**Appunti da: Build a Large Language Model (From Scratch)**

**Introduzione: Le 3 Fasi Principali per Costruire un LLM**

Il libro guida l'utente attraverso la costruzione di un Large Language Model (LLM) partendo da zero. L'intero processo è suddiviso in tre fasi fondamentali, come illustrato nello schema seguente.

Immagine che contiene testo, schermata, Carattere, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.

**Figura 0.1 – Schema delle 3 fasi di sviluppo di un LLM.**

Le tre fasi principali della programmazione di un **modello linguistico di grandi dimensioni (LLM)** sono:

1. **Implementazione dell'Architettura e Preparazione dei Dati (Fase 1)**: Si definisce la struttura del modello e si preparano i dati di testo per l'addestramento.
2. **Pre-addestramento (Fase 2)**: Si addestra il modello su un'enorme quantità di dati non etichettati per creare un **modello di base** (*foundation model*) con una comprensione generale del linguaggio.
3. **Fine-tuning (Fase 3)**: Si adatta il modello di base a compiti specifici (come rispondere a domande o classificare testo) addestrandolo su un dataset più piccolo e mirato.

**Capitolo 1: Comprendere i Large Language Models (LLM)**

**1.1 Cos'è un LLM?**

Un LLM è una **rete neurale profonda** addestrata su enormi quantità di dati testuali. Il termine "Large" (grande) si riferisce a due aspetti:

* **Dimensione del modello**: Numero di parametri (pesi interni) che vanno da milioni a miliardi.
* **Dimensione del dataset**: Quantità immensa di testo usata per l'addestramento.

Il compito fondamentale su cui si basano è la **predizione della parola successiva** in una sequenza. Questo semplice obiettivo permette al modello di apprendere grammatica, contesto, stile e relazioni complesse all'interno del linguaggio.

**1.2 Fasi di Costruzione e Utilizzo**

Il processo di creazione di un LLM si articola in due momenti principali:

1. **Pre-addestramento (Pre-training)**:
   * Il modello viene addestrato su un vasto corpus di testo non etichettato (libri, articoli, Wikipedia, ecc.).
   * Il risultato è un **modello di base** (*foundation model*), capace di completare frasi ma non ancora specializzato.
2. **Messa a punto (Fine-tuning)**:
   * Il modello di base viene ulteriormente addestrato su un dataset più piccolo e specifico, contenente esempi etichettati.
   * Questo processo adatta il modello a compiti specifici come la classificazione del testo, la traduzione o l'assistenza conversazionale.

Immagine che contiene testo, diagramma, schermata, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 1.1 – Flusso di Pre-addestramento e Fine-tuning.**

**1.3 Introduzione all'Architettura Transformer**

Gli LLM moderni, come GPT, si basano sull'architettura **Transformer**. La sua caratteristica chiave è il **meccanismo di auto-attenzione (self-attention)**, che permette al modello di pesare l'importanza di parole diverse all'interno di una sequenza per comprendere meglio il contesto.

L'architettura Transformer originale era composta da due parti:

* **Encoder**: Elabora il testo di input e lo converte in una rappresentazione numerica.
* **Decoder**: Prende la rappresentazione dell'encoder e genera il testo di output.

Modelli come **BERT** si concentrano sull'**encoder** (ottimi per classificazione), mentre modelli come **GPT** si basano sul **decoder** (ideali per la generazione di testo).

Immagine che contiene testo, schermata, diagramma, numero

Il contenuto generato dall'IA potrebbe non essere corretto.

**Figura 1.2 – Confronto tra architetture basate su encoder (BERT) e decoder (GPT).**

I modelli GPT sono **autoregressivi**: generano testo una parola alla volta, e ogni nuova parola generata viene aggiunta all'input per la predizione successiva.

Immagine che contiene testo, schermata, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 1.3 – Funzionamento iterativo di un modello GPT.**

**Capitolo 2: Lavorare con i Dati di Testo**

Prima di addestrare un LLM, il testo grezzo deve essere convertito in un formato numerico.

**2.1 Word Embedding**

Un **embedding** è una rappresentazione vettoriale (una lista di numeri) di una parola. Parole con significati simili avranno vettori simili nello spazio multidimensionale.

**2.2 Tokenizzazione**

La **tokenizzazione** è il processo di suddivisione di un testo in unità più piccole chiamate **token** (parole, punteggiatura, ecc.). I modelli GPT usano tecniche avanzate come il **Byte Pair Encoding (BPE)**, che scompone le parole sconosciute in sotto-unità, permettendo di gestire qualsiasi vocabolo.

Immagine che contiene testo, Carattere, linea, schermata

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 2.1 – Esempio di come BPE scompone una parola sconosciuta.**

**2.3 Creazione degli ID Token e del Vocabolario**

Dopo la tokenizzazione, ogni token viene mappato a un numero intero (**ID Token**) tramite un **vocabolario**.

* **Encode**: Converte da testo a ID Token.
* **Decode**: Converte da ID Token a testo.

**2.4 Campionamento dei Dati e Positional Encoding**

Per addestrare il modello, il testo viene trasformato in coppie di **input-target** usando una "finestra scorrevole" (*sliding window*). Il target è semplicemente l'input spostato di una posizione in avanti.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 2.2 – Creazione di coppie input-target.**

Infine, agli embedding dei token viene sommato un **positional embedding**, un vettore che fornisce al modello informazioni sulla posizione di ogni token nella sequenza, superando un limite intrinseco dell'architettura Transformer.

Immagine che contiene testo, schermata, Carattere, numero

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 2.3 – L'embedding finale è la somma del token embedding e del positional embedding.**

**Capitolo 3: Programmare i Meccanismi di Attenzione**

Questo capitolo esplora il cuore dell'architettura Transformer: il meccanismo di attenzione.

**3.1 Il Problema delle Sequenze Lunghe: Limiti delle Architetture Precedenti**

Le architetture precedenti agli LLM, come le **Reti Neurali Ricorrenti (RNN)**, faticavano a gestire sequenze lunghe. Esse comprimevano l'intera informazione della frase di input in un singolo vettore di stato ("hidden state"), creando un **collo di bottiglia contestuale**. Questo rendeva difficile per il modello ricordare informazioni presenti all'inizio di un lungo testo.

Immagine che contiene testo, elettronica, schermata, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 3.1 – Le RNN comprimono l'informazione in un singolo stato, perdendo dettagli su lunghe distanze.**

**3.2 Auto-Attenzione (Self-Attention): Il Concetto Base**

L'auto-attenzione risolve questo problema permettendo a ogni token di "guardare" direttamente a tutti gli altri token nella sequenza di input. In questo modo, il modello può calcolare l'importanza relativa di ogni parola rispetto alle altre per costruire una rappresentazione più ricca di contesto.

Il processo si può riassumere in 3 passaggi:

1. **Calcolo degli Attention Score**: Per ogni token ("query"), si calcola un punteggio di somiglianza (solitamente tramite *dot product*) con tutti gli altri token.
2. **Normalizzazione in Attention Weight**: I punteggi vengono normalizzati con una funzione **softmax**, trasformandoli in pesi che sommano a 1. Questi pesi indicano "quanta attenzione" un token deve prestare agli altri.
3. **Creazione del Vettore di Contesto (Context Vector)**: I pesi di attenzione vengono usati per calcolare una somma pesata dei vettori di embedding originali. Il risultato è un nuovo vettore "arricchito di contesto" per ogni token.

**3.3 Auto-Attenzione con Pesi Addestrabili (Scaled Dot-Product Attention)**

Nella pratica, il meccanismo è più sofisticato. Gli embedding di input non vengono usati direttamente, ma vengono proiettati in tre spazi distinti attraverso delle matrici di peso addestrabili (**Wq, Wk, Wv**):

* **Query (Q)**: Rappresenta il token corrente che sta "cercando" informazioni.
* **Key (K)**: Rappresenta i token nella sequenza che "offrono" informazioni.
* **Value (V)**: Rappresenta il contenuto effettivo dei token.

Il calcolo avviene confrontando la Query di un token con le Key di tutti gli altri per ottenere i pesi di attenzione, che vengono poi applicati ai vettori Value. La formula include anche un fattore di scala per stabilizzare l'addestramento.

Immagine che contiene testo, diagramma, schermata, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.

**Figura 3.2 – Flusso completo dell'auto-attenzione con matrici Query, Key e Value.**

**3.4 Attenzione Causale (Causal Attention Mask)**

Per i modelli generativi come GPT, che predicono il futuro una parola alla volta, è fondamentale impedire al modello di "sbirciare" i token successivi durante l'addestramento. A questo scopo si applica una **maschera causale** (*causal mask*), che azzera i pesi di attenzione per tutti i token che si trovano in posizioni future rispetto a quella corrente.

Immagine che contiene testo, schermata, Carattere, Parallelo

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 3.3 – La maschera causale impedisce a un token di "vedere" i token successivi.**

**3.5 Attenzione Multi-Testa (Multi-Head Attention)**

Invece di eseguire un singolo calcolo di attenzione, l'architettura Transformer lo esegue più volte in parallelo. Ognuna di queste esecuzioni parallele è chiamata **"testa" (head)**.

* **Obiettivo**: Permettere al modello di catturare diversi tipi di relazioni contestuali contemporaneamente (es. una testa potrebbe focalizzarsi sulle relazioni sintattiche, un'altra su quelle semantiche).
* **Funzionamento**: Gli output di tutte le teste vengono concatenati e proiettati di nuovo nella dimensione originale, combinando le diverse informazioni apprese.

Immagine che contiene testo, schermata, diagramma, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.  
**Figura 3.4 – L'attenzione Multi-Testa esegue più meccanismi di attenzione in parallelo.**

Con questi mattoncini fondamentali (self-attention, causal mask, multi-head attention), siamo pronti a costruire il blocco Transformer completo, che sarà l'elemento centrale del nostro LLM.