

TRANSFORMERS

Nuove architetture neurali, introdotte da Google nel 2016.

Dizionario

Definiamo questa struttura dati, il dizionario D , come coppie chiave valore:

$$D = \{(k_1, v_1) \dots (k_m, v_m)\} \quad k_i \in \mathbb{R}^d, v_i \in \mathbb{R}^d \quad \forall i=1 \dots m$$

Quando riceviamo una query $q \in \mathbb{R}^d$ (nel nostro caso, consideriamo q come un token o singolo word embeddings) si fa l'operazione di lookup nel dizionario

$$\text{Lookup}(q, D) = \begin{cases} v_i & \text{se } q = k_i \\ \perp & \text{altrimenti} \end{cases}$$

Questa operazione ritorna sempre la chiave più vicina alla query se quest'ultima non è presente in D . Se ci sono più chiavi equidistanti, coincidenti con i più vicini a q ? Possiamo migliorare, ritrovando una combinazione lineare delle chiavi. Questa operazione è detta di soft-lookup:

$$\text{soft-Lookup}(q, D) = \sum_{i=1}^m \underbrace{\alpha(q, k_i)}_{\text{pesi}} \cdot v_i \equiv \tilde{v}$$

Abbiamo una combinazione lineare di tutti i dati, pesata coi pesi $\alpha(q, k_i) \in \mathbb{R}$ detti anche **ATTENTION WEIGHTS**. L'operazione di soft-lookup è detta **ATTENTION POOLING**.

Hard Lookup o Hard Pooling

$$\begin{cases} \alpha(q, k_i) = 1 & \text{se } q = k_i \\ 0 & \text{altrimenti} \end{cases}$$

Average Pooling

$$\alpha(q, k_i) = \frac{1}{m} \rightarrow \text{centrale di tutte le chiavi}$$

CALCOLO DEGLI ATTENTION WEIGHTS

Un approccio molto comune è quello di misurare la distanza fra q e k_i , per poi calcolare il relativo attention weight come inversamente proporzionale a questa distanza.

Chiamiamo questo similarity score, attention score $\alpha(q, k_i)$.

Possiamo ricavare una combinazione convessa dei pesi (ovvero con $\alpha_i > 0$ e $\sum \alpha_i = 1$) usiamo la softmax sugli attention score:

$$\begin{aligned} \alpha(q, k_i) &= \text{softmax} \left([\alpha(q, k_1) \dots \alpha(q, k_m)] \right) = \\ &= \frac{\exp(\alpha(q, k_i))}{\sum_{j=1}^m \exp(\alpha(q, k_j))} \end{aligned}$$

Il nostro obiettivo è però il calcolo dell'attention score.

Il modo più efficiente per calcolarlo è usare la similarità fra q e k_i , che è data dal prodotto scalare $q^T k_i$.

Se assumiamo che le componenti di q e k_i sono

indipendenti e identicamente distribuite con media nulla
e varianza unitaria ($\mu=0$, $\sigma^2=1$), è facile da provare
che anche il loro prodotto scalare $q^T K_i$ ha $\mu=0$ e
 $\sigma^2=d$, dove d è il numero di componenti dei vettori.
Questo numero (d) può esplodere facilmente, quindi per
mantenerlo circa unitario, riscaliamo $q^T K_i$ usando \sqrt{d} ,
ottenendo il **prodotto scalare scalato degli attention score**:

$$\alpha(q, k_i) = \frac{q^T k_i}{\sqrt{d}} \quad \forall i=1 \dots m$$

Rappresentando q e K_i come vettori riga:

$$q \in \mathbb{R}^{1 \times d} \quad \left\{ \begin{array}{l} q = [\text{---}] \end{array} \right.$$

$$k_i \in \mathbb{R}^{1 \times d} \quad \left\{ \begin{array}{l} \text{m vettori riga} \end{array} \right.$$

$$v_i \in \mathbb{R}^{1 \times V} \quad \left\{ \begin{array}{l} \text{m vettori riga} \end{array} \right.$$

$$K = \begin{bmatrix} \rightarrow K_1 \rightarrow \\ \rightarrow K_2 \rightarrow \\ \vdots \\ \rightarrow K_m \rightarrow \end{bmatrix} \quad \in \mathbb{R}^{m \times d}$$

$$V = \begin{bmatrix} \rightarrow V_1 \rightarrow \\ \rightarrow V_2 \rightarrow \\ \vdots \\ \rightarrow V_m \rightarrow \end{bmatrix} \quad \in \mathbb{R}^{m \times V}$$

$$\frac{1}{\sqrt{d}} [q K_1^T, \dots, q K_m^T] = \frac{1}{\sqrt{d}} [\rightarrow q \rightarrow] \begin{bmatrix} \uparrow & \uparrow \\ K_1 \dots K_m \\ \downarrow & \downarrow \end{bmatrix} = \frac{q K^T}{\sqrt{d}}$$

Quindi l'output \tilde{V} dello soft-Cookup al prodotto scalare degli attention score è:

$$\tilde{V} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

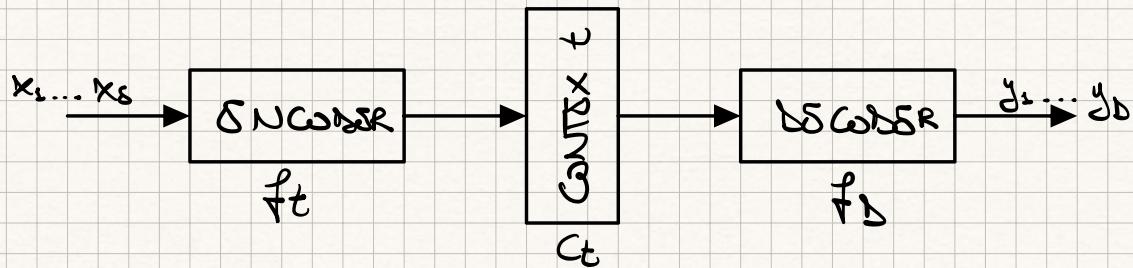
Questo è l'**ATTENTION FORMULA**

Se abbiamo N query rappresentate da vettori q_1 e vogliamo un output, possiamo riarrangiare le matrici come segue:

$$Q = \begin{bmatrix} \leftarrow q_1 \rightarrow \\ \vdots \\ \leftarrow q_N \rightarrow \end{bmatrix} \quad \tilde{V} = \begin{bmatrix} \leftarrow \tilde{V}_1 \rightarrow \\ \vdots \\ \leftarrow \tilde{V}_N \rightarrow \end{bmatrix}$$

Per ottenere $\tilde{V} = \text{attention}(Q, K, V)$

Modulo SSQ2SSQ



Dynamic Context C_t

$$C_t = \sum_{i=1}^n \alpha_i (h_{t-1}^d, h_i^e) h_i^e$$

Annotations:

- A green bracket groups h_{t-1}^d and h_i^e , with a blue arrow pointing to it from the text "Encoder hidden state che si comporta come chiave-value".
- A green bracket groups h_i^e with a blue arrow pointing to it from the text "Stato precedente che si comporta da query per il livello corrente".

Dato che stiamo lavorando con sequenze, qualche volta potremo dover **forzare** il meccanismo di attention per ignorare alcune coppie chiave valore. Questo è detto **masked attention** e viene implementato facilmente tramite la **masked softmax function**. Se la coppia K_e, V_e deve essere mascherata, bisogna settare a mano lo score della query q .

$$Q(q, K_e) = -\infty \rightarrow \underline{\underline{-10^6}}$$

Da cui ricaviamo: $Q(q, K_e) = \emptyset$

Così facendo ignoriamo il contributo del valore V_e .

Generalmente, le query, le chiavi e i valori che useremo sono generati da delle NN. Fra l'altro non conosciamo a priori le dimensioni di q , K e V , né tantomeno sappiamo se vengono generate cose compatibili. Quindi, comuneamente vengono trasformate q , K e V in vettori che hanno una rappresentazione compatibile:

$$q \in \mathbb{R}^{1 \times d} \longrightarrow q' = q W^q \in \mathbb{R}^{1 \times pd} \quad \text{con } W^q \in \mathbb{R}^{d \times pd}$$

$$K_i \in \mathbb{R}^{1 \times d} \longrightarrow K'_i = K_i W^K \in \mathbb{R}^{1 \times pd} \quad \text{con } W^K \in \mathbb{R}^{d \times pd}$$

$$V_i \in \mathbb{R}^{1 \times d} \longrightarrow V'_i = V_i W^V \in \mathbb{R}^{1 \times pd} \quad \text{con } W^V \in \mathbb{R}^{d \times pd}$$

Mettendo queste matrici nell'attention:

$$\tilde{V} = \text{attention}(q' W^q, K' W^K, V' W^V)$$

Detto anche **ATTENTION HEAD**

$$\tilde{V} = \text{attentionHead}(Q, K, V | W^q, W^K, W^V) \in \mathbb{R}^{m \times pd}$$

↳ batch di
in query

TRANSFORMERS BUILDING BLOCKS

1) SELF ATTENTION - Primo blocco

È una sorta di attention head dove la query, le chiavi e i valori sono identici: $x_i \in \mathbb{R}^d$

$$\text{Per } i=1, \dots, n \Rightarrow X = \begin{bmatrix} \leftarrow x_1 \rightarrow \\ \vdots \\ \leftarrow x_n \rightarrow \end{bmatrix}$$

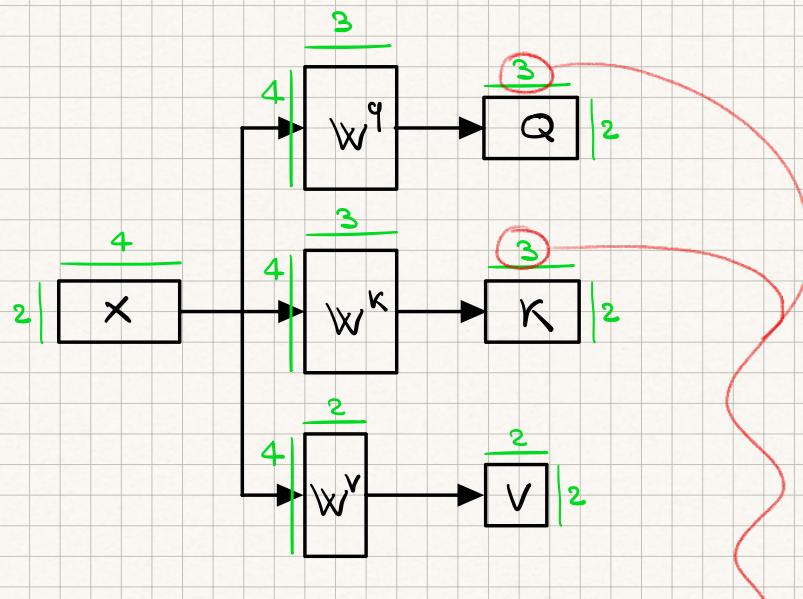
Quindi

$$\tilde{V} = \text{attention}(XW^q, XW^k, XW^v)$$

$$\tilde{V} = \text{attentionHead}(X, X, X | W^q, W^k, W^v)$$

Scenario:

Con due parole all'ingresso w_1, w_2 e relativi word embeddings $x_1, x_2 \in \mathbb{R}^{1 \times 4}$, messi nella matrice $X \in \mathbb{R}^{2 \times 4}$



Abbiamo:

$$\tilde{V} \in \mathbb{R}^{2 \times 2} \quad \tilde{V} = \text{softmax} \left(\frac{QK^T}{\sqrt{3}} \right) V$$

$$\tilde{V} = \text{softmax} \left(\frac{1}{\sqrt{3}} \begin{bmatrix} q_1 K_1^T & q_1 K_2^T \\ q_2 K_1^T & q_2 K_2^T \end{bmatrix} \right) V =$$

$$= \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix} = \begin{bmatrix} \xleftarrow{\tilde{V}_1} \xrightarrow{\tilde{V}_1} \\ \xleftarrow{\tilde{V}_2} \xrightarrow{\tilde{V}_2} \end{bmatrix}$$

MULTIPLE (SELF) ATTENTION

Usiamo la multiple (self) attention. Assumiamo di avere h diverse

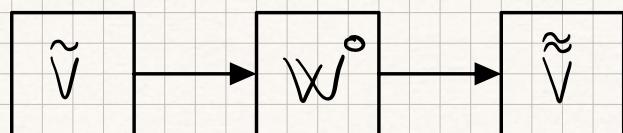
$$\tilde{V}_i = \text{attentionHead}(X, X, X | W_i^q, W_i^K, W_i^V) \quad \forall i=1 \dots h$$

$\tilde{V}_i \in \mathbb{R}^{m \times p_V}$, dove m è la lunghezza di $x_1 \dots x_m$.

Uniamo orizzontalmente tutte le $\tilde{V}_1 \dots \tilde{V}_h$ in una unica matrice $\tilde{V} \in \mathbb{R}^{m \times hp_V}$:

$$\tilde{V} = \begin{bmatrix} \tilde{V}_1 & | & \tilde{V}_2 & | & \cdots & | & \tilde{V}_h \end{bmatrix}$$

Proiettiamo ora \tilde{V} in una rappresentazione imparabile da una NN:



Dove:

$$\begin{aligned}\tilde{V} &\in \mathbb{R}^{M \times hP_V} \\ \tilde{\tilde{V}} &\in \mathbb{R}^{M \times P_O} \\ W^o &\in \mathbb{R}^{hP_V \times P_O}\end{aligned}$$

$$\tilde{\tilde{V}} = \tilde{V} W^o \equiv \text{MULTI HEAD ATTENTION}(x)$$

Note: Bisogna tenere sotto controllo la varianza, in modo che tutti i neuroni della rete ricevano i giusti aggiornamenti e non aggiornamenti minimi o nulli.

2a) LAYER NORMALIZATION - Secondo blocco, a

Sia $x \in \mathbb{R}^M$ l'ingresso (In realtà è un vettore riga, ma lo consideriamo colonna per semplicità)

$$x \rightarrow \frac{x - \mu_x}{\sigma_x} = \text{Layer Normalization}(x)$$

Dove: $\mu_x = \frac{1}{M} \sum_{i=1}^M x_i$; $\sigma = \frac{1}{M} \sum_{i=1}^M (x_i - \mu_x)^2 + \epsilon$

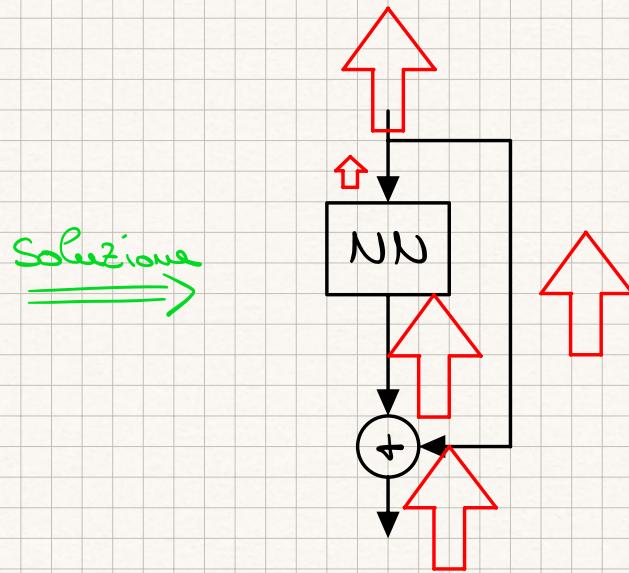
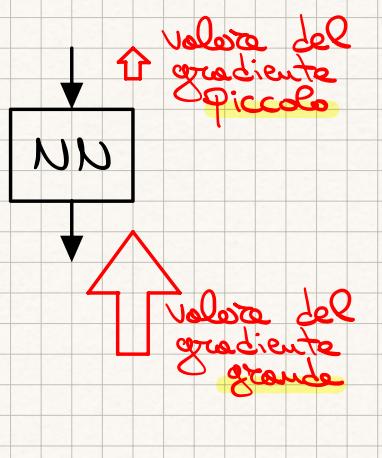
Aggiungiamo ϵ per evitare che $\sigma = 0$ e quindi di dividere per zero.

In qualche implementazione, il layer normalization presenta due parametri imparabili γ e β :

$$x \rightarrow \gamma \frac{x - \mu_x}{\sigma_x} + \beta$$

2b) SKIP CONNECTION - Secondo blocco, b

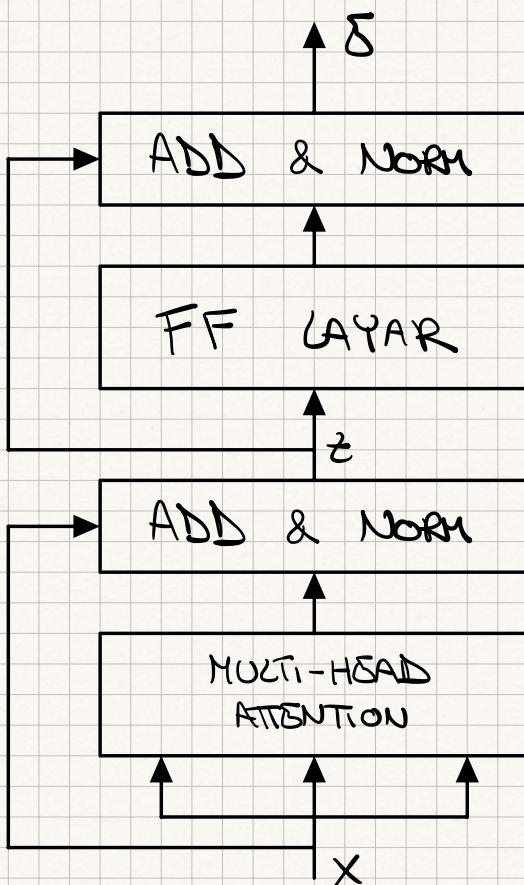
È un trucco di ottimizzazione, per evitare che il gradiente durante la backpropagation, vada a svanire (**Vanishing gradient**):



Difatti costituiscono la rete.

Attenzione: questa soluzione fa scoppiare la varianza. Va sempre usato in coppia con un meccanismo di controllo della varianza (zo)

3) ENCODER - Terzo blocco



Dove:

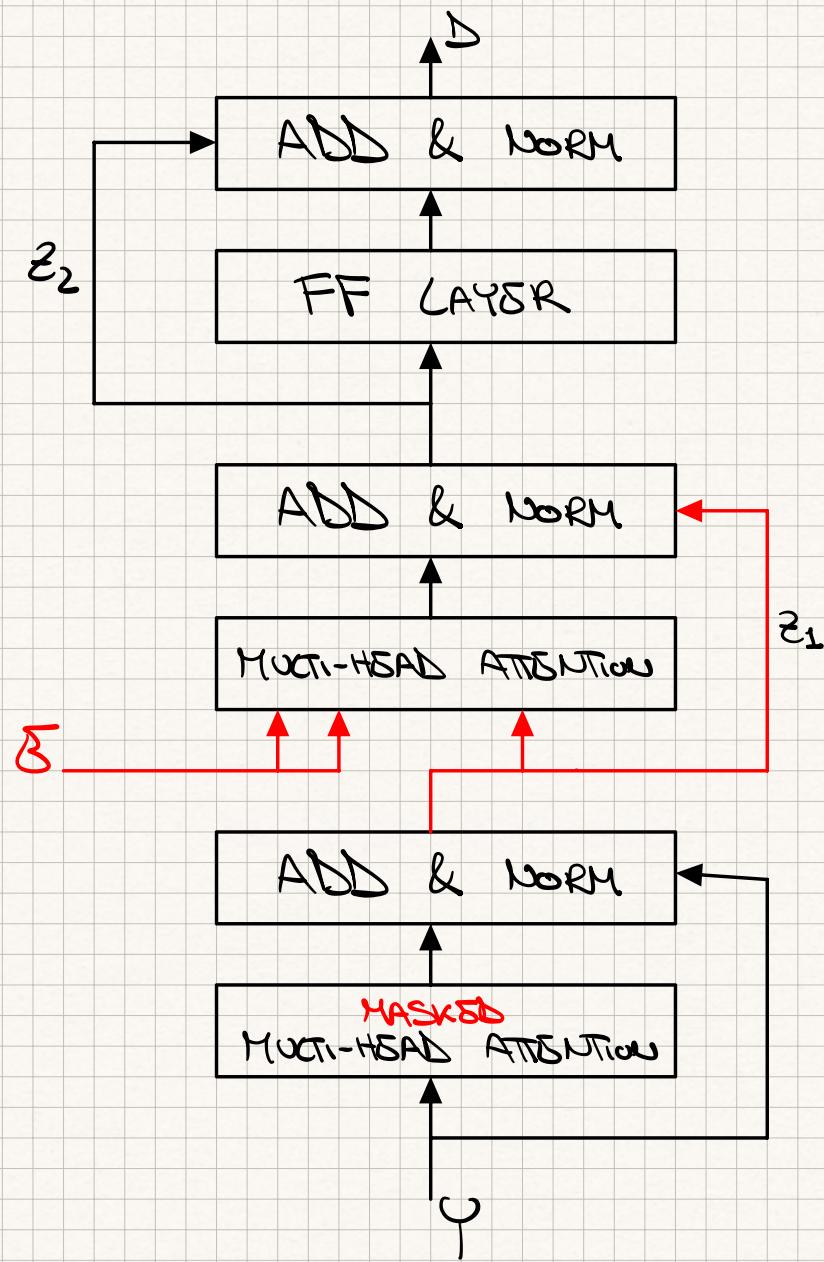
$$1) X \in \mathbb{R}^{n \times d}$$

$$2) Z = \text{Layer Norm}(\text{Multi-head -Attention}(x) + x)$$

$$3) S = \text{Layer Norm}(\text{Feed Forward}(Z) + Z)$$

Il livello FF è composto da due livelli lineari con ercole Gaussiano (Gaussian Outer Linear Unit - GOLU) che è una versione modificata della RSLU, non lineare

4) Decoder - quarto blocco



Dove:

$$1) \gamma \in \mathbb{R}^{m \times d}$$

$$2) \sigma \in \mathbb{R}^{m \times d}$$

$$3) z_1 = \text{LayerNorm}(\text{Masked-Multi head - Attention}(\gamma) + \gamma)$$

$$4) z_2 = \text{LayerNorm}(\text{Multi head - Attention}(z_1, \sigma, \sigma) + z_1)$$

$$5) D = \text{LayerNorm}(\text{Feed Forward}(z_2) + z_2)$$

5) INPUT EMBEDDINGS - quinto blocco

Abbiamo la nostra sequenza di ingresso fatta di parole:

$w_1 \dots w_n \in V^m$. Ogni parola è codificata con i vettori
 $x_1 \dots x_n \in \mathbb{R}^d$:

$$x_i = e_i + p_i$$

Dove:

1) e_i = word embedding

2) p_i = position embedding

Come al solito i word embedding si calcolano:

$$e_i = \sum_{j=1}^n \underbrace{\delta_j}_{\text{one-hot vector}} \underbrace{W}_{\text{embedding matrix}}$$

Per quanto riguarda i position embedding, che danno info sulla posizione della parola nella sequenza di ingresso, si usa un meccanismo non imparabile da una NN.

I modi più semplici:

- 1) Dare un valore da 0 a 1, coerente con la posizione della parola nella frase. Ma ci serve sapere la lunghezza della frase
- 2) Dare un valore incrementale, non dipendente dalla lunghezza della frase. Ma questo numero esplode facilmente.

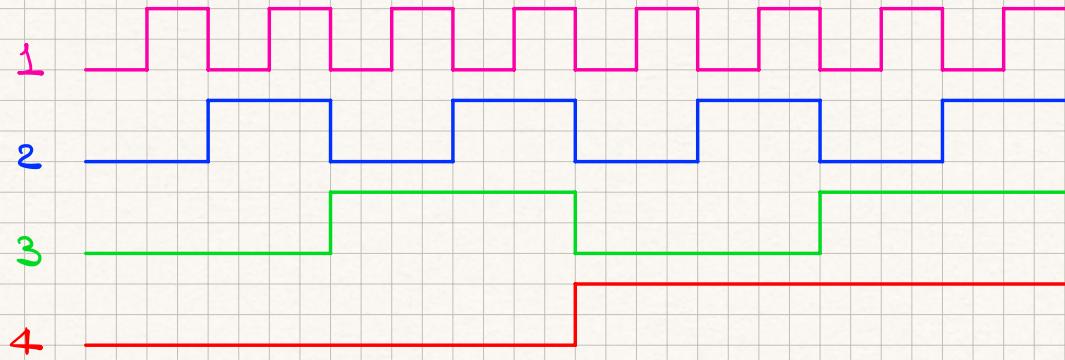
Nella realtà, assumiamo che $\mathbf{p}_i \in \mathbb{R}^d$ sia pari a:

$$\mathbf{p}_i = \begin{bmatrix} p_{i,1} \\ p_{i,2} \\ p_{i,3} \\ \vdots \\ p_{i,d-1} \\ p_{i,d} \end{bmatrix} = \begin{bmatrix} \sin(\omega_1 \cdot i) \\ \cos(\omega_1 \cdot i) \\ \sin(\omega_2 \cdot i) \\ \cos(\omega_2 \cdot i) \\ \vdots \\ \sin(\omega_d \cdot i) \\ \cos(\omega_d \cdot i) \end{bmatrix} \quad \text{con } \omega_k = \frac{1}{10000^{2k/d}}$$

Per capire l'intuizione supponiamo di voler rappresentare i primi 15 numeri binari:

$0: \textcolor{red}{\emptyset} \textcolor{green}{\emptyset} \textcolor{blue}{\emptyset} \textcolor{magenta}{\emptyset}$	$8: \textcolor{red}{1} \textcolor{green}{\emptyset} \textcolor{blue}{\emptyset} \textcolor{magenta}{\emptyset}$
$1: \textcolor{red}{\emptyset} \textcolor{green}{\emptyset} \textcolor{blue}{\emptyset} \textcolor{magenta}{1}$	$9: \textcolor{red}{1} \textcolor{green}{\emptyset} \textcolor{blue}{\emptyset} \textcolor{magenta}{1}$
$2: \textcolor{red}{\emptyset} \textcolor{green}{\emptyset} \textcolor{magenta}{1} \textcolor{blue}{\emptyset}$	$10: \textcolor{red}{1} \textcolor{green}{\emptyset} \textcolor{magenta}{1} \textcolor{blue}{\emptyset}$
$3: \textcolor{red}{\emptyset} \textcolor{green}{\emptyset} \textcolor{magenta}{1} \textcolor{blue}{1}$	$11: \textcolor{red}{1} \textcolor{green}{\emptyset} \textcolor{magenta}{1} \textcolor{blue}{1}$
$4: \textcolor{red}{\emptyset} \textcolor{magenta}{1} \textcolor{green}{\emptyset} \textcolor{blue}{\emptyset}$	$12: \textcolor{red}{1} \textcolor{magenta}{1} \textcolor{green}{\emptyset} \textcolor{blue}{\emptyset}$
$5: \textcolor{red}{\emptyset} \textcolor{magenta}{1} \textcolor{green}{\emptyset} \textcolor{magenta}{1}$	$13: \textcolor{red}{1} \textcolor{magenta}{1} \textcolor{green}{\emptyset} \textcolor{magenta}{1}$
$6: \textcolor{red}{\emptyset} \textcolor{magenta}{1} \textcolor{green}{1} \textcolor{blue}{\emptyset}$	$14: \textcolor{red}{1} \textcolor{magenta}{1} \textcolor{green}{1} \textcolor{blue}{\emptyset}$
$7: \textcolor{red}{\emptyset} \textcolor{magenta}{1} \textcolor{green}{1} \textcolor{magenta}{1}$	$15: \textcolor{red}{1} \textcolor{magenta}{1} \textcolor{green}{1} \textcolor{magenta}{1}$

Guardando i singoli digits si può notare un certo andamento sinusoidale:



Possiamo usare la periodicità di ogni digit, per identificare la posizione relativa a ogni parola.

Esempio

Analizziamo i position embedding di ogni parola della frase: "I am a robot".

Per semplicità usiamo 100 al posto di 10000 e d=4:

$$\sin\left(\frac{i}{100^{2k/4}}\right) \quad \cos\left(\frac{i}{100^{2k/4}}\right)$$

	i	$K=0$	$K=0$	$K=1$	$K=1$
I	\emptyset	$\sin(\emptyset) = 0$	$\cos(\emptyset) = 1$	$\sin(\emptyset) = 0$	$\cos(\emptyset) = 1$
am	1	$\sin(1/1) = 0.84$	$\cos(1/1) = 0.54$	$\sin(1/10) = 0.10$	$\cos(1/10) = 0.995$
a	2	$\sin(2/1) = 0.91$	$\cos(2/1) = -0.42$	$\sin(2/10) = 0.20$	$\cos(2/10) = 0.98$
robot	3	$\sin(3/1) = 0.54$	$\cos(3/1) = -0.99$	$\sin(3/10) = 0.30$	$\cos(3/10) = 0.96$

Dalla matrice ricaviamo i vari P_i :

$$1) I \rightarrow P_0 = [0.0 \quad 1.0 \quad 0.0 \quad 1.0]$$

$$2) am \rightarrow P_1 = [0.84 \quad 0.54 \quad 0.10 \quad 0.395]$$

$$3) a \rightarrow P_2 = [0.91 \quad -0.42 \quad 0.20 \quad 0.38]$$

$$4) robot \rightarrow P_3 = [0.14 \quad -0.99 \quad 0.30 \quad 0.96]$$

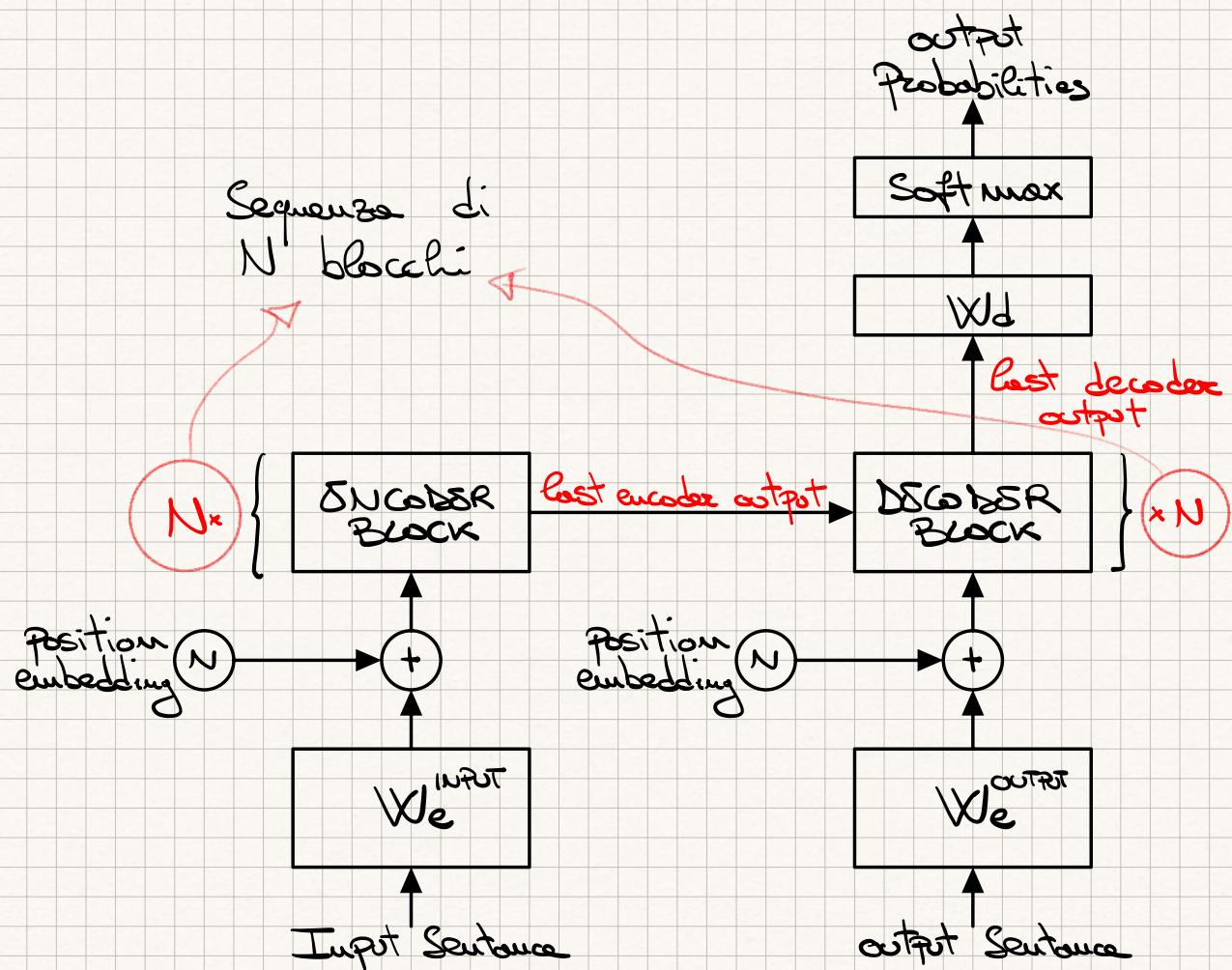
6) OUTPUT EMBEDDINGS - sesto blocc

Arrivati qui, abbiamo un gruppo valori di output $y_i \in \mathbb{R}^{1 \times d}$ che va convertita in una distribuzione di probabilità $Q(w_i)$. Per farlo si usa la softmax. Prima però bisogna riproiettare i componenti della dimensione d alla dimensione del vocabolario V . Per farlo si usa l'embedding matrix $W_d \in \mathbb{R}^{d \times V}$:

$$Q(w_i) = \text{softmax}(y_i W_d)$$

La corrispondente parola data in output è quella che massimizza $Q(w_i)$.

TRANSFORMER SCHEMA



Questa architettura non si usa spesso per intero, ma molti modelli usano o solo la parte di Encoder (BERT) o solo la parte di Decoder (GPT).

In origine i Transformer sono stati introdotti per compiti di traduzione, ovvero usandoli con un apprendimento **supervisionato**. Ma si è presto capito che sono fatti anche con problemi **unsupervised**, usando dataset enormi per ottenere un LM.

Training Objectives for unsupervised Pre-training:

1) Causal Language model

2) Masked Language model

Così facciamo dei modelli pre-addestrati, che sanno solo predire del testo. Bisogna specializzarli in task specifici, senza riallevarli da zero, ma usando quelli pre-addestrati come base.

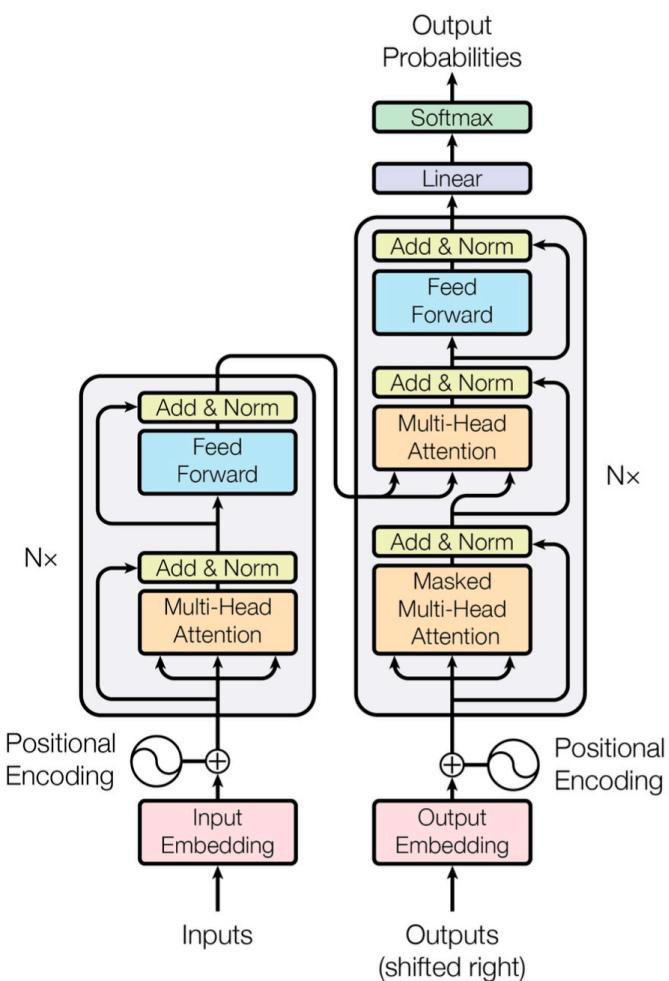


Figure 1: The Transformer - model architecture.