

Deep Learning & Applied AI

Self-attention and transformers

Emanuele Rodolà
rodola@di.uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA

Sequential data

Example: numeric 1D sequential data ([time series](#))



Per dati sequenziali si intendono tutti quei dati che hanno una continuità spaziale e/o temporale, come in questo caso

Sequential data

Example: numeric 1D sequential data ([time series](#))



Prototypical task: predict the [next numbers](#) in the sequence

Sequential data

Example: Brownian motion of a particle in 3D space

Sequential data

Example: 3D shape motions



Quindi un task potrebbe essere riconoscere l'animazione come comportamento

Sequential data

Example: 3D shape motions



Prototypical task: **classify** the entire sequence (e.g. “running”)

Sequential data

Example: Text ([symbolic](#))

“the little brown fox”

the, little, brown, fox

t, h, e, , l, i, t, t, l, e, , b, r, o, w, n, , f, o, x

Sequential data

Example: Text (symbolic) simbolico perché sta al designer scegliere il vocabolario da utilizzare (parole composte, lettere, segni, ecc..)

“the little brown fox”

the, little, brown, fox vocabolario di parole

t, h, e, , l, i, t, t, l, e, , b, r, o, w, n, , f, o, x

vocabolario di lettere

Prototypical task: text translation (e.g. “茶色の小狐”)

Sequence-to-sequence model

Key property:

The **same weights** apply to sequences of **different lengths**.

Sequence-to-sequence model

Key property:

The [same weights](#) apply to sequences of [different lengths](#).

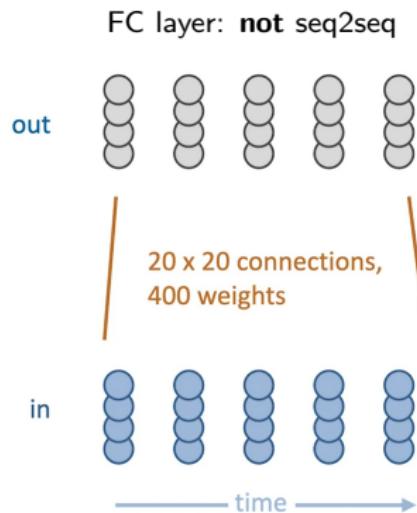
If the output is another sequence, it's a [sequence-to-sequence](#) model.

Sequence-to-sequence model

Key property:

The **same weights** apply to sequences of **different lengths**.

If the output is another sequence, it's a **sequence-to-sequence** model.

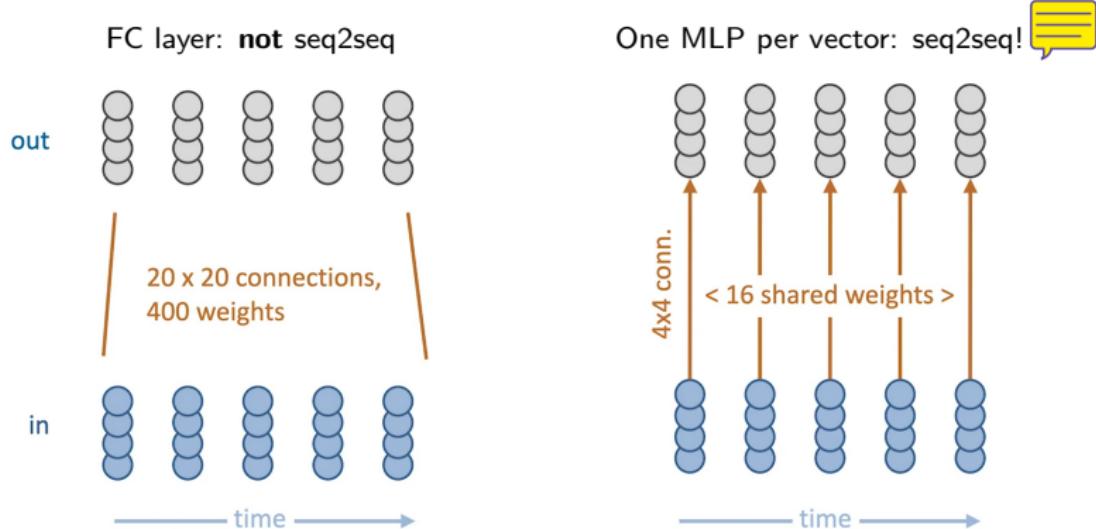


Sequence-to-sequence model

Key property:

The **same weights** apply to sequences of **different lengths**.

If the output is another sequence, it's a **sequence-to-sequence** model.

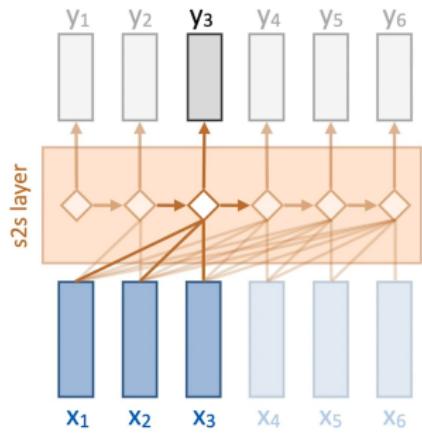


Causal vs. non-causal layers

S2S layers admit input and output with different lengths.

Causal vs. non-causal layers

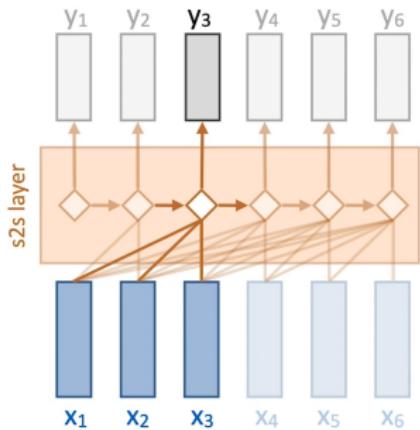
S2S layers admit input and output with different lengths.



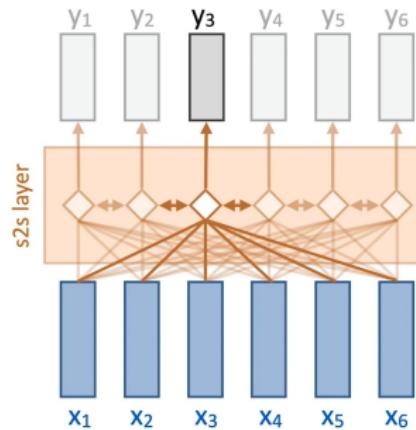
"causal" layer
can only look backward in time

Causal vs. non-causal layers

S2S layers admit input and output with different lengths.



“causal” layer
can only look backward in time

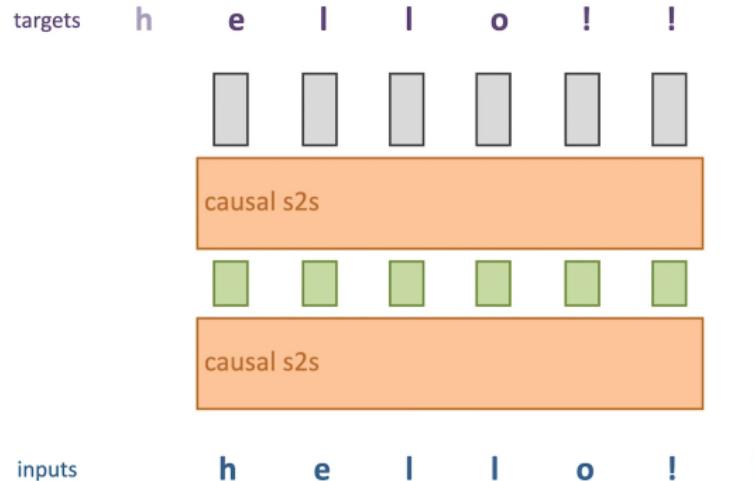


“non-causal” layer
no restrictions

nel modello causale un output può essere generato
solo da input precedenti a livello temporale

Autoregressive modeling

Given an unlabeled dataset of sequences, take a subsequence and train to predict the [next character](#):

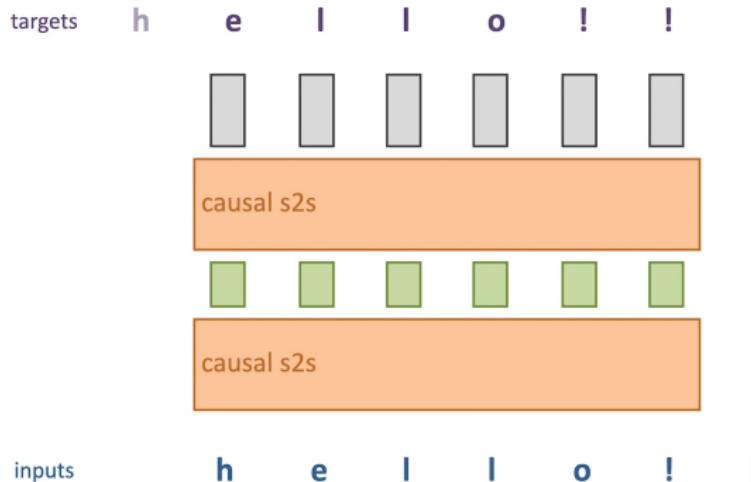


Autoregressive modeling



casi d'uso: generazione del testo

Given an unlabeled dataset of sequences, take a subsequence and train to predict the [next character](#):



Since **causal** layers are used, the model can not “cheat” by looking ahead in the sequence.

Autoregressive modeling

The model outputs, for each token, a **probability distribution** over the set of symbols (e.g. over the letters of the alphabet).

Once trained, one can generate sequences by **sequential sampling**:

Input: seed $[s_1, s_2, s_3, \dots]$

Output: distribution $p(c \mid s_1, s_2, s_3, \dots)$

Autoregressive modeling

The model outputs, for each token, a **probability distribution** over the set of symbols (e.g. over the letters of the alphabet).

Once trained, one can generate sequences by **sequential sampling**:

Input: seed $[s_1, s_2, s_3, \dots]$

Output: distribution $p(c \mid s_1, s_2, s_3, \dots)$

- Sample the output distribution
- Append the sampled symbol to the seed
- Iterate

Autoregressive modeling

The model outputs, for each token, a **probability distribution** over the set of symbols (e.g. over the letters of the alphabet).

Once trained, one can generate sequences by **sequential sampling**:

Input: seed $[s_1, s_2, s_3, \dots]$ la sottostringa

Output: distribution $p(c \mid s_1, s_2, s_3, \dots)$

- Sample the output distribution
- Append the sampled symbol to the seed
- Iterate

“I like to eat hot  dogs pancakes”

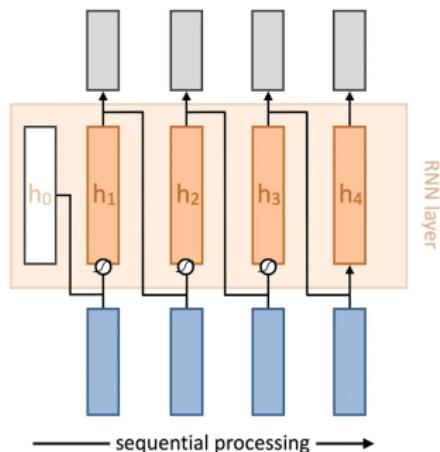
Sequence-to-sequence layers

How do we actually design a S2S layer?

Sequence-to-sequence layers

How do we actually design a S2S layer?

Many possible choices.



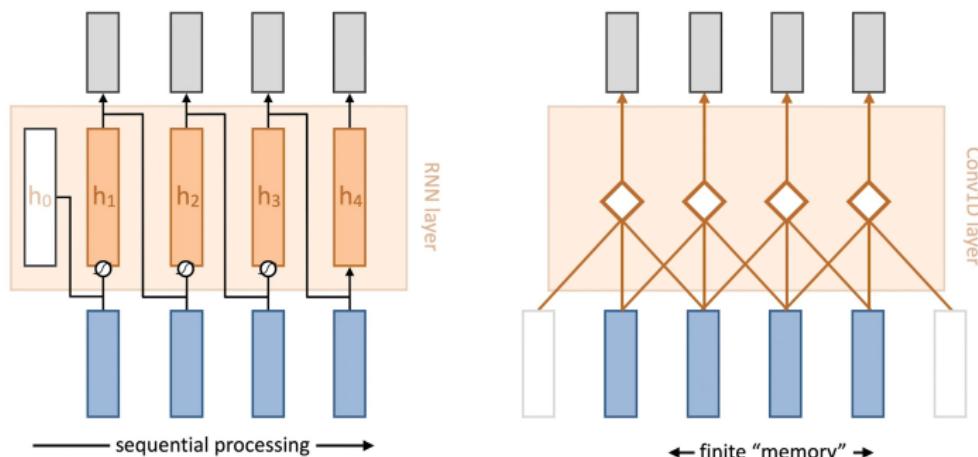
can look indefinitely far back in the sequence
processing must be done sequentially

Sequence-to-sequence layers

How do we actually design a S2S layer?

Many possible choices.

è possibile ottenere entrambe le caratteristiche positive?...



can look indefinitely far back in the sequence
processing must be done sequentially

limited range backward or forward
parallel computation

Self-attention

Can we have both, parallel computation and long dependencies?

Self-attention

Can we have both, parallel computation and long dependencies?

Consider a linear model from **input** to **output**:

$$y_i = \sum_j w_{ij} x_j$$

Self-attention

Can we have both, parallel computation and long dependencies?

Consider a linear model from **input** to **output**:

$$y_i = \sum_j w_{ij} x_j$$

However, the w_{ij} are **not** trainable weights.

Self-attention

Can we have both, parallel computation and long dependencies?

Consider a linear model from **input** to **output**:

$$\textcolor{brown}{y}_i = \sum_j \textcolor{brown}{w}_{ij} \textcolor{brown}{x}_j$$

However, the $\textcolor{brown}{w}_{ij}$ are **not** trainable weights.

Instead, for a given position i , compute the correlations over all j :

$$\textcolor{brown}{w}'_{ij} = \textcolor{brown}{x}_i^\top \textcolor{brown}{x}_j$$

Self-attention ... si con questi modelli

Can we have both, parallel computation and long dependencies?

Consider a linear model from **input** to **output**:

$$y_i = \sum_j w_{ij} x_j$$

quindi in questo caso, y_i è la parola che vogliamo trovare e l'insieme delle x_j sono le parole che costituiscono la sequenza X in input

However, the w_{ij} are **not** trainable weights.

Instead, for a given position i , compute the correlations over all j :

$$w'_{ij} = x_i^\top x_j$$

In questo modo, w_{ij} rappresenta la probabilità che le posizioni j abbiano una relazione con la posizione i

and transform them so that each $w_{ij} > 0$ and $\sum_j w_{ij} = 1$:

$$w_{ij} = \frac{e^{w'_{ij}}}{\sum_j e^{w'_{ij}}} \quad \text{softmax}$$

quindi la somma delle attenzioni per una parola x_i rispetto a tutte le altre x_j è uguale a 1

riprendendo l'esempio delle parole, ogni carattere viene messo in correlazione con ogni altro carattere per vedere quanto si influenzano

Self-attention

In matrix notation:

$$\textcolor{brown}{w}'_{ij} = \textcolor{blue}{x}_i^\top \textcolor{blue}{x}_j \Rightarrow \mathbf{W}' = \mathbf{X}^\top \mathbf{X}$$

Self-attention

In matrix notation:

$$\mathbf{w}'_{ij} = \mathbf{x}_i^\top \mathbf{x}_j \Rightarrow \mathbf{W}' = \mathbf{X}^\top \mathbf{X}$$

$$w_{ij} = \frac{e^{\mathbf{w}'_{ij}}}{\sum_j e^{\mathbf{w}'_{ij}}} \Rightarrow \mathbf{W} = \begin{pmatrix} & & & \vdots \\ & & \text{softmax}(-\mathbf{w}'_{i:} -) & \\ & & & \vdots \end{pmatrix}$$

Self-attention

In matrix notation:

$$\textcolor{brown}{w}'_{ij} = \textcolor{blue}{x}_i^\top \textcolor{blue}{x}_j \Rightarrow \mathbf{W}' = \mathbf{X}^\top \mathbf{X}$$

$$w_{ij} = \frac{e^{\textcolor{brown}{w}'_{ij}}}{\sum_j e^{\textcolor{brown}{w}'_{ij}}} \Rightarrow \mathbf{W} = \begin{pmatrix} & & & \vdots \\ & \text{softmax}(-\textcolor{brown}{w}'_{i:} -) & & \\ & & & \vdots \end{pmatrix}$$

$$y_i = \sum_j \textcolor{brown}{w}_{ij} \textcolor{blue}{x}_j \Rightarrow \mathbf{Y}^\top = \mathbf{W} \mathbf{X}^\top$$

Self-attention

In matrix notation:

$$\mathbf{w}'_{ij} = \mathbf{x}_i^\top \mathbf{x}_j \Rightarrow \mathbf{W}' = \mathbf{X}^\top \mathbf{X}$$

$$\mathbf{w}_{ij} = \frac{e^{\mathbf{w}'_{ij}}}{\sum_j e^{\mathbf{w}'_{ij}}} \Rightarrow \mathbf{W} = \begin{pmatrix} & & & \vdots \\ & \text{softmax}(-\mathbf{w}'_{i:} -) & & \\ & & & \vdots \end{pmatrix}$$

$$y_i = \sum_j \mathbf{w}_{ij} \mathbf{x}_j \Rightarrow \mathbf{Y}^\top = \mathbf{W} \mathbf{X}^\top$$

Matrix \mathbf{W} is diagonally dominant, since typically $\mathbf{w}'_{ii} \geq \mathbf{w}'_{ij}$.

Self-attention

In matrix notation:

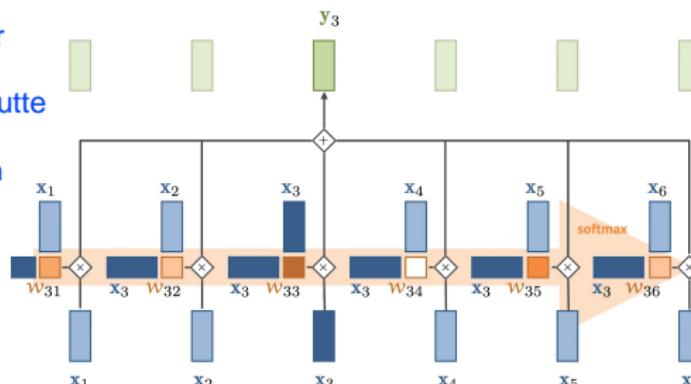
$$\mathbf{w}'_{ij} = \mathbf{x}_i^\top \mathbf{x}_j \Rightarrow \mathbf{W}' = \mathbf{X}^\top \mathbf{X}$$

$$w_{ij} = \frac{e^{\mathbf{w}'_{ij}}}{\sum_j e^{\mathbf{w}'_{ij}}} \Rightarrow \mathbf{W} = \begin{pmatrix} & & & & \vdots & \\ & \text{softmax}(-\mathbf{w}'_{i:} -) & & & & \\ & & & & & \\ & & & & & \vdots \\ & & & & & \end{pmatrix}$$

somma ponderata
dei pesi per l'output $y_i = \sum_j w_{ij} x_j \Rightarrow \mathbf{Y}^\top = \mathbf{W} \mathbf{X}^\top$

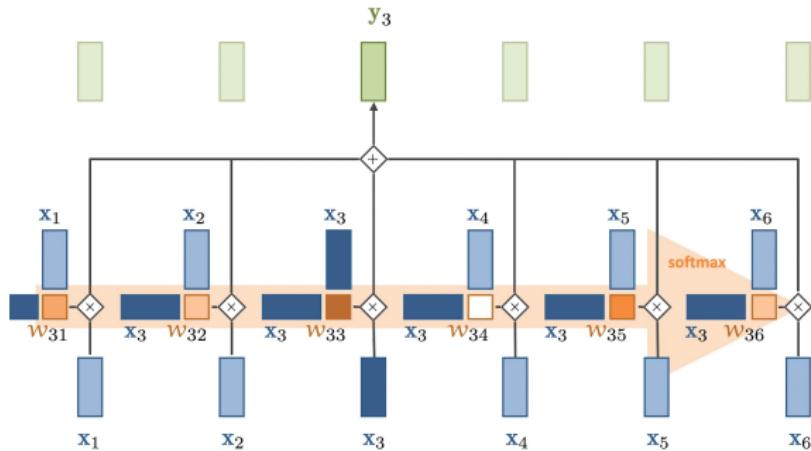
Matrix \mathbf{W} is diagonally dominant, since typically $w'_{ii} \geq w'_{ij}$.

quindi il dot product per
una parte di sequenza
in input viene fatto tra tutte
le altre parti disponibili
(quindi i layer sono non
causali perché...)



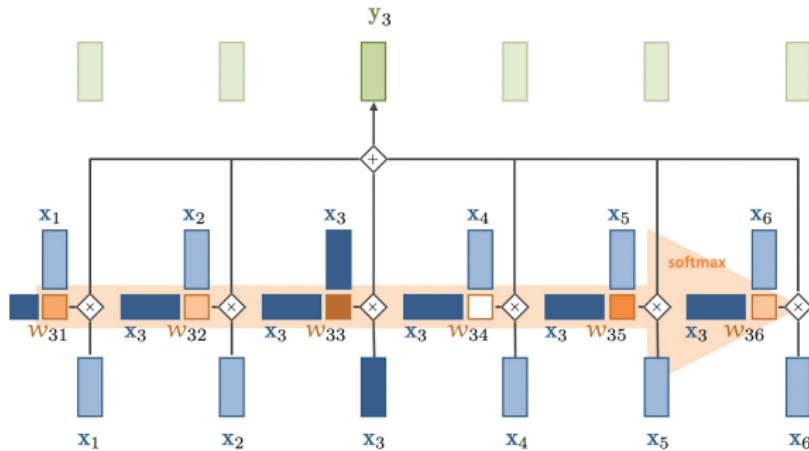
Self-attention

Self-attention is just a transformation. There are no trainable parameters. However, the [input](#) may be the result of a learned transformation.



Self-attention

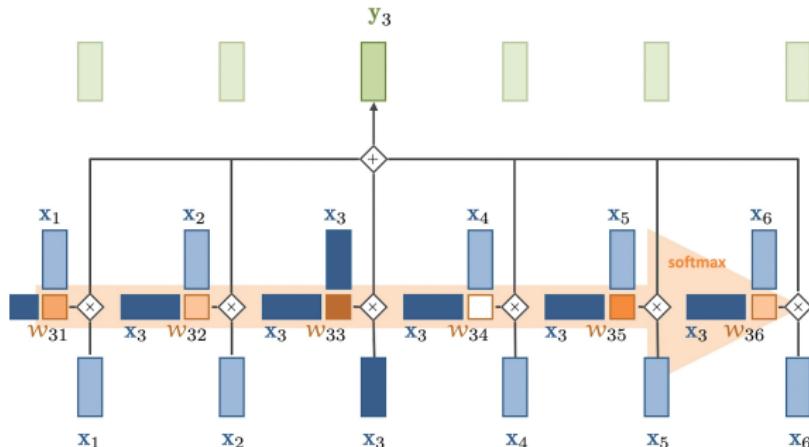
Self-attention is just a transformation. There are no trainable parameters. However, the [input](#) may be the result of a learned transformation.



Moreover, no temporal information is used. Sequences are seen as [sets](#).

Self-attention

Self-attention is just a transformation. There are no trainable parameters. However, the [input](#) may be the result of a learned transformation.



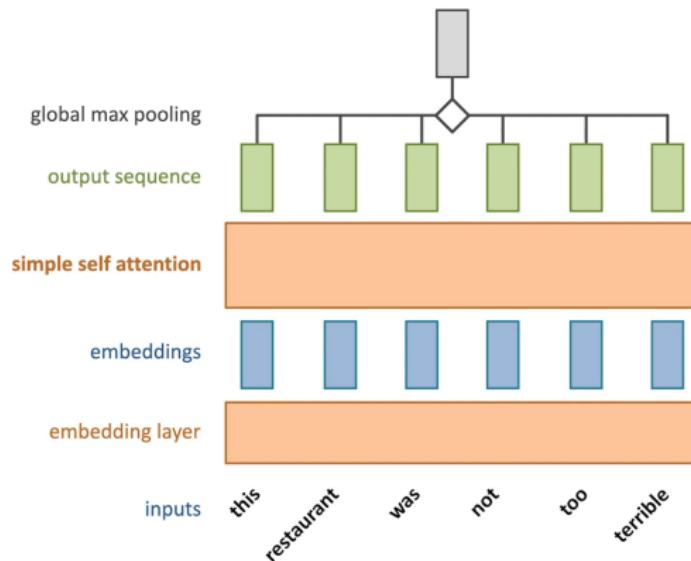
Moreover, no temporal information is used. Sequences are seen as [sets](#).

Self-attention is [permutation-equivariant](#):

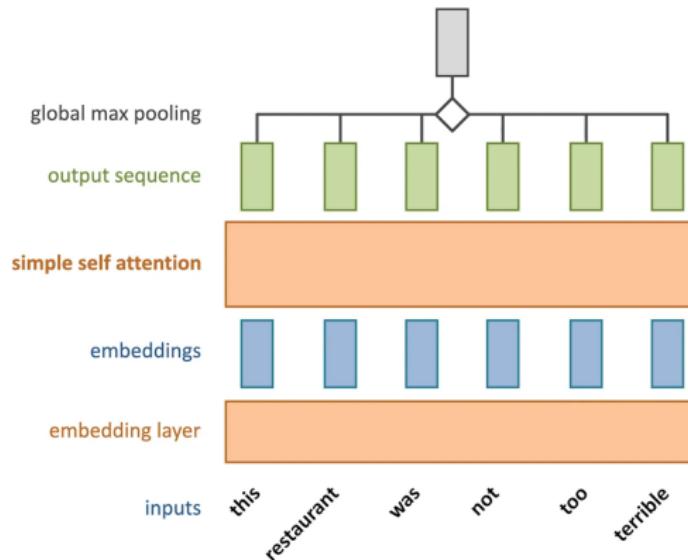
$$\pi(\text{sa}(\mathbf{x})) = \text{sa}(\pi(\mathbf{x}))$$

... con questo modello il tempo non è rappresentato (tuttavia esiste la variante con layer causali)

Example



Example



The word “not” directly affects “terrible”

i.e., the dot product $x_{\text{not}}^\top x_{\text{terrible}}$ should be large.

Key, value, query

Each input vector plays three roles:

$$\mathbf{w}'_{ij} = \underbrace{\mathbf{x}_i^\top}_{\text{query}} \underbrace{\mathbf{x}_j}_{\text{key}}$$

$$y_i = \sum_j \mathbf{w}_{ij} \underbrace{\mathbf{x}_j}_{\text{value}}$$

Key, value, query



Each input vector plays three roles:

$$w'_{ij} = \underbrace{x_i^\top}_{\text{query}} \underbrace{x_j}_{\text{key}}$$

$$y_i = \sum_j w_{ij} \underbrace{x_j}_{\text{value}}$$

We can now introduce **trainable** weights and biases:

$$w'_{ij} = q_i^\top k_j \quad \begin{matrix} q = \text{query} \\ k = \text{key} \end{matrix}$$
$$y_i = \sum_j w_{ij} v_j \quad \begin{matrix} v = \text{value} \end{matrix}$$

where $q = Qx + b$ (and similarly for k and v).



calcolo della self-attention
nella fase di addestramento



generazione
della parola
successiva

Causal self-attention

If we need an [auto-regressive](#) model, we must avoid looking ahead.

Causal self-attention

If we need an **auto-regressive** model, we must avoid looking ahead.

We do this by only summing over the **previous** tokens in the sequence:

$$\textcolor{brown}{y}_i = \sum_j \textcolor{brown}{w}_{ij} \textcolor{brown}{x}_j$$

Causal self-attention

If we need an **auto-regressive** model, we must avoid looking ahead.

We do this by only summing over the **previous** tokens in the sequence:

$$\textcolor{brown}{y}_i = \sum_{j \leq i} \textcolor{brown}{w}_{ij} \textcolor{brown}{x}_j$$

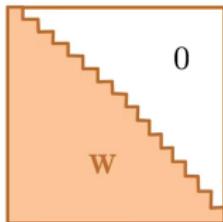
Causal self-attention

If we need an **auto-regressive** model, we must avoid looking ahead.

We do this by only summing over the **previous** tokens in the sequence:

$$y_i = \sum_{j \leq i} w_{ij} x_j$$

In matrix notation, we simply set:



This is also known as **masking**.

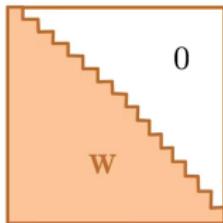
Causal self-attention

If we need an **auto-regressive** model, we must avoid looking ahead.

We do this by only summing over the **previous** tokens in the sequence:

$$\textcolor{brown}{y}_i = \sum_{j \leq i} \textcolor{brown}{w}_{ij} \textcolor{brown}{x}_j$$

In matrix notation, we simply set:



This is also known as **masking**.

Other **priors** can be encoded by enforcing a structure on **W**.

Position information

In some applications, the sequential structure is informative.
i.e., permutation-equivariance is not always desired.

Position information

In some applications, the sequential structure is informative.
i.e., permutation-equivariance is not always desired.

- Position embedding

Learn a vector embedding for each position, sum it to the token

$$1 + \mathbf{v}_1, 2 + \mathbf{v}_2, 3 + \mathbf{v}_3, \dots$$

Position information

In some applications, the sequential structure is informative.
i.e., permutation-equivariance is not always desired.

- Position embedding

Learn a vector embedding for each position, sum it to the token

- Position encoding

Define position embeddings *a priori* using some mathematical rule

$$1 + \mathbf{v}_1, 2 + \mathbf{v}_2, 3 + \mathbf{v}_3, \dots$$

Position information

In some applications, the sequential structure is informative.
i.e., permutation-equivariance is not always desired.

- Position embedding

Learn a vector embedding for each position, sum it to the token

- Position encoding

Define position embeddings *a priori* using some mathematical rule

- Relative positions

Embed/encode the relative rather than the absolute positions

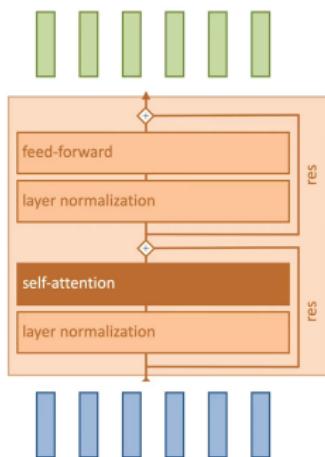
$$\boxed{1} + \mathbf{v}_1, \quad \boxed{2} + \mathbf{v}_2, \quad \boxed{3} + \mathbf{v}_3, \quad \dots$$

Transformers

A **transformer** is any model that primarily uses self-attention to propagate information across the basic tokens (e.g. vectors in a sequence, pixels in a grid, nodes in a graph, etc.).

Transformers

A **transformer** is any model that primarily uses self-attention to propagate information across the basic tokens (e.g. vectors in a sequence, pixels in a grid, nodes in a graph, etc.).



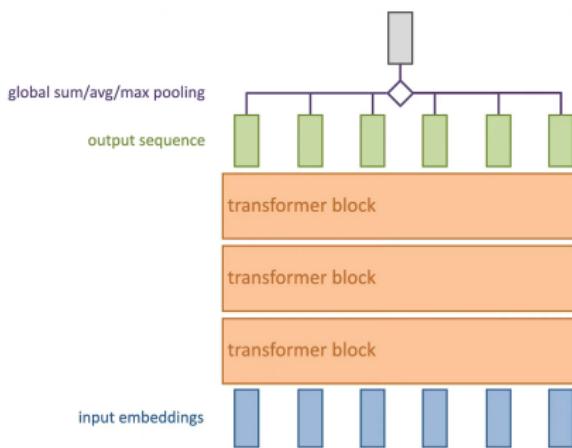
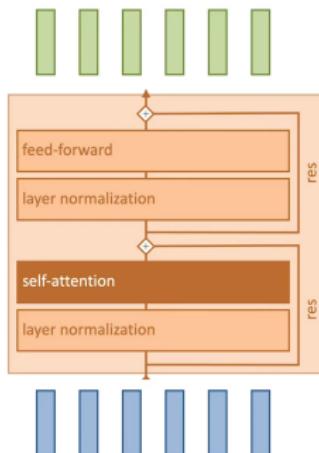
Transformers

I transformers sono reti neurali in cui le informazioni di diversi elementi si mescolano solo attraverso operazioni di self-attention

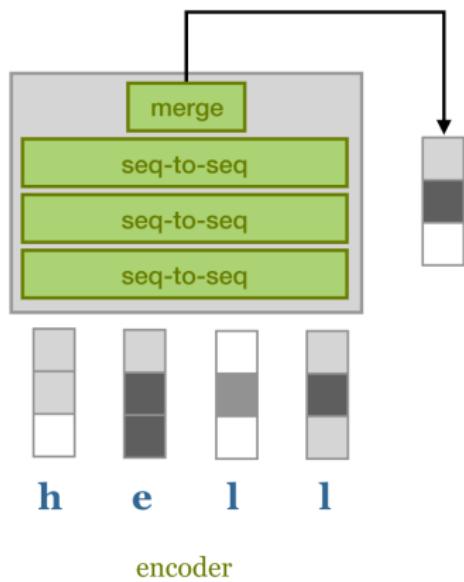
A **transformer** is any model that primarily uses self-attention to propagate information across the basic tokens (e.g. vectors in a sequence, pixels in a grid, nodes in a graph, etc.).

Main idea:

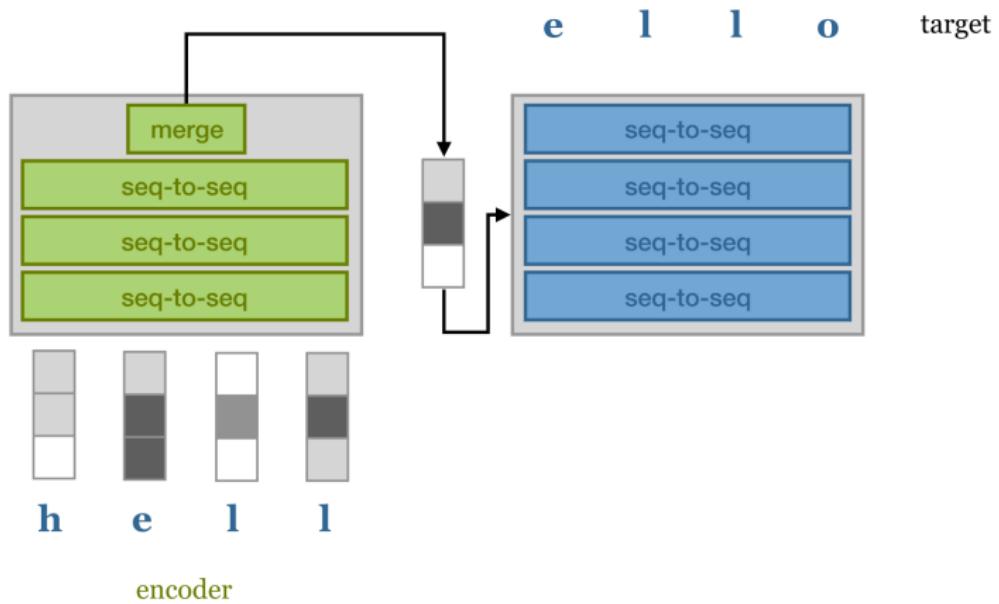
Define a generic transformer block, and compose it several times.



Encoder-decoder model

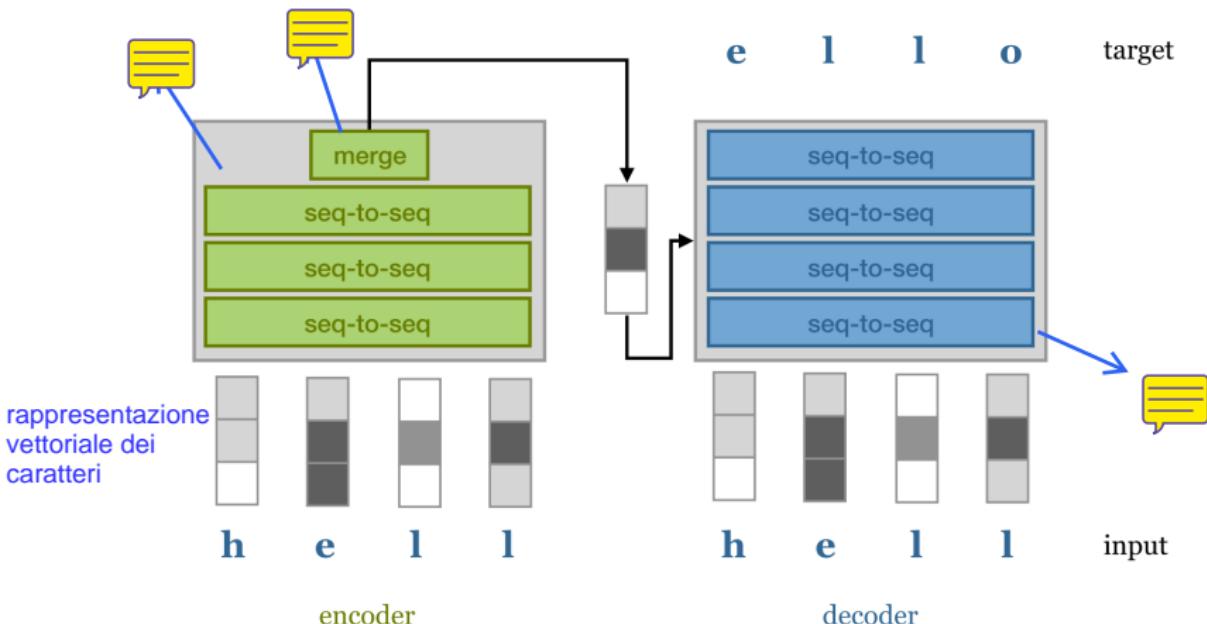


Encoder-decoder model



Encoder-decoder model

Il decoder genera la sequenza di output "ello", token per token.



I modelli Seq2Seq (come i transformer) sono tipicamente composti da due componenti principali: un encoder e un decoder.

Casi d'uso: traduzione automatica, riassunto di un testo