TRANSFORMER: Il meccanismo di attenzione

Marco Ligabue marco.ligabue1@studenti.unimi.it

Introduzione

I modelli sequenziali, come le reti neurali ricorrenti (RNN) e convolutive (CNN), dominavano la traduzione automatica e il modeling di sequenze.

I transformer vogliono risolvere i maggiori problemi delle RNN:

1. Le RNN processano le serie in modo sequenziale, il che limita la parallelizzazione.
2. Aumentare la lunghezza della sequenza causa difficoltà a modellare le dipendenze a lungo raggio (problema della scomparsa del gradiente)

Introduzione

Un Transformer è un tipo di modello di deep learning introdotto nel paper «Attention Is All You Need» (2017, Google Brain). Questi modelli sono diventati rapidamente fondamentali nell'elaborazione del linguaggio naturale (NLP) e sono stati applicati a una vasta gamma di compiti nell'apprendimento automatico e nell'intelligenza artificiale

Innovazioni

1. **Positional Encoding**: Fornisce informazioni sulla posizione di ciascun token (parti dell'input, come parole o sottoparole in NLP) nella sequenza, consentendo al modello di considerare le informazioni sequenziali della sequenza.

2. **Self-Attention**: L'attenzione è un meccanismo che calcola i pesi per ogni token nella sequenza in relazione a ogni altro token, in modo che il modello possa prevedere i token che probabilmente compariranno nella sequenza.

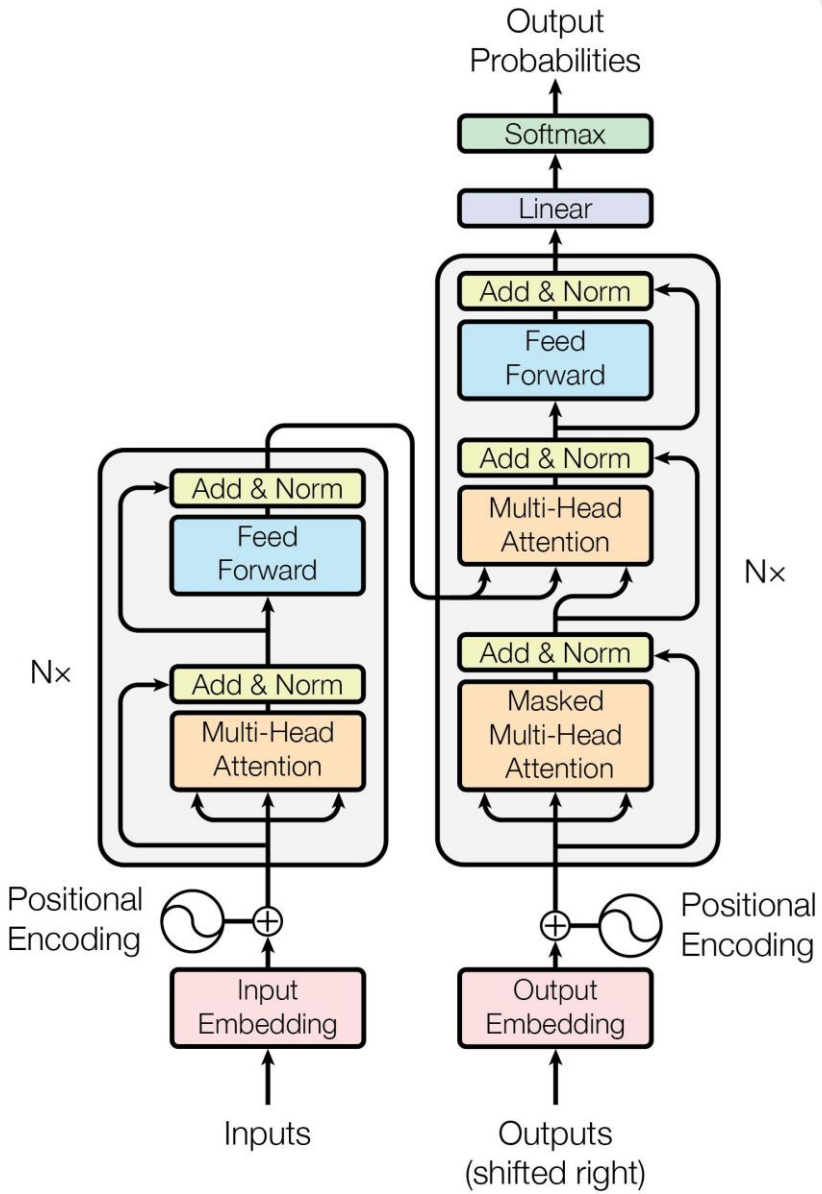
Esempio di funzionamento

Immaginiamo di dover convertire una frase inglese in italiano. Questi sono i passaggi necessari per svolgere questo compito con un modello transformer:

1. **Input embeddings**: La frase di input viene prima trasformata in rappresentazioni numeriche chiamate *embeddings*.
2. **Positional encoding**: insieme di valori o vettori aggiuntivi che vengono sommati agli *embeddings* dei token.
3. **Multi-head attention**: L'*attention* è eseguita in molteplici "test di attenzione" per catturare diversi tipi di relazioni tra i token.
4. **Layer normalization e residual connections**: Il modello utilizza la *layer normalization* e le *residual connections* per stabilizzare e velocizzare l'addestramento.

Esempio di funzionamento

5. **Feedforward neural networks**: L'output del livello di *self-attention* viene passato attraverso strati *feed-forward*.
6. **Stacked layers**: Tipicamente vi sono più livelli impilati uno sopra l'altro. Ogni livello elabora l'output del livello precedente.
7. **Output layer**: Può essere aggiunto un modulo *decoder* separato sopra l'*encoder* per generare la sequenza di output.
8. **Training**: I modelli vengono generalmente addestrati utilizzando l'apprendimento supervisionato, dove imparano a minimizzare una funzione di perdita che quantifica la differenza tra le previsioni del modello e la ground-truth.
9. **Inference**: Dopo l'addestramento, il modello può essere utilizzato per l'*inference* su nuovi dati



Architettura

L'architettura segue uno schema ENCODER-DECODER:

- L'**encoder** trasforma l'input in una rappresentazione continua.
- Il **decoder** genera una sequenza output utilizzando le informazioni dall'encoder.

Encoder

L'encoder è composto da una pila di layer identici.

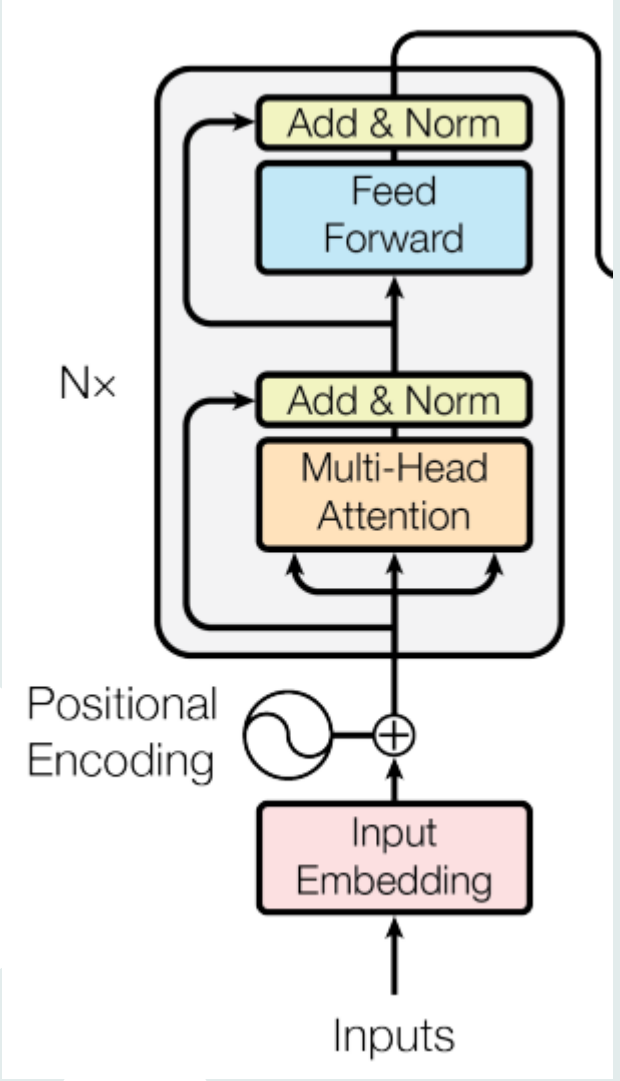
Ogni layer ha due sublayer:

1. Meccanismo di self-attention multi-head
2. Rete feed-forward completamente connessa, applicata per posizione.

Si utilizza una **Residual Connection** attorno a ciascuno dei due sublayer, seguita da una normalizzazione del layer .

L'output di ciascun sublayer è $LayerNorm(x + Sublayer(x))$, dove $Sublayer(x)$ è la funzione implementata dal sublayer stesso.

Per facilitare queste connessioni residue, tutti i sublayer nel modello, così come i layer di embedding, producono output con la medesima dimensione d_{model}.



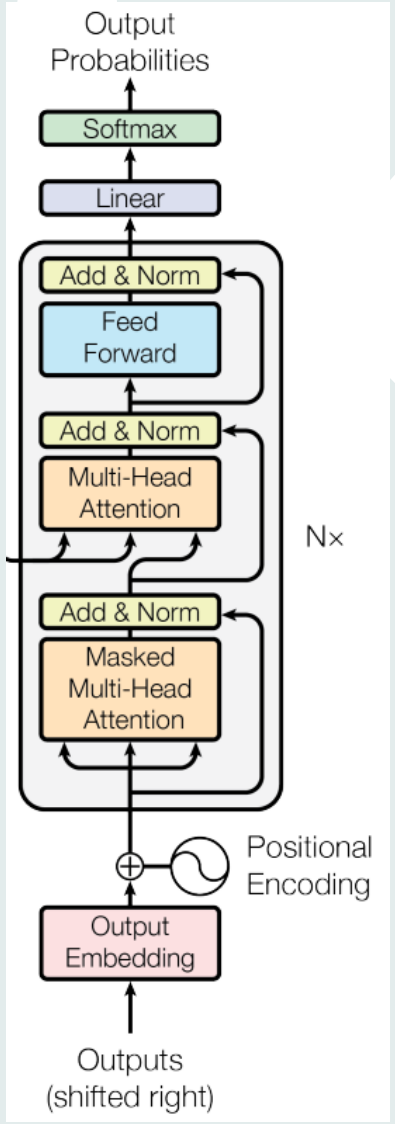
Decoder

Anche il decoder è composto da una pila di layer identici.

Ha un sublayer in più, che esegue una **multi-head attention** sull'output della pila di encoder.

Come nell'encoder, vengono utilizzate **residual connections** attorno a ciascuno dei sublayer, seguite dalla normalizzazione del layer.

Il sublayer di self-attention nella pila del decoder viene modificato per impedire che le posizioni possano fare attenzione alle posizioni successive.



Attention Mechanism

Una funzione di **Attention** può essere descritta come una mappatura di una query e di un insieme di coppie key-value su un output, in cui query, keys, values e output sono tutti vettori.

L'output viene calcolato come una somma ponderata dei valori, dove il peso assegnato a ciascun valore è determinato da una funzione di compatibilità tra la query e la chiave corrispondente.

Due tipi comuni di attention:

- Dot-product
- Additive (usa una rete feed-forward)

Scaled Dot-Product Attention

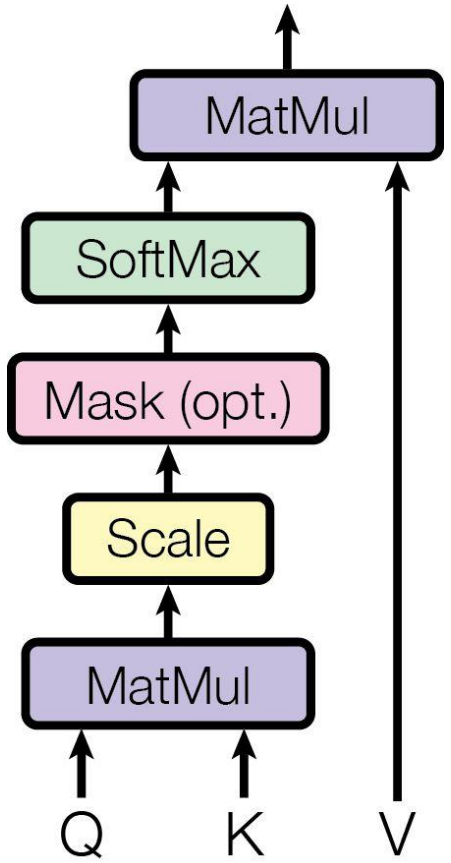
Utilizza query e chiavi di dimensione d_k e valori di dimensione d_v.

L'attenzione su un set di query viene calcolata simultaneamente, raggruppando query, chiavi e valori nelle matrici $Q, K\ e\ V$.

L'output è dato da:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Per grandi d_k, il prodotto scalare aumenta, portando *softmax* in regioni con gradienti molto piccoli; lo scaling aiuta a gestire questo effetto.



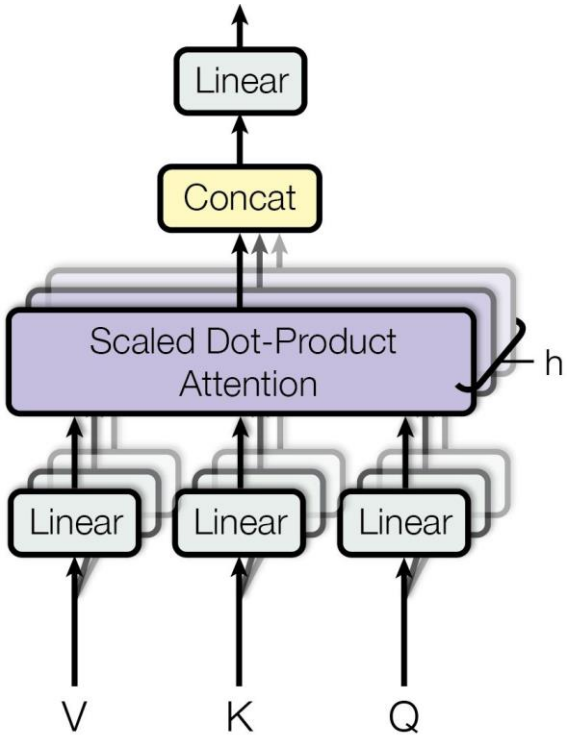
Multi-Head Attention

La *multi-head attention* consente al modello di prestare attenzione congiuntamente a informazioni provenienti da diversi sottospazi di rappresentazione in posizioni differenti.

$$MultiHead(Q,K,V) = Concat(head1,\dots,headh)W^O$$
$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Dove le proiezioni sono matrici dei parametri $W_i^Q \in \mathbb{R}^{d_{model} x d_k}$, $W_i^K \in \mathbb{R}^{d_{model} x d_k}$, $W_i^V \in \mathbb{R}^{d_{model} x d_v}$, $W^O \in \mathbb{R}^{h d_v x d_{model}}$

Con una singola *attention head*, la media ostacola questo processo. Le *queries*, *keys* e *values* vengono poi usate per eseguire la funzione di *attention* in parallelo, producendo output di dimensione d_v.



Encoding posizionale

Per consentire al modello di sfruttare l'ordine della sequenza (dato che non contiene ricorrenza o convoluzione), vengono aggiunti i **positional encodings** agli embeddings in input, alla base delle stack di encoder e decoder.

I positional encodings hanno la stessa dimensione d_{model} degli embeddings, così da poter essere sommati. Sono disponibili diverse opzioni per i positional encodings, sia apprese che fisse.

ESEMPIO: si utilizzano funzioni seno e coseno a diverse frequenze:

Per le dimensioni pari: $$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Per le dimensioni dispari: $$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Dove pos è la posizione e i è la dimensione. Ogni dimensione dell'encoding posizionale corrisponde a una sinusoide con lunghezze d'onda che formano una progressione geometrica da 2π a 10000·2π.

Confronto tra self-attention, layer ricorrenti e convoluzionali

Complessità computazionale per layer:

Self-attention ha una complessità più bassa rispetto ai layer ricorrenti quando la lunghezza della sequenza n è minore della dimensionalità d della rappresentazione.

Parallelizzazione:

Self-attention connette tutte le posizioni con un numero costante di operazioni sequenziali, mentre i layer ricorrenti richiedono O(n) operazioni sequenziali.

Dipendenze a lungo raggio:

Self-attention consente percorsi più brevi tra posizioni di input e output rispetto ai layer ricorrenti e convoluzionali, facilitando l'apprendimento delle dipendenze a lungo raggio.

Layer convoluzionali:

Richiedono più layer O(n/k) o O(logk(n)) per collegare tutte le posizioni, aumentando la lunghezza dei percorsi.

Confronto tra self-attention, RNN e CNN

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n: sequence length
d: representation dimension
k: kernel size of convolutions
r: size of the neighborhood in restricted self-attention.

Grazie per l’attenzione!