Apprentissage Automatique

Recurrent Neural Networks

Thibaud Leteno (thibaud.leteno@univ-st-etienne.fr)

Based on the course of Barbara Martin et Ava Amini

Mars 2025

télécom saint-étienne
école d'ingénieurs nouvelles technologies

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

Introduction to RNN

- Idea behind RNN

- The core of RNN

- The Vanish Gradient Problem

- Long-short term memory

- Bi-directional RNN

Introduction to RNN

Classification and Regression

Artificial Neural Networks

Computer Vision

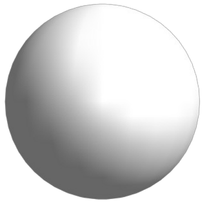
Convolutional Neural Networks

Time series and NLP

Recurrent Neural Networks

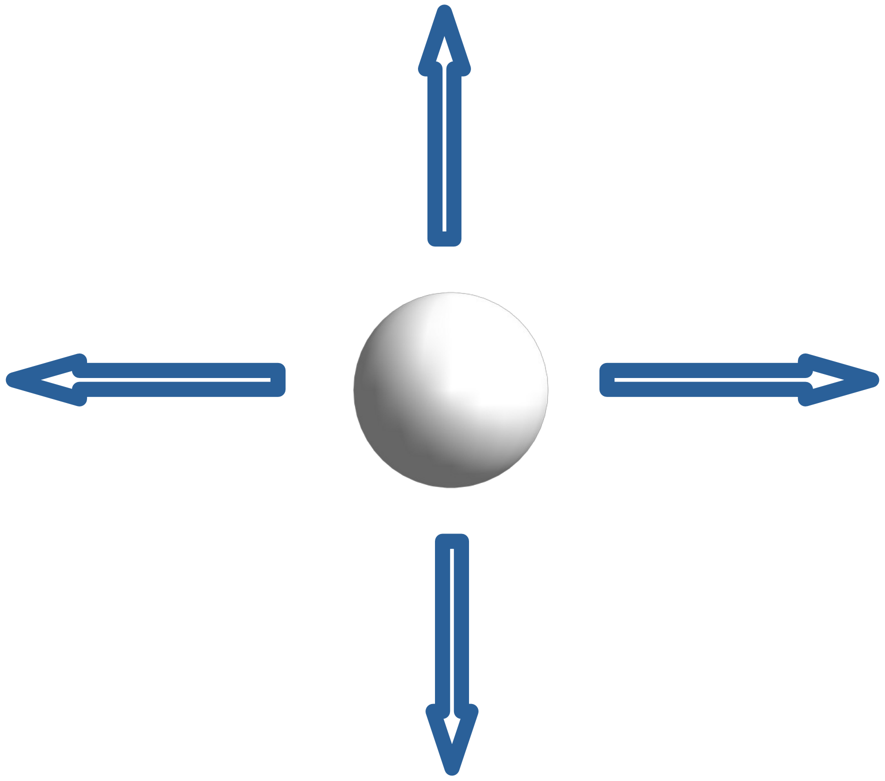
Notion of sequence

Next position of the ball ?



Notion of sequence

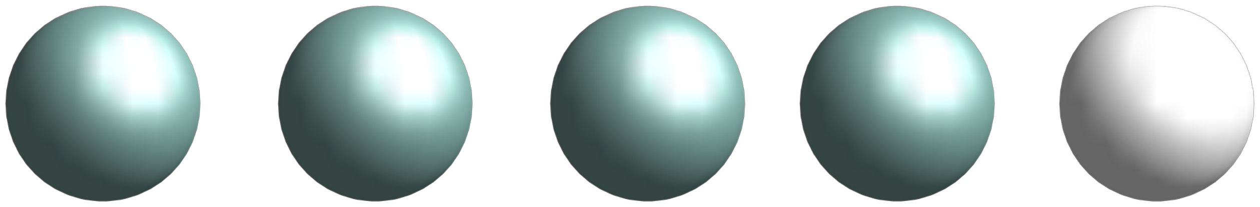
Next position of the ball ?



No prior information. Could be any position.

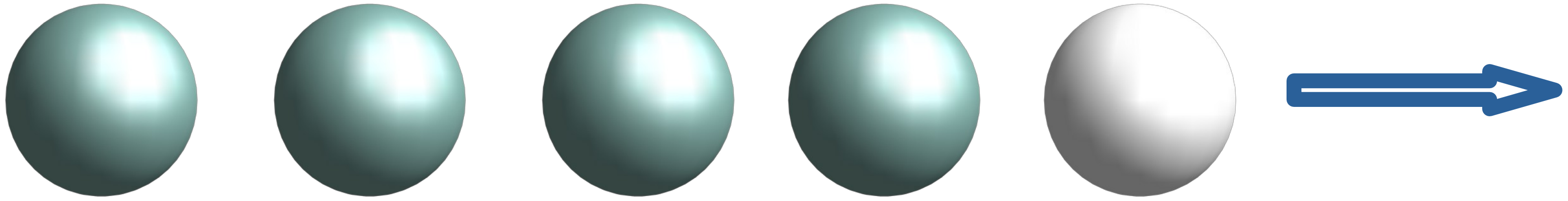
Notion of sequence

Next position of the ball ?



Notion of sequence

Next position of the ball ?



With prior information, we can guess what position is most likely to be next. Our prediction is guided.

Notion of sequence

A sequence can be :

- Audio
- Text (sequence of characters or words)
- Medical Signal (ECG)
- Financial markets
- Biological sequences encoded in DNA
- Patterns in the climate

What questions when dealing with sequences ?

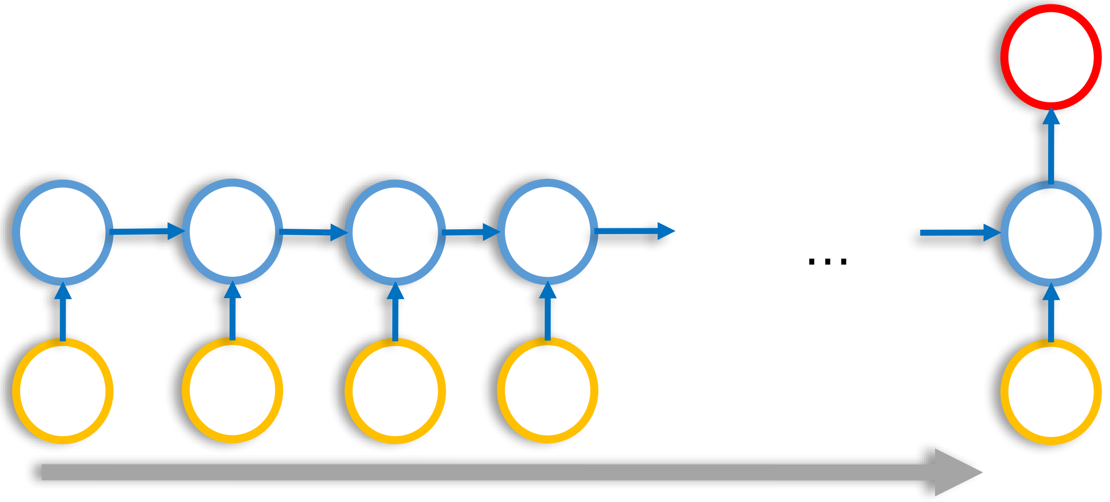
So far with FFN, **One-to-one** configuration (classification, regression)

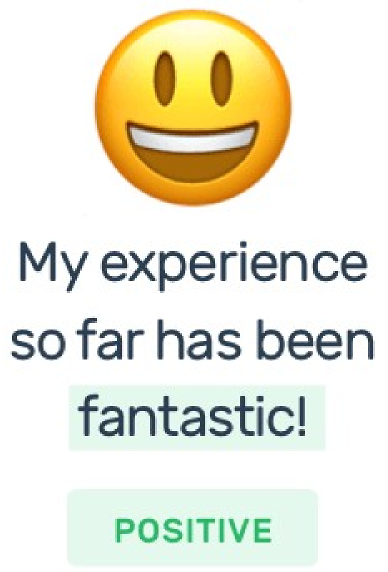
Some notations :

x, the input.
y, the associated true label.
ŷ, the predicted label.

What questions when dealing with sequences ?

Many-to-one

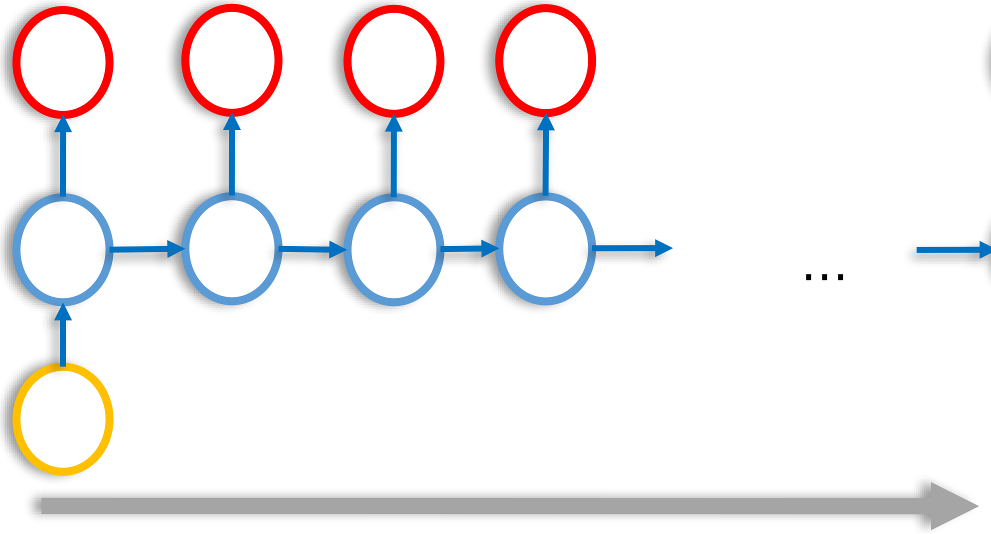




Example : Sentiment analysis

What questions when dealing with sequences ?

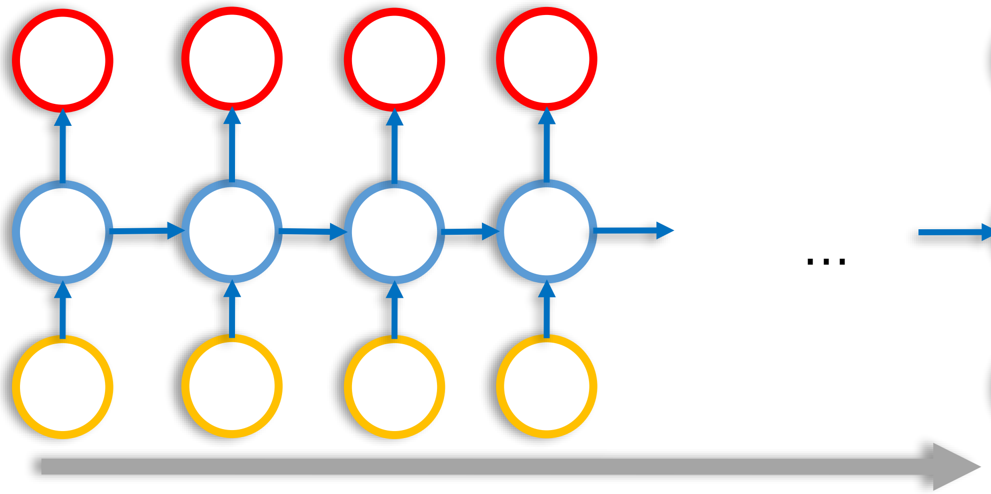
One-to-many

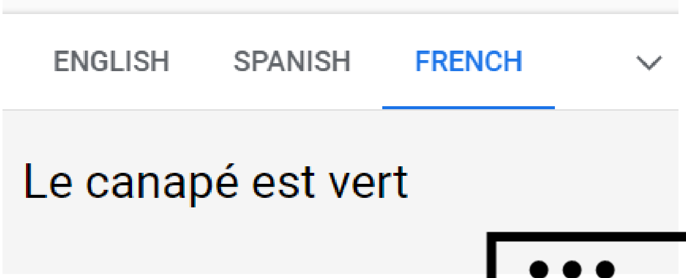


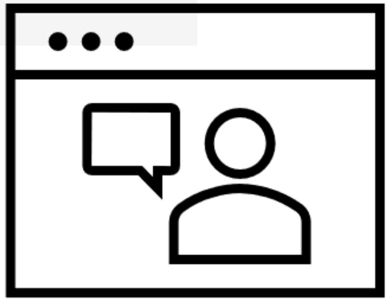
Example : Image captioning

What questions when dealing with sequences ?

Many-to-many





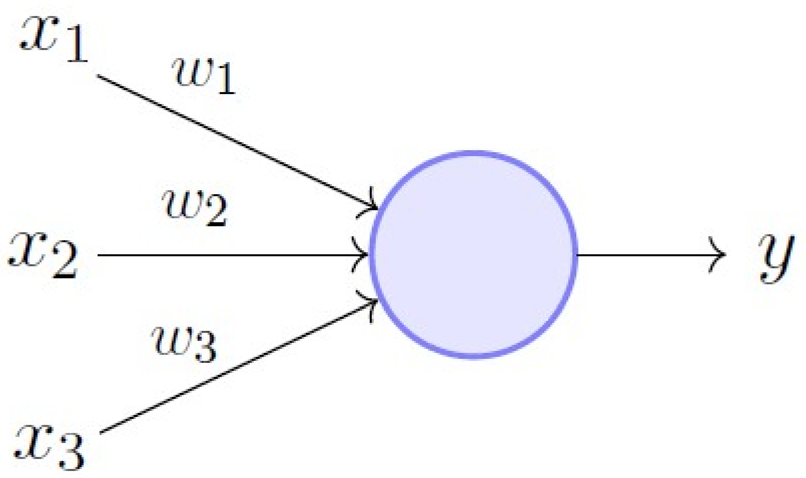


Example : Translations, Chatbot

What solutions ?

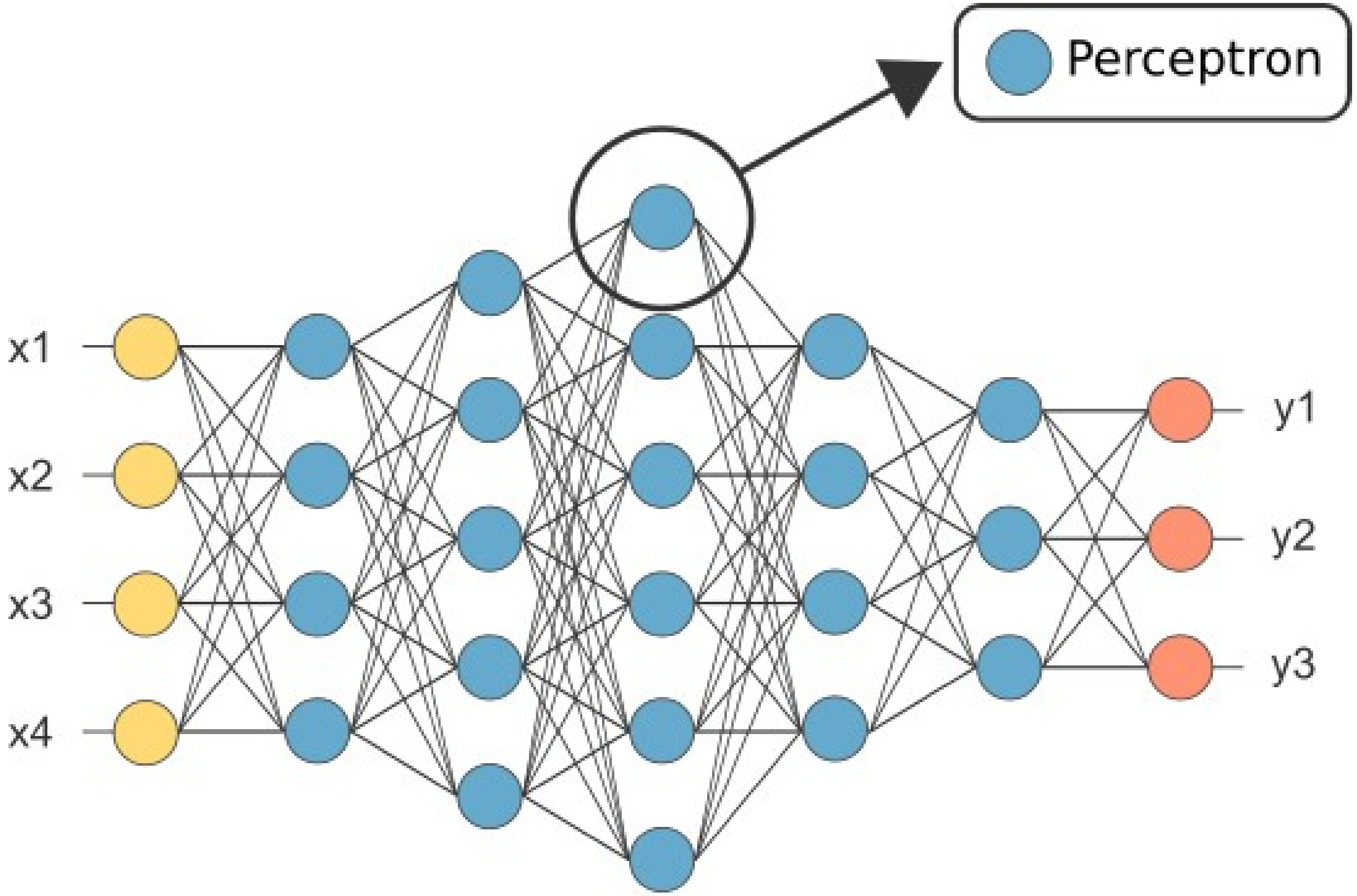
What Neural Networks can we build to tackle this type of problems ?

The perceptron, reminder



Perceptron Model (Minsky-Papert in 1969)

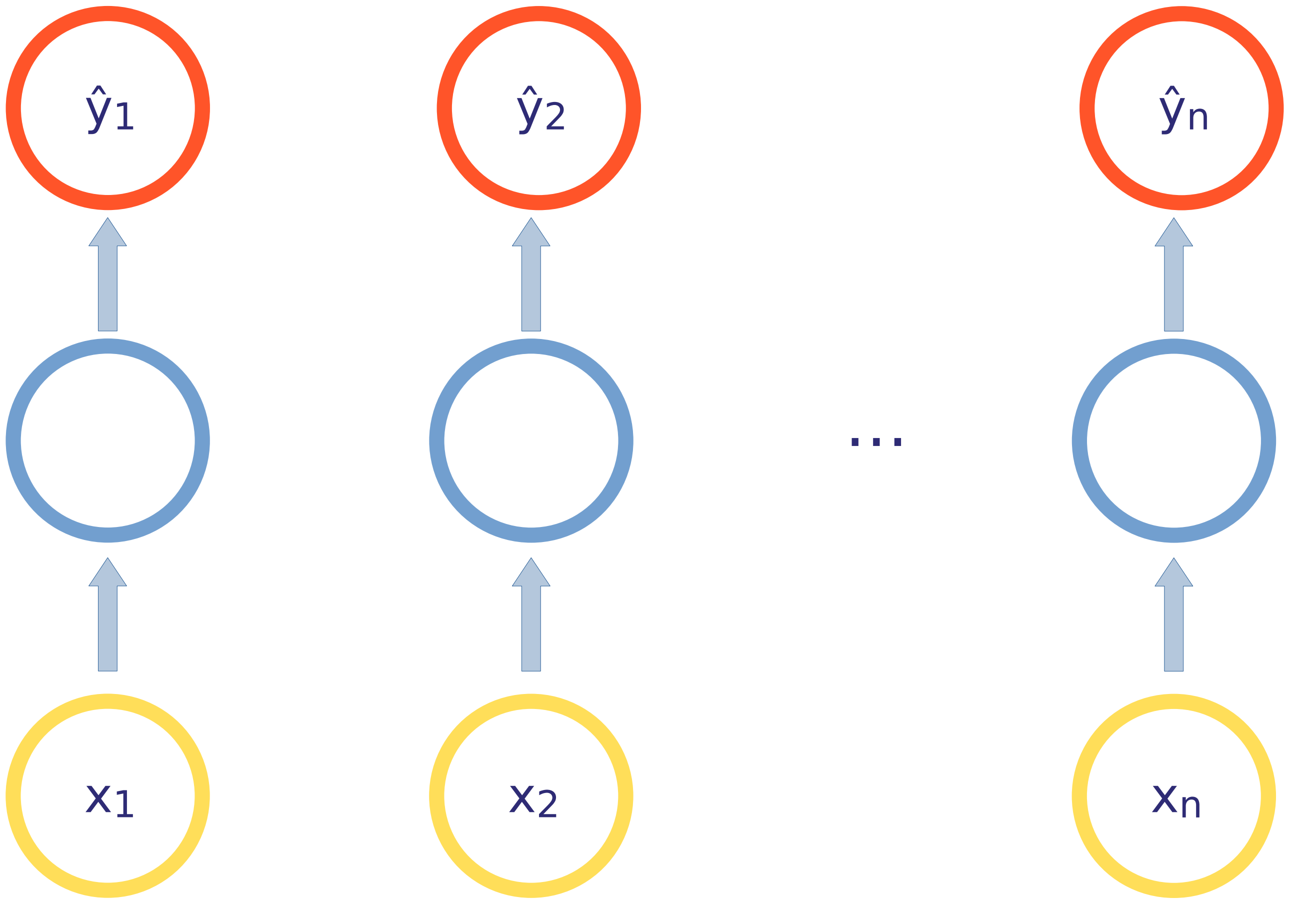
The perceptron, reminder



No notions of sequence or temporal processing...

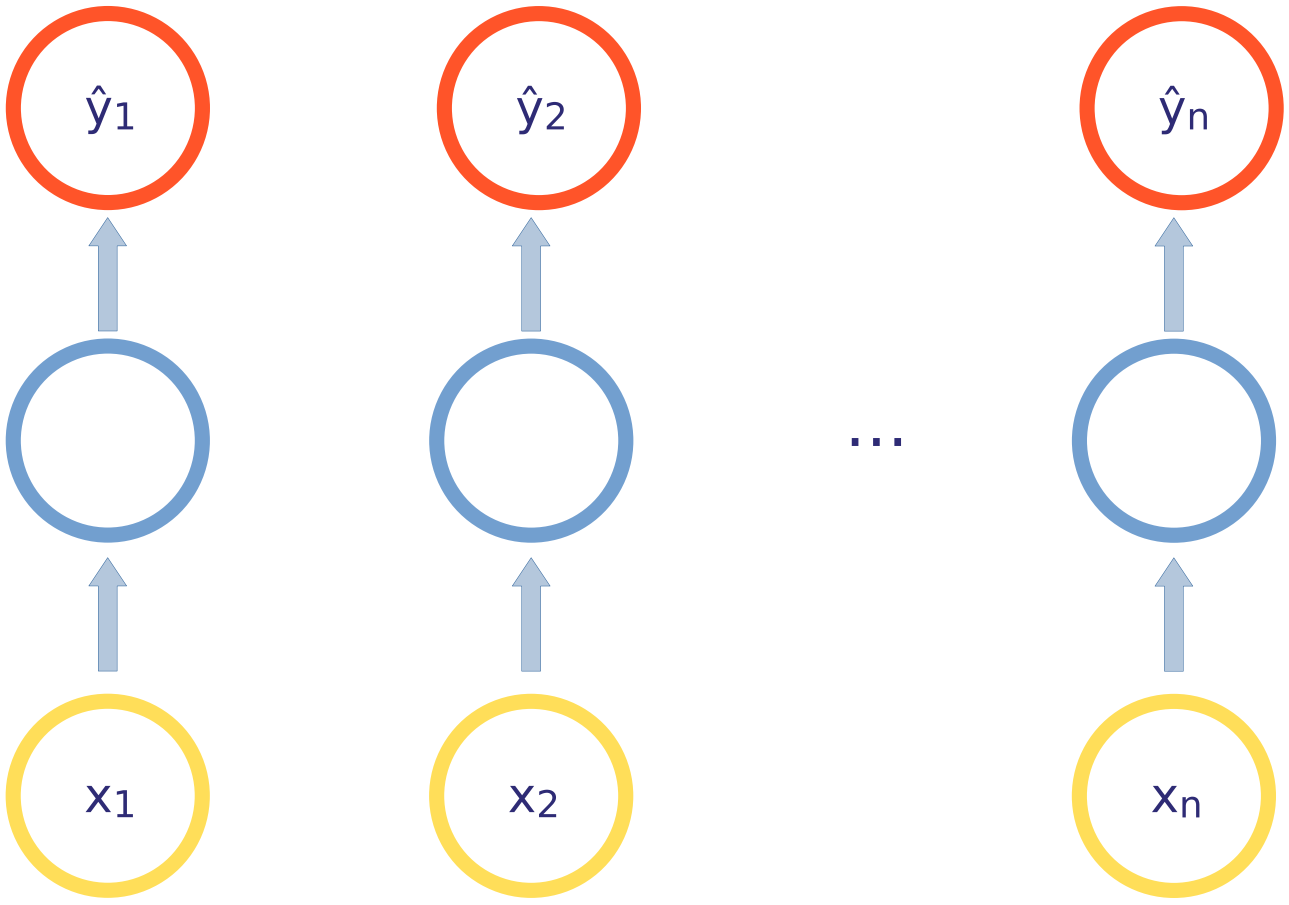
Handling individual time steps

Let consider a sequence $X = \{x_1, x_2, \ldots, x_n\}$



Handling individual time steps

Let consider a sequence $X = \{x_1, x_2, \ldots, x_n\}$



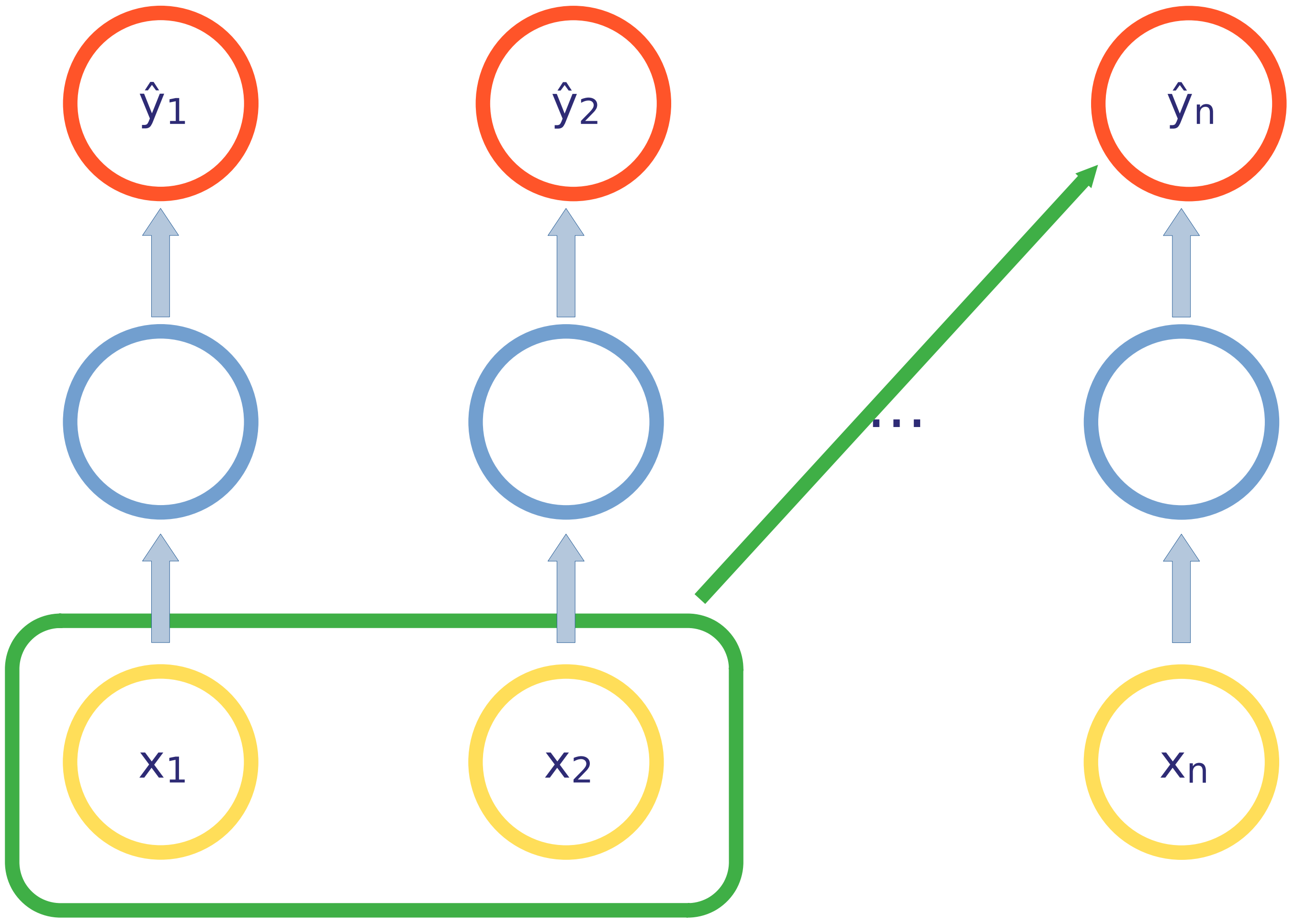
We have

$$\hat{y}_t = f(x_t)$$

with f learned and defined by the weights of the neural network.

Recurrent Neural Networks

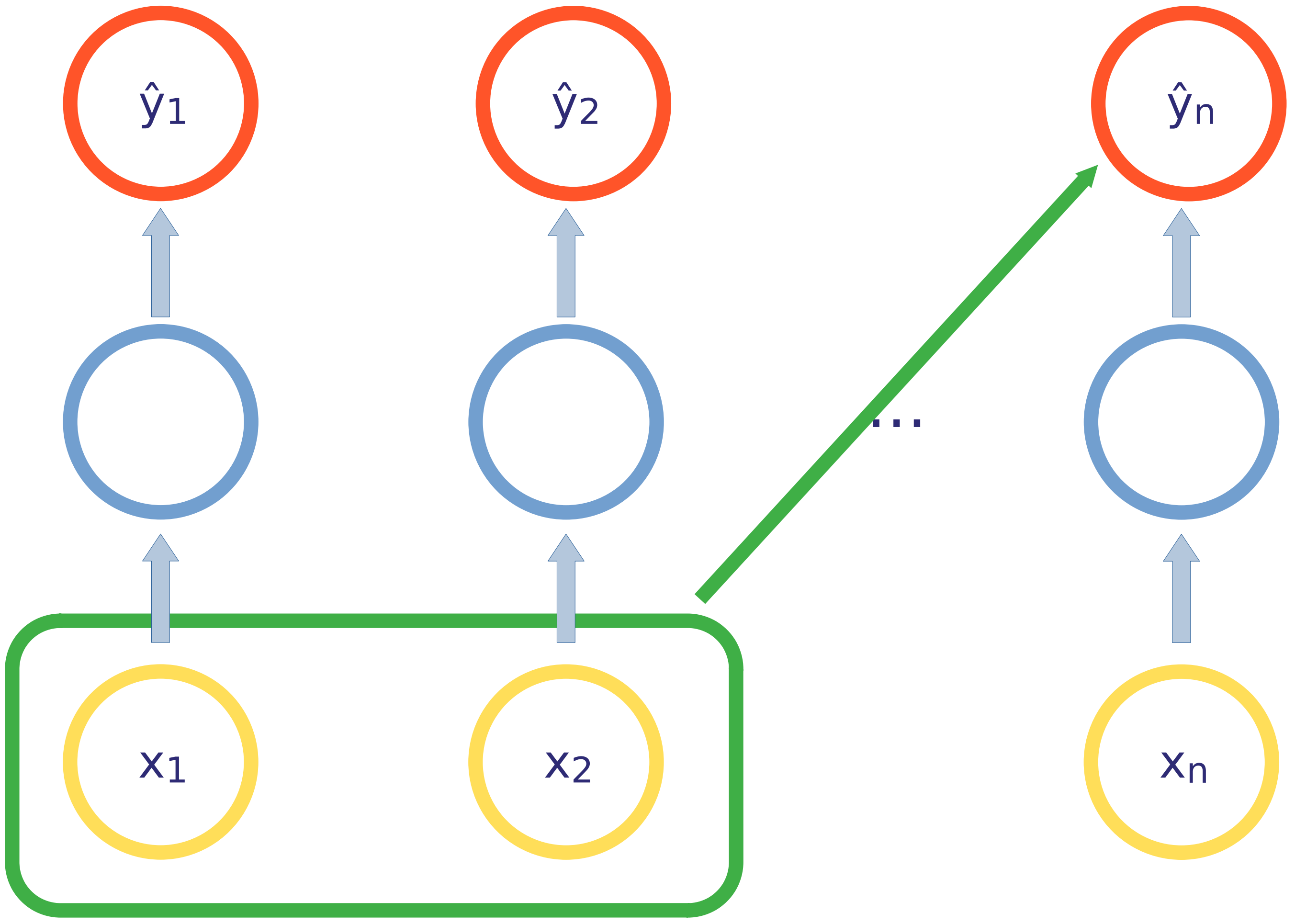
x_t have dependencies not taken into considerations.



$\hat{y}_n$ could depends of previous inputs

Recurrent Neural Networks

x_t have dependencies not taken into considerations.

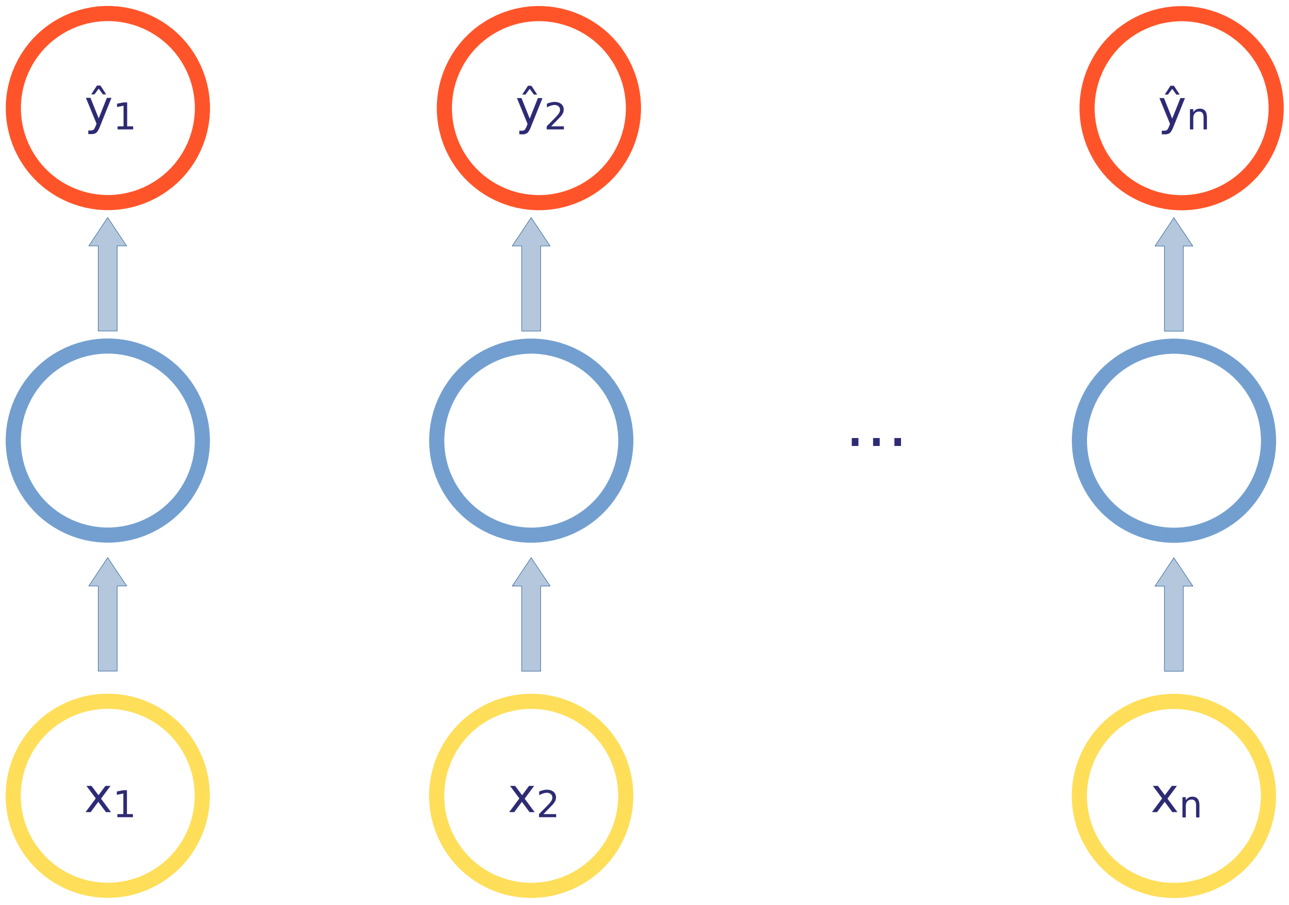


$\hat{y}_n$ could depends of previous inputs

How can we define a relation that links network computation of the different steps ?

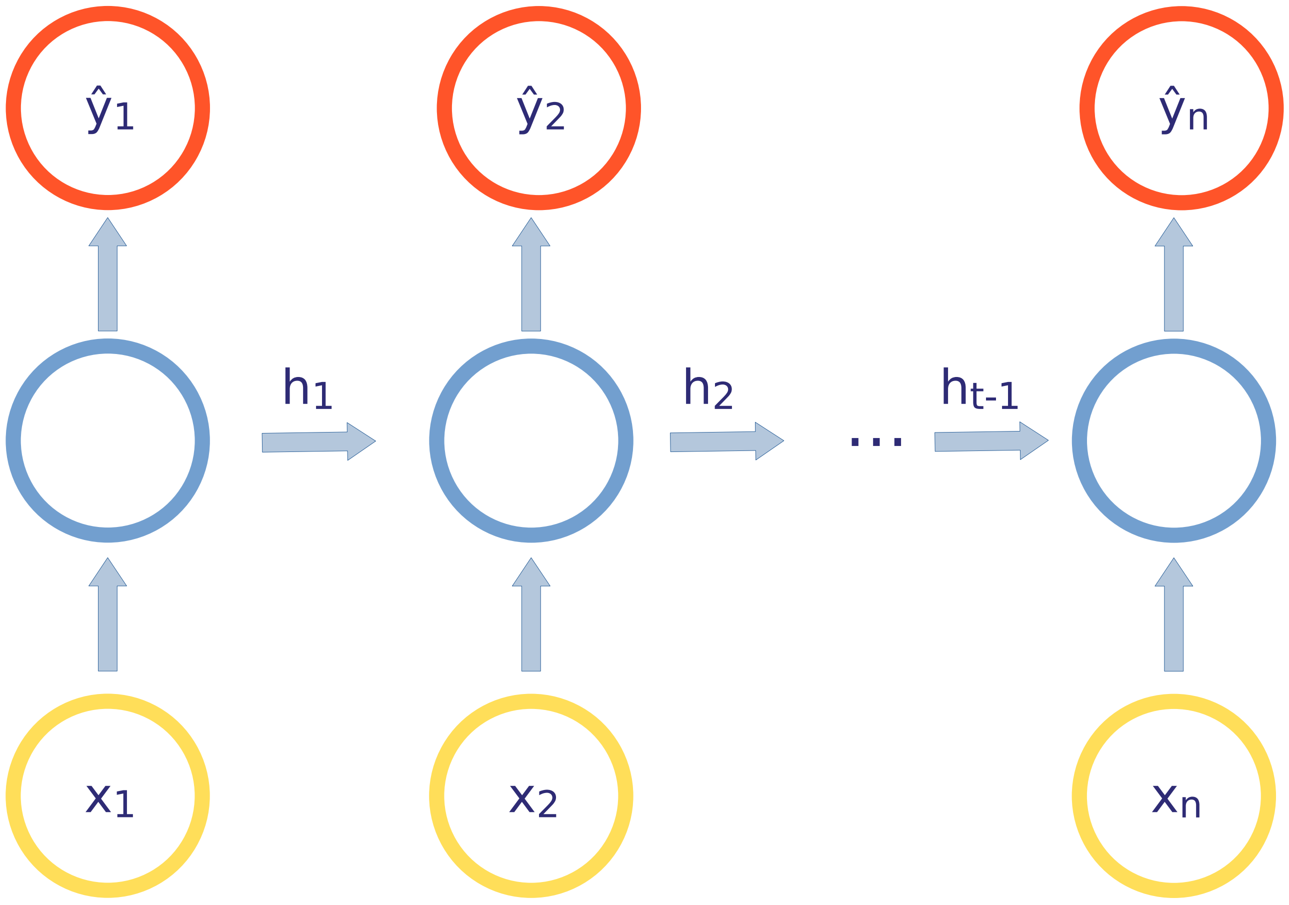
Recurrent Neural Networks

We want to pass the information from the previous computations to the next step.



Recurrent Neural Networks

We want to pass the information from the previous computations to the next step.

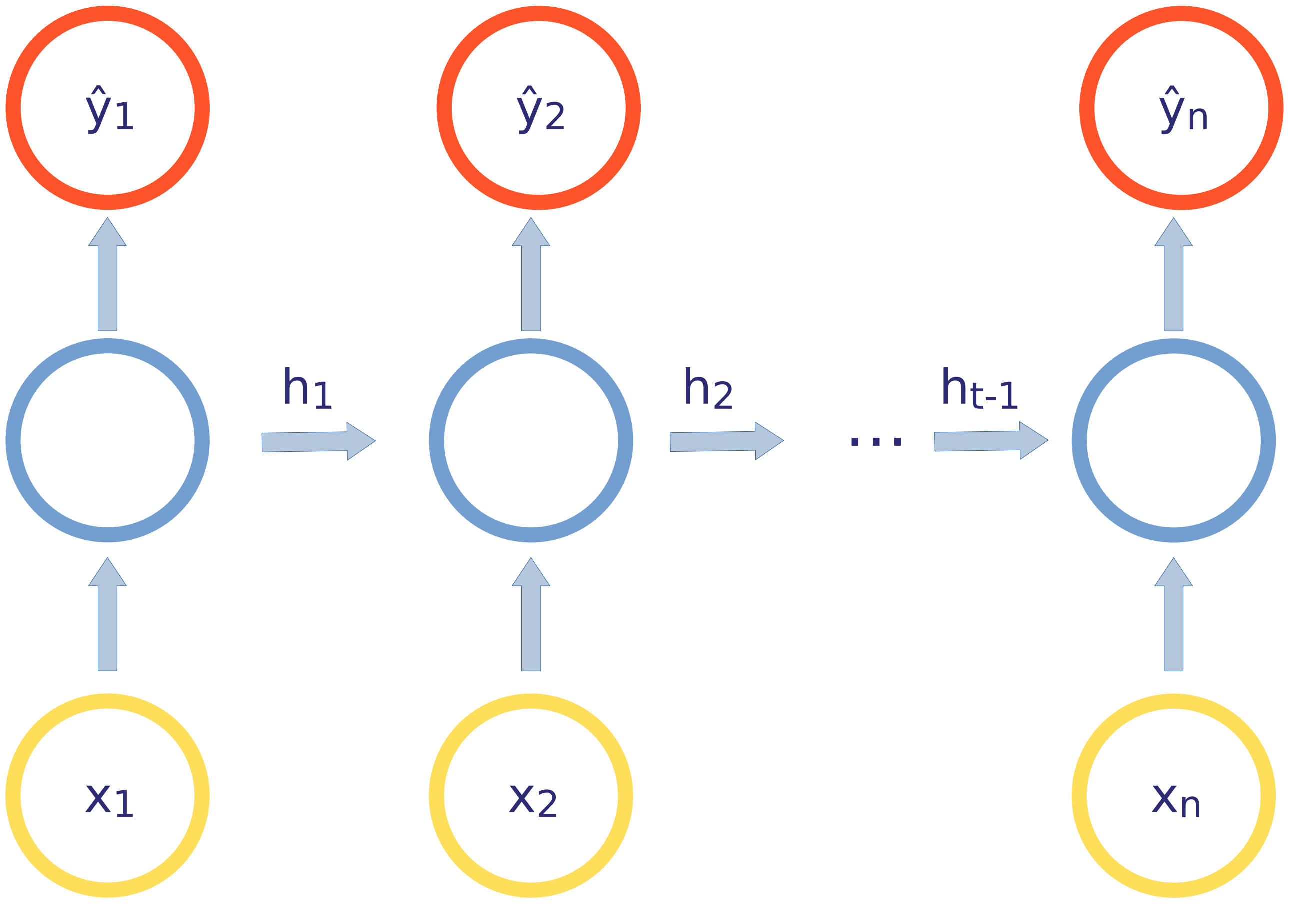


We define this as the **internal states** or **memory term** :

Variable h_t

Recurrent Neural Networks

We want to pass the information from the previous computations to the next step.

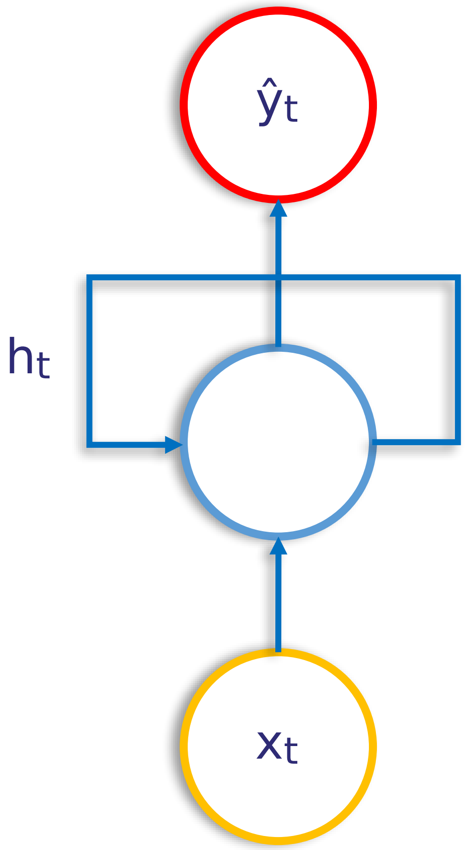


The output becomes

$$\hat{y}_t = f(x_t, h_{t-1})$$

and depends on the input and the past information.

Intermediate sum up : the recurrence relation



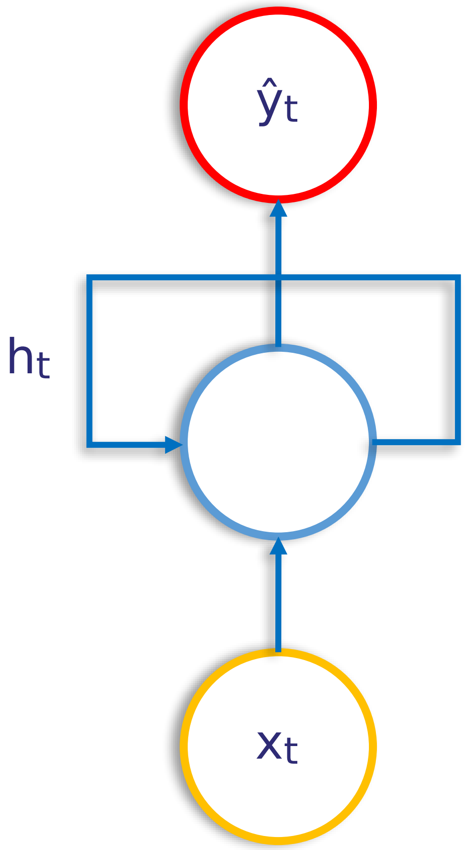
RNN seen as a loop.

Recurrence relation captures how we update internal state h of t.

$$h_t = f_w(x_t, h_{t-1}) \qquad f \mathrel{!}= f_w$$

Same function f_w and weights w at every steps.

Intermediate sum up : the recurrence relation

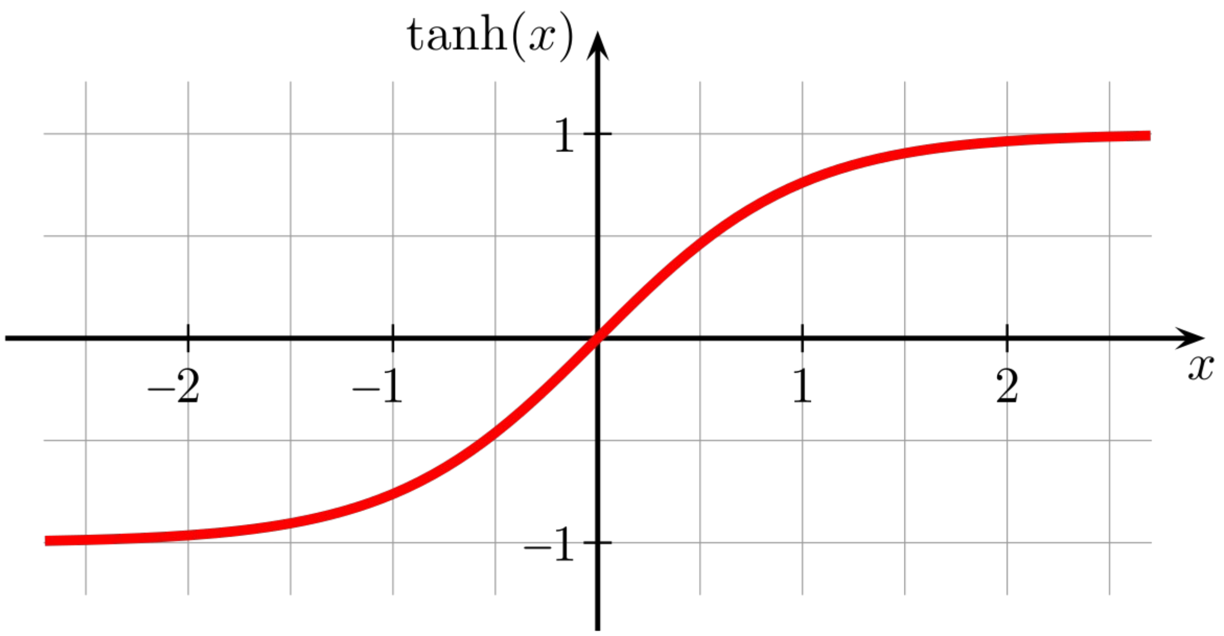


RNN seen as a loop.

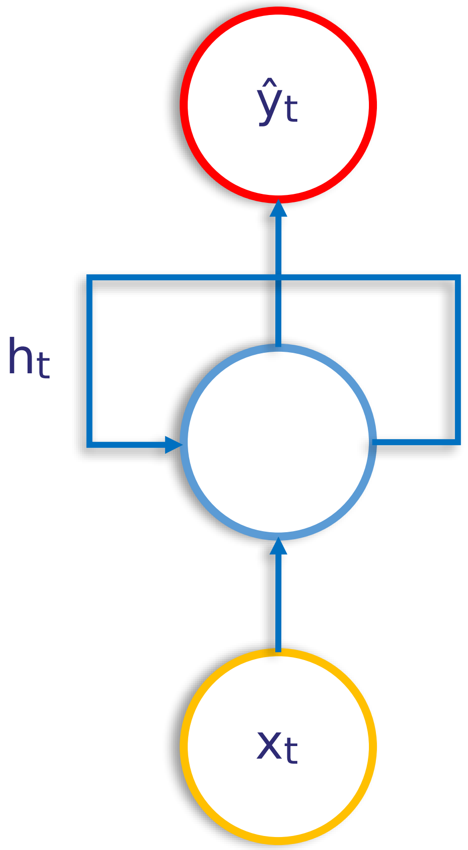
How do we compute h_t ?

$$h_t = \tanh(w_{hh}^T * h_{t-1} + w_{xh}^T * x_t)$$

Warning : several weights matrix are used !



Intermediate sum up : the recurrence relation



RNN seen as a loop.

Intuition (pseudo-code)

```
rnn = RNN()
hidden_states = [0, 0, 0, 0]
sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
        prediction, hidden_states = rnn(word, hidden_states)

next_word = prediction
# next_word = "networks"
```

Criteria to get a robust and reliable network (for sequences)

Characteristics to fulfill :

- Deal with sequences of different lengths
- Learn long-term dependencies
- Maintain information in order
- Share parameters across the sequence

Do we fulfill everything ?

Length of sequences and Language

Neural Networks understand numbers not words.

We need the sequences to be of fixed size.

How to represents the sentence numerically ?

Length of sequences and Language

Neural Networks understand numbers not words.

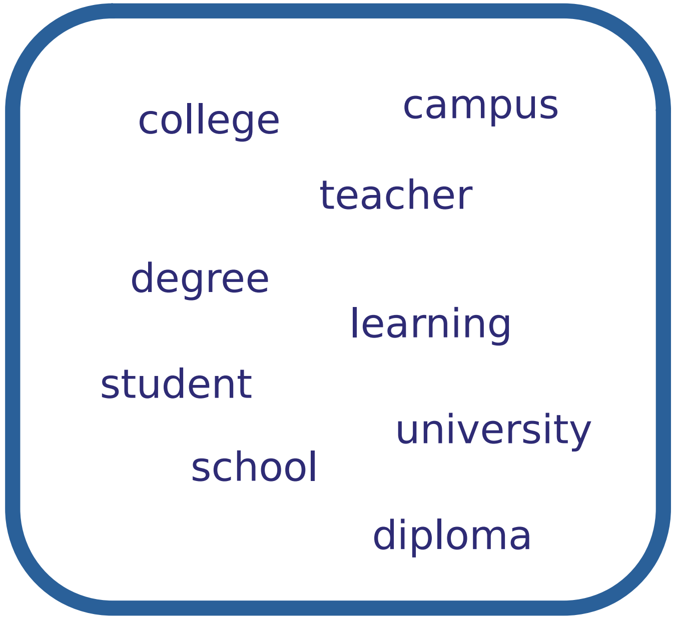
We need the sequences to be of fixed size.

How to represents the sentence numerically ?

Word Embeddings

Words embeddings

Embedding = mapping into a vector of numbers of fixed size.



1. Vocabulary

Words embeddings

Embedding = mapping into a vector of numbers of fixed size.

college
campus
teacher
degree
learning
student
university
school
diploma

1. Vocabulary

college → 0
campus → 1
teacher → 2
learning → 3

…

diploma → n

2. Indexing
(words to index)

Words embeddings

Embedding = mapping into a vector of numbers of fixed size.

college
campus
teacher
degree
learning
student
university
school
diploma

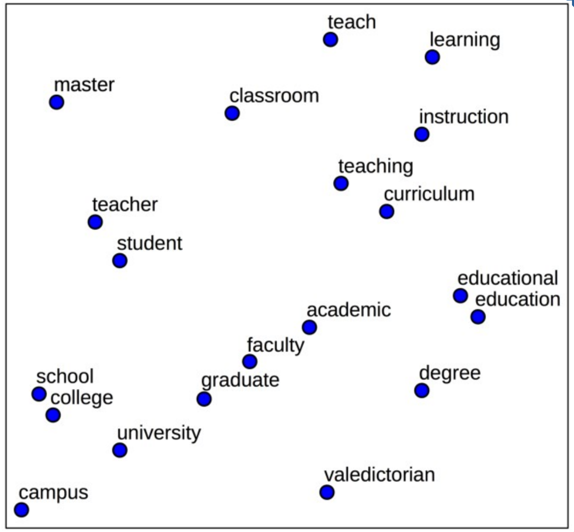
1. Vocabulary

college → 0
campus → 1
teacher → 2
learning → 3

…

diploma → n

2. Indexing
(words to index)



3. Embedding

Words embeddings

One-hot encoding

college = [1, 0, 0, 0, …, 0]
campus = [0, 1, 0, 0, …, 0]
teacher = [0, 0, 1, 0, …, 0]
learning = [0, 0, 0, 1, …, 0]

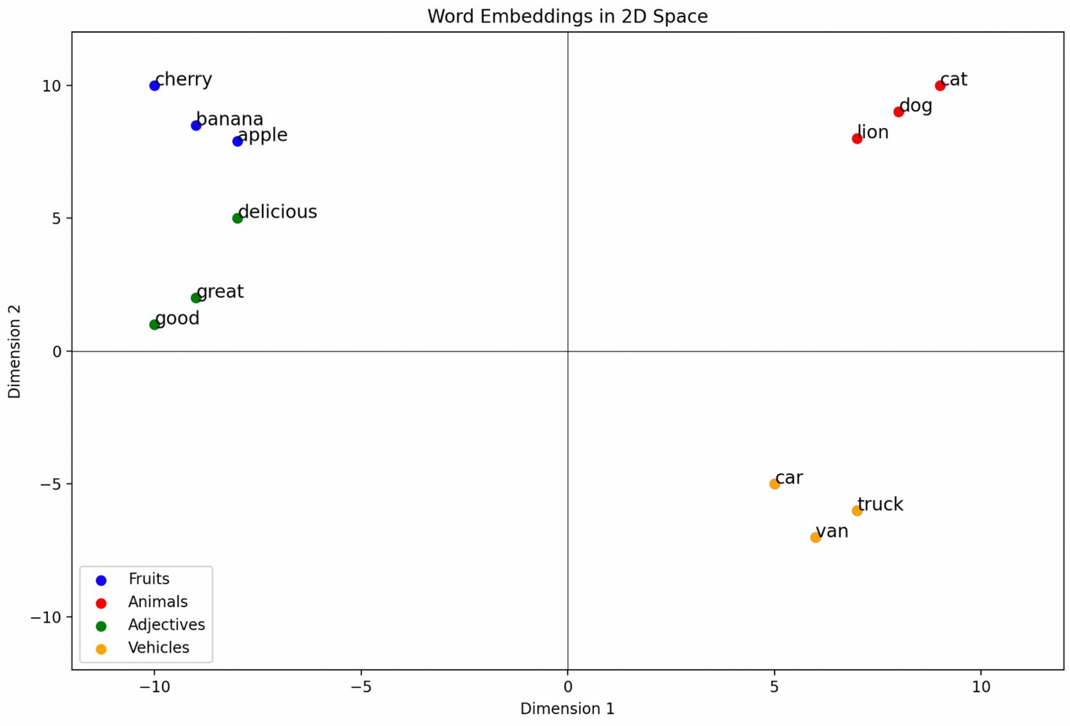
…

diploma = [0, 0, 0, 0, …, 1]

Vectors is the size of the vocabulary.

No notion of meaning.

Learned embedding



Captures some meaning from the data.

Words embeddings

Padding

Sentence_1 = [‘I’, ‘love’, ‘neural’, ‘networks’]
Sentence_2 = [‘I’, ‘use’, ‘recurrent’, ‘neural’, ‘networks’, ‘everyday’]

Embedding_1 = [0.2, 0.5, 0.35, 0.4]
Embedding_2 = [0.2, 0.9, 0.65, 0.35, 0.4, 0.8]

Embedding_1 and Embedding_2 have different size…

With **padding**:

Embedding_1 = [0.2, 0.5, 0.35, 0.4, 0.0, 0.0]
Embedding_2 = [0.2, 0.9, 0.65, 0.35, 0.4, 0.8]

Padding can be done to fixed size (with truncation) or max size.

Criteria to get a robust and reliable network (for sequences)

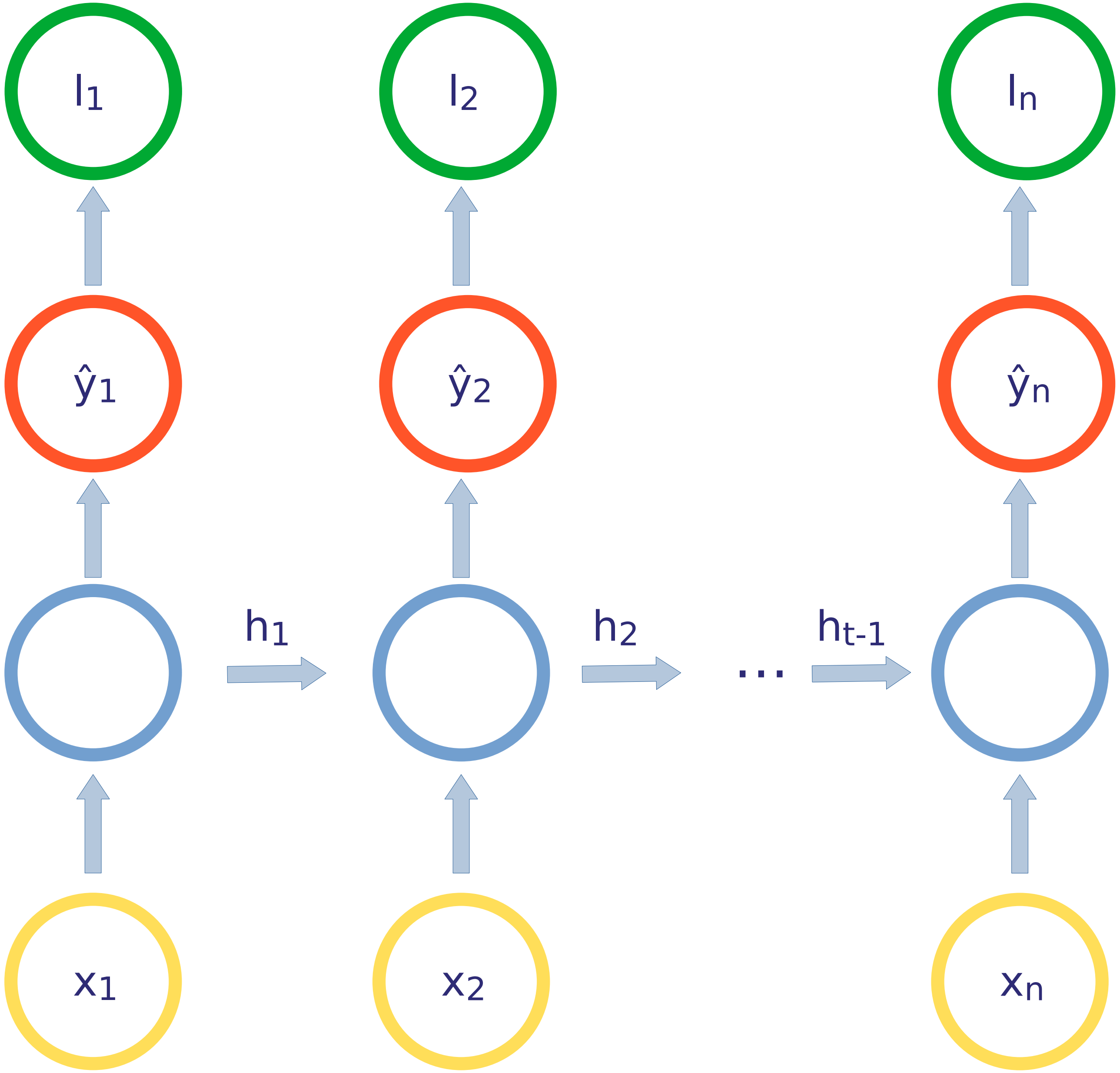
Characteristics to fulfill :

- Deal with sequences of different lengths ✓
- Learn long-term dependencies
- Maintain information in order ✓
- Share parameters across the sequence ✓

We have seen how to process a sequence,
but how do we train the network ?

How do we compute the loss when
we have N predictions ?

Loss computation with RNN



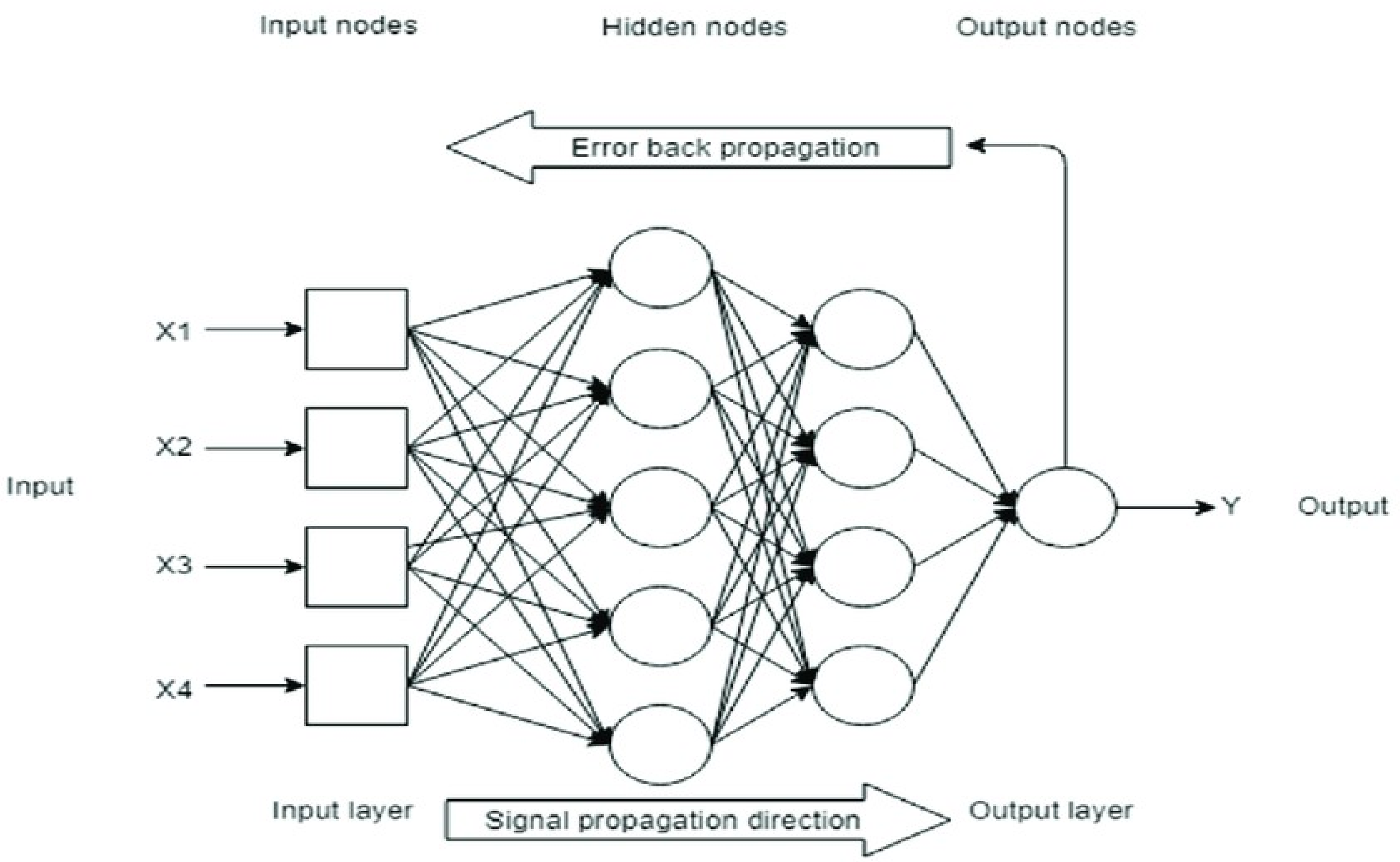
Compute a loss at every steps.

$\hat{y}_t = y_t$?

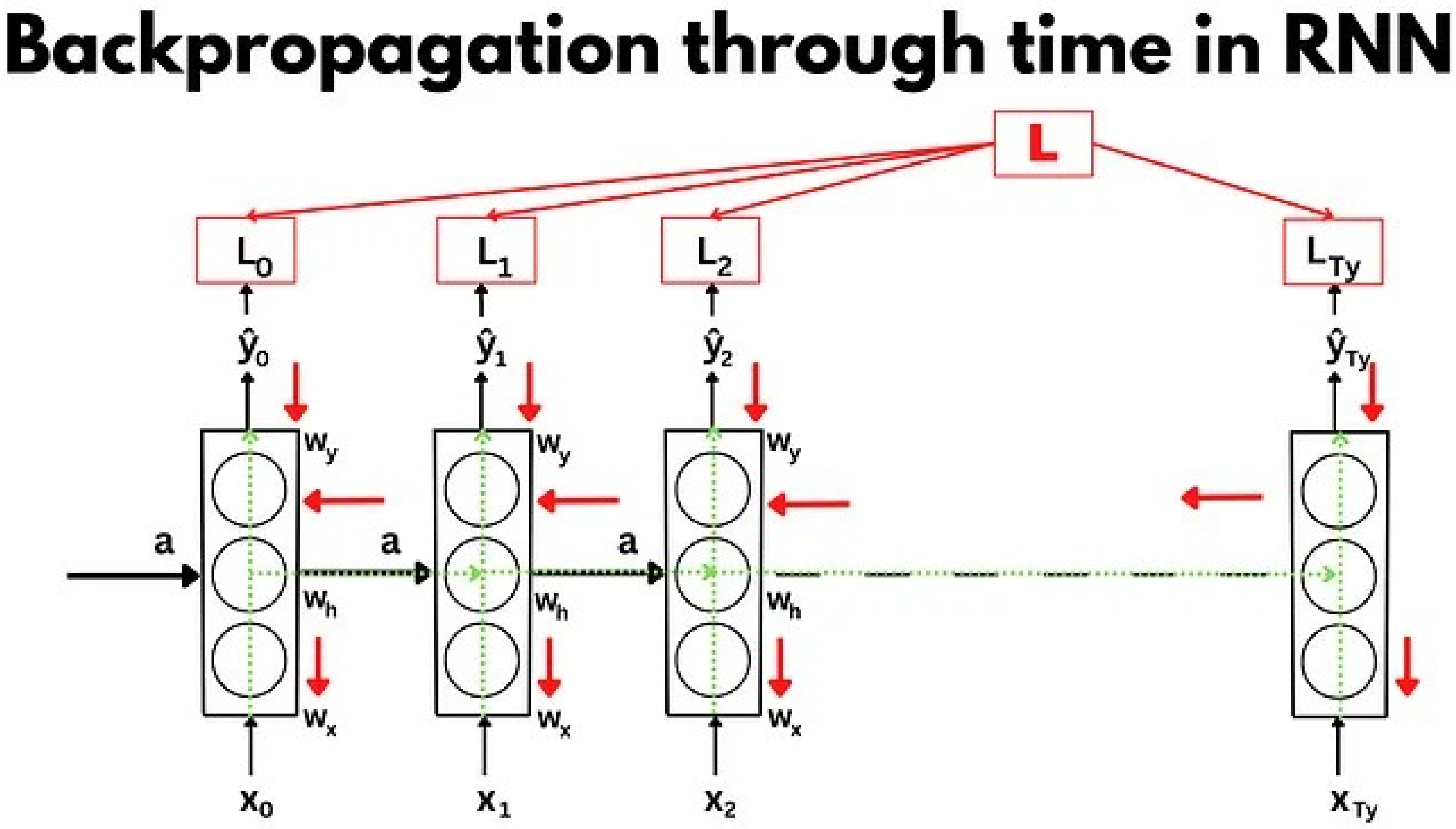
Total loss for a given sequence :

$$L = l_1 + l_2 + \ldots + l_n$$

Backpropagation (Feed Forward Networks)



Backpropagation Through Time (RNN)



Backpropagation Through Time (RNN)

RNN have individuals losses across steps :

→ When back-propagating we have to propagate the loss through each individuals steps
= Back-propagation Through Time

→ We take the predictions and back-propagate back through the network to define and update the loss with regards to each parameters in the network and adjust it.

Backpropagation Through Time (RNN)

RNN have individuals losses across steps :

→ When backpropagating we have to propagate the loss through each individuals steps = Backpropagation Through Time

→ We take the predictions and backpropagate back through the network to define and update the loss with regards to each parameters in the network and adjust it.

Example at step 2 :

$$\frac{\partial L_2}{\partial wy} = \frac{\partial L_2}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial y_2} * \frac{\partial y_2}{\partial wy}$$

Compute the gradient (chain rule)

$$wy = wy - \alpha * \frac{\partial L}{\partial wy}$$

Update parameters

Problem : RNN’s backpropagation is tricky !

Many repeated computations and multiplication in order to compute the gradients wrt. to the first step → issues with the gradient.

Problem : RNN’s backpropagation is tricky !

Many repeated computations and multiplication in order to compute the gradients wrt. to the first step → issues with the gradient.

If the values of the gradient get :

- Too large → **Exploding gradient** problem
  → Impossible to train the network

- Too small → **Vanishing gradient** problem
  → Can’t update the parameters and train the model properly

Problem : RNN’s backpropagation is tricky !

→ **Exploding gradient** problem

Simple solution : weights clipping (scale the weights at reasonable values)

→ **Vanishing gradient** problem

Three tools to mitigate the problems :

- Activation function
- Weights initialization
- Network architecture

Why is the vanishing gradient a problem ?

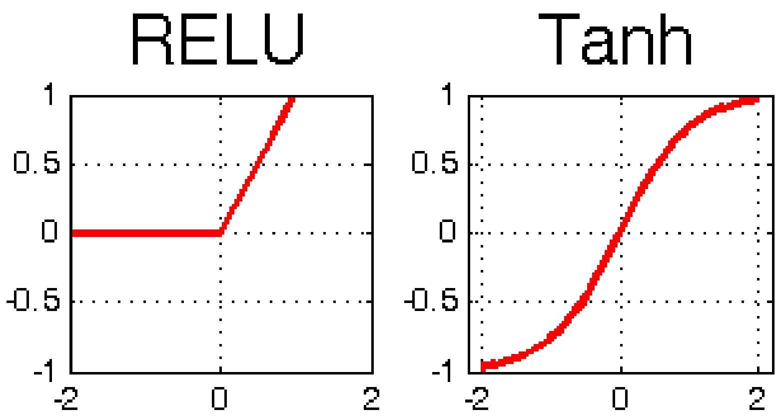
Multiplying many small number with small number → Gradient get smaller and smaller.

→ In case of short sequences, not a problem.

→ In case of long sequences, we need information from further back in the sequence to do the prediction. The problem then appears because of the multiplicity of operations to do.

Vanishing gradient problem and solutions

1) Activation function : switch from Tanh to ReLU.
→ prevents the derivative to shrink
the gradients when x > 0.



2) Weights initialization :
At initialization :
- weights are set to the identity matrix.
- bias are set to 0.

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

→ prevents the weights from shrinking to 0

Vanishing gradient problem and solutions

3) Adjust the architecture (most efficient)

→ Objective : controlling the flow of information in the network to filter out what is not important.

→ Add gated cells to selectively add or remove information within each recurrent unit.

→ Techniques : LSTM, GRU, etc.

We will focus on LSTM.

Long-Short Term Memory

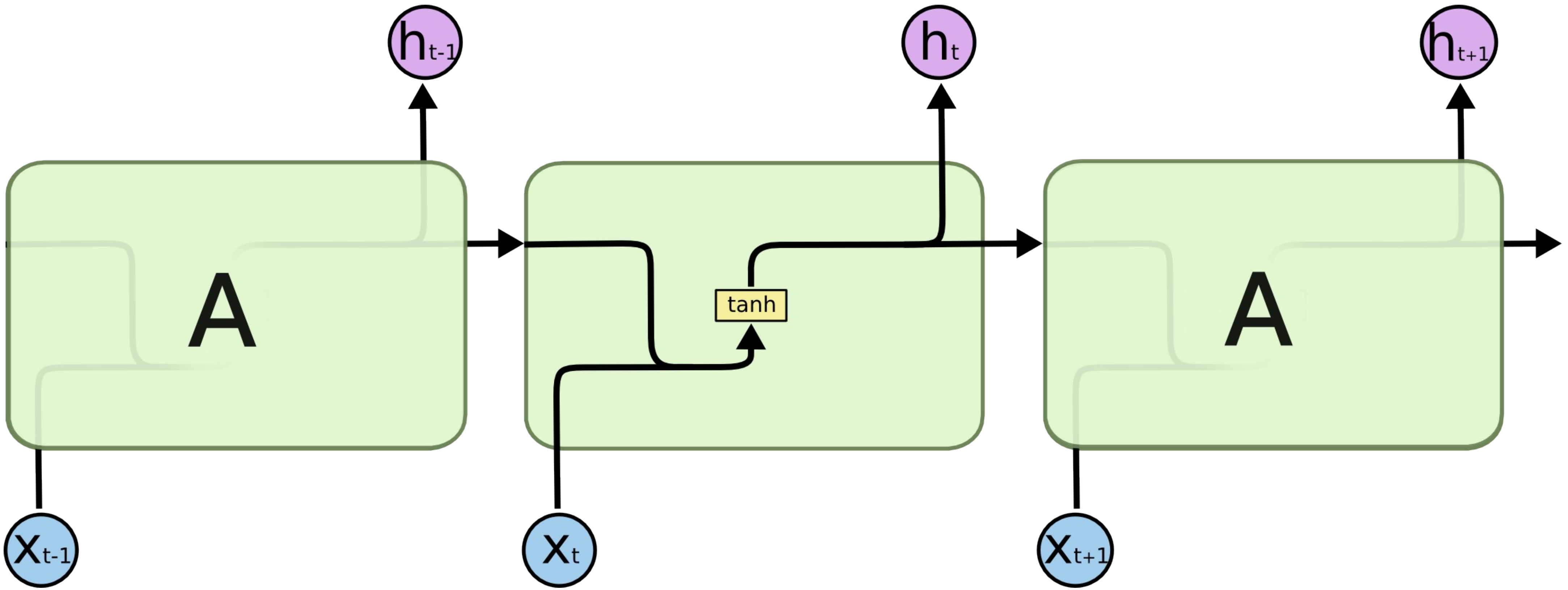
Key concepts :

1) LSTM maintains a cell state c_t (like RNN) but it is independent from what is outputted.

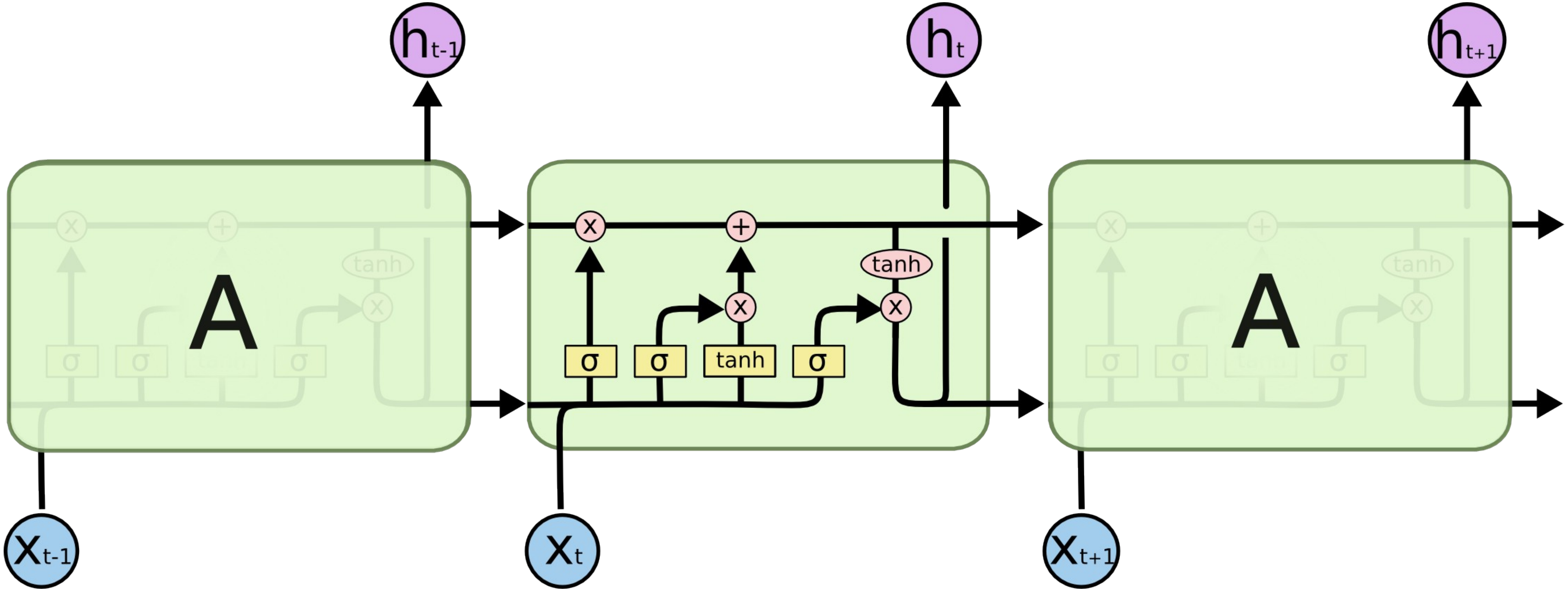
2) Cell state is updated according to the gates that control the information flow :

→ **Forget** gate get rids of irrelevant information.
→ **Store** relevant information from the current input.
→ Selectively **update** cell state
→ **Output** gate returns a filtered version of the cell state.

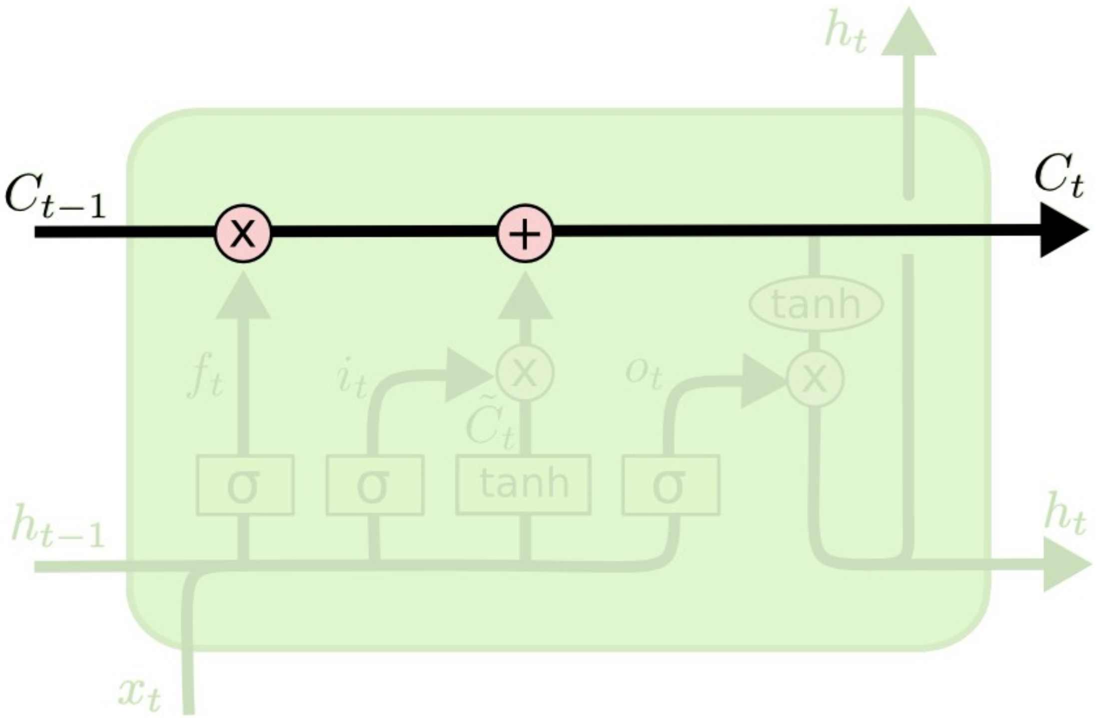
RNN reminder



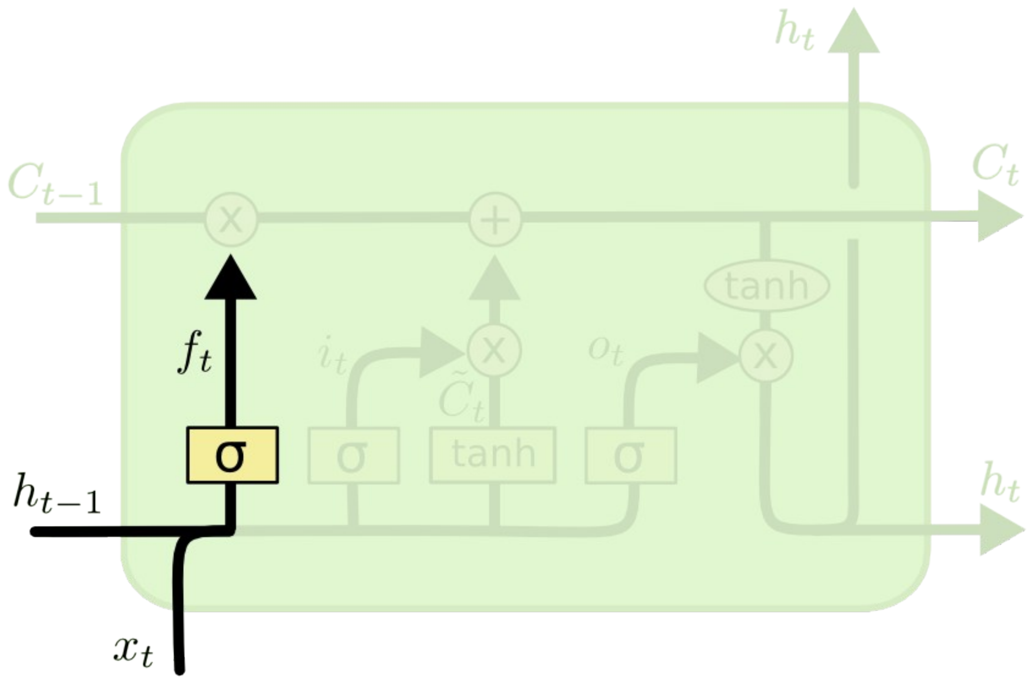
LSTM, global view



LSTM maintain a cell state through steps



LSTM, forget gate



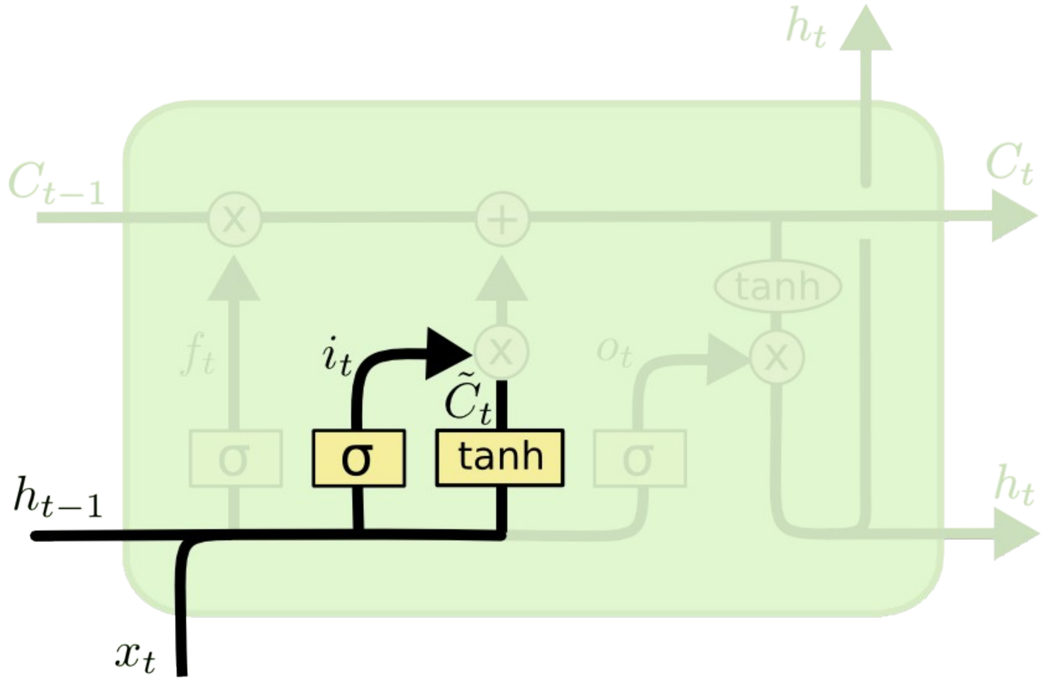
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Uses a sigmoid activation function

If close to 0, information is forgotten;
If close to 1, the information is retained.

Retrieve internal state and get rid of irrelevant information.

LSTM, store relevant information from input



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

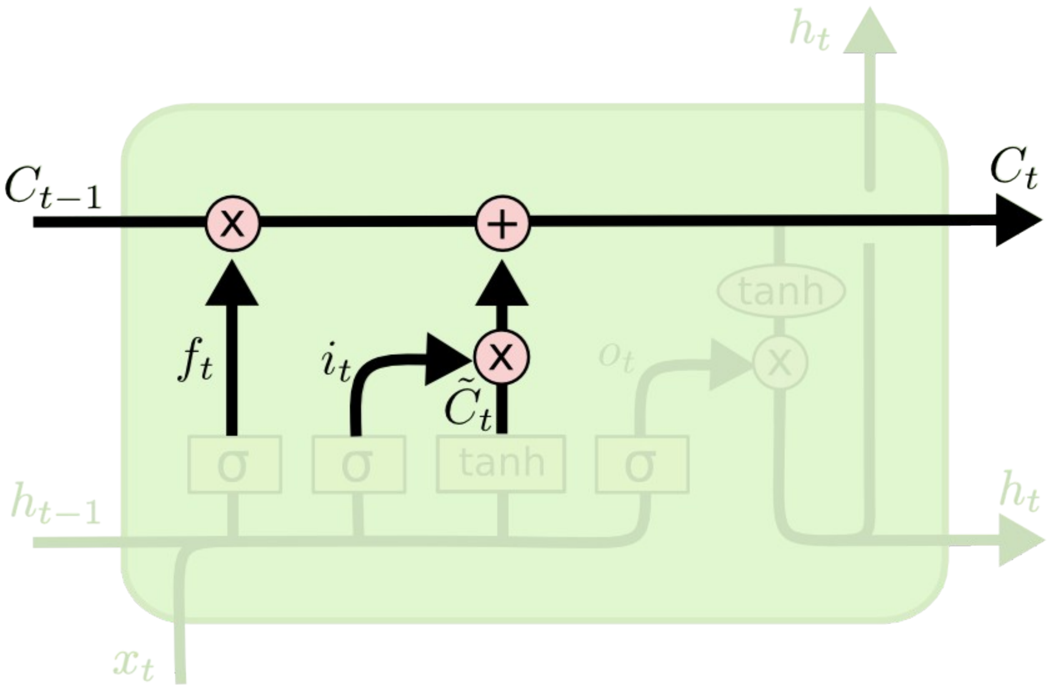
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

sigmoid function determine if new information is accepted (close to 1) or not (close to 0)

tanh function creates a candidate value that can be added to the cell state

Input gate: retrieve internal state and extract relevant information.

LSTM, update cell state

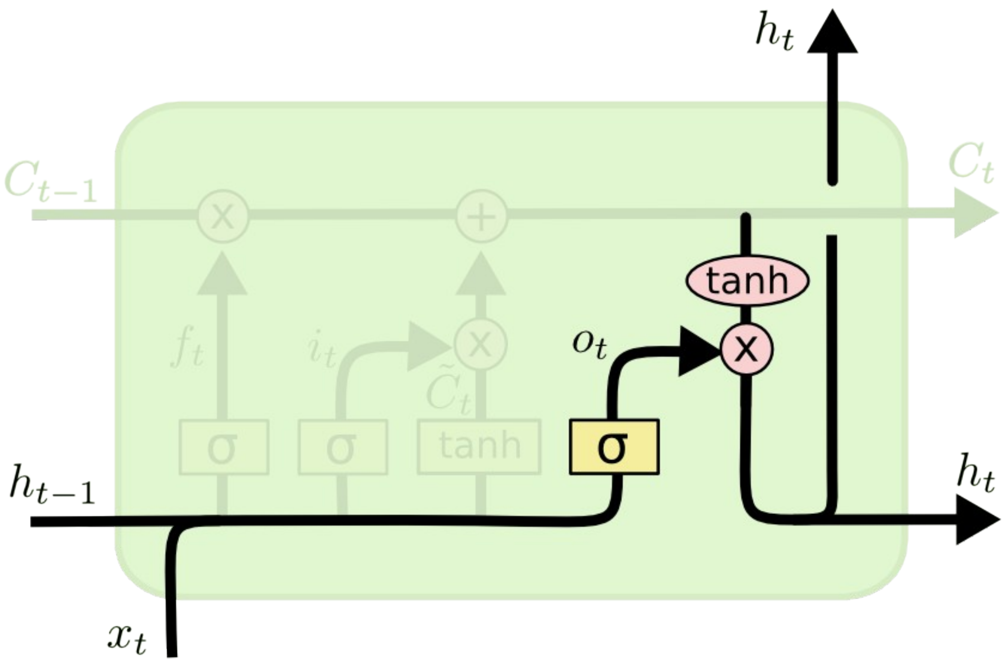


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Old information is updated based on the importance of the new input

Update with remove and added information.

LSTM, output filtered version of the cell state



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

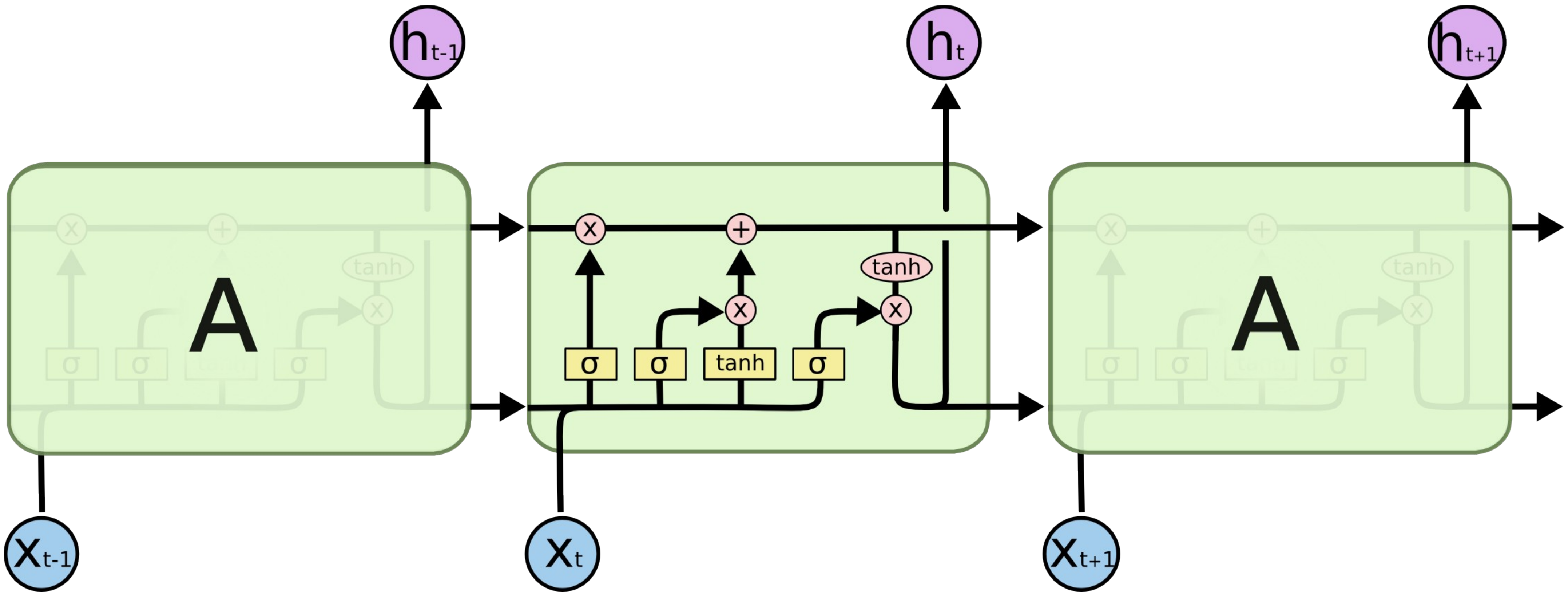
$$h_t = o_t * \tanh\left(C_t\right)$$

sigmoid function decides what portion of the cell state should be passed.

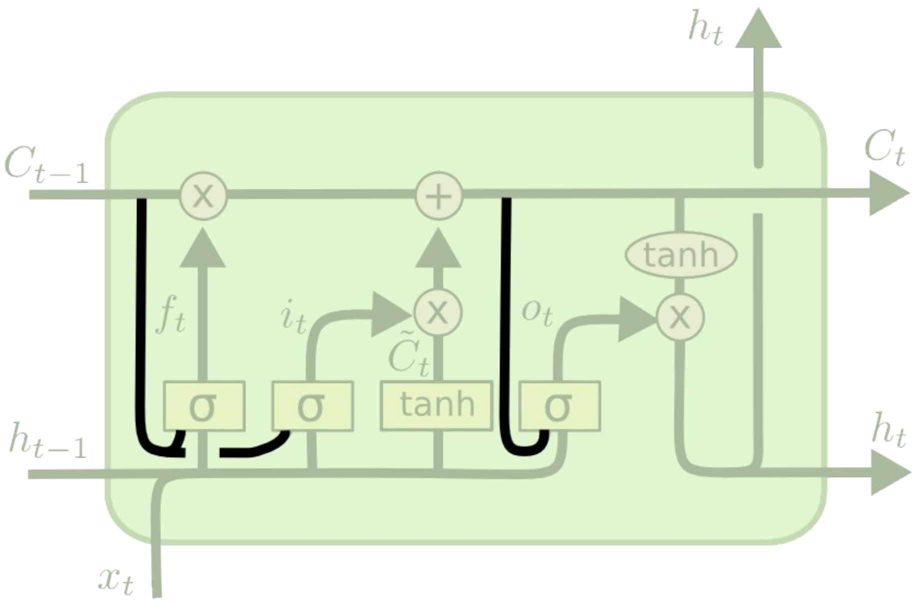
Compute the hidden state based on the cell state with tanh function.

Transmit less information to allow long-term sequences.

LSTM, global view



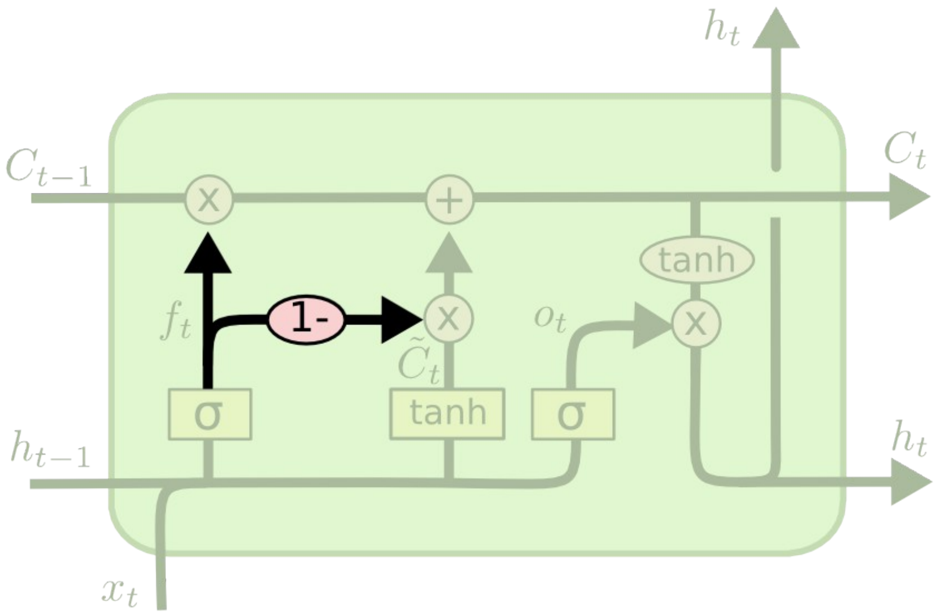
LSTM variations



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Gers & Schmidhuber (2000), adding “peephole” connections (the gate layers (forget and input) look at the cell state)

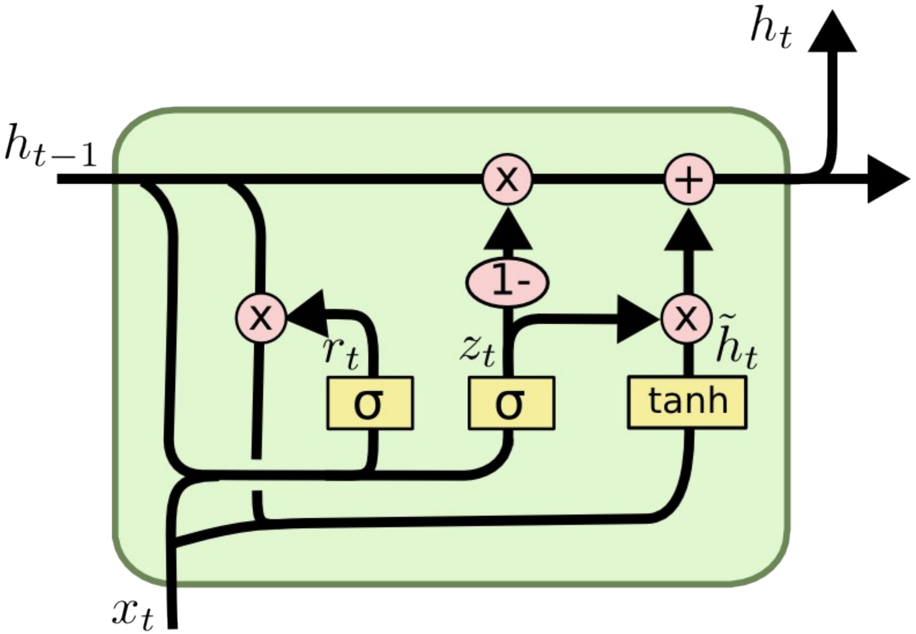
LSTM variations



$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Variation : coupled forget and input gates

LSTM variations



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gated Recurrent Unit (GRU, Cho, et al. (2014)) :

→ combine the forget and input gates into a single “update gate”

→ merge the cell state and hidden state

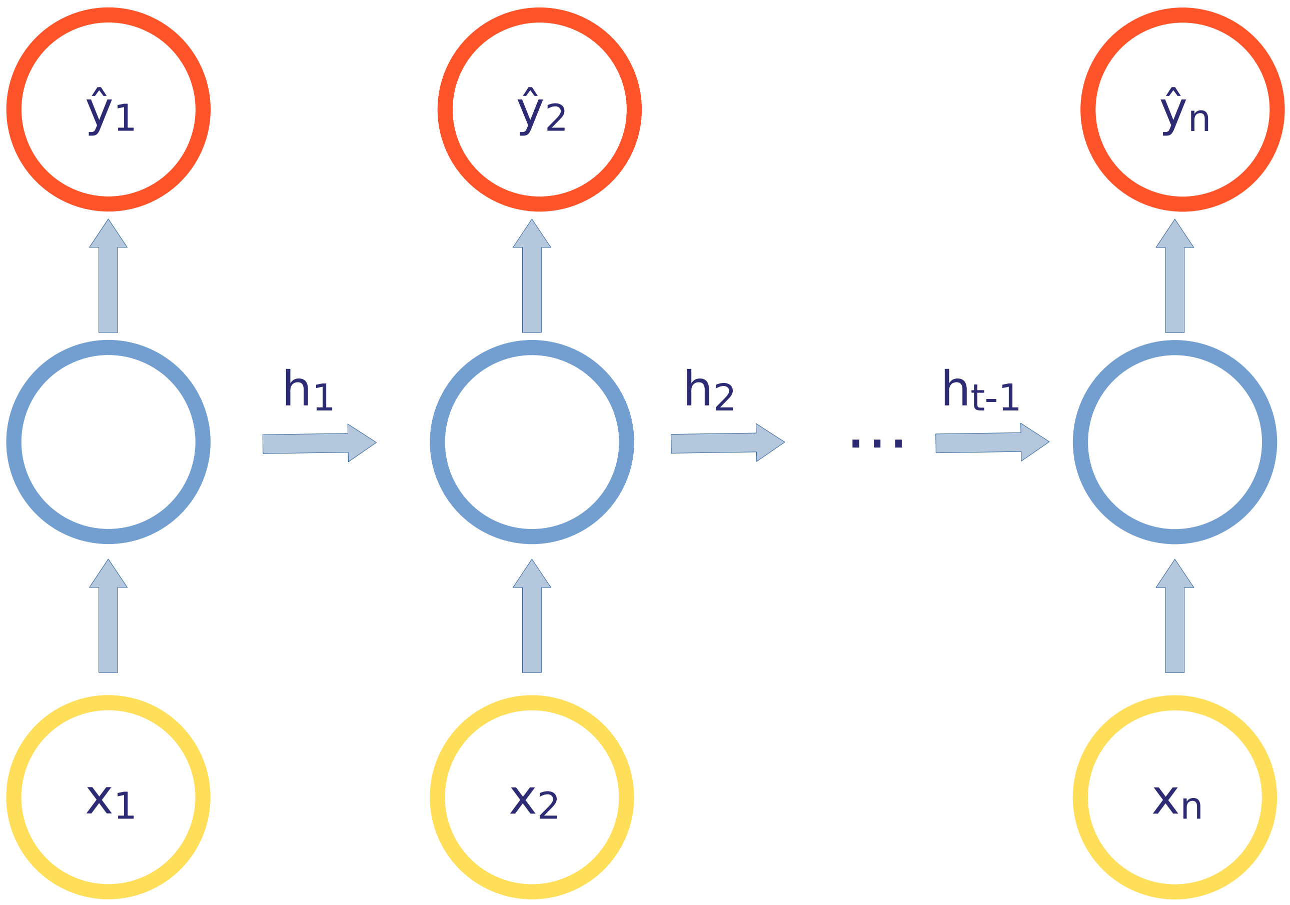
Criteria to get a robust and reliable network (for sequences)

Characteristics to fulfill :

- Deal with sequences of different lengths ✓
- Learn long-term dependencies ✓ (at least longer)
- Maintain information in order ✓
- Share parameters across the sequence ✓

Bi-directional RNN

Limits of RNN



→ RNN processes sequences in a single direction.
(left-to-right or right-to-left)

→ Only information from previous steps can be used.

Limits of RNN

Example :

→ Apple is my favorite _____.

Limits of RNN

Example :

→ Apple is my favorite fruit/company/phone.

Limits of RNN

Example :

→ Apple is my favorite fruit/company/phone.

→ Apple is my favourite _____, and I work there.

Limits of RNN

Example :

→ Apple is my favorite fruit/company/phone.

→ Apple is my favourite company, and I work there.

Limits of RNN

Example :

→ Apple is my favorite fruit/company/phone.

→ Apple is my favourite company, and I work there.

→ Apple is my favorite ______, and I am going to buy one.

Limits of RNN

Example :

→ Apple is my favorite fruit/company/phone.

→ Apple is my favourite company, and I work there.

→ Apple is my favorite phone, and I am going to buy one.

Limits of RNN

Example :

→ Apple is my favorite fruit/company/phone.

→ Apple is my favourite company, and I work there.

→ Apple is my favorite phone, and I am going to buy one.

We need later information to make a good prediction.

Bi-directional RNN or Bi-RNN

Objective :

→ Capture the information in the input data by processing it in both directions.

→ Bi-RNN = RNN that processes data in forward **and** backward directions.

Idea :

→ Combine the outputs of **two** RNNs.

Bi-directional RNN or Bi-RNN

Idea :

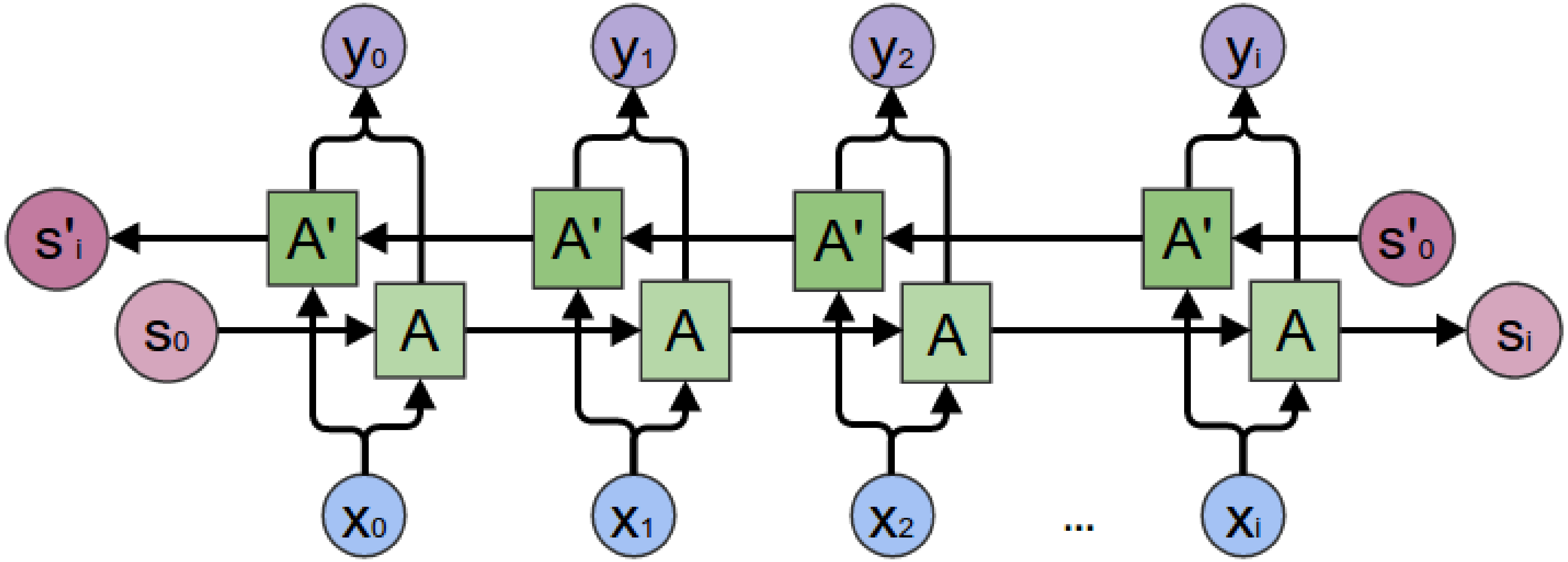
→ Combine the outputs of **two** RNNs :

- Forward RNN processes the data from left to right.

- Backward RNN processes the data from right to left.

Bi-directional RNN or Bi-RNN

→ Combine the outputs of **two** RNNs :



How to combine outputs of the RNNs ?

→ Concatenation (default): outputs of the forward and backward RNNs are concatenated together. Output tensor is twice the size of the input vector.

→ Sum: outputs of the forward and backward RNNs are added together element-wise. Output tensor of the same size as the input.

→ Average: outputs of the forward and backward RNNs are averaged element-wise. Output tensor of the same size as the input.

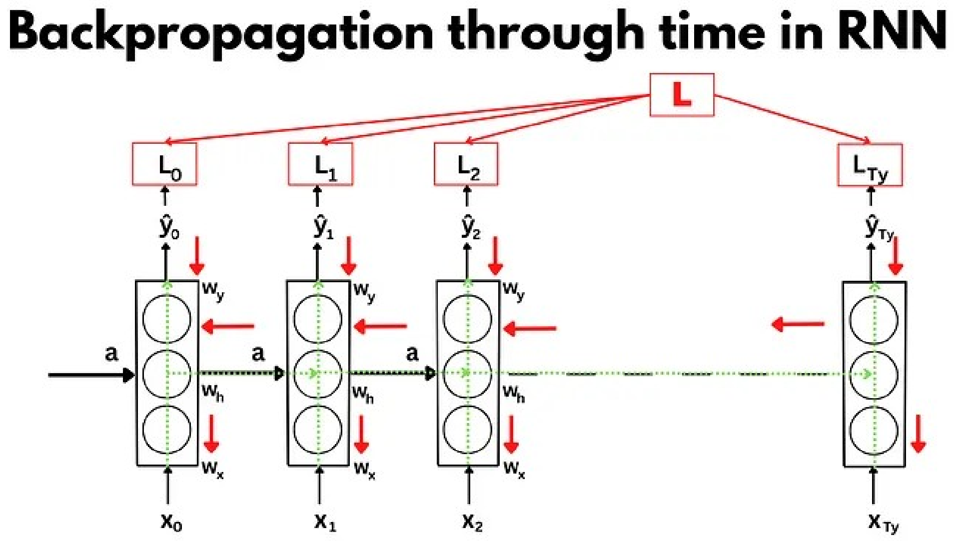
→ Maximum: maximum value of the forward and backward outputs is taken at each step. Output tensor of the same size as the input.

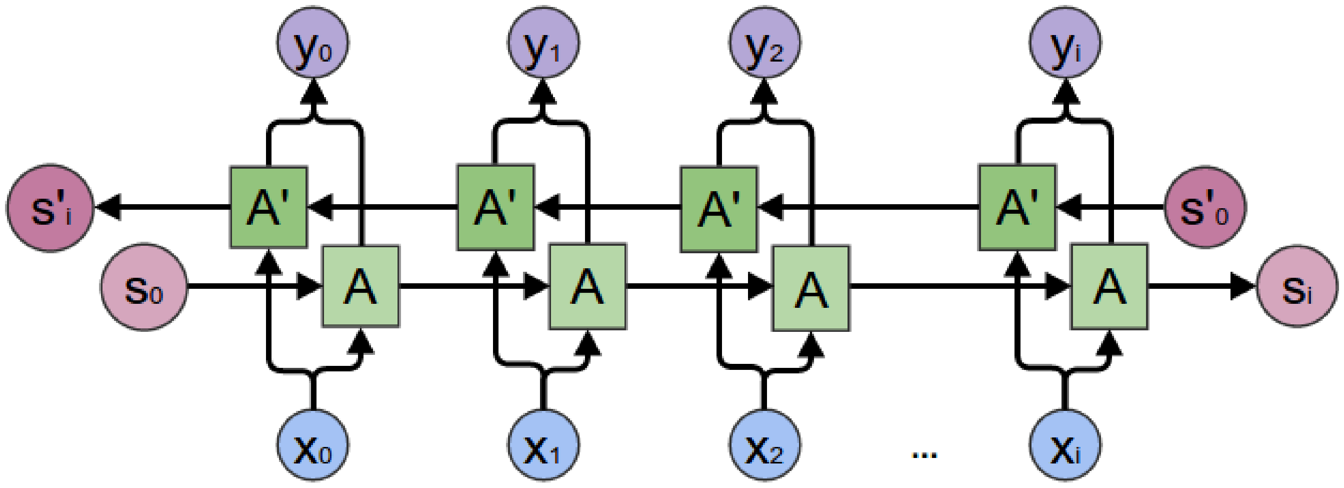
How to back-propagate during training ?

→ Back-propagation through time (BPTT) (as for RNN).

→ 2 separate BPTT (one for each network).

→ use the same output to compute the loss.





How to back-propagate during training ?

→ Back-propagation through time (BPTT) (as for RNN).

→ 2 separate BPTT (one for each network).

→ In the end, training a bi-RNN boils down to training :

1) a RNN to predict the next word knowing the previous words.

2) a second RNN to predict the previous word knowing the next ones.

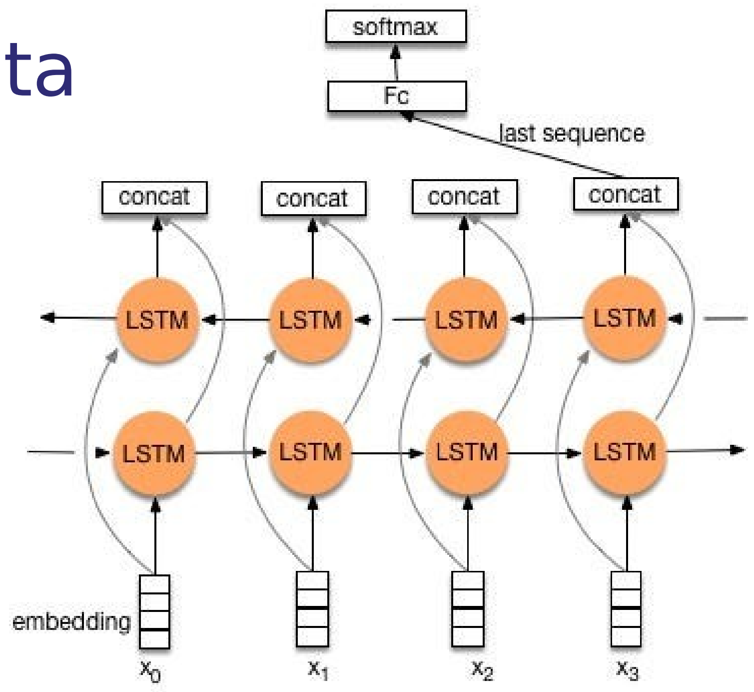
→ Bi-RNN consider information from previous and next steps when making predictions.

Bi-RNN : Advantages

→ Better performance for sequential data processing since it consider previous and future steps. Outperfom RNN in many tasks.

→ Capture long-term dependencies in the data (same reason). Can be combine with LSTM.

→ Better handling of complex data.
Bi-RNNs can capture complex patterns in the input data.



Bi-RNN : Drawbacks

→ Increased computational complexity.

- more computational resources.
- more difficult to implement.
- less efficient regarding runtime performance.
- requires more memory to store the weights.

→ Harder to optimize.

More parameters implies more difficulties to optimize.
Slower convergence and gradients can interfere.

→ Need for longer input sequences to capture long-term dependencies.

Next time : practical session

Implementation of a Recurrent Neural Network.

télécom saint-étienne
école d'ingénieurs nouvelles technologies

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

Apprentissage Automatique

Neural Networks for sequences (Part 2)

Thibaud Leteno (thibaud.leteno@univ-st-etienne.fr)

Based on the course Ava Amini.

Avril 2025

Introduction to RNN

- Seq2Seq architecture

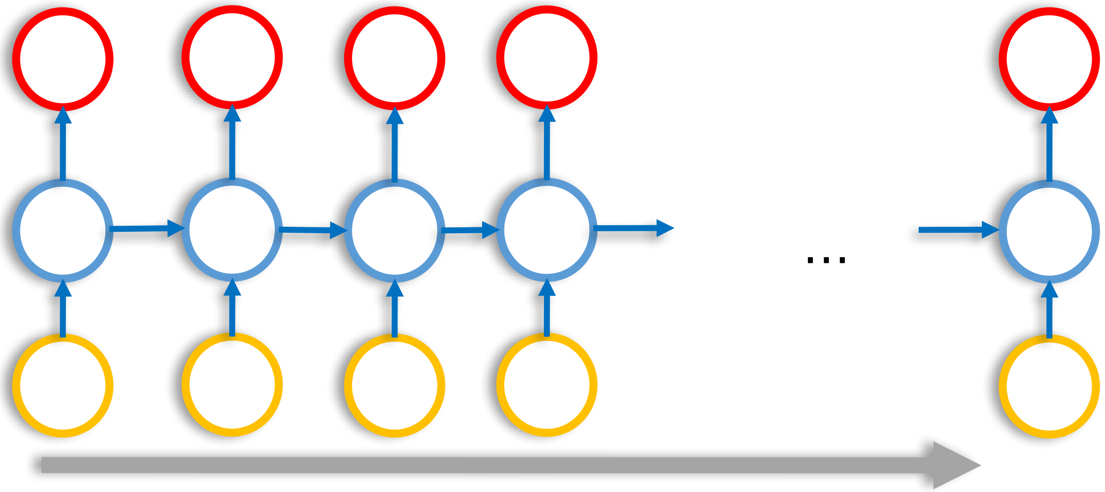
- Attention mechanism

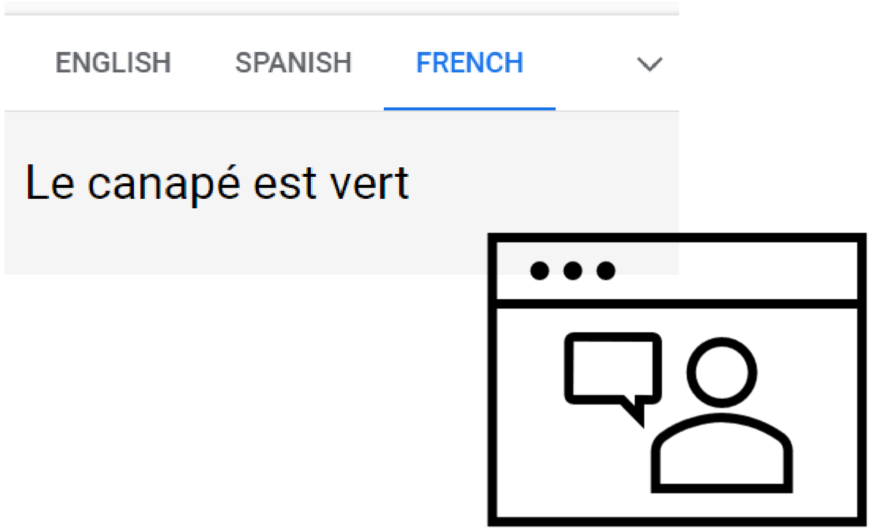
- Transformers

Seq2Seq architecture

Limitations of RNN for many-to-many cases

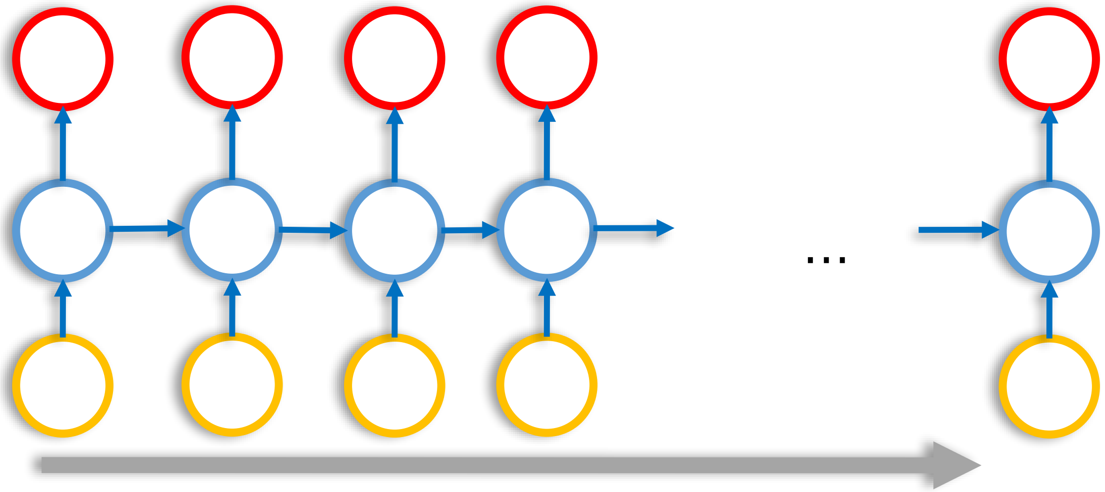
Many-to-many





Limitations of RNN for many-to-many cases

Many-to-many



Translation English-Chinese.

“What are you doing today?”
“ 今天你在做什麼？ ”

From 5 words to 7 symbols.

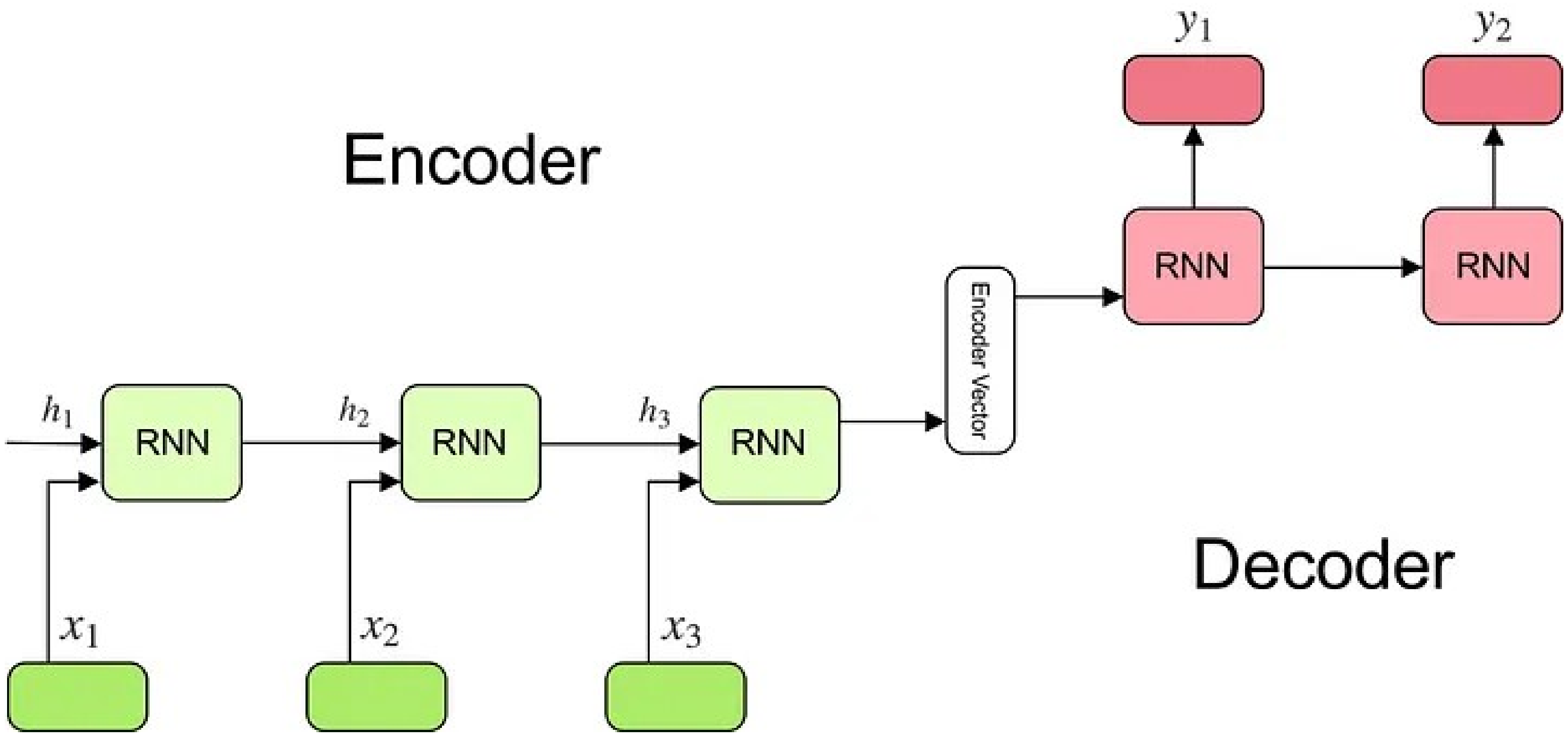
Definitions and applications

→ Seq2Seq (Sutskever et al., 2014) aims at mapping a sequence of size N to a sequence of size M.

→ Applications :

- Translation
- Speech recognition
- Video captioning
- Text generation (Chatbot)

Seq2Seq architecture



3 components : Encoder, Encoder Vector, Decoder.

Seq2Seq architecture

→ Encoder (stack of recurrent units (RNN, LSTM or GRU))

- $h_t = f_h(w_{hh}^T * h_{t-1} + w_{xh}^T * x_t)$

Seq2Seq architecture

→ Encoder (stack of recurrent units (RNN, LSTM or GRU))

- $h_t = f_h(w_{hh}^T * h_{t-1} + w_{xh}^T * x_t)$

→ Encoder Vector

- final hidden state produced from the encoder part, contains the information from the input elements.

- initial state of the decoder.

Seq2Seq architecture

→ Encoder (stack of recurrent units (RNN, LSTM or GRU))

- $h_t = f_h(w_{hh}^T * h_{t-1} + w_{xh}^T * x_t)$

→ Encoder Vector

- final hidden state produced from the encoder part, contains the information from the input elements.

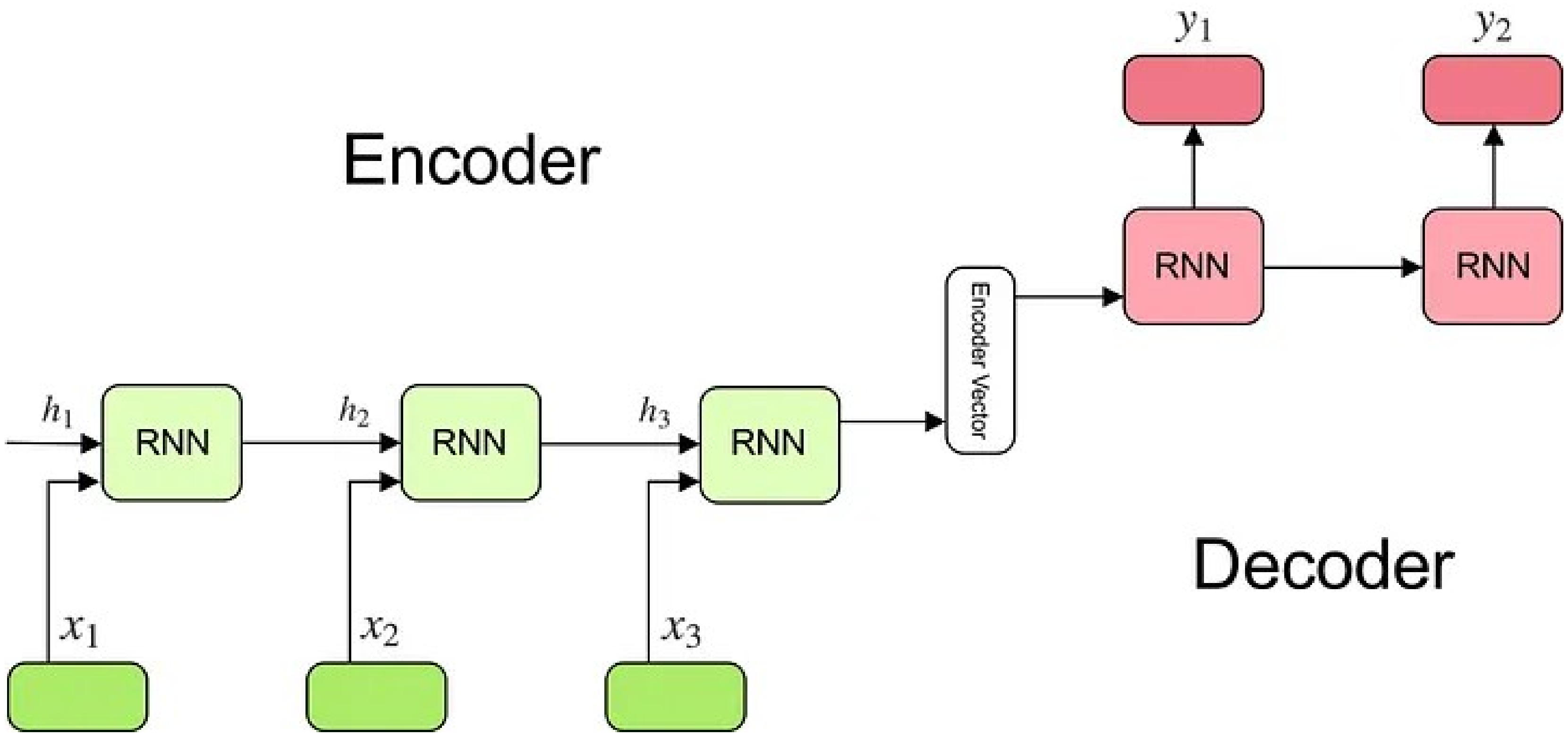
- initial state of the decoder.

→ Decoder (stack of recurrent units (RNN, LSTM or GRU))

- each unit takes a hidden state from the previous unit and produces an output and its own hidden state..

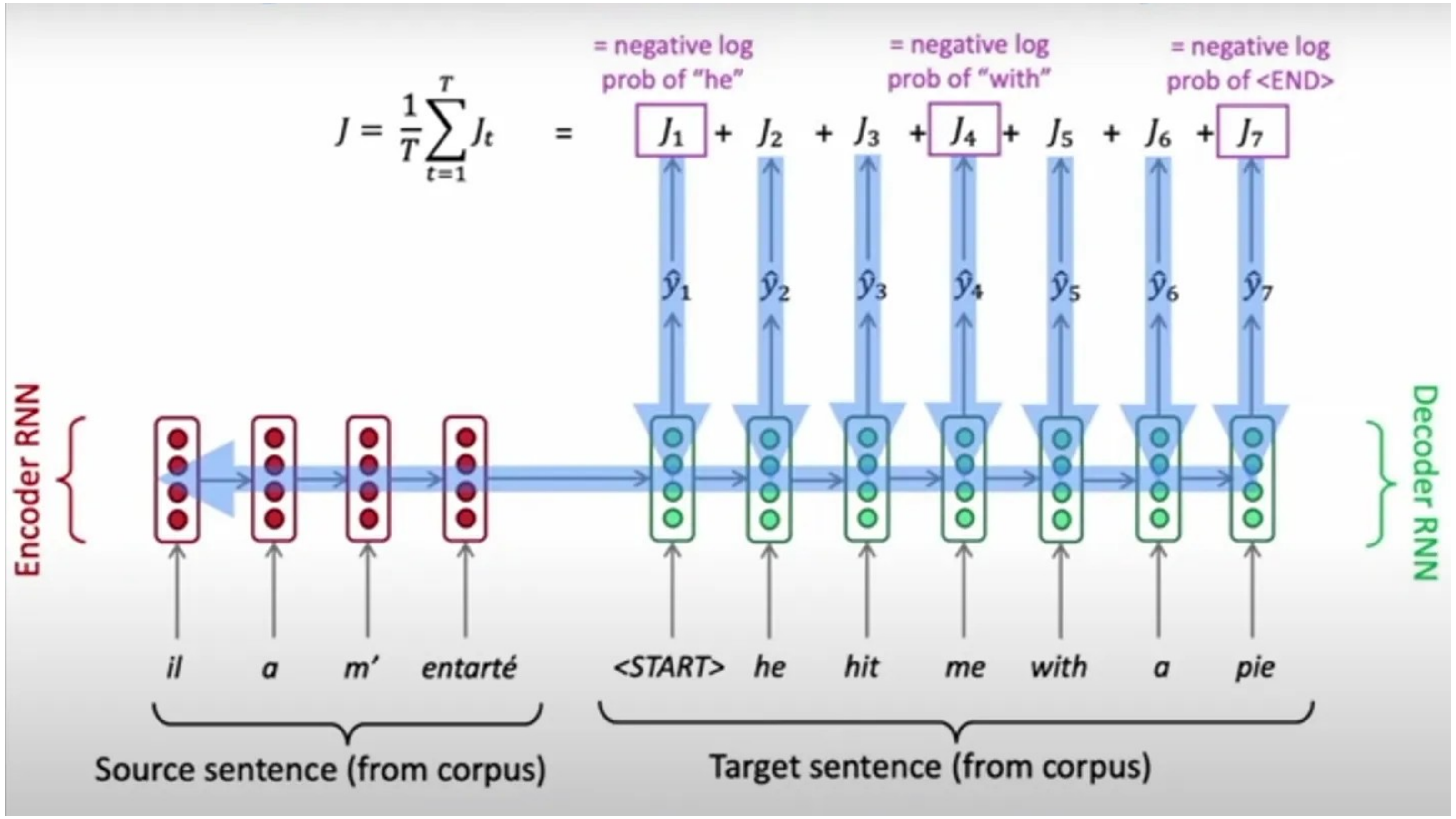
- $h'_t = f'_h(w_{hh}^T * h'_{t-1})$ and $\hat{y}_t = f_y(w_y * h_t)$

Seq2Seq architecture



3 components : Encoder, Encoder Vector, Decoder.

Seq2Seq back-propagation



Attention mechanism

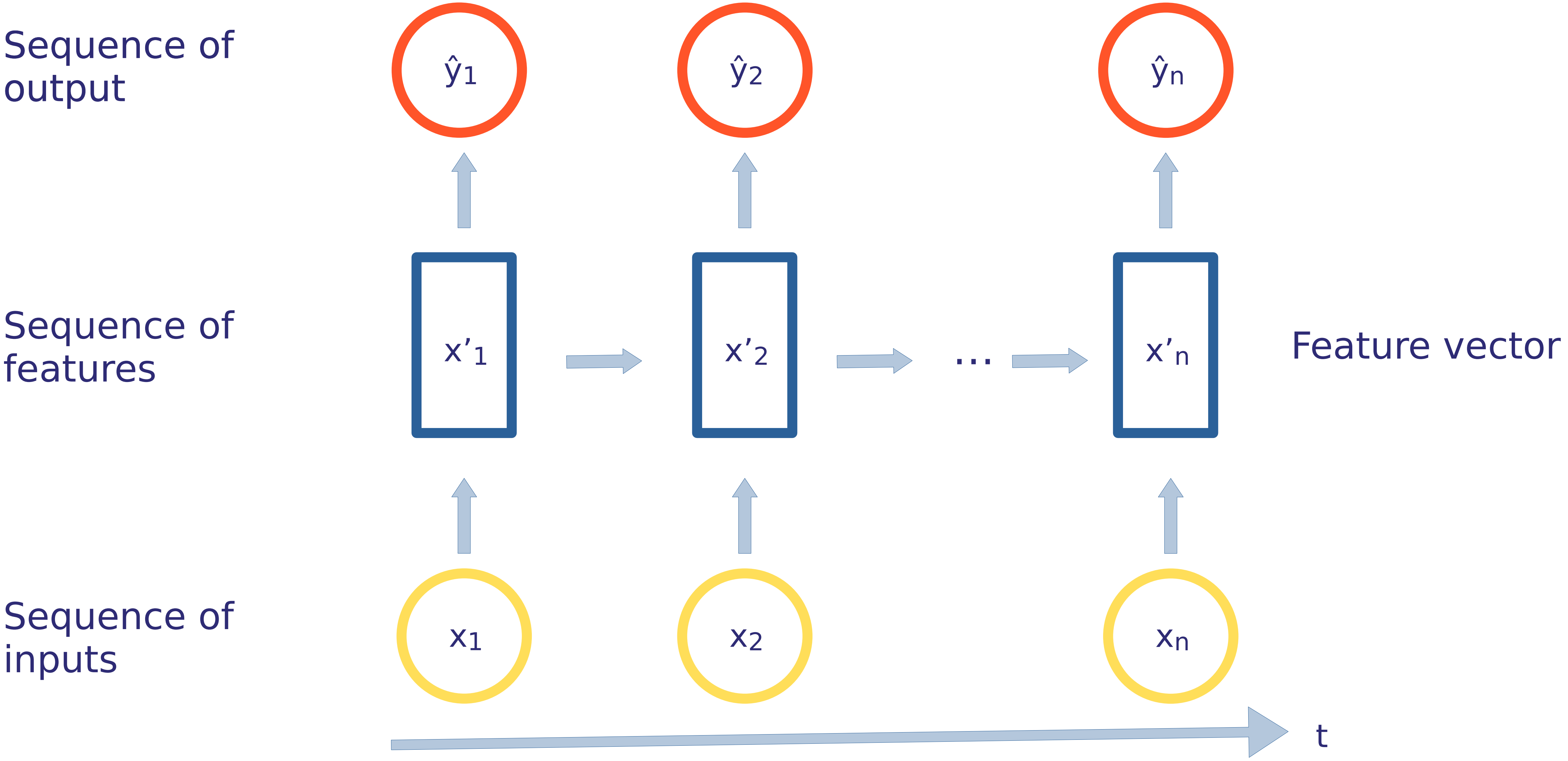
Limitations of recurrent architecture

→ Encoding bottleneck
  → Sequences are passed step-by-step
  → Hard to keep information through the pipeline
  → Loss of information in practice

→ Slow (step-by-step), no parallelization

→ Not (so) long memory

Reminder : data through the pipeline



Problem reformulation

RNN use recurrence to model sequence dependencies (with limitations).

We want :

- continuous stream
- parallelization
- long memory

Problem reformulation

RNN use recurrence to model sequence dependencies (with limitations).

We want :

- continuous stream
- parallelization
- long memory

Problems come from the step-by-step processing.
Can we **eliminate** the need for **recurrence** ?

Problem reformulation

RNN use recurrence to model sequence dependencies (with limitations).

We want :

- continuous stream
- parallelization
- long memory

Problems come from the step-by-step processing.
Can we **eliminate** the need for **recurrence** ?

Idea : Identify and focus on what is important !

Attention Is All You Need
(Vaswani et al., 2017; Bahdanau et al., 2014)

Intuition behind self-attention

Example : Identify the brands of the cars present on the image.

Intuition behind self-attention

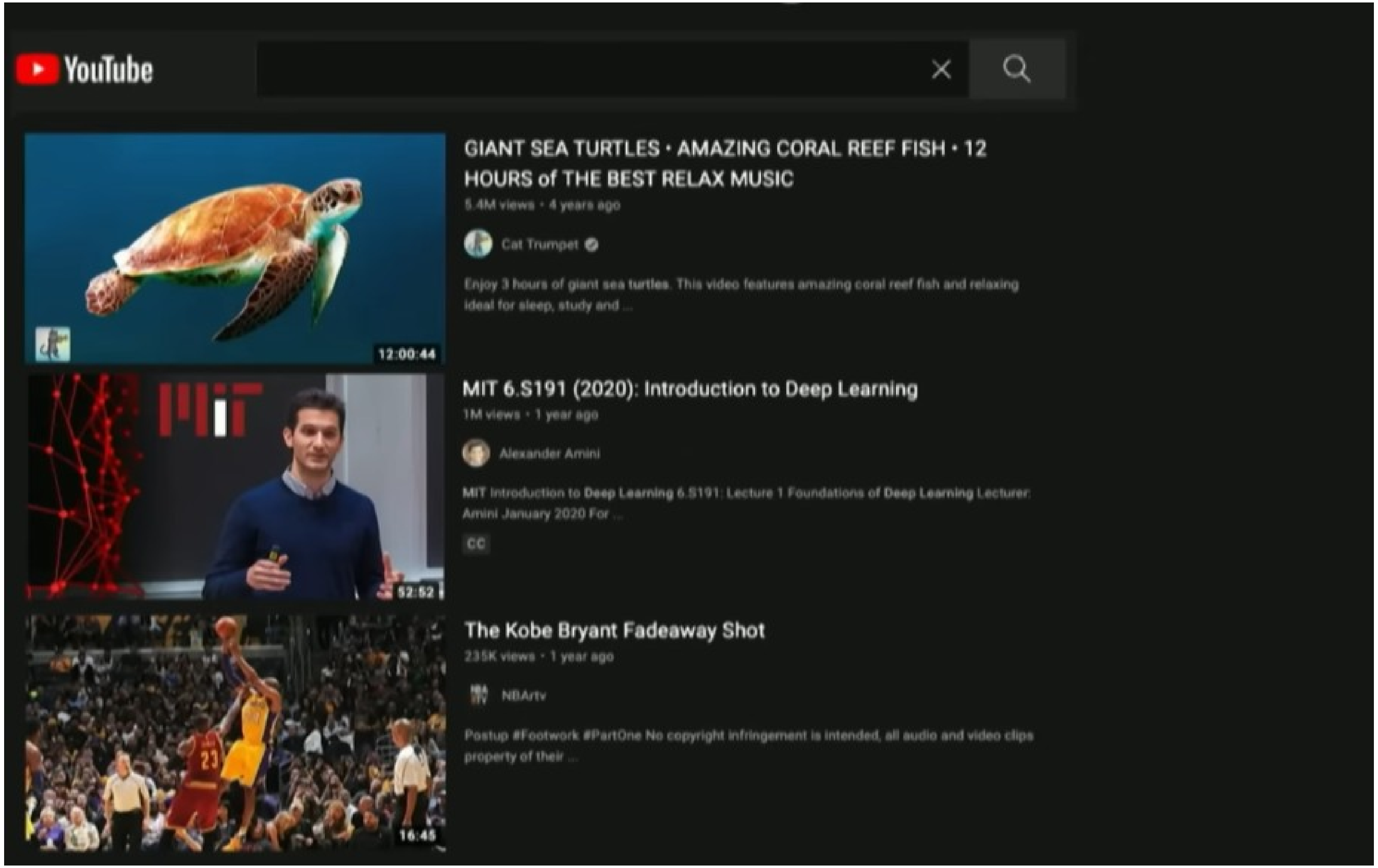
Example : Identify the brands of the cars present on the image.

1) Identify object to focus on.

2) Extract the features with high attention.

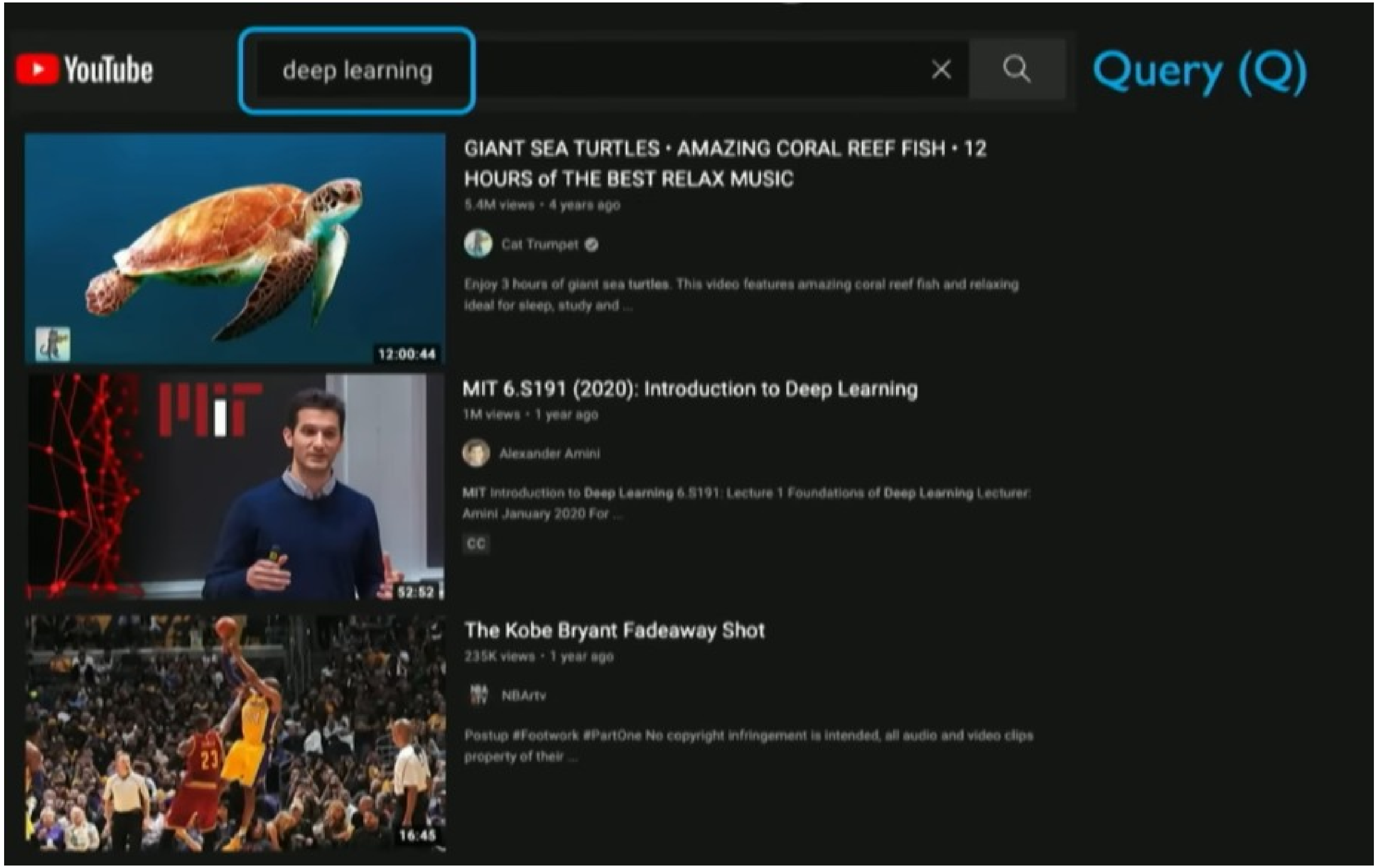
Intuition behind self-attention

Most challenging part… Similar to a search on Internet.

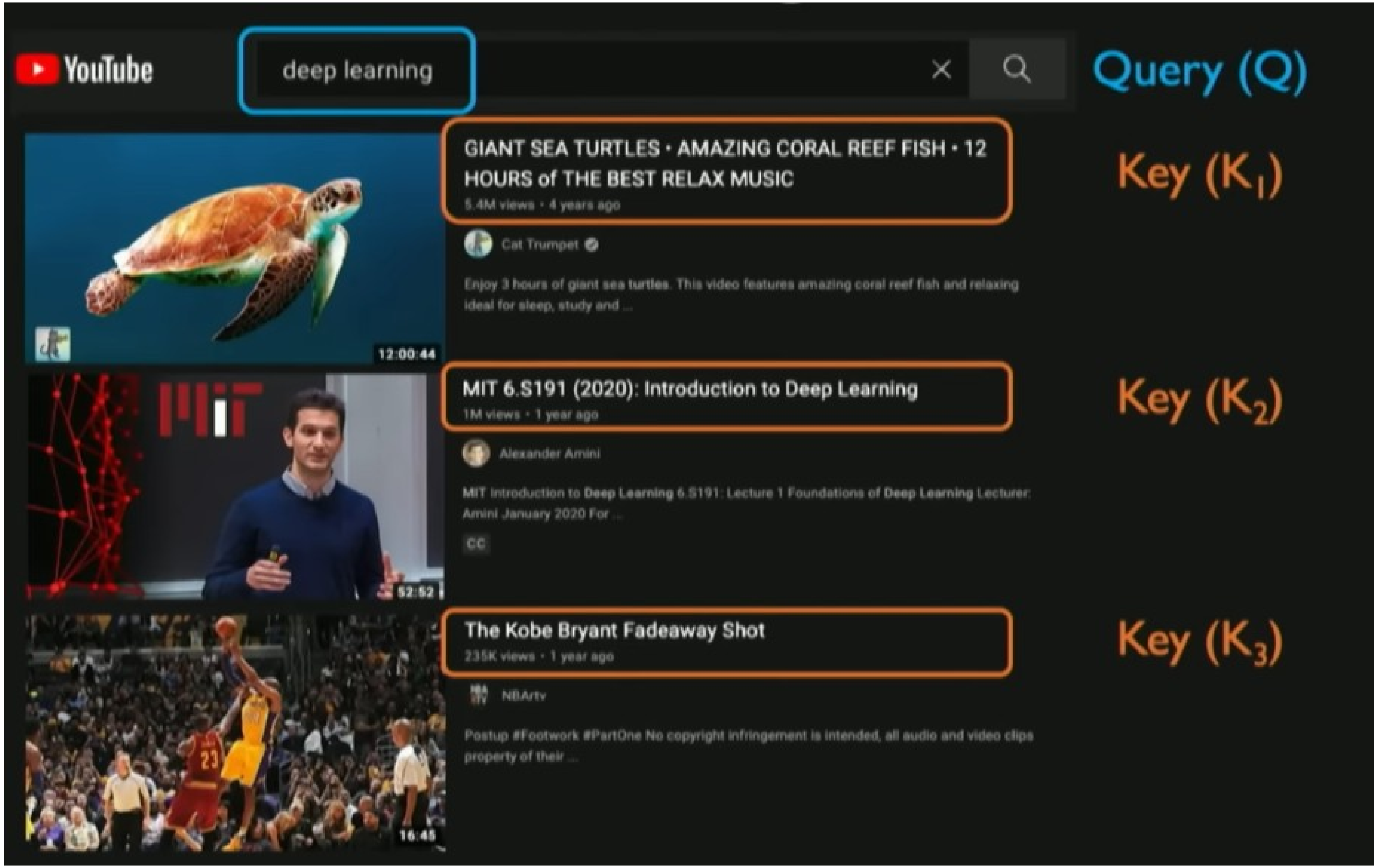


Intuition behind self-attention

Most challenging part... Similar to a search on Internet.

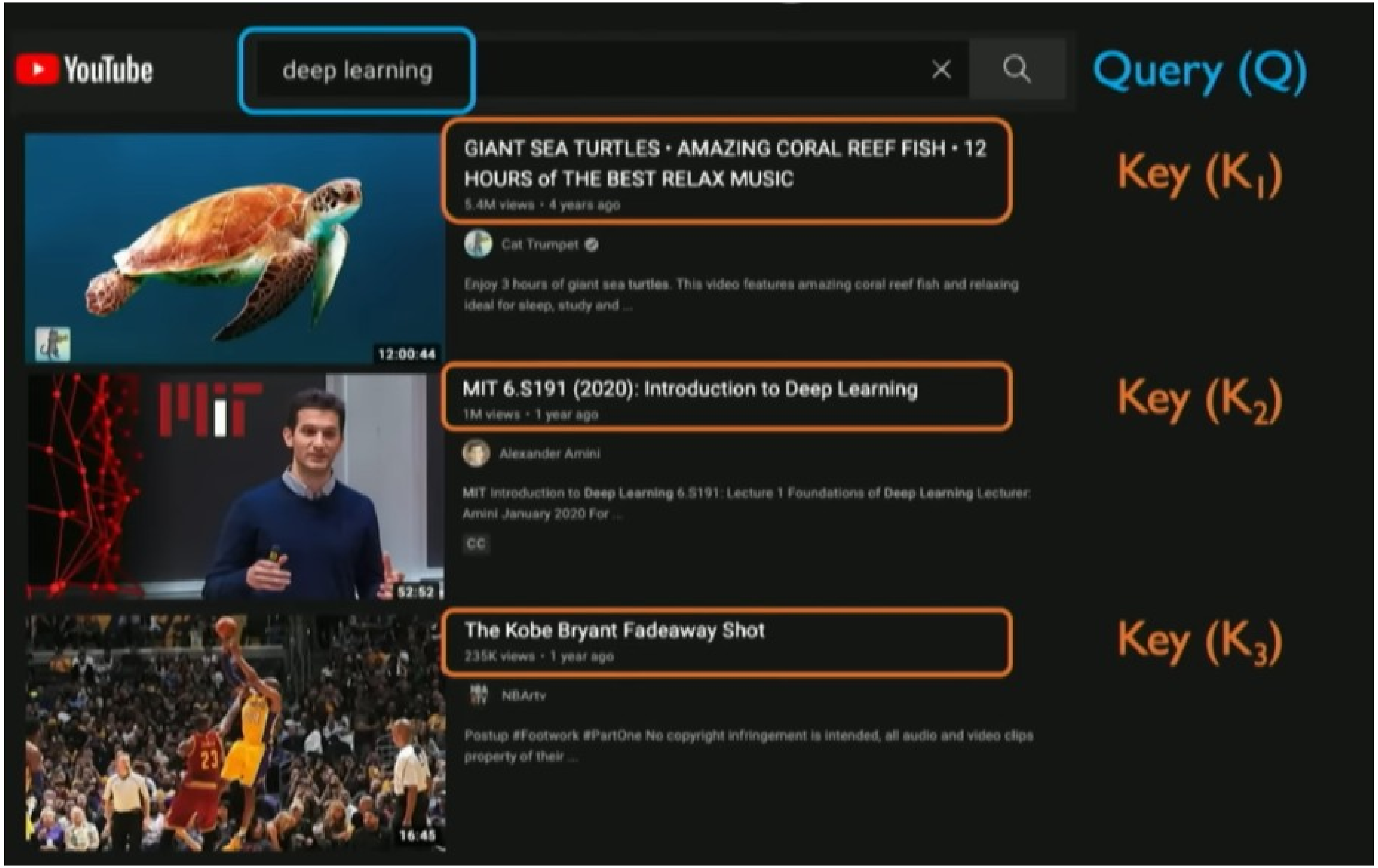


Intuition behind self-attention



→ For every videos, key information related (e.g. title)

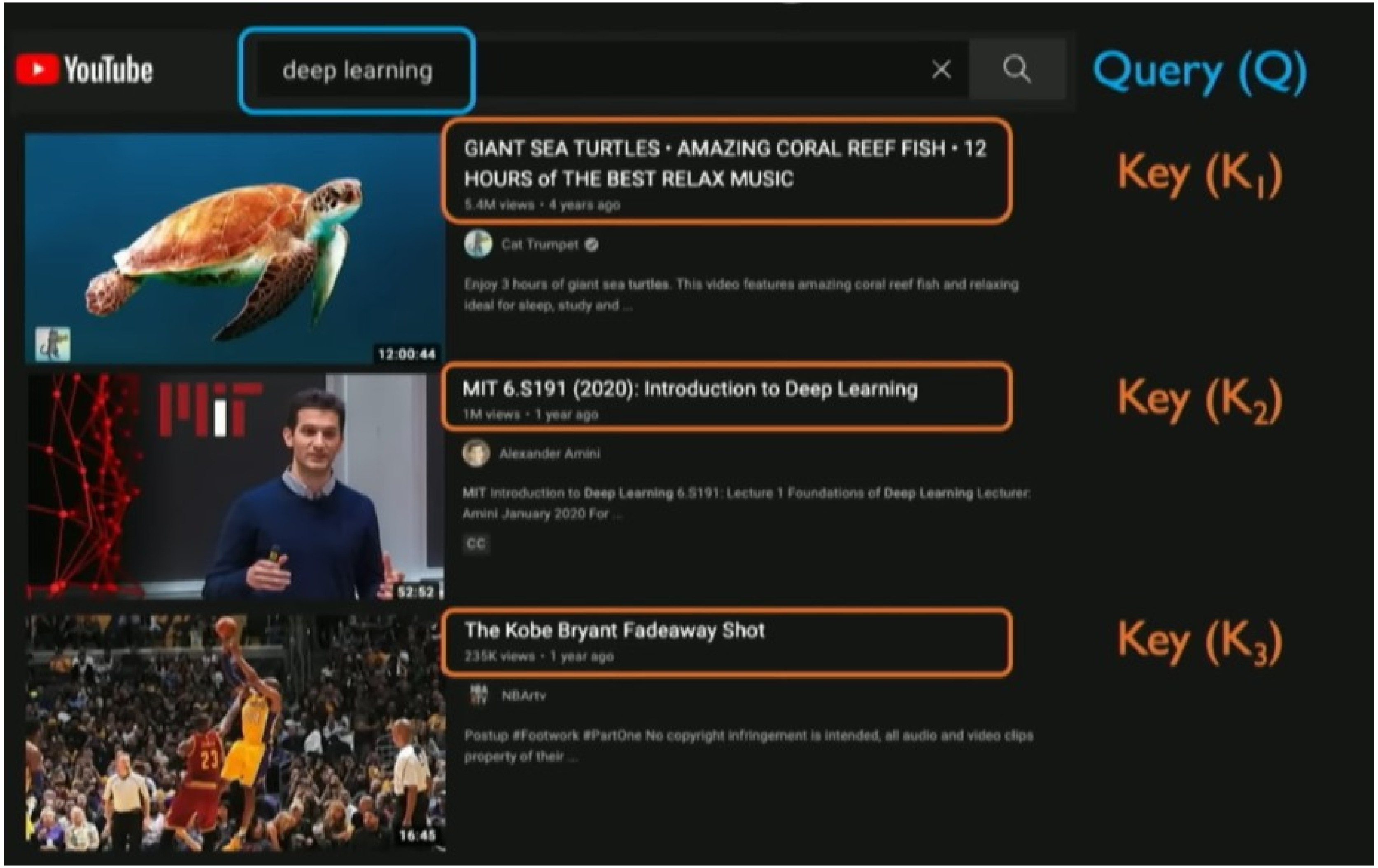
Intuition behind self-attention



→ For every videos, key information related (e.g. title)

→ We want to find the correspondence between the search (Query) and the title (Keys).

Intuition behind self-attention



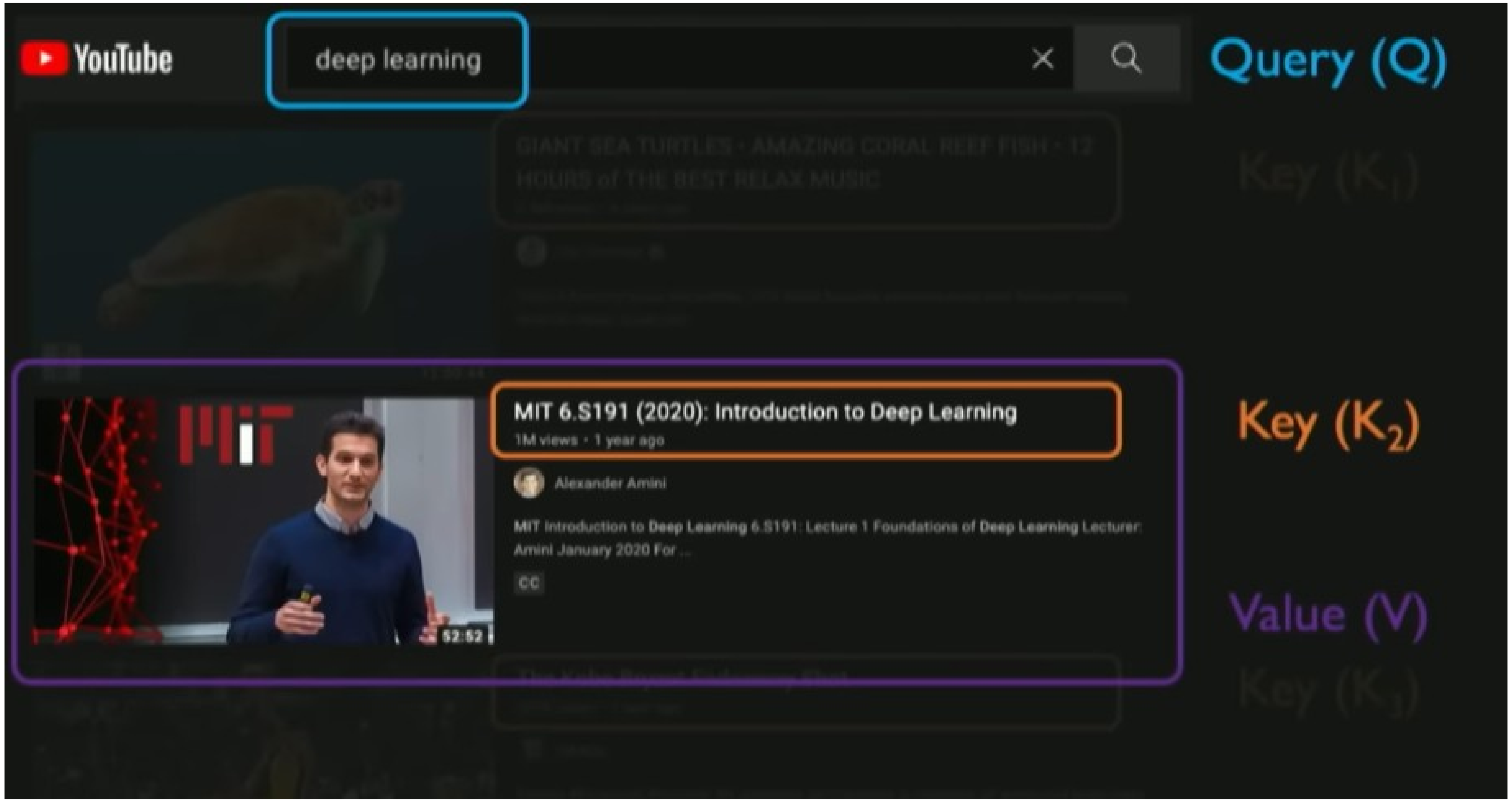
→ For every videos, key information related (e.g. title)

→ We want to find the correspondence between the search (Query) and the title (Keys).

→ Compute **metric of similarity** between Key and Query.

How similar is each Key to the Query ?

Intuition behind self-attention



→ Last step extract the relevant information.

→ Extract Values (videos).

Basis of self-attention

→ Identify and attend the most important feature in the input.

- We consider a sequence x.
- Data is feed all at once.
- We still need information on the **order**.

→ Learning self-attention with Neural Networks.

1) Encode **positional encoding** to capture the order of the sequence.

2) Extract **Query**, **Key**, **Value**.

3) Compute the **attention weighting**.

4) Extract features with **high attention**.

1) Positional Encoding

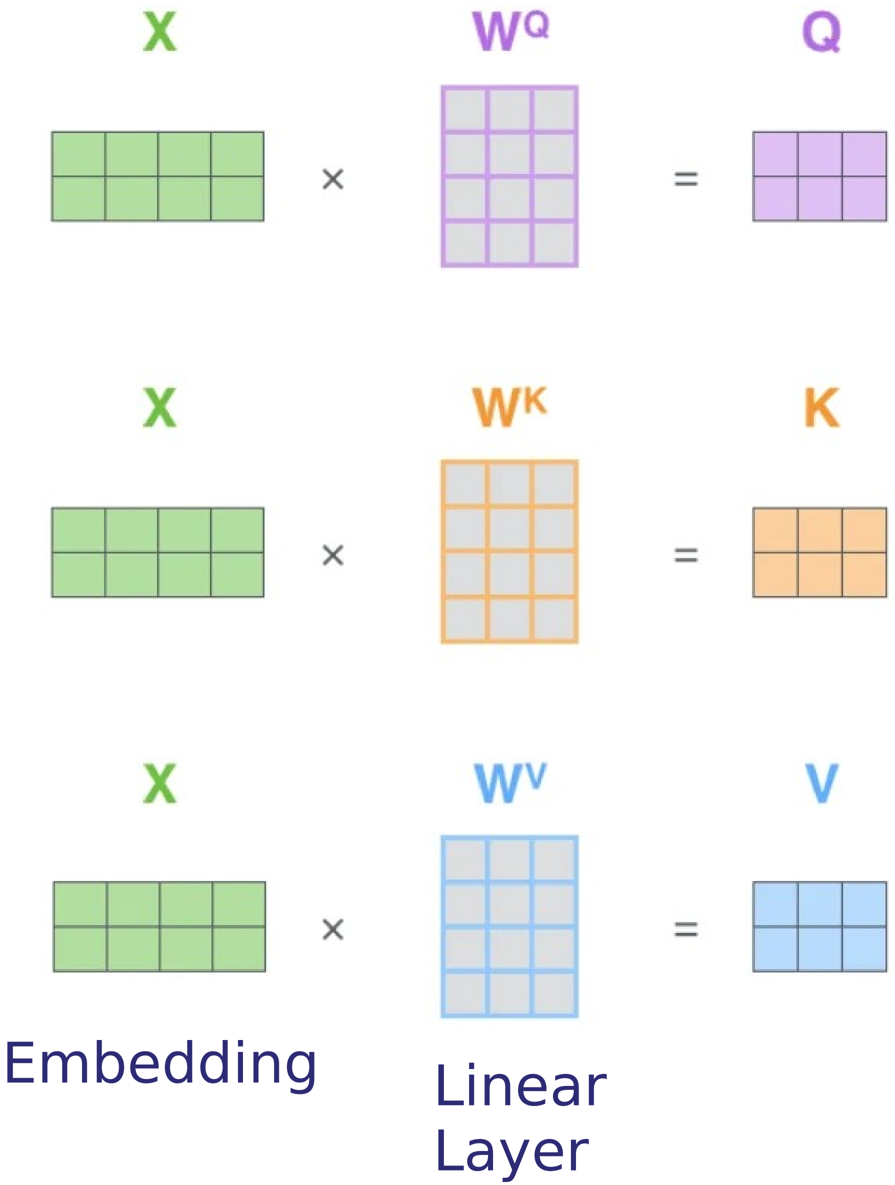
Data is feed all **at once**.
Need information on the **order**.

Positional Encoding gives information on the order of the words directly in the embedding.

Input sequence	I	am	a	Robot
Word embedding	v_0 = embedding vector(I)	v_1 = embedding vector(am)	v_2 = embedding vector(a)	v_3 = embedding vector(Robot)
		+		
Positional Encoding Matrix	P_0 = Positional vector(I)	P_1 = Positional vector(am)	P_2 = Positional vector(a)	P_3 = Positional vector(Robot)
		=		
Output of positional encoding layer	y_0 = Positional encoding(I)	y_1 = Positional encoding(am)	y_2 = Positional encoding(a)	y_3 = Positional encoding(Robot)

2) Extract Key, Query, Values

Using Neural Network layers, we compute the Key, Query and Value matrices.



3) Compute the attention weighting

→ Attention scores : compute the pairwise similarity between each Key and Query.

→ $\dfrac{Q \cdot K^T}{\text{scaling}}$ (scaled dot product)

→ Equivalent to the cosine similarity.

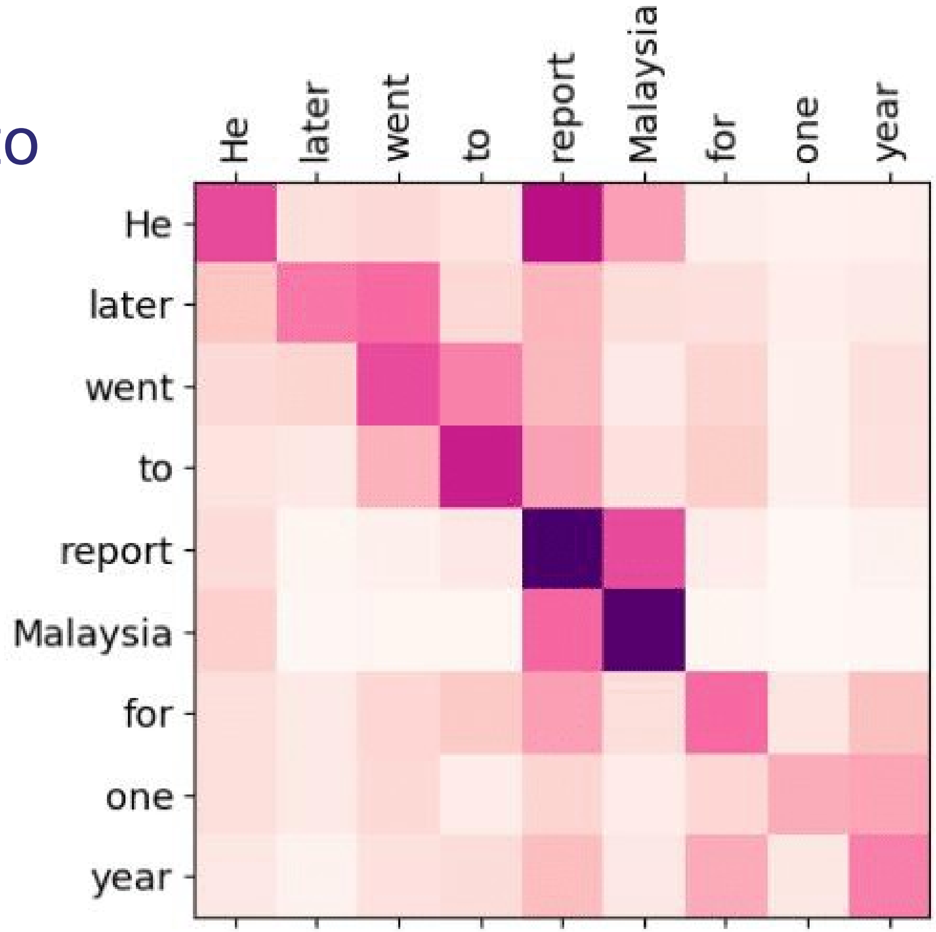
→ In other words, Q and K being vectors, are they going in the same direction?

4) Extract features with highest attention

→ Attention matrix gives indications on where to find the related information (how components are related to each others).

→ Features with highest attention :

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Attention mechanism sum-up

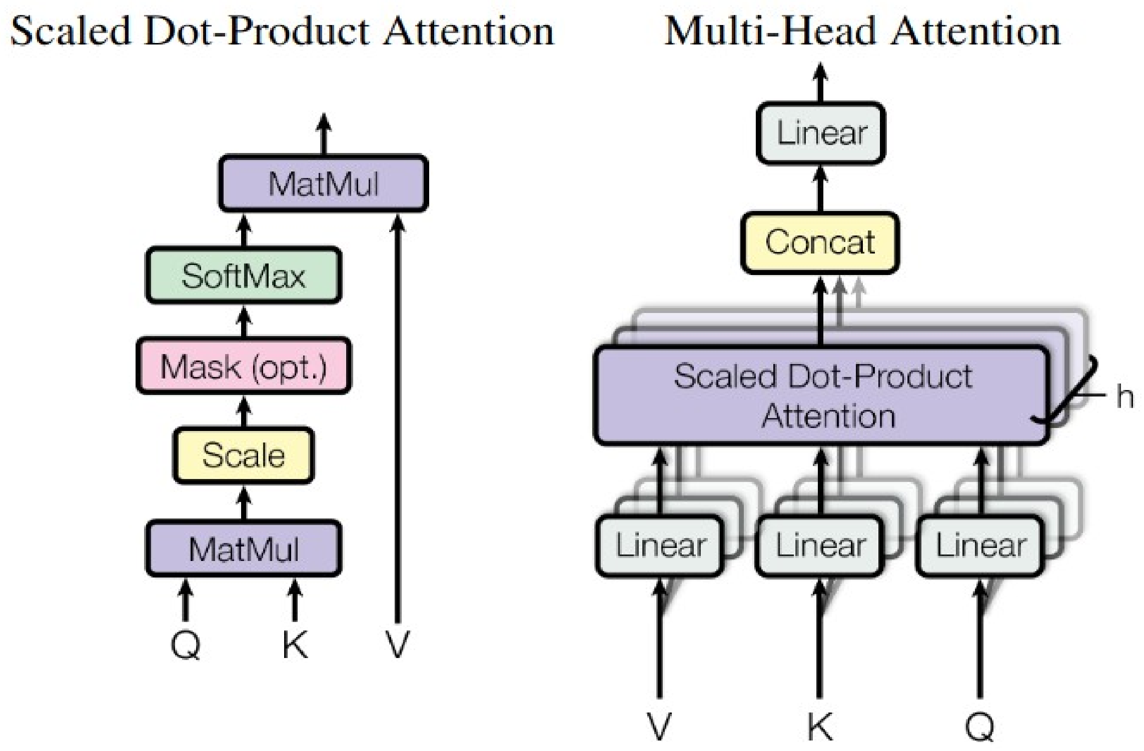
Objective: Identify and attend to the most important features in the input.

Input data → Positional Encoding
→ Embedding
→ Key, Query, Values
→ Compute self-attention scores.
→ Extract representations of the data where we focus on important information.

Attention based architecture

So far, single self-attention head, multiple ones can be layered together to build larger NN.

Each head extracts different information to get a rich representation of the data ! (grammar, semantic, meaning...)

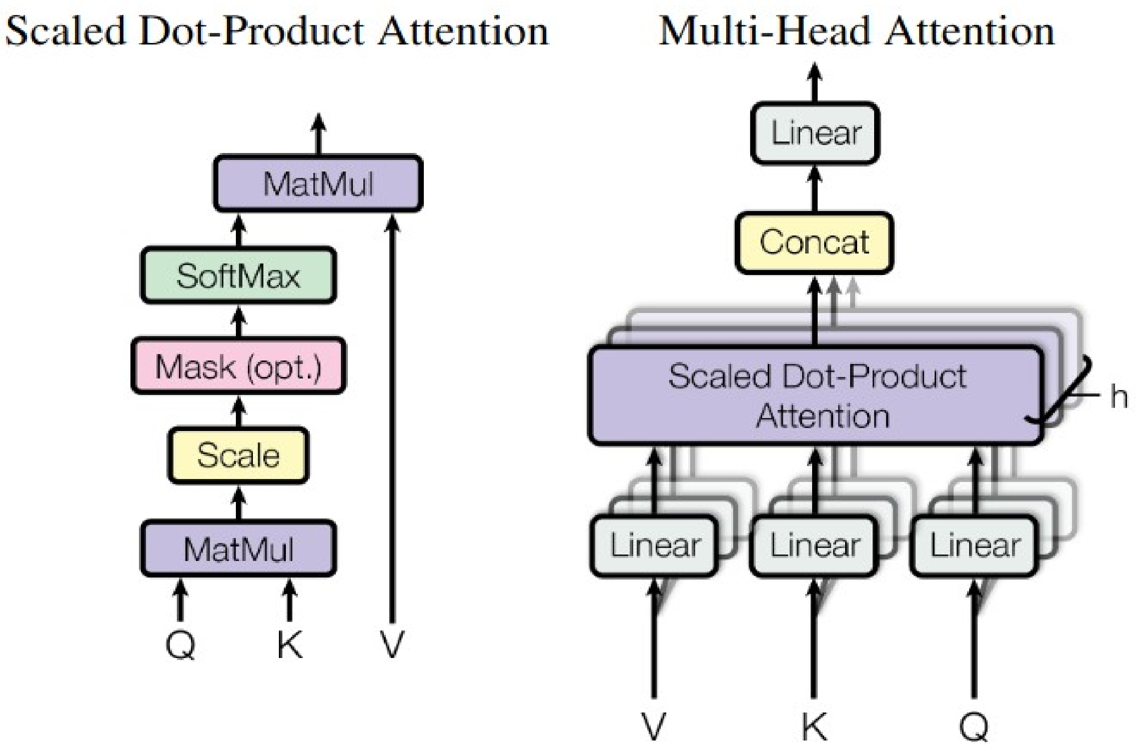


Attention based architecture

So far, single self-attention head, multiple ones can be layered together to build larger NN.

Each head extracts different information to get a rich representation of the data ! (grammar, semantic, meaning...)

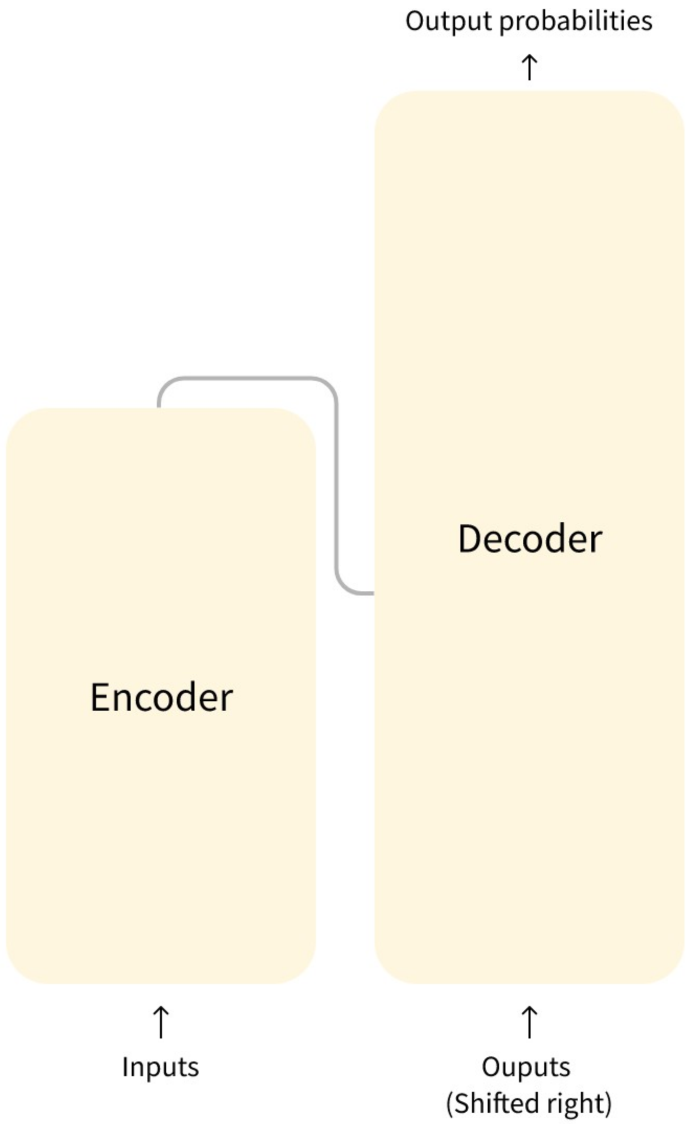
Transformers !



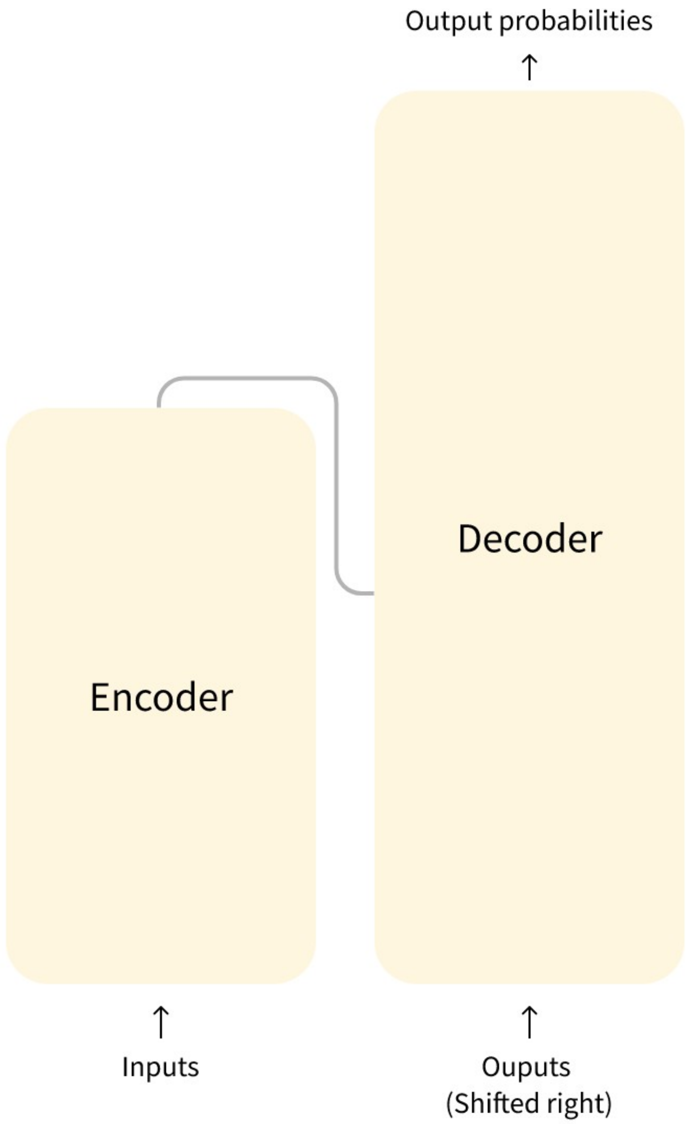
Transformers

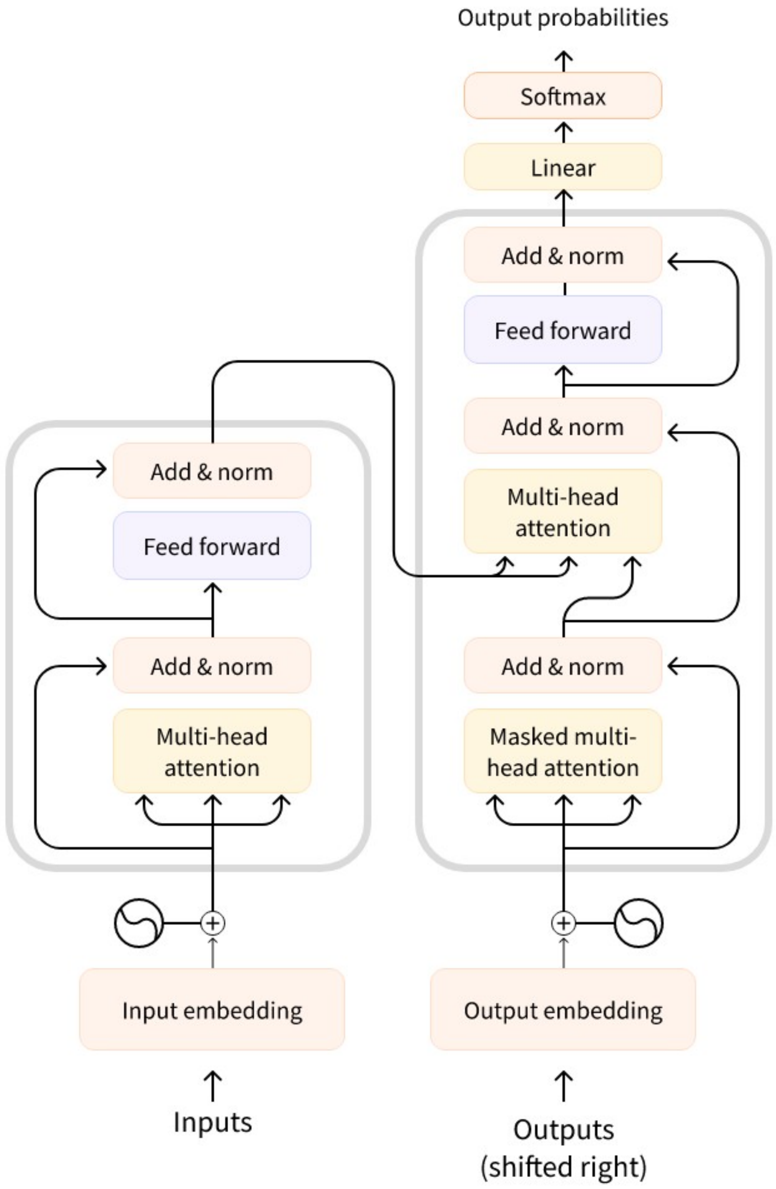
State-of-the-art (for now...)

Transformer architecture



Transformer architecture





Transformer-based models (for NLP)

→ Encoder only

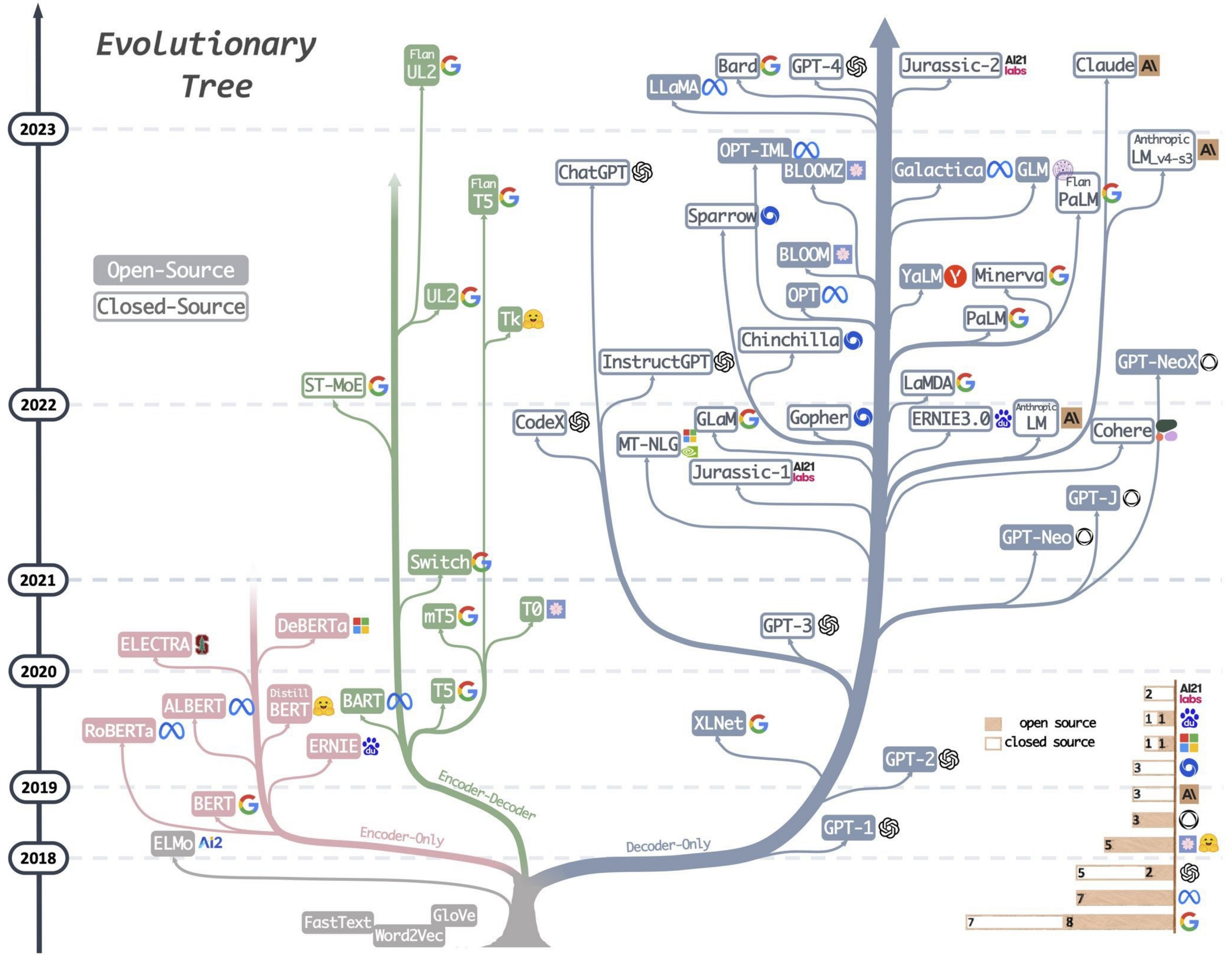
- input oriented tasks
- text classification
- entity recognition

→ Decoder only

- generative tasks
- text generation

→ Encoder-Decoder

- generative tasks requiring knowledge on the input
- translation
- summarization



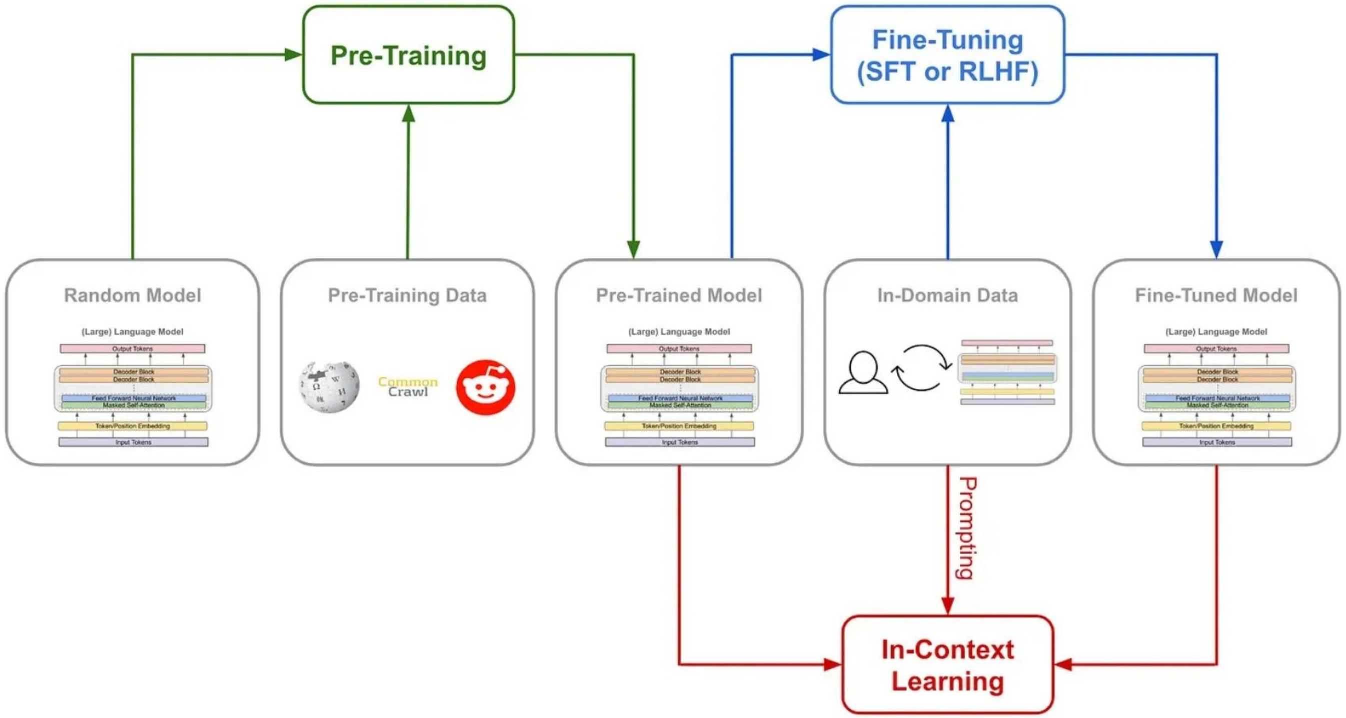
Pre-trained models

Life-cycle of LLMs

- Pre-training
  → learn global 'understanding' of the language

- Tuning
  → further training to fit a task

- Deployment



Example of Transformers in NLP, the Large Language Models

→ Pre-training

- Masked Language Modeling (MLM)

  Some tokens are randomly masked and the model must predict them.
  Helps to learn representations (e.g. Encoder-based models).

- Causal Language Modeling (CLM) or Next Word Prediction

  Predicts the next word in a sequence given the previous words.
  Good for generative models.

- Contrastive Learning

  Distinguish between similar and dissimilar sentences.
  Helps to learn representations.

Example of Transformers in NLP, the Large Language Models

→ Tuning

- Full fine-tuning (best performance)

All the models' parameters are updated for a task (e.g. classification).

- Parameter-Efficient fine-tuning (most efficient)

  - Adapters (small sub-network are trained and added to the model).
  - Low-Rank Adapters (weights matrices are trained and added to the model).
  - Prefix-tuning (special prefix embeddings are learned and added to the embeddings, *no weights changes*). -
  - Prompt-tuning (learnable prompts are add to inputs, *no weights changes*).

Example of Transformers in NLP, the Large Language Models

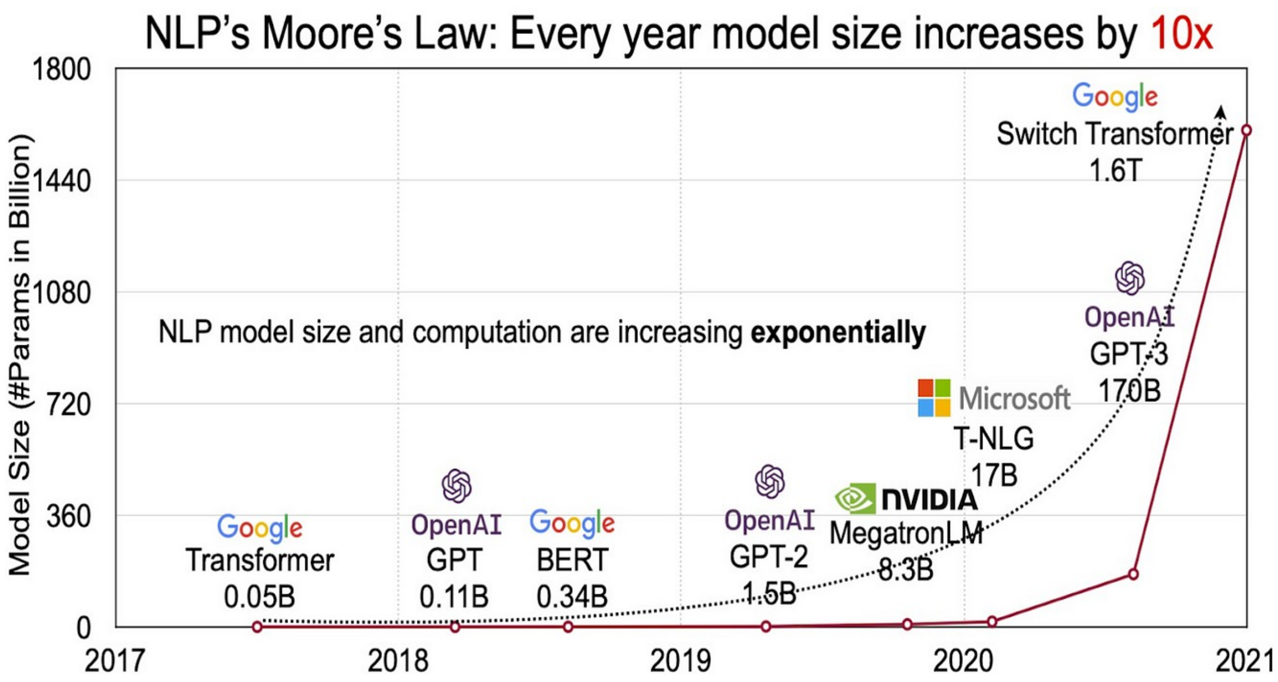
→ Tuning

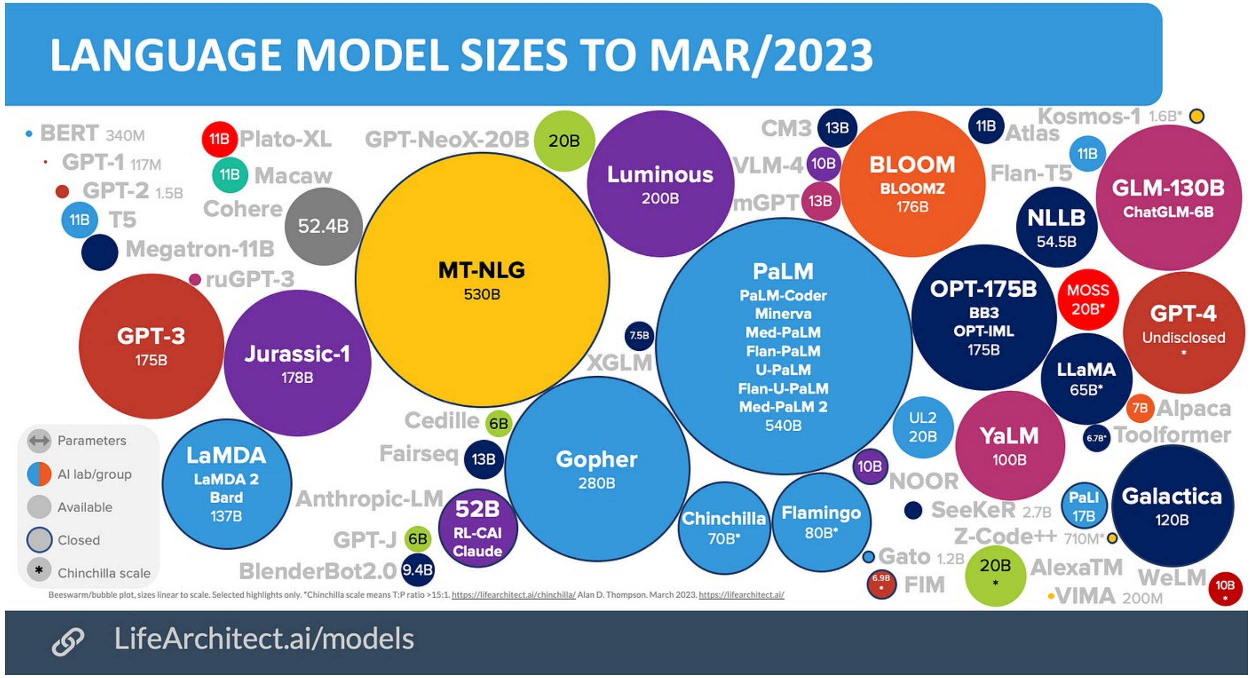
- Reinforcement Learning (best alignment with human preferences) Use of **human feedback** to improve the model.

    - Proximal Policy Optimization (use a reward model to score the output of the model and upgrade its weights).

    - Direct Policy Optimization (use pairs of sentences (accepted-rejected) and train the model to prefer the ‘accepted sentence’).

Large language models





Models are larger and larger → New challenges...

- deployment on small devices
- data privacy
- intellectual property of the training data

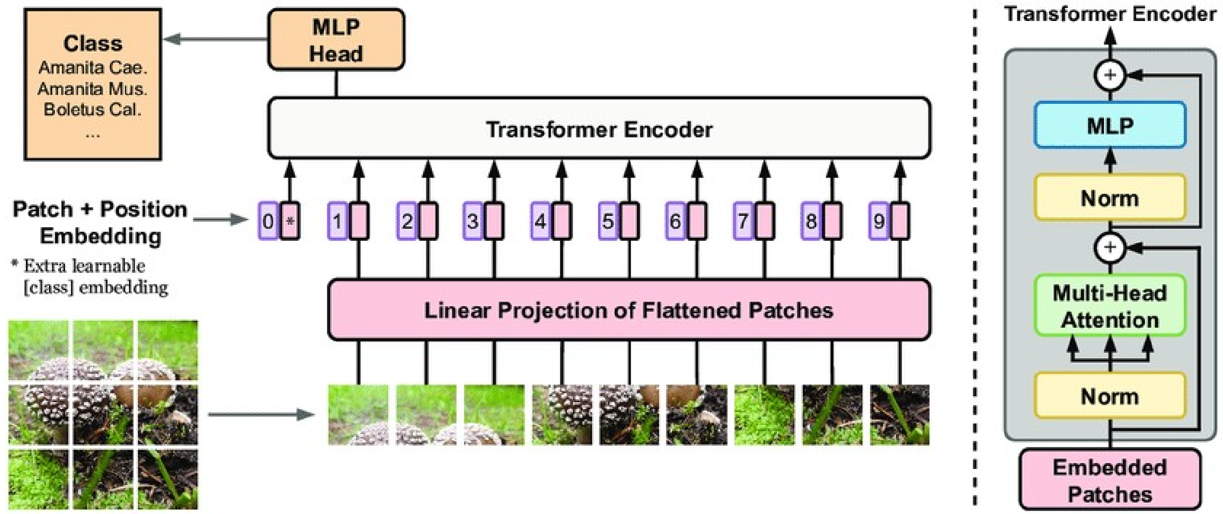
Transformers, other applications

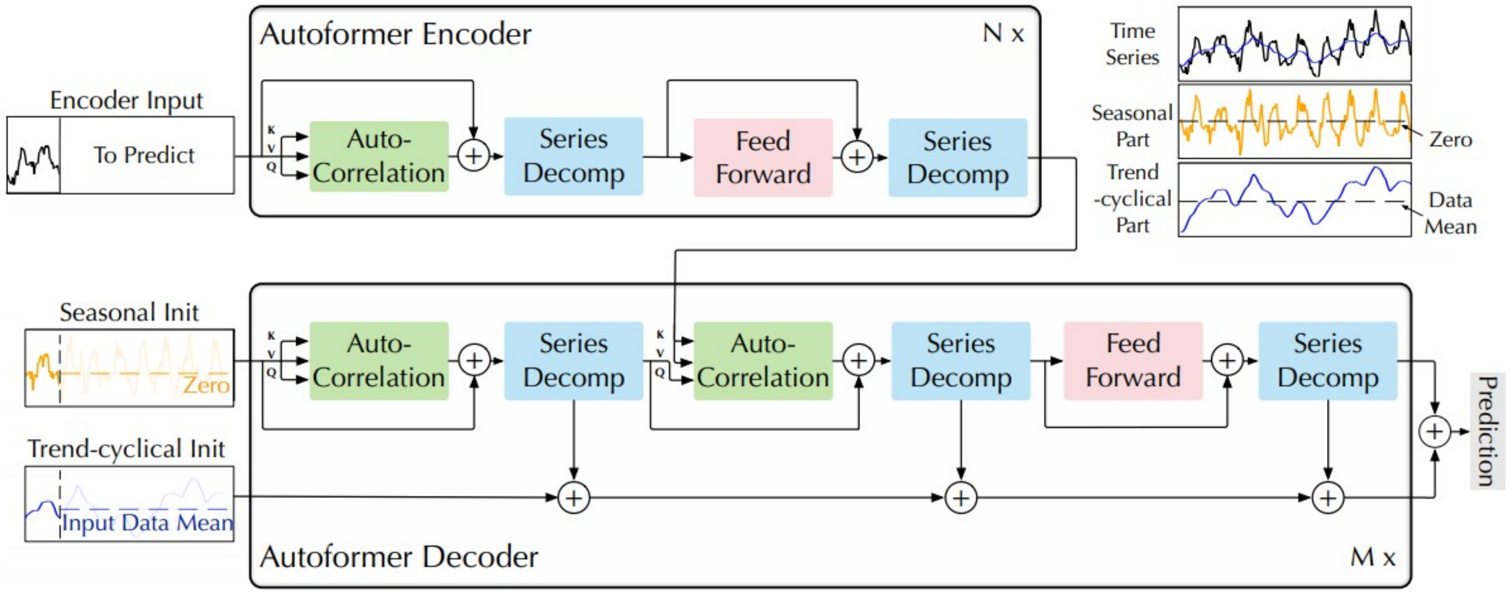
→ Biological sequences (e.g. AlphaFold2 (Jumper et al., 2021))

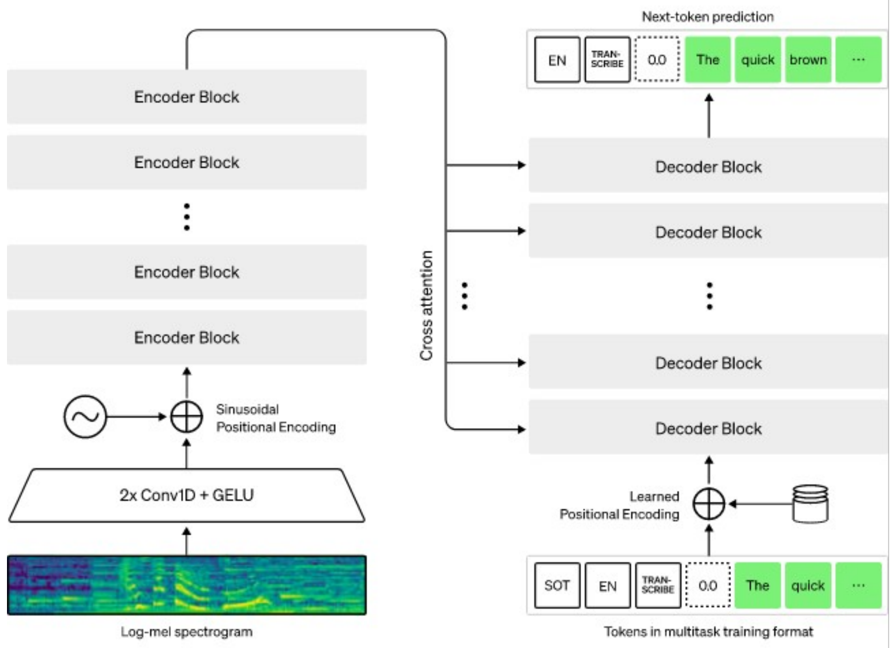
→ Images (e.g. Vision Transformer (ViT) (Dosovitskiy et al. 2020))

→ Audio (e.g. Whisper (Radford et al., 2022) for Speech-to-text)

→ Time series







Next : Practical Session

Implementation of a Seq2Seq model.