Who's next?
La predizione nell'era dell'attenzione

Dario Comanducci

***Sommario*—Il meccanismo dell'attenzione combinato con l'architettura dei Transformers ha modificato radicalmente l'approccio al Natural Language Processing, così come in generale gli aspetti di predizione su dati di natura sequenziale: questo report riassume gli aspetti principali di tale paradigma di analisi.**

I. Introduzione

L'apprendimento automatico della rilevanza degli elementi in input ad un sistema di Intelligenza Artficiale permette di valutare l'importanza di qualsiasi porzione dell'input e tenerne conto durante l'apprendimento ed esecuzione di tale sistema: la soluzione più comune a questo problema è un meccanismo noto come *attenzione* [2].

Storicamente le prime idee sulla rilevanza automatica nei dati di input sono nate nell'ambito della Computer Vision, volte in particolare ad identificare le parti "importanti" all'interno di un'immagine o video su cui focalizzare maggiormente l'elaborazione ad alta risoluzione [4], [5], mentre per quanto riguarda il settore del Natural Language Processing il concetto di attenzione è stato adottato per la prima volta in [1], superando le limitazioni dei precedenti approcci (RNN e LSTM) basati su modelli *Encoder-Decoder*, per la traduzione tra lingue [2].

A. Modelli Encoder-Decoder

I modelli encoder-decoder sono utilizzati soprattutto per attività in cui la sequenza di input e quella di output possono avere lunghezze diverse e la mappatura tra un token nell'input e uno nell'output può essere molto "distante": il caso applicativo più tipico è la traduzione automatica.[1]

Come illustrato in Fig. 1, alla base di queste modello c'è una rete neurale di codifica (l'*encoder*) che prende una sequenza di input $\{\mathbf{x}_1 \dots \mathbf{x}_n\}$ e ne crea una rappresentazione contestualizzata (il *contesto*, per l'appunto) che viene quindi passata a un decodificatore (il *decoder*) che genera una sequenza di output $\{\mathbf{y}_1 \dots \mathbf{y}_m\}$ specifica per il problema in questione: tipici esempi di reti per l'encoder ed il decoder possono essere le RNN, LSTM, reti convoluzionali e Transformers [3, § 8.7].

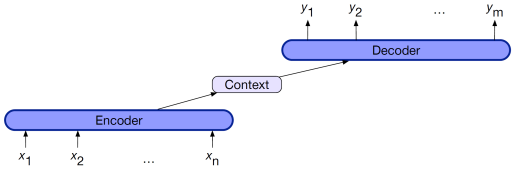


Figura 1. Generico modello Encoder-Decoder.

[1]Altri scenari possono essere quello di riassumere un testo oppure quello di fornire una risposta ad una domanda.

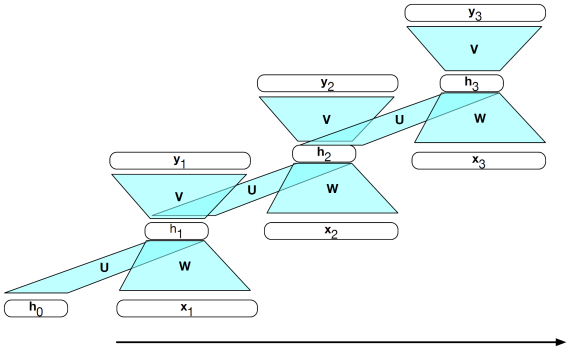


Figura 2. RNN semplice sviluppata nel tempo.

Tralasciando le reti LSTM (delle particolari RNN) e le reti convoluzionali, verranno invece approfonditi propedeuticamente prima i modelli Encoder-Decoder basati su RNN per poi passare a quelli con i Transformers.

B. Recurrent Neural Networks (RNN)

Una rete neurale ricorrente (RNN) è una qualsiasi rete che contiene un ciclo nelle sue connessioni, ossia il valore di una certa unità dipende direttamente o indirettamente dai suoi output precedenti come input: in particolare Fig. 2 mostra l'esempio di una RNN *semplice* (o *rete di Elman*) in cui lo stato interno al tempo t è dato da una combinazione degli input correnti e dallo stato interno al tempo precedente (il contesto) pesato secondo un'opportuna matrice (quadrata) $\mathtt{U}$ [3, § 8.1]

$$\mathbf{h}_t = g(\mathtt{W}\mathbf{x}_t + \mathtt{U}\mathbf{h}_{t-1})$$
$$\mathbf{y}_t = f(\mathtt{V}\mathbf{h}_t)$$

In particolare consideriamo il caso in cui $g(\cdot)$ è una funzione di attivazione come $\tanh(\cdot)$ o $\mathrm{ReLU}(\cdot)$, mentre $f(\cdot) = \mathrm{softmax}(\cdot)$ (per cui l'uscita modella una distribuzione di probabilità sugli elementi di $\mathbf{y}_t$ condizionata allo stato $\mathbf{h}_t$).

Il vettore $\mathbf{x}$ rappresenta qui un *token*, e le sue componenti vengono dette *features*: un token corrisponde in generale ad un elemento di un testo, solitamente una parola o un suo frammento (es. "playing" può essere divisa in 2 token, "play" e "ing") o elementi della punteggiatura, all'interno di un vocabolario V di possibili token, e ciascun token viene mappato nel vettore di *embegging* $\mathbf{x}$ (i vettori di embedding catturano proprietà semantiche elementari, ad esempio mappando parole con significati simili a posizioni vicine nello spazio di embedding).[2]

[2]Per semplicità discorsiva, nel proseguo un token verrà associato alla parola w (del vocabolario V), già convertito nel suo vettore di embedding. In contesti diversi da NLP, un token potrebbe essere una patch all'interno di un'immagine o un amminoacido all'interno di una proteina.

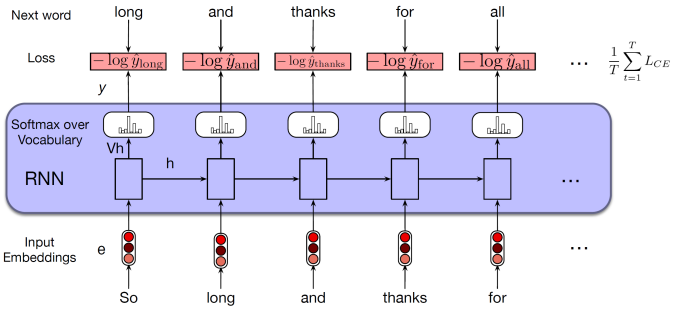


Figura 3. Addestramento di una RNN semplice, nell'apprendere "so long and thanks for all".

L'addestramento di una RNN come modello linguistico segue il paradigma *self-supervised learning* (Fig. 3):

- dato che in ogni sequenza di addestramento $w_1 \dots w_t$ (formata da parole w_i in un vocabolario V) la successiva parola w_j è nota, abbiamo che $\mathbb{P}[w_k|w_1 \dots w_{j-1}] = \mathbf{1}(w_k = w_j)$ modella la vera distribuzione di $w_k \in V$ date le precedenti parole $w_1 \dots w_{j-1}$
- il k-esimo elemento di $\mathbf{y}_j$ rappresenta la probabilità (stimata) di una generica parola w_k date le precedenti $w_1 \dots w_{j-1}$ ossia, con leggero abuso di notazione

$$\mathbf{y}_j[w_k] = \hat{\mathbb{P}}[w_k = w_j|w_1 \dots w_{j-1}] \tag{1}$$

- ad ogni passo la *cross-entropy loss* tra la vera distribuzione $\mathbb{P}[w_k|w_1 \dots w_{j-1}]$ e quella stimata $\hat{\mathbb{P}}[w_k = w_j|w_1 \dots w_{j-1}]$ vale

$$\begin{aligned} L_{CE}(w_j) &= \sum_{w_k \in V} \mathbb{P}[w_k|w_1 \dots w_{j-1}] \log \frac{1}{\hat{\mathbb{P}}[w_k|w_1 \dots w_{j-1}]} \\ &= -\sum_{w_k \in V} \mathbf{1}(w_k = w_j) \log \hat{\mathbb{P}}[w_k|w_1 \dots w_{j-1}] \\ &= -\log \mathbf{y}_j[w_j] \end{aligned}$$

- la rete viene quindi addestrata minimizzando sull'intero corpus di testo la funzione di costo complessiva

$$\frac{1}{T}\sum_{j=1}^{T} L_{CE}(w_j) = -\frac{1}{T}\sum_{j=1}^{T} \log \mathbf{y}_j[w_j] \tag{2}$$

Ad ogni passo la sequenza in ingresso alla rete è quella corretta, e non quella predetta dalla rete fino a quel punto: tale approccio di addestramento è detto *teacher forcing*.

Possiamo dare un'interpretazione ad Eq. (2): essendo la probabilità di una sequenza di parole $w_1 \dots w_t$ espressa da

$$\mathbb{P}[w_1 \dots w_t] = \mathbb{P}[w_1] \cdot \mathbb{P}[w_2|w_1] \cdot \dots \cdot \mathbb{P}[w_t|w_1 \dots w_{t-1}]$$

e tenuto conto di Eq. (1) abbiamo che

$$\hat{\mathbb{P}}[w_1 \dots w_t] = \prod_{i=1}^{t} \mathbf{y}_i[w_i] \tag{3}$$

Eq. 3 costituisce la *verosimiglianza* per la sequenza $w_1 \dots w_t$ fornita dalla rete addestrata: massimizzare Eq. 3 equivale a minimizzare $-\log \hat{\mathbb{P}}[w_1 \dots w_t]$ ossia minimizzare Eq. (2).[3]

[3]Una RNN addestrata è quindi un *predittore a massima verosimiglianza*.

C. Encoder-Decoder con reti RNN

Fig. 4 sintetizza il funzionamento di un encoder-decoder basato su RNN:

- la frase in ingresso e quella in uscita vengono concatenate inserendo tra le due un token fittizio <s> di separazione, ed al termine della concatenazione un altro token <\s> di chiusura, eseguendo su tale sequenza il procedimento di addestramento già visto per le RNN (Fig. 4a);
- durante la fase di inferenza l'output dell'encoder ad ogni passo della sequenza viene ignorato, prendendo in considerazione solo al termine della prima frase lo stato interno dell'encoder che costituisce il contesto $\mathbf{h}$ che viene passato al decoder per procedere alla decodifica avendo come primo ingresso <s>, fino a generare il token di terminazione <\s> (Fig. 4b).

In una migliore formalizzazione (Fig. 4c) il contesto prodotto dall'encoder viene passato al decoder non solo al primo passo ma anche nei successivi.

II. Meccanismo di attenzione

Il problema principale del modello encoder-decoder basato su RNN è l'uso dell'ultimo stato interno $\mathbf{h}_t$ dell'encoder come contesto $\mathbf{c}$, costituendo di fatto un collo di bottiglia (*bottleneck*) in quanto deve condensare al suo interno *l'intero contenuto* della sequenza in ingresso al decoder.

Una possibile strategia per alleviare tale situazione e permettere al decoder di avere visibilità anche sugli stati interni dell'encoder nei passi intermedi è quella di combinarli linearmente in $\mathbf{c}$, in maniera specifica per ogni passo di traduzione: i pesi indicano quanto i passati stati $\mathbf{h}_{e|1} \dots \mathbf{h}_{e|t-1}$ dell'encoder nella sequenza da tradurre siano rilevanti in ciascuno i-esimo punto della sequenza tradotta, ossia

$$\mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}_{e|j} \tag{4}$$

Tale strategia è alla base del meccanismo di *attenzione*, riflessa dai pesi α_{ij} (Fig. 5). In generale L'attenzione permette di costruire rappresentazioni contestuali del significato di un token integrando le informazioni dai token circostanti, aiutando il modello a imparare come i token si relazionano tra loro su ampi intervalli.

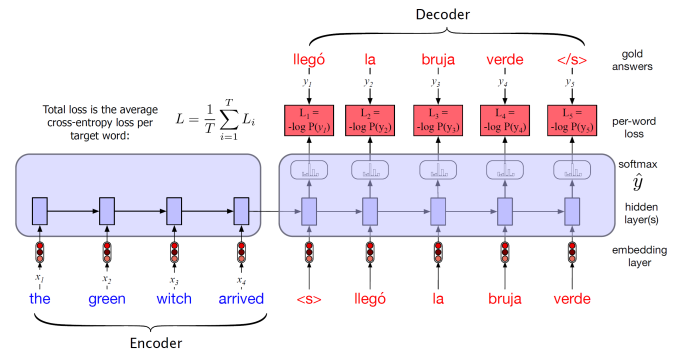
A. Calcolo dei pesi

Per determinare il valore dei pesi α_{ij} è necessario introdurre una funzione di *rilevanza*, per valutare quanto lo stato interno $\mathbf{h}_{d|i-1}$ del decoder in ingresso al passo i sia simile rispetto agli stati assunti $\mathbf{h}_{e|j}$ dall'encoder:

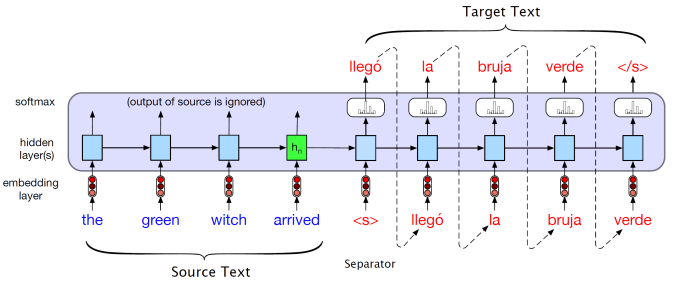
$$\text{score}(\mathbf{h}_{d|i-1}, \mathbf{h}_{e|j}) = \mathbf{h}_{d|i-1}^{\top} \mathbf{h}_{e|j} \tag{5}$$

dove è stato usato il prodotto scalare come indice di similiarità (*dot-product attention*). L'impiego della funzione softmax$(\cdot)$ permette infine di normalizzare complessivamente tale indice su tutti gli stati interni assunti dall'encoder:

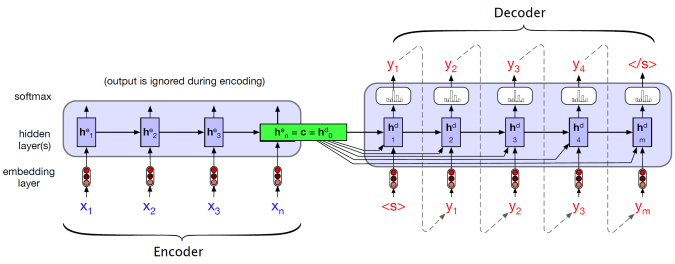
$$\alpha_{ij} = \text{softmax}\big(\text{score}(\mathbf{h}_{d|i-1}, \mathbf{h}_{e|j})\big) = \frac{\exp\left(\mathbf{h}_{d|i-1}^{\top} \mathbf{h}_{e|j}\right)}{\sum_k \exp\left(\mathbf{h}_{d|i-1}^{\top} \mathbf{h}_{e|k}\right)} \tag{6}$$



(a)



(b)



(c)

Figura 4. Modello Encoder-Decoder con RNN semplici per tradurre "the green witch arrived" in "Illegó la bruja verde": in (a), la fase di addestramento; in (b), il processo di traduzione. L'approccio in (c) costituisce infine un'evoluzione del modello, in cui il contesto prodotto dall'encoder viene ricevuto ad ogni passo di decodifica.

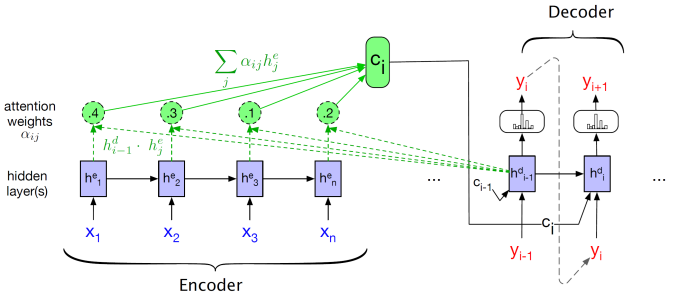


Figura 5. Meccanismo di attenzione per encoder-decoder basati su RNN.

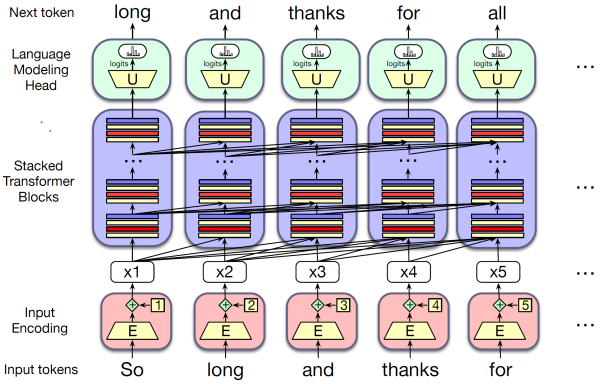


Figura 6. Natural Language Model basato su Transformers.

Una versione più raffinata di Eq. (5) è la seguente:

$$\text{score}(\mathbf{h}_{d|i-1}, \mathbf{h}_{e|j}) = \mathbf{h}_{d|i-1}^{\top} \mathtt{W}_s \mathbf{h}_{e|j} \tag{7}$$

dove i valori di $\mathtt{W}_s$ vengono appresi nella fase di addestramento. Eq. (7) permette inoltre di avere dimensioni distinte tra gli stati dell'encoder e del decoder.

III. I Transformers e l'attenzione

Un Transformer è una particolare rete neurale, che include un meccanismo chiamato *auto-attenzione* (self-attention), con una struttura organizzata in colonne successive di elaborazione come illustrato in Fig. 6:

- alla base è presente un componente di codifica (*Input encoding*) che elabora le parole (token, in generale) in input convertendole in una rappresentazione vettoriale *contestuale* $\mathbf{x}_i$, utilizzando una matrice di embedding $\mathtt{E}$ e un meccanismo per la codifica della posizione del token.
- seguono più blocchi impilati (gli *stacked Transformers blocks*), ognuno dei quali composto da una rete multi-strato (un livello di self-attention, reti feedforward e un livello di normalizzazione), che mappano un vettore di input $\mathbf{x}_i$ nella colonna i (corrispondente al token di input i) a un vettore di output $\mathbf{h}_i$;
- ogni colonna è seguita da un ultimo strato (il *Language Modeling Head*), che passa $\mathbf{h}_i$ attraverso una matrice di *unembedding* $\mathtt{U}$ e una softmax sul vocabolario per generare un singolo token per la colonna in questione.

A. L'attenzione nei Transformers

Il contesto in Eq. (4), con i pesi α_{ij} forniti da Eq. (6), è una forma semplificata di attenzione, qui ripresa in termini non più degli stati interni dell'encoder ma impiegando i token $\mathbf{x}_j$ presenti all'interno di una *finestra di contesto* (Fig. 7):[4]

$$\mathbf{a}_i = \sum_{j \leqslant i} \alpha_{ij} \mathbf{x}_j \tag{8}$$

[4]La trattazione viene limitata a modelli linguisti *causali* che operano da sinistra verso destra, per cui il modello non ha accesso ai token successivi a quello corrente ma solo a quelli passati.

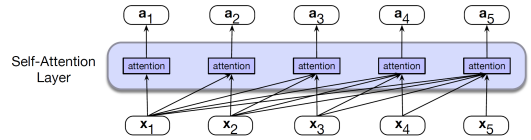


Figura 7. Self-attention causale: ad ogni input $\mathbf{x}_i$, il modello presta attenzione a tutti gli input fino a $\mathbf{x}_i$ incluso.

1) Singola testa di attenzione: La *testa di attenzione* (attention head) è il tipo di attenzione impiegata nei Transformers, in cui gli input $\mathbf{x}_i \in \mathbf{R}^d$ (d è la *dimensionalità del modello*) assumono tre diversi ruoli grazie alle matrici di proiezione $\mathtt{Q} \in \mathbb{R}^{d_k \times d}$, $\mathtt{K} \in \mathbb{R}^{d_k \times d}$, $\mathtt{V} \in \mathbb{R}^{d_v \times d}$, ossia

query in quanto elemento corrente confrontato con gli elementi precedenti

$$\mathbf{q}_i = \mathtt{Q}\mathbf{x}_i \tag{9}$$

chiave nel ruolo di input precedente che viene confrontato con l'elemento corrente per determinare un peso di similarità

$$\mathbf{k}_i = \mathtt{K}\mathbf{x}_i \tag{10}$$

valore come valore di un elemento precedente che viene ponderato e sommato per calcolare l'output per l'elemento corrente

$$\mathbf{v}_i = \mathtt{V}\mathbf{x}_i \tag{11}$$

Attraverso Eq. (9)–(11) Eq. (8) diventa

$$\mathbf{a}_i = \sum_{j \leqslant i} \alpha_{ij}\mathbf{v}_j = \sum_{j \leqslant i} \alpha_{ij}\mathtt{V}\mathbf{x}_j = \mathtt{V}\sum_{j \leqslant i} \alpha_{ij}\mathbf{x}_j \tag{12a}$$

$$\alpha_{ij} = \mathrm{softmax}\big(\mathrm{score}(\mathbf{x}_i, \mathbf{x}_j)\big) \tag{12b}$$

$$\mathrm{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}} = \frac{\mathbf{x}_i^\top \mathtt{Q}^\top \mathtt{K}\mathbf{x}_j}{\sqrt{d_k}} \tag{12c}$$

Fig. 8 mostra il calcolo dell'attenzione nel caso del terzo elemento $\mathbf{a}_3$.

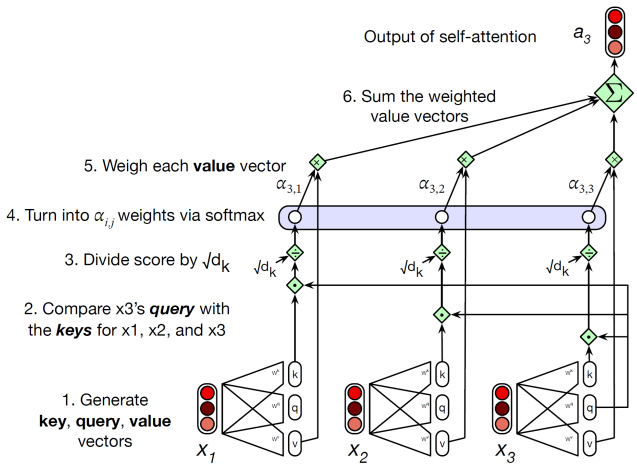


Figura 8. Rappresentazione grafica di Eq. (12).

2) Auto-attenzione: Eq. 12 costituisce una singola testa di attenzione; l'idea alla base dell'auto-attenzione (o *multi-head attention*), impiegando più teste di attenzione, è che invece ogni testa possa prestare attenzione al contesto per scopi diversi come rappresentare diverse relazioni linguistiche tra gli elementi del contesto e il token corrente, o per cercare particolari tipi di modelli nel contesto: ad esempio, alcuni schemi potrebbero essere rilevanti per i tempi verbali, mentre altri potrebbero essere associati al vocabolario.

L'utilizzo di una singola testa di attenzione può portare a una media di questi effetti; in alternativa, possiamo utilizzare più teste di attenzione in parallelo: queste consistono in copie strutturate in modo identico della singola testa, con parametri apprendibili indipendenti che regolano il calcolo delle matrici di query, chiave e valore.

Quindi nell'auto-attenzione abbiamo H teste di attenzione separate che risiedono in strati paralleli alla stessa profondità in un modello, ciascuna con il proprio set di matrici $\mathtt{Q}_h$, $\mathtt{K}_h$, $\mathtt{V}_h$ ($h = 1 \ldots H$):[5]

$$\mathbf{q}_i^h = \mathtt{Q}_h\mathbf{x}_i \tag{13a}$$

$$\mathbf{k}_i^h = \mathtt{K}_h\mathbf{x}_i \tag{13b}$$

$$\mathbf{v}_i^h = \mathtt{V}_h\mathbf{x}_i \tag{13c}$$

$$\alpha_{ij}^h = \mathrm{softmax}\big(\mathrm{score}^h(\mathbf{x}_i, \mathbf{x}_j)\big) \tag{13d}$$

$$\mathrm{score}^h(\mathbf{x}_i, \mathbf{x}_j) = \frac{{\mathbf{q}_i^h}^\top \mathbf{k}_j^h}{\sqrt{d_k}} = \frac{\mathbf{x}_i^\top \mathtt{Q}_h^\top \mathtt{K}_h \mathbf{x}_j}{\sqrt{d_k}} \tag{13e}$$

$$\mathbf{t}_i^h = \sum_{j \leqslant i} \alpha_{ij}^h \mathbf{v}_j^h \tag{13f}$$

$$\mathbf{a}_i = \mathtt{O}\begin{bmatrix} \mathbf{t}_i^1 \\ \vdots \\ \mathbf{t}_i^H \end{bmatrix} \tag{13g}$$

B. Gli altri elementi dei Transformers

Una volta chiarito il meccanismo di attenzione, vediamo più in dettaglio i restanti elementi che compongono i Transformers.

1) Embedding e codifica della posizione: Poiché nei Transfomers non sono presenti nè ricorrenze né convoluzioni, affinché il modello possa utilizzare l'ordine della sequenza, è necessario iniettare alcune informazioni sulla posizione relativa o assoluta dei token nella sequenza.

A tal fine agli embedding per i token in ingresso vengono aggiunti degli *embedding posizionali*, con la stessa dimensione d degli embedding, in modo che i due possano essere sommati (Fig. 9); ad esempio in [6] è impiegata una combinazione di funzioni seno e coseno con frequenze diverse in un modo da catturare le relazioni intrinseche tra le posizioni:[6]

$$\mathrm{pe}(p, 2k) = \sin \frac{p}{c^{2k/d}}$$

$$\mathrm{pe}(p, 2k+1) = \cos \frac{p}{c^{2k/d}}$$

[5] Si noti che la l'auto-attenzione formulata in Eq. (13), analoga a quella adottata in letteratura, include una certa ridondanza nella moltiplicazione successiva della matrice $\mathtt{V}_j$ per ogni testa j con la matrice di output $\mathtt{O}$.

[6] Vale a dire, cattura il fatto che la posizione 4 in un input è più strettamente correlata alla posizione 5 che alla posizione 17.

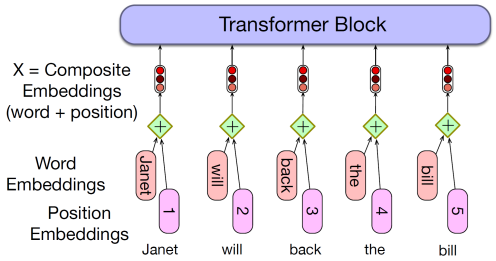


Figura 9. Coodifica della posizione di ciascun token.

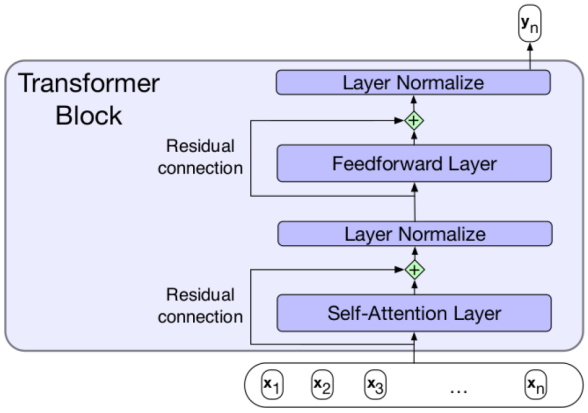


Figura 10. Componenti di un blocco Transformer.

con $c = 10000$, essendo p la posizione del token mentre l'indice k varia tra 0 e d lungo il vettore di embedding posizionale.

2) Transformer block: Oltre al layer di self-attention, ogni blocco Transfomer include altri tre componenti (Fig. 10), ossia[7]

- degli *strati di normalizzazione* sul vettore in ingresso $\mathbf{u}_i = [u_{i1} \dots u_{in}]^\top \in \mathbb{R}^d$ alla posizione i

$$\text{lnorm}(\mathbf{u}_i) = \gamma \frac{\mathbf{u}_i - \mu_i}{\sigma_i} + \beta \tag{14a}$$

$$\mu_i = \frac{1}{n} \sum_k u_{ik} \tag{14b}$$

$$\sigma_i = \sqrt{\frac{1}{n} \sum_k (u_{ik} - \mu)^2} \tag{14c}$$

  dove i valori di γ e β vengono impostati grazie all'addestramento.
- una *rete feed-forward* con uno strato nascosto[8]

$$\text{ff}(\mathbf{u}) = \mathtt{W}_2 \max(\mathbf{0}, \mathtt{W}_1 \mathbf{u} + b_1) + b_2 \tag{15}$$

[7]Indichiamo con $\mathbf{u}$ in Eq. (14)-(15) sia il vettore in ingresso allo strato di normalizzazione che alla rete feed-forward.

[8]Finora la non linearità entrerebbe solo attraverso i pesi di attenzione tramite la funzione softmax, ma vincolando così i vettori di output a trovarsi nel sottospazio formato dai vettori di input: questo limita le capacità espressive dello strato di attenzione, per cui possiamo migliorare la flessibilità dei Transformers post-elaborando l'output di ogni strato utilizzando una rete neurale non lineare standard.

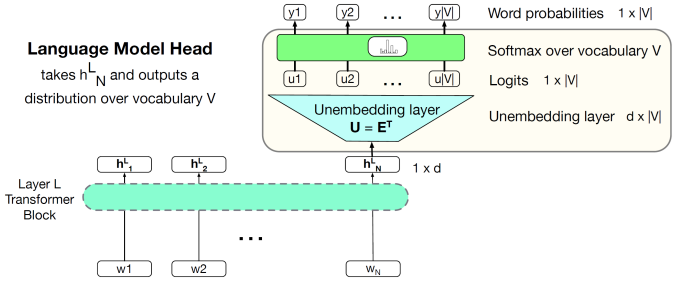


Figura 11. Language Modeling Head.

con $\max(\mathbf{a}, \mathbf{b}) = [\max(a_1, b_1) \dots \max(a_n, b_n)]^\top$. I pesi in $\mathtt{W}_1$ e $\mathtt{W}_2$ sono gli stessi per ogni posizione i, ma variano da blocco a blocco.[9]
- *connessioni residue*, che passano le informazioni da uno strato inferiore a uno superiore senza passare attraverso lo strato intermedio.

3) Language Modeling Head: Come dice il nome Qui utilizziamo la parola testa per indicare il circuito neurale aggiuntivo che modelliamo aggiungendolo all'architettura di base del trasformatore quando applichiamo modelli di trasformatore pre-addestrati a varie attività.

Il compito del Language Modeling Head è di prendere l'output $\mathbf{h}_N^L$ dello strato finale L del Transformer block dall'ultimo token N e usarlo per predire la parola successiva nella posizione $N + 1$, producendo una distribuzione di probabilità sulle parole (Fig. 11):

$$\mathbf{y} = \text{softmax}(\mathtt{E}^\top \mathbf{h}_N^L) \tag{16}$$

Ricordando che $\mathtt{E}$ costituisce la matrice di embedding che converte le parole dal vocabolario V nei vettori numerici $\mathbf{x}$, in Eq. (16) la matrice $\mathtt{E}^\top$ svolge il ruolo opposto.

C. Encoder-Decoder con i Transformers

Fig. 12 illustra come applicare i Transformers nell'architettura Encoder-Decoder: l'encoder utilizza i Transformer blocks già visti, mentre il decoder impiega anche un meccanismo di *attenzione incrociata* (cross-attention) applicato alle uscite terminali dei Transformer blocks nell'encoder.[10]

La cross-attention ha la stessa forma della self-attention in un normale Trasformer block, eccetto che mentre le query provengono dal livello precedente del decodificatore, le chiavi e i valori provengono dall'output del codificatore (Fig. 12b). Indicando con $\mathbf{h}_i$ le uscite di ciascuna testa di attenzione dell'encoder, sia $\mathbf{h}_E = [\mathbf{h}_1^\top \dots \mathbf{h}_H^\top]^\top$; il vettore $\mathbf{a}_c$ di cross-attention viene calcolata quindi come

$$\mathbf{a}_c = \text{softmax}\left(\frac{\mathbf{h}_E^\top \mathtt{Q}_c^\top \mathtt{K}_c \mathbf{h}_E}{\sqrt{d_k}}\right) \mathtt{V}_c \mathbf{h}_E$$

dove $\mathtt{Q}_c$, $\mathtt{K}_c$ e $\mathtt{V}_c$ sono le matrici di proiezione in query, chiave e valori per lo strato di cross-attention.

[9]La stessa rete condivisa viene applicata a ciascuno dei vettori di output così da preservare la capacità del Transformer di elaborare sequenze di lunghezza variabile; la rete neurale può essere migliorata utilizzando una connessione residua sulla sua uscita.

[10]L'attenzione multi-head nell'encoder, tuttavia, è autorizzata a guardare avanti all'intero testo della lingua di origine, quindi non è mascherata.

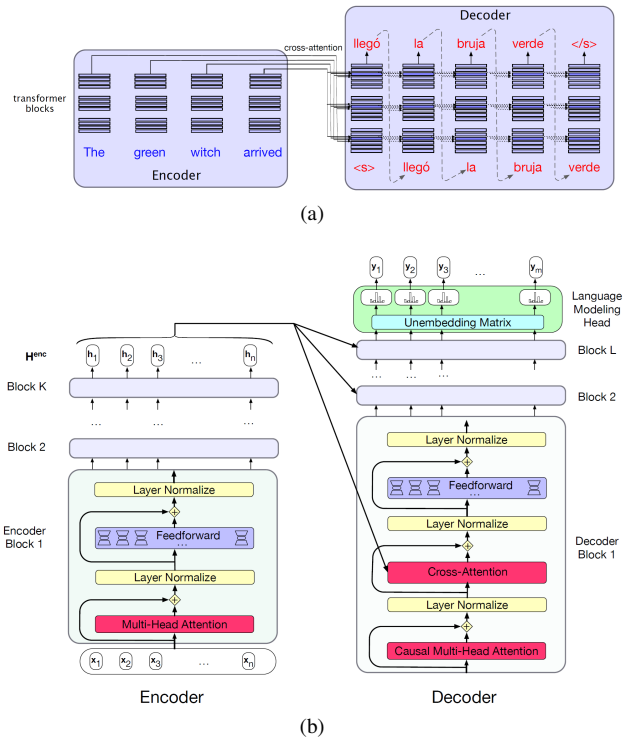


Figura 12. In (a), schema generale per il modello encoder-decoder basato su Transformers. In (b), schema dettagliato del meccanismo di cross-attention.

Per addestrare tale modello viene impiegato la stessa tecnica di self-supervised training già utilizzata per l'encoder-decoder basato su RNN: alla rete viene fornito il testo sorgente e quindi, a partire dal token separatore, viene addestrata in modo autoregressivo per predire il token successivo utilizzando la cross-entropy loss.

IV. Conclusioni sull'attenzione

Da quanto esposto, possiamo ricavare uno schema generale nel meccanismo di attenzione su un insieme di dati [2]; riprendendo il linguaggio introdotto nei Transformers,

- uno dei dati in ingresso riveste il ruolo centrale di *query* $\mathbf{q}$, rispetto al quale enfatizzare o meno l'attenzione dagli altri dati (nel caso degli encoder-decoder basati su RNN la query è il corrente stato interno $\mathbf{h}_i$ della RNN);
- il nucleo del meccanismo di attenzione mappa una sequenza di K vettori $\{\mathbf{k}_j\}$, le *chiavi*, su una distribuzione $\mathbf{a} = [a_1 \dots a_K]^\top$ di pesi a_j, dove i vettori $\mathbf{k}_j$ codificano le caratteristiche dei dati su cui viene calcolata l'attenzione (nel caso degli encoder-decoder basati su RNN le chiavi sono i precedenti stati interni dell'encoder);
- la query $\mathbf{q}$ per il dato i-esimo viene confrontata con ciascuna delle chiavi $\mathbf{k}_j$ per definire un punteggio (*score*) di similarità, su cui costruire una distribuzione di probabilità nei pesi a_{ij} del vettore di attenzione $\mathbf{a}_i$ (solitamente impiegando la softmax sui punteggi di similarità);
- spesso è necessario che la sequenza di chiavi $\mathbf{k}_j$ venga rappresenta biunivocamente anche in una forma diversa, ossia la sequenza dei *valori* $\{\mathbf{v}_j\}$ (nel caso dell'encoder-decoder con RNN tuttavia questa diversificazione non esiste);
- i pesi nel vettore $\mathbf{a}_i$ e la sequenza $\{\mathbf{v}_j\}$ sono infine combinati per produrre una rappresentazione compatta del vettore di *contesto* $\mathbf{c}$.

Riferimenti bibliografici

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(10):4291–4308, 2021.

[3] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 2024. Online manuscript released August 20, 2024.

[4] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[5] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, Red Hook, NY, USA, 2017. Curran Associates Inc.