

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](#))



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



Sequential data

Example: 3D shape motions



Prototypical task: **classify** the sequence (e.g. "running")

Sequential data

Example: Text ([symbolic](#))

"the little fox"

the, little, fox

t, h, e, , l, i, t, t, l, e, , f, o, x

Sequential data

Example: Text ([symbolic](#))

"the little fox"

the, little, fox

t, h, e, , l, i, t, t, l, e, , f, o, x

Prototypical task: text [translation](#) (e.g. "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.

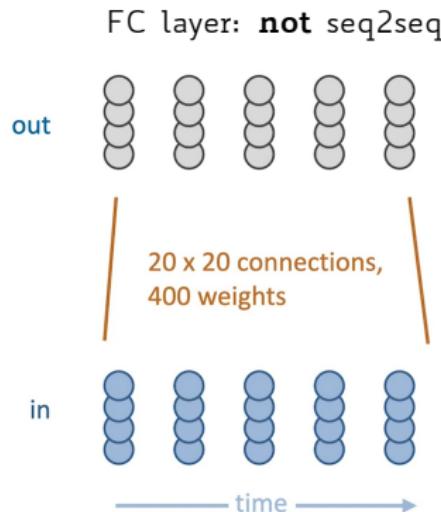
If the output is a sequence: [sequence-to-sequence](#) model.

Sequence-to-sequence model

Key property:

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

If the output is a sequence: **sequence-to-sequence** model.

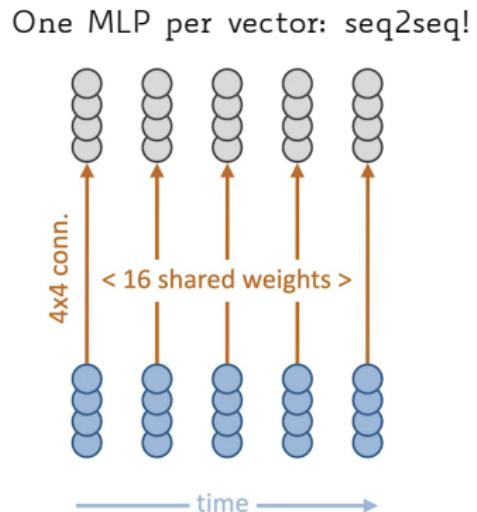
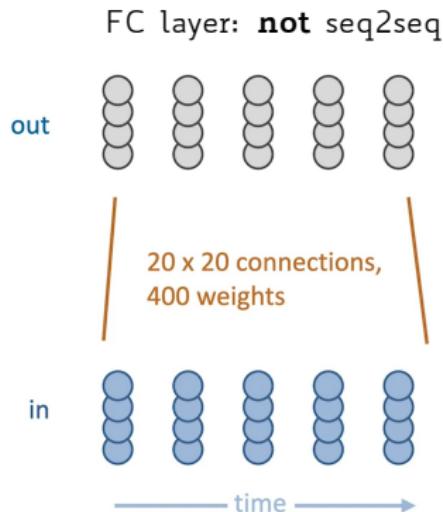


Sequence-to-sequence model

Key property:

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

If the output is a sequence: **sequence-to-sequence** model.

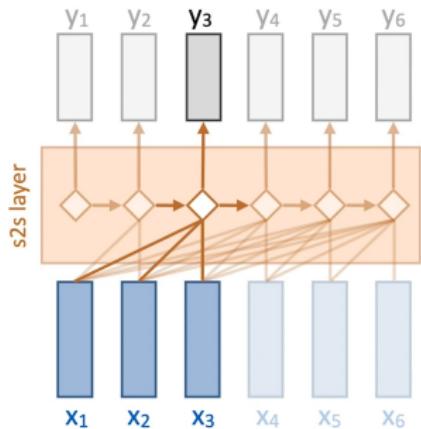


Causal vs. non-causal layers

S2S layers admit input and output with different lengths.

Causal vs. non-causal layers

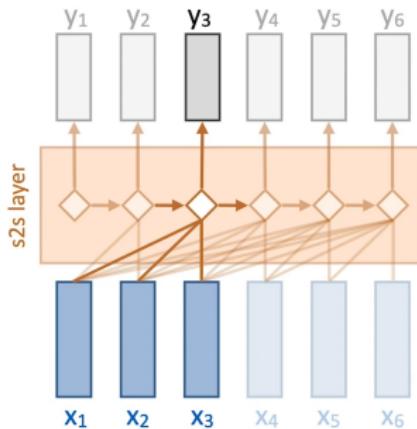
S2S layers admit input and output with different lengths.



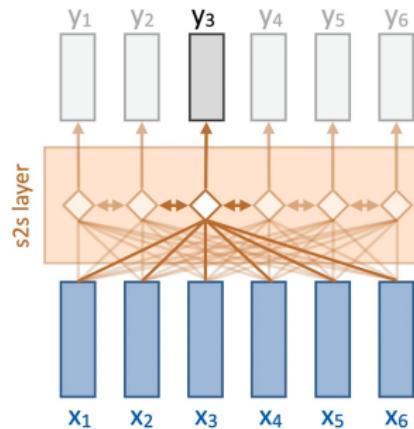
"causal" layer
can only look backward in time

Causal vs. non-causal layers

S2S layers admit input and output with different lengths.



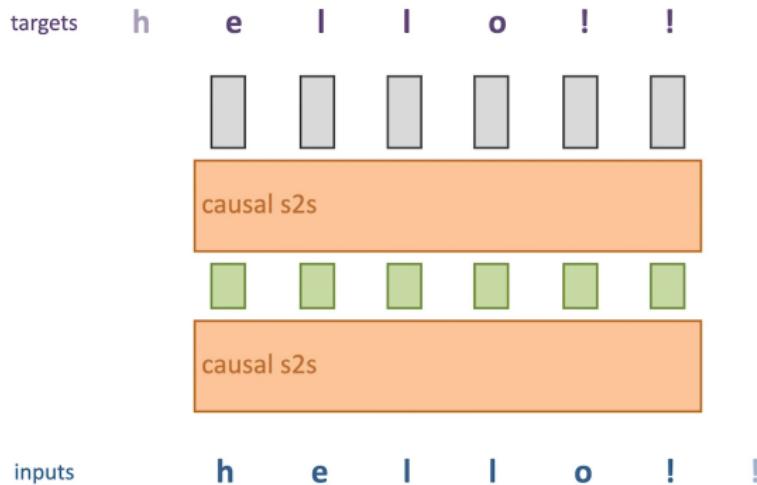
"causal" layer
can only look backward in time



"non-causal" layer
no restrictions

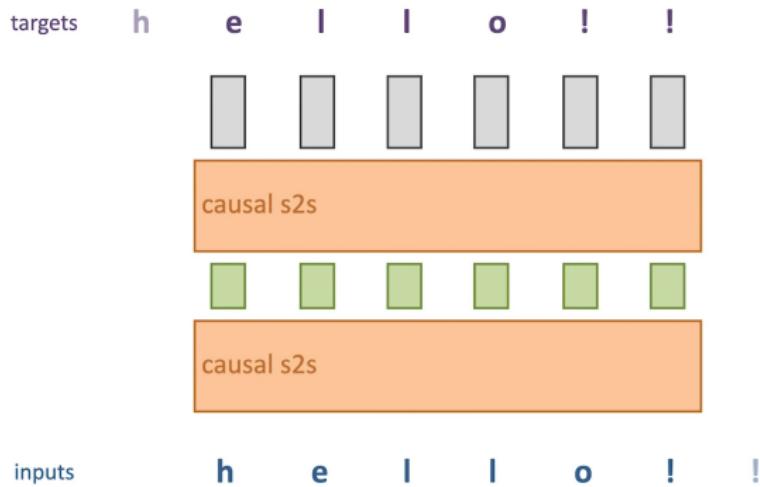
Autoregressive modeling

Given an unlabeled dataset of sequences, take a subsequence and train to predict the **next character**:



Autoregressive modeling

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

For each token, output a **probability distribution** over the set of symbols (e.g. alphabet letters).

Once trained, 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

For each token, output a **probability distribution** over the set of symbols (e.g. alphabet letters).

Once trained, 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

For each token, output a **probability distribution** over the set of symbols (e.g. alphabet letters).

Once trained, 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

"I like to eat hot dogs pancakes"
80% 20%

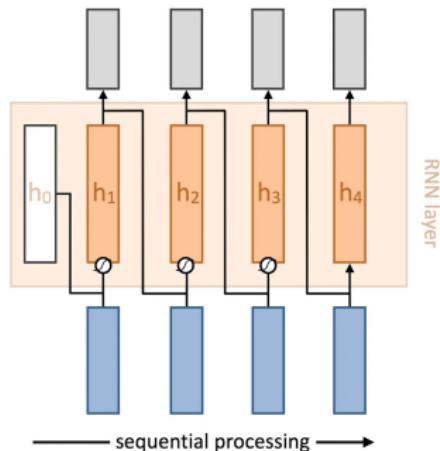
Sequence-to-sequence layers

How do we actually design a S2S layer?

Sequence-to-sequence layers

How do we actually design a S2S layer?

Many possible choices.

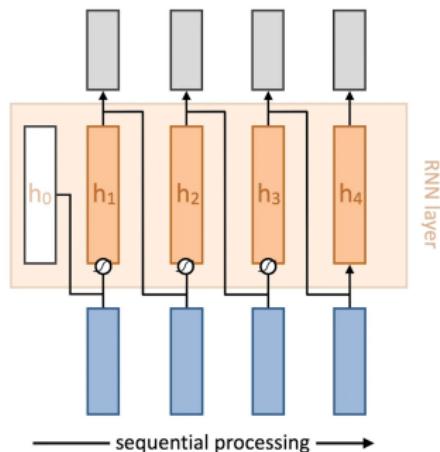


can look far back in the sequence
sequential processing

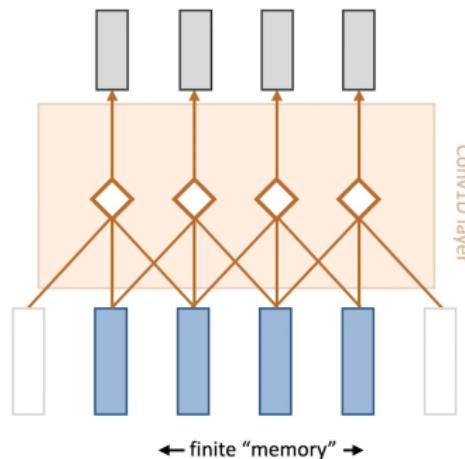
Sequence-to-sequence layers

How do we actually design a S2S layer?

Many possible choices.



can look far back in the sequence
sequential processing



limited range backward or forward
parallel computation

Self-attention

Can we have parallelism **and** long dependencies?

Self-attention

Can we have parallelism **and** long dependencies?

Consider the linear transformation (**input** → **output**):

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

Self-attention

Can we have parallelism **and** long dependencies?

Consider the linear transformation (**input** → **output**):

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

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

Self-attention

Can we have parallelism **and** long dependencies?

Consider the linear transformation (**input** → **output**):

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

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

Given a position i , compute the correlations over all j :

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

Self-attention

Can we have parallelism **and** long dependencies?

Consider the linear transformation (**input** → **output**):

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

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

Given a position i , compute the correlations over all j :

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

and transform such that $\textcolor{brown}{w}_{ij} > 0$ and $\sum_j \textcolor{brown}{w}_{ij} = 1$:

$$\textcolor{brown}{w}_{ij} = \frac{e^{\textcolor{brown}{w}'_{ij}}}{\sum_j e^{\textcolor{brown}{w}'_{ij}}}$$

Self-attention

In matrix notation:

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

Self-attention

In matrix notation:

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

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

Self-attention

In matrix notation:

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

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

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

Self-attention

In matrix notation:

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

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

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

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

Self-attention

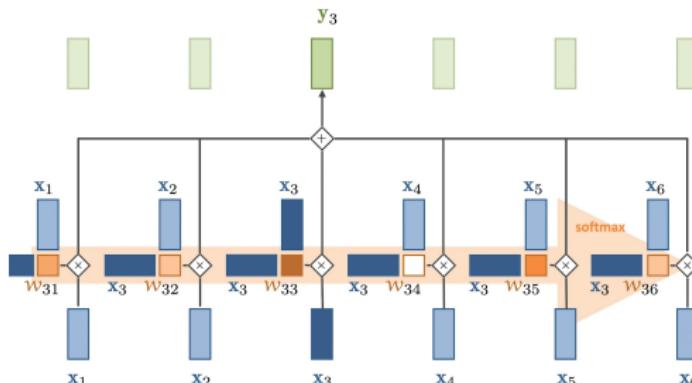
In matrix notation:

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

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

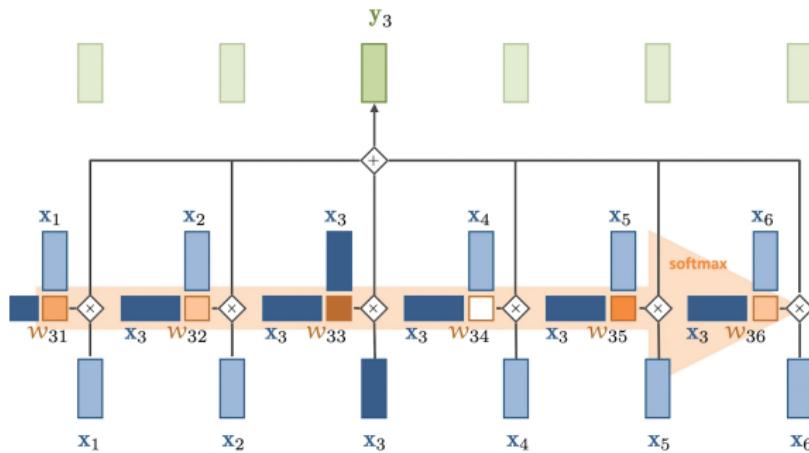
$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{x}_j \Rightarrow \mathbf{Y}^\top = \mathbf{W} \mathbf{X}^\top$$

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



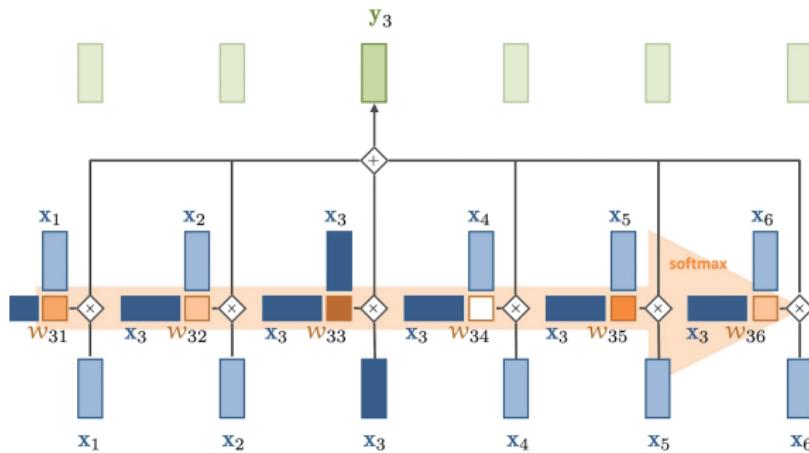
Self-attention (SA)

SA is just a transformation (no trainable parameters).
However, the **input** may be the result of a learned transformation.



Self-attention (SA)

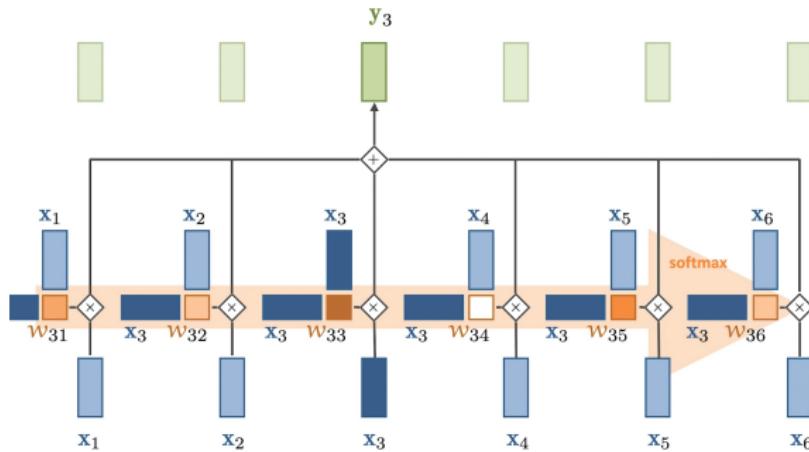
SA is just a transformation (no trainable parameters).
However, the **input** may be the result of a learned transformation.



No temporal information is used! Sequences are just **sets**.

Self-attention (SA)

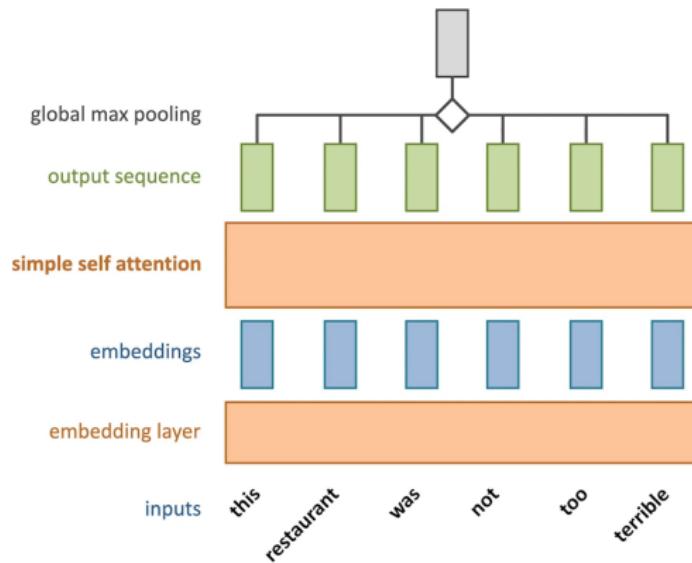
SA is just a transformation (no trainable parameters).
However, the `input` may be the result of a learned transformation.



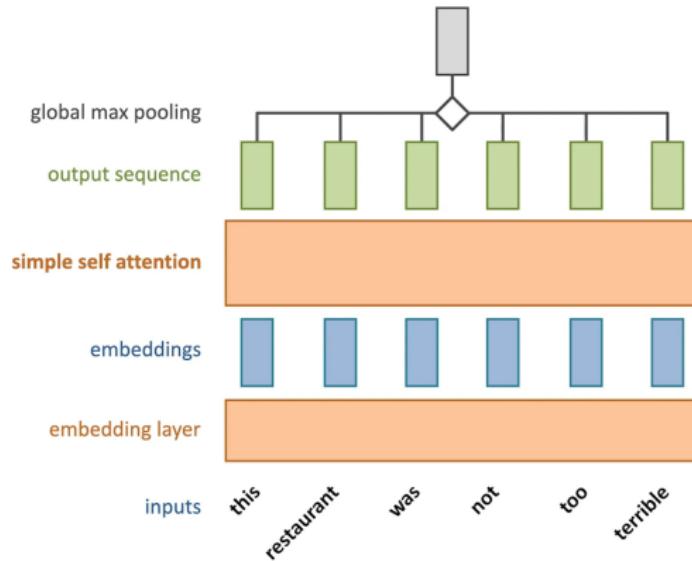
No temporal information is used! Sequences are just **sets**.
SA is **permutation-equivariant**:

$$\pi(\text{sa}(\textcolor{blue}{x})) = \text{sa}(\pi(\textcolor{blue}{x}))$$

Example



Example



The word "not" directly affects "terrible"
i.e., the dot product $x_{\text{not}}^T x_{\text{terrible}}$ should be large.

Key, value, query

Each input vector plays three roles:

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

$$\textcolor{brown}{y}_i = \sum_j \textcolor{brown}{w}_{ij} \underbrace{\textcolor{blue}{x}_j}_{\text{value}}$$

Key, value, query

Each input vector plays three roles:

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

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

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

$$\mathbf{w}'_{ij} = \mathbf{q}_i^\top \mathbf{k}_j$$

$$y_i = \sum_j \mathbf{w}_{ij} \mathbf{v}_j$$

where $\mathbf{q} = Q\mathbf{x} + b$ (and similarly for \mathbf{k} and \mathbf{v}).

Causal self-attention

An **auto-regressive** model must not look ahead!

Causal self-attention

An **auto-regressive** model must not look ahead!

Just sum over the **previous** tokens in the sequence:

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

Causal self-attention

An **auto-regressive** model must not look ahead!

Just sum over the **previous** tokens in the sequence:

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

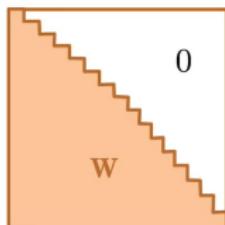
Causal self-attention

An **auto-regressive** model must not look ahead!

Just sum over the **previous** tokens in the sequence:

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

In matrix notation, simply set:



This is also known as **masking**.

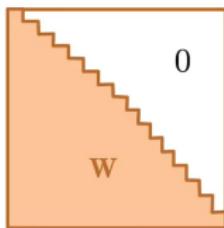
Causal self-attention

An **auto-regressive** model must not look ahead!

Just sum over the **previous** tokens in the sequence:

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

In matrix notation, simply set:



This is also known as **masking**.

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

Position information

When the **sequential** structure is informative,
permutation-equivariance is not desired.

Position information

When the **sequential** structure is informative,
permutation-equivariance is not 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

When the **sequential** structure is informative, permutation-equivariance is not 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

When the **sequential** structure is informative, permutation-equivariance is not 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

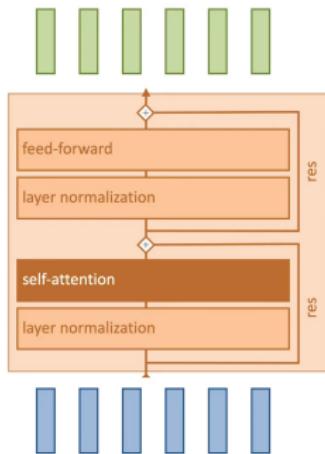
$$1 + \mathbf{v}_1 , 2 + \mathbf{v}_2 , 3 + \mathbf{v}_3 , \dots$$

Transformers

A **transformer** is any model that **primarily** uses SA across tokens (e.g. vectors in a sequence, pixels in a grid, etc.).

Transformers

A **transformer** is any model that **primarily** uses SA across tokens (e.g. vectors in a sequence, pixels in a grid, etc.).

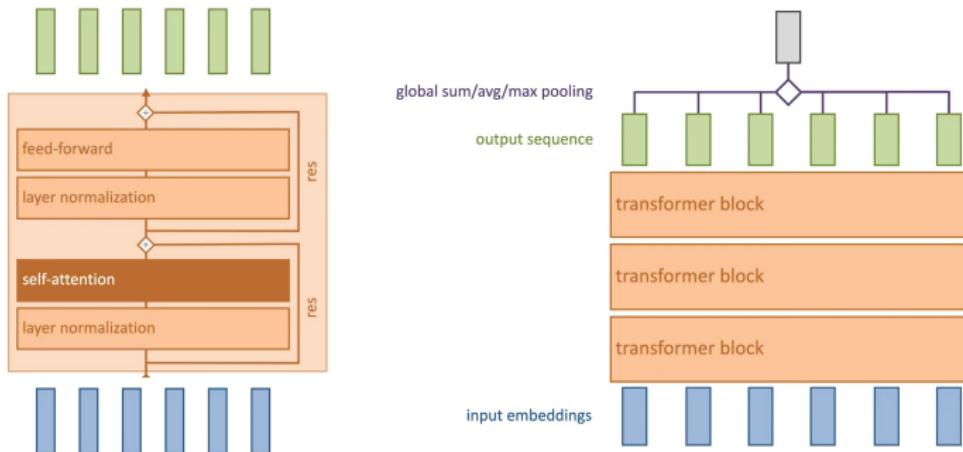


Transformers

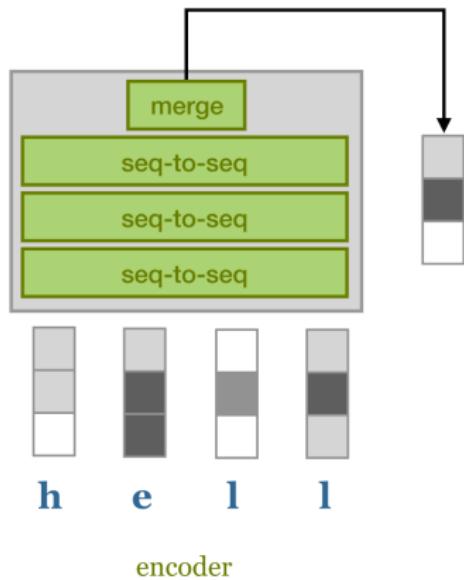
A **transformer** is any model that **primarily** uses SA across tokens (e.g. vectors in a sequence, pixels in a grid, etc.).

Main idea:

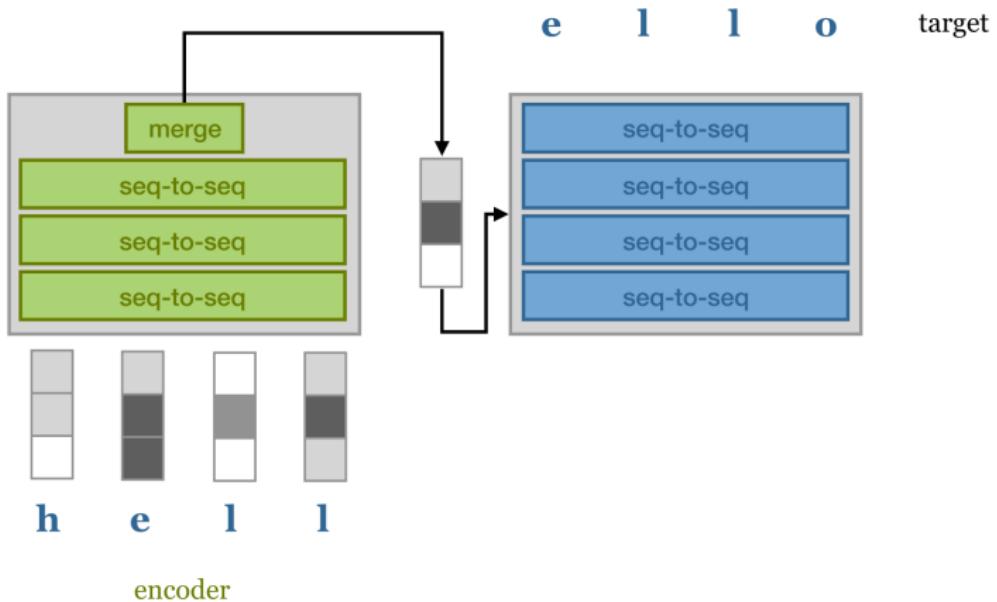
Define a generic transformer block, compose it many times.



Encoder-decoder model



Encoder-decoder model



Encoder-decoder model

