UniGe | MaLGa

From Recurrent Neural Networks to Transformers

Advanced Machine Learning
Master degree in Computer Science

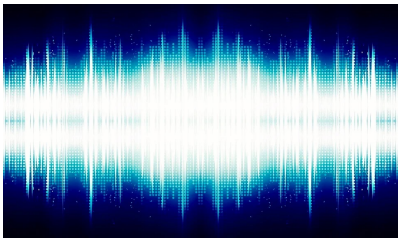
Nicoletta Noceti
Machine Learning Genoa Center – University of Genoa

UniGe | MaLGa

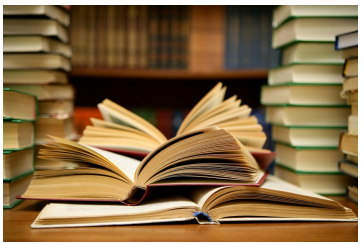
Sequential data and problem formulation

Dealing with sequential data

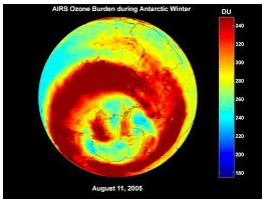
- Today we consider data in the form of sequences



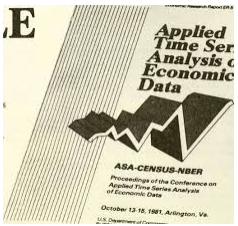
Audio



Text

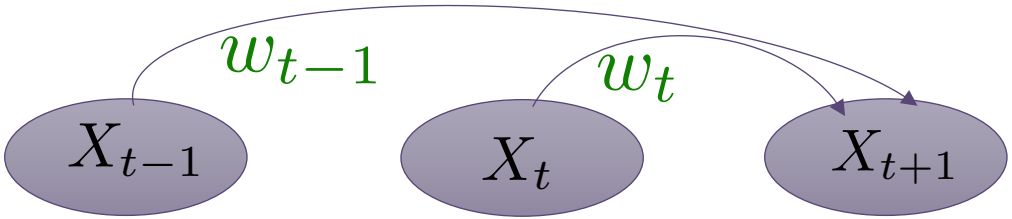


IoT

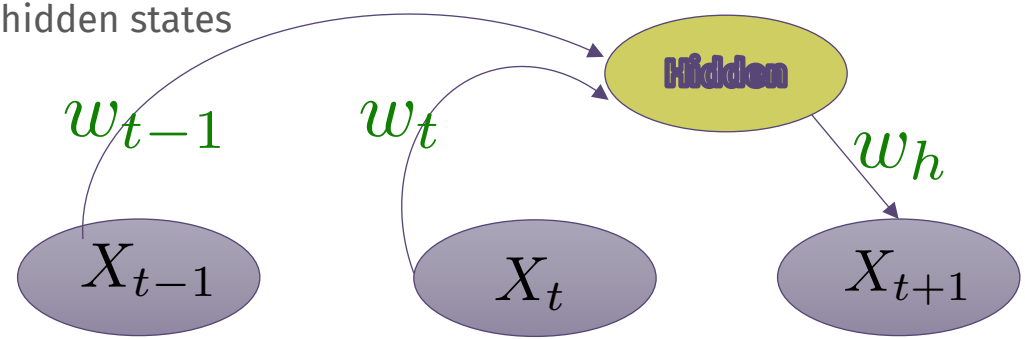


Dealing with sequences: a first summary

- Autoregressive models



- "Using" hidden states



Different formulations of the problem

- Let us introduce a variety of problem formulations involving a special kind of sequential data, i.e. text

"This morning I am happy since I am taking my dog for a walk"

- What are the problems that we may want to address?

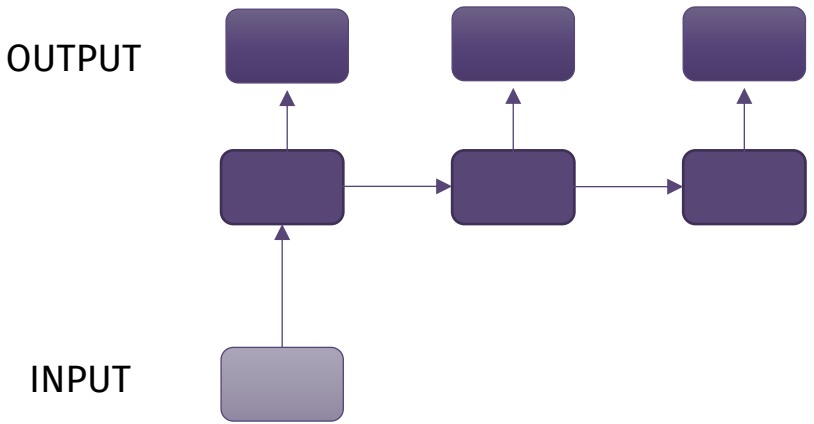
"This morning I am happy since I am taking my dog for a walk"

"This morning I am happy since I am taking my dog for a walk"

Different formulations of the problem

One-to-may: the input is in a standard format (not a sequence!), the output is a sequence

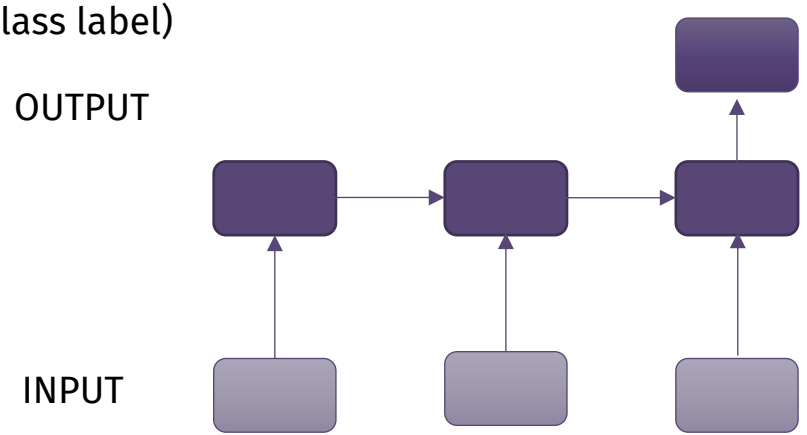
Example of applications: image captioning (input: image, output: text describing the image content)



Different formulations of the problem

Many-to-one: the input is a sequence, the output is a fixed-size vector (not a sequence!)

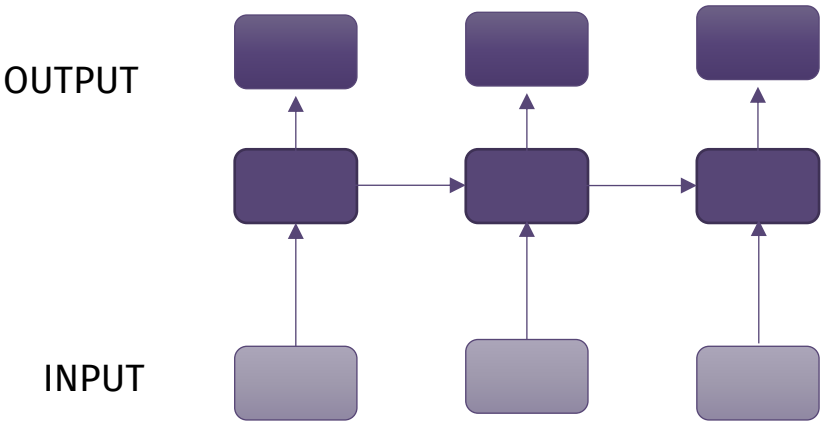
Example of applications: sentiment analysis (input: text, output: class label)



Different formulations of the problem

Direct Many-to-many: input and output are both sequences

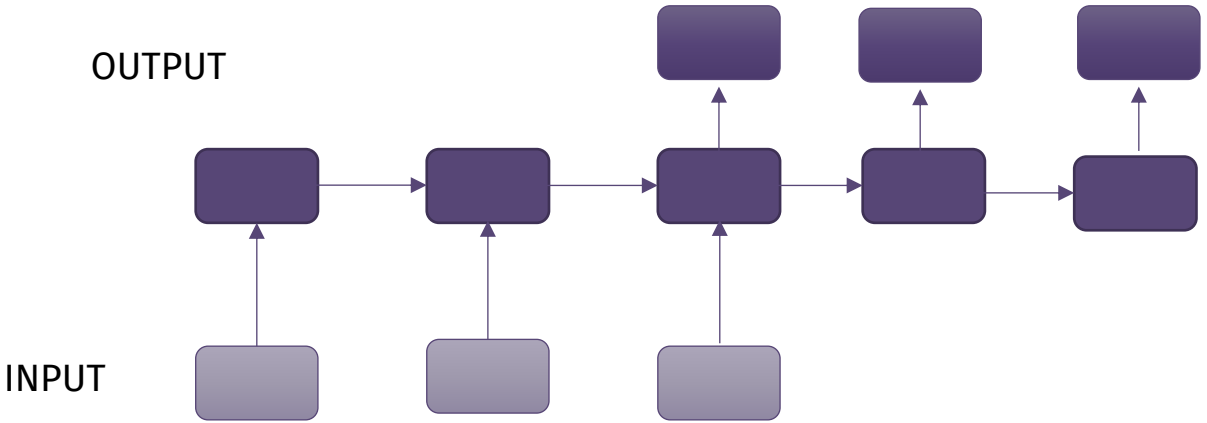
Example of applications: video captioning (input is a sequence of images, output is text)



Different formulations of the problem

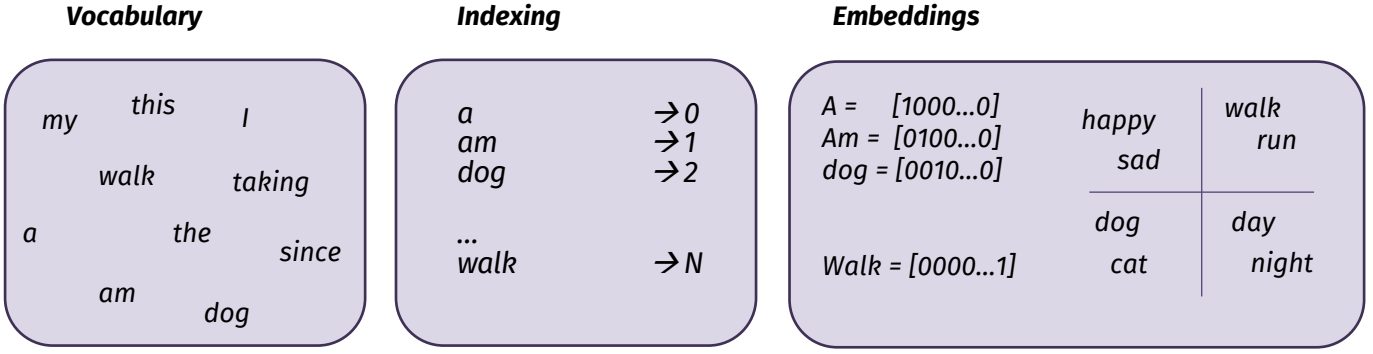
Delayed Many-to-many: input and output are both sequences

Example of applications: language translation (input is a text, output is a text)

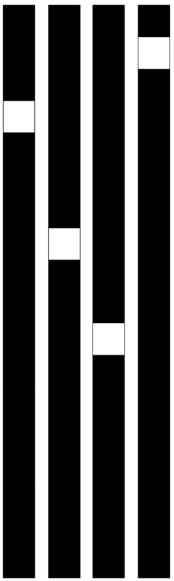


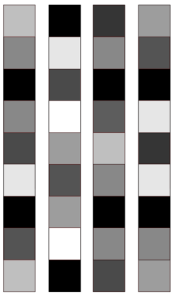
A special case: text data

- ML can not (directly) handle text data… we need a numerical descriptor
- Word embeddings can do the job



One-hot encoding vs word embeddings





One-hot encoding:
- *Very sparse*
- *High dimensional*
- *Hard-coded*

Word embeddings:
- *Dense*
- *Lower dimensional*
- *Learned from data*

UniGe | MaLGa

An example

An example: predicting the next word

“This morning I took my dog for a walk”

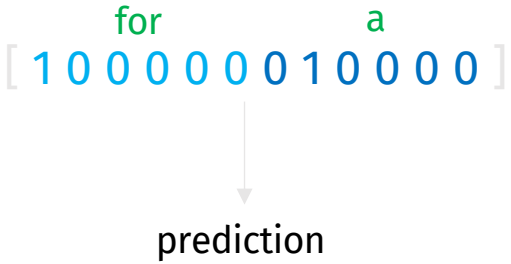
- Predicting the last word in the sentence given all or some of the previous words

- Let’s reason on possible solutions

Solution 1

"This morning I took my dog for a walk"

- Using small fixed windows and one-hot encoding



Problem: what if the few words are not informative?
e.g. *"Italy is where I grew up but now I live in London. I speak fluent _____"*
Longer-term dependences may be needed!

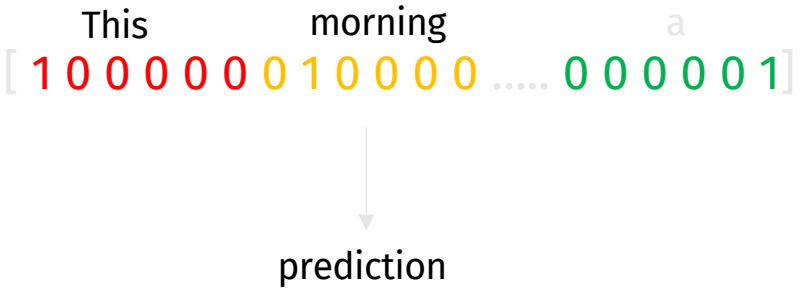
Solution 2

- Using all the available words to form an embedding

Problem: order is not preserved
e.g. *"The food was good, not bad at all" vs "The food was bad, not good at all"*

Solution 3

- Going back to solution 1 with larger windows



Problem: no parameter sharing, and the knowledge we may derive does not transfer is words appear elsewhere

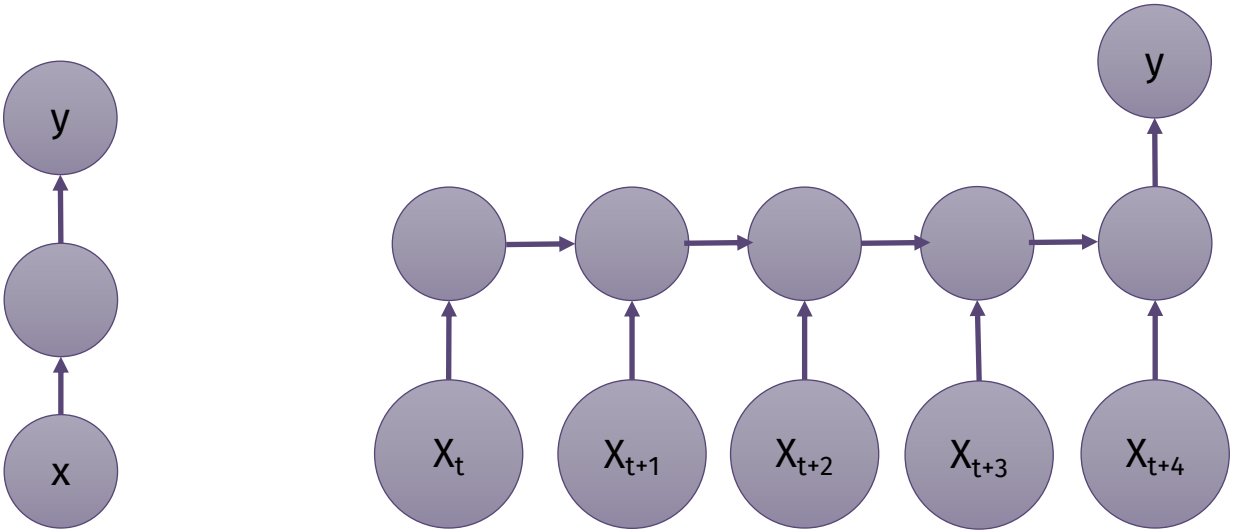
How to approach sequence modelling

- Handling sequences of **different lengths**
- Taking into account **short** and **long term** dependences
- Considering **order** between elements
- **Sharing parameters** across the sequence

UniGe | MaLGa

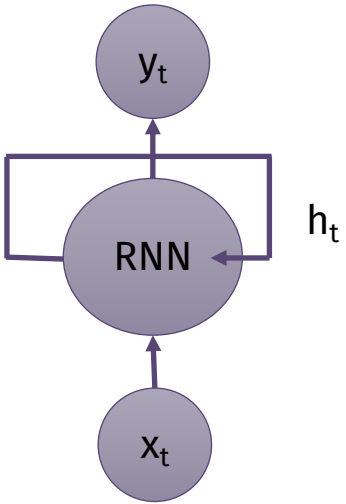
Recurrent Neural Networks

Modelling sequences



Standard one-to-one vanilla network

Recurrent Neural Networks



A recurrence relation is applied at each time step to model the sequence

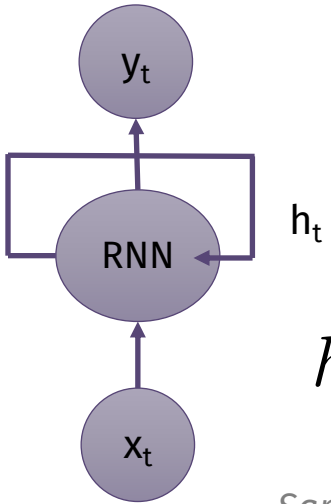
$$h_t = f_W(h_{t-1}, x_t)$$

Same function and weights are used for each time step, so they are a way to share weights 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 combine to determine how they respond to new data
- It is finding correlations between events separated by many moments, and these correlations are called "*long-term dependencies*"

RNNs: Forward propagation



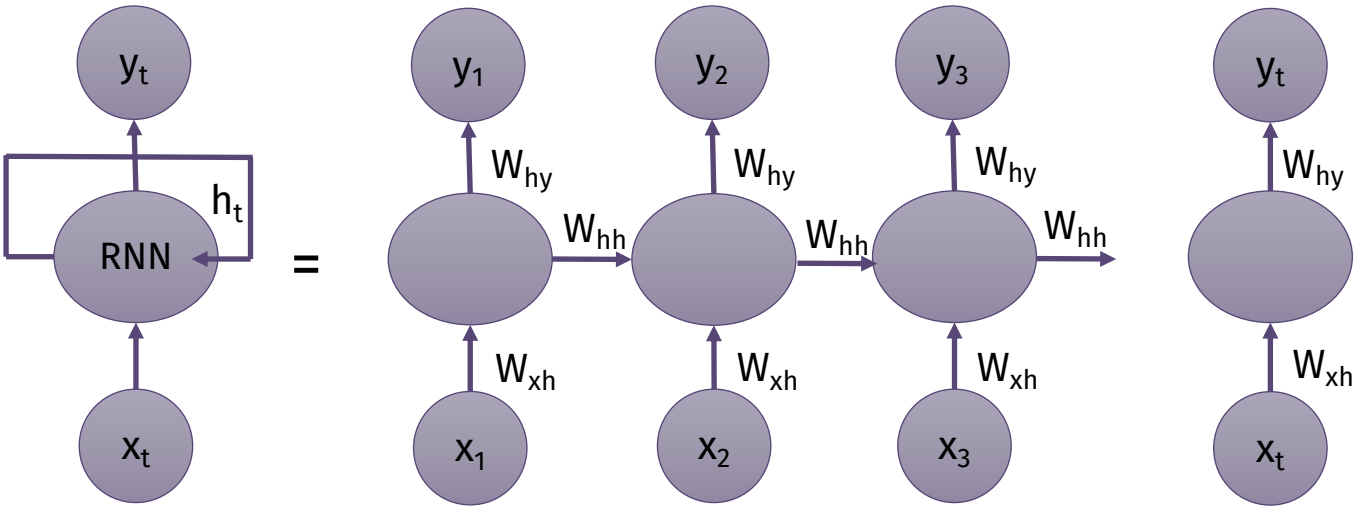
$$y_t = W_{hy} h_t$$

$$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t)$$

Same function and weights are used for each time step, so they are a way to share weights over time

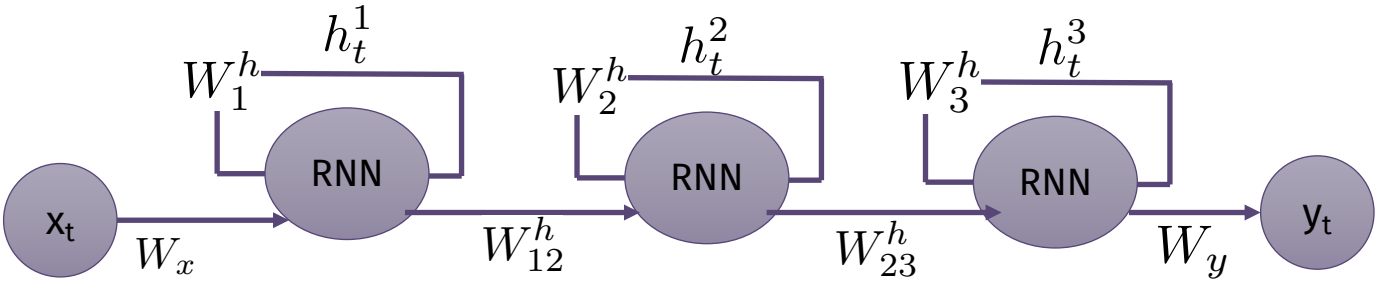
Unrolling RNNs over time

A RNN can be seen as a sequence of multiple, communicating copies of the same network

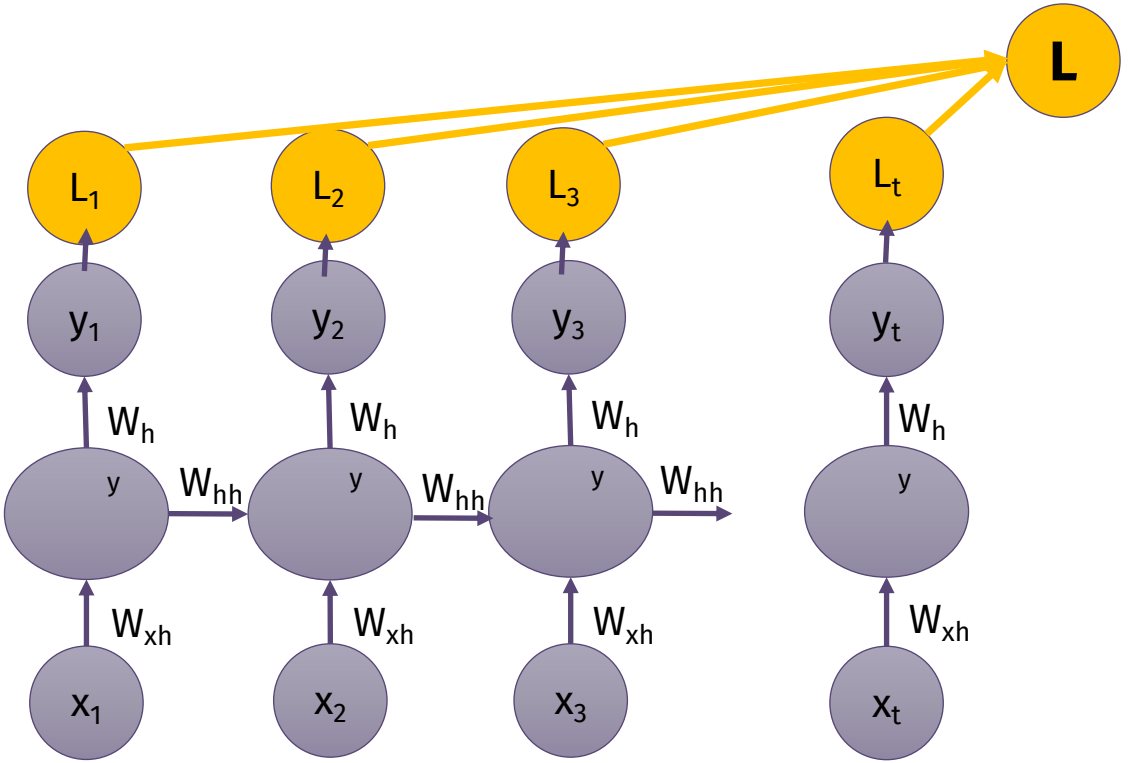


The weight matrices are filters that determine ***how much importance to give to both the present input and the past hidden state***

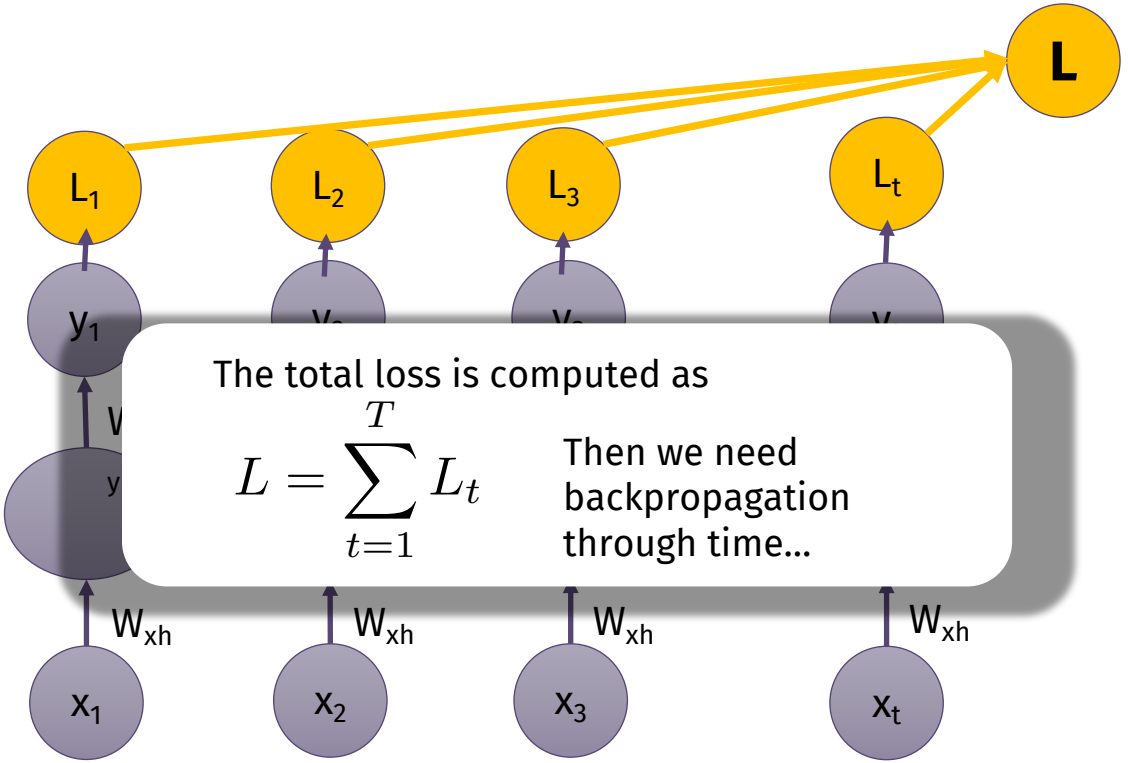
What about having multiple hidden layers?



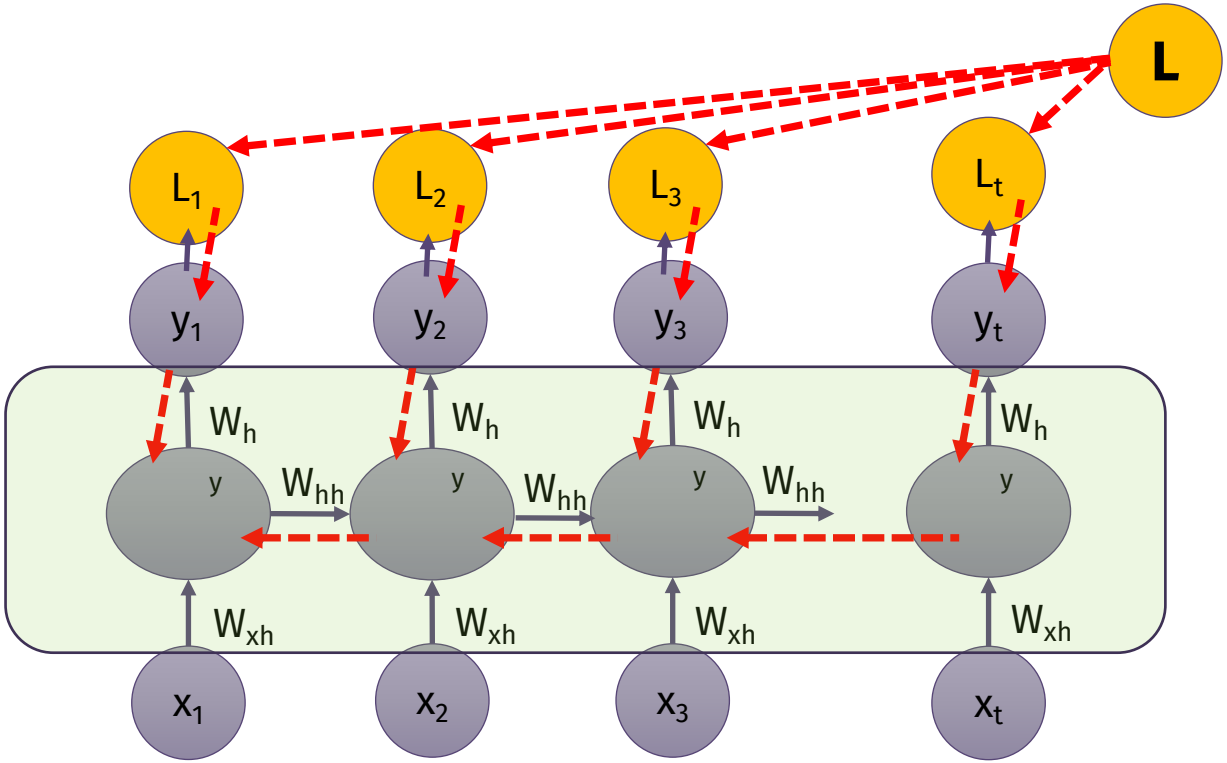
Again on forward propagation



Again on forward propagation



Backward propagation



Loss and backpropagation

$$L = \sum_{t=1}^{T} L_t$$

$$\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \left(\sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \right)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}}$$

Computed by multiplying consecutive time steps

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

$$\Theta_{new} = \Theta_{old} - \alpha \nabla_{\Theta} L(\Theta)$$

- 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

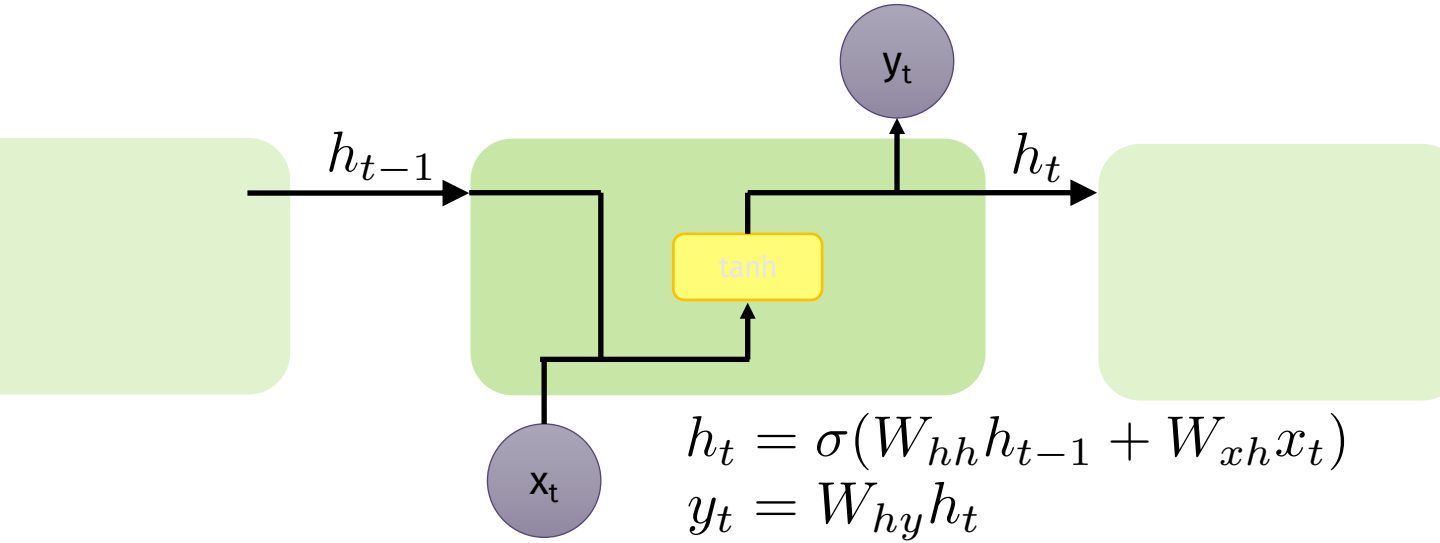
- A possible solution to learn long-term dependences in the data is to use **gated cells**

UniGe | MaLGa

Long-Short Term Memory

RNNs cells

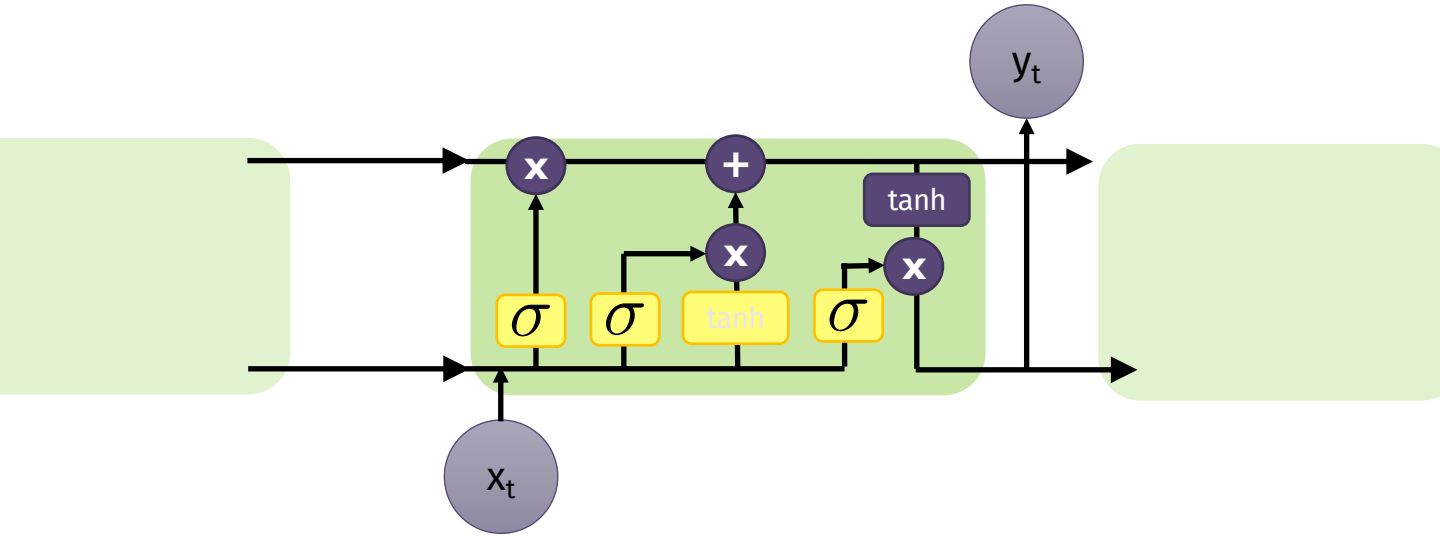
- In standard RNNs, the cells contain a simple computation and their state is constantly re-written



$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$y_t = W_{hy}h_t$$

Gated cells

- Gated cells contain computational blocks that control information flow



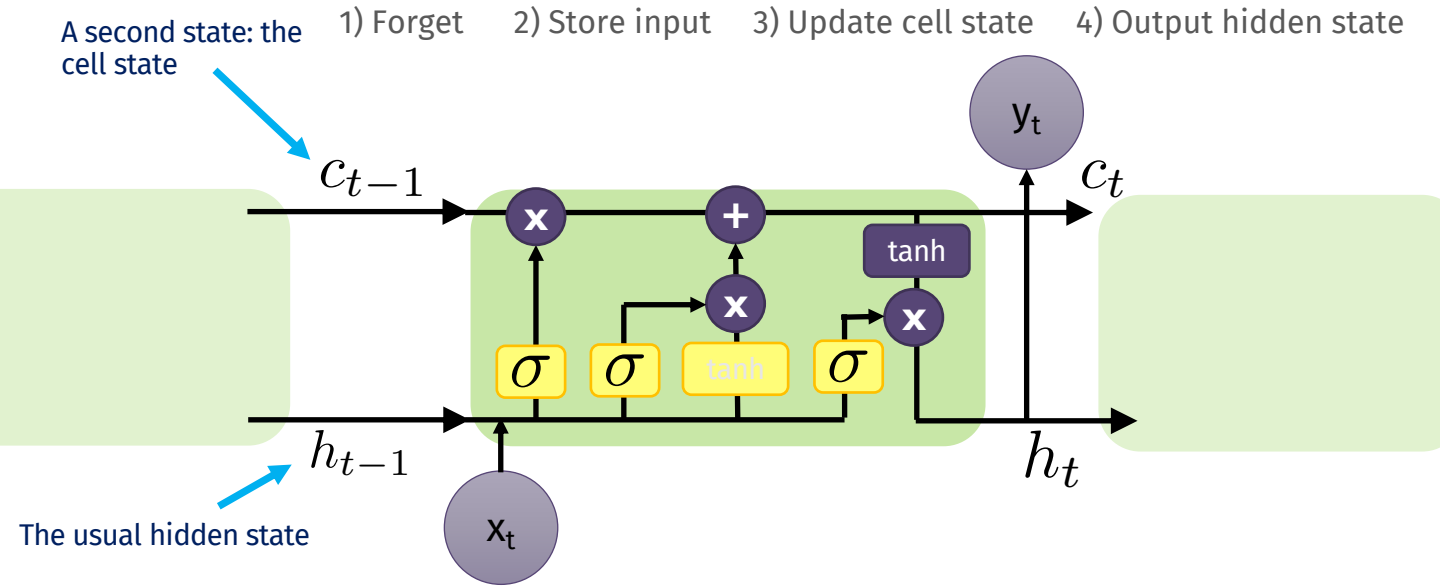
Long-short term memory (LSTM, 1997)

- 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 RNN 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)

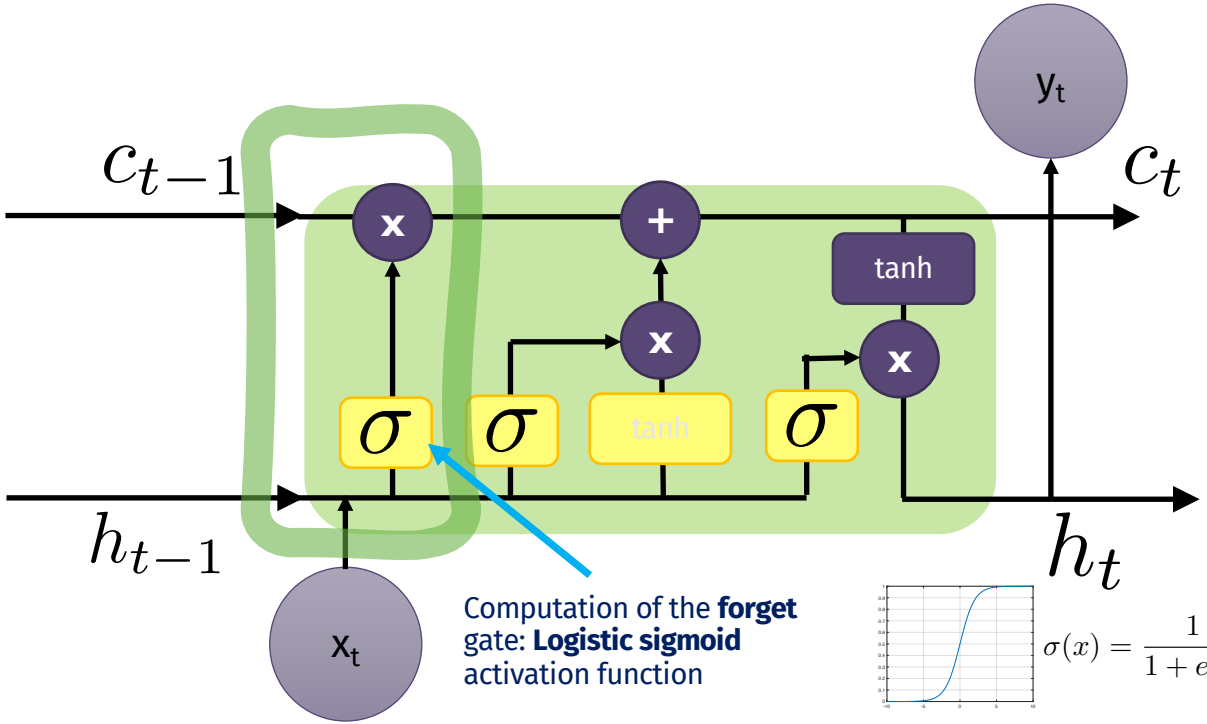
LSTM cell

- 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: how does it work



LSTM cell – 1) Forget

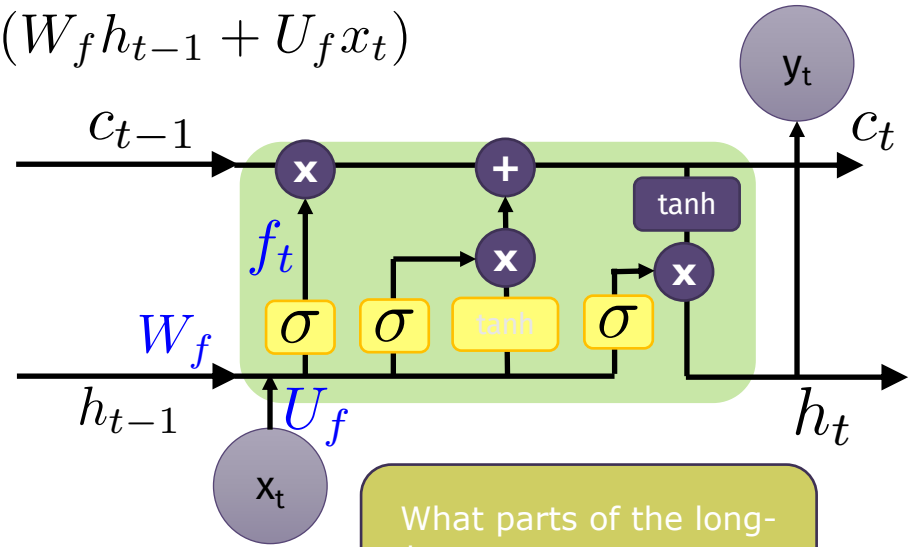


Forget gate

- It decides which information to keep and what to forget from the previous **cell state**

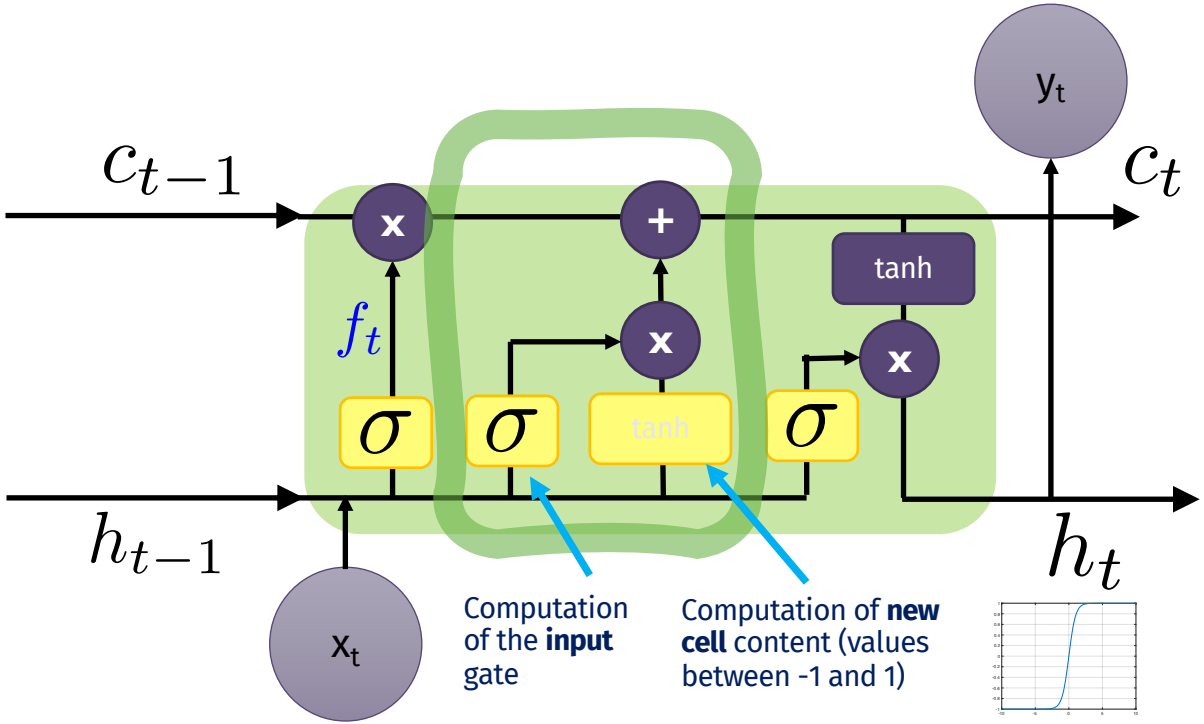
$$f_t = \sigma(W_f h_{t-1} + U_f x_t)$$

Sigmoid gate: all values are between 0 and 1



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

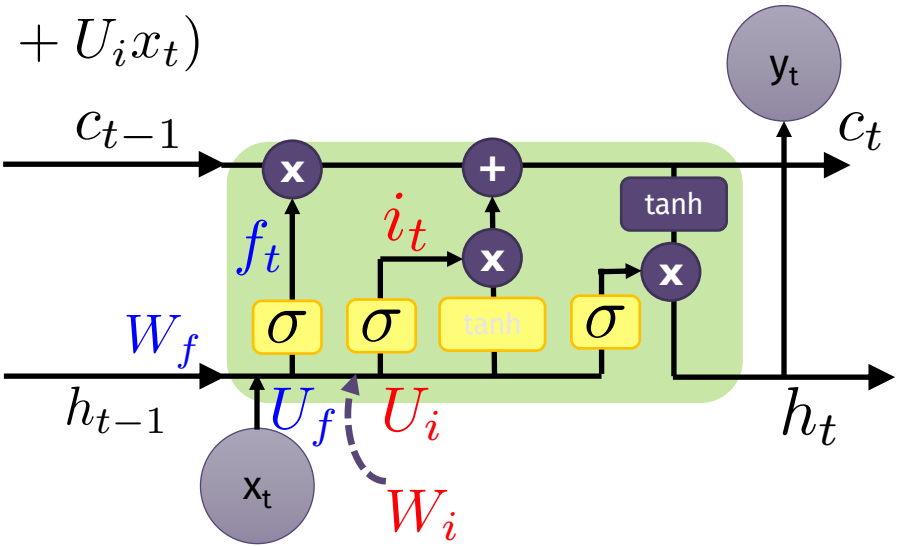
LSTM cell – 2) Store input



Input gate

- It decides what new information (from previous hidden state and new input data) storing in the cell states

$$i_t = \sigma(W_i h_{t-1} + U_i x_t)$$

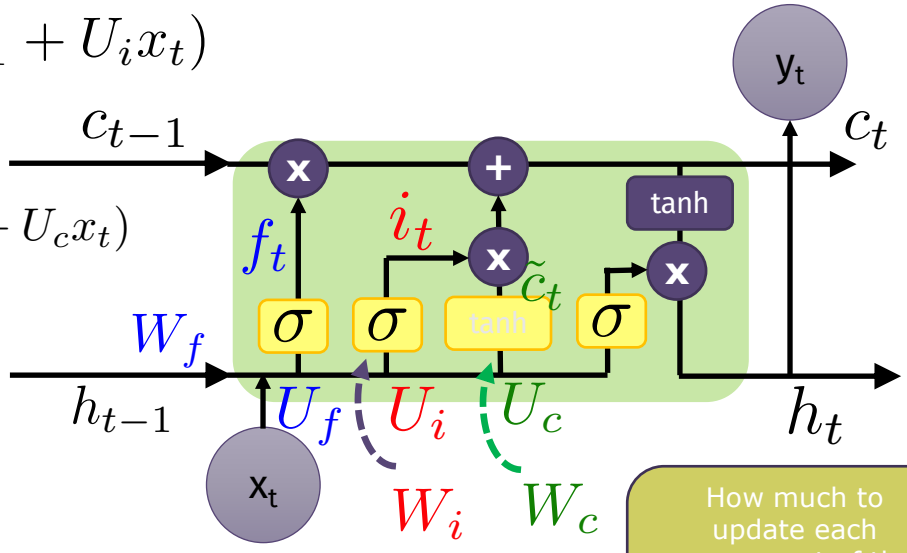


Input gate

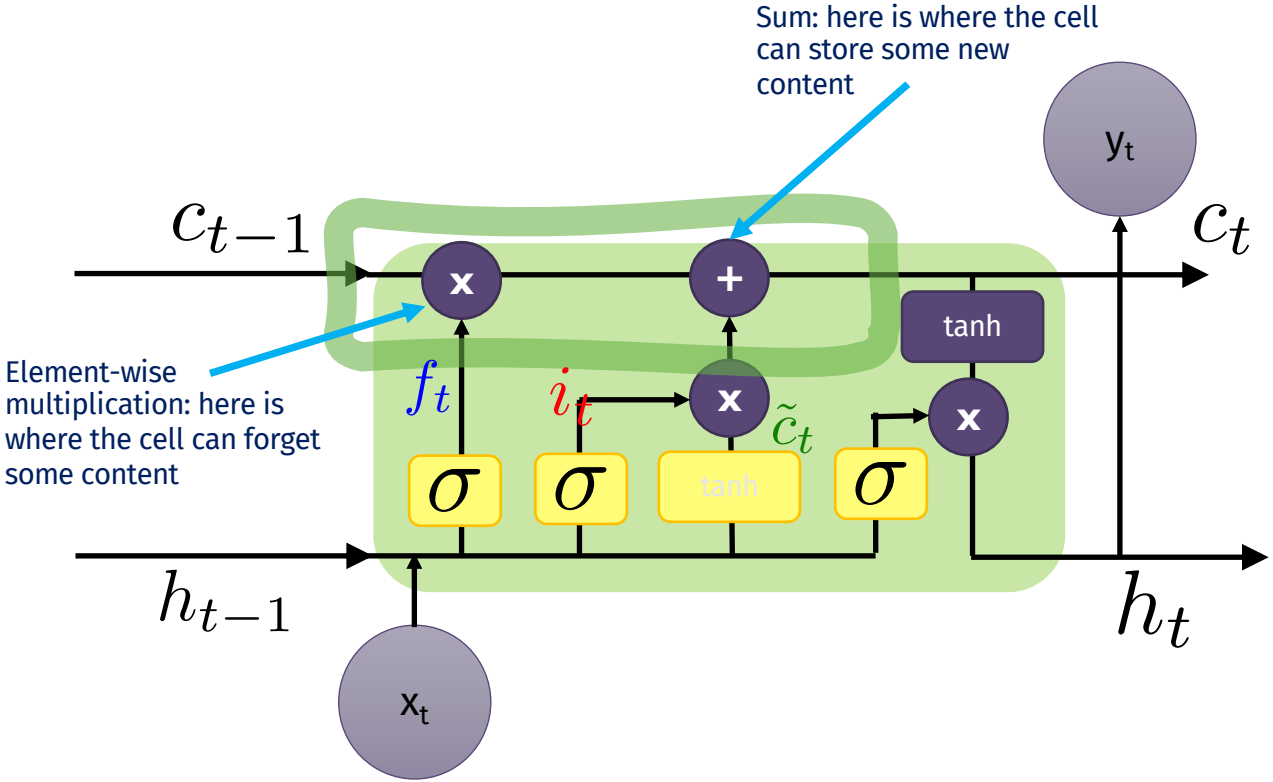
- It decides what new information (from previous hidden state and new input data) storing in the cell states

$$i_t = \sigma(W_i h_{t-1} + U_i x_t)$$

$$\tilde{c}_t = tanh(W_c h_{t-1} + U_c x_t)$$



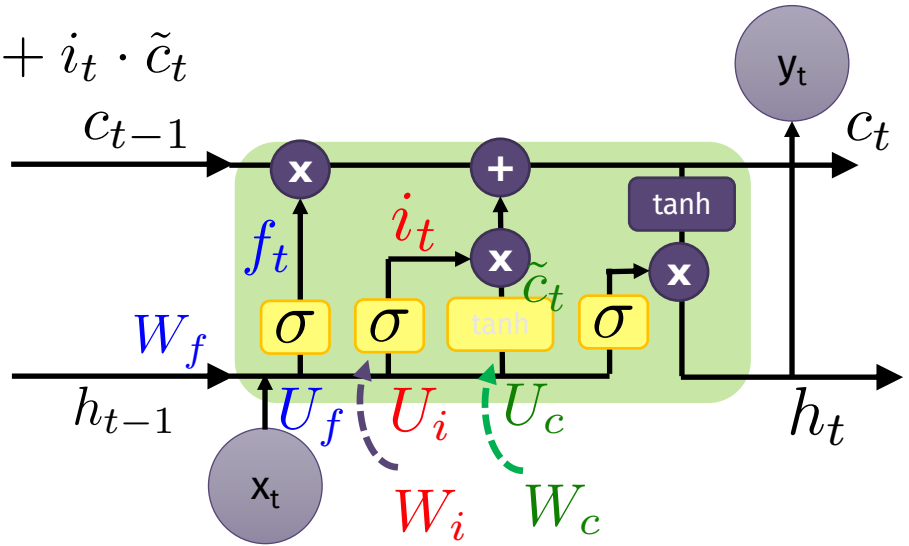
LSTM cell – 3) Update



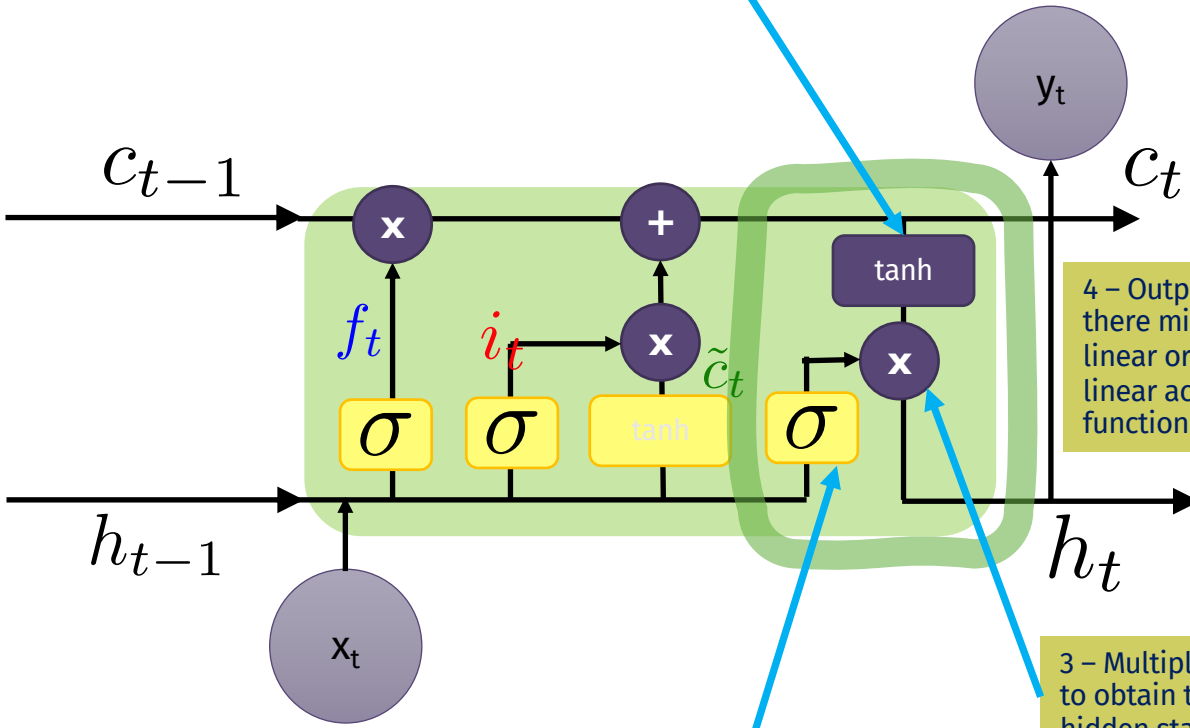
Update

- The cell state values are selectively updated

$$c_t = c_{t-1} \cdot f_t + i_t \cdot \tilde{c}_t$$



LSTM cell – 4) Output



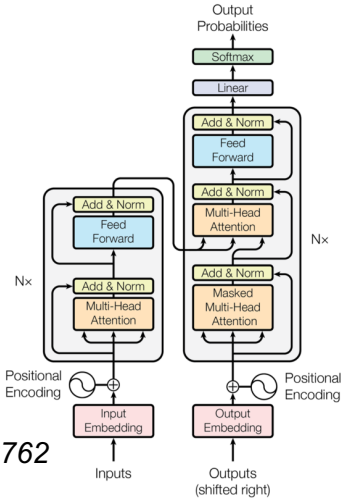
Variations on the theme and recent advances

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 forge

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

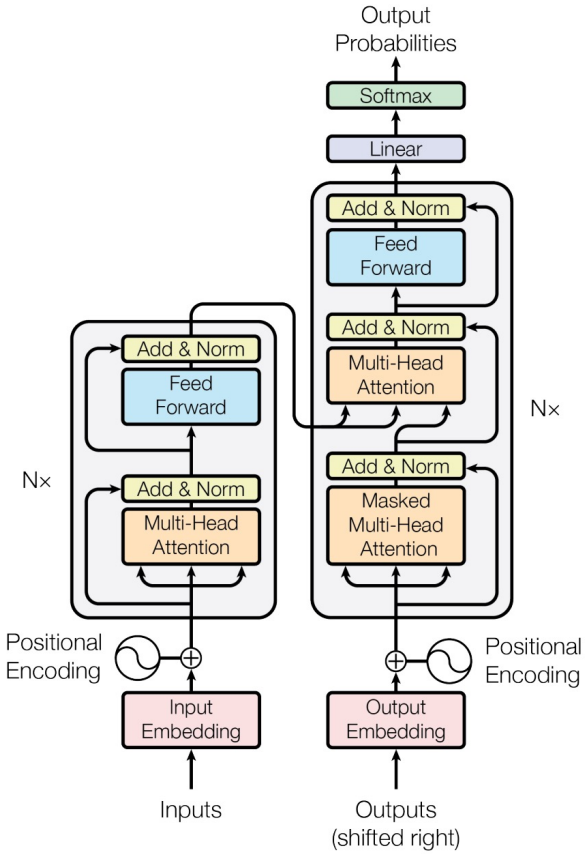


Figure 1: The Transformer - model architecture.

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

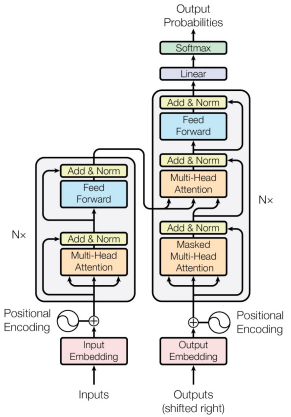


Figure 1: The Transformer - model architecture.

$(x_1, \ldots, x_n)$ 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

Transformer: main structure

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

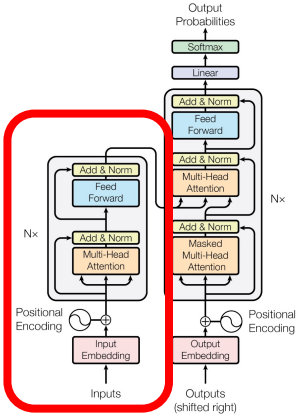
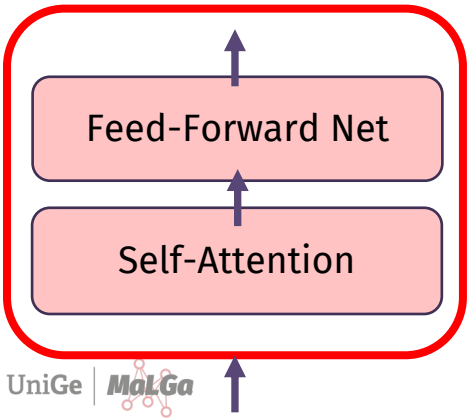


Figure 1: The Transformer - model architecture.



Transformer: main structure

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

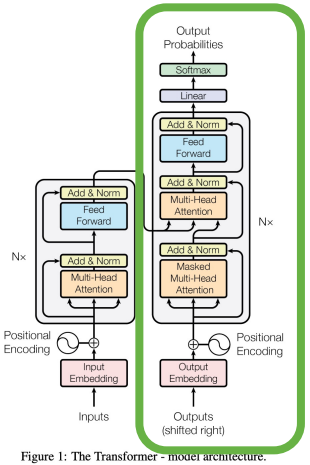
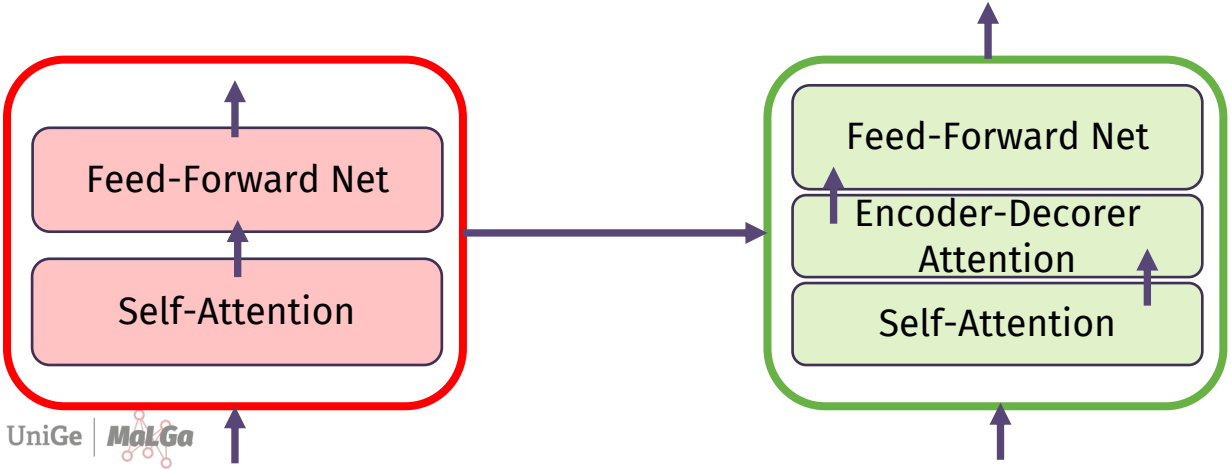


Figure 1: The Transformer - model architecture.

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

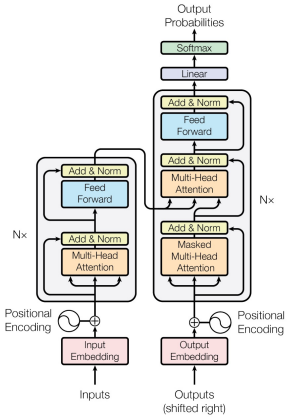


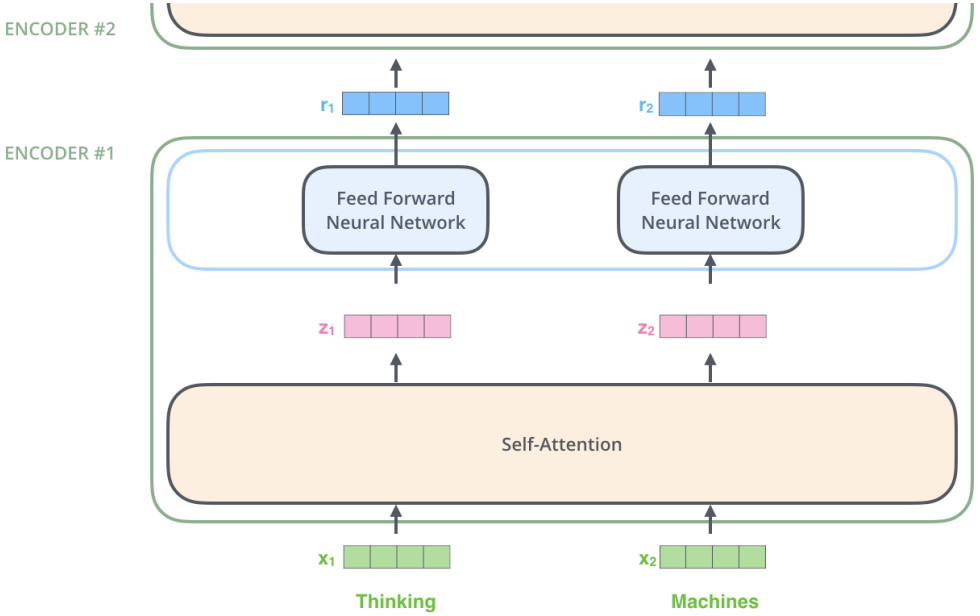
Figure 1: The Transformer - model architecture.

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

<u>RESIDUAL CONNECTION</u>: the output of each layer is

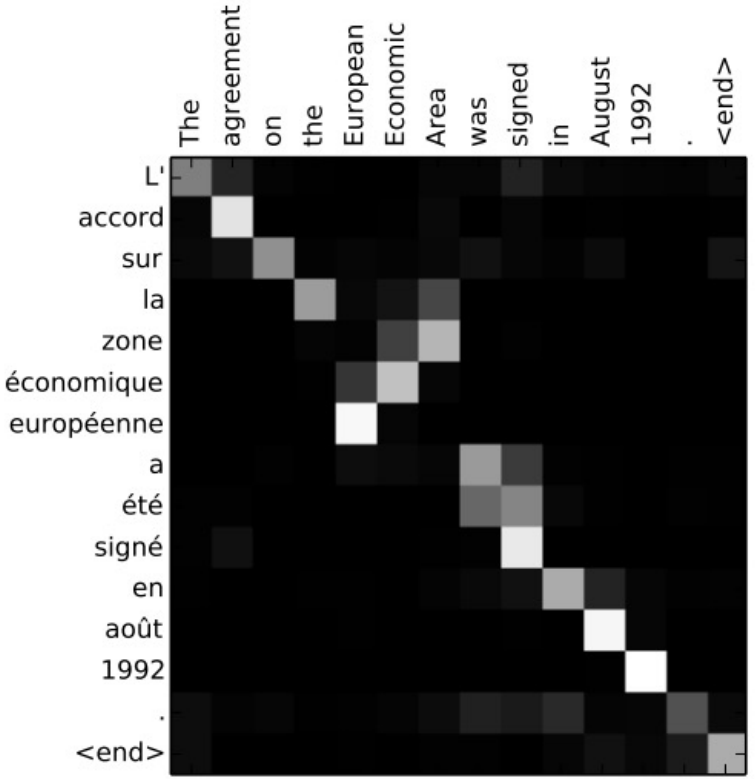
$$\sigma(x + f(x))$$

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



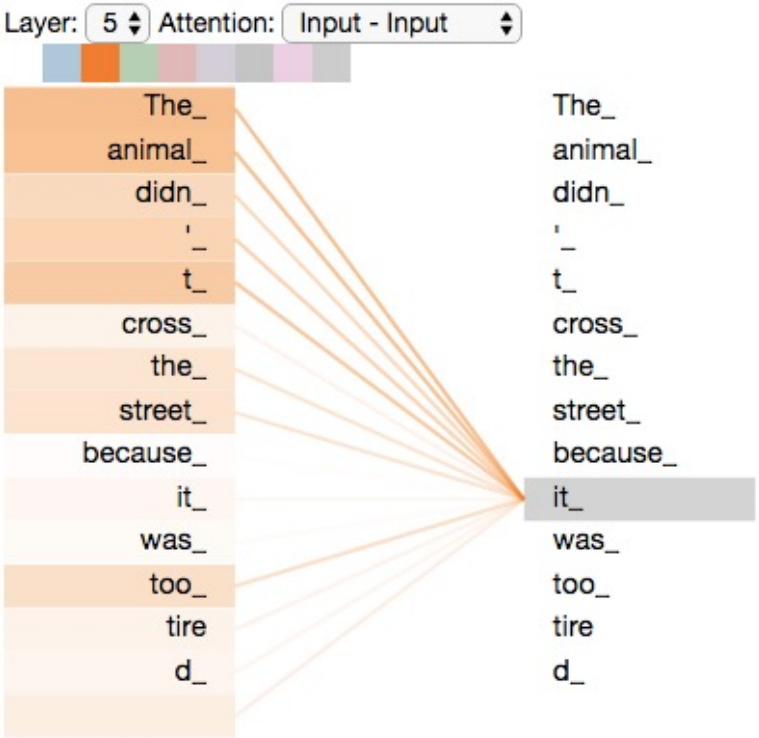
From http://jalammar.github.io/illustrated-transformer/

Why attention?



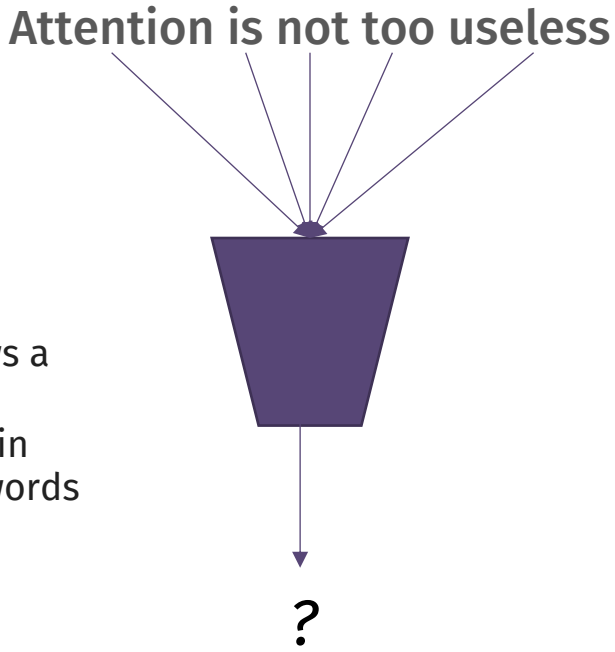
From «Neural Machine Translation by Jointly Learning to Align and Translate" (2015)

Why self-attention?



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

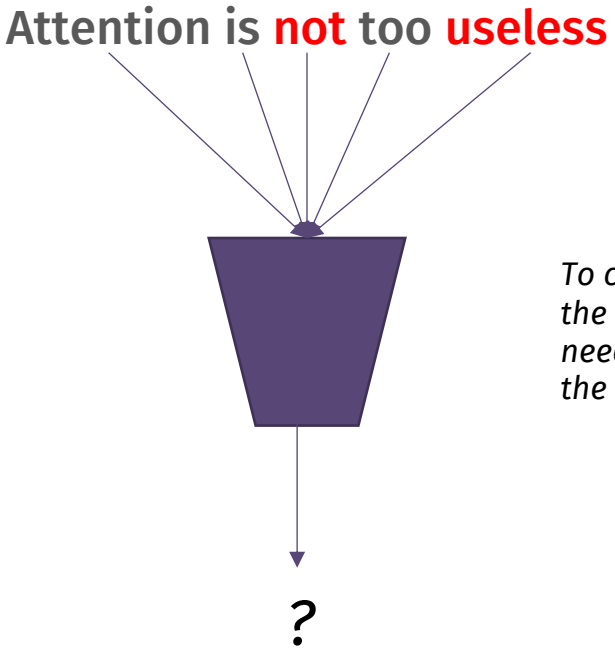
What is self-attention?
An example on sentiment classification



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

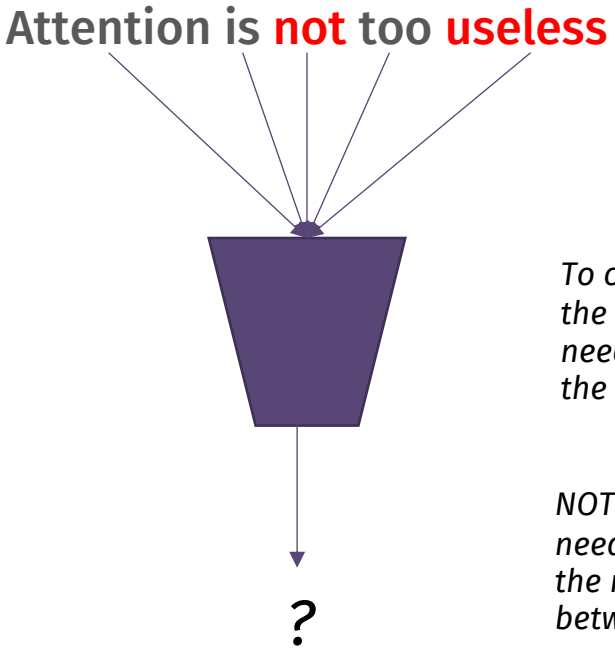
What is self-attention?
An example on sentiment classification



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?
An example on sentiment classification

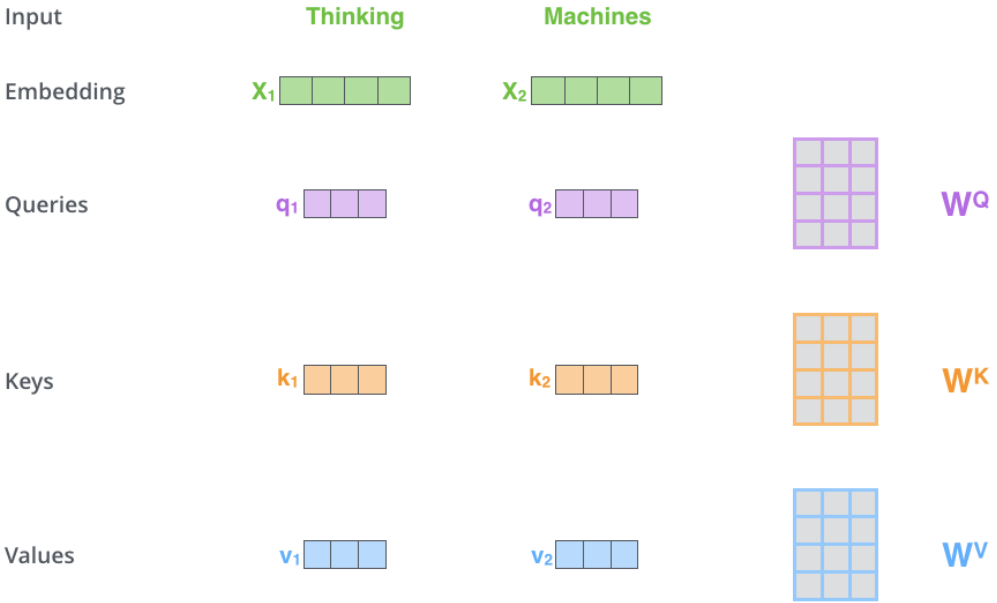


To correctly classify the sentence we need to consider all the words in it

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

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

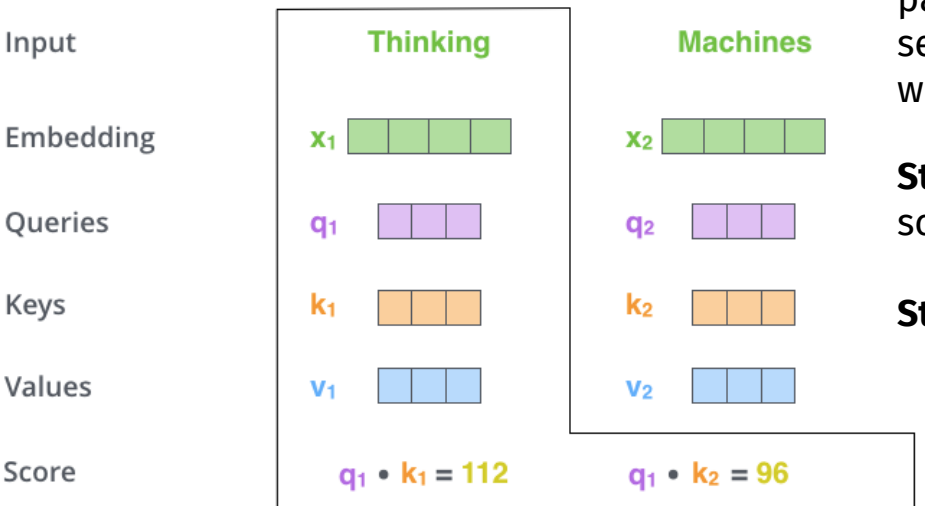
Let's proceed step by step



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

Step 1 - For each word, we create a Query vector, a Key vector, and a Value vector

Let's proceed step by step



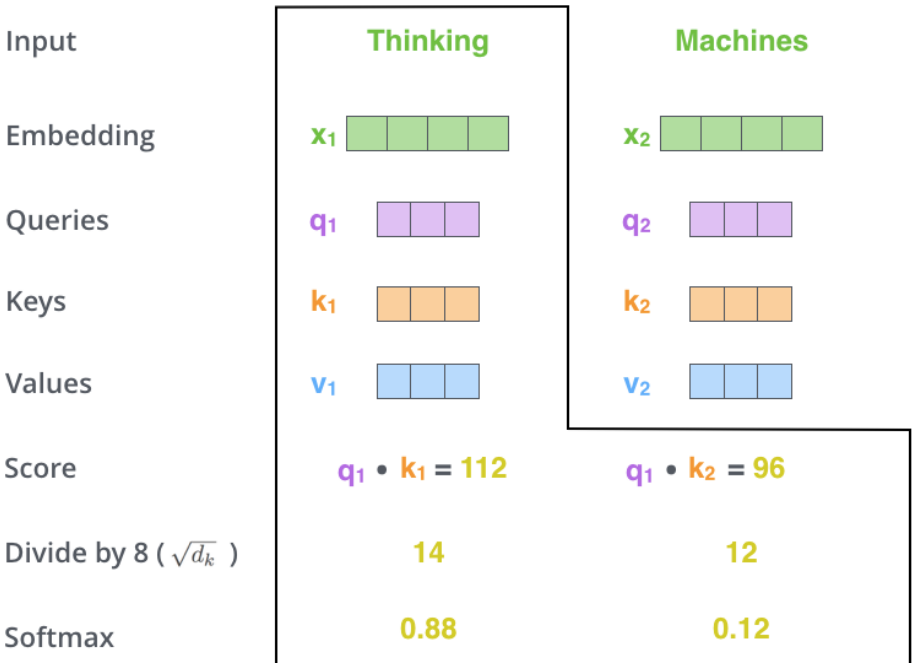
Step 2 - For each embedding, we compute the score that determines the importance of other parts of the input sentence as we encode a word at a certain position.

Step 3 – Normalize the scores

Step 4 – Apply softmax

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

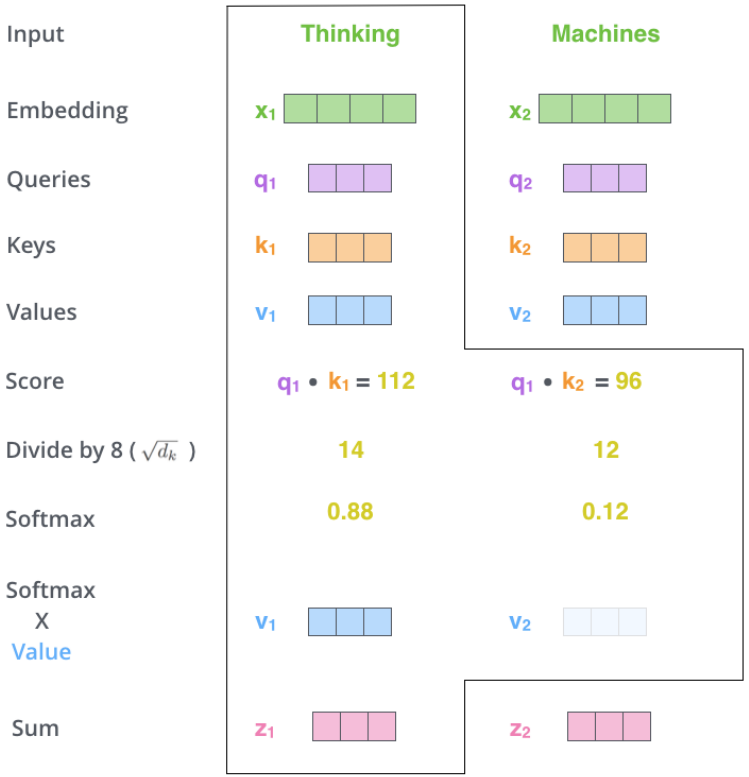
Let's proceed step by step



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

The softmax score determines the "level of expression" of the word at this position.

Let's proceed step by step

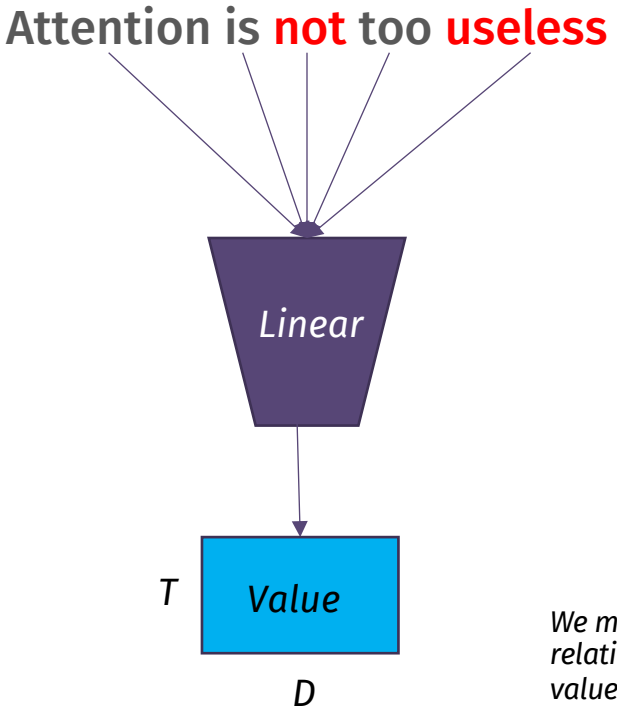


Step 5 – Multiply each value vector by the softmax score. This keeps intact the values of the word(s) we want to focus on, and attenuate the contribution of irrelevant words

Step 6 - Sup the weighted value vectors. This is the output of the self-attention layer at this position (for the first word).

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

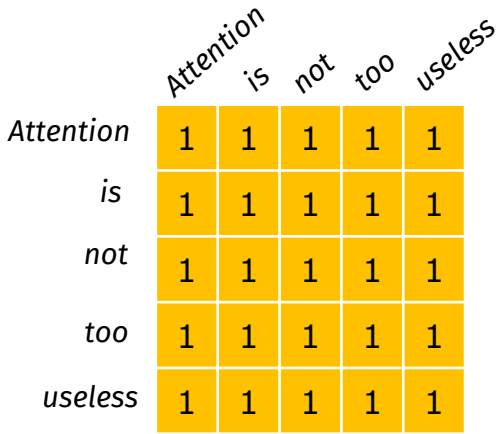
What is self-attention?
An example on sentiment classification

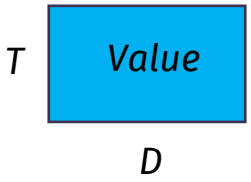


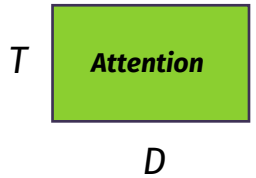
We may encode relationships between values by summation

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

Combining values/words





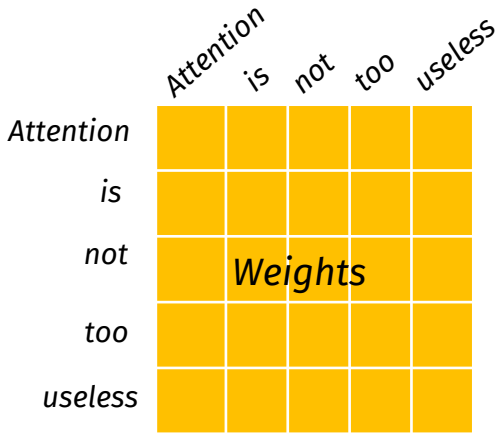


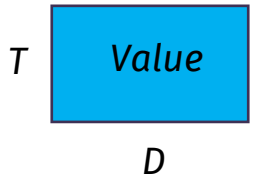
Here we would assume the same importance for all relationships… but we would like that, for instance, the relation between «not» and «useless» is more important than the one between «is» and «too»

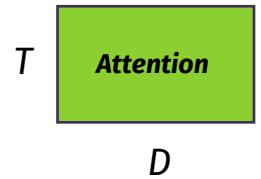
Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

Learning the attention weights

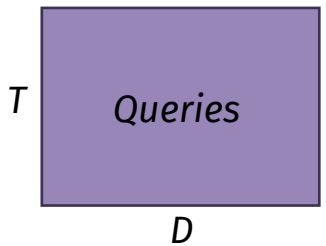
Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

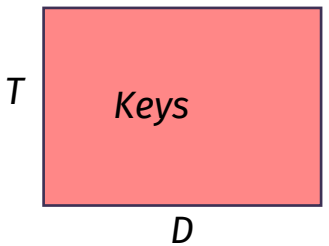






$$W = Q \cdot K^T$$

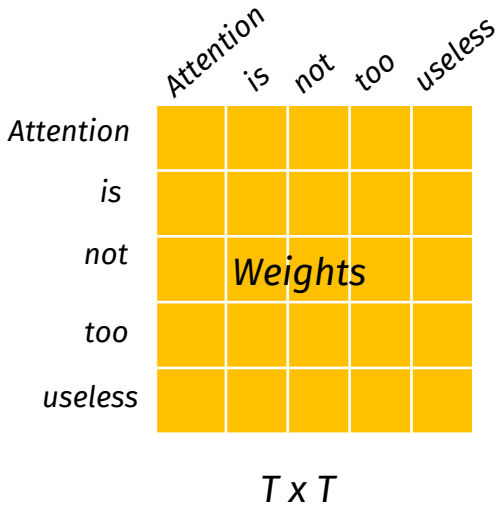


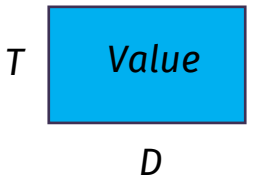


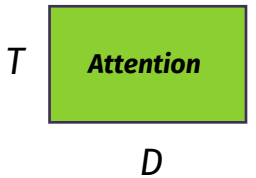
Learning the attention weights

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

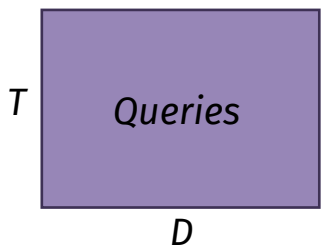
***Intuition**: we want the W matrix to weight the relationship between word$_i$ as a context for word$_j$. We employ other two linear nets*

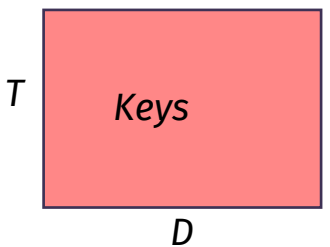




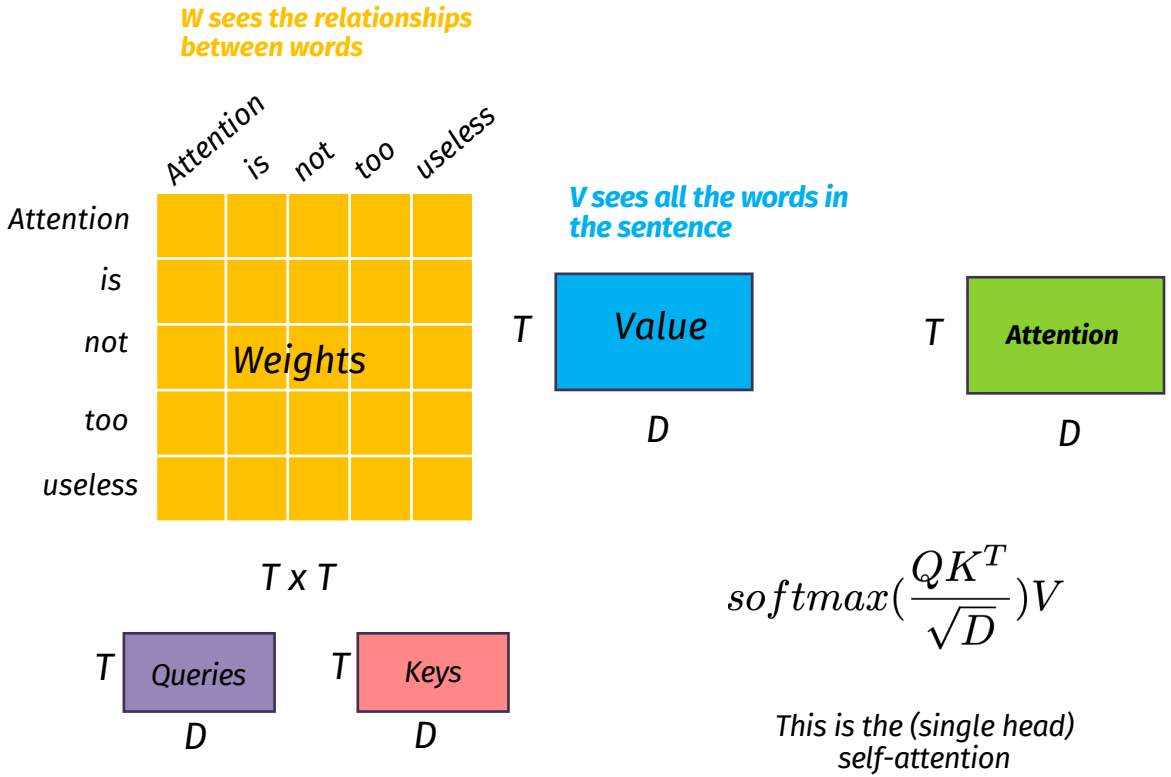


$$W = Q \cdot K^T \rightarrow \frac{Q \cdot K^T}{\sqrt{D}}$$



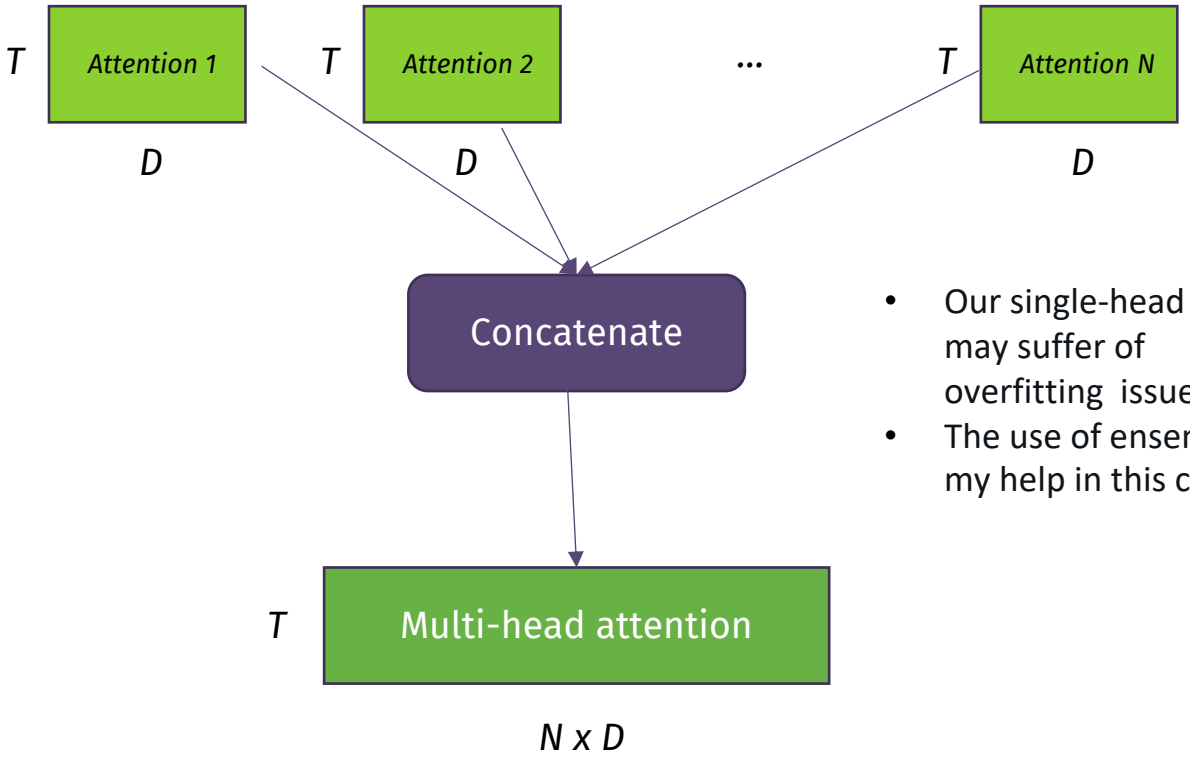


Single-head attention



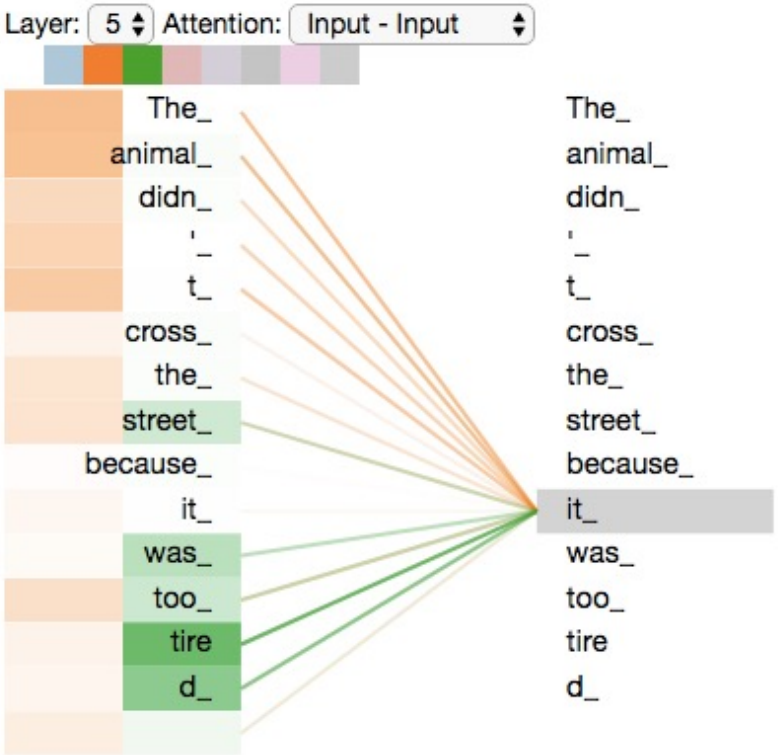
Inspired by https://twitter.com/MishaLaskin/status/1479246928454037508

Multi-head attention



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

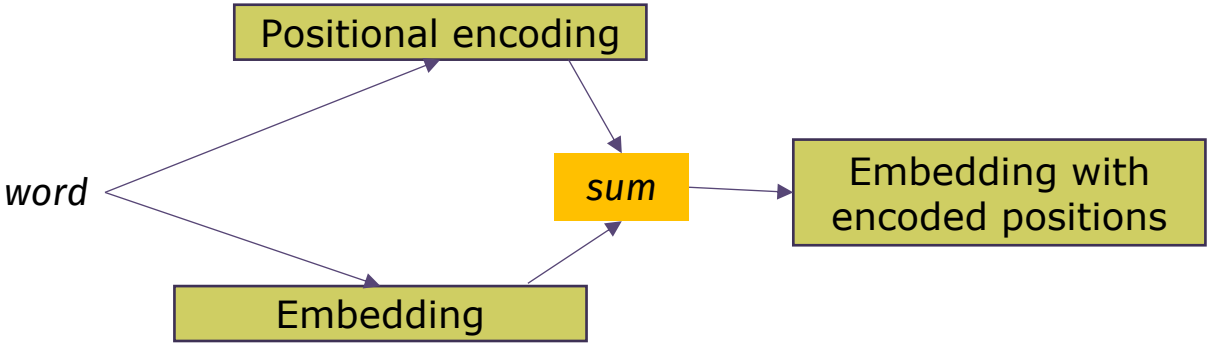
Back to the example



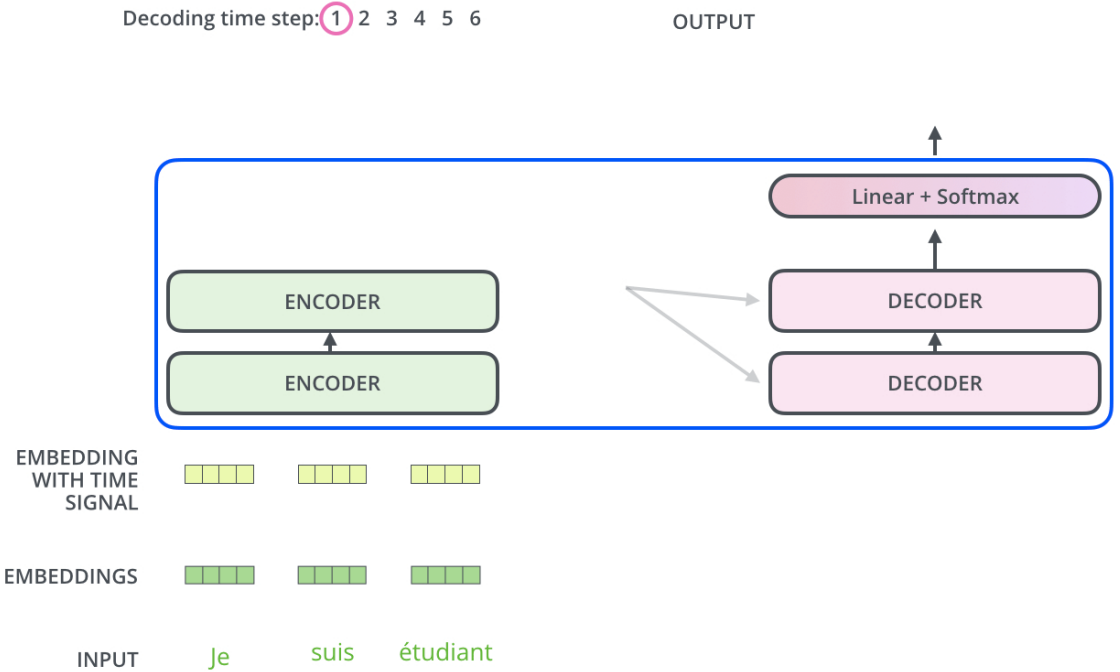
http://jalammar.github.io/illustrated-transformer/

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

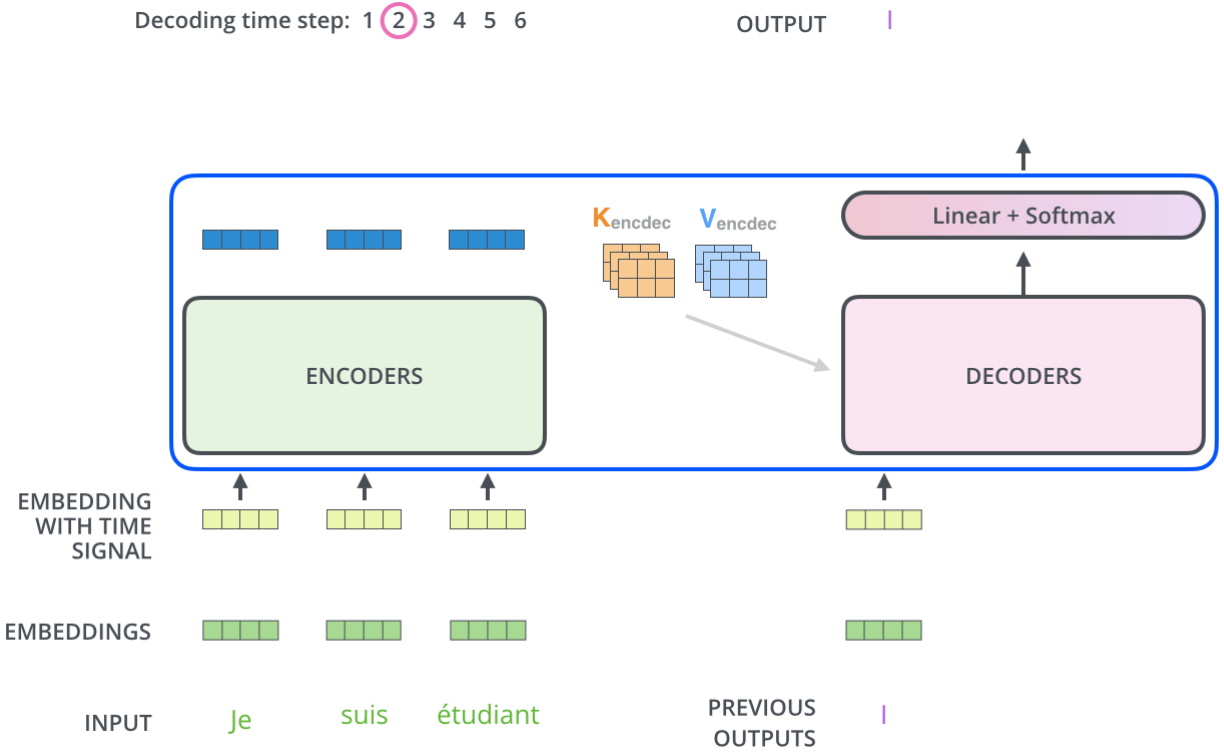


The decoder side



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 to earlier positions in the output sequence (it can not “see” the future)

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

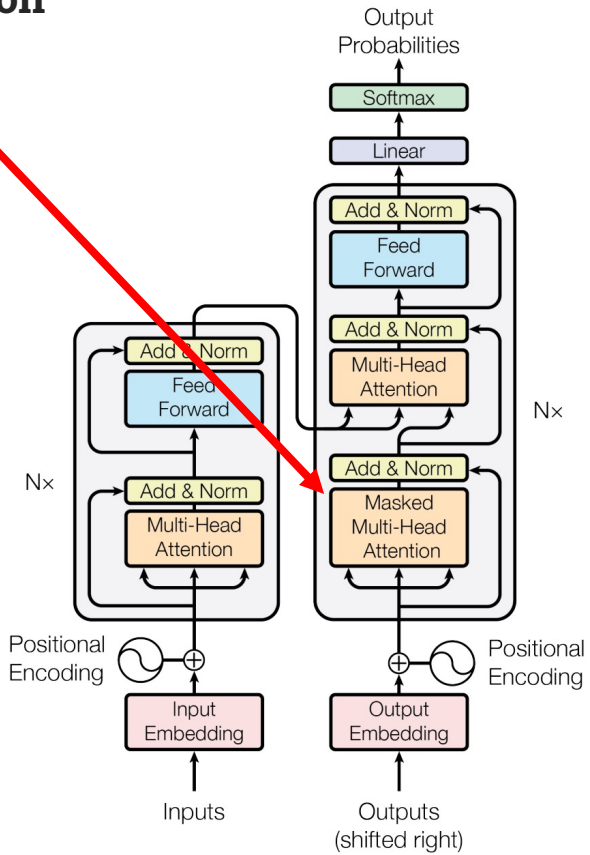


Figure 1: The Transformer - model architecture.

UniGe