Traduzioni Inglese-Italiano Mediante un Modello Sequence to Sequence con Attention

Andrea Valenti
a.valenti5@studenti.unipi.it

August 26, 2018

Corso:	HLT
Anno Accademico	2017-18

1 Introduzione

La neural machine translation (NMT) è un approccio alla machine translation in cui una rete neurale (tipicamente ricorrente o convoluzionale) è addestrata end-to-end a tradurre frasi da un linguaggio sorgente ad un linguaggio destinazione. Negli ultimi anni, i modelli di NMT hanno dimostrato la loro affidabilità e accuratezza nelle traduzioni, piazzandosi consistentemente tra i vincitori delle più importanti competizioni internazionali [1]. Al momento della scrittura di questa relazione rappresentano infatti lo stato dell'arte della traduzione automatica, il tutto con un consumo di risorse (in particolar modo in termini di memoria) decisamente più contenuto rispetto alle loro controparti "tradizionali" [1][2].
In questa relazione è presentato un modello per la NMT, ispirato sull'architettura sequence to sequence descritta in [3]. Dopo una descrizione del modello in sezione 2, nella sezione 3 sono presentati gli esperimenti effettuati e i relativi risultati ottenuti. Infine, in sezione 4 questi risultati sono analizzati e discussi, e vengono fornite delle linee guida per un ulteriore aumento delle prestazioni del modello.

2 Modello

Questa sezione descrive l'implementazione del modello sequence to sequence. L'architettura di quest'ultimo è stata in gran parte ispirata da [2], [3] e [5]. Sostanzialmente, questo modello è composto da due moduli principali: un encoder ed un decoder. L'encoder, tipicamente implementato mediante una recurrent neural network, ha il compito di codificare una frase di lunghezza arbitraria in un vettore di dimensione fissa (impostata a priori) che rappresenta il significato della frase. Questo vettore è denotato con il nome di thought vector. Il though vector è successivamente dato in input al decoder (un'altra recurrent neural network), che ha il compito di generare la traduzione nel linguaggio di destinazione. Dopo una descrizione delle tecniche di preprocessing utilizzate, sono descritti in modo dettagliato i diversi componenti del modello.

2.1 Preprocessing

Il preprocessing si divide in varie fasi successive. Inizialmente avviene la pulizia e normalizzazione del testo grezzo: sono eliminati dal testo tutti i caratteri che non siano lettere e i più comuni segni di punteggiatura `.,:;!?`. L'alfabeto di riferimento è l'alfabeto latino con l'aggiunta delle lettere accentate `àèéìòù`, necessarie per una corretta rappresentazione della lingua italiana. Tutte le lettere sono convertite in minuscolo. L'output di questa fase è una sequenza di parole minuscole e/o segni di interpunzione, separati da un singolo spazio (ad esempio, la frase `"Corri, è tardi!"` viene trasformata in `"corri , è tardi !"`). Infine, ad ogni frase sono aggiunti i simboli speciali di inizio e fine frase (rispettivamente `<start>` e `<end>`).
Successivamente, si procede con la creazione di un indice, assegnando ad ogni parola presente nel dataset un identificativo intero univoco, che sarà poi usato per codificare le parole date in input all'encoder ed al decoder. Allo stesso tempo viene creato l'indice inverso, da numeri interi a parole. Il valore 0 non è assegnato a nessuna parola, bensì è riservato per il carattere speciale di padding `<pad>`. Questo carattere è utilizzato nell'ultima fase di preprocessing, dove la lunghezza delle frasi del dataset viene uniformata a quella della frase più lunga. Alla fine della fase di preprocessing ogni frase avrà quindi una lunghezza fissa T.

2.2 Encoder

L'encoder è formato da due componenti principali: un layer di embedding ed un layer di encoding. Il layer di embedding ha il compito di trasformare l'indice ricevuto in input in una rappresentazione neurale densa

$$e_t = Ew_t \tag{1}$$

$E \in \mathbb{R}^{m \times k}$ rappresenta la matrice dei word embedding. m e k sono, rispettivamente, la dimensionalità del word embedding e la dimensione del vocabolario. e_t rappresenta il word embedding dell'input al t-esimo timestep w_t. Il risultato dell'embedding viene passato a un layer di Gated Recurrent Unit bidirezionale (biGRU), che si occupa di effettuare l'encoding vero e proprio. I forward state dell'encoder sono calcolati nel seguente modo:

$$\overrightarrow{h}_t = \begin{cases} (1 - \overrightarrow{z}_t) \otimes \overrightarrow{h}_{t-1} + \overrightarrow{z}_t \otimes f_t & t > 0 \\ 0 & t = 0 \end{cases} \tag{2}$$

dove

$$\begin{aligned} f_t &= tanh(\overrightarrow{W} e_t + \overrightarrow{U}(\overrightarrow{r}_t \otimes \overrightarrow{h}_{t-1})) \\ \overrightarrow{z}_t &= \sigma(\overrightarrow{W_z} e_t + \overrightarrow{U_z} \overrightarrow{h}_{t-1}) \\ \overrightarrow{r}_t &= \sigma(\overrightarrow{W_r} e_t + \overrightarrow{U_r} \overrightarrow{h}_{t-1}) \end{aligned} \tag{3}$$

$\overrightarrow{z}_t$ e $\overrightarrow{r}_t$ rappresentano, rispettivamente, l'update ed il reset gate della biGRU. $\otimes$ rappresenta la moltiplicazione elemento per elemento di due vettori. $\overrightarrow{W}, \overrightarrow{W_z}, \overrightarrow{W_r} \in \mathbb{R}^{n \times m}$ rappresentano le matrici dei pesi di input della GRU; $\overrightarrow{U}, \overrightarrow{U_z}, \overrightarrow{U_r} \in \mathbb{R}^{n \times n}$ rappresentano le matrici dei pesi ricorrenti della biGRU. n è il numero di unità nascoste della biGRU. In questo caso, $n = 512$ e $m = 256$. σ rappresenta la funzione sigmoidale. Tutte le variabili sopra citate si riferiscono alla parte forward della biGRU. I backward state $\overleftarrow{h}_t$ sono calcolati in modo del tutto analogo ai forward state.

Le unità GRU offrono vantaggi simili ad altre gated unit, come le LSTM, per quanto riguarda l'apprendimento di dipendenze a lungo termine; però, a differenza di quest'ultime, sono caratterizzate da una maggiore efficienza computazionale. Inoltre, nei linguaggi naturali è pittosto comune trovare parole il cui significato dipende non solo dalle parole precedenti, ma anche (o esclusivamente) da quelle successive, e l'utilizzo di un layer bidirezionale permette di catturare proprio questo tipo di dipendenze all'interno della frase,

migliorando quindi la qualità complessiva della traduzione [4].
L'encoder restituisce quindi la sequenza dei thought vector

$$h_t = \begin{bmatrix} \overrightarrow{h}_t \\ \overleftarrow{h}_{T-t+1} \end{bmatrix} \tag{4}$$

uno per ogni timestep.

2.3 Decoder

Il decoder, in modo analogo all'encoder è formato da un layer di embedding e da un layer ricorrente di decoding. Il layer di embedding è del tutto analogo a quello dell'encoder. Il layer ricorrente è una GRU monodirezionale, a cui è stato aggiunto il meccanismo di attention [5]. Gli stati del decoder sono calcolati nel seguente modo

$$s_t = \begin{cases} (1 - z_t) \otimes s_{t-1} + z_t \otimes g_t & t > 0 \\ \begin{bmatrix} \overrightarrow{h}_T \\ \overleftarrow{h}_1 \end{bmatrix} & t = 0 \end{cases} \tag{5}$$

dove

$$\begin{aligned} g_t =& tanh(We_t + U(r_t \otimes s_{t-1}) + Cc_t) \\ z_t =& \sigma(W_z e_t + U_z s_{t-1} + C_z c_t) \\ r_t =& \sigma(W_r e_t + U_r s_{t-1} + C r_t) \end{aligned} \tag{6}$$

z_t e r_t rappresentano, rispettivamente, l'update ed il reset gate della GRU. $\otimes$ rappresenta la moltiplicazione elemento per elemento di due vettori. $W, W_z, W_r \in \mathbb{R}^{2n \times m}$ rappresentano le matrici dei pesi di input della GRU; $U, U_z, U_r \in \mathbb{R}^{2n \times 2n}$ rappresentano le matrici dei pesi ricorrenti delle GRU; $C, C_z, C_r \in \mathbb{R}^{2n \times 2n}$ rappresentano le matrici dei pesi del context vector. Da notare che, in questo caso, il numero delle unità nascoste della GRU è $2n$, mentre m, la dimensionalità del word embedding, rimane invariata.
Il context vector è ricalcolato ad ogni timestep dal meccanismo di attention

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j \tag{7}$$

dove

$$\begin{aligned} \alpha_{tj} =& \frac{exp(a_{tj})}{\sum_{k=1}^{T} exp(a_{tk})} \\ a_{tj} =& v_a^\top tanh(W_a s_{t-1} + U_a h_j) \end{aligned} \tag{8}$$

h_j rappresenta il thought vector generato dall'encoder al timestep j (cfr. equazione 4), $v_a \in \mathbb{R}^T$, $W_a \in \mathbb{R}^{T \times 2n}$, $U_a \in \mathbb{R}^{T \times 2n}$.
Se impostiamo $\forall t : c_t = h_T$ otteniamo l'architettura sequence to sequence "classica", senza attention. Questo presuppone che h_T sia in grado di riassumere tutte le informazioni rilevanti per rappresentare perfettamente il significato della frase da tradurre, condizione che non sempre si verifica, in particolare quando la frase contiene dipende a lungo termine particolarmente difficili da catturare per l'encoder. Il meccanismo di attention cerca di superare questa limitazione, facendo in modo che il decoder abbia accesso all'intera sequenza di thought vector generata dall'encoder, e lasciandogli "valutare" ad ogni timestep quali thought vector siano i più rilevanti per la traduzione della parola corrente [5].

3 Esperimenti

In questa sezione sono descritti il training processo di training e di valutazione utilizzato per il modello sopra descritto. Sono inoltre riportati alcuni esempi di traduzioni ottenute con il modello finale.

3.1 Training

Il training del modello è stato eseguito mediante il metodo di ottimizzazione Adam [6]. I parametri utilizzati sono $\alpha = 0.001$ (learning rate), $\beta_1 = 0.9$, $\beta_2 = 0.999$ come consigliato nell'articolo originale. Per la valutazione dell'errore è stata utilizzata la cross-entropy (dove i token speciali di padding `<pad>` non sono stati inclusi nel calcolo dell'errore).
Tutti i dataset sono stati suddivisi in batch da 64 esempi ciascuna. Il 20% dei dati sono stati messi in disparte in un validation set. Il training è stato eseguito per 10 epoche, alla fine di ciascuna epoca il modello è stato valutato sul validation set per monitorare l'eventuale overfitting. Durante il training del decoder è stata applicata la tecnica del teacher forcing: ad ogni timestep,

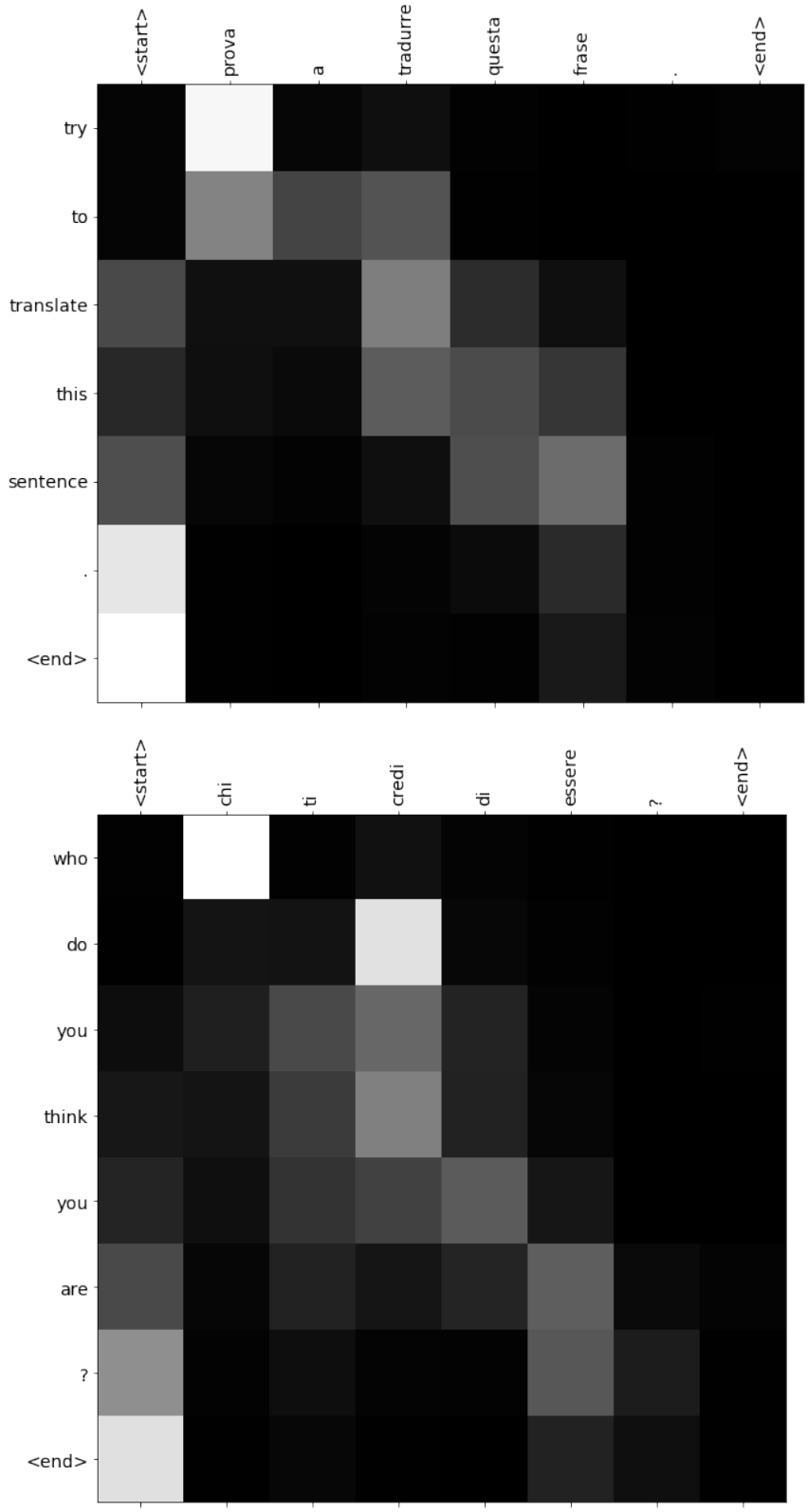

Figure 1: Visualizzazione degli attention weights generati dal modello durante la traduzione di alcune frasi.

invece di dare in input al decoder l'output dello stesso al timestep precedente, viene dato direttamente il target, come se il decoder avesse sempre eseguito la traduzione corretta. Ciò, in genere, può portare ad una più rapida convergenza del processo di training.
Gli esperimenti sono basati sul dataset *Italian-English Tab-delimited bilingual sentece pairs*, raccolto nell'ambito del Tatoeba project[7], che consiste in una raccolta di traduzioni inglese-italiano di frasi generiche, ordine in ordine crescente di lunghezza e complessità, partendo dalle singole parole sino ad arrivare a semplici frasi complete di senso compiuto.

3.2 Risultati

Di seguito sono riportati esempi di traduzioni eseguite dal modello qui presentato (la spaziatura delle traduzioni generate è stata modificata per garantire una maggiore facilità di lettura). Per confronto, è fornita anche la stessa traduzione ottenuta con Google Translate. In figura 1 sono visualizzati gli attention weights generati dal decoder durante la traduzione di alcune di queste frasi.

Dataset *Italian-English Tab-delimited bilingual sentece pairs*:

- – *Input:* "Prova a tradurre questa frase."
  – *Output:* "Try to translate this sentence."
  – *Google Translate:* "Try to translate this sentence."
- – *Input:* "Non ne posso più!"
  – *Output:* "I can't make it anymore."
  – *Google Translate:* "I can not stand it anymore!"
- – *Input:* "Chi ti credi di essere?"
  – *Output:* "Who do you think you are?"
  – *Google Translate:* "Who do you think you are?"
- – *Input:* "La mela rossa è caduta dall'albero."
  – *Output:* "The apple fell off the tree."
  – *Google Translate:* "The red apple has fallen from the tree"

- – *Input:* "Sbrigati, il tempo stringe."
  – *Output:* "Hurry up, the weather will be short."
  – *Google Translate:* "Hurry up, time is running out."
- – *Input:* "Le frasi lunghe sono le più difficili da tradurre."
  – *Output:* " The art is twenty more difficult."
  – *Google Translate:* "The long sentences are the most difficult to translate."

4 Discussione e sviluppi futuri.

In questa relazione è stato descritto un modello sequence to sequence applicato ad un task di neural machine translation. Su frasi semplici e di lunghezza contenuta la traduzione ottenuta da questo modello è comparabile a quella di Google Translate, mentre la qualità degrada rapidamente all'aumentare della lunghezza della frase. Questo problema, seppur mitigato dal meccanismo di attention, rimane uno dei problemi tipici delle architteture sequence to sequence [2]. In ogni caso i risultati ottenuti, sebbene non perfetti, possono essere considerati più che soddifacenti, considerando in particolar modo la relativa semplicità del modello. Per migliorare le prestazioni si possono aumentare il numero di layer, la dimensione dei singoli layer, o il numero di epoche di training, anche se questo potrebbe probabilmente portare ad una situazione di overfitting. Pertanto, in aggiunta a fare ciò sarebbe opportuno considerare l'utilizzo di tecniche di regularization, come il dropout [8].

References

[1] Ajay Anand Verma, Pushpak Bhattacharyya
Literature Survey: Neural Machine Translation.
http://www.cfilt.iitb.ac.in/resources/surveys/NMT-survey-paper.pdf

[2] Ilya Sutskever, Oriol Vinyals, Quoc V. Le
Sequence to Sequence Learning with Neural Networks.
https://arxiv.org/pdf/1409.3215.pdf

[3] Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio
Neural Machine Translation by Jointly Learning to Align and Translate.
https://arxiv.org/pdf/1409.0473v7.pdf

[4] Mike Schuster, Kuldip K. Paliwal
Bidirectional Recurrent Neural Networks.
https://maxwell.ict.griffith.edu.au/spl/publications/papers/ieeesp97_schuster.pdf

[5] Minh-Thang Luong, Hieu Pham, Christopher D. Manning
Effective Approaches to Attention-based Neural Machine Translation.
https://arxiv.org/pdf/1508.04025.pdf

[6] Diederik P. Kingma, Jimmy Lei Ba
Adam: A Method for Stochastic Optimization.
https://arxiv.org/pdf/1412.6980v8.pdf

[7] Tatoeba Project
Tab-delimited Bilingual Sentence Pairs.
http://www.manythings.org/anki/

[8] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov
Improving neural networks by preventing co-adaptation of feature detectors.
https://arxiv.org/pdf/1207.0580.pdf