Natural Language Processing

AIMS

Romain Benassi

Course 4

Spring 2023

Course Schedule

- **Course 1**: NLP introduction
- **Course 2**: Word embedding
- **Course 3**: Long Short-Term Memory (LSTM) architecture
- **Course 4**: “Attention” mechanism and Transformer architectures
- **Course 5**: Large Language Models and Generative AI

Evaluation

1. **A graded exam will be used as evaluation and will be done at the beginning of the last course (Course 5)**

This exercise will contain

- multiple-choice-questions (MCQ)
- theoretical questions
- Coding questions

2. **A graded project**

Real life dataset for the Graded Project

Data related to a Data for Good project

- **Data For Good** is a **non-profit** association whose point is to use data for the common good
- The data corresponds to **article headlines** (in French)
- The point is to **recognize automatically** those related to **climate**, in order to **challenge the media on their coverage of the subject**
- Solving this (open) use-case corresponds to **Question 4** of the **Graded Project**

Course Schedule

- **Course 1**: NLP introduction
- **Course 2**: Word embedding
- **Course 3**: Long Short-Term Memory (LSTM) architecture
- **Course 4**: “Attention” mechanism and Transformer architectures
- **Course 5**: Large Language Models and Generative AI

Course Schedule

- **Course 1**: NLP introduction
- **Course 2**: Word embedding
- **Course 3**: Long Short Term Memory (LSTM) architecture
- **Course 4**: **"Attention" mechanism and Transformer architectures**
  - Encoder-decoder
  - Attention Mechanism
  - Transformer architectures
  - BERT model
  - Hugging Face

This Week

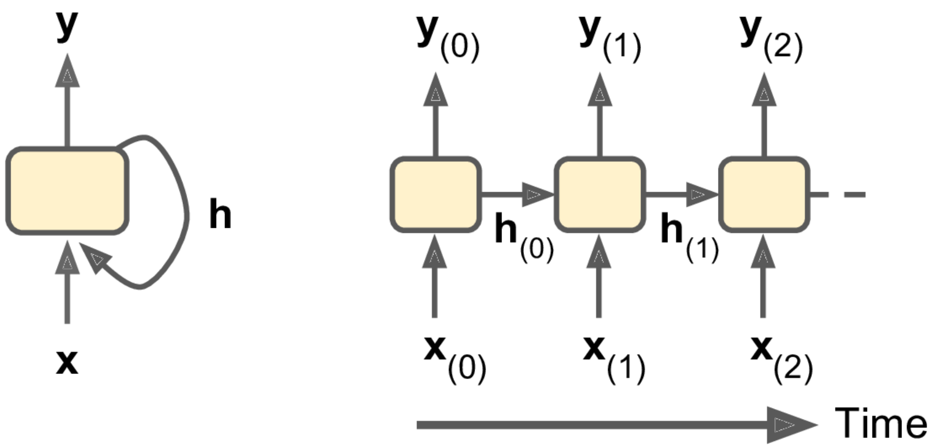
- **Course 5**: Large Language Models and Generative AI

Previously in Course 3

Recurrent Neural Network (RNN): Definition

- A **RNN** is a kind of neural network dealing with **recurrent** connection
- Allows to deal with **temporal sequences**
- E.g., on the figure below, a sequence **X** is given as **input**, and we get a sequence **Y** as **output**
- **A cell hidden state h and the output y may be different**

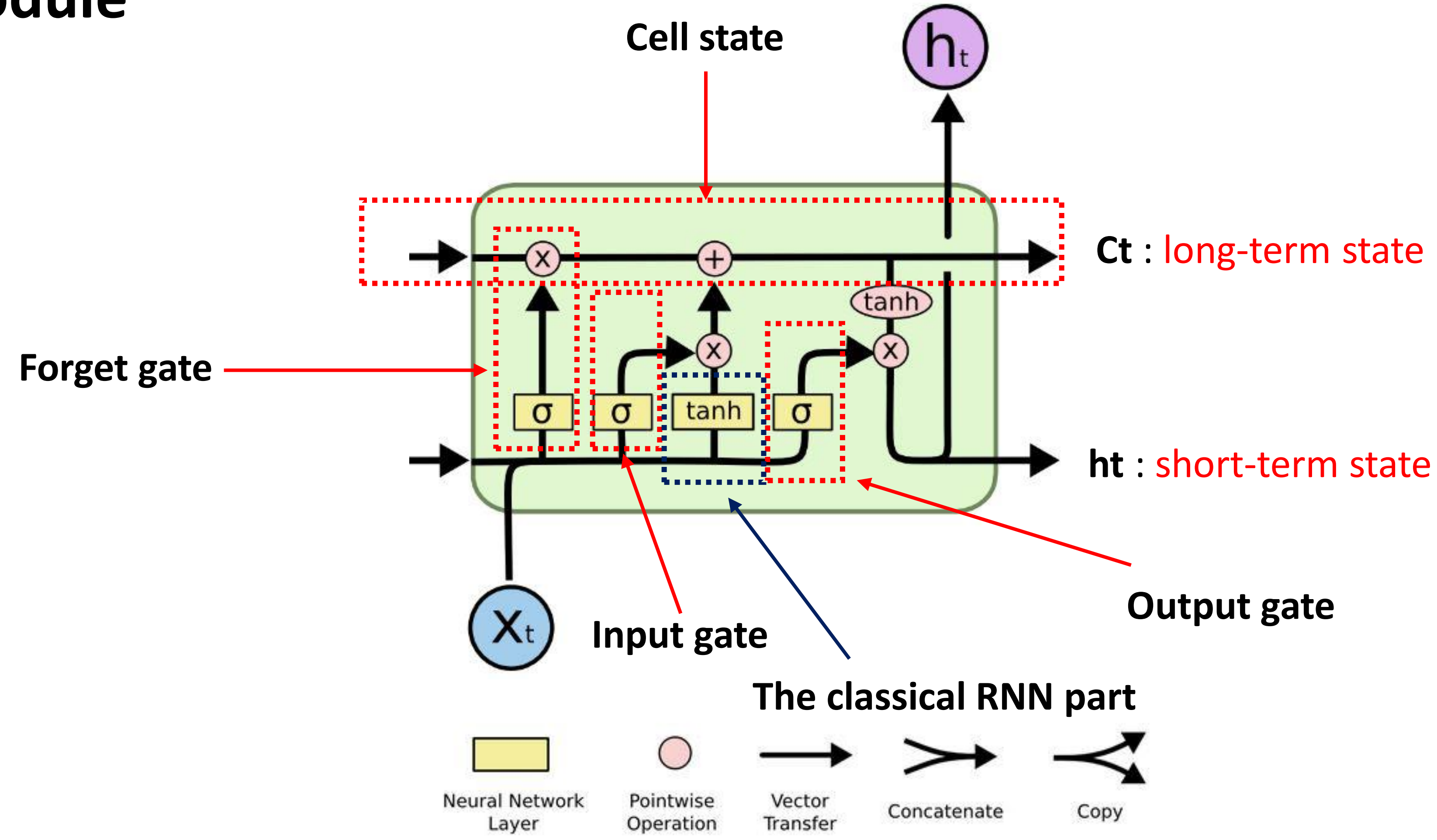
A hidden state RNN



Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

LSTM (Long Short Term Memory)

LSTM module

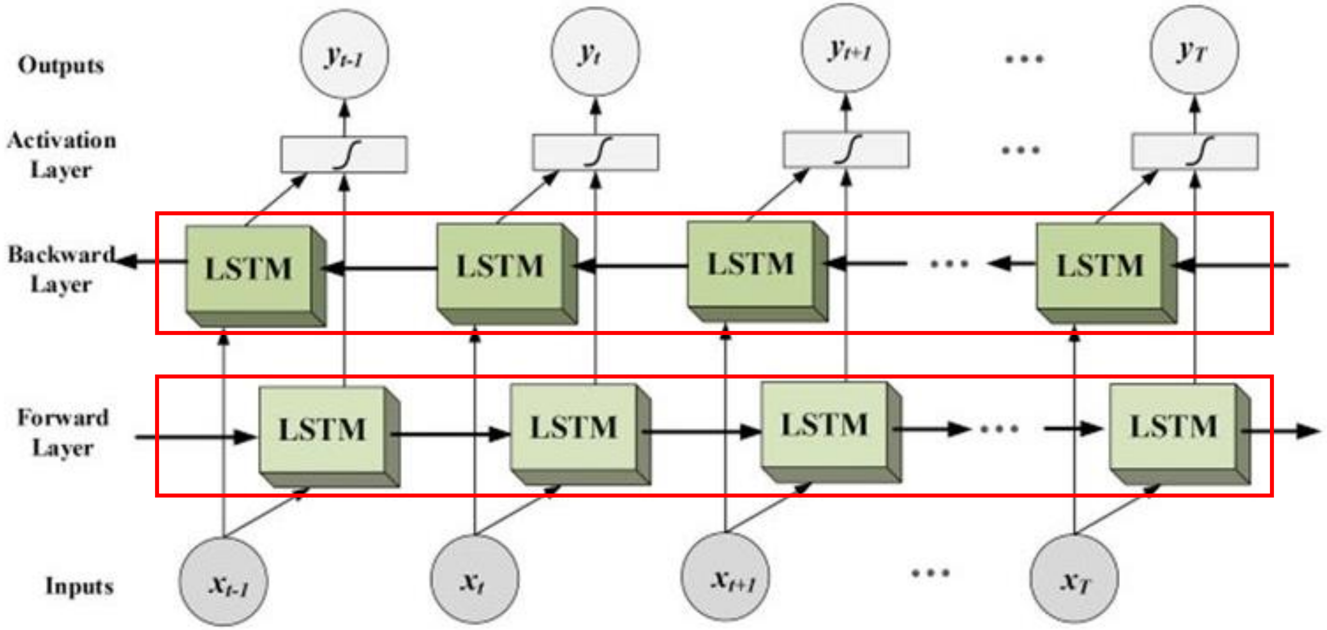


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Bidirectional LSTM

In a **classic LSTM** architecture, the sentence/sequence is processed only in **one direction** (generally from the past to the present or future)

The **bidirectional** structure process the data in **both directions** => can be useful for some applications



https://medium.com/the-official-integrate-ai-blog/what-you-need-to-know-about-natural-language-processing-2c8240e6c38e

The Queen of the United Kingdom

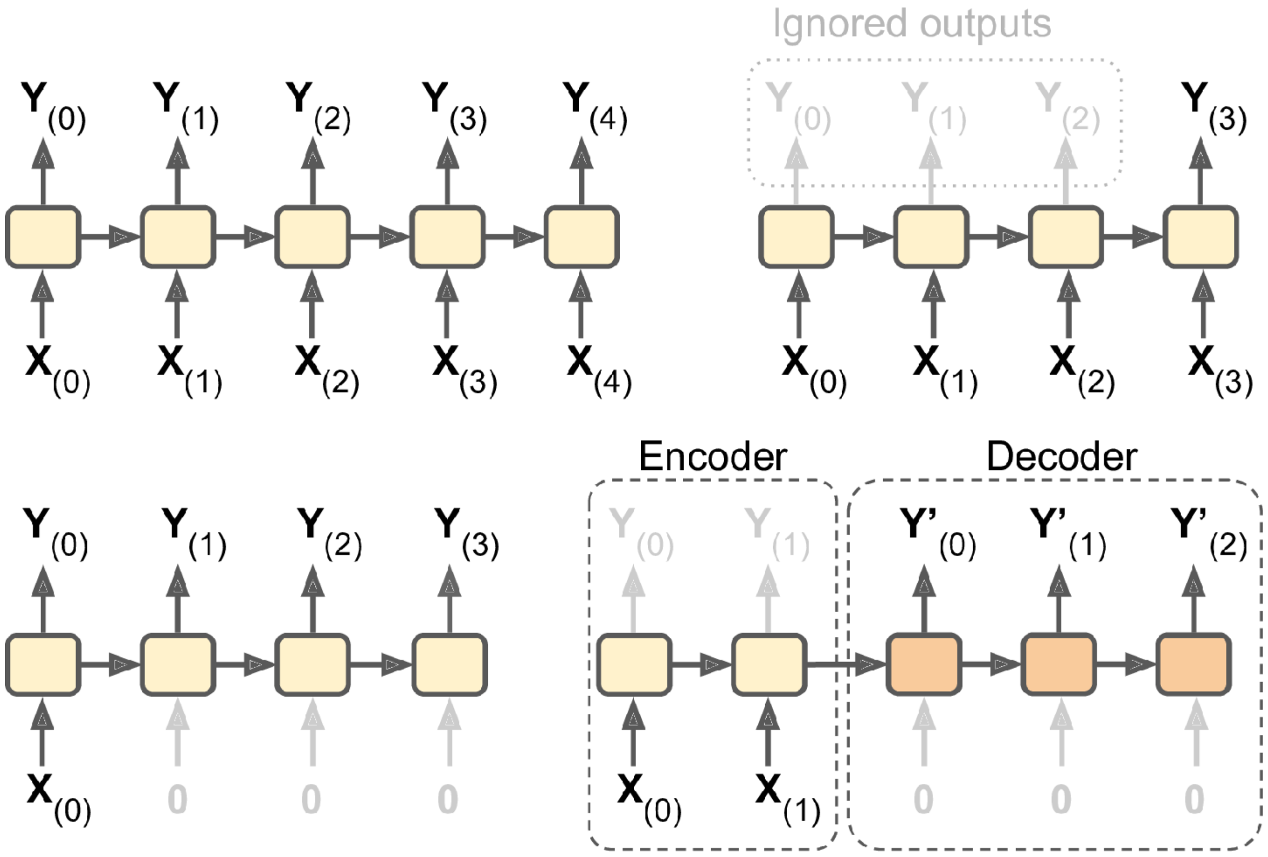
The queen of hearts

The queen of bees

Here, the word queen will be encode differently
if read in reverse order

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Use cases

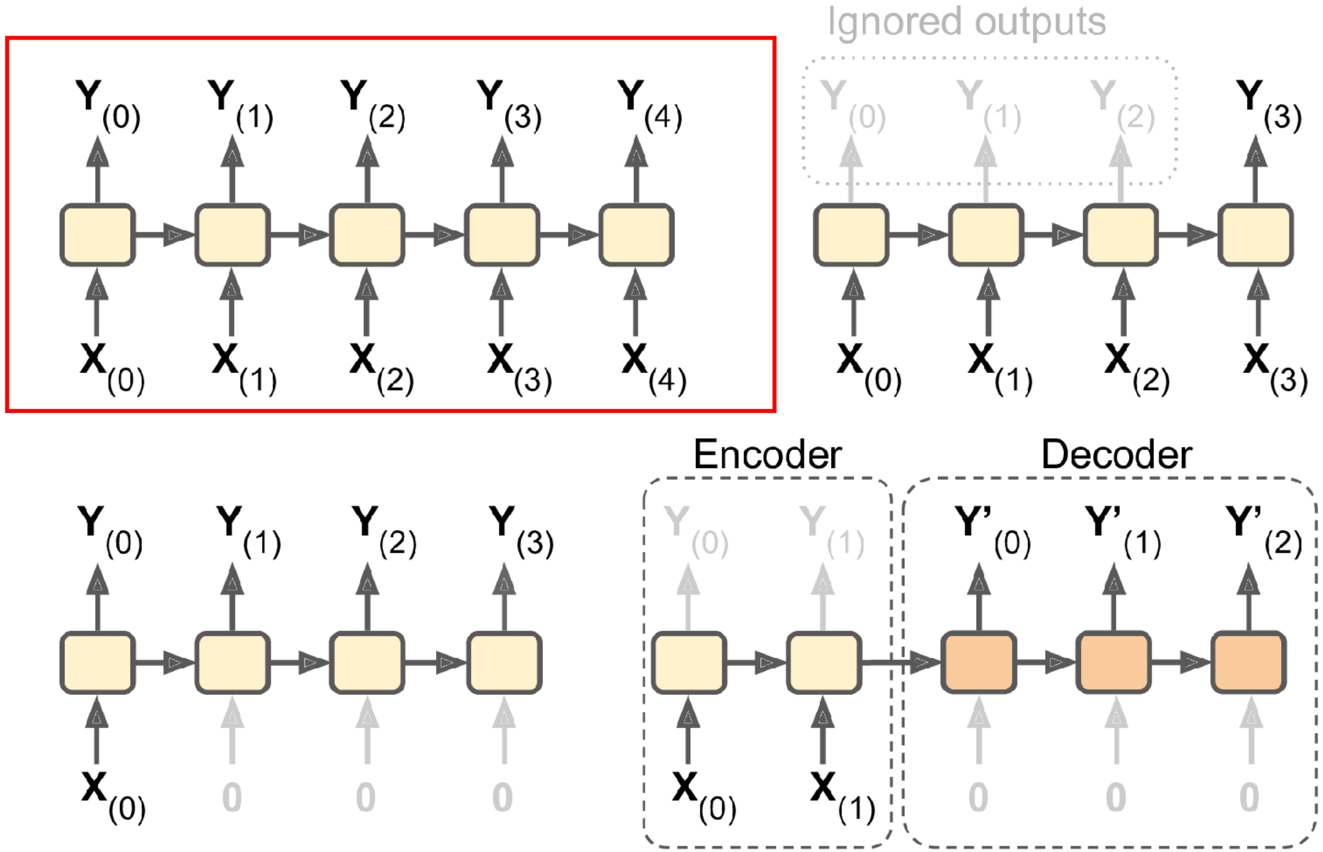


Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Use cases

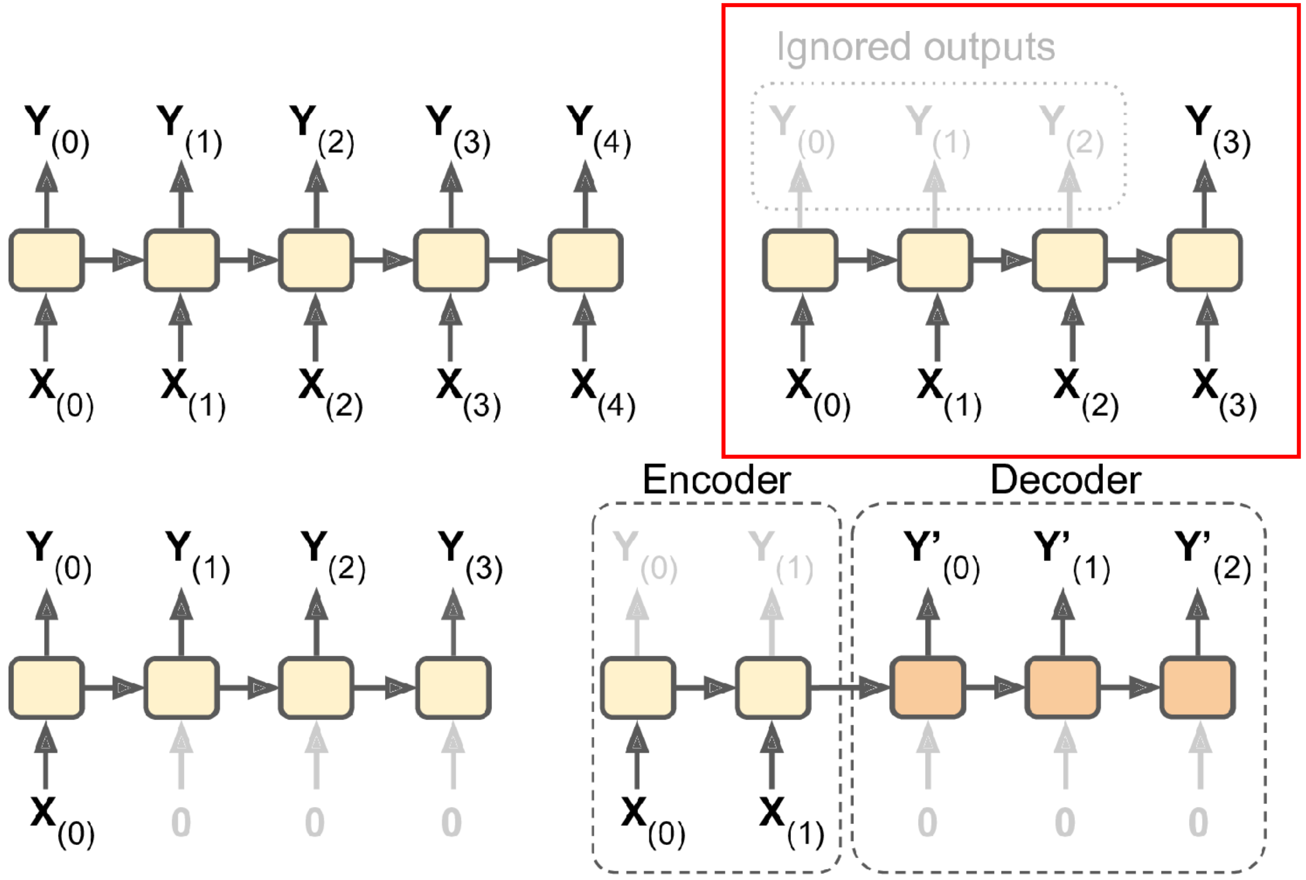
sequence-to-sequence

e.g. predicting time series



Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Use cases



sequence-to-vector

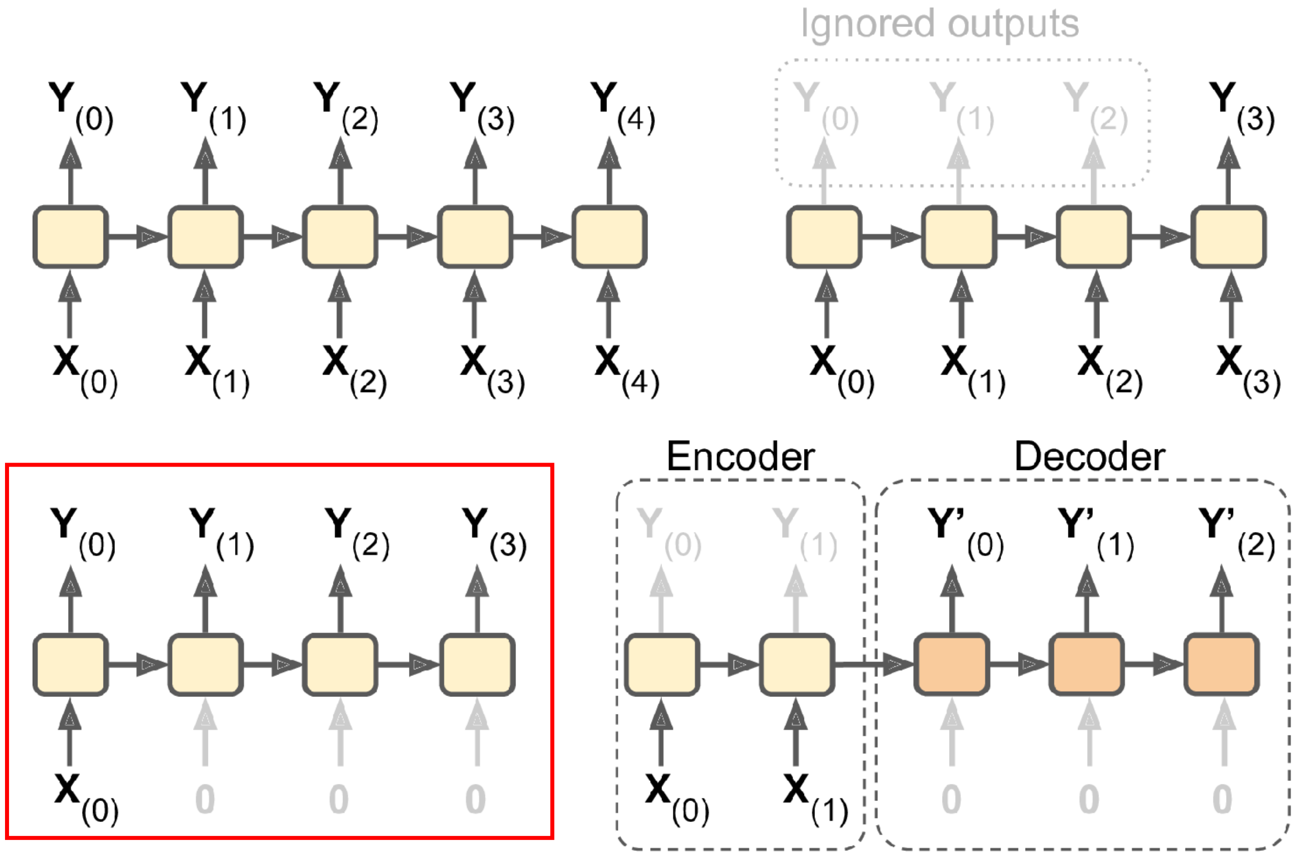
e.g. sentiment analysis

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Use cases

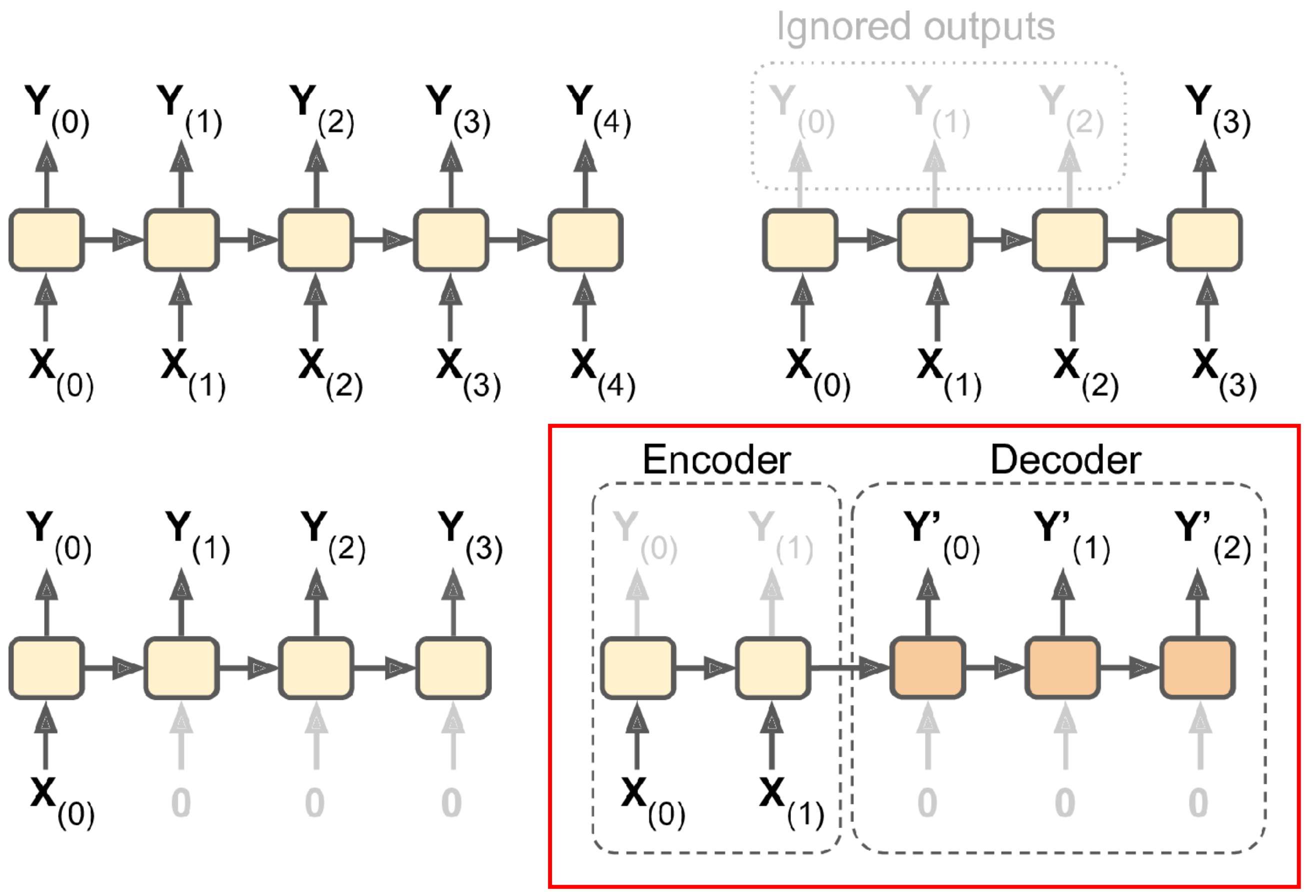
vector-to-sequence

e.g. image caption generation



Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Recurrent Neural Network (RNN): Use cases



sequence-to-vector

e.g. translation

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Course 4: Attention Mechanism and Transformer Architectures

Encoder-decoder

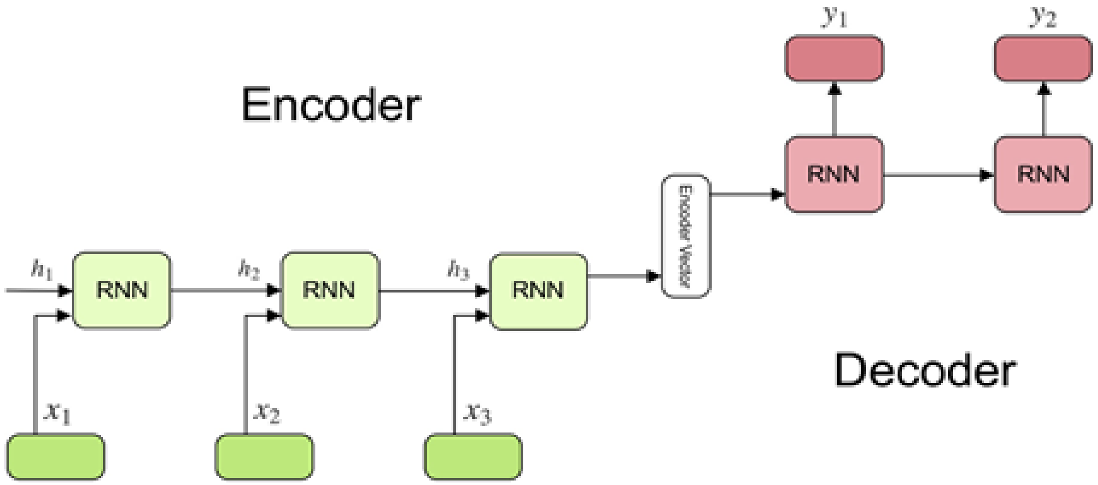
Encoder-decoder: A core concept

Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

Architecture

- An encoder
- An encoded vector
- A decoder



Encoder-decoder sequence to sequence model

https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

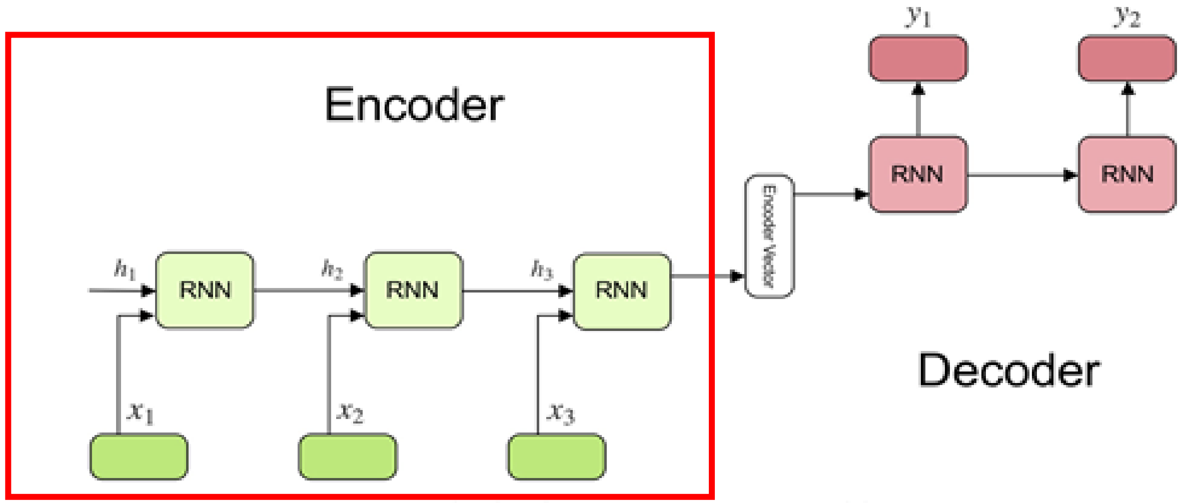
Encoder-decoder: A core concept

Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

Architecture

- **An encoder**
- An encoded vector
- A decoder



Encoder-decoder sequence to sequence model

https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

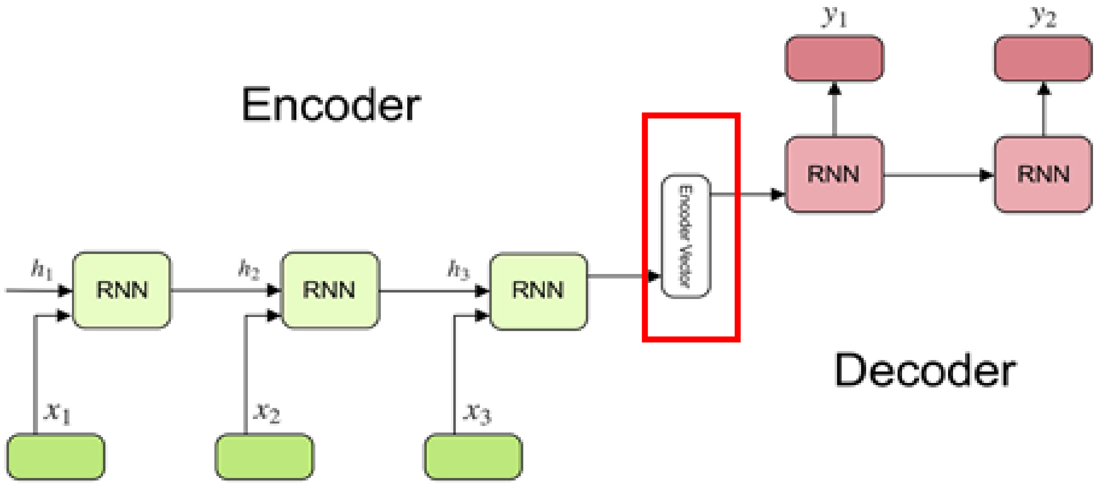
Encoder-decoder: A core concept

Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

Architecture

- An encoder
- **An encoded vector**
- A decoder



Encoder-decoder sequence to sequence model

https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

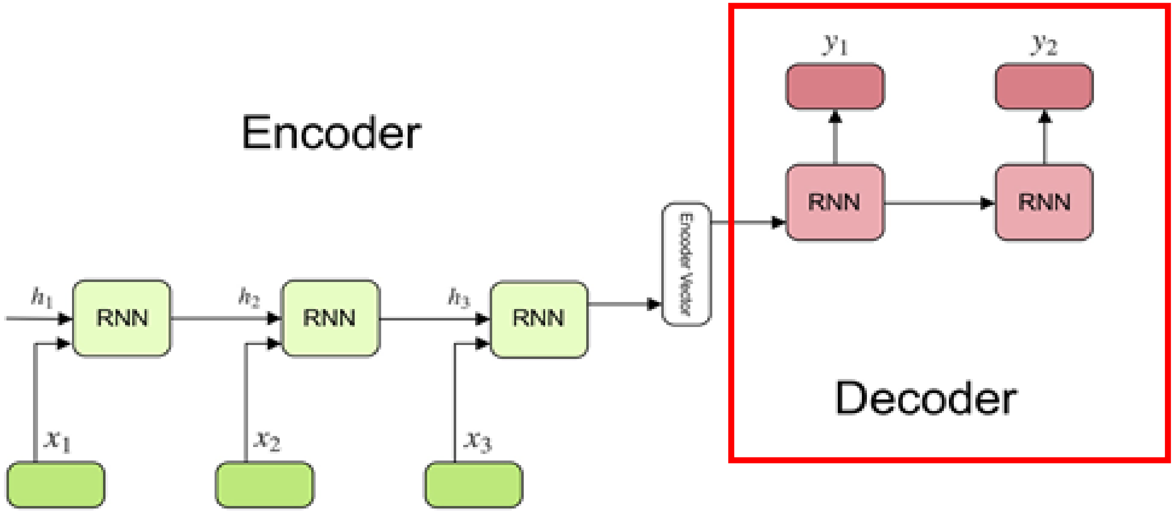
Encoder-decoder: A core concept

Principle

The goal is to project the input values into a smaller vector space before returning into the larger vector space expected

Architecture

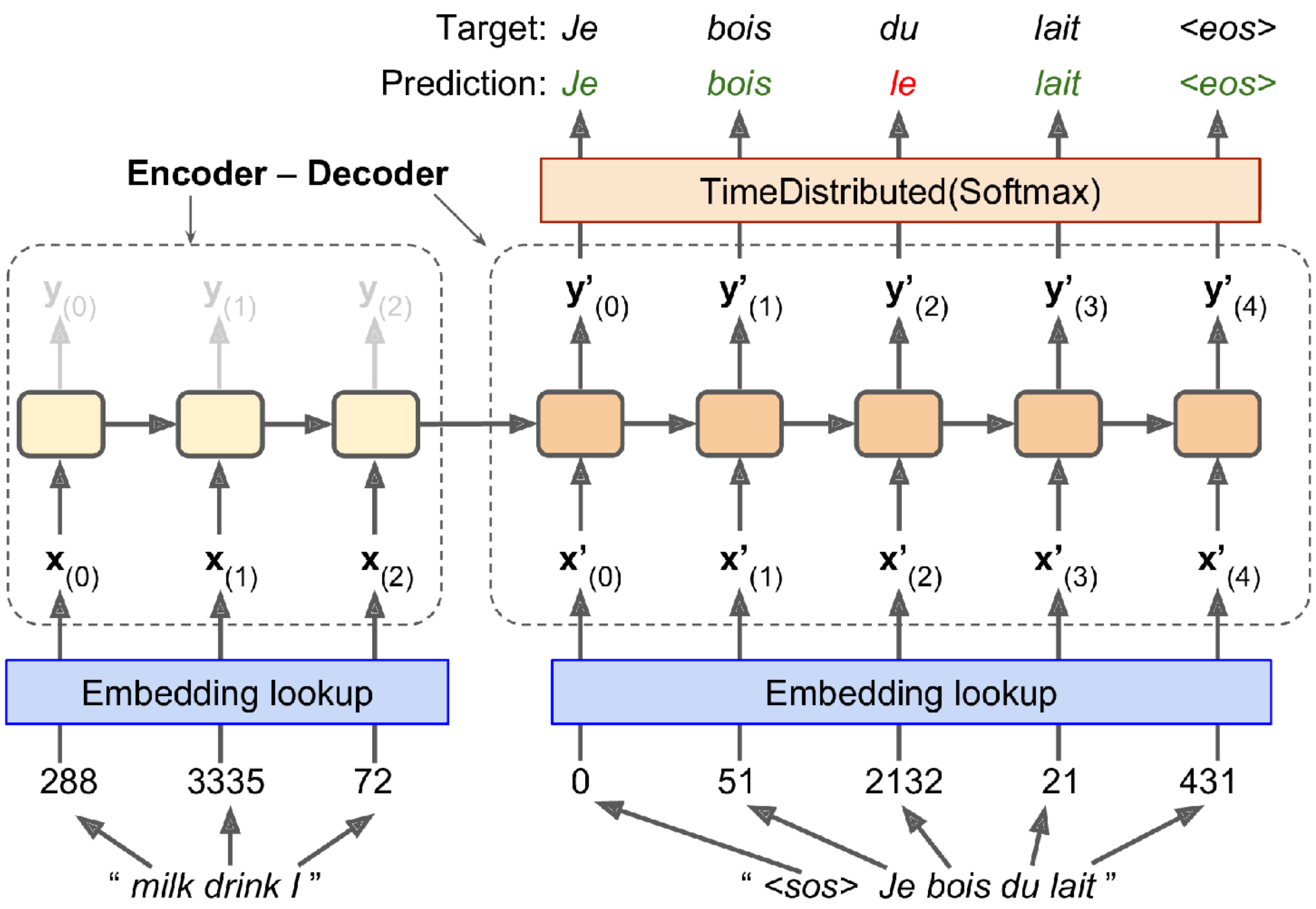
- An encoder
- An encoded vector
- **A decoder**



Encoder-decoder sequence to sequence model

https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

Encoder-decoder: Translation example



Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Encoder-decoder: Keras

```python
import tensorflow_addons as tfa

encoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
decoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
sequence_lengths = keras.layers.Input(shape=[], dtype=np.int32)

embeddings = keras.layers.Embedding(vocab_size, embed_size)
encoder_embeddings = embeddings(encoder_inputs)
decoder_embeddings = embeddings(decoder_inputs)

encoder = keras.layers.LSTM(512, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_embeddings)
encoder_state = [state_h, state_c]

sampler = tfa.seq2seq.sampler.TrainingSampler()

decoder_cell = keras.layers.LSTMCell(512)
output_layer = keras.layers.Dense(vocab_size)
decoder = tfa.seq2seq.basic_decoder.BasicDecoder(decoder_cell, sampler,
                                                 output_layer=output_layer)
final_outputs, final_state, final_sequence_lengths = decoder(
    decoder_embeddings, initial_state=encoder_state,
    sequence_length=sequence_lengths)
Y_proba = tf.nn.softmax(final_outputs.rnn_output)

model = keras.Model(inputs=[encoder_inputs, decoder_inputs, sequence_lengths],
                    outputs=[Y_proba])
```

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Encoder-decoder: Keras

```
import tensorflow_addons as tfa

encoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
decoder_inputs = keras.layers.Input(shape=[None], dtype=np.int32)
sequence_lengths = keras.layers.Input(shape=[], dtype=np.int32)

embeddings = keras.layers.Embedding(vocab_size, embed_size)
encoder_embeddings = embeddings(encoder_inputs)
decoder_embeddings = embeddings(decoder_inputs)
```

encoder

```
encoder = keras.layers.LSTM(512, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_embeddings)
encoder_state = [state_h, state_c]
```

allow to get both ht and Ct

```
sampler = tfa.seq2seq.sampler.TrainingSampler()
```

indicate to the decoder at each step what was the previous output

decoder

```
decoder_cell = keras.layers.LSTMCell(512)
output_layer = keras.layers.Dense(vocab_size)
decoder = tfa.seq2seq.basic_decoder.BasicDecoder(decoder_cell, sampler,
                                                 output_layer=output_layer)
final_outputs, final_state, final_sequence_lengths = decoder(
    decoder_embeddings, initial_state=encoder_state,
    sequence_length=sequence_lengths)
Y_proba = tf.nn.softmax(final_outputs.rnn_output)
```

model construction

```
model = keras.Model(inputs=[encoder_inputs, decoder_inputs, sequence_lengths],
                    outputs=[Y_proba])
```

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Attention Mechanism

Attention Mechanism: Introduction

Problem to solve

- Even with a LSTM architecture, it can be difficult to characterize long sentences in an efficient way

Attention mechanism principle

- To put more weight on specific word positions of the sentence depending **both** on the **context** and the **relative positions of words**

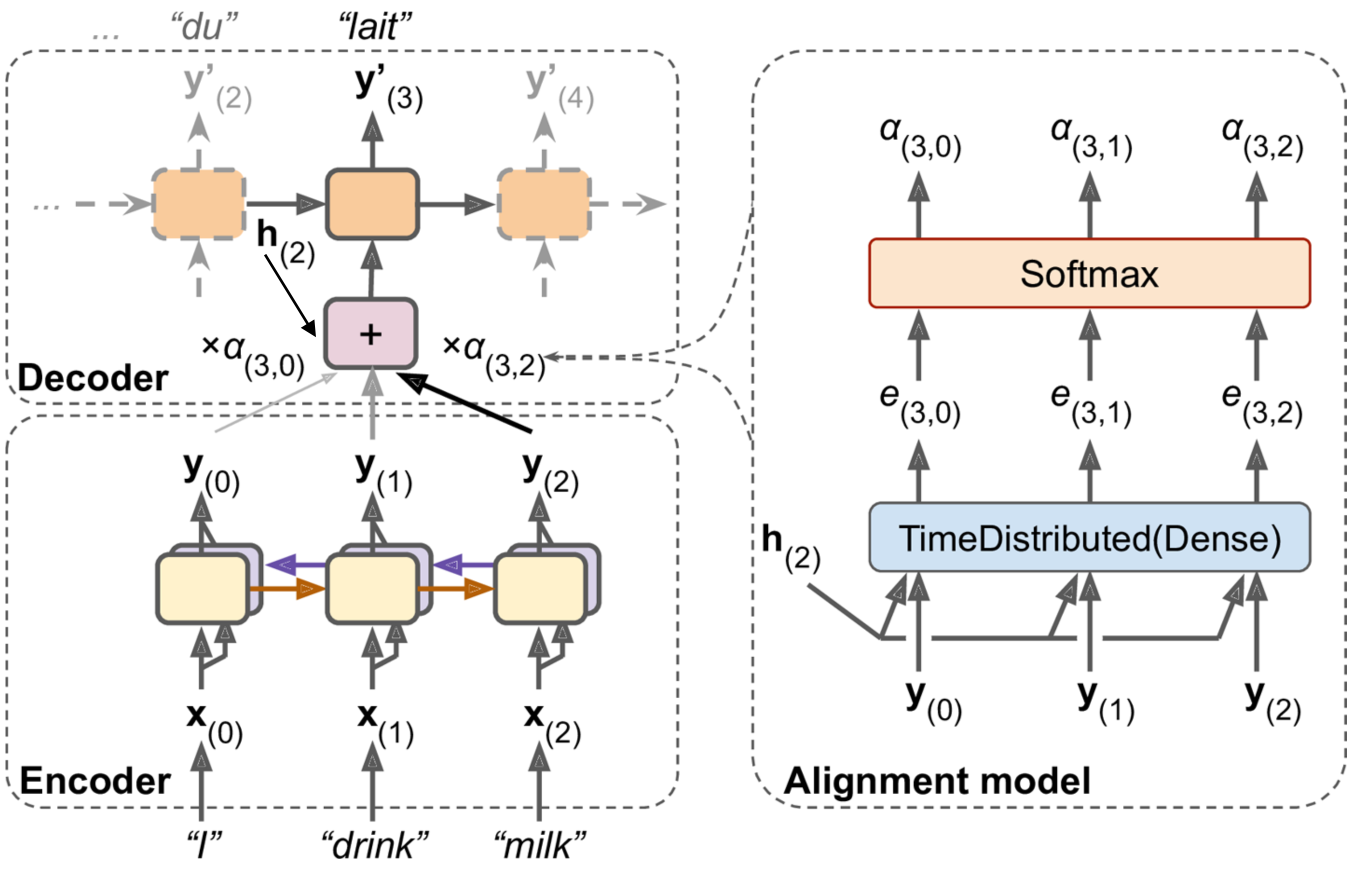
Example

Despite being from Uttar Pradesh, as he was brought up in Bengal, he is more comfortable in Bengali.

To predict the word *Bengali*, the expressions *brought up* and *Bengal* must be associated to higher weights.

https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/

Attention Mechanism: Translation illustration



Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Attention Mechanism: Get the intuition

Imagine you want to translate the sentence using an **encoder-decoder**:

"They play chess…"

Actually, you expect the encoder to deal with it that way:

"*They* *play* *chess…*" ⇨ {**subject**: *they* ; **verb**: *play* ; …}

subject verb

encoder

Encoded information

(in a very simplified way)

Just imagine you already translated the **first word** and are looking for the **next one**:

Ils ???

subject verb?

You are looking for a **verb** in the encoder information:

Values

{**subject**: *they* ; **verb**: *play* ; …}

Query

Keys

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Attention Mechanism: Get the intuition

Imagine you want to translate the sentence using an **encoder-decoder**:

"They play chess…"

Actually, you expect the encoder to deal with it that way:

"They play chess…" ⇨ {**subject**: *they* ; **verb**: *play* ; …}

subject verb

encoder

Encoded information

(in a very simplified way)

Just imagine you already translated the **first word** and are looking for the **next one**:

Ils ???

subject verb?

You are looking for a **verb** in the encoder information:

{**subject**: *they* ; **verb**: *play* ; …}

Query

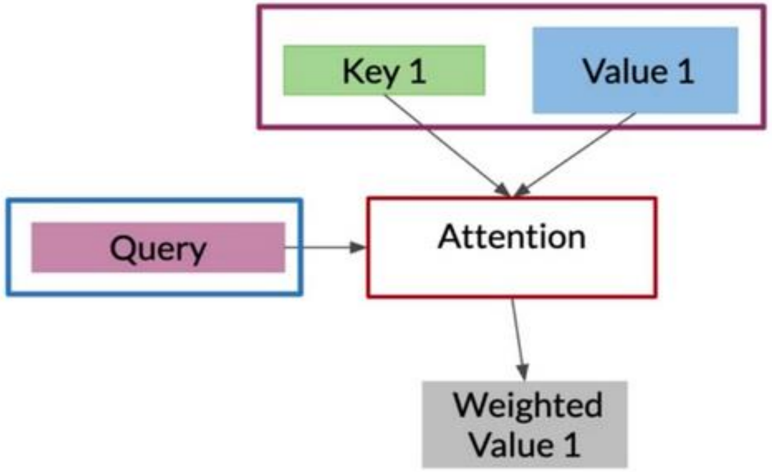
Keys and Values are oftenly fused together in the encoder vector, so you take Keys = Values

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2019

Attention Mechanism: Principle

Formalism

- To associate a ***query*** and a set of (***key***-***value***) tuples with an ***output***
- The **query**, the (**key**-**value**) set and the **ouput** are all vectors
- The output corresponds to the **weighted sum** of values, with weights determined from a **compatibility function** between the query and the corresponding key



https://www.coursera.org/specializations/natural-language-processing

Attention Mechanism: Formula

Formula

The mathematical expression generally used to compute the attention function is

$$\boxed{softmax(QK^T)V}$$

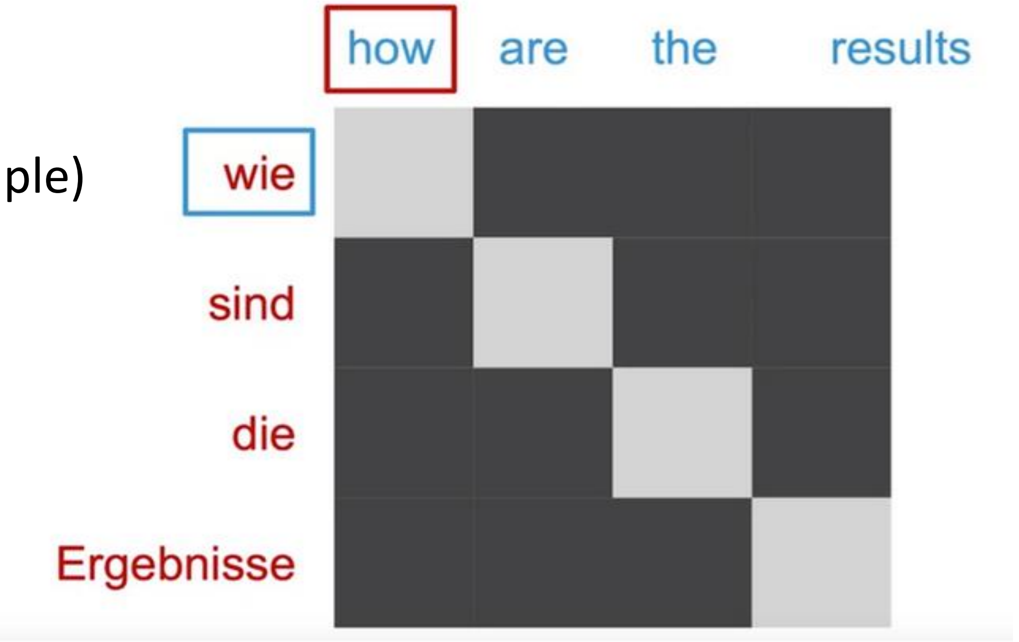
With ***Q****,* ***K*** and ***V*** respectively the *query*, *key* and *value*, matrices.

The ***softmax*** function gives normalized values between 0 and 1

Attention Mechanism: Illustration 1

Example for the case of English-German translation

- The bright squares show where the algorithm « **look** » to translate a word
- Here, words in English and German **share the same order**
- So, the match is only on the **diagonal**
- No need to look other word positions (on this specific example)

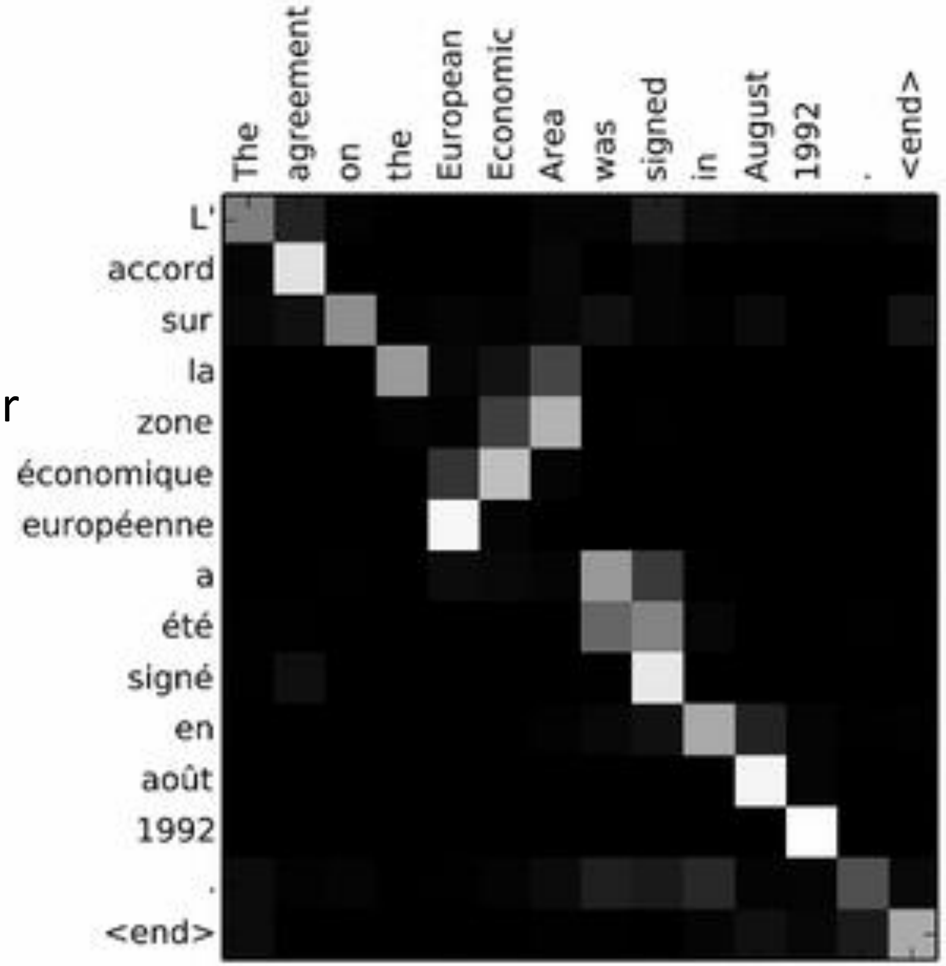


https://www.coursera.org/specializations/natural-language-processing

Attention Mechanism: Illustration 2

Example for the case of English-French translation

- Here, the match is **not on a specific word only**
- The figure shows the contribution of each word
- **Several words** can contribute to the translation of another



https://www.coursera.org/specializations/natural-language-processing

Attention Mechanism: Examples

Intuition behind query and (key, value) concepts

Depends on the application

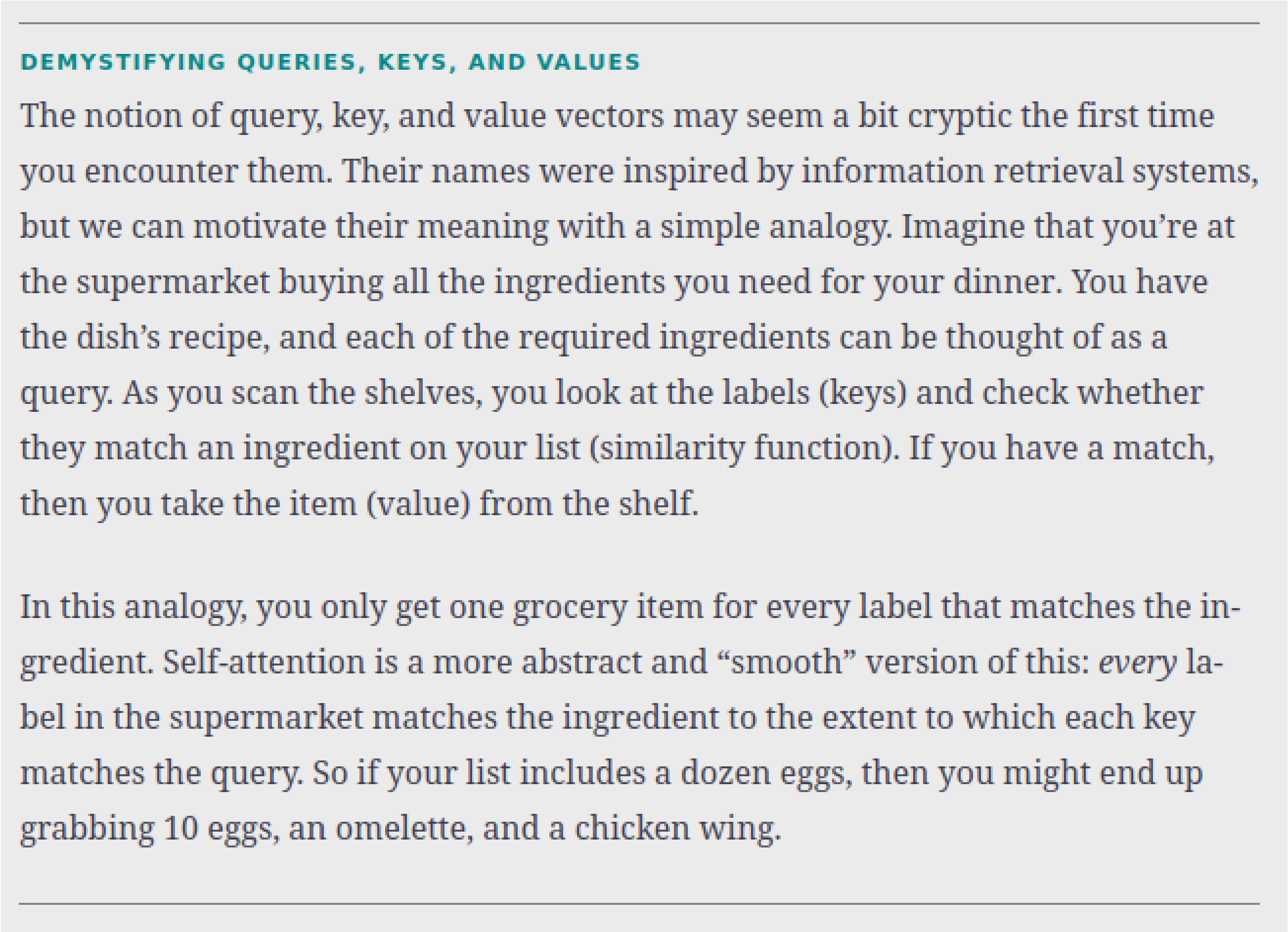
In NLP, usually, the key and value correspond to the **same vectors**:

- **Translation**: the **query** can be the encoded sentence vector in **one language** and both **key** and **value** can be the **encoded** sentence vector in the **other language**
- **Text similarity**: the **query** can be the sequence embedding of the **first piece of text** and both **key** and **value** can be the sequence embedding of the **second piece of text**

However, in some cases, the concepts **key** and **value** **can be different**:

- if you are looking for a video in *Youtube*, the **query** can be derived from the **text you entered**, the **keys** can correspond to **the video descriptions** and the **values** can be **the videos themselves**

Attention Mechanism


DEMYSTIFYING QUERIES, KEYS, AND VALUES

The notion of query, key, and value vectors may seem a bit cryptic the first time you encounter them. Their names were inspired by information retrieval systems, but we can motivate their meaning with a simple analogy. Imagine that you're at the supermarket buying all the ingredients you need for your dinner. You have the dish's recipe, and each of the required ingredients can be thought of as a query. As you scan the shelves, you look at the labels (keys) and check whether they match an ingredient on your list (similarity function). If you have a match, then you take the item (value) from the shelf.

In this analogy, you only get one grocery item for every label that matches the ingredient. Self-attention is a more abstract and "smooth" version of this: *every* label in the supermarket matches the ingredient to the extent to which each key matches the query. So if your list includes a dozen eggs, then you might end up grabbing 10 eggs, an omelette, and a chicken wing.

L. Tunstall, L. Von Werra, T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*, (2022)

Visual Attention: Illustration

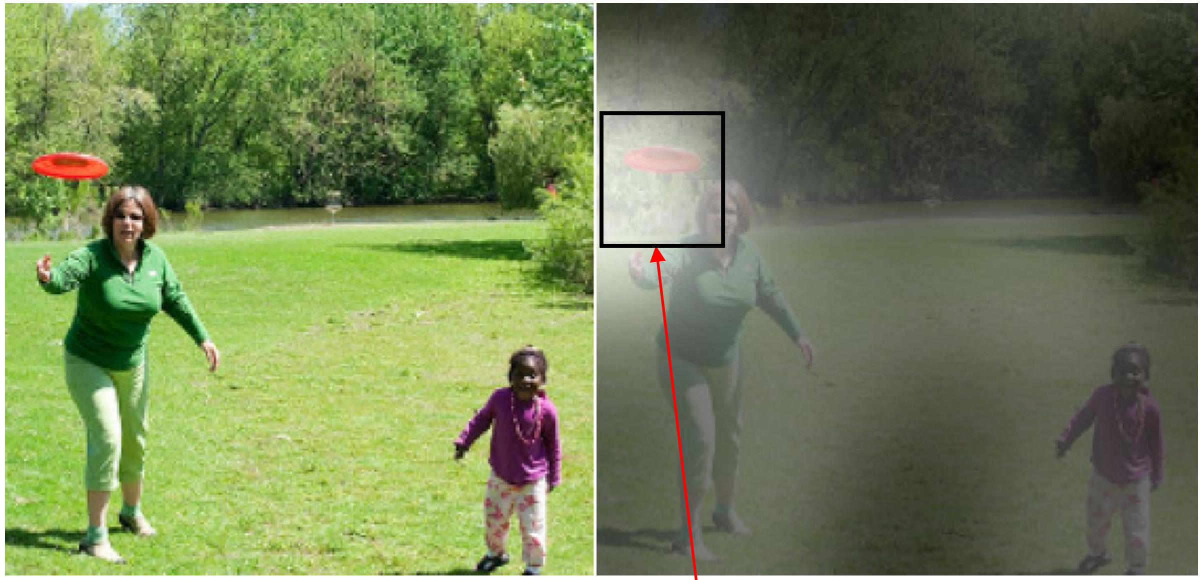
Goal: Generate automatically the caption of a picture with an attention mechanism

"A woman is throwing a frisbee in a park"

Kelvin Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention," *Proceedings of the 32nd International Conference on Machine Learning* (2015): 2048–2057.

Visual Attention: Illustration

Goal: Generate automatically the caption of a picture with an attention mechanism



“A woman is throwing a ***frisbee*** ***in a park****”*

Kelvin Xu et al., “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention,” *Proceedings of the 32nd International Conference on Machine Learning* (2015): 2048–2057.

Transformer architecture

Transformer: Introduction

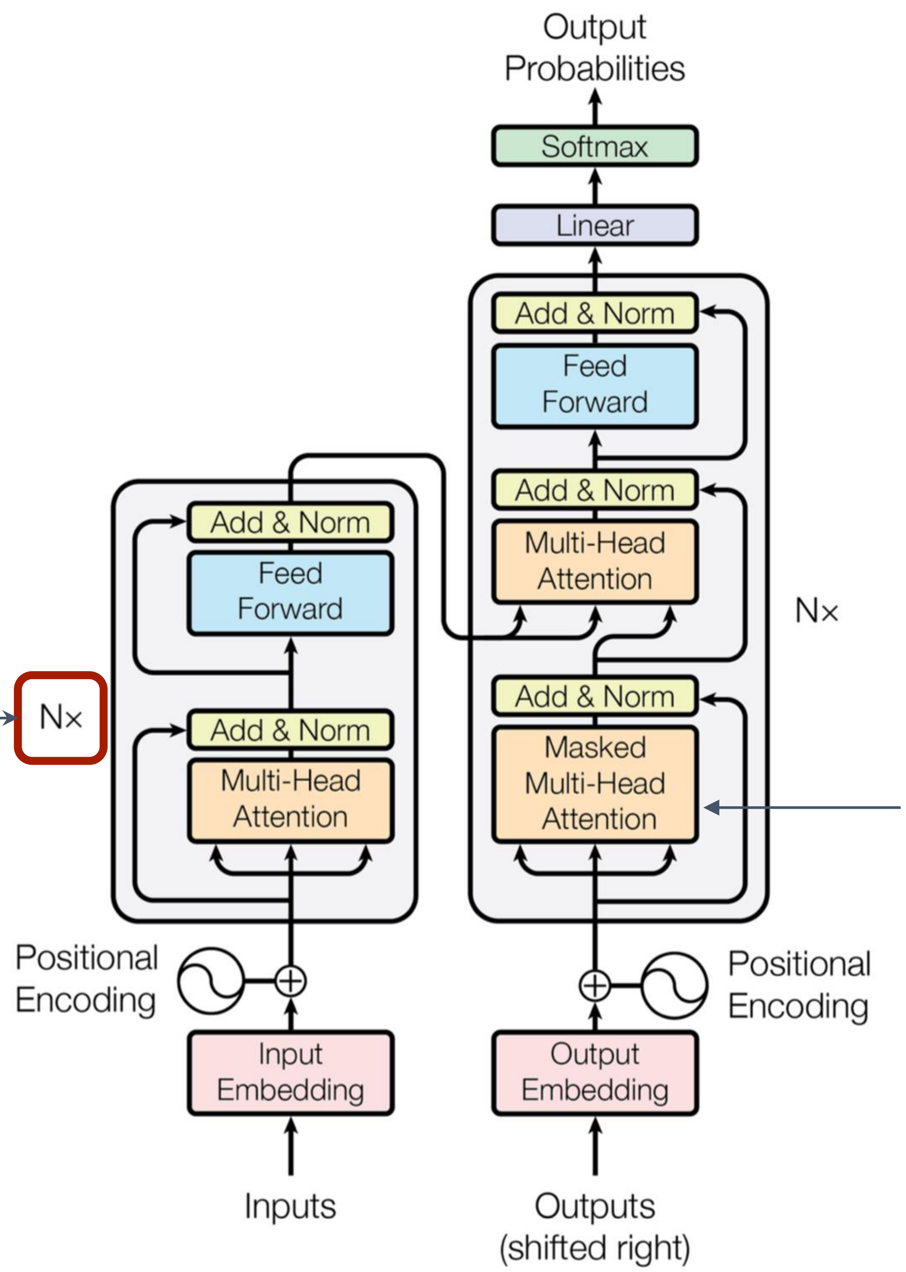
Principle

- A quite new deep learning model (2017), introduced in the **game changing** article *Attention is all you need*
- Relies heavily on the **Attention mechanism**
- **Does not need to process a sequence in a specific order**
- Solves the issue of keeping into memory the information related to distant words (and that even **without LSTM**)
- Makes possible a significant use of **parallelization** computing
- At every moment, the algorithm can access the complete set of the successive states visited during the procedure

The main idea is that the Attention mechanism alone, without any recurrent sequential procedure, is powerful enough to reach the state of the art

Transformer

Hyperparameter:
The number of
encoders

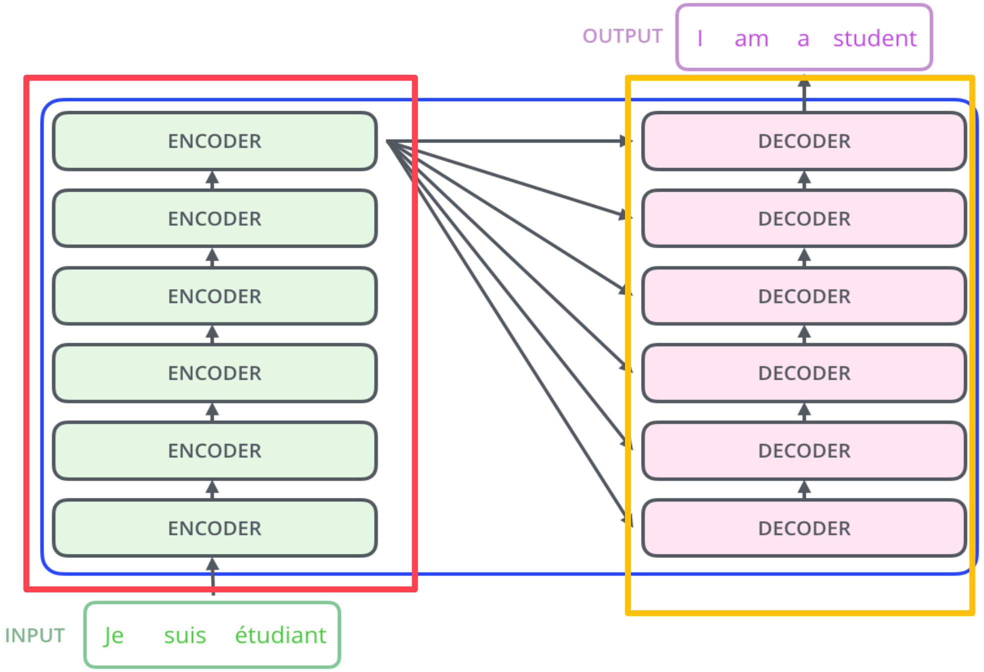


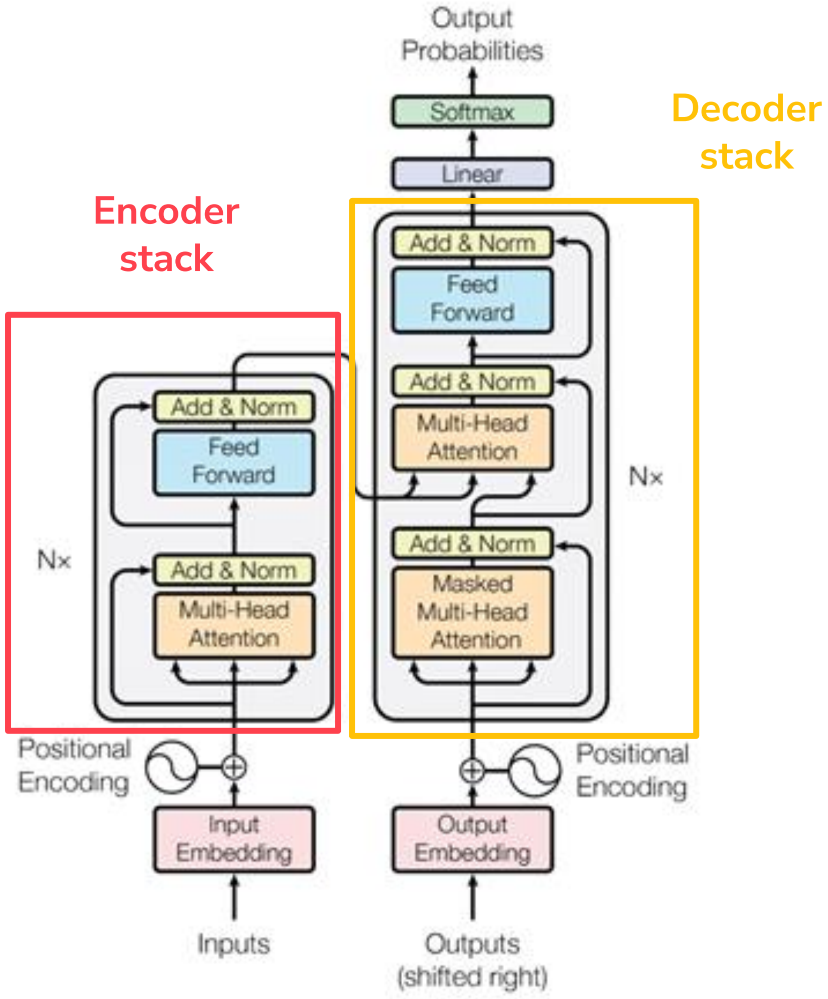
Hyperparameter:
The number of
attention heads

Transformer: Architecture

The Transformer is made of two main components:

- A stack of **identical encoders** (independents form each other)
- Followed by a stack of ***identical decoders*** (independents form each other)



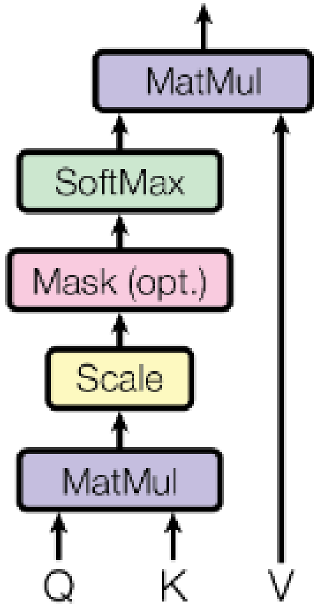


Transformer: Scaled Dot-Product Attention

Formula

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

dk corresponds to the query and key dimension



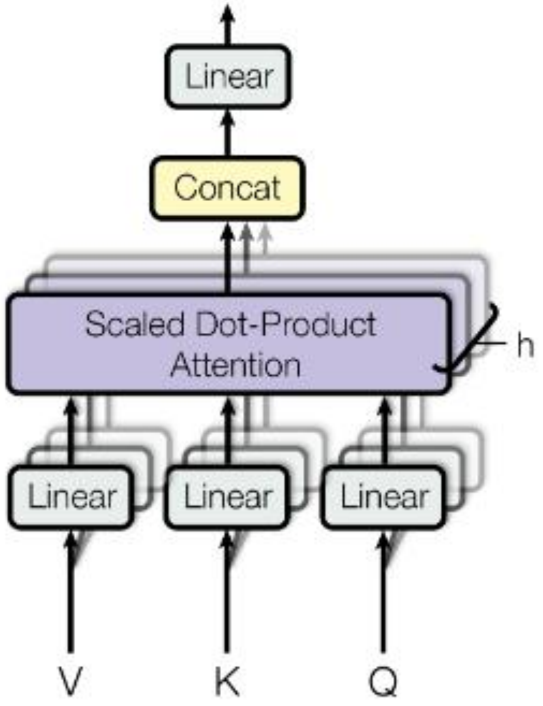
- The **normalization** prevents the result of the dot-product to reach high values and, doing so, **prevents the gradient to vanish** into small values
- An **additive** variant of the attention function exists, but the dot-product version is preferred here for the efficiency of the matrix calculations it allows

Transformer: Multi-Head Attention

Principle

- **Several Attention** layers computed in parrallel
  - the *queries*, *keys* and *values* are projeted *h* times
  - the h results are concatenated
  - then projeted one last time
- Enable the use of different sub-space information
- The mathematical expression is the following

$$\boxed{\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O}$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

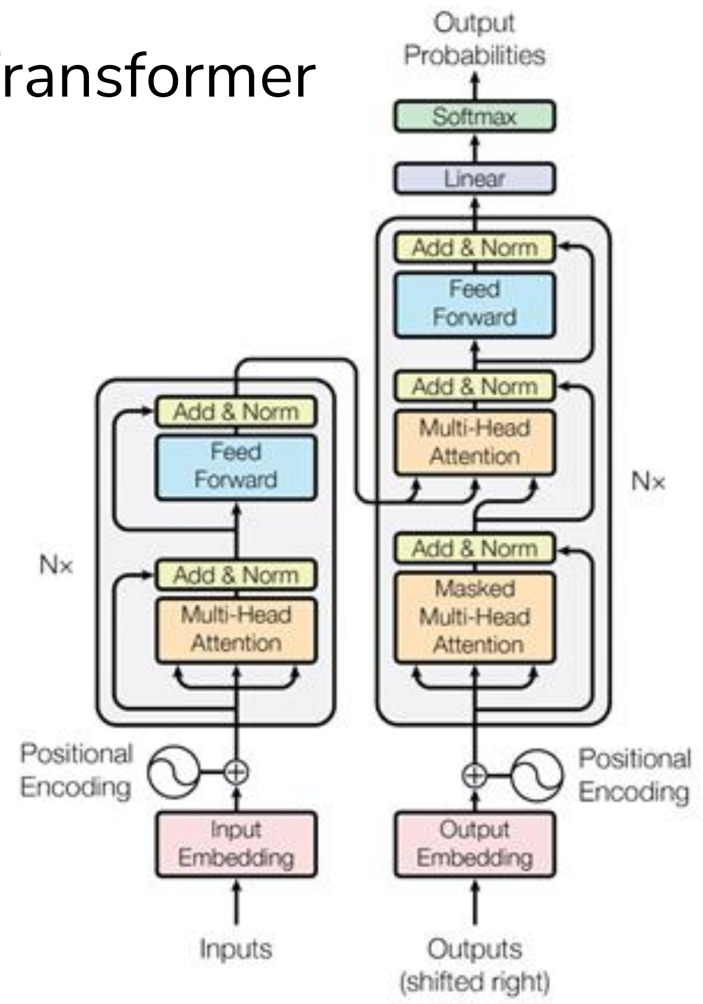


Where the Wi* corresponds to the respective projection matrices

Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

- Inside the encoder
- Inside the decoder
- At the encoder-decoder interface

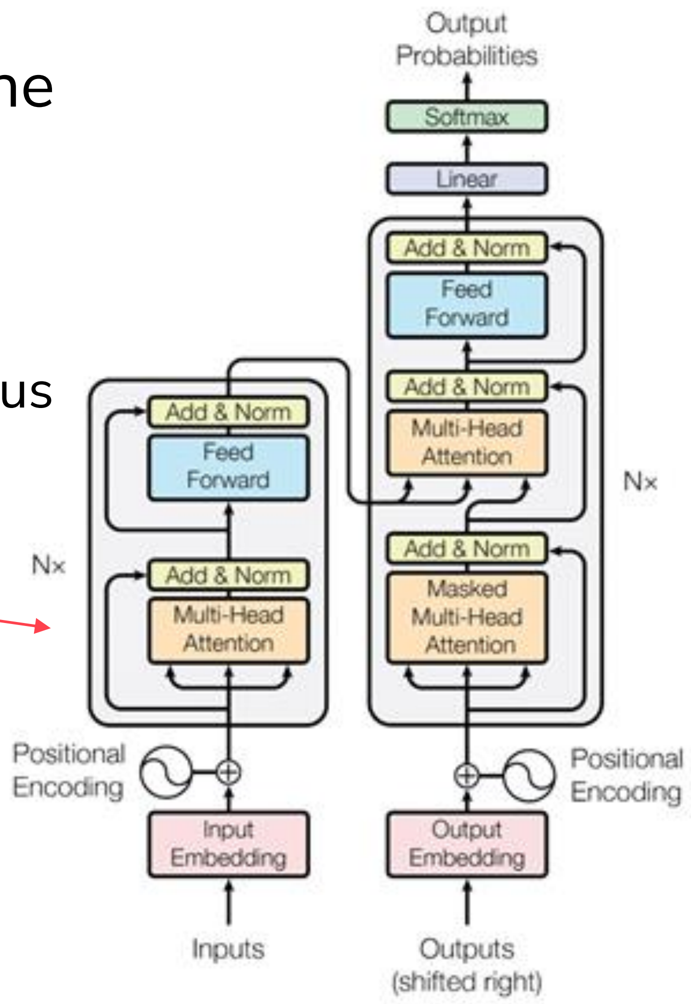


Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

- **Inside the encoder**
  - the queries, keys and values all come from the previous encoder output
- Inside the decoder
- At the encoder-decoder interface

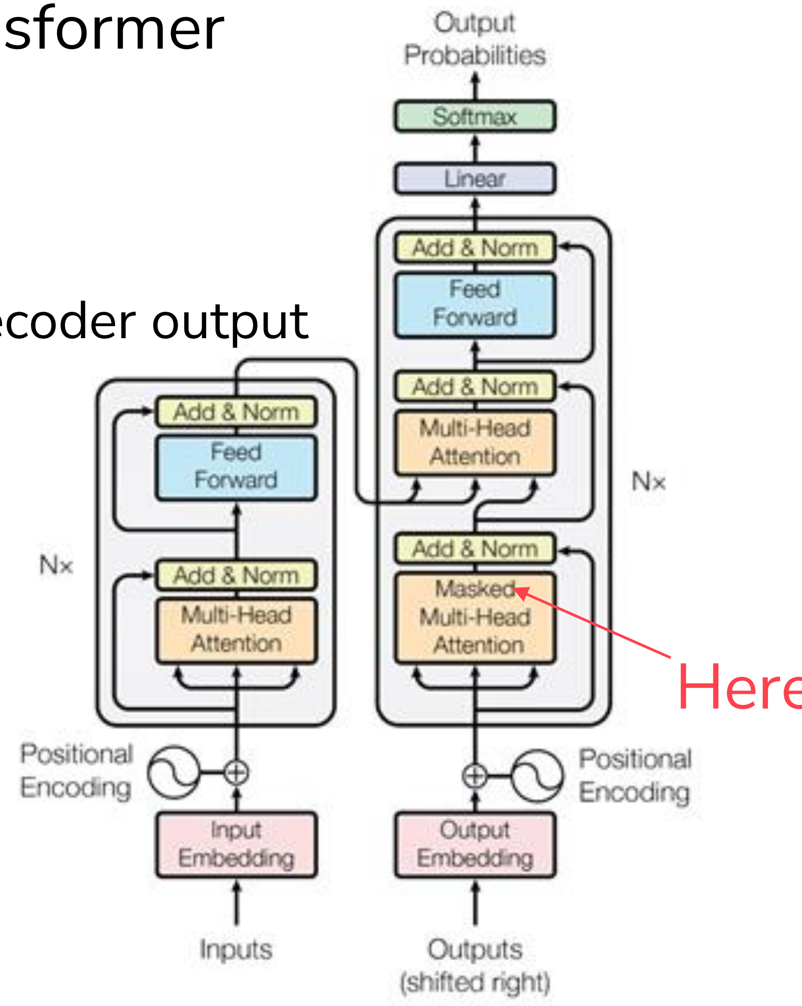
Here



Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

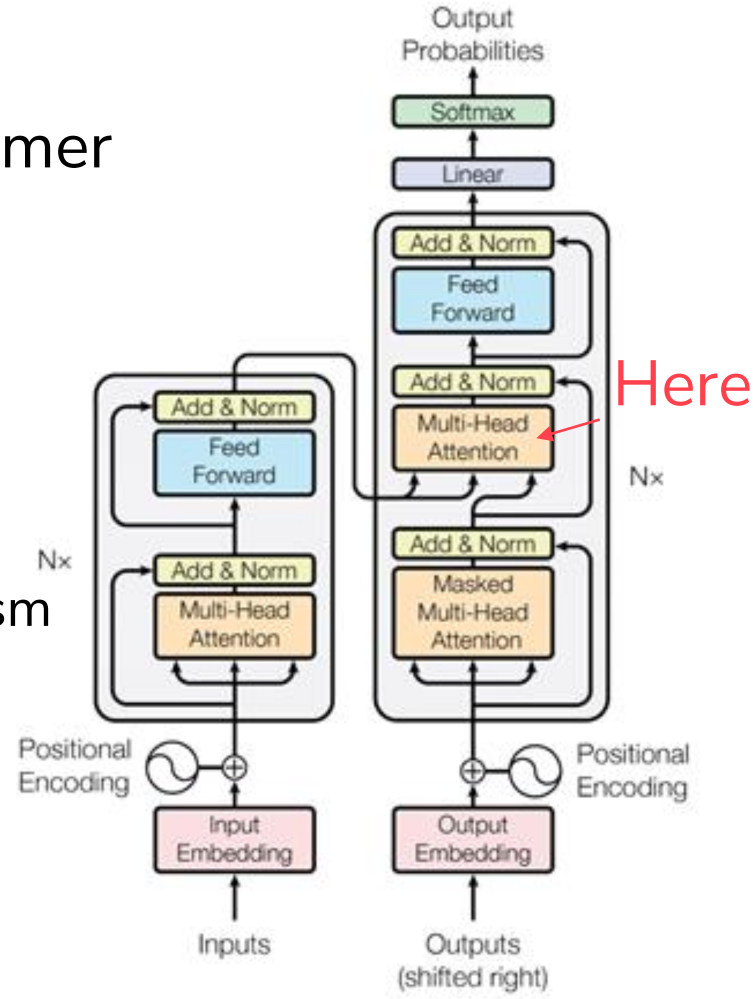
- Inside the encoder
- **Inside the decoder**
  - the queries, keys and values all come from the previous decoder output
  - use of a « mask » to decode only from known elements
- At the encoder-decoder interface



Transformer: Multi-Head Attention

The **Multi-Head Attention** is used three times in the Transformer

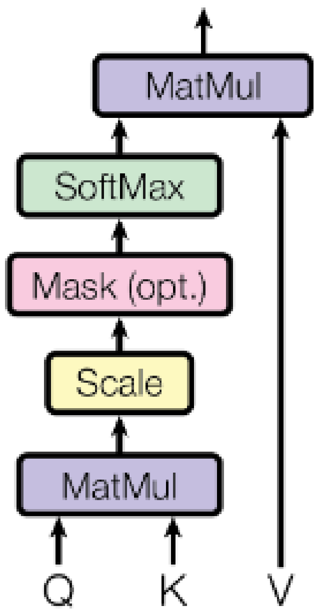
- Inside the encoder
- Inside the decoder
- **At the encoder-decoder interface**
    - the queries come from the previous decoder layer
    - the keys and values come from the encoder outputs
    - identical to the classical “encoder-decoder” Attention mechanism



Transformer: Attention

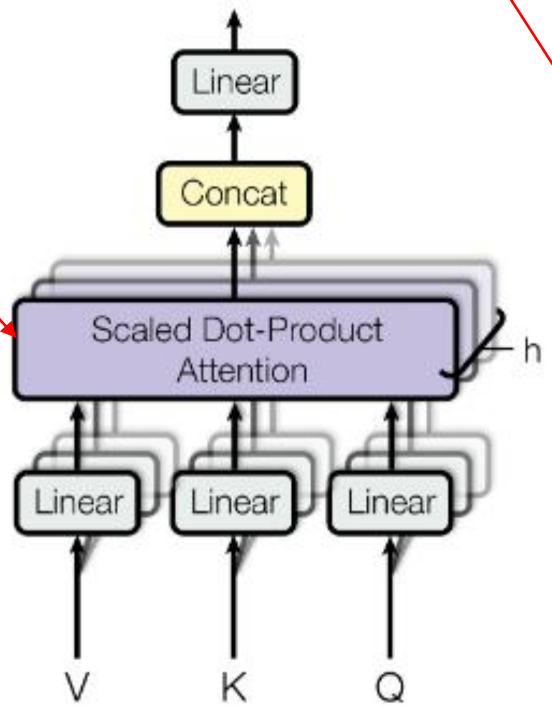
Transformer

Attention



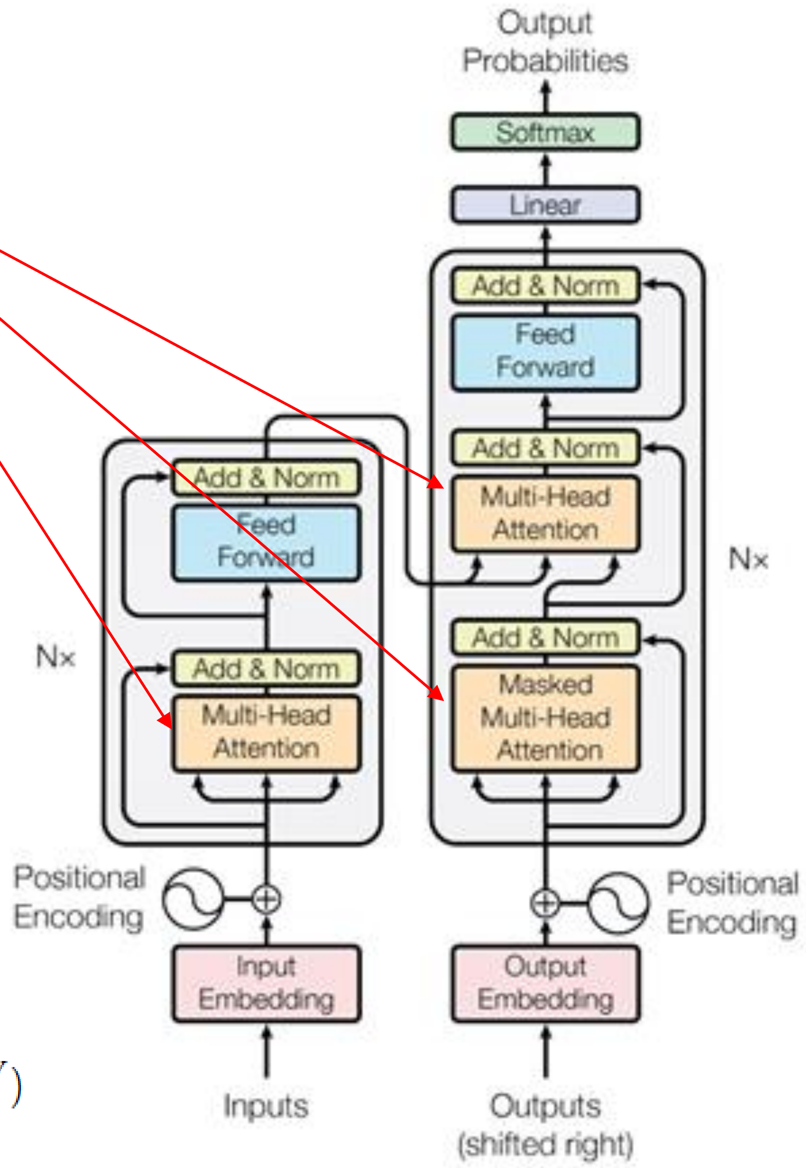
$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

multi-head Attention



$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$

$$\text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



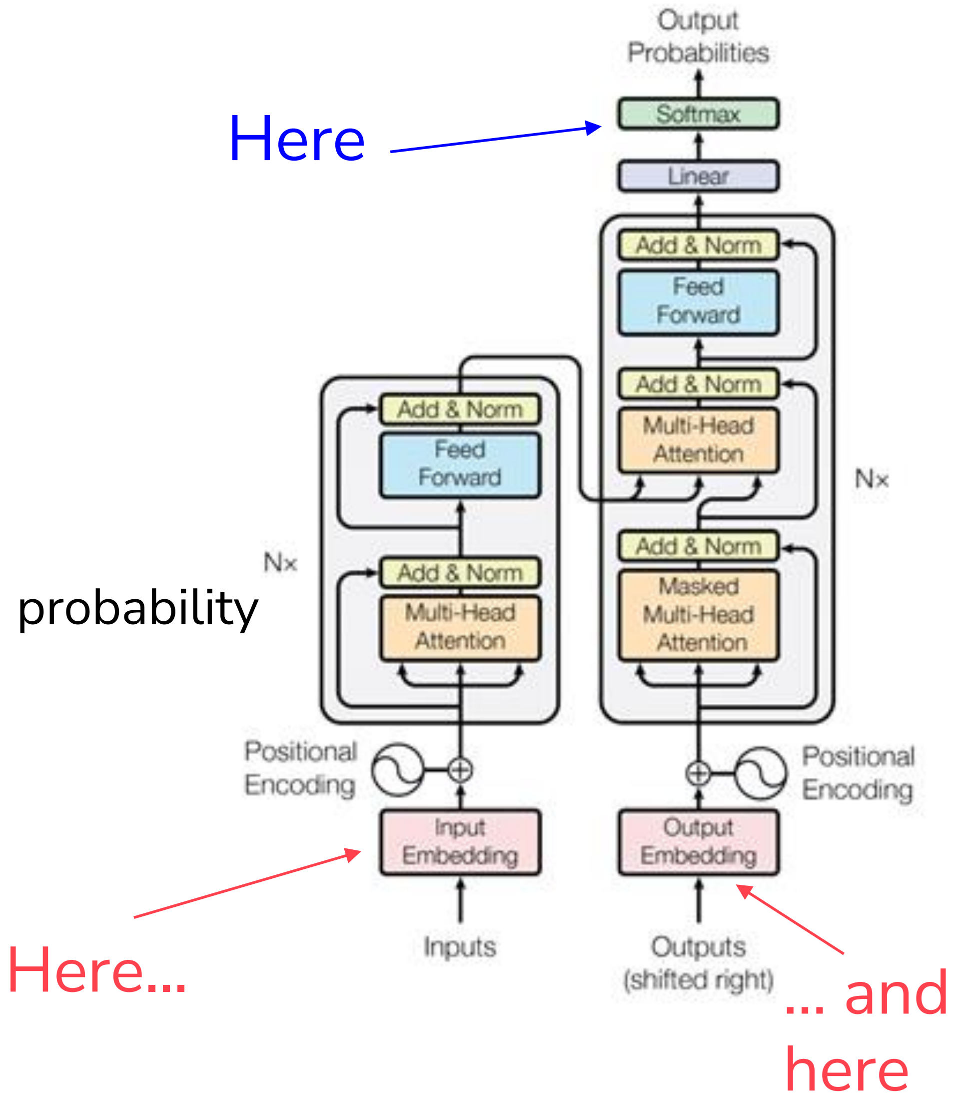
Transformer: Embeddings et Softmax

Embedding learning in order to convert

- The input *tokens*
- The output *tokens*

Use of a *Softmax* layer

- Convert the *decoder* output into the next token prediction probability



Transformer: Positional Encoding

Principle

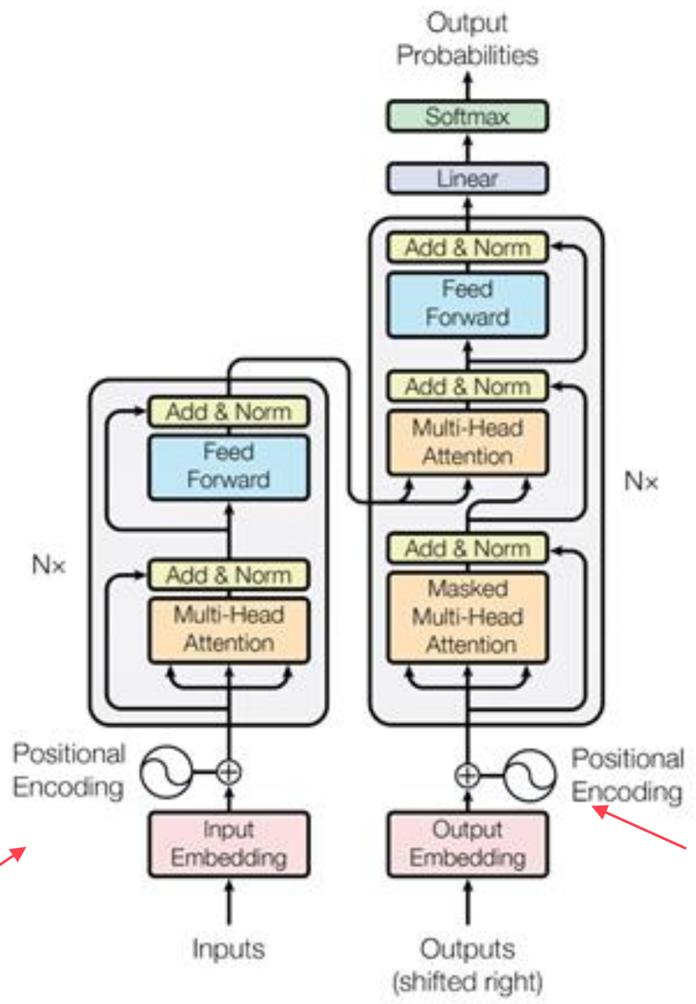
- Keep in memory the *token* positions in the sequence
- Add of "*positional encodings*" to the input *embeddings*
- Used just before the *encoder-decoder* stacks

Formulas

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

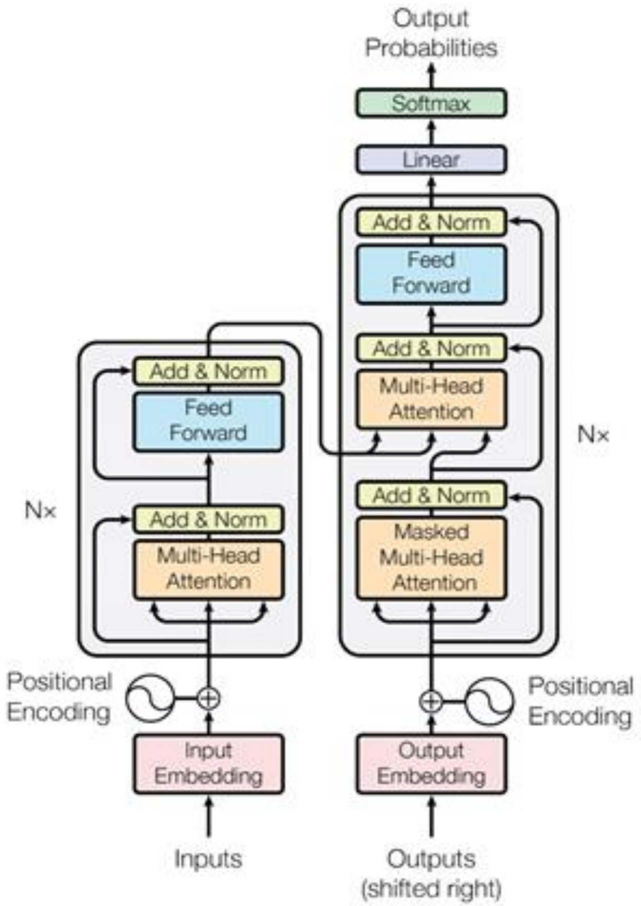
- with *pos* the position and *i* the dimension
- Each dimension corresponds to a sinusoïd
- For each *k*, *PEpos+k* is a linear function of *PEpos*

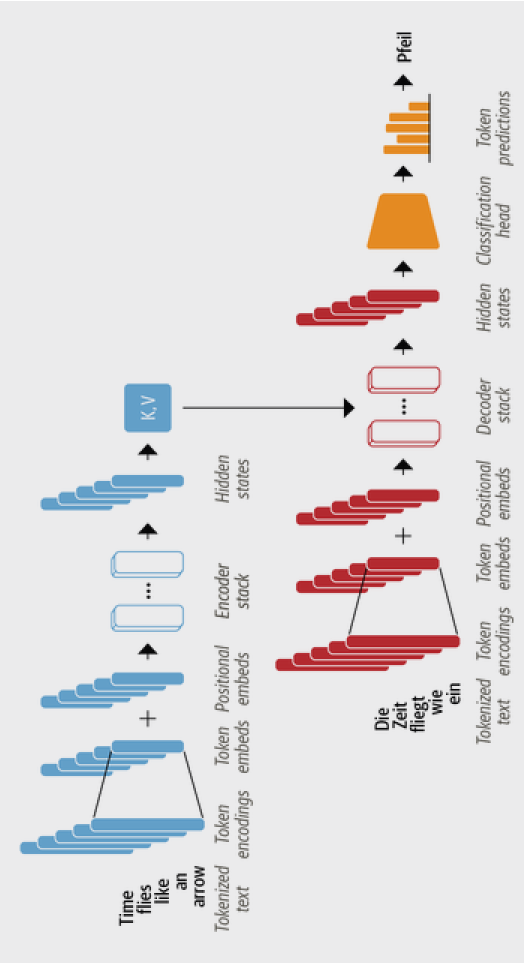
Here...



... and here

Transformer: Illustration





L. Tunstall, L. Von Werra, T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*, (2022)

Transformer: get the intuition

DEMYSTIFYING ENCODER-DECODER ATTENTION

Let's see if we can shed some light on the mysteries of encoder-decoder attention. Imagine you (the decoder) are in class taking an exam. Your task is to predict the next word based on the previous words (decoder inputs), which sounds simple but is incredibly hard (try it yourself and predict the next words in a passage of this book). Fortunately, your neighbor (the encoder) has the full text. Unfortunately, they're a foreign exchange student and the text is in their mother tongue. Cunning students that you are, you figure out a way to cheat anyway. You draw a little cartoon illustrating the text you already have (the query) and give it to your neighbor. They try to figure out which passage matches that description (the key), draw a cartoon describing the word following that passage (the value), and pass that back to you. With this system in place, you ace the exam.

L. Tunstall, L. Von Werra, T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*, (2022)

Transformer: Results on the Benchmark *BLEU*

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet	23.75			
Deep-Att + PosUnk		39.2		$1.0 \cdot 10^{20}$
GNMT + RL	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer: Tutorial notebook

course4_transformer_tutorial.ipynb

Goal: Illustration of how a Transformer can be build step by step in Python

Remarks:

- This notebook comes from a **tutorial notebook** **freely accessible** (the original reference is given at the very beginning, in the first cell)

BERT model

BERT: Introduction

BERT (2018) is for ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers*

- A Transformer variant with a **Bidirectional** specificity
- Thanks to it, there is no need for the end sequence token masking constraint anymore (into the decoders Attention mechanism)
- Build an efficient and generic language representation

The two main BERT steps

- A **pre-training** step on non-labelled data
- A **specialization** step ("***fine tuning***") for a specialized task, from pre-trained parameters
    - There is only need to train **the « last » layers** of the model to get such a specialization

French versions of BERT exists, the most famous one being probably **CamemBERT**

BERT: Architecture

The initial **BERT** architecture (2018) is the following:

- Length of the processed sequences: **512** (Limitation to contain a quadratic explosion)
- Stack of **Nx** = **12 blocks** of encoders and decoders (a 24-blocks variant exists)
- **h = 12** attention heads
- It means there are h x Nx = 12 x 12 = **144** distinct attention mechanisms
- Each head gives an output vector of size: **64**
- A hidden layer of h x 64 = **768** dimensions
- Roughly **110 millions** of parameters for the basic version (12 encoders) and **340 millions** for the large version (24 encoders)

Reminder, the initial Transformer architecture (2017)

- Length of the processed sequences: **512** as well
- Stack of **Nx = 6 blocks** of encoders and decoders
- **h = 8** attention heads

BERT: Pre-training

BERT is pre-trained from two unsupervised tasks:

Masked Language Modeling (MLM)

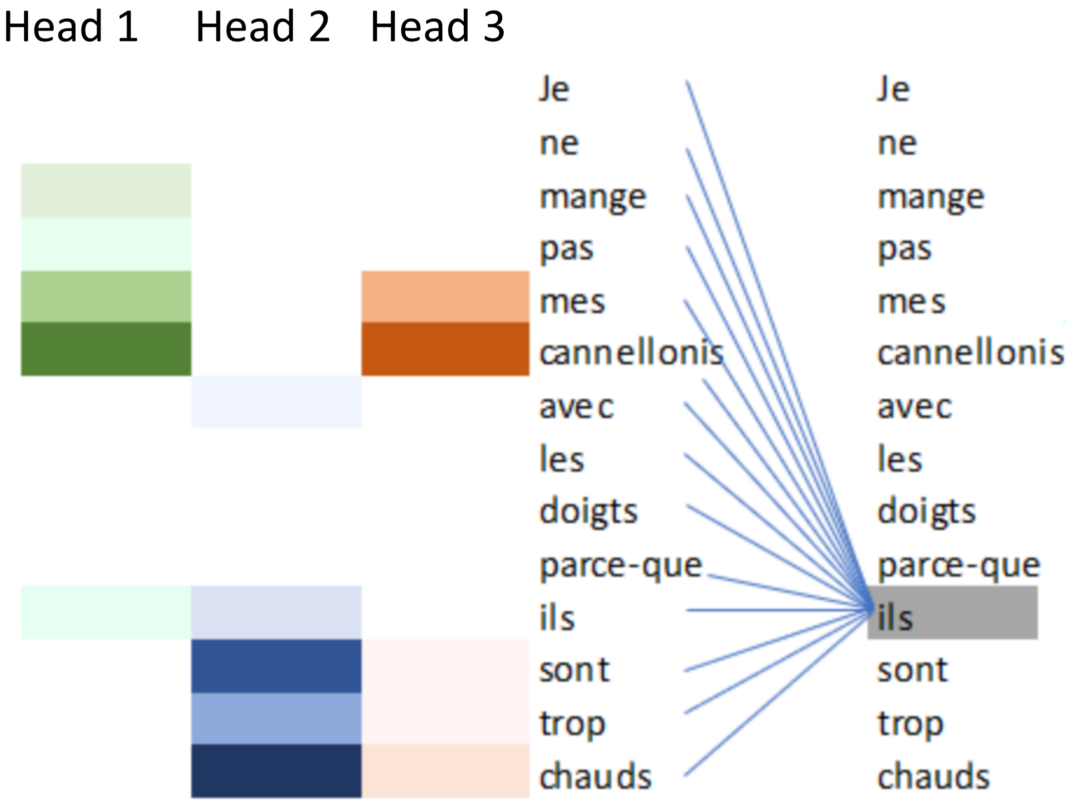
- The model is pre-trained from sequences decomposed into *tokens*
- The goal is to predict some of these tokens (15%) randomly masked in the text

Next Sentence Prediction (NSP)

- The goal is to understand the relationship between two successive sentences
- To do that, the algorithm is trained in order to predict the sentence following a given sentence
- The training set is composed of sentence tuples (A,B)
  - In 50% of the cases, B is the **sentence following** the sentence A (the tuple is then labelled *IsNext*)
  - In 50% of the cases, B is a sentence **randomly** chosen in the corpus (*NotNext* label)

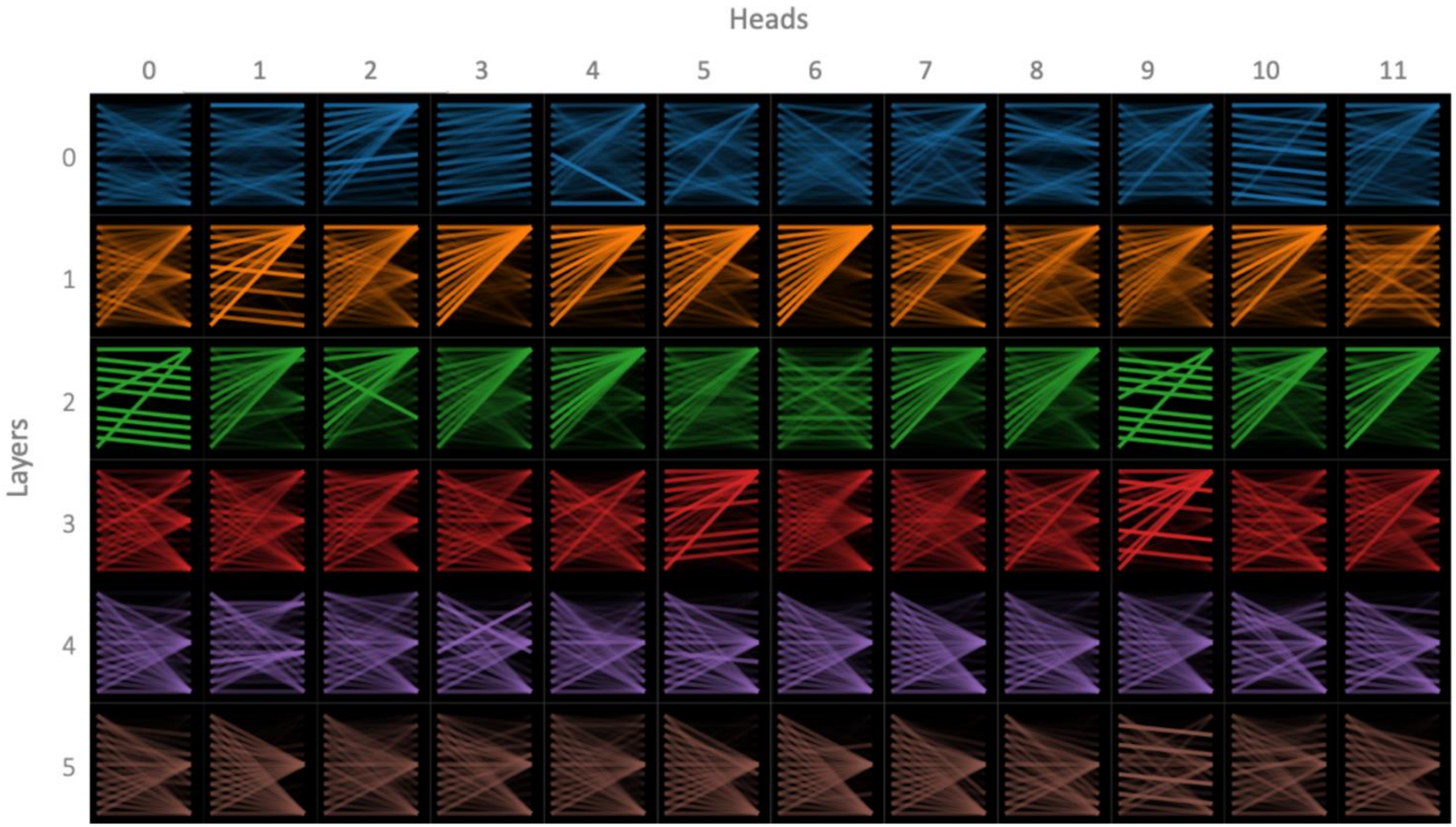
CamemBERT and Attention mechanism

Attention level, for each head, relativly to the french word “ils” (« they »). Each head “see” **different** things from others (**complementarity**)



CamemBERT and Attention mechanism

On the first 6 layers (out of 12), attention mechanism visualization example over the 12 heads



BERT: Fine-tuning

The initial pre-training step needs:

- A **huge** corpus of **generic** texts
- Very **high** computational resources

The fine-tuning step relies on the already pre-trained algorithm, so it only needs :

- A **reduce** and **specialized** corpus
- Significantly **reduces** resources

The aim is obviously to move from a **generic language** modelization to a **task specific one**

Classify text with BERT: Exercise

course4_classify_text_with_BERT.ipynb

Goal: Illustration of how a pre-trained Transformer model such as BERT can be used to solve practical and specific use case

Remarks:

- Shows how to train, from BERT outputs, a classifier model with only few layers
- Can help you solve **question 3** of the **Graded Project**
- This notebook comes from a **tutorial notebook** **freely accessible** (the original reference is given in the correction)

Hugging Face

Hugging Face

- Hugging Face is an open-source provider of NLP technologies for Python
- A Hugging Face **Transformer** package is available and quite popular
- It gives access to pretrained models such as BERT but also to modules to perform fine-tuning
- It previously supported only **Pytorch**, but now **Tensorflow** is more and more supported as well

Transformer libraries: Tutorial notebook

course4_transformer_model_libraries.ipynb

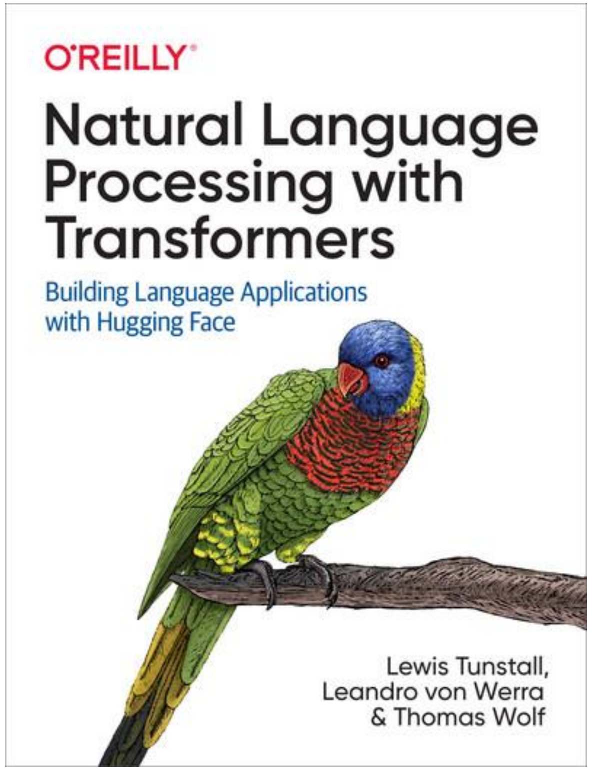
<u>Goal</u>: Illustration of how pre-trained Transformer can be loaded and used from open-source libraries such as Hugging Face

<u>Remarks:</u>

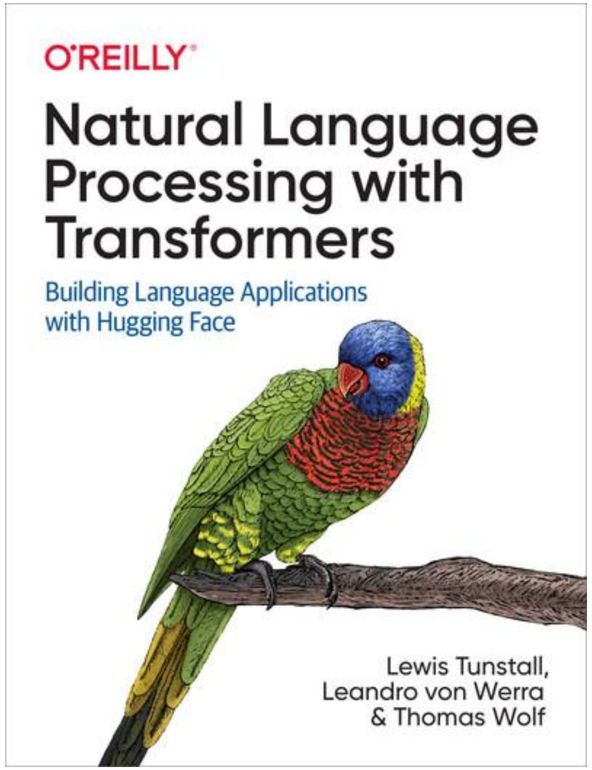
- This notebook comes from a **tutorial notebook** **freely accessible** (the original reference is given at the very beginning, in the first cell)

Hugging Face: *NLP with Transformers*

- Reference book written by **Hugging Face** employees
- Explain how to use pre-trained Transformer models to solve most of classic **NLP** use case
  - text generation
  - text summarization
  - text classification
  - text tagging
  - …
- Access to the book **is not free** but a **GitHub** repo containing all the main explanations and code is **freely** accessible (!): https://github.com/nlp-with-transformers/notebooks/tree/main

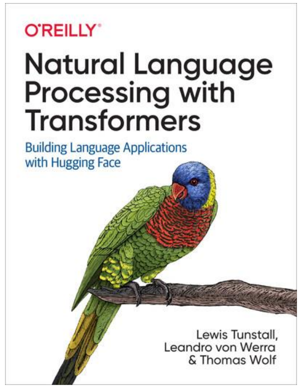


Hugging Face: *NLP with Transformers*



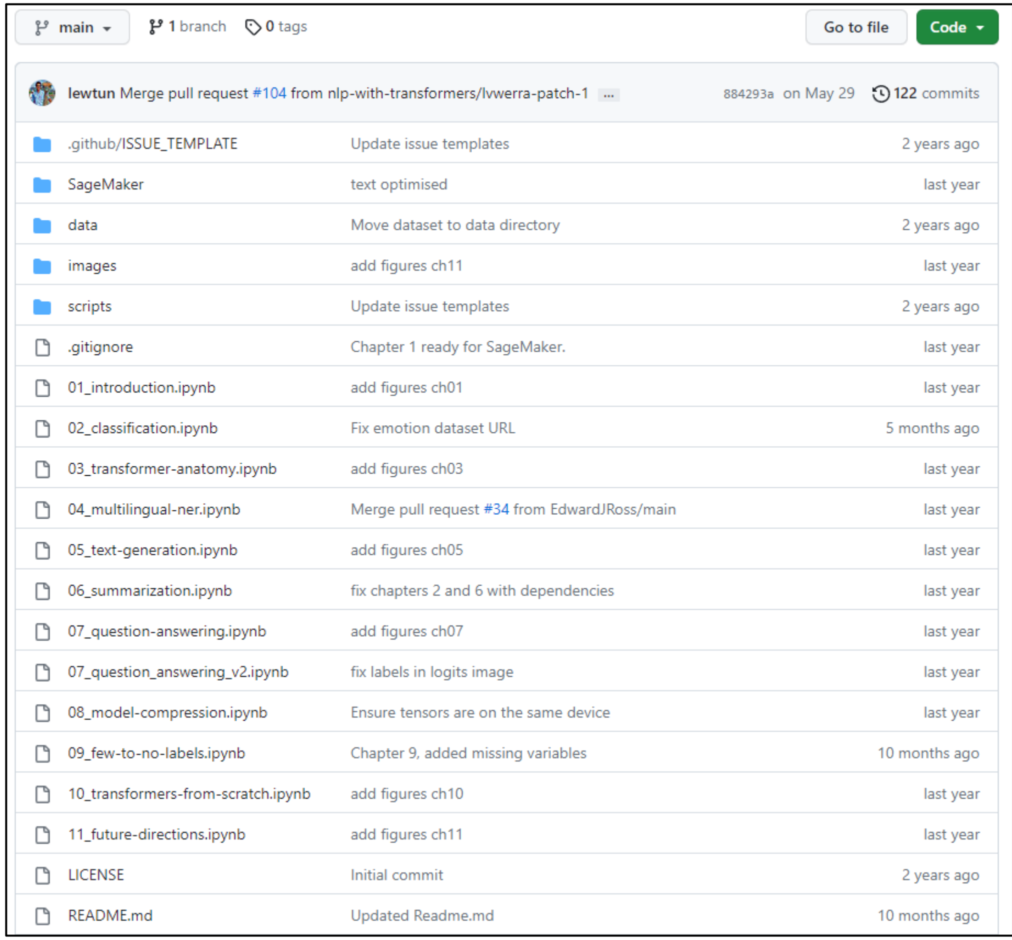
- Reference book written by **Hugging Face** employees
- Explain how to use pre-trained Transformer models to solve most of classic **NLP** use case
  - text generation
  - text summarization
  - **text classification** ⇨ can be useful for **question 3** of the **Graded Project**
  - text tagging
  - …
- Access to the book **is not free** but a **GitHub** repo containing all the main explanations and code is **freely** accessible (!): https://github.com/nlp-with-transformers/notebooks/tree/main

Hugging Face: *NLP with Transformers*

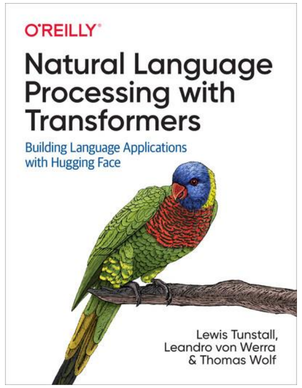


If you want to use these notebooks directly on Google Colab, you may need to:

- manually **copy-paste** the content (error with direct import)
- change the models by **lighter** versions
  - "gpt2" instead of "gpt2-xl"
  - "t5-base" instead of "t5-large"
  - …
- change the **datasets** since some are not available any longer



Hugging Face: *NLP with Transformers*

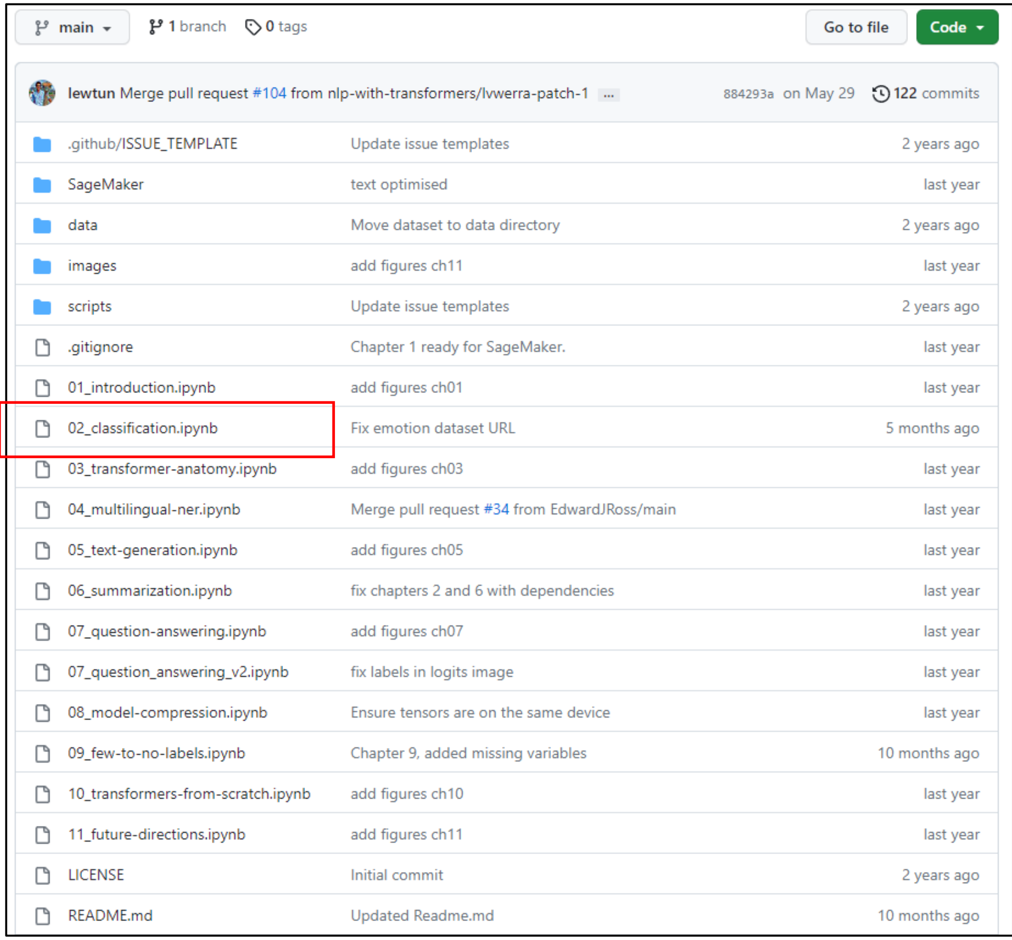


If you want to use these notebooks directly on Google Colab, you may need to:

- manually **copy-paste** the content (error with direct import)
- change the models by **lighter** versions
  - "gpt2" instead of "gpt2-xl"
  - "t5-base" instead of "t5-large"
  - …
- change the **datasets** since some are not available any longer

⇩

This is the case for ***02_classifcation.ipynb***
If you want to get some inspiration from there for question 3 of **the Graded Project**, there will be some adaptations to make.
However, you can simply get inspiration from exercise ***course4_classify_text_with_BERT.ipynb*** if that make things easier for you



Take-away from Course 4

- **LSTM** architectures are **historically important** but there **are not** the state-of-the-art **any more**
- The **"Attention" mechanism** allowed a **significant performance improvement** than using LSTM only
- It has been demonstrated in the seminal paper ***Attention is all you need*** **(2017)**, thanks to the **Transformer architecture** introduced in it, that "Attention" **alone** is enough to reach the state-of-the-art (no LSTM needed and no need to process a sequence in a specific order)
- **BERT** is an example of such **pre-trained Transformer models**, and it can be **fined-tuned** for specific tasks
- **Hugging Face** offers quite convenient tools **for using and fine-tuning BERT**
- Some **freely** accessible **code and resources** can be found online to help you get the most from the **Transformer pre-trained models** available (see e.g. the *NLP with Transformers* **GitHub** repo)

References

Books

- A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2019)
- L. Tunstall, L. Von Werra, T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*, (2022)
- F. Chollet, *Deep Learning with Python* (2021)

Online formations

- https://www.udemy.com/course/nlp-natural-language-processing-with-python
- https://www.coursera.org/specializations/natural-language-processing
- https://www.coursera.org/learn/natural-language-processing-tensorflow

Internet sites

- https://arxiv.org/abs/1706.03762 *(Attention is al you need)*
- https://www.youtube.com/watch?reload=9&v=OyFJWRnt_AY
- http://jalammar.github.io/illustrated-transformer/
- https://medium.com/dissecting-bert/dissecting-bert-part-1-d3c3d495cdb3
- https://camembert-model.fr/
- https://huggingface.co/
- https://lesdieuxducode.com/blog/2019/4/bert--le-transformer-model-qui-sentraine-et-qui-represente

Online notebooks

- https://colab.research.google.com/github/tensorflow/text/blob/master/docs/tutorials/transformer.ipynb
- https://colab.research.google.com/github/ziadloo/attention_keras/blob/master/examples/colab/LSTM.ipynb