Large Language Models: dalle RNN all'Attenzione. Analisi tecnica e introduzione alle allucinazioni.

Eduardo De Filippis
www.eduardodefilippis.com

L'avvento dei large language models (LLM) basati su architettura Transformer ha segnato un punto di discontinuità nella storia del natural language processing (NLP). Se le precedenti architetture ricorrenti (es. RNN e LSTM) trattavano il linguaggio come una sequenza strettamente lineare, soffrendo il problema del "vanishing gradient" su lunghe dipendenze, i moderni LLM parallelizzano l'elaborazione posizionale introducendo il concetto di self-attention. È bene ricordare che ciò è vero durante il training (grazie al masking). Durante l'inferenza (generazione), il processo resta intrinsecamente sequenziale (autoregressivo) a causa della dipendenza dal token precedente.

Tuttavia, ridurre un LLM a un semplice "chatbot" sarebbe un errore. Infatti, da un punto di vista statistico, questi sistemi sono modellatori di distribuzioni di probabilità condizionata su sequenze discrete di token.

Di seguito viene presentata una parte della matematica dietro il meccanismo di attenzione, l'iniezione posizionale e le implicazioni stocastiche delle strategie di decoding. Dettagli e precisazioni troppo tecniche sono omessi per evitare di appesantire la lettura.

1 Definizione formale e probabilistica

Un language model è, nella sua essenza, una funzione che assegna una probabilità a una sequenza di parole (token) $w_1, w_2, ..., w_T$. L'obiettivo in fase di pre-training (spesso self-supervised learning) è massimizzare la verosimiglianza della distribuzione congiunta $P(w_1, ..., w_T)$, fattorizzabile secondo la chain rule:

$$P(w_1, ..., w_T) = \prod_{t=1}^{T} P(w_t | w_1, ..., w_{t-1})$$

Il modello apprende i parametri θ che minimizzano la cross-entropy loss sul corpus di addestramento:

$$\mathcal{L}(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \log P(w_t | w_{<t}; \theta)$$

Si precisa che, nella sommatoria della Loss, spesso si usa $t = 1$ fino a T, ma il primo token w_1 di solito è un token speciale, come <sos> o [CLS], che non ha un contesto precedente). Dunque, la predizione inizia tecnicamente dal secondo token basandosi sul primo.

2 L'architettura Transformer

Sebbene l'Attenzione sia il componente più celebre, l'efficacia del Transformer risiede nella sinergia di tre componenti fondamentali che permettono di superare i limiti delle RNN.

2.1 Scaled Dot-Product Attention

L'innovazione del paper "Attention is All You Need" (Vaswani et al., 2017) consiste nel proiettare ogni token in tre vettori tramite matrici di pesi W^Q, W^K, W^V: **Query** (ciò che cerco), **Key** (ciò che offro), **Value** (il contenuto). La formula canonica è:

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

Dove il termine di scaling $\frac{1}{\sqrt{d_k}}$ previene la saturazione della Softmax.

Inoltre, l'architettura utilizza la Multi-Head Attention. Di conseguenza, il calcolo viene parallelizzato su h "teste" differenti, permettendo al modello di catturare contemporaneamente relazioni in sottospazi semantici diversi (es. una testa si focalizza sulla sintassi e un'altra sulle anafore).

2.2 Positional encoding

Un punto critico spesso trascurato è che l'equazione dell'attenzione è "permutation invariant". Ovvero, senza un'informazione esplicita, per il modello, la frase "Il cane morde l'uomo" è matematicamente identica a "L'uomo morde il cane". Per ovviare a ciò, viene sommato (o iniettato) un vettore posizionale P_t all'embedding del token E_t.

Nelle architetture moderne (come LLaMA), si utilizzano i "rotary positional embeddings" (RoPE), che ruotano i vettori Query e Key nello spazio complesso di un angolo proporzionale alla loro posizione assoluta m:

$$f(x, m) = xe^{im\theta}$$

In termini formali, per una coppia di componenti del vettore (x_i, x_{i+1}), la trasformazione RoPE agisce come una rotazione nel piano:

$$\begin{pmatrix} f_q(x,m)_i \\ f_q(x,m)_{i+1} \end{pmatrix} = \begin{pmatrix} \cos m\theta_i & -\sin m\theta_i \\ \sin m\theta_i & \cos m\theta_i \end{pmatrix} \begin{pmatrix} x_i \\ x_{i+1} \end{pmatrix}$$

Questo permette al prodotto scalare QK^T di dipendere solo dalla distanza relativa tra i token, migliorando la generalizzazione su sequenze lunghe.

Si evidenzia che, nella realtà, $\theta_i = 10000^{-2i/d}$. Dove d è la dimensione della testa (d_k), non del modello intero (d_{model}).

Per essere estremamente precisi, in RoPE l'indice i della θ si riferisce alla coppia di dimensioni. Se d è la dimensione della testa, i varia da 0 a $d/2 - 1$.

Ancora, è bene ricordare che il valore 10000 è la base standard introdotta originariamente. Nelle implementazioni più recenti per estendere la context window (come in Llama 3 o modelli Long-Context), tale base viene spesso modificata (es. 500 000)

2.3 Feed-Forward Networks (FFN)

Dopo il blocco di attenzione, l'informazione passa attraverso una rete densa (MLP) che lavora "position-wise" (indipendentemente su ogni token). Dunque, l'Attenzione serve a "mescolare" le informazioni tra parole diverse, l'FFN serve a elaborare e "ragionare" sulle informazioni raccolte.

3 Applicazione pratica: implementazione dell'Attenzione

Per comprendere la complessità computazionale quadratica $O(N^2)$, viene proposta di seguito una implementazione da zero del meccanismo di attenzione.

```python
import numpy as np
from scipy.special import softmax

####################

def scaled_dot_product_attention(query, key, value, mask=None):
    """
    Args:
        query: (batch_size, num_heads, seq_len_q, depth)
        key:   (batch_size, num_heads, seq_len_k, depth)
        value: (batch_size, num_heads, seq_len_k, depth_v)
```

```
        mask:  maschera opzionale (look-ahead)
    """
    d_k = query.shape[-1]

###################

    # Step 1 - Matmul: (Batch, Heads, Seq_Q, Seq_K)
    # trasponiamo K per allineare le dimensioni per il prodotto scalare
    scores = np.matmul(query, key.transpose(0, 1, 3, 2))

    # Step 2 - Scaling
    scores = scores / np.sqrt(d_k)

    # Step 3 - Masking
    # per decoder autoregressivi
    if mask is not None:
        scores += (mask * -1e9)
        # precisazione: questa impostazione di scores presuppone che la mask sia composta da 1 dove si vuole oscurare e 0 dove si vuole mantenere. Solitamente le librerie (come PyTorch) usano maschere booleane invertite.

    # Step 4 - Softmax
    attention_weights = softmax(scores, axis=-1)
    # nota tecnica importante: La funzione scipy.special.softmax    sicura, ma se dovessimo implementarla manualmente, dovremmo sottrarre il valore massimo (scores - np.max(scores)) per evitare overflow degli esponenziali. Utilizzando SciPy, il problema non sussiste.

    # Step 5 - Output
    # somma pesata dei valori
    output = np.matmul(attention_weights, value)

    return output, attention_weights

###################

# esempio dummy
np.random.seed(2025)
seq_len, d_model = 4, 8
Q = np.random.rand(1, 1, seq_len, d_model)
K = np.random.rand(1, 1, seq_len, d_model)
V = np.random.rand(1, 1, seq_len, d_model)

output, weights = scaled_dot_product_attention(Q, K, V)
print(f"Shape Output: {output.shape}")
```

Listing 1: Implementazione low-level della scaled dot-product attention

Si evidenzia che la maschera (masking) è fondamentale per evitare il "data leakage". Infatti, durante il training, il token al tempo t non deve poter "vedere" i token $t+1$.

4 Il dilemma del decoding e le "allucinazioni"

Una volta calcolate le probabilità (logits), il modello deve selezionare il token successivo.

4.1 Temperature (τ) e Top-P

La temperatura modifica la forma della distribuzione prima del campionamento:

$$P_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

Con $\tau > 1$, la distribuzione si appiattisce ("creatività" e rischio allucinazione). Con $\tau < 1$, i picchi si accentuano. Il Top-P (nucleus sampling) seleziona il più piccolo insieme di token la cui probabilità cumulativa supera una soglia p (es. 0.9), adattando dinamicamente il pool di scelte alla "sicurezza"

del modello nel contesto corrente. Ciò previene la dispersione delle probabilità quando il modello è incerto.

L'allucinazione è spesso statistica: il modello campiona dalla "coda lunga" della distribuzione o confida in pattern appresi che non corrispondono alla realtà fattuale (conoscenza parametrica errata).

5 Conclusioni

Gli LLM non sono database di conoscenza, ma motori di ragionamento probabilistico. La comprensione della distinzione tra i pesi del modello (conoscenza parametrica) e il contesto fornito (conoscenza in-context) è essenziale. Le sfide future vertono sull'efficienza (es. quantizzazione, pruning) e sull'integrazione di meccanismi di verifica fattuale per mitigare la natura stocastica della generazione.