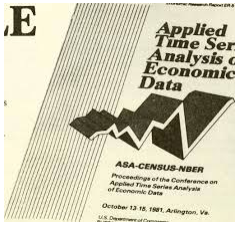
UniGe | MaLGa

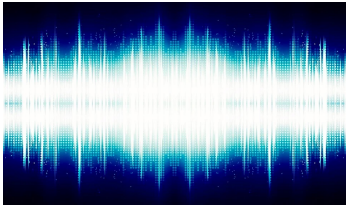
From Recurrent Neural Networks to Transformers

Nicoletta Noceti
Nicoletta.noceti@unige.it

Dealing with sequential data

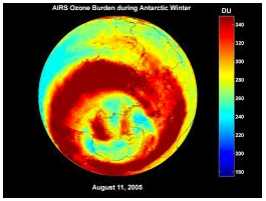


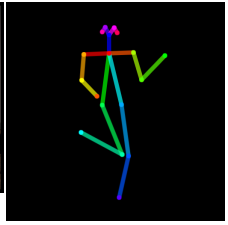
Text

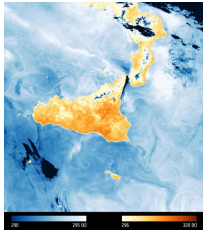


Audio

IoT





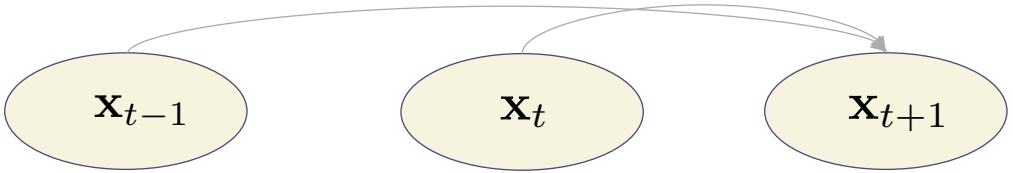


Sequences of visual data

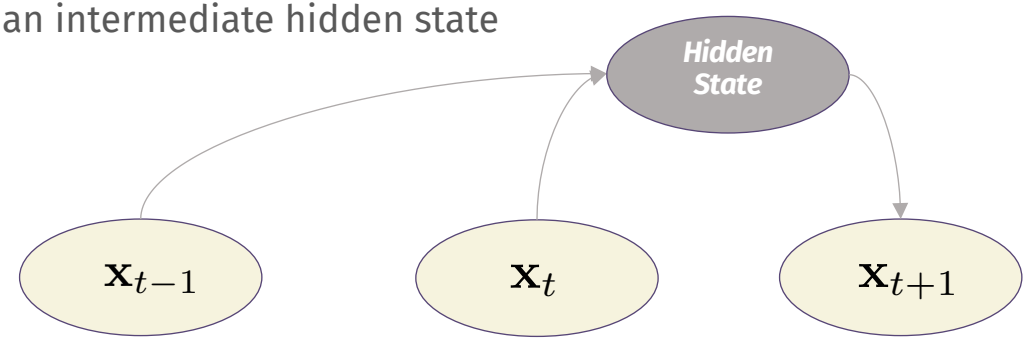
UniGe | MaLGa

Dealing with sequences: a first summary

- Autoregressive models



- "Using" an intermediate hidden state



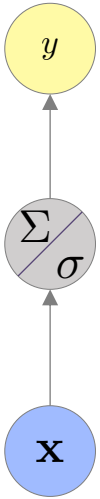
Dealing with sequences: issues

- Handling sequences of **different lengths**
- Taking into account **short** and **long term** dependences
- Considering **order** between elements

UniGe | MaLGa

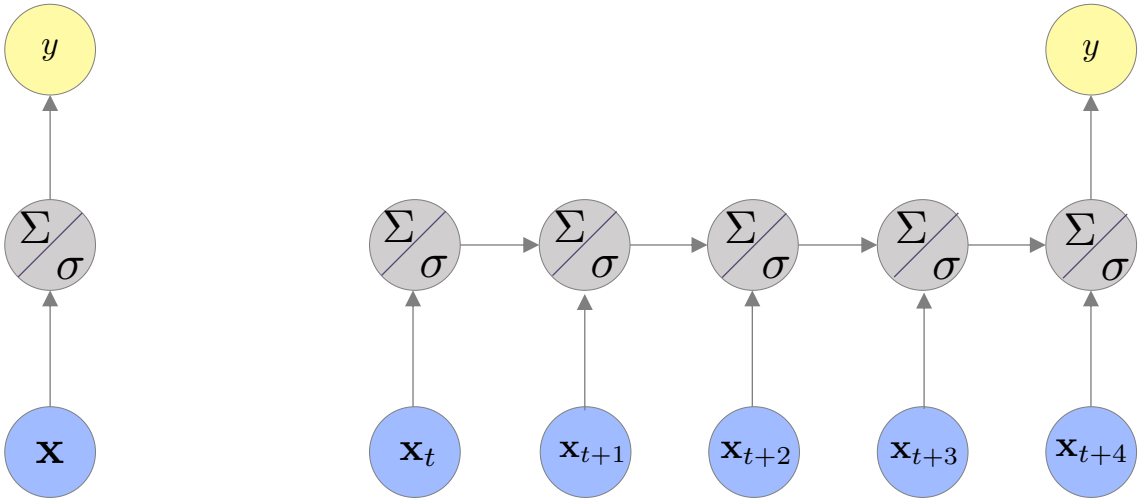
Recurrent Neural Networks

Modelling sequences



Standard one-to-one vanilla network

Modelling sequences



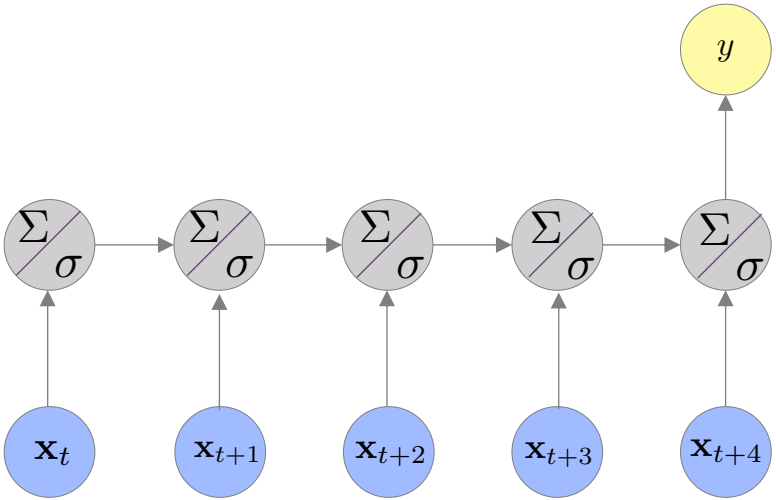
Standard one-to-one vanilla network

Adding recurrence over time

Recurrent Neural Networks (1986)

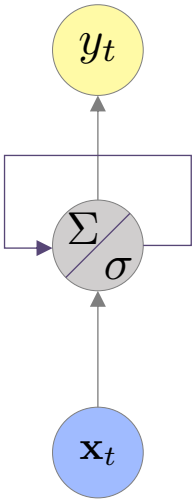
- Recurrence adds memory to the NN
- It also provides a way to model causal relationships between observations: the decision a recurrent net reached at time step t-1 affects the decision it will reach at time step t
- RNNs have two sources of input: the present and the recent past, which are combined to determine how they respond to new data

Recurrent Neural Networks



Recurrent Neural Networks

The weight matrices are filters that determine ***how much importance to give to both the present input and the past hidden state***



$$\mathbf{x}_t \in \mathbb{R}^m$$

$$y_t \in \mathbb{R}$$

$$\mathbf{h}_t \in \mathbb{R}^p$$

$$\mathbf{h}_t = \sigma(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

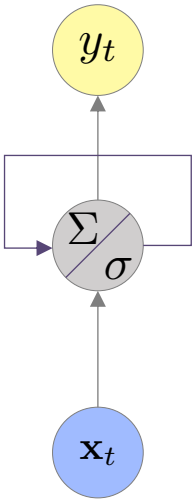
$$y_t = \sigma(W_y \mathbf{h}_t)$$

$$W_h \in \mathbb{R}^{p \times p} \qquad W_y \in \mathbb{R}^p$$

$$W_x \in \mathbb{R}^{p \times m}$$

Recurrent Neural Networks

The weight matrices are filters that determine ***how much importance to give to both the present input and the past hidden state***



$$\mathbf{x}_t \in \mathbb{R}^m$$

$$y_t \in \mathbb{R}$$

$$\mathbf{h}_t \in \mathbb{R}^p$$

This is a hyper-parameter of the method

$$\mathbf{h}_t = \sigma(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

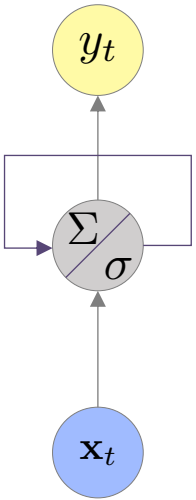
$$y_t = \sigma(W_y \mathbf{h}_t)$$

$$W_h \in \mathbb{R}^{p \times p} \qquad W_y \in \mathbb{R}^p$$

$$W_x \in \mathbb{R}^{p \times m}$$

Recurrent Neural Networks

The weight matrices are filters that determine ***how much importance to give to both the present input and the past hidden state***



$$\mathbf{x}_t \in \mathbb{R}^m$$

$$y_t \in \mathbb{R}$$

$$\mathbf{h}_t \in \mathbb{R}^p$$

This is a hyper-parameter of the method

Usually a tanh

$$\mathbf{h}_t = \sigma(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

$$y_t = \sigma(W_y \mathbf{h}_t)$$

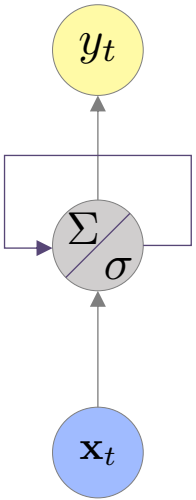
It depends on the problem

$$W_h \in \mathbb{R}^{p \times p} \qquad W_y \in \mathbb{R}^p$$

$$W_x \in \mathbb{R}^{p \times m}$$

Recurrent Neural Networks

The weight matrices are filters that determine ***how much importance to give to both the present input and the past hidden state***



$$\mathbf{x}_t \in \mathbb{R}^m$$

$$y_t \in \mathbb{R}$$

$$\mathbf{h}_t \in \mathbb{R}^p$$

This is a hyper-parameter of the method

Usually a tanh

$$\mathbf{h}_t = \sigma(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

$$y_t = \sigma(W_y \mathbf{h}_t)$$

It depends on the problem

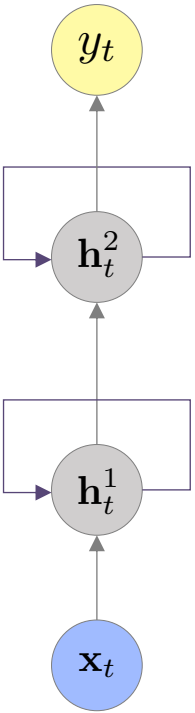
$$W_h \in \mathbb{R}^{p \times p} \qquad W_y \in \mathbb{R}^p$$

$$W_x \in \mathbb{R}^{p \times m}$$

IMPORTANT: weights do not depend on time → It's a parameter sharing

Recurrent Neural Networks

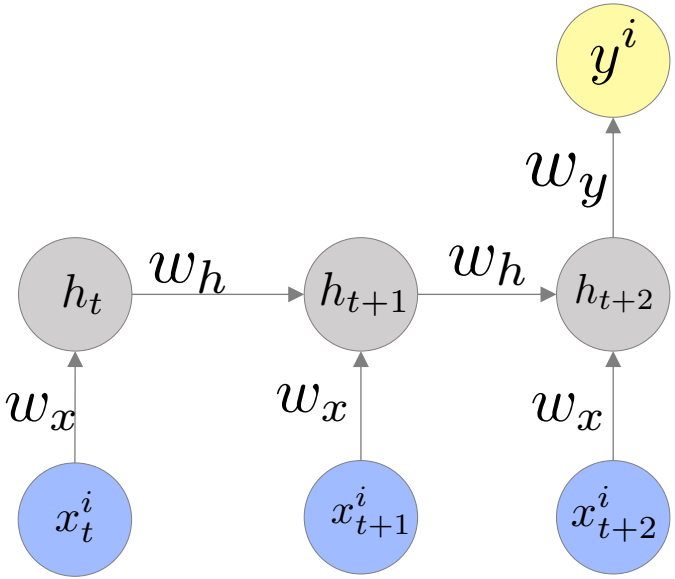
Having multiple hidden layers



$$\mathbf{h}_t^2 = \sigma(W_h^2 \mathbf{h}_{t-1}^2 + W_{12}\mathbf{h}_t^1)$$

$$\mathbf{h}_t^1 = \sigma(W_h^1 \mathbf{h}_{t-1}^1 + W_x \mathbf{x}_t)$$

Backpropagation: an example



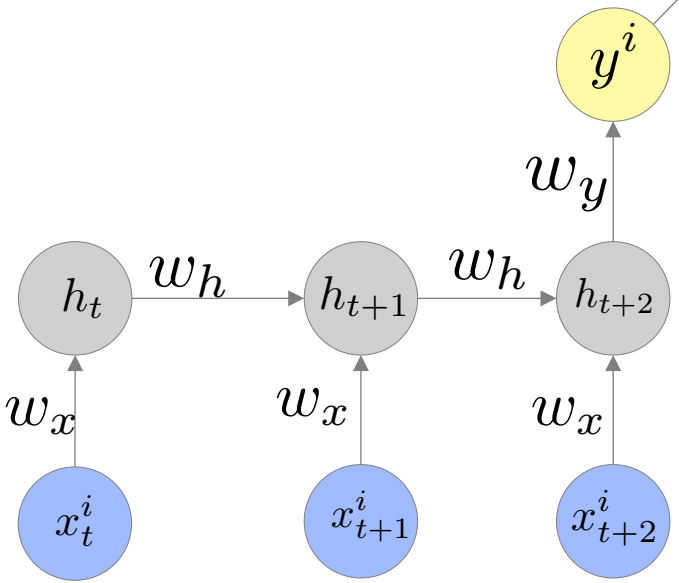
$$\hat{y}^i = f_\sigma(w_y h_{t+2})$$

Let's assume everything (input, output, state, weights) is one-dimensional

$$h_{t+2} = f_\sigma(w_h h_{t+1} + w_x x_{t+2}^i)$$
$$h_{t+1} = f_\sigma(w_h h_t + w_x x_{t+1}^i)$$
$$h_t = f_\sigma(w_h h_{t-1} + w_x x_t^i)$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$



$$\hat{y}^i = f_\sigma(w_y h_{t+2})$$

Let's assume everything (input, output, state, weights) is one-dimensional

$$h_{t+2} = f_\sigma(w_h h_{t+1} + w_x x_{t+2}^i)$$
$$h_{t+1} = f_\sigma(w_h h_t + w_x x_{t+1}^i)$$
$$h_t = f_\sigma(w_h h_{t-1} + w_x x_t^i)$$

Backpropagation: an example

$$J(S; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} (y^i - \hat{y}^i)^2$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

Let's consider the cost related to a single sample

$$J^i(\mathbf{w}) = (y^i - \hat{y}^i)^2$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

Let's consider the cost related to a single sample

$$J^i(\mathbf{w}) = (y^i - \hat{y}^i)^2$$

We want to estimate the partial derivative with respect a specific weight in the network

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x}$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

Let's consider the cost related to a single sample

$$J^i(\mathbf{w}) = (y^i - \hat{y}^i)^2$$

We want to estimate the partial derivative with respect a specific weight in the network

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial w_x} + \\ + \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial w_x} + \\ + \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial w_x}$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

Let's consider the cost related to a single sample

$$J^i(\mathbf{w}) = (y^i - \hat{y}^i)^2$$

We want to estimate the partial derivative with respect a specific weight in the network

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial w_x} + \\ + \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial w_x} + \\ + \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\frac{\partial h_{t+2}}{\partial w_x}$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

Let's consider the cost related to a single sample

$$J^i(\mathbf{w}) = (y^i - \hat{y}^i)^2$$

We want to estimate the partial derivative with respect a specific weight in the network

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\left(\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial w_x} + \right.$$
$$+ \frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial w_x} +$$
$$\left. + \frac{\partial h_{t+2}}{\partial w_x}\right)$$

Backpropagation: an example

$$J(S;\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

Let's consider the cost related to a single sample

$$J^i(\mathbf{w}) = (y^i - \hat{y}^i)^2$$

We want to estimate the partial derivative with respect a specific weight in the network

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\left(\frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t}\frac{\partial h_t}{\partial w_x} + \frac{\partial h_{t+2}}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial w_x} + \frac{\partial h_{t+2}}{\partial w_x}\right)$$

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial h_{t+2}}\sum_{k=t}^{t+2}\left(\frac{\partial h_{t+2}}{\partial h_k}\frac{\partial h_k}{\partial w_x}\right)$$

Backpropagation: an example

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i} \frac{\partial \hat{y}^i}{\partial h_{t+2}} \sum_{k=t}^{t+2} \left(\frac{\partial h_{t+2}}{\partial h_k} \frac{\partial h_k}{\partial w_x} \right)$$

Backpropagation: an example

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i} \frac{\partial \hat{y}^i}{\partial h_{t+2}} \sum_{k=t}^{t+2} \left(\frac{\partial h_{t+2}}{\partial h_k} \frac{\partial h_k}{\partial w_x} \right)$$

$$\frac{\partial h_{t+2}}{\partial h_k} = \prod_{j=k+1}^{t+2} \frac{\partial h_j}{\partial h_{j-1}}$$

Backpropagation: an example

$$\frac{\partial J^i(\mathbf{w})}{\partial w_x} = \frac{\partial J^i(\mathbf{w})}{\partial \hat{y}^i} \frac{\partial \hat{y}^i}{\partial h_{t+2}} \sum_{k=t}^{t+2} \left(\frac{\partial h_{t+2}}{\partial h_k} \frac{\partial h_k}{\partial w_x} \right)$$

$$\frac{\partial h_{t+2}}{\partial h_k} = \prod_{j=k+1}^{t+2} \frac{\partial h_j}{\partial h_{j-1}}$$

- How to incorporate the contributions from all samples in the training set?
- What if multiple hidden layers are present?

- What if the output is a sequence?

Gradients-related issues for long-term dependences

- The computation of the loss gradient as successive multiplication leads to instability of the gradient and may take very long training times
- Many values < 1 lead to **vanishing** gradient problems
- Many values > 1 lead to **exploding** gradient problems

Exploding gradient

- Many values > 1 lead to **exploding** gradient problems: the update with SGD is done with very large steps, leading to bad results

- A possible solution is gradient clipping: if the gradient is greater than some threshold, scale it down before applying SGD update

- You make a step in the same direction but with a smaller step

Vanishing gradient

- Many values < 1 lead to **vanishing** gradient problems: gradient signal far over time is lost because it's much smaller than gradient signal from closer times

- Model weights are updated only with respect to near effects, not long-term effects

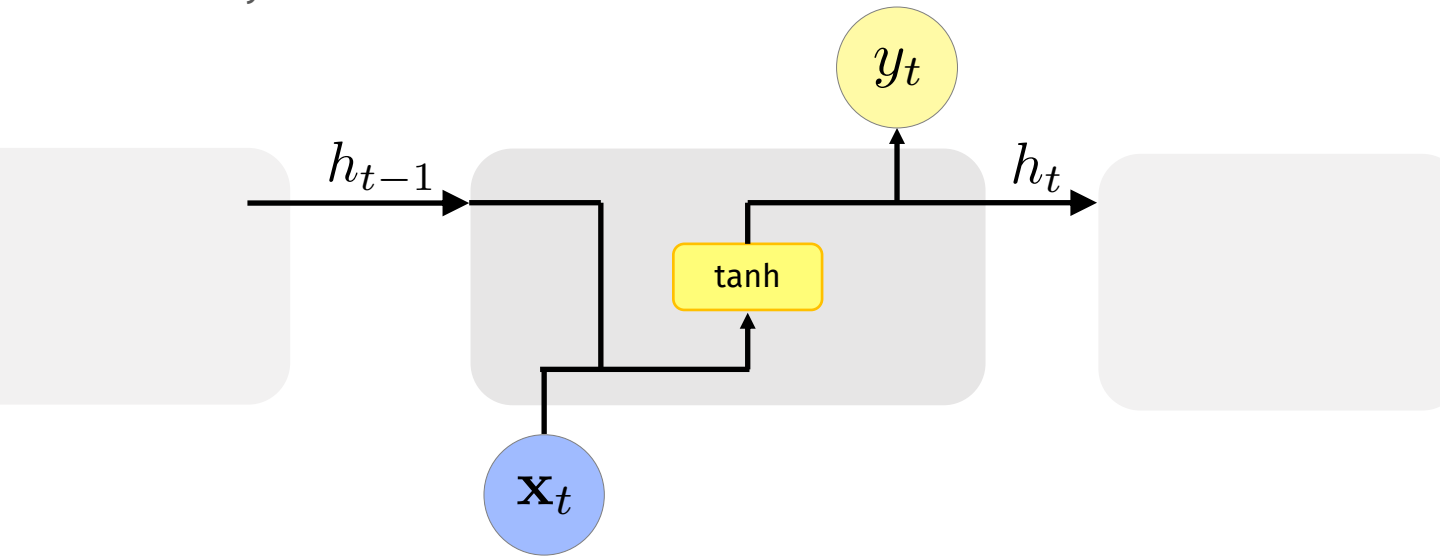
- A possible solution to learn long-term dependences in the data is to use **gated cells**

UniGe | MaLGa

Long-Short Term Memory

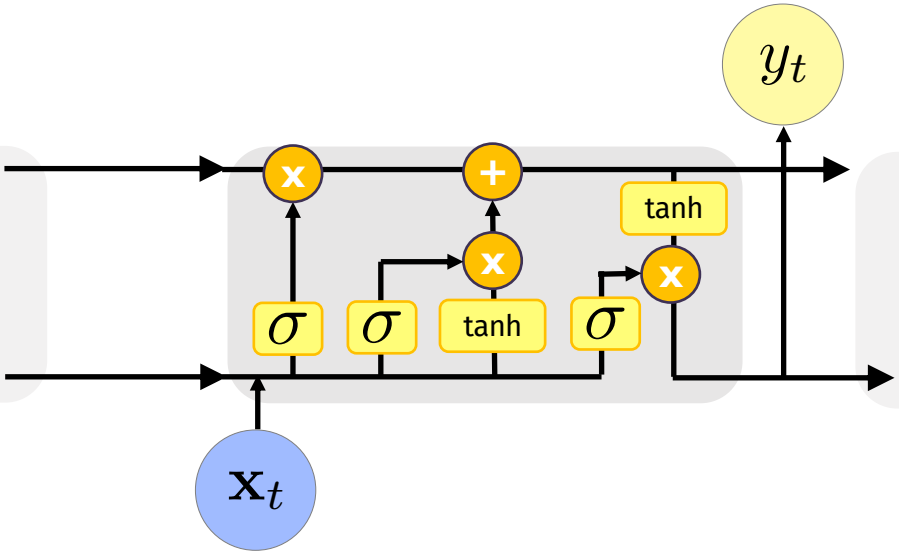
RNNs cells

In standard RNNs, the cells contain a simple computation and their state is constantly re-written



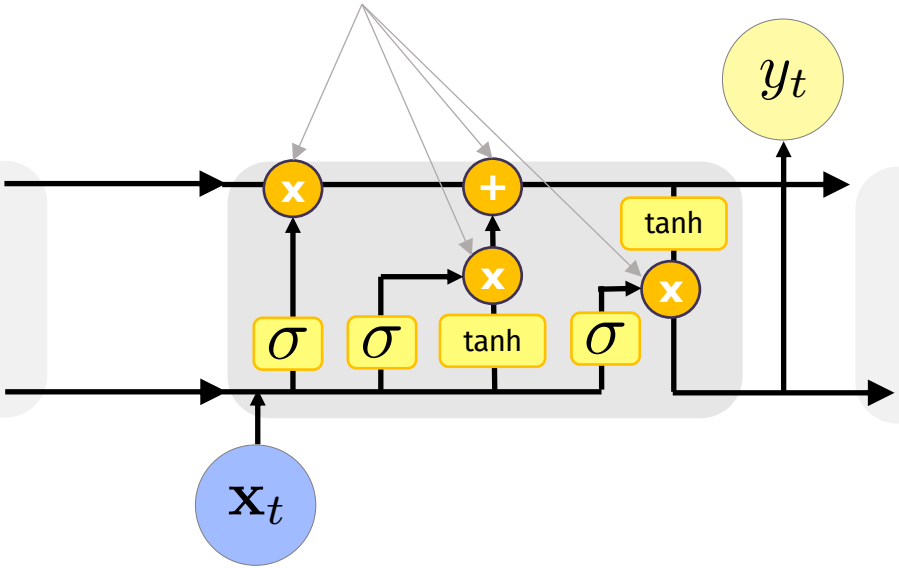
Gated cells

Gated cells contain computational blocks that control information flow



Gated cells

Gated cells contain computational blocks that control information flow



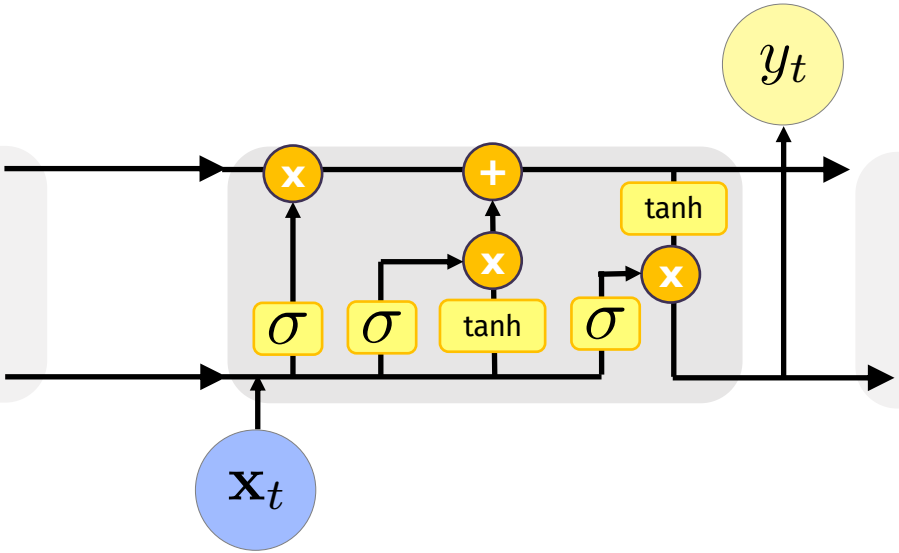
Long-short term memory (LSTM, 1997)

- They consider **connection weights that may change at each time step**
- Information is accumulated over a long duration
- Once the information has been used, it may be useful for the layer to forget the old state or keep the information
- The LSTM learns how to decide when to do that (this is in fact the role of gated units)

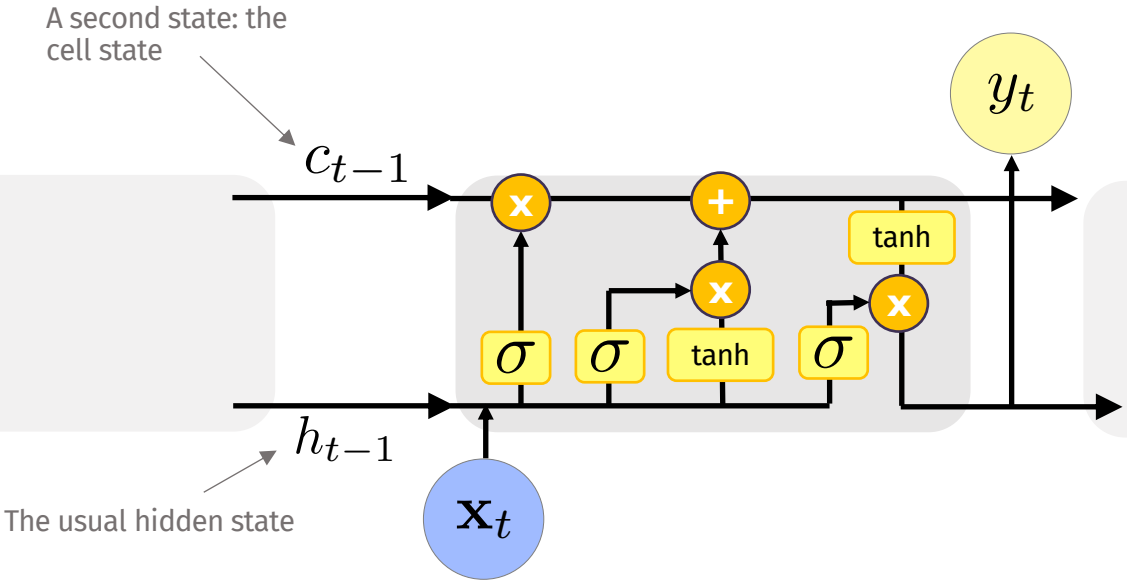
LSTM cell

- It includes two states: a **hidden state** and a **cell state**, both vectors of length n
- The cell stores long-term information. The LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**, vectors again of length n
- Each element of the gates can be open (1), closed (0), or somewhere in-between
- The gates are dynamic: their value is computed based on the current context

LSTM cell: phases of the work

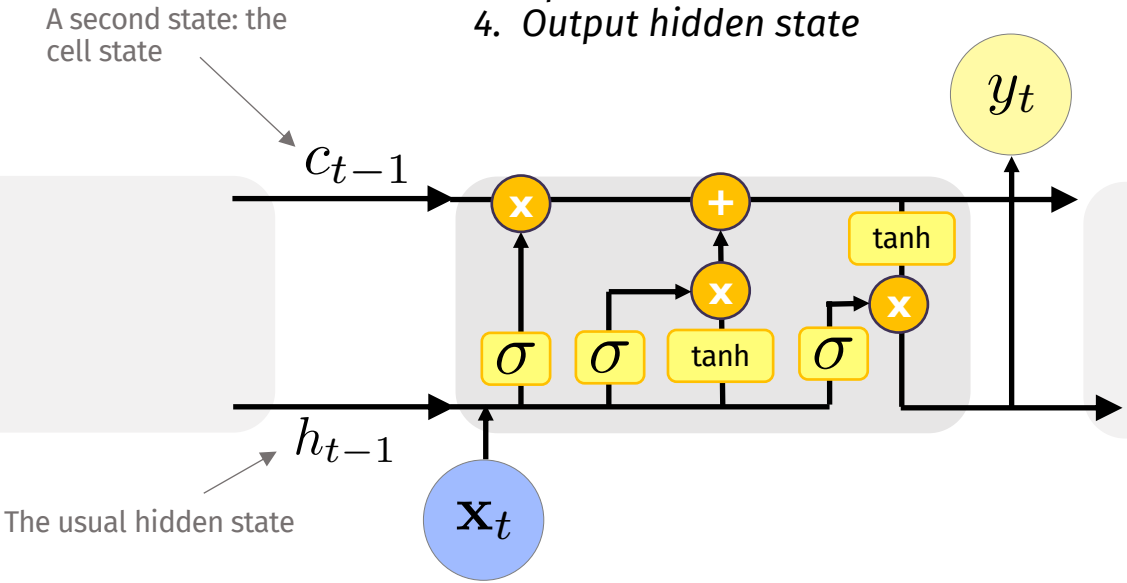


LSTM cell: phases of the work



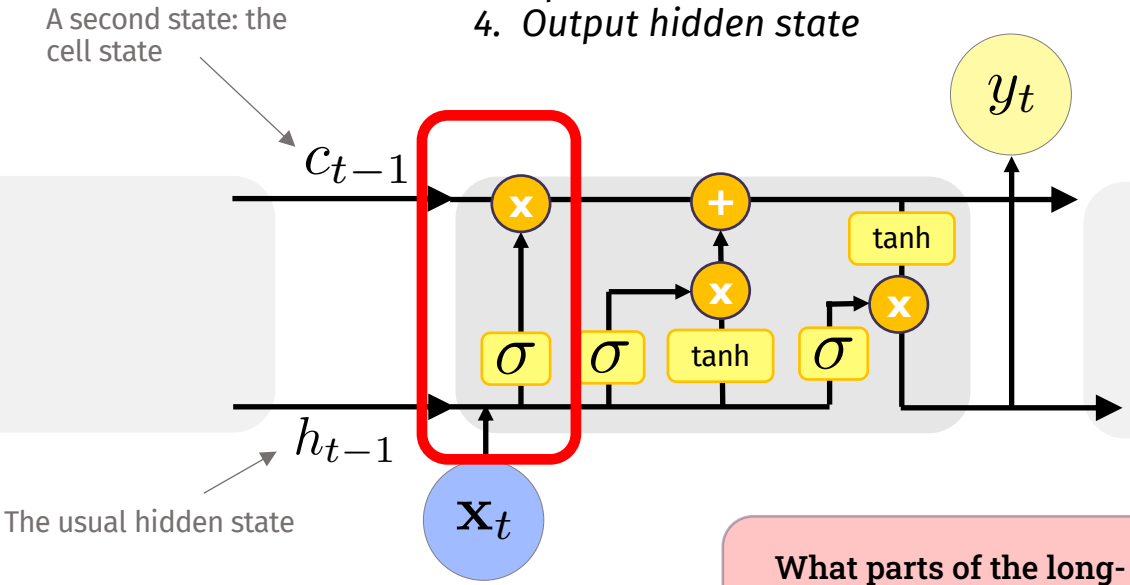
LSTM cell: phases of the work

1. *Forget*
2. *Store input*
3. *Update cell state*
4. *Output hidden state*



LSTM cell: phases of the work

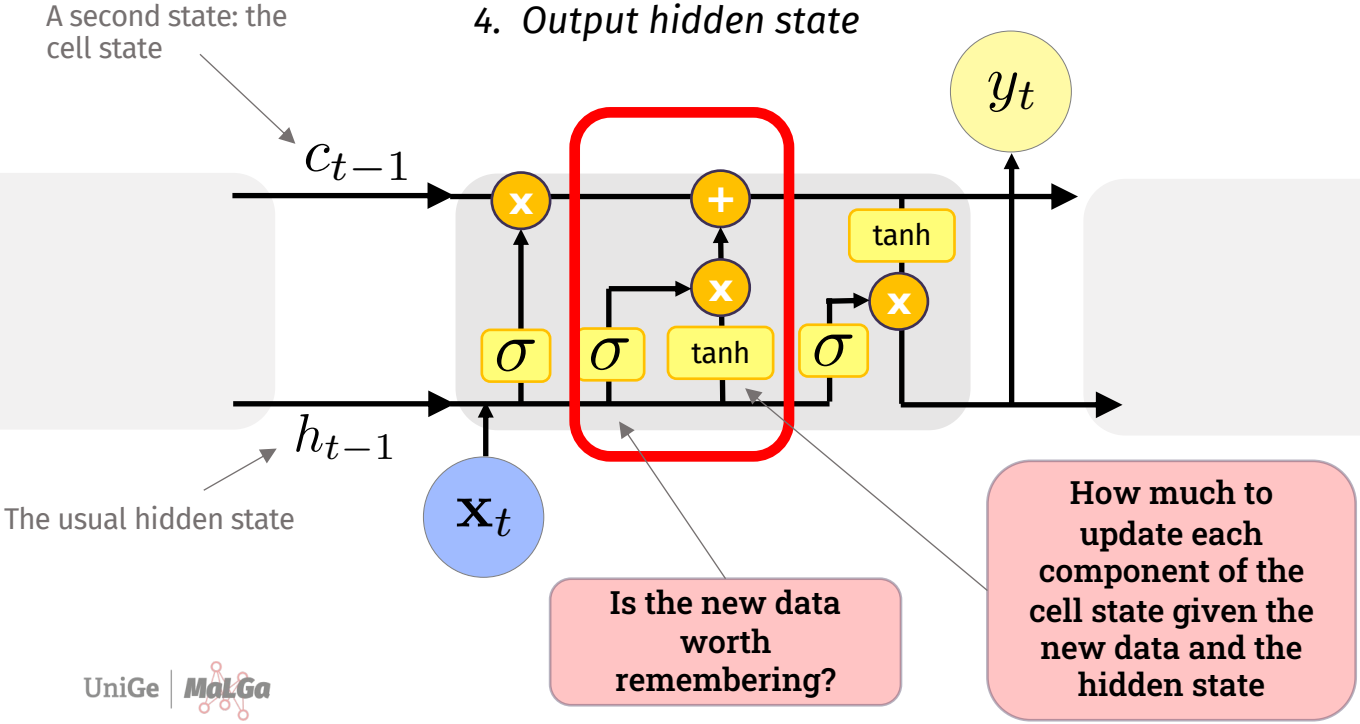
1. ***Forget***
2. *Store input*
3. *Update cell state*
4. *Output hidden state*



What parts of the long-term memory can now be forgotten?

LSTM cell: phases of the work

1. *Forget*
2. ***Store input***
3. *Update cell state*
4. *Output hidden state*



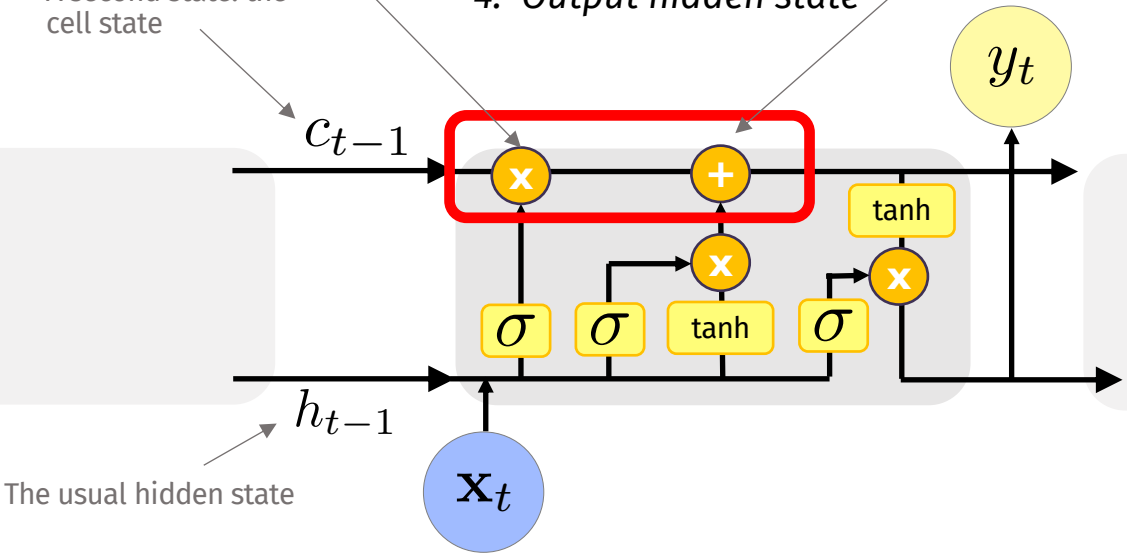
LSTM cell: phases of the work

Element-wise multiplication: here is where the cell can forget some content

1. *Forget*
2. *Store input*
3. ***Update cell state***
4. *Output hidden state*

Sum: here is where the cell can store some new content

A second state: the cell state

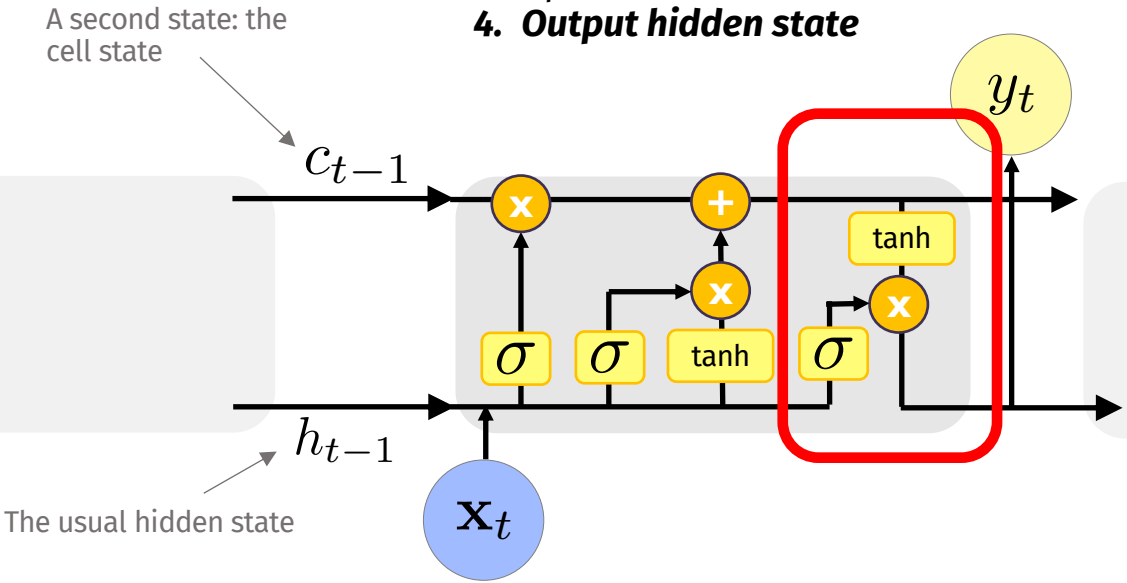


The usual hidden state

The cell state values are selectively updated

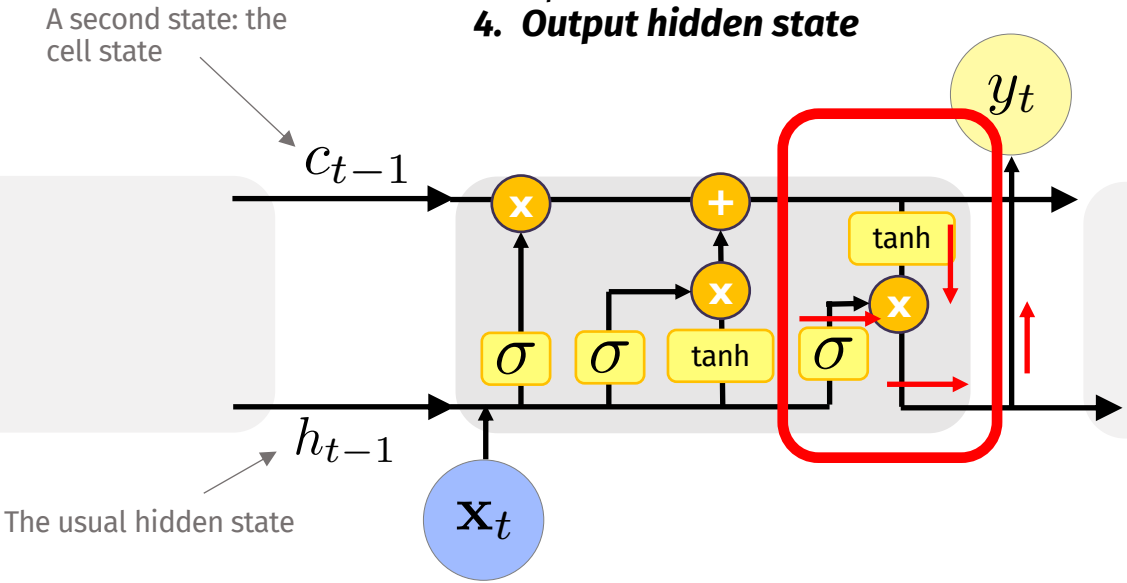
LSTM cell: phases of the work

1. *Forget*
2. *Store input*
3. *Update cell state*
4. ***Output hidden state***



LSTM cell: phases of the work

1. *Forget*
2. *Store input*
3. *Update cell state*
4. ***Output hidden state***



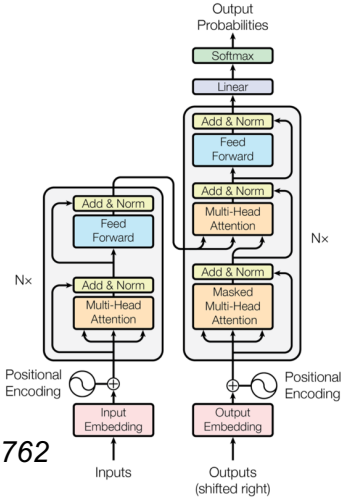
Variations on the theme and recent advances

GRU (Gated Recurrent Unit)

- It includes only the hidden state
- It also has two gates:
  - Update gate: it decides what information to keep and what to throw away
  - Reset gate: it decides how much past information to forget

Transformers

- It includes an attention mechanism that decides at each step which other part of a certain sequence is important



https://arxiv.org/abs/1706.03762

UniGe | MaLGa

Self-attention and Transformers

Transformer (2017)

Originally proposed for language translation, they can be highly parallelized

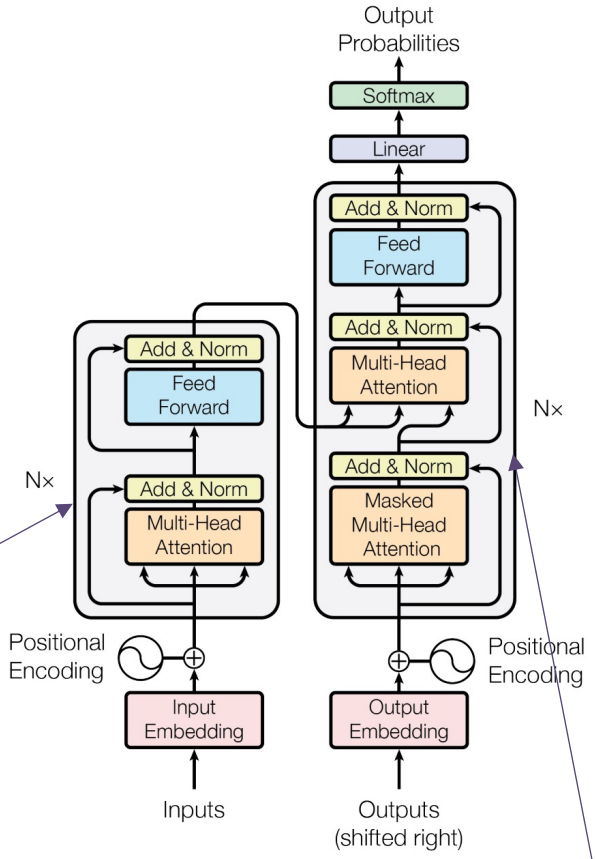


Figure 1: The Transformer - model architecture.

Encoding layers

Decoding layers

From https://arxiv.org/abs/1706.03762

Transformer

- As other models for sequence transduction (e.g. language translation), they are based on an encoder-decoder structure

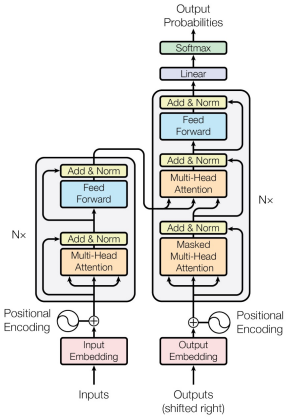


Figure 1: The Transformer - model architecture.

$(x_1, \ldots, x_n)$ Input sequence

ENCODING

$\mathbf{z} = (z_1, \ldots, z_n)$ Sequence of representations

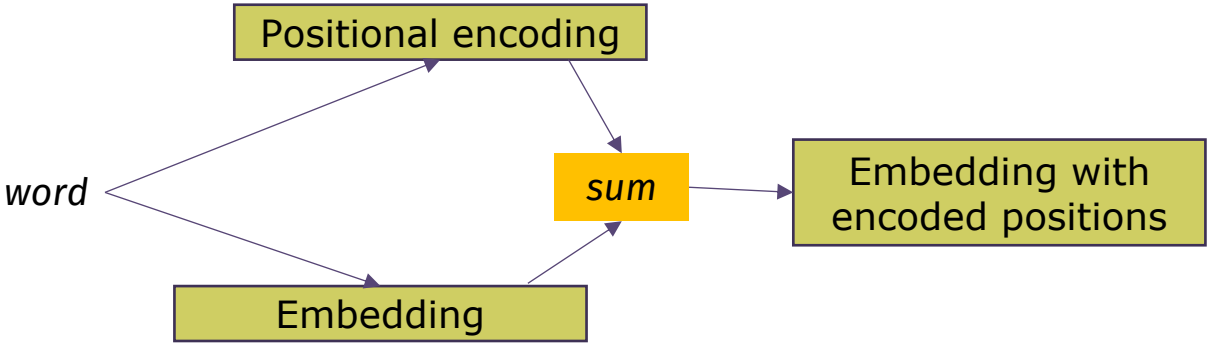
DECODING

$(y_1, \ldots, y_n)$ Output, generated one element at a time

At each time step the model is auto-regressive, as the previously generated symbols are used as additional output in the decoder

What about the position of the words in the sequence?

- We need to «guide» the network to see the words in the correct order
- This is done by means of positional encoding



Transformer: main structure

A stack of N identical layers each one composed by multi-head self-attention and a fully connected net

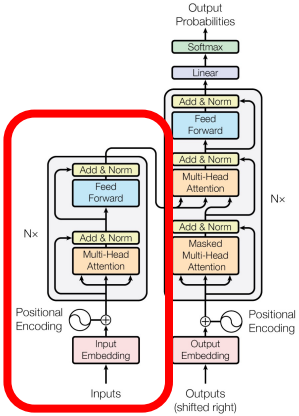
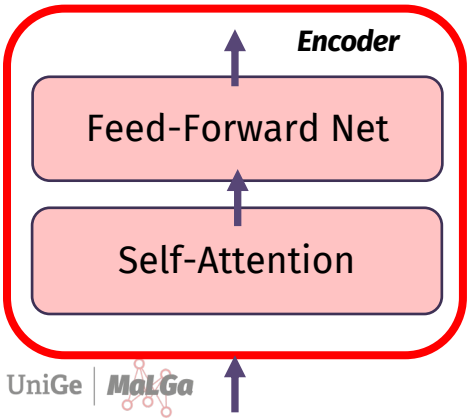


Figure 1: The Transformer - model architecture.



Transformer: main structure

A stack of N identical layers each one composed by multi-head self-attention and a fully connected net

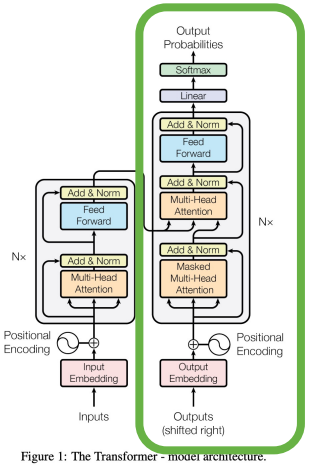
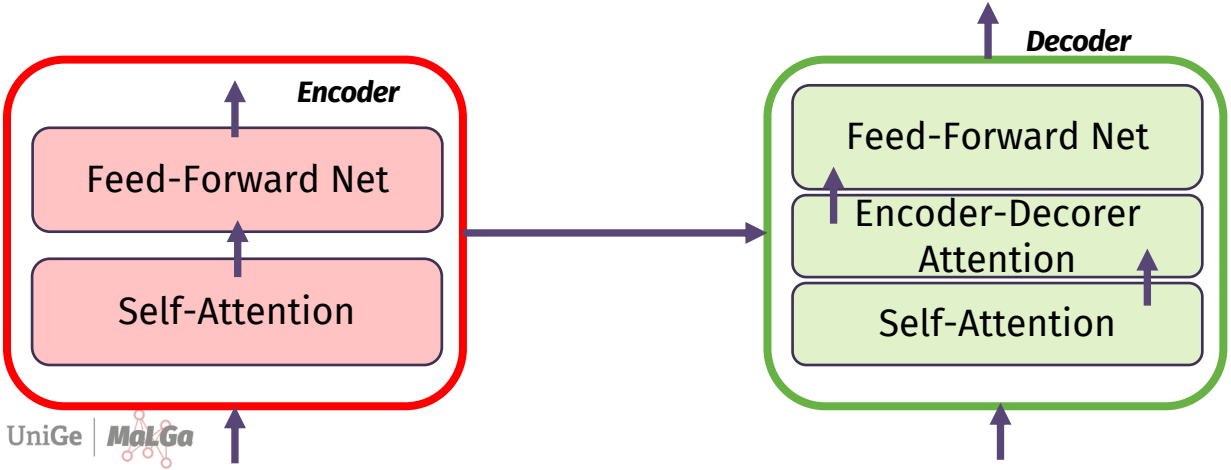


Figure 1: The Transformer - model architecture.

A stack of N identical layers each one composed by masked multi-head self-attention, multi-head self-attention, and a fully connected net



Transformer: main structure

A stack of N identical layers each one composed by multi-head self-attention and a fully connected net

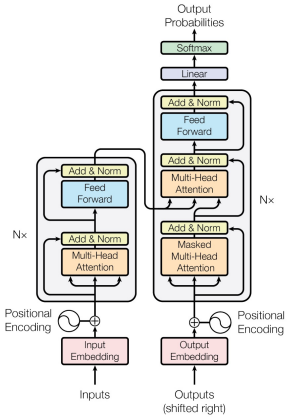


Figure 1: The Transformer - model architecture.

A stack of N identical layers each one composed by masked multi-head self-attention, multi-head self-attention, and a fully connected net

RESIDUAL CONNECTION: the output of each layer is

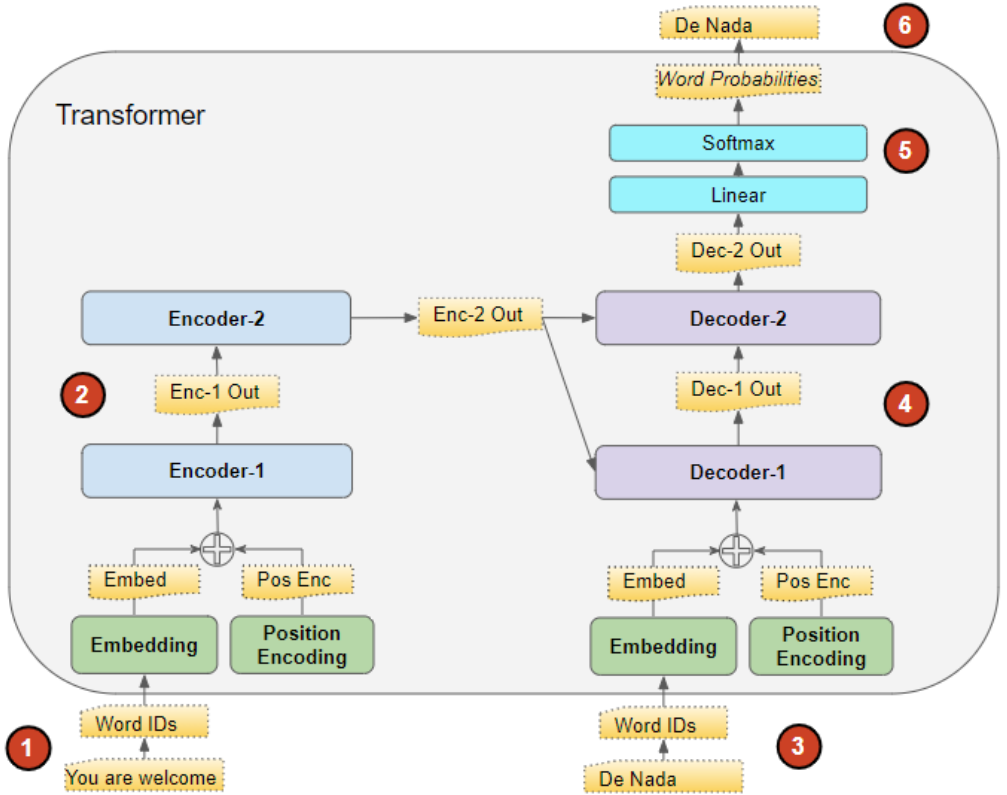
$$\sigma(x + f(x))$$

Where x is the input to the layer, while f(x) is the function implemented by the layer itself

What is self-attention?

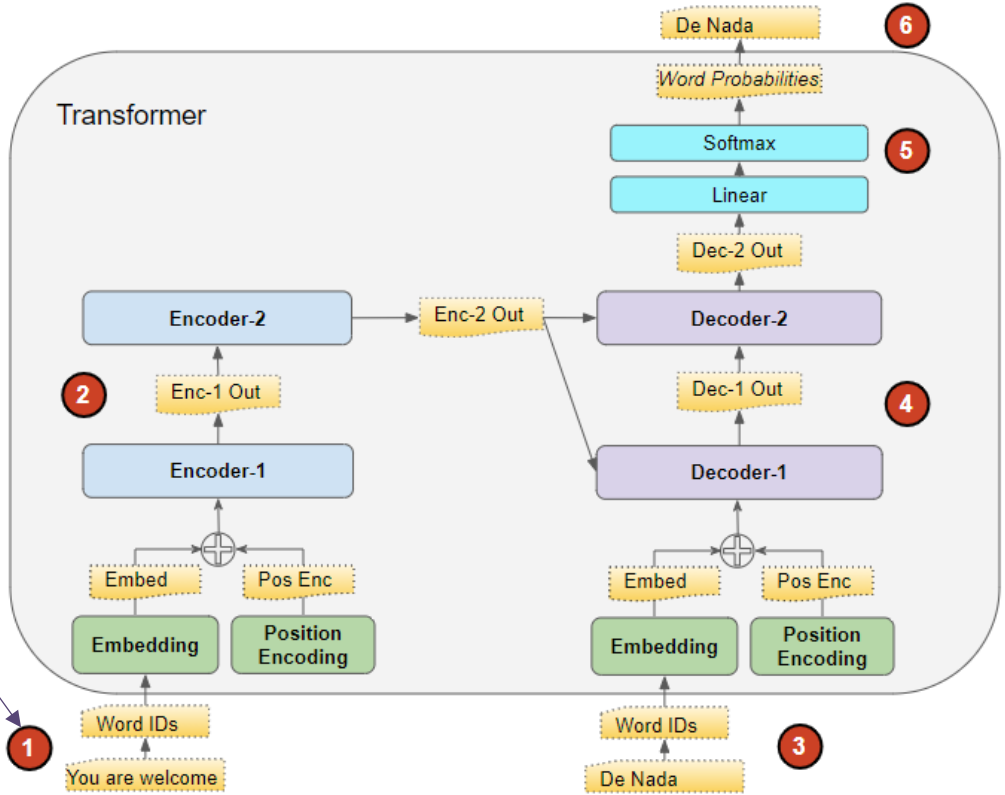
Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Training a Transformer



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Training a Transformer

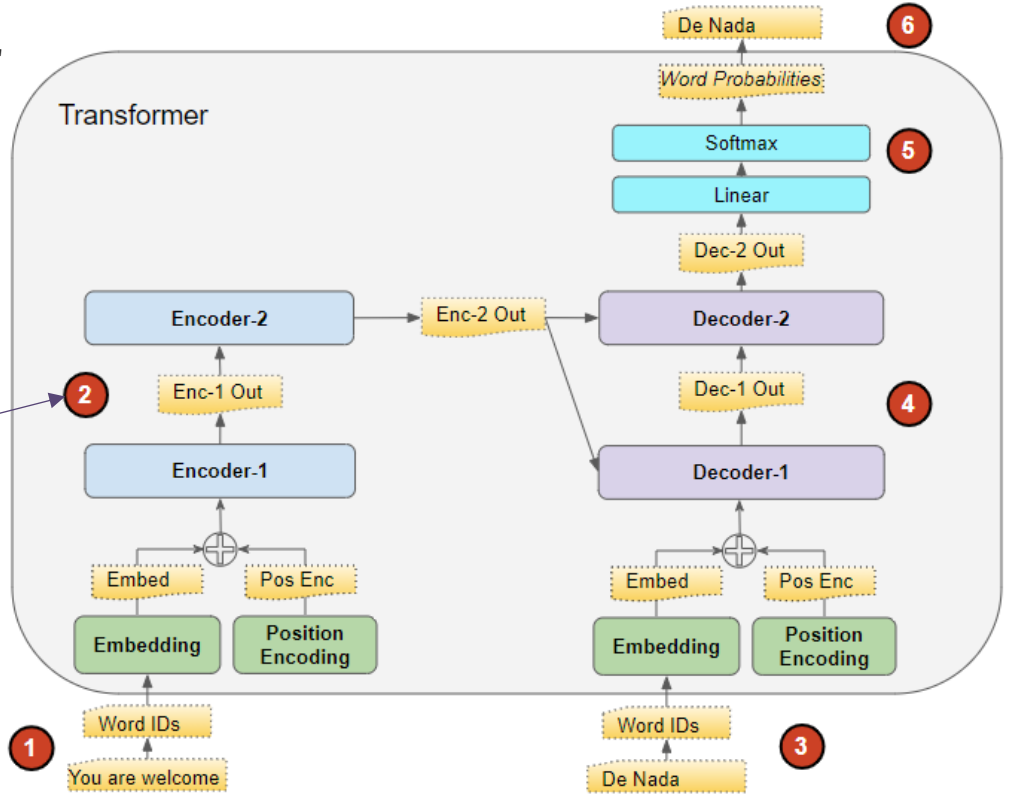


The input sequence is converted into embeddings+positional encoding

Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Training a Transformer

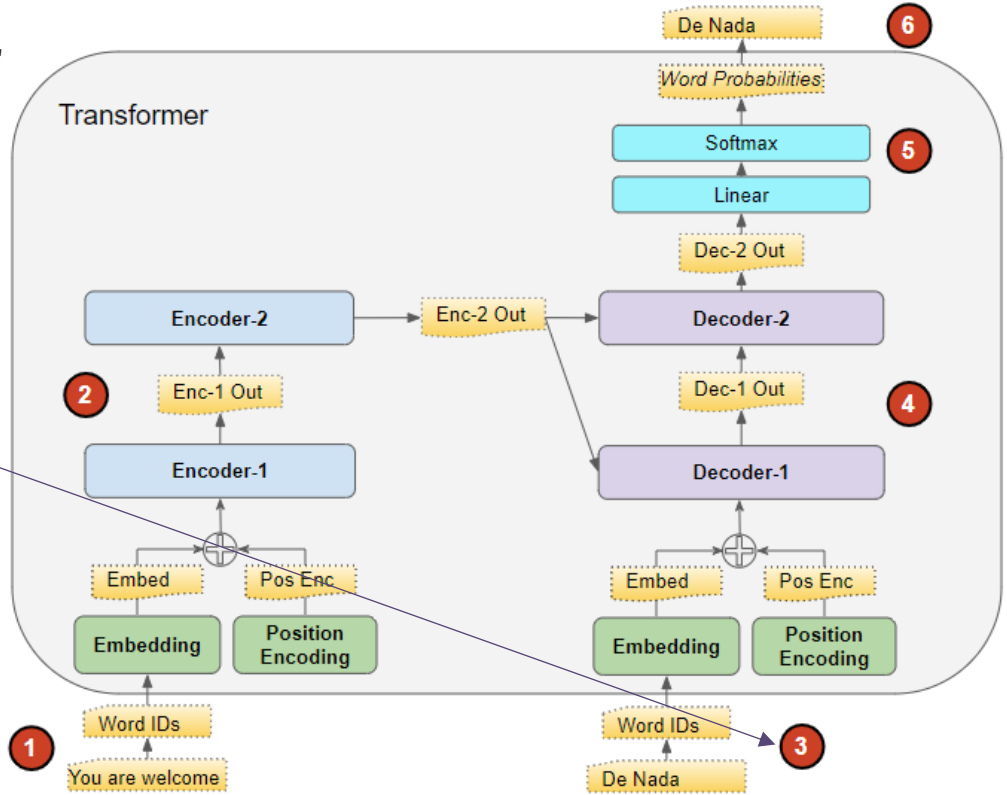
The encoders in the stack process the embeddings and produce an encoded representation



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Training a Transformer

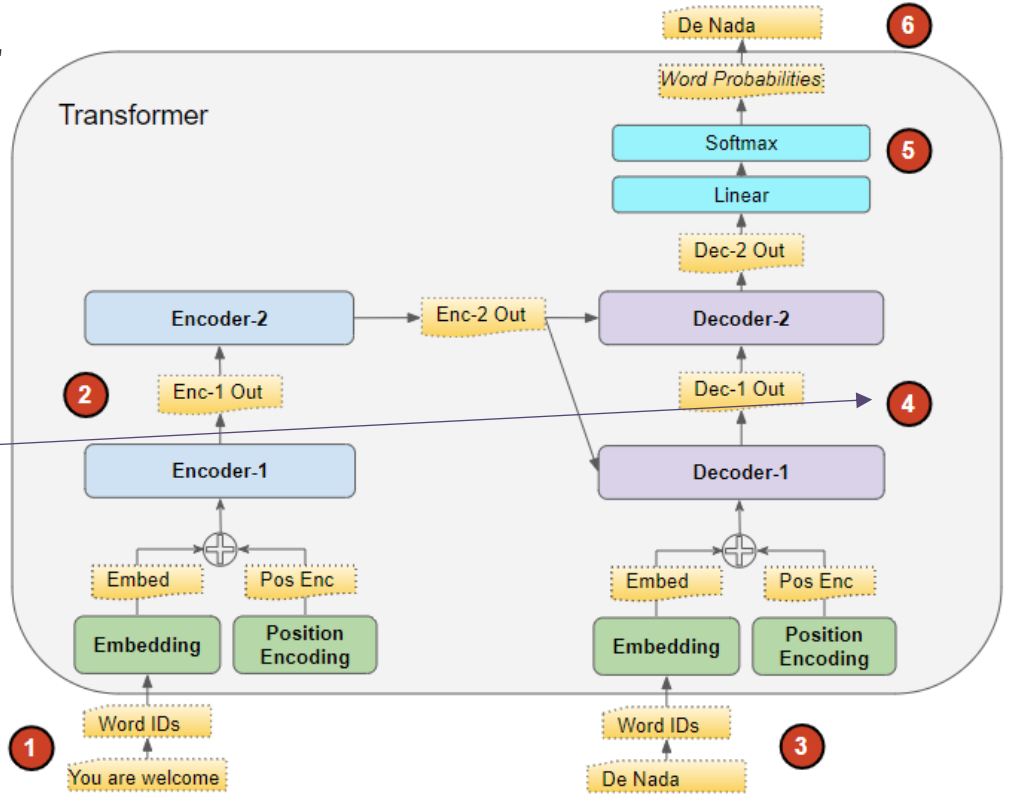
The target sequence is converted into embeddings+positional encoding



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Training a Transformer

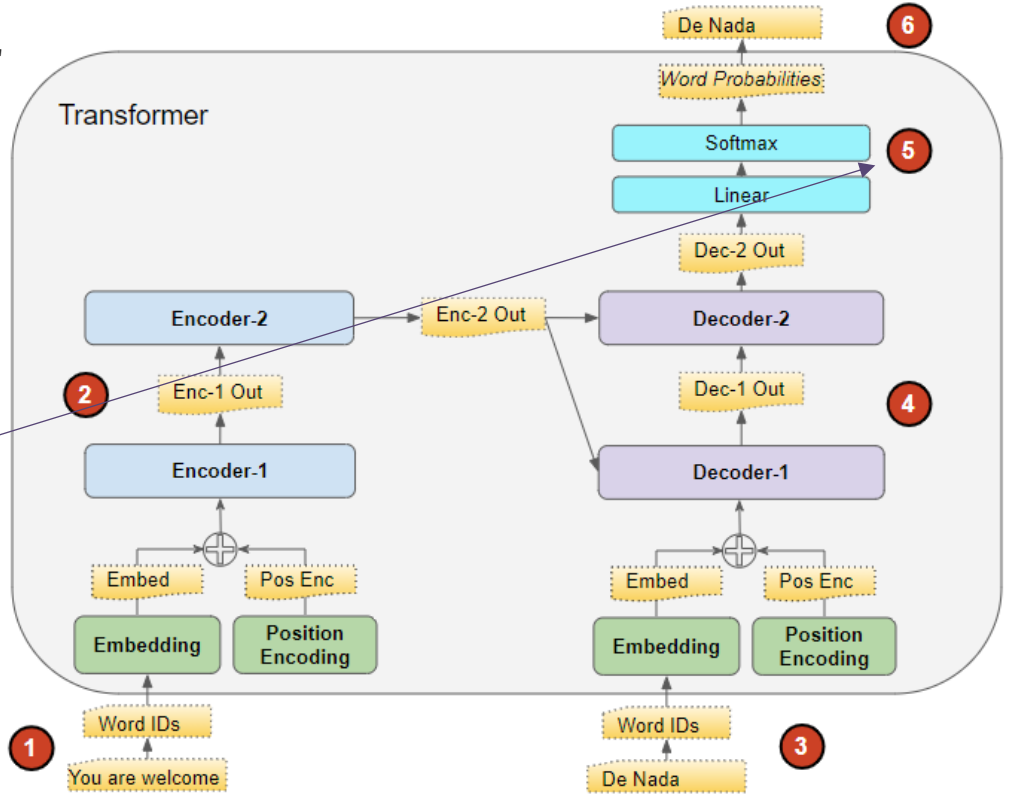
The decoders in the stack processes the embeddings + the encoded representation to produce a decoded representation of the target sequence.



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

Training a Transformer

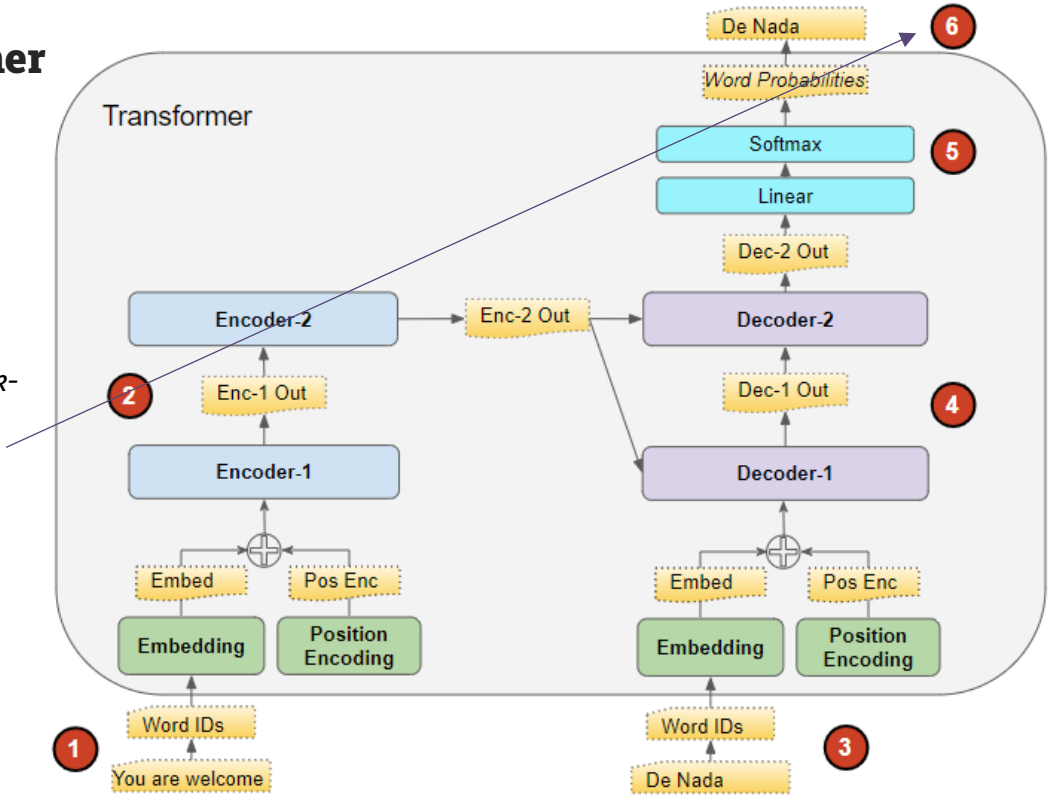
The output layer converts the decoded representations into word probabilities and produce the output



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

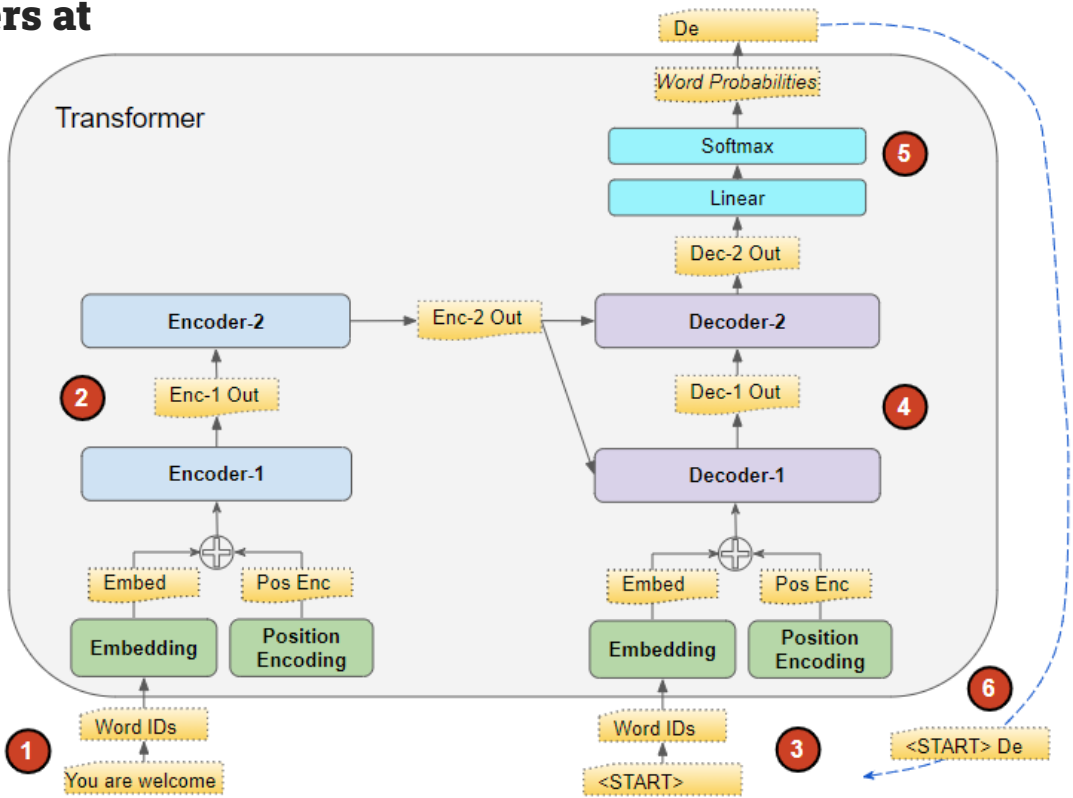
Training a Transformer

The loss function is computed for the back-propagation



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

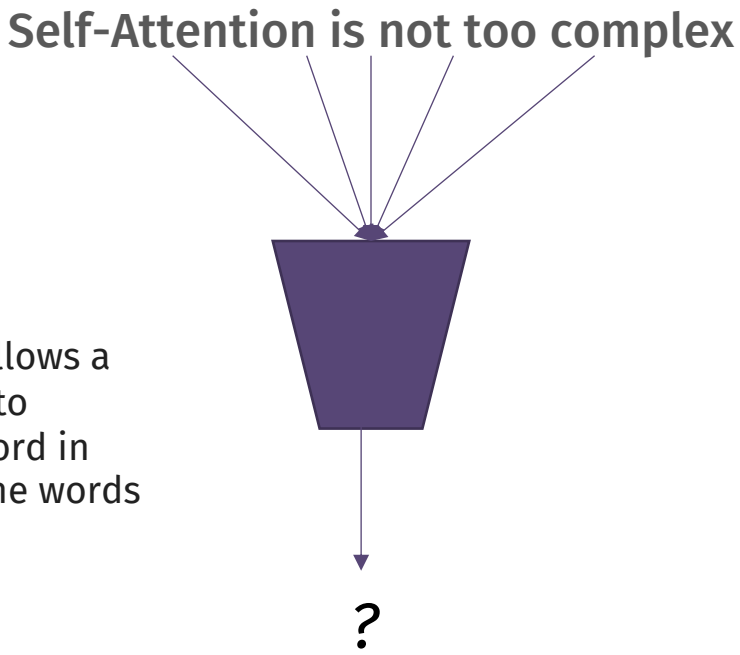
Transformers at inference time



Example from https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

What is self-attention?

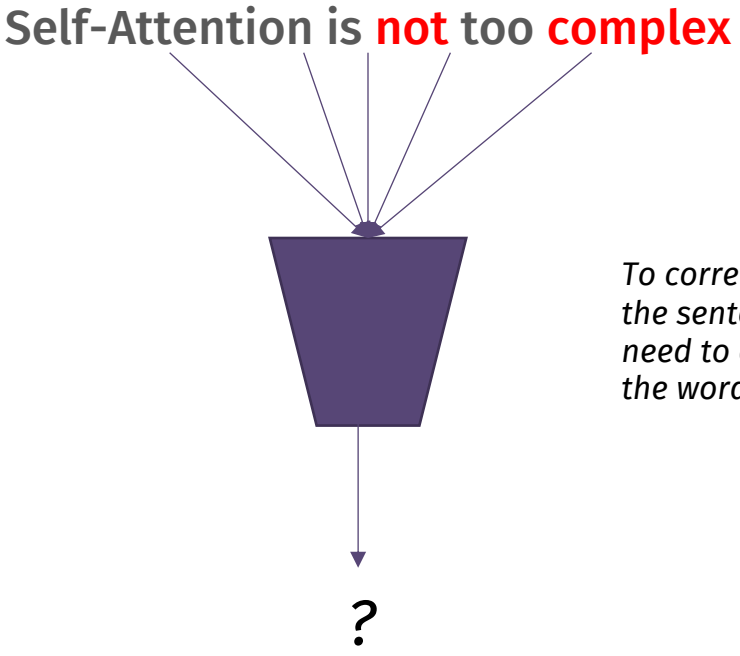
Another example (on sentiment classification)



Self-attention allows a neural network to understand a word in the context of the words around it.

Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

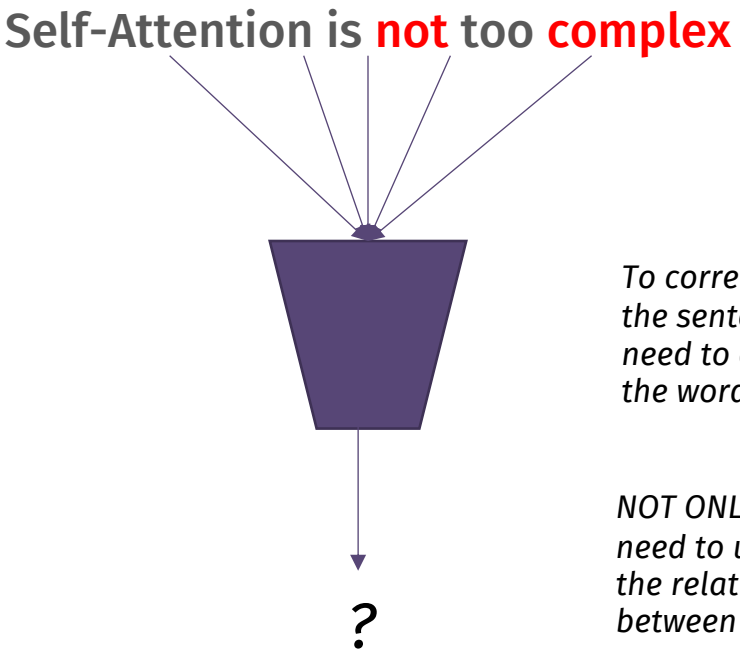
What is self-attention?
Another example (on sentiment classification)



To correctly classify the sentence we need to consider all the words in it

Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

What is self-attention?
Another example (on sentiment classification)



To correctly classify the sentence we need to consider all the words in it

NOT ONLY: we also need to understand the relations between them

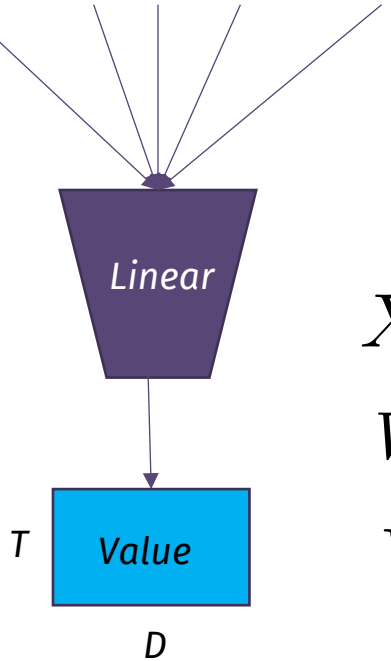
Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

What is self-attention?
An example on sentiment classification

Self-Attention is not too complex

We encode each word (text) into a numerical embedding of size N

We pass the T words embeddings through a linear layer, obtaining the values, each one of size D



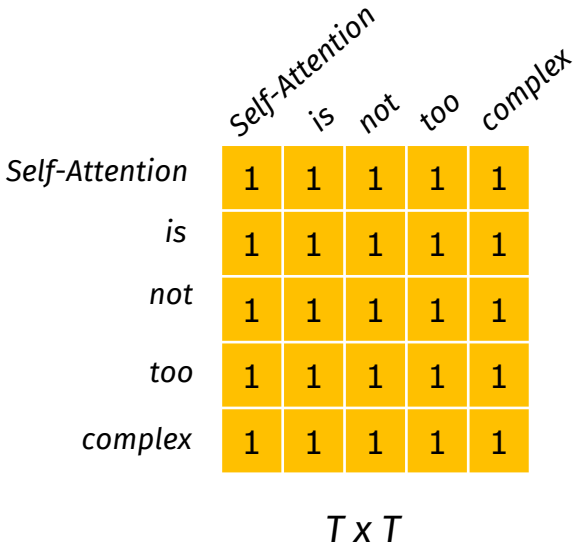
$$X \in \mathbb{R}^{T \times N}$$

$$W_V \in \mathbb{R}^{N \times D}$$

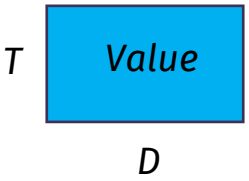
$$V = XW_V$$

Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

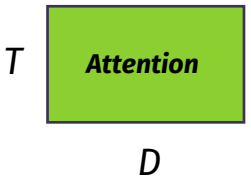
Combining values/words



$$V = XW_V$$



This is what we want to obtain

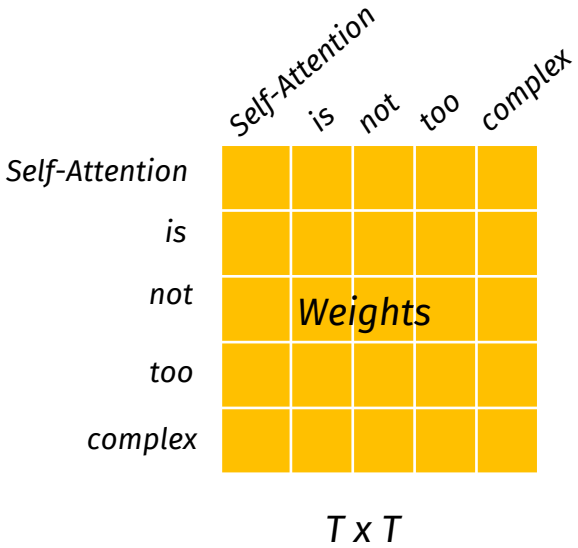


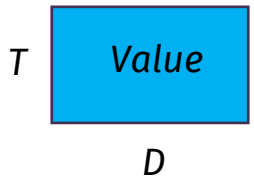
Here we would assume the same importance for all relationships... but we would like that, for instance, the relation between «not» and «complex» was more important than the one between «is» and «too»

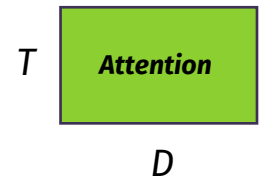
Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

Learning the attention weights

Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508



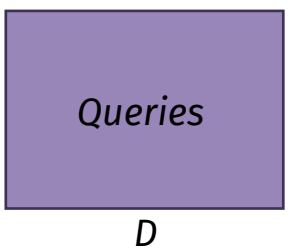


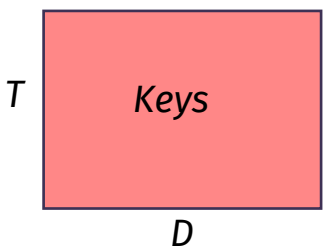


$$W = Q \cdot K^T$$

$$Q = XW_Q$$

$$W_Q \in \mathbb{R}^{N \times D}$$





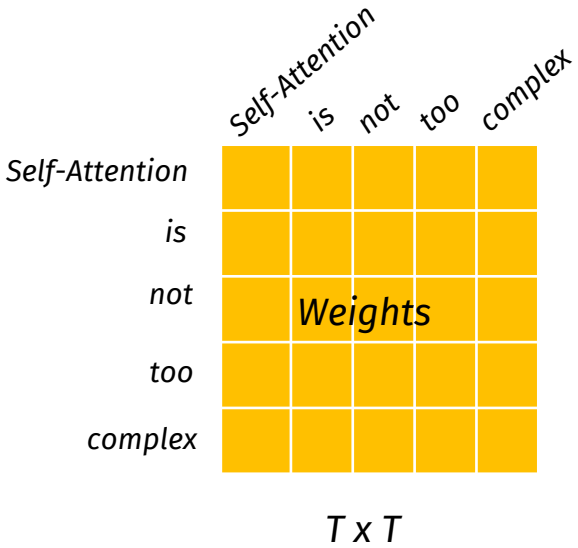
$$K = XW_K$$

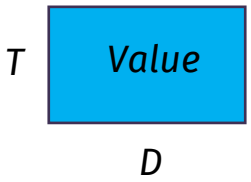
$$W_K \in \mathbb{R}^{N \times D}$$

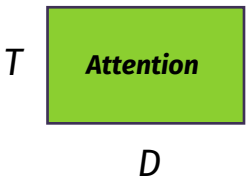
Learning the attention weights

Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

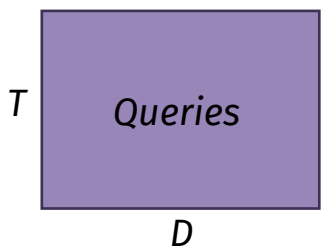
***Intuition**: we want the W matrix to weight the relationship between word$_i$ as a context for word$_j$. We employ other two linear nets*

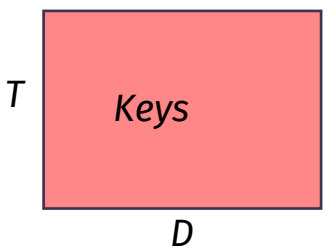




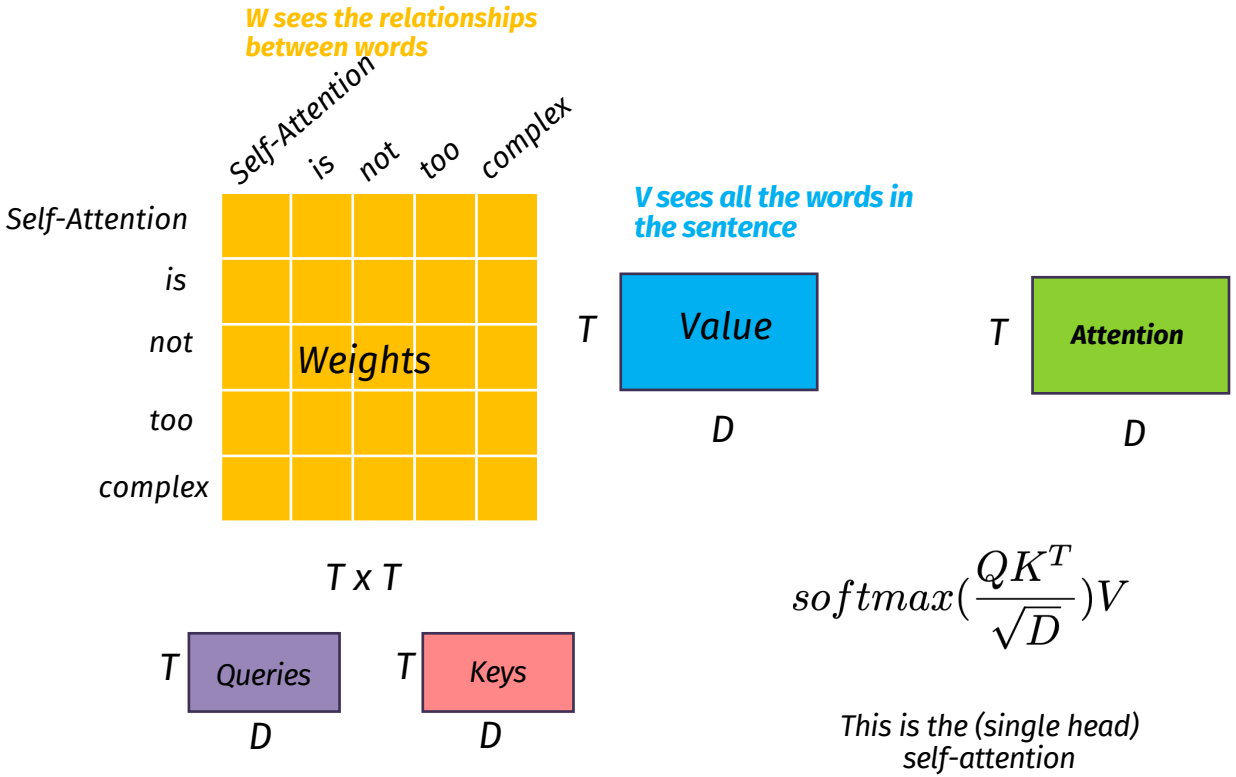


$$W = Q \cdot K^T \rightarrow \frac{Q \cdot K^T}{\sqrt{D}}$$



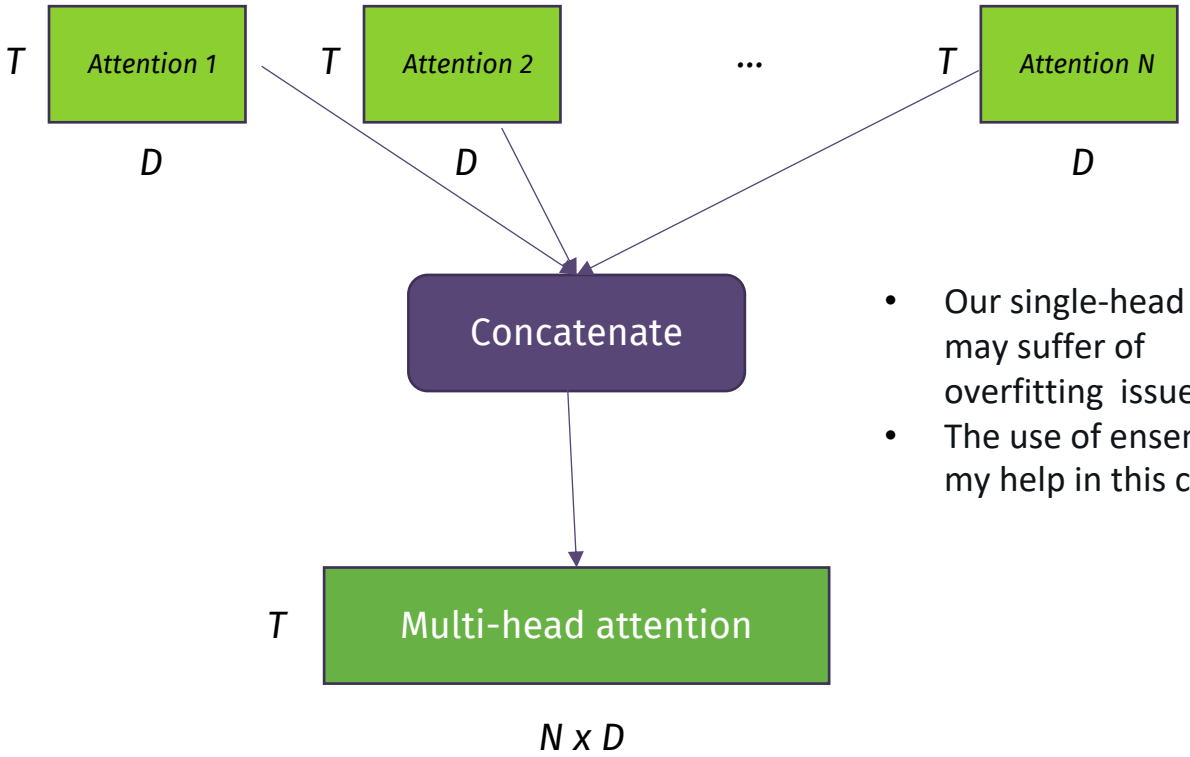


Single-head attention



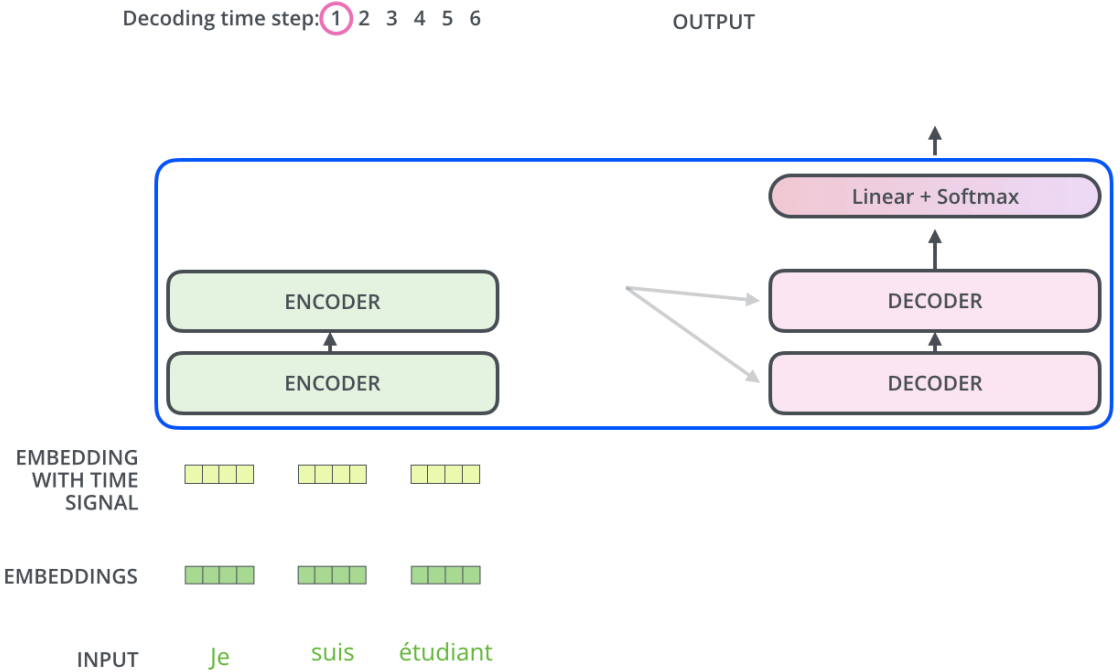
Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

Multi-head attention



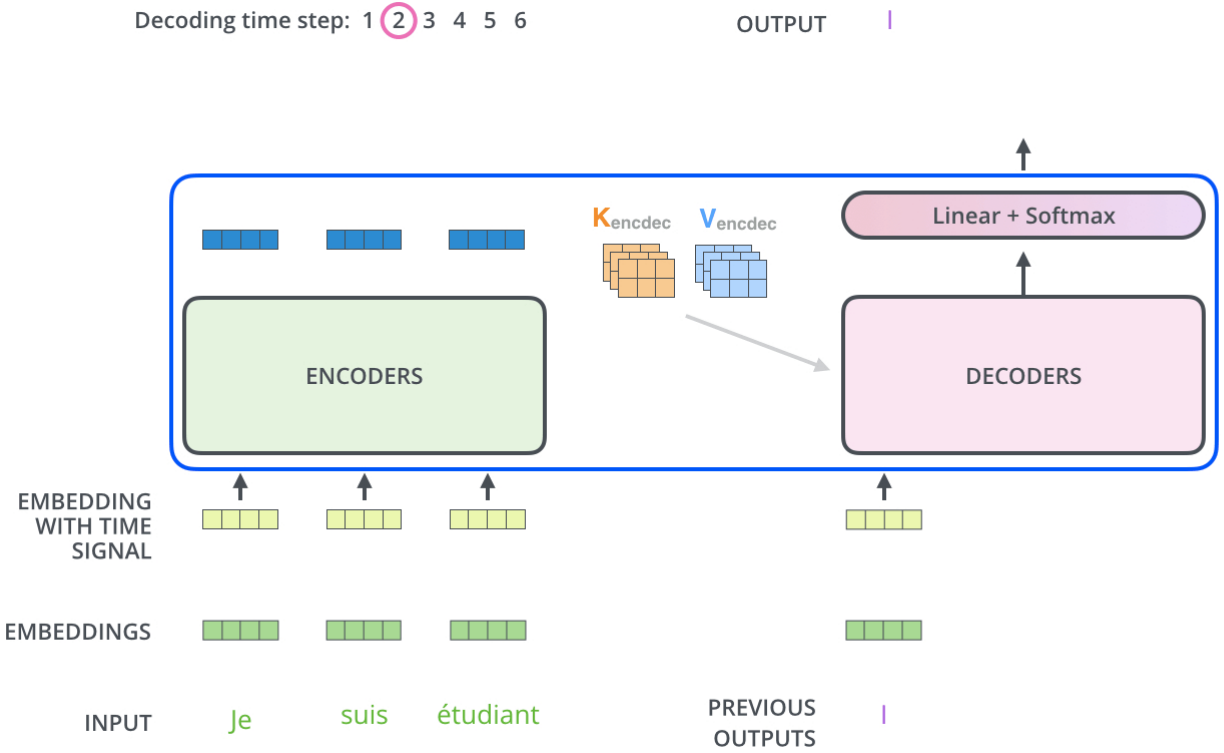
- Our single-head net may suffer of overfitting issues
- The use of ensembles my help in this case

The decoder side



From http://jalammar.github.io/illustrated-transformer/

The decoder side



From http://jalammar.github.io/illustrated-transformer/

The decoder side: masked attention

- In the decoder, the self-attention layer is only allowed to consider earlier positions in the output sequence (it can not “see” the future)

- This is achieved using masked attention: future positions are set to –inf before the softmax step

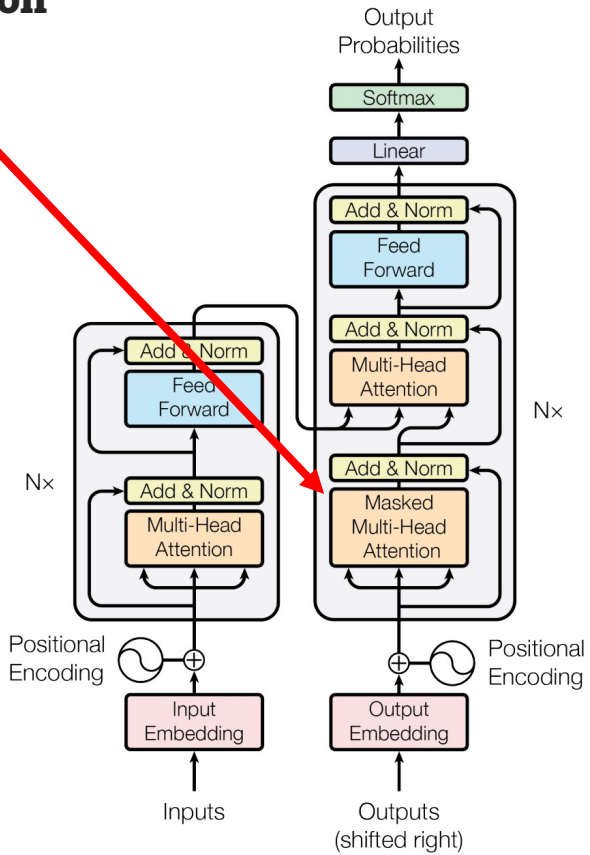


Figure 1: The Transformer - model architecture.

UniGe