



Intelligenza Artificiale

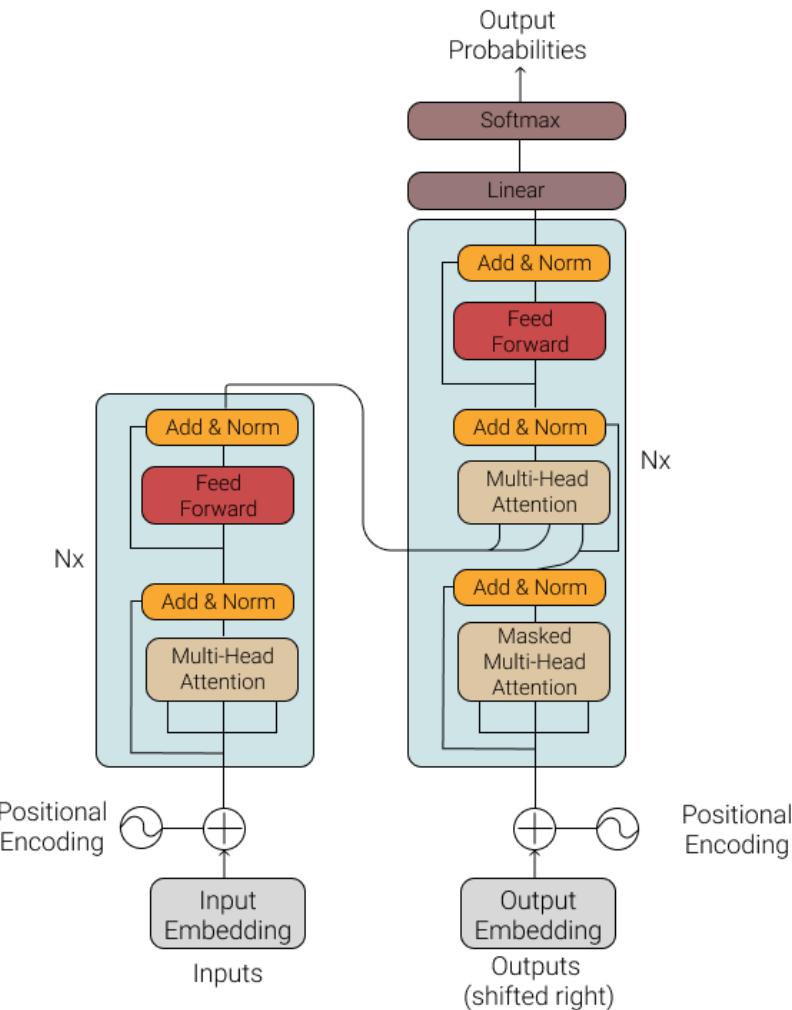
**Encoder-Decoder
BERT-GPT2-T5**



Outline

- ▶ Introduzione
- ▶ Tokenizzazione
- ▶ Encoder-Decoder Framework
- ▶ Architettura Transformer
- ▶ Transfer Learning
 - ▶ BERT
 - ▶ GPT-2
 - ▶ T5

Introduzione



▶ Nel 2017, i ricercatori di Google hanno pubblicato un paper in cui proponevano una nuova architettura di rete neurale per la modellazione di sequenze: **Transformer**

Introduzione (2)

- ▶ Tale architettura utilizza l'**attention** per aumentare la velocità con cui i modelli possono essere addestrati
- ▶ Inoltre, combinando l'architettura Transformer con tecniche di apprendimento non supervisionato, si elimina la necessità di addestrare un modello da zero

Tokenizzazione

- ▶ I modelli basati su Transformer non possono ricevere come input stringhe grezze, ma presuppongono che il testo sia stato tokenizzato e codificato in vettori numerici
- ▶ La tokenizzazione è il processo di suddivisione di una stringa in unità atomiche, denominate **token**
 - ▶ Character-level
 - ▶ Word-level
 - ▶ Subword-level

Character-level

- ▶ Suddivisione di una stringa in una lista di singoli caratteri

```
text = "Tokenizing text is a core task of NLP."
tokenized_text = list(text)
print(tokenized_text)

['T', 'o', 'k', 'e', ' ', 't', 'e', 'x', 't', ' ', 'i', 's', ' ', 'a', ' ', 'c', 'o', 'r', 'e', ' ', 't', 'a', 's', ' ', 'k', ' ', 'o', 'f', ' ', 'N', 'L', 'P', '.']
```

- ▶ Numericalization
- ▶ One-hot encoding

```
Token: T
Tensor index: 5
One-hot: tensor([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

- ▶ Vantaggio
 - ▶ Utile in presenza di errori ortografici e parole rare
- ▶ Svantaggio
 - ▶ Ignora qualsiasi struttura del testo e tratta l'intera stringa come un flusso di caratteri

Word-level

- ▶ Piuttosto che suddividere il testo in caratteri, esso viene suddiviso in parole, ciascuna delle quali viene mappata in un intero
 - ['Tokenizing', 'text', 'is', 'a', 'core', 'task', 'of', 'NLP.']
- ▶ Numericalization - One-hot encoding
- ▶ Vantaggio
 - ▶ L'utilizzo di parole fin dall'inizio permette ad un modello di evitare l'apprendimento delle parole dai caratteri, riducendo così la complessità del processo di addestramento
- ▶ Svantaggio
 - ▶ La dimensione del vocabolario può essere molto grande (a causa di errori ortografici o parole rare)

Subword-level

- ▶ Un buon compromesso tra la tokenizzazione character-level e word-level
 - ▶ Combina i migliori aspetti dei due approcci
- ▶ Suddivide parole rare in unità più piccole per permettere al modello di considerare le parole frequenti come entità uniche e quindi di limitare la dimensione degli input
- ▶ Viene definita in base al corpus ed è solitamente basata su regole statistiche e algoritmi
- ▶ Byte Pair Encoding (BPE) – Approccio Greedy
 - ▶ Esamina la frequenza combinata di ciascuna coppia di subtoken (bytes) e considera quella con la frequenza più alta

BPE

```
"hug", "pug", "pun", "bun", "hugs"
```

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]

Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)

Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"]

Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)

Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]

Corpus: ("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)

BPE (2)

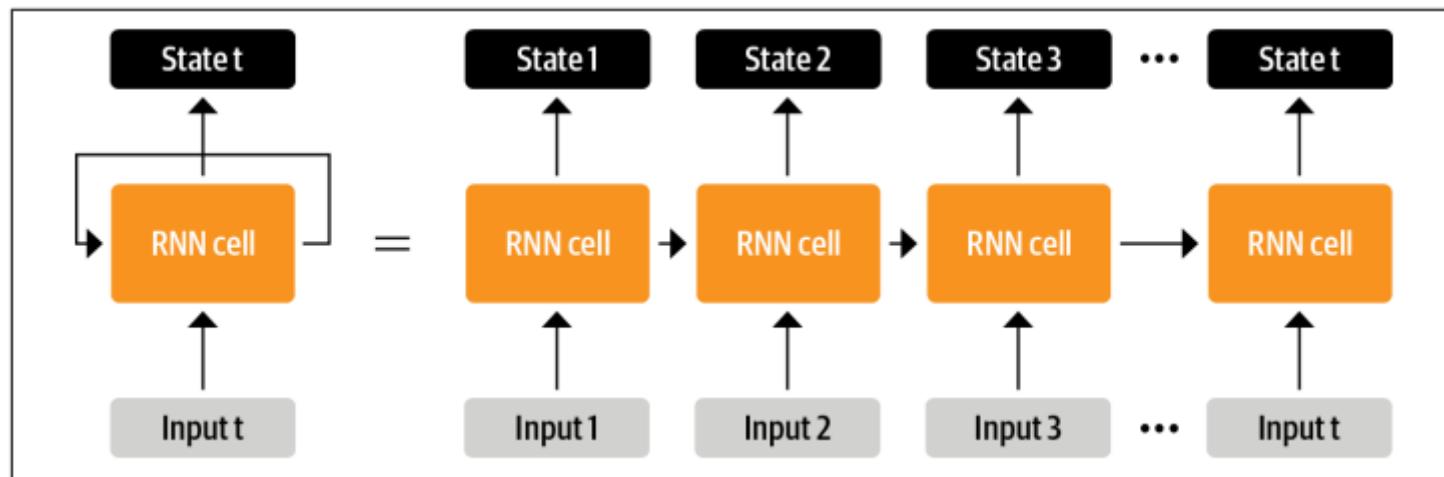
Number	Token	Frequency
1	li	3
2	l	5
3	ea	2
4	eb	4
5	near	6
6	bra	2
7	al	4
8	n	5
9	r	1
10	ge	11
11	g	7
12	e	8

linear = **li** + **near** *or* **li** + **n** + **ea** + **r**

algebra = **al** + **ge** + **bra** *or* **al** + **g** + **e** + **bra**

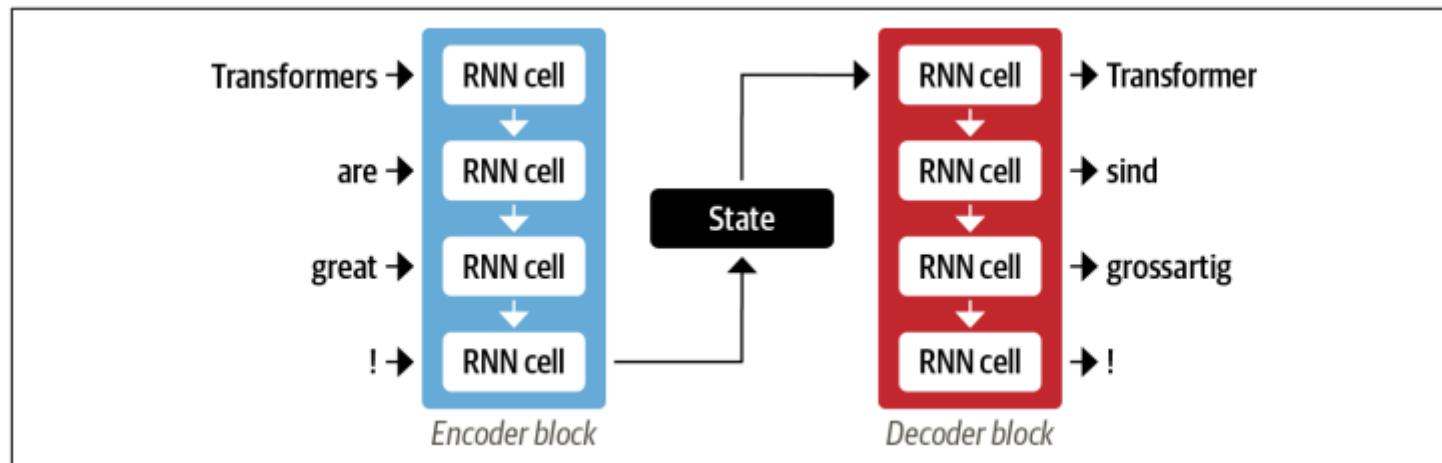
Encoder-Decoder Framework

- ▶ Queste architetture contengono un ciclo di feedback nelle connessioni di rete che consente alle informazioni di propagarsi da uno step all'altro, rendendole ideali per la modellazione di dati sequenziali come i testi



Encoder-Decoder Framework (2)

- ▶ Il compito dell'encoder è quello di codificare le informazioni della sequenza di input in una rappresentazione numerica che spesso viene denominata **last hidden state**
- ▶ Questo stato viene poi passato al decoder, che genera la sequenza di output

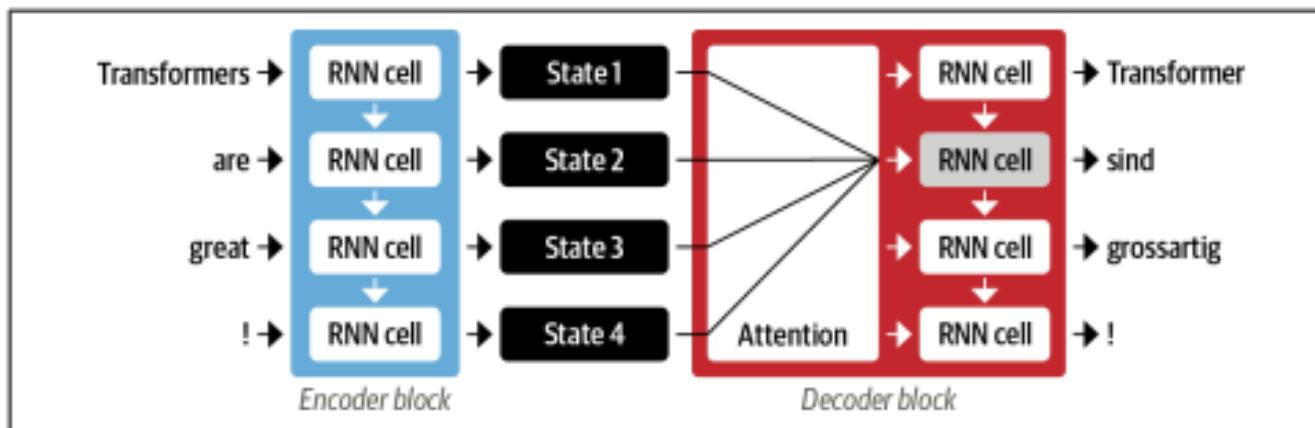


Encoder-Decoder Framework (3)

- ▶ Sebbene sia elegante nella sua semplicità, una debolezza di questa architettura è che il last hidden state dell'encoder crea un **collo di bottiglia dell'informazione**
- ▶ Esso deve rappresentare il significato dell'intera sequenza di input perché è tutto ciò a cui il decoder ha accesso quando genera l'output
- ▶ È possibile superare questo problema consentendo al decoder di accedere a tutti gli hidden state dell'encoder

Attention

- ▶ L'Attention è un meccanismo che permette alle reti neurali di assegnare un peso differente o 'attenzione' a ciascun elemento in una sequenza
- ▶ Per sequenze di testo, gli elementi sono gli embedding dei token. Ovvero, rappresentazioni vettoriali a dimensione fissata

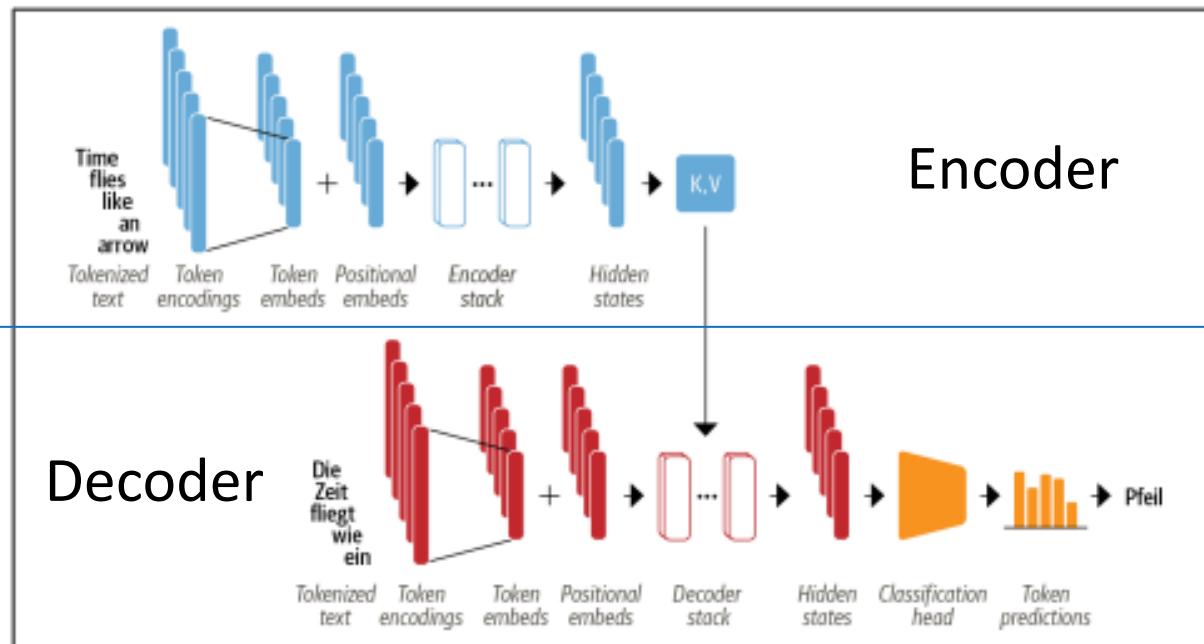


Attention (2)

- ▶ Sebbene l'attenzione abbia permesso di ottenere ottime performance su molti task di NLP, c'è ancora un grosso difetto nell'utilizzo di modelli ricorrenti per la codifica e la decodifica
- ▶ I calcoli sono intrinsecamente sequenziali e non possono essere parallelizzati sulla sequenza

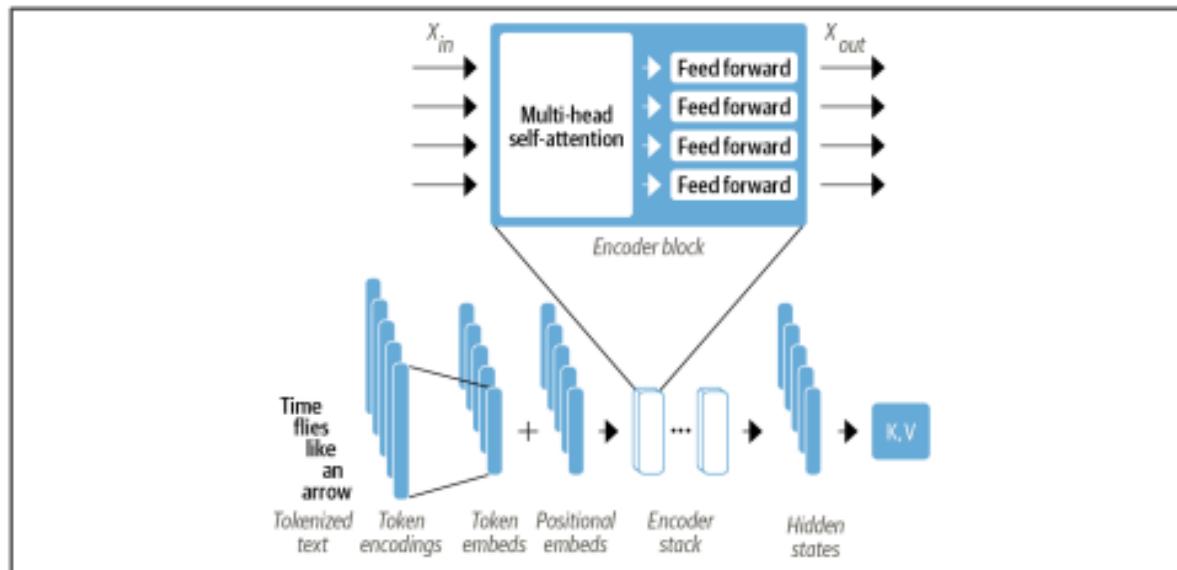
Architettura Transformer

- ▶ Tale architettura è composta da due componenti:
 - ▶ **Encoder:** Converte la sequenza dei token di input in una sequenza di vettori di embedding, spesso denominata ‘hidden states’
 - ▶ **Decoder:** Utilizza l’output dell’encoder per generare iterativamente una sequenza di token di output, uno ad ogni step



Encoder

- ▶ L'encoder è composto da uno stack di **encoder layer** disposti sequenzialmente
- ▶ Ogni encoder layer riceve una sequenza di embedding che fornisce in input ai seguenti sublayer:
 - ▶ Multi-head self-attention layer
 - ▶ Fully connected feed-forward layer

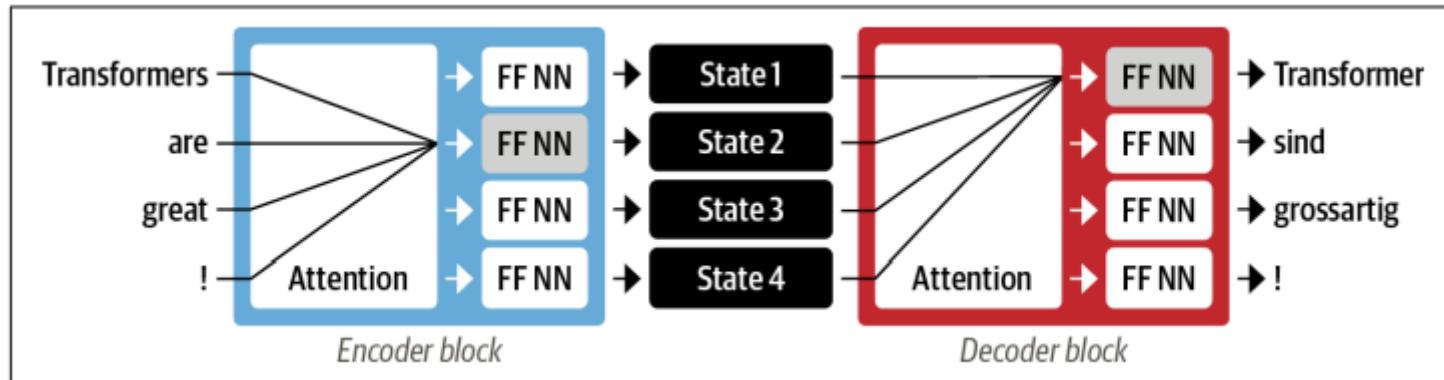


Encoder (2)

- ▶ Il ruolo principale dello stack di encoder layer è quello di ‘aggiornare’ gli embedding di input al fine di ottenere rappresentazioni che codificano alcune informazioni contestuali
- ▶ **Self-attention Layer**

Self-Attention

- ▶ La parte ‘**Self**’ del meccanismo di Self-Attention si riferisce al fatto che i pesi vengono calcolati per tutti gli hidden state dello stesso insieme, ad esempio quelli dell’encoder
- ▶ Al contrario, il meccanismo di Attention associato ai modelli ricorrenti prevede il calcolo della rilevanza di ogni hidden state dell’encoder rispetto a quello del decoder in un determinato time step di decodifica



Self-Attention (2)

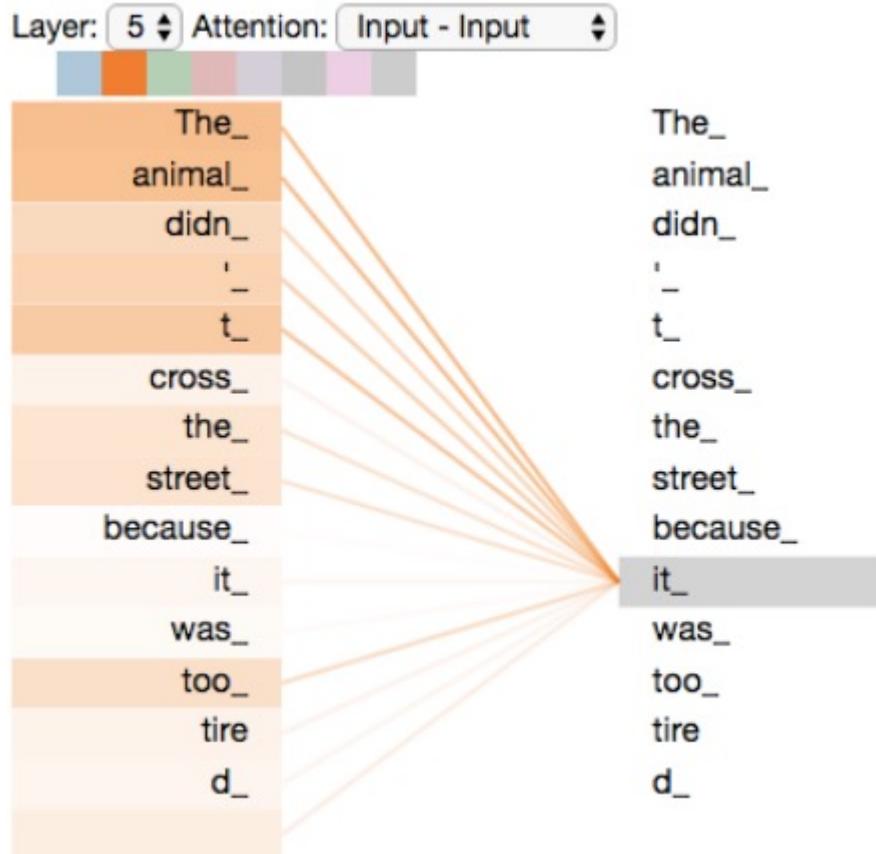
- ▶ L'idea principale che sta alla base è che, invece di usare un embedding fisso per ogni token, possiamo usare l'intera sequenza per calcolare una **media ponderata** di ogni embedding
- ▶ In altre parole, data una sequenza di embedding di token x_1, \dots, x_n , il meccanismo produce una nuova sequenza x'_1, \dots, x'_n , dove ogni x'_i è una combinazione lineare di tutte le rappresentazioni x_j :

$$x'_i = \sum_{j=1}^n w_{ji} x_j$$

- ▶ I coefficienti w_{ji} vengono definiti '**attention weights**' e sono normalizzati in modo tale che $\sum_j w_{ji} = 1$.
- ▶ Invece, gli embedding generati in questo modo vengono definiti '**contextualized embeddings**'

Self-Attention (3)

"The animal didn't cross the street because it was too tired"

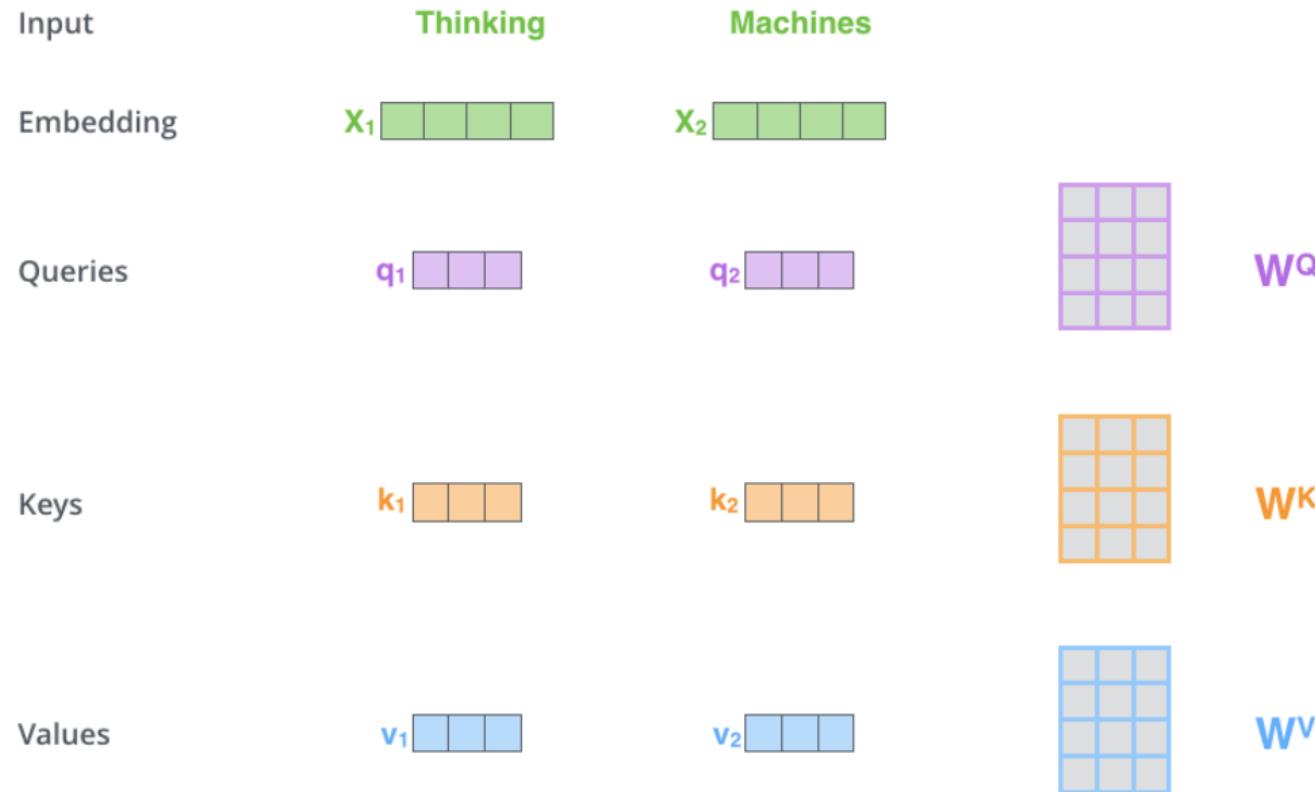


- ▶ A cosa si riferisce 'it' in questa frase? Alla strada o all'animale?

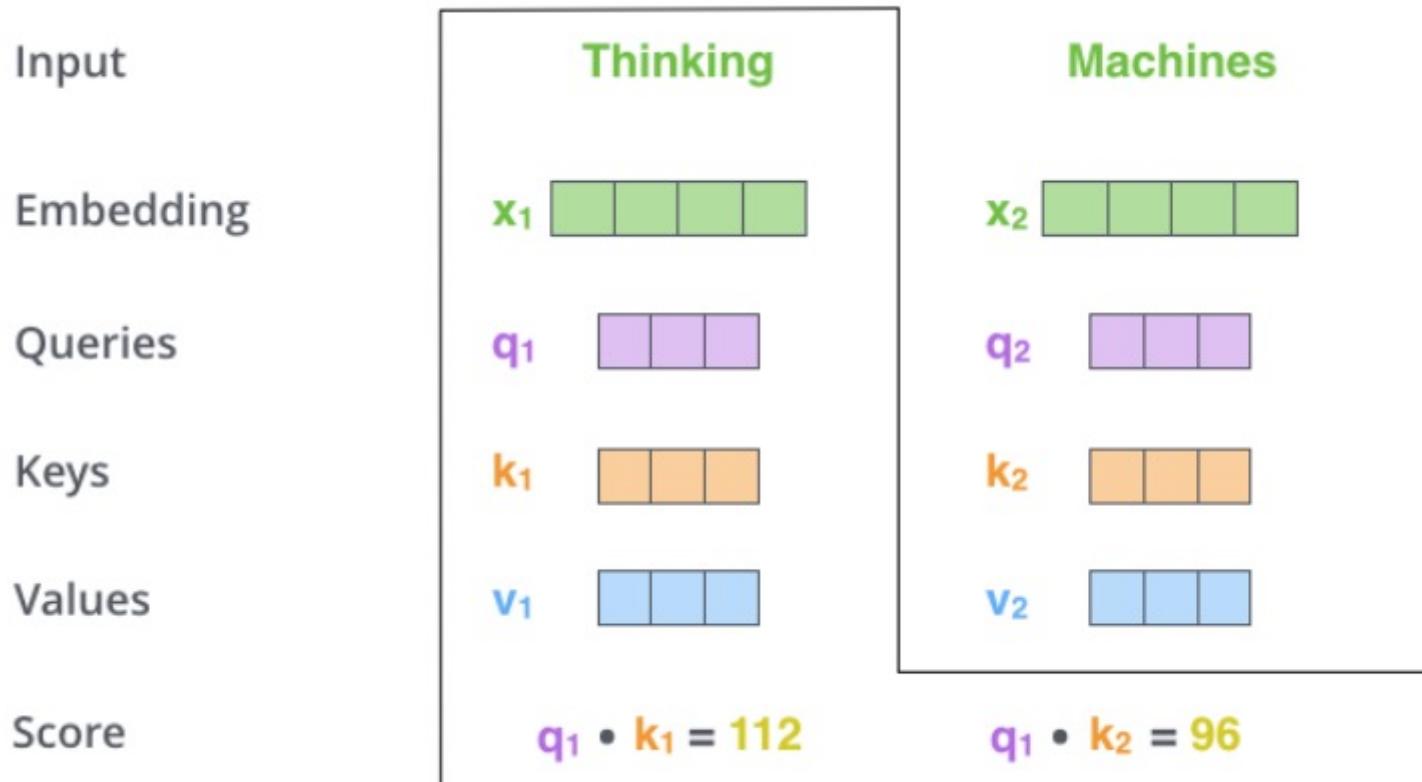
Scaled dot-product attention

- ▶ Esistono diversi modi per implementare il self-attention layer, ma il più comune è lo ‘scaled dot-product attention’
- ▶ Per implementare questo meccanismo sono necessari quattro passaggi principali:
 1. Proiettare ogni embedding di input in tre vettori denominati **Query, Chiave e Valore**
 2. Calcolare gli ‘**attention score**’
 - ▶ Ogni valore viene calcolato eseguendo il **prodotto scalare** tra il vettore Query della parola che stiamo valutando e il vettore Chiave di tutte le altre parole della sequenza. Quindi, se stiamo elaborando la parola in posizione #1 , il primo punteggio è dato dal prodotto scalare tra q_1 e k_1 . Il secondo punteggio sarebbe il prodotto scalare di q_1 e k_2 , ecc.
 3. Calcolare gli ‘**attention weights**’
 4. Aggiornare gli embedding dei token
 - ▶ Tale operazione viene eseguita moltiplicando gli attention weights con i vettori Valore e sommando le rappresentazioni risultanti ($x'_i = \sum_j w_{ji} v_j$)

Scaled dot-product attention (2)



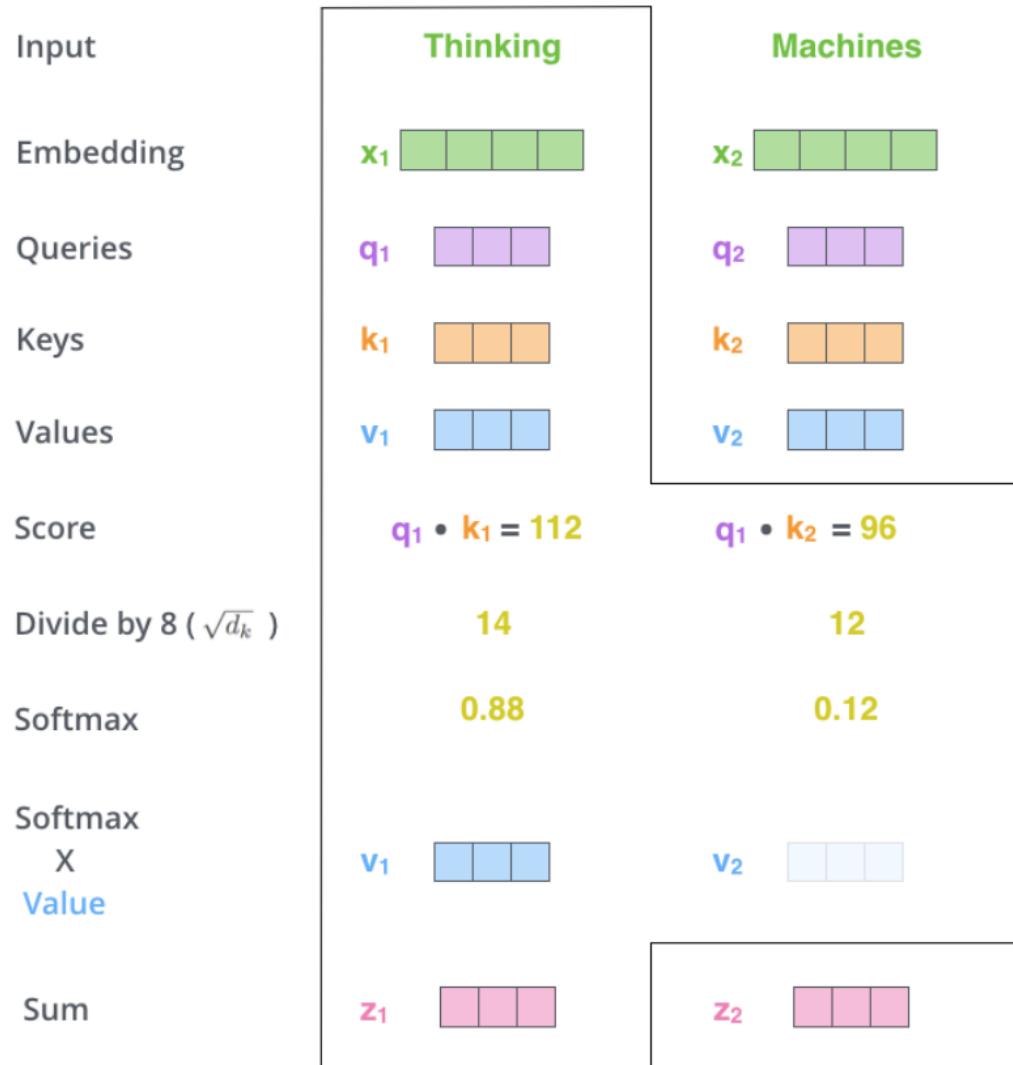
Scaled dot-product attention (3)



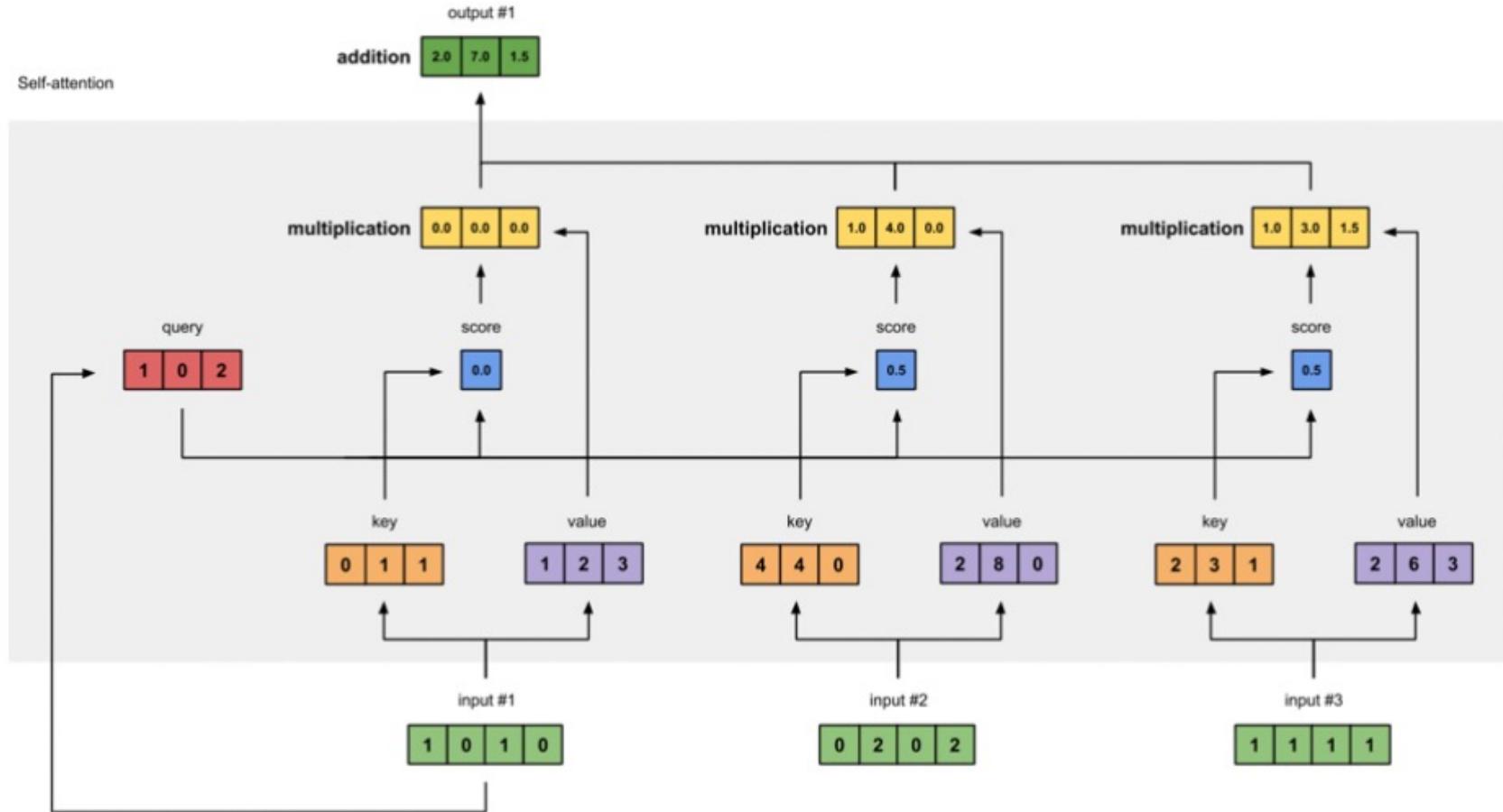
Scaled dot-product attention (4)

Input	Thinking	Machines
Embedding	x_1	x_2
Queries	q_1	q_2
Keys	k_1	k_2
Values	v_1	v_2
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12

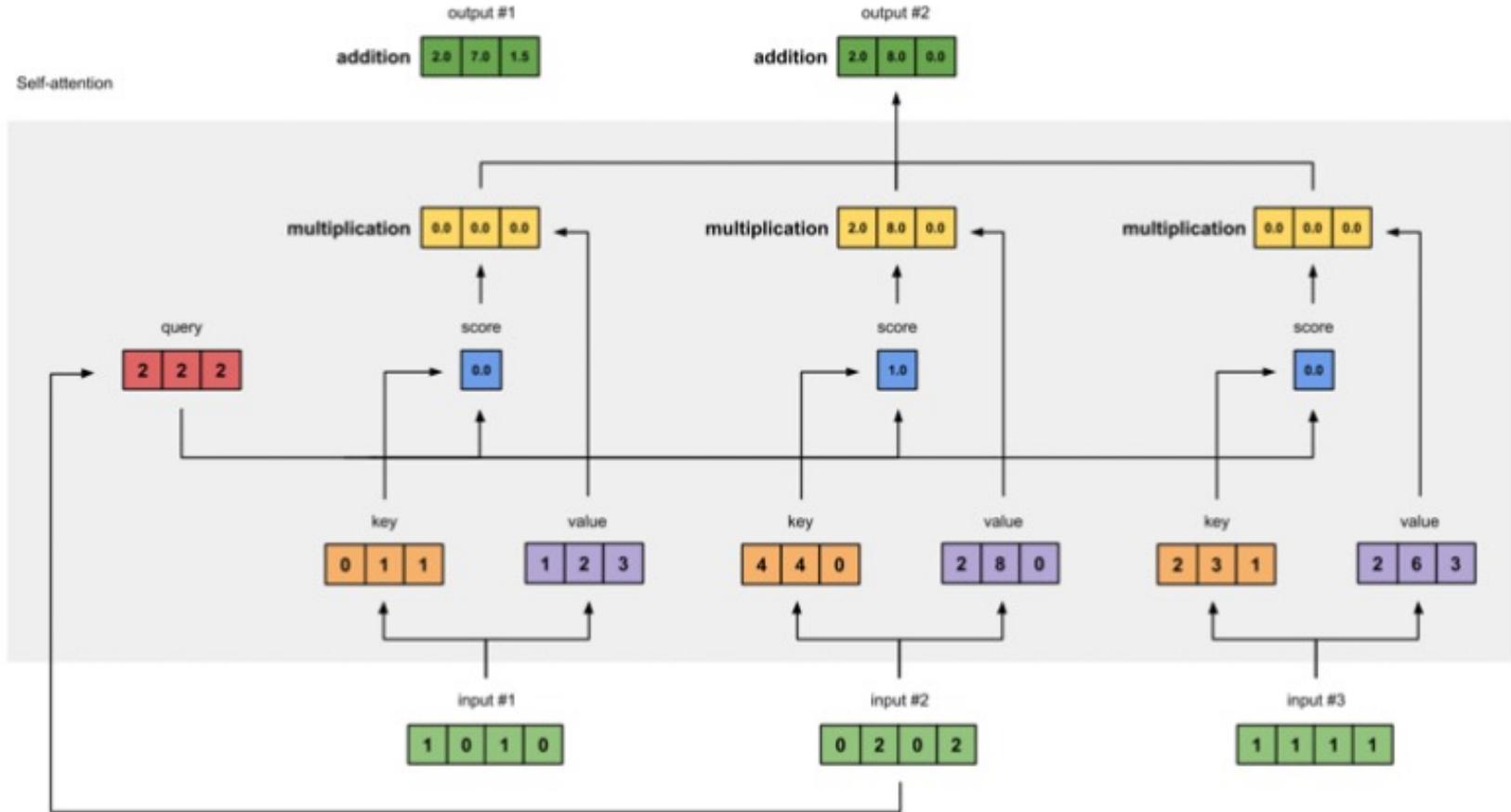
Scaled dot-product attention (5)



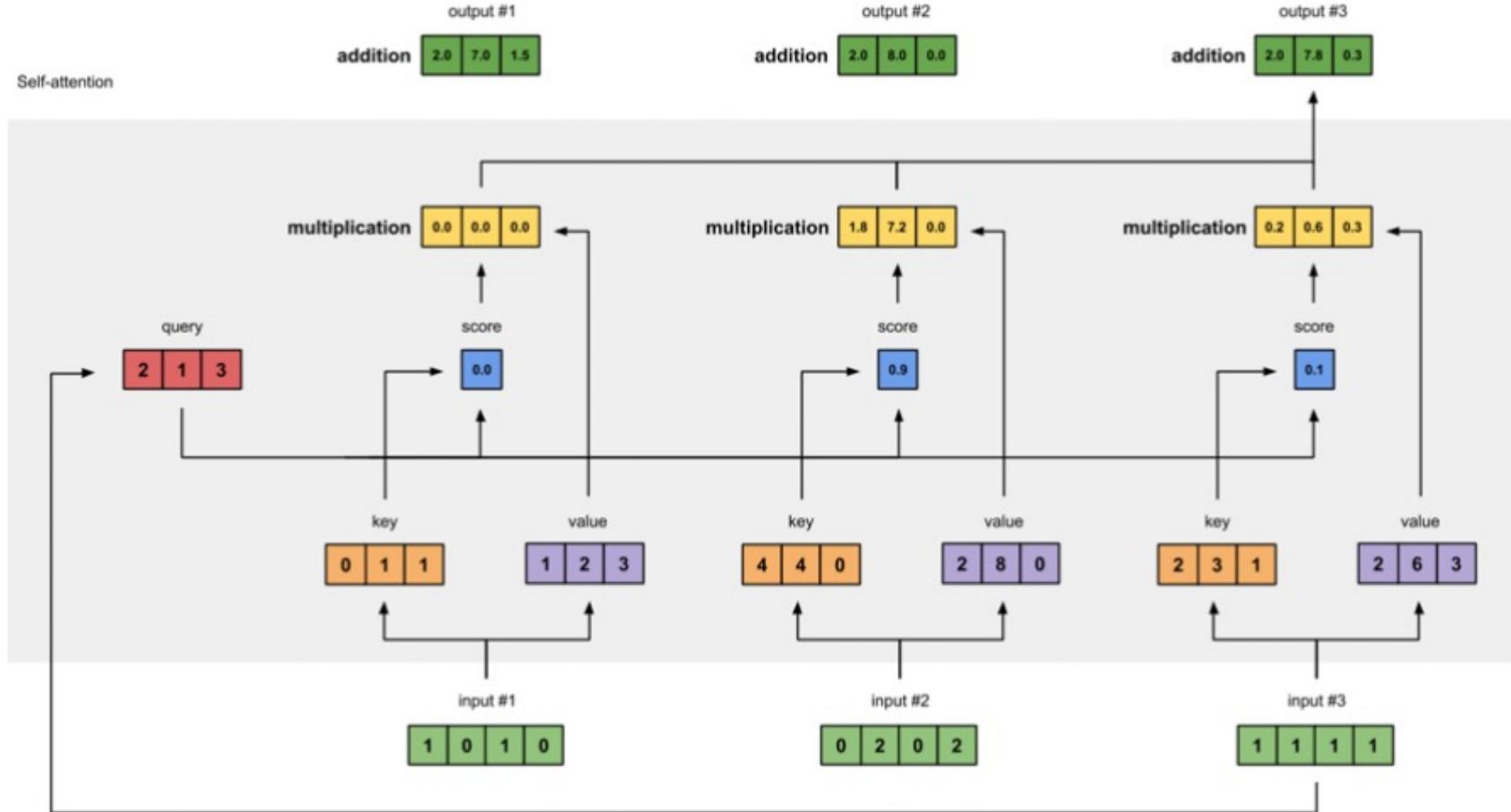
Scaled dot-product attention (6)



Scaled dot-product attention (7)



Scaled dot-product attention (8)



Vettori Query, Chiave e Valore

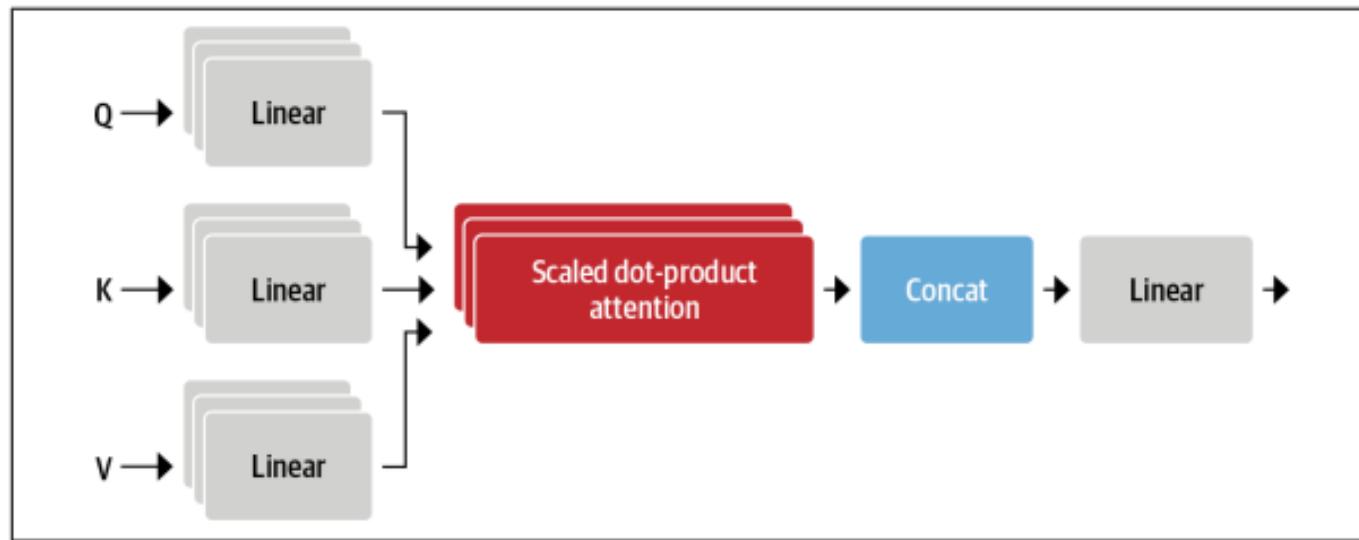
- ▶ La nozione di vettori query, chiave e valore può sembrare un po' criptica la prima volta che la si incontra. I loro nomi sono stati ispirati dai sistemi di **information retrieval**, ma possiamo motivarne il significato con una semplice analogia
- ▶ Immaginate di trovarvi al supermercato per acquistare tutti gli ingredienti necessari per la vostra cena
 - ▶ Ricetta del piatto – Ingredienti (query)
 - ▶ Passeggiamo tra gli scaffali osservando le etichette (chiave)
 - ▶ Controlliamo se corrispondono ad un ingrediente della lista
 - ▶ Se c'è una corrispondenza, si prende il prodotto dallo scaffale (valore)

Cosa manca?

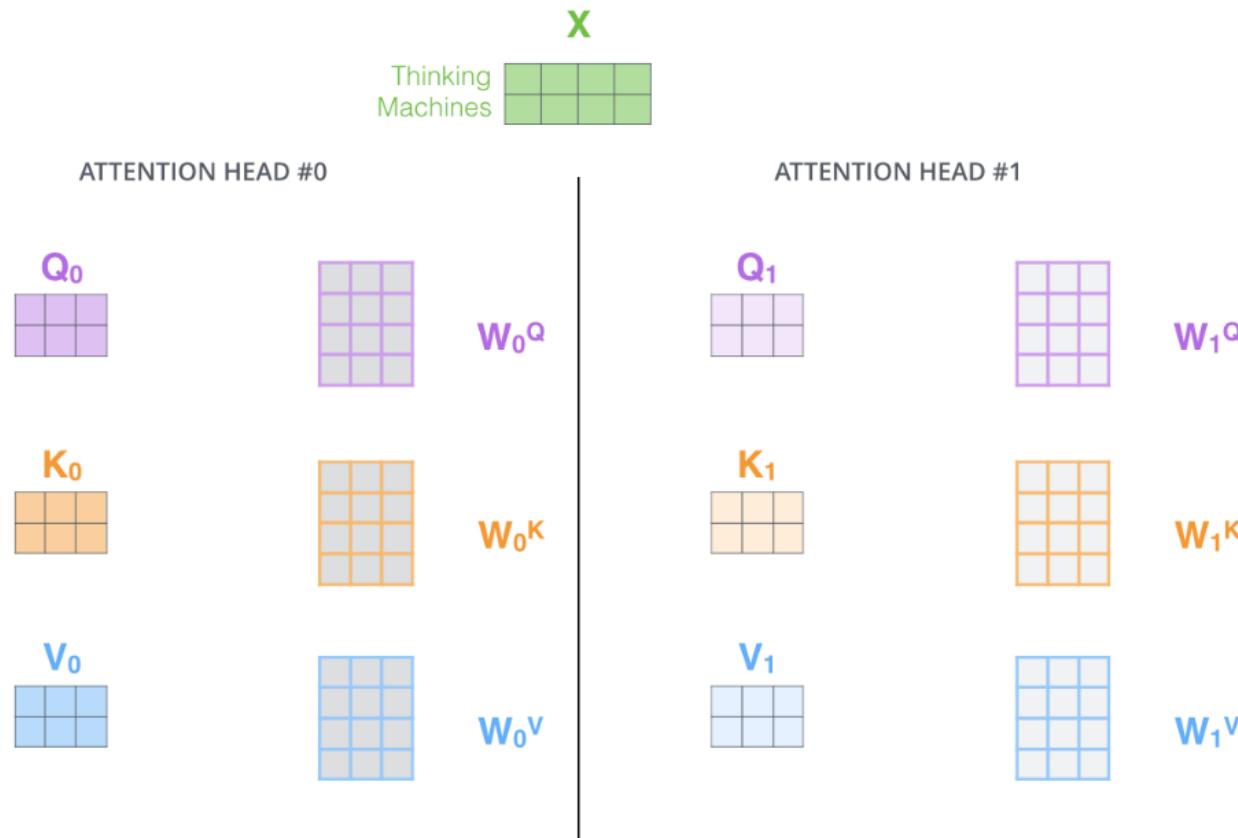
- ▶ Le trasformazioni proiettano gli embedding e ogni proiezione porta con sé il proprio insieme di parametri che consentono al self-attention layer di concentrarsi su diversi aspetti semantici della sequenza
- ▶ Il problema è che il softmax layer tende a concentrarsi su un singolo aspetto

Multi headed-Attention

- ▶ Risulta vantaggiosa la presenza di più serie di proiezioni lineari, ognuna delle quali rappresenta una cosiddetta **attention head**
- ▶ La presenza di più head consente al modello di concentrarsi su più aspetti contemporaneamente

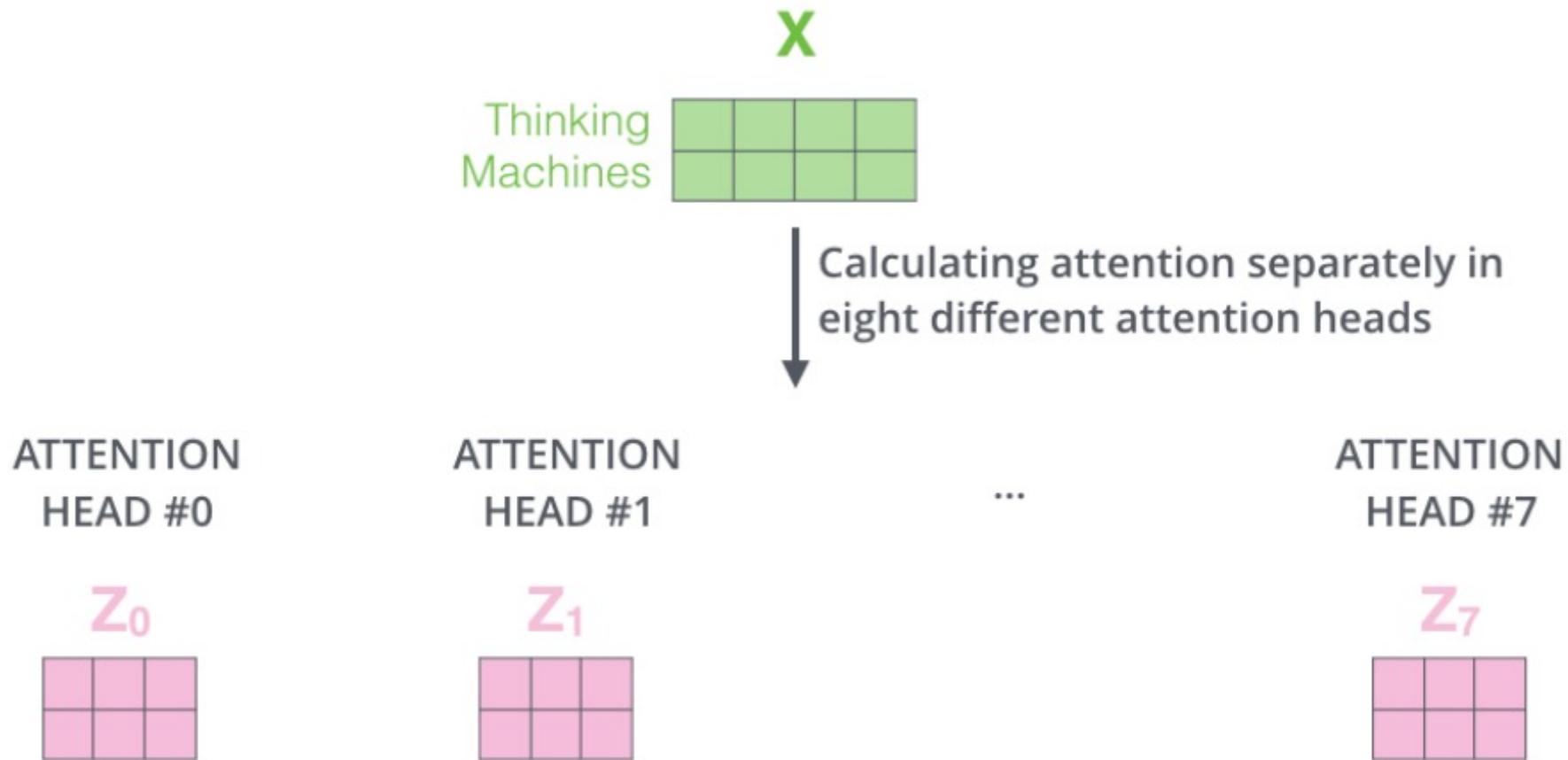


Multi headed-Attention (2)



- ▶ Eseguendo le operazioni precedentemente introdotte e utilizzando diverse matrici di peso in ogni attention head, otteniamo diverse matrici Z

Multi headed-Attention (3)



- ▶ Problema: Il fully connected feed-forward layer si aspetta una **singola** matrice

Multi headed-Attention (4)

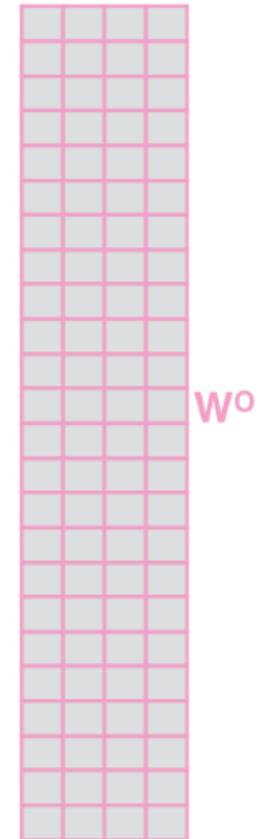
- ▶ Concateniamo le matrici Z e le moltiplichiamo per la matrice W^o

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

\times



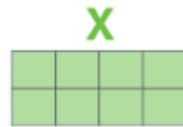
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

$$= \begin{matrix} & Z \\ & \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} \end{matrix}$$

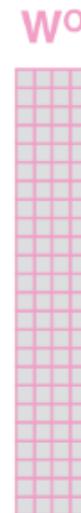
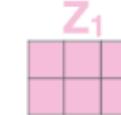
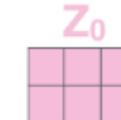
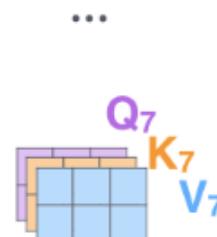
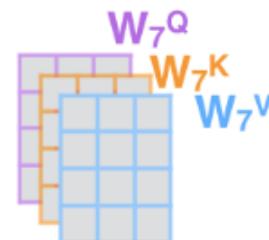
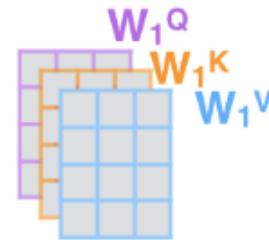
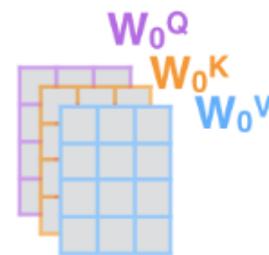
Multi headed-Attention (5)

- 1) This is our input sentence* X
- 2) We embed each word* R
- 3) Split into 8 heads. We multiply X or R with weight matrices W_0^Q, W_0^K, W_0^V , W_1^Q, W_1^K, W_1^V , ..., W_7^Q, W_7^K, W_7^V
- 4) Calculate attention using the resulting $Q/K/V$ matrices Q_0, K_0, V_0 , Q_1, K_1, V_1 , ..., Q_7, K_7, V_7
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

Thinking
Machines



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

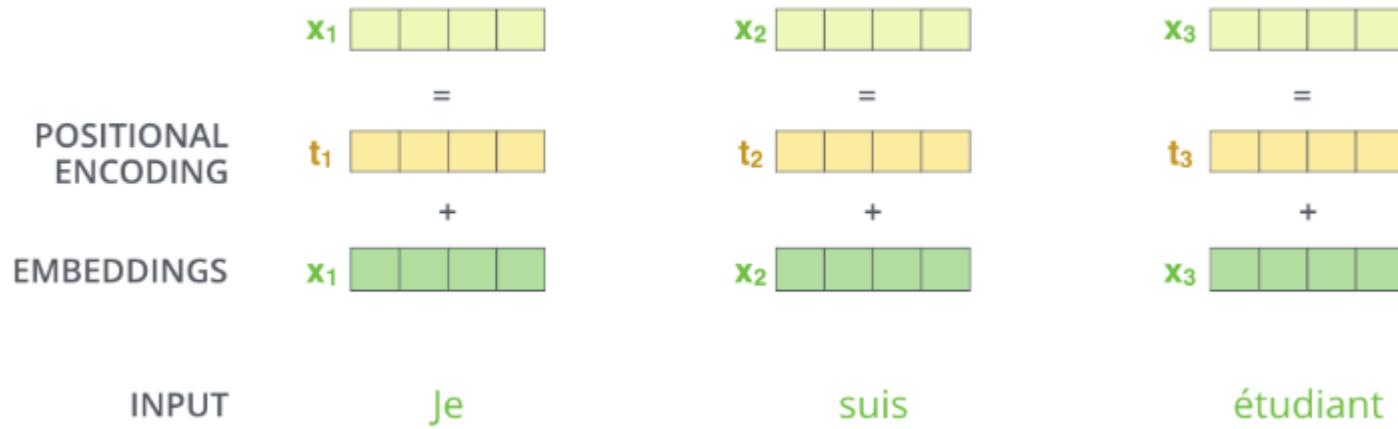


Encoder layer

- ▶ Questo è tutto ciò che è necessario per implementare un encoder layer!
- ▶ Tuttavia, c'è un problema
- ▶ Per come li abbiamo implementati, gli encoder layer non tengono conto dell'ordine (posizione) delle parole nella sequenza di input
- ▶ Fortunatamente, esiste un trucco molto semplice per codificare tali informazioni: **positional embeddings**

Positional embeddings

- ▶ I positional embeddings si basano su un'idea semplice, ma molto efficace: aggiungere un **position-dependent pattern** negli embedding dei token
- ▶ Se il pattern è caratteristico per ogni posizione, le attention head e i feed-forward layer di ogni stack possono imparare a codificare le informazioni sulla posizione nelle loro trasformazioni



Positional embeddings (2)

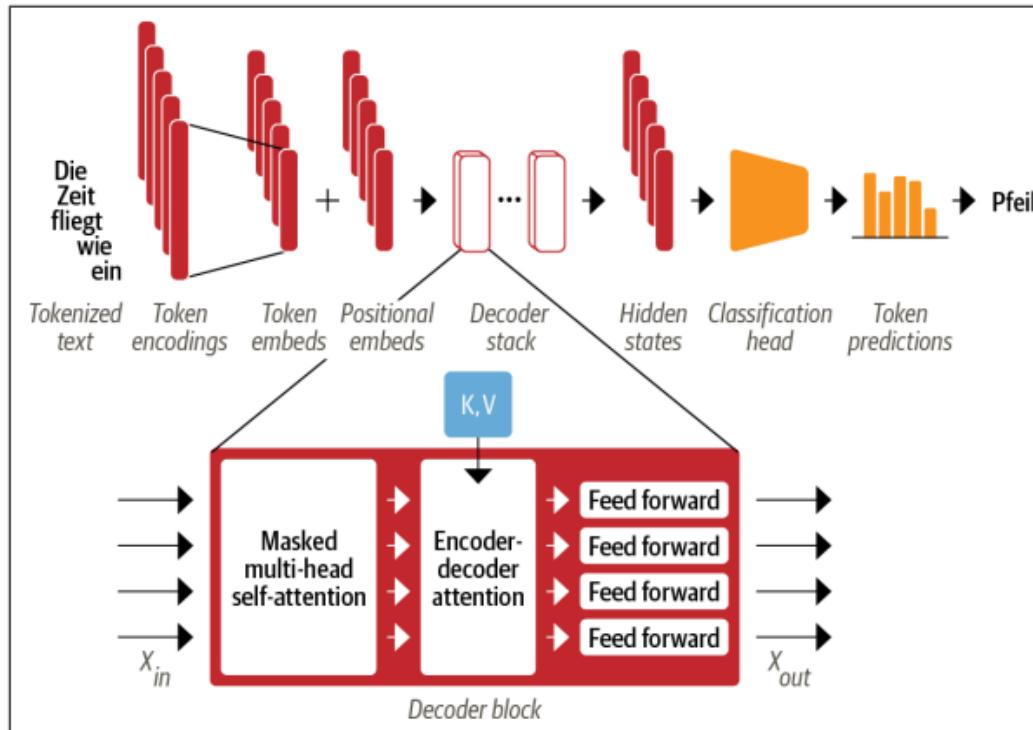
- ▶ Sebbene i position-dependent pattern siano facili da implementare e ampiamente utilizzati, esistono alcune alternative:
 - ▶ **Absolute positional representations**
 - ▶ Pattern statici definiti tramite segnali modulati con le funzioni seno e coseno. Questo metodo funziona particolarmente bene quando non sono disponibili grandi volumi di dati
 - ▶ **Relative positional representations**
 - ▶ Sebbene le posizioni assolute siano importanti, si può sostenere che quando si calcola un embedding, i token circostanti sono più importanti

Fully connected feed-forward layer

- ▶ Il fully connected feed-forward layer presente nell'encoder e nel decoder è una semplice rete neurale a due strati completamente connessa, ma con una modifica
- ▶ Invece di elaborare l'intera sequenza di embedding come un unico vettore, elabora ogni embedding in modo indipendente
- ▶ Per questo motivo, questo layer viene spesso chiamato ‘**position-wise feed-forward layer**’

Decoder

- ▶ La differenza principale tra un decoder e un encoder è che il primo ha due attention sublayer:
 - ▶ Masked multi-head self-attention layer
 - ▶ Encoder-decoder attention layer



Masked multi-head self-attention layer

- ▶ Vediamo le modifiche da apportare per includere la mascheratura nel self-attention layer
- ▶ Il trucco consiste nell'introdurre una **mask matrix** con degli uni sulla diagonale inferiore e degli zeri su quella superiore

```
tensor([[1., 0., 0., 0., 0.],  
       [1., 1., 0., 0., 0.],  
       [1., 1., 1., 0., 0.],  
       [1., 1., 1., 1., 0.],  
       [1., 1., 1., 1., 1.]])
```

Masked multi-head self-attention layer (2)

- ▶ Una volta applicata la mask matrix, possiamo impedire a ogni attention head di ‘sbirciare’ i token futuri sostituendo tutti gli zeri con $-\infty$

```
tensor([[[26.8082,      -inf,      -inf,      -inf,      -inf],
        [-0.6981,  26.9043,      -inf,      -inf,      -inf],
        [-2.3190,   1.2928,  27.8710,      -inf,      -inf],
        [-0.5897,   0.3497,  -0.3807,  27.5488,      -inf],
        [ 0.5275,   2.0493,  -0.4869,   1.6100,  29.0893]]],  
grad_fn=<MaskedFillBackward0>)
```

Linear e Softmax Layer

- ▶ Lo stack del decoder emette un vettore di float. Come lo trasformiamo in una parola?
- ▶ Questo è il compito del Linear Layer finale, il quale è seguito da un Softmax Layer
- ▶ Il Linear Layer è una semplice rete neurale completamente connessa che proietta il vettore prodotto dallo stack di decoder layer in un vettore molto più grande chiamato **vettore logits**
- ▶ Il Softmax Layer trasforma i punteggi in probabilità

Linear e Softmax Layer (2)

Which word in our vocabulary
is associated with this index?

am

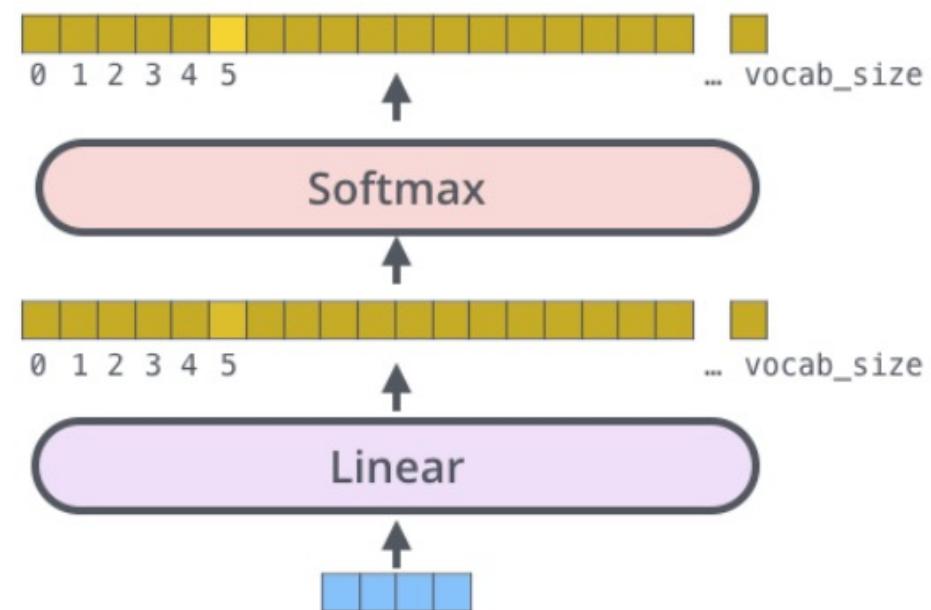
Get the index of the cell
with the highest value
(argmax)

5

log_probs

logits

Decoder stack output

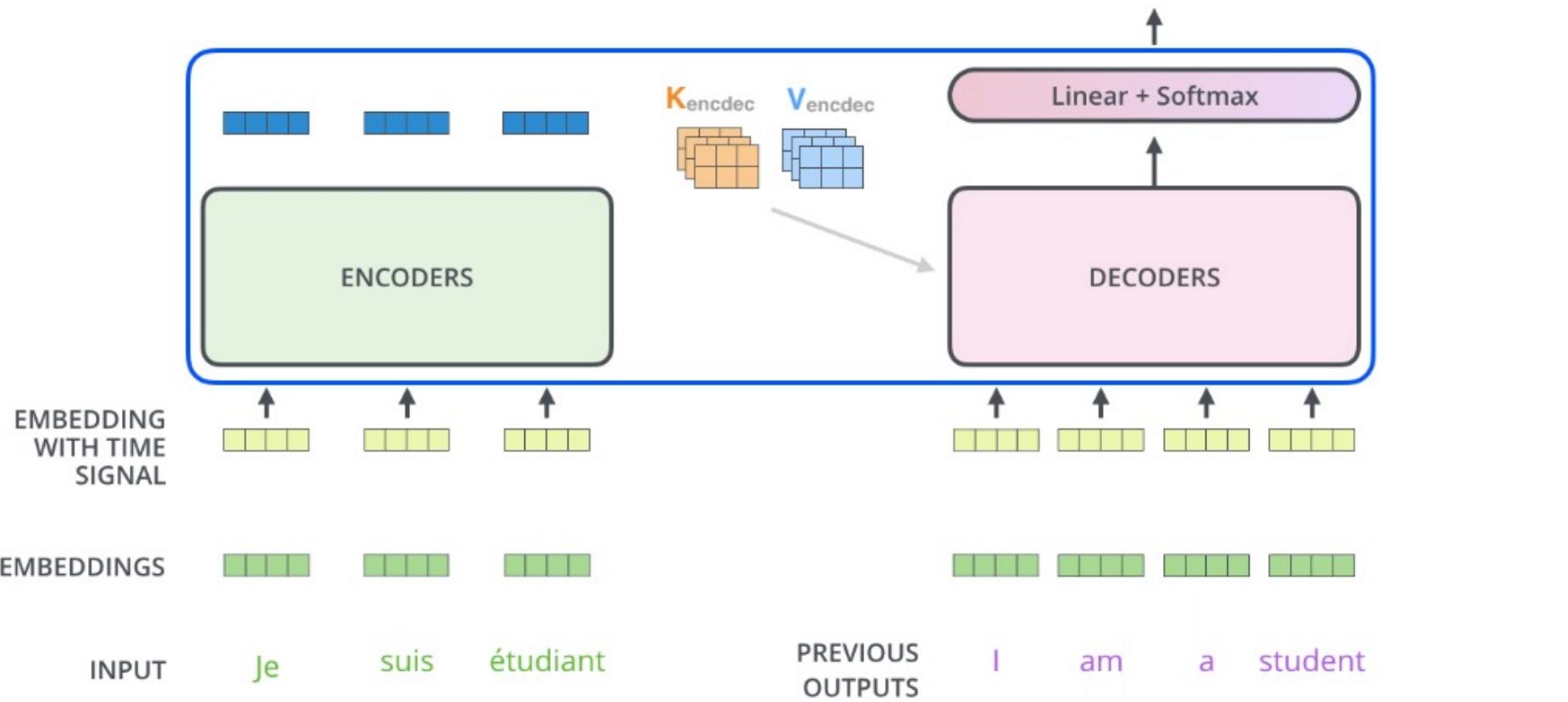


Transformer

Decoding time step: 1 2 3 4 5 6

OUTPUT

I am a student <end of sentence>

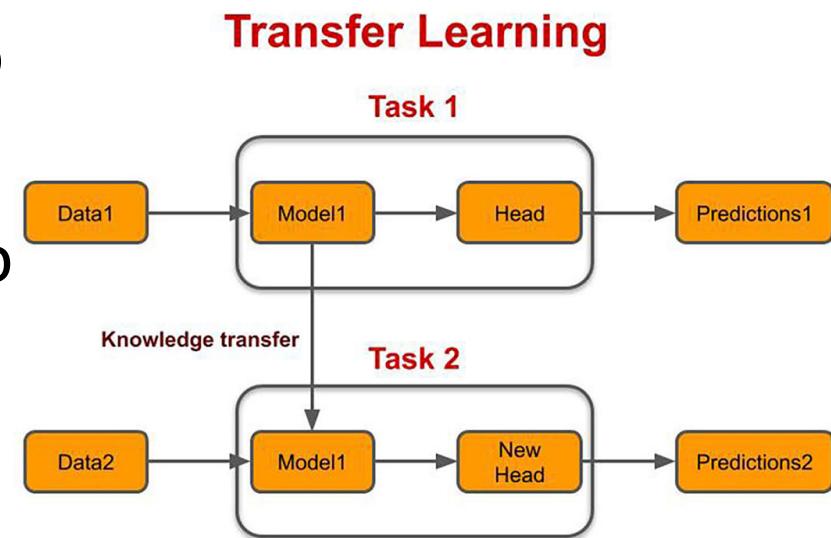


Transformer

- ▶ Immaginate di essere in classe a sostenere un esame (decoder)
- ▶ Il vostro compito è prevedere la parola successiva in base alle parole precedenti (input del decoder)
- ▶ Fortunatamente, un vostro amico (l'encoder) ha il testo completo
- ▶ Sfortunatamente, si tratta di uno studente erasmus e il testo è nella sua lingua madre
- ▶ Da studenti astuti quali siete, trovate un modo per imbrogliare comunque
 - ▶ Disegnate un piccolo fumetto che illustra il testo che avete già (la query) e lo date al vostro amico
 - ▶ Lui cerca di capire quale parte del suo testo corrisponde a tale descrizione (la chiave) e disegna una vignetta che descrive la parola che segue quella parte (il valore)
 - ▶ Utilizzate tale informazione per completare il testo

Transfer Learning

- ▶ Una delle problematiche più rilevanti quando si ha a che fare con modelli complessi, è la mancanza di dati etichettati sufficienti per riuscire ad addestrarli efficacemente.
- ▶ Il Transfer Learning è una tecnica di Machine Learning in cui un modello, addestrato per uno specifico scopo, viene riproposto per un problema diverso, ma appartenente ad un contesto correlato.



TL: Strategie di addestramento

- ▶ **Addestrare l'intera architettura:** addestrare ulteriormente l'intero modello pre-addestrato sul set di dati. In questo caso, il calcolo dei pesi viene propagato all'indietro attraverso l'intera architettura e i pesi pre-addestrati del modello vengono aggiornati in base al nuovo set di dati;
- ▶ **Addestrare alcuni layer, congelando altri:** un altro modo per utilizzare un modello pre-addestrato consiste nell'addestrarlo parzialmente. Bisogna mantenere congelati i pesi dei layer iniziali del modello e si riaddestrano i layer più alti;
- ▶ **Congelare l'intera architettura:** è possibile congelare tutti i layer del modello pre-addestrato, collegare nuovi layer alla rete neurale e addestrare il nuovo modello. Durante la fase di addestramento verranno aggiornati solo i pesi dei layer collegati al modello pre-addestrato.

Transfer Learning: vantaggi

► Sviluppo più rapido

I pesi dei modelli pre-addestrati codificano già molte informazioni sul linguaggio. Di conseguenza, è sufficiente solo modellarli durante la fase di addestramento, richiedendo complessivamente molto meno tempo per il completamento del processo.

► Meno dati richiesti

Grazie ai pesi pre-addestrati, è possibile utilizzare un set di dati più piccolo rispetto a quello che sarebbe richiesto per costruire un modello da zero.

Infatti, uno dei principali svantaggi dei modelli di NLP definiti da zero è che spesso per raggiungere una precisione elevata c'è bisogno di un set di dati dalle dimensioni proibitive per la fase di addestramento.

► Risultati migliori

È stato dimostrato che questa tipologia di modelli permette di ottenere risultati all'avanguardia per un'ampia varietà di attività: classificazione, inferenza linguistica, generazione di testo, ecc...

Tipologie di approcci

- ▶ **Quelli che si servono solo degli Encoder**
 - ▶ Bidirectional Encoder Representations from Transformers (BERT)
- ▶ **Quelli che sfruttano solo i Decoder**
 - ▶ Generative Pre-trained Transformer (GPT)
- ▶ **Quelli che utilizzano sia Encoder che Decoder**
 - ▶ Text-To-Text Transfer Transformer (T5)

BERT

- ▶ Bidirectional Encoder Representations from Transformers (BERT) è un modello di rappresentazione del linguaggio basato su Transformer e sviluppato da Google. Rilasciato alla fine del 2018.
- ▶ BERT **base**: 12 encoder, 12 attention head
BERT **large**: 24 encoder, 16 attention head
- ▶ Pre-addestrato su Wikipedia e su Book Corpus, un dataset composto da più di 10.000 libri, di diverso genere.
- ▶ L'encoder del Transformer legge l'intera sequenza di parole in entrambe le direzioni, contemporaneamente. Per tale motivo è considerato **bidirezionale**.

BERT (2)

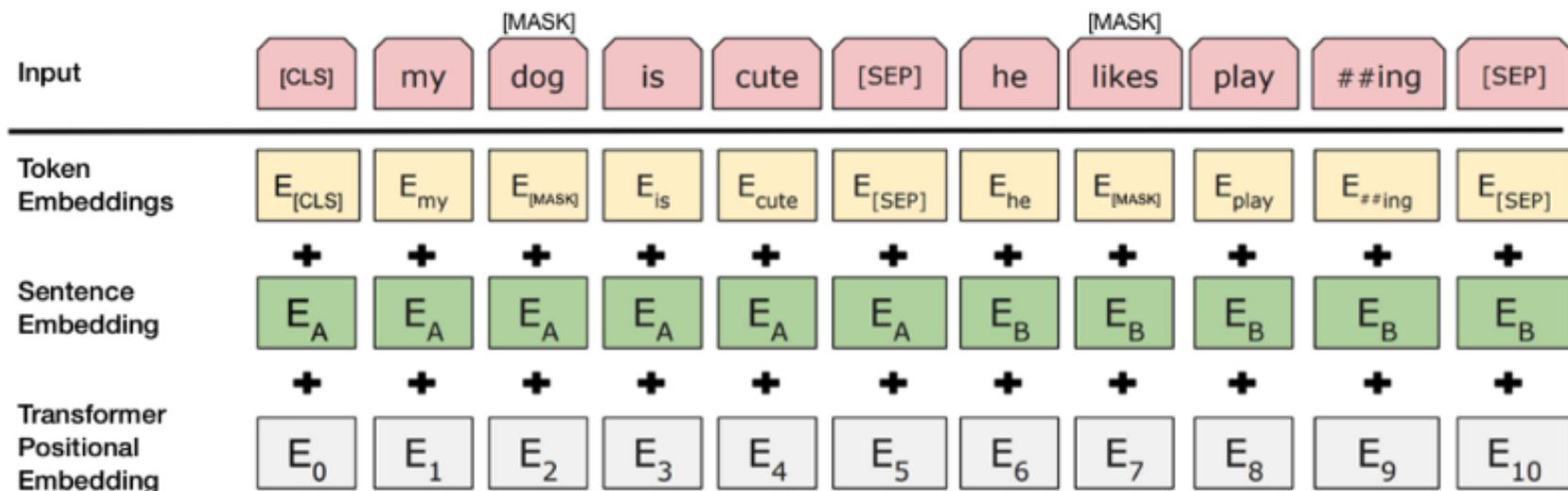
- ▶ Le capacità di BERT di elaborare l'input bidirezionalmente ha segnato l'inizio di una nuova era per il Natural Language Processing.
- ▶ A differenza dei modelli linguistici precedenti, BERT tiene conto contemporaneamente dei token precedenti e di quelli successivi. I modelli LSTM combinati da sinistra a destra e da destra a sinistra mancavano di questa parte di «contemporaneità».
- ▶ Esempi:
 - ▶ “The woman went to the store and bought a _____ of shoes.”
 - ▶ “I accessed the bank account”

BERT (3)

- ▶ L'obiettivo ultimo di BERT è generare un modello di rappresentazione del linguaggio, codificando un input tramite il meccanismo dell'attenzione, ragion per cui necessita solo della parte degli encoder.
- ▶ Per poter essere processato da BERT l'input necessita di essere coadiuvato da una serie di metadata.
 - ▶ **Token embeddings:** viene aggiunto un token [CLS] ai token delle parole in input all'inizio della prima frase e un token [SEP] viene inserito alla fine di ogni frase.
 - ▶ **Segment embeddings:** ad ogni token viene aggiunto un marcitore che indica la frase A o la frase B. Questo permette all'encoder di distinguere tra la frase A e quella B.
 - ▶ **Positional embeddings:** ad ogni token viene aggiunto un positional embedding per indicare la sua posizione nella frase.

BERT (4)

- ▶ L'obiettivo ultimo di BERT è generare un modello di rappresentazione del linguaggio, codificando un input sfruttando il meccanismo dell'attenzione, ragion per cui necessita solo della parte degli encoder.

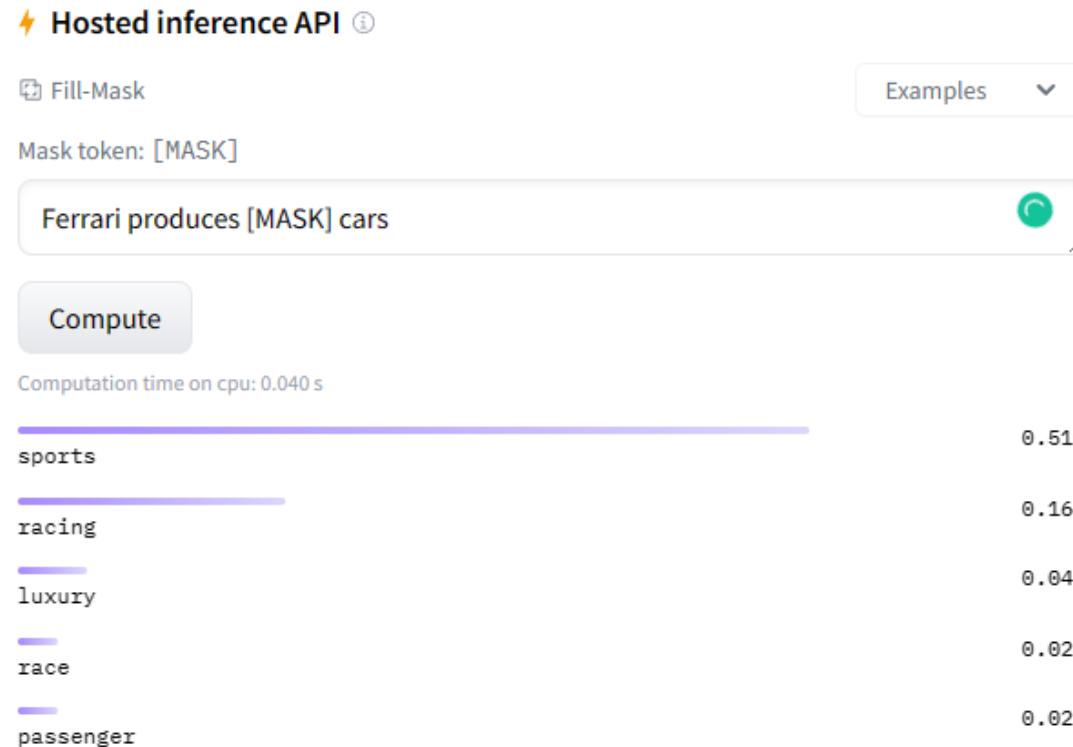


BERT (5)

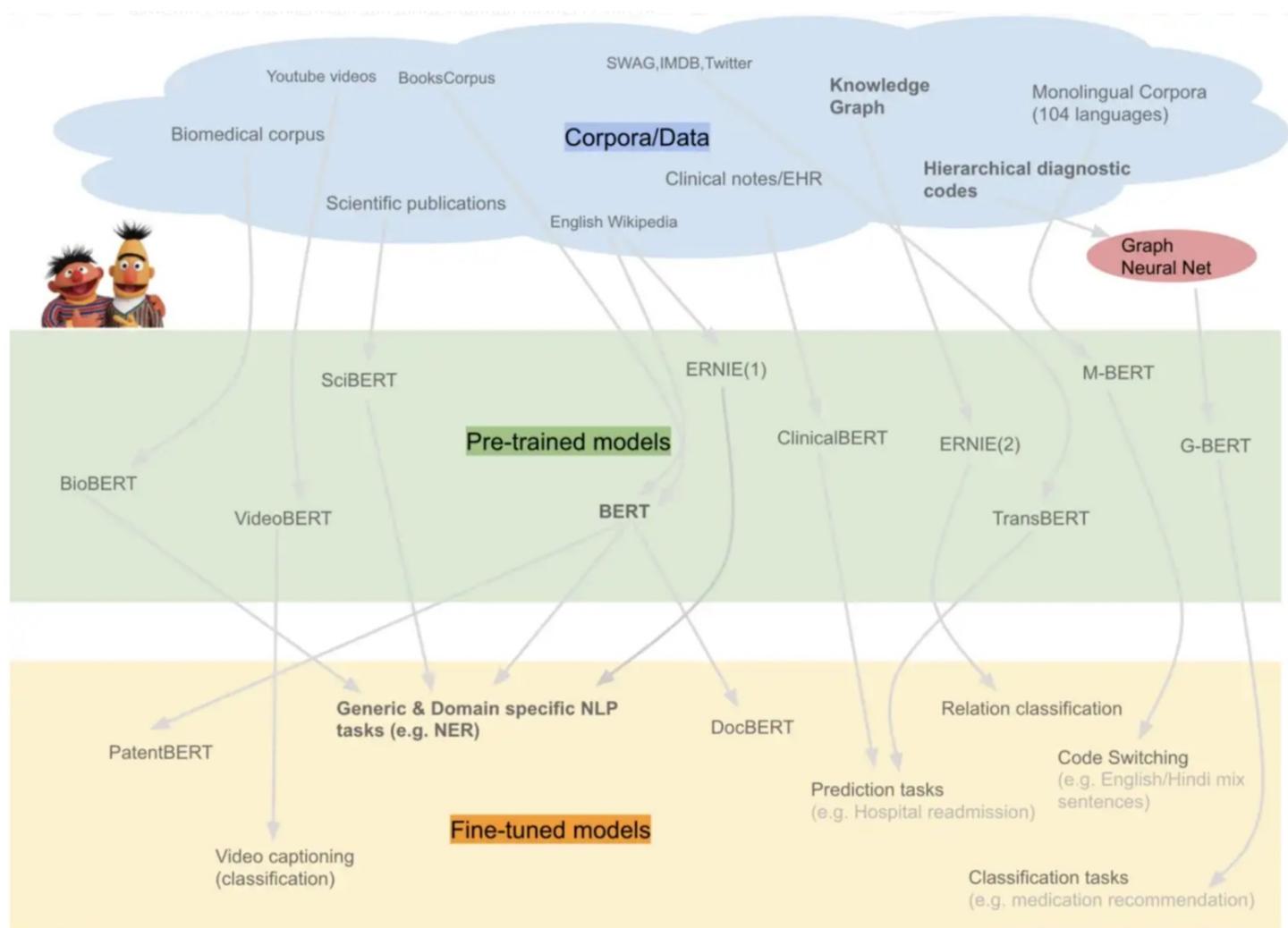
- ▶ BERT si serve di due strategie di addestramento:
 - ▶ **Masked Language Modeling (Masked LM):** prima di inserire sequenze di parole in BERT, il 15% delle parole in ciascuna sequenza vengono sostituite con il token [MASK]. Il modello tenta quindi di prevedere il valore originale delle parole mascherate, in base al contesto fornito dalle altre parole non mascherate nella sequenza.
 - ▶ **Next Sentence Prediction (NSP):** durante il processo di addestramento, BERT riceve in input coppie di frasi e impara a prevedere se la seconda frase nella coppia è successiva alla prima nel documento.
 - 50% dell'input è composto da coppie di frasi in cui la seconda frase è effettivamente la frase successiva nel documento.
 - L'altro 50% è composto da coppie di frasi in cui la seconda frase viene scelta casualmente dal corpus.

BERT (6)

Accedendo al link <https://huggingface.co/bert-base-uncased> è possibile testare in modo interattivo BERT nella sua variante «base».



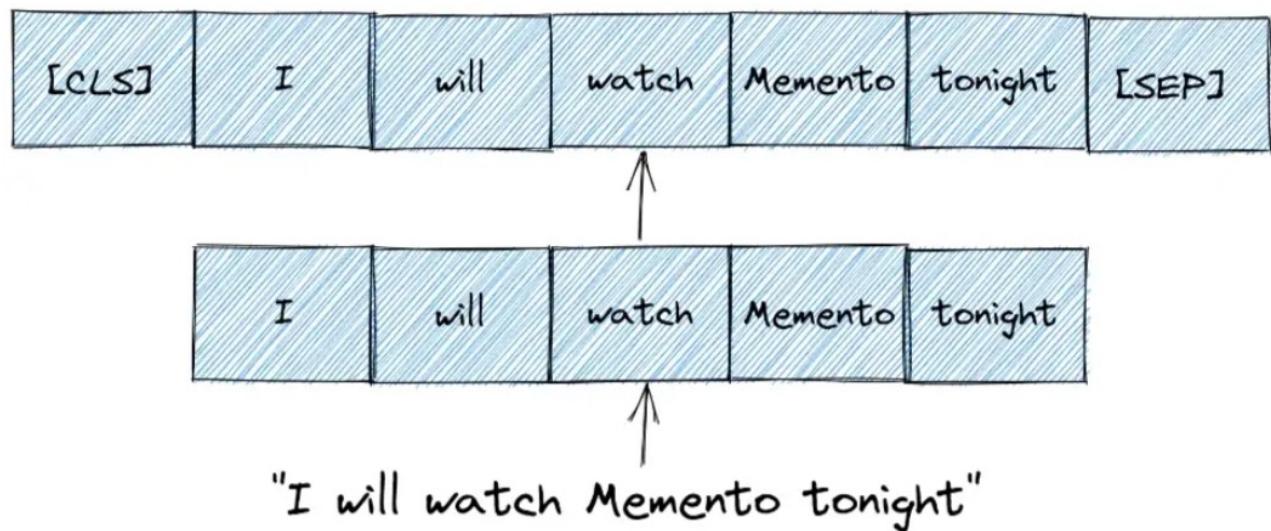
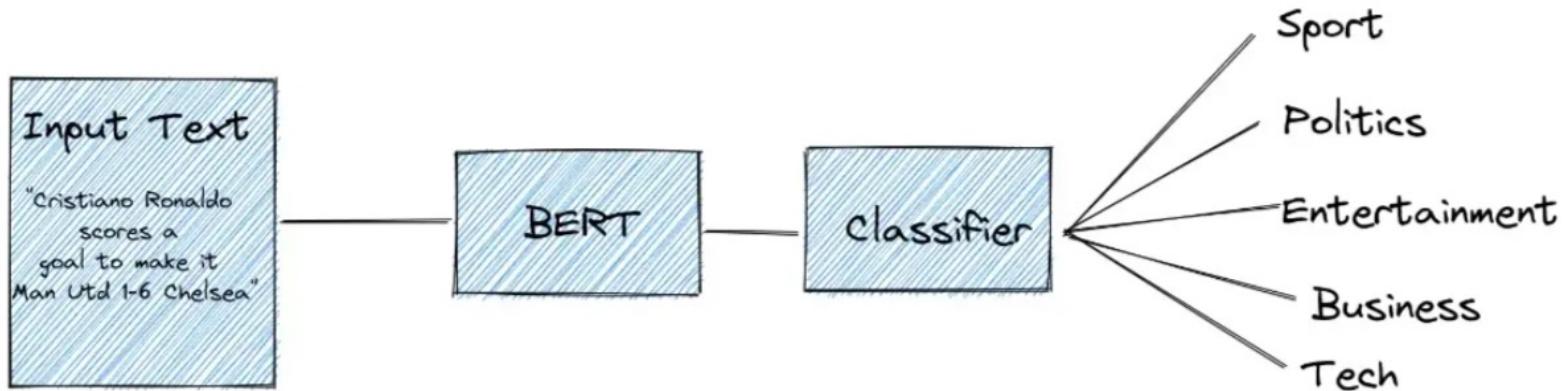
FINE-TUNING BERT



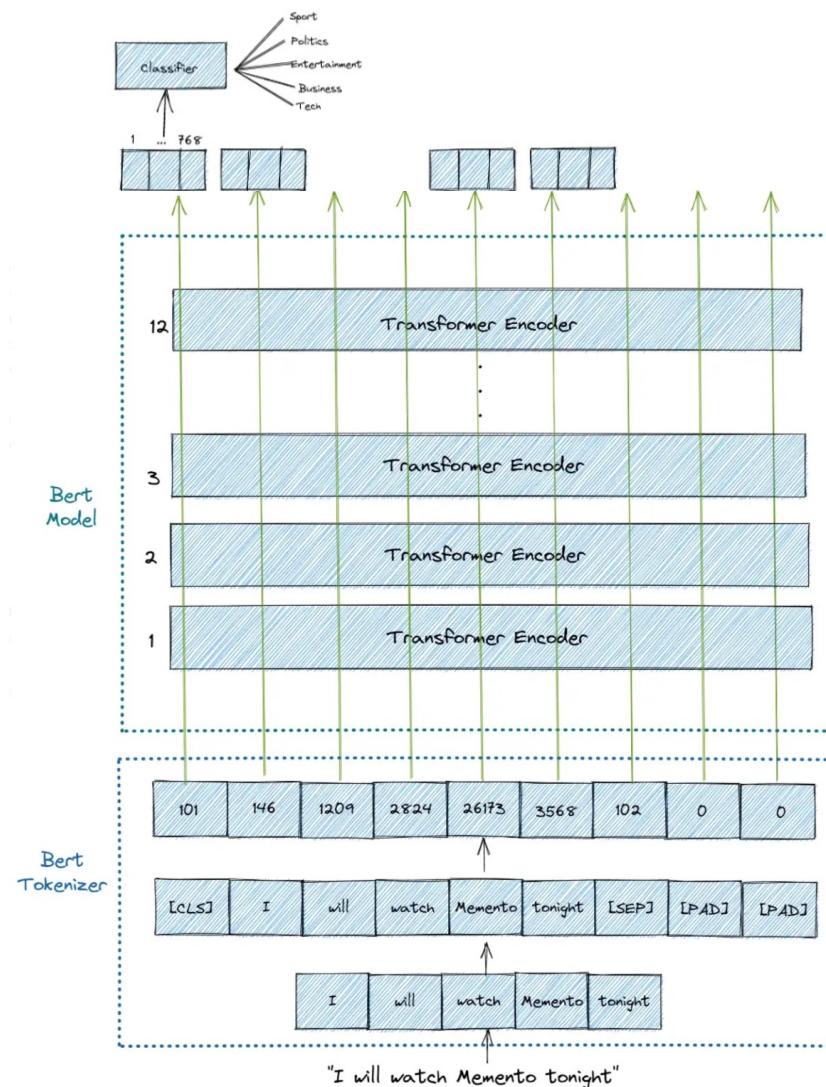
FINE-TUNING BERT (1)

- ▶ Al fine di personalizzare il comportamento di BERT per un task specifico è sufficiente aggiungere un layer in cima al modello base di BERT.
- ▶ Infatti, il modello BERT produce in output un vettore di embedding di 768 elementi per ciascuno dei token. Questi vettori possono essere passati come input per diversi altri modelli e per diverse applicazioni NLP, che si tratti di classificazione di testi, previsione di frasi successive, Named-Entity-Recognition (NER), o question-answering.

FINE-TUNING BERT (2)

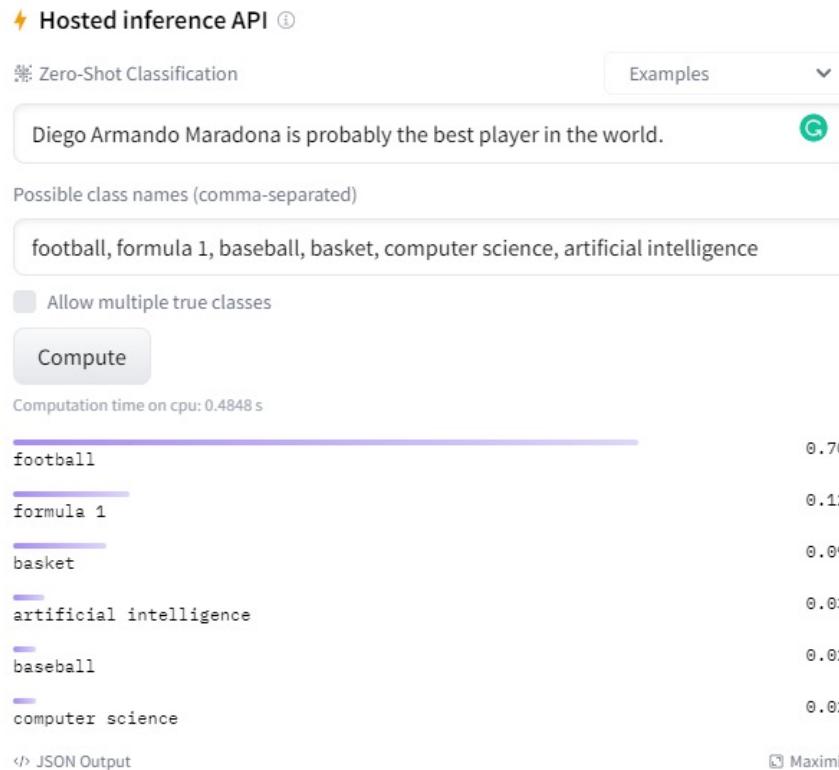


FINE-TUNING BERT (3)



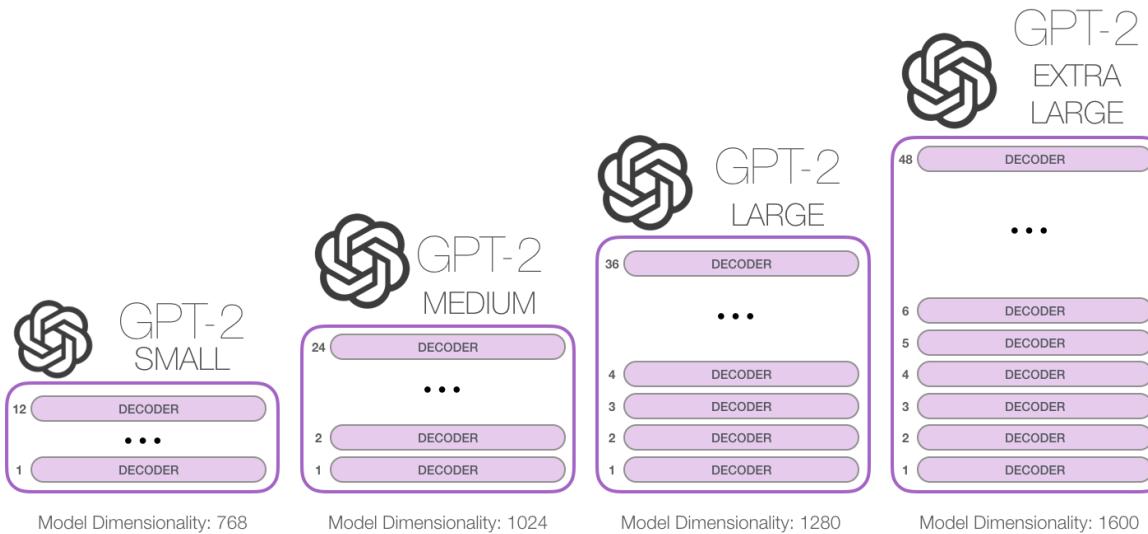
Zero-shot classification

Accedendo al link <https://huggingface.co/Narsil/deberta-large-mnli-zero-cls> potete testare in modo interattivo il modello DeBERTa addestrato per un task di Zero-shot classification.



GPT-2

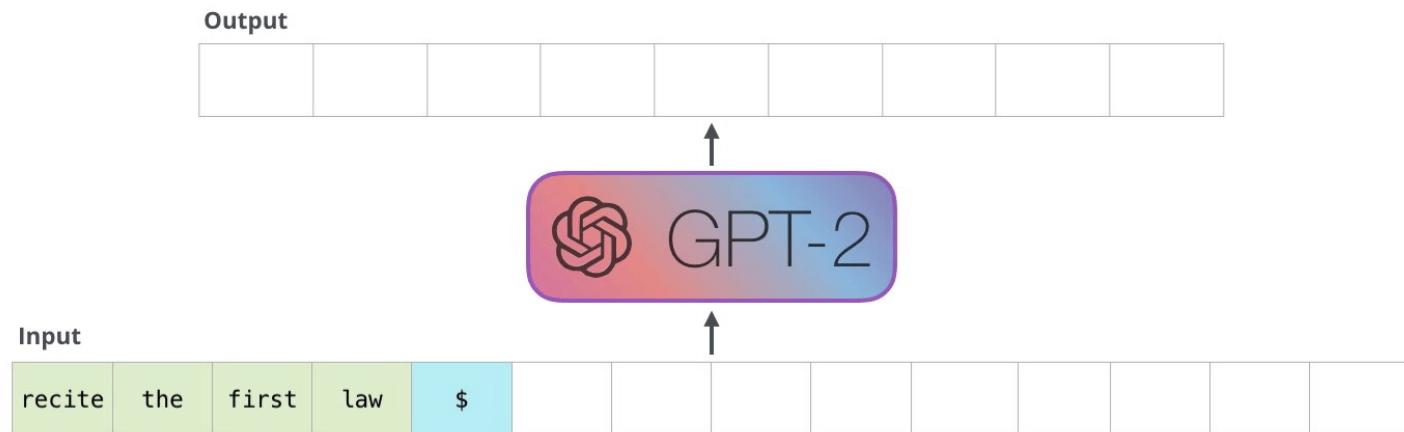
- ▶ GPT-2 è un modello di machine learning sviluppato da OpenAI, un gruppo di ricerca sull'intelligenza artificiale con sede a San Francisco. GPT-2 è in grado di generare testi grammaticalmente corretti e notevolmente coerenti.
- ▶ La versione più «grande» di GPT-2 ha 1,5 miliardi di parametri ma non è stata rilasciata al pubblico temendo che il modello venisse utilizzato per scopi nefasti.



- ▶ Sono stati rilasciati modelli più piccoli con 124-1558 milioni di parametri e uno strumento per la messa a punto di tali modelli su altri dati.
- ▶ Addestrato su dataset di 40GB, su 8 Milioni di pagine, prese da reddit.

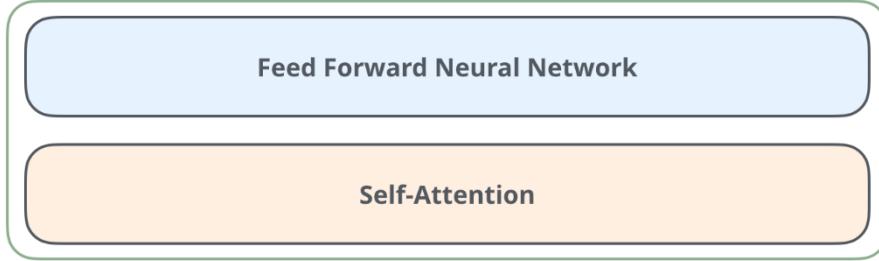
GPT-2 (2)

- ▶ GPT-2 è costruito utilizzando solo il decoder transformer model. BERT, invece, utilizza solo gli encoder.
- ▶ La differenza è che tra i due GPT-2, come i modelli linguistici tradizionali, produce un token alla volta.
- ▶ Chiediamo ad esempio a un GPT-2 ben addestrato di recitare la prima legge della robotica:



GPT-2 (3)

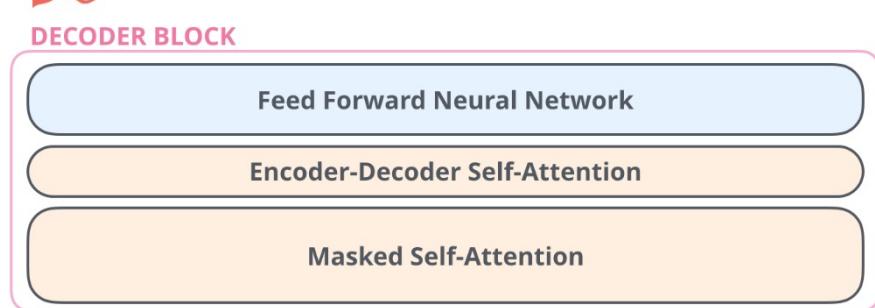
THE TRANSFORMER ENCODER BLOCK



robot	must	obey	orders	<eos>	<pad>	...	<pad>
1	2	3	4	5	6		512

- ▶ Una differenza fondamentale nel livello di self-attention è che il decoder maschera i token futuri, non cambiando la parola in [MASK] come il BERT, ma interferendo nel calcolo della self-attention e bloccando le informazioni dei token che si trovano a destra della posizione calcolata.

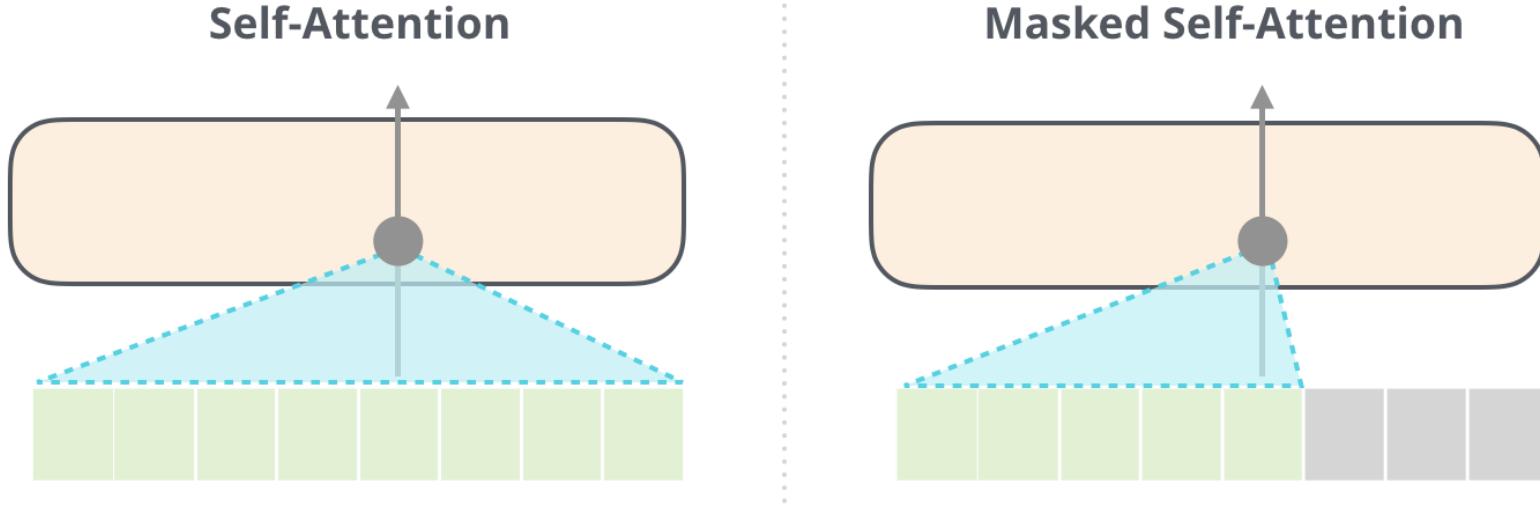
THE TRANSFORMER DECODER BLOCK



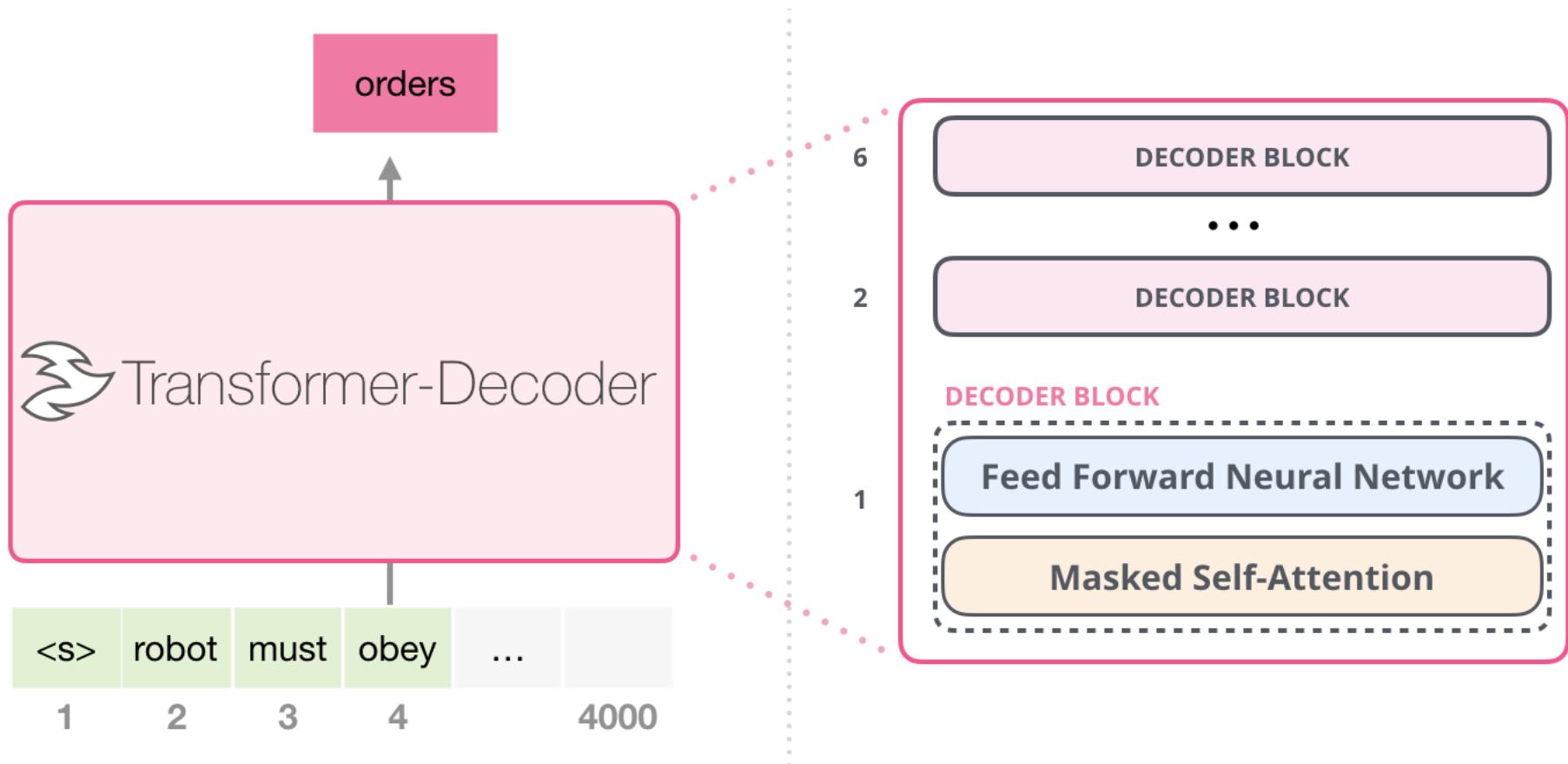
Input	<s>	robot	must	obey						512
	1	2	3	4	5	6				

GPT-2 (4)

- ▶ È importante che sia chiara la distinzione tra self-attention (quella che usa BERT) e Masked self-attention (quella che usa GPT-2). Un normale self-attention block permette a un token di «dare una sbirciata» sia ai token alla sua sinistra che quelli alla sua destra.
- ▶ La masked self-attention, invece, impedisce che ciò avvenga.



GPT-2 (5)



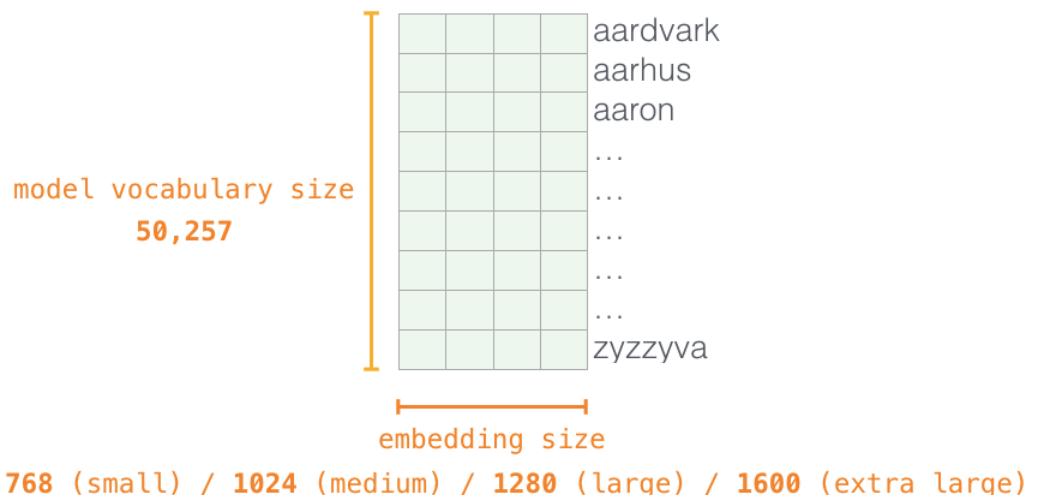
GPT-2 (6)

- ▶ Ma come funziona?
 - ▶ GPT-2 può essere eseguito in due modalità distinte
1. **Generate unconditional samples:** al modello non viene fornito alcun input, e gli viene chiesto di generare un testo partendo da un singolo token, ovvero `<|endoftext|>`. Il quale normalmente rappresenta la fine di un blocco di testo e da questo iniziare a generare un testo.
 2. **Generate interactive conditional samples:** al modello vengono fornite alcune parole, che nello specifico vengono chiamati **prompt**, i quali guidino il modello verso la generazione di un testo afferente alla topica inferita tramite essi.

GPT-2 (7)

- ▶ Partiamo dall'input. Come in altri modelli NLP il modello cerca l'embedding della parola in ingresso nella sua matrice di embedding.
- ▶ Ogni riga è un word embedding: un elenco di numeri che rappresentano una parola e ne catturano parte del significato. La dimensione di questo elenco varia a seconda delle dimensioni del modello GPT2. Il modello più piccolo utilizza una dimensione di embedding di 768 token per word.

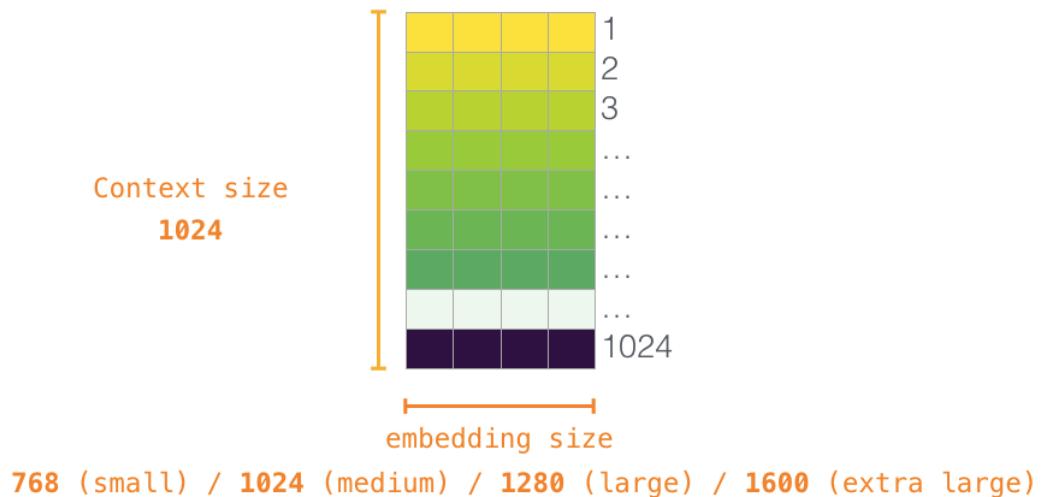
Token Embeddings (wte)



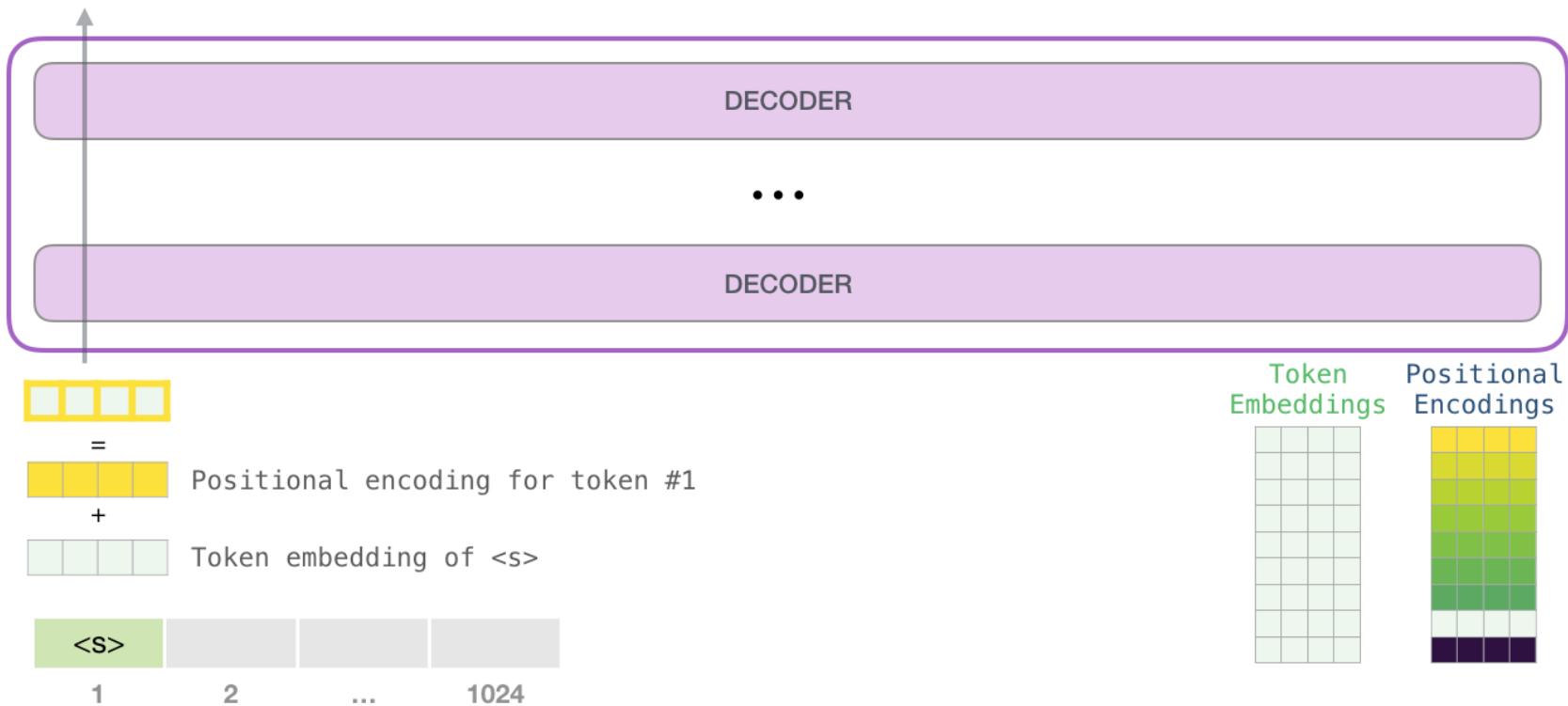
GPT-2 (8)

- ▶ All'inizio, quindi, cerchiamo l'embedding del token iniziale <s> nella matrice di embedding.
- ▶ Poi, prima di trasmetterlo al primo blocco del decoder, incorporiamo la sua positional encoding.

Positional Encodings (wpe)



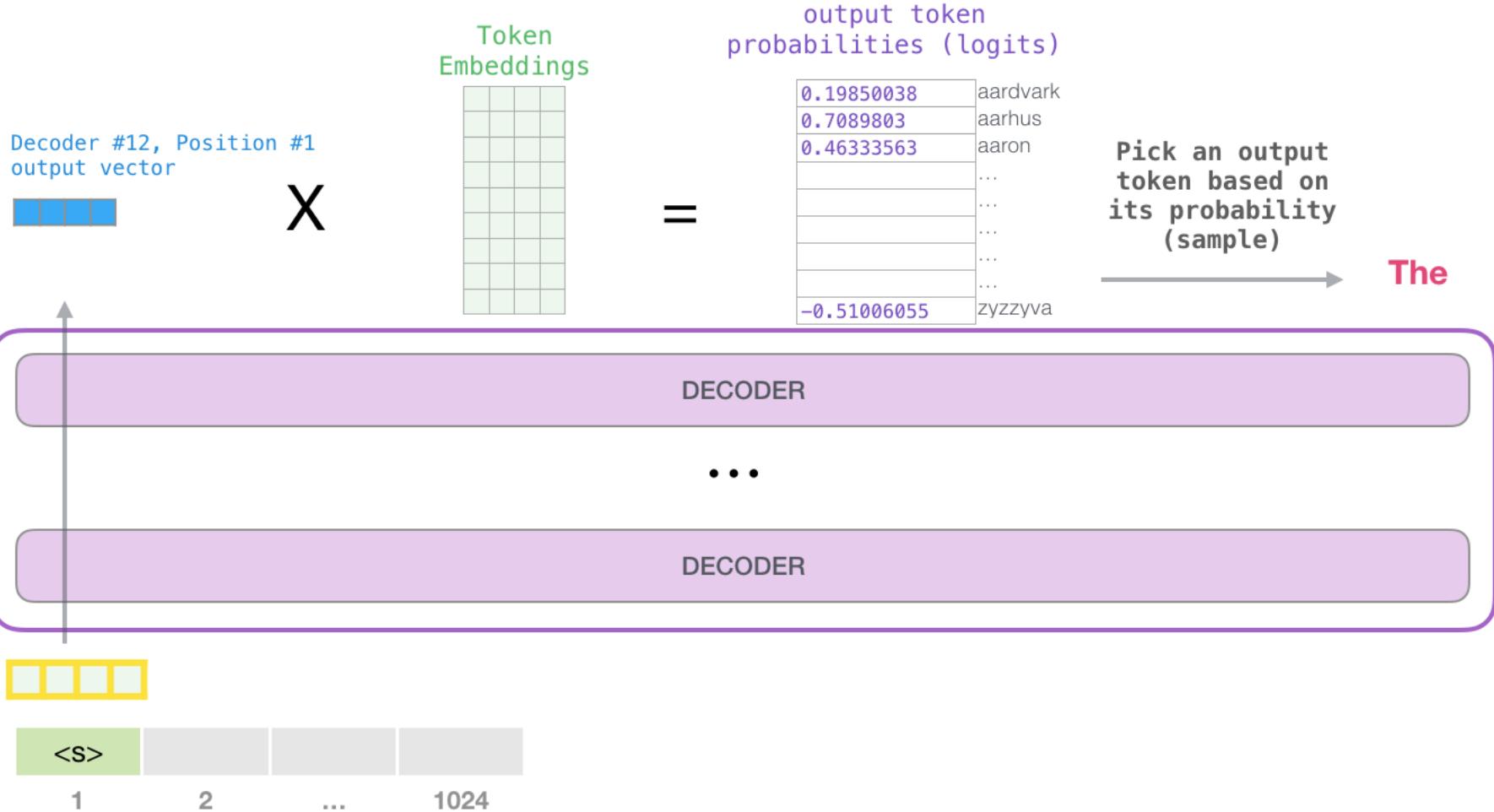
GPT-2 (9)



GPT-2 (10)

- ▶ Il primo blocco può ora elaborare il token facendolo passare prima attraverso il processo di masked self-attention e poi attraverso la feed forward neural network.
- ▶ Una volta che il primo blocco transformer ha elaborato il token, invia il vettore risultante su per lo stack per essere elaborato dal blocco successivo.
- ▶ Il processo è identico in ogni blocco, ma ogni blocco trattiene i propri pesi in entrambi i sublayer della masked self-attention e FFNN.
- ▶ Quando l'ultimo blocco, quello più in alto dell'architettura, produce il suo vettore di output, il modello moltiplica tale vettore per la matrice di embedding, ciò produce uno score associabile alle probabilità che l'n-esimo token presente nel vocabolario, sia quello da generare.

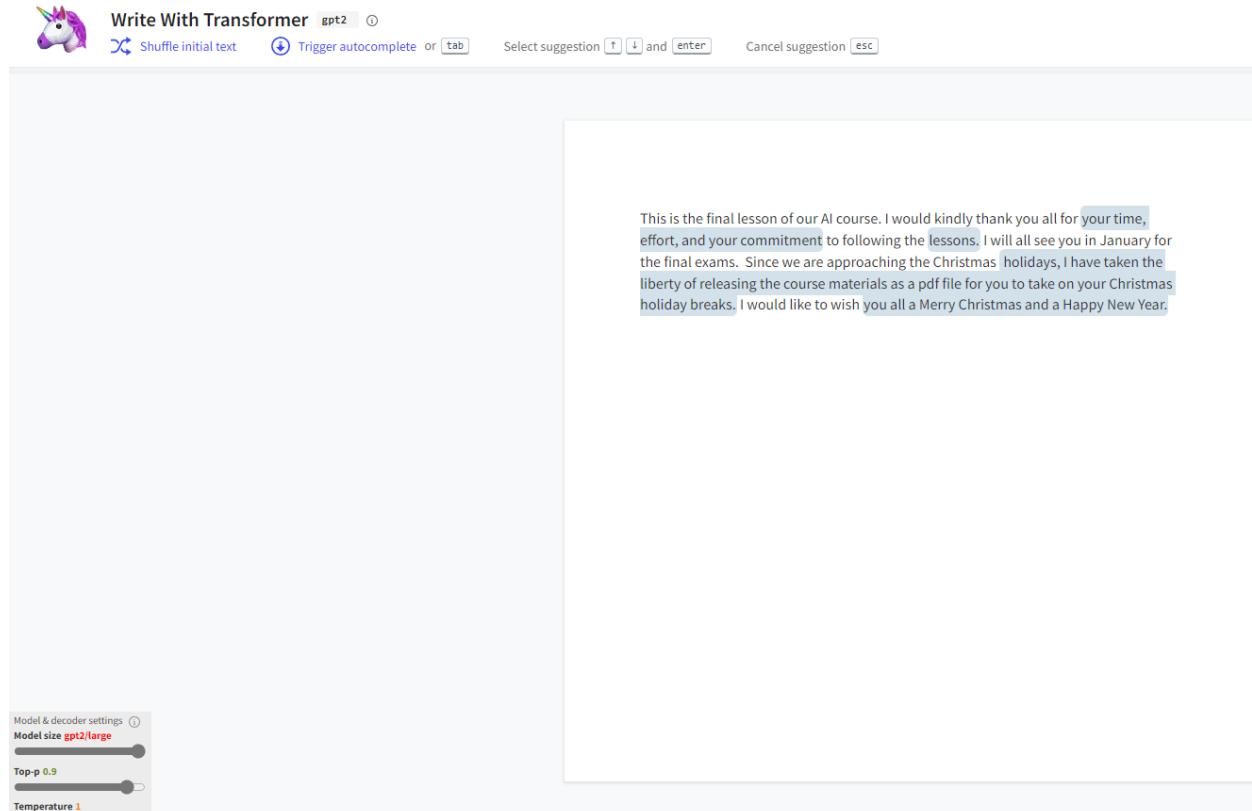
GPT-2 (11)



GPT-2 (12)

Completamento del testo tramite GPT-2 <https://transformer.huggingface.co/doc/gpt2-large>

- **Model size:** la quantità di dati su cui è addestrato il modello.
- **Top-p:** limita le ipotesi generate ad una specifica probabilità cumulativa.
- **Temperature:** Un valore più alto consente al modello di generare frasi più «pazze». È consigliato tenerlo tra 0.7 e 1.
- **Max time:** tempo massimo concesso al tool per generare il testo.



T5

- ▶ Il modello Text-to-Text-Transfer-Transformer proposto da Google, si è posto l'obiettivo di riprogettare tutti i compiti del NLP in un formato unificato text-to-text, in cui l'input e l'output sono sempre stringhe di testo.
- ▶ Gli autori suggeriscono di utilizzare lo stesso modello, la stessa loss function e gli stessi iperparametri per tutti i compiti NLP. In questo approccio, gli input sono modellati in modo tale che il modello riconosca il compito e l'output è semplicemente la versione "testuale" del risultato atteso.



T5 (2)

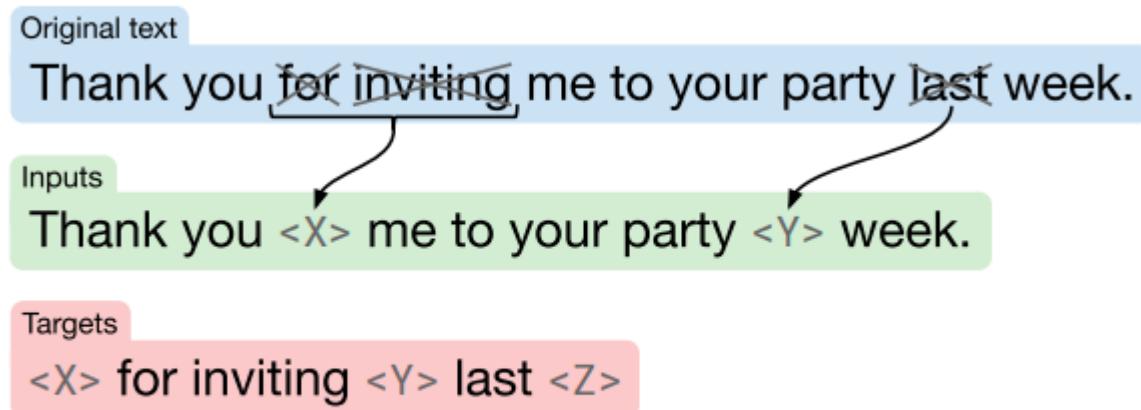
- ▶ Il modello è stato addestrato sul dataset Colossal Clean Crawled Corpus (C4), ottenuto tramite processo di scrape di pagine web risalente ad Aprile 2019.
- ▶ Le pagine ottenute, sono state filtrate con i seguenti passaggi:
 1. Mantenere le frasi che terminano solo con un segno di punteggiatura terminale valido (punto, punto esclamativo, punto interrogativo o virgolette finali).
 2. Rimozione di qualsiasi pagina contenente parole offensive che compaiono nella "List of Dirty, Naughty, Obscene or Otherwise Bad Words".
 3. Gli avvisi del tipo "JavaScript deve essere abilitato" vengono rimossi filtrando tutte le righe che contengono la parola JavaScript.
 4. Le pagine con testo segnaposto come "lorem ipsum" vengono rimosse.
 5. I codici sorgente vengono rimossi eliminando tutte le pagine che contengono una parentesi graffa "{" (poiché le parentesi graffe sono presenti in molti linguaggi di programmazione noti).
 6. Per rimuovere i duplicati, si considerano gli intervalli di tre frasi. Tutte le occorrenze doppie delle stesse 3 frasi vengono filtrate.
 7. Infine, poiché i compiti a valle riguardano soprattutto la lingua inglese, si utilizza langdetect per filtrare tutte le pagine che non sono classificate come inglese con una probabilità di almeno 0,99.

T5 (3)

- ▶ Uno dei problemi principali di T5 è quello di rendere possibile l'approccio unificato da text-to-text.
- ▶ Per utilizzare lo stesso modello per tutti i compiti a valle, si aggiunge un prefisso testuale specifico per il compito da svolgere, all'input originale che viene fornito al modello. Questo prefisso testuale viene anche considerato come un iperparametro.
- ▶ Ad esempio, per chiedere al modello di tradurre la frase "That is good." dall'inglese al tedesco, il modello dovrebbe ricevere la sequenza "translate English to German: That is good." ed essere addestrato a produrre "Das ist gut.".
- ▶ Allo stesso modo, per i task di classificazione, il modello predice una singola parola corrispondente all'etichetta "target".

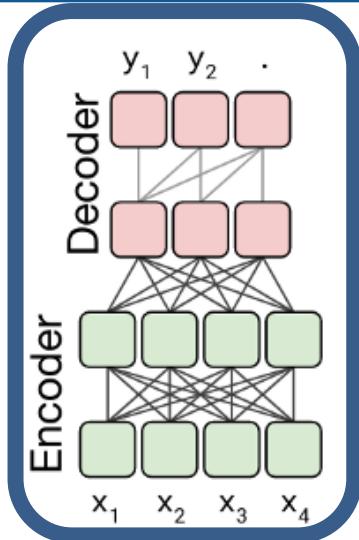
T5 (4)

- ▶ Anche T5 è addestrato con lo stesso obiettivo di BERT, ovvero il modello di linguaggio mascherato con una piccola modifica.
- ▶ La sottile differenza che il T5 impiega è quella di sostituire più token consecutivi con una singola parola chiave Mask, a differenza del BERT che usa il token Mask per ogni parola.
- ▶ Poiché l'obiettivo finale è quello di addestrare un modello che prende in input testo e emette testo, i target sono stati progettati per produrre una sequenza, a differenza di BERT, che cerca di emettere una parola attraverso una FFNN e un softmax a livello di output.

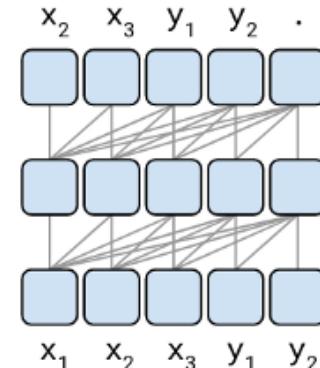


T5's mask language modeling (Raffel et al., 2019)

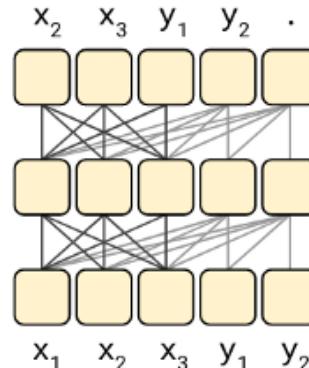
T5 (5)



Language model

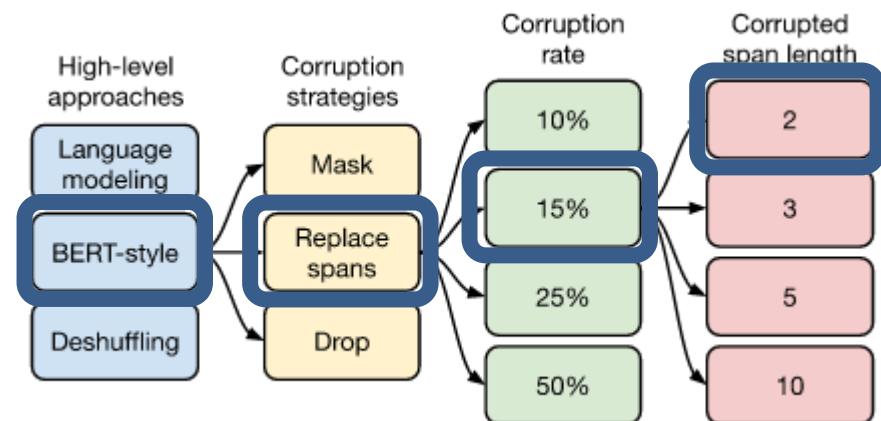


Prefix LM



Architetture di training

Tuning degli iperparametri



T5 (6)

Performance sulle varianti architetturali

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	<i>2P</i>	<i>M</i>	83.28	19.24	80.88	71.36	26.98	39.82	27.65
	Enc-dec, shared	<i>P</i>	<i>M</i>	82.81	18.78	80.63	70.73	26.72	39.03	27.46
	Enc-dec, 6 layers	<i>P</i>	<i>M/2</i>	80.88	18.97	77.59	68.42	26.38	38.40	26.95
	Language model	<i>P</i>	<i>M</i>	74.70	17.93	61.14	55.02	25.09	35.28	25.86
	Prefix LM	<i>P</i>	<i>M</i>	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	<i>2P</i>	<i>M</i>	79.56	18.59	76.02	64.29	26.27	39.17	26.86
	Enc-dec, shared	<i>P</i>	<i>M</i>	79.60	18.13	76.35	63.50	26.62	39.17	27.05
	Enc-dec, 6 layers	<i>P</i>	<i>M/2</i>	78.67	18.26	75.32	64.06	26.13	38.42	26.89
	Language model	<i>P</i>	<i>M</i>	73.78	17.54	53.81	56.51	25.23	34.31	25.38
	Prefix LM	<i>P</i>	<i>M</i>	79.68	17.84	76.87	64.86	26.28	37.51	26.76

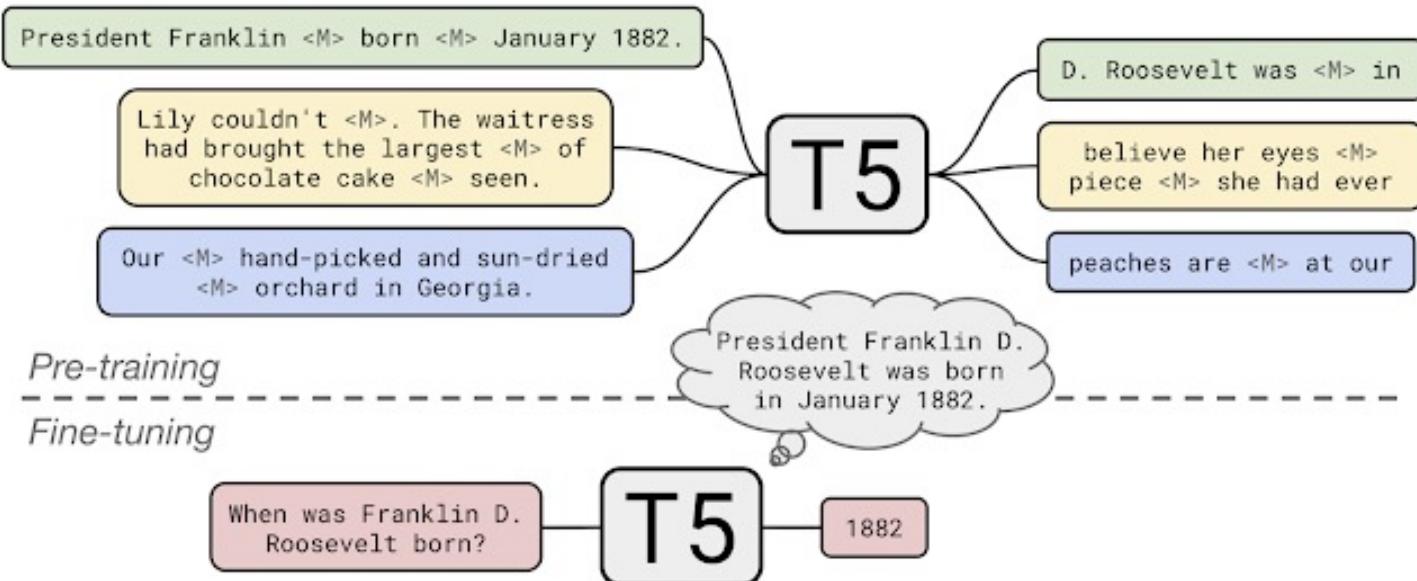
Performance sugli obiettivi di training

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you <M> <M> me to your party apple week .	(original text)
Deshuffling	party me for your to . last fun you inviting week Thank	(original text)
MASS-style Song et al. (2019)	Thank you <M> <M> me to your party <M> week .	(original text)
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	26.86	39.73	27.49
BERT-style (Devlin et al., 2018)	82.96	19.17	80.65	69.85	26.78	40.03	27.41
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62

T5 (7)

- ▶ T5 viene prima pre-addestrato sul dataset C4 per l'approccio di denoising (BERT-style), andando a mascherare span di token e con un'architettura Encoder-Decoder.
- ▶ Viene poi fatto fine-tuning sui compiti a valle tramite un approccio supervisionato, con un'appropriata modellazione degli input per l'impostazione text-to-text.



Machine Translation

Accedendo al link <https://huggingface.co/t5-base> potete testare in modo interattivo il modello T5 addestrato per un task di Machine Translation.

⚡ Hosted inference API ⓘ

Translation Examples

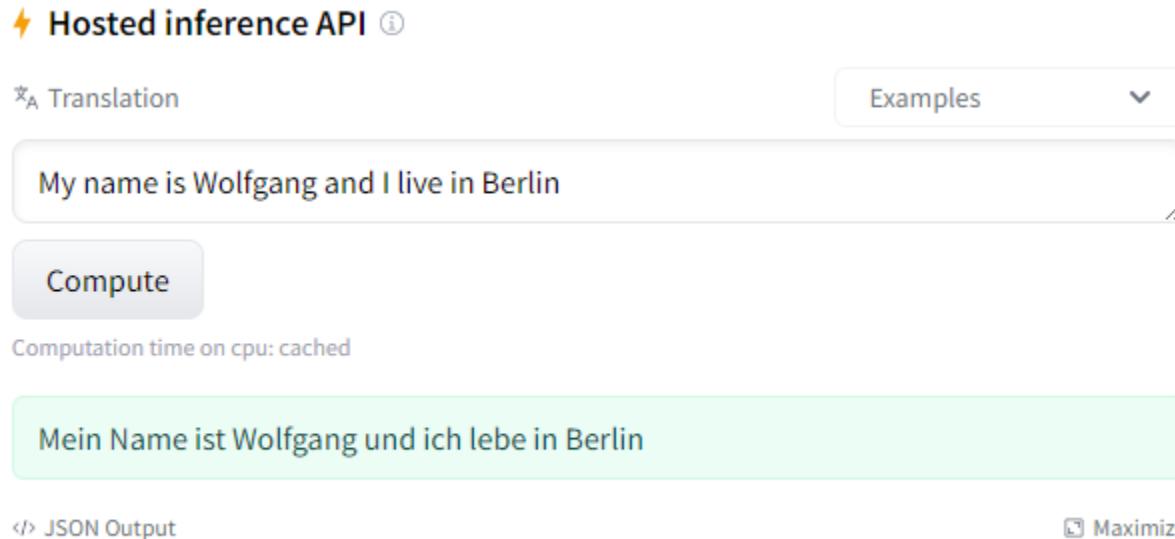
My name is Wolfgang and I live in Berlin

Compute

Computation time on cpu: cached

Mein Name ist Wolfgang und ich lebe in Berlin

</> JSON Output Maximize



Chat-GPT

Accedendo al link <https://chat.openai.com/> potete testare in modo interattivo il modello GPT-3 capace di generare testi molto complessi in risposta alle vostre richieste.



Linas Beliūnas
@linas.beliunas

Time it took to reach **1 million users**:

Netflix - 3.5 years

Airbnb - 2.5 years

Facebook - 10 months

Spotify - 5 months

Instagram - 2.5 months

iPhone - 74 days

ChatGPT - 5 days

ChatGPT is one of those rare moments in technology that **will reshape everything** going forward.

The screenshot shows two interactions with the AI:

Interaction 1: User asks for company name suggestions. AI responds with a list of 10 suggestions:

- 1. InnoVenture
- 2. NextGenTech
- 3. BlueOcean
- 4. BoldIdeas
- 5. BrightMind
- 6. FutureForward
- 7. VisionVenture
- 8. CleverCore
- 9. BoldThink
- 10. ProdigyWorks

Interaction 2: User asks for shorter and more memorable suggestions. AI responds with a new list of 10 suggestions:

- 1. InnoTech
- 2. BoldPro
- 3. BrightFlow
- 4. FutureFlex
- 5. CleverWise
- 6. ProdigyPrime
- 7. QuickThink
- 8. NextGenPro
- 9. BlueWave
- 10. VisionCore

Both interactions include a note from the AI: "These are just suggestions, and you can use them as a starting point to come up with your own unique name. Keep in mind that the name should reflect the mission and values of your company, as well as be memorable and easy to pronounce and spell. Good luck!"



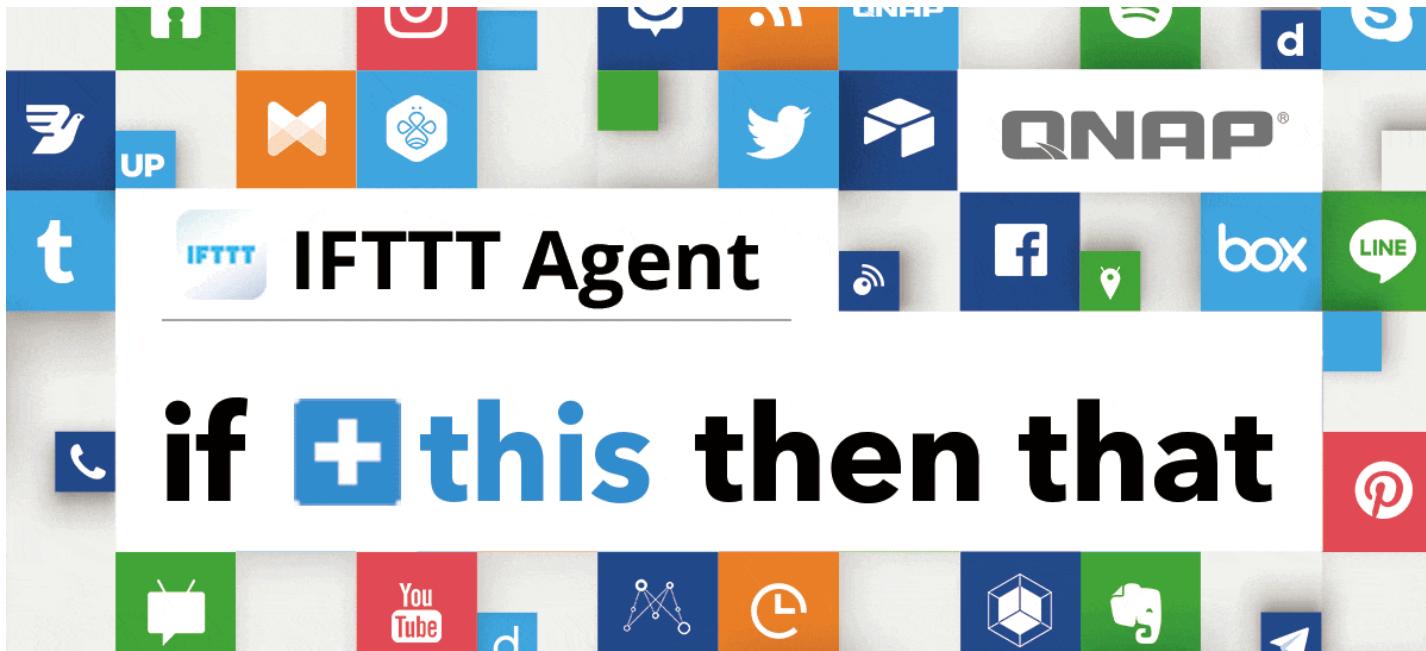
Intelligenza Artificiale

Proposte Tesi



Un caso applicativo IFTTT

IFTTT è una piattaforma di programmazione trigger-action in un dominio IoT.



Categorizzazione delle applet

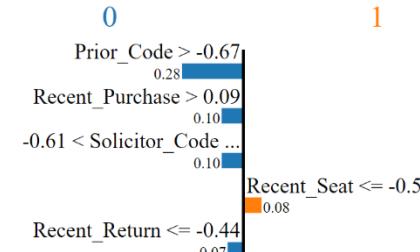
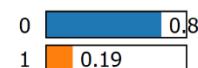
- ▶ **Innocua:** applet apparentemente innocua (a cui è stata associata la classe 0)
- ▶ **Personale:** causa la perdita di dati sensibili (a cui è stata associata la classe 1)
 - ▶ Questo danno è autoinflitto poiché qualsiasi danno è il risultato del comportamento dell'utente
- ▶ **Fisico:** danno alla salute fisica o beni (a cui è stata associata la classe 2)
 - ▶ Questo danno è esterno in quanto una third party può potenzialmente infliggere il danno
- ▶ **Cybersecurity:** interruzione di un servizio online o distribuzione di malware (a cui è stata associata la classe 3)
 - ▶ Questo danno è esterno in quanto una third party può potenzialmente infliggere il danno

Explainable AI (XAI)

- ▶ Il paradigma XAI si occupa di chiarire le ragioni che stanno alla base dei risultati di un modello di IA
- ▶ Quali sono le migliori tecniche che permettono ad un utente con uno scarso background tecnico di comprendere il danno segnalato?

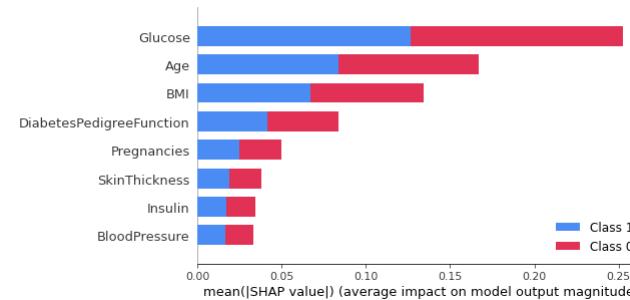


Prediction probabilities



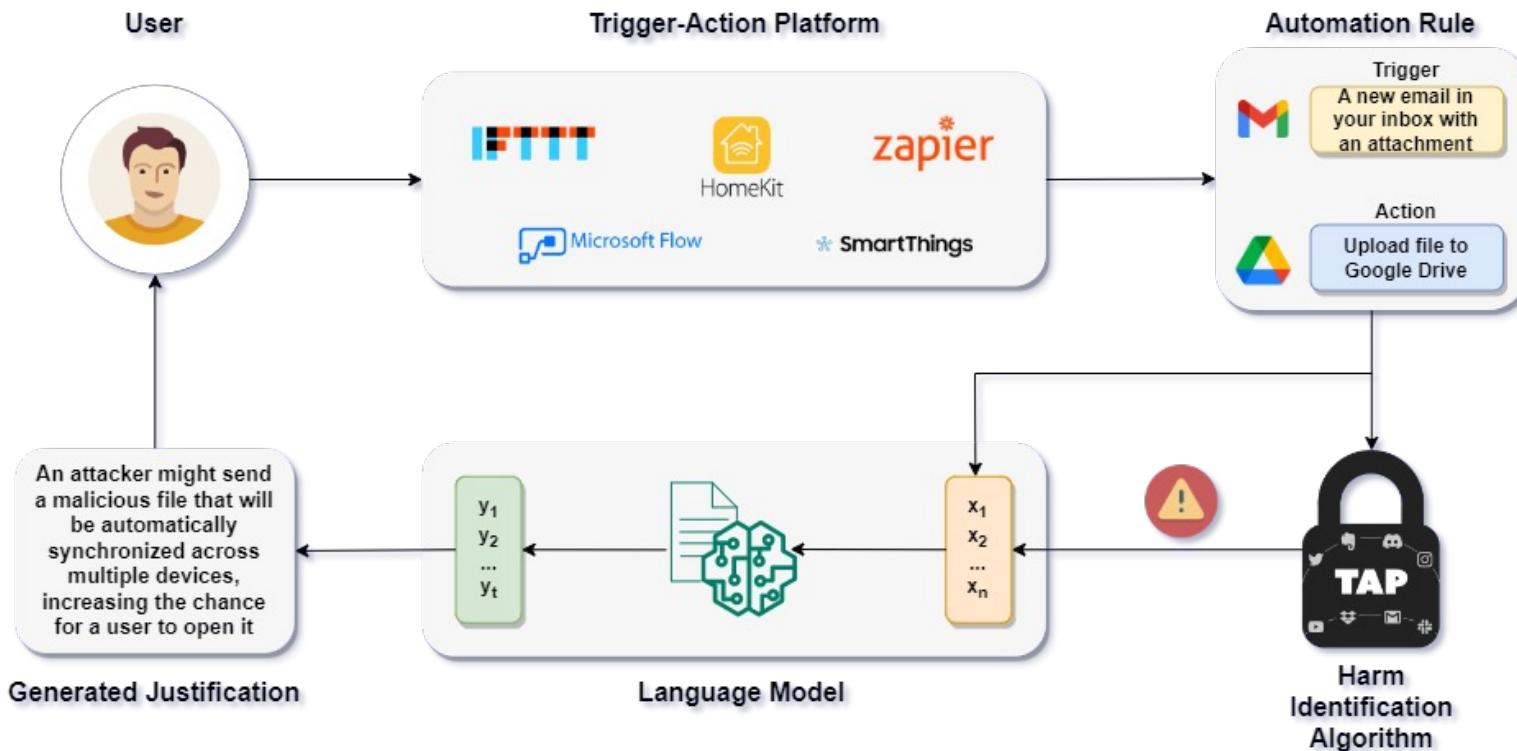
Feature	Value
Prior_Code	1.62
Recent_Purchase	0.36
Solicitor_Code	0.01
Recent_Seat	-0.53
Recent_Return	-0.44

Variable Importance Plot - Global Interpretation



Explainable AI with Natural Language Explanations (Natural-XAI)

- ▶ Gli approcci di Natural-XAI mirano a costruire modelli di IA in grado di generare frasi che giustifichino le predizioni



Explainable AI with Natural Language Explanations (Natural-XAI)

Natural Language Generation

T5

Enter Keywords:

India XCapital XNew Delhi XPress enter to add more

Generate text

Generated Sentence:

The capital of India is New Delhi.



GPT-2
SMALL

117M Parameters



GPT-2
MEDIUM

345M Parameters



GPT-2
LARGE

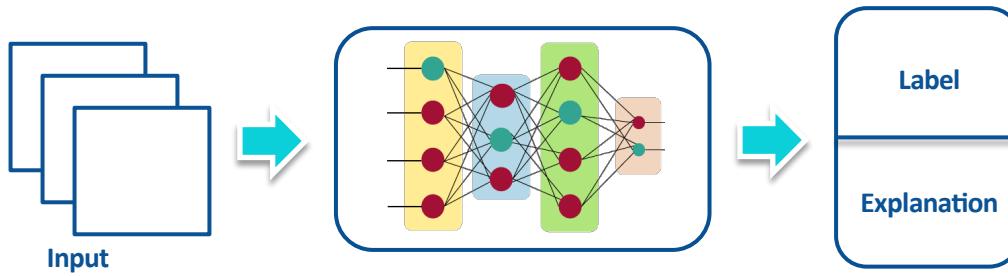
762M Parameters



GPT-2
EXTRA
LARGE

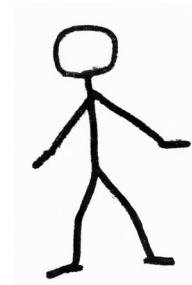
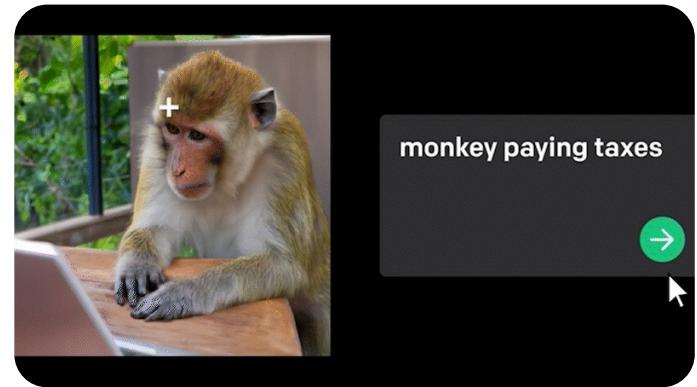
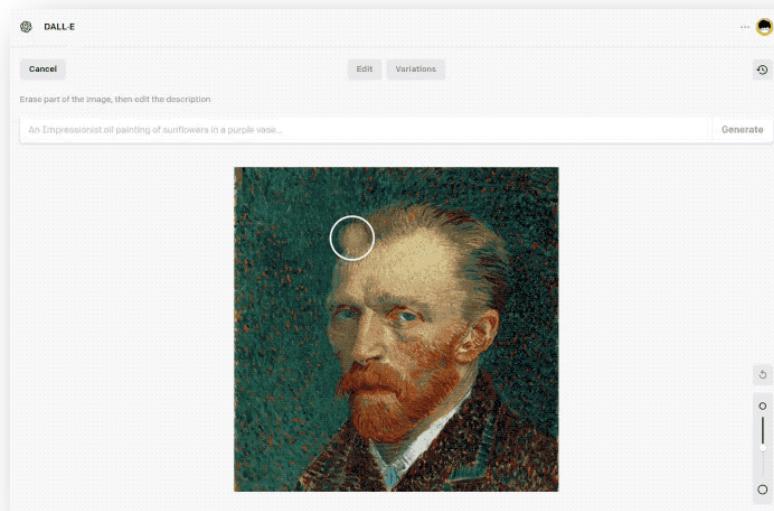
1,542M Parameters

Self-rationalization

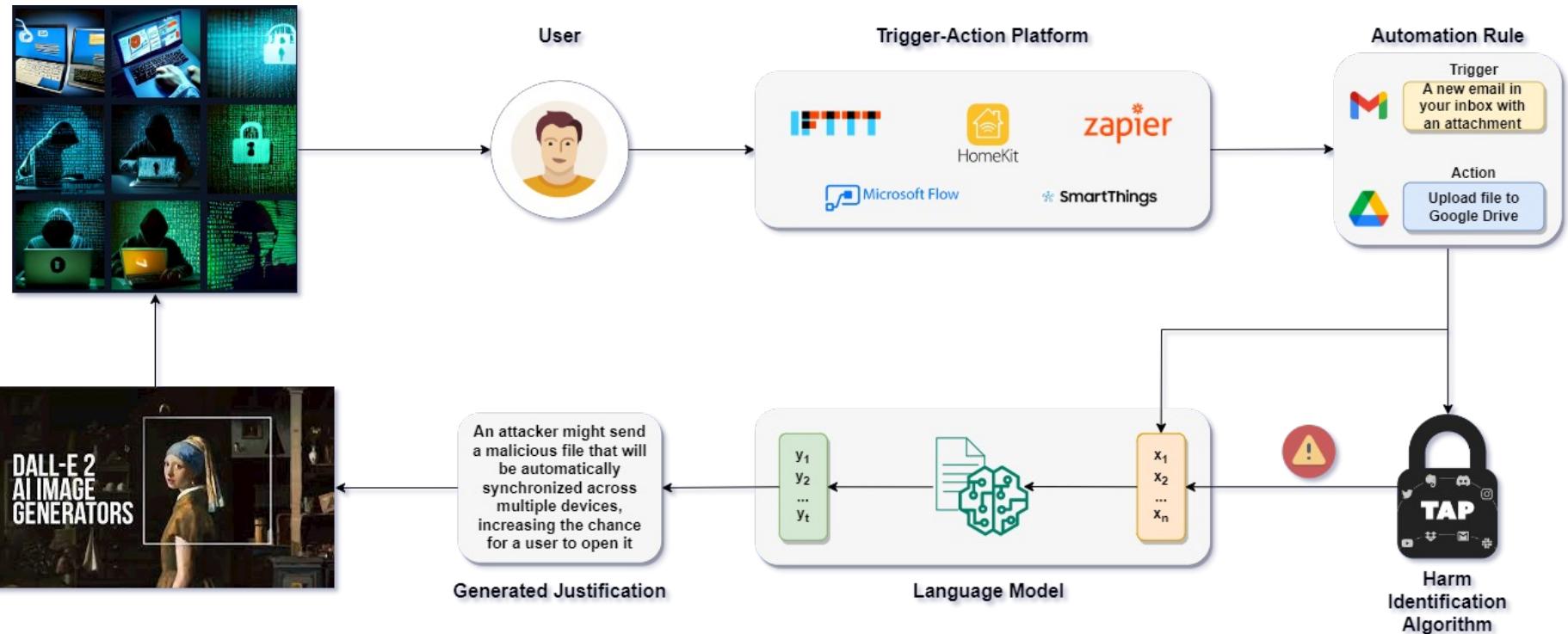


DALL-E 2

- ▶ DALL-E 2 è un sistema di IA in grado di creare immagini e opere d'arte realistiche da una descrizione in linguaggio naturale

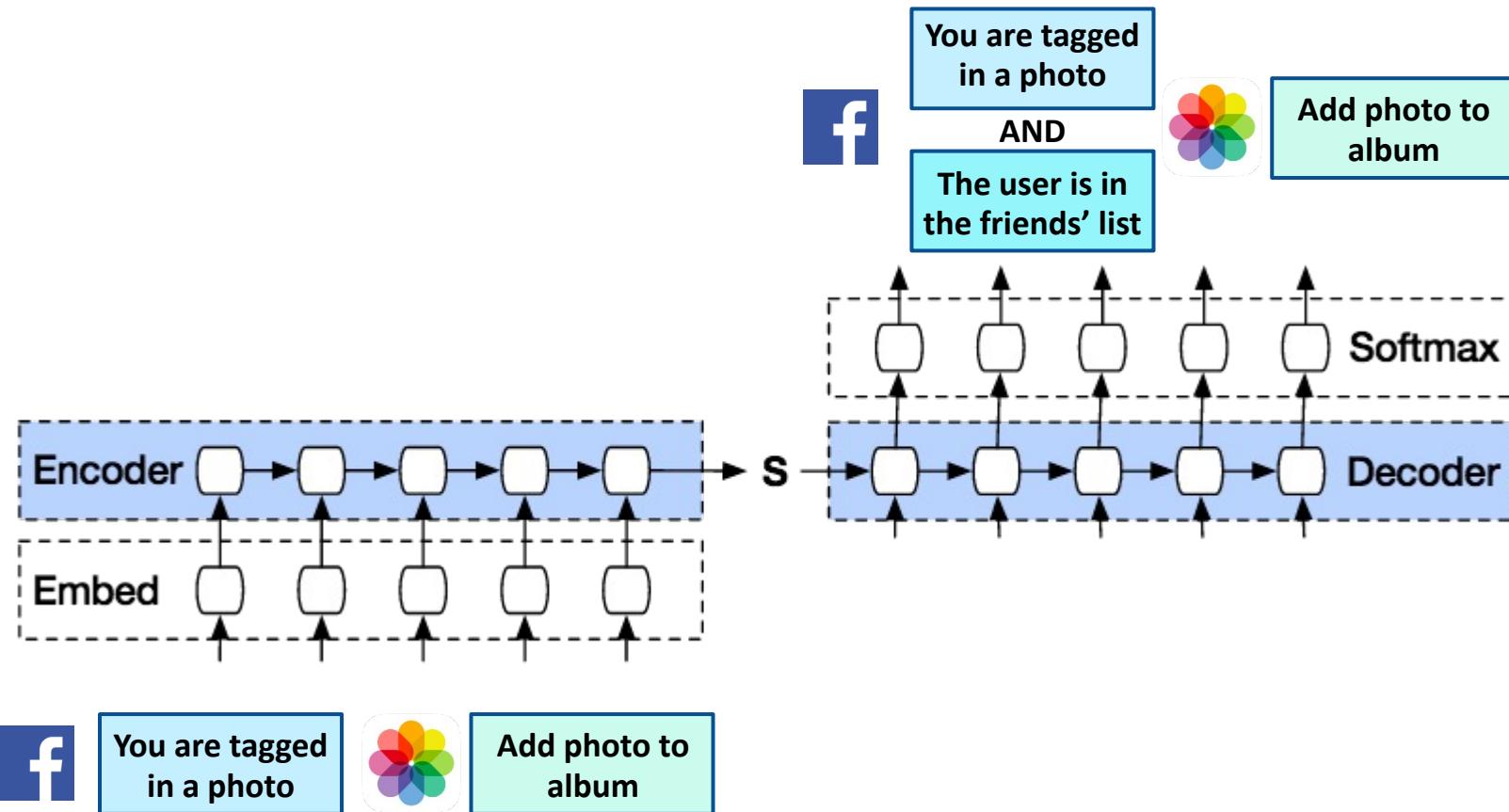


DALL-E 2



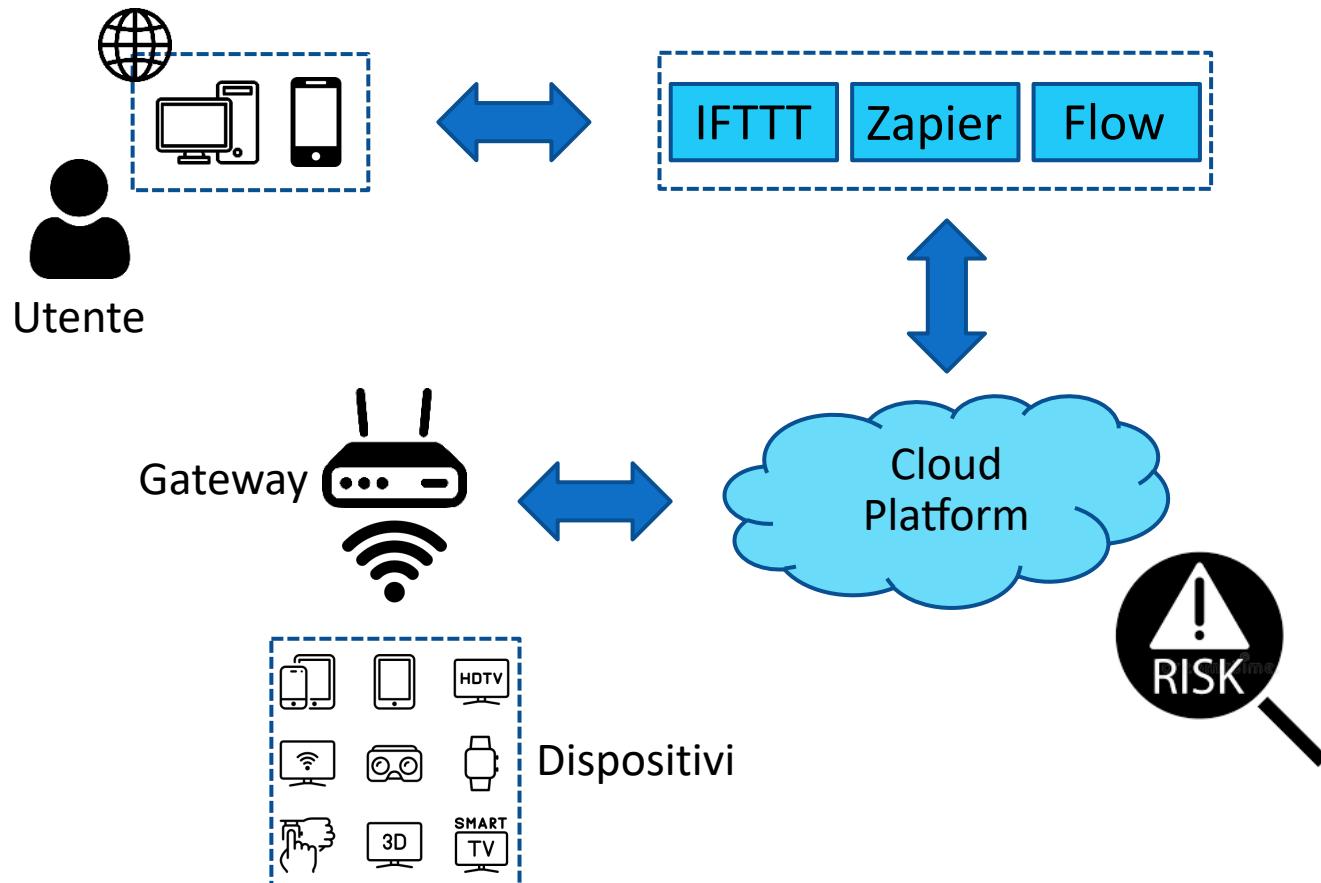
Mitigazione del rischio

- ▶ Implementazione di modelli sequence2sequence per ‘tradurre’ una regola che comporta un potenziale danno in una **sicura**



Identificazione di nuovi rischi

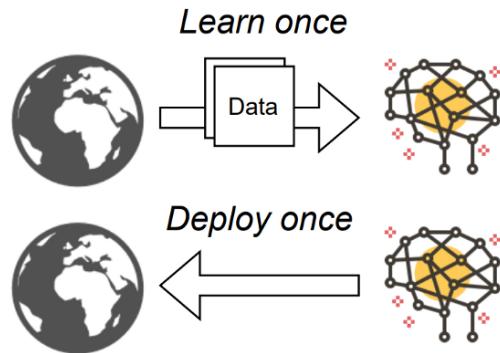
- ▶ Implementazione moduli di IA in grado di identificare rischi sia a design-time (quando una regola viene definita) che a run-time (quando una regola è già stata connessa)



Continual Learning

- ▶ Solitamente quando si acquisiscono nuovi dati l'approccio utilizzato è quello **cumulativo**
- ▶ Semplici approcci, come quello di reiterare il training di un modello su nuovi campioni, non funzionano nella pratica a causa di un problema denominato **catastrophic forgetting**

Static ML



Adaptive ML

