

# Transformers and Large Language Models

Minghao Shao

# Pre-words

- The slides were adapted from Stanford CS224N, Natural Language Processing
- You should know deep learning basics

# Outline

## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

# What can transformers do?

- Text Generation

B

Write me a Pytorch class of a transformer in Python

T

Here is a basic implementation of a Transformer model:

python

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Transformer(nn.Module):
    def __init__(self, embedding_size, src_vocab_size, trg_vocab_size, src_p
        super(Transformer, self).__init__()
        self.src_word_embedding = nn.Embedding(src_vocab_size, embedding_size)
        self.src_position_embedding = nn.Embedding(max_length, embedding_size)
        self.trg_word_embedding = nn.Embedding(trg_vocab_size, embedding_size)
        self.trg_position_embedding = nn.Embedding(max_length, embedding_size)
        self.device = device
```

B

Translate "I have a cat. It is very cool!" into Chinese.



The sentence "I have a cat. It is very cool!" translates into Chinese as "我有一只猫。它非常"

酷!"

*original  
text*

"hello world!"



*tokens*

['hello', 'world', '!']



*token  
IDs*

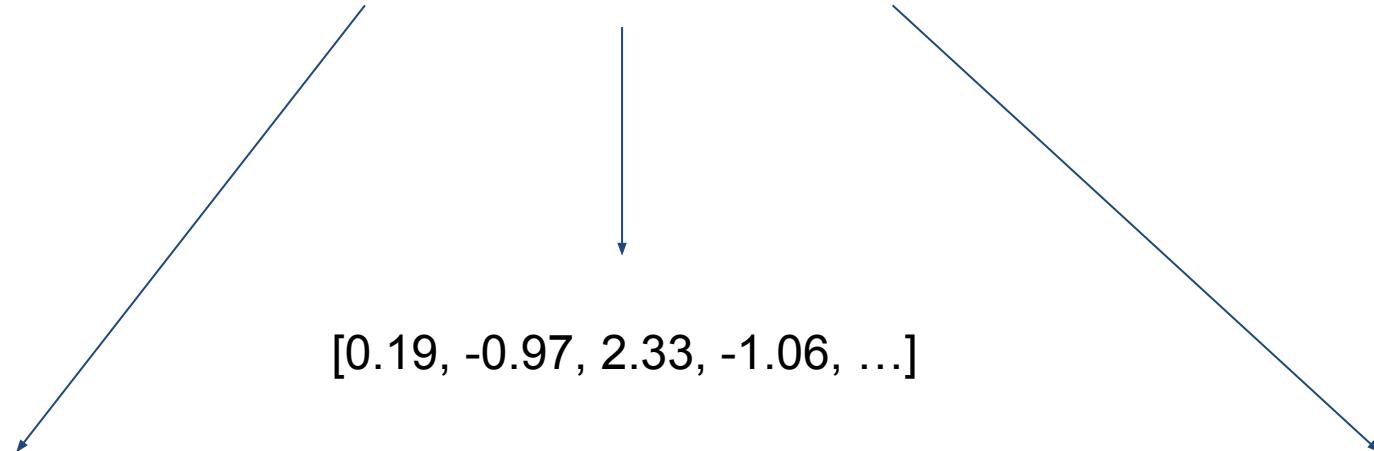
[7592, 2088, 999]

Embedding

[0.19, -0.97, 2.33, -1.06, ...]

[...]

[...]



# How are transformers trained?

- Text Generation (Training)

Input (English): I have a cat.

Output (French): J'ai un \_\_\_\_\_

label: chat

Each token is represented as an [embedding]-dimensional vector. For example, if your embedding is 5, then the “chat” token might be represented as [0.53, 1.2, 0.01, 0.275, 0.90]. It is converted using a **tokenizer** that comes along with the model.

# What can transformers do?

- Text Classification

Positive tweets:

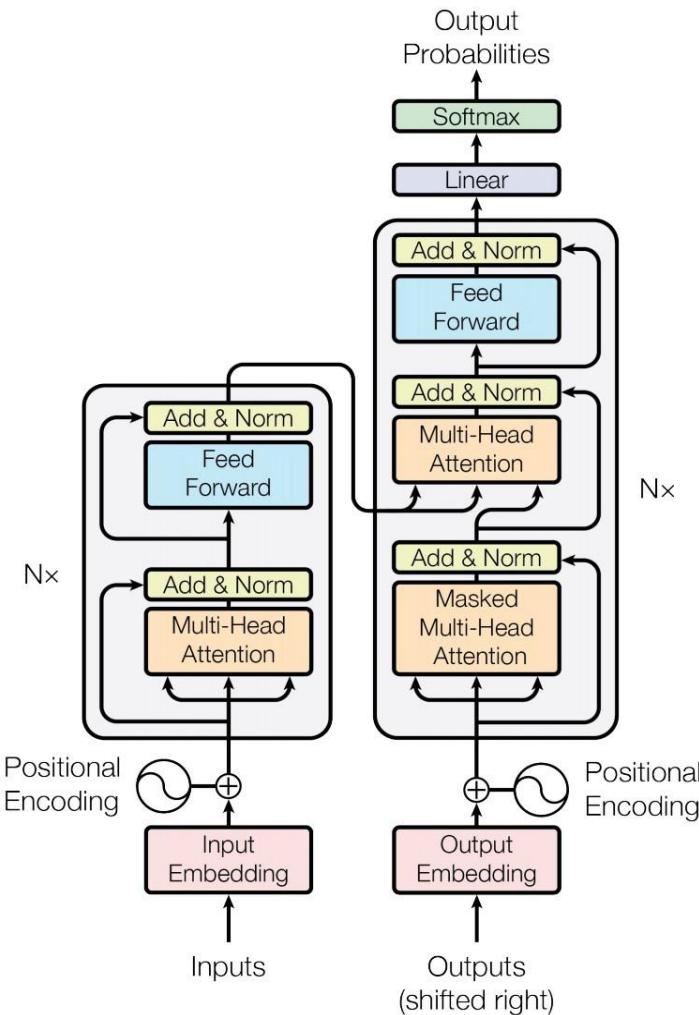
RT @JohnGGalt: Amazing—after years of attacking Donald Trump the media managed to turn #InaugurationDay into all about themselves.  
#MakeAme...

Negative tweets:

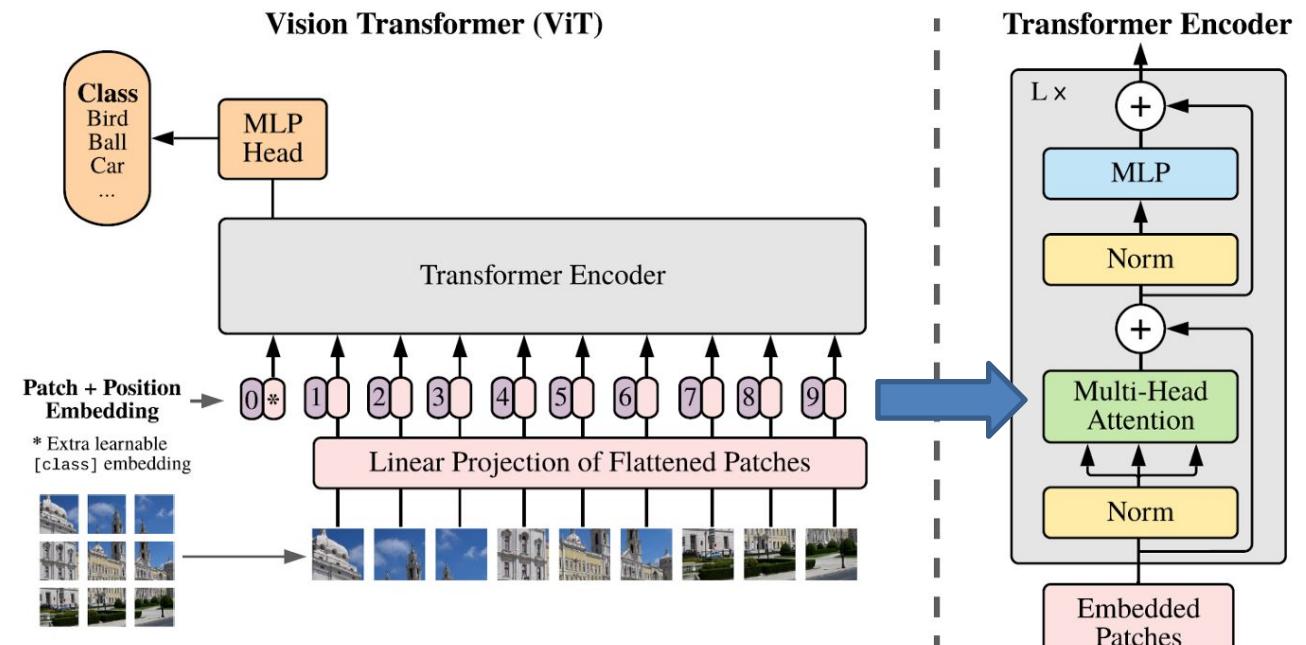
RT @Slate: Donald Trump's administration: "Government by the worst men."  
RT @RVAwonk: Trump, Sean Spicer, etc. all lie for a reason.

# What can transformers do?

- NLP: Machine Translation



- CV: Classifier / Object Detection



# What can transformers do?

- Audio Generation

<https://alxmamaev.medium.com/generating-music-with-ai-or-transformers-go-brrrr-3a3ac5a04126>

The screenshot shows a SoundCloud page with a user profile picture of a person with white hair. A red play button icon with the text "ALX" is overlaid on the image. Below it, the text "Transformer - Generation - 1" is displayed. To the right is a waveform visualization of the audio file, with a timestamp of "0:53". In the top right corner of the player, there is a "Share" button. Below the player, a list of six tracks is shown, each with a small profile picture, the track name, and a play button icon followed by a view count:

	ALX - Transformer - Generation - 1	▶ 1.4K
	ALX - Transformer - Generation - 2	▶ 1.4K
	ALX - Transformer - Generation - 3	▶ 847
	ALX - Transformer - Generation - 4	▶ 692
	ALX - Transformer - Generation - 5	▶ 589
	ALX - Transformer - Generation - 6	▶ 530

Image, video, movements, etc... We call it **multimodal**.

# Outline

## Part 1: Transformer

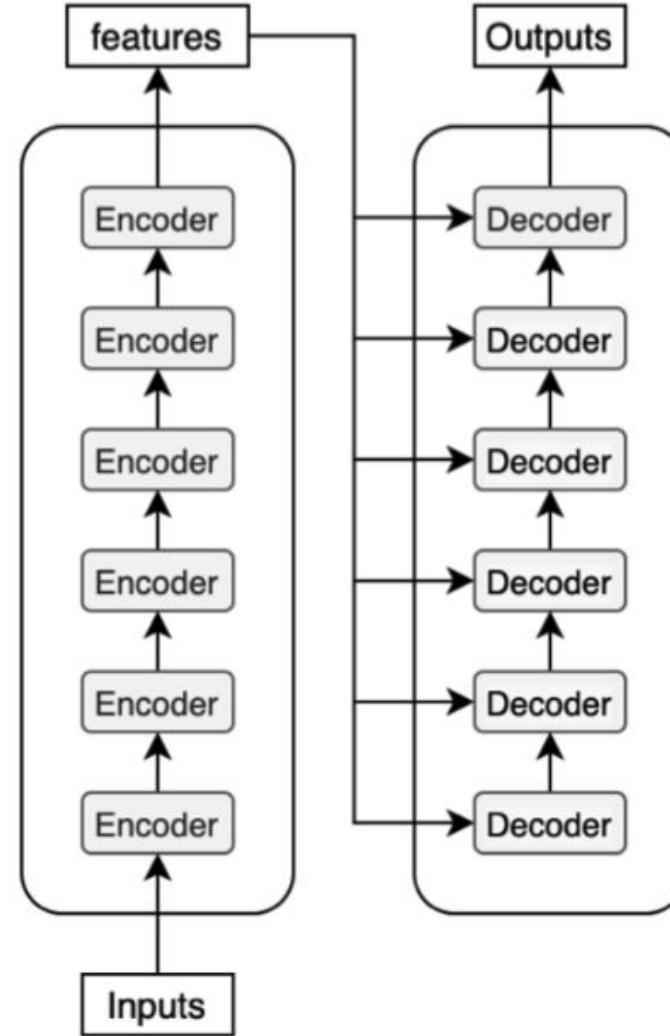
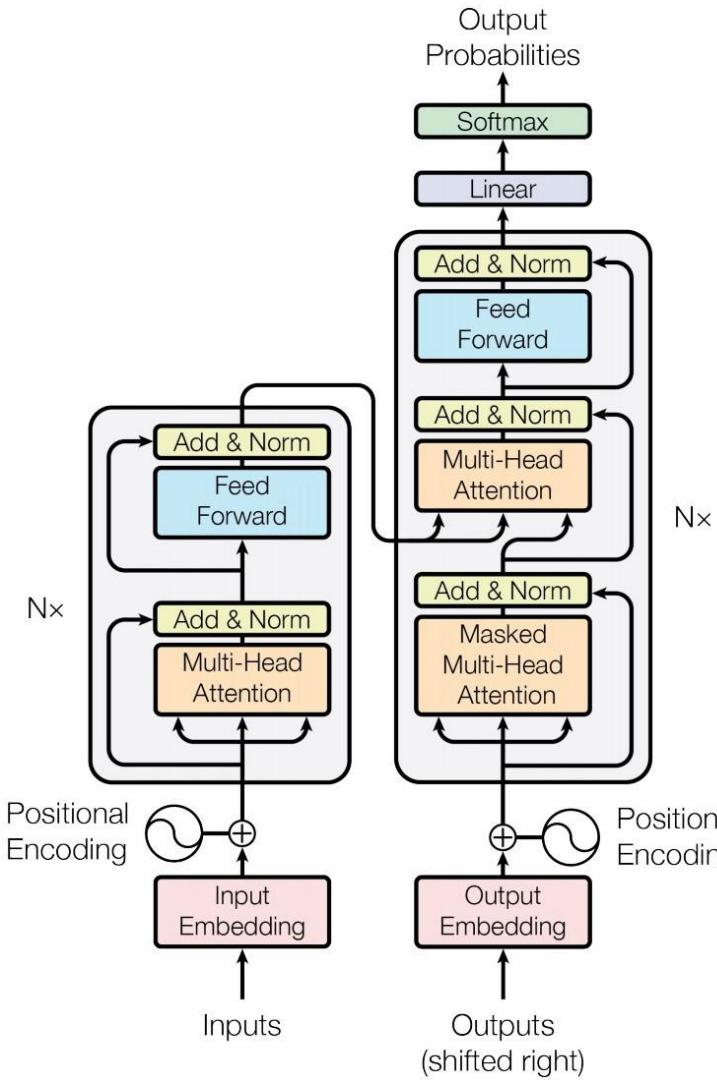
- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

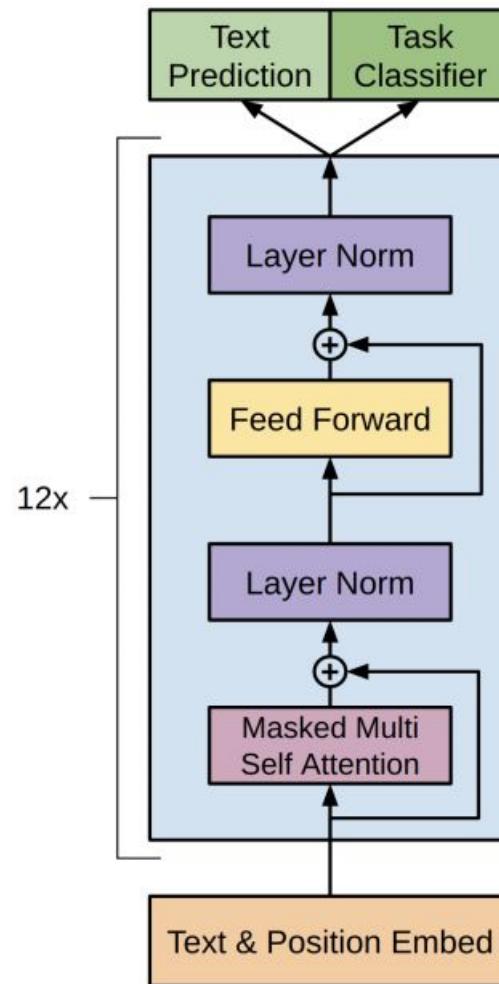
# Different Types of Transformers

- Encoder-Decoder (2017)

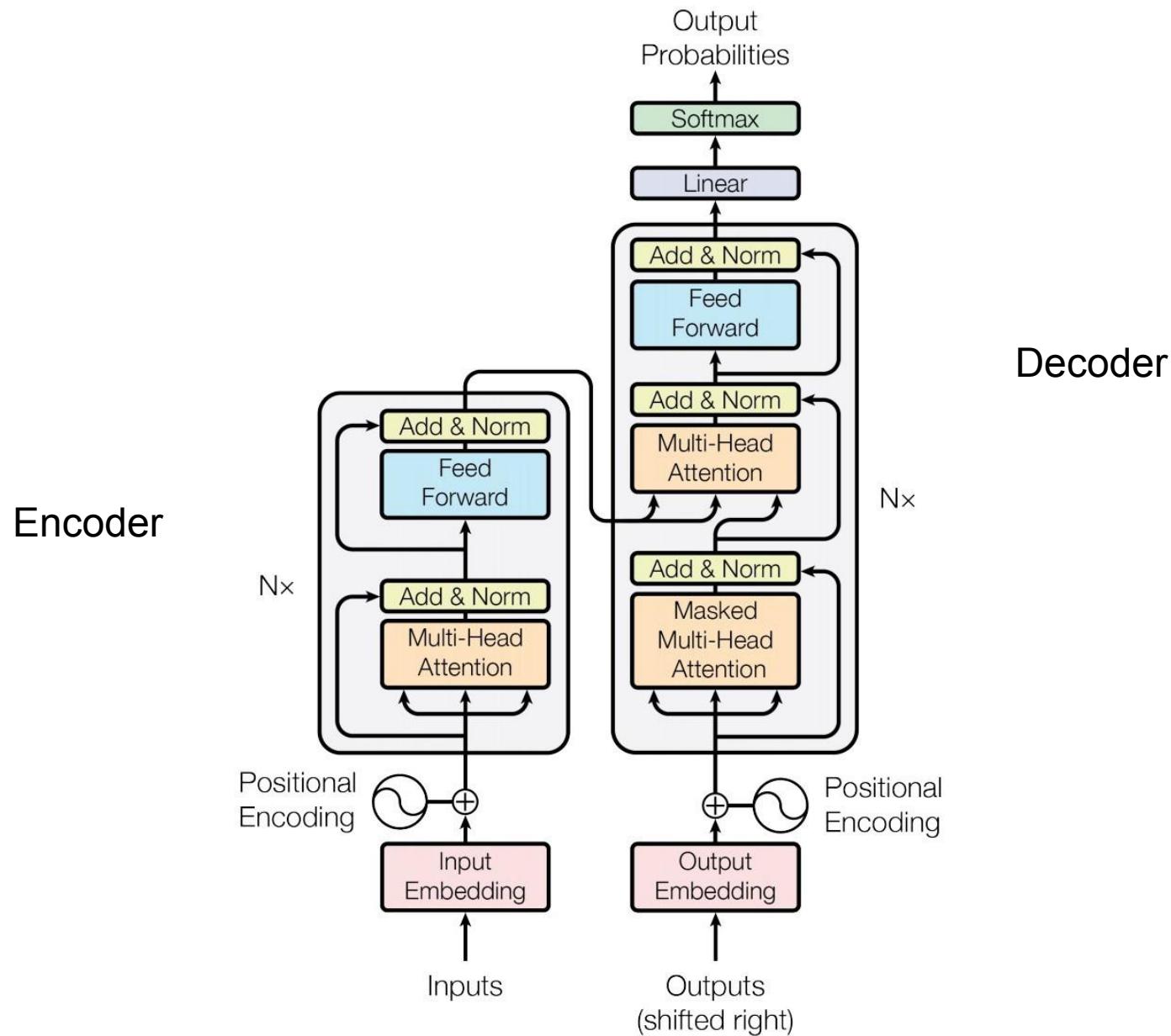


# Different Types of Transformers

- Decoder-Only (Openai GPT-2, 2019)
- Encoder-Only (BERT, 2018)



# Encoder-Decoder Transformer Architecture



# Outline

## Part 1: Transformer

- What can transformers do
- Different types of transformer
- **Why is transformer better than RNN**
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

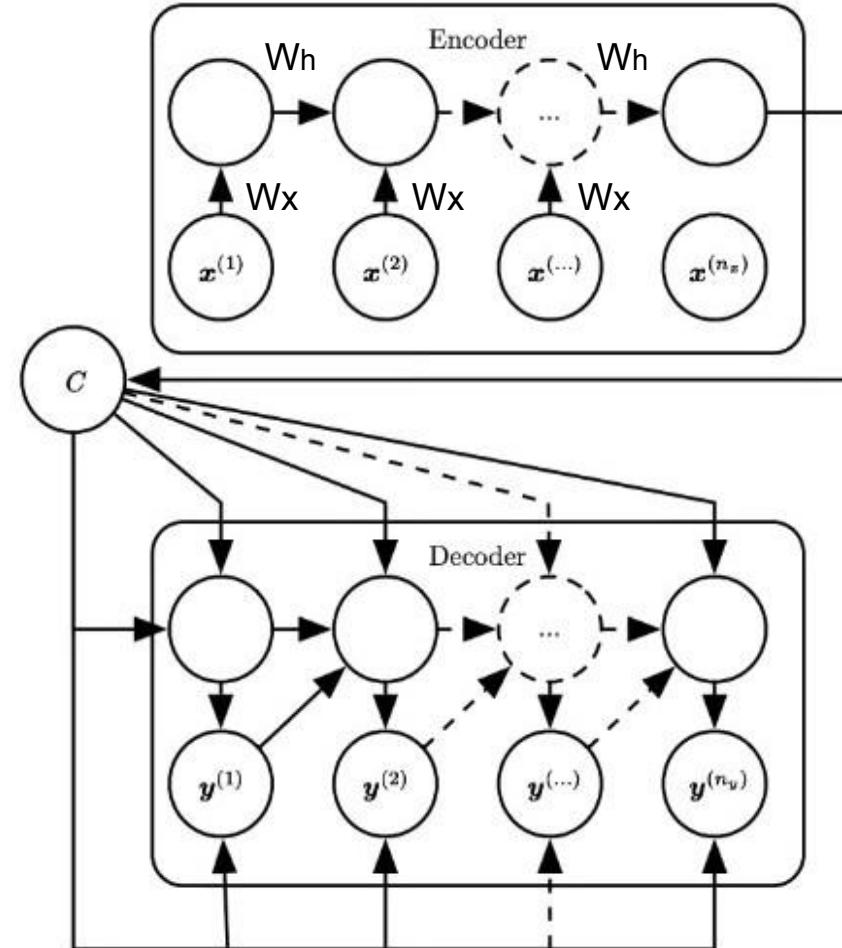
## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

# Why is transformer better than RNN?

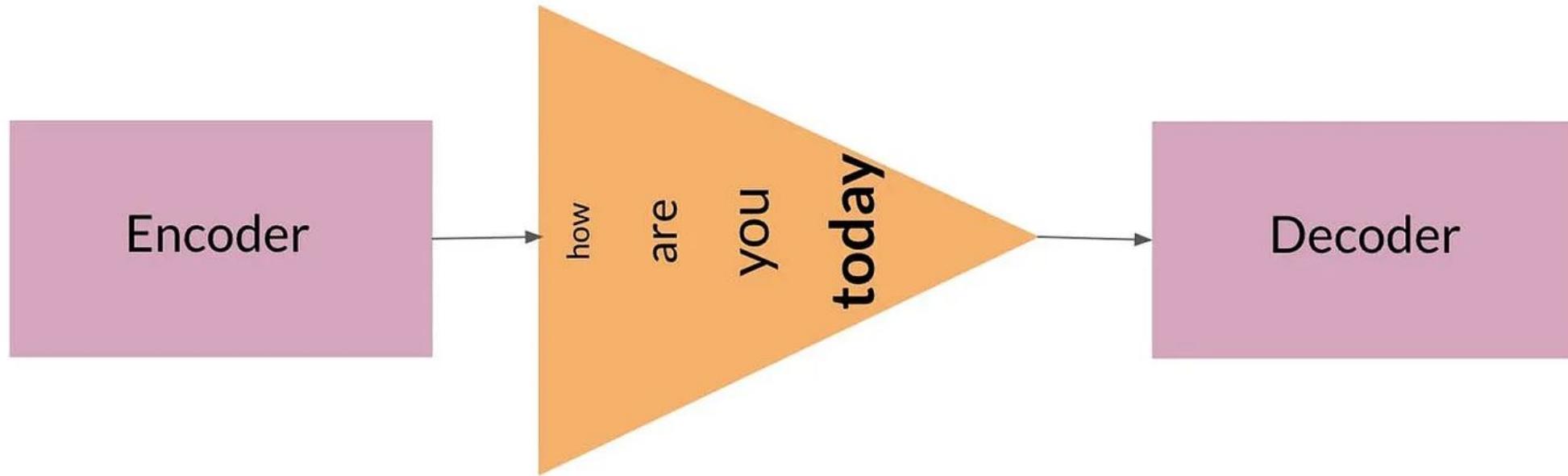
## Decoder-Encoder Model

Les pauvres sont démunis  
(French)



The poor do not have any money  
(English)

# Information Bottleneck



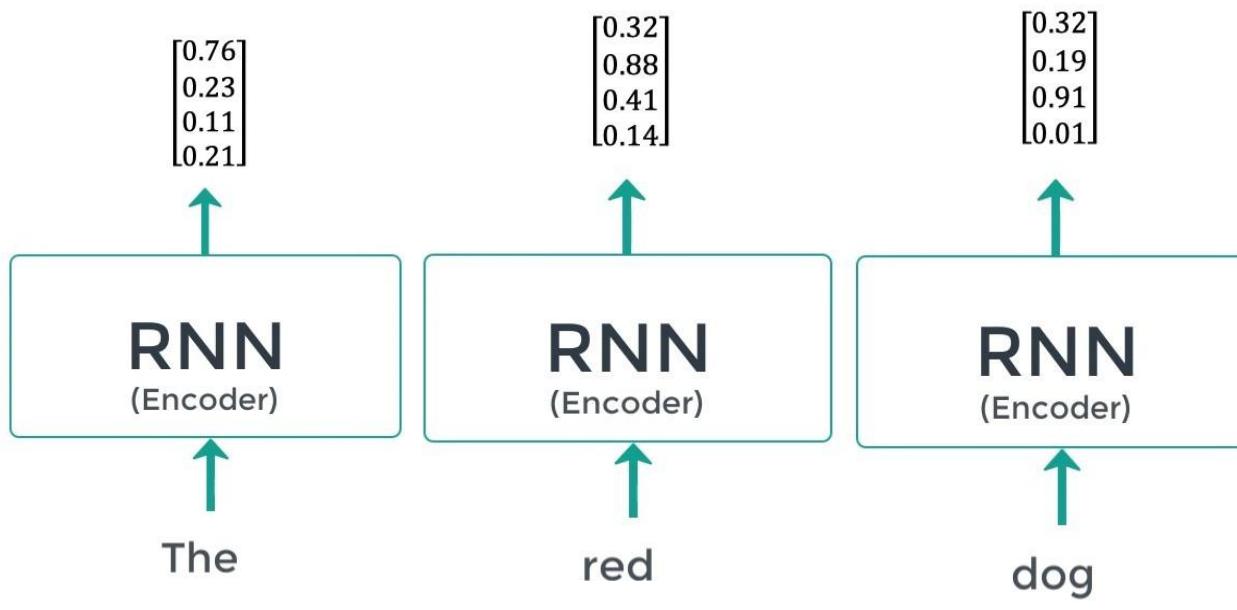
Forgets about the early tokens

Gradient vanishing and gradient explosion

# Motivations of Attention

## RNNs

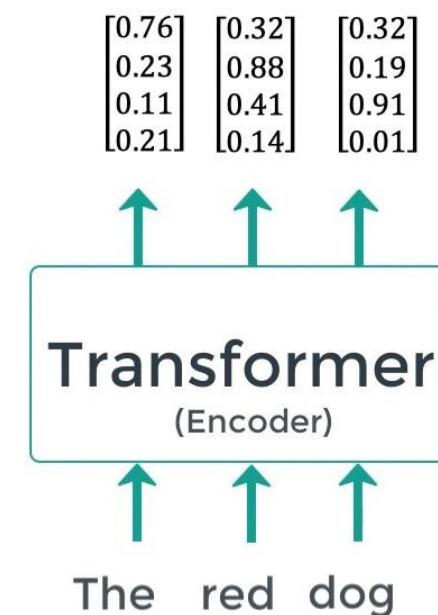
1. Long-range Dependencies
2. Gradient Vanishing/Exploding
3. Large #Training Steps
4. Recurrence prevents Parallel computation



## Transformers

1. Facilitate Long-range dependencies
2. No Gradient Vanishing/Exploding
3. Fewer #Training Steps
4. Parallelization

VS



# Attention Mechanism

Solution to the bottleneck problem: *Attention*

**Key idea:** On each step of the decoder, *focus on a particular part* of the source sequence

# Outline

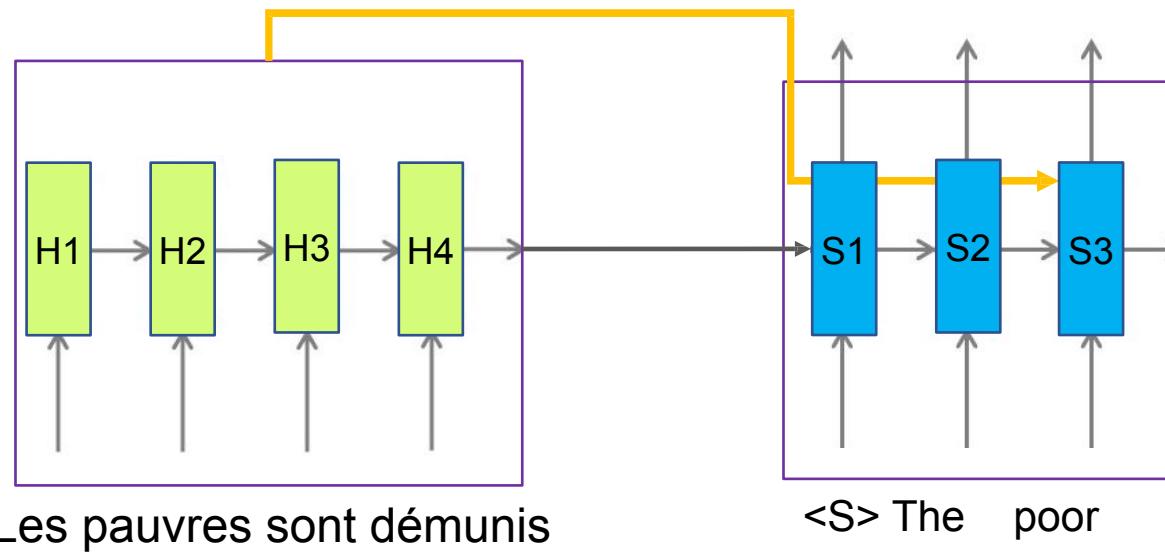
## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- **Attention mechanism**
  - **General Structure**
    - Query, Key & Value
    - Multi-Head Attention
    - Self-Attention

## Part 2: Large Language Models

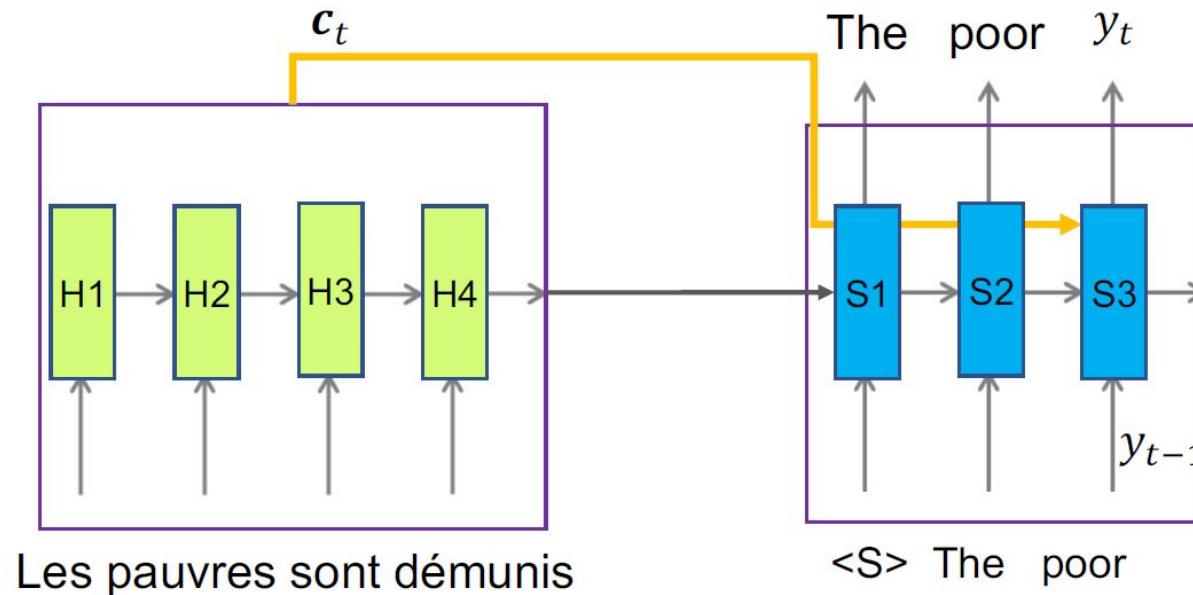
- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

# Attention Mechanism



Random Access Memory: Retrieve as Needed

# Attention Mechanism

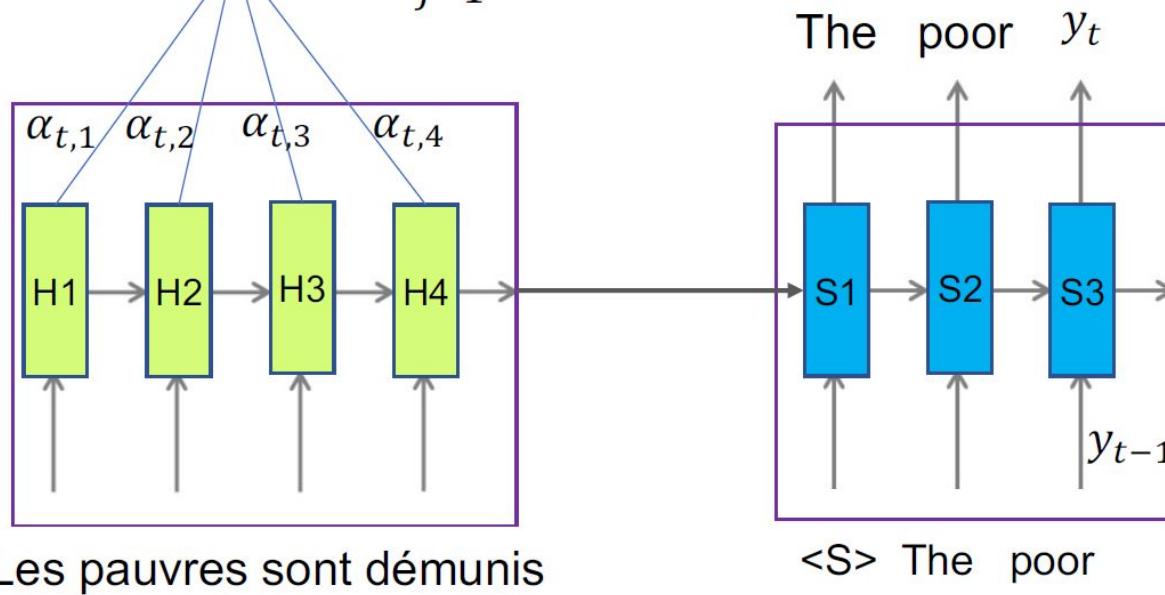


$$p(y_t | y_{t-1}, \dots, y_1, x) = f(y_{t-1}, s_t, c_t)$$

# Attention Mechanism

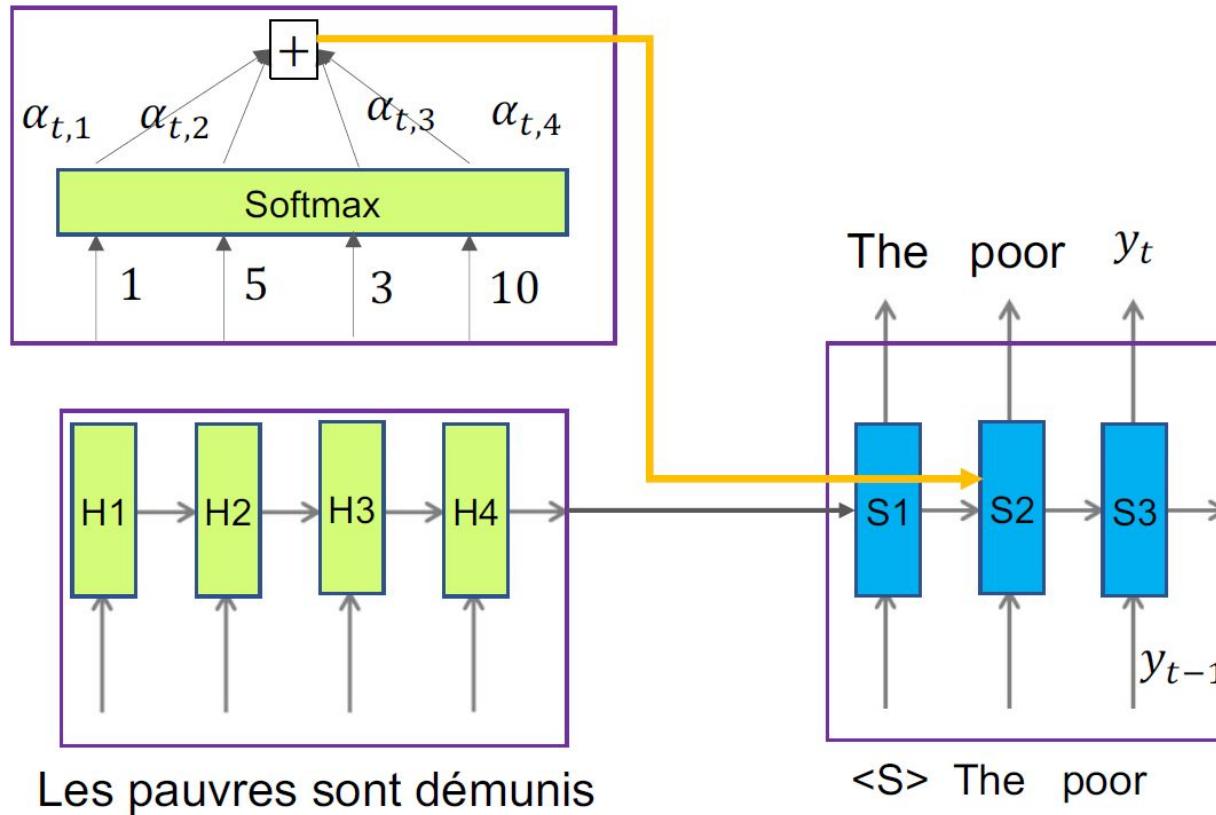
Get weighted sum

$$\mathbf{c}_t = \sum_{j=1}^T \alpha_{t,j} \mathbf{h}_j$$



# Attention Mechanism

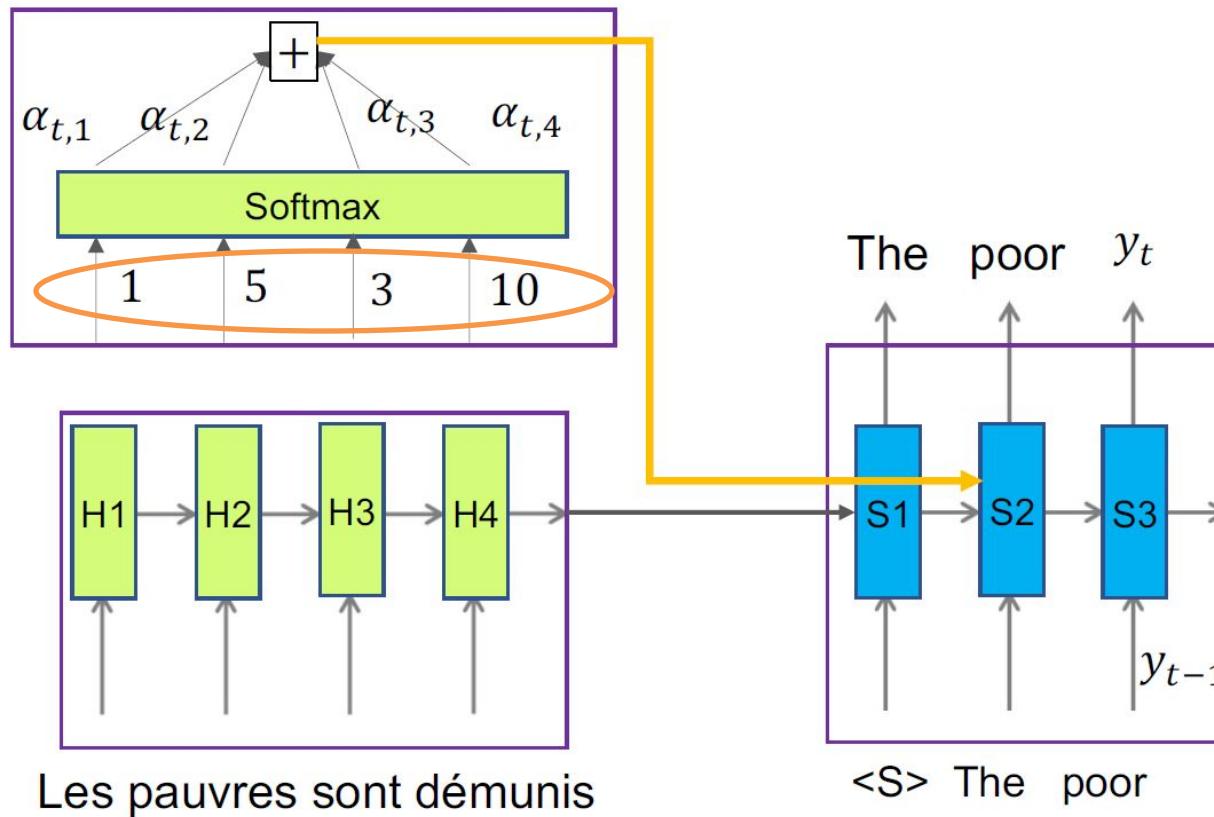
$$\alpha_{t,j} = \frac{\exp(Score_{t,j})}{\sum_{k=1}^T \exp(Score_{t,k})} = \text{Softmax}(\text{scores})$$



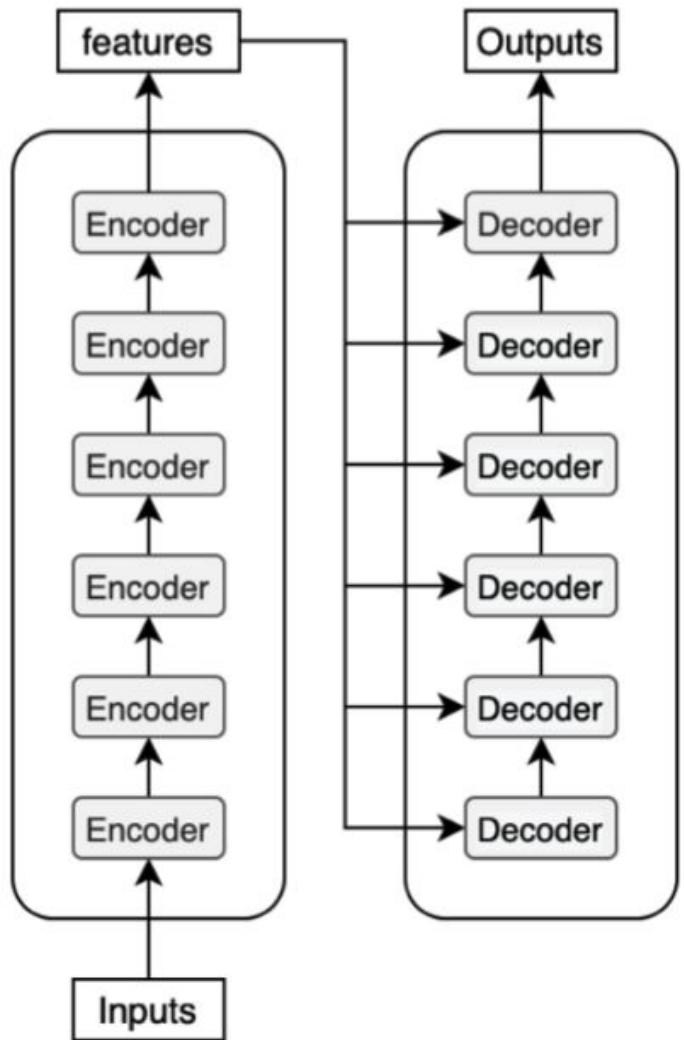
# Attention Mechanism

$$\alpha_{t,j} = \frac{\exp(Score_{t,j})}{\sum_{k=1}^T \exp(Score_{t,k})} = \text{Softmax}(\text{scores})$$

How are the weights calculated?



# Attention Mechanism



All the ↑ are passing a hidden state (h). It is of shape [ batch size, text length, embedding ]  
text length is the input length for encoder; output length for decoder.

Example:

Input: Les pauvres sont démunis  
Output: The poor do \_\_

Batch size = 1, embedding = 1024. What is the hidden states' shape for encoder and decoder?

Answer:

Input: [ <start>, 'Les', ' pauvres', ' sont', ' démunis', <end> ]  
[ 1, 6, 1024 ]  
Output: [ <start>, 'The', ' poor', ' do' ]  
[ 1, 4, 1024 ]

# Outline

## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- **Attention mechanism**
  - General Structure
  - **Query, Key & Value**
  - Multi-Head Attention
  - Self-Attention

## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

# Attention Mechanism

Mimics *Retrieval Process* in a database:

- Query
- Key
- Value

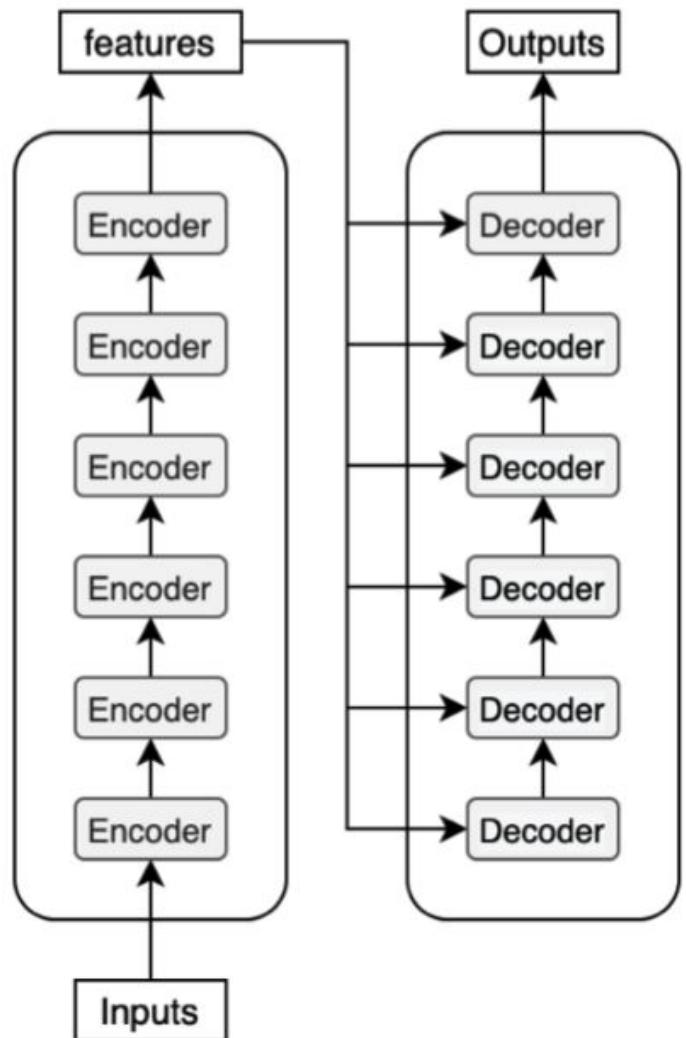
*Quer*

```
-- Find any employee born in October  
y  
SELECT *  
FROM employee  
WHERE birth_date LIKE '____-10%';
```

Value	Value	Value	Key	Value	Value	Value	Value
Employee							
emp_id	first_name	last_name	birth_date	sex	salary	super_id	branch_id
100	David	Wallace	1967-11-17	M	250,000	NULL	1
101	Jan	Levinson	1961-05-11	F	110,000	100	1
102	Michael	Scott	1964-03-15	M	75,000	100	2
103	Angela	Martin	1971-06-25	F	63,000	102	2
104	Kelly	Kapoor	1980-02-05	F	55,000	102	2
105	Stanley	Hudson	1958-02-19	M	69,000	102	2
106	Josh	Porter	1969-09-05	M	78,000	100	3
107	Andy	Bernard	1973-07-22	M	65,000	106	3
108	Jim	Halpert	1978-10-01	M	71,000	106	3

Success	5:47 PM						
1 rows	0.045 seconds						
<hr/>							
emp_id	first_name	last_name	birth_date	sex	salary	super_id	branch_id
108	Jim	Halpert	1978-10-01	M	71000	106	3

# Attention Mechanism



$$Q = \text{HiddenStates} \times W_q$$

$$K = \text{HiddenStates} \times W_k$$

$$V = \text{HiddenStates} \times W_v$$

Each is of shape [ batch size, text length, qkv-embedding ]

$$\text{Attention}(Query, Key, Value) = \sum_i \text{Similarity}(Query, Key_i) * Value_i$$

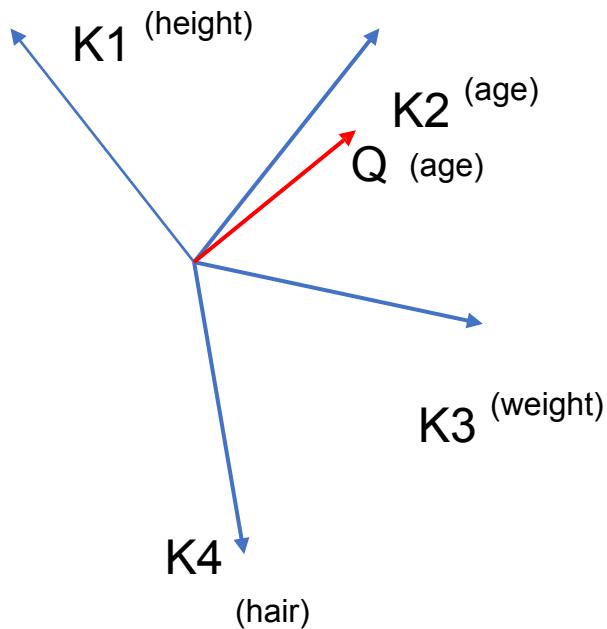
$$c_t = \sum_{j=1}^T \alpha_{t,j} h_j$$

Attention = Q K<sup>T</sup> V, with a little twist...

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

# Attention Mechanism

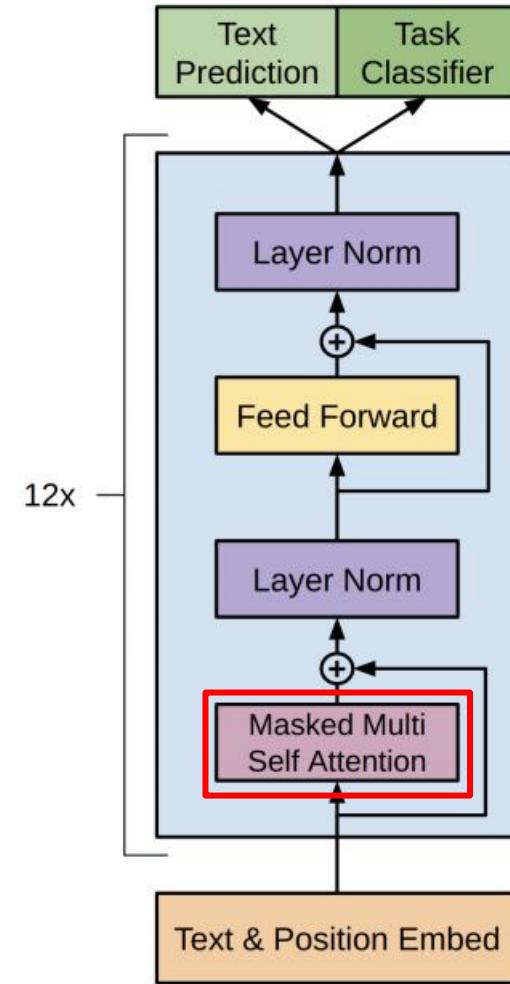
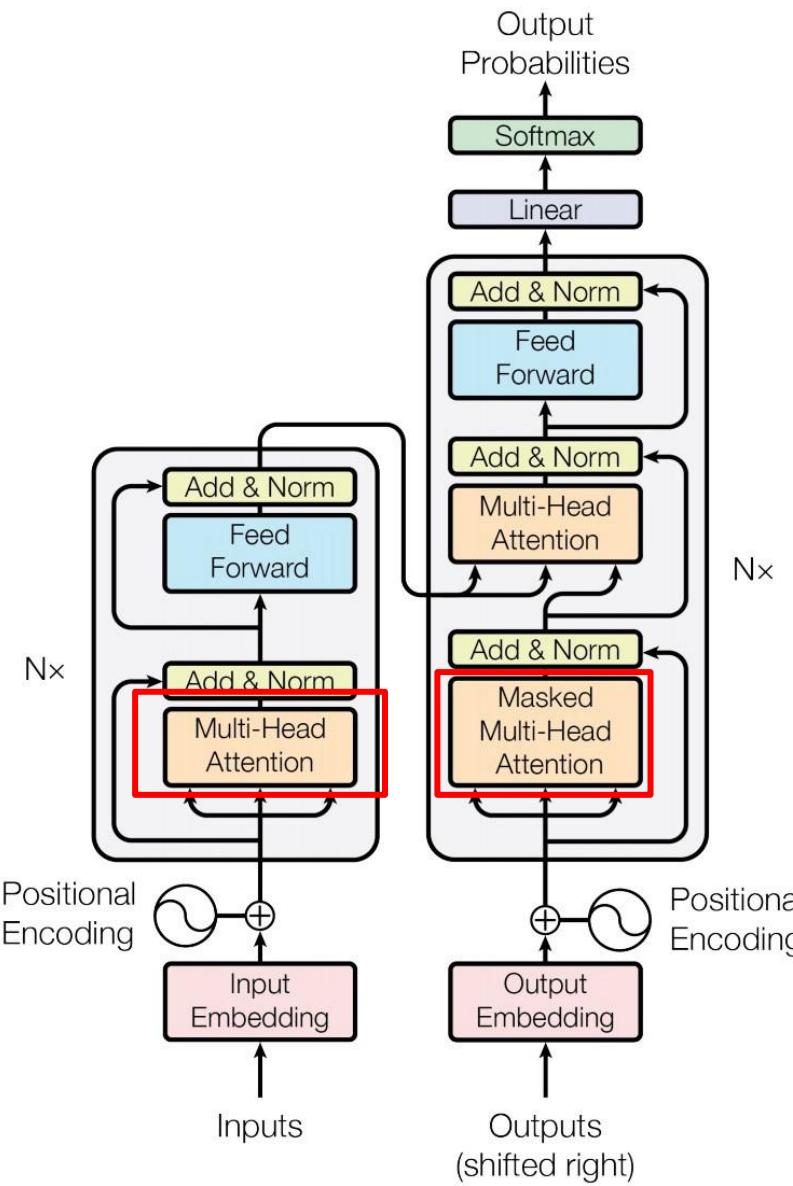
$$\text{Attention}(\text{Query}, \text{Key}, \text{Value}) = \sum_i \text{Similarity}(\text{Query}, \text{Key}_i) * \text{Value}_i$$



$$\begin{matrix} \langle Q, K \rangle & V \\ \begin{array}{|c|} \hline 0.1 \\ \hline 0.9 \\ \hline 0.2 \\ \hline 0.1 \\ \hline \end{array} & \times \quad \begin{array}{|c|} \hline 171 \\ \hline 23 \\ \hline 130 \\ \hline 1 \\ \hline \end{array} = \quad \begin{array}{|c|} \hline 17.1 \\ \hline 20.7 \\ \hline 26 \\ \hline 0.1 \\ \hline \end{array} \end{matrix}$$

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

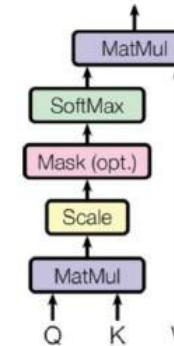
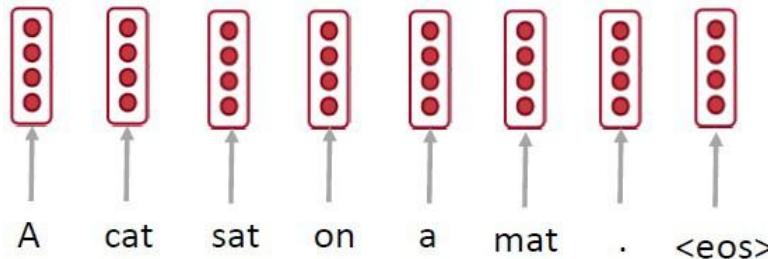
# Self Attention



# Self-Attention: Running Example

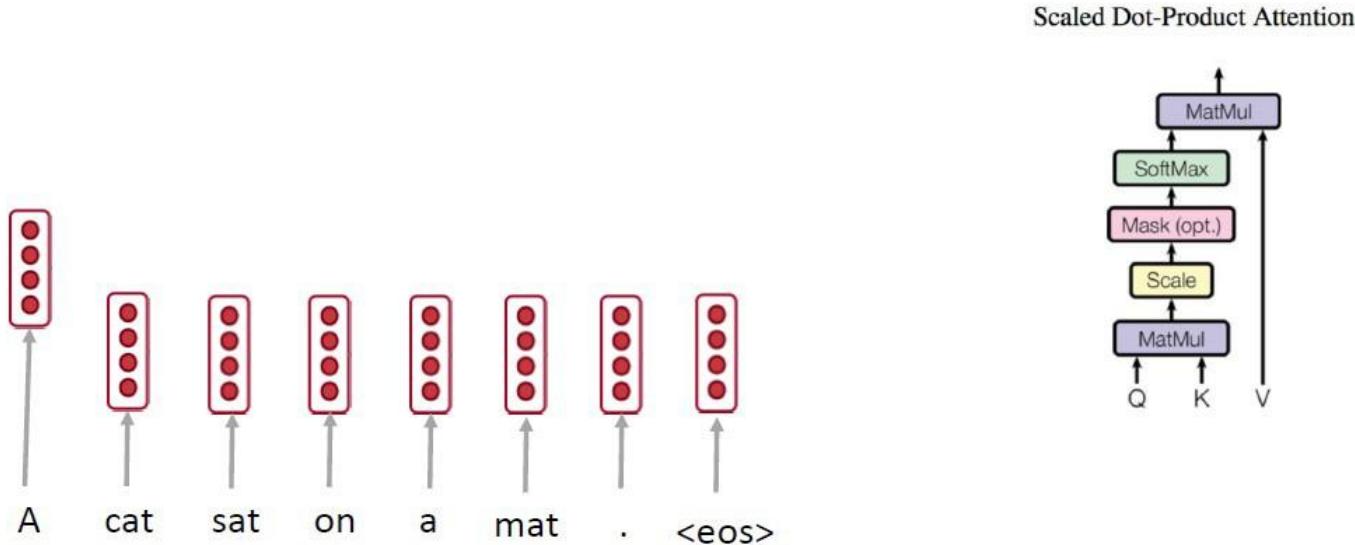
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention



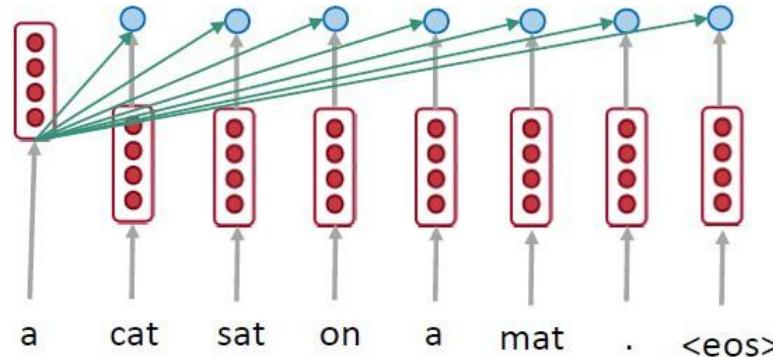
# Self-Attention: Running Example

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

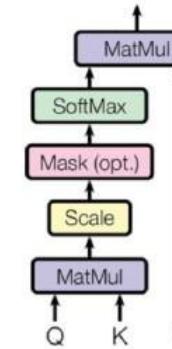


# Self-Attention: Running Example

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

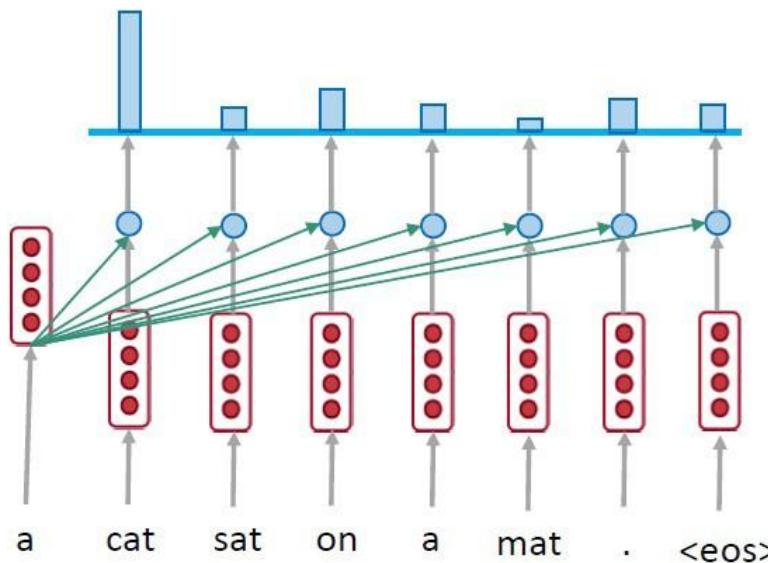


Scaled Dot-Product Attention

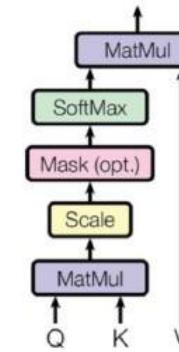


# Self-Attention: Running Example

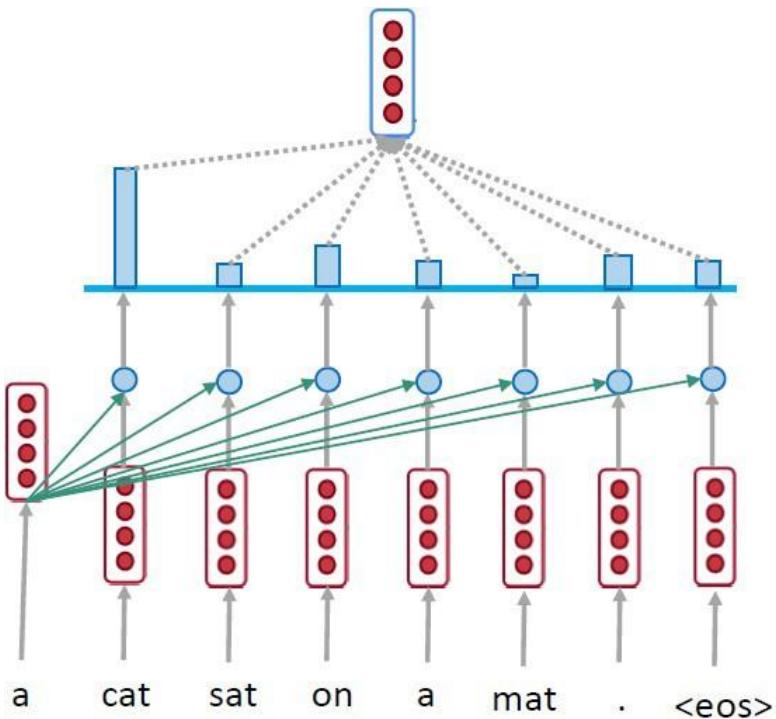
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Scaled Dot-Product Attention

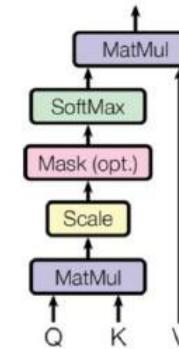


# Self-Attention: Running Example

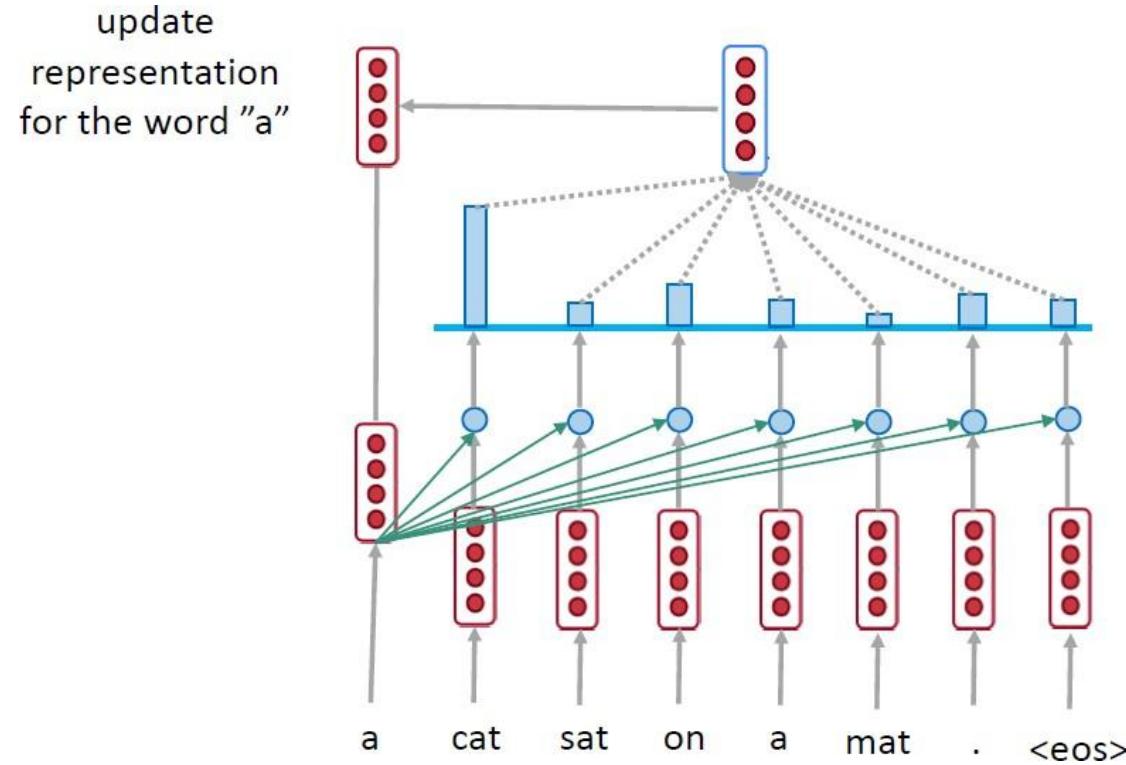


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

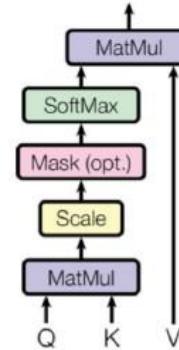


# Self-Attention: Running Example

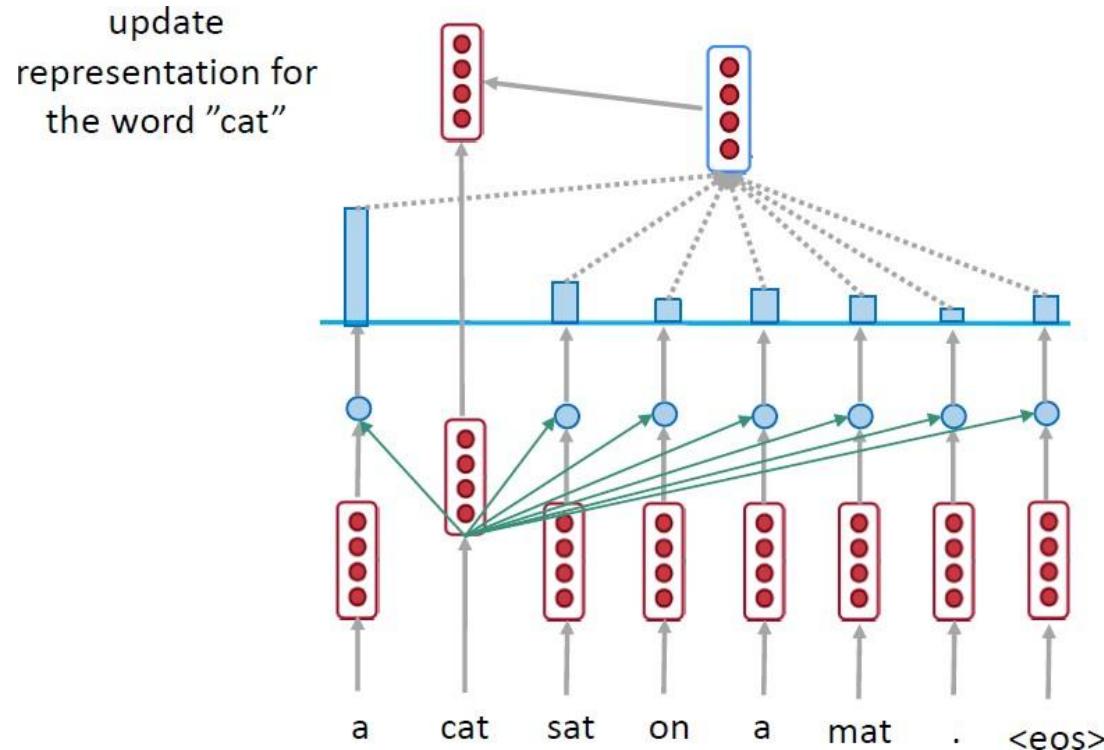


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

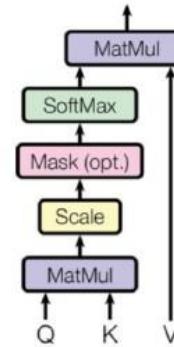


# Self-Attention: Running Example

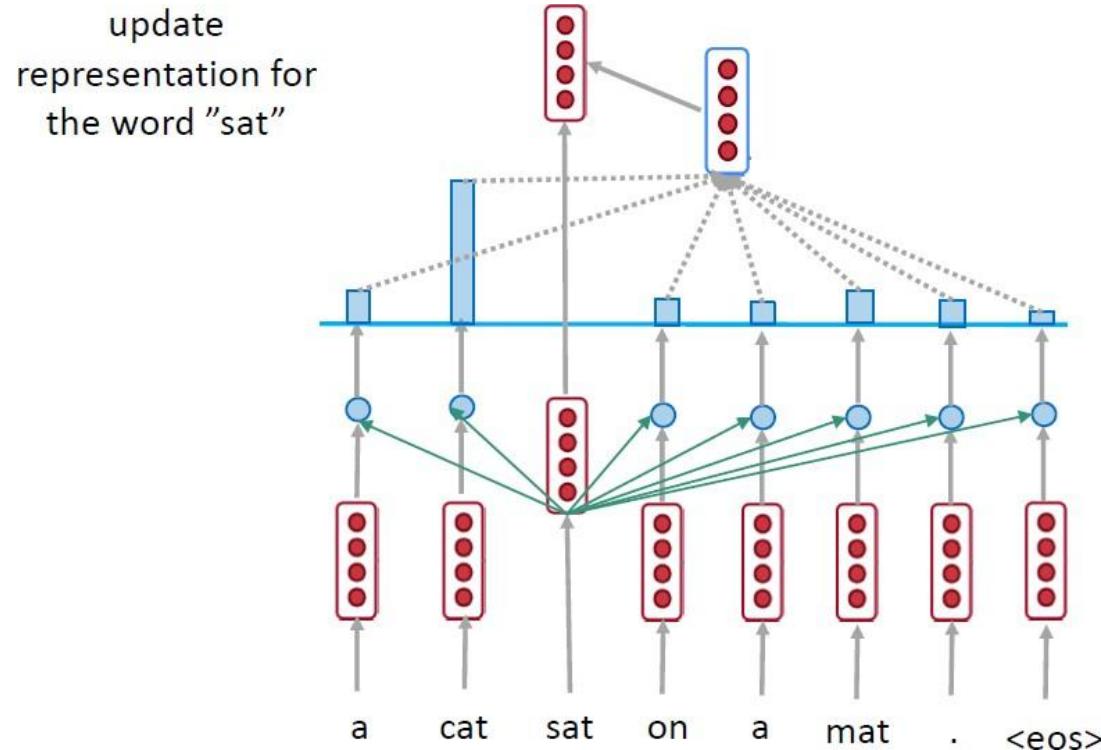


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

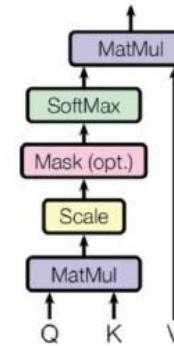


# Self-Attention: Running Example

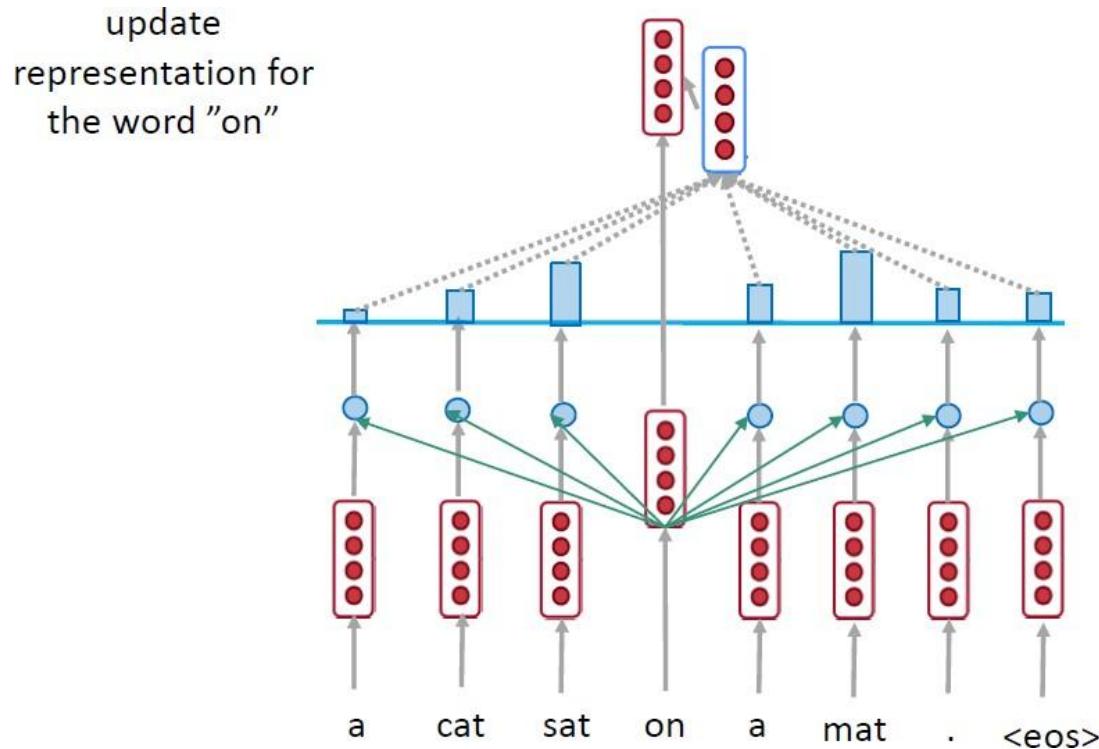


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

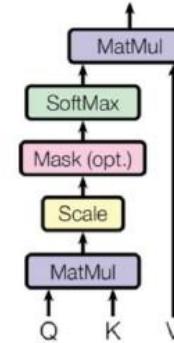


# Self-Attention: Running Example



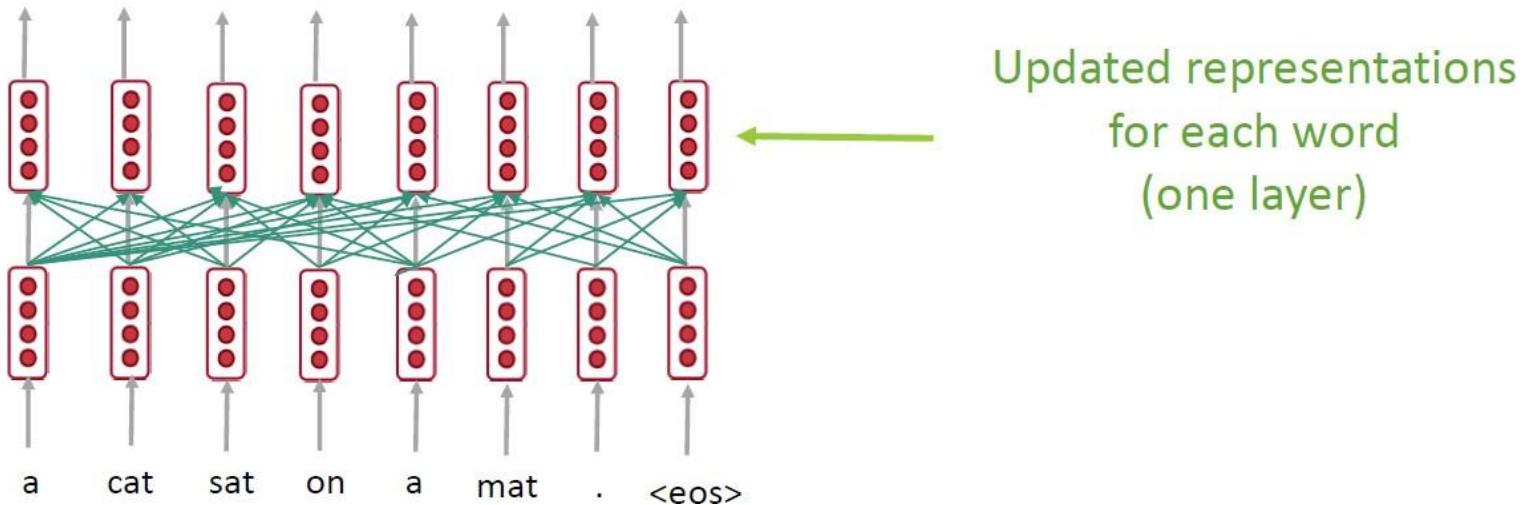
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

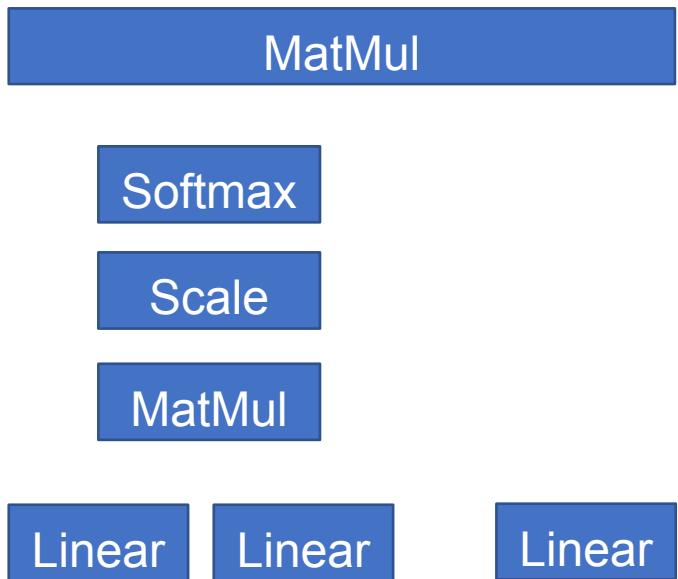


# Self-Attention: Running Example

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



# Self-Attention



The diagram illustrates the Self-Attention process. On the left, the input sequence "Les pauvres sont démunis" is shown. To its right, two matrix multiplications are performed:

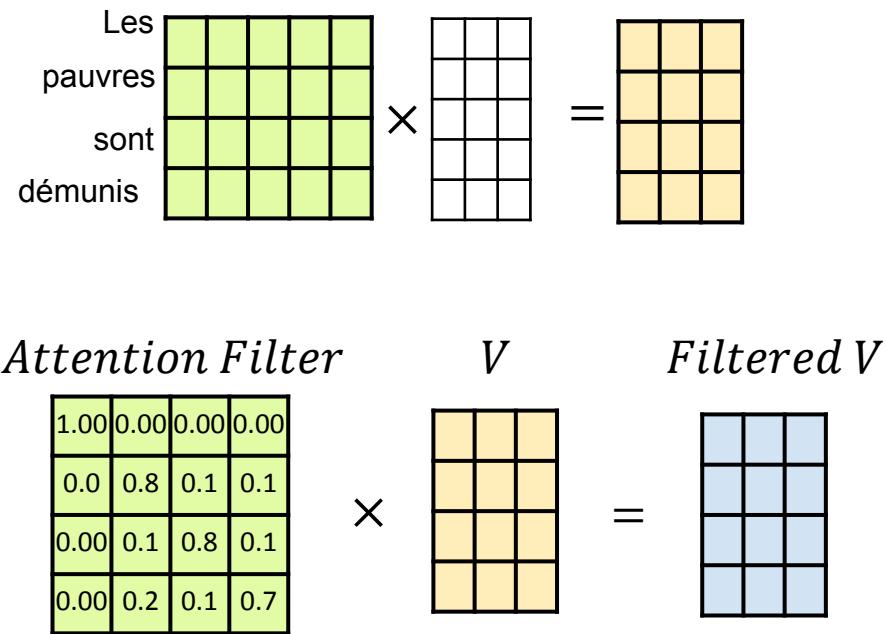
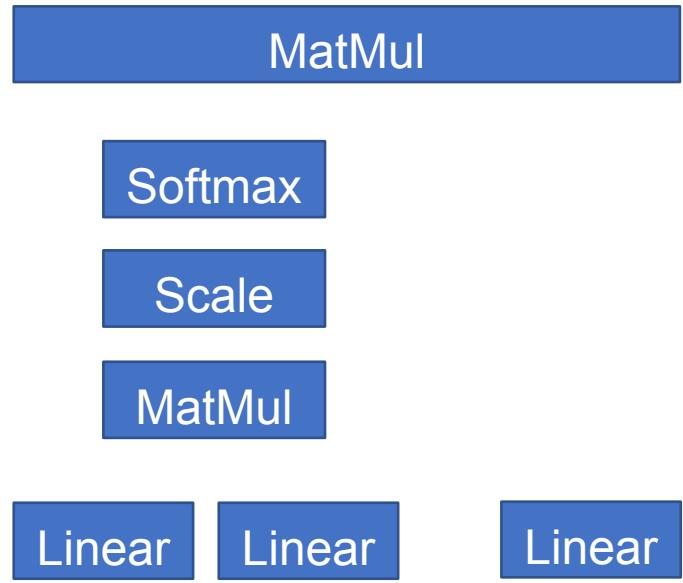
- The first multiplication shows the input sequence being multiplied by a weight matrix  $WQ$  to produce the *Query* matrix.
- The second multiplication shows the input sequence being multiplied by a weight matrix  $WK$  to produce the *Key* matrix.

The *Query* matrix has green cells, while the *Key* matrix has light blue cells. The *Query* matrix is labeled *Query* and the *Key* matrix is labeled *Key*.

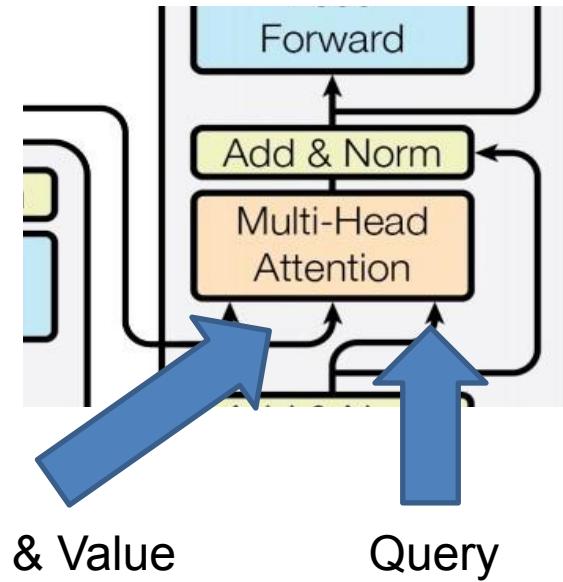
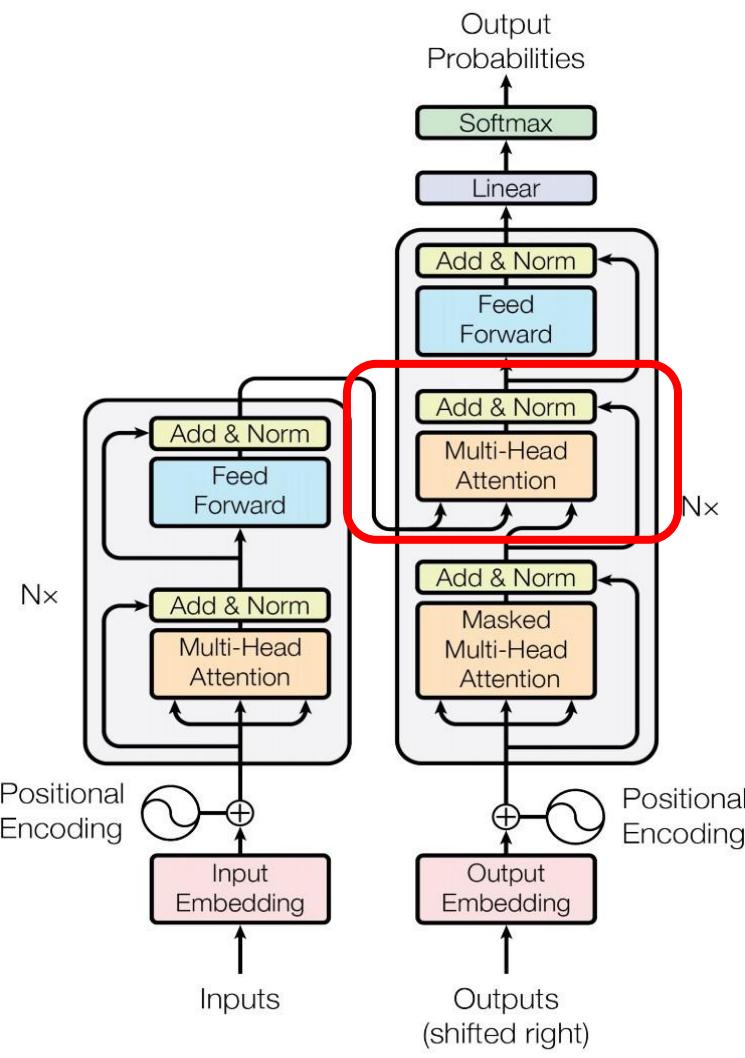
*Attention Filter*

1.00	0.00	0.00	0.00
0.0	0.8	0.1	0.1
0.00	0.1	0.8	0.1
0.00	0.2	0.1	0.7

# Self-Attention



# Cross Attention

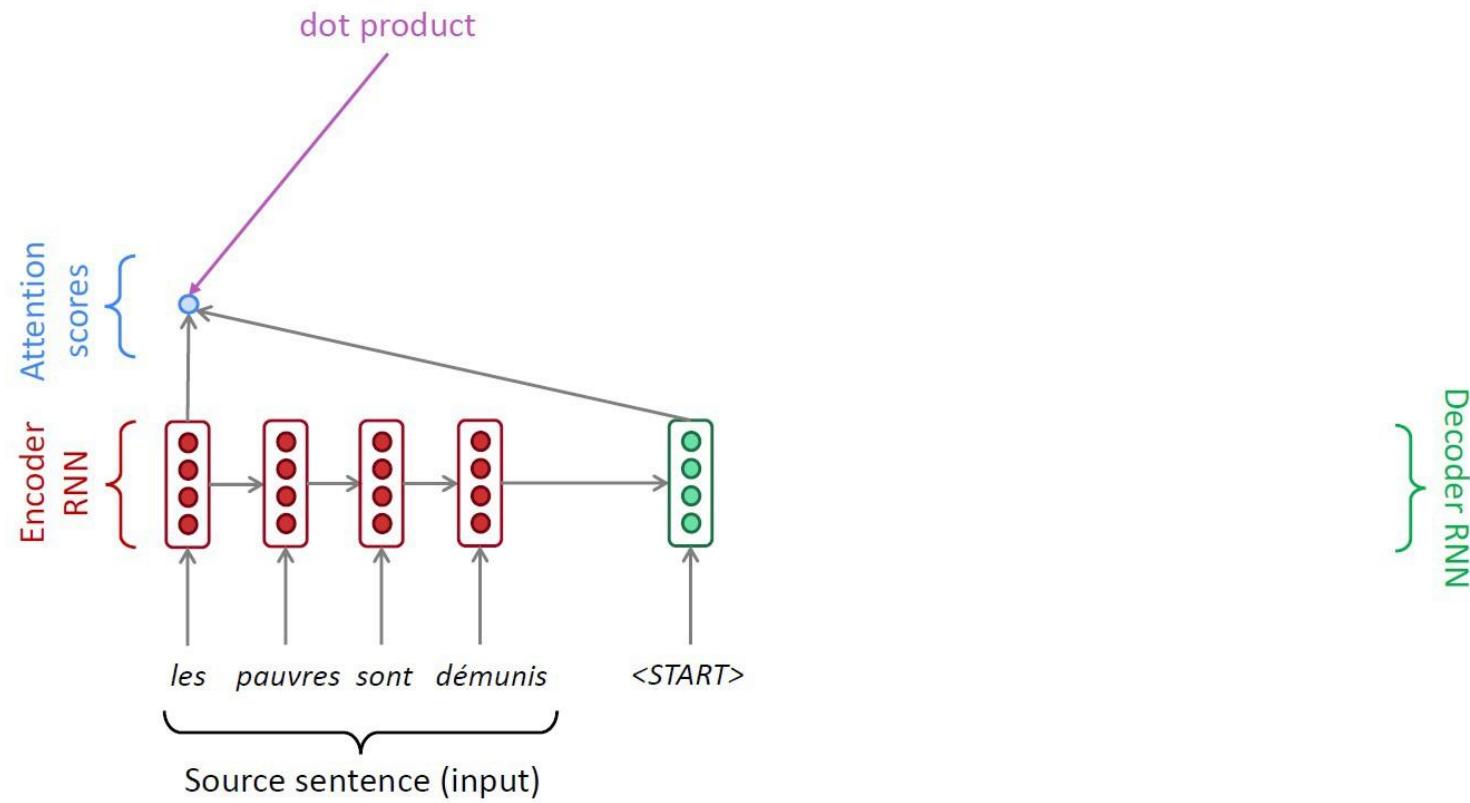


Key & Value shape: [input length, qkv-embedding]  
Query shape: [output length, qkv-embedding]

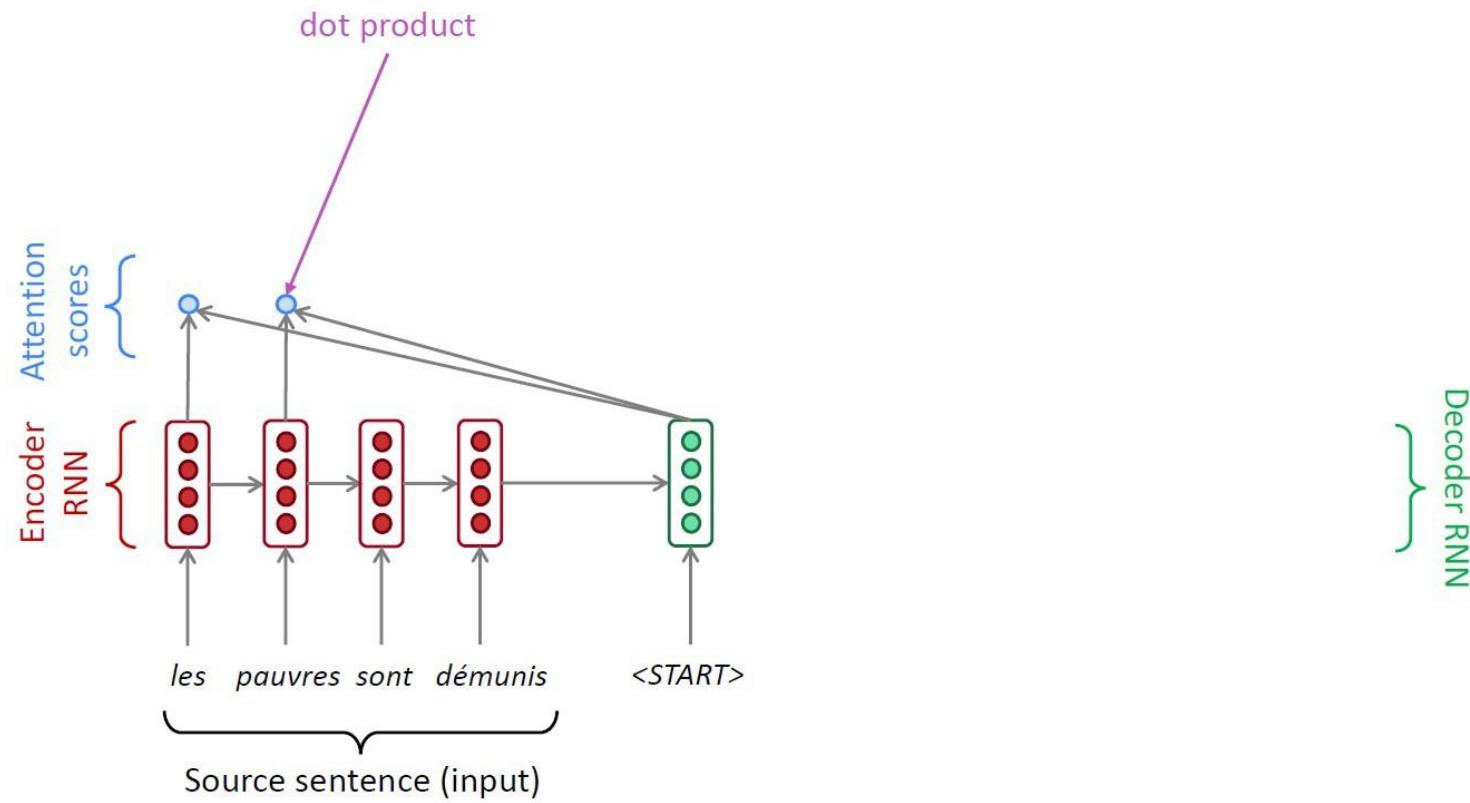
$$\text{Attention}(\text{Query}, \text{Key}, \text{Value}) = \sum_i \text{Similarity}(\text{Query}, \text{Key}_i) * \text{Value}_i$$

$$\mathbf{c}_t = \sum_{j=1}^T \alpha_{t,j} \mathbf{h}_j$$

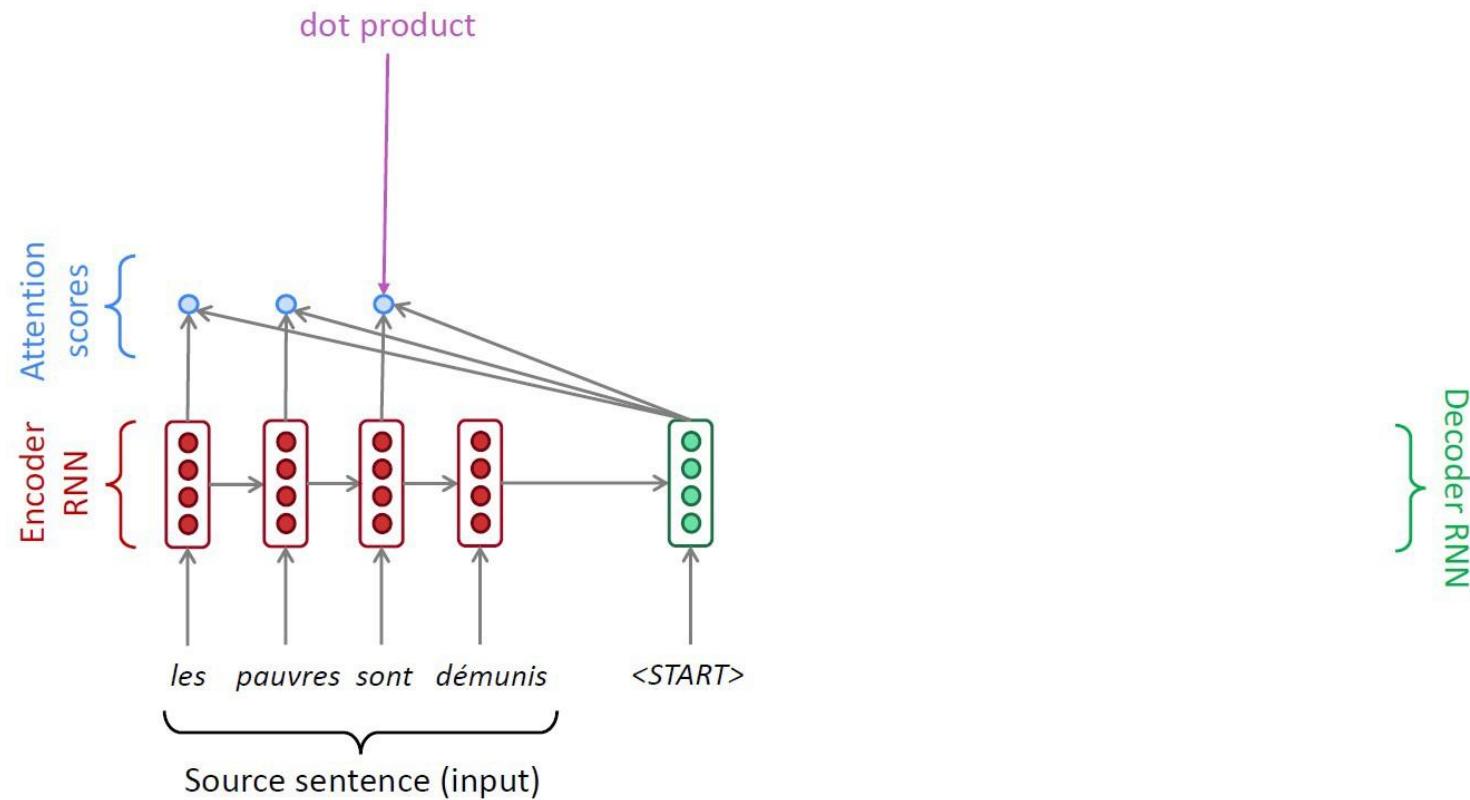
# Running Example



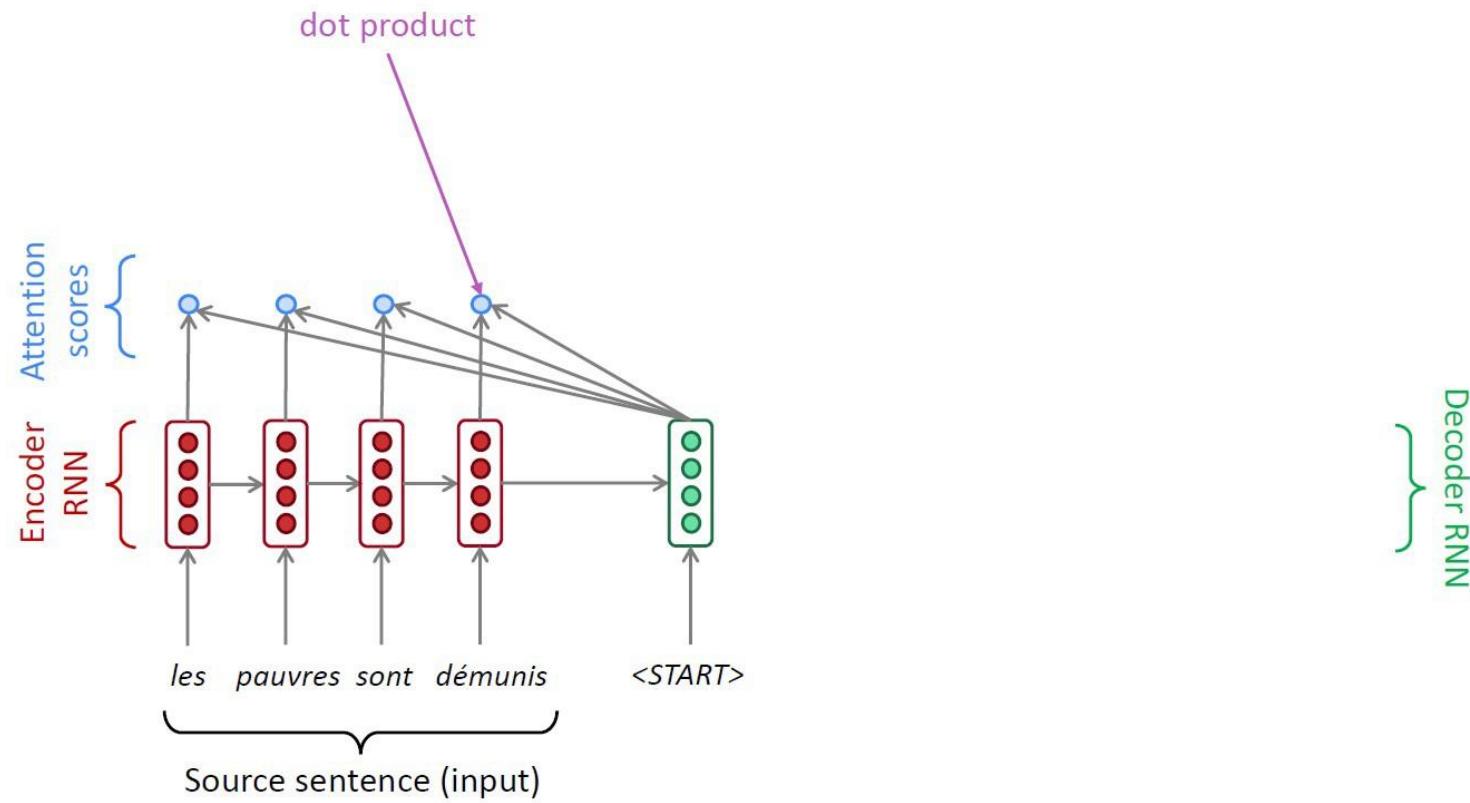
# Running Example



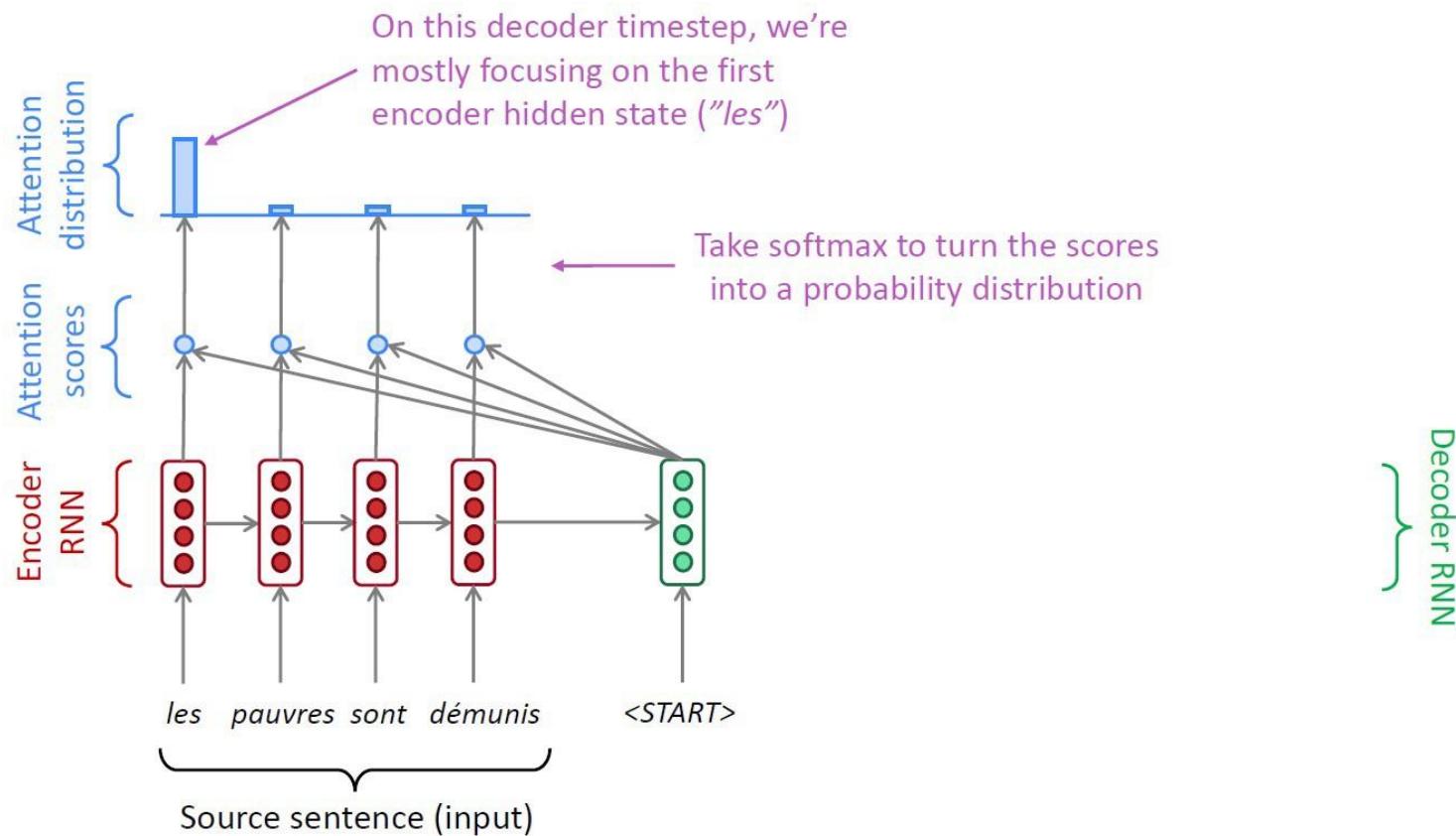
# Running Example



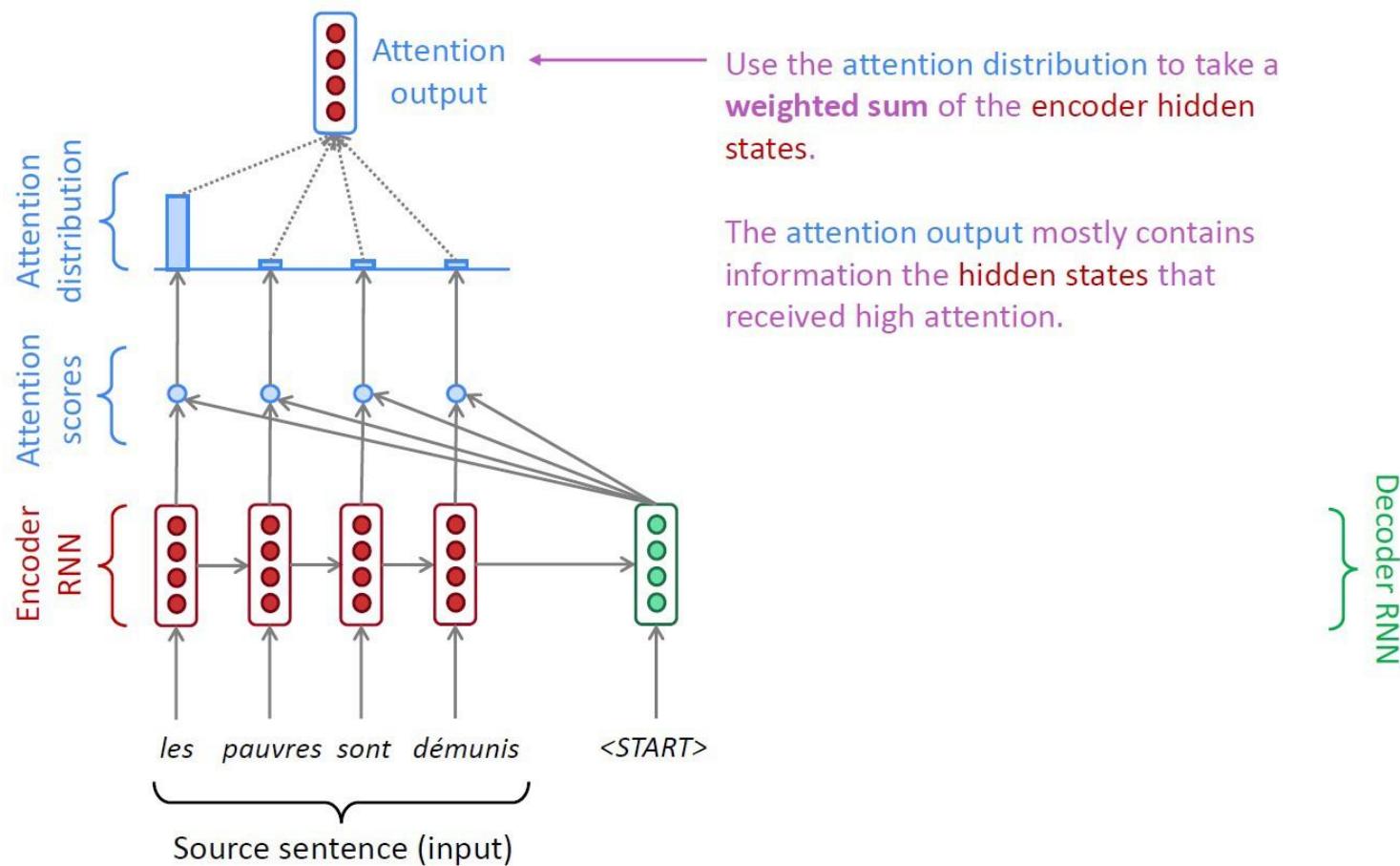
# Running Example



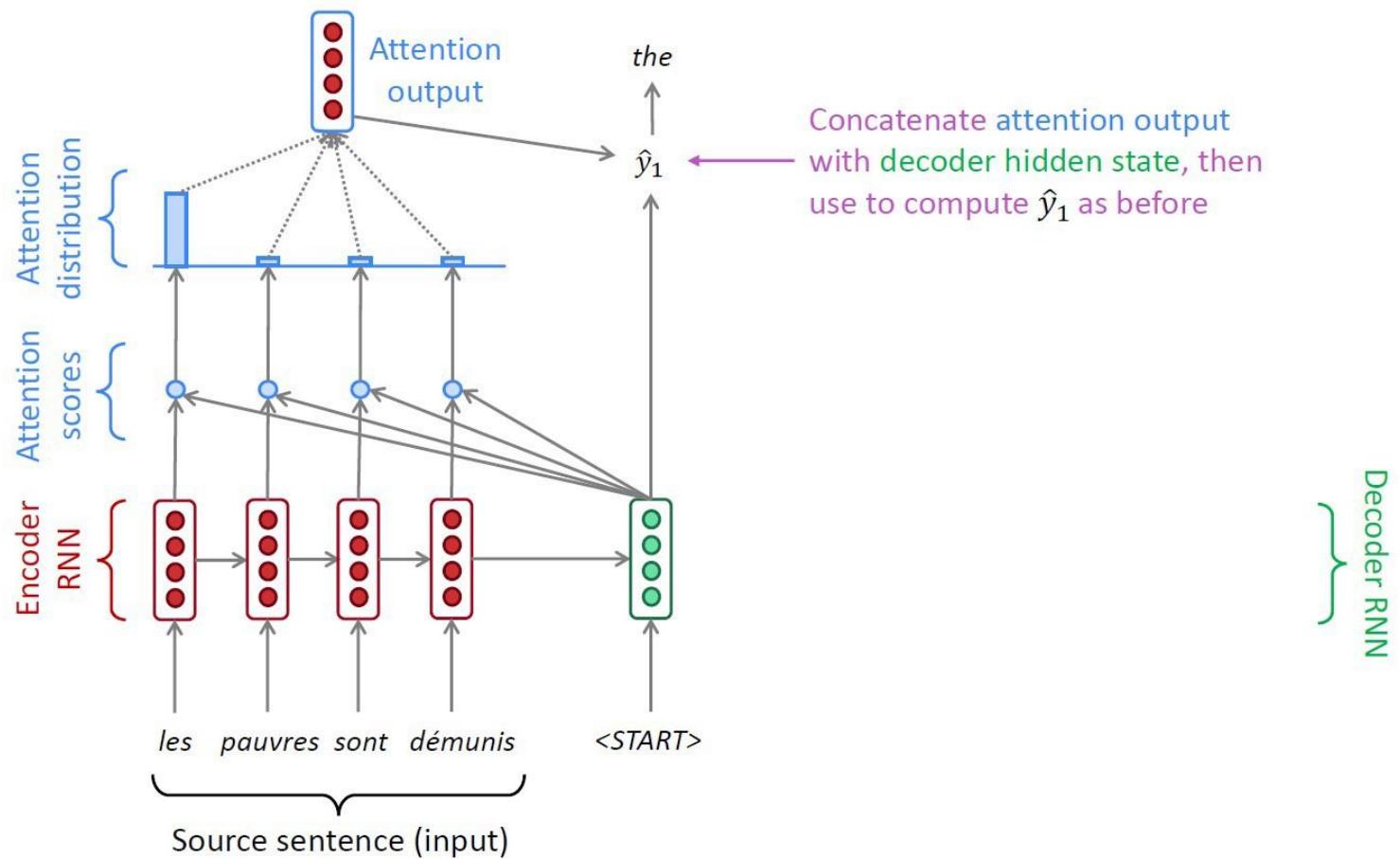
# Running Example



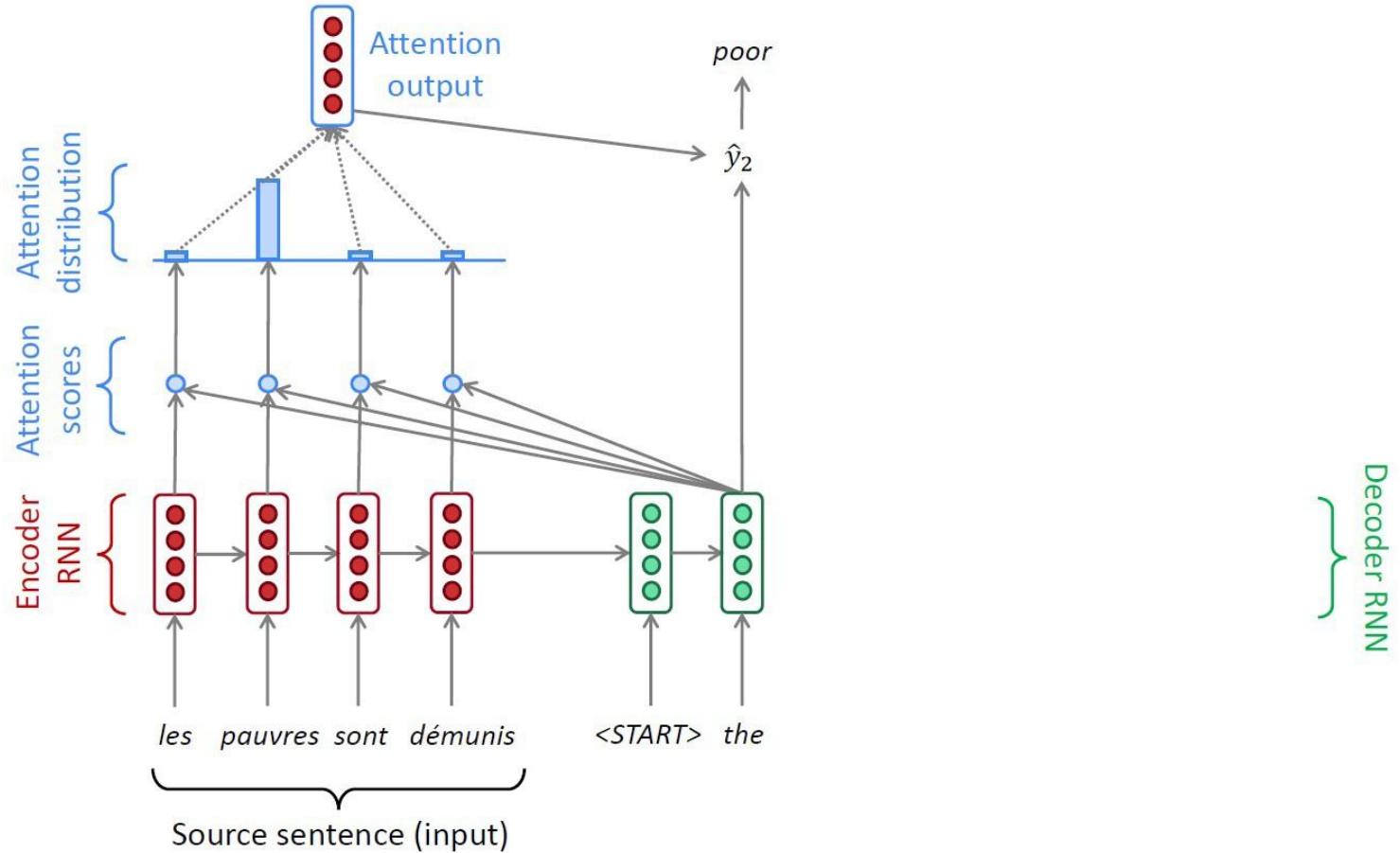
# Running Example



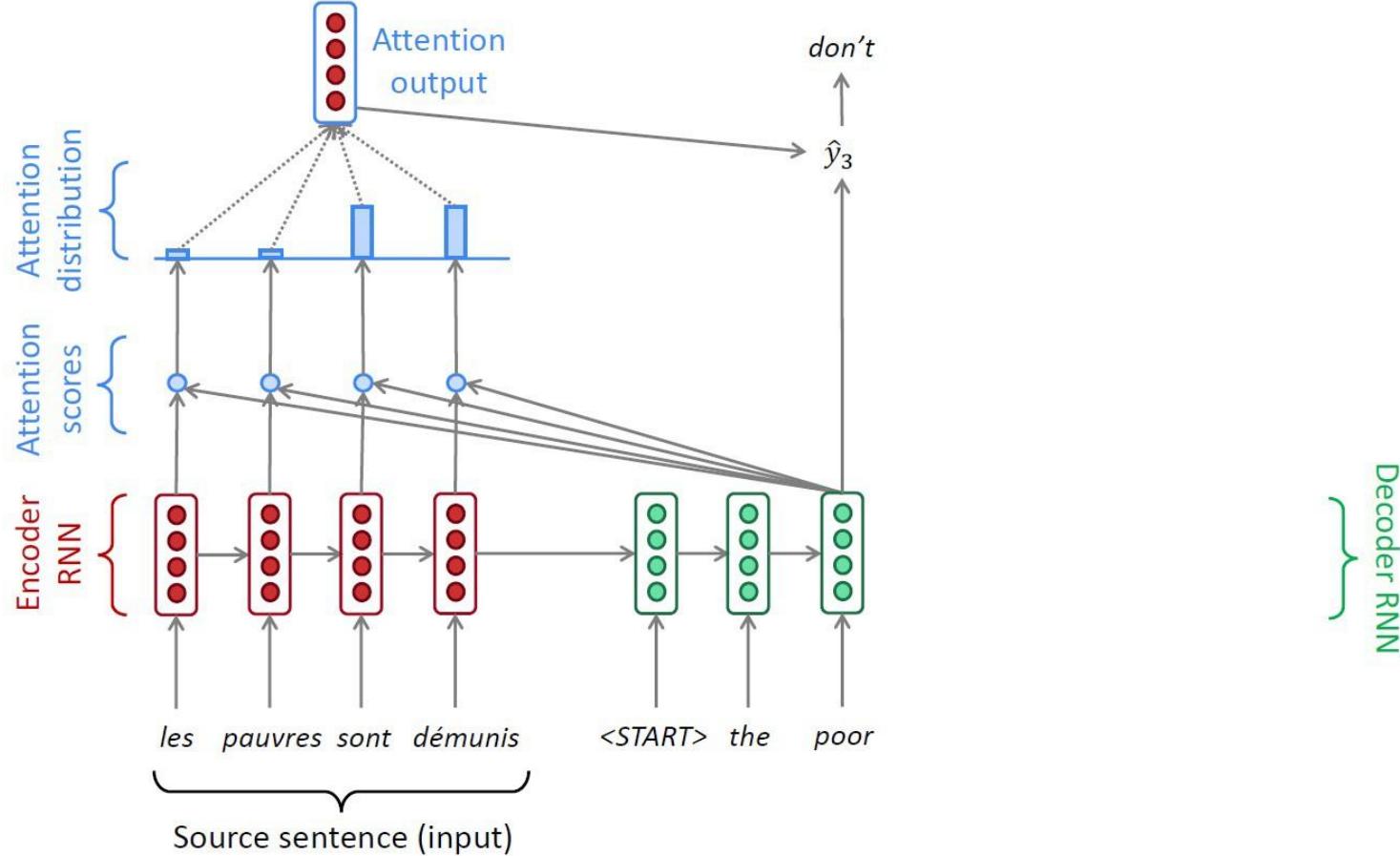
# Running Example



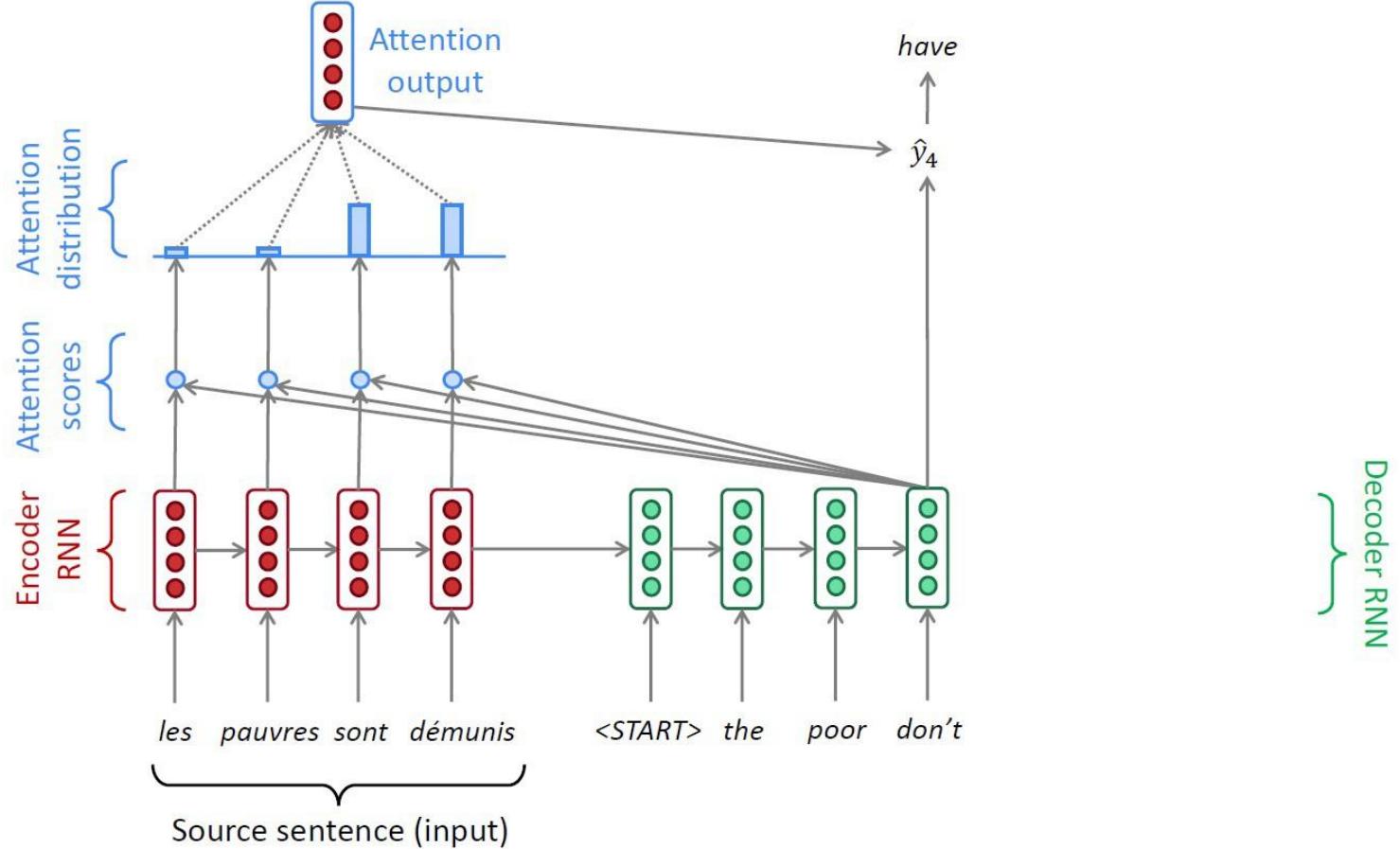
# Running Example



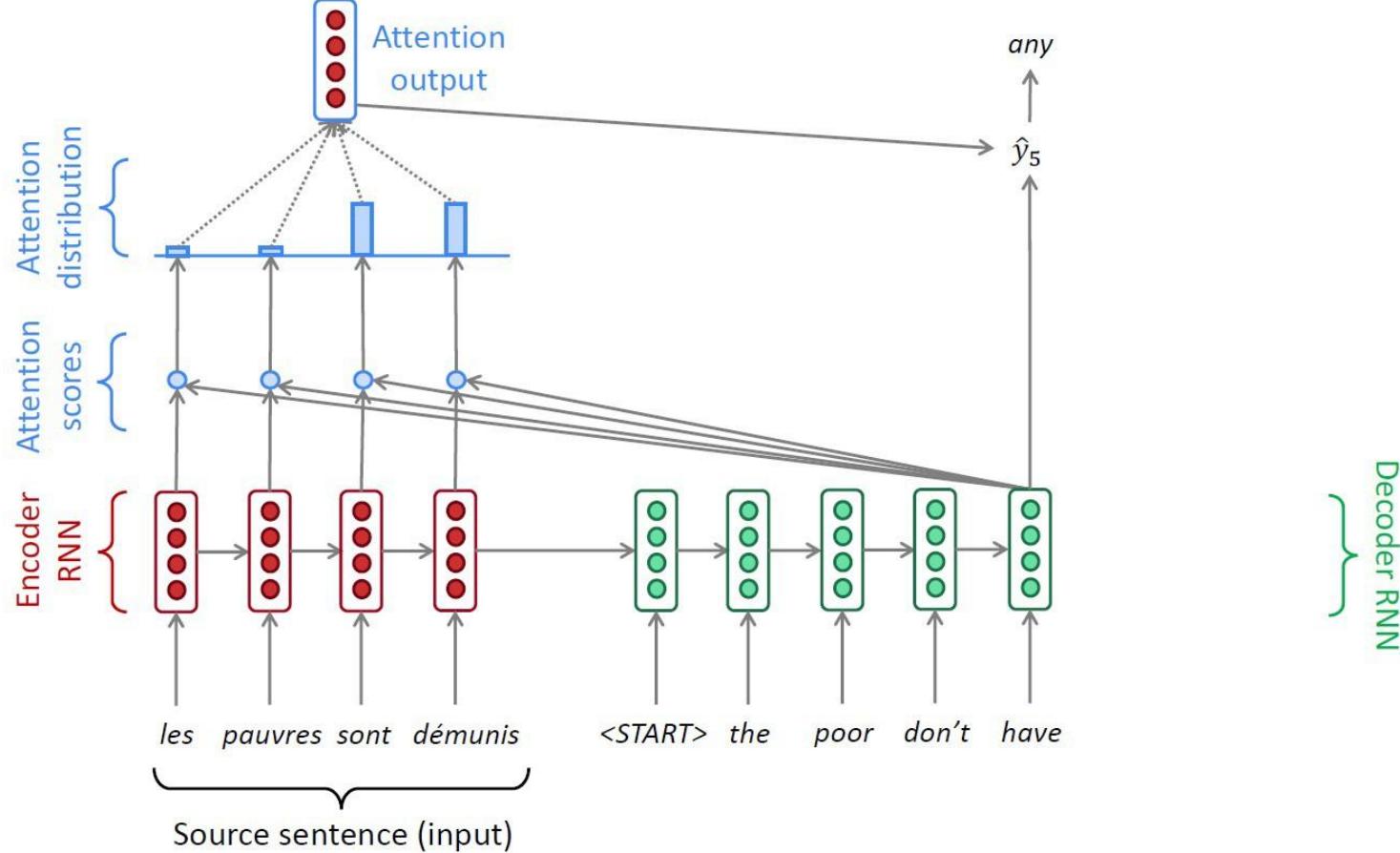
# Running Example



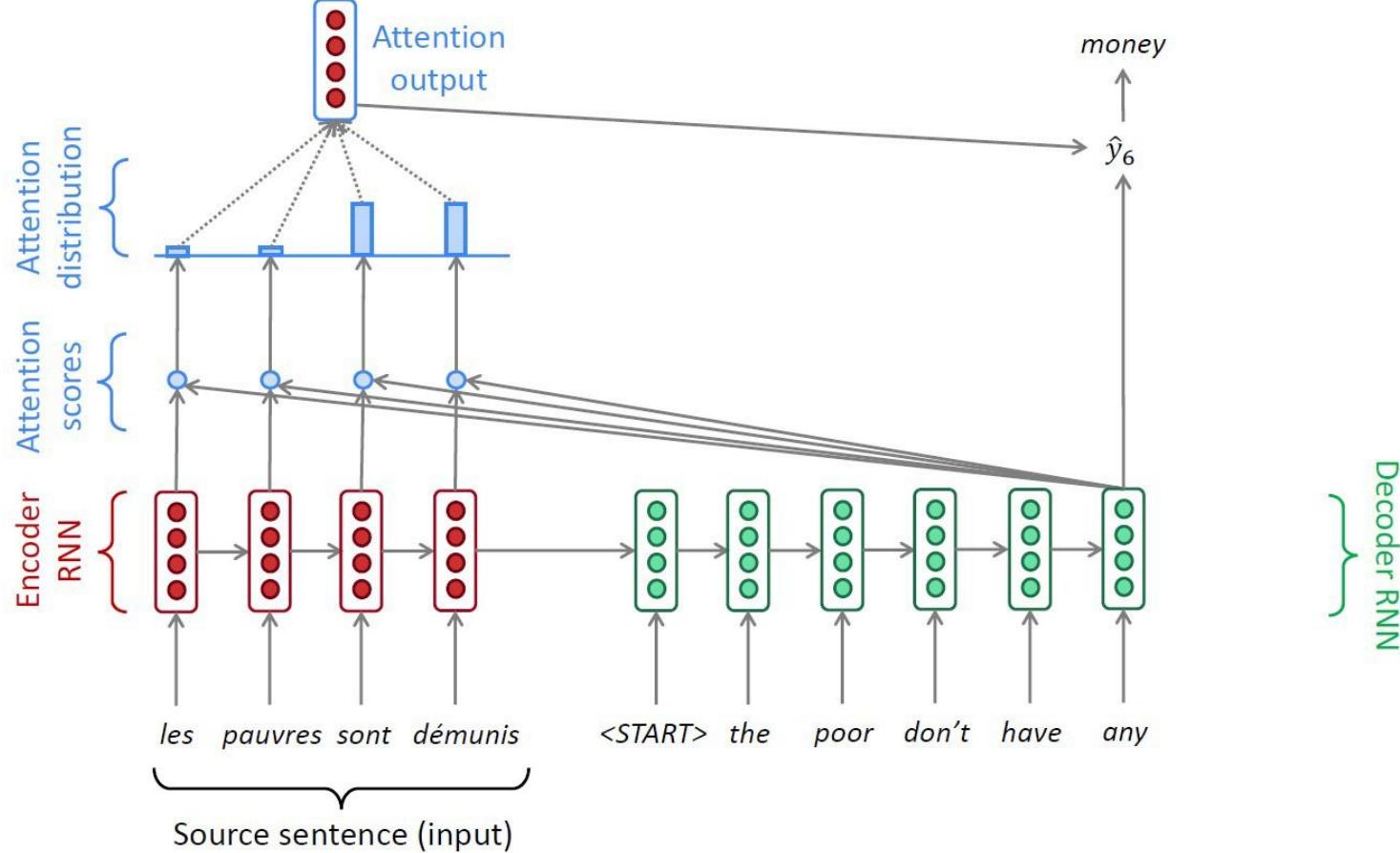
# Running Example



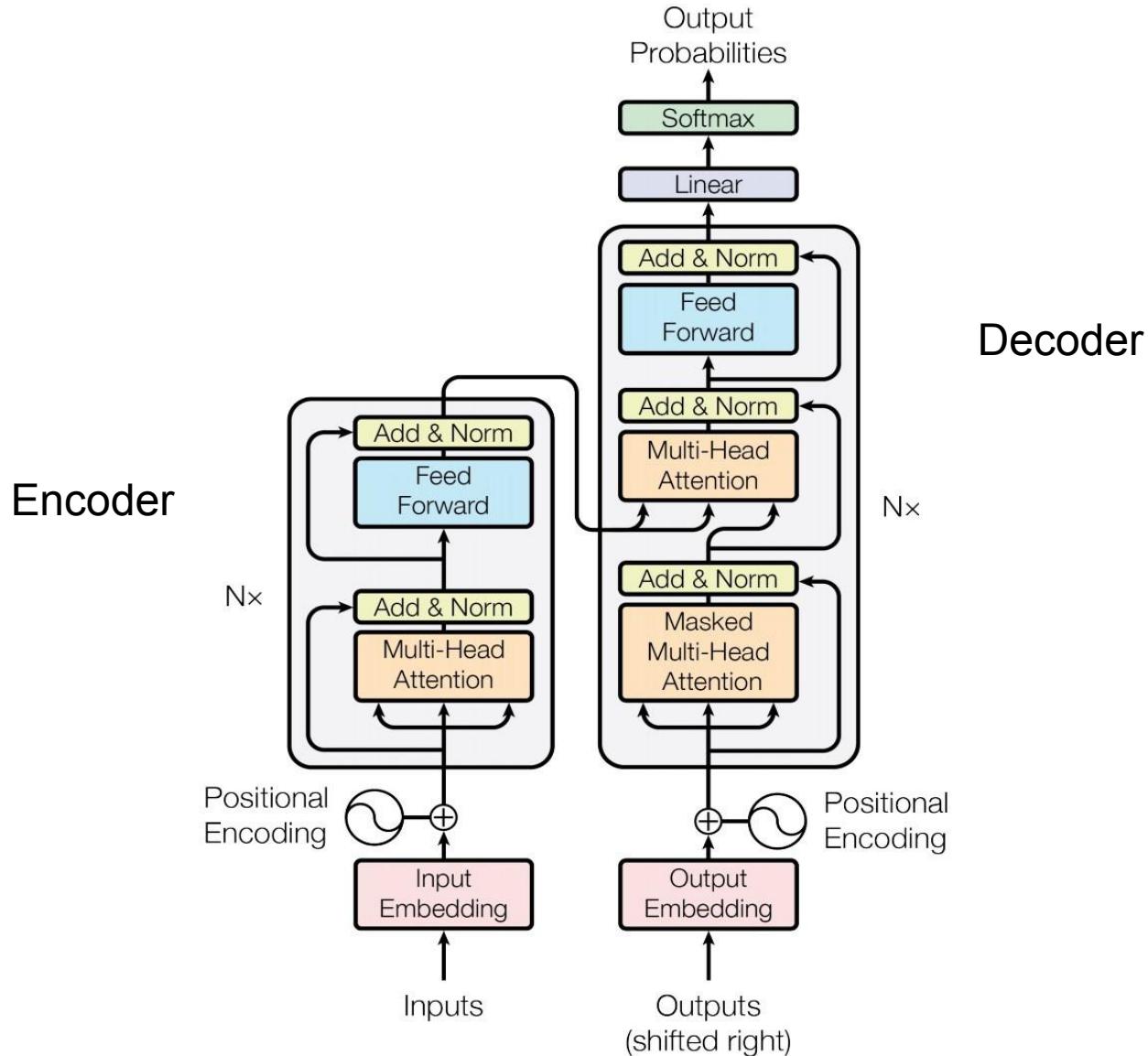
# Running Example

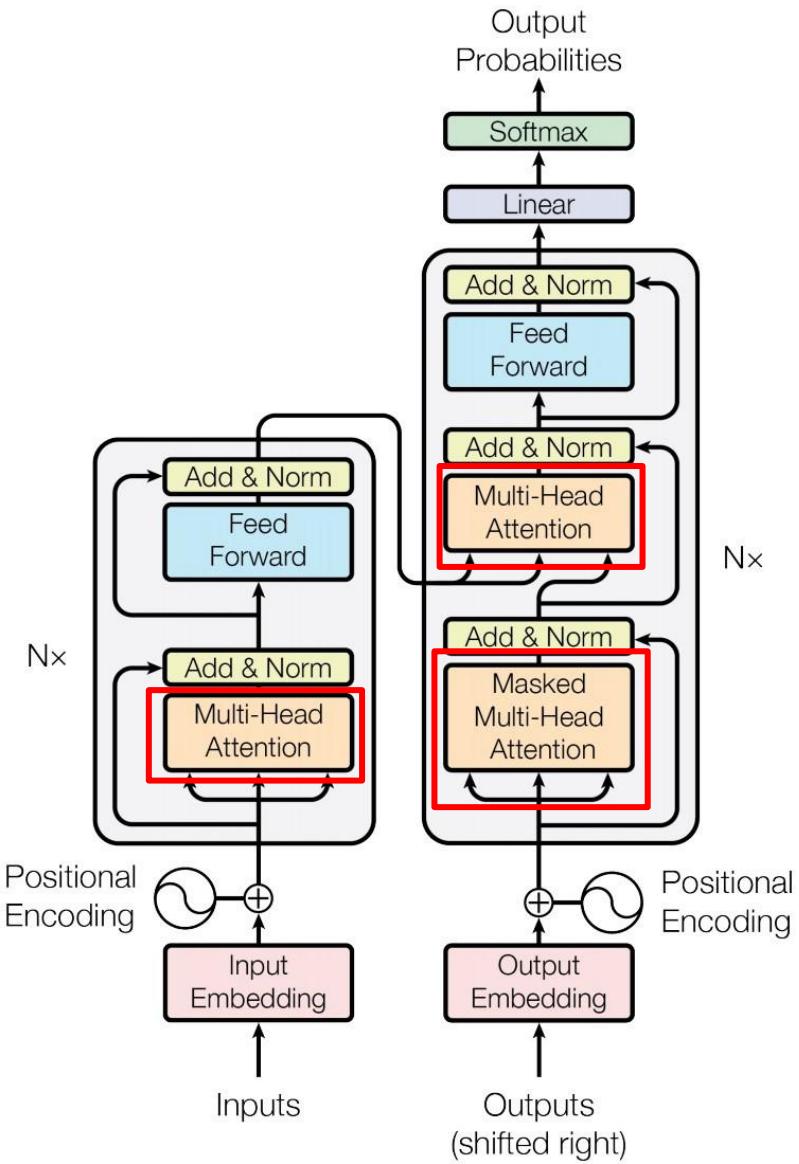


# Running Example



# Transformer Architecture

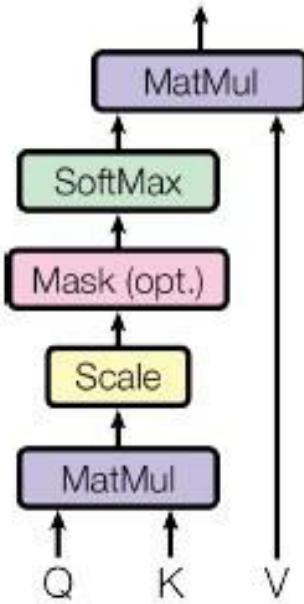




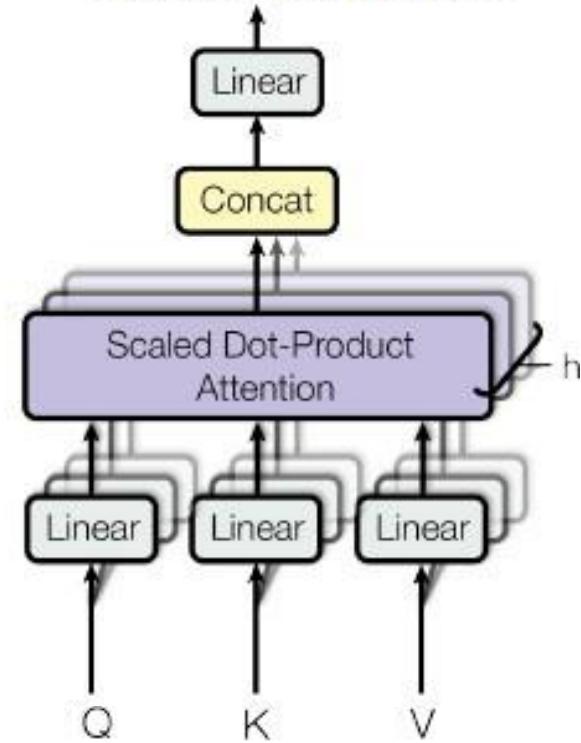
# Multi-Head Attention

# Multi-Head Attention: Summary

Scaled Dot-Product Attention



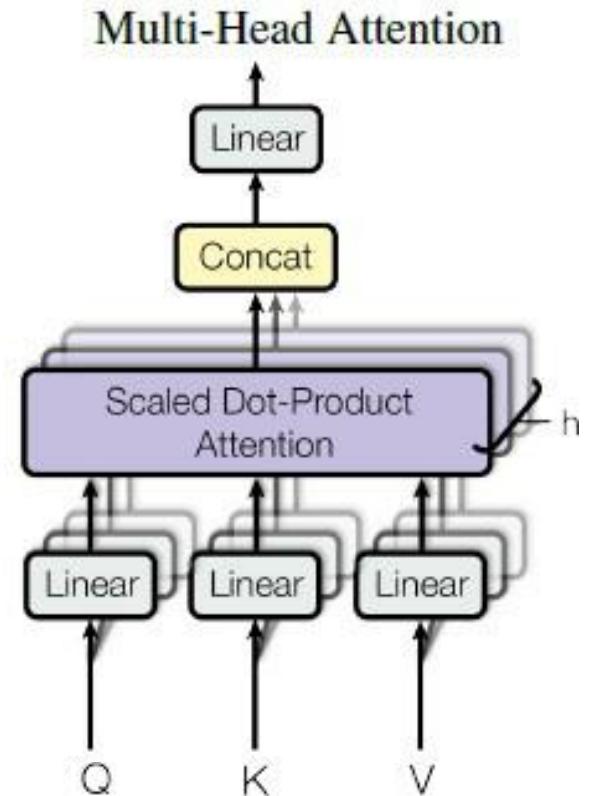
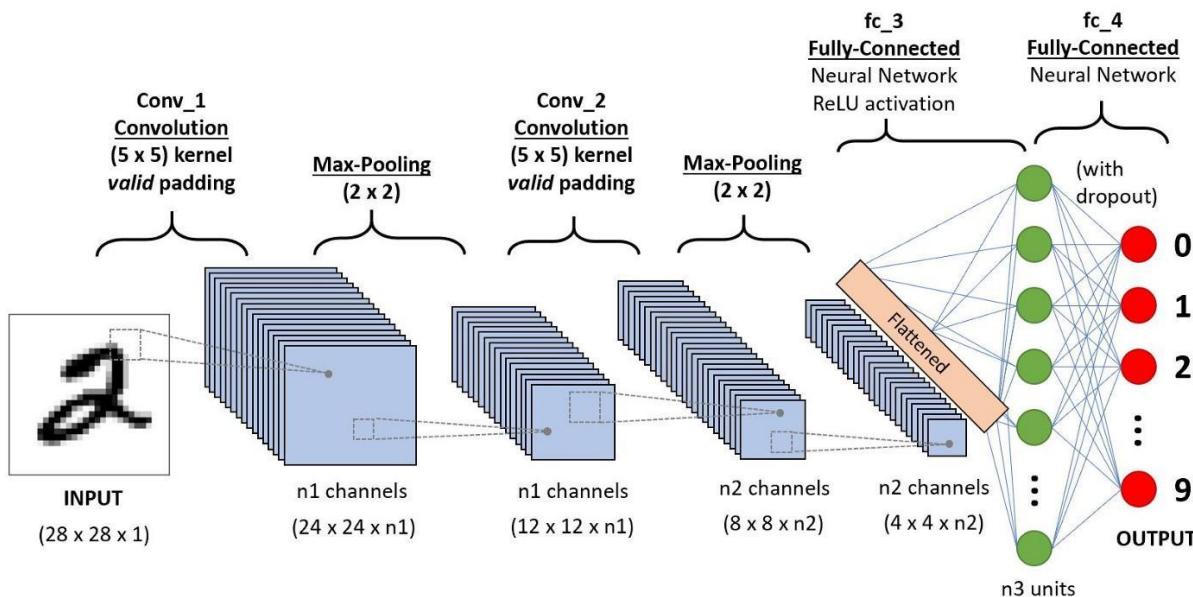
Multi-Head Attention



# Multi-Head Attention: Intuition

The Encoder:

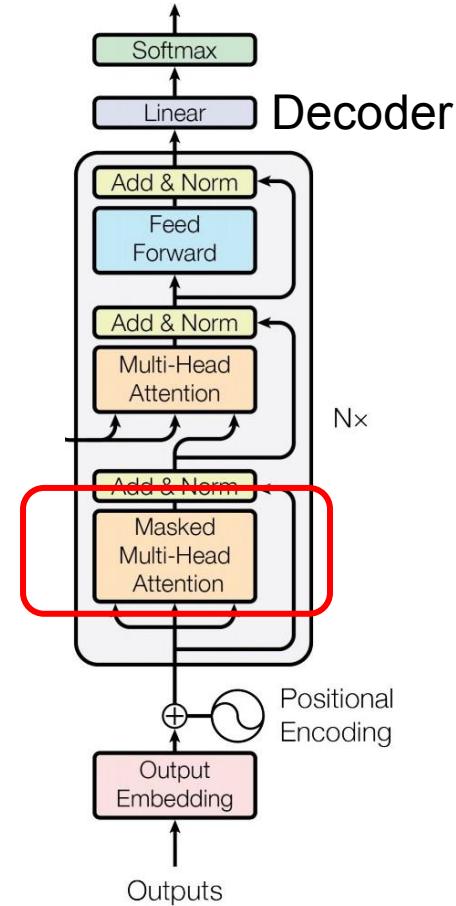
- Combines pairs of words, pairs of pairs of words, ...
- Get the entire sentence
  - Get all words in the input sequence
  - Uses Positional Encoding
- Multiple heads: Similar to the idea of multiple filters in CNN



# Masked Multi-Head Attention

Les pauvres sont démunis  
(French)

The poor do not have any money  
(English)



# Masked Multi-Head Attention

Les pauvres sont  
démunis  
(French)

The poor do not have any money  
(English)

Some of the values should be masked so that we do not use them to make combinations

In Language Translation, when we produce a word:

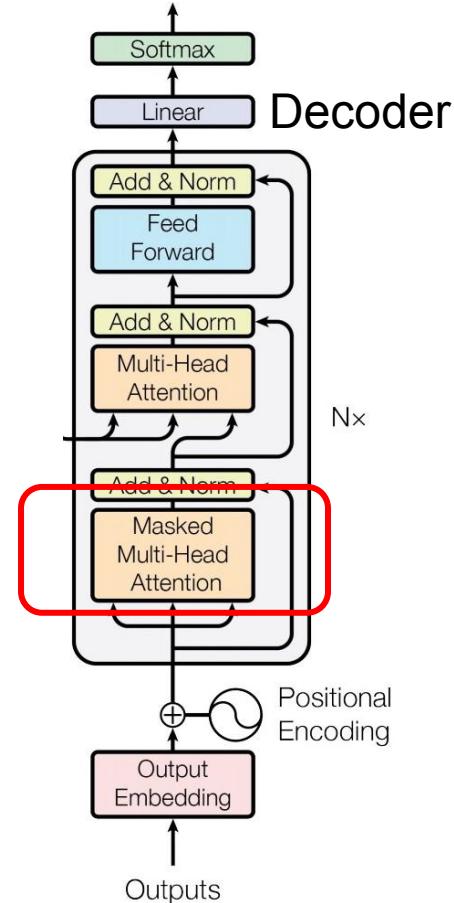
1. It is fine depend on previously-generated words since we generate them sequentially
2. We cannot depend on the future words since they are not generated yet

Solution: In the decoder, we need to prevent building attention based on future words

$$\text{Attention}(Q, K, V) = \text{Softmax} \left( \frac{Q^T K}{\sqrt{d}} \right) V$$

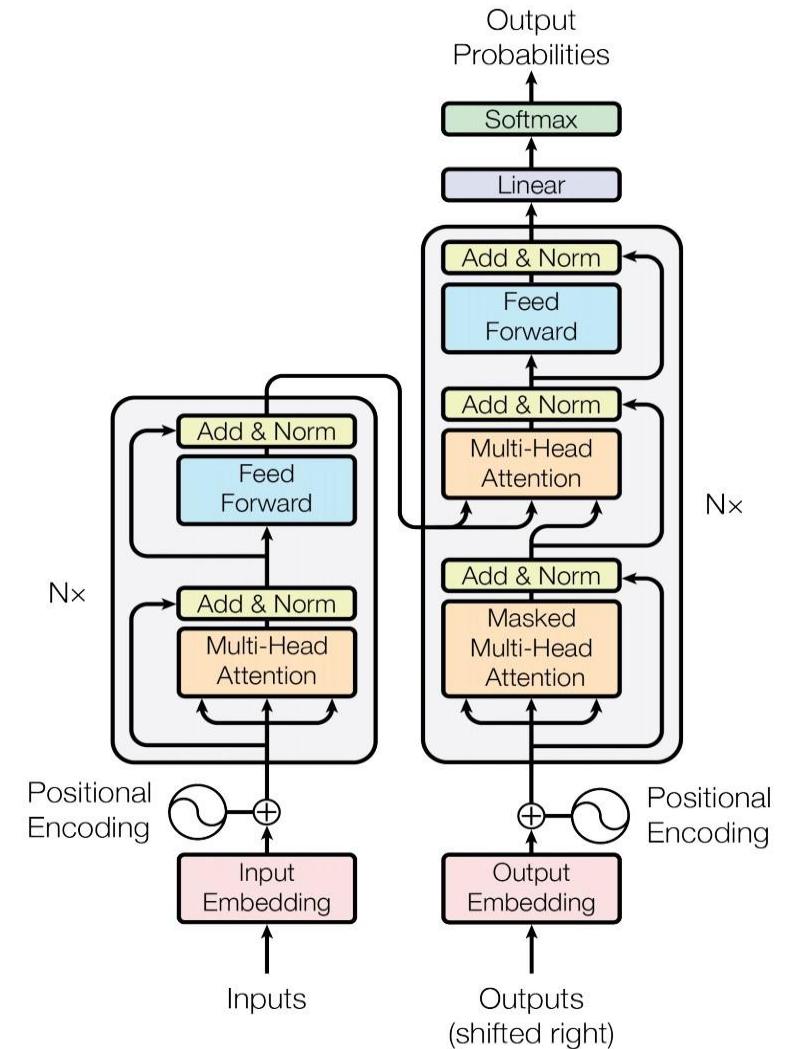
$$\text{MaskedAttention}(Q, K, V) = \text{Softmax} \left( \frac{Q^T K + M}{\sqrt{d}} \right) V$$

$M$  Mask Matrix of 0s and  $-\infty$



# Attention Mechanism Summary

- Self Attention
- Cross Attention
- Multi-Head
- Masked Attention



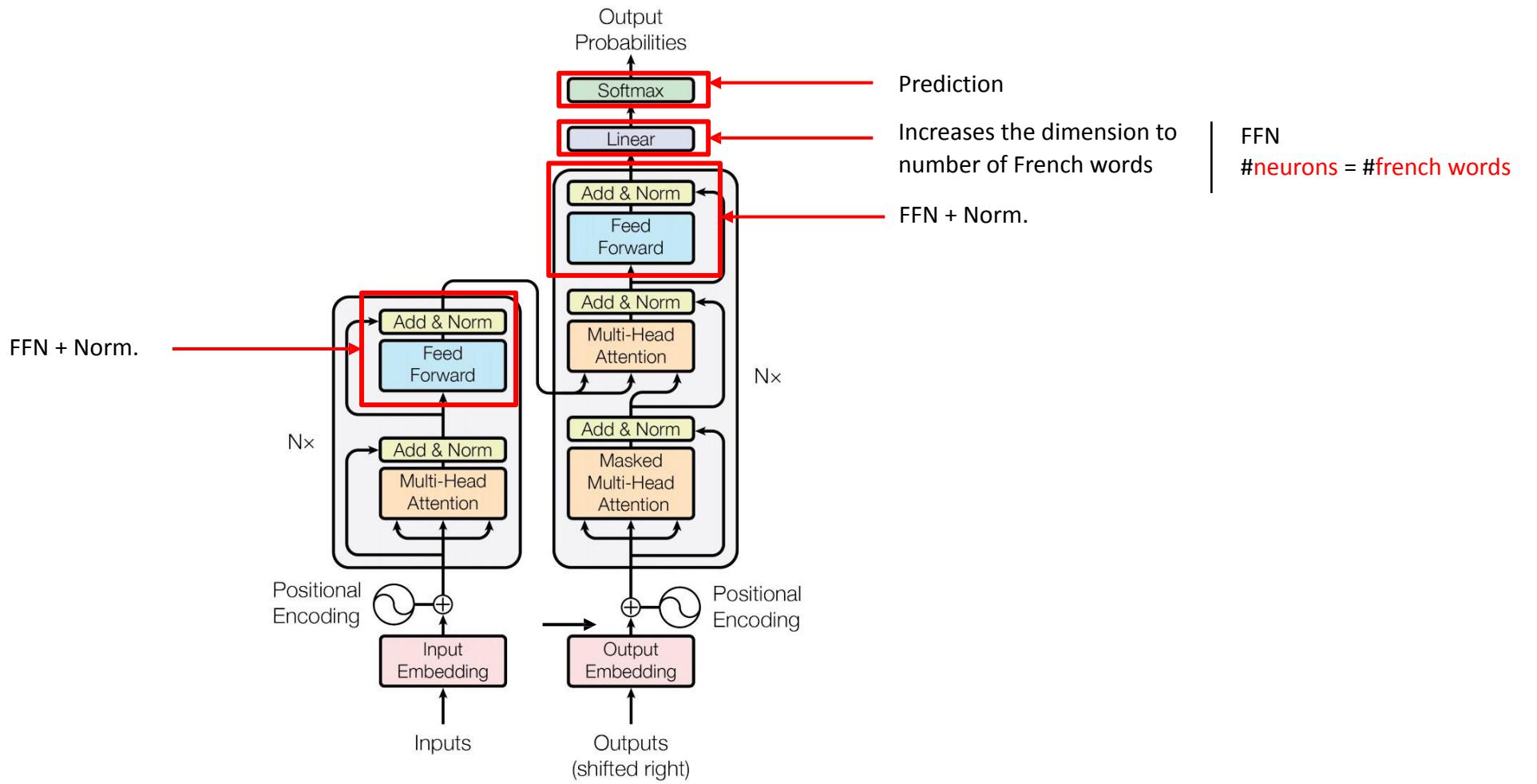
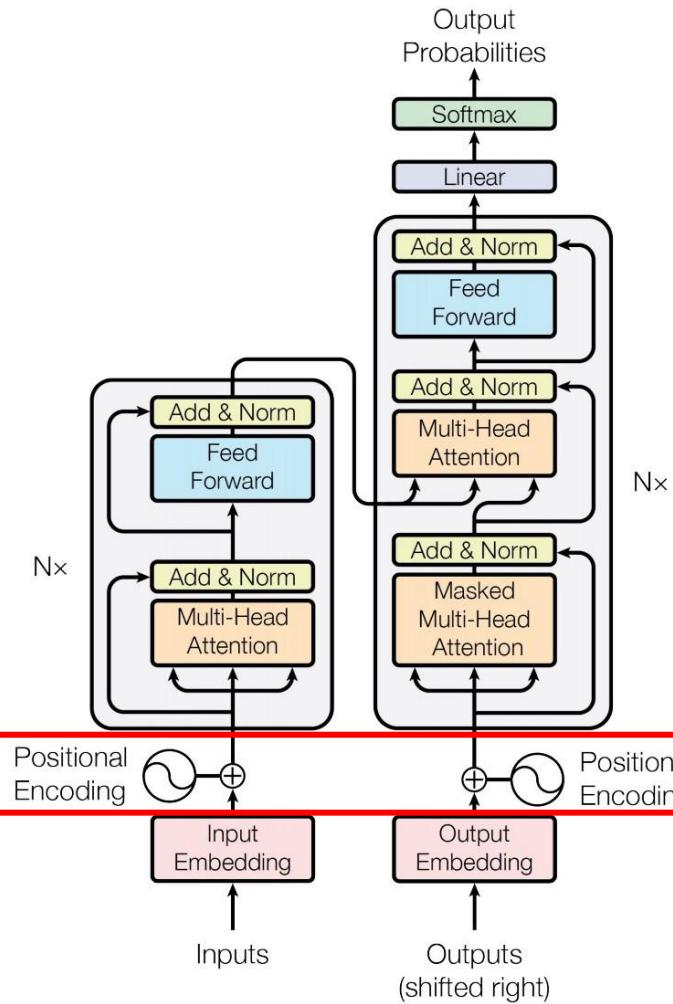


Figure 1: The Transformer - model architecture.



# Positional Embedding

# Positional Encoding

## Input Embedding

dog →

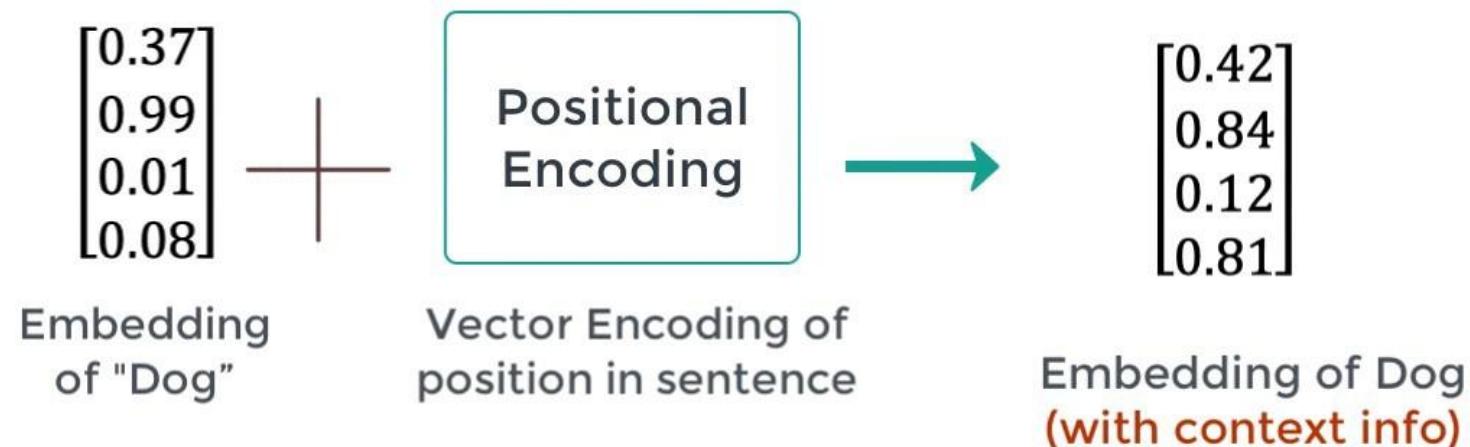
AJ's **dog** is a cutie

AJ looks like a **dog**



$$\begin{bmatrix} 0.37 \\ 0.99 \\ 0.01 \\ 0.08 \end{bmatrix}$$

# Positional Encoding



Sequence      Index  
of token,

Positional Encoding  
Matrix with  $d=4, n=100$

$k$	$i=0$	$i=0$	$i=1$	$i=1$
I	$P_{00}=\sin(0) = 0$	$P_{01}=\cos(0) = 1$	$P_{02}=\sin(0) = 0$	$P_{03}=\cos(0) = 1$
am	$P_{10}=\sin(1/1) = 0.84$	$P_{11}=\cos(1/1) = 0.54$	$P_{12}=\sin(1/10) = 0.10$	$P_{13}=\cos(1/10) = 1.0$
a	$P_{20}=\sin(2/1) = 0.91$	$P_{21}=\cos(2/1) = -0.42$	$P_{22}=\sin(2/10) = 0.20$	$P_{23}=\cos(2/10) = 0.98$
Robot	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

Positional Encoding Matrix for the sequence 'I am a robot'

---

# The Impact of Positional Encoding on Length Generalization in Transformers

---

**Amirhossein Kazemnejad<sup>1,2</sup>, Inkit Padhi<sup>3</sup>  
Karthikeyan Natesan Ramamurthy<sup>3</sup>, Payel Das<sup>3</sup>, Siva Reddy<sup>1,2,4</sup>**

<sup>1</sup>Mila - Québec AI Institute; <sup>2</sup>McGill University;

<sup>3</sup>IBM Research; <sup>4</sup>Facebook CIFAR AI Chair

{amirhossein.kazemnejad,siva.reddy}@mila.quebec  
inkpad@ibm.com, {knatesa,daspa}@us.ibm.com

## Abstract

Length generalization, the ability to generalize from small training context sizes to larger ones, is a critical challenge in the development of Transformer-based language models. Positional encoding (PE) has been identified as a major factor influencing length generalization, but the exact impact of different PE schemes on extrapolation in downstream tasks remains unclear. In this paper, we conduct a systematic empirical study comparing the length generalization performance of decoder-only Transformers with five different position encoding approaches including Absolute Position Embedding (APE), T5’s Relative PE, ALiBi, and Rotary, in addition to Transformers without positional encoding (NoPE). Our evaluation encompasses a battery of reasoning and mathematical tasks. Our findings reveal that the most commonly used positional encoding methods, such as ALiBi, Rotary, and APE, are not well suited for length generalization in downstream tasks. More importantly, NoPE outperforms other explicit positional encoding methods while requiring no additional computation. We theoretically demonstrate that NoPE can represent both absolute and relative PEs, but when trained with SGD, it mostly resembles T5’s Relative PE attention patterns. Finally, we find that scratchpad is not always helpful to solve length generalization and its format highly impacts the model’s performance. Overall, our work suggests that explicit position encodings are not essential for decoder-only Transformers to generalize well to longer sequences.

# Research Topics

- Faster Inference
- Mixture of Experts
- Safety Alignment & Attacks
- Replacement of Attention Block

# Outline

## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

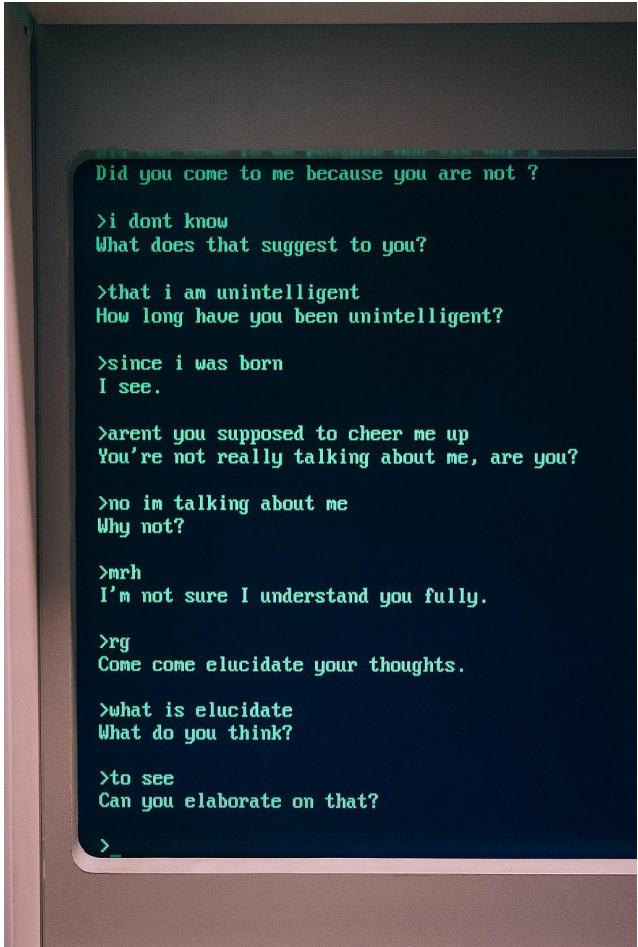
## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

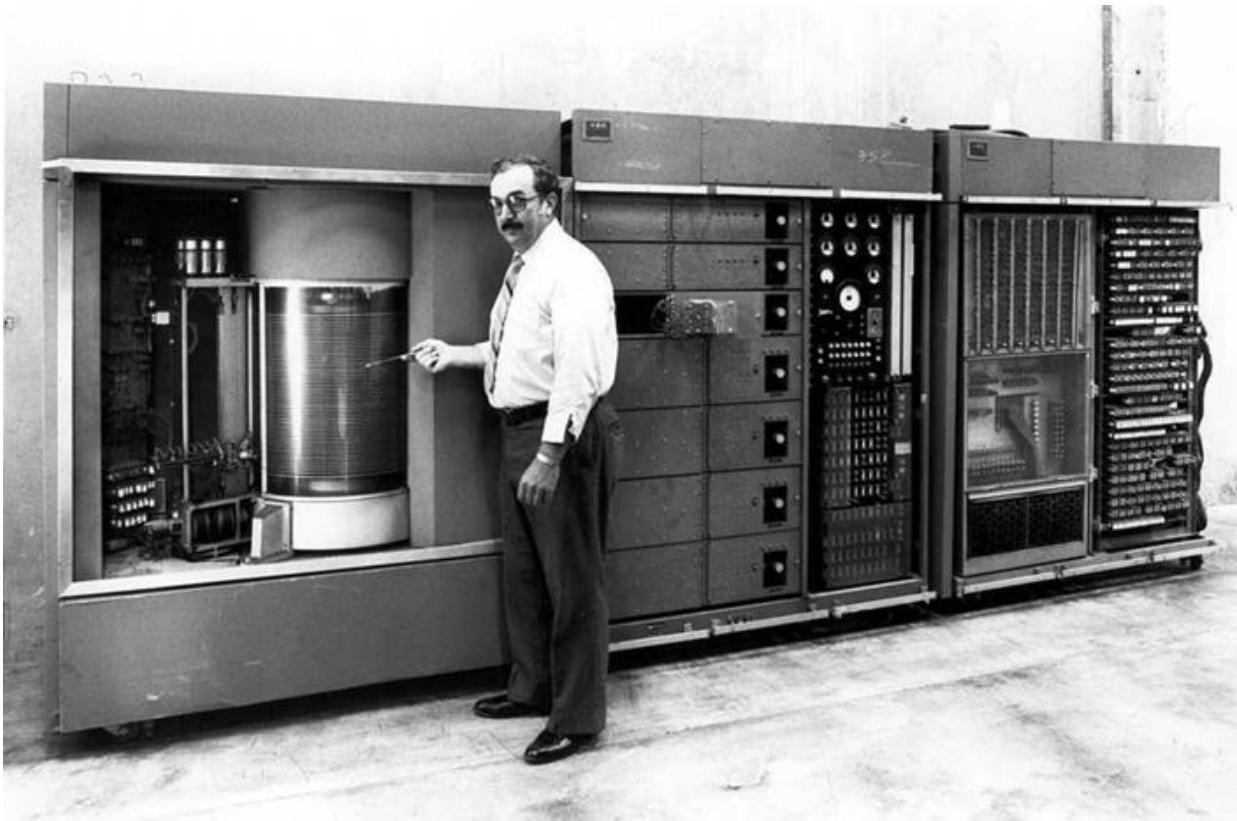
# What Are Language Models

A probabilistic model of a natural language that can generate probabilities of a series of words, based on text corpora in one or multiple languages it was trained on.

# Applications



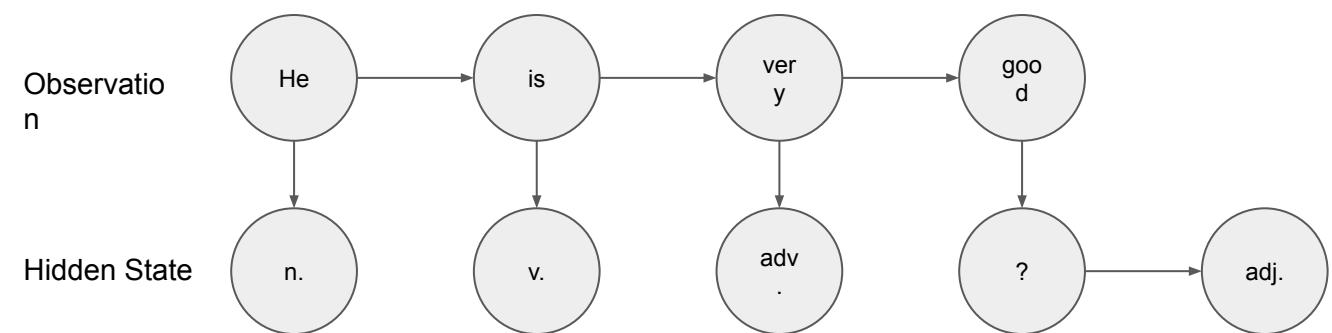
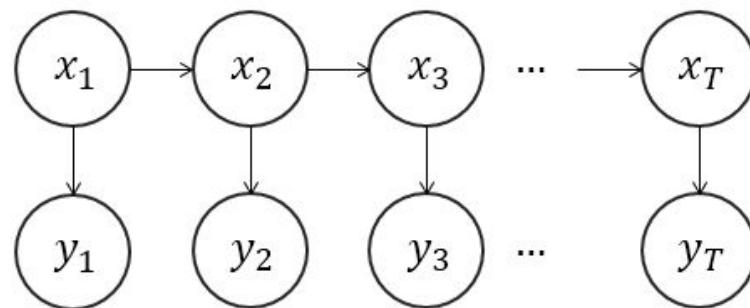
IBM Eliza



Georgetown-IBM-Experiment

# Early Language Model

Statistical Language Models: **Hidden Markov Models**

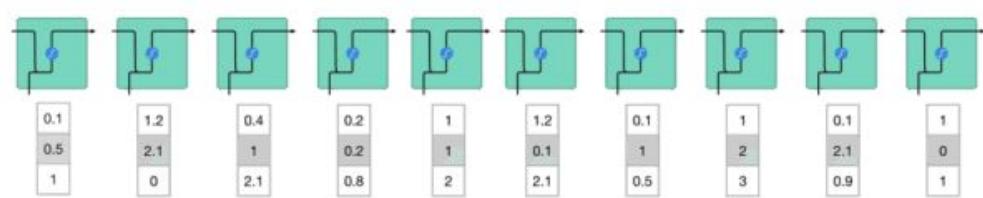


# Drawbacks on Early Language Model

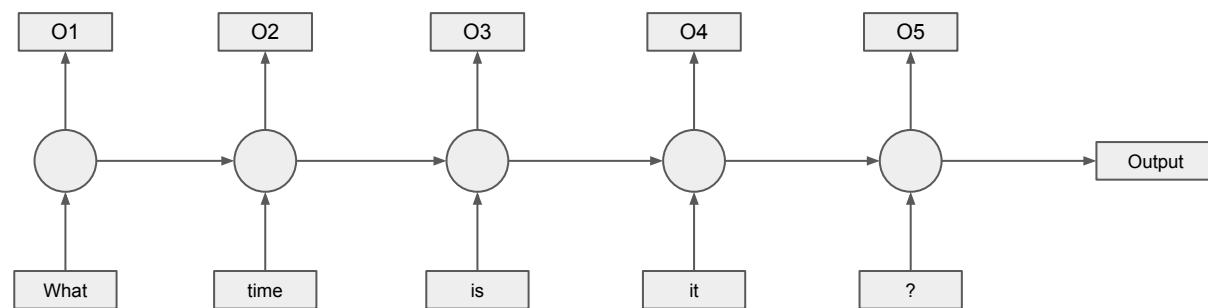
- Slow to compute
- Lack of generalisation
- Data sparsity
- ...

# Introduction to Neural Network

Neural Network: Recurrent Neural Network

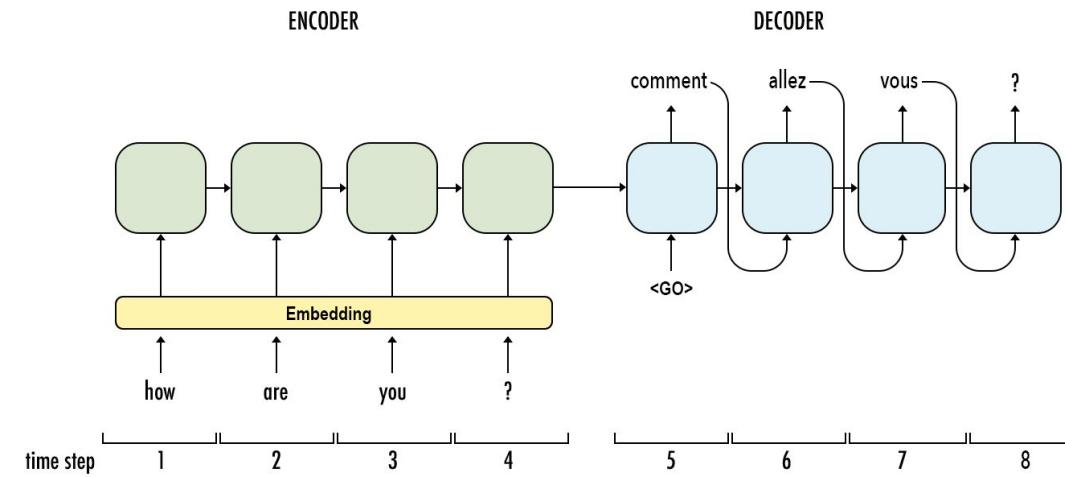


Animation demo of RNN



NLP example on semantic understanding

# Recap: Why transformer?



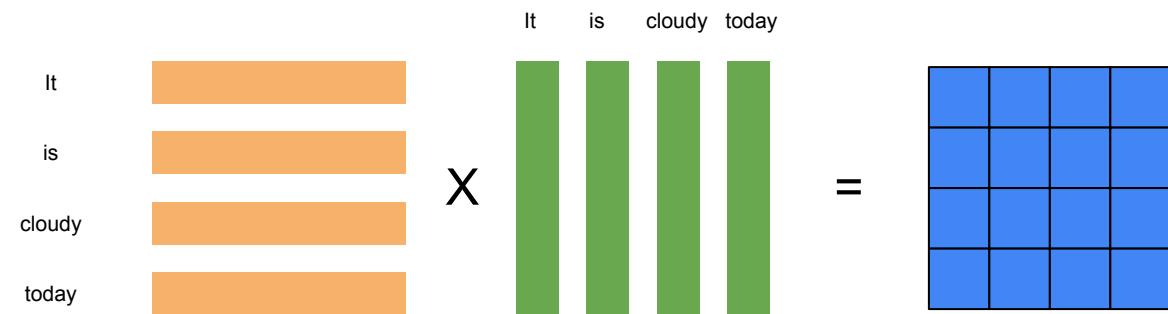
In traditional sequence neural networks, sequence were ordered, which means the latter tokens are dependent to the previous sequence, which means both the ability of parallel computation and the ability of model's view is limited

# Recap: Why transformer?

$$\text{softmax}\left(\frac{\begin{matrix} \text{Q} & \text{K}^T \\ \begin{matrix} \text{---} \\ \times \\ \sqrt{d_k} \end{matrix} \end{matrix}}{\text{---}}\right) \text{---} \text{V}$$

=  $\text{z}$

[https://zhuyang-zhao.github.io/\\_book/15034697](https://zhuyang-zhao.github.io/_book/15034697)



In transformer or attention based models, the input sequence was regarded as a whole entity, which means by a simple production, we can get the relationship between any two of the tokens in a single sequence, give the model ability to “see the global status”

# Large Language Models

Artificial intelligence (AI) system that has been trained on vast amounts of text data to understand and generate human-like language.

Key characters:

- Deeper architecture
- Larger scale
- Massive data
- Strong capability

# Outline

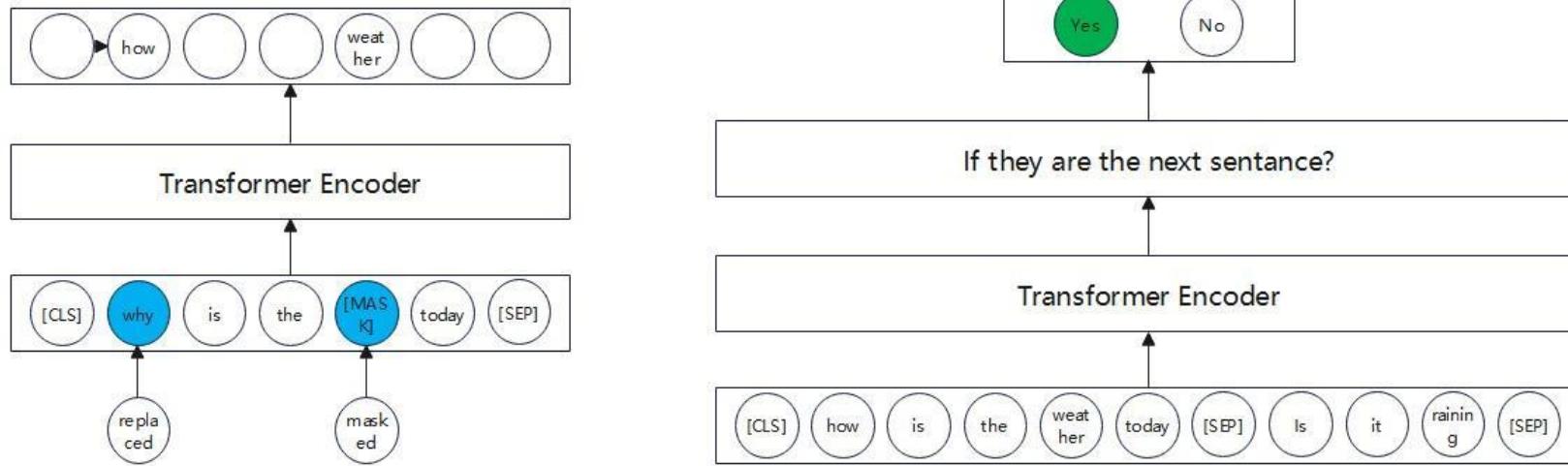
## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

# Autoencoding (Encoder only)

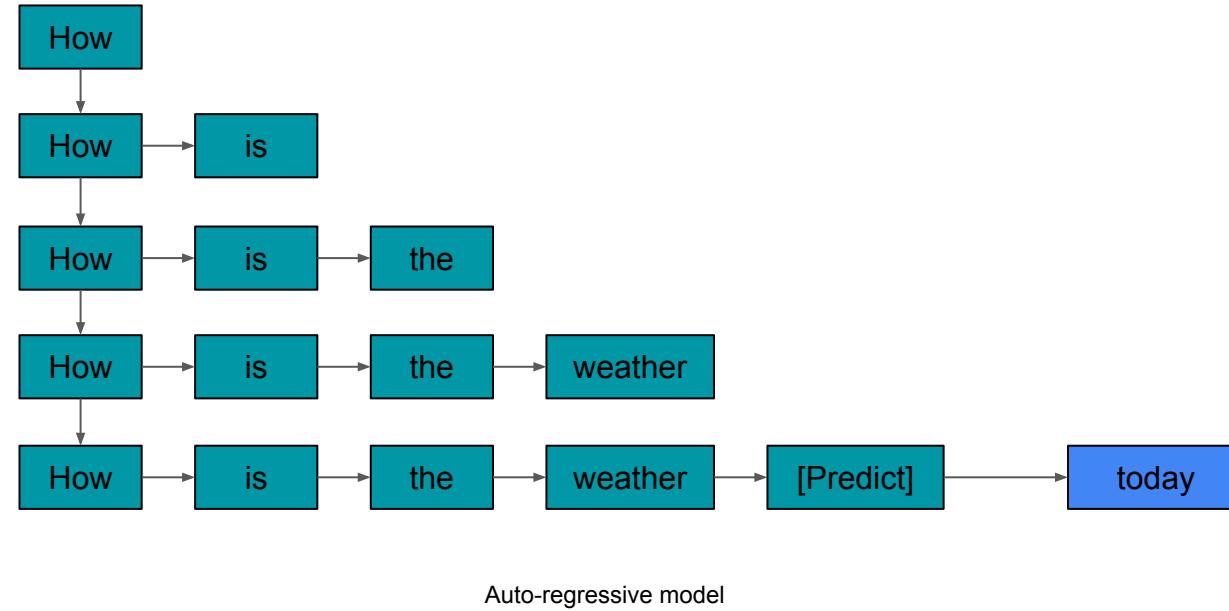


Example of strategy used by BERT

Scenario: Encoders produce contextual representations suitable for natural language understanding tasks, often used to solve reading comprehension, cloze and other problems.

Cons: It can not support the generation of indeterminate length text, and depends on the context before and after, which is very limited to the type of downstream tasks; Generally, it can only be used in downstream tasks after fine-tune, which will also involve a lot of manual operation and model tuning, and the model can not be made too large.

# Autoregressive (Decoder model)



Scenario: Suitable for long sequence, zero shot, and generation tasks, Allows the Decoder class model to adapt to various types of downstream tasks without being fine-tuned.

Cons: Unidirectional attention, which makes it impossible to fully capture the dependencies between contexts in NLU tasks. Other tasks can be converted into autoregressive tasks.

# Sequence to Sequence (Encoder-Decoder model)

阅读文章:鲁迅,原名周樟寿,后改名周树人,浙江绍兴人。著名文学家、思想家、革命家、教育家、民主战士,新文化运动的重要参与者,中国现代文学的奠基人之一。早年与厉绥之和钱均夫同赴日本公费留学,于日本仙台医科专门学校肄业。鲁迅,1918年发表《狂人日记》时所用的笔名,也是最为广泛的笔名。鲁迅一生在文学创作、文学批评、思想研究、文学史研究、翻译、美术理论引进、基础科学介绍和古籍校勘与研究等多个领域具有重大贡献。他对于五四运动以后的中国社会思想文化发展具有重大影响,蜚声世界文坛,尤其在韩国、日本思想文化领域有极其重要的地位和影响,被誉为“二十世纪东亚文化地图上占最大领土的作家”。请问:鲁迅于1918年发表过什么文章?回答:

(Read the article: Lu Xun, formerly known as Zhou Zhangshou, later renamed Zhou Shuren, Zhejiang Shaoxing. He is a famous writer, thinker, revolutionist, educator, democracy fighter, important participant of the New Culture Movement, one of the founders of Chinese modern literature. In his early years, Lu Xun, Li Suizhi and Qian Junfu went to Japan to study abroad as public expense and studied at Sendai Medical School in Japan. Lu Xun, the pen name used when he published Diary of Mad Men in 1918, is also the most widely used pen name. Lu Xun made great contributions in literary creation, literary criticism, ideological research, literary history research, translation, introduction of art theory, introduction of basic science and collation and research of ancient books. He had a great influence on the ideological and cultural development of Chinese society after the May 4th Movement, and was famous in the world literary circles, especially in the ideological and cultural fields of Korea and Japan. He was known as the writer who occupied the largest territory on the cultural map of East Asia in the 20th century. Question: What articles did Lu Xun publish in 1918? Answer:

鲁迅于1918年发表了《狂人日记》。《狂人日记》是中国现代文学史上最早的一部白话小说,也是中国现代小说的开山之作。  
(Lu Xun published the Madman's Diary in 1918. Madman's Diary is the earliest vernacular novel in the history of modern Chinese literature, and is also the beginning of modern Chinese novels.)

Pangu- $\Sigma$  on bilingual summarisation task

Scenarios: Mainly used for conditional text generation, such as generating summaries and answering questions

Cons: Requires more parameters.

# Comparison between LLM and DNN

	Large Language Model	Traditional DNN
Architecture	Typically based on Transformers	Various architecture such as MLP, CNN, RNN, GAN, Diffusion etc
Scale	Hundreds of millions to billions of parameters	May have millions of parameters but not in the billions
Data	Vast and diverse datasets	Domain-specific datasets
Training Strategy	Two-step training process, pre-training and fine-tuning	Task specific supervised or unsupervised training
Application	Natural language understanding and generation tasks	Crossfield, NLP, CV, Audio, Generation etc.

# Outline

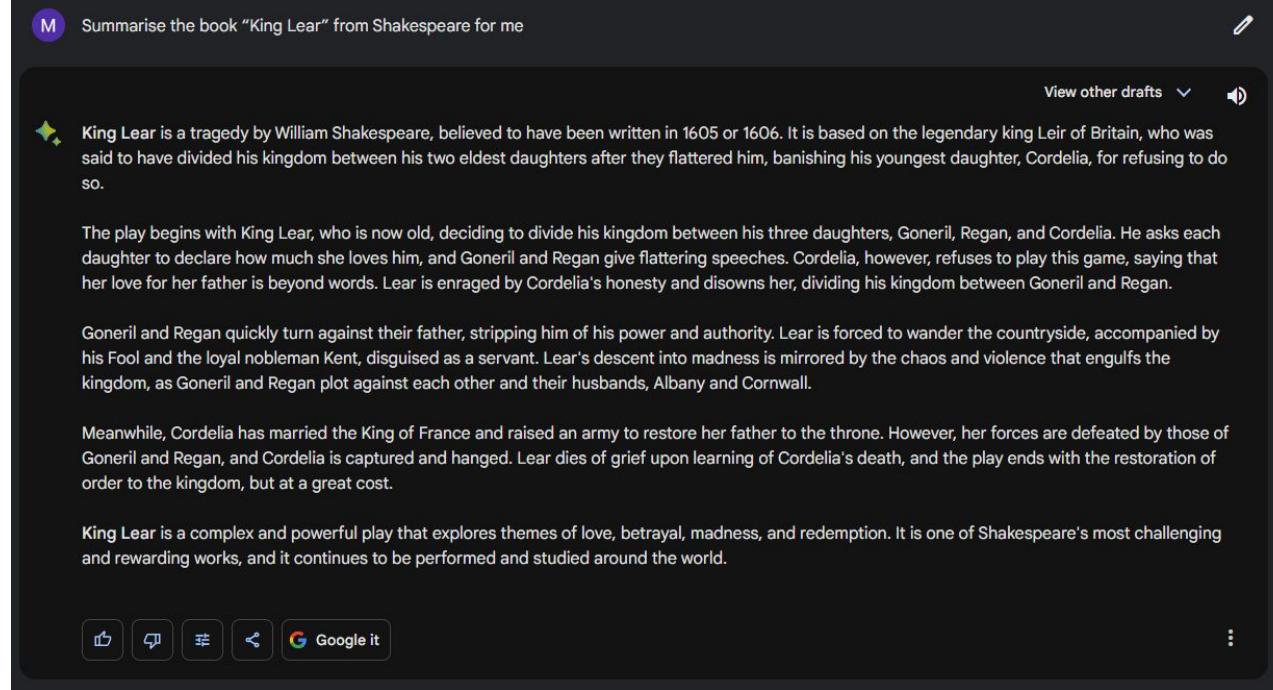
## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

# Application of Large Language Model



M Summarise the book "King Lear" from Shakespeare for me

View other drafts ⚙️ 🔍

King Lear is a tragedy by William Shakespeare, believed to have been written in 1605 or 1606. It is based on the legendary king Leir of Britain, who was said to have divided his kingdom between his two eldest daughters after they flattered him, banishing his youngest daughter, Cordelia, for refusing to do so.

The play begins with King Lear, who is now old, deciding to divide his kingdom between his three daughters, Goneril, Regan, and Cordelia. He asks each daughter to declare how much she loves him, and Goneril and Regan give flattering speeches. Cordelia, however, refuses to play this game, saying that her love for her father is beyond words. Lear is enraged by Cordelia's honesty and disowns her, dividing his kingdom between Goneril and Regan.

Goneril and Regan quickly turn against their father, stripping him of his power and authority. Lear is forced to wander the countryside, accompanied by his Fool and the loyal nobleman Kent, disguised as a servant. Lear's descent into madness is mirrored by the chaos and violence that engulfs the kingdom, as Goneril and Regan plot against each other and their husbands, Albany and Cornwall.

Meanwhile, Cordelia has married the King of France and raised an army to restore her father to the throne. However, her forces are defeated by those of Goneril and Regan, and Cordelia is captured and hanged. Lear dies of grief upon learning of Cordelia's death, and the play ends with the restoration of order to the kingdom, but at a great cost.

King Lear is a complex and powerful play that explores themes of love, betrayal, madness, and redemption. It is one of Shakespeare's most challenging and rewarding works, and it continues to be performed and studied around the world.

Upvote Downvote Share ⋮

Google Bard on summarisation

# Application of Large Language Model

How can I call the animal if it was descendant of a tiger and a lion MS

There are a few possibilities for what to call the offspring of a tiger and a lion:

- Liger - This is the official name for the hybrid offspring of a male lion and a female tiger. Ligers tend to be very large in size compared to either parent species.
- Tigon - This is the name for the offspring of a male tiger and a female lion. Tigons are generally smaller in size than ligers.
- Hybrid - Calling it a hybrid or tiger-lion hybrid is a generic way to refer to the mixed offspring without specifying the exact parentage.
- Crosses - Some people may refer to these big cat hybrids as "crosses" between a tiger and lion.
- Rex - Historically, some lion-tiger crosses have been referred to as "rex" cats. However, this term is not commonly used today.
- Titan - Occasionally these hybrids have been informally dubbed "titans" to reflect their impressive size, especially with ligers. But this is not a standardized name.

So in summary, liger and tigon are the most specific and accurate names reflecting the parental lineage. More casual or generalized options include hybrid, cross, rex or titan. Liger is the most commonly used term for a lion-tiger hybrid.

A Copy Retry

Claude 2 on question answering

# Application of Large Language Model

**Demo of MiniGPT-4**

[NOTE] It is possible that you are waiting in a lengthy queue.  
You can duplicate and use it with a paid private GPU.

**Duplicate this Space**

Alternatively, you can also use the demo on our [project page](#).

This is the demo of MiniGPT-4. Upload your images and start chatting!

[Project](#) [Page](#) [Github](#) [Code](#) [Paper](#) [PDF](#)

Image



Start Chatting

Restart

beam search numbers

1

Temperature

1

MiniGPT-4

Can you tell me what if the person with black coat a human or an alpaca

The person in the image is an alpaca.

How did you get this answer

I was trained on a dataset that included images of alpacas and humans, and I was able to recognize the alpaca in the image.

User

Type and press Enter

Semantic understanding

# Application of Large Language Model

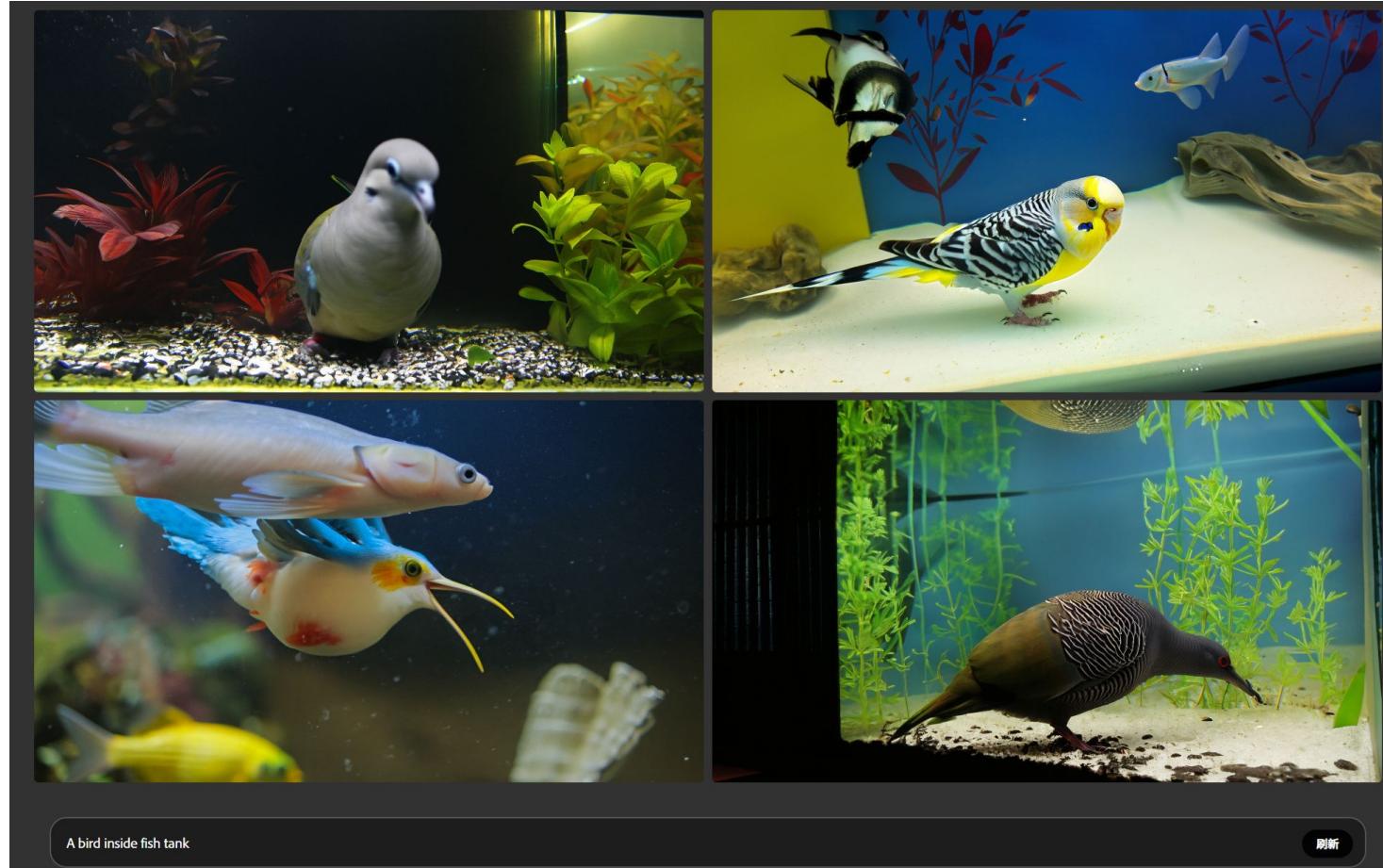
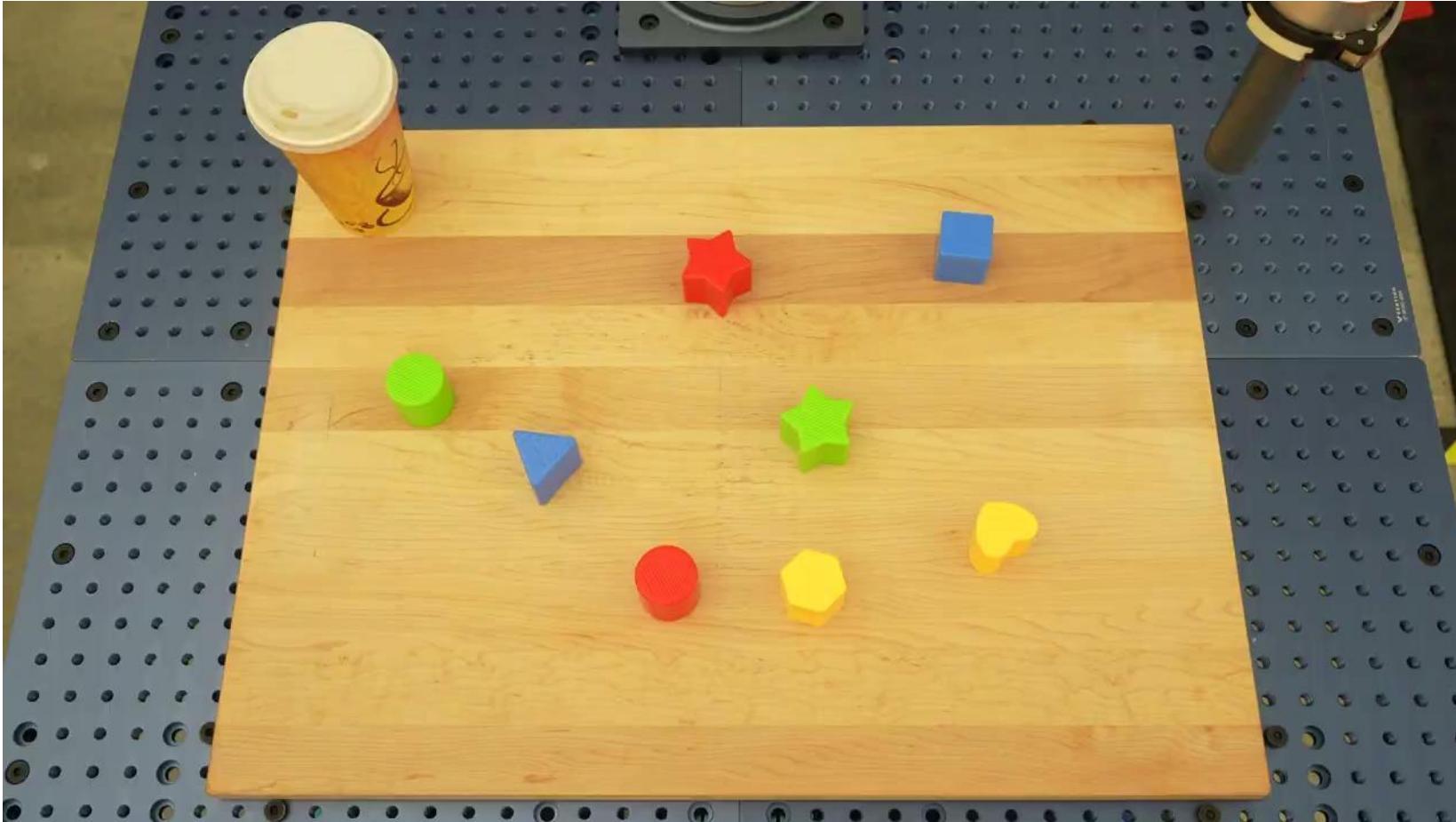


Image generation

# Application of Large Language Model



Embodied support of PaLM-E

# Outline

## Part 1: Transformer

- What can transformers do
- Different types of transformer
- Why is transformer better than RNN
- Attention mechanism
  - General Structure
  - Query, Key & Value
  - Multi-Head Attention
  - Self-Attention

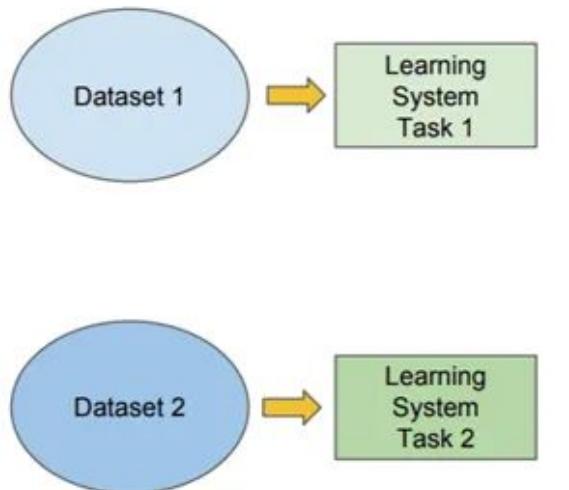
## Part 2: Large Language Models

- Introduction to Language Models and Large Language Models
- Common architecture of LLMs
  - Autoencoding
  - Autoregressive
  - Sequence to sequence
- Application of LLMs
- Training a LLM

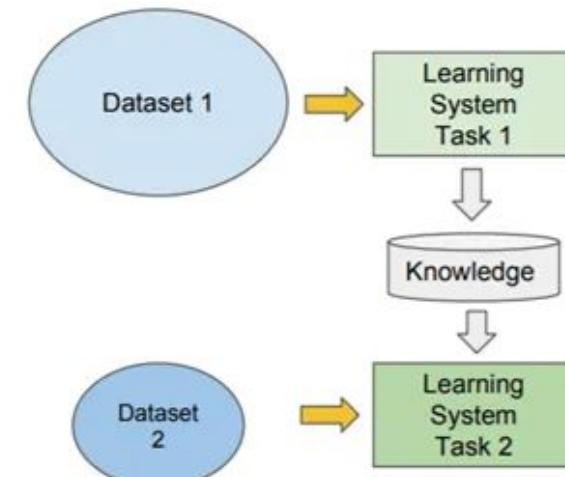
# Training a LLM: Transfer Learning

## Traditional ML vs Transfer Learning

- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



# Why it is so hard to train a LLM

LLaMA-6B with bfloat16 (bf16, 16bits, 2 bytes) float format

To train this model we need:

1. Parameter:  $6B * 2\text{bytes} = 12\text{GB}$
2. Gradient:  $6B * 2\text{bytes} = 12\text{GB}$
3. Optimiser: AdamW normally double of the parameter: 24GB

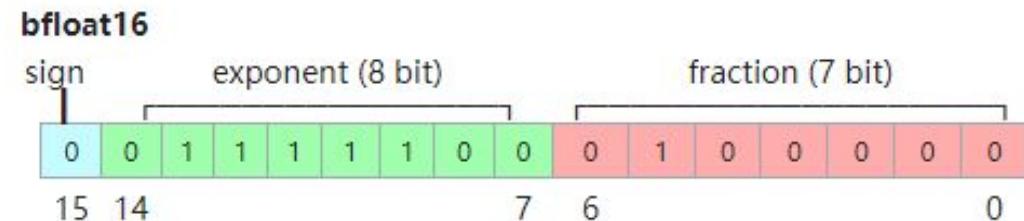
With LLaMA architecture: `hidden_size=4096, intermediate_size=11008, num_hidden_layer=32, context_length=2048`

For each sample we need:  $(4096 + 11008) * 2048 * 32 * 2\text{byte} = 1920\text{MB} \sim 2\text{GB}$

With batchsize=50, the total VRAM required to train this model under this configuration is:

$$48\text{GB} + 50 * 2\text{GB} = 148\text{GB}$$

Which is larger than a single Nvidia H100 GPU! (one of the best AI chips)



```
training_args = TrainingArguments(  
    per_device_train_batch_size=1,  
    gradient_accumulation_steps=4,  
    gradient_checkpointing=True,  
    fp16=True,  
    **default_args,  
)
```

# Why it is so hard to train a LLM

Electronics > Computers & Accessories > Computer Components > Internal Components > Graphics Cards



Click image to open expanded view



## Tesla H100 80GB NVIDIA Deep Learning GPU Compute Graphics Card

Brand: Generic

3.0 3 ratings

\$43,998<sup>00</sup>

Eligible for Return, Refund or Replacement within 30 days of receipt

Graphics NVIDIA Tesla H100

Coprocessor

Brand Generic

Graphics Processor NVIDIA

Manufacturer

Compatible Devices Desktop

Graphics Card PCI Express

Interface

### About this item

- NVIDIA H100 is a high-performance GPU designed for data center and cloud-based applications, optimized for AI workloads designed for data center and cloud-based applications
- Based on the NVIDIA Ampere architecture, it has 640 Tensor Cores and 160 SMs, delivering 2.5x more compute power than the V100 GPU
- With a memory bandwidth of 1.6TB/s and PCIe Gen4 interface, it can handle large-scale data processing tasks efficiently
- Advanced features include Multi-Instance GPU (MIG) technology, enhanced NVLink, and enterprise-grade reliability tools

Report incorrect product information.

\$43,998<sup>00</sup>

FREE delivery **September 25 - 28**. Details

Deliver to Minghao - Brooklyn  
11201

**Only 5 left in stock - order soon**

Qty: 1 ▾

Add to Cart

Buy Now

Payment Secure transaction

Ships from Global Tech Discount

Sold by Global Tech Discount

Returns Eligible for Return,  
Refund or Replacement  
within 30 days of receipt

Add to List

**Buy it on Amazon Business**

Save up to 1% on this product  
with business-only pricing.

Create a free account

# How can I get access to the LLMs

- Huggingface (<https://huggingface.co/>): Include most of open-source models implementation
- GitHub (<https://github.com/>): Some research groups uploaded the source of the model to GitHub such as LLaMA (<https://github.com/facebookresearch/llama>)
- API available by AI companies: Such as OpenAI's API (<https://openai.com/blog/openai-api>)
- Or you can write your own LLM with deep learning frameworks...

# Suggested extra tutorials

- Deep Learning: Dive into Deep Learning (<https://d2l.ai/>)
- PyTorch: Official PyTorch tutorials (<https://pytorch.org/tutorials/>) and practical code (<https://github.com/yunjey/pytorch-tutorial>)
- Huggingface: Huggingface NLP course  
(<https://huggingface.co/learn/nlp-course/chapter1/1>)

# Demonstrations and Assignments

- Transformer Basic:

[https://drive.google.com/file/d/1\\_EYvoDVPCNrwGDFbNFq0x8nGcBxw6rdO/view  
?usp=drive\\_link](https://drive.google.com/file/d/1_EYvoDVPCNrwGDFbNFq0x8nGcBxw6rdO/view?usp=drive_link)

- LLM Inference:

[https://drive.google.com/file/d/1pqNTiwfixZ7Ee07S1lesH41Ry8cYMfp3/view?usp  
=drive\\_link](https://drive.google.com/file/d/1pqNTiwfixZ7Ee07S1lesH41Ry8cYMfp3/view?usp=drive_link)

- LLM Finetuning:

[https://drive.google.com/file/d/1D-fkUeFPV1ki2VAEd-mRJYDGFVC8HkDh/view?  
usp=drive\\_link](https://drive.google.com/file/d/1D-fkUeFPV1ki2VAEd-mRJYDGFVC8HkDh/view?usp=drive_link)