujo, and A. Duque Fernandez. Semgraph: Extracting keyphrases following a novel semantic graph-based approach. *Journal of the Association for Information Science and Technology*, 67(1):71–82, 2016.
- [53] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [54] R. Meng, S. Han, Y. Huang, D. He, and P. Brusilovsky. Knowledge-based content linking for online textbooks. In *Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on*, pages 18–25. IEEE, 2016.
- [55] R. Meng, S. Zhao, S. Han, D. He, P. Brusilovsky, and Y. Chi. Deep keyphrase generation. *arXiv preprint arXiv:1704.06879*, 2017.
- [56] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [57] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.
- [58] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [59] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *hlt-Naacl*, volume 13, pages 746–751, 2013.
- [60] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *1990 IJCNN International Joint Conference on Neural Networks*, pages 21–26 vol.3, June 1990.

- [61] S. Nohara and R. Saga. Preprocessing method topic-based path model by using word2vec. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2017.
- [62] C. Olah. Understanding lstm networks. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015. consultato il 20-10-2017 alle 07:30:00.
- [63] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [64] D. M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.
- [65] S. Raschka. *Python Machine Learning*. Packt Publishing, Birmingham, UK, 2015.
- [66] T. W. Rauber. Redes neurais artificiais. *Universidade Federal do Espírito Santo*, 2005.
- [67] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [68] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [69] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.
- [70] R. J. Schalkoff. *Artificial neural networks*, volume 1. McGraw-Hill New York, 1997.
- [71] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [72] S. Siddiqi and A. Sharan. Keyword and keyphrase extraction techniques: a literature review. *International Journal of Computer Applications*, 109(2), 2015.
- [73] E. Siegel. *Predictive analytics: The power to predict who will click, buy, lie, or die*. Wiley Hoboken (NJ), 2016.
- [74] S. K. Sienčnik. Adapting word2vec to named entity recognition. In *Proceedings of the 20th Nordic Conference of Computational Linguistics, NODALIDA 2015, May 11-13, 2015, Vilnius, Lithuania*, number 109, pages 239–243. Linköping University Electronic Press, 2015.

- [75] K. N. Singh, H. M. Devi, and A. K. Mahanta. Document representation techniques and their effect on the document clustering and classification: A review. *International Journal of Advanced Research in Computer Science*, 8(5), 2017.
- [76] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3):1464–1468, 1997.
- [77] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.
- [78] L. Sterckx, T. Demeester, C. Develder, and C. Caragea. Supervised keyphrase extraction as positive unlabeled learning. In *EMNLP2016, the Conference on Empirical Methods in Natural Language Processing*, pages 1–6, 2016.
- [79] G. Thimm and E. Fiesler. High-order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, 8(2):349–359, 1997.
- [80] T. Tieleman and G. Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical report, 2012. 31.
- [81] P. D. Turney. Learning algorithms for keyphrase extraction. *Information retrieval*, 2(4):303–336, 2000.
- [82] F. Vallet. The hebb rule for learning linearly separable boolean functions: learning and generalization. *EPL (Europhysics Letters)*, 8(8):747, 1989.
- [83] P. van den Driessche and X. Zou. Global attractivity in delayed hopfield neural network models. *SIAM Journal on Applied Mathematics*, 58(6):1878–1890, 1998.
- [84] K. Warburton. Deep learning and education for sustainability. *International Journal of Sustainability in Higher Education*, 4(1):44–56, 2003.
- [85] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [86] F. Xie, X. Wu, and X. Zhu. Efficient sequential pattern mining with wildcards for keyphrase extraction. *Knowledge-Based Systems*, 115:27–39, 2017.
- [87] B. Xue, C. Fu, and Z. Shaobin. A study on sentiment computing and classification of sina weibo with word2vec. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 358–363. IEEE, 2014.

- [88] M. Yousefi-Azar and L. Hamey. Text summarization using unsupervised deep learning. *Expert Systems with Applications*, 68:93–105, 2017.
- [89] E. Zhang and Y. Zhang. *Average Precision*, pages 192–193. Springer US, Boston, MA, 2009.
- [90] G. P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.
- [91] N. ZHANG and X.-r. CHEN. The improvement studies on jordan neural networks [j]. *Journal of Guizhou University (Natural Science Edition)*, 1:012, 2009.
- [92] Q. Zhang, Y. Wang, Y. Gong, and X. Huang. Keyphrase extraction using deep recurrent neural networks on twitter. In *EMNLP*, pages 836–845, 2016.