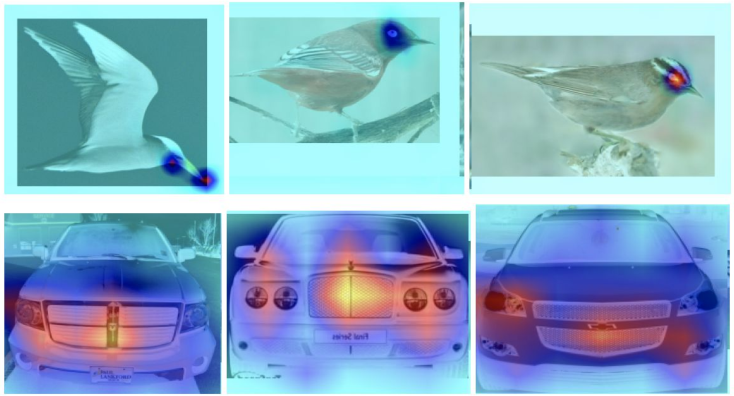
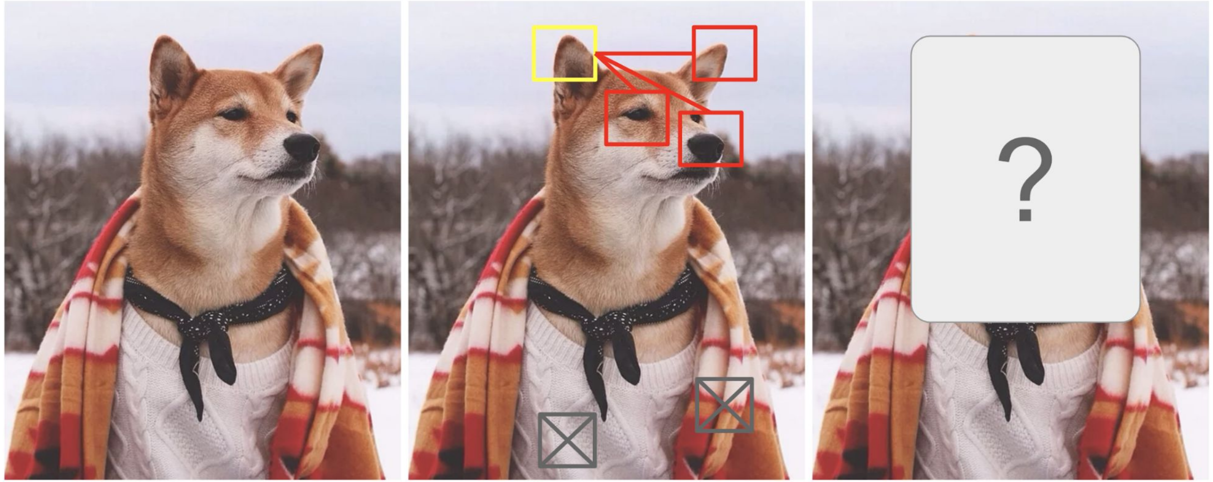
Attention in Deep Learning



Visual Attention & Attention in Text

Attention is **the ability to choose and concentrate on relevant stimuli**.In neural networks, attention is a technique that mimics cognitive attention. The effect enhances some parts of the input data while diminishing other parts. In other words, **attention is the way how we pay visual attention to different regions** of an image or correlate words in one sentence.



Attention in Deep Learning

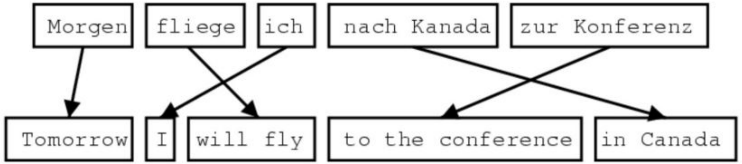
Attention was initially designed in the context of Neural Machine Translation using Seq2Seq Models

Machine Translation (MT) is the task of translating a sentence *x* from one language (the source language) to a sentence *y* in another language (the target language).

x: *L'homme est né libre, et partout il est dans les fers*

y: *Man is born free, but everywhere he is in chains*

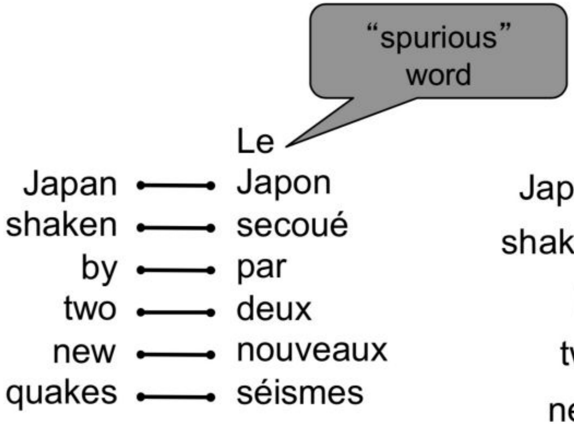
Alignment, i.e. word-level correspondence between source sentence x and target sentence y

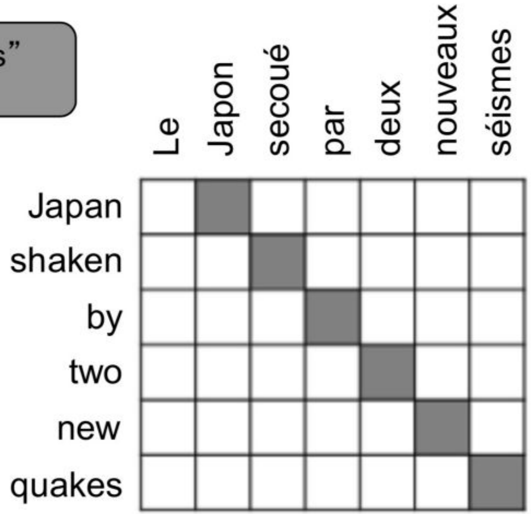


What is Alignment

Alignment is the correspondence between particular words in the translated sentence pair.

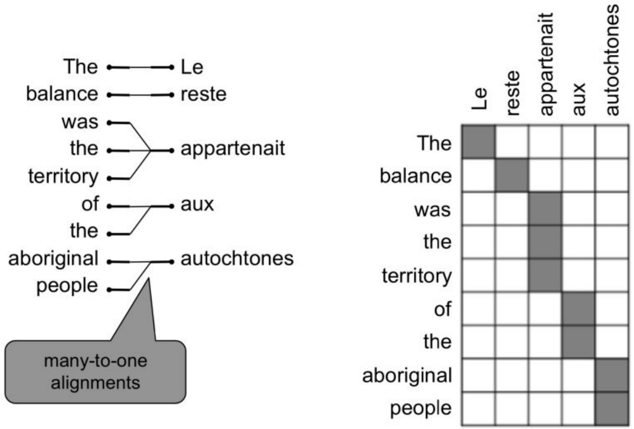
- Typological differences between languages lead to complicated alignments!
- Note: Some words have no counterpart



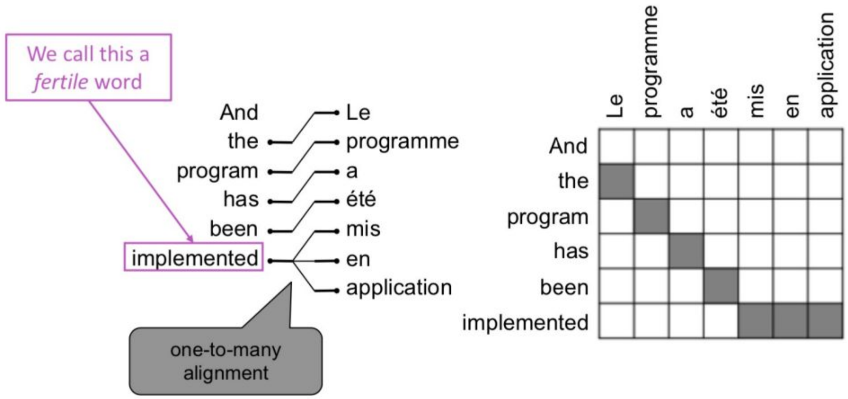


Alignment is Complex

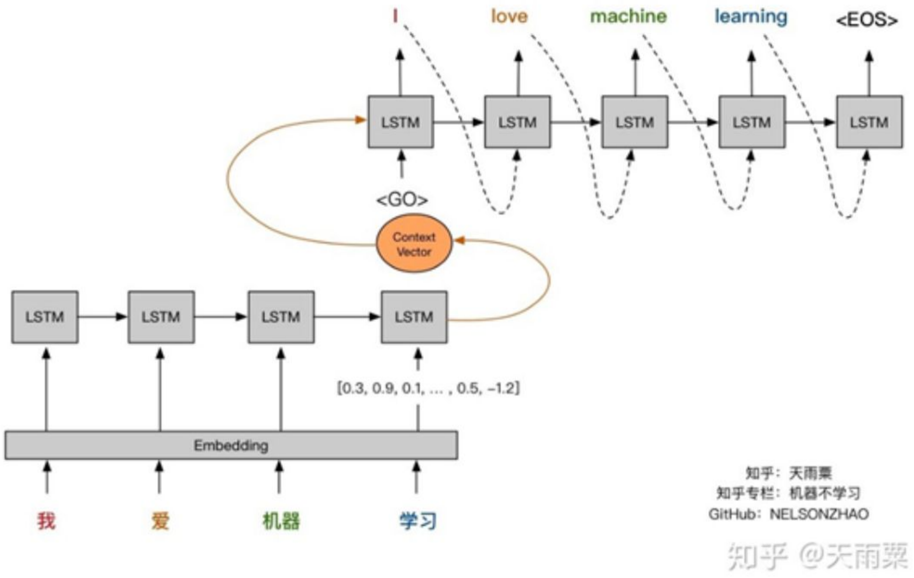
Alignment can be many-to-one



Alignment can be one-to-many

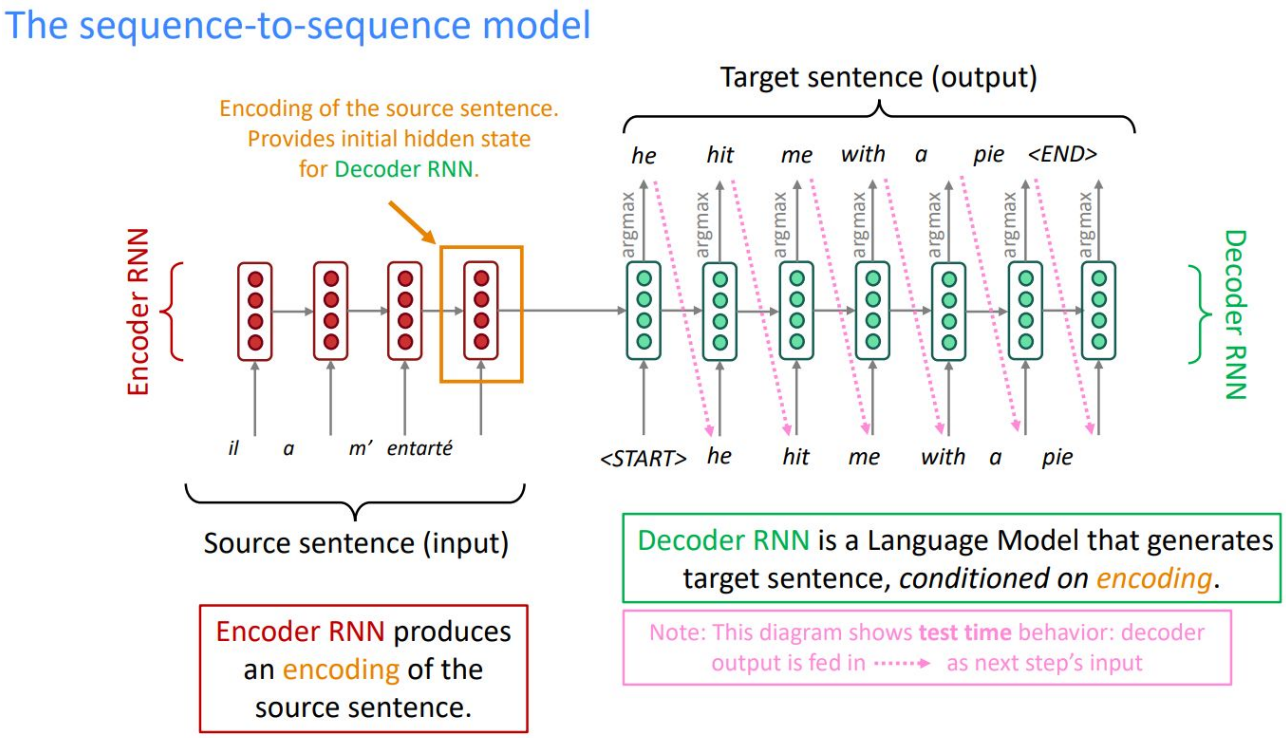


Encoder-Decoder in seq2seq



Encoder-Decoder with simple fixed context vector

Neural Machine Translation aka seq2seq

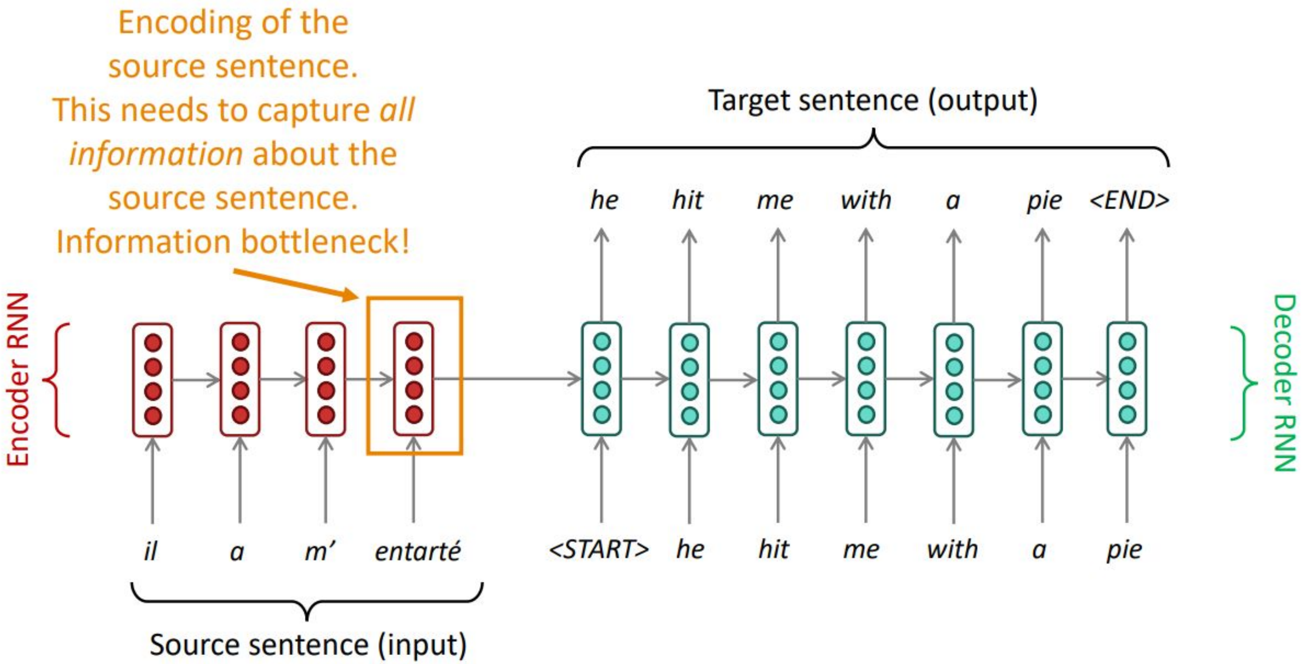


encoder processes the input sequence and compresses the information into a context vector of a *fixed length*.

This representation is expected to be a good summary of the meaning of the *whole* source sequence.

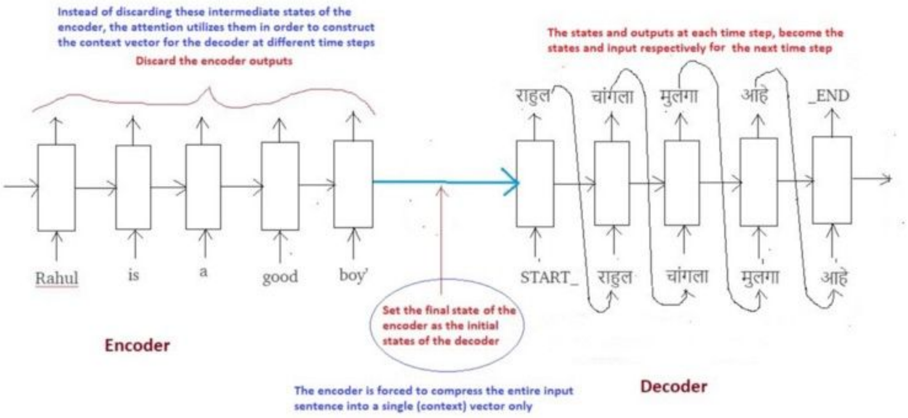
!! A critical and apparent disadvantage of this fixed-length context vector design is incapability of remembering long sentences.

Sequence-to-sequence: the bottleneck problem

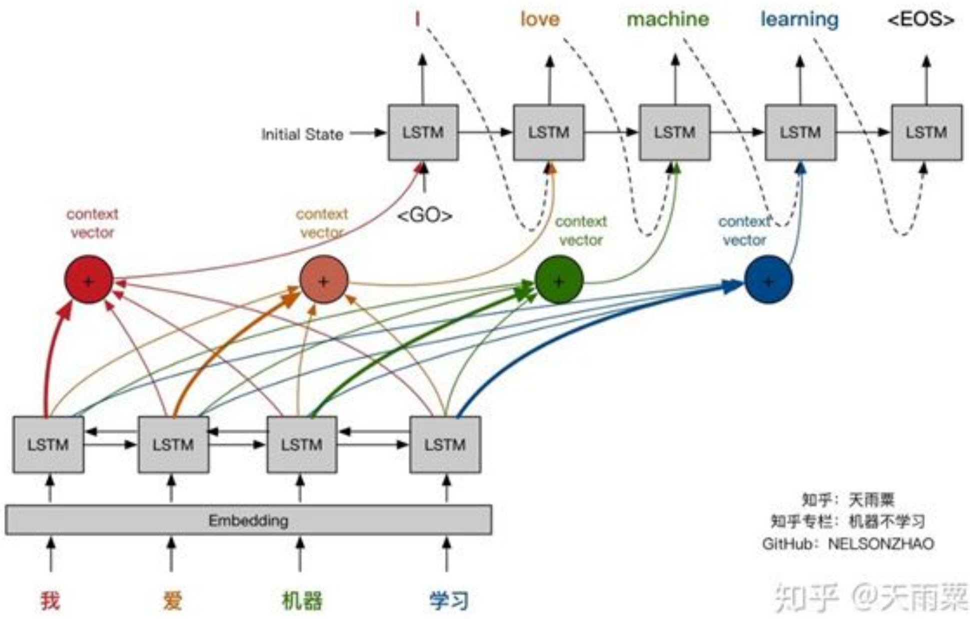


Born for Translation : Attention

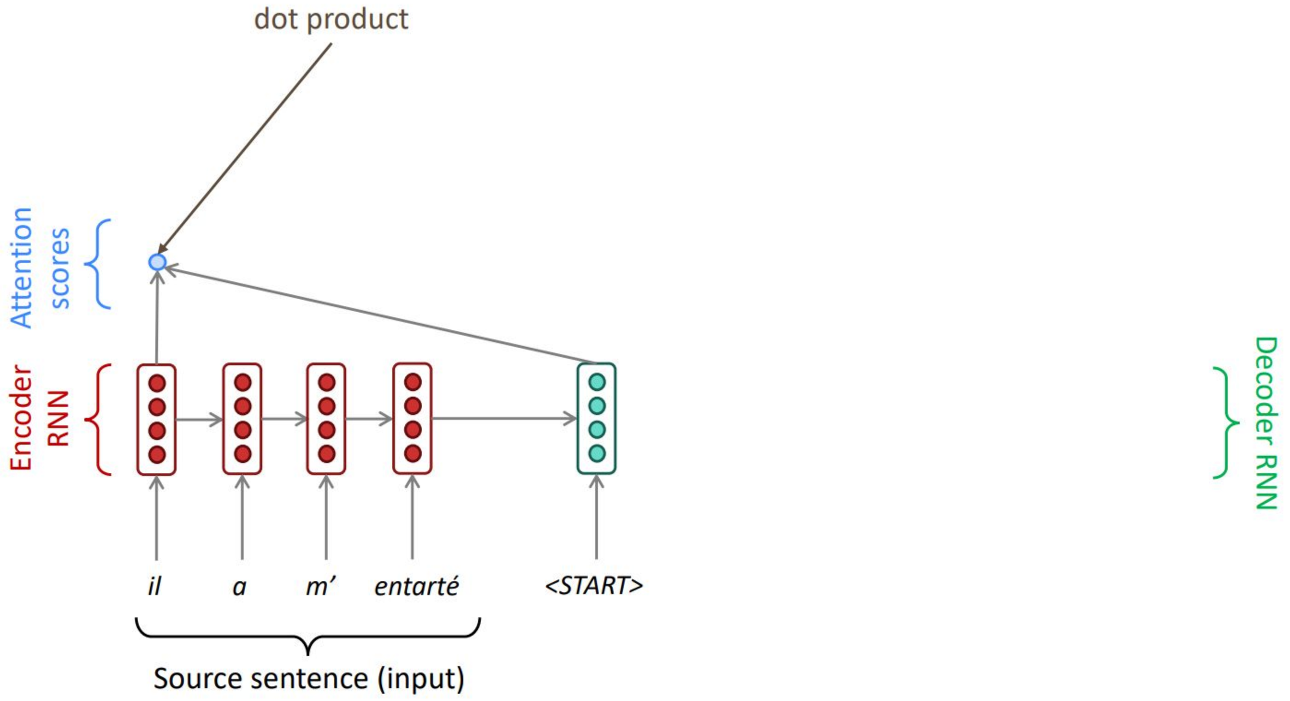
The attention mechanism was born to help memorize long source sentences in neural machine translation. In the traditional **Seq2Seq** model, we discard all the intermediate states of the encoder and **use only its final states (vector) to initialize the decoder.** This technique works good for smaller sequences, however as the length of the sequence increases, a single vector becomes a bottleneck and it gets very difficult to summarize long sequences into a single vector. The central idea behind **Attention is not to throw away those intermediate encoder states but to utilize all the states in order to construct the context vectors** required by the decoder to generate the output sequence.



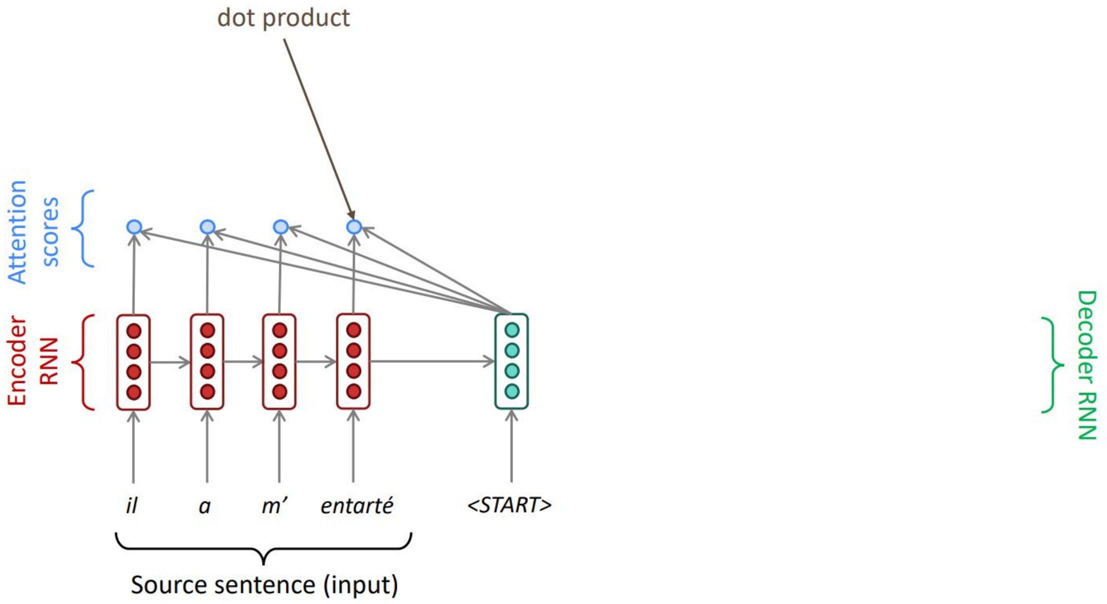
Encoder-decoder with attention-based mechanism



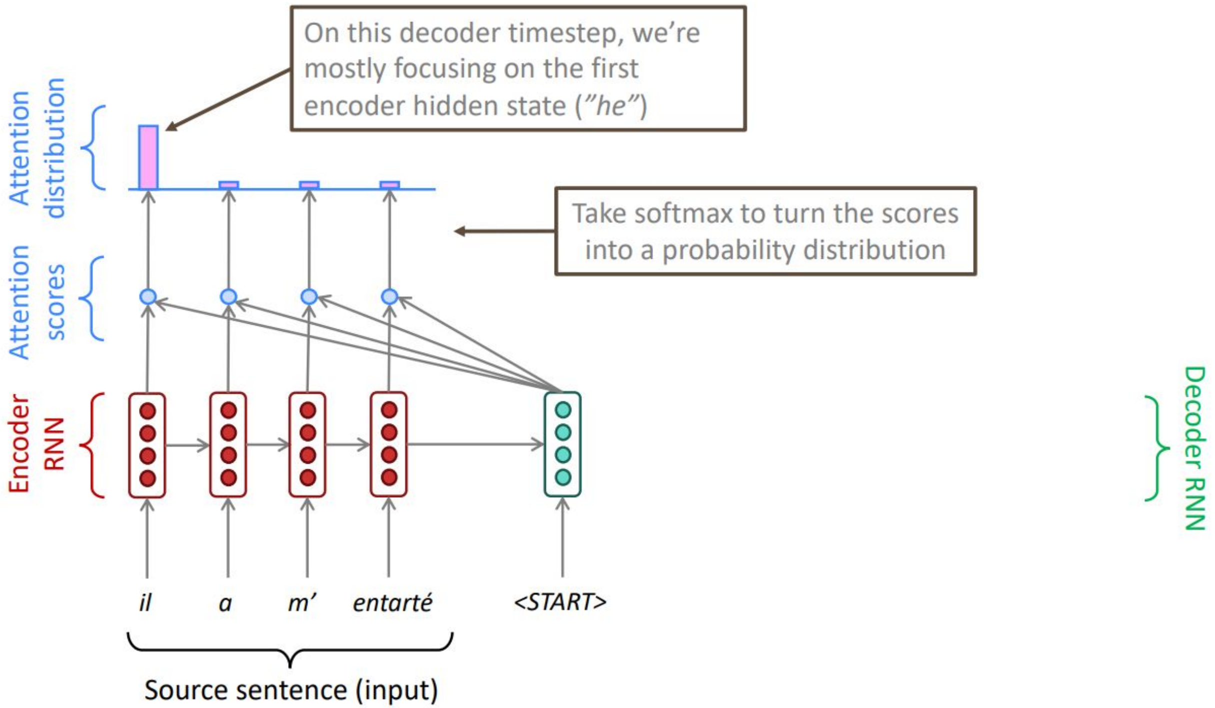
Sequence to sequence with Attention



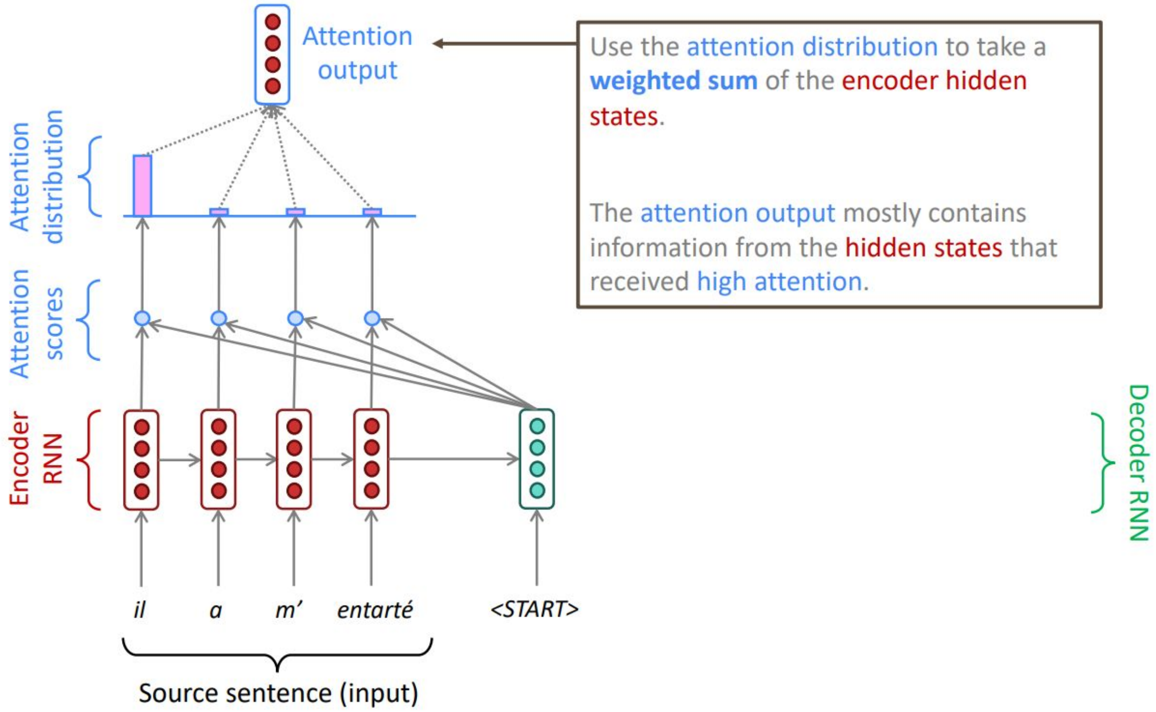
Sequence to sequence with Attention



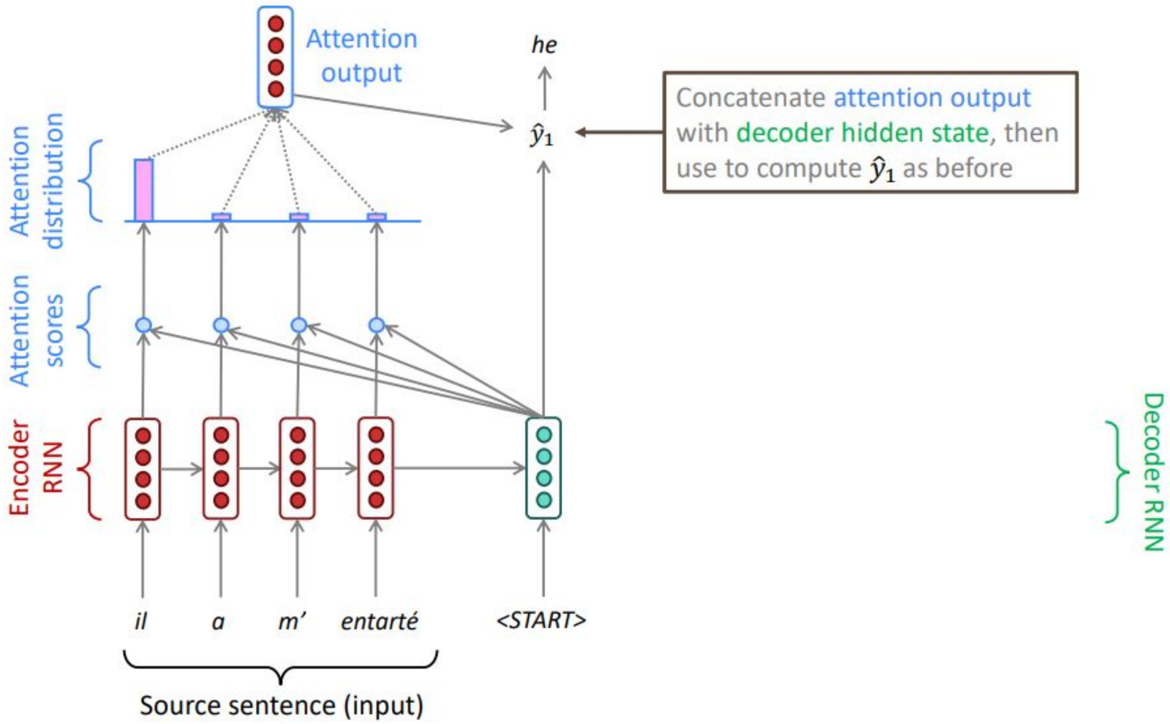
Sequence to sequence with Attention



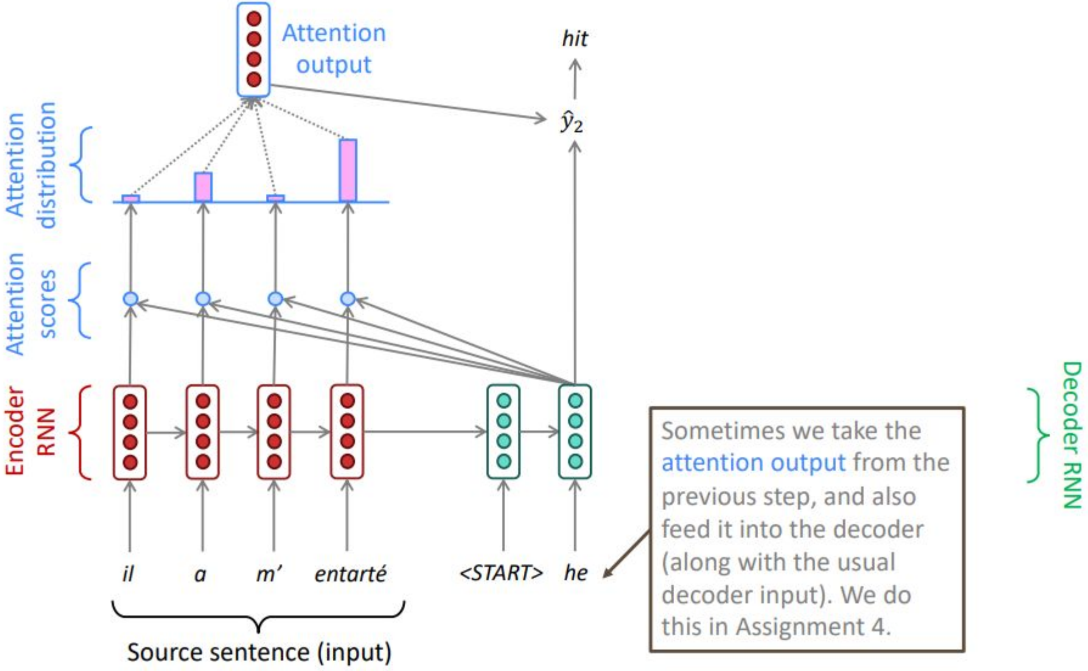
Sequence to sequence with Attention



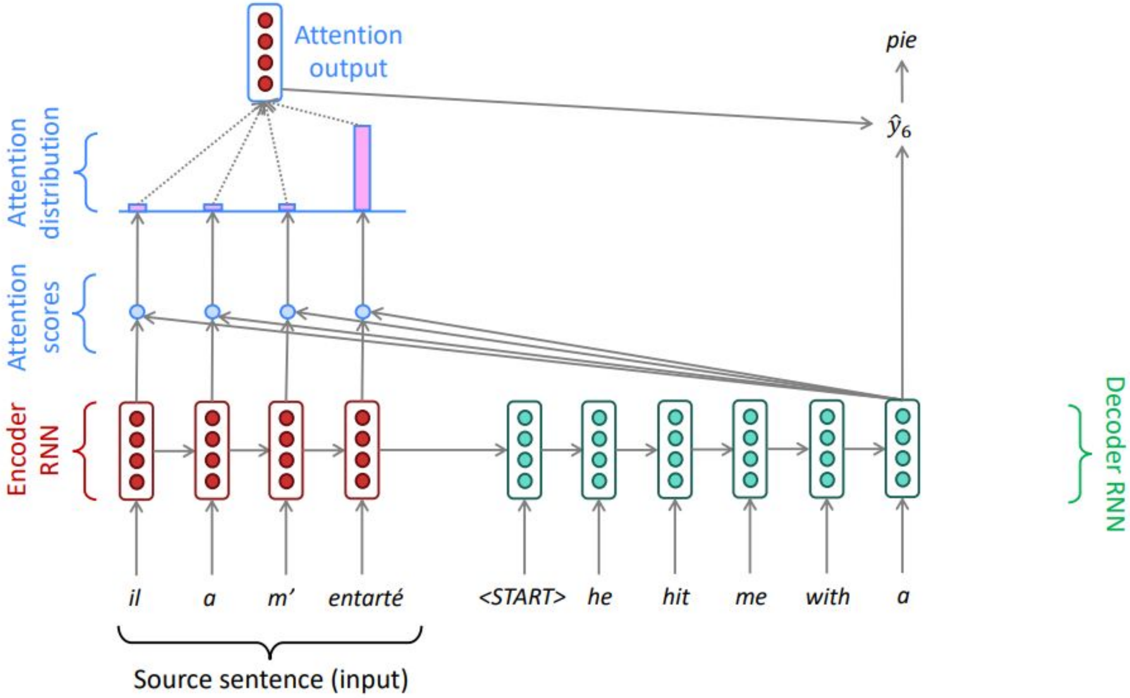
Sequence to sequence with Attention



Sequence to sequence with Attention



Sequence to sequence with Attention



Attention: in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Variants in Attention

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$
  2. Taking softmax to get *attention distribution* α:

$$\alpha = \text{softmax}(\boldsymbol{e}) \in \mathbb{R}^N$$

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

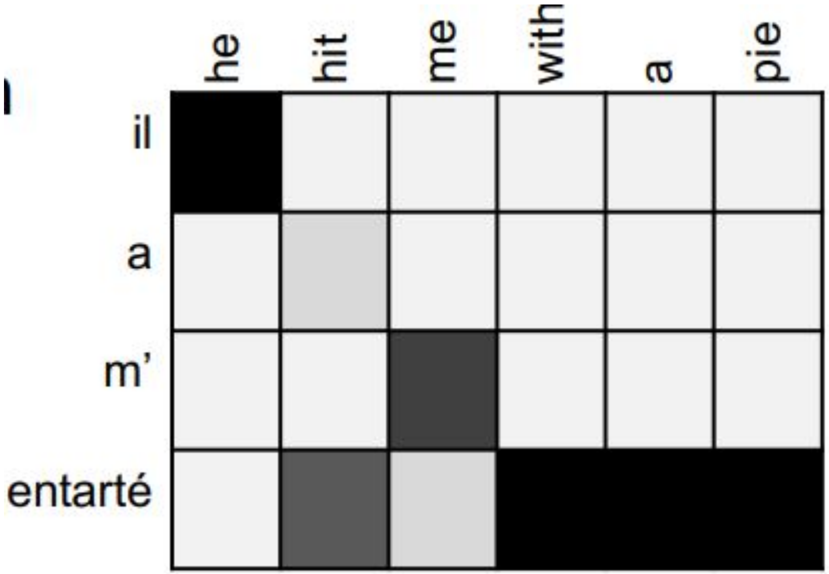
thus obtaining the *attention output* ***a*** (sometimes called the *context vector*)

There are multiple ways to do this

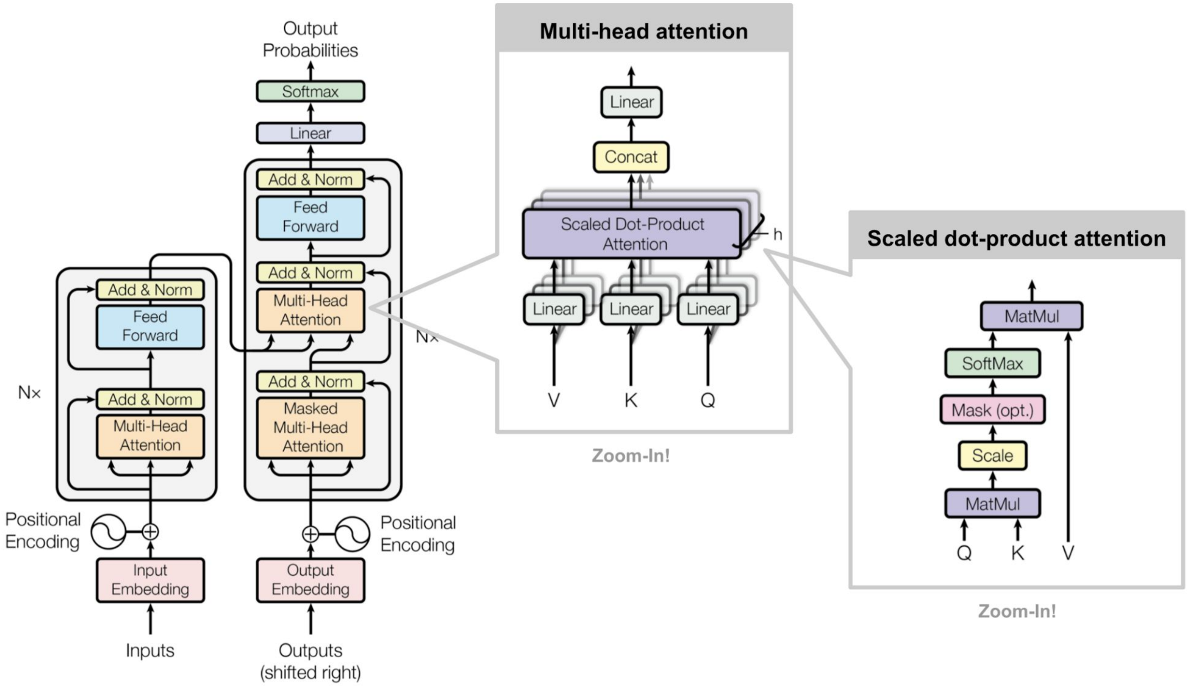
Types of Score Calculation

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Attention Result



Attention is All you Need with Transformer Architecture



Encoder: Self-Attention

- Step 1: For each word x_i , calculate its **query**, **key**, and **value**.

  $$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between **query and keys.**

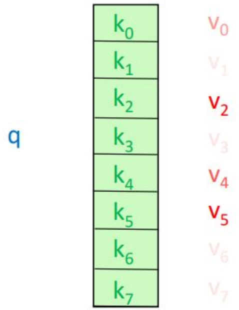
  $$e_{ij} = q_i \cdot k_j$$

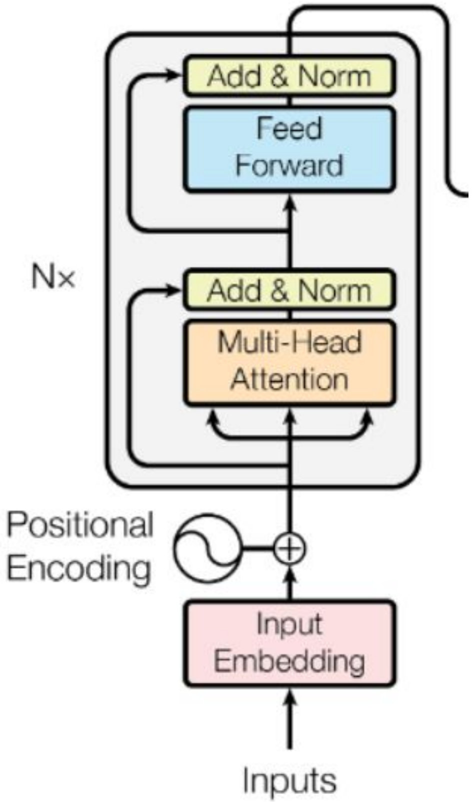
- Step 3: Take the softmax to normalize attention scores.

  $$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**.

  $$Output_i = \sum_j \alpha_{ij} v_j$$





Positional Encoding

Suppose we have an input sequence of length L and we require the position of the kth object within this sequence.

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

k: Position of an object in input sequence, $0 \leq k < L/2$

d: Dimension of the output embedding space

$P(k, j)$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

n: User defined scalar. Set to 10,000 by the authors of Attention is all You Need.

i: Used for mapping to column indices $0 \leq i < d/2$. A single value of i maps to both sine and cosine functions

Positional Encoding Example

Positional Encoding Matrix with d=4, n=100

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

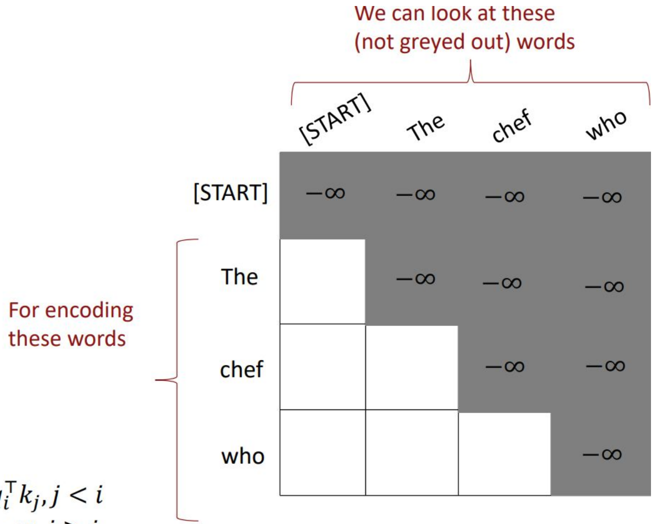
Positional Encoding Matrix for the sequence 'I am a robot'

Masked Multi-Head Attention

At a high-level, we hide (mask) information about future tokens from the model

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^{\top} k_j, j < i \\ -\infty, j \geq i \end{cases}$$



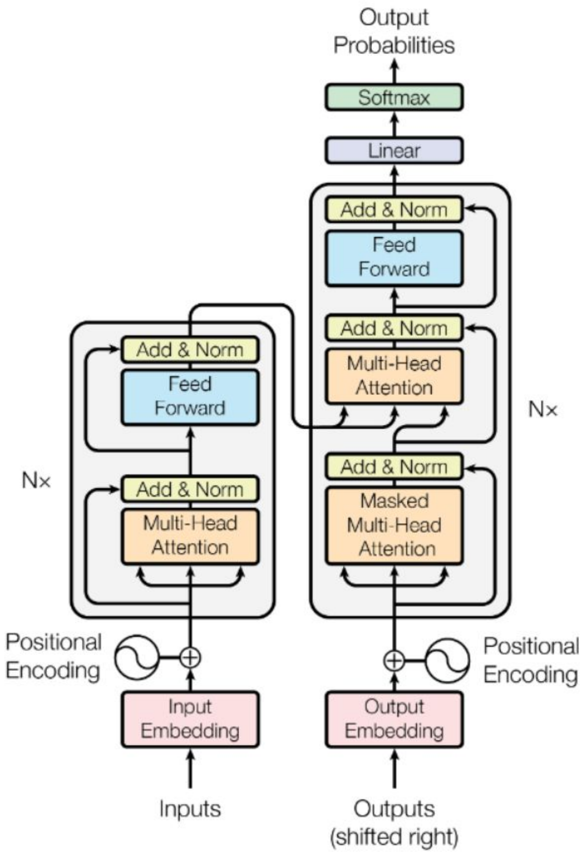


Image captioning

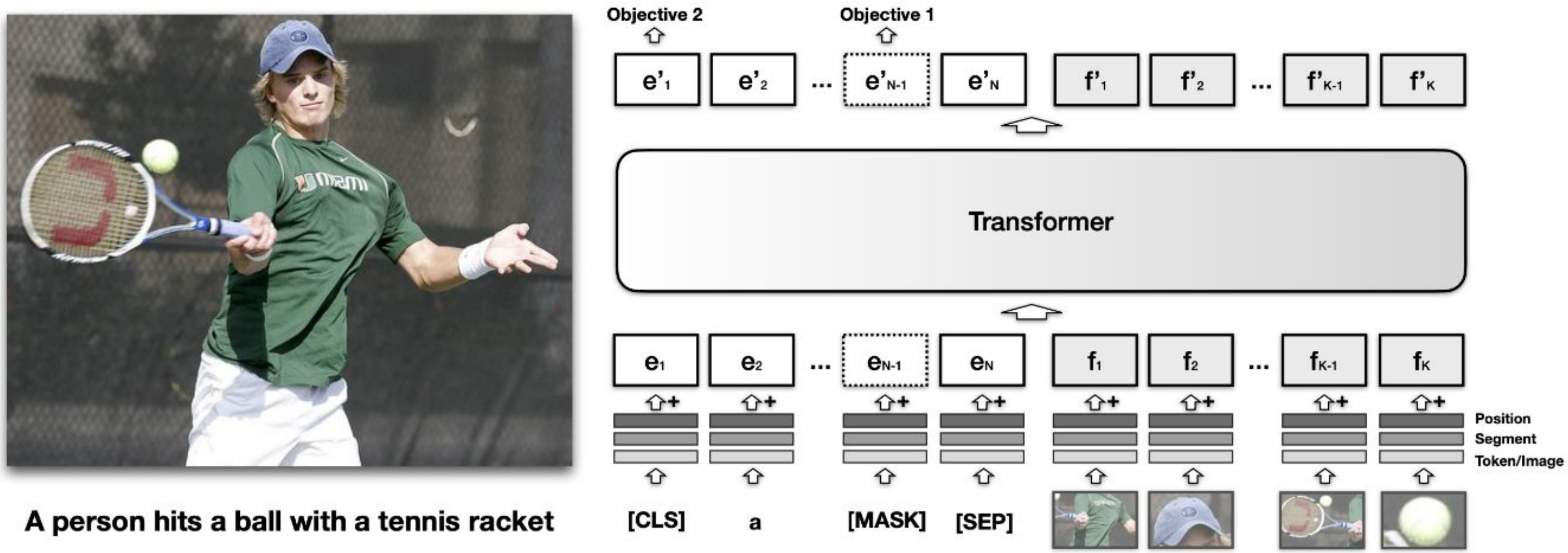


Fig. 1. VisualBERT is trained on the combination of both text and image embeddings. (Image source: Li et al. 2019)

Image patches to transformer

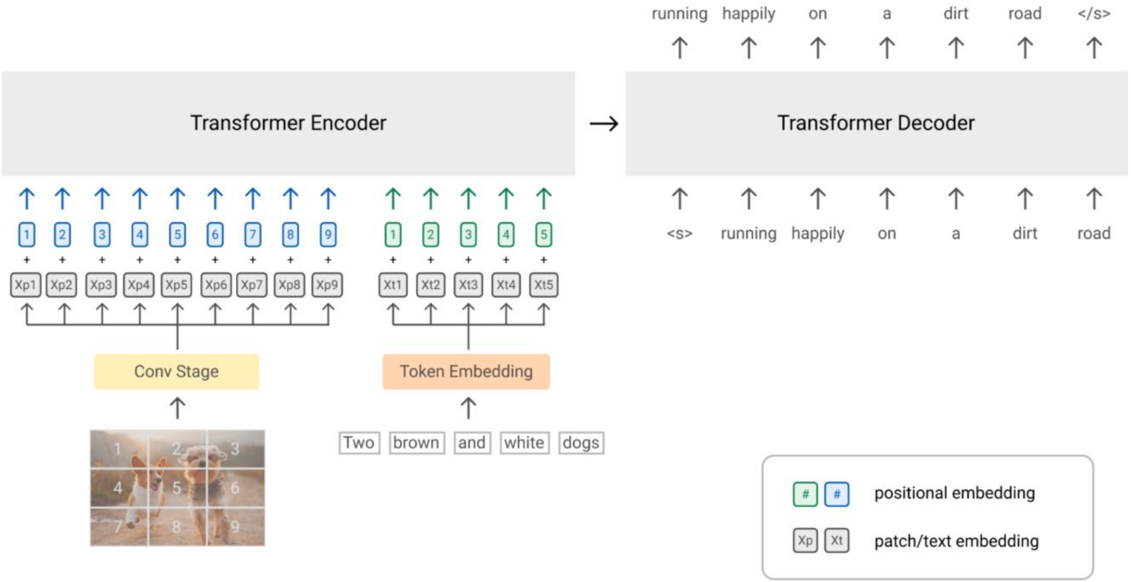


Fig. 3. Training architecture for SimVLM, where the image patches are processed by the cross-attention encoder and the text decoder has causal attention. (Image source: Wang et al. 2022)

Question answering via Transformer

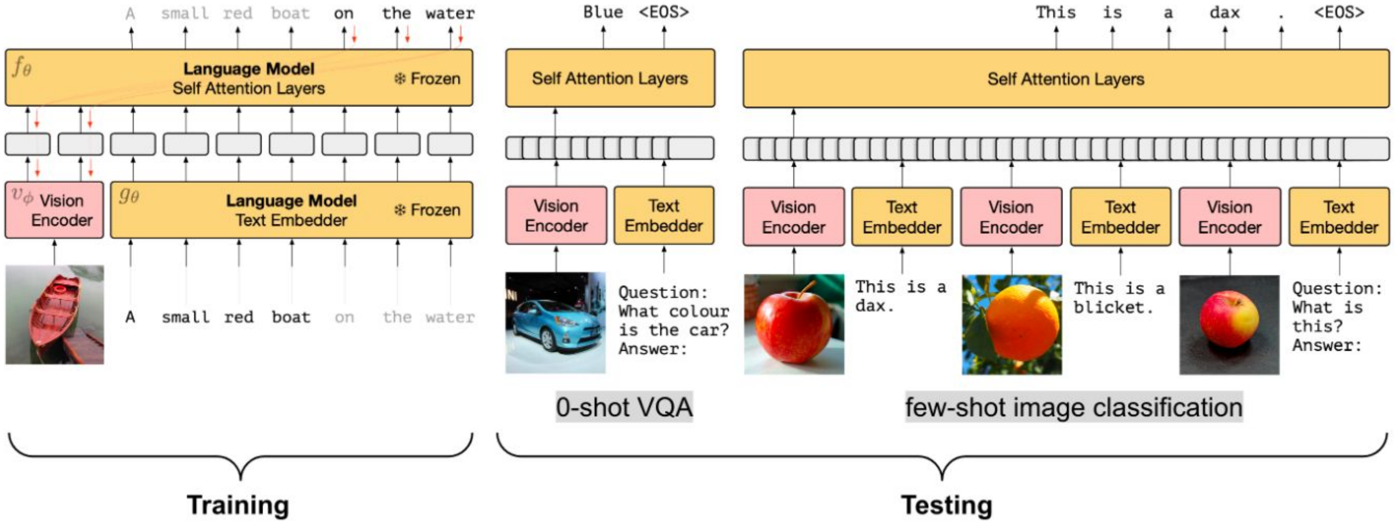


Fig. 6. Illustration of Frozen model (left) training architecture and (right) testing pipeline. (Image source: Tsimpoukelli et al. 2021)

Thank you for your attention :)