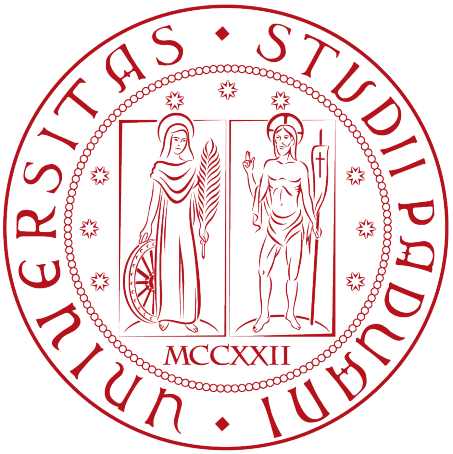
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Modelli di Reti Neurali e Natural Language Processing applicati all'Information Retrieval

Tesi di laurea

Relatore

Prof. Lamberto Ballan

Laureando

Gianpiero Giuseppe Tovo

Anno Accademico 2021/2022

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecentoventi ore, dal laureando Gianpiero Giuseppe Tovo presso l'azienda Zucchetti S.p.A.
Gli obiettivi da raggiungere hanno riguardato due tematiche inerenti l'*information retrieval*: il riconoscimento di *query* e l'analisi sintattica di frasi in linguaggio naturale. È stato svolto uno studio delle tecnologie coinvolte e sono state progettate delle reti neurali per sperimentare l'efficienza di varie soluzioni.
Il compito è stato portato a termine impiegando la libreria *ml.js*, che fornisce strumenti utili alla realizzazione di modelli di *machine learning*.

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Lamberto Ballan, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con a etto la mia famiglia ed i miei amici ed amiche per il sostegno durante gli anni di studio.

Padova, 21 Settembre 2022 Gianpiero Giuseppe Tovo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 L'azienda

L'esperienza del tirocinio che ho svolto ha avuto luogo negli uffici di Zucchetti S.p.A., un'azienda italiana che produce servizi e prodotti software e hardware per aziende, banche, assicurazioni, professionisti e associazioni di categoria. Tra le diverse sedi sparse in tutto il mondo, quella padovana si occupa di ricerca e sviluppo.

Da diversi anni l'azienda collabora con l'Università di Padova in diverse occasioni: sia come proponente di capitolati per il progetto di gruppo del corso di Ingegneria del software, sia come impresa partecipante all'iniziativa *STAGE-IT*, evento organizzato annualmente per mettere in contatto studenti ed aziende del territorio.

È proprio grazie a quest'ultima occasione che sono venuto a conoscenza delle diverse attività proposte da Zucchetti S.p.A., tra le quali mi ha attirato quella riguardante l'ambito del *machine learning*^{G}, argomento che desideravo approfondire da tempo e che ho quindi scelto di affrontare durante il tirocinio.

1.2 I progetti

Lo stage ha riguardato lo sviluppo di due progetti inerenti l'*information retrieval*^{G}ed il *natural language processing*^{G}, in particolare sperimentando in questi ambiti l'utilizzo di *reti neurali feed-forward*^{G}e la loro efficienza in compiti per la quale tipicamente vengono impiegate soluzioni diverse e più avanzate.

Nell'ambito dell'*information retrieval*, il primo progetto si è proposto di verificare il comportamento di una rete *feed-forward* implementata per riconoscere *query* in linguaggio naturale^G, verificando l'ipotesi pessimistica iniziale dovuta ai limiti della tecnica impiegata e delle risorse disponibili. Il prodotto della rete è risultato infatti insoddisfacente per cause imputabili alle prestazioni di una soluzione *feed-forward* con un *corpus*^{G}di dimensioni ridotte.

Nel secondo progetto, il modello di rete neurale è stato impiegato per realizzare un *part-of-speech tagger*^{G}ed ottenere quindi, a partire da una frase in linguaggio naturale, il ruolo grammaticale di ciascuna parola della quale la frase è composta. La bontà dei risultati di questo secondo prodotto è stata giudicata soddisfacente ma con un certo margine di miglioramento, colmabile agendo sulla qualità sia della logica degli script

sviluppati, sia dei corpus di dati utilizzati.

Assieme ai prodotti ricavati, relativamente a ciascuno dei due progetti è stato redatto un breve documento riguardante la spiegazione dei problemi trattati, il funzionamento del codice prodotto e i risultati ottenuti.

1.3 Strumenti utilizzati

L'azienda ha lasciato libertà nella scelta degli strumenti, fatta eccezione per la richiesta di utilizzare *Javascript*[G] e la libreria *ml.js* per lo sviluppo dei prodotti. Il lavoro è stato portato a termine principalmente sulla mia macchina personale con il sistema operativo *Xubuntu*.

Gli strumenti di sviluppo e di supporto utilizzati sono i seguenti:

Visual Studio Code Ambiente di sviluppo software multipiattaforma caratterizzato da una grande possibilità di personalizzazione. L'ho utilizzato per la familiarità che avevo già acquisito in precedenza e la disponibilità di plugin utili con la quale l'ho integrato, principalmente per assistere la formattazione ed il versionamento del codice, come ad esempio: *Syntax Highlighter*, *Brackets Pair Colorizer* e *GitLens*.

Mozilla Firefox Web browser open-source; utilizzato per l'esecuzione ed il debugging degli script.

Git Software utilizzato per il versionamento del codice.

GitHub Piattaforma di hosting per progetti software, utilizzato per hostare il backup del progetto versionato.

Google Sheet e Docs Servizi per l'elaborazione di fogli di testo e di calcolo, utilizzati per l'annotazione di appunti e risorse.

LaTeX Software di formattazione testi; utilizzato per la stesura del presente documento e dei report finali sulle attività e sui prodotti.

Per quanto riguarda invece le tecnologie principali utilizzate:

Javascript Linguaggio di programmazione ad alto livello; utilizzato per la codifica degli script dei prodotti.

Node.js Server di esecuzione per Javascript; utilizzato per eseguire pacchetti e script senza l'impiego di un web browser.

ml.js Libreria Javascript che fornisce strumenti utili alla realizzazione di modelli di machine learning.

JSdoc Generatore di documentazione Javascript utilizzato per produrre in modo automatico versioni ben leggibili e formattate del codice commentato.

Jest Framework Javascript utilizzato per sviluppare le suite di test del codice.

1.4 Organizzazione del testo

Per ciascuno dei due progetti, trattati nelle parti tematiche ricerca tramite rete neurale feed-forward e POS tagging con rete neurale feed-forward, vengono descritti il problema, la ricerca del materiale utile, lo sviluppo del prodotto richiesto e le conclusioni del lavoro.

Infine nella parte conclusiva vengono discussi i risultati finali e le conoscenze acquisite con l'esperienza dello stage.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *parola*^{G};
* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

Capitolo 2

Ricerca con reti neurali feed-forward

2.1 Analisi del problema

Il primo progetto trattato ha introdotto l'utilità e le proprietà dell'*information retrieval*, ovvero le tecniche per l'interrogazione, il prelievo e la classificazione di informazioni in corpus di dati. L'*information retrieval* trova la sua applicazione più esemplare nei motori di ricerca, che a partire da semplici *query* di ricerca hanno il compito di discriminare tra milioni di possibili documenti e di presentare i risultati nell'ordine più adatto, a seconda del contenuto e dei *metadati*.

Nella fattispecie del lavoro svolto, ha riguardato l'implementazione e la comparazione di due differenti tecniche per la ricerca tramite *query* in linguaggio naturale. Le interrogazioni sono state eseguite su un corpus di piccole dimensioni, composto dalle voci di un menu di un'applicazione fittizia, fornita dall'azienda.

Prima di tutto era richiesta la comprensione delle caratteristiche ed il funzionamento dell'applicazione dimostrativa che implementa il sistema *BM25* per fornire la voce più pertinente a partire da una *query*, ipoteticamente ricavata dalla comunicazione dell'utente con un *chatbot* o tramite una funzione di comando vocale.
In seguito era previsto lo sviluppo di una soluzione basata su una rete neurale *feed-forward* per compararne il comportamento.

L'utilizzo di reti neurali di tipo *feed-forward* è stato un requisito posto dall'azienda proponente, per sperimentarne il comportamento in compiti per la quale tipicamente vengono impiegate soluzioni diverse e più avanzate.

2.2 Approfondimento del dominio applicativo

2.2.1 Esempi di applicazione

Il lavoro è incominciato a seguito di una breve introduzione da parte del tutor aziendale delle varie tecniche impiegate per l'analisi di frasi, l'elaborazione del contenuto delle *query* e per la ricerca nei corpus con l'algoritmo *Okapi BM25*. Mi è stata presentata una serie di materiali e manuali informativi riguardo l'estrapolazione di comandi da frasi in

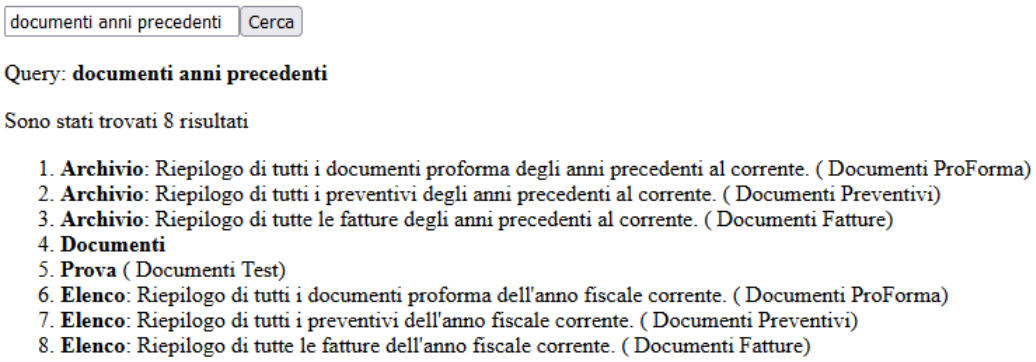


Figura 2.1: Screenshot dimostrativo dell'applicazione fittizia che utilizza BM25

linguaggio naturale e documentazioni di alcune applicazioni che hanno implementato le tecnologie descritte.
Attraverso una semplice applicazione dimostrativa, ho quindi potuto analizzare il funzionamento di un'implementazione dell'algoritmo tramite il modulo Javascript `BM25c.js`, sviluppato da Zucchetti S.p.A..

L'applicazione si compone di una pagina web con una *text-box*, tramite la quale è possibile inserire le *query*, ed un contenitore che viene popolato con il prodotto della ricerca. Il corpus è composto da 23 elementi che rappresentano le voci della barra di un menu contestuale dell'applicazione fittizia. Ciascuna voce è strutturata in più campi, tra i quali i più significativi sono la posizione dell'elemento nella gerarchia del menu (percorso dalla voce più generica a quella più specifica) ed il campo contenente una descrizione estesa della voce.

Seguono gli approfondimenti sul funzionamento delle tecniche studiate ed implementate per l'elaborazione e la ricerca delle *query*.

2.2.2 Reti neurali feed-forward

La tecnologia principale su cui si sono concentrati i due progetti è la *feed-forward neural network*, una delle prime reti neurali ideate e caratterizzata dalla struttura non-ciclica da cui è composta.
Le reti neurali trovano il loro utilizzo in numerosi campi, generalmente in compiti per cui risulta complesso formalizzare il problema o la sua soluzione. La tipologia *feed-forward* in particolare risulta efficiente nelle attività di riconoscimento di pattern.

Si tratta di un modello di apprendimento automatico formato da livelli di *neuroni* (che idealmente replicano il funzionamento dei neuroni umani) attraverso il quale l'input varia e si muove unidirezionalmente fino ad arrivare ad uno strato finale che rappresenta l'output della rete.

In particolare, ciascun neurone va a comporre uno strato e produce in output la somma dei neuroni del livello precedente, pesata secondo dei pesi associati al livello. Il risultato della combinazione lineare precedente viene poi introdotto in una funzione di attivazione che riduce i valori ad un intervallo compreso tra 0 e 1, implementando una soglia di attivazione per i neuroni a seconda della funzione utilizzata.

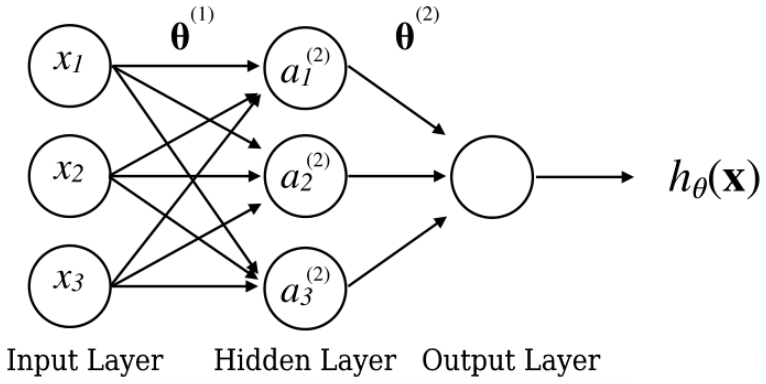


Figura 2.2: Uno schema esemplificativo di una rete neurale *feed-forward* con un solo *hidden layer*

La fase più complessa riguarda l'allenamento della rete, durante la quale lo stato interno viene corretto seguendo una procedura chiamata *backpropagation*. Viene quindi introdotto nella rete un set di input *labeled*, ovvero provvisti dell'output desiderato (modello di apprendimento detto *supervised*[G]), e se ne compara l'errore rispetto all'effettivo risultato che il modello ha previsto, tramite una specifica funzione di costo. A partire dall'*hidden layer* finale. ciascun livello corregge i propri pesi minimizzando l'errore di costo sfruttando la discesa del gradiente, un metodo iterativo per individuare il minimo locale delle funzioni, per poi propagare l'errore ai *layer* precedenti, così che ripetano a loro volta il procedimento.

Come altri modelli di apprendimento, le reti *feed-forward* sono prone al rischio di *overfit*, ovvero la possibilità che si adattino eccessivamente ai dati di allenamento e producano molti errori; oppure all'*underfit* se la rete risulta incapace di generalizzare e cogliere la relazione tra i dati. I sintomi dei problemi vanno ricercati attuando rigorose fasi di validazione ed agendo generalmente sul numero delle *features*, aumentando la grandezza del set di allenamento o agendo su specifici parametri nelle funzioni di costo utili a stabilizzare il comportamento.

Nei due progetti svolti i problemi maggiori sorti hanno riguardato l'inadeguatezza del corpus dati e i vincoli del modello che richiede l'utilizzo di input a grandezza costante.

2.2.3 BM25

L'applicazione fittizia approfondita implementa *BM25* (*Best Matching*), una funzione di *ranking* utilizzata per stimare la rilevanza e la pertinenza dei documenti di cui un corpus è composto rispetto ad una *query*.
La tecnica opera sulla base di due fattori:

* l'occorrenza dei termini della *query* nei singoli documenti;
* l'occorrenza dei termini della *query* in ciascun documento rispetto agli altri documenti.

In questo modo si penalizzano parole che risultano meno informative rispetto alle altre, come ad esempio termini presenti in grandi quantità e in molti documenti.

La procedura prevede il calcolo del valore denominato *Inverse Document Frequency* (abbreviato a *IDF* per comodità) per ogni parola presente nel corpus, mentre il punteggio di classificazione per pertinenza di ogni documento rispetto all'intera *query* viene calcolato sommando il *rango IDF* delle parole ricercate e, se presenti, classificate in precedenza.

Tra le differenti varianti esistenti è di particolare interesse la *BM25-F*, ideata per operare su documenti strutturati in campi con diversi gradi di importanza. Nell'implementazione dell'applicazione dimostrativa *BM25.js* risulta infatti utile, ad esempio, per dare maggiore rilevanza alle parole contenute dal titolo di una voce, rispetto a quelle contenute nel campo descrizione.
Prima di essere impiegati per le operazioni, i termini dei documenti e delle *query* vengono sottoposti ad una serie di elaborazioni, descritte in seguito.

Tokenization, stemming e rimozione stop-words

Ciascuna stringa viene prima di tutto *tokenizzata*, ovvero suddivisa in sezioni unitarie composte da un termine, dalla quale vengono rimossi i caratteri di punteggiatura e le *stop-words*: termini come articoli, preposizioni e parole che non risultano rilevanti per il funzionamento di *BM25*.

Sui termini viene poi effettuato lo *stemming*, operazione utile per ricavare la forma radice delle parole, utilizzando l'algoritmo *Porter Stemmer*, sviluppato da terzi [1].
A partire da parole diverse come *dispari*, *disparato*, *disparate* e *disparire*, lo *stemmer* ricava quindi la forma base (o *tema*) della parola, in questo caso *dispar*.

Queste procedure vengono eseguite sia sui documenti, per calcolare lo *score* delle parole contenute, sia sui termini delle *query*. In questo modo l'algoritmo sarà in grado di proporre i documenti contenenti parole che condividono lo *stem* con *token* della *query*, nella quale i termini appaiono però con una derivazione diversa. Il prezzo di questo comportamento è la presenza nel risultato anche di voci meno pertinenti rispetto alla *query* (maggiore *recall* ma minore *precisione*).

L'applicazione fittizia memorizza lo *score IDF* sia della radice delle parole, sia della forma completa e il punteggio viene pesato dando una maggiore rilevanza alle parole complete, in modo da "premiare" la presenza di un abbinamento esatto tra i termini nella *query* e quelli nei documenti. Questo significa che i risultati delle *query* contenenti le parole *disparate*, *disparo* o *dispari* conterranno le voci del corpus con la parola *dispari*, ma l'interrogazione contenente esattamente la parola *dispari* avrà uno *score* migliore.

2.2.4 Difetti di BM25 e soluzioni comuni

In termini computazionali, l'algoritmo risulta essere molto semplice ed efficiente. Tuttavia, rispetto ad approcci più avanzati in grado di cogliere più a fondo il significato delle *query* in linguaggio naturale, l'utente dovrà effettuare ricerche "ragionando come il computer" cercando di intuire le parole chiave migliori per ottenere la lista di risultati desiderata.
Nel caso di input più complessi in linguaggio naturale, l'algoritmo potrebbe faticare a

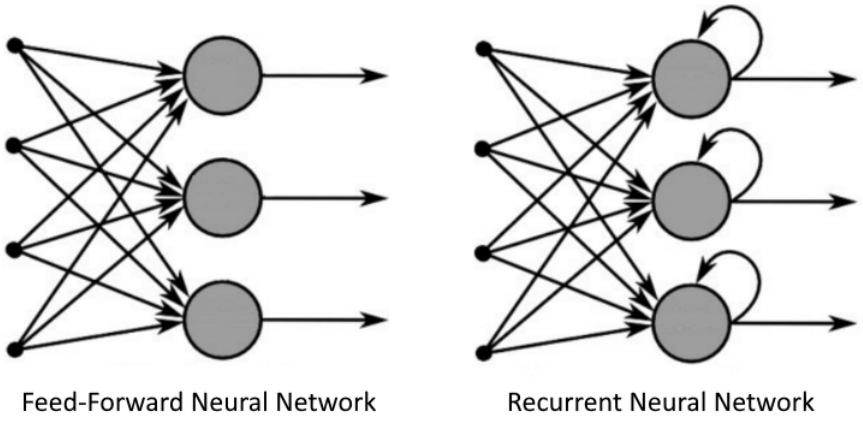


Figura 2.3: I nodi delle reti neurali ricorrenti possono presentare cicli ricorsivi

trovare risultati adatti e precisi, in particolare se confrontati con soluzioni più avanzate.

Due delle soluzioni alternative comunemente utilizzate per problemi di questo tipo prendono il nome di reti neurali di tipo ricorrente (*RNN*) e tecniche di *n-gram*.

Recurrent Neural Networks

Questo tipo di rete neurale, rispetto alle reti *feed-forward* dalla quale deriva, è caratterizzato da una struttura che permette di raggiungere uno stato interno più complesso, chiamato *memory*. Tra i nodi che la compongono è infatti prevista la presenza di collegamenti che possono dar vita a cicli. In questo modo gli input possono essere condizionati da quelli precedentemente elaborati.
La *memoria* interna, capace di conservare informazioni sugli input, rende le reti neurali ricorrenti adatte a problemi che richiedono l'elaborazione di sequenze di dati a lunghezza variabile, come ad esempio il riconoscimento della scrittura, l'*image captioning* e per l'appunto problemi riguardanti il *natural language processing*.

N-grammi

Gli *n-gram* esprimono invece la probabilità di una sequenza contigua di elementi di apparire rispetto ad un corpus di parole.
Gli elementi costitutivi della sequenza possono essere caratteri, sillabe o parole e la lunghezza di ciascuna sequenza determina lo specifico nome dell'*n-gramma*, ad esempio una sequenza lunga due parole prenderà il nome di *2-gramma* (*bi-gramma*).
Il caso appena citato, utile ad approssimare la probabilità condizionale di un elemento, stato o evento della sequenza dato l'elemento precedente, è un *n-gramma* molto utilizzato in diversi modelli statistici che prende il nome di *Markov chain*.

Entrambe le soluzioni citate permettono di adottare comportamenti più dinamici, variando a seconda dell'input e risultando opzioni alternative molto valide per la soluzione al problema trattato nel primo progetto.

Libreria ml.js

Per verificare l'efficacia di una rete *feed-forward* in questo contesto, il tutor aziendale ha richiesto di sperimentare una sua implementazione utilizzando *ml.js* [2].
La libreria è sviluppata in Javascript e dispone di numerose classi per implementare tecniche di analisi numerica e *machine learning* (sia di tipo *supervised* che *unsupervised*), tra le quali sono presenti modelli di *Principal Component Analysis (PCA)*, *K-Nearest Neighbor (KNN)*, *Decision Tree Classifier*, modelli di regressione e reti neurali.
Dispone inoltre di alcune classi di supporto e utilità, ad esempio per decomporre matrici, gestire e ricavare informazioni da *array* o effettuare *cross-validation*.

Nei progetti sviluppati è stata utilizzata la classe `FeedForwardNeuralNetworks`, utile alla configurazione di una rete *feed-forward* e alla sua gestione nelle fasi di *training* e *predict*.

La classe permette di modificare i seguenti parametri:

* **hiddenLayers** *array* contenente le dimensioni degli *hidden layers*;
* **iterations** numero di iterazioni a *training time*;
* **learning rate** grandezza correzione da propagare durante la *backpropagation*;
* **regularization** parametro per stabilizzare l'apprendimento;
* **activation** funzione di attivazione tra *tanh*, *identity*, *logistic*, *arctan*, *softsign*, *relu*, *softplus*, *bent*, *sinusoid*, *sinc*, *gaussian*, *parametric-relu*, *exponential-relu*, *soft-exponential*;
* **activationParam** eventuale parametro per la funzione di attivazione.

Permette inoltre di utilizzare e gestire la rete con i seguenti metodi:

* **load**(model: *object*) carica un modello di rete neurale, utile per la deserializzazione;
* **predict**(features: (*Array* | *Matrix*)) predice l'output a partire da un set di *features*;
* **toJSON()** serializza il modello in *JSON*;
* **train**(features: (*Matrix* | *Array*), labels: (*Matrix* | *Array*)) allena la rete con le *features* fornite e le relative *labels*.

Il calcolo dell'accuratezza dei modelli è stato invece ricavato usando le funzioni della classe `ConfusionMatrix`.

2.3 Sviluppo del prodotto

2.3.1 Sviluppo del corpus e one-hot encoding

Dopo aver approfondito le tecnologie coinvolte ed il dominio applicativo, ho proceduto sviluppando un corpus di frasi da utilizzare per l'allenamento della rete. L'intenzione era quella di insegnare alla rete quale voce del menu proporre al presentarsi di specifiche frasi o insiemi di parole.

Stesura corpus dati

Per ogni possibile risultato (una voce del menu) è quindi stata stilata ed assegnata una lista di frasi in linguaggio naturale, ipotizzando il tipo di *query* inserite dall'utente. Le frasi prendono la forma di "*Visualizza i preventivi degli anni precedenti al corrente*" o "*Mostra l'elenco di tutti i clienti*".

```
const FRASI =
  ‘ [
            {
              "intent": "Archivio",
              "phrases": [
                "Riepilogo dei documenti proforma degli anni
                    precedenti al corrente",
                "Visualizza fatture degli anni precedenti al
                    corrente",
                "Mostra riepilogo fatture degli anni precedenti
                    al corrente",
                "Riepilogo di tutti i preventivi degli anni
                    precedenti al corrente",
                ...
                },
            {
              "intent": "Clienti",
              "phrases": [
                "Mostra l’elenco di tutti i clienti",
                "Visualizza l’elenco dei i clienti",
                "Visualizza l’elenco dei clienti principali",
                "Mostra i clienti",
                "Mostrami i clienti",
                "Mostra la lista dei clienti",
            ...
```

Per ciascuna frase viene memorizzata l'associazione con il suo *intent*, ovvero la relativa voce del menu alla quale fa riferimento.

```
const INTENT = ‘ {
                  "0" : "Invalid",
                  "1" : "Archivio",
                  "2" : "Elenco",
                  "3" : "Clienti",
                  "4" : "Dipendenti"
                }‘;
```

Il corpus aveva quindi bisogno di essere presentato in un formato comprensibile alla rete; ho quindi sviluppato una classe per codificare ciascuna voce in forma *one-hot*, descritta in seguito.

One-hot encoding

La codifica *one-hot* prevede, per ciascun input, la costruzione di un vettore contenente solamente i valori zero o uno i quali indicano la presenza (1) o l'assenza (0) nell'input dei differenti termini che compongono l'universo dei possibili valori. In questo caso le righe *one-hot* indicano, per ciascuna frase, quali parole fanno parte del dizionario che

compone il corpus.
Questa tecnica è fondamentale nei modelli di *machine learning*, che calcolano predizioni con funzioni matematiche a partire da valori di tipo numerico, per trasformare valori di tipo nominale in valori comprensibili dagli algoritmi di apprendimento automatico. L'utilizzo di due soli numeri per codificare la presenza o l'assenza di un valore impedisce inoltre che una rete individui nella relazione d'ordine tra gli altri numeri una discriminante (*bias*) che tuttavia è indesiderata.

La classe *One-Hot* sviluppata contiene i metodi per preparare i dati seguendo le fasi di *tokenizzazione*, *stemming* e rimozione di *stop-words* descritte in precedenza, memorizzare la lista di frasi codificate in forma *one-hot* e serializzare o importare una matrice *one-hot* in formato *JSON*.

L'associazione *intent-frase* codificata è stata quindi utilizzata per allenare la rete e configurare i suoi nodi e pesi in modo che a *prediction time*, introdotta una *query* anch'essa in forma *one-hot vector* rispetto all'insieme di parole del corpus, stabilisca per ogni possibile voce del menù la probabilità di essere quello desiderato.

2.3.2 Sviluppo della rete neurale

Ho proseguito il lavoro con la definizione di una prima architettura per la rete neurale. I vincoli del caso hanno configurato un problema di classificazione *multi-classe*: ogni possibile voce del menu è una classe e l'output desiderato è la probabilità di ciascuna classe di essere quella corretta.
Per le funzioni di attivazione dei livelli intermedi e del livello output è stata quindi utilizzata la funzione *logistica*.

Sono stati definiti due *hidden-layer* da 20 neuroni ciascuno, il *learning rate* è stato fissato al valore 0.01. Sono stati sperimentati diversi valori per le iterazioni di allenamento, da poche centinaia fino a 2000 *epochs*^{G}.

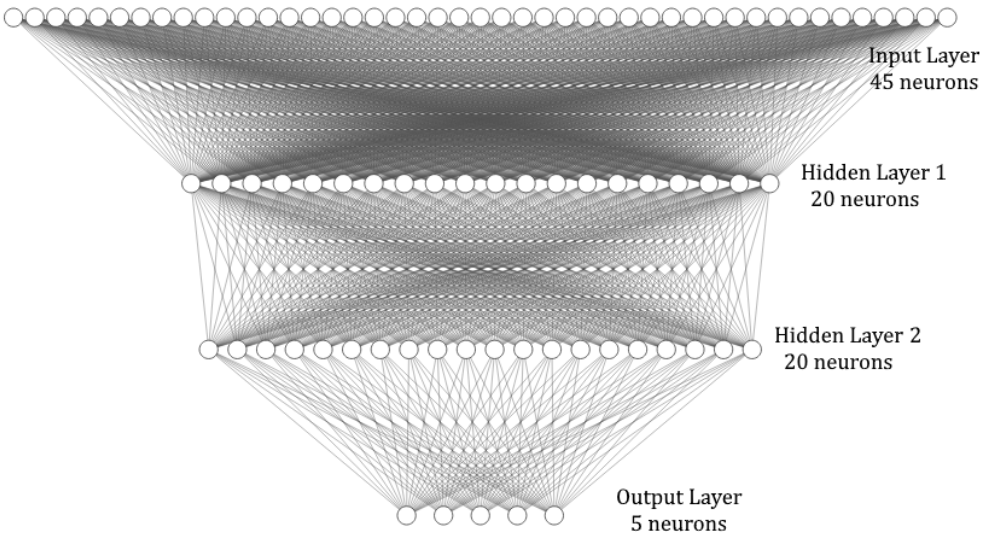


Figura 2.4: Schema rete neurale per la ricerca di query in natural language

Nei primi tentativi il comportamento della rete è risultato poco preciso e apparentemente arbitrario. L'output si rivelava talvolta completamente errato e sensibile anche a parole non pertinenti al contesto (ad esempio produceva risultati la parola "gatto").

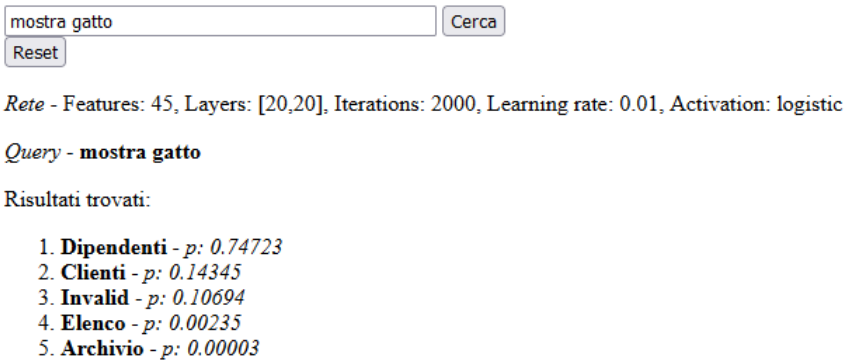


Figura 2.5: Ricerca con parole non affini al corpus; i risultati vengono ordinati secondo probabilità decrescente, stampata accanto ciascuna voce

Le prime modifiche apportate hanno riguardato la variazione del numero di iterazioni con lo scopo di allenare più a lungo la rete.

In una discussione dei risultati assieme al tutor aziendale è emersa la possibilità di sperimentare l'implementazione e l'allenamento con una classe aggiuntiva "*Invalid*", intesa per riconoscere gli input indesiderati, come ad esempio *query* composte da singole parole.
Il comportamento ha ancora una volta portato a risultati negativi: la *label* speciale *Invalid*, avendo imparato tutte le parole del vocabolario, seppur singolarmente, risultava essere il risultato più probabile ad ogni *query*.

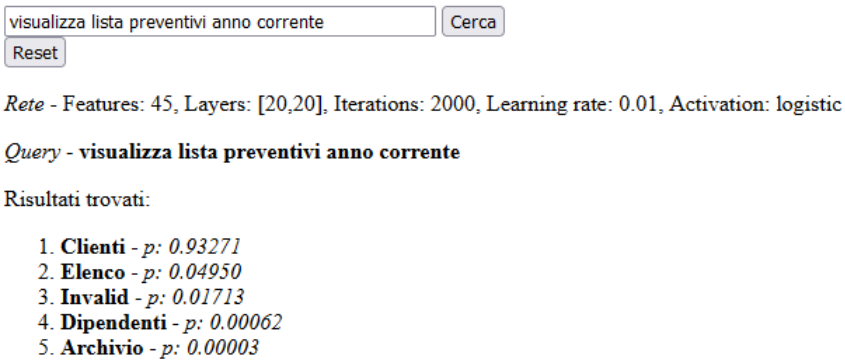


Figura 2.6: Screenshot dimostrativo della ricerca utilizzando la rete neurale

Si è quindi allenata la rete con una classe per ogni caso indesiderato, introducendo le singole parole del dizionario una alla volta, tentativo che ha prodotto ancora risultati non corretti, oltre a risultare una soluzione poco pratica e più onerosa, rendendo

necessario ulteriore tempo di allenamento.

Al termine del tempo previsto per lo sviluppo del progetto, dati gli esiti dei tentativi compiuti, è stato necessario concludere che le dimensioni limitate del corpus di allenamento non permettono la realizzazione di una soluzione soddisfacente per il problema.

2.3.3 Documentazione e test automatici

Allo scopo di verificare il corretto funzionamento del codice è stata implementata una suite di test automatici utilizzando il framework Javascript *Jest* [3].
Inoltre per facilitare un eventuale ulteriore sviluppo da parte dell'azienda è stato utilizzato *JSDoc* [4] per produrre ed estrarre in modo automatico la documentazione del codice.

La maggior parte dei file servono all'implementazione della rete e alla visualizzazione dei risultati; per questo la documentazione e la suite di test sono state realizzate solamente per la classe *One-Hot*. In particolare, i test operano verificando che i metodi della classe *OneHot* funzionino a dovere comparandone il comportamento con gli oggetti fittizi creati da *Jest*.

Il report redatto per l'azienda alla fine del progetto riguarda invece le premesse ed il funzionamento dell'intero codice prodotto.

2.4 Resoconto finale

2.4.1 Prodotti ottenuti

I prodotti ricavati al termine del lavoro sono i seguenti:

* una pagina *HTML* per eseguire le *query* con la rete neurale;
* una pagina *HTML* per eseguire le *query* contemporaneamente sulla rete neurale e con l'algoritmo *BM25* per compararne i risultati;
* una classe contenente metodi di utilità per gestire gli *one-hot vector*;
* una suite di test per verificare il funzionamento della classe *OneHot*;
* documentazione del codice della classe *OneHot*.

Inoltre è stato redatto un breve documento riguardante la spiegazione dei problemi trattati, il funzionamento del codice prodotto e i risultati ottenuti.

2.4.2 Conclusione

I risultati ottenuti hanno confermato l'ipotesi posta inizialmente, scettica rispetto alla possibilità di portare a termine il compito con buona accuratezza.
Si può concludere che il tipo di rete neurale utilizzato, ma soprattutto le dimensioni limitate del corpus di allenamento non permettono la realizzazione di una soluzione soddisfacente per il problema.

Riguardo a soluzioni alternative, ipotizzando di voler procedere ugualmente con un sistema *BM25*, questo potrebbe eventualmente venire implementarlo chiedendo all'utente di riformulare la *query* in caso si riceva una richiesta complessa o ambigua (comportamento impiegato da *ELIZA*, uno dei primi *chatterbot* sviluppati).
Un'altra possibilità è quella di sfruttare le *query* input al *BM25* per produrre progressivamente nel tempo un corpus di dimensioni sufficienti ad allenare un sistema più complesso.

Per questo progetto risultano quindi più adatte al caso le soluzioni discusse nella sezione precedente, dotate di un comportamento più dinamico e capaci di raggiungere maggiore efficacia.

Capitolo 3

POS tagging con reti neurali feed-forward

3.1 Analisi del problema

La seconda parte dello stage è stata dedicata allo studio delle tecniche di *part-of-speech tagger* e ad una implementazione di queste tramite rete neurale *feed-forward*. Il compito ha previsto l'approfondimento dei metodi comuni e la realizzazione di un modello che si occupa quindi di stabilire, a partire da una frase in linguaggio naturale, il ruolo grammaticale (marcatore o *tag*) di ciascuna delle parole della quale una frase è composta. Il corpus di allenamento utilizzato, e quindi si assume anche l'input che viene *taggato*, è composto da frasi in lingua italiana.
Il prodotto di questi processi permette generalmente di ricavare informazioni sulla semantica delle frasi, risultando utile in attività di ricerca ed analisi di testi o comandi vocali.

L'utilizzo di una rete neurale di tipo *feed-forward* è stato un requisito posto dall'azienda proponente, per sperimentarne il comportamento in compiti per la quale tipicamente vengono impiegate soluzioni diverse e più avanzate.
Gran parte della difficoltà è stata riscontrata nella scelta delle *feature* da introdurre come input nella rete e nella scelta del comportamento del modello in situazioni di ambiguità. Infine è stata implementata una soluzione che ha permesso di raggiungere una precisione accettabile.

3.2 Approfondimento del dominio applicativo

La prima attività che si è svolta nella seconda parte dello stage è stata una presentazione da parte del tutor aziendale delle tecnologie coinvolte nel *tagging* di frasi in linguaggio naturale. Inoltre è stato introdotto il materiale di lavoro e sono state condivise delle risorse utili allo studio di questo.

Ho quindi proceduto approfondendo autonomamente le tecniche di *part-of-speech tagging*.

Part-of-speech tagging

Le tecniche di *path-of-speech tagging*, chiamate anche *grammatical tagging*, sono utili all'individuazione della categoria lessicale di parole nel contesto in cui appaiono. Nei processi che utilizzano corpus in linguaggio naturale, il *POS tagging* viene impiegato nelle prime fasi di elaborazione del testo per produrre un'analisi preliminare utile ad una certa varietà di attività successive.
Possibili applicazioni possono essere nella traduzione tra lingue, l'analisi del sentimento espresso o l'estrazione di informazioni e comandi da frasi ottenute in input ad esempio da assistenti vocali.

Le difficoltà principali del problema stanno nell'identificazione dell'utilizzo di una parola nel suo esatto contesto, dal momento che molti termini possono assumere significati e ruoli diversi a seconda delle circostanze.
Un esempio evidente in lingua inglese dell'ambiguità che possono sorgere in certe frasi è la proposizione

"The sailor dogs the hatch"

dove la parola *dogs*, ordinariamente utilizzata come nome, assume il ruolo di verbo (*"to dog"*: assicurare, fissare) per via delle parole circostanti.

La marcatura può avvenire in diversi modi: seguendo un modello *stocastico*, basato sulla frequenza e probabilità di parole e sequenze di *tag* di apparire, implementato ad esempio dal modello di *Markov*; oppure sfruttando tecniche *rule-based*, come il metodo di *Brill*, che fanno utilizzo di regole determinate manualmente.

Brill Tagger

Una delle tecniche di *part-of-speech tagging* approfondite, che all'interno del secondo progetto è stata utilizzata per produrre uno dei corpus utilizzati, è quella di *Brill*.
Il metodo prevede innanzitutto che le parole di un corpus vengano marcate secondo certe regole predefinite, come ad esempio assegnando il *tag* con la quale si presentano più di frequente, la presenza o meno della iniziale maiuscola o il tipo di suffissi e prefissi. In seguito il *tag* provvisorio viene progressivamente corretto applicando delle regole (per questo il metodo viene detto *transformation-based* ed *error-driven*) basate sul contesto locale della parola. Il contesto è definito dalle parole immediatamente precedenti e successive.
Le regole prendono la forma di

tag1 $\rightarrow$ *tag2* `IF` *Condizione*

ovvero al verificarsi della condizione, *tag1* viene sostituito con *tag2*; e queste vengono iterate fino al raggiungimento della condizione nella quale non rimangono regole applicabili.
Per raggiungere una precisione soddisfacente il metodo richiede la definizione di una certa quantità di regole, solitamente nell'ordine di qualche centinaia, che possono essere definite manualmente o con l'ausilio di modelli di *machine learning*.

Materiale

L'allenamento e il test della rete sono stati effettuati utilizzando due corpus di dati forniti dal tutor:

* *Morph-it!* [5]: dizionario italiano contenente per ciascuna parola gli attributi morfologici estratti tramite regole basate su espressioni regolari. Il testo con la quale è stato elaborato il dizionario proviene da un *dataset* di articoli de *la Repubblica*. Le voci sono 500'000 e sono presenti diverse variazioni e coniugazioni per ciascuna parola.

* Set di circa 99'900 frasi *POS taggate* provenienti da articoli de *la Repubblica*.

3.3 Sviluppo del prodotto

3.3.1 Elaborazione corpus e classi di supporto

Il tutor aziendale ha fornito diverse varianti del corpus *Morph-it!*, derivanti da precedenti progetti dell'azienda. Ciascuna contiene derivazioni ed elaborazioni differenti delle parole, come ad esempio forme base, *tag* non normalizzati, parole duplicate o dettagli grammaticali superflui.
Data la dispersione delle informazioni e la presenza di dati non indispensabili ho cominciato la fase di sviluppo con la realizzazione di una serie di script *Javascript* per la configurazione della codifica e per il riordino del formato e del contenuto dei file. In particolare ho prodotto un file che ha integrato il dizionario *Morph-it!* con le parole del corpus *taggato* de *la Repubblica*.
Per ogni parola sono state inoltre contate le occorrenze di ciascun *tag* con la quale appariva ed è stato prodotto un file utile a memorizzare i *tag* raccolti.
I marcatori utilizzati per lo sviluppo della rete sono quindi stati i seguenti:

PROPN nome proprio,

VERB verbo,

ADP preposizione,

NOUN nome,

ADJ aggettivo,

NUM numero,

PUNCT punteggiatura,

AUX verbo ausiliario,

DET articolo determinativo,

CCONJ congiunzione coordinante,

PRON pronome,

ADV avverbio,

SCONJ congiunzione subordinante.

Classe Object

Il dizionario integrato ed il file con la lista totale dei *tag* presenti, così come altri file di supporto con dizionari o prodotti intermedi degli script, sono tutti stati memorizzati serializzando oggetti *Javascript* di tipo `Object` in formato *JSON*.
Questo perché già dalle prime ottimizzazioni della logica del codice è stato evidente che, specialmente con i file più grandi, sarebbe stato essenziale minimizzare le letture e le iterazioni complete dei corpus. Si è quindi fatto un largo uso della classe nativa `Object`, utilizzata per memorizzare dizionari di dati strutturati e che essendo implementata come *hashmap*, consente l'accesso in tempo costante ai propri membri.

Dopo aver consultato alcune risorse che trattavano soluzioni di problemi affini [6][7][8][9], generalmente con altri vincoli e lingue diverse, ho potuto cominciare lo sviluppo della rete neurale.

3.3.2 Sviluppo della rete

Le circostanze hanno delineano un problema di classificazione di tipo *multi-classe* in cui l'output desiderato è la probabilità di ciascun *tag* di essere il descrittore corretto di una parola nel contesto (*window*) in cui è situata. Per le funzioni di attivazione dei livelli intermedi e del livello output è stata utilizzata la funzione *tanh*.

Sistema a *sliding window*

Nel corso dello sviluppo sono stati sperimentati diversi set di *features*, ovvero delle informazioni selezionate per comporre l'input, e nella logica generale del modello.
Dato il vincolo delle reti *feed-forward* di supportare un input a grandezza definita e costante, da subito ho ipotizzato un sistema di *"sliding windows"*, ovvero un set di *features* provenienti dalle parole circostanti al termine da *taggare*, in modo da codificare il suo contesto ed ottenere in output il marcatore corretto.

La previsione dei *tag* di un'intera frase richiede che questa venga scorsa e le finestre ricavate vengano proposte alla rete una alla volta.

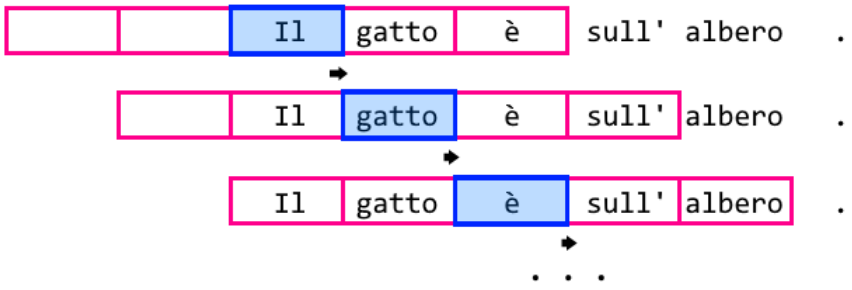


Figura 3.1: Dimostrazione dello scorrimento di una *window* di grandezza 5 (in rosa) che predice il marcatore della parola centrale (in blu)

Si è inoltre ipotizzata l'aggiunta di *tag* speciali per segnalare l'inizio e la fine della frase. In seguito si è poi proceduto assumendo che questi casi venissero codificati introducendo il vettore delle *features* vuoto (nessun *tag* assegnato) nello slot interessato della *window*.

Rete 1x100 neuroni, *window size* 3 e 5

Il primo sistema di *word embedding* utilizzato, allo scopo di trasformare le parole della *window* in un valore numerico comprensibile alla rete neurale, si è basato sulle probabilità dei *tag* con cui ogni parola si può presentare.
La soluzione era un tentativo di creare una sorta di "impronta" di ciascuna parola che compone la *window*, mentre una frase sarebbe stata composta da una sequenza di "impronte".

I primi tentativi hanno coinvolto una rete neurale formata da un livello di 100 neuroni ed una *sliding window* di grandezza 3 e 5 parole e non hanno prodotto buoni risultati, né introducendo un'*array* contenente le probabilità dei *tag* con cui le parole della *window* si possono presentare, né introducendo gli stessi valori ma sotto forma di *one-hot vector* (valore 1 se la parola ha una probabilità non nulla di apparire con un certo marcatore in un certo contesto).

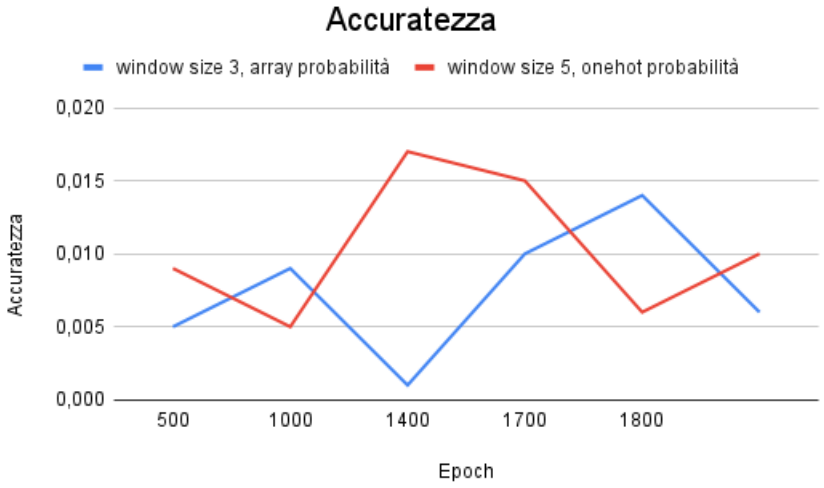


Figura 3.2: Accuratezza rete 1x100 neuroni

Esempio dei primi due tentativi di *word embedding* della parola ***morso*** (non dell'intera *window*), che secondo il dizionario appare con i marcatori `NOUN 74%, VERB 21%, ADJ 5%`:

* vettore contenente le probabilità per una parola di presentarsi con i *tag*

morso: `[0.0, 0.21, 0.0, 0.74, 0.05, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]`

* *one-hot vector* del vettore delle probabilità

morso: `[0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]`

Rete 1x50 neuroni, *window size* 5, sistema con permutazioni

Si è quindi ipotizzato di creare per ciascuna parola una serie di *one-hot vector*, ciascuno utile a rappresentare la presenza di uno (e uno solo) dei *tag* con cui la parola si può presentare.
È stato quindi necessario introdurre nella rete tutte le permutazioni degli *one-hot vector* prodotti da ciascuna *window*, con lo scopo di presentare alla rete tutti i casi possibili. Inoltre rispetto ai primi tentativi, è stato aumentata a 5 la grandezza della *window*, in modo da cogliere una parte più grande del contesto.

morso: `[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`,
`[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`,
`[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]`

Vengono quindi presentate alla rete le permutazioni (una alla volta) tra i vettori prodotti dalla parola **morso** e quelli prodotti dalle altre parole della *window* di cui fa parte.
Per questi tentativi è stata sperimentata una rete formata da un livello con 50 neuroni e una *window* ampia 5 parole.
Una volta sottoposta ogni permutazione possibile al modello, il risultato considerato valido è quello che presenta l'indice di *confidence* migliore.

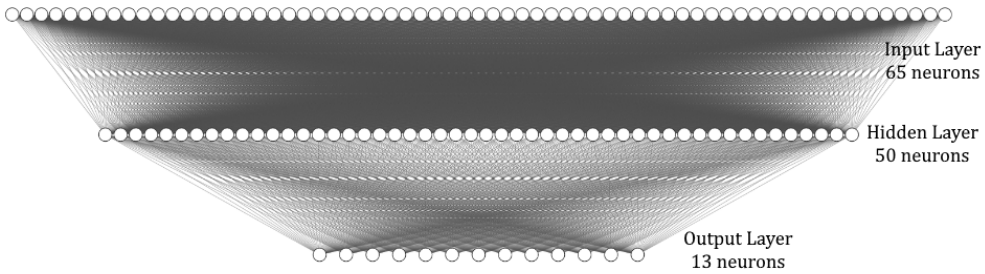


Figura 3.3: Schema rete neurale per il part-of-speech tagging

3.3.3 Risultati raggiunti

Da un esempio di frase come la seguente:

`Il gatto è salito sull' albero .`

a *prediction time* verrebbe quindi prodotta la rispettiva lista di *POS tag*, nell'esempio la seguente:

`DET NOUN AUX VERB ADP NOUN PUNCT`

Quest'ultima soluzione ha introdotto una maggiore complessità nella logica degli script e presenta dei limiti nelle performance ma ha permesso di ottenere risultati migliori e più soddisfacenti.

È stato possibile raggiungere un'accuratezza del 60%. Sono stati sperimentati diversi tentativi variando i parametri a disposizione e i risultati migliori sono stati raggiunti allenando la rete su 1232 *training sample* (ovvero singole *window*, ricavate da un totale di 40 frasi) per 2000 *epochs* con un singolo *layer* di 50 neuroni.

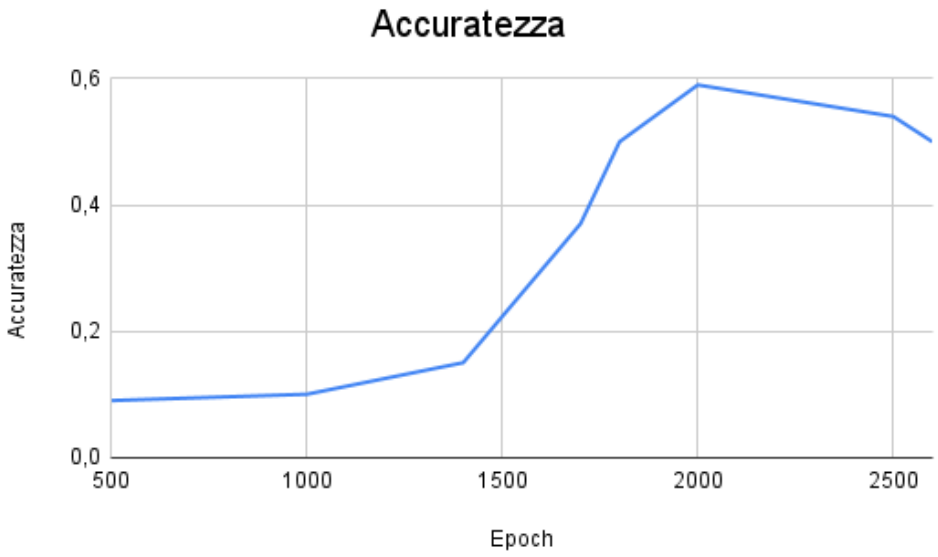


Figura 3.4: Rete 1x50 neuroni, *window size* 5, permutazioni delle *features*

3.4 Resoconto finale

3.4.1 Prodotti ottenuti

Lo sviluppo del *POS Tagger* ha prodotto:

* degli script Javascript di utilità per normalizzare i corpus di dati;
* dei file dizionario che raccolgono informazioni ricavate dai vari corpus, utili per l'utilizzo della rete;
* degli script per creare ed allenare la rete sui corpus di dati utilizzando la libreria *ml.js*.

Inoltre è stato redatto un breve documento riguardante la spiegazione dei problemi trattati, il funzionamento del codice prodotto e i risultati ottenuti.

3.4.2 Conclusione

Il modello sviluppato ha permesso di raggiungere buoni risultati considerando i limiti dei vincoli posti, in particolare l'utilizzo di una rete *feed-forward* e il tipo di *word embedding* impiegato.

Ci sono tuttavia diverse possibilità per tentare di migliorare ulteriormente l'accuratezza dei risultati.
È possibile sperimentare ad esempio soluzioni alternative agendo sui parametri della rete, oppure migliorando la qualità del corpus dati (quello utilizzato presenta degli errori di *tagging*). Si potrebbe inoltre intervenire scegliendo delle *features* diverse, ad esempio utilizzando relazioni probabilistiche tra le parole e i *tag*, oppure agendo cambiando la logica dello script.

Capitolo 4

Conclusioni

4.1 Obiettivi raggiunti

Verrà di seguito riportata la tabella contenente lo stato di completamento degli obiettivi concordati con il tutor esterno nel piano di lavoro dello stage.
Si farà riferimento ai requisiti secondo le seguenti notazioni:

* **O** per i requisiti obbligatori, vincolanti in quanto obiettivo primario richiesto dal committente;
* **D** per i requisiti desiderabili, non vincolanti o strettamente necessari, ma dal riconoscibile valore aggiunto;
* **F** per i requisiti facoltativi, rappresentanti valore aggiunto non strettamente competitivo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo del requisito.

Tutti gli obiettivi obbligatori sono stati soddisfatti ad eccezione di `005.1`. La libreria Java *Apache OpenNLP* infatti non è stata approfondita in quanto il progetto si è concentrato sull'utilizzo del linguaggio Javascript della libreria *ml.js*.

Codice	Descrizione	Raggiunto
O01	Ricerca e studio tecniche tipiche usate per la classificazione di documenti	Sì
O02	Ricerca e studio tecniche tipiche usate per lo stemming di parole	Sì
O03	Sviluppo di una rete neurale per il riconoscimento di frasi in linguaggio naturale	Sì
O04	Confronto dell'efficienza della rete neurale con i metodi funzionali	Sì
O05.1	Studio della libreria Apache OpenNLP	No
O05.2	Studio del sistema di tagging di Brill	Sì
O06	Stesura di test e documentazione di resoconto sulle attività svolte	Sì
D01	Sviluppo POS Tagger in Javascript	Sì
F01	Sviluppo microservizio per ottimizzare comunicazione browser-server	No
F02	Ottimizzazione architettura rete neurale e gestione dei casi di ambiguità	Sì

Tabella 4.1: Obiettivi raggiunti

4.2 Risultati ottenuti

Al termine del tirocinio sono state sviluppate due reti neurali, degli script per preparare e processare i dati necessari e una suite di test per verificare il corretto funzionamento di alcune porzioni di codice. Inoltre è stata redatta la documentazione dei sorgenti e dei report finali sul loro funzionamento.

Nello specifico, la soluzione realizzata per la ricerca di *query* in linguaggio naturale non ha prodotti grandi risultati, in linea con l'ipotesi pessimistica iniziale dovuta ai limiti della tecnica impiegata e delle risorse disponibili.

Il modello di *POS tagger* ha invece permesso di raggiungere buoni risultati, lasciando tuttavia un certo margine di miglioramento colmabile agendo sulla logica ed il corpus impiegati dalla soluzione.

4.3 Conoscenze acquisite

Il progresso maggiore è stato fatto nell'impiego della libreria *ml.js*, che non avevo mai utilizzato prima, e più in generale anche nell'uso di Javascript per il tipo di applicazione che ho avuto modo di farne. L'utilizzo del linguaggio infatti è stato differente rispetto alle mie precedenti esperienze circoscritte allo sviluppo di piccoli *script* per pagine web, ed ho quindi approfondito nuovi tipi di oggetti e operatori, funzionalità e diverse proprietà di cui dispone. Ho trovato interessante anche approfondire *node.js* che, seppur impiegato in modo limitato, non avevo mai utilizzato.

Riguardo al tema principale dei progetti svolti, ho acquisito più familiarità nell'ambito delle reti neurali *feed-forward* e ne ho ricavato un'interessante panoramica su alcune tecniche di *word embedding* e *natural language processing*.

Inoltre ho avuto modo di affrontare autonomamente una ricerca approfondita che ha previsto l'analisi di materiale e documentazione ma soprattutto la consultazione di *paper* scientifici, svolgendo un lavoro che non avevo mai compiuto in precedenza.

4.4 Valutazione personale

L'esperienza portata a termine nelle otto settimane di *stage* si è rivelata molto interessante e formativa. Nel corso del lavoro sono incorso in molti problemi mai affrontati ed ho commesso vari errori che hanno rallentato lo sviluppo, alcuni a causa di una debole progettazione iniziale, altri per la mancanza di pratica nell'utilizzo di certi strumenti o per la difficoltà che talvolta si trova nell'interpretazione dei comportamenti e dei risultati di una rete neurale.
Tuttavia con ulteriore approfondimento e con l'aiuto del tutor aziendale ne ho ricavato lezioni molto utili sulle migliori pratiche da seguire e su quali comportamenti e soluzioni invece evitare.

A prescindere dai risultati ottenuti, ho potuto di lavorare autonomamente su problemi mai affrontati in un contesto lavorativo nuovo con l'opportunità di mettere in pratica le conoscenze apprese durante il percorso di studi, oltre a quella di acquisire una maggiore sicurezza nell'utilizzo delle tecnologie coinvolte.

L'esperienza ha inoltre rinnovato il mio interesse iniziale verso l'argomento, che in futuro non mancherò senz'altro di sviluppare ed approfondire ulteriormente.

Appendice A

Esempio corpus utilizzati

A.1 Porzione esemplificativa del corpus *Morph-it!*

```
morbi_NOUN
morbida_ADJ
morbide_ADJ
morbidezza_NOUN
morbidezze_NOUN
morbidi_ADJ
morbidissima_ADJ
morbidissime_ADJ
morbidissimi_ADJ
morbidissimo_ADJ
morbido_ADJ
morbilli_NOUN
morbillo_NOUN
morbo_NOUN
```

A.2 Porzione corpus articoli di *la Repubblica POS-taggati*

```
Il_DET gatto_NOUN selvatico_ADJ sardo_NOUN (_PUNCT "_PUNCT Felis_PROPN lybica_ADJ
sarda_ADJ "_PUNCT Lataste_PROPN ,_PUNCT 1885_NUM )_PUNCT è_VERB un_DET felino_NOUN
carnivoro_ADJ ._PUNCT _VERB insieme_ADV alla_ADP volpe_NOUN sarda_ADJ il_DET
più_ADV grande_ADJ mammifero_ADJ predatore_NOUN in_ADP Sardegna_PROPN ._PUNCT
La_DET collocazione_NOUN sistematica_ADJ del_ADP gatto_NOUN selvatico_ADJ
sardo_NOUN _VERB ancora_ADV incerta_ADJ e_CCONJ controversa_ADJ ,_PUNCT
perciò_SCONJ ricorrono_VERB nomi_NOUN scientifici_ADJ riconducibili_ADJ
```

a_ADP tre_NUM differenti_ADJ posizioni_NOUN tassonomiche_ADJ ._PUNCT

Glossario

Corpus raccolta di documenti e informazioni correlate, genericamente dallo stesso autore. 1, 30

Epoch iterazione (che comprende le fasi di *feedforward* e *backpropagation*) dell'intero set dati di allenamento in una rete neurale. 12, 30

Information Retrieval insieme delle tecniche utilizzate per gestire la rappresentazione, la memorizzazione, l'organizzazione e l'accesso a di informazioni. 1, 30

Javascript linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella *programmazione web* lato client. 2, 30

Linguaggio naturale linguaggio *scritto o parlato* utilizzato dagli *umani* per comunicare. 1, 30

Machine Learning branca dell'*intelligenza artificiale* che utilizza metodi statistici per migliorare la performance di un algoritmo nell'identificare *pattern* nei dati, imparando da questi a svolgere delle funzioni piuttosto che attraverso la programmazione esplicita. 1, 30

Natural Language Processing processo di trattamento automatico mediante un calcolatore elettronico delle informazioni scritte o parlate in *linguaggio umano*. 1, 30

Part-Of-Speech Tagger processo di marcatura (*tagging*) di una parola in un testo come corrispondente a una parte particolare del discorso, basata sia sulla sua definizione che sul suo contesto nella frase. 1, 30

Reti Neurali Feed-Forward semplice modello di *rete neurale artificiale* dove le connessioni tra le unità *non formano cicli*, differenziandosi dalle reti neurali ricorrenti. 1, 30

Supervised tipologia di tecniche di apprendimento automatico e intelligenza artificiale che migliorano la propria accuratezza correggendo i pesi interni rispetto ad un set di input e output corretti (*labeled dataset*) forniti durante la fase di apprendimento. 7, 30

Bibliografia

[1] *JavaScript Snowball Stemmers - Github Repository*. URL: https://github.com/mazko/jssnowball (cit. a p. 8).

[2] ml.js. *ml.js - Library Documentation*. URL: https://mljs.github.io/feedforward-neural-networks/ (cit. a p. 10).

[3] *Jest - Javascript Testing Framework*. URL: https://jestjs.io/ (cit. a p. 14).

[4] *JSDoc - Github Repository*. URL: https://github.com/jsdoc/jsdoc (cit. a p. 14).

[5] Eros Zanchetta e Marco Baroni. «Morph-it! A free corpus-based morphological resource for the Italian language». In: (2005) (cit. a p. 19).

[6] Nuno Marques et al. «Neural Networks, Part-of-Speech Tagging and Lexicons». In: (feb. 1970) (cit. a p. 20).

[7] Nuno Marques e Gabriel Lopes. «A Neural Network Approach to Part-of-Speech Tagging». In: (lug. 2008) (cit. a p. 20).

[8] Jan A. Botha et al. «Natural Language Processing with Small Feed-Forward Networks». In: (2017). DOI: 10.48550/ARXIV.1708.00214. URL: https://arxiv.org/abs/1708.00214 (cit. a p. 20).

[9] Helmut Schmid. «Part-of-Speech Tagging with Neural Networks». In: *CoRR* abs/cmp-lg/9410018 (1994). arXiv: cmp-lg/9410018. URL: http://arxiv.org/abs/cmp-lg/9410018 (cit. a p. 20).

[10] *MDN - Mozilla Developers Web Docs*. URL: https://developer.mozilla.org.

[11] *W3Schools - Javascript Tutorial*. URL: https://www.w3schools.com/js/.

[12] *Standford University RCPedia - Word Embeddings*. URL: https://rcpedia.stanford.edu/topicGuides/textProcessingWord_Embeddings.html.

[13] Achmad F. Abka. «Evaluating the use of word embeddings for part-of-speech tagging in Bahasa Indonesia». In: (2016), pp. 209–214. DOI: 10.1109/IC3INA.2016.7863051.