ITMO UNIVERSITY

Saint Petersburg, Russia

Специализированные технологии машинного обучения / Advanced Machine learning Technologies

Lecture 7 – Attention Models and Transformers

Outline

1. Seq2Seq problems challenges
2. RNN and LSTM
3. Attention mechanism
4. Transformers: “Attention is all you need”
5. Vision Transformers

Attention history

Attention mechanism was first proposed in the NLP field and still actively researched.
Key milestoned are the following, including developing Transformers ("Attention is All you need") in 2017 and Vision Transformers in 2020:

- Seq2Seq, or RNN Encoder-Decoder (Cho et al. (2014), Sutskever et al. (2014))
- Alignment models (Bahdanau et al. (2015), Luong et al. (2015))
- Visual attention (Xu et al. (2015))
- Hierarchical attention (Yang et al. (2016))
- Transformer (Vaswani et al. (2017))

Long-sequence problem

Example:

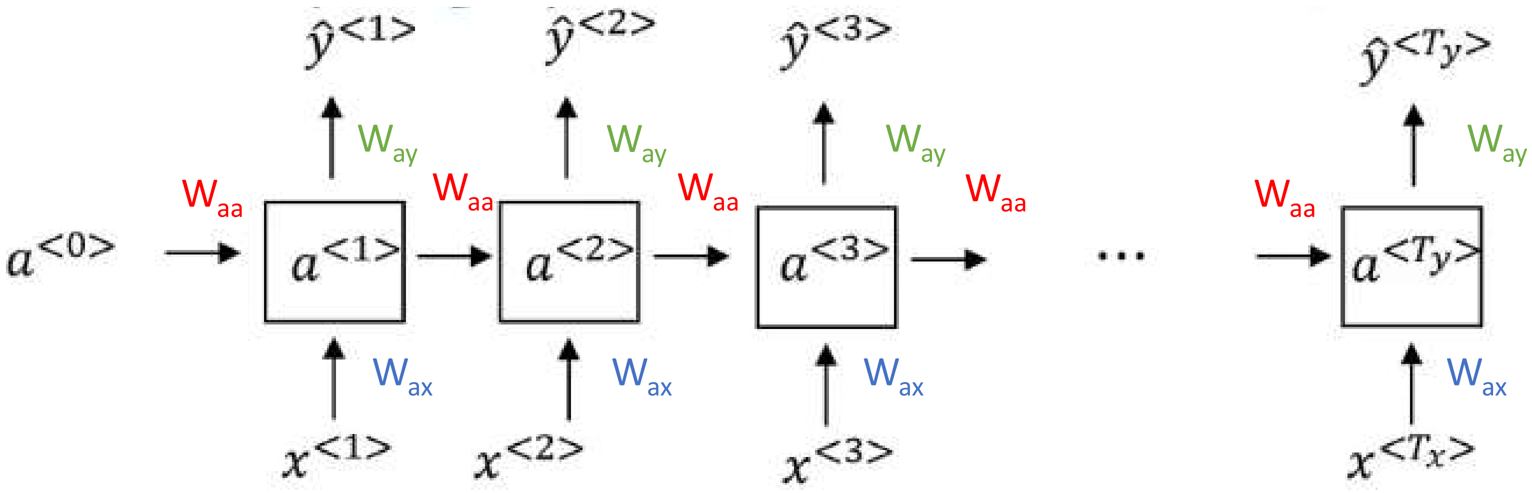
- I have been to **Paris!**
- And how was it?
- It was wonderful! In childhood I spent a lot of time there.
  I have a perfect opportunity to learn ______ language.

23 words later – "Paris"

– it is the only reason to put "French" into the gap.

- The model should 'remember' old (and very old) inputs to be able to use them when it is the reason for it
- In classic dense networks the weights of "early" words is exponentially small.
- We should capture long- (and very long-) term dependencies!

Parameters of RNN



W_{ax} - matrix of parameters for "input gate"

W_{ay} - matrix of parameters for "output gate"

W_{aa} - matrix of parameters for "hidden state"

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

- Shared weights for every state of a cell.
- We have T_x copies of the RNN cell

Problems with “classic” RNN

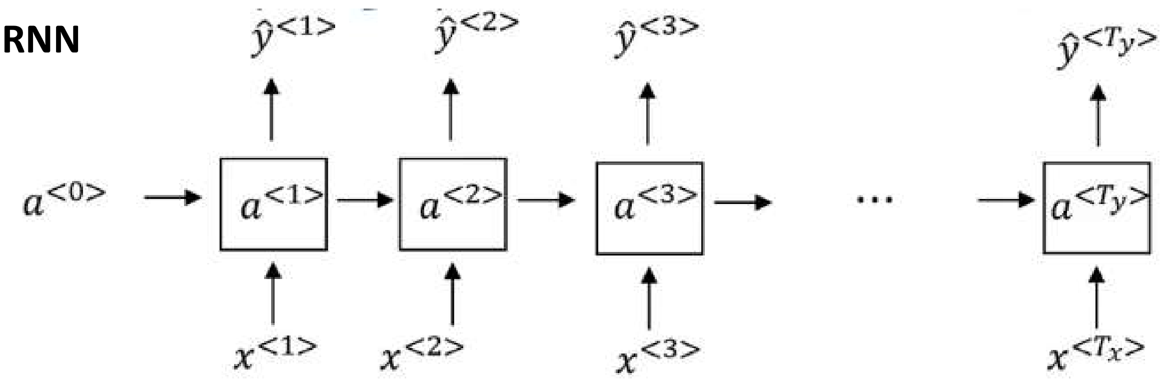
- **Exploding gradients**

Gradients increase exponentially with the number of layers (length of the sequence)

Solution:

→ Clipping values of gradients

RNN

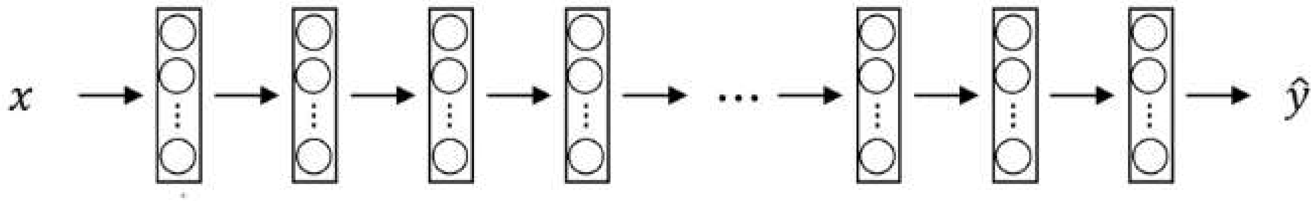


- **Vanishing gradients**

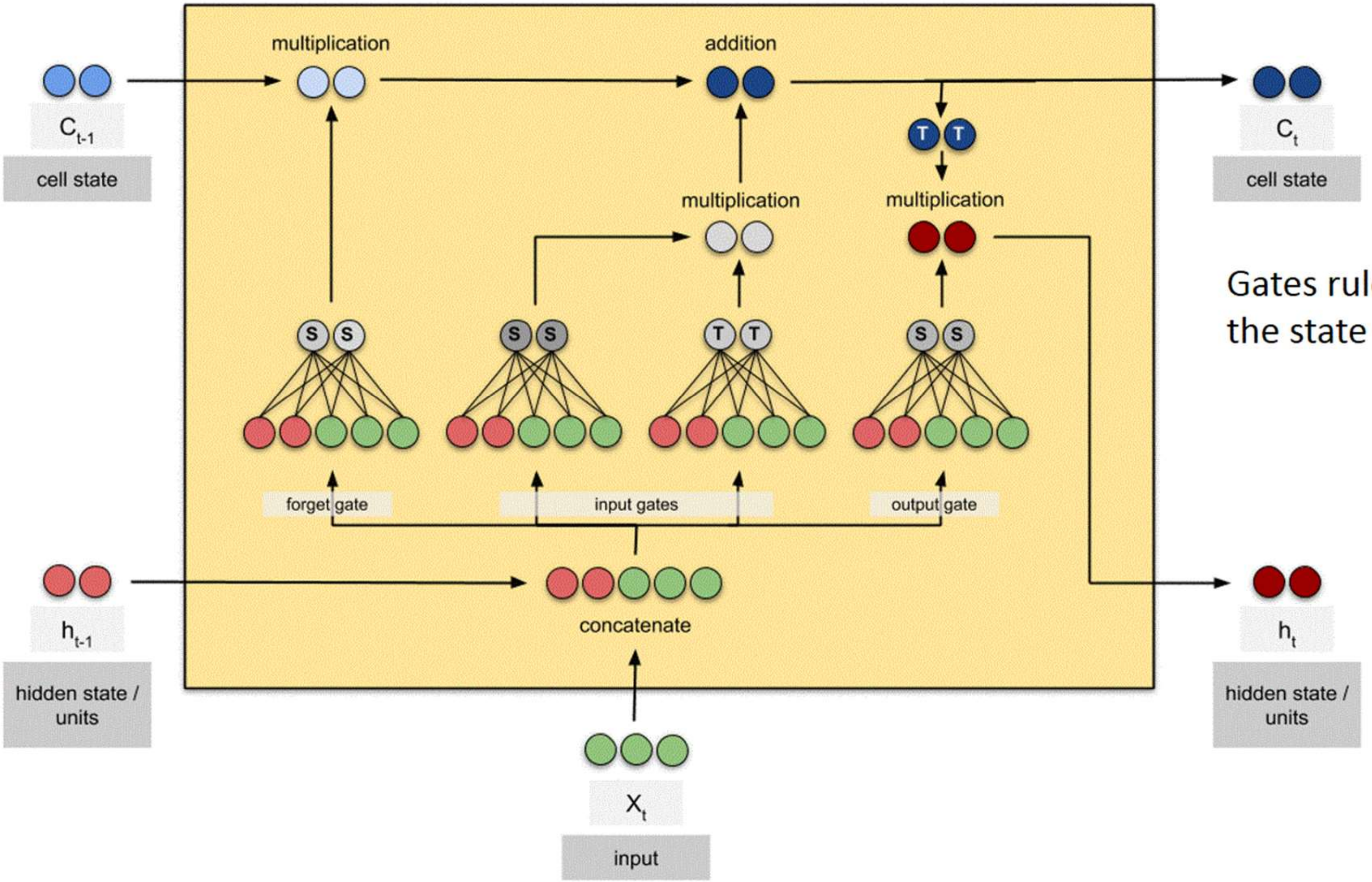
Gradients decrease exponentially with the number of layers (length of the sequence)

Solution - ?

Plane Deep Network



May be smth like residual connections like in plane deep networks (ResNet)?...



Gates rule when to update the state of the cell

Long Short-Term Memory (LSTM) cell

ITMO UNIVERSITY

$$\begin{aligned}
c'_t &= \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_{c'}\right) && \textit{candidate cell state} \\
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right) && \textit{input gate} \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) && \textit{forget gate} \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right) && \textit{output gate} \\
c_t &= f_t \odot c_{t-1} + i_t \odot c'_t, && \textit{cell state} \\
h_t &= o_t \odot \tanh(c_t) && \textit{block output}
\end{aligned}$$

"Constant error carousel": (assuming f_t=1):

$$c_t = c_{t-1} + i_t \odot c'_t.$$

$$\frac{\partial c_t}{\partial c_{t-1}} = 1.$$

There is no non-linearity in dependence c_t from c_{t-1} so it provides gradient propagation through time without vanishing;
- Problems with exploding gradients => shrinking and bordering the absolute values of gradients

Vectorization:

$$\begin{pmatrix} W_{xc} & & & & W_{hc} & & & \\ & W_{xi} & & & & W_{hi} & & \\ & & W_{xf} & & & & W_{hf} & \\ & & & W_{xo} & & & & W_{ho} \end{pmatrix} \begin{pmatrix} x_t \\ x_t \\ x_t \\ x_t \\ h_{t-1} \\ h_{t-1} \\ h_{t-1} \\ h_{t-1} \end{pmatrix} = \begin{pmatrix} W_{xc}x_t + W_{hc}h_{t-1} \\ W_{xi}x_t + W_{hi}h_{t-1} \\ W_{xf}x_t + W_{hf}h_{t-1} \\ W_{xo}x_t + W_{ho}h_{t-1} \end{pmatrix}.$$

→ Parallelization!

Machine Translation

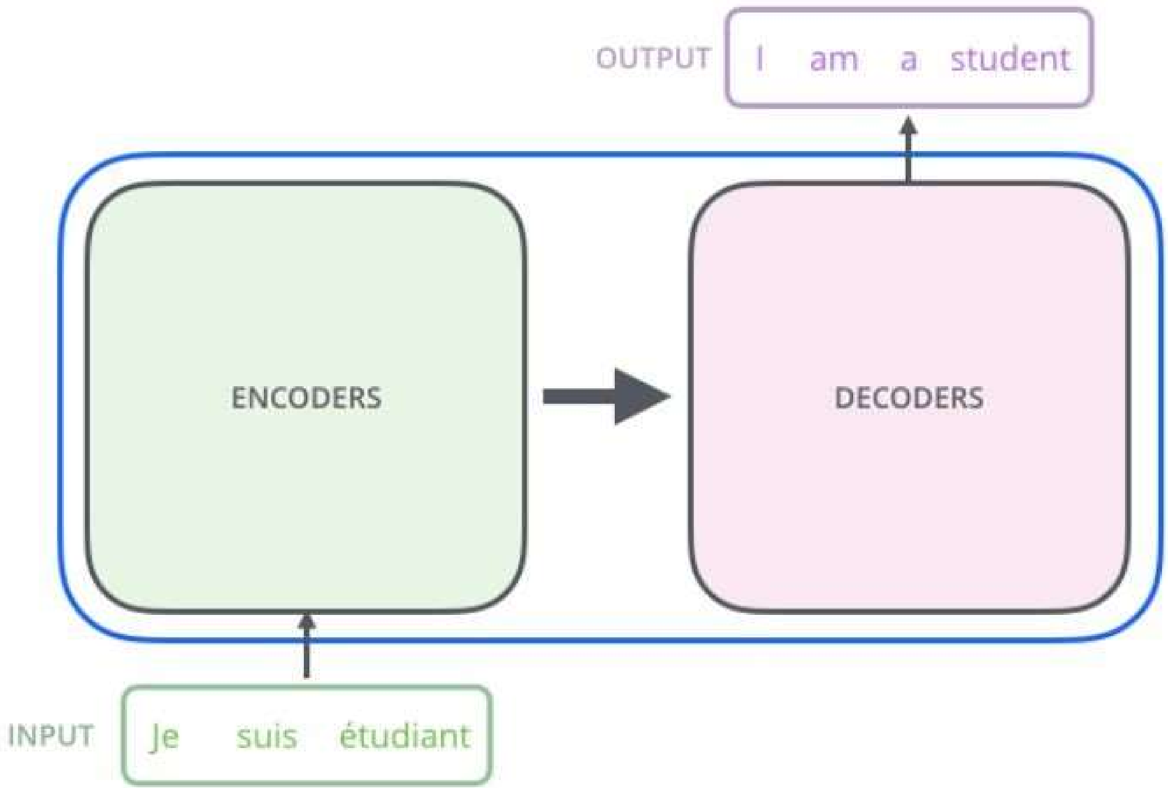
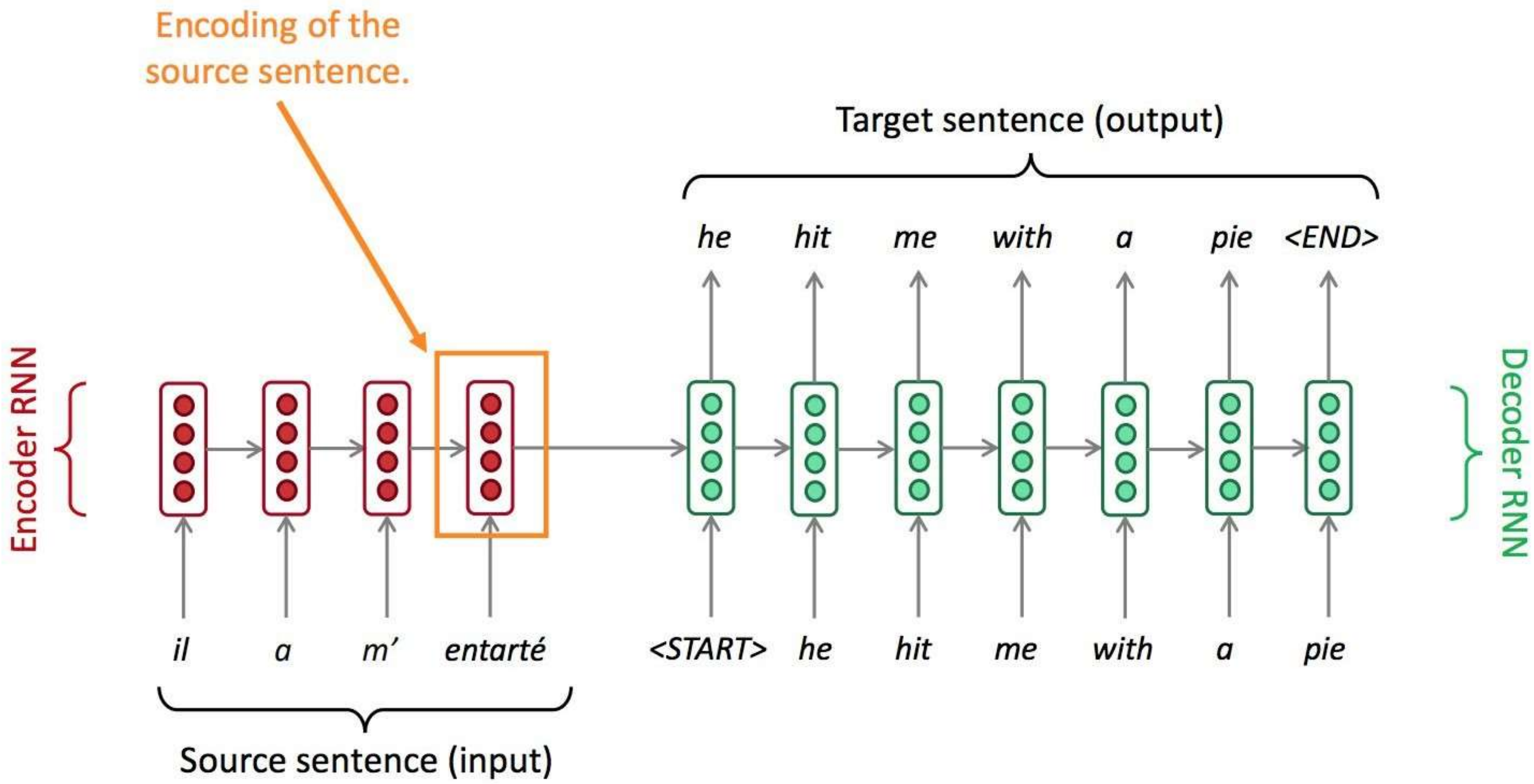


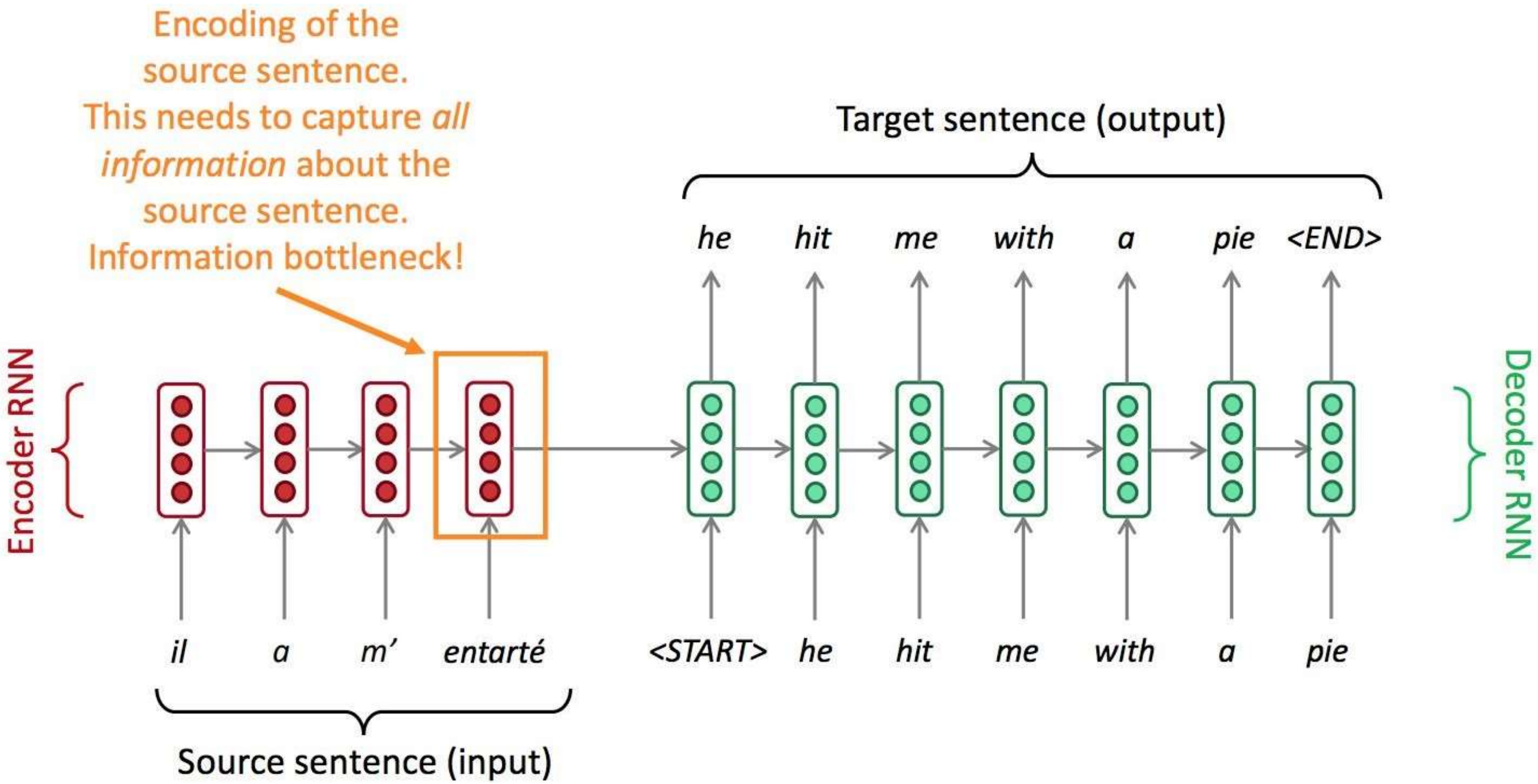
Image Credit: Jay Alammar

Machine Translation



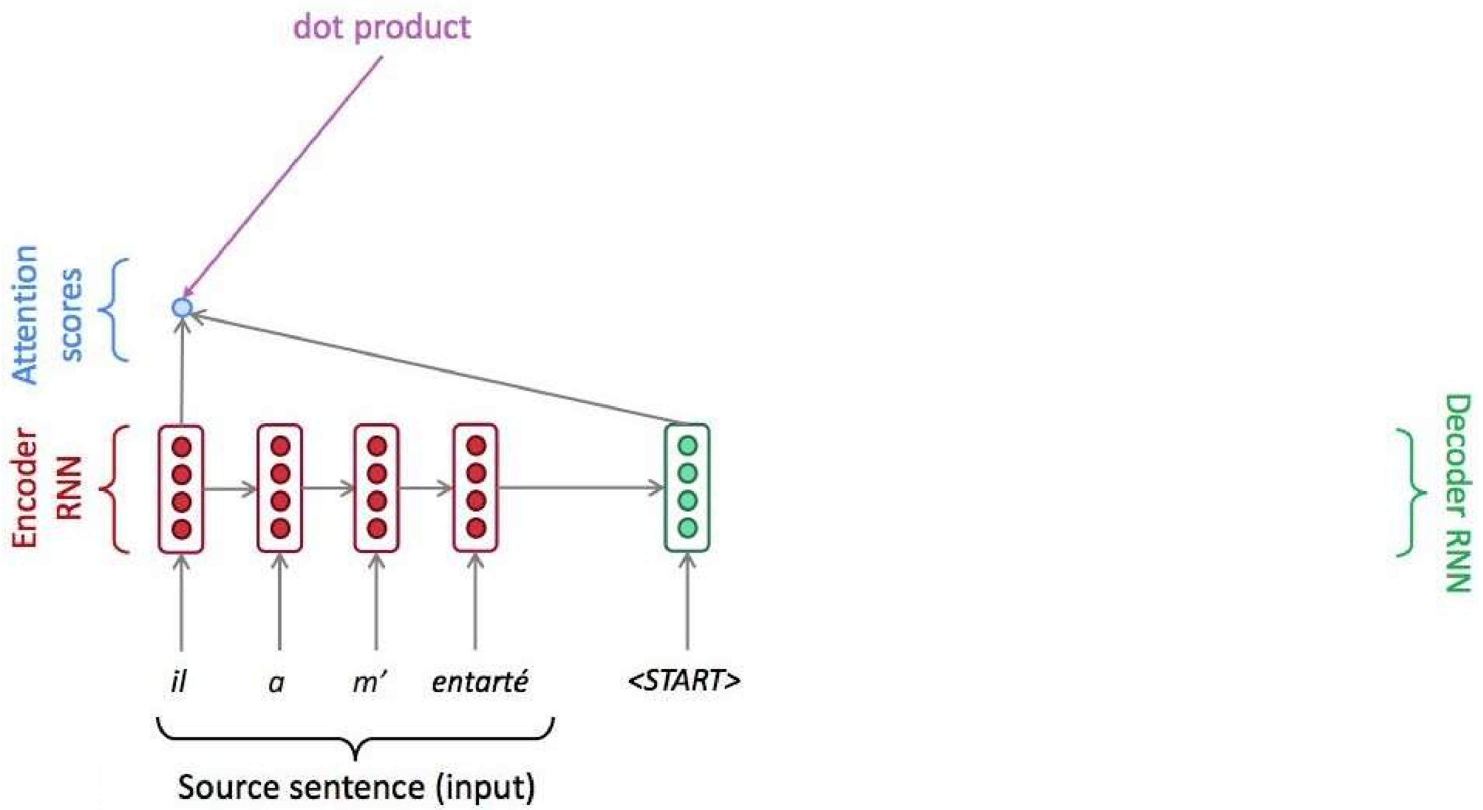
Slide Credit: Abigail See, Stanford CS224n

Machine Translation



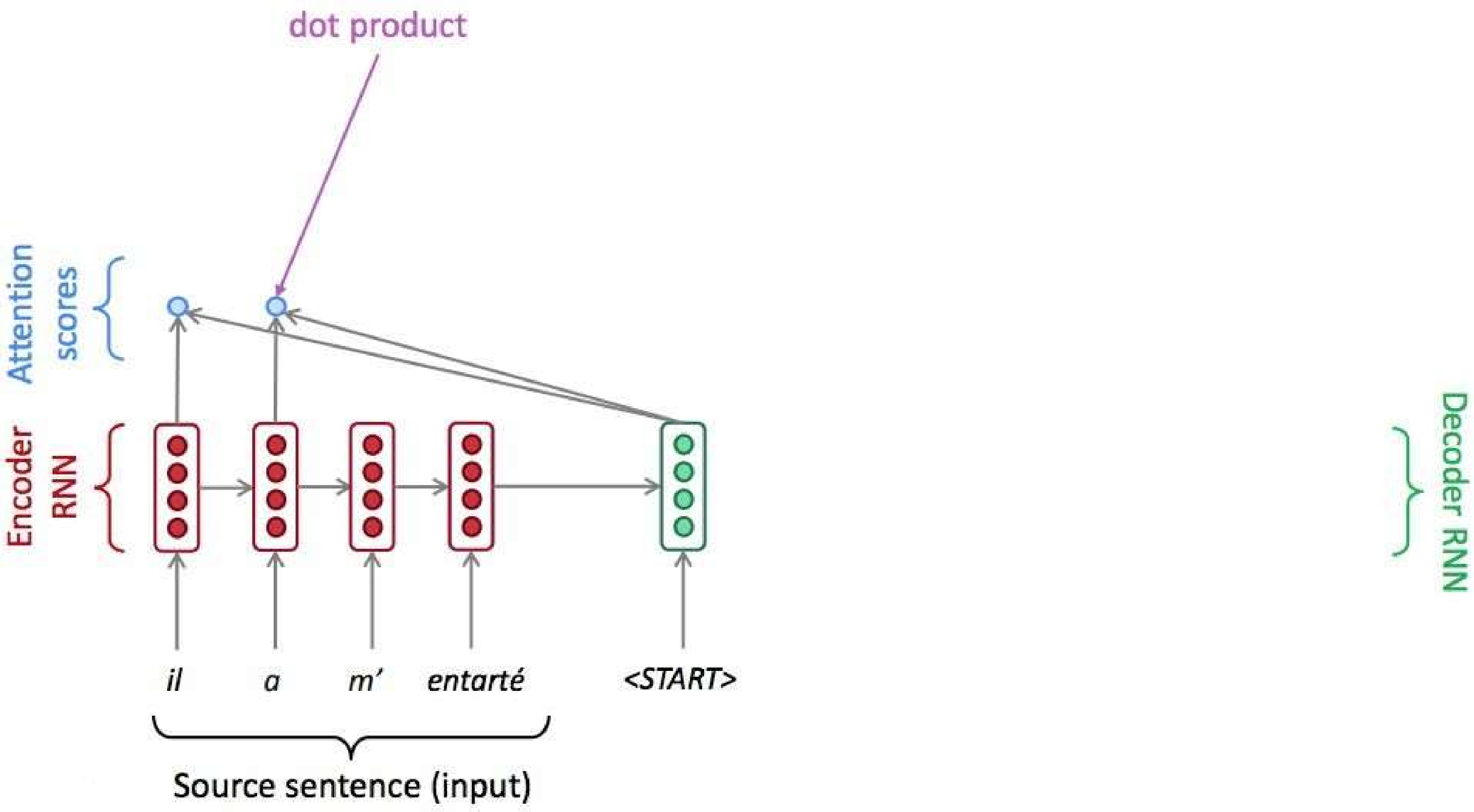
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



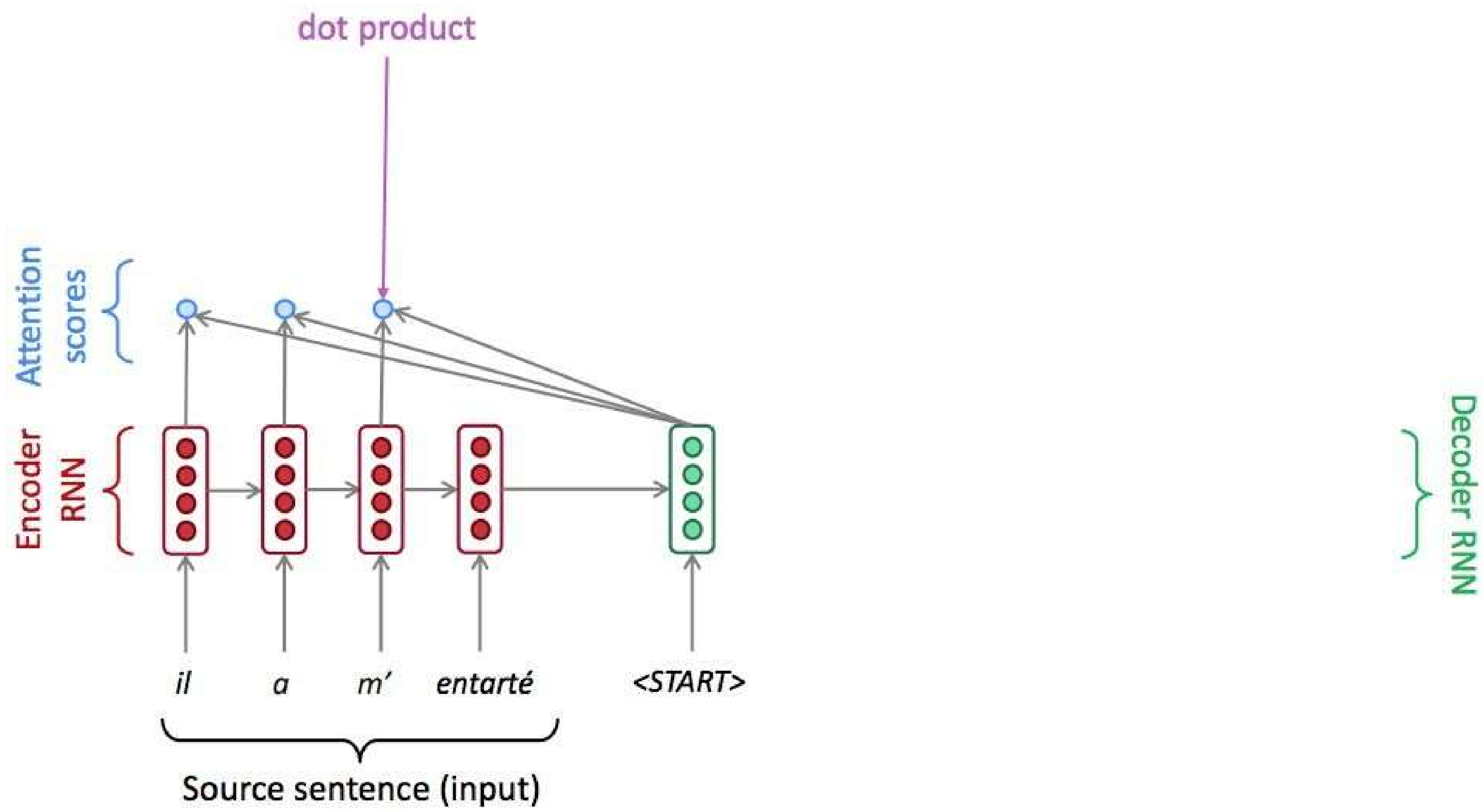
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



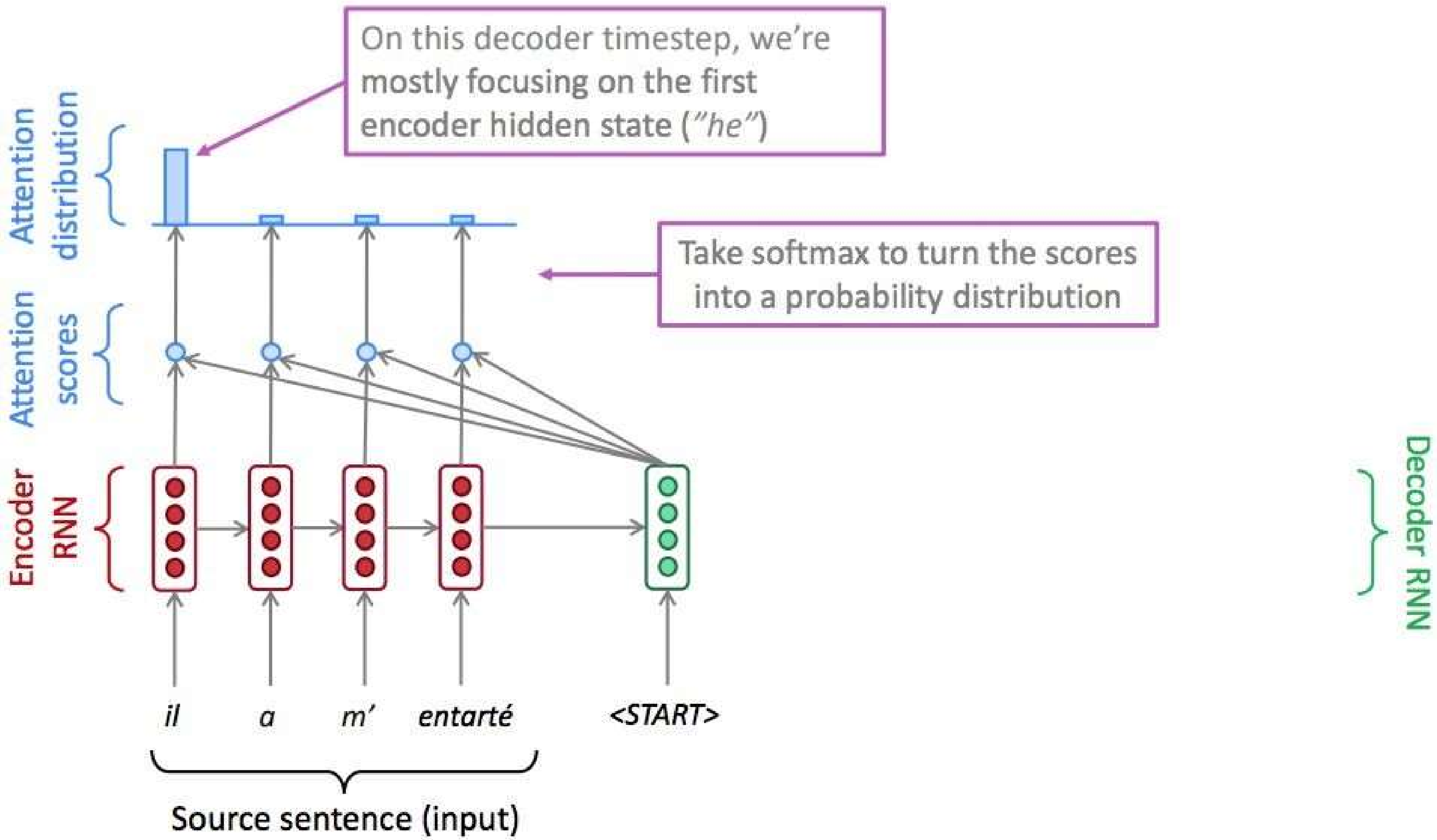
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



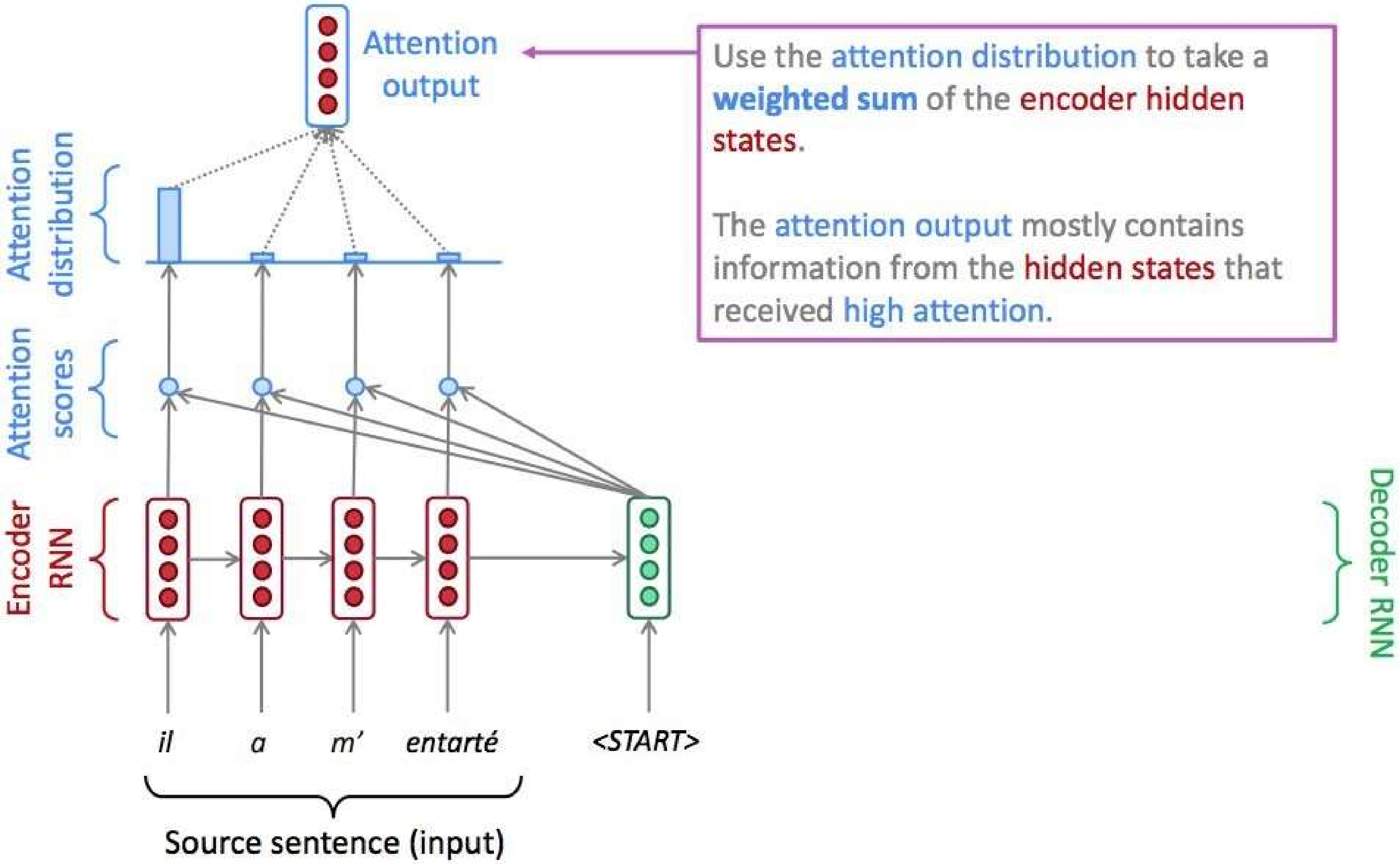
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention

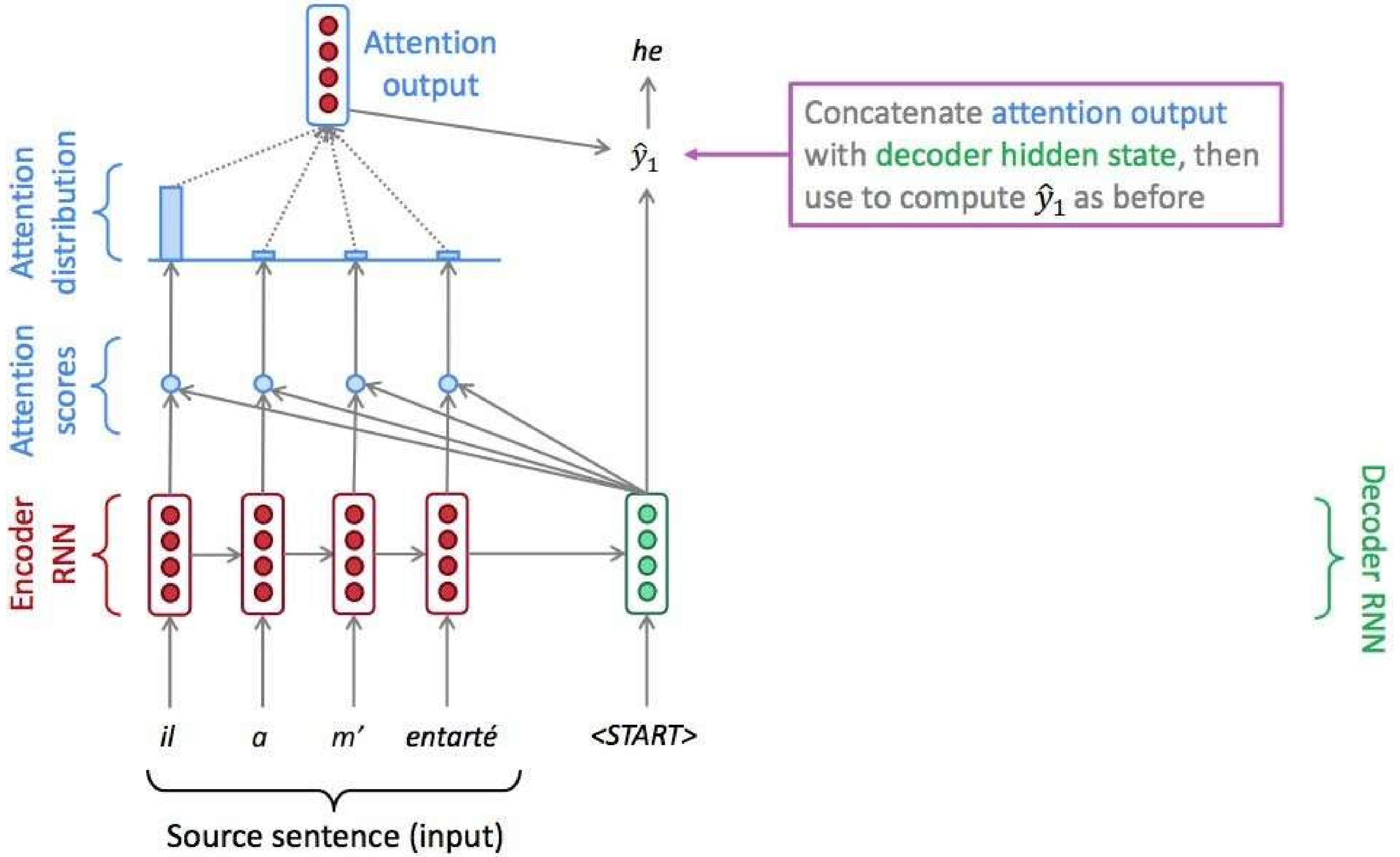


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention.

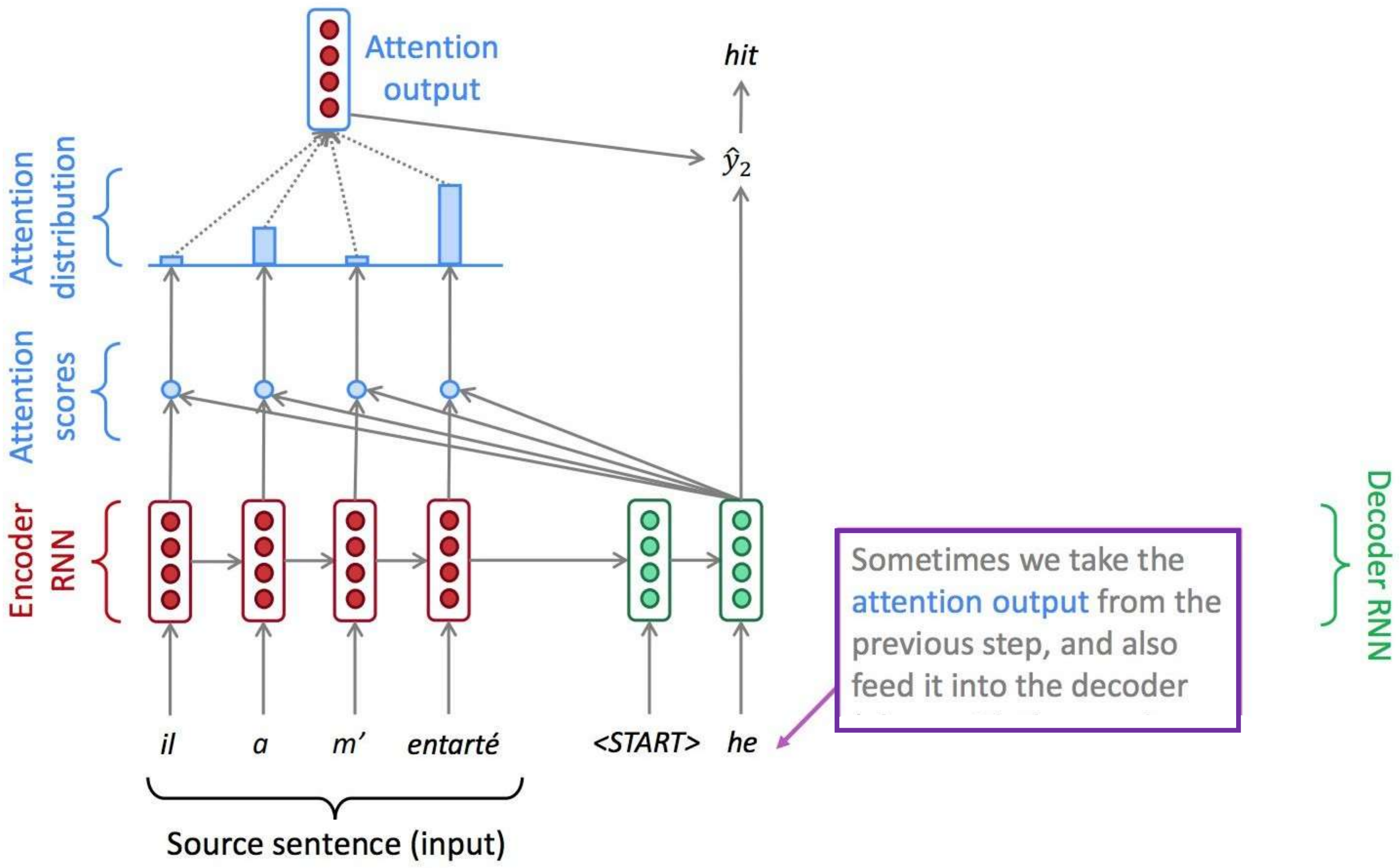
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



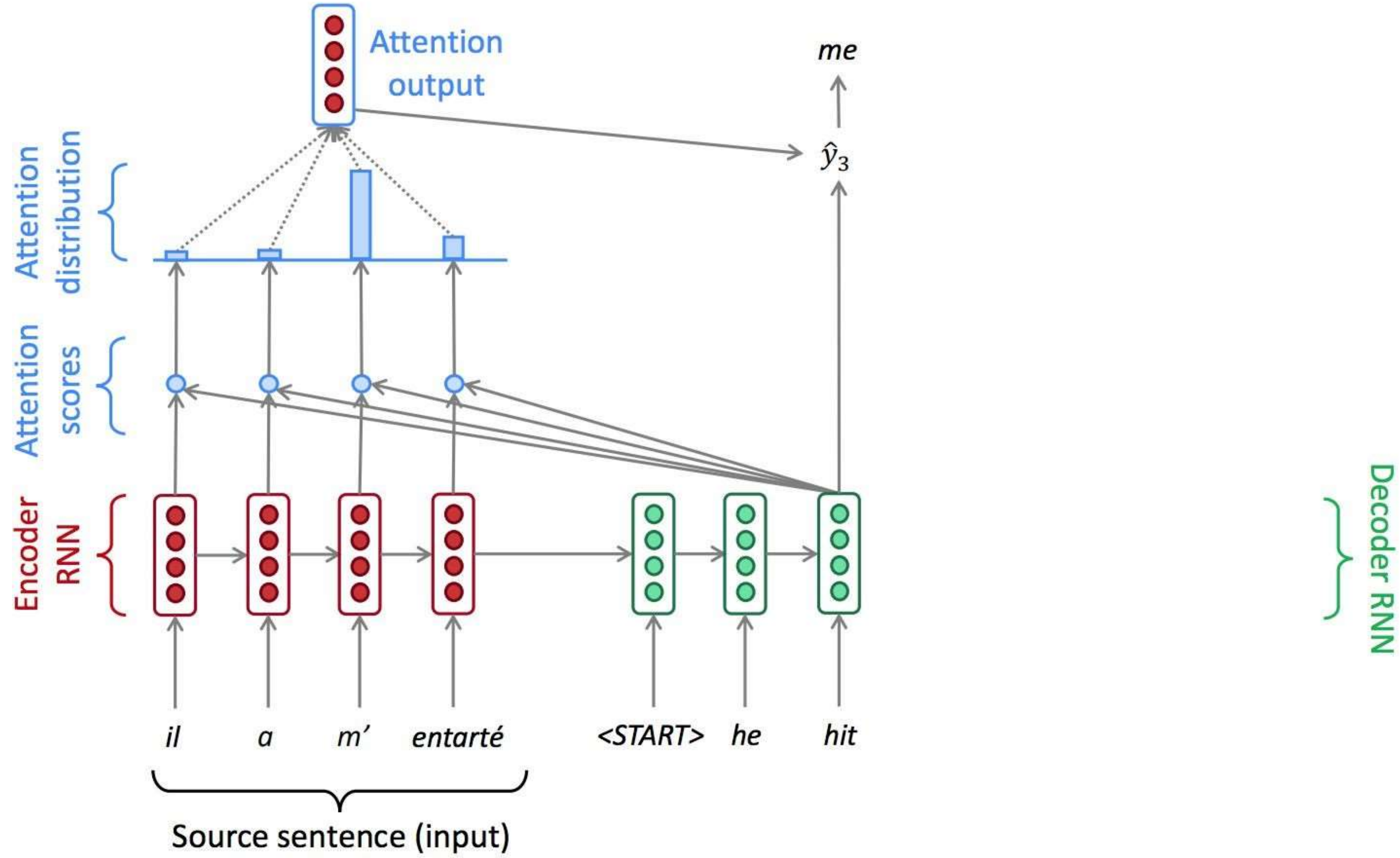
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



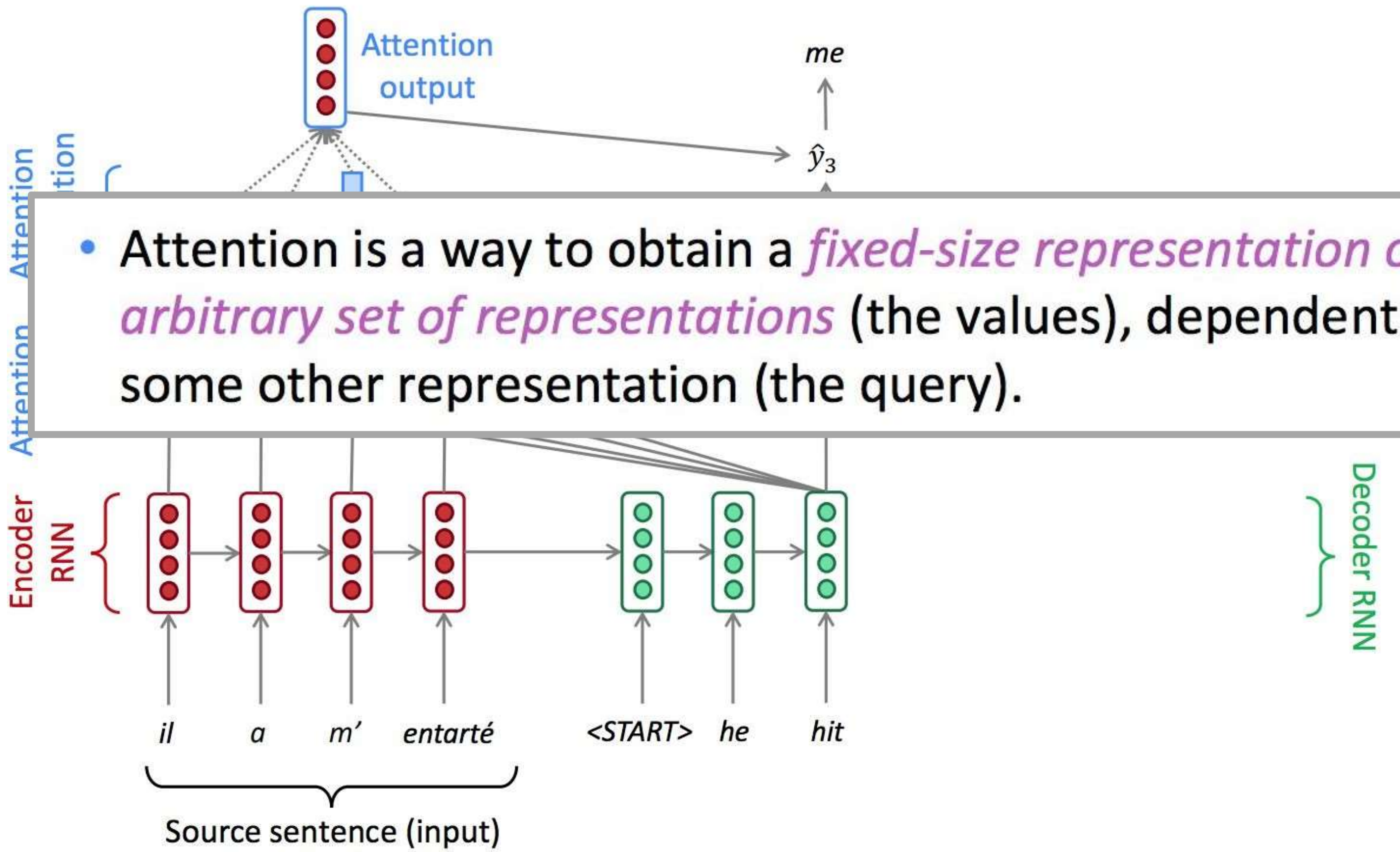
Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



Slide Credit: Abigail See, Stanford CS224n

Machine Translation + Attention



- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

Slide Credit: Abigail See, Stanford CS224n

Attention mechanism

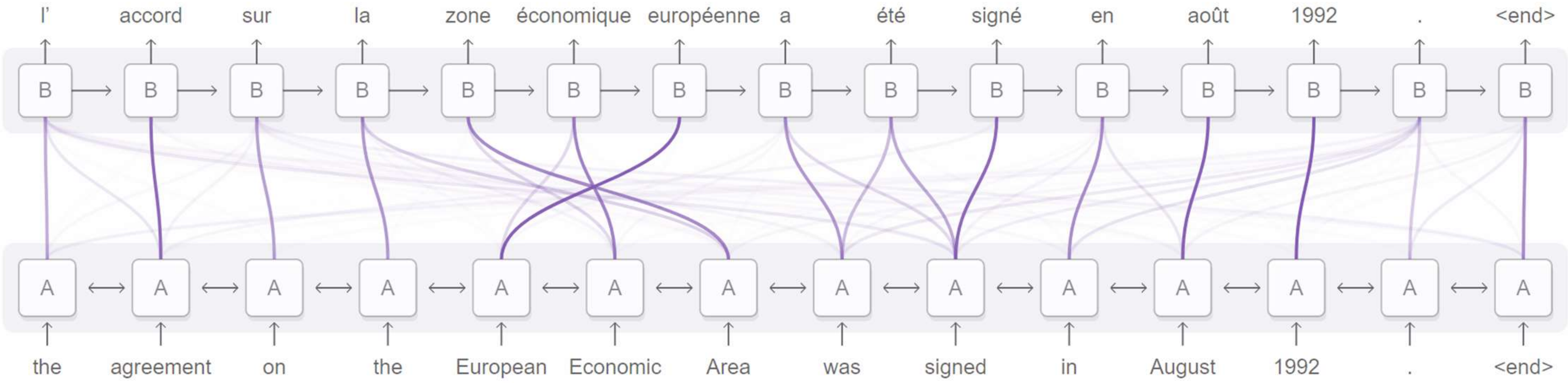
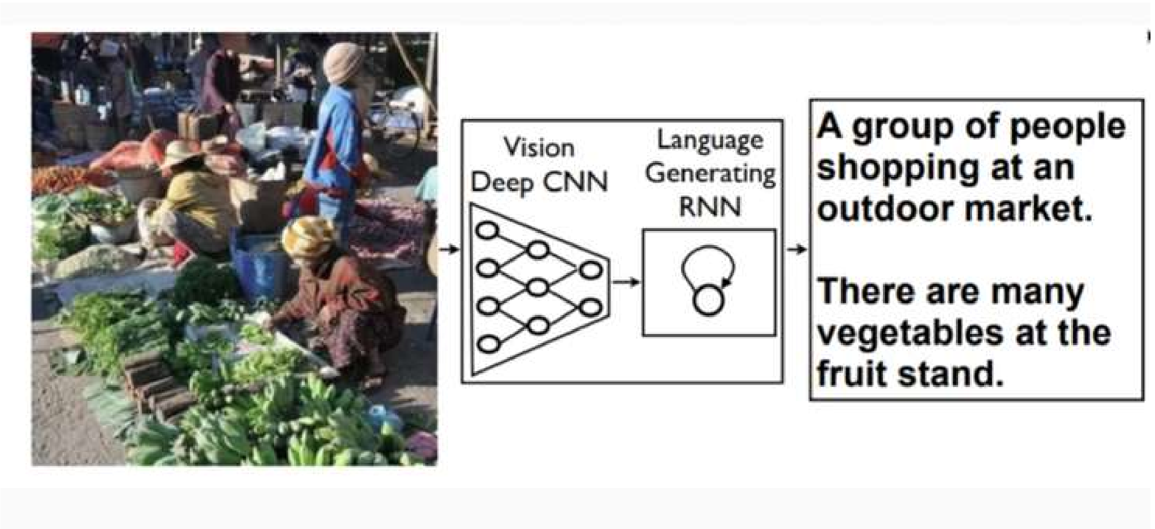


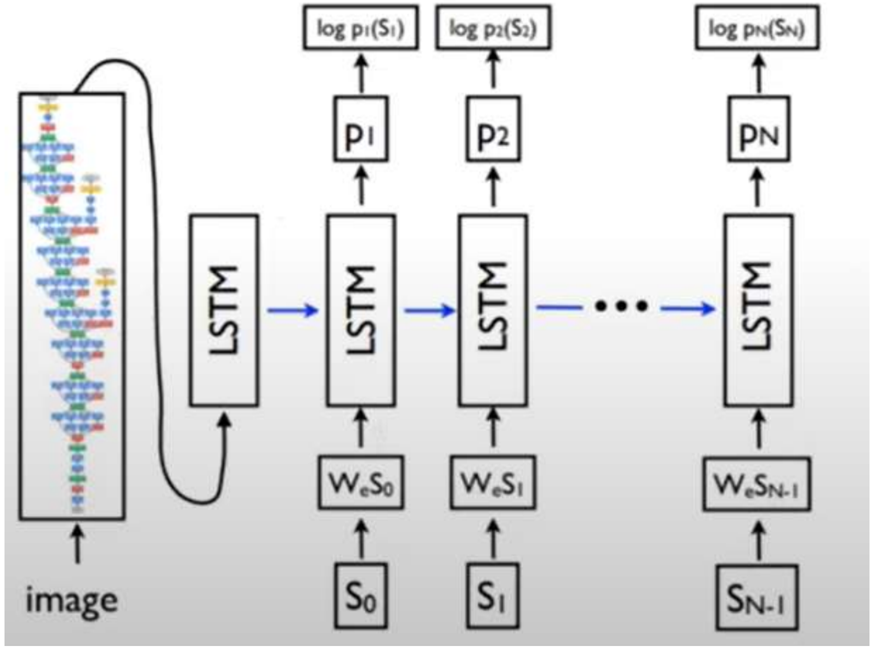
Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

The strength of connections depends on the magnitude of dependence on each word

“Show and tell” (2015)

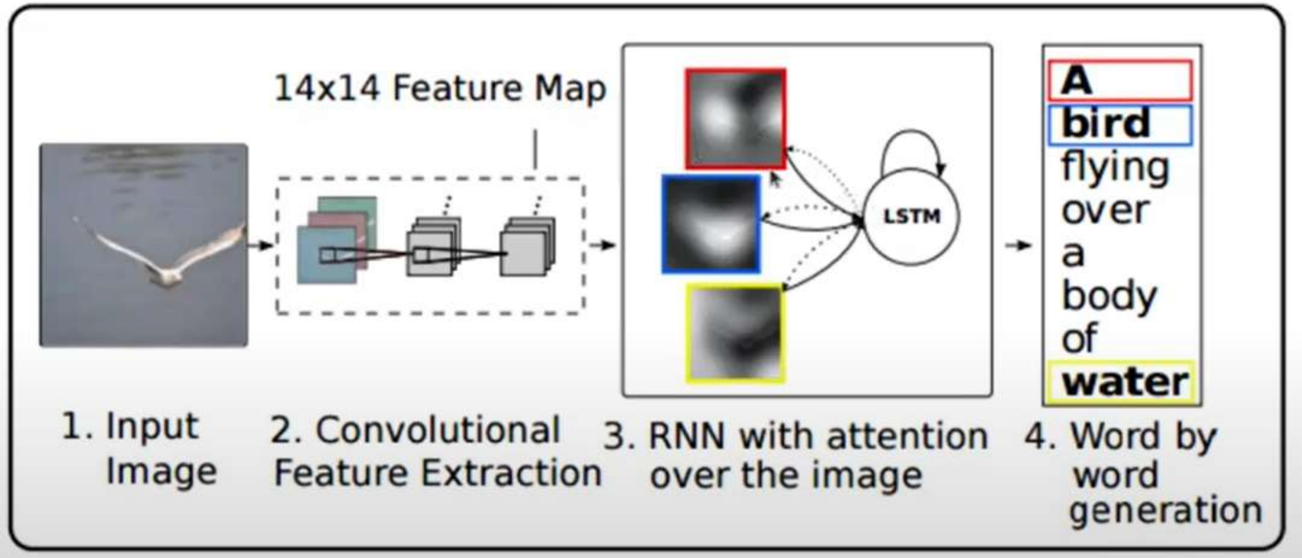
- Image captioning model with attention module



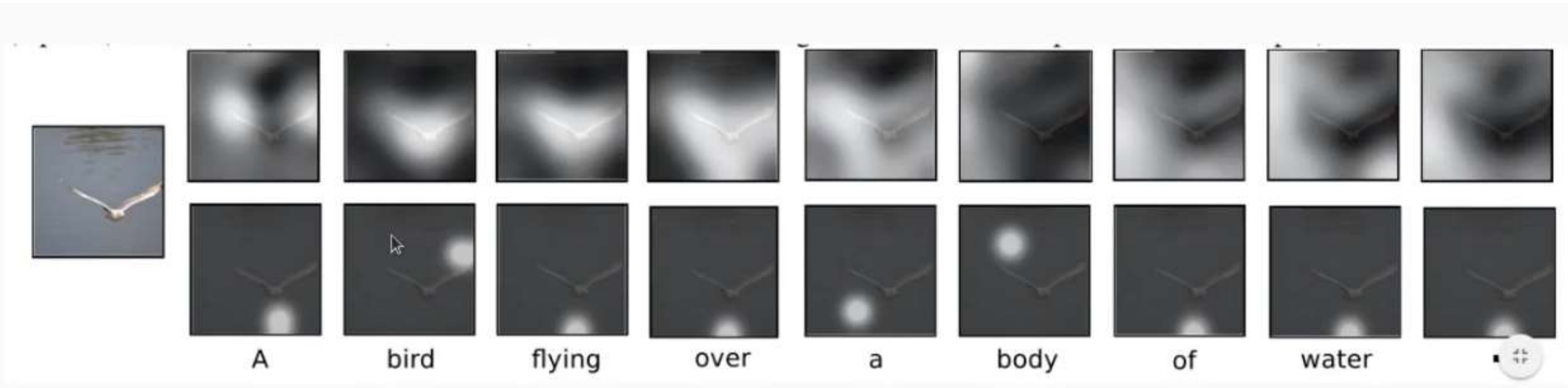


Vinyals et.al, 2015, Show and Tell: A Neural Image Caption Generator

“Show, attend and tell” (2015)

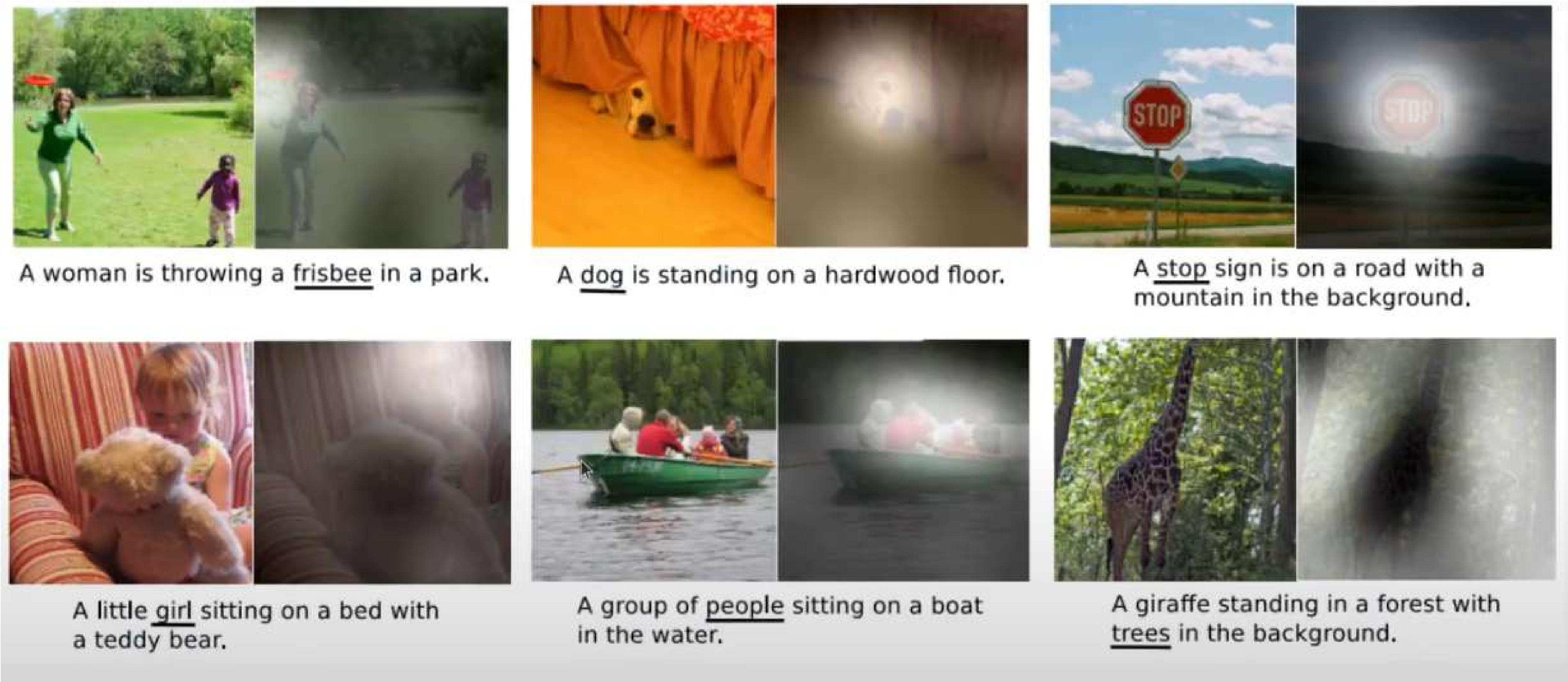


Xu et.al., Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2015



“Show, attend and tell” (2015)

ITMO UNIVERSITY



Xu et.al., Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2015

Advantages of Attention Mechanism

ITMO UNIVERSITY

- Attention significantly improves performance **in many applications**
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on

ITsMOre than a UNIVERSITY

Slide Credit: Abigail See, Stanford CS224n

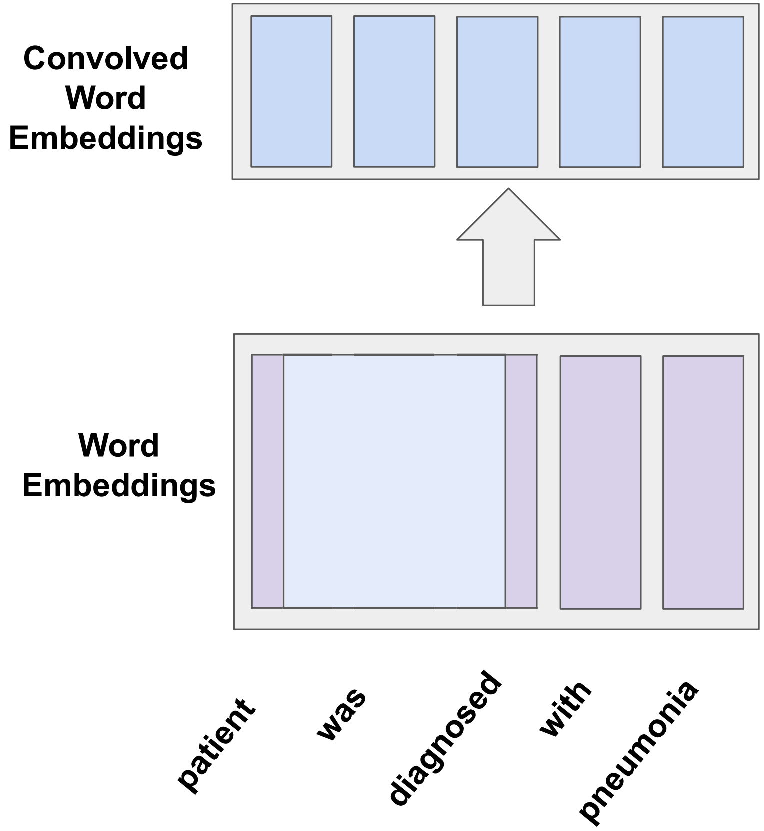
Problems with RNNs – still remains…

- RNNs involve sequential computation - can’t parallelize => time-consuming;
- RNNs “forget” past information;
- No explicit modeling of long and short range dependencies;

Slide Credit: Sarah Wiegreffe

Convolution?

- Trivial to parallelize (per layer);
- Exploits local dependencies; models local context;
- Long-distance dependencies require many layers;



Slide Credit: Sarah Wiegreffe Ashish Vaswani & Anna Huang, Stanford CS224n

Transformers

“Attention is All You Need”

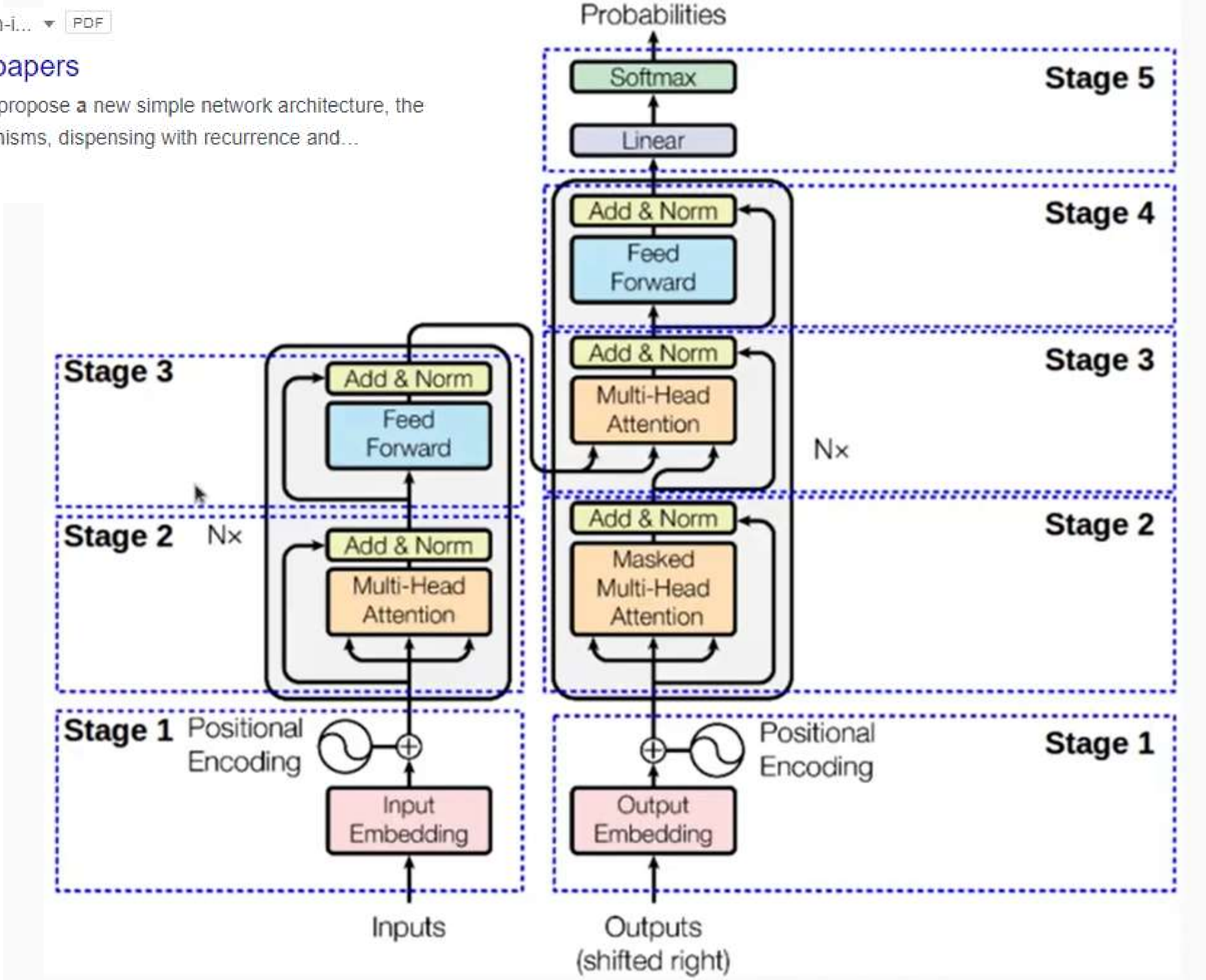
http://papers.neurips.cc › paper › 7181-attention-i... ▾ PDF

Attention is All you Need - NIPS papers

Автор: A Vaswani · Цитируется: 53585 – We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and...

11 страниц

Encoder →



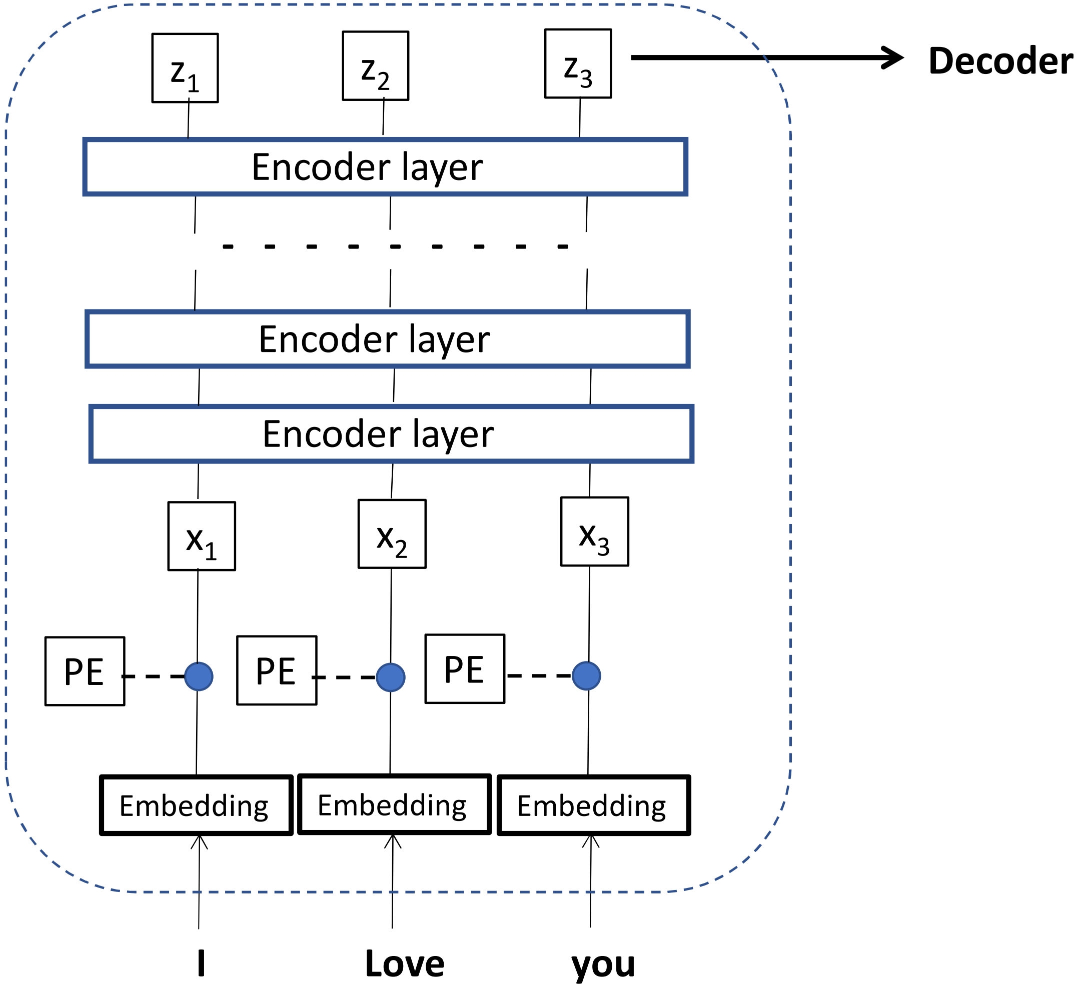
← Decoder

Figure Credit: Vaswani et. al

Basic concepts of Transformer model

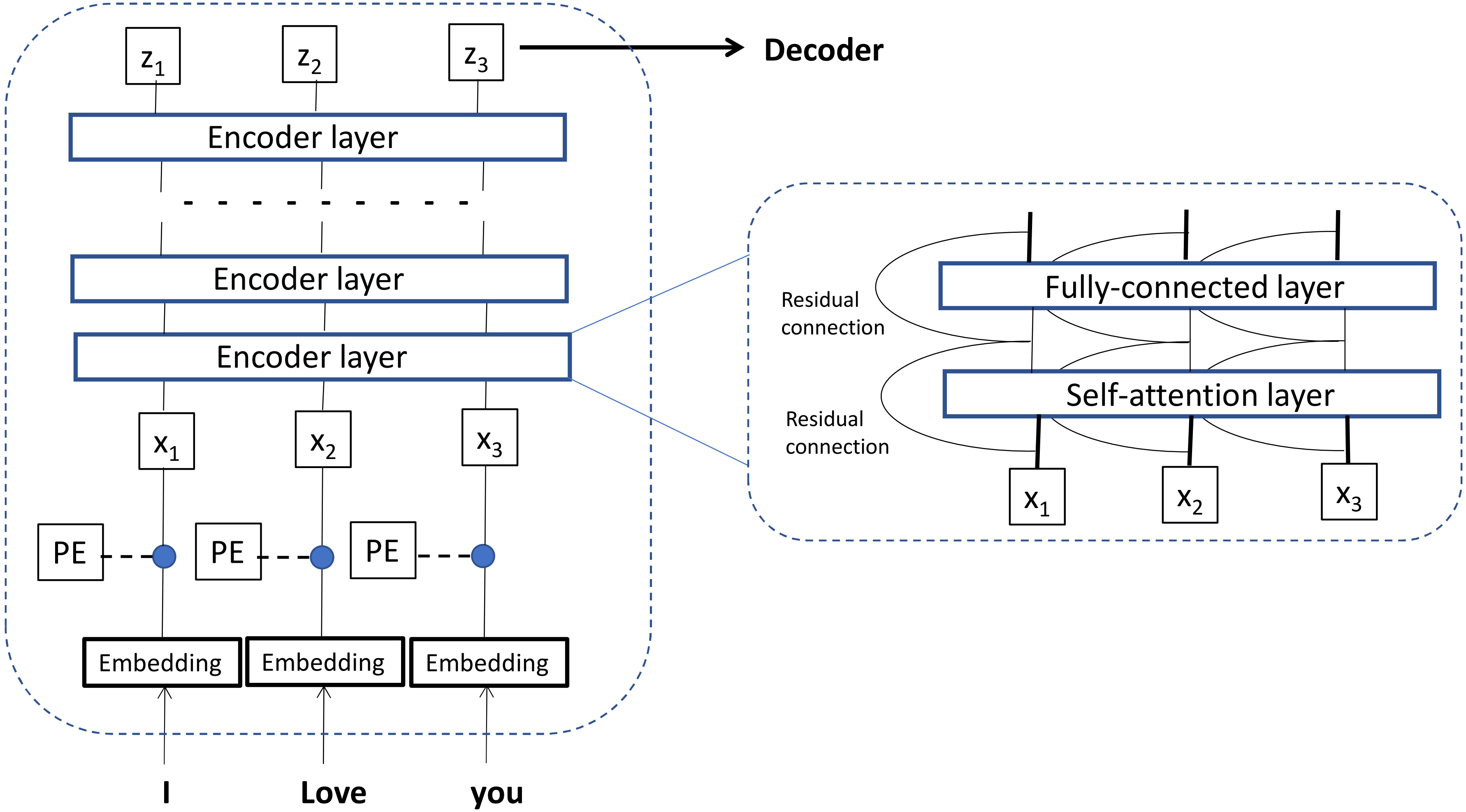
1. *Encoding-decoding architecture;*
2. *Self-attention mechanism: query, key and value;*
3. *Positional Encoding;*
4. *Multi-head Attention;*
5. *Masked Multi-head attention;*
6. *Residual connections*

Encoder



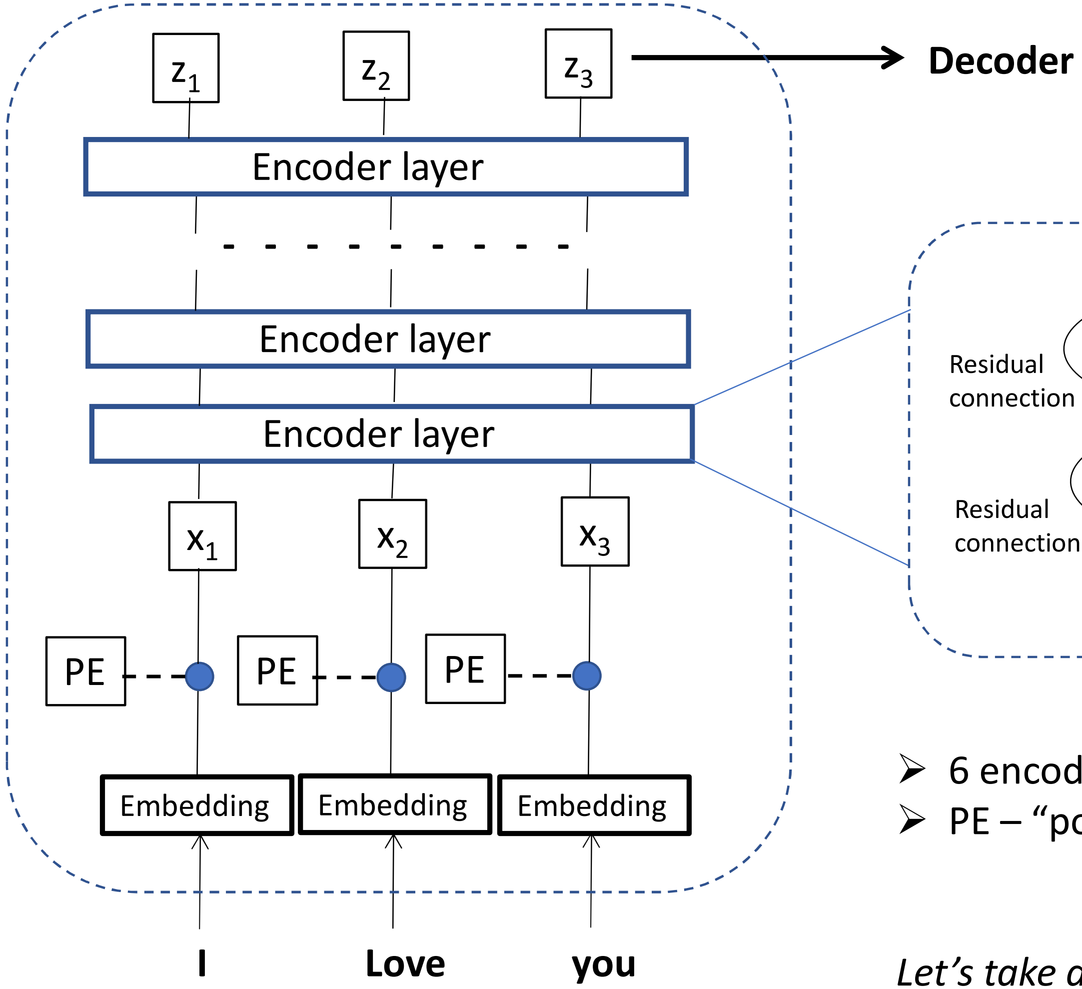
z_1
z_2
z_3
Decoder
Encoder layer
Encoder layer
Encoder layer
x_1
x_2
x_3
PE
PE
PE
Embedding
Embedding
Embedding
I
Love
you

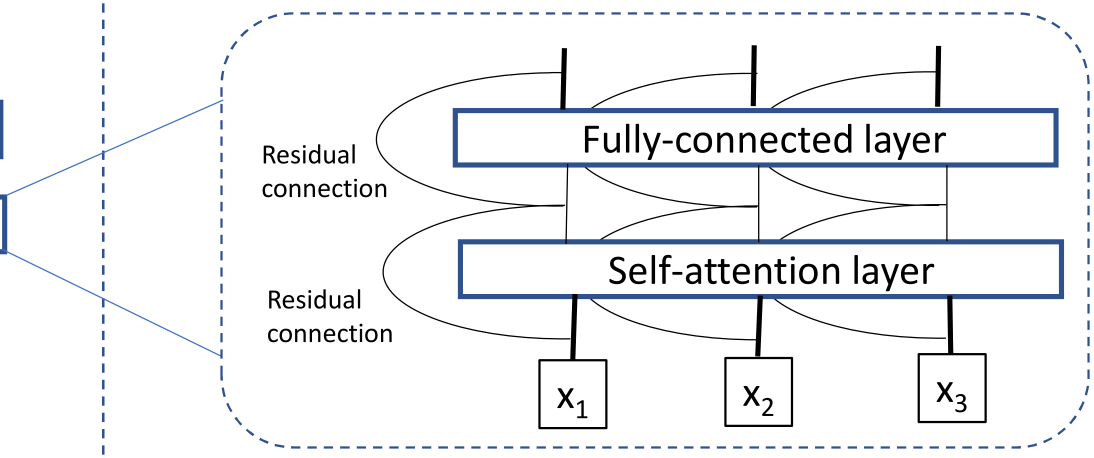
Encoder



z_1
z_2
z_3
Decoder
Encoder layer
Encoder layer
Encoder layer
x_1
x_2
x_3
PE
PE
PE
Embedding
Embedding
Embedding
I
Love
you
Residual connection
Fully-connected layer
Residual connection
Self-attention layer
x_1
x_2
x_3

Encoder

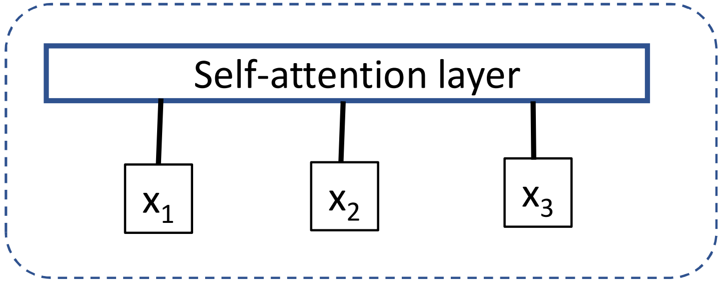




- 6 encoder layers (in initial paper)
- PE – "positional encoding"

Let's take a look into details inside self-attention layer.

Self-attention

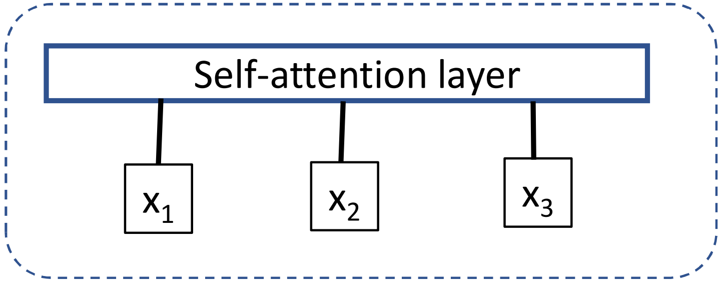


For each input embedding vector:

$$x_i \rightarrow (q_i,\ k_i,\ v_i)$$

Query Key Value

Self-attention

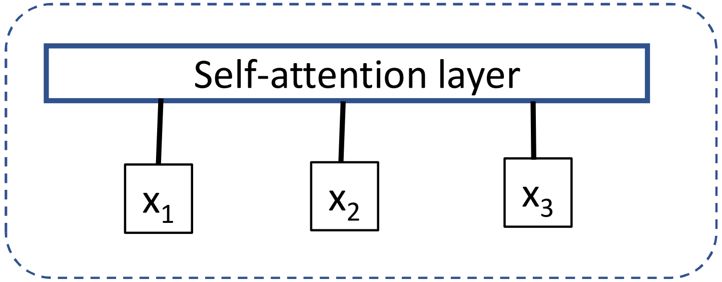


For each input embedding vector:

$$x_i \rightarrow (\underset{\substack{\uparrow \\ \text{Query}}}{q_i},\ \underset{\substack{\uparrow \\ \text{Key}}}{k_i},\ \underset{\substack{\uparrow \\ \text{Value}}}{v_i})$$

$\longrightarrow a_{ij} = (q_i^T \ k_j)$ - how much (in current state) x_i relates to x_j - or "relevance" – how much *j* is relevant to *i*.

Self-attention



For each input embedding vector:

$$x_i \rightarrow (q_i, \; k_i, \; v_i)$$

Query Key Value

$\longrightarrow \; a_{ij} = (q_i^T \; k_j)$ - how much (in current state) x_i relates to x_j - or "relevance" – how much *j* is relevant to *i*.

We go through all the keys of the input vector with query q_i:

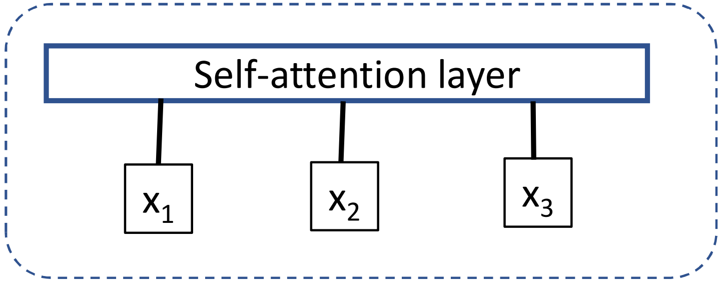
$$x_n \rightarrow (q_n, \; k_n, \; v_n) \;\rightarrow\; (q_i^T \; k_n)$$

$$\cdots$$

$$x_2 \rightarrow (q_2, \; k_2, \; v_2) \;\rightarrow\; (q_i^T \; k_2)$$

$$x_1 \rightarrow (q_1, \; k_1, \; v_1) \;\rightarrow\; (q_i^T \; k_1)$$

Self-attention



For each input embedding vector:

$$x_i \rightarrow (\underset{\text{Query}}{\underset{\uparrow}{q_i}},\ \underset{\text{Key}}{\underset{\uparrow}{k_i}},\ \underset{\text{Value}}{\underset{\uparrow}{v_i}})$$

$\longrightarrow$ $a_{ij} = (q_i^T\ k_j)$ - how much (in current state) x_i relates to x_j - or "relevance" – how much *j* is relevant to *i*.

We go through all the keys of the input vector with query q_i:

$$x_n \rightarrow (q_n,\ k_n,\ v_n)\ \rightarrow\ (q_i^T\ k_n)$$

$$\cdots$$

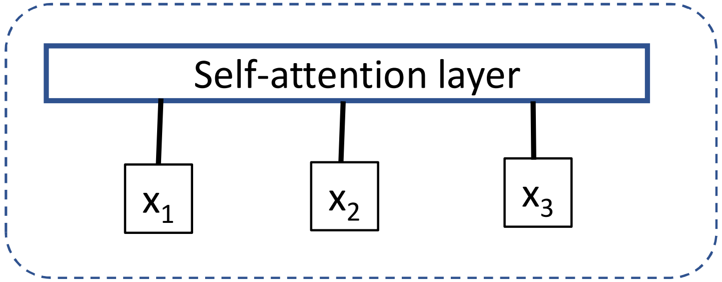
$$x_2 \rightarrow (q_2,\ k_2,\ v_2)\ \rightarrow\ (q_i^T\ k_2)$$

$$x_1 \rightarrow (q_1,\ k_1,\ v_1)\ \rightarrow\ (q_i^T\ k_1)$$

Evaluating a final weighted and normalized output of the layer

$$x_i \rightarrow (q_i,\ k_i,\ v_i)\ \rightarrow y_i = \sum_{j=1}^{n} Softmax(\frac{1}{\sqrt{d}}\ q_i^T\ k_j)\ v_j$$

Self-attention



For each input embedding vector:

$$x_i \rightarrow (q_i,\ k_i,\ v_i)$$

Query Key Value

$$\longrightarrow a_{ij} = (q_i^T\ k_j)$$

- how much (in current state) x_i relates to x_j - or "relevance" – how much *j* is relevant to *i*.

We go through all the keys of the input vector with query q_i:

$$x_n \rightarrow (q_n,\ k_n,\ v_n) \rightarrow (q_i^T\ k_n)$$

$$\cdots$$

$$x_2 \rightarrow (q_2,\ k_2,\ v_2) \rightarrow (q_i^T\ k_2)$$

$$x_1 \rightarrow (q_1,\ k_1,\ v_1) \rightarrow (q_i^T\ k_1)$$

Evaluating a final weighted and normalized output of the layer

$$x_i \rightarrow (q_i,\ k_i,\ v_i) \rightarrow y_i = \sum_{j=1}^{n} Softmax\left(\frac{1}{\sqrt{d}}\, q_i^T\ k_j\right) v_j$$

Q, K, V matrices and multi-head attention

$$x_i \in R^d, q_i \in R^m, \; k_i \in R^m, \; v_i \in R^l$$

$$W_Q^{(1)} X \to Q^{(1)}$$

$$W_K^{(1)} X \to K^{(1)} \quad \longrightarrow \quad Y^{(1)} = Softmax\left(\frac{1}{\sqrt{m}} QK^T\right) V$$

$$W_V^{(1)} X \to V^{(1)}$$

X

Q, K, V matrices and multi-head attention

$$x_i \in R^d, q_i \in R^m, \;\; k_i \in R^m, \;\; v_i \in R^l$$

$$W_Q^{(1)} X \to Q^{(1)}$$

$$W_K^{(1)} X \to K^{(1)} \quad \longrightarrow \quad Y^{(1)} = Softmax(\frac{1}{\sqrt{m}} QK^T)\, V$$

$$W_V^{(1)} X \to V^{(1)}$$

$$W_Q^{(2)} X \to Q^{(2)}$$

$$W_K^{(2)} X \to K^{(2)} \quad \longrightarrow \quad Y^{(2)} = Softmax(\frac{1}{\sqrt{m}} QK^T)\, V$$

$$W_V^{(2)} X \to V^{(2)}$$

$$X$$

Q, K, V matrices and multi-head attention

$$x_i \in R^d, q_i \in R^m, \ k_i \in R^m, \ v_i \in R^l$$

Apply the same operations S times in different branches – “heads”. Each heads learn some specific information from embedding

X

(1)
$$W_Q^{(1)}X \to Q^{(1)}$$
$$W_K^{(1)}X \to K^{(1)} \longrightarrow Y^{(1)} = Softmax(\frac{1}{\sqrt{m}}QK^T)\,V$$
$$W_V^{(1)}X \to V^{(1)}$$

(2)
$$W_Q^{(2)}X \to Q^{(2)}$$
$$W_K^{(2)}X \to K^{(2)} \longrightarrow Y^{(2)} = Softmax(\frac{1}{\sqrt{m}}QK^T)\,V$$
$$W_V^{(2)}X \to V^{(2)}$$

. . .

(S)
$$W_Q^{(n)}X \to Q^{(n)}$$
$$W_K^{(n)}X \to K^{(n)} \longrightarrow Y^{(n)} = Softmax(\frac{1}{\sqrt{m}}QK^T)\,V$$
$$W_V^{(n)}X \to V^{(n)}$$

S branches

Concatenation

Q, K, V matrices and multi-head attention

ITMO UNIVERSITY

$$x_i \in R^d, q_i \in R^m, \ k_i \in R^m, \ v_i \in R^l$$

Apply the same operations S times in different branches – “heads”. Each heads learn some specific information from embedding

$$X \left\{ \begin{array}{ll}
(1) & \begin{array}{l} W_Q^{(1)}X \to Q^{(1)} \\ W_K^{(1)}X \to K^{(1)} \\ W_V^{(1)}X \to V^{(1)} \end{array} \longrightarrow Y^{(1)} = Softmax(\frac{1}{\sqrt{m}}QK^T)\,V \\
(2) & \begin{array}{l} W_Q^{(2)}X \to Q^{(2)} \\ W_K^{(2)}X \to K^{(2)} \\ W_V^{(2)}X \to V^{(2)} \end{array} \longrightarrow Y^{(2)} = Softmax(\frac{1}{\sqrt{m}}QK^T)\,V \\
& \cdots \\
(S) & \begin{array}{l} W_Q^{(n)}X \to Q^{(n)} \\ W_K^{(n)}X \to K^{(n)} \\ W_V^{(n)}X \to V^{(n)} \end{array} \longrightarrow Y^{(n)} = Softmax(\frac{1}{\sqrt{m}}QK^T)\,V
\end{array} \right\} \begin{pmatrix} Y^{(1)} \\ Y^{(2)} \\ \cdots \\ Y^{(S)} \end{pmatrix} \times W_O = Z_O$$

S branches

Concatenation

Final output of the Multi-head module

ITsMOre than a UNIVERSITY

Multi-head Attention

ITMO UNIVERSITY

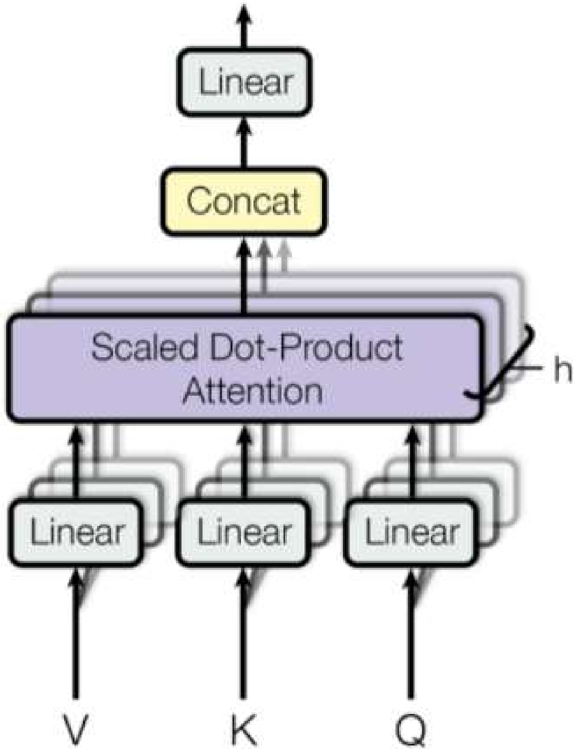
➢ **Multi-head Attention** is a module for attention mechanisms which runs through an attention mechanism **several times in parallel**. (in 1st paper – 8 attention heads and 6 encoder/decoder layers);

➢ The independent attention outputs are then concatenated and linearly transformed into the expected dimension.

➢ Intuitively, multiple attention heads allows for attending to parts of the sequence differently (e.g. longer-term dependencies versus shorter-term dependencies).

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Above ***W*** are all learnable parameter matrices.



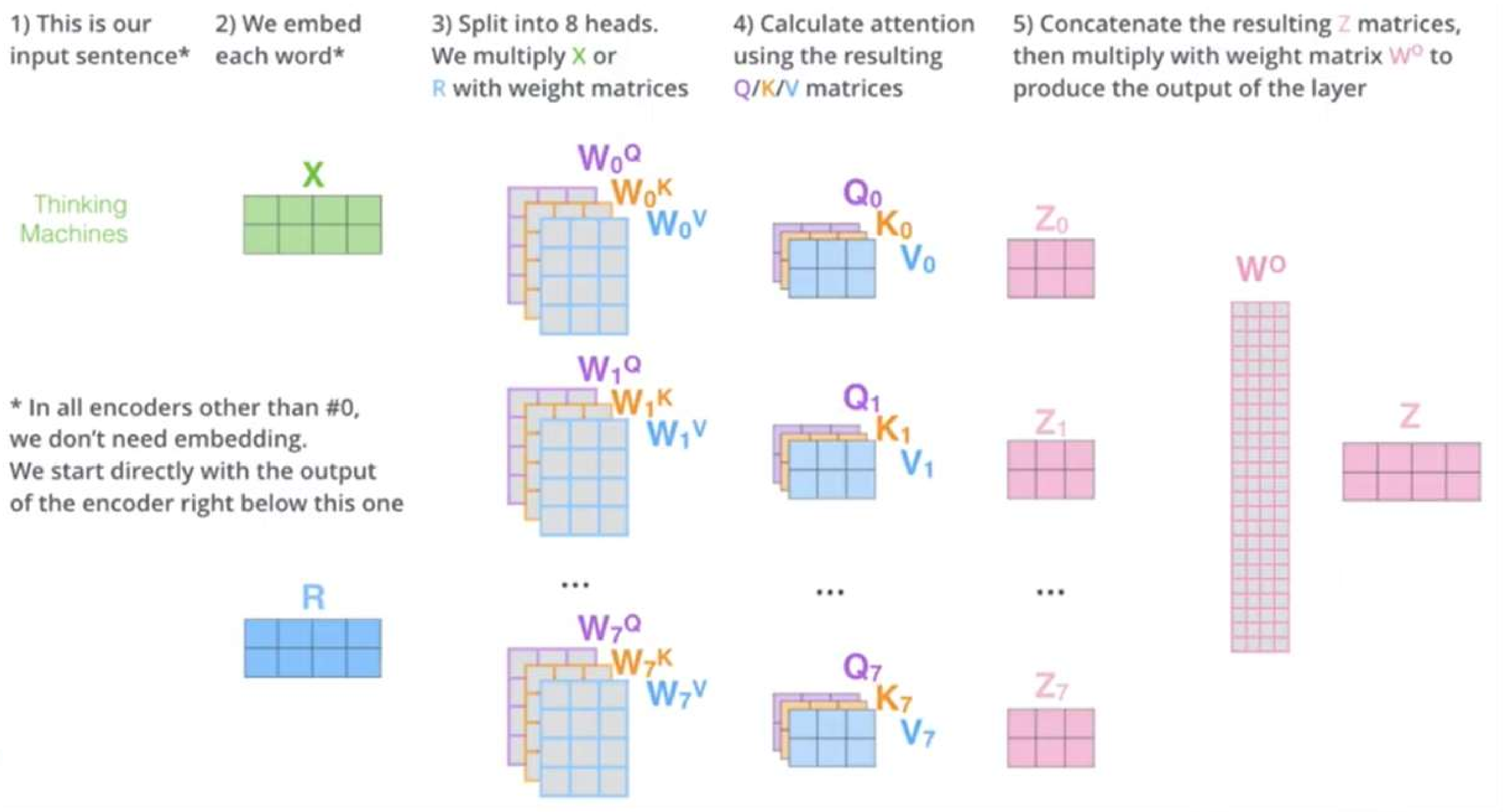
Display attention from different layers and heads:
https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

Vaswani et.al., 2017

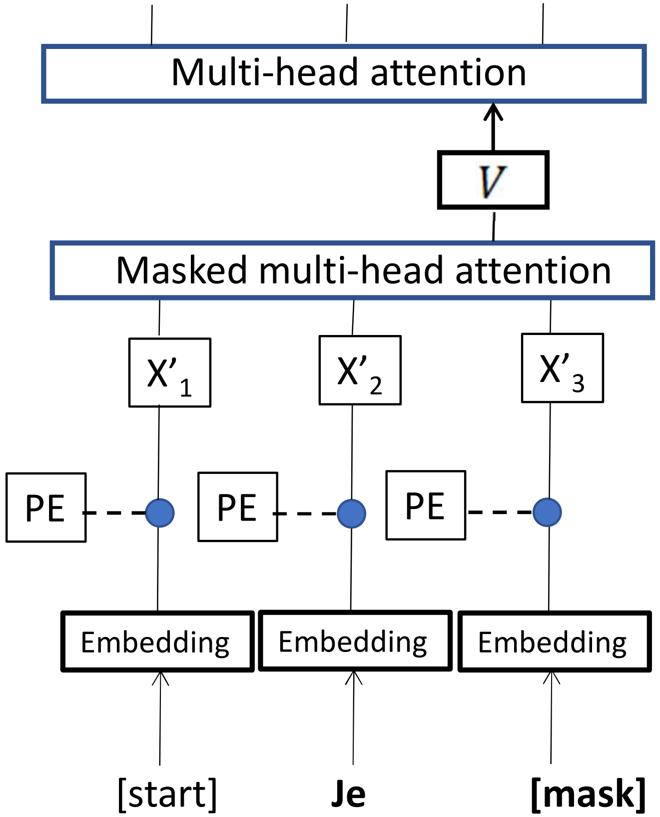
IT's MOre than a UNIVERSITY

Encoder: summary

ITMO UNIVERSITY



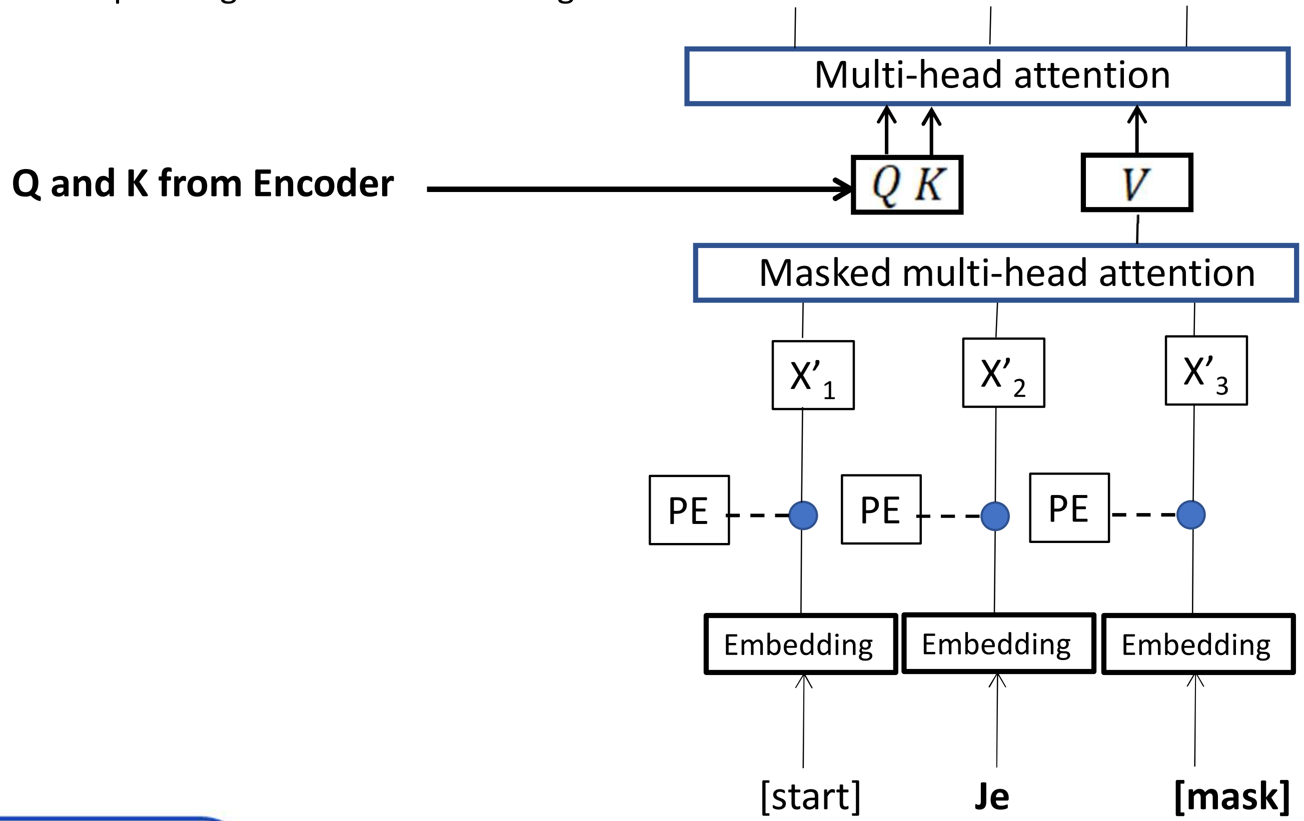
Decoder



Multi-head attention
V
Masked multi-head attention
X'1
X'2
X'3
PE
PE
PE
Embedding
Embedding
Embedding
[start]
Je
[mask]

Decoder

Decoder takes **Q** and **K** from encoder, and **V** from corresponding candidate embedding.



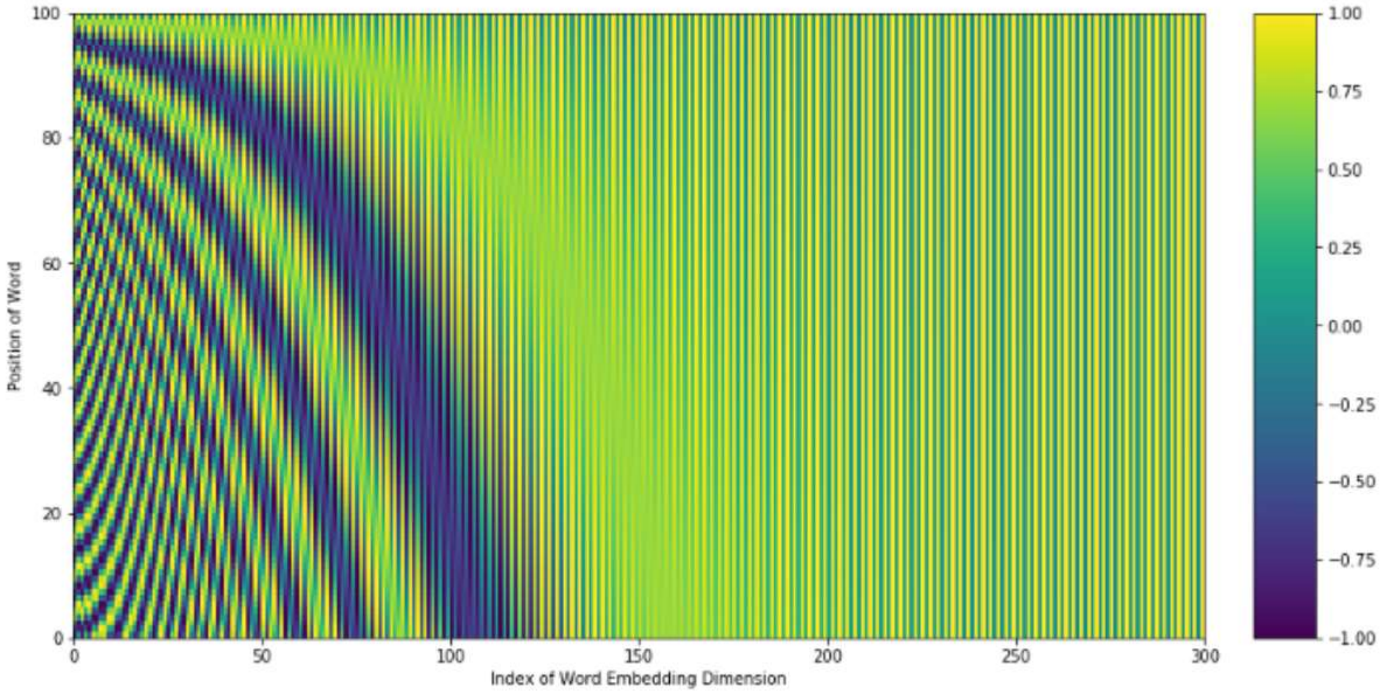
Positional Encoding

Each word in the input sequence is assigned a vector:

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

i – index of the dimensional index of word embedding;
pos – position of the current word in the sequence;
d_model – word embedding dimension;

In order to capture positional information, each element of the positional embedding varies according to a word's position and the index of the element within the dimension of the word embedding *d*. This is achieved by varying frequencies.



That's how transformer learns positions of each word.

PE is added to initial embedding of the word: $Z^w_{new} = Z^w + \mathbf{PE}$

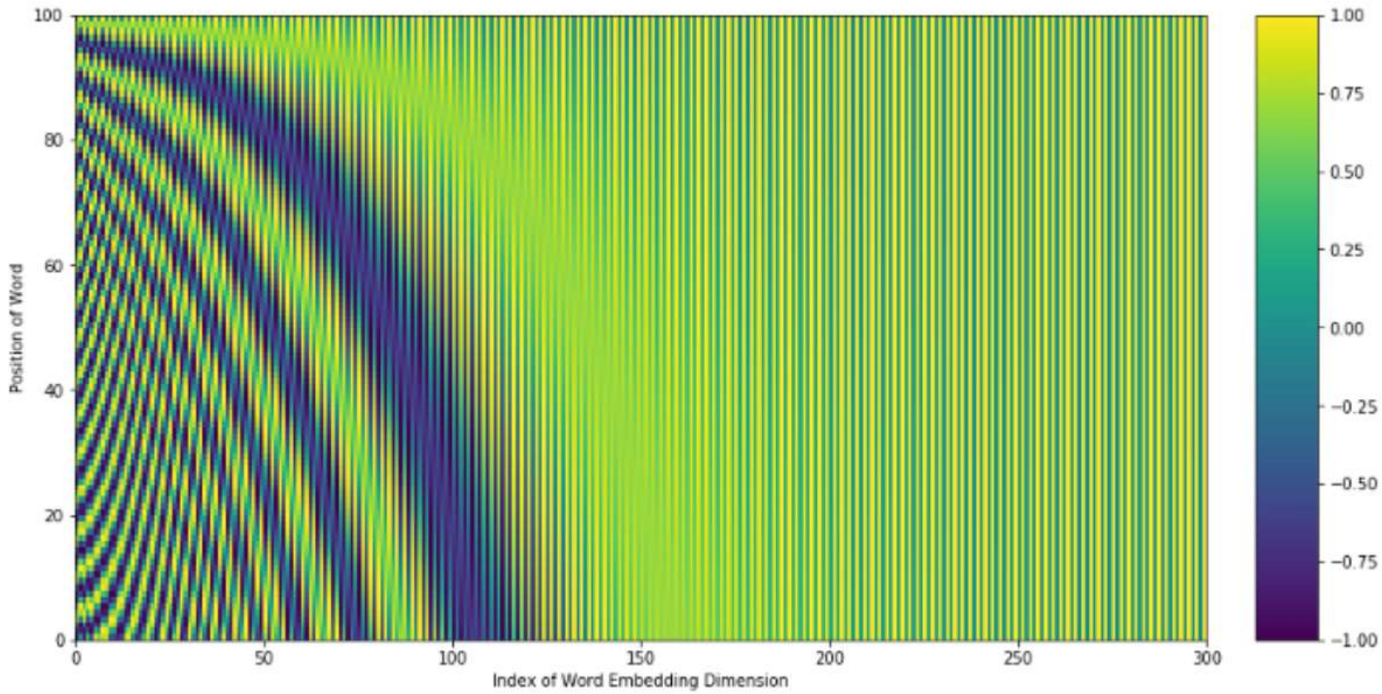
Positional Encoding

Each word in the input sequence is assigned a vector:

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

i – index of the dimensional index of word embedding;
pos – position of the current word in the sequence;
d_model – word embedding dimension;

In order to capture positional information, each element of the positional embedding varies according to a word's position and the index of the element within the dimension of the word embedding *d*. This is achieved by varying frequencies.



That's how transformer learns positions of each word.

PE is added to initial embedding of the word: $\mathbf{Z}^w_{new} = Z^w + \mathbf{PE}$

For the case of 4-dimensional embedding:

$$\mathbf{Z^w_{new}} = Z^w + [\sin(pos/10000^{\frac{2*0}{4}}), \cos(pos/10000^{\frac{2*0}{4}}), \sin(pos/10000^{\frac{2*1}{4}}), \cos(pos/10000^{\frac{2*1}{4}})]$$

Attention is cheap

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O(n^2 \cdot d)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$

n – the length of the input vector;
d – the length of latent representation (embedding);
r - radius of restricted-length attention.

- Cheaper when n < d
- No long-range dependencies;
- No sequential operations;

… and IT WORKS!

Table Credit: Vaswani et. al 2017

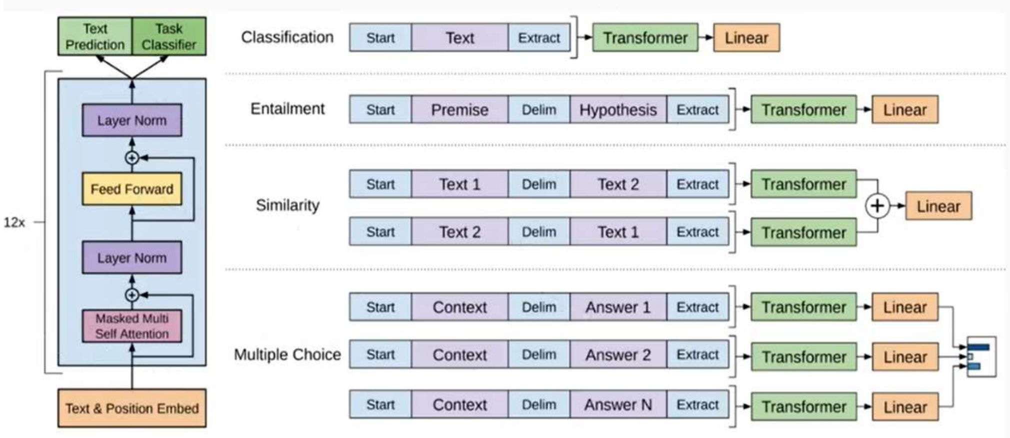
Transformer-based models for text problems

ITMO UNIVERSITY

Transformers
(Encoder + Decoder)

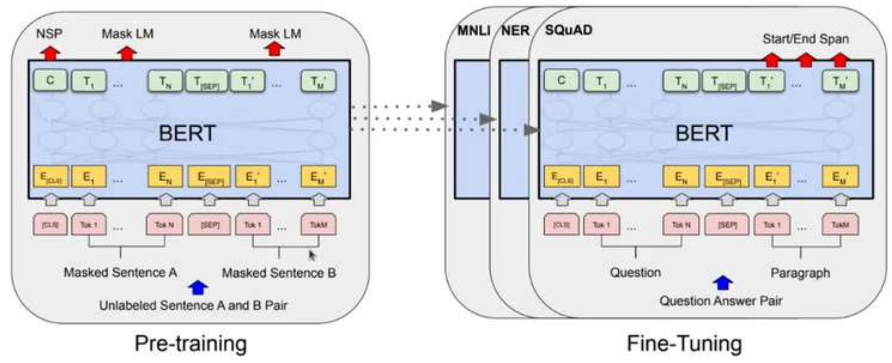
GPT
(Generative Pretrained Transformer) models
<u>Decoder</u>
-language models-

BERT
(Bidirectional Encoder Representations from Transformers) models
<u>Encoder</u>
- text-related tasks -



Examples of GPT models

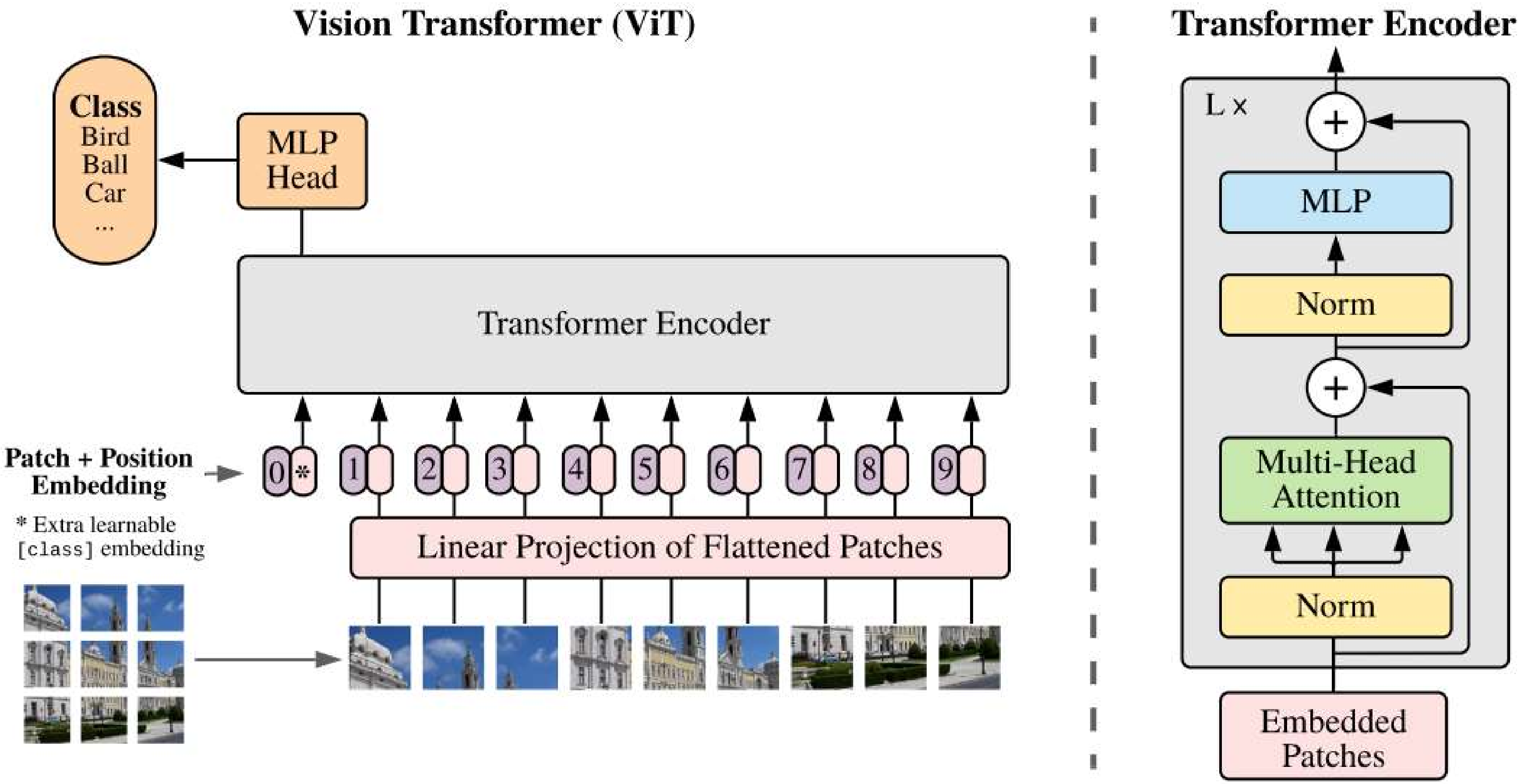
Masking random words (N-gramms/sentences) inside input sentence and try to predict it (fill in the gaps)



Vision Transformers

ViT model

It was published in early December 2020 (implementation).



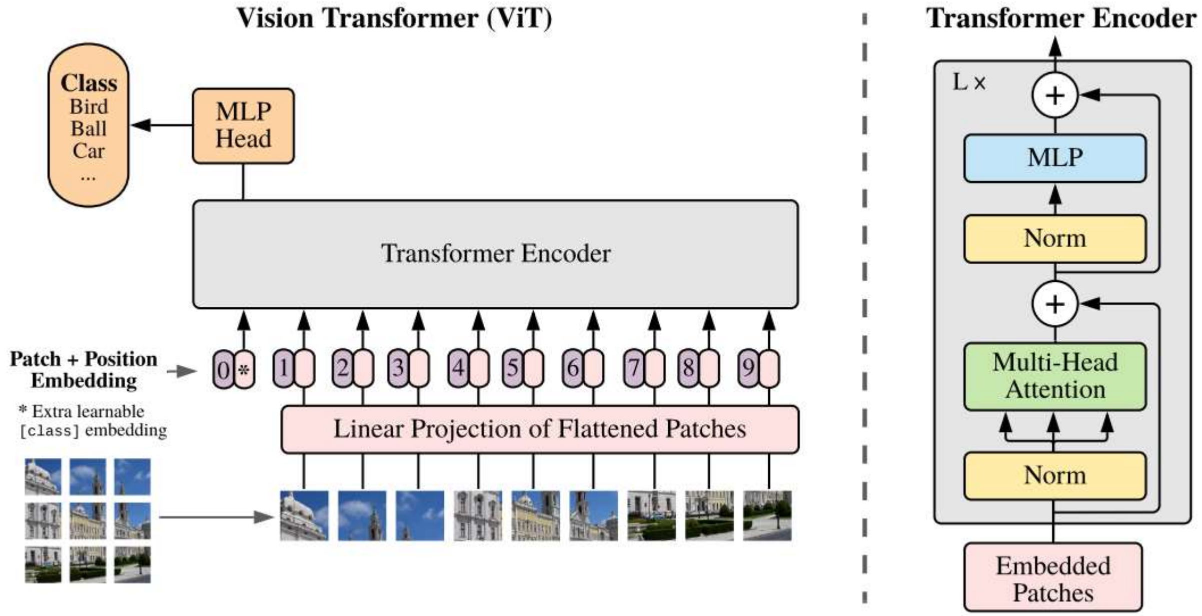
Dosovitsky et.al., 2020

ViT model

It was published in early December 2020 (implementation).
Dosovitsky et.al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2020

- ➢ The picture is divided into mini-sections 16 * 16.
- ➢ Each section is fed into the transformer as a "word", supplemented by a positional encoder.
- ➢ And, suddenly, it all worked!

- ➢ **Deit** – improvement of **ViT** from Facebook - greatly simplified training and inference.
- ➢ On large datasets, this approach still holds the first places in almost all classifications https://paperswithcode.com/paper/going-deeper-with-image-transformers
- ➢ But, with a dataset of ~ 2-3 thousand pictures, doesn't work very well – classic ResNet more stable and better.



Dosovitsky et.al., 2020

Key advantages of transformers

- Easier to train, more efficient;
- Transfer learning works (pre-trained models can be fine-tuning for new tasks)
- Can be trained on unsupervised texts (all the world text data is now valid training data)

<u>LSTM still good when:</u>

- Sequence length long or infinite (Transformers are $O(N^2)$)
- Need real-time control for robotics
- Can't pretrain on a large corpus.

Links to additional resources & tutorials

ITMO UNIVERSITY

https://jalammar.github.io/illustrated-transformer/
https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/
https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0
https://arxiv.org/pdf/1706.03762.pdf
https://arxiv.org/pdf/2010.11929.pdf

IT's MOre than a UNIVERSITY

ITMO UNIVERSITY

Saint Petersburg, Russia

Thank you!