# Bert Transformer

Una breve introduzione

# Introduzione

Prima dei transformer i metodi utilizzati per risolvere i problemi di NLP erano basati su RNN, che tuttavia possono presentare limitazioni:

- Non gestiscono bene grandi quantità di testo, poiché si dimenticano quelle più vecchie
- Sono lente da allenare, a causa della loro natura sequenziale

# Cosa è un transformer?

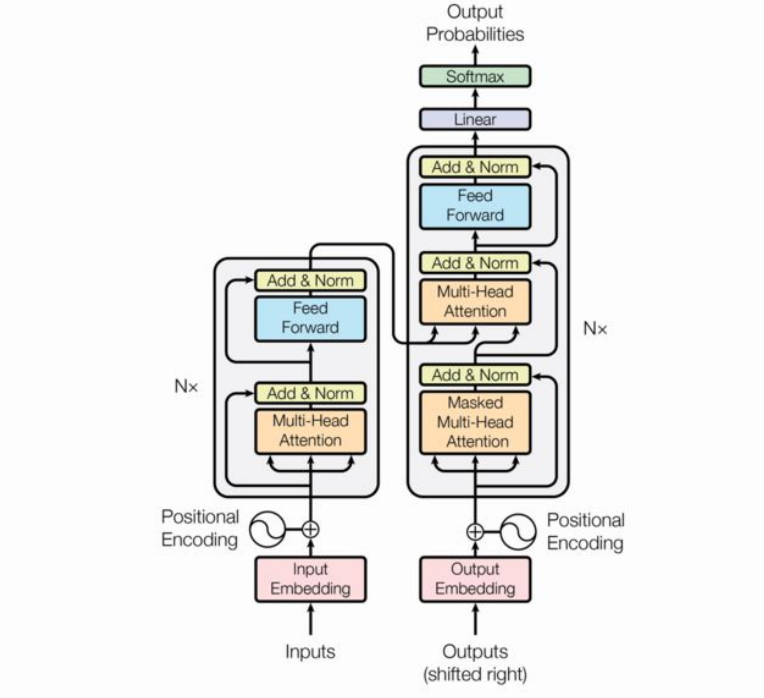
I transformers, come appena detto, sono un tipo di architettura specificamente sviluppate per poter essere allenate in parallelo, permettendo quindi  l'utilizzo di un dataset di grandissime dimensioni durante la fase di training, si parla di 40+ Terabyte di testo.

Questo, in contrapposizione alle RNN, ha permesso di allenare modelli come BERT che hanno centinaia di milioni di parametri.

# Come Funziona Un Transformer

Il funzionamento di un transformer si basa su 3 concetti chiave che lo differenziano dalle RNN:

- Positional Encoding
- Attention
- Self-Attention



# Positional encoding

Il positional encoding Permette di risolvere il problema della sequenzialità delle RNN accorpando ad ogni parola un indice posizionale.

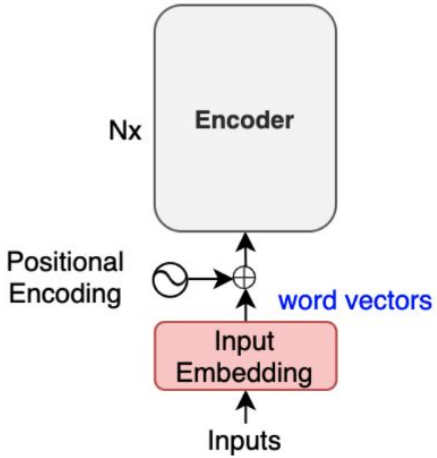
Questo indice verrà utilizzato durante la fase di training per tenere conto dell'ordine in cui si presentano le parole, e allo stesso tempo permettendo di parallelizzare l'allenamento.

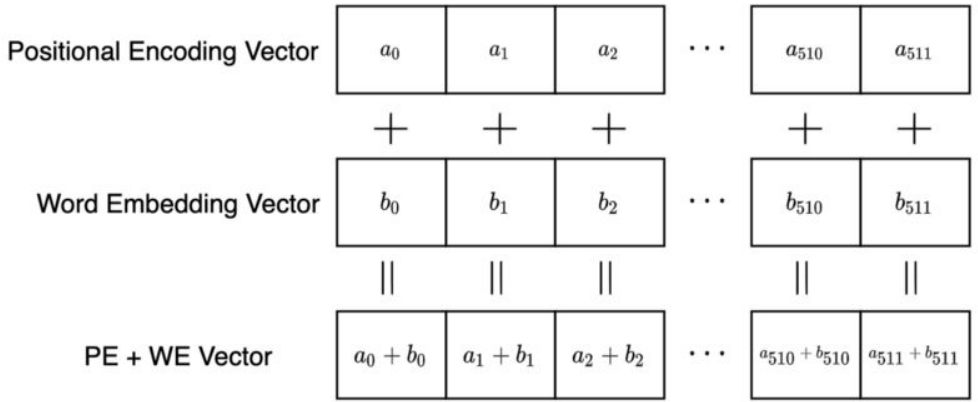
"***Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence.***"

Estratto dal paper "Attention is all you need", che ha introdotto per primo i Transformer.

# Positional encoding - Nel dettaglio

Nella pratica il positional encoding è rappresentato da un vettore di interi che andrà a sommarsi con la rappresentazione vettorizzata della parola, può essere visualizzata come una traslazione nello spazio multidimensionale.



# Positional encoding - Perché si somma?

Ci potrebbe sorgere la domanda:

*“Non sarebbe meglio concatenare i due vettori?”*


oppure

*“Come può il modello distinguere il word embedding dal positional encoding?”*

La risposta è che per ridurre i costi in memoria, e per evitare di aggiungere altri iperparametri, la somma risulta essere il metodo più efficace, dato che il modello riesce ad interpretare comunque bene il nuovo word embedding.

# Positional encoding - Come si calcola

I vari indici del **Positional Encoding Vector** si ottengono con la seguente funzione:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Che sostanzialmente applica il coseno agli elementi di indice dispari, e il seno a quelli pari, assegnando ad ogni indice un valore che varia tra [-1,1], che all'aumentare dell'indice tenderà ad oscillare tra [0,1]

# Positional encoding - Esempio

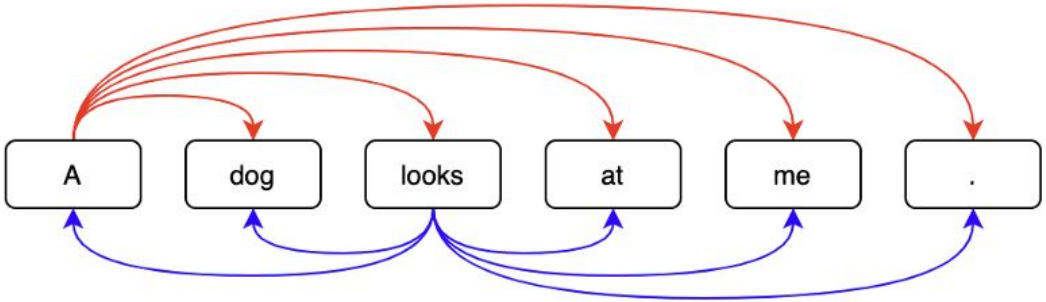
Di seguito un esempio di **Positional Encoding**:

$$PE(0) = \left( \sin\left(\frac{0}{10000^{\frac{0}{512}}}\right), \cos\left(\frac{0}{10000^{\frac{0}{512}}}\right), \sin\left(\frac{0}{10000^{\frac{2}{512}}}\right), \cos\left(\frac{0}{10000^{\frac{2}{512}}}\right), \ldots, \sin\left(\frac{0}{10000^{\frac{510}{512}}}\right), \cos\left(\frac{0}{10000^{\frac{510}{512}}}\right) \right)$$
$$= (\sin(0), \quad \cos(0), \quad \sin(0), \quad \cos(0), \quad \ldots, \quad \sin(0), \quad \cos(0))$$
$$= (0, \quad 1, \quad 0, \quad 1, \quad \ldots, \quad 0, \quad 1)$$

$$PE(1) = \left( \sin\left(\frac{1}{10000^{\frac{0}{512}}}\right), \cos\left(\frac{1}{10000^{\frac{0}{512}}}\right), \sin\left(\frac{1}{10000^{\frac{2}{512}}}\right), \cos\left(\frac{1}{10000^{\frac{2}{512}}}\right), \ldots, \sin\left(\frac{1}{10000^{\frac{510}{512}}}\right), \cos\left(\frac{1}{10000^{\frac{510}{512}}}\right) \right)$$
$$= (0.8414, \quad 0.5403, \quad 0.8218, \quad 0.5696, \quad \ldots, \quad 0.0001, \quad 0.9999)$$

# Positional encoding - conclusione

Per concludere è utile capire come fa il modello a comprendere la posizione relativa dell'input.

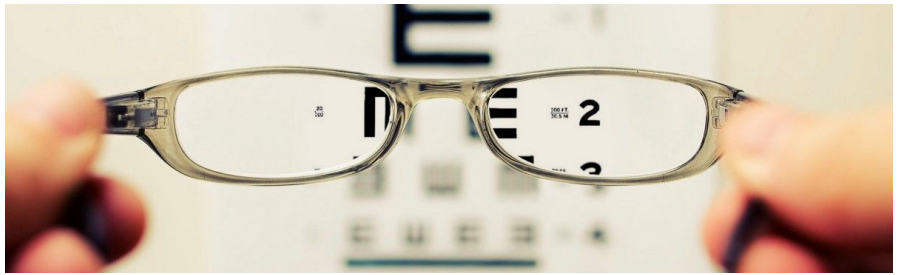



Grazie alla scelta delle funzioni sinusoidali è possibile manipolare le posizioni relative tra una parola ed un'altra tramite una matrice di rotazione, che è un'operazione lineare, e quindi permette al modello di capire le posizioni relative.

# ATTENTION

L’attenzione è una tecnica introdotta per affrontare i problemi di compressione di input grandi legati alle RNN, in cui si possono perdere parti importanti del testo.

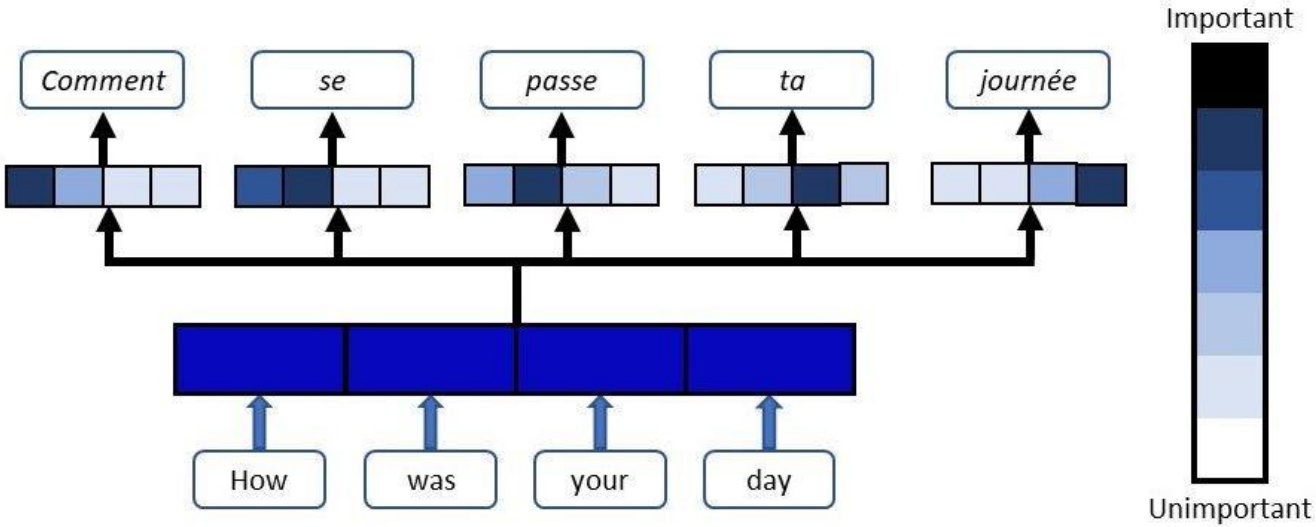
È una tecnica che prova a simulare il comportamento umano, ovvero il fatto che ci si concentra su una piccola parte dell’input per volta, le cui parti sono correlate tra di loro.



*“**Attention is all you need.**”*

# ATTENTION - NELLA TEORIA

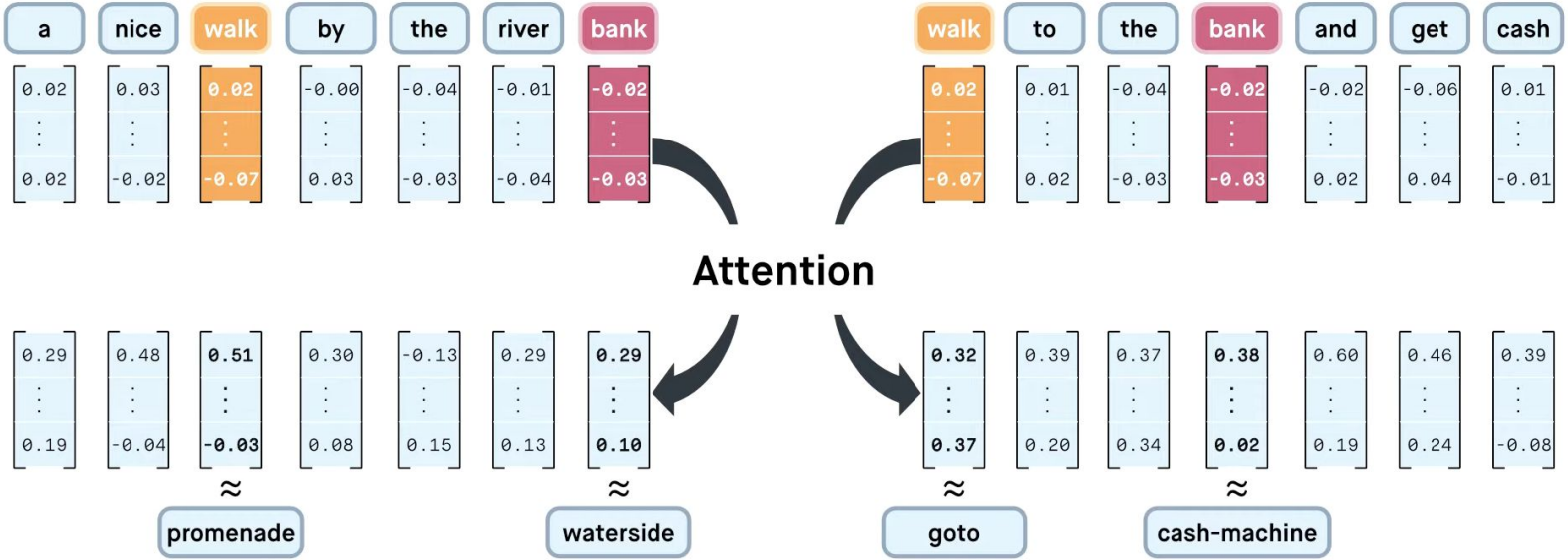
L’attenzione viene implementata come un vettore, chiamato context vector, contenente valori normalizzati che si riferiscono all’importanza da dare a determinate parti dell’input a seconda dell’output che è stato generato al time step corrente.



# ATTENTION

Normalmente gli word embedding vector non hanno valori che sono influenzati dal contesto della frase in cui compaiono.

L'attention prova a incorporare il contesto nei valori di embedding.



# ATTENTION - NELLA PRATICA

Nella pratica l'attention è implementata utilizzando 3 componenti:

- Alignment Scores - per ogni hidden state $\mathbf{h_i}$, e per ogni output già decodificato $\mathbf{s_{t-1}}$, indica quanto bene gli elementi della sequenza di input si allineano con l'output corrente nella posizione **t**. Formalmente si esprime con la funzione

$$\mathbf{e}_{t,i} = a(s_{t-1}, h_i)$$

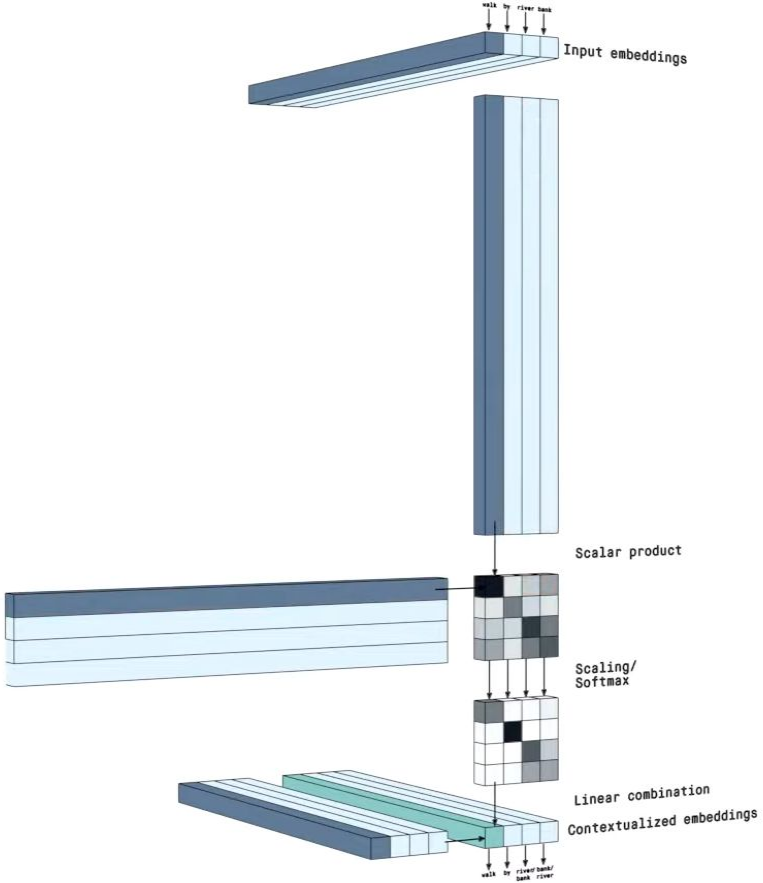
- Weights - applica la softmax all'Alignment Score per calcolare dei pesi la cui somma cumulata è 1. Formalmente

$$\boldsymbol{\alpha}_{t,i} = \text{softmax}(e_{t,i})$$

- Context Vector - è ciò che viene dato in input al decoder ad ogni time step. È calcolato dalla somma pesata di tutti gli hidden state **T**, dell'encoder.

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$$

# Scaled dot-product Self-Attention




La Scaled dot-product Self-Attention si basa sul prodotto scalare tra i word embeddings dell'input.

Il prodotto scalare ritorna un valore che indica la somiglianza tra i due vettori moltiplicati. Possiamo utilizzare questo valore per capire quanto due word embeddings sono vicini nello spazio N-dimensionale nel quale sono rappresentati, questo ci fa capire che due embeddings vicini sono sicuramente correlati fra loro e perciò devono essere presi in considerazione entrambi allo stesso momento.

Al risultato del prodotto scalare applicheremo la funzione softmax per normalizzare ed introdurre non linearità al calcolo.

Per finire andremo a fare una combinazione lineare tra il world embedding iniziale e la colonna corrispondente a quel word embedding nella matrice ottenuta in precedenza così da avere il valore dei Contextualized embeddings.

# Scaled dot-product Self-Attention

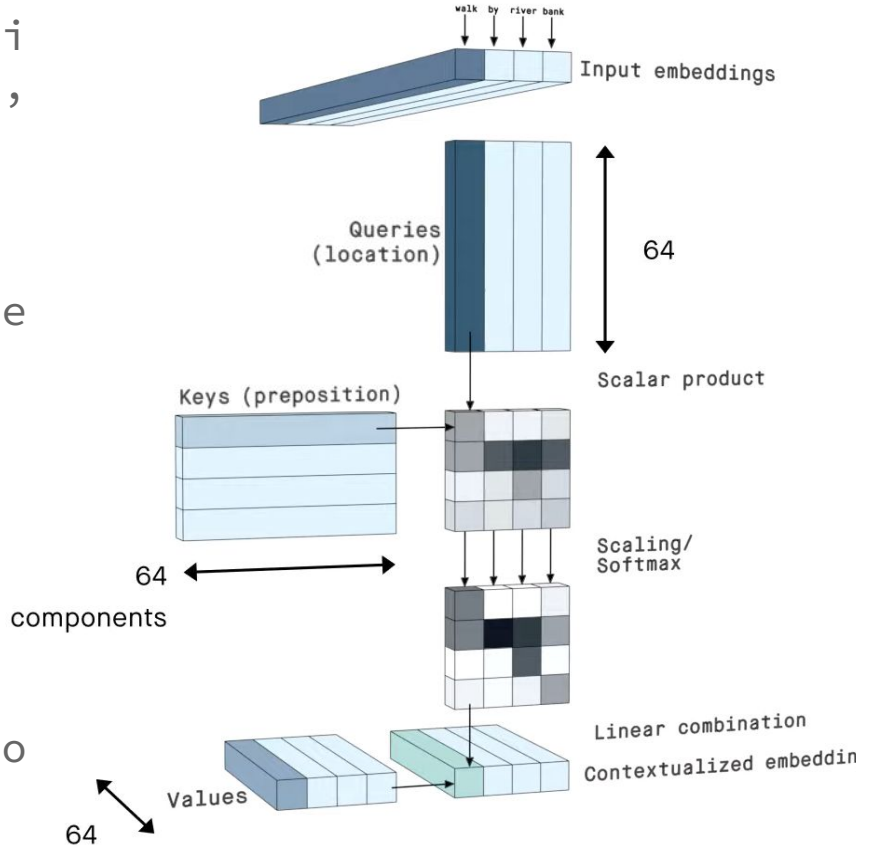
Una possibile ottimizzazione è quella di utilizzare differenti parti del vettore, la cui scelta verrà appresa dalla rete stessa durante il training.

Questo processo evidenzia solo determinate parti del vettore (posizione della parola, preposizioni, significato).

Queste tre parti verranno chiamate:

- Queries
- Keys
- Values

Nell'esempio di Bert abbiamo che la loro lunghezza è prefissata a 64 entry.



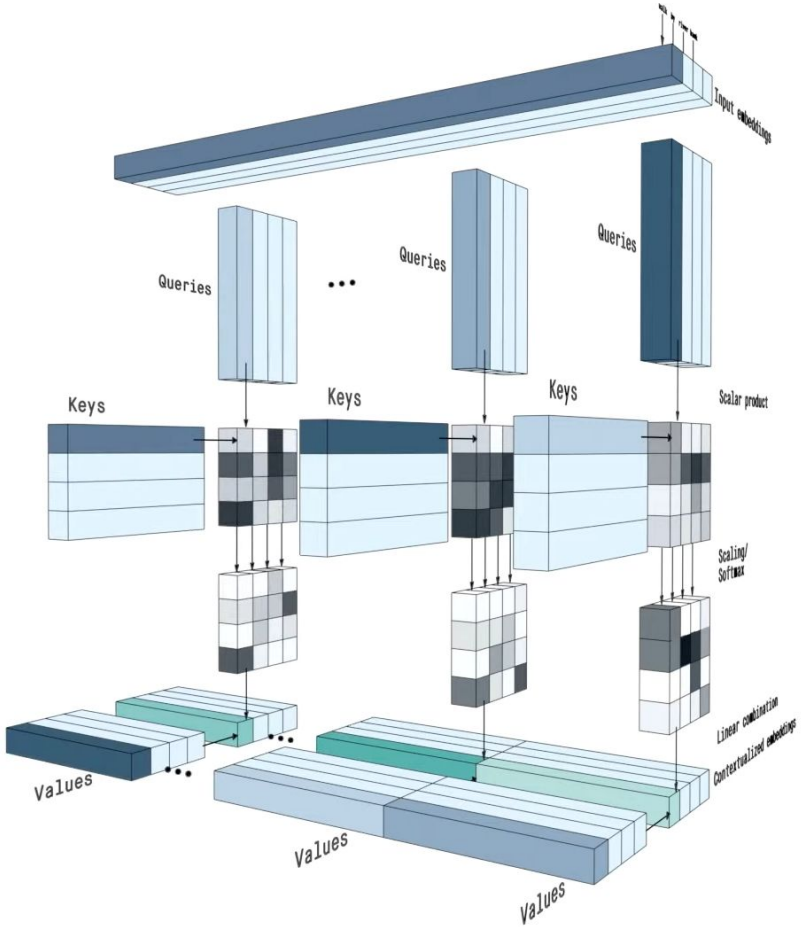
# Multi Head Attention

Nel caso della Multi Head Attention possiamo ripetere l'applicazione della Self-Attention più volte.

Questa idea è molto importante in quanto così potremo prendere in esame più parti dell'array di word embedding di input e cogliere differenti aspetti del linguaggio come ad esempio:

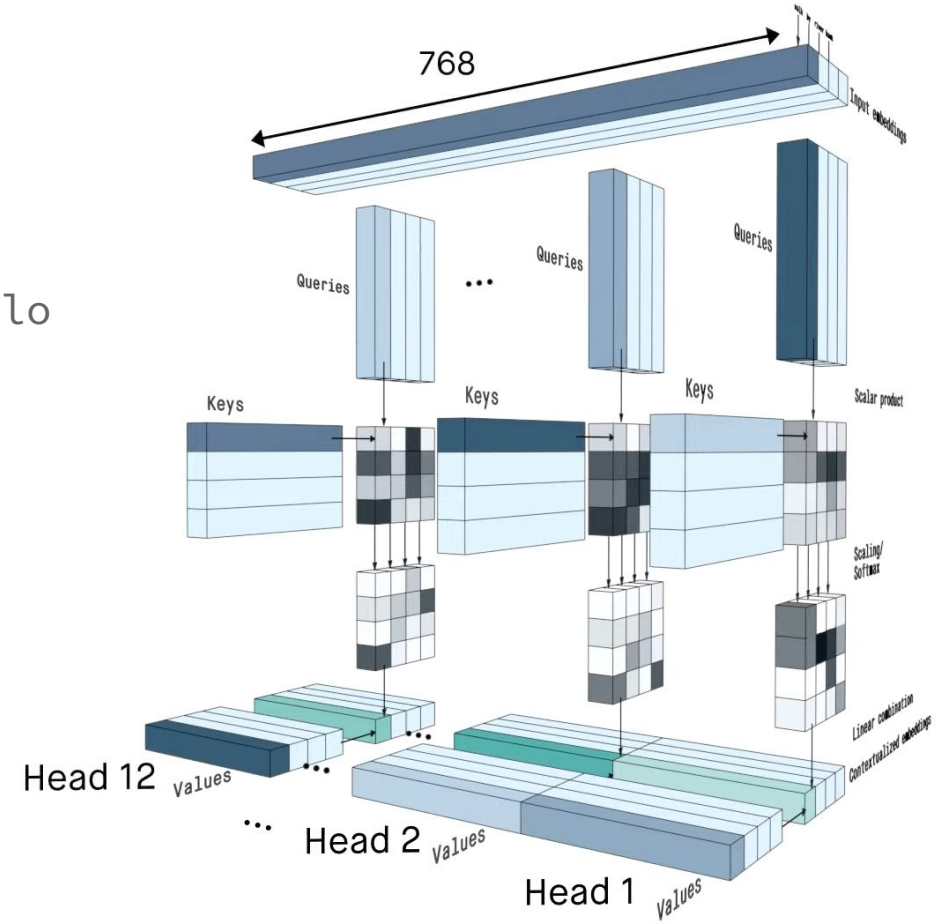
- Punteggiatura
- Posizionamento nella frase
- Preposizioni

Alla fine dell'applicazione della Multi Head Attention il modello andrà a concatenare i vari risultati nel context vector che verrà passato ai nodi successivi della rete.



# Multi Head Attention Bert Specifics

Nel caso specifico di BERT, il modello utilizza 12 testine differenti che vanno in totale a creare un context vector di 768 entry, esattamente grande quanto i word embeddings utilizzati per ogni parola presa in analisi.

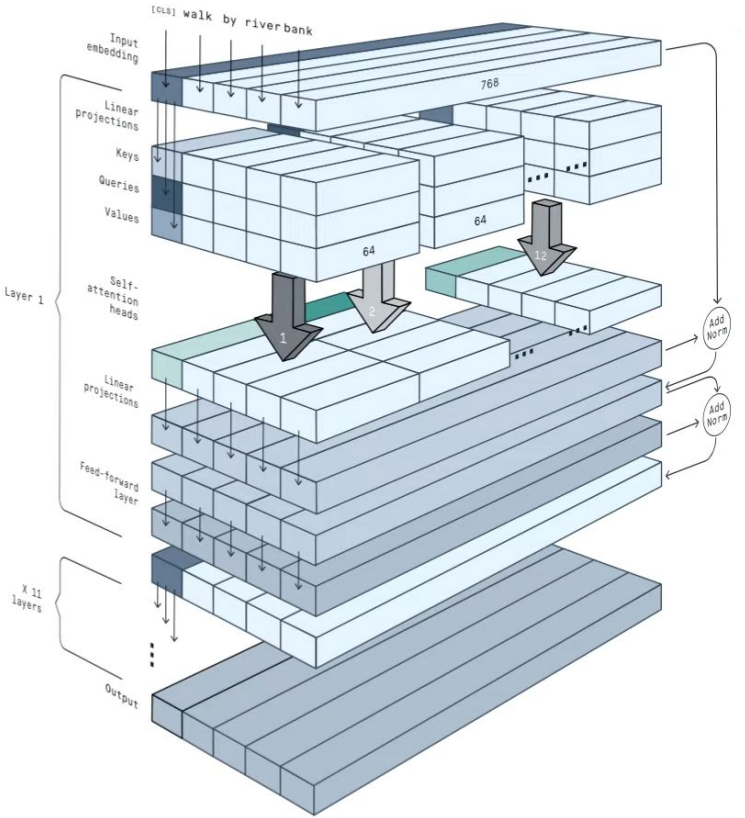


# Bidirectional Encoder Representations from Transformers: BERT

L’intuizione dietro BERT è quella di voler creare un'architettura di NLP che non sia Task-Specific.

Questo obiettivo è stato realizzato da Jacob Devlin et al. i quali hanno utilizzato una componente di pre training di tipo unsupervised per fornire al modello molti contesti per apprendere delle rappresentazioni del linguaggio.

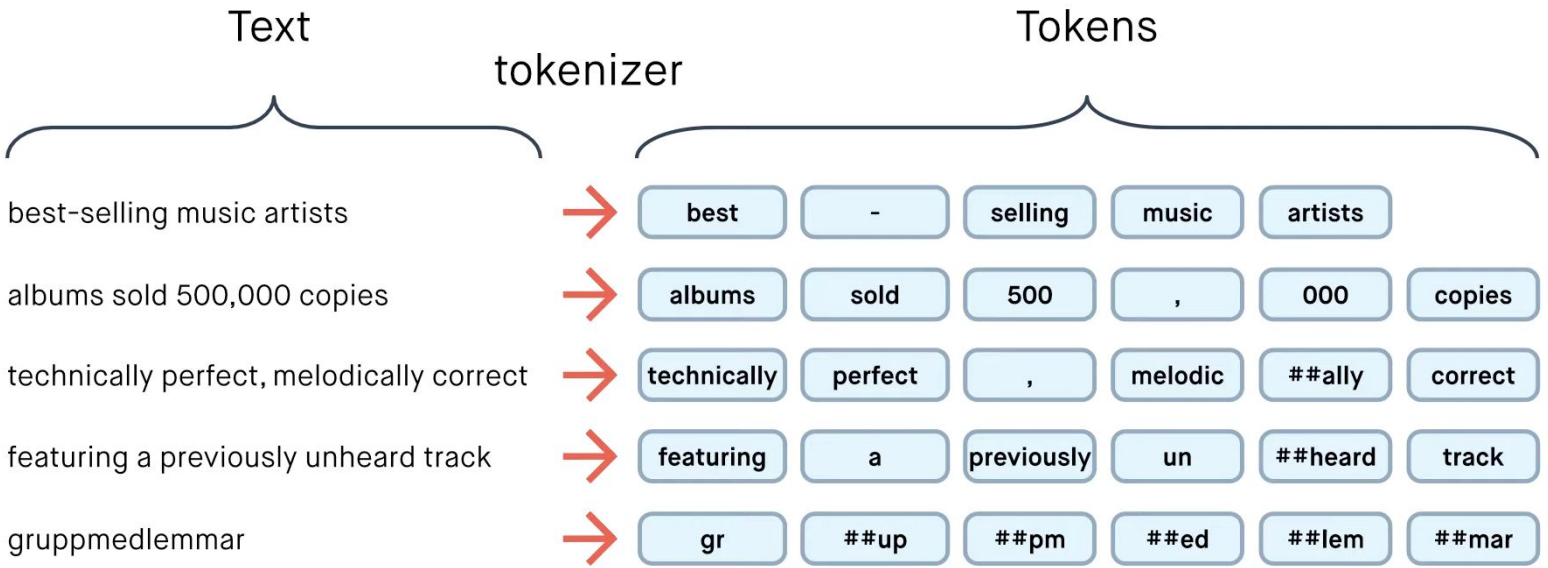
Inoltre la natura unsupervised di questa fase permette di sfruttare corpus enormemente più grandi di quelli usati dal transformer originale.



# Bert Tokenizer: WordPiece

Prima di prendere in input il testo BERT utilizza un tokenizer per individuare le parole e la punteggiatura: **WordPiece.**

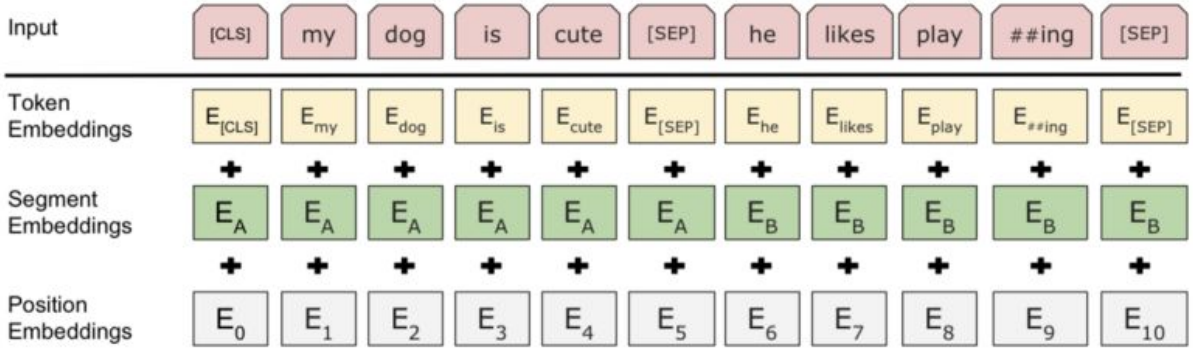
WordPiece permette anche di individuare suffissi e di ricondurre parole mai viste prima ad altre già viste dividendole in sillabe



# Bidirectional Encoder Representations from Transformers: BERT

Per accomodare un pre training che copre vari obiettivi, Jacob Devlin et al. hanno usato la WordPiece Tokenization e hanno anche introdotto dei token speciali:

- [CLS] - Primo token di ogni sequenza di frasi, utilizzato per specificare il task di classificazione.
- [SEP] - Separa due frasi mettendole in relazione Domanda - Risposta è utilizzata su task multifrase.
- [MASK] - Token che permette di mascherare alcuni token di input, per la text prediction.



# Perché bidirectional?

Differentemente dai Language Model classici, bert non ha un approccio “left-to-right” o “right-to-left” (in base a come viene letta la frase), ma per apprendere un task di text prediction sfrutta il meccanismo di self-attention combinato con una maschera che va a “nascondere” parti dell’input, che dovranno essere predette dal modello (nel paper chiamano questo approccio self-supervised).

Il bidirectional dunque nasce dal fatto che l’attention non considera solo le parole che precedono quella che vogliamo predirre, ma considera tutte quelle che la circondano, donando alla classificazione questa bidirezionalità.

Questa tecnica prende il nome di **Bi-directional cross attention**.

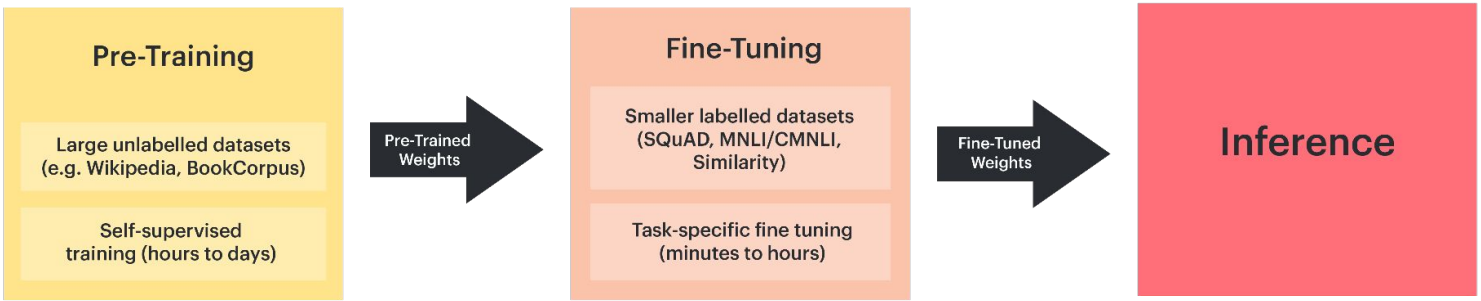
# BERT Fine-Tuning

Grazie al processo di pre training, per utilizzare BERT basta eseguire uno step di fine-tuning, aggiustando dunque i parametri del modello per i task che ci interessano, ma senza necessità di allenare nuovamente il tutto.

Alcuni esempi di fine tuning sono:

- Per i task di classificazione, inseriamo la rappresentazione finale **[CLS]** all'output layer.

- Per le attività con più frasi, l'encoder può processare una coppia di testo concatenata (usando **[SEP]**) con la bi-directional cross attention tra due frasi.



# Riferimenti

- Positional Encoding: Articolo su positional encoding
- Attention: Articolo su attention di base
- Self-attention : How to get meaning from text with language model BERT | AI Explained
- Bert: Articolo su BERT
- Paper Originale: Attention is all you need

# A cura delle Typing Monkeys 

Tommaso Romani

Nicolò Posta

Nicolò Vescera