Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

STRUMENTO DI MECHANISTIC INTERPRETABILITY PER MODELLI LINGUISTICI BASATI SU TRANSFORMER

Relatore:
Chiar.mo Dott.
STEFANO PIO ZINGARO

Controrelatore:
Chiar.mo Prof.
MAURIZIO GABBRIELLI

Presentata da:
DANIELE POLIDORI

Sessione III
Anno Accademico 2024/2025

Introduzione

I modelli di *machine learning* basati su *Transformer* [10] rivestono un ruolo di ampio rilievo nel panorama attuale, in particolare per quanto riguarda l'elaborazione del linguaggio naturale (NLP) [5, 3]. Si tratta di modelli estremamente potenti, tuttavia carenti nell'interpretabilità [3, 2, 1]. Risulta socialmente necessario l'utilizzo di sistemi sicuri, affidabili e trasparenti. Tale obiettivo richiede una comprensione esaustiva del relativo funzionamento interno, rendendo un imperativo cruciale fare luce sui processi decisionali da cui sono guidati [2, 3, 1].

Questo lavoro si prefigge lo scopo di trovare uno strumento per interpretare il funzionamento dei moderni modelli linguistici basati su Transformer.

I metodi di *explainable AI*, nel panorama di ricerca tipicamente presentato, forniscono spiegazioni per specifici *stakeholder* in particolari contesti. Kästner e Crook [2] definiscono tale scenario fondamentalmente incompleto, suggerendo di considerare anche il potenziale di strategie di ricerca coordinate al fine di scoprire l'organizzazione funzionale dei sistemi. Un approccio che si propone di compensare questa mancanza è quello della *mechanistic interpretability*, un campo di ricerca emergente, che parte dalla premessa che i sistemi di intelligenza artificiale, sufficientemente complessi, vanno studiati e spiegati tramite le stesse lenti usate per gli organismi biologici, piuttosto che trattarli come meri artefatti tecnologici [2].

Nell'ambito della mechanistic interpretability, Elhage e colleghi [9] hanno creato un *framework* matematico per descrivere a livello teorico la struttura e il funzionamento dei modelli linguistici basati su Transformer, facendo

un primo passo verso il *reverse engineering* dei calcoli eseguiti da tali sistemi. Nella loro ricerca, essi hanno utilizzato una libreria — senza rilasciarne l'implementazione — chiamata *Garçon* [8], uno strumento per l'analisi degli elementi interni dei modelli, in particolare quelli di grandi dimensioni, che facilita l'accesso alle attivazioni e ai parametri [9].
Neel Nanda ha guidato lo sviluppo di una libreria chiamata *TransformerLens* [6] — le cui caratteristiche fondamentali sono state fortemente ispirate dall'interfaccia di Garçon [8] e dal relativo framework matematico [9] — che consente l'analisi di modelli linguistici basati su Transformer, secondo un approccio di mechanistic interpretability, esponendo le relative attivazioni interne.
Tale strumento si pone come risposta all'esigenza presentata inizialmente, in linea con l'obiettivo prefissato. TransformerLens, infatti, consente di verificare ipotesi sul funzionamento di un modello linguistico basato su Transformer.

Il lavoro sarà strutturato nella maniera seguente.
Nel Cap. 1 verrà presentato il panorama di riferimento in cui si inserisce questa ricerca. In particolare, nella Sez. 1.1 si esporrà lo stato dell'arte, con particolare riferimento alla mechanistic interpretability, illustrandone tecniche e un possibile flusso di lavoro (Sez. 1.1.1). Nella Sez. 1.2 si illustrerà il framework matematico presentato da Elhage e colleghi [9].
Nel Cap. 2 si parlerà della libreria TransformerLens [6], mostrandone le caratteristiche principali (Sez. 2.1), la corrispondenza con il framework matematico [9] (Sez. 2.2) e l'utilizzo pratico di tale strumento (Sez. 2.3).
Infine, nelle Conclusioni, si enunceranno i limiti e i possibili sviluppi futuri di questo lavoro.

Indice

Elenco delle figure

Elenco dei codici

Capitolo 1

Mechanistic Interpretability

Ad oggi, i principali modelli utilizzati nel campo dell'elaborazione del linguaggio naturale (NLP) sono basati sull'architettura *Transformer* [10, 5, 3]. Si tratta di modelli estremamente potenti, tuttavia carenti nell'interpretabilità [3, 2, 1]. I programmatori stessi difettano di una piena comprensione dei meccanismi interni di tali sistemi, che, così come gran parte degli altri modelli di reti neurali, vengono utilizzati come delle "*black-box*" [4, 3].
Risulta socialmente necessario, in particolare nei campi ad alto rischio, come quello medico, finanziario e gli scenari in cui è in gioco la vita umana, l'utilizzo di sistemi sicuri, affidabili e trasparenti. Tale obiettivo, reso prioritario dalla rapida crescita e ampia diffusione di tali modelli, richiede una comprensione esaustiva del relativo funzionamento interno, rendendo un imperativo cruciale fare luce sui processi decisionali da cui sono guidati [2, 3, 1].
Questo lavoro si inserisce nell'ampio panorama di ricerca volto all'individuazione di strategie, approcci e strumenti in grado di far fronte all'opacità dei modelli.

1.1 Stato dell'Arte

Nel *machine learning (ML)* e in particolare nel *deep learning*, le reti neurali artificiali profonde (ANN) sono modelli spesso complessi e opachi. Il loro

funzionamento viene espresso come approssimazione di funzioni attraverso il contributo di milioni o miliardi di parametri della rete, i cui valori vengono appresi durante un processo di addestramento automatizzato (*training*). Se, da un lato, questo permette di ottenere comportamenti molto sofisticati, dall'altro non esclude la possibilità di averne di indesiderati e non previsti. Infatti, essendo l'organizzazione funzionale della rete appresa in automatico, il funzionamento interno, che fa corrispondere un determinato output a un certo input, spesso rimane, almeno in parte, sconosciuto [2].

Per far fronte a questa situazione e al bisogno di sistemi trasparenti, necessari, come già detto, per un uso sicuro nella società, è stato sviluppato il campo di ricerca dell'*explainable AI (XAI)* [2]. Uno dei pilastri principali è l'interpretabilità[1], cioè la capacità di spiegare il comportamento dei modelli di ML, in termini comprensibili per un essere umano[3, 4].

Conoscere tali strumenti permette di avere una chiara visione sulle rispettive capacità e limiti, in modo tale da individuare possibili *bug* e *bias*, nonché aree di miglioramento delle performance, evitando rischi e aprendo la strada a modelli più robusti, efficaci e affidabili [3]. L'interpretabilità favorisce il monitoraggio delle capacità del modello nel corso del tempo, permette di confrontare modelli differenti e consente lo sviluppo di strumenti affidabili, etici e sicuri da poter essere impiegati nelle applicazioni del mondo reale [3].

Questo lavoro si concentra sull'interpretabilità di modelli linguistici (*language model*) basati sull'architettura Transformer. La maggior parte dei *large language model (LLM)* moderni — tra i quali, alcuni esempi più noti sono *GPT-2* (e i successivi dell'omonima serie, rilasciati da *OpenAI*) e *Claude* (rilasciato da *Anthropic*) — sono basati sui Transformer [3, 5]. Per via dell'opacità dovuta alla vastità dei dati d'addestramento, delle dimensioni dell'architettura e della complessità della struttura appresa, interpretare tali modelli risulta più difficile [3].

[1] È una questione di ricerca dibattuta la differenza tra *interpretability* ed *explainability*. Zhao e colleghi [3] utilizzano i due termini in maniera interscambiabile. Ai fini di questo lavoro, risulta secondario entrare nel merito di tale questione. Con il termine "interpretabilità", quindi, si farà riferimento al concetto così come presentato.

L'XAI si occupa in larga parte di sviluppare soluzioni analitiche con cui rendere trasparenti sistemi opachi [2, p. 4]. Dal momento che il tipo d'informazione, necessaria per fare luce su un determinato sistema, dipende da svariati fattori, i ricercatori hanno sviluppato una gamma di metodi con proprietà differenti. La ricerca in tale ambito si propone di fornire la spiegazione più indicata per l'applicazione in esame, nel relativo contesto e per lo specifico obiettivo [2]. Tra gli esempi più noti ci sono i metodi basati su *feature attribution*, che puntano a misurare l'importanza di ciascuna *feature* in input, come parole, frasi e porzioni di testo, rispetto alla predizione del modello. Ne fanno parte le tecniche di attribuzione basate sul gradiente, quelle di decomposizione e i metodi basati su modelli surrogati. Le prime misurano l'importanza analizzando le derivate parziali dell'output rispetto a ogni dimensione dell'input. Le seconde mirano a scomporre il punteggio di rilevanza in contributi lineari provenienti da componenti del modello, quali *attention head*, *token* e attivazioni neuronali. Gli ultimi usano modelli più semplici per spiegare singole previsioni di altri più complessi; tali sistemi surrogati includono alberi di decisione, modelli lineari e altri sistemi *white-box* che, per loro natura, sono interpretabili *by-design*, quindi comprensibili più facilmente per un umano. Un noto esempio è LIME (*Local Interpretable Model-Agnostic Explanations*): per generare spiegazioni per un caso specifico, il modello surrogato viene addestrato su dati campionati localmente attorno a quel caso, per approssimare il comportamento dell'originale nella regione locale. In questo contesto, si inserisce anche SHAP (*SHapley Additive exPlanations*), un approccio basato sulla teoria dei giochi, che considera le feature come giocatori in un gioco cooperativo di predizione e assegna a ciascun loro sottoinsieme un valore, che riflette il rispettivo contributo alla predizione del modello. Anziché costruire un modello di spiegazione locale per ciascuna istanza, SHAP calcola i valori utilizzando l'intero *dataset* [3].

1.1.1 Strategie di Ricerca per Scoprire l'Organizzazione Funzionale dei Sistemi

Come già detto, i metodi di XAI, nel panorama di ricerca tipicamente presentato, forniscono spiegazioni per specifici *stakeholder* in particolari contesti. Kästner e Crook [2] definiscono tale scenario fondamentalmente incompleto, suggerendo di considerare anche il potenziale di strategie di ricerca coordinate al fine di scoprire l'organizzazione funzionale dei sistemi. Le tecniche tradizionali di XAI sono in grado di fare luce su quali feature in input influenzino un determinato esito o su come queste debbano essere modificate per produrre un cambiamento nel risultato. Tuttavia, questo non risulta sufficiente per comprendere come questi sistemi funzionano nella loro interezza. Solo comprendendo come l'organizzazione funzionale suscita pattern comportamentali, potremo anticipare la risposta dei modelli a nuovi input e il loro comportamento in contesti inesplorati. L'interesse è rivolto alla struttura interna appresa dalla rete, detta *emergent structure*, che emerge attraverso le procedure di addestramento automatizzato. Solitamente, una comprensione profonda di questa struttura non è estrapolabile applicando metodi di XAI individuali personalizzati per contesti specifici [2]. Per di più, la rapida crescita dei modelli rende i metodi tradizionali, orientati a sistemi addestrati secondo il paradigma del *traditional fine-tuning*, inadeguati per i modelli di grandi dimensioni addestrati secondo il paradigma del *prompting*. Tale distinzione — che segue la classificazione presentata da Zhao e colleghi [3] — si basa sulla modalità di adattamento dei sistemi al dominio specifico. Appartengono alla prima categoria i modelli linguistici che vengono inizialmente sottoposti a *pre-training*, su un ampio corpus di dati testuali non etichettati, per poi essere sottoposti a fine-tuning, su un insieme di dati etichettati provenienti da uno specifico dominio applicativo. Appartenenti alla seconda categoria, invece, sono i modelli che prevedono l'utilizzo di *prompt* — per esempio, frasi in linguaggio naturale contenenti spazi vuoti, che il modello deve completare — per consentire l'*in-context learning (ICL)* di tipo *zero-shot* (cioè senza esempi) o *few-shot* (cioè con pochi esempi), senza

richiedere dati di addestramento aggiuntivi. Come già detto, all'aumentare delle dimensioni dei modelli e dei dati d'addestramento, corrisponde una maggiore complessità nel comprendere i relativi processi decisionali. L'utilizzo di tecniche tradizionali, come quelle di feature attribution, richiederebbe considerevoli risorse computazionali, risultando inadeguate per sistemi con centinaia di miliardi di parametri, o anche più. In aggiunta, questi sistemi mostrano abilità emergenti che richiedono nuove prospettive per chiarirne i meccanismi sottostanti. Tali modelli si basano su abilità di ragionamento che rendono meno significative le spiegazioni locali o specifiche per determinati esempi e gli intricati funzionamenti interni e processi di ragionamento sono troppo complessi per essere catturati in maniera efficace semplificandoli con modelli surrogati [3].

Un approccio che si propone di compensare questa mancanza è quello della *mechanistic interpretability (MI)*, un campo di ricerca emergente, che parte dalla premessa che i sistemi di intelligenza artificiale, sufficientemente complessi, vanno studiati e spiegati tramite le stesse lenti usate per gli organismi biologici, piuttosto che trattarli come meri artefatti tecnologici. In tal senso, i sistemi di intelligenza artificiale vengono caratterizzati in termini della rispettiva organizzazione funzionale, ovvero l'insieme delle attività organizzate delle componenti rilevanti dal punto di vista funzionale. Tali componenti possono essere una qualsiasi unità o struttura della rete, che contribuisce al raggiungimento di una specifica funzione all'interno del sistema; senza limitarsi a entità prespecificate, come neuroni o *layer*, sono comprese anche strutture complesse, come circuiti o rappresentazioni distribuite. Si tratta, quindi, di comprendere quali proprietà del sistema supportino il relativo comportamento e come le proprie funzioni siano implementate dalla sinergia tra le componenti rilevanti. Ciò richiede l'applicazione di *discovery strategies* coordinate, mutuate dalle scienze della vita, come *pattern recognition*, *functional decomposition*, *localisation* e *systematic experimental manipulations*. In quanto tale, la ricerca in MI può essere dispendiosa a livello di risorse, sia

in termini di tempo che di lavoro, ma, di contro, si otterrà una comprensione, sul funzionamento di questi sistemi, più profonda e olistica, ovvero, ancora, in termini di come l'organizzazione funzionale dei sistemi implementa il rispettivo comportamento [2]. I risultati di una ricerca efficace in MI offrono una maggiore capacità predittiva e, permettendo interventi mirati, un maggiore controllo sul comportamento del sistema, consentendo ai ricercatori di potenziare specifiche capacità o correggere comportamenti indesiderati [2, 1].

Come già detto, le ANN acquisiscono la propria organizzazione funzionale attraverso procedure d'addestramento automatizzate. In questa maniera, le componenti del sistema assumono ruoli specifici, che i programmatori, solitamente, non sono in grado di anticipare, e vengono organizzate in modi che consentono loro di collaborare per produrre i comportamenti che osserviamo [2]. L'obiettivo dei ricercatori in MI è proprio quello di scoprire le componenti rilevanti e la loro organizzazione, spiegando il funzionamento interno di questi modelli [2, 1]. La MI si focalizza sul *reverse engineering* dei componenti del modello, per trasformarli in algoritmi comprensibili all'essere umano, in questo modo spiegando e prevedendo i comportamenti delle reti [4, 1, 9].
La ricerca in MI considera le ANN come grafi computazionali e i circuiti come sottografi con funzionalità distinte [4, 1, 2]. Un circuito, infatti, connette neuroni o feature attraverso diversi layer, formando un meccanismo coerente e interpretabile [1]. In quanto unità funzionali in cui i neuroni si auto-organizzano durante il processo d'addestramento, i circuiti fanno parte della struttura appresa dal modello [2]. Questi rivelano come i singoli neuroni contribuiscono a funzioni d'alto livello [1], dal momento che i comportamenti delle reti neurali sono implementati da algoritmi all'interno del grafo computazionale del modello [4].
Secondo la classificazione di Gantla [1], le tecniche di MI si dividono in osservazionali (*observational*) e interventistiche (*interventional*). Sebbene le prime — tra cui rientrano il *probing* e gli *sparse autoencoder (SAE)* — forniscano informazioni sulle attivazioni neuronali, esse potrebbero rivelare soltan-

to correlazioni piuttosto che relazioni causali. Le seconde affrontano questa limitazione, alterando le attivazioni neuronali per osservare i cambiamenti nell'output del modello. Ne fanno parte le tecniche di *ablation*, che esplorano relazioni causali rimuovendo o disattivando sistematicamente componenti del modello e osservando l'effetto sulle prestazioni del sistema nel compiere il *task*. Questo aiuta a identificare i componenti dei circuiti computazionali che svolgono compiti specifici, fornendo evidenza causale della loro esistenza. Tuttavia, la rimozione completa di componenti può introdurre effetti indesiderati, motivando l'uso di metodi più raffinati, come il *path patching* e il *knockout*. Il primo consente un'analisi più precisa del contributo dei componenti, attraverso interventi selettivi sui percorsi computazionali. Questa tecnica prevede l'esecuzione del modello su due input differenti e lo scambio degli output di componenti specifici, per analizzare gli effetti di sottoinsiemi differenti di archi, nel grafo computazionale dei modelli. Il secondo offre un approccio meno dirompente, sostituendo gli output dei componenti con valori neutri — le attivazioni medie provenienti da un dataset di riferimento — anziché rimuoverli completamente, cancellando così il relativo contenuto informativo. Questo preserva la stabilità del modello, pur consentendo l'analisi del ruolo di un componente [1].
Conmy e colleghi [4] hanno sistematizzato il flusso di lavoro comune seguito da molte ricerche sulla MI con approccio interventistico [1], che si è rivelato fruttuoso per individuare circuiti nei modelli, delineando gli elementi essenziali di questo processo. Essi identificano il flusso di lavoro, che porta eventualmente all'individuazione di un circuito, come articolato in tre fasi. I ricercatori:

1. **Selezionano comportamento, prompt e metrica.**
   Scelgono un comportamento dell'ANN da analizzare, curano una collezione di prompt che stimolino tale comportamento nel modello e selezionano una metrica per quantificare il grado di esecuzione del compito da parte del modello.
   Il più delle volte, optano per un comportamento chiaramente definito,

al fine di isolare l'algoritmo relativo a un compito specifico. Così facendo, il circuito sarà più semplice da interpretare, rispetto a un mix di circuiti corrispondenti a un comportamento vago.

2. **Dividono la rete neurale in un grafo di unità più piccole.**
   Per individuare circuiti relativi al comportamento di interesse, è necessario rappresentare gli elementi interni del modello come un grafo aciclico diretto (DAG) computazionale.
   Ciascuno studio definisce il livello di astrazione del grafo computazionale, in base al grado di dettaglio richiesto per le spiegazioni del comportamento del modello. Ad esempio, a un livello di astrazione più grossolano, i grafi computazionali possono rappresentare le interazioni tra le attention head e gli *MLP (Multi-Layer Perceptron)*. A un livello più granulare, invece, potrebbero includere le attivazioni separate di *query*, chiavi (*key*) e valori (*value*), le interazioni tra i singoli neuroni oppure prevedere un nodo per ciascuna posizione dei token.

3. **Eseguono il *patching* delle attivazioni del modello, per isolare il sottografo rilevante.**
   Una volta specificato il DAG computazionale, si può procedere alla ricerca degli archi che formano il circuito, per comprendere quali unità sono coinvolte nel comportamento.
   Si testa l'importanza degli archi utilizzando il patching ricorsivo delle attivazioni (*recursive activation patching*):

   (a) si sovrascrive il valore di attivazione di un nodo o di un arco con un'attivazione corrotta (*corrupted*),

   (b) si esegue un *forward pass* attraverso il modello,

   (c) si confrontano i valori di output del nuovo modello con quelli del modello originale, utilizzando la metrica precedentemente scelta.

   In genere, si inizia dal nodo di output, si determinano gli archi in ingresso importanti e poi si investigano tutti i nodi genitori (*parent*

node) attraverso questi archi, procedendo allo stesso modo.
L'obiettivo è quello di rimuovere dal modello quanti più componenti e connessioni superflue possibili.

Dopo aver isolato con successo un sottografo, si è individuato un circuito. Il ricercatore può quindi formulare e testare ipotesi sulle funzioni implementate da ciascun nodo all'interno del sottografo, ripetendo i tre passi precedenti con prompt o granularità leggermente differenti, finché non saranno soddisfatti delle spiegazioni dei componenti del circuito e l'algoritmo del modello non sarà compreso [4].

1.2 Framework Matematico

Elhage e colleghi [9], del gruppo di ricerca in MI di Anthropic, hanno creato un *framework* matematico per descrivere a livello teorico la struttura e il funzionamento dei modelli linguistici basati su Transformer, facendo un primo passo verso il reverse engineering dei calcoli eseguiti da tali sistemi. Il loro lavoro si è concentrato su modelli estremamente semplici — in particolare, Transformer con al massimo due layer, che contengono esclusivamente blocchi d'attenzione — con l'obiettivo di scoprire pattern algoritmici elementari o schemi ricorrenti che possano successivamente essere applicati a modelli di dimensioni e complessità maggiori. Concettualizzando il funzionamento dei Transformer in un modo nuovo, ma matematicamente equivalente, essi hanno successo nella comprensione di questi modelli di piccole dimensioni, acquisendo una comprensione significativa del loro funzionamento interno. Di particolare rilievo è l'individuazione di specifiche attention head, definite "*induction head*". Queste si sviluppano solamente in modelli con almeno due layer d'attenzione e possono spiegare l'ICL in questi sistemi di piccole dimensioni. Su questa linea, Olsson e colleghi [7], del medesimo gruppo di ricerca di Anthropic, hanno mostrato che sia il framework matematico per comprendere i Transformer, sia il concetto di induction head, continuano a essere rilevanti, almeno parzialmente, per modelli molto più grandi e realisti-

ci, pur rimanendo lontani da un reverse engineering completo di tali modelli [9].

La MI richiede di scomporre i modelli in componenti interpretabili dall'essere umano. Riconcettualizzando i Transformer, come già detto, secondo modalità equivalenti — seppur non convenzionali, perché computazionalmente inefficienti — Elhage e colleghi [9] individuano una rappresentazione che facilita il ragionamento sui modelli. Essi apportano una modifica sostanziale alla struttura dei Transformer, concentrandosi su una versione semplificata, di tipo "*attention-only*", ovvero privi di layer MLP. Inoltre, non considerano né i bias né la *layer normalization* — cambiamenti più superficiali[2] — allo scopo di favorire chiarezza e semplicità.
Il lavoro si concentra su modelli linguistici basati su Transformer di tipo *decoder-only*[3] autoregressivi[4], come GPT-2. Un Transformer di questo tipo (Fig. 1.1) inizia con un *token embedding*, seguito da una serie di blocchi ("*residual block*"), e termina con un *token unembedding*. Ogni blocco è costituito da un layer di attenzione, seguito da un layer MLP. Sia i layer di attenzione che quelli MLP "leggono" il proprio input dal *residual stream*, eseguendo una proiezione lineare, e successivamente "scrivono" il risultato nel residual stream, sommandovi una nuova proiezione lineare. Ogni layer di attenzione è composto da più teste (attention head), che operano in parallelo [9].

[2] Al netto di complicanze implementative, infatti, i classici modelli possono essere simulati da sistemi riformulati secondo questa semplificazione [9].

[3] Vaswani e colleghi, nel lavoro originale sui Transformer [10], utilizzavano una specifica architettura *encoder-decoder* per compiti di traduzione, ma molti modelli di linguaggio moderni hanno omesso tale struttura [9, 5].

[4] I Transformer di tipo decoder-only autoregressivi prevedono il token successivo basandosi solo sui token precedenti, grazie alla *masked self-attention* [10] che limita l'attenzione al contesto precedente, simulando il processo di generazione token-per-token (cioè senza accesso a informazioni future non ancora prodotte) [5].

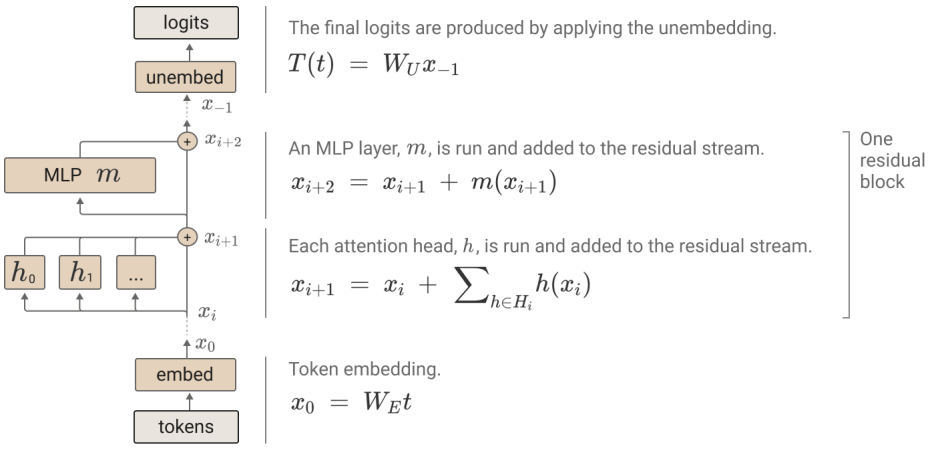


Figura 1.1: L'architettura d'alto livello di un Transformer di tipo decoder-only. Tratta da [9].

1.2.1 Residual Stream come Canale di Comunicazione

Una delle caratteristiche principali dell'architettura d'alto livello di un Transformer è che ogni layer, come già detto, aggiunge i propri risultati a quello che Elhage e colleghi [9] definiscono "residual stream" (Fig. 1.2). Questo è semplicemente la somma dell'output di tutti i layer precedenti e dell'embedding originale. Generalmente viene considerato come un canale di comunicazione, poiché non effettua alcuna elaborazione in sé e tutti i layer comunicano attraverso di esso. Il residual stream presenta una struttura profondamente lineare. Ogni layer esegue una trasformazione lineare arbitraria per "leggere" le informazioni dal residual stream, per poi eseguirne un'altra prima dell'addizione per "scrivere" il proprio output nel residual stream.
Questa struttura lineare e additiva del residual stream comporta numerose implicazioni importanti. Una conseguenza particolarmente utile è che si possono concepire dei "pesi virtuali" ("*virtual weights*") impliciti (Fig. 1.3), che collegano direttamente qualsiasi coppia di layer — anche quelli separati da molti altri layer — moltiplicando le loro interazioni attraverso il residual stream. Questi pesi virtuali sono il prodotto dei pesi di output di un layer

Figura 1.2: Il residual stream di un Transformer. Tratta da [9].

con i pesi di input[5] di un altro (ossia $W_I^q W_O^p$, dove p e q sono indici dei layer e $p < q$) e descrivono la misura in cui un layer successivo legge le informazioni scritte da un layer precedente [9].

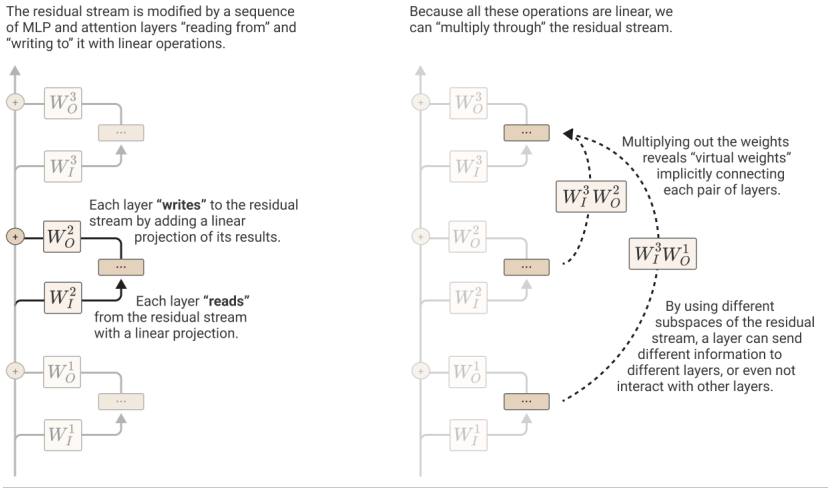


Figura 1.3: I "pesi virtuali" che collegano implicitamente ogni coppia di layer. Tratta da [9].

Il residual stream rappresenta uno spazio vettoriale ad alta dimensionalità (Fig. 1.4). Nei modelli più piccoli può contare centinaia di dimensioni, mentre in quelli più grandi può raggiungere le decine di migliaia. Questo significa

[5]Si noti che per i layer di attenzione esistono tre diversi tipi di pesi di input: W_Q, W_K e W_V. Per semplicità e generalità, Elhage e colleghi [9] considerano i layer come dotati semplicemente di pesi di input e output [9].

che i layer possono inviare informazioni diverse a layer diversi, archiviandole in sottospazi differenti. Ciò risulta particolarmente importante nel caso delle attention head, poiché ogni singola testa opera su sottospazi comparativamente ridotti (spesso 64 o 128 dimensioni) e può scrivere molto facilmente in sottospazi completamente disgiunti, evitanto l'interazione. Una volta aggiunta, l'informazione persiste in un sottospazio, a meno che un altro layer attivamente non la cancelli. Da questa prospettiva, le dimensioni del residual stream diventano qualcosa di simile a una "memoria" o una "larghezza di banda". Elhage e colleghi [9] riferiscono indizi secondo cui determinati neuroni MLP e attention head svolgono una sorta di ruolo di "gestione della memoria", cancellando le dimensioni del residual stream, impostate da altri layer, attraverso la lettura delle informazioni e la scrittura della versione negativa [9].

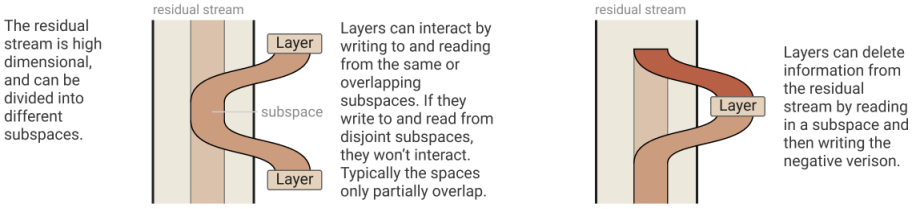


Figura 1.4: Il residual stream rappresenta uno spazio vettoriale ad alta dimensionalità, che può essere diviso in sottospazi differenti. Tratta da [9].

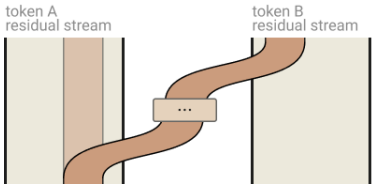
1.2.2 Attention Head Indipendenti e Additive

Elhage e colleghi [9] concepiscono i layer di attenzione dei Transformer come costituiti da diverse attention head $h \in H$ completamente indipendenti, che operano in parallelo e aggiungono ciascuna il proprio output al residual stream. Nel lavoro originale di Vaswani e colleghi sui Transformer [10], l'output di un layer d'attenzione veniva descritto attraverso la concatenazione dei vettori risultato $r^{h_1}, r^{h_2}, \ldots, r^{h_n}$ e la successiva moltiplicazione per una matrice di output W_O^H. Elhage e colleghi [9] suddividono W_O^H in blocchi di

uguale dimensione per ciascuna testa ($[W_O^{h_1}, W_O^{h_2}, \ldots, W_O^{h_n}]$). Si osserva che:

$$W_O^H \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \cdots \\ r^{h_n} \end{bmatrix} = \left[W_O^{h_1}, W_O^{h_2}, \ldots, W_O^{h_n}\right] \cdot \begin{bmatrix} r^{h_1} \\ r^{h_2} \\ \cdots \\ r^{h_n} \end{bmatrix} = \sum_i W_O^{h_i} r^{h_i}$$

rivelando che la formulazione originale risulta equivalente all'esecuzione indipendente delle teste, alla moltiplicazione di ciascuna per la propria matrice di output e alla loro aggiunta nel residual stream.
L'azione fondamentale delle attention head consiste nel trasferimento di informazioni. Esse leggono informazioni dal residual stream di un token e le scrivono nel residual stream di un altro token (Fig. 1.5). Di particolare importanza è il fatto che la scelta dei token da cui prelevare le informazioni è completamente separabile dall'informazione che viene effettivamente "letta", per essere trasferita, e dalle modalità con cui essa viene "scritta" nella destinazione. Per comprendere questo aspetto, essi descrivono il meccani-



Attention heads copy information from the residual stream of one token to the residual stream of another. They typically write to a different subspace than they read from.

Figura 1.5: Le attention head leggono informazioni dal residual stream di un token e le scrivono nel residual stream di un altro token. Tratta da [9].

smo di attenzione in modo non convenzionale. Dato un pattern d'attenzione A, il calcolo dell'output di un'attention head viene tipicamente descritto attraverso tre passaggi:

1. si calcola il *value vector* per ciascun token, a partire dal residual stream ($v_i = W_V x_i$);
2. si calcola il "*result vector*" mediante una combinazione lineare dei value vector, sulla base del pattern d'attenzione ($r_i = \sum_j A_{i,j} v_j$);

3. si calcola l'*output vector* della testa per ciascun token ($h(x)_i = W_O r_i$).

Ciascuno di questi passaggi può essere espresso come moltiplicazione matriciale, il che li rende combinabili in un'unica operazione. Considerando x come una matrice bidimensionale, composta da un vettore per ciascun token, questa viene moltiplicata su lati differenti. Le matrici W_V e W_O moltiplicano il lato corrispondente al "vettore per token", mentre la matrice A moltiplica il lato corrispondente alla "posizione". Utilizzando i prodotti tensoriali (denotati con il simbolo $\otimes$), possiamo descrivere il processo d'applicazione dell'attenzione come segue:

$$h(x) = \underbrace{(\mathrm{Id} \otimes W_O)}_{\substack{\text{Proietta i result vector per ciascun token} \\ (h(x)_i = W_O r_i)}} \cdot \underbrace{(A \otimes \mathrm{Id})}_{\substack{\text{Combina i value vector} \\ \textit{tra (across)} \text{ i token, per calcolare i result vector} \\ (r_i = \sum_j A_{i,j} v_j)}} \cdot \underbrace{(\mathrm{Id} \otimes W_V)}_{\substack{\text{Calcola il value vector per ciascun token} \\ (v_i = W_V x_i)}} \cdot \; x$$

Applicando la proprietà mista del prodotto tensoriale (*mixed product property*) e semplificando le identità, si ottiene:

$$h(x) = \underbrace{(A \otimes W_O W_V)}_{\substack{A \text{ combina tra i token, mentre} \\ W_O W_V \text{ agisce su ciascun vettore} \\ \text{indipendentemente.}}} \cdot \; x$$

Dunque, un'attention head applica due operazioni lineari, A e $W_O W_V$, che operano su dimensioni differenti e agiscono in modo indipendente. La matrice A governa da quali token e verso quali token viene trasferita l'informazione. Il prodotto $W_O W_V$ governa quale informazione viene letta dal token sorgente e come essa viene scritta nel token di destinazione — in altre parole, determina quale sottospazio del residual stream viene utilizzato dalla attention head per la lettura e la scrittura durante il trasferimento dell'informazione. In particolare, le attention head possono essere concepite come dotate di due computazioni sostanzialmente indipendenti: un circuito QK ("*query-key*"), che calcola il pattern d'attenzione, e un circuito OV ("*output-value*"), che determina in che modo ciascun token influenza l'output, quando viene selezionato dal meccanismo d'attenzione (Fig. 1.6).

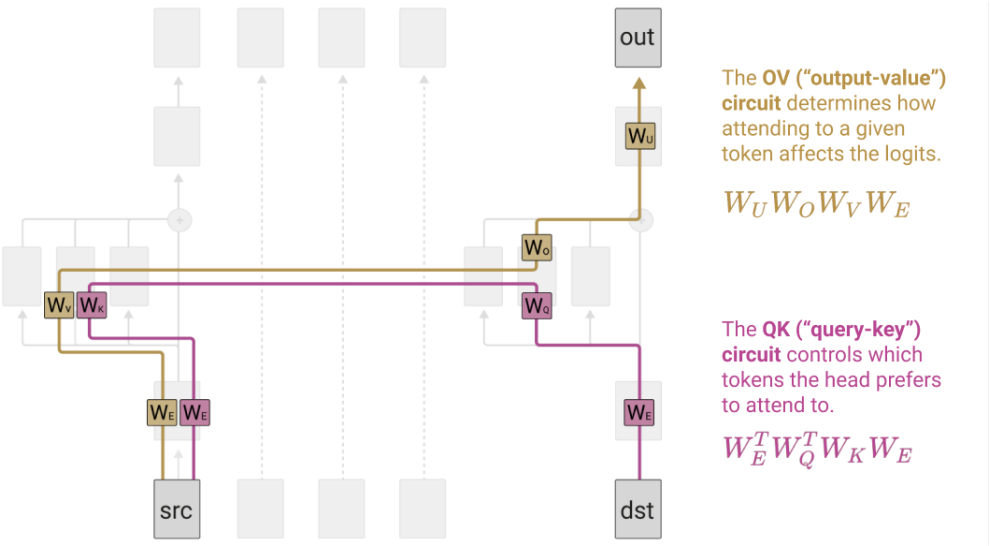


Figura 1.6: I circuiti QK (per l'attenzione) e OV (per l'output), due operazioni lineari separabili di ogni attention head. Tratta da [9].

Per quanto riguarda il pattern d'attenzione A, tipicamente, si calcolano le chiavi $k_i = W_K x_i$, si calcolano le query $q_i = W_Q x_i$ e successivamente si calcola il pattern d'attenzione a partire dai prodotti scalari di ciascun vettore chiave e query: $A = \text{softmax}(q^T k)$. Tuttavia, l'intera operazione può essere eseguita in un unico passaggio, senza fare riferimento a chiavi e query:

$$A = \text{softmax}(x^T W_Q^T W_K x)$$

Le matrici W_Q e W_K operano sempre congiuntamente e non sono mai indipendenti l'una dall'altra. Analogamente, anche W_O e W_V operano sempre in combinazione. Per questo motivo, Elhage e colleghi [9] definiscono variabili che rappresentano queste matrici combinate: $W_{OV} = W_O W_V$ e $W_{QK} = W_Q^T W_K$.
Riformulando le attention head nella forma presentata, essi hanno fatto emergere una struttura che in precedenza poteva risultare più difficile da osservare. In particolare, il fatto che i prodotti di attention head si comportano in modo molto simile alle attention head stesse. Per la proprietà distributiva, infatti,

si ha che:

$$(A^{h_2} \otimes W_{OV}^{h_2}) \cdot (A^{h_1} \otimes W_{OV}^{h_1}) = (A^{h_2} A^{h_1}) \otimes (W_{OV}^{h_2} W_{OV}^{h_1})$$

Il risultato di questo prodotto può essere interpretato come funzionalmente equivalente a un'attention head, a cui si riferiscono con il nome di "*virtual attention head*", con un pattern d'attenzione dato dalla composizione delle due teste $A^{h_2}A^{h_1}$ e una matrice *output-value* $W_{OV}^{h_2}W_{OV}^{h_1}$ [9].

1.2.3 Risultati

Compiendo il reverse engineering di questi semplici modelli di tipo attention-only, in particolare studiando i relativi circuiti QK e OV, Elhage e colleghi [9] ottengono i seguenti risultati nella comprensione di tali sistemi:

0-layer I Transformer a zero layer modellano le statistiche dei *bigram*.

1-layer I Transformer, di tipo attention-only, a un singolo layer costituiscono un insieme di modelli di bigram e "*skip-trigram*" (sequenze della forma "$A \dots BC$"), implementando una forma molto semplice di ICL.

2-layer I Transformer, di tipo attention-only, a due layer possono implementare algoritmi considerevolmente più complessi mediante la composizione di attention head, per creare le cosiddette "induction head", un algoritmo di ICL molto generale, costituendo un punto di transizione importante, che risulterà rilevante per i modelli di dimensioni maggiori [9].

Capitolo 2

TransformerLens

Elhage e colleghi, in corrispondenza dell'articolo sul framework matematico [9], ne hanno rilasciato un secondo [8], per fornire un contesto sull'infrastruttura utlizzata durante la ricerca [8]. In questo articolo [8], essi descrivono una libreria — senza rilasciarne l'implementazione — chiamata *Garçon*, uno strumento per l'analisi degli elementi interni dei modelli, in particolare quelli di grandi dimensioni, che facilita l'accesso alle attivazioni e ai parametri [9]. Esso è uno dei componenti principali dell'infrastruttura utilizzata per la ricerca in MI presso Anthropic. Questo approccio implica l'analisi dei modelli in modo ampio e flessibile. Tale strumento permette di esaminare singole attivazioni all'interno di layer arbitrari del modello, condurre esperimenti in cui si modificano o rimuovono componenti individuali e, più in generale, accedere e lavorare con gli elementi interni del modello, non solo con i suoi input e output.
Per i modelli di piccole dimensioni, il lavoro di MI può essere svolto utilizzando *notebook Colab* o *Jupyter*, o strumenti simili, impiegando funzionalità come gli *hook* di *PyTorch*, per accedere alle attivazioni intermedie. Tuttavia, quando il modello scala oltre un singolo nodo computazionale[1] — come un

[1]Gli LLM moderni, infatti, possono raggiungere dimensioni tali da rendere impraticabile l'esecuzione efficiente su una singola GPU o su un singolo computer dotato di più GPU [8].

computer o un server — non esiste un modo evidente per trasferire questo flusso di lavoro. Garçon è stato progettato per risolvere questo problema e consentire il lavoro di MI su modelli di dimensioni arbitrarie.
Garçon è costituito da un'interfaccia basata sugli hook, che consente di accedere e modificare lo stato interno durante il forward pass. I modelli espongono un insieme di "*probe point*", ognuno dei quali denota una posizione specifica all'interno del modello, in cui un tensore può essere acceduto o modificato. È possibile collegarsi a qualsiasi di questi punti ed eseguire un forward pass — oppure un *backward pass*, calcolando i gradienti rispetto a una funzione di loss fornita dall'utente — mantenendo il collegamento attivo. L'interfaccia prevede la possibilità di fornire "*probe function*", che accettano due argomenti. Il primo è un "contesto di salvataggio" ("*save context*"), utilizzabile per memorizzare attivazioni o dati per un utilizzo successivo, mentre il secondo è il tensore rappresentato in quel particolare punto del modello. Le probe function possono restituire un tensore aggiornato, il quale sostituirà il tensore ispezionato nel calcolo.
Garçon è stato creato dando priorità alla flessibilità piuttosto che alle prestazioni. Questo fa sì che si possa lavorare direttamente su un qualsiasi modello, indipendentemente dalla scala o dalla specifica architettura [8].

Neel Nanda, ex membro del gruppo di ricerca in MI di Anthropic, coautore dei relativi articoli a cui si è fatto riferimento [9, 8, 7], ha guidato lo sviluppo di una libreria chiamata *TransformerLens* [6], che consente l'analisi di modelli linguistici basati su Transformer, secondo un approccio di MI, esponendo le relative attivazioni interne.
È un lavoro indipendente da Anthropic, dal momento che la libreria è stata creata dopo che Nanda ha lasciato quella compagnia per intraprendere ricerca indipendente, in cui ha riscontrato una profonda insoddisfazione rispetto allo stato degli strumenti open source disponibili per analizzare gli aspetti interni dei modelli e applicare tecniche di reverse engineering volte a comprenderne il funzionamento — problema che questa libreria cerca di risolvere. Tuttavia, il

collegamento con il lavoro di Elhage e colleghi [9] è affermato da Nanda stesso — che si ricorda essere coautore dell'articolo di riferimento [9] — riferendo che le caratteristiche fondamentali di TransformerLens sono state fortemente ispirate dall'interfaccia di Garçon[2].
La libreria è tuttora[3] mantenuta[4] e fornisce una buona documentazione[5].

2.1 Caratteristiche Principali

TransformerLens[6] è una libreria *Python open source*, che consente di utilizzare tecniche di MI su modelli linguistici basati su Transformer[7]. Il principio cardine che ha guidato la progettazione è stato quello di consentire un'analisi di tipo esplorativo[8]. La libreria permette di caricare oltre cinquanta diversi modelli linguistici open source — da semplici modelli di tipo attention-only, studiati da Elhage e colleghi [9] (Sez. 1.2), fino a modelli come GPT-2 — ed espone le attivazioni interne del modello all'utente. È possibile catturare e conservare qualsiasi attivazione interna del modello e aggiungere funzioni per modificare, rimuovere o sostituire tali attivazioni durante l'esecuzione del modello[9].

[2]Per riferimenti, si consulti la pagina *GitHub* della libreria (URL: `https://github.com/TransformerLensOrg/TransformerLens`).

[3]L'ultimo *commit* rilevato, in data 5 dicembre 2025, sulla pagina GitHub della libreria, risale al 9 luglio 2025.

[4]Anche se non più da Nanda, ma da Bryce Meyer. Per riferimenti, si consulti la pagina GitHub della libreria.

[5]Con il termine "documentazione" si farà riferimento alle risorse presenti nel seguente URL: `https://transformerlensorg.github.io/TransformerLens/`.

[6]Per evitare una possibile confusione, si precisa il fatto che TransformerLens era precedentemente denominata "EasyTransformer". Per riferimenti, si consulti la documentazione.

[7]Da notare che in tale categoria rientra la tipologia di modelli studiati da Elhage e colleghi [9] (Sez. 1.2).

[8]Per riferimenti, si consulti la documentazione.

[9]Per riferimenti, si consulti la pagina GitHub della libreria.

TransformerLens è implementata come pacchetto *PyPI*[10] installabile via `pip install transformer_lens`[11]. L'uso di base della libreria è il seguente:

```
1 import transformer_lens
2
3 # Load a model (eg GPT-2 Small)
4 model = transformer_lens.HookedTransformer.from_pretrained("
    gpt2-small")
5
6 # Run the model and get logits and activations
7 logits, activations = model.run_with_cache("Hello World")
```

Codice 2.1: L'uso di base di TransformerLens. Tratto dalla pagina GitHub della libreria.

Dopo aver importato la libreria, si può caricare un modello (Cod. 2.1, riga 4), in questo caso *GPT-2 Small*, ed eseguirlo restituendo *logit* e attivazioni (Cod. 2.1, riga 7)[12]. Da queste prime linee di codice, emergono già le due classi principali della libreria: `HookedTransformer`[13] e `ActivationCache`[14]. La prima costituisce l'elemento cardine di TransformerLens. Nelle implementazioni standard dei modelli in PyTorch, risulta relativamente semplice estrarre i pesi del modello, mentre l'estrazione delle attivazioni presenta maggiori complessità. TransformerLens si propone di semplificare questa operazione mediante l'inserimento di hook su ogni attivazione rilevante all'interno del modello, consentendo l'ispezione e la modifica delle attivazioni nei singoli componenti, come le attention head e i layer MLP. `HookedTransformer` implementa un Transformer completo, inserendo un `HookPoint` su ogni attiva-

[10] URL: `https://pypi.org/project/transformer-lens/`.

[11] Per riferimenti, si consulti la pagina GitHub della libreria.

[12] Per riferimenti, si consulti la pagina GitHub della libreria.

[13] Per riferimenti e maggiori dettagli, si consulti la relativa pagina nella documentazione (URL: `https://transformerlensorg.github.io/TransformerLens/generated/code/transformer_lens.HookedTransformer.html`).

[14] Per riferimenti e maggiori dettagli, si consulti la relativa pagina nella documentazione (URL: `https://transformerlensorg.github.io/TransformerLens/generated/code/transformer_lens.ActivationCache.html`).

zione di interesse — questo adattamento trae ispirazione, dichiaratamente[15], dall'impiego di probe point da parte di Garçon. Tipicamente, si procede all'inizializzazione di un modello precaricato in TransformerLens mediante il metodo `from_pretrained()`, sebbene sia possibile creare istanze con pesi inizializzati casualmente attraverso il metodo `__init__()`. Una volta completata l'inizializzazione del modello, un passaggio comune consiste nel verificare che esso sia in grado di eseguire il compito oggetto dell'analisi. Tale verifica può essere effettuata utilizzando la funzione `test_prompt()`[16].
`ActivationCache`, a sua volta, costituisce il fulcro operativo di TransformerLens. Si tratta di un *wrapper* che memorizza tutte le attivazioni rilevanti provenienti da un forward pass del modello e fornisce una serie di funzioni ausiliarie (alcune delle quali saranno presentate nella Sez. 2.3) per analizzarle. Il metodo abituale per accedervi consiste nell'eseguire il modello utilizzando `run_with_cache()`[17].

2.2 Corrispondenza con il Framework Matematico

Seguendo l'approccio di Elhage e colleghi [9] (Sez. 1.2), Nanda — che si ricorda essere coautore dell'articolo di riferimento [9] — implementa e analizza la struttura interna di `HookedTransformer` secondo la riconcettualizzazione del funzionamento dei Transformer presentata nella Sez. 1.2, come riferito nei notebook in documentazione[18].

[15]Per riferimenti e maggiori dettagli, si consulti il notebook Main Demo nella documentazione.

[16]Per riferimenti e maggiori dettagli, si consulti la pagina relativa alla classe `HookedTransformer` nella documentazione.

[17]Per riferimenti e maggiori dettagli, si consulti la pagina relativa alla classe `ActivationCache` nella documentazione.

[18]Per riferimenti e maggiori dettagli, si consultino i notebook *Clean Transformer Demo* (URL: `https://colab.research.google.com/github/TransformerLensOrg/TransformerLens/blob/clean-transformer-demo/Clean_Transformer_Demo.ipynb`), *Exploratory Analysis Demo* (URL: `https://colab.research.google.com/github/`

`HookedTransformer` costituisce una variante, computazionalmente identica, dell'architettura di GPT-2, con alcune modifiche implementative. Le più significative riguardano la struttura interna delle attention head (confronta Sez. 1.2.2). Le matrici di pesi `W_K` (chiavi), `W_Q` (query) e `W_V` (valori) sono tre matrici separate, anziché un'unica grande matrice concatenata e, assieme a `W_O` (output) e alle attivazioni, presentano assi separati per `head_index`, che indica l'indice della testa, e `d_head`, che indica la sua dimensione interna, anziché appiattiti in un unico grande asse[19]. Durante l'analisi di un

Iperparametro	**Descrizione**
`n_layers`	Il numero di blocchi presenti nel modello (ognuno contiene un layer d'attenzione e un layer MLP).
`n_heads`	Il numero di attention head per ogni layer d'attenzione.
`d_model`	La dimensione del residual stream.
`d_head`	La dimensione interna di un'attention head.
`d_mlp`	La dimensione interna dei layer MLP.
`d_vocab`	Il numero di token presenti nel vocabolario.
`n_ctx`	Il numero massimo di token consentito in un prompt di input.

modello, ciascun layer d'attenzione può essere riformulato nel contributo delle proprie teste, suddividendo la matrice dei pesi di output (W_O^H) in una matrice di output per ciascuna testa ($W_O^{h_i}$, in questo caso indicata come `model.blocks[k].attn.W_O`, di forma `[head_index, d_head, d_model]`) e sommandole. Inoltre, ciascuna di queste può essere indagata a livello del

`TransformerLensOrg/TransformerLens/blob/main/demos/Exploratory_Analysis_Demo.ipynb`) e *Main Demo* (URL: `https://colab.research.google.com/github/TransformerLensOrg/TransformerLens/blob/main/demos/Main_Demo.ipynb`) nella documentazione.

[19]Per riferimenti e maggiori dettagli, si consulti il notebook Main Demo nella documentazione.

singolo circuito QK o OV, applicando il patching in modo mirato al solo pattern d'attenzione oppure ai soli value vector, distinguendo così quale di queste due operazioni sia rilevante[20].

Il residual stream viene rappresentato come una struttura lineare e additiva, costituita dalla somma degli output di ciascun layer e dell'embedding originale (Sez. 1.2.1). Tale struttura si evince dal codice presentato nel notebook *Clean Transformer Demo*[21], in particolare dall'implementazione delle classi `DemoTransformer` (Cod. 2.2), che rappresenta il Transformer completo, e `TransformerBlock` (Cod. 2.3), che rappresenta un singolo blocco del Transformer. Si può notare infatti come il residual stream sia costituito dalla somma dell'embedding (Cod. 2.2, riga 15) e dell'output di ciascun blocco (Cod. 2.2, riga 17) — ognuno costituito dagli output dei relativi due layer, quello d'attenzione (Cod. 2.3, riga 15) e quello MLP (Cod. 2.3, riga 19)[22].

```
1  class DemoTransformer(nn.Module):
2      def __init__(self, cfg):
3          super().__init__()
4          self.cfg = cfg
5          self.embed = Embed(cfg)
6          self.pos_embed = PosEmbed(cfg)
7          self.blocks = nn.ModuleList([TransformerBlock(cfg)
                                        for _ in range(cfg.n_layers)])
8          self.ln_final = LayerNorm(cfg)
9          self.unembed = Unembed(cfg)
10
11     def forward(self, tokens):
12         # tokens [batch, position]
```

[20]Per riferimenti e maggiori dettagli, si consulti il notebook Exploratory Analysis Demo nella documentazione.

[21]Per brevità e semplicità, si farà riferimento al codice di TransformerLens come viene presentato nel notebook Clean Transformer Demo. Nanda stesso, nella documentazione, afferma che tale notebook costituisce un'implementazione pulita e minimale di GPT-2, con la stessa architettura interna e gli stessi nomi delle attivazioni di `HookedTransformer`, ma significativamente più chiara e meglio documentata.

[22]Per riferimenti e maggiori dettagli, si consulti il notebook Clean Transformer Demo nella documentazione.

```
        embed = self.embed(tokens)
        pos_embed = self.pos_embed(tokens)
        residual = embed + pos_embed
        for block in self.blocks:
            residual = block(residual)
        normalized_resid_final = self.ln_final(residual)
        logits = self.unembed(normalized_resid_final)
        # logits have shape [batch, position, logits]
        return logits
```

Codice 2.2: Implementazione della classe `DemoTransformer`, rappresentante il Transformer completo. Tratta dal notebook Clean Transformer Demo, nella documentazione di TransformerLens.

```
class TransformerBlock(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.cfg = cfg

        self.ln1 = LayerNorm(cfg)
        self.attn = Attention(cfg)
        self.ln2 = LayerNorm(cfg)
        self.mlp = MLP(cfg)

    def forward(self, resid_pre):
        # resid_pre [batch, position, d_model]
        normalized_resid_pre = self.ln1(resid_pre)
        attn_out = self.attn(normalized_resid_pre)
        resid_mid = resid_pre + attn_out

        normalized_resid_mid = self.ln2(resid_mid)
        mlp_out = self.mlp(normalized_resid_mid)
        resid_post = resid_mid + mlp_out
        return resid_post
```

Codice 2.3: Implementazione della classe `TransformerBlock`, rappresentante un singolo blocco del Transformer. Tratta dal notebook Clean Transformer Demo, nella documentazione di TransformerLens.

2.3 Uso

Questo lavoro si concentra sull'utilizzo di TransformerLens per l'analisi in ICL di modelli linguistici — già addestrati, secondo il paradigma del prompting [3] — basati su Transformer di tipo decoder-only autoregressivi, secondo il flusso di lavoro sistematizzato da Conmy e colleghi [4] (Sez. 1.1.1).
Quando si indaga un particolare comportamento di un modello, un primo passo molto comune consiste nel cercare di comprendere quali componenti del modello siano maggiormente responsabili di tale comportamento. Ad esempio, se si sta analizzando il prompt:

"Why did the chicken cross the" $\rightarrow$ *" road"*

potrebbe essere utile comprendere se esiste uno specifico *sublayer* (layer MLP o d'attenzione) responsabile della predizione " road" da parte del modello. Dopo aver caricato il modello ed eseguito l'inferenza sul prompt di interesse (in questo caso "Why did the chicken cross the"), si procede alla decomposizione del residual stream utilizzando il metodo `decompose_resid()`, della classe `ActivationCache`. Successivamente, per identificare quale componente contribuisce maggiormente alla predizione del token desiderato (ad esempio " road"), si calcola l'attribuzione dei logit attraverso il metodo `logit_attrs()`, della medesima classe `ActivationCache`. Questo calcolo genera un tensore che rappresenta il contributo di ciascun layer alla predizione finale. L'analisi di questo tensore, attraverso operazioni come `argmax()`, permette di individuare il componente specifico che esercita l'influenza più significativa sul risultato del modello.
È inoltre possibile analizzare i dati con maggiore granularità, utilizzando `get_full_resid_decomposition()`, della classe `ActivationCache`, per ottenere il residual stream suddiviso per singoli componenti (neuroni MLP e singole attention head). In tal caso, il metodo di utilizzo di `logit_attrs()` rimane invariato. Analogamente, potrebbe essere utile determinare se il modello incontra difficoltà nel costruire tali output linguistici fino agli ultimi

layer, oppure se il compito risulta banale e i primi layer sono sufficienti[23].

[23]Per riferimenti e maggiori dettagli, si consulti la pagina relativa alla classe `ActivationCache` nella documentazione.

Conclusioni

L'obiettivo di questo lavoro, come illustrato nell'Introduzione, era quello di individuare uno strumento per interpretare modelli linguistici basati su Transformer di tipo decoder-only autoregressivi.
Ho individuato la libreria TransformerLens [6] (Cap. 2), che consente l'utilizzo di tecniche interventistiche [1] su modelli linguistici basati su Transformer di grandi dimensioni, addestrati secondo il paradigma del prompting [3], indagando l'ICL secondo il flusso di lavoro presentato da Conmy e colleghi [4] (Sez. 1.1.1).
TransformerLens si basa su una riconcettualizzazione del funzionamento dei Transformer presentata da Elhage e colleghi [9], di cui ho verificato la corrispondenza (Sez. 1.2). Il vantaggio derivante da tale allineamento è duplice. Da una parte, sostiene la riconcettualizzazione dei Transformer implementata nella libreria. Dall'altra, fa sì che si possa utilizzare TransformerLens per analizzare un modello secondo l'approccio di Elhage e colleghi [9], ad esempio indagando i circuiti QK e OV all'interno delle attention head (Sez. 1.2.2).

Limiti e Sviluppi Futuri

Il limite principale di questo lavoro è la mancata validazione di TransformerLens su un caso di studio, con un approccio sistematico e comprensivo, che deve considerarsi il primo obiettivo per un ulteriore sviluppo di questa ricerca. In particolare, risulta necessaria una validazione su casi di studio in letteratura, ad esempio quelli che sono stati oggetto del lavoro di Conmy e colleghi [4], oltre a un'indagine approfondita e complessiva su un ulteriore

caso di studio innovativo.

Per concludere, un secondo limite di questo lavoro consiste nell'aver fatto riferimento, nel verificare la corrispondenza di TransformerLens con il framework matematico (Sez. 2.2), al codice presentato da Nanda nel notebook Clean Transformer Demo — dal momento che lui stesso afferma che in tale notebook viene presentata la medesima architettura interna e gli stessi nomi delle attivazioni di `HookedTransformer` (vedi nota 21) — invece che riferirsi al codice effettivamente implementato nella libreria.

Bibliografia

[1] Gantla, "Exploring Mechanistic Interpretability in Large Language Models: Challenges, Approaches, and Insights", 2025 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI), 2025.

[2] Kästner e Crook, "Explaining AI Through Mechanistic Interpretability", European Journal for Philosophy of Science, 2024.

[3] Zhao et al., "Explainability for Large Language Models: A Survey", Association for Computing Machinery, 2024.

[4] Conmy et al., "Towards Automated Circuit Discovery for Mechanistic Interpretability", Advances in Neural Information Processing Systems, 2023.

[5] Zhang et al., "Dive into Deep Learning", Cambridge University Press, 2023.

[6] Nanda e Bloom, "TransformerLens", 2022. URL: `https://github.com/TransformerLensOrg/TransformerLens`

[7] Olsson et al., "In-context Learning and Induction Heads", Transformer Circuits Thread, 2022.

[8] Elhage et al., "Garcon", Transformer Circuits Thread, 2021.

[9] Elhage et al., "A Mathematical Framework for Transformer Circuits", Transformer Circuits Thread, 2021.

[10] Vaswani et al., “Attention is All you Need”, Advances in Neural Information Processing Systems, 2017.