# Transformers

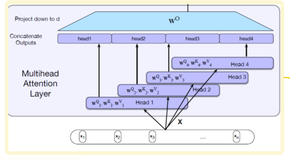
## 1. Introduzione

- Il principio base su cui si basa il funzionamento dei transformers è il Meccanismo di **Self-Attention**
  - Questo fa si che ogni token tenga conto degli altri token del contesto
  - Grazie a ciò
    - I Transformer operano completamente in parallelo
      - A differenza delle **Reti Ricorrenti (RNN)** dove l'input è processato sequenzialmente
    - La self-attention permette a ogni token di pesare dinamicamnete tutti gli altri nella sequenza
      - Questo ci permette di avere una **context-size** molto ampia
        - Detta anche finestra di contesto
        - Rappresenta il numero massimo di token che il modello può considerare in input
          - Ex. GPT-2 max 1024 token
          - Ex. GPT-4 max 32000 token
        - Questo parametro influenza direttamente
          - La lunghezza delle sequenze gestibili
          - La capacità del modello di mantenere coerenza a lungo termine
- **Struttura dei Transformer Blocks**
  - Ogni Transformer Block contiene due componenti principali
    - **Self-Attention Layer**
      - Consente a ogni token in input di produrre un output dipendente da tutti gli altri token della sequenza
    - **Feedforward Layer**
    - Entrambi sono seguiti da residual connection e da un layer di normalizzazione
  - I blocchi più moderni presentano la **multi-head attention**

## 2. Funzionamento del blocco di self-attention

- Dato un input di n token
  - La self-attention produce n output
    - Uno per ogni token
- Per ogni token **i** si calcola uno score che indica quanto è importante ogni altro token **j** rispetto a **i**
- **Query, key, value**
  - Ogni token xi viene proiettato in tre spazi distinti
    - $q_i = x_i W^Q$
      - **Query**
      - Rappresenta l'elemento che sto valutando
    - $k_j = x_j W^K$
      - **Key**
      - Rappresenta i valori che servono per valutare la query (frecce in entrata)
    - $v_j = x_j W^V$
      - **Value**
      - Serve per l'output
  - Il peso di attention che il token **i** assegna al token **j** risulta
    - $\text{score}_{ij} = \frac{q_i \cdot k_j^\top}{\sqrt{d_k}}$
      - dove dk è la dimensione dei vettori key
  - Dopo aver calcolato tutti gli score per il token i, si applica una softmax per ottenere pesi normalizzati
  - L'output per il token i è infine la combinazione pesata dei valori
    - $\text{output}_i = \sum_j \text{softmax}(\text{score}_{ij}) \cdot v_j$

## 3. Multi-head Attention Layer




- La Multi-Head Attention è un'estensione del meccanismo di Self-Attention
  - Consente al modello di apprendere più rappresentazioni dell'input da differenti spazi di attenzione
- Ogni testa di attenzione (head) lavora con una proiezione diversa dell'input originale
- Sia
  - $X \in \mathbb{R}^{n \times d_{model}}$
    - L'input alla Multi-Head Attention
    - Numero di token
      - n
    - Dimensione del modello (512 o 768)
      - dmodel
- Consideriamo le dimensioni delle matrici coinvolte
  - $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{model} \times d_k}$
    - Per ogni head **i** vengono apprese tre matrici di pesi
    - $d_k = d_{model}/h$
      - Tipicamente
      - h numero di head
      - Dimensione per ciascuna head
        - dk
        - dk = 64
        - Ex. se dmodel = 512, h = 8
  - con $Q_i, K_i, V_i \in \mathbb{R}^{n \times d_k}$
    - $Q_i = XW_i^Q \quad K_i = XW_i^K \quad V_i = XW_i^V$
      - Ogni head calcola
  - $\text{head}_i = \text{Attention}(Q_i, K_i, V_i) \in \mathbb{R}^{n \times d_k}$
    - Ogni head calcola la self-attention indipendentemente
  - $\text{Concat}(\text{head}_1, \ldots, \text{head}_h) \in \mathbb{R}^{n \times d_{model}}$
    - I risultati di tutte le h head vengono concatenati lungo la dimensione dei feature
  - $\text{MultiHead}(X) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$
    - $W^O \in \mathbb{R}^{d_{model} \times d_{model}}$
      - Infine, viene applicata una proiezione finale tramite
- Questa struttura consente di apprendere diverse relazioni tra i token
  - Ogni head può specializzarsi su un tipo di pattern
    - Ex.
      - Posizione
      - Sintassi
      - Semantica

## 4. Codifica Posizionale dei Token

- Il Transformer non ha un'idea implicita dell'ordine delle parole
  - Ha bisogno di un modo per capire dove si trovano i token nella sequenza
- Per questo, si aggiunge ad ogni token un **embedding posizionale**
  - Vettore che rappresenta la sua posizione nella frase
- Il risultato finale è dato dalla somma
  - $\text{input}_i = \text{Embedding}_{\text{token}} + \text{Embedding}_{\text{posizione}}$
- Questi embedding di posizione possono essere appresi dal modello durante il training
- **Limite**
  - Questi metodi funzionano bene per parole vicine
    - Non sempre permettono al modello di cogliere bene la relazione tra parole molto distanti nella sequenza
- **Alternativa**
  - Alcuni modelli usano le **posizioni relative** invece di quelle assolute
    - Invece di dire
      - Questo token è in posizione 7
      - Si rappresenta
        - Questo token è 2 posizioni dopo l'altro
    - Questo aiuta a modellare meglio le dipendenze tra parole anche lontane