

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 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](#))

“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

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](#).

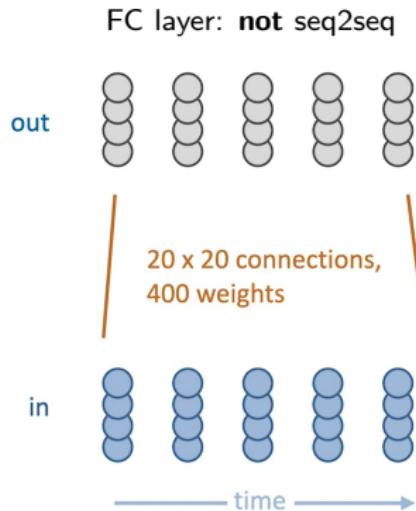
If the output is another sequence, it's a [sequence-to-sequence](#) model.

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.

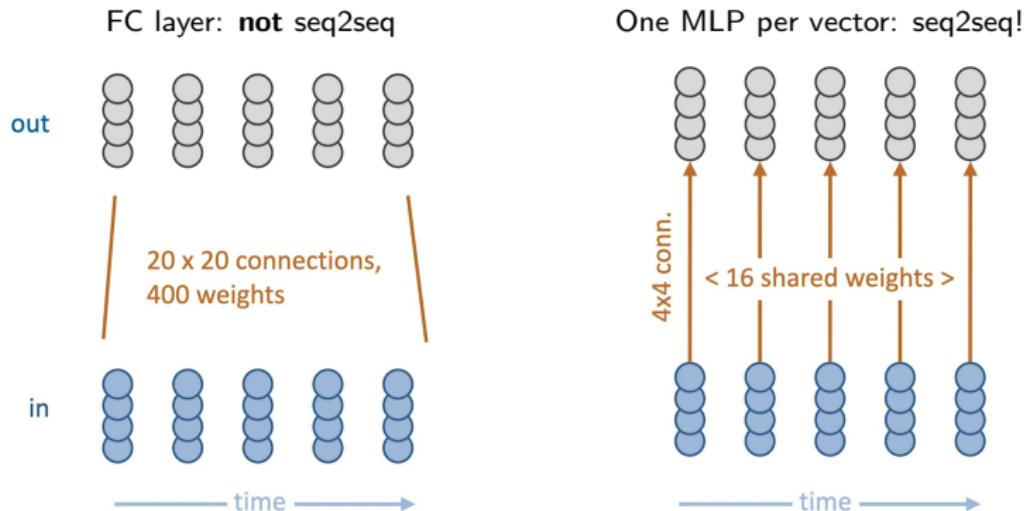


Sequence-to-sequence model

Key property:

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

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

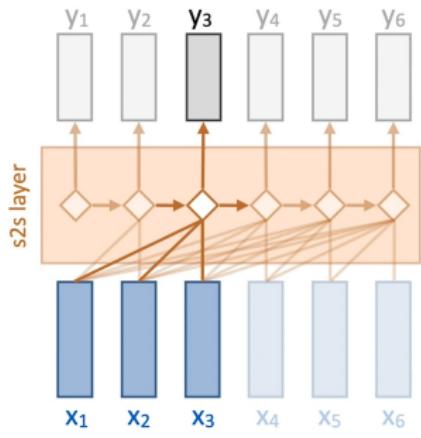


Causal vs. non-causal layers

S2S layers admit input and output with different lengths.

Causal vs. non-causal layers

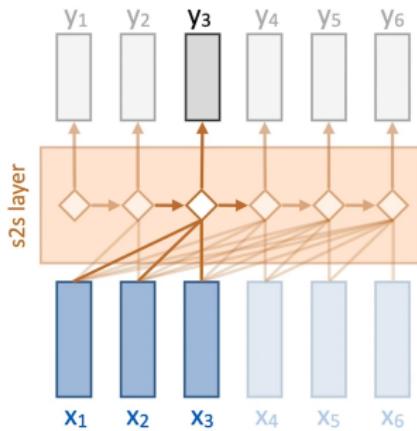
S2S layers admit input and output with different lengths.



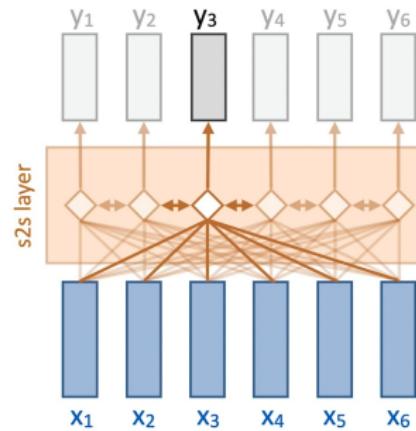
“causal” layer
can only look backward in time

Causal vs. non-causal layers

S2S layers admit input and output with different lengths.



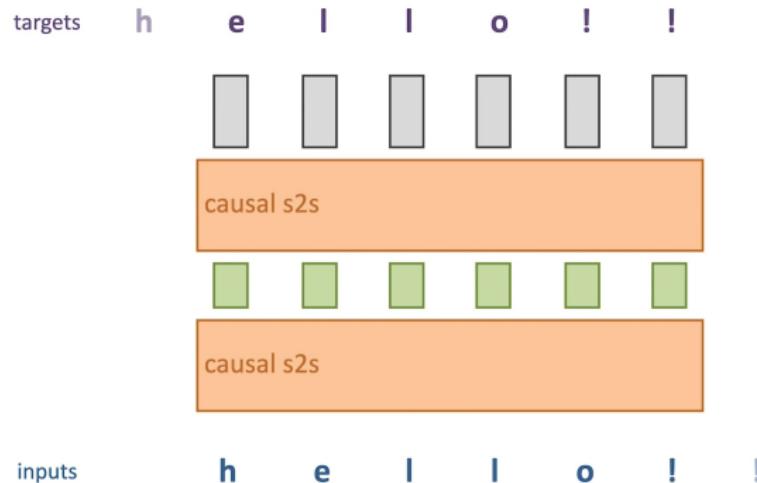
"causal" layer
can only look backward in time



"non-causal" layer
no restrictions

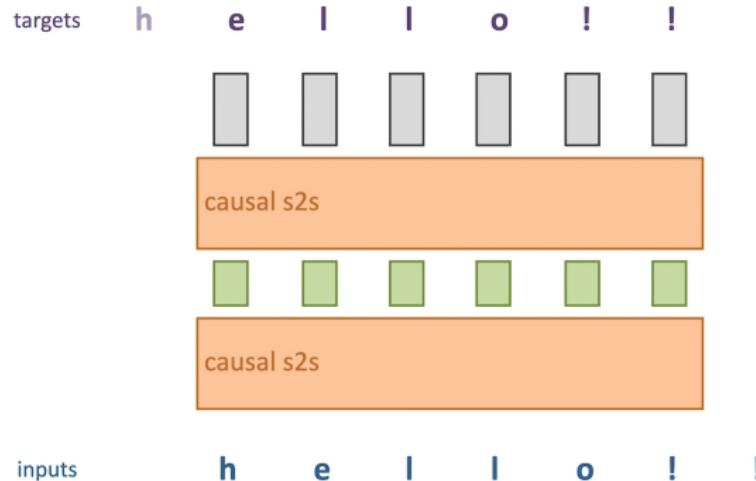
Autoregressive modeling

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

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 $[s_1, s_2, s_3, \dots]$

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%

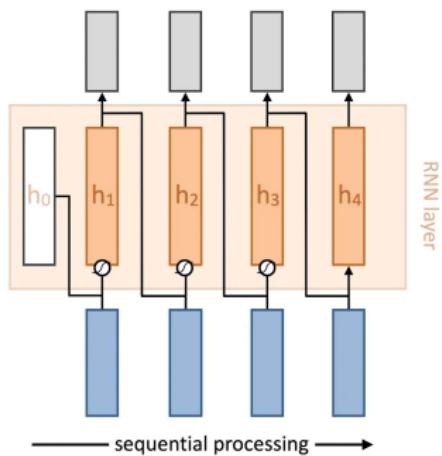
Sequence-to-sequence layers

How do we actually design a S2S layer?

Sequence-to-sequence layers

How do we actually design a S2S layer?

Many possible choices.

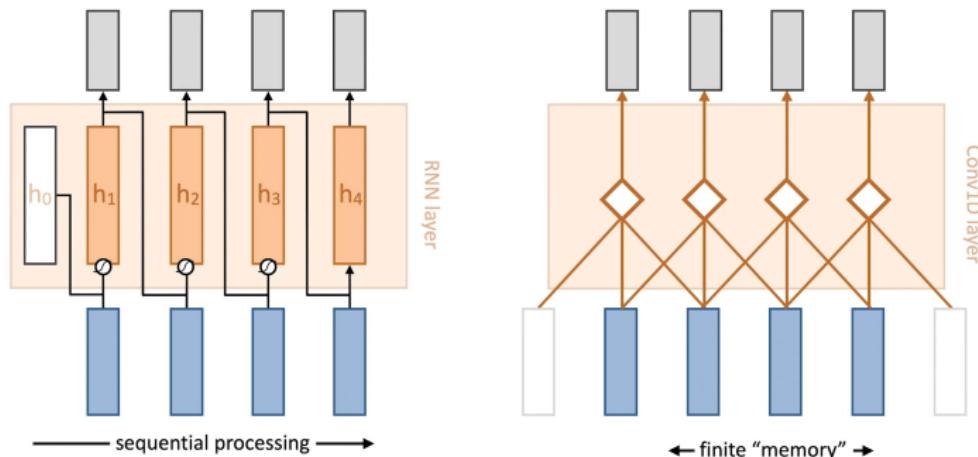


can look indefinitely far back in the sequence
processing must be done sequentially

Sequence-to-sequence layers

How do we actually design a S2S layer?

Many possible choices.



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

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 :

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

and transform them so that each $\textcolor{brown}{w}_{ij} > 0$ 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:

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

Self-attention

In matrix notation:

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

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

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:

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

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

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

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

Self-attention

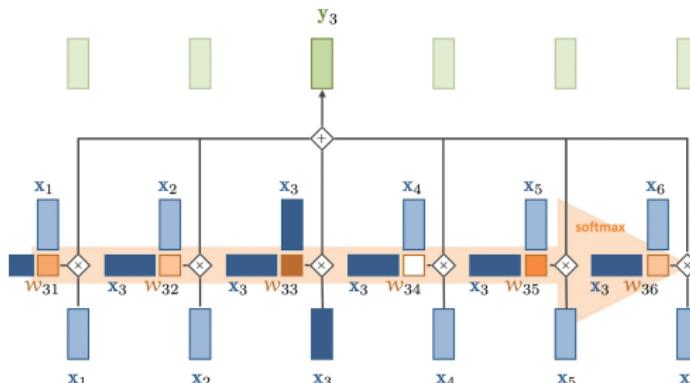
In matrix notation:

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

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

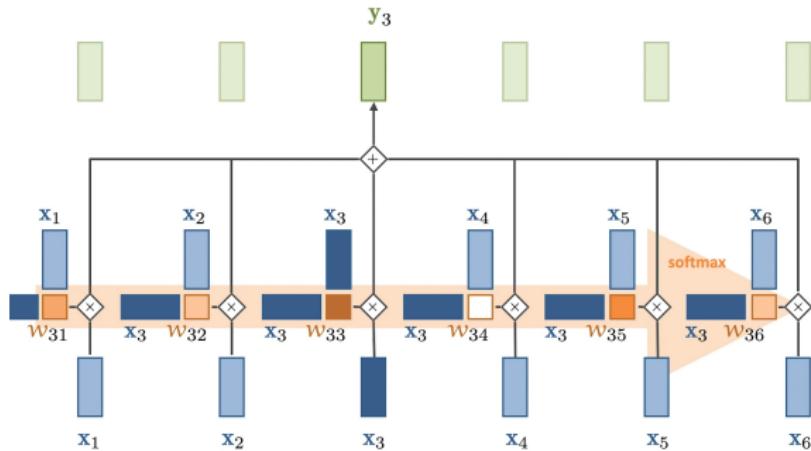
$$y_i = \sum_j w_{ij} \mathbf{x}_j \Rightarrow \mathbf{Y}^\top = \mathbf{W} \mathbf{X}^\top$$

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



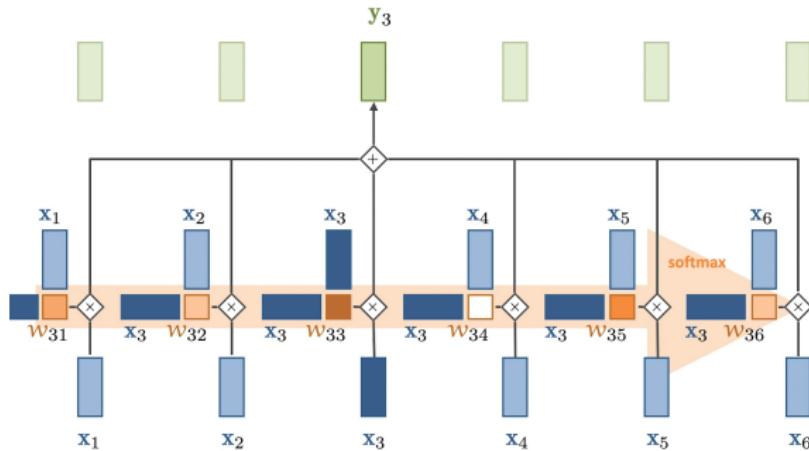
Self-attention

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



Self-attention

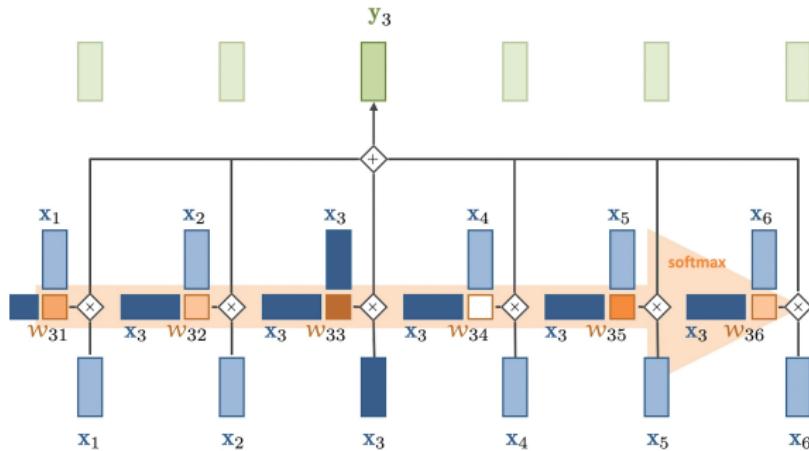
Self-attention is just a transformation. There are no trainable parameters.
However, the [input](#) may be the result of a learned transformation.



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.

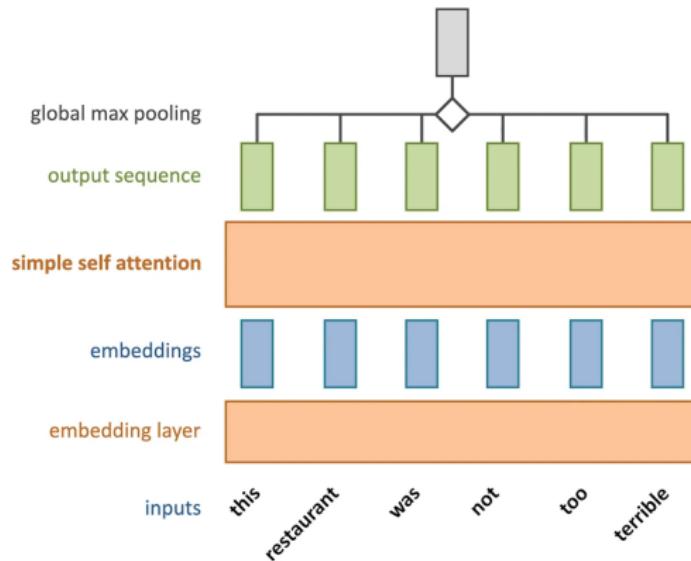


Moreover, no temporal information is used. Sequences are seen as [sets](#).

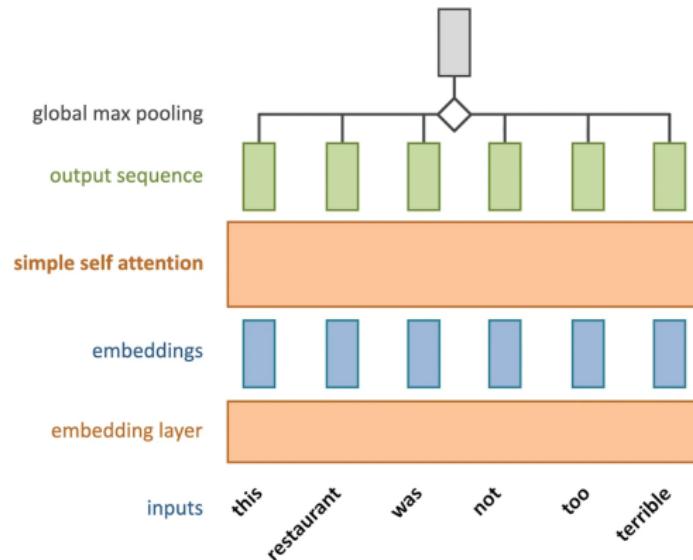
Self-attention is [permutation-equivariant](#):

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

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:

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

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

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

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:

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

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:

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

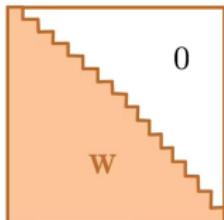
Causal self-attention

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**.

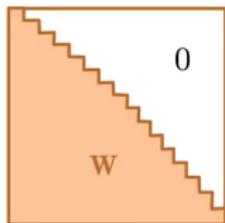
Causal self-attention

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:

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

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

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

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

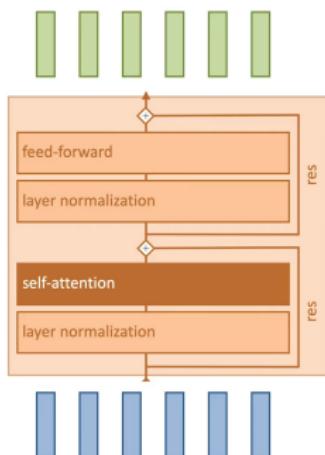
$$\boxed{1} + \mathbf{v}_1 , \quad \boxed{2} + \mathbf{v}_2 , \quad \boxed{3} + \mathbf{v}_3 , \dots$$

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.).

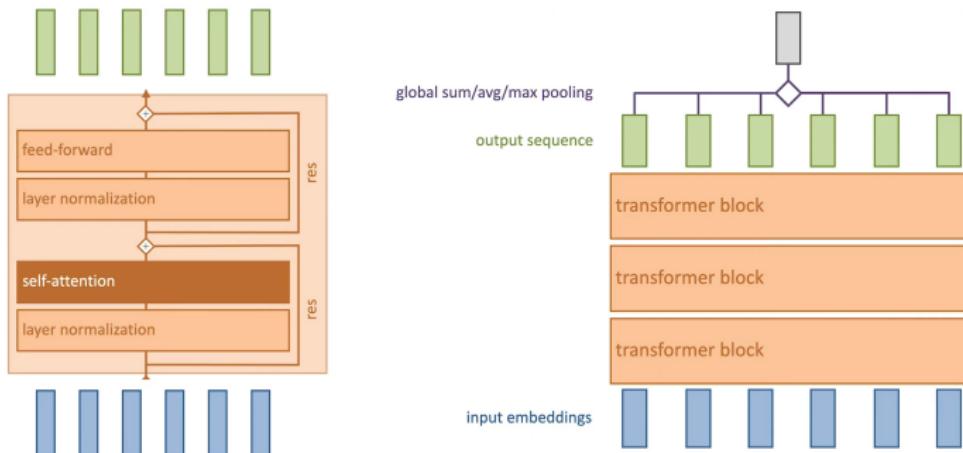


Transformers

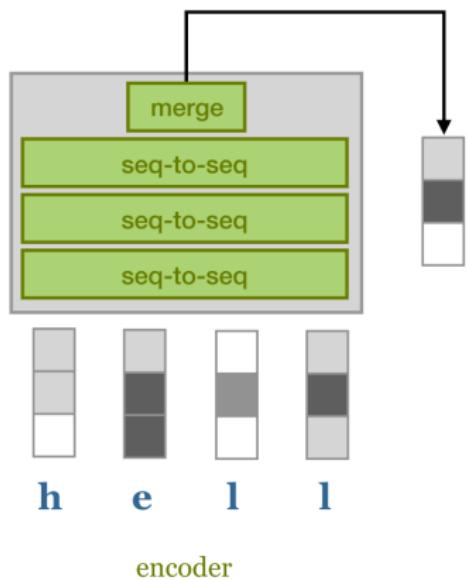
A **transformer** is any model that primarily uses self-attention to propagate information across the basic tokens (e.g. vectors in a sequence, pixels in a grid, nodes in a graph, etc.).

Main idea:

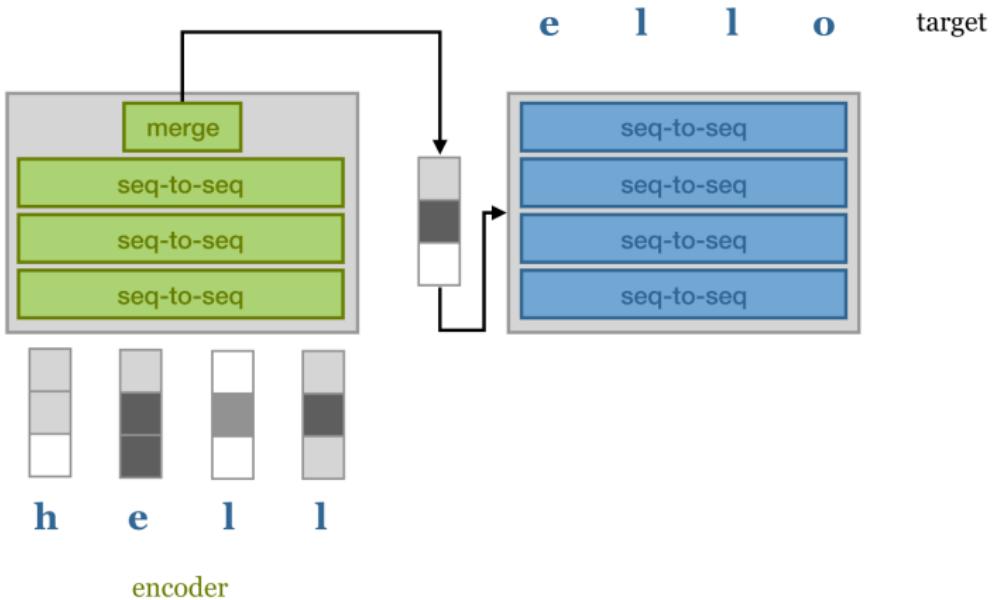
Define a generic transformer block, and compose it several times.



Encoder-decoder model



Encoder-decoder model



Encoder-decoder model

