Transformer

CORSO DI NATURAL LANGUAGE PROCESSING (ELABORAZIONE DEL LINGUAGGIO NATURALE)

a.a. 2025/2026

Prof. Roberto Pirrone

Gli LLM sono costituiti da Transformer

- Transformer: un tipo specifico di architettura di rete, simile a una rete feedforward più sofisticata, ma basata sull'attenzione.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

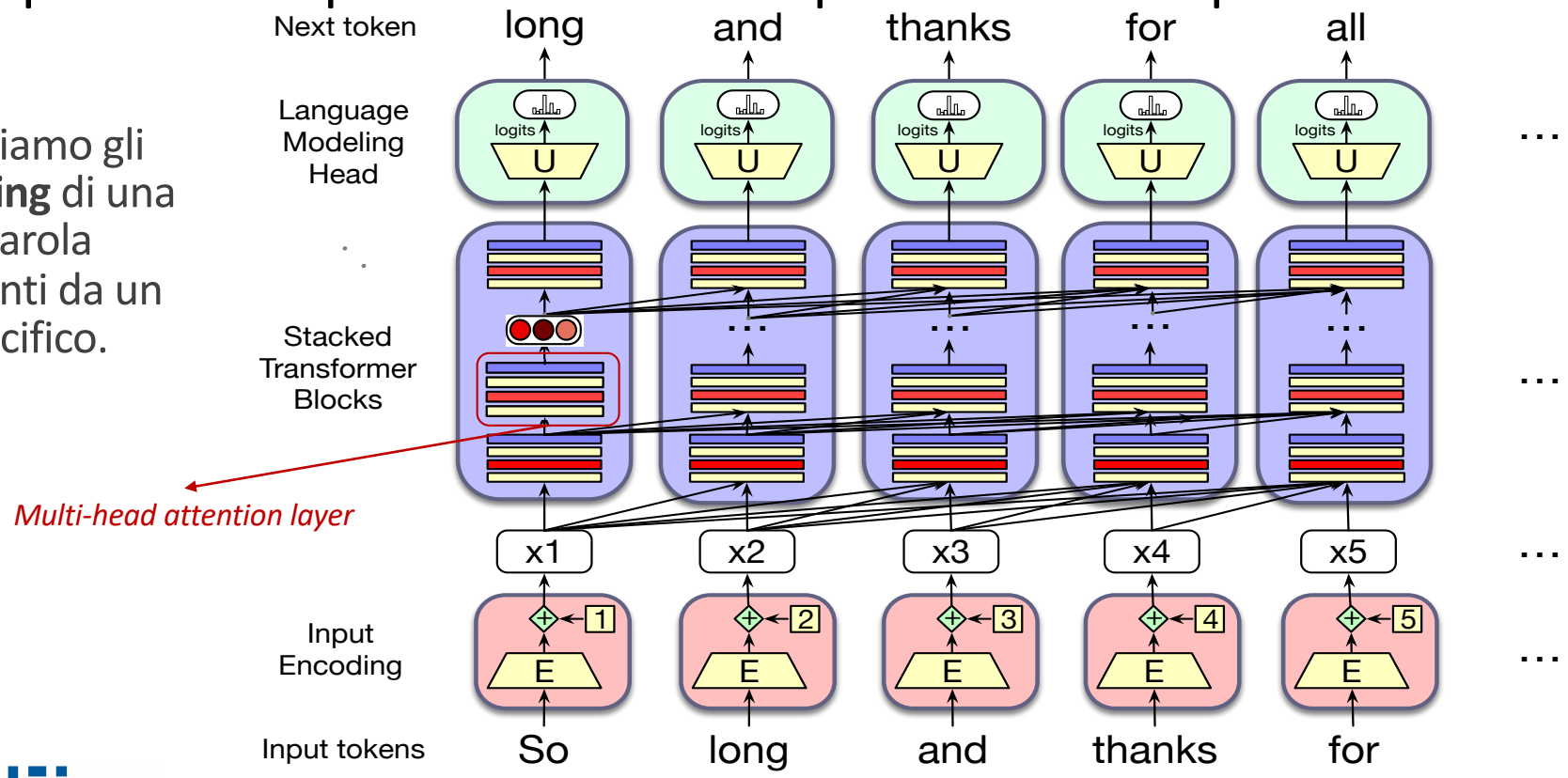
Una semplice linea del tempo

- 1990 Static Word Embeddings
- 2003 Neural Language Model
- 2008 Multi-Task Learning
- 2015 Attenzione
- 2017 Transformer
- 2018 Contextual Word Embeddings and Pretraining
- 2019 Prompting

Attenzione

Al posto di partire con il quadro completo …

Consideriamo gli **embedding** di una singola parola provenienti da un layer specifico.



Problemi con gli embedding statici (word2vec)

- Sono **statici!**
L'embedding di una parola non riflette come il suo significato cambi a seconda del contesto.

- `The chicken didn't cross the road because it was too tired`

- Qual è il significato rappresentato nell'embedding statico della parola "*it*"?

Embedding contestuali

- *Intuizione*: la rappresentazione del significato di una parola dovrebbe essere ***diversa nei diversi contesti***!

- ***Embedding contestuale***: ogni parola ha un ***vettore differente*** che esprime ***significati diversi*** a seconda delle parole che la circondano.

- Come calcolare gli embedding contestuali?
  👉 ***Self-attention***

Embedding contestuali

- `The chicken didn't cross the road because it`

- Quali potrebbero essere le proprietà di "*it*"?

- `The chicken didn't cross the road because it was too` **`tired`**
- `The chicken didn't cross the road because it was too` **`wide`**

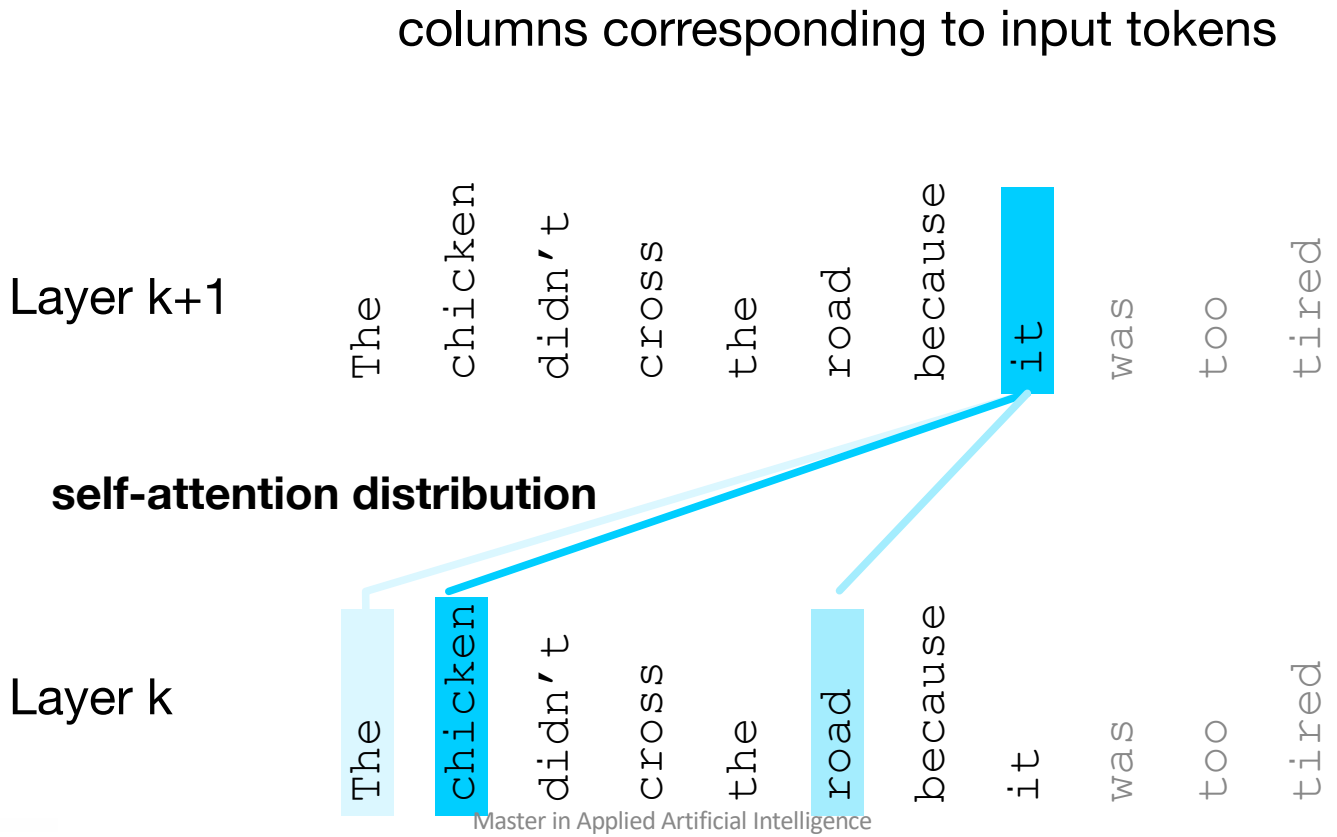
- A questo punto della frase, probabilmente si riferisce o al pollo o alla strada.

Intuizione della self-attention

- Costruire l'embedding contestuale di una parola integrando selettivamente le informazioni provenienti da tutte le parole vicine.

- Si dice che una parola "presta attenzione" (*attends to*) ad alcune parole vicine ***più che ad altre***.

Intuizione della self-attention

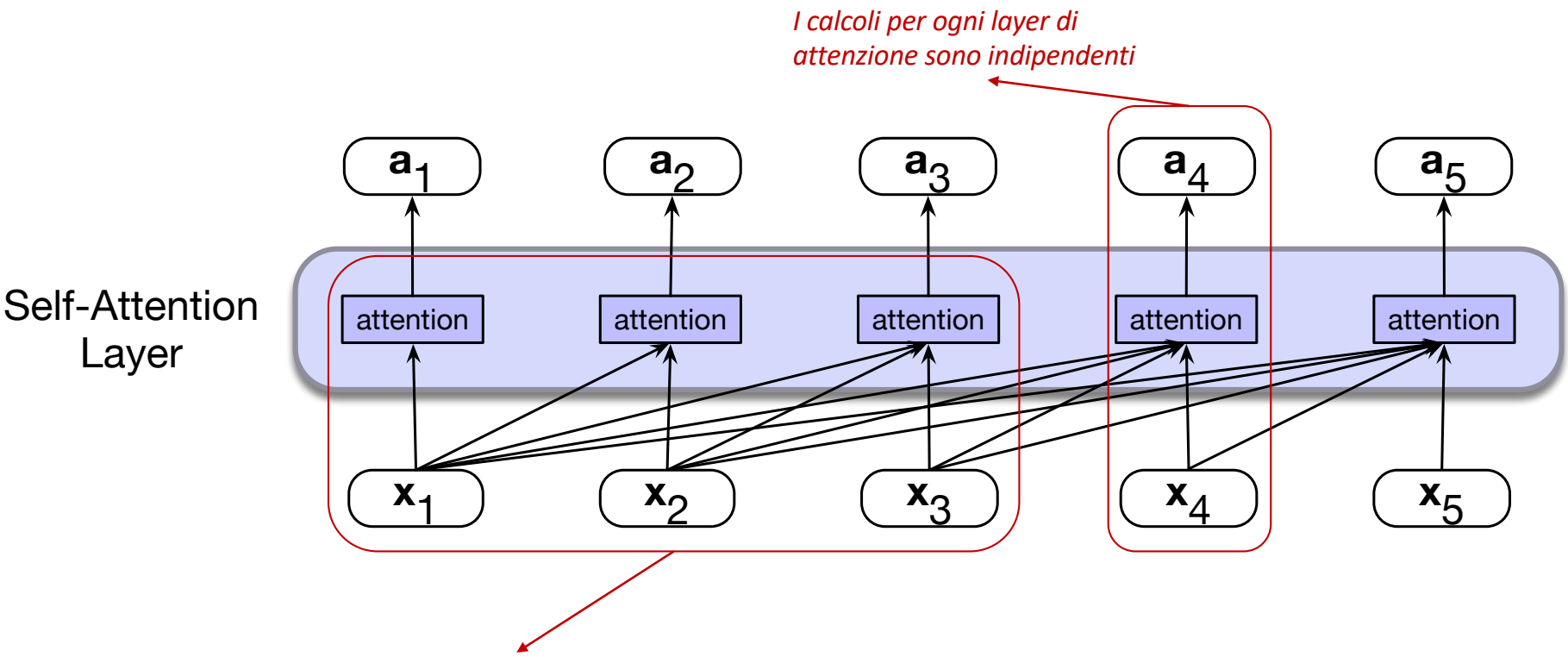


Definizione di attenzione

- Un meccanismo che aiuta a calcolare l'embedding di un token prestando selettivamente attenzione e integrando le informazioni provenienti dai token circostanti (nel livello precedente).

- Più formalmente: un metodo per eseguire una somma pesata di vettori.

L’attenzione va da sinistra a destra



Versione semplificata del meccanismo di attenzione

Data una sequenza di embedding di token:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

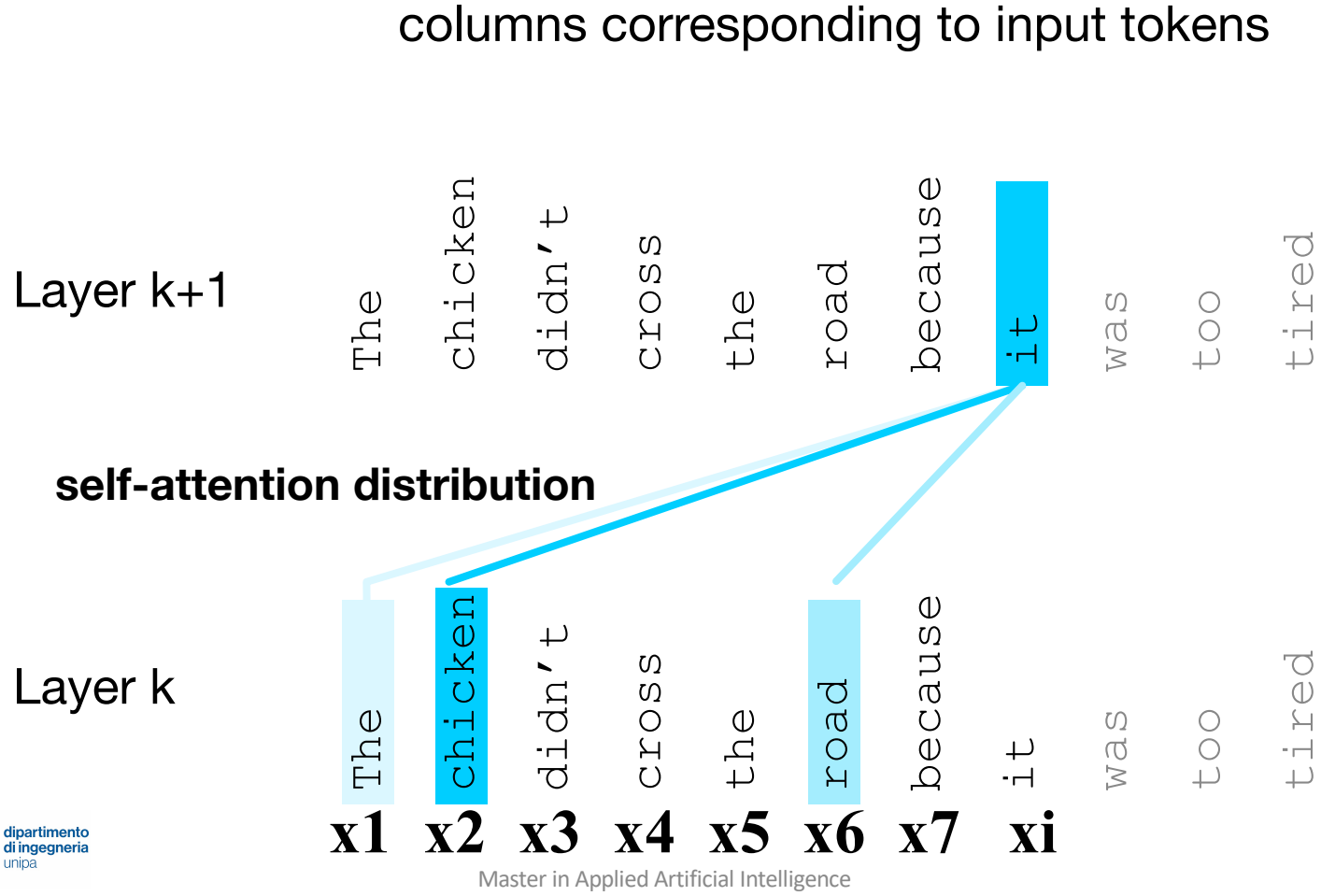
Produci $\mathbf{a}_i$ = somma pesata da $\mathbf{x}_1$ a $\mathbf{x}_7$ (e $\mathbf{x}_i$) pesati rispetto alla loro similarità con $\mathbf{x}_i$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

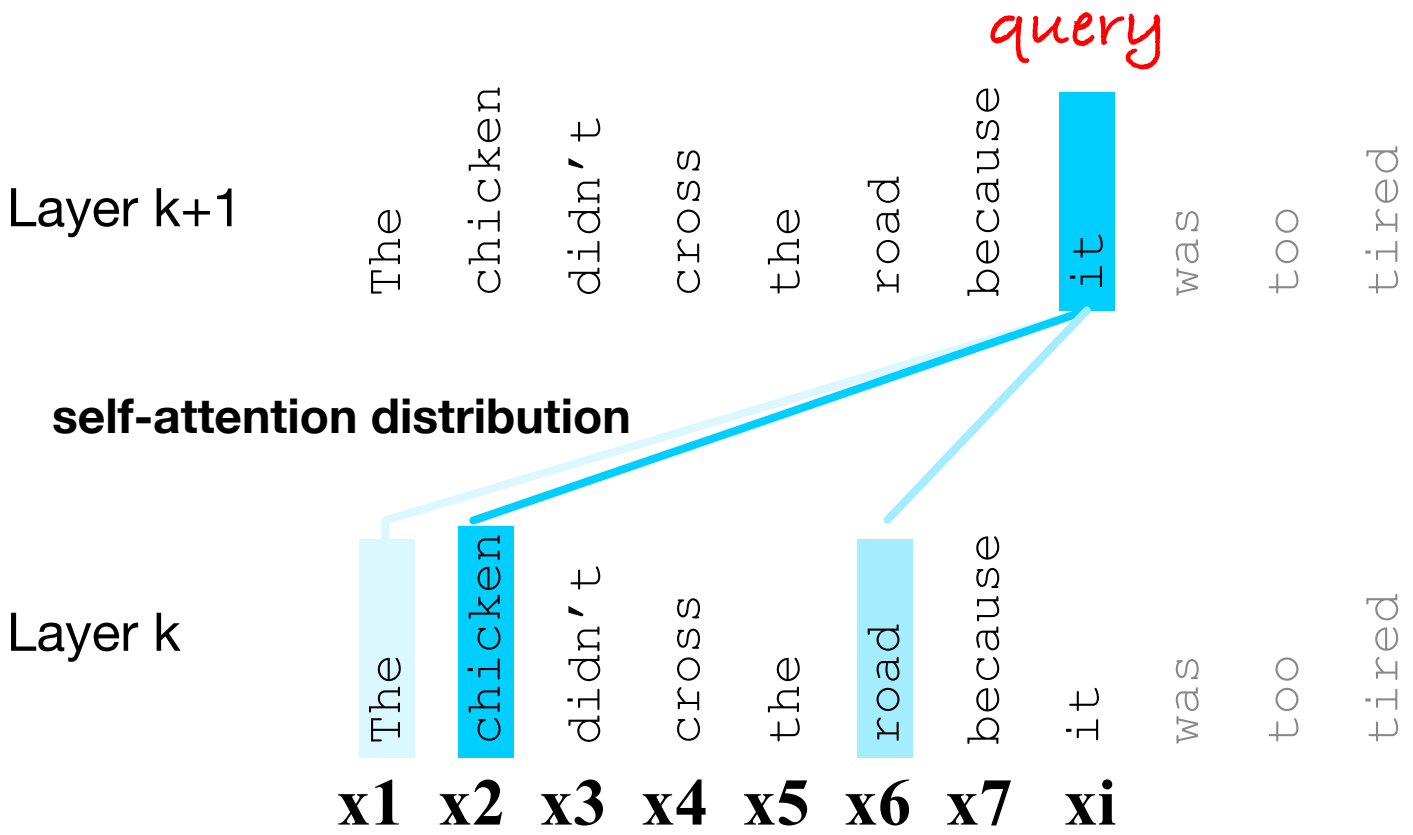
Intuizione della self-attention



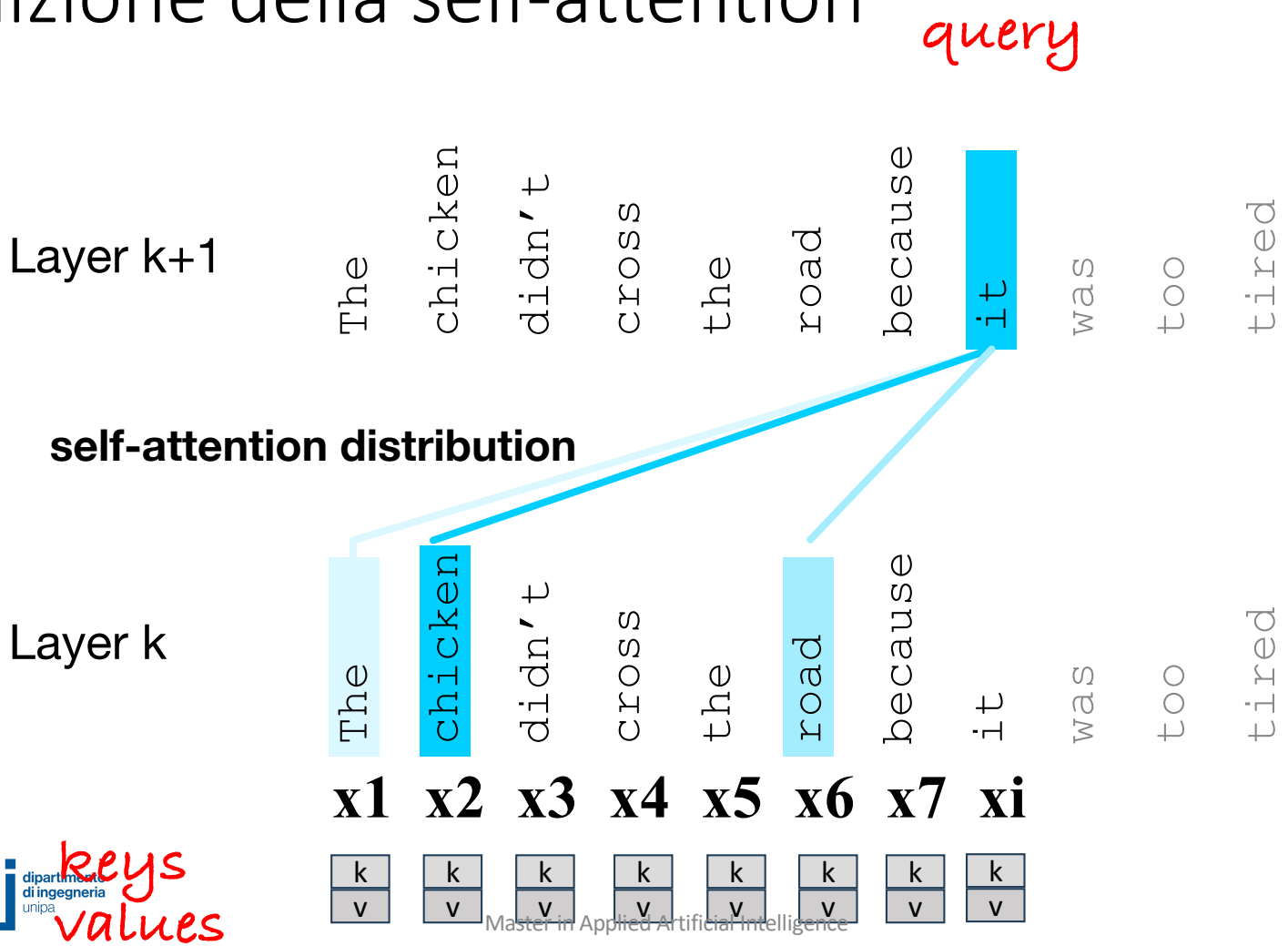
Una vera testa di attenzione: leggermente più complessa

- Idea generale: invece di usare direttamente i vettori (come x_i e x_4), rappresenteremo *tre ruoli distinti* che ciascun vettore x_i svolge:
- **Query:** rappresenta l'*elemento corrente* che viene confrontato con gli input precedenti.
- **Key:** rappresenta un *input precedente* che viene confrontato con l'elemento corrente per determinare una misura di somiglianza.
- **Value:** rappresenta il *valore di un elemento precedente* che viene *pesato e sommato* per calcolare l'output dell'elemento corrente.

Intuizione della self-attention



Intuizione della self-attention



Una vera testa di attenzione: leggermente più complessa

- Useremo matrici per proiettare ogni vettore $\mathbf{x}_i$ nella rappresentazione del suo ruolo come query, chiave e valore:
- **query**: $\mathbf{W}^Q$
- **key**: $\mathbf{W}^K$
- **value**: $\mathbf{W}^V$

Implementazione originale

$d = 512$ *Dimensionalità del modello*

$d_k = d_v = 64$

$\mathbf{x}_i$: $(1, d)$

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^Q; \quad \mathbf{k}_i = \mathbf{x}_i\mathbf{W}^K; \quad \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^V$$

$(1, d_k)$ (d, d_k) $(1, d_k)$ (d, d_k) $(1, d_v)$ (d, d_v)

Una vera testa di attenzione: leggermente più complessa

- Date queste tre rappresentazioni di $\mathbf{x}_i$

$$\mathbf{q}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_i = \mathbf{x}_i\mathbf{W}^{\mathbf{V}}$$

- Per calcolare la somiglianza tra l'elemento corrente $\mathbf{x}_i$ e un elemento precedente $\mathbf{x}_j$
- useremo il prodotto scalare tra $\mathbf{q}_i$ e $\mathbf{k}_j$.
- E invece di sommare $\mathbf{x}_j$, sommiamo $\mathbf{v}_j$

Equazioni finali per una testa di attenzione

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{V}}$$

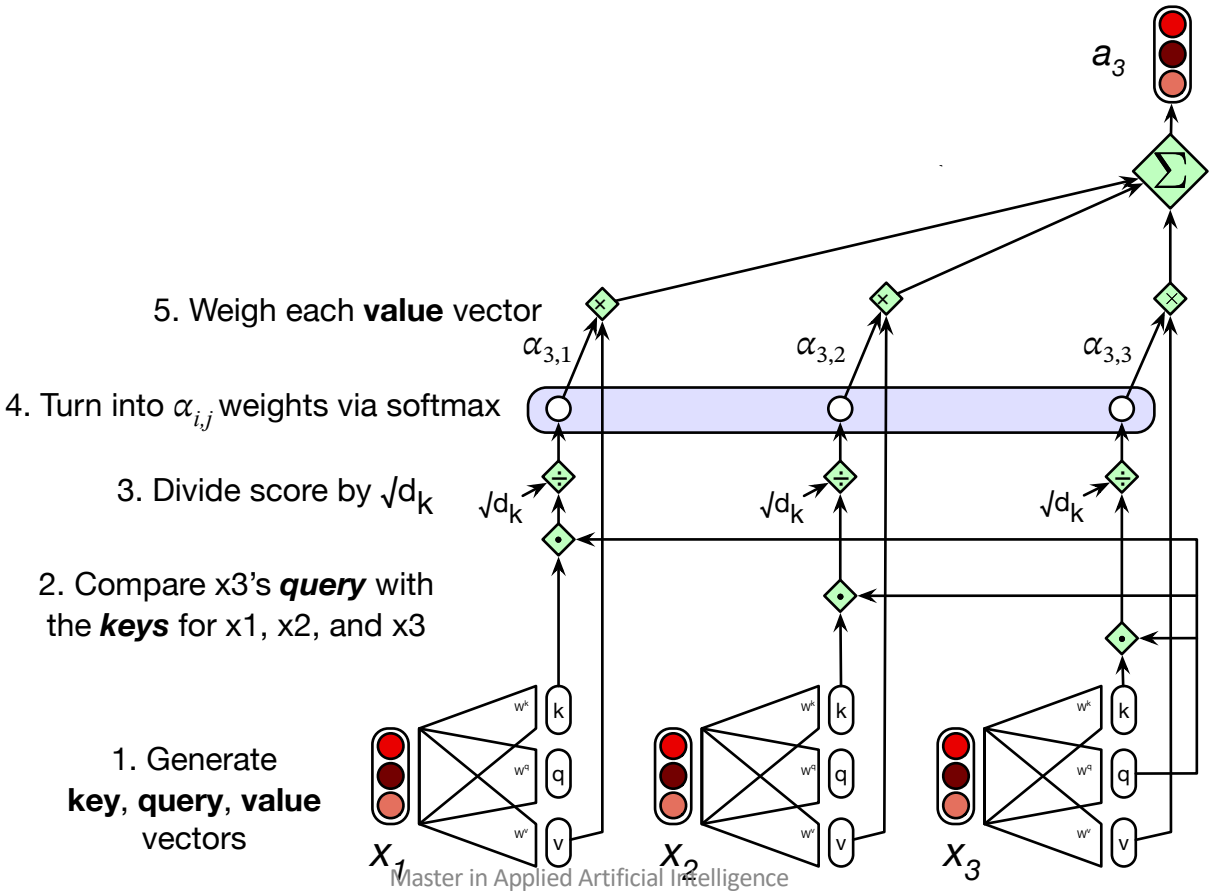
$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

La scalatura previene valori troppo grandi per ilprodotto scalare

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Calcolo del valore di a_3



Una vera attenzione: ancora più complessa

- Invece di una sola testa di attenzione, ne avremo ***molte!***
- *Intuizione*: ogni testa potrebbe ***prestare attenzione al contesto per scopi diversi***, cogliendo relazioni linguistiche o schemi differenti presenti nel contesto.

$$\mathbf{q}_i^c = \mathbf{x}_i \mathbf{W}^{\mathbf{Qc}}; \quad \mathbf{k}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{Kc}}; \quad \mathbf{v}_j^c = \mathbf{x}_j \mathbf{W}^{\mathbf{Vc}}; \quad \forall\, c \quad 1 \le c \le h$$

$$\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}}$$

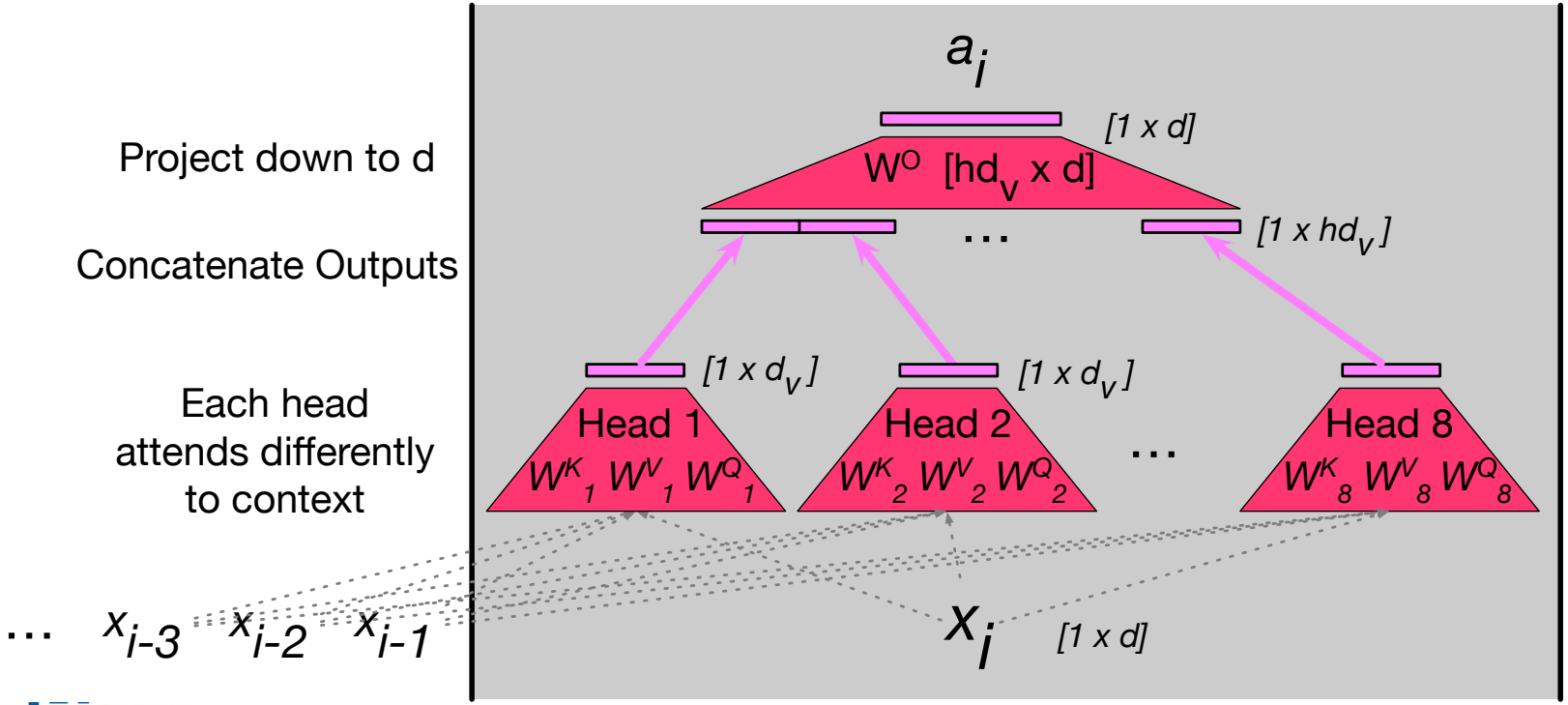
$$\alpha_{ij}^c = \text{softmax}(\text{score}^c(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \le i$$

$$\mathbf{head}_i^c = \sum_{j \le i} \alpha_{ij}^c \mathbf{v}_j^c$$

$$\mathbf{a}_i = (\mathbf{head}^1 \oplus \mathbf{head}^2 \ldots \oplus \mathbf{head}^h)\mathbf{W}^O$$

$$\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \cdots, \mathbf{x}_N]) = \mathbf{a}_i$$

Multi-head self-attention

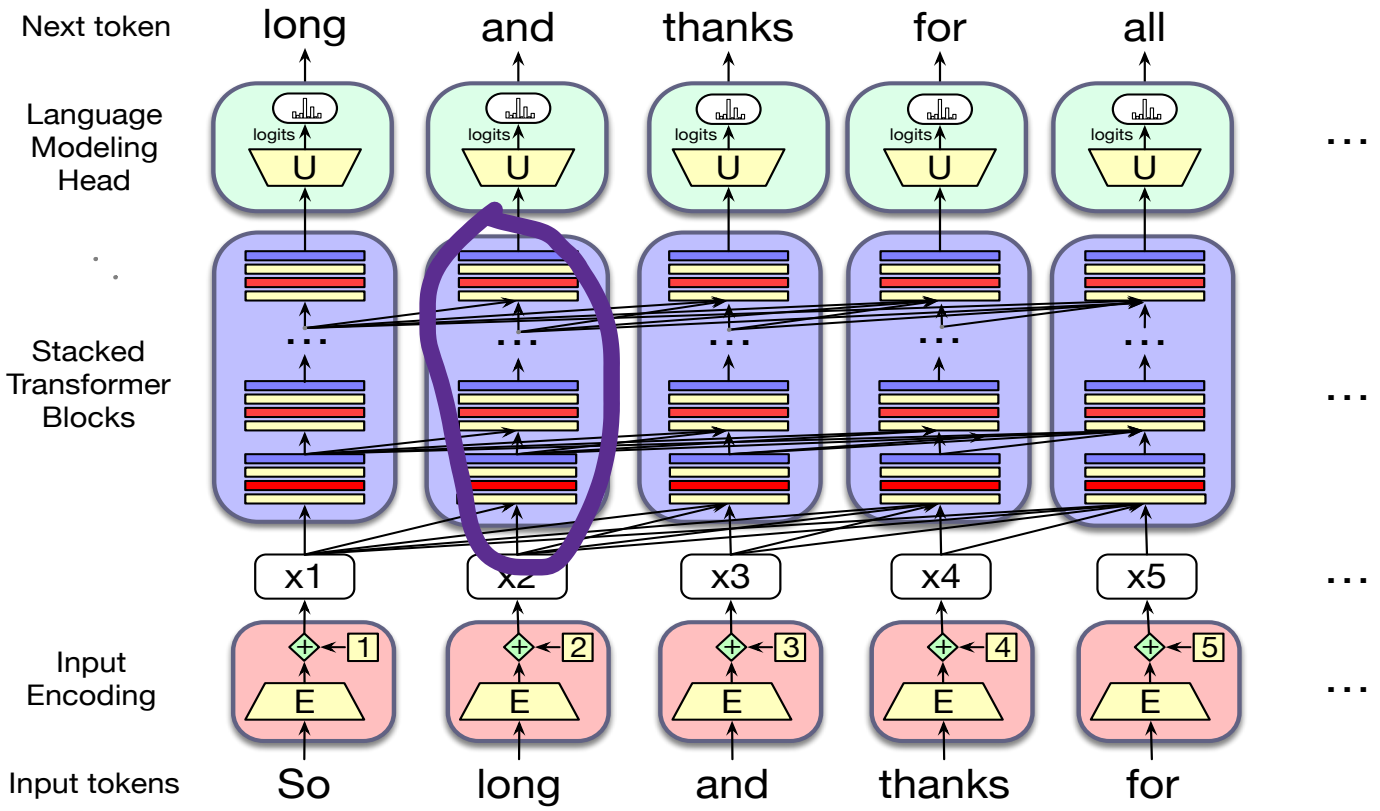


Summary

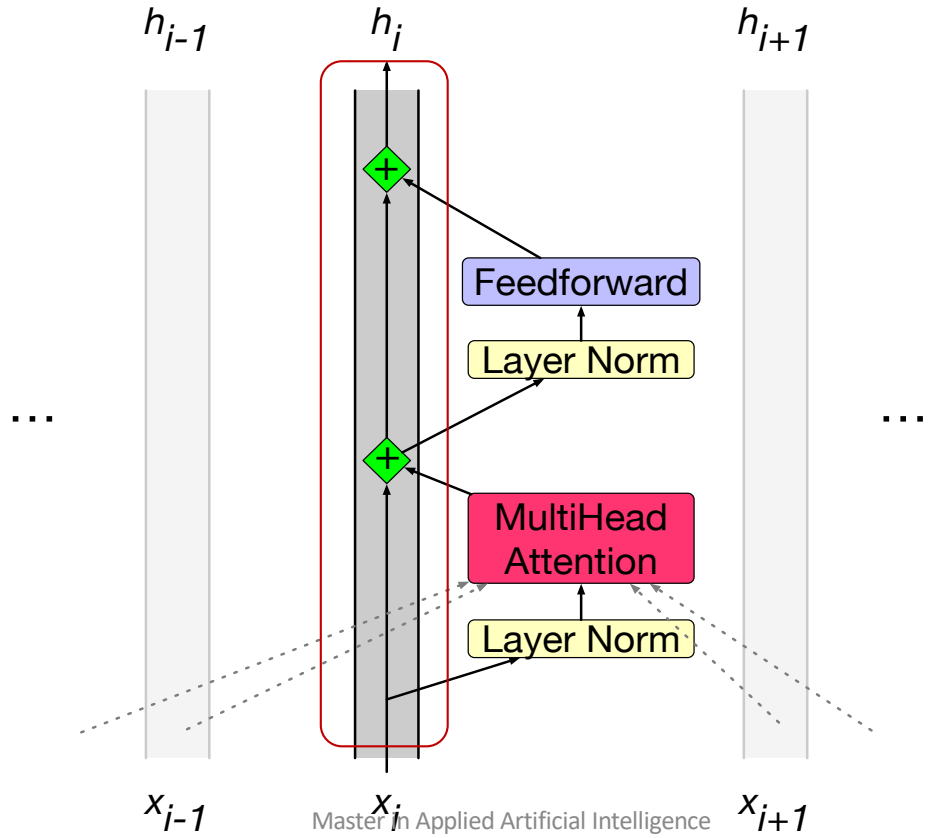
- L'attenzione è un metodo per arricchire la rappresentazione di un token incorporando le informazioni contestuali.
- Il risultato: l'embedding di ogni parola sarà ***diverso a seconda del contesto!***
- Gli embedding contestuali rappresentano quindi il ***significato di una parola nel suo contesto*.**
- vedremo che l'attenzione può anche essere vista come un modo per ***trasferire informazioni da un token a un altro***.

Il blocco Transformer

Ricordiamoci il Transformer LM

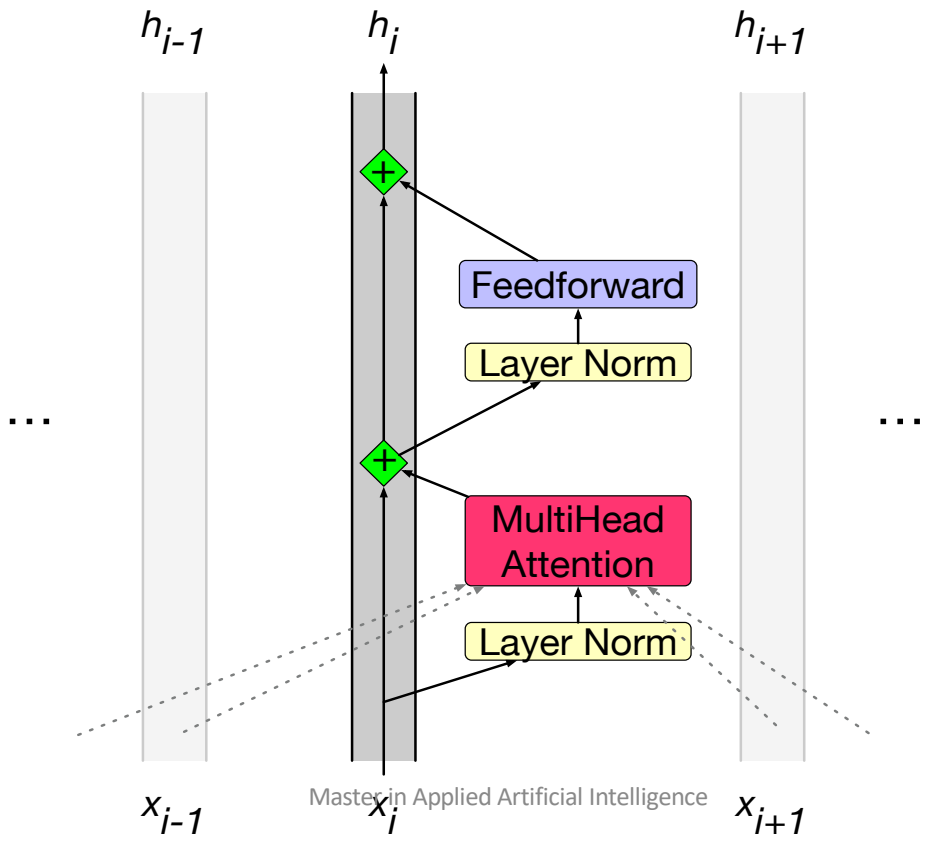


Il “residual stream”: ogni token viene trasmesso ai livelli superiori e modificato

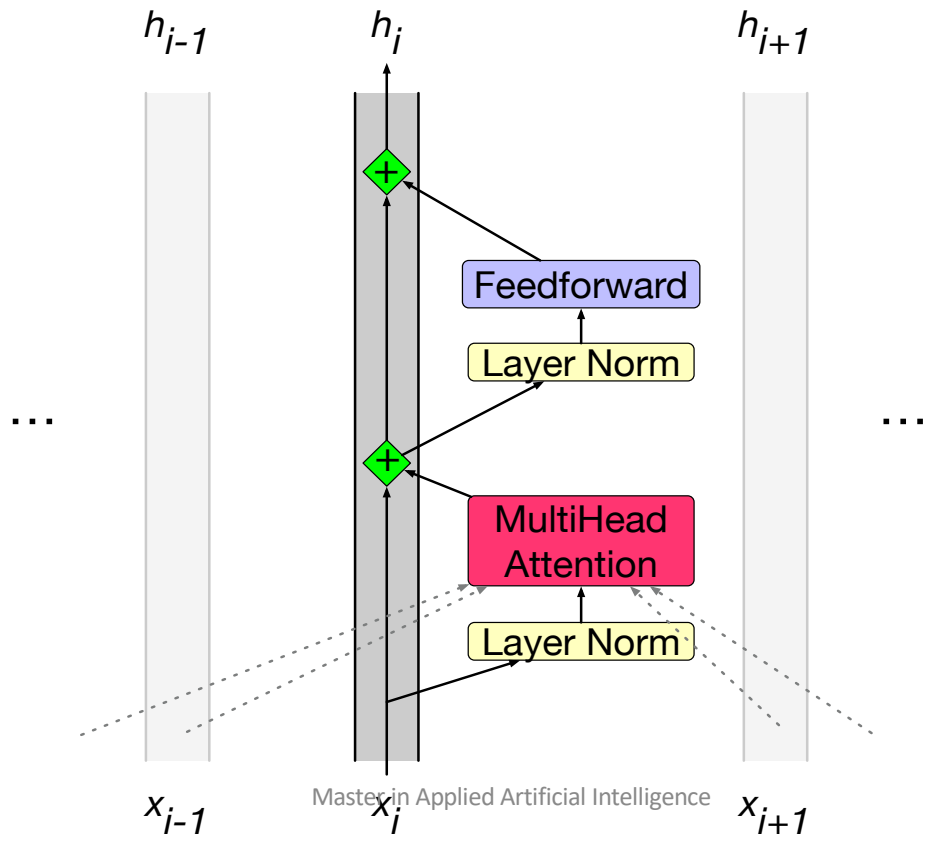


Ci servono non linearità e quindi inseriamo un layer feedforward

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2$$



Layer normalization: il vettore $\mathbf{x}_i$ viene normalizzato due volte



Layer normalization

- La layer normalization è una variazione dello *z-score* usato in statistica, applicata a un singolo vettore in un livello nascosto.

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i$$

Lungo le dimensioni

$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

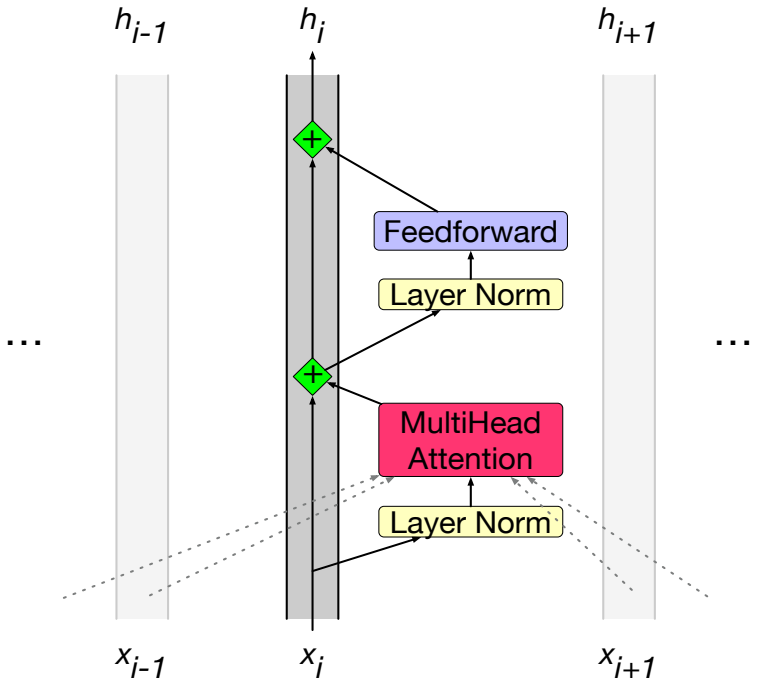
$$\sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

Per ogni vettore di embedding

$$\text{LayerNorm}(\mathbf{x}) = \gamma \frac{(\mathbf{x} - \mu)}{\sigma} + \beta$$

Le operazioni sono elemento per elemento

Combiniamo tutti gli elementi in un singolo blocco del transformer.

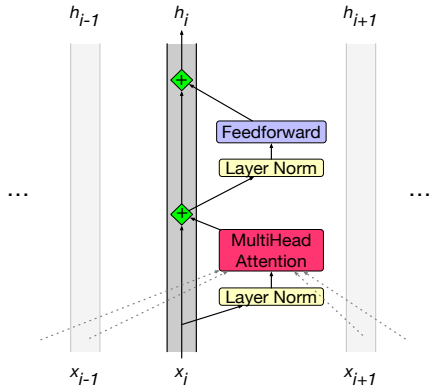


$$\mathbf{t}_i^1 = \text{LayerNorm}(\mathbf{x}_i)$$

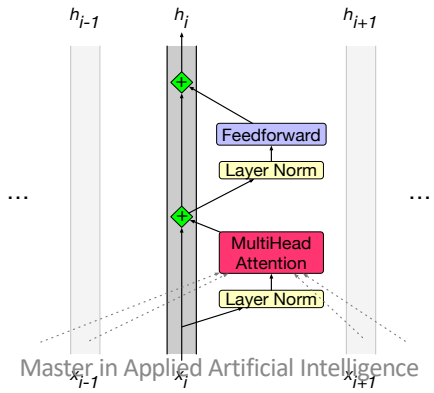
$$\mathbf{t}_i^3 = \mathbf{t}_i^2 + \mathbf{x}_i$$
$$\mathbf{t}_i^4 = \text{LayerNorm}(\mathbf{t}_i^3)$$
$$\mathbf{t}_i^5 = \text{FFN}(\mathbf{t}_i^4)$$
$$\mathbf{h}_i = \mathbf{t}_i^5 + \mathbf{t}_i^3$$

Un transformer è una struttura composta da uno stack di questi blocchi

Block 2



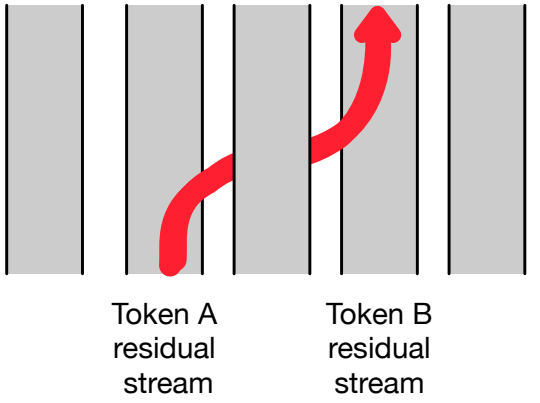
Block 1



tutti i vettori hanno la stessa dimensionalità *d*

Stream residuali e attenzione

- Si noti che tutte le parti del blocco del transformer si applicano a un unico flusso residuo (cioè un token)
- *Ad eccezione del meccanismo di attenzione*, che utilizza informazioni provenienti da altri token.
- Elhage et al. (2021) mostrano che è possibile interpretare le teste di attenzione come elementi che spostano letteralmente informazioni dal flusso residuo di un token vicino verso il flusso corrente.



Parallelizzazione del flusso di attenzione

Parallelizzazione del calcolo usando **X**

- Nel blocco di attenzione/transformer, finora abbiamo calcolato un singolo output in un singolo passo temporale *i* all'interno di un unico flusso residuale.
- Ma possiamo "impacchettare" *N* token della sequenza di ingresso in una singola matrice **X** di dimensione [$N \times d$].
- Ogni riga di **X** è l'embedding di un token di input.
- **X** può avere da 1K a 32K righe, ognuna pari alla dimensionalità degli embedding *d*

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \;\; \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \;\; \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

QK^T

- Possiamo fare un'unica moltiplicazione di matrice per combinare Q and K^T

N

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

N

Parallelizzare l'attenzione

- Scaliamo gli score, eseguiamo la softmax e moltiplichiamo il risultato per **V** ottenendo una matrice di dimensione $N \times d$
  - Un vettore attenzione per ogni token di input

$$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

Mascheriamo il futuro

$$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

- A che serve questa funzione di mascheratura?
- $\mathbf{QK}^T$ ha uno scor per ogni prodotto query-chiave, *inclusi quelli che seguono la query*.
- Indovinare la prossima parola è veramente semplice se già la conosciamo!

Mascheriamo il futuro

$$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

- Aggiungiamo $-\infty$ alle celle del triangolo superiore
- La softmax calcolerà 0 in questi punti

q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

N

Un’altro punto: l’attenzione è quadratica

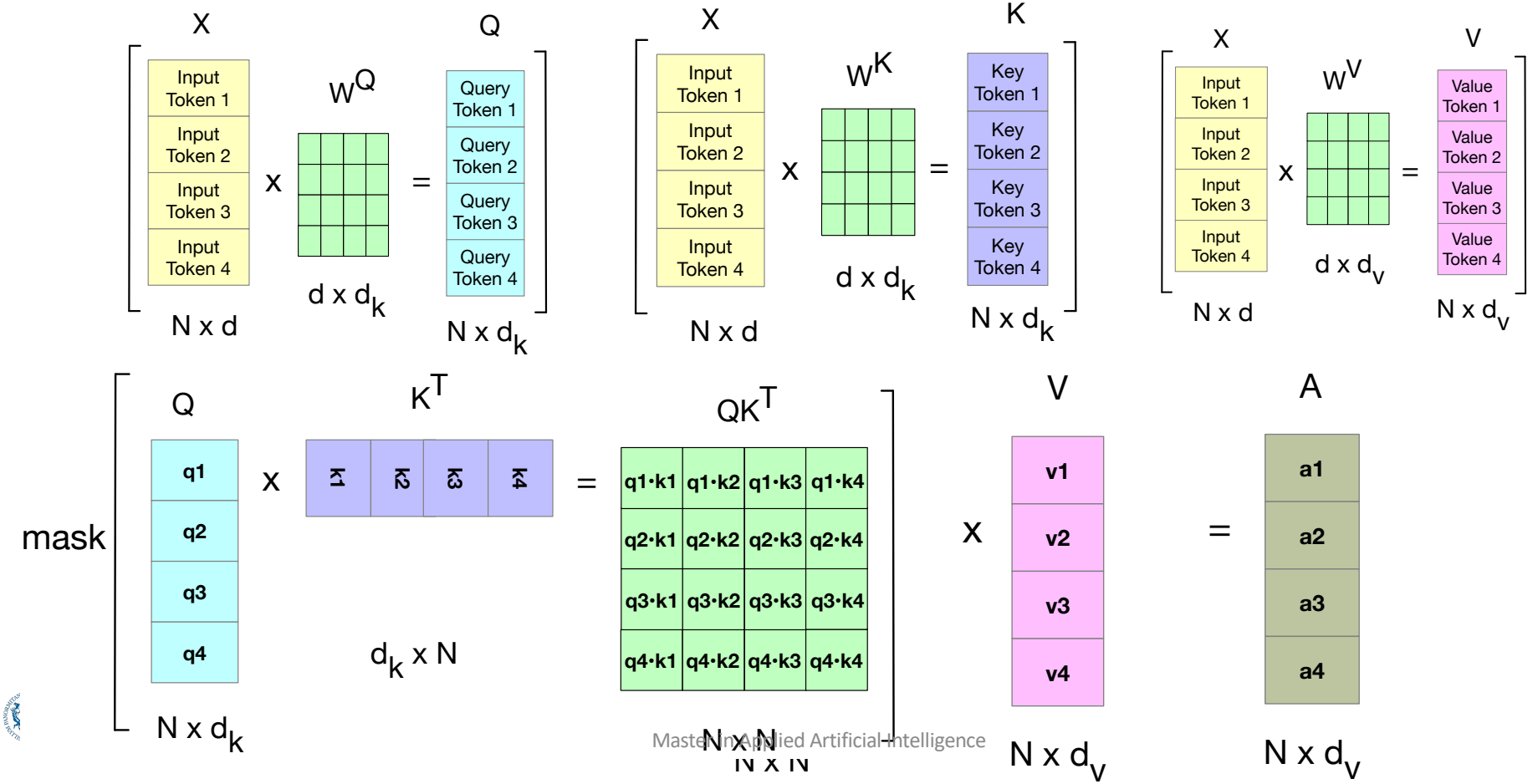
$$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{Q}\mathbf{K}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V}$$

N

q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

Flusso dei tensori nell'attenzione



Parallelizzazione della Multi-head Attention

$$\mathbf{Q^i} = \mathbf{X}\mathbf{W^{Qi}}; \quad \mathbf{K^i} = \mathbf{X}\mathbf{W^{Ki}}; \quad \mathbf{V^i} = \mathbf{X}\mathbf{W^{Vi}}$$

$$\mathbf{head}_i = \text{SelfAttention}(\mathbf{Q^i}, \mathbf{K^i}, \mathbf{V^i}) = \text{softmax}\left(\frac{\mathbf{Q^i}\mathbf{K^{iT}}}{\sqrt{d_k}}\right)\mathbf{V^i}$$

$$\text{MultiHeadAttention}(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W^O}$$

Parallelizzazione della Multi-head Attention

$$\mathbf{O} = \text{LayerNorm}(\mathbf{X} + \text{MultiHeadAttention}(\mathbf{X}))$$
$$\mathbf{H} = \text{LayerNorm}(\mathbf{O} + \text{FFN}(\mathbf{O}))$$

- or

$$\mathbf{T}^1 = \text{MultiHeadAttention}(\mathbf{X})$$
$$\mathbf{T}^2 = \mathbf{X} + \mathbf{T}^1$$
$$\mathbf{T}^3 = \text{LayerNorm}(\mathbf{T}^2)$$
$$\mathbf{T}^4 = \text{FFN}(\mathbf{T}^3)$$
$$\mathbf{T}^5 = \mathbf{T}^4 + \mathbf{T}^3$$
$$\mathbf{H} = \text{LayerNorm}(\mathbf{T}^5)$$

Position embeddings e la testa di LM

Token e Position Embedding

- La matrice **X** (di dimensione $[N \times d]$) contiene un embedding per ogni parola nel contesto.
- Questo embedding viene creato ***sommando due embedding distinti*** per ciascun input:
- token embedding
- Embedding posizionali

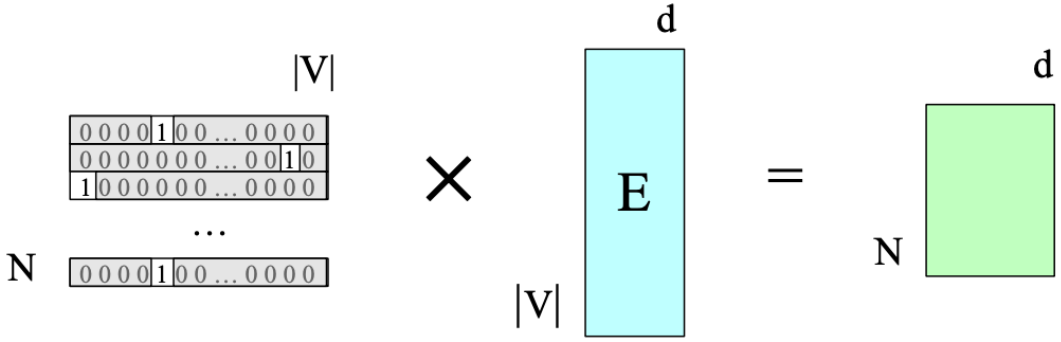
Token Embedding

- La matrice degli embedding **E** ha forma [|V | × d].
- Una riga per ognuno dei |V | token del vocabolario.
- Ogni parola è un vettore riga d-dimensionale

- Data la stringa "*Thanks for all the*"
- *1.* tokenizza con BPE e converti in iindici del vocabolario
- **w** = [5,4000,10532,2224]
- 2. Seleziona le righe corrispondanti da **E**, essendo ognuna un embedding
- (row 5, row 4000, row 10532, row 2224).

Embedding posizionali

- Possiamo rappresentare l'intera sequenza di token come una matrice di one-hot vector, uno per ciascuna delle N posizioni nella context window del transformer.

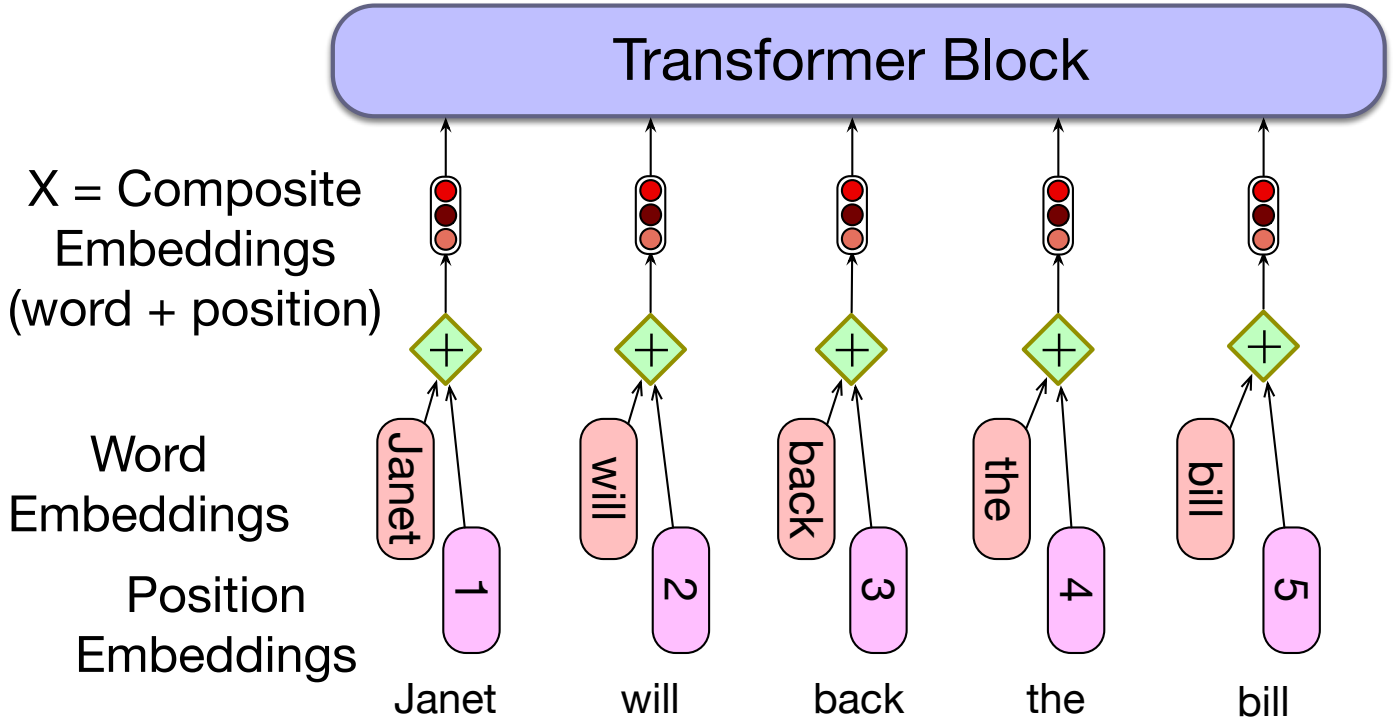


- Questi embedding *non dipendono* dalla posizione

Embedding posizionali

- Ci sono molti modi, ma noi descriveremo il più semplice: la posizione assoluta.
- Goal: apprendere una matrice di embedding posizionali E_{pos} di dimensione $[1 \times N]$.
- Partiamo con embedding inizializzati casualmente, uno per ogni intero fino ad una lunghezza massima.
- Cioè così come abbiamo un ambedding per la parola *fish*, avremo un embedding per la posizione 3 oppure la 17.
- Come per i word embedding, I position embedding vengono appreso durante l'addestramento come gli altri paramateri.

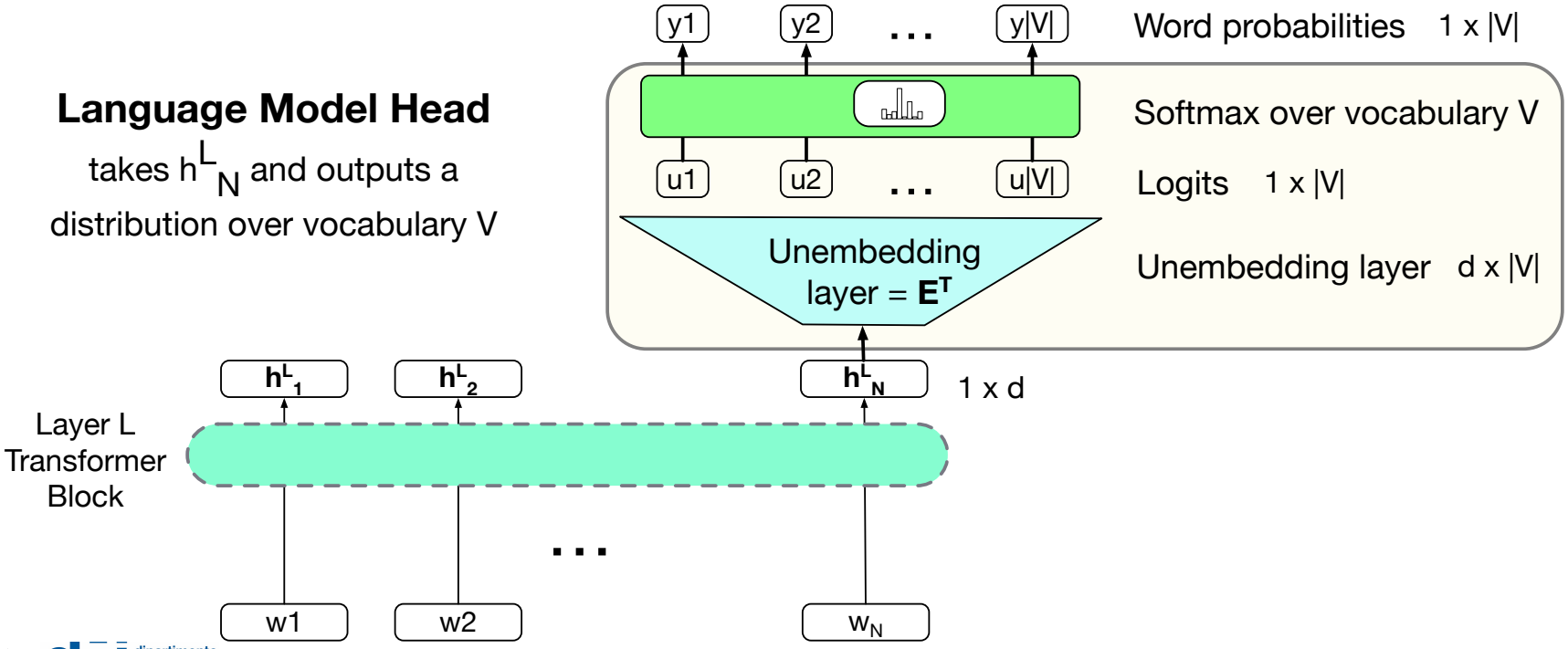
Sommiamo word e position embedding



Testa di LM

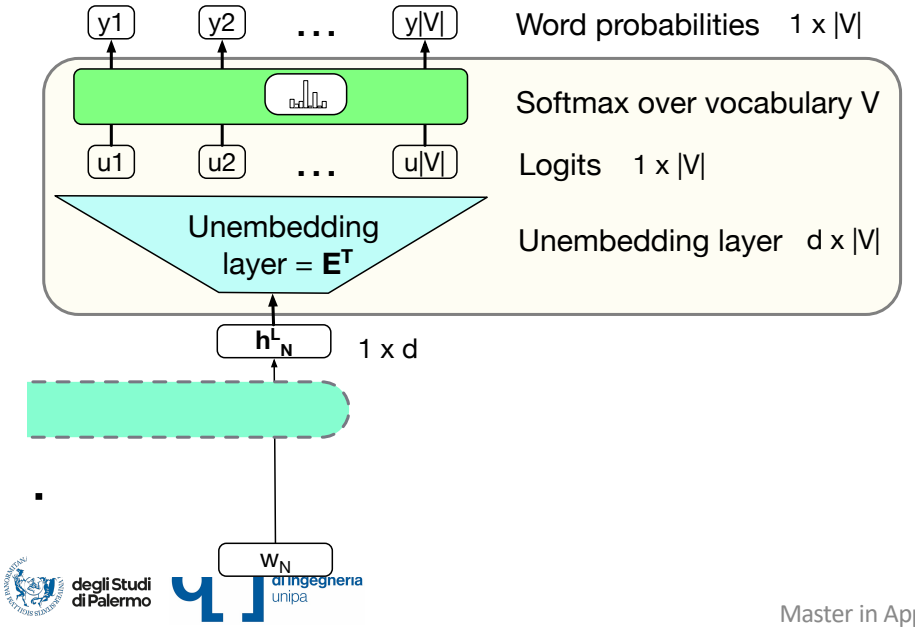
Language Model Head

takes h^L_N and outputs a distribution over vocabulary V



Testa di LM

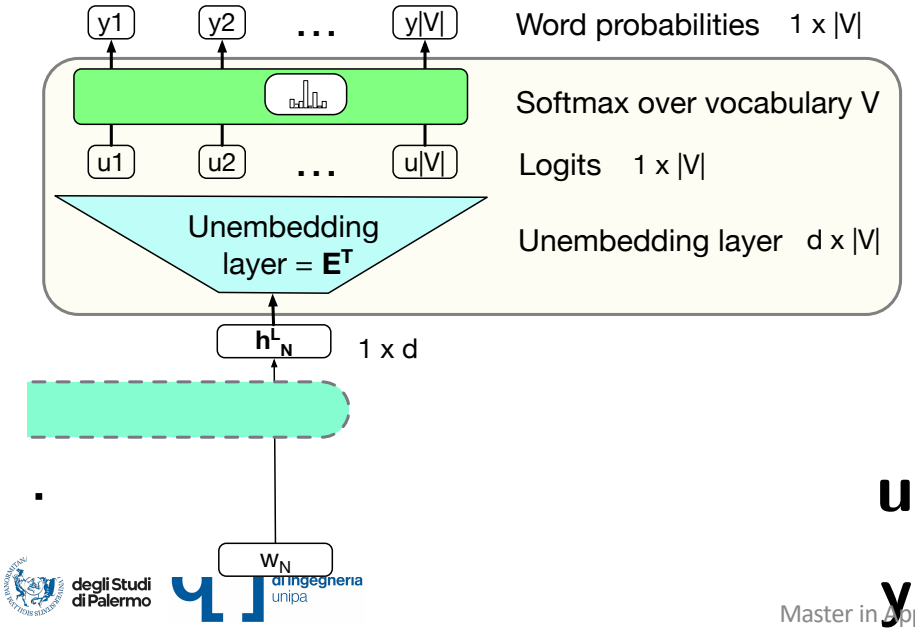
Unembedding layer: layer lineare che proietta h^L_N (di dimensione $[1 \times d]$) in un vettore di logit di dimensione 1 x $|V|$



Perché "unembedding"? **Weight Tiyng** con $\mathbf{E^T}$

Usiamo gli stessi pesi per le due matrici

Testa di LM



I logit sono il vettore degli score **u**

Uno score per ognuna delle $|V|$ possibili parole nel vocabolario V. Dimensione $1 \times |V|$.

La softmax trasforma I logit in probabilità sul vocabolario. Shape $1 \times |V|$.

$$\mathbf{u} = \mathbf{h}_N^L \mathbf{E}^T$$

$$\mathbf{y} = \text{softmax}(\mathbf{u})$$

Il modello Transformer finale