

```
linuxday25@poul:~$ python
```

```
build_your_transformer.py
```

Link to code 

linuxday25@poul:~\$ man Questo Talk

## @ Perché?

- ~ Questa tecnologia sta entrando nelle nostre vite e dispositivi
- ~ Per costruire ed usare applicazioni open con questi modelli è importante conoscerli (almeno un pochino)

## @ Come?

- ~ Teoria:
  - \$ Machine Learning
  - \$ Transformer Block
- ~ Pratica:
  - \$ Implementazione in PyTorch (no training)

```
linuxday25@poul:~$ man Machine Learning
```

## @ Codice

~ Ingredienti:

\$ Conoscenza del programmatore

~ Risultato:

\$ Il programmatore impartisce alla macchina le operazioni da compiere per risolvere il problema

## @ Machine Learning

~ Ingredienti:

\$ Esempi di input – output

\$ Un modello e come addestrarlo

~ Risultato:

\$ Il modello impara a produrre l'output atteso dato un input





Come **parla** un Transformer?

```
linuxday25@poul:~$ man Rappresentazione
```

## @ Immagini

- ~ Pixel
- ~ **RGB**, Tre numeri identificano univocamente un colore!

## @ Testo

- ~ Parole? E se faccio errori?
- ~ Vettori multidimensionali? Ma che valore posso dare ad concetto?

```
linuxday25@poul:~$ man Rappresentazione
```

## @ Immagini

- ~ Pixel
- ~ **RGB**, Tre numeri identificano univocamente un colore!

## @ Testo

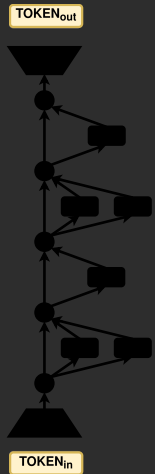
- ~ **Parole? E se faccio errori?**
- ~ Vettori multidimensionali? Ma che valore posso dare ad concetto?

```
linuxday25@poul:~$ man Tokenizzazione
```

## @ Token

- ~ Un Token può essere visto come un “**Pixel per il Testo**”
- ~ É una **sotto-parola**:
  - \$ Evita il problema degli errori grammaticali, sarebbe impossibile avere una rappresentazione univoca per ogni possibile errore grammaticale
  - \$ Permette di assegnare rappresentazioni parzialmente condivise tra parole e le loro radici. Ad esempio Tokenizer è diviso in Token-izer

```
linuxday25@poul:~$ python Tokenizzazione.py
```



[Code Session]

```
linuxday25@poul:~$ man Rappresentazione
```

## @ Immagini

- ~ Pixel
- ~ **RGB**, Tre numeri identificano univocamente un colore!

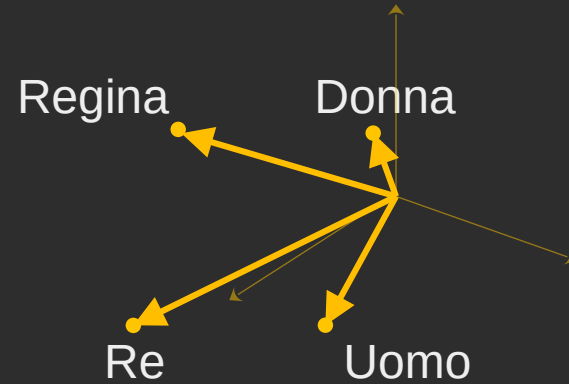
## @ Testo

- ~ Parole? E se faccio errori?
- ~ **Vettori multidimensionali? Ma che valore posso dare ad concetto?**

# linuxday25@poul:~\$ man Embedding

## @ Embedding

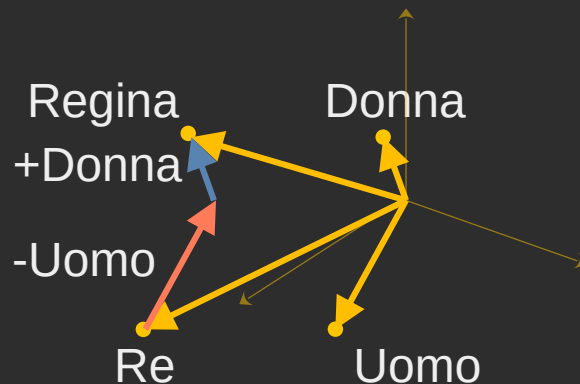
- ~ Un Token viene associato ad una rappresentazione in uno spazio multi-dimensionale
- ~ **Angolo Tra Vettori = Somiglianza di Significato**



# linuxday25@poul:~\$ man Embedding

## @ Embedding

- ~ Un Token viene associato ad una rappresentazione in uno spazio multi-dimensionale
- ~ **Angolo Tra Vettori = Somiglianza di Significato**
- ~ Bag of Word e Contesto



# linuxday25@poul:~\$ man Decoding

## @Language Modeling Head

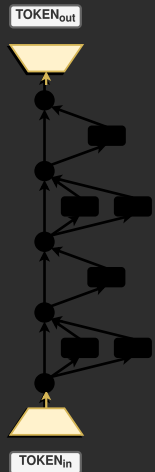
- ~ La LMH è il componente responsabile di effettuare l'operazione di un-embedding
- ~ E' una mappa dallo spazio vettoriale usato dal modello al vocabolario del tokenizer

## @ Scelta del Token

- ~ Il risultato ritornato dalla LMH è un “punteggio” per ogni prossimo token
- ~ Con una SoftMax possiamo ottenere una distribuzione probabilistica

$$\text{SoftMax}(\vec{x}_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

```
linuxday25@poul:~$ python Embed&Decode.py
```



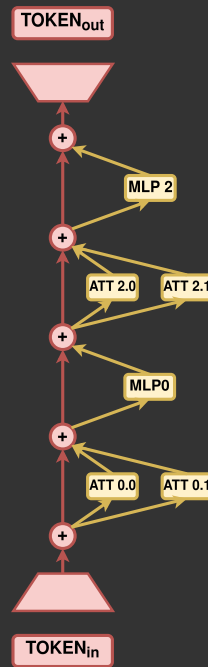
[Code Session]

# Dentro un Transformer

linuxday25@poul:~\$ man Transformer

## @ Residuo

- ~ Tutti i componenti leggono il loro input e scrivono il loro output su un residuo comune
- ~ All'inizio il residuo rappresenta l'input, alla fine il token successivo



## @ Componenti

- ~ In una struttura modulare in blocchi (layer)
- ~ Ogni modello presenta Normalizzazione, Attenzione e Multi-Layer Perceptron

linuxday25@poul:~\$ man MLPs

## @ Come?

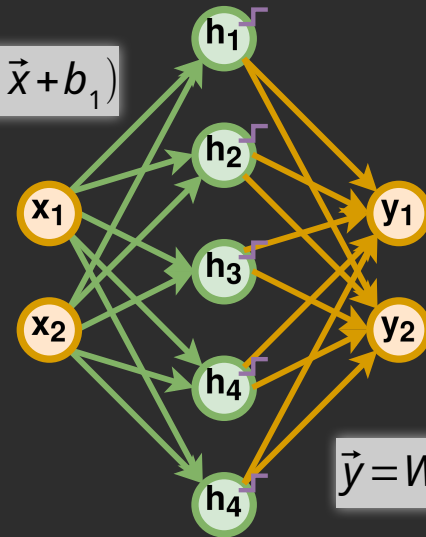
~ Trasformazioni lineari

$$\vec{y} = W \vec{x} + b$$

~ Intervallate da funzioni di attivazione  
non-lineari

$$y = GELU(x)$$

$$\vec{h} = GELU(W_1 \vec{x} + b_1)$$



$$\vec{y} = W_2 \vec{h} + b_2$$

```
linuxday25@poul:~$ man MLPs
```

### @ Teoria

- ~ Un MLP a singolo layer sufficientemente largo può approssimare qualsiasi funzione
- ~ Creare più layer riduce la quantità di neuroni richiesti

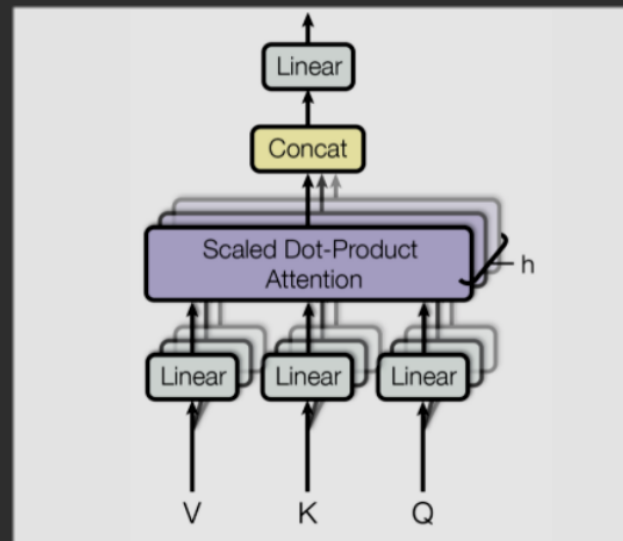
### @ Negli LLM

- ~ Svolgono il ruolo di essere la “memoria” dell’LLM (*probabilmente*)
- ~ Compongono gran parte dei parametri del modello
- ~ Intervengono solo sul token corrente

linuxday25@poul:~\$ man **Attenzione**

## @ Pratica

- ~ É l'elemento che permette al modello di “ragionare”
- ~ Raccoglie informazioni dai residui associati a **TUTTI I TOKEN!**



linuxday25@poul:~\$ man **Attenzione**

## @ Come?

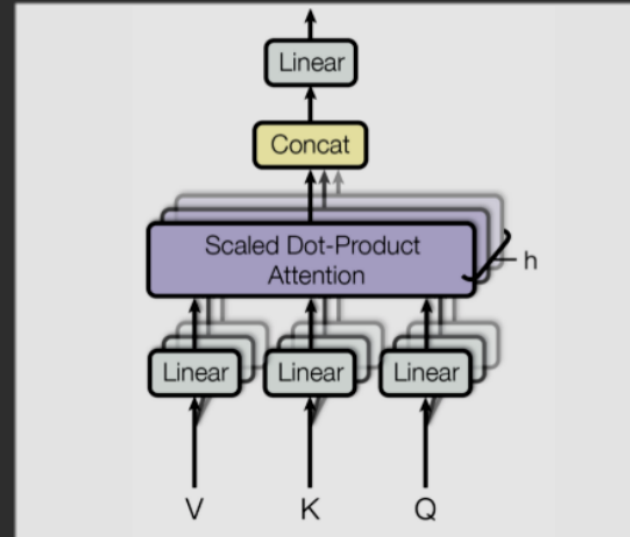
~ Proietta i residui in tre spazi vettoriali:

\$ **Query**

\$ **Key**

\$ **Value**

~ Copia i valori sulla base di  
corrispondenza tra key e query



linuxday25@poul:~\$ man **Attenzione**

## @ Intuizione

~ Prompt: “Le mele sono **verdi**, la mela è di **colore** \_\_\_\_”

~ Query = “cosa cerco”

→ pos.` colore`: “Il colore della mela”

~ Key = “cosa sono”

→ pos.` verdi`: “io sono colore delle mele”

~ Value = “che informazione porto”

→ pos.` verdi`: “verde”

Query e Key sono molto simili e quindi voglio copiare l'informazione associata a ` verdi` nel residuo corrente in modo da “raccolgere” l'informazione sul colore

linuxday25@poul:~\$ man **Attenzione**

## @ Come?

~ Proietta i residui in ogni posizione  $i$  in tre spazi vettoriali usando 3 layer lineari:

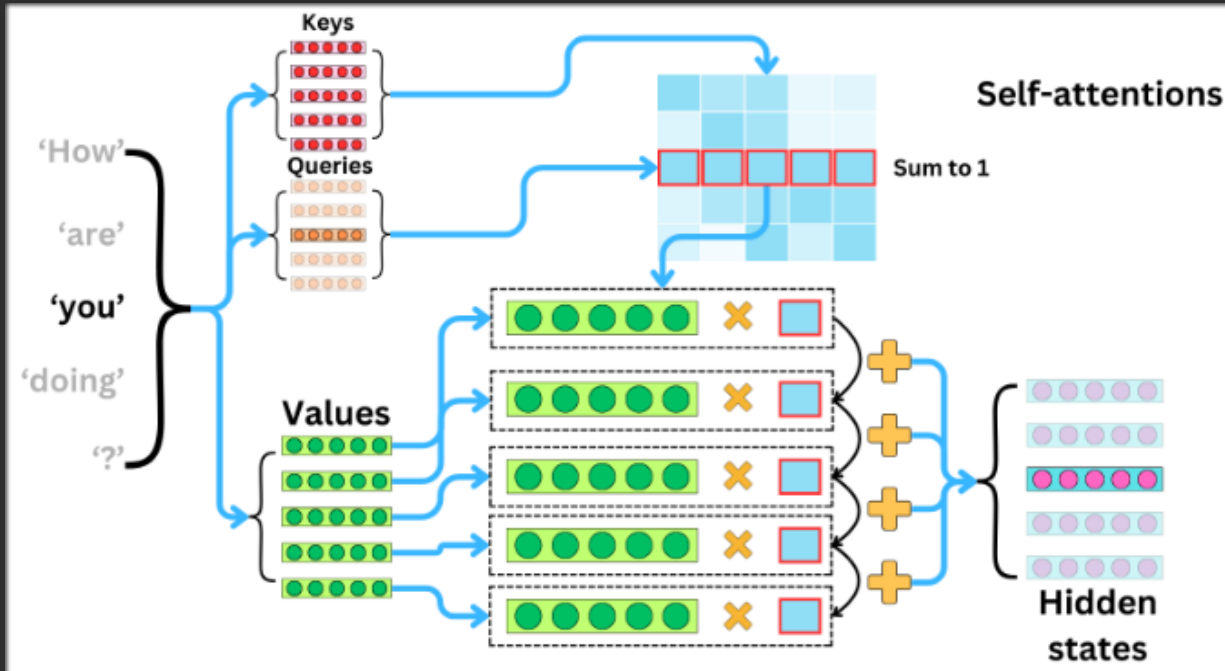
$$\mathbf{query}_i = \mathbf{W}_q \cdot \mathbf{r}_i + \mathbf{b}, \quad \mathbf{key}_i = \mathbf{W}_k \cdot \mathbf{r}_i + \mathbf{b}_k, \quad \mathbf{value}_i = \mathbf{W}_v \cdot \mathbf{r}_i + \mathbf{b}_v$$

~ Copia i valori sulla base di corrispondenza tra key e query:

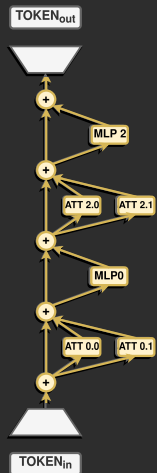
1. Calcola la matrice delle **cosine similarities** (  $\text{cosim}_{ij}(q_i, v_j) = (q_i \cdot v_j) / (\|q_i\| \cdot \|v_j\|)$  )
2. Per ogni query applica una softmax ottenendo la matrice degli **scores**
3. Copia nel residuo *di*  $q_i$ , i valori  $v_j$  per ogni  $j \leq i$  pesandoli in base a  $\text{scores}[i, j]$

linuxday25@poul:~\$ man Attenzione

@ Come?



```
linuxday25@poul:~$ python GPTux.py
```



[Code Session]

linuxday25@poul:~\$ man Training

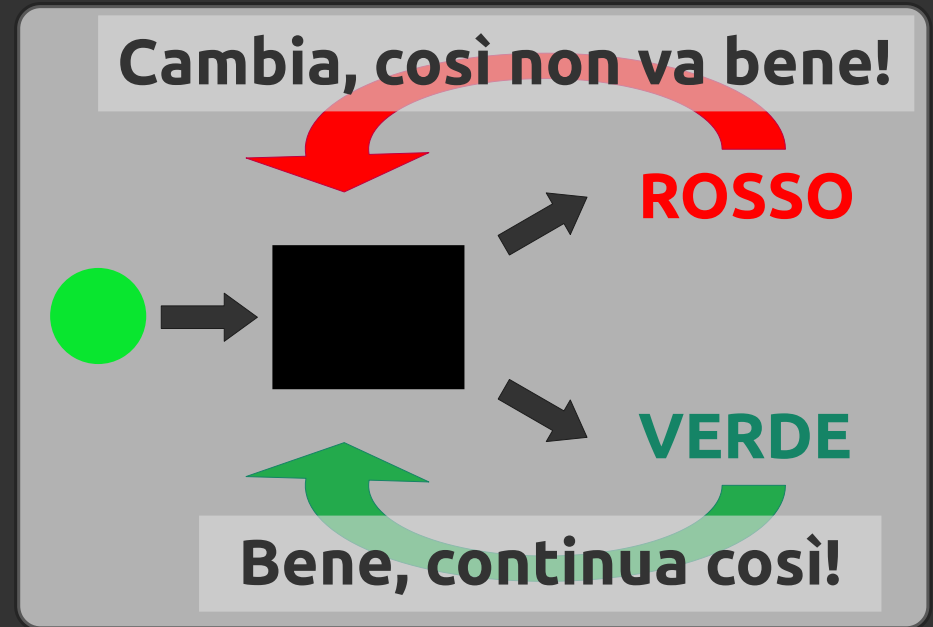
## @ Machine Learning

### ~ Ingredienti:

- \$ Esempi di input – output
- \$ Un modello e come addestrarlo

### ~ Risultato:

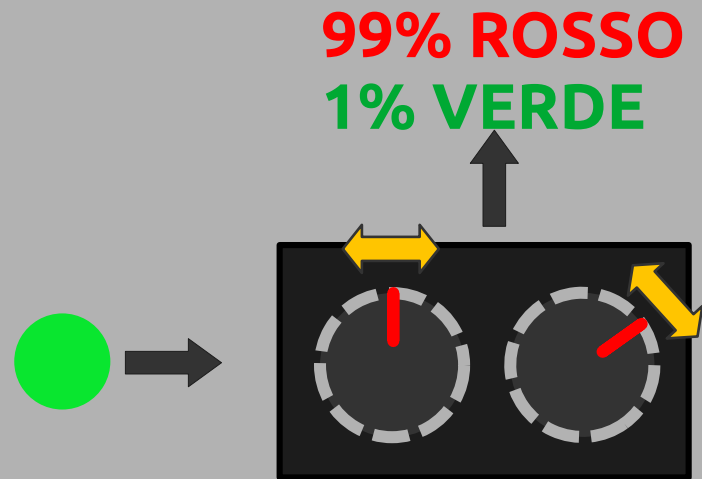
- \$ Il modello impara a produrre l'output atteso dato un input



linuxday25@poul:~\$ man **Back Propagation**

### @ Idea

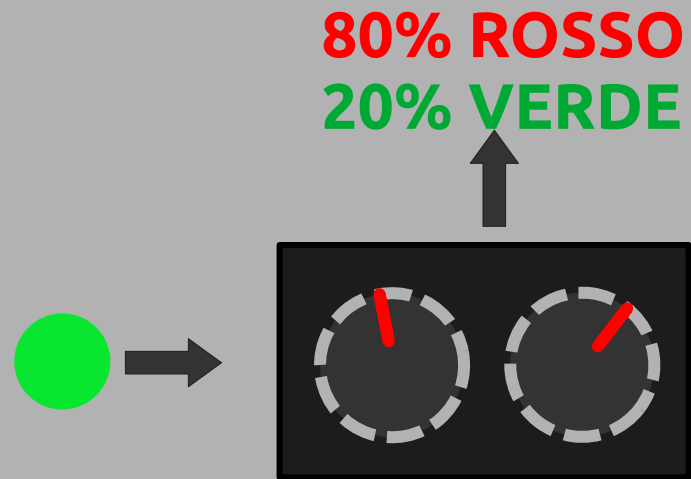
- ~ Il modello al suo interno ha tante manopole
- ~ Non sapendo cosa fa una manopola la giro un pochino a destra e sinistra vedendo se il modello fa meglio o peggio (calcolo la derivata discreta!)



linuxday25@poul:~\$ man **Back Propagation**

### @ Idea

- ~ Il modello al suo interno ha tante manopole
- ~ Non sapendo cosa fa una manopola la giro un pochino a destra e sinistra vedendo se il modello fa meglio o peggio (calcolo la derivata discreta!)



```
linuxday25@poul:~$ python Training.py
```

**[Code Session]**

[Grazie per l' **Attenzione**]