Document Classification con HAN, DocBERT e LSTM

Francesco Gradi
Matricola: 7017190
francesco.gradi@stud.unifi.it

Abstract

Nell'ambito del Natural Language Processing, un task molto importante riguarda la classificazione automatica dei documenti dopo aver addestrato un modello tramite apprendimento supervisionato. All'interno di questo progetto sono stati replicati, dopo una breve recensione della letteratura relativa, alcuni dei più recenti e promettenti lavori di Document Classification mediante tecniche di Deep Leaning, le quali utilizzano reti ricorrenti o meccanismi di attenzione. Sono stati valutati su 3 diversi datasets di Sentiment Analysis i modelli HAN, BERT, LSTM e una modifica di quest'ultimo tramite la tecnica di addestramento conosciuta come Knowledge Distillation. I risultati ottenuti sono stati generalmente in linea con quelli descritti in altri papers citati, con il modello BERT che raggiunge un'accuracy del 57.7% su IMDB e 77.4% su Yelp 2014. Sono stati analizzati anche i meccanismi di attenzione previsti in HAN mediante delle rappresentazioni visive e è stata fatta un'analisi quantitativa sui tempi di inferenza con le varie configurazioni. Tutto il codice è open source e reperibile su GitHub.

1. Introduzione

La classificazionde le testo è, all'interno della branca del Natural Language Processing (NLP), un task fondamentale. Essa consiste nella maggior parte dei casi nell'assegnare delle labels a un testo, queste possono riguardare una certa categoria di appartenenza (come l'assegnazione a un topic), un sentimento (cioè, come verrà analizzato all'interno di questo report, nel valutare il grado di apprezzamento di un argomento sulla base di una valutazione numerica rispetto a una recensione) oppure spam detection.

Al fine di ottenere un buon livello di accuratezza su datasets testuali muniti di ground truth (apprendimento supervisionato), sono stati utilizzati, negli scorsi decenni, metodi tradizionali che prevedevano la rappresentazione del documento mediante features lessicali sparse, come gli *n-grammi*, utilizzate per addestrare modelli lineari o basati su kernels, come SVM [1].

Negli ultimi anni, tuttavia, i modelli basati su *deep neural networks* hanno ottenuto risultati migliori e sono lo stato dell'arte per quanto riguarda NLP. Sono state molto utilizzate in particolare le Recurrent Neural Neworks (RNN), particolari reti in grado di processare sequenze e di ricordarsi gli input precedenti, e, in certi lavori, anche le Convolutional Neural Networks [2].

Dal momento che la sfida, per queste tecniche di Machine Learning, è caratterizzata dal comprendere l'importanza delle parole, non solo per quello che vogliono dire di per sè, ma anche in relazione al contesto (una parola in un diverso contesto può avere un diverso significato e questo complica il processo di apprendimento e di classificazione), sono stati proposti dei modelli di Attenzione Gerarchica o Hierarchical Attention Networks (HAN) [3]. In particolare, gli autori hanno cercato di incorporare la conoscenza della struttura del documento nell'architettura della rete. Infatti, per quanto riguarda la classificazione, per esempio di una recensione, non tutte le frasi presenti sono rilevanti allo stesso modo e, scalando di gerarchia, non tutte le parole presenti in un periodo sono rilevanti allo stesso modo. Il processo di attenzione di HAN è concepito, come vedremo, in due livelli , uno per le parole e uno per le frasi, questo per cercare di utilizzare maggiormente il contesto per prendere la decisione nella classificazione.

Alcuni articoli seguenti, come (Adihikari et al. 2019) [4], sostengono che modelli complessi, con meccanismi di attenzione sofisticati, non portino in realtà particolari vantaggi, ma anzi peggiorino soltanto il tempo di inferenza in una eventuale fase di deployment. Un modello basato su LSTM bidirezionale, mediante alcuni accorgimenti di regolarizzazione, può ottenere gli stessi risultati se non superare una rete complessa con meccanismi di attenzione, con evidenti vantaggi di semplicità e di utilizzo di risorse.

Lo stato dell'arte per quanto riguarda i tasks di NLP è rappresentato da BERT [5], il quale utilizza diversi

layers di trasformatori (12 per la versione Base e 24 per la Large). Il modello base è molto complesso, possiede all'incirca 109 milioni di parametri: l'approccio per utilizzarlo è prendere il modello pretrained (dagli autori di Google stessi) e adattarlo alla classificazione mediante dei layer aggiuntivi e del fine-tuning, come proposto da Adihikari et al. 2019 (DocBERT) [6].

Sempre Adihikari et al., nuovamente al fine di creare un'architettura più leggera, hanno provato a riaddestrare la loro precedente versione LSTM con BERT, utilizzando quest'ultimo come insegnante per la rete studente snella con un meccanismo di Knowledge Distillation (KD) [17].

2. Contenuto del Progetto

Questo progetto ha avuto l'obiettivo di replicare i principali risultati recenti di Natural Language Processing, in particolare per quanto concerne il task della classificazione del testo. I passi salienti sono stati:

- Replica del modello di attenzione gerarchica HAN, utilizzato per attenzionare frasi e parole interessanti nel processo di classificazione.
- Fine-tuning di BERT per la text-classification, è stato utilizzato il modello base con 12 layer trasformatori.
- Modello LSTM bidirezionale con embedding dropout.
- Knowledge Distillation di LSTM con BERT.
- Per valutare l'accuratezza dei modelli sono stati utilizzati i datasets IMDB 2 classi (dispondibile da tensorflow.datasets), IMDB 10 classi e Yelp 2014.

L'intero codice del progetto è open source e reperibile su GitHub [8]. Per utilizzare il codice e replicare gli esperimenti, si consulti il relativo README.md.

3. Reti Ricorrenti

3.1. Recurrent Neural Networks

Le Recurrent Neural Networks (RNNs) sono una classe di reti neurali dove le connessioni tra i vari neuroni formano un ciclo e questa proprietà le rende particolarmente adatte a processare delle sequenze temporali di dati in input grazie alla loro intrinseca natura chain-like, come le parole [9]. Mantengono nello stato la memoria di degli istanti temporali passati e condividono i parametri, al fine di identificare parti importanti della sequenza a prescindere dall'ordine in cui arrivano (a differenza delle reti tradizionali *feedforward*.

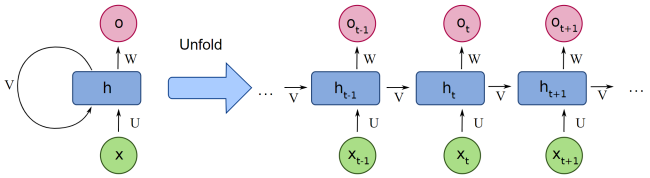

Figure 1. La struttura di una rete ricorsiva e la controparte unfolded.

Queste reti possono essere rappresentate mediante dei grafi computazioni che contengono dei cicli rispetto agli hidden states, ma è possibile visualizzare il grafo *unfolded*, mediante una disposizione estesa che rappresenta ogni singolo istante di tempo, o elemento della sequenza in input, come si nota in Figura 1. Una RNN può essere pensata come tante copie della stessa rete, ognuna che riceve in input l'elemento a un certo istante temporale della sequenza e l'output della rete precedente.

Un problema delle RNN è che, durante il calcolo della backpropagation, i gradienti tendono a scomparire o, più raramente, ad esplodere: questo fenomeno è detto *vanishing gradients* e rende difficile alla rete imparare le dipendenze tra elementi della sequenza di più lungo termine. Per evitare questo problema e ottenere un buon addestramento della rete sono stati ideati dei layers appositi come GRU e LSTM.

3.2. Gated Recurrent Unit

GRU è una rete ricorrente, derivante da LSTM, che cerca di mantenere le dipendenze a lungo termine in una sequenza in input [10]. Essa utilizza un meccanismo di *gating* per controllare il flusso informativo attraverso le unità della rete, senza utilzzare delle celle di memoria separate per tenere traccia dello stato.

Ogni cella contiene due diversi tipi di gates: uno di *reset* r_t e uno di *update* z_t. Essi sono responsabili dell'aggiornamento dell'informazione a un determinato istante di tempo.

Ad un istante di tempo t, il calcolo per il nuovo stato nascosto h_t è interpolato linearmente rispetto al precedente e al stato candidato corrente $\tilde{h}_t$:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{1}$$

In particolare il gate z_t controlla quanta informazione passata considerare e quanta nuova informazione aggiungere. Il gate di update è infatti così generato:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2}$$

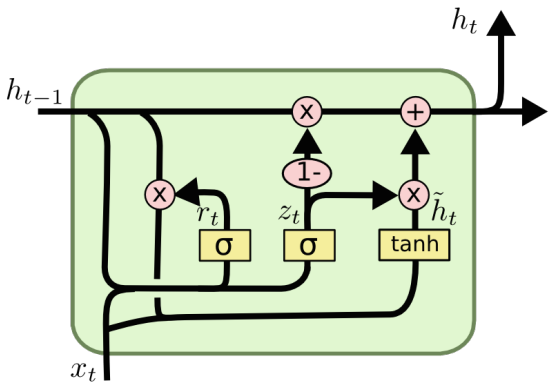

Figure 2. Modello di una unità GRU. Il nuovo stato è calcolato attraverso l'input e lo stato nascosto precedente, attraverso l'uso di due gates controllati da sigmoidi per mantenere le memoria a lungo termine.

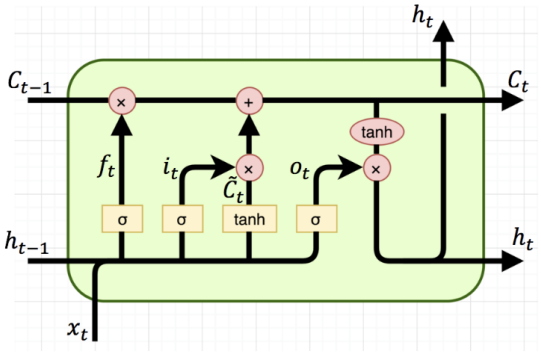

Figure 3. Unità LSTM: si noti il doppio stato nascosto che permette di separare l'informazione riguardante la memoria a breve rispetto a quella a lungo termine.

Dove $W_z, U_z e b_z$ sono pesi addestrabili e x_t l'elemento della sequenza nell'istante t. Lo stato candidato $\tilde{h}_t$ è calcolato tramite tangente iperbolica:

$$\tilde{h}_t = \tanh(W_h x_t + r_t \odot (U_h h_{t-1}) + b_h) \tag{3}$$

Il gate di reset controlla quanto devono pesare i contributi passati rispetto allo stato candidato. Se è 0, allora si dimentica lo stato passato. L'aggiornamento del reset gate è calcolato come segue:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{4}$$

Il problema dei vanishing gradients è qui mitigato grazie ai gates di controllo, poiché ad ogni istante temporale non viene ricalcolato interamente il prossimo hidden state, ma normalmente viene mantenuto lo stato precedente con una piccola modifica rispetto al nuovo input. Quindi nel momento in cui si fa backtracking non si dovrebbero avere più gradienti troppo piccoli (che quindi non riescono a cambiare i pesi da addestrare) o troppo grandi (che rendono le soluzioni instabili).

3.3. Long Short Term Memory

Le reti LSTM sono state il primo esempio di RNN in grado di risolvere il problema delle dipendenze a lungo termine, sono quindi capaci, come GRU, di ricordarsi sequenze di lunghezza maggiore rispetto alle RNN base, ma hanno una struttura leggermente più complicata [11].

Ciascuna cella del grafo virtual unrolled di una rete LSTM è formata stavolta da 3 gates, attivati da sigmoidi che restituiscono un output compreso tra 0 e 1. Inoltre il modello ha uno stato C_t (cell state) in più, oltre allo stato nascosto h_t. Lo stato C_t si occupa di mantenere in memoria le dipendenze a lungo termine, mentre h_t quelle a breve termine.

Il primo gate, chiamato *forget gate* f_t calcola la sigmoide tra i suoi pesi addestrabili e lo stato nascosto precedente con l'input:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{5}$$

Il secondo gate, chiamato *input*, si occupa di determinare se aggiornare C_t insieme al vettore dei nuovi candidati $\tilde{C}_t$. Vengono calcolati quindi:

$$\begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C[h_{t-1}, x_t] + b_C) \end{aligned} \tag{6}$$

Da cui viene ricalcolato il nuovo stato C_t pesandolo grazie ai valori restituiti dai gates appena calcolati:

$$C_t = f_t \odot C_{t-1} + i_t * \tilde{C}_t \tag{7}$$

Il nuovo stato h_t viene calcolato tramite un ulteriore gate apposito di output moltiplicato con il vettore C_t filtrato con una tangente iperbolica:

$$h_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \odot \tanh(C_t) \tag{8}$$

Il ruolo dei gates di input e forget è molto simile a quello del gate di update in GRU con la differenza che quest'ultimo riesce a controllare soltanto quanta informazione aggiungere, mentre in LSTM si controlla quanta memorizzarne e cancellarne mediante due gate separati e indipendenti [12].

Per quanto riguarda le performance, GRU possiede meno pesi da addestrare, quindi rende la fase di traning più agile; d'altra parte il modello di memoria più sofisticato di LSTM promette sulla carta risultati leggermente superiori.

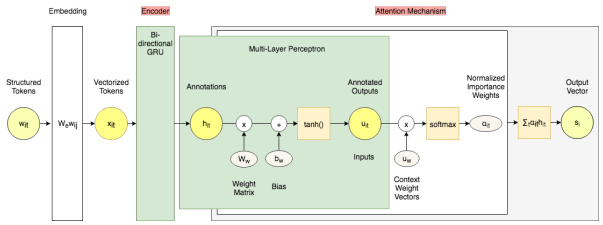

Figure 4. Il word level di HAN, costituito dall'embedding delle parole, la rete ricorrente GRU (encoder) e il layer di attenzione.

4. Hierarchical Attention Networks

Come trattato nell'articolo di Yang et al. 2016, è stato proposto un modello di classificazione del testo basato su GRU e una struttura gerarchica che attenziona le parole e le frasi più importanti all'interno del documento in input. Questo dovrebbe permettere di trovare una buona soluzione riguardo le questioni:

- Non tutte le parole in una frase o tutte le frasi nel documento sono ugualmente importanti per predire la label.
- Il significato delle parole può cambiare in base al contesto e questo può essere più o meno indicativo rispetto alla classificazione finale.

Supponiamo quindi di avere un documento con L frasi e che ciascuna frase contenga al massimo T parole, quindi w_{it} rappresenta la parola t-esima della frase i-esima del documento. A partire da questi, si costruiscono i vari vettori della struttura gerarchica, la quale implementa lo stesso procedimento di encoding-meccanismi di attenzione due volte nella pipeline, all'inizio per le parole e successivamente per le frasi.

4.1. Word Level

Il primo passo è quello di fare l'*embedding* delle parole, che vengono trasformate in vettori numerici, tramite una matrice di embedding W_e, la quale può essere, in generale, addestrata tramite un layer apposito oppure è possibile utilizzarne una pretrained. L'embeddings trasforma una parola in un vettore cercando di catturare la semantica e la relazione che intercorre tra le parole, sia questa di similarità, di opposizione. Per esempio una coppia di parole come *"Stockholm"* e *"Sweden"* ha la stessa relazione che c'è fra *"Cairo"* e *"Egipt"*.

$$x_{it} = W_e w_{it}, t \in [1, T] \tag{9}$$

Al fine di ottenere un training più veloce, all'interno del progetto è stata utilizzata una matrice di embedding pretrained, in particolare *GloVe 100d* (versione addestrata su Wikipedia 2014) [13]. Un'alternativa potebbe essere *word2vec* [14].

A questo punto si inserisce, come meccanismo di encoding, la nostra rete ricorrente GRU, in una sua versione bidirezionale, in modo da ottenere delle annotazioni sul contesto delle parole utilizzando leggendo la sequenza delle parole da entrambe le direzioni.

$$\begin{aligned} \overrightarrow{h}_{it} &= \overrightarrow{GRU}(x_{it}), t \in [1, T] \\ \overleftarrow{h}_{it} &= \overleftarrow{GRU}(x_{it}), t \in [T, 1] \\ h_{it} &= [\overrightarrow{h}_{it}, \overleftarrow{h}_{it}] \end{aligned} \tag{10}$$

Si ottengono quindi le annotazioni contestuali per una parola concatenando i due hidden states di uscita. Le annotazioni vengono date in pasto al *Word Attention Layer*.

Si calcolano adesso le parole che sono importanti per il significato della frase e si aggrega l'informazione di queste words sotto forma di *sentence vectors*. Si costruiscono dei vettori di contesto u_{it} migliorati tramite un Multilayer Perceptron con tangente iperbolica (per limitarli tra -1 e 1):

$$u_{it} = \tanh(W_w h_{it} + b_w) \tag{11}$$

Si misura l'importanza attraverso un word level sentence vector u_w (addestrabile) che va a normalizzare u_{it} attraverso una *softmax*:

$$\alpha_{it} = \frac{\exp(u_{it}^\top u_w)}{\sum_t \exp(u_{it}^\top u_w)} \tag{12}$$

Infine si calcola il vettore sentence vector s_i come una somma pesata delle annotazioni:

$$s_i = \sum_t \alpha_{it} h_{it} \tag{13}$$

4.2. Sentence Level

A partire dai sentence vectors s_i, si ripete lo stesso procedimento di encoding e attenzione per le frasi, al fine di ottenere un vettore v che racchiude l'informazione di tutte le frase del documento.

Si procede con l'encoding tramite le GRU bidirezionali:

$$\begin{aligned} \overrightarrow{h}_i &= \overrightarrow{GRU}(s_i), i \in [1, L] \\ \overleftarrow{h}_i &= \overleftarrow{GRU}(s_i), i \in [L, 1] \\ h_i &= [\overrightarrow{h}_i, \overleftarrow{h}_i] \end{aligned} \tag{14}$$

Si attenzionano le frasi più importanti:

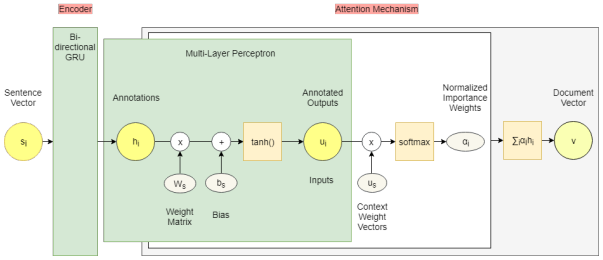

Figure 5. Il sentence level di Han che, dopo l'encoding con GRU, attenziona le frasi più importanti e le sintetizza in un vettore v utile per l'estrazione di features.

$$
\begin{aligned}
u_i &= \tanh(W_s h_i + b_s) \\
\alpha_i &= \frac{\exp(u_i^\top u_s)}{\sum_i \exp(u_i^\top u_s)} \\
v &= \sum_i \alpha_i h_i
\end{aligned}
\tag{15}
$$

4.3. Classificazione

Il vettore v in uscita dal Sentence Attention Layer può essere utilizzato come features per la classificazione del documento tramite una softmax:

$$
p = softmax(W_c v + b_c) \tag{16}
$$

5. DocBERT

Bidirectional Encoder Representations from Transformers (BERT) rappresenta lo stato dell'arte di diversi tasks in NLP, avendo superato in accuratezza i modelli precedenti. Nel loro articolo del 2019, Adhikari et al. propongono di utilizzare BERT per la classificazione. L'idea di base è utilizzare il modello di BERT pretrained aggiungendo dei layers finali ed effettuare del fine tuning della rete (cioè un riaddestramento sulla base del dataset specifico di document classification).

5.1. Attention is all you need

Questo è il titolo dell'articolo di Vaswani et al. 2017 [15] che cerca di enfatizzare il fatto che il modello sviluppato dagli autori non utilizzi RNN, ma soltanto meccanismi di attenzione i quali vanno a sostituire l'idea di ricorrenza.

Il trasformatore è l'elemento che effettua l'encoding-decoding, come le GRU in HAN. Utilizza un meccanismo di attenzione che prende in ingresso il word embedding (cioè la rappresentazione della parola tramite un vettore di numeri) e lo moltiplica per 3 diverse matrici con pesi da addestrare, *Query*, *Key* e *Value*. Esse

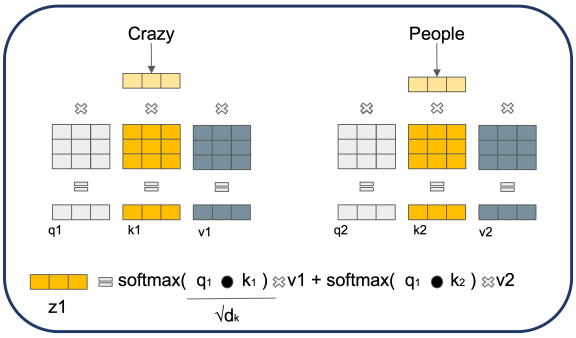

Figure 6. Struttura del meccanismo di attenzione del trasformatore: viene fatto l'embedding di ciascuna parola e si moltiplica rispetto a 4 matrici addestrabili. Si combinano i vari risultati per ottenere un vettore di attenzioni.

sono 3 diverse rappresentazioni dell'embedding iniziale della parola.

Per attenzionare le parole importanti, gli embeddings di due parole diverse vengono moltiplicati fra loro estraendo un nuovo vettore, che avrà valori maggiori nelle posizioni delle parole più importanti. Questo meccanismo è in realtà ripetuto 8 volte, per questo prende il nome di *Multi-head Attention*. Avendo 8 vettori di attention head per ogni parola, si effettua una moltiplicazione con una matrice di pesi addestrabili al fine di ottenere un vettore della dimensione giusta, più compatta, per il prossimo layer, e che cerca di contenere la maggior parte dell'informazione delle attention heads.

Per rappresentare l'ordine delle parole, che è cruciale per comprendere in generale un linguaggio, il modello utilizza degli appropriati *positional encodings* che rappresentano la posizione della parola all'interno della frase. Questi vettori vengono aggiunti agli embeddings in input e possono essere in generale addestrati o fissati. Vaswani et al. 2017 hanno proposto un'implementazione che sfrutta le funzioni seno e coseno con diverse frequenze dipendenti dalla posizione della parola e dalla dimensione.

Dopo ogni Multi-head attention layer, è aggiunto lo stesso input all'uscita, al fine di mantenere l'informazione e la posizione delle parole per i sublayers successivi ed evitare il problema dei vanishing gradients. Tale soluzione è chiamata *residual connections*.

5.2. BERT

L'archiettura di BERT si basa sui trasformatori appena descritti, in particolare ne utilizza 12 bidirezionali (ciascuno munito dei meccanismi di Multi-head Attention) nella versione BERT_{base} (109 milioni di parametri) e 24 layers nella BERT_{large} (304 milioni di parametri).

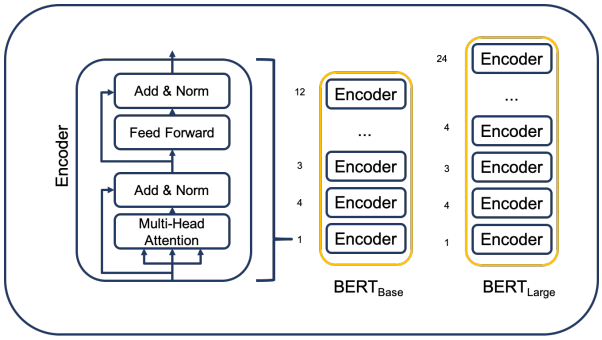

Figure 7. Architettura di BERT, si notino i trasformatori encoder, ciascuno dei quali è strutturato con i multi-head attention e una rete feed forward. Le frecce di lato rappresentano le connessioni residue.

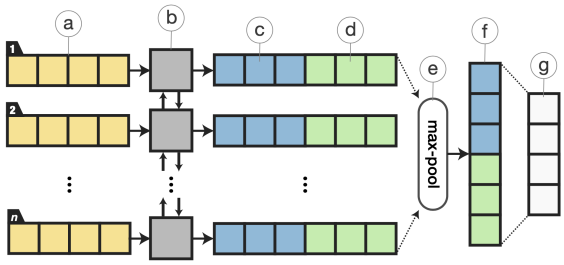

Figure 8. Il modello di LSTM: (a) input word embeddings, (b) Bilateral LSTM, (c, d) concatenazione delle hidden features backward e forward, (e) Max-Pooling, (f) document feature vector e (g) softmax.

BERT prevede in input parole specificatamente tokenizzate attraverso il WordPiece Tokenizer e aggiunge dei tokens speciali per la classificazione e la separazione delle frasi; inoltre prende in ingresso, oltre ai tokens delle parole del testo (indicizzate), una maschera e altri indici per specifici contesti di NLP.

Per utilizzare BERT nel task della document classification, si introduce un layer finale fully-connected sopra lo stato finale di BERT corrispondente al token speciale $[CLS]$ e si procede con il fine tuning. La loss è calcolata mediante *cross entropy* (dopo aver normalizzato l'output mediante una softmax finale).

6. LSTM e Knowledge Distillation

Gli autori di DocBERT hanno evidenziato la complessità di BERT in termini di quantità di parametri e tempo di inferenza, sostenendo ancora una volta la tesi che un modello più semplice può essere migliore in fase di deployment. Hanno preso perciò un loro precedente lavoro, LSTM$_{reg}$ [4] e hanno applicato una tecnica di distillazione dell'informazione da BERT durante il training.

6.1. LSTM

Al posto di una architettura complessa, basata su sofisticati meccanismi di attenzione gerarchici e molti layers, Adikhari et al. nel loro articolo *"Rethinking complex neural net- work architectures for document classification"*, cercano di dimostrare che è possibile ottenere risultati simili, se non migliori, attraverso una singola rete LSTM bilaterale, se correttamente regolarizzata. Hanno quindi una visione completamente opposta rispetto a Vaswani, non utilizzano affatto meccanismi di attenzione.

Di base il modello comprende uno strato di embedding (non è specificato se si utilizza una matrice pretrained oppure si addestra direttamente), un single layer BiLSTM che estrae i vettori di contesto in avanti e indietro e li concatena (in un modo simile alle GRU in HAN in pratica) e un layer di max-pooling con softmax finale.

Per regolarizzare la rete sono state inoltre applicate due diverse tecniche di *dropout* specifiche per le reti ricorrenti: infatti, è rischioso applicare il *dropout* direttamente sugli hidden states, poiché questo, in una RNN, potrebbe limitare la memoria a lungo termine.

- Weight-dropped: al posto di fare dropout rispetto alle unità di RNN, lo si fa sulle connessioni, secondo il risultato di una variabile aleatoria, tramite il drop-connect [16].
- Embedding-dropout: il momento più opportuno di fare dropout classico è sugli embeddings delle parole, è un metodo molto simile al rimuovere delle parole. Questo consente di regolarizzare e di rendere il modello più robusto a diversi input.

Nella implementazione LSTM di questo progetto è stato utilizzato, per semplicità, soltanto il secondo metodo di regolarizzazione.

6.2. Knowledge Distillation

A seguito dell'avvento dei trasformatori e di BERT, Adhikari et al. hanno dovuto prendere atto che effettivamente i meccanismi di attenzione sono necessari per aumentare l'accuratezza. Hanno però provato a distillare parte dell'informazione imparata da BERT in una rete più piccola e maneggevole come LSTM$_{reg}$ appena descritta, in modo da migliorarne l'accuratezza.

La tecnica in questione è chiamata *Knowledge Distillation* e si basa sull'utilizzo di una rete complessa addestrata, che ha raggiunto un alto grado di accuratezza su un determinato dataset, detta *teacher* e una piccola rete *student* da addestrare [17].

L'idea di base è quella di addestrare lo studente e di calcolare la loss relativa non solo rispetto alle labels del dataset, ma anche ai soft-targets prodotti dal suo insegnante. La rete teacher, infatti, produrrà in output una distribuzione di probabilità delle classi (per la softmax) che dovrà essere il più vicino a quella prodotta dalla rete student.

La loss totale della rete sarà quindi una composizione tra quella tradizionale (cross entropy) e quella rispetto ai soft targets dell'insegnante (è stata proposta la divergenza di Kullback-Leibler):

$$\mathcal{L} = \mathcal{L}_{classification} + \lambda \mathcal{L}_{distillation} \tag{17}$$

con $\lambda \in \mathbb{R}$ peso per i due diversi contributi.

7. Esperimenti

Sono stati condotti diversi esperimenti con i modelli appena descritti, in particolare HAN, BERT, LSTM e KD-LSTM, durante i quali si è cercato di valutare le prestazioni di ognuno ottimizzando, per quanto si sia rivelato possibile, i parametri in gioco.

L'implementazione del codice di è stata effettuata utilizzando il framework Tensorflow 2, mentre le altre con il backend Pytorch, poiché con Tf non è stato possibile ricavare la rete di BERT pretrained mediante la libreria Transformers (ottimizzata per Pytorch, nonostante che l'implementazione originale di BERT giri su Tensorflow 1).

Gli esperimenti sono stati condotti utilizzando una gpu Nvidia 1060 6GB, per questo è possibile che alcuni passaggi siano stati un po' semplificati a causa delle risorse hardware dispinibili. I fine tuning di BERT, con un po' di pazienza, è comunque andato a buon fine con risultati soddisfacenti, nonostante sia stata presa in considerazione soltanto la versione BERT_{base}.

7.1. Datasets

Per valutare i modelli sono stati usati dei datasets di classificazione del testo standard e descritti in letteratura, come IMDB e Yelp 2014. In particolare è stato possibile confrontare i risultati di questo progetto con quelli ottenuti dal Adhikari et al. e Yang et al..

IMDB è un dataset di recensioni di films, ciascuna munita di un sentimento relativo catalogato da 1 a 10. Contiene circa 135000 recensioni ed è stato reperito direttamente dalla repository GitHub del progetto di Adhikari [18]. È stata utilizzata, come prova, anche una versione ristretta (25000 documenti) con labels binarie, recensioni positive e negative, resa direttamente disponibile da tensorflow.datasets [19].

Yelp è un dataset di recensioni di ristoranti e attività commerciali con una valutazione da 1 a 5 stelle. Non è stato possibile reperire la versione 2014, ma soltanto quella con tutte le recensioni fino al 2019; quindi è stata fatta una ricostruzione in codice Python prendendo soltanto le recensioni in data 2014. Il dataset risultante è leggermente più piccolo rispetto a quello descritto da [6], circa 900000 documenti contro poco più di un milione, forse perché in Yelp 2014 originale sono state incluse delle recensioni del 2013 o 2015. I risultati ottenuti con il dataset ricostruito sono simili a quelli con l'originale.

Al fine di ottenere risultati più accurati e una verifica della stabilità del valore di accuratezza trovato, come suggerito da Adhikari et. al., sono stati presi in considerazione sia il test set che il validation set durante la costruzione della tabella finale.

7.2. Preprocessing

Il preprocessing dei datasets di HAN è stato diverso rispetto a quello di BERT, il quale ha bisogno di inputs dedicati e un suo particolare tokenizer pretrained.

In HAN si sono specificati i parametri di MAX_SENTENCE_NUM e MAX_WORD_NUM, specifici per ogni datasets, rispettivamente 20 e 40 per Yelp e 25 e 40 per IMDB. Il massimo numero di parole distinte è fissato 200 000 e la dimensione del word embedding è 100, visto che sono stati utilizzati i vettori GloVe a dimensione 100.

Dopo aver letto le parole di un documento, viene effettuato un *cleaning* che rimuove le maiuscole e le stop words più frequenti, prese da *ntlk.corpus*[20]. Viene successivamente costruito un indice che associa le parole ad un intero, il quale viene utilizzato, con i vettori di GloVe, per creare la nostra matrice degli embeddings W_e. Si associano le frasi a sequenze di numeri con l'indice e si effettua il classico split. Come suggerito dagli autori di HAN, si è optato per uno splitting 80 train, 10 validatione e 10 test. Tutte le informazioni vengono salvate per non effettuare il preprocessing prima di ogni train.

Per BERT si è effettuato subito lo split 80/10/10 e, in seguito, le parti sono state date in pasto ad una classe apposita custom. Tale classe, grazie al tokenizer già integrato di BERT, restituisce sia le sequenze di parole, sia la maschera che i token_type_ids nel formato appropriato, in particolare ritorna tensori di Pytorch. Tale metodo è consigliato dala documentazione di transformers [21]. Il numero massimo di tokens per BERT è stato limitato a 128, per questioni di memoria.

7.3. Training e Iperparametri

Per HAN, il training è stato abbastanza veloce (circa 2 ore sul dataset più grande, Yelp 2014) e sono state impiegate 45, 35 e 25 epoche rispettivamente per imdb

small, IMDB e Yelp. Come ottimizzatore è stato usato, come suggeriva Han et al., stochastic gradiente descent con momentum 0.9. In aggiunta al modello originale, è stato applicato un layer di embedding dropout con rate di 0.1, utile per regolarizzare. Al layer responsabile degli embeddings delle parole è stata data in input la matrice degli embeddings, come già discusso; è stato provato a rendere quei pesi sia addestrabili (quindi mediante una sorta di fine-tuning vengono migliorati durante l'addestramento) che non, ma non si sono riscontrate differenze apprezzabili sia in termini di velocità di training sia di accuracy del modello sul testo.

Il fine-tuning di BERT ha richiesto rispettivamente 2, 3 e 3 epoche per imdb small, IMDB e Yelp 2014. Quest'ultimo è stato in fase di training per circa 13 ore e mezzo sulla gpu utilizzata. Avendo utilizzato come parametro numero massimo di parole 128 (MAX_LEN), i valori consigliati per il batch size dagli stessi autori di BERT sono 16 e 32, ma per risorse disponibili, è stato scelto 16 per tutti e 3 i datasets. L'ottimizzatore scelto è stato Adam, con learning rate 10^{-5}.

Per LSTM sono stati addestrati anche i pesi dell'embedding, poiché non è stata utilizzata alcuna matrice di embedding pretrained, ma sono state date in input soltanto le parole indicizzate dal tokenizer di BERT. Il motivo della scelta riguarda la facilità di integrazione del modello con BERT, utile soprattutto nell'ultimo modello proposto in questo progetto, cioè KD-LSTM. Si è scelta una dimensione di embedding pari a 50 e 256 unità nascoste per la rete ricorrente (come nel codice di Adhikari et al.); il numero massimo di parole è 128, come in BERT. L'ottimizzatore utilizzato è stato Adam, learning rate 10^{-3}. Il training ha richiesto un totale di 9 epoche per 3 ore e 25 minuti di addestramento con il dataset Yelp 2014.

Nella Knowledge Distillation di LSTM sono stati mantenuti gli stessi parametri utilizzati nella versione base, è stata soltanto modificata, come da manuale, la loss, aggiungendo la KL divergence tra i soft targets di BERT e l'uscita della softmax di LSTM. L'iperparametro λ, che pesa il contributo della distillazione, ha assunto valore 1 in tutti e tre gli esperimenti con i diversi datasets, come suggerito da Adhikari.

8. Analisi dei Risultati

Osservando la tabella dei risultati, la quale esprime i valori di accuratezza raggiunti dai vari modelli implementati, in confronto rispetto a quelli dei papers originali, è lecito concludere che i valori trovati sono generalmente vicini, con qualche lieve discrepanza.

HAN è la rete meno recente, nonostante unisca le reti ricorrenti con l'idea di avere dei meccanismi di attenzione, e infatti raggiunge, come previsto risultati inferiori rispetto ai modelli successivi. In particolare raggiunge risultati di poco inferiori rispetto a LSTM con Yelp 2014, ma è peggiore significativamente negli altri due datasets. Con il dataset IMDB in questo progetto sono stati ottenuti risultati inferiori rispetto a Yang et al. 2016; cercando di ragionare sulle differenze, la matrice degli embeddings utilizzata nel loro paper è stata word2vec (invece di GloVe) con dimensione di word embedding 200 (invece di 100). D'altra parte, per normalizzare, è stato implementato dell'embedding dropout in questo paper, che, controllando il training, contribuisce significativamente a migliorare l'accuracy (su IMDB ha permesso di guadagnare un buon 3%).

BERT_{base} è il modello più pesante, ma con prestazioni decisamente migliori. Si ottengono i valori di accuracy più alti su tutti e 3 i datasets, con valori molto stabili tra test set e validation set. Il prezzo da pagare è, come sarà discusso più in dettaglio, il costo di training e il tempo di inferenza. Nell'articolo di Adhikari et al. 2019 è inserito anche BERT_{large}, che ottiene accuracy superiore del 2% circa, ed è stata fatta un'analisi più approfondita rispetto al numero massimo di parole ammesse dalla rete. Secondo il loro articolo, nei datasets utilizzati, all'aumentare del numero massimo di parole, aumenta anche l'accuracy, come è lecito pensare. Dal momento che in questa implementazione, per ragioni esclusivamente di risorse hardware disponibili, viene utilizzato come MAX_LEN 128, sarebbe interessante in futuro, vedere se effettivamente l'accuracy cresce ancora aumentando tale valore, siccome i risultati di questo paper superano già quelli di Adhikari et al.

LSTM raggiunge onestamente dei buoni risultati, con una buona stabilità, su tutti e 3 i datasets, senza mai comunque raggiungere BERT. Rispetto alle implementazioni precedenti, i risultati sono comparabili.

KD-LSTM porta benefici su tutti i datasets considerati per quanto concerne l'accuracy finale. IMDB Small migliora di poco, complice anche il fatto che BERT ha risultati simili a LSTM in questa configurazione. Su Yelp si ha un miglioramento piccolo, ma percettibile dello 0.7%, risultato confrontabile anche rispetto ai risultati di Adhikari. Nel dataset IMDB si ha un miglioramento sostanziale, con risultati simili, se non superiori, a BERT: la nota negativa riguarda però la distribuzione delle predizioni effettuate dalla rete distillata. IMDB, come dataset, è abbastanza sbilanciato al negativo, ci sono più recensioni negative rispetto alle positive (circa 3 volte più recensioni con label 0 rispetto alla label 9) e KD-LSTM predice quasi esclusivamente le labels più negative, ottenendo comunque risultati finali molto buoni. BERT, invece, si impegna

#	Model	IMDB Small		IMDB		Yelp 2014	
		Val. Acc.	Test Acc.	Val. Acc.	Test Acc.	Val. Acc.	Test Acc.
1	HAN Orig.	-	-	-	49.4	-	70.5
2	HAN	89.0	86.6	45.9	46.4	68.6	69.0
3	BERT_{base} Orig.	-	-	54.4	54.2	72.1	72.0
4	BERT_{base}	94.6	**94.6**	57.5	**57.7**	77.3	**77.4**
5	LSTM_{reg} Orig.	-	-	53.4	52.8	69.0	68.7
6	LSTM_{reg}	94.3	94.2	52.8	52.7	71.1	71.1
7	KD-LSTM_{reg} Orig.	-	-	54.5	53.7	69.7	69.4
8	KD-LSTM_{reg}	94.4	94.6	58.0	58.5	71.6	71.7

Table 1. Risultati, per ogni modello, sul validation e test sets. I modelli con la dicitura 'Orig.' sono stati presi da Yang et al. 2016 (HAN) e Adhikari et al. 2019 (tutti gli altri).

a predire tutte le label, anche sbagliando leggermente di più, ma avendo un comportamento più *"naturale"*.

8.1. Visualizzazione dell'attenzione in HAN

Il modello di HAN, con i suoi layer di attenzione gerarchici suddivisi in frasi e parole, consente di avere un'idea riguardo al processo di learning della rete. In particolare è possibile capire se la rete è riuscita a dare la giusta importanza alle parole in base al contesto in cui sono inserite.

Yang et al. 2016 hanno analizzato ad esempio la parola *'good'* e hanno notato che essa non compariva soltanto nelle recensioni positive, ma anche in quelle negative, con, naturalmente, minore frequenza. Questo proprio perché anche una parola positiva può avere un diverso significato in base al contesto.

L'idea di questa ulteriore verifica è quella di visionare le parole e le frasi più importanti per HAN, estraendo i pesi dei layers intermedi relativi ai meccanismi di attenzione, della rete già addestrata. In pratica si va ad utilizzare parti della rete (quella relative all'attention words e quella per l'attenzione sulle frasi) con l'obiettivo di estrarre risultati parziali.

Data una recensione in ingresso, viene fatta una previsione con ciascuna rete parziale e vengono così restituite sua la frasi che le parole attenzionate. Viene inoltre predetta la label tramite il modello di rete completo.

Con i vettori che identificano le parole e le frasi più importanti, è stata creata una semplice interfaccia grafica da terminale, la quale cerca di esemplificare con i colori il diverso livello di attenzione nelle gerarchie del documento. In rosso scuro sono evidenziate le parole più importanti, seguite da quelle minormente attenzionate in rosso chiaro. Lo stesso principio è applicato alle frasi: quelle più importanti sono associate ai livelli di blu più intensi, quelle meno ai blu chiari.

Dalle recensioni presentate nelle Figure 8, 9 e 10, tutte provenienti dal dataset Yelp, si può notare come la rete riconosca abbastanza bene le parole più significative, sia positive che negative. Non solo: in base al contesto, una parola può essere interpretata come negativa o positiva, cosa che presumibilmente accade anche in questi casi con *'bad'*, *'recommend'* e *'fond'*.

Le recensioni con etichetta media rappresentano, come è lecito aspettarsi, le predizioni più difficili, spesso anche un umano potrebbe sbagliarsi nel classificare questi testi. Tuttavia la rete, ma volendo generalizzare anche tutti i modelli proposti, quando sbagliano sono comunque piuttosto vicini alla label giusta (cercando di interpretare i risultati soltanto qualitativamente, sarebbe interessante fare dei test quantitativi ad hoc).

Dataset	HAN	LSTM	BERT
IMDB Small	3 (1x)	13 (4x)	34 (11x)
IMDB	10 (1x)	102 (10x)	216 (22x)
Yelp 2014	30 (1x)	121 (4x)	710 (24x)

Table 2. Tempi di inferenza sul test sets per ogni modello, risultati in secondi. È stata usata come baseline la rete con inferenza più veloce, HAN.

8.2. Analisi della Complessità dei Modelli

Facendo riferimento alla Table 2, possiamo valutare, come fatto da Adhikari, la complessità dei modelli, valutandola soprattutto in una prospettiva di deployment di una rete di classificazione del testo, con risorse fisiche limitate.

L'accuratezza è a favore di BERT, quindi se i requisiti di sistema prevedono di ottenere precisione elevata senza badare a spese hardware, allora BERT è il modello da utilizzare oggi, in particolare si potrebbe pensare di scegliere addirittura la versione *large*.

Viceversa, è necessario almeno porsi l'interrogativo di usare una rete più piccola, come LSTM o HAN. Quest'ultima ha un tempo di inferenza ridotto di un

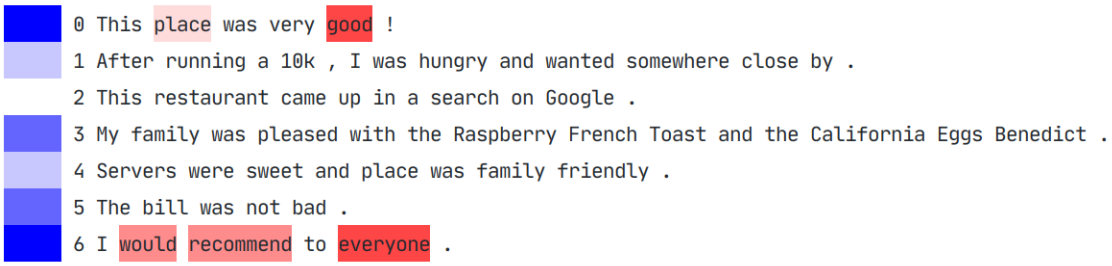

Figure 9. **PREDIZIONE di HAN: 5, TARGET: 5.** La prima e l'ultima frase sono state attenzionate maggiormente, con le relative parole più importanti. Nella frase *'The bill was not bad'*, anch'essa avente abbastanza peso nel processo decisionale, la parola *'bad'* è stata probabilmente ben interpretata nel contesto.

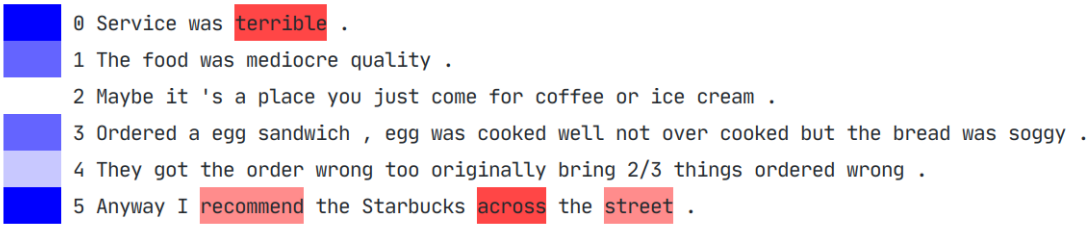

Figure 10. **PREDIZIONE di HAN: 1, TARGET: 1.** In questa recensione negativa, è stata giustamente enfatizzata la parola *'terrible'*, ma avuto molto peso anche l'ultima frase, in questo contesto *'recommend'* ha una valenza negativa, a differenza della recensione precedente.

fattore 20 su datasets reali, ma, soprattutto su IMDB, ottiene un'accuratezza parecchio inferiore.

LSTM potrebbe essere un giusto compromesso: ha risultati più che soddisfacenti e un tempo di inferenza fino a 7 volte inferiore rispetto a BERT. Inoltre, se si possiedono sufficienti risorse hardware in fase di addestramento, potrebbe essere una buona idea provare a fare Knowledge Distillation con BERT, se si vuole ottenere la massima precisione su un ambiente di deployment tutto sommato limitato.

9. Conclusione e Sviluppi Futuri

In questo progetto è stato affrontato il problema della classificazione di documenti testuali, prendendo come riferimento dei datasets ben conosciuti di Sentiment Analysis. Dopo una breve descrizione dei metodi utilizzati della letteratura, sono stati presi in considerazione i modelli più recenti di *Deep Learning* e valutate due diverse filosofie: le reti ricorrenti e i meccanismi di attenzione.

Sono stati reimplementati i modelli HAN, è stato acquisito BERT al fine di effettuare del fine-tuning sui datasets in possesso e è stato fatto un excursus riguardante LSTM, con meccanismi di regolarizzazione, e la Knowledge Distillation, come metodi più leggeri alternativi a BERT.

I risultati sembrano suggerire come vincitore, se il metro di giudizio si ferma alla valutazione dell'accuracy sul test set, BERT, con i suoi meccanismi di attenzione (all'interno degli encoders trasformatori) distaccati dall'idea di ricorrenza. Il modello ibrido di HAN, risalente al 2016, ottiene risultati ben peggiori, nonostante cerchi di unire i due diversi concetti. Una rete più minimale, come la versione proposta di LSTM, riesce ad avere risultati quasi paragonabili a BERT, avendo tempi di training e inferenza molto inferiori, quindi costi finali minori in una prospettiva di deployment.

Quindi, le reti ricorrenti sono destinate a scomparire? È complicato, nonostante i trasformatori sembrino prendersi la scena, una semplice rete LSTM ottiene comunque degli ottimi risultati. Forse la strada intrapresa dagli autori di HAN, cioè utilizzare reti ricorrenti combinate con metodi di attenzione, potrebbe rivelarsi più interessante del previsto e ottenere in futuro risultati migliori, magari andando a complicare la struttura del modello.

References

[1] Thorsten Joachims. 1998. *Text categorization with support vector machines: Learning with many rele-*

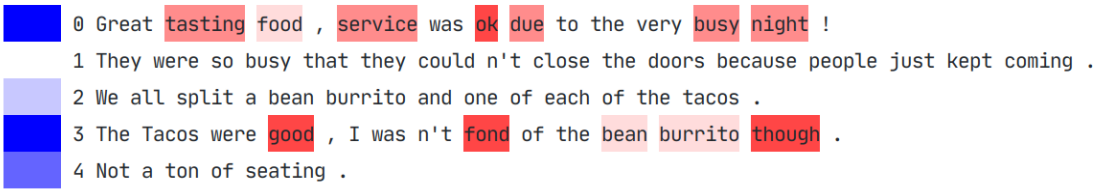

Figure 11. **PREDIZIONE di HAN: 3, TARGET: 4.** In questo caso, il modello ha di poco sbagliato la predizione, ma, nonostante ci siano molte parole positive, solo dal contesto si può dedurre che la recensione non è così entusiasta, come per la parola *'fond'*.

vant features. Springer.

[2] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017a. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124.

[3] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchi- cal attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.

[4] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. Rethinking complex neural net- work architectures for document classification. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4046–4051.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Tech- nologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

[6] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. 2019. *DocBERT: BERT for Document Classification*. Arvix.

[7] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. *Distilling the knowledge in a neural network.* arxiv/1503.02531.

[8] Repostory Github del progetto, `https://github.com/FrancescoGradi/DocumentClassificationwithHANandBERT`

[9] Ian Goodfellow and Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Ben- gio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv*:1409.0473.

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

[12] Un articolo su LSTM: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

[13] Vettori di embedding pretrained *GloVe*: `https://nlp.stanford.edu/projects/glove/`

[14] Vettori di embedding pretrained *word2vec*: `https://code.google.com/archive/p/word2vec/`

[15] Vaswani, A. u. a. (2017) „Attention is all you need“, in *Advances in Neural Information Processing Systems*. Neural information processing systems foundation.

[16] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural net- works using dropconnect. In *International Confer- ence on Machine Learning*, pages 1058–1066.

[17] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. *arxiv/1503.02531.*

[18] Repository Github Hedwig: `https://github.com/castorini/hedwig`

[19] Tensorflow datasets:
https://www.tensorflow.org/datasets

[20] NTLK, tool per NLP: https://www.nltk.org

[21] Libreria per BERT pretrained:
https://github.com/huggingface/transformers