

Machine Translation. Attention. Transformers

MIPT

4.03.2021

Anton Emelianov, Alena Fenogenova.

Today

- Machine Translation
- Seq2Seq
- Attention
- Early attention models
- Transformer
 - High-level
 - Deeper



Machine translation

How are you today?  Wie geht es dir heute?

$x=(x_1,x_2,\dots,x_{(T_x)})$

$y=(y_1,y_2,\dots,y_{(T_y)})$

Translation task => finding the target sequence that is the most probable given the input;

the target sequence that maximizes the conditional probability: $y^* = \arg \max_y p(y|x)$

Machine translation:

- between natural languages
- between programming languages
- any sequences of tokens

by **Machine translation** we will mean **any general sequence-to-sequence task**

Machine translation

Machine translation systems learn a function: $p(y|x, \theta)$

We try to find target sequence that maximizes the conditional probability:

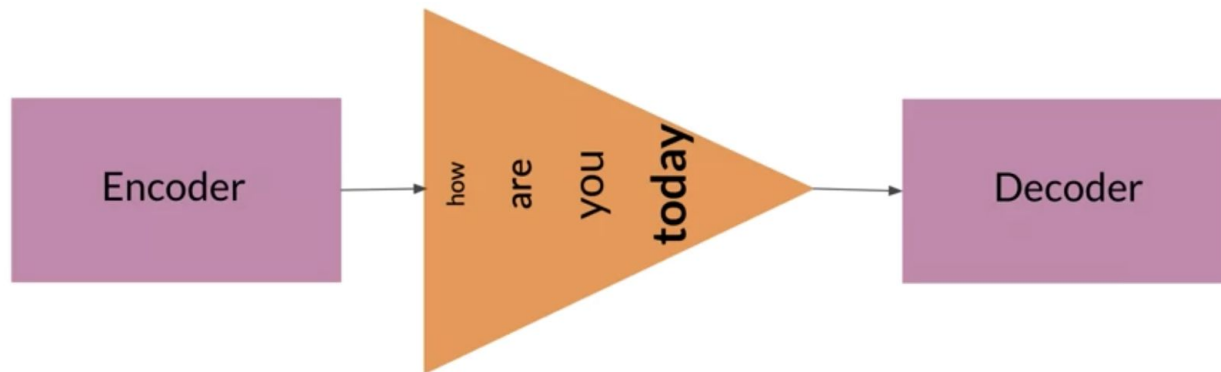
$$\hat{y} = \operatorname{argmax}_y p(y|x, \theta)$$

where θ – model parameters that determine probability distribution

What to do?

1. modeling - how does the model for $p(y|x, \theta)$ look like?
2. learning - how to find the parameters θ ?
3. inference - how to find the best y ?

Machine translation



Seq2Seq

Introduced by Google in 2014

Encoder - reads source sequence and produces its representation;

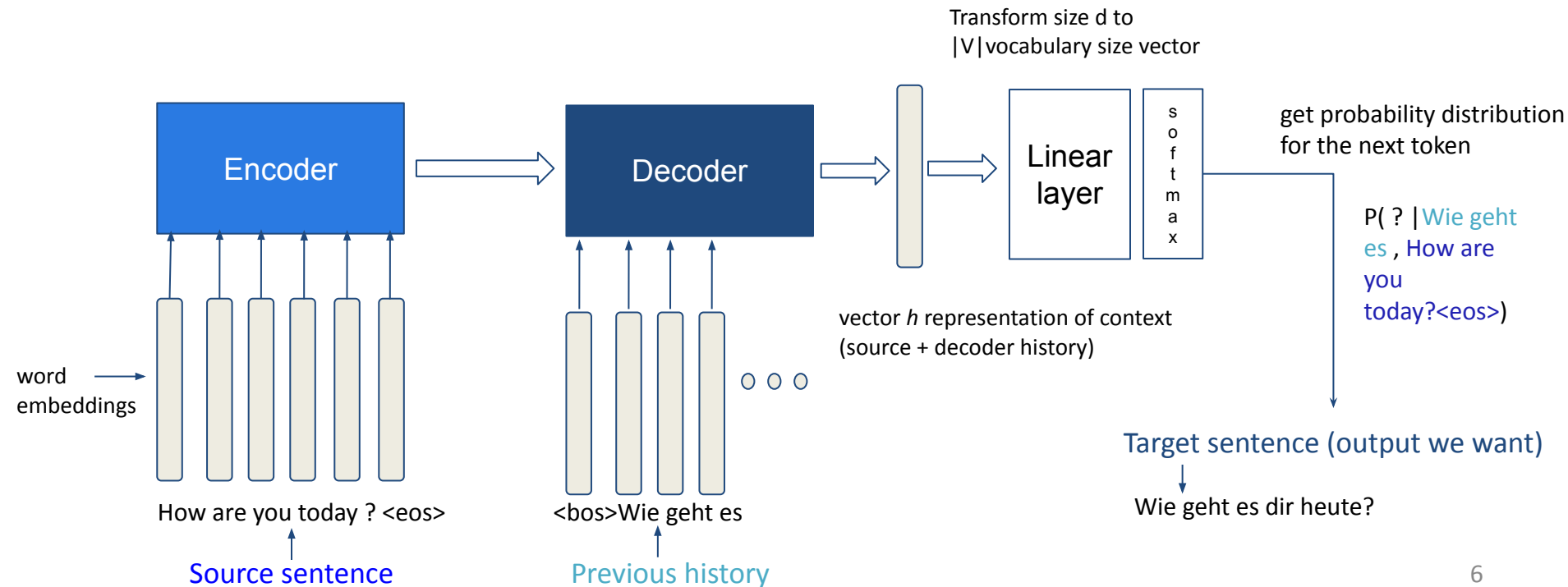
Decoder - uses source representation from the encoder to generate the target sequence.

Maps variable-length sequences to fixed-length memory

Commonly used LSTMs, GRUs in encoder, decoder

Machine translation. Conditional LM

Language models estimate the *unconditional probability* $p(y)$ of a sequence y ,
Sequence-to-sequence models need to estimate the *conditional probability* $p(y|x)$ of a
sequence y given a source x



Machine translation

Pipeline:

- feed source and previously generated target words into a network;
- get vector representation of context (both source and previous target) from the networks decoder;
- from this vector representation, predict a probability distribution for the next token.

Training:

we train to predict probability distributions of the next token given previous context (source and previous target tokens). At each step we maximize the probability a model assigns to the correct token.

Cross-entropy: looking for parameters


$$Loss(p^*, p) = -p^* \log(p) = - \sum_{i=1}^{|V|} p_i^* \log(p_i)$$

Seq2seq is optimized as a single system. Backpropagation operates “end-to-end”

Machine translation. Inference

Ok, everything is great, but **how to generate?**

How to find argmax?


$$y' = \arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t | y_{<t}, x)$$

Actually, the total number of hypotheses is $|V|^n$

We don't try to find exact solution, we approximate it.

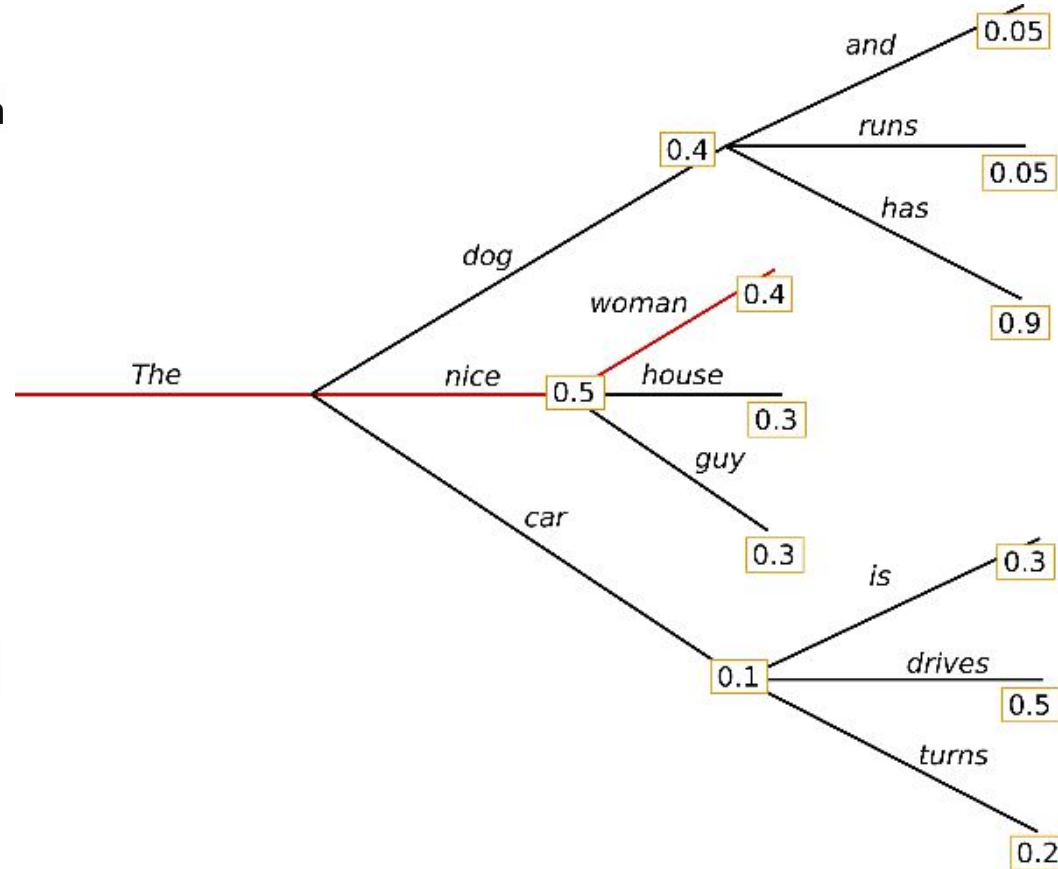
There are several methods to do it ...

Machine translation. Inference

Greedy Decoding:

At each step, pick the most probable token

Take argmax on each step of the decoder



Problems:

- the best next token \neq best sequence
- has no way to undo decisions

Machine translation. Inference

Beam Search:

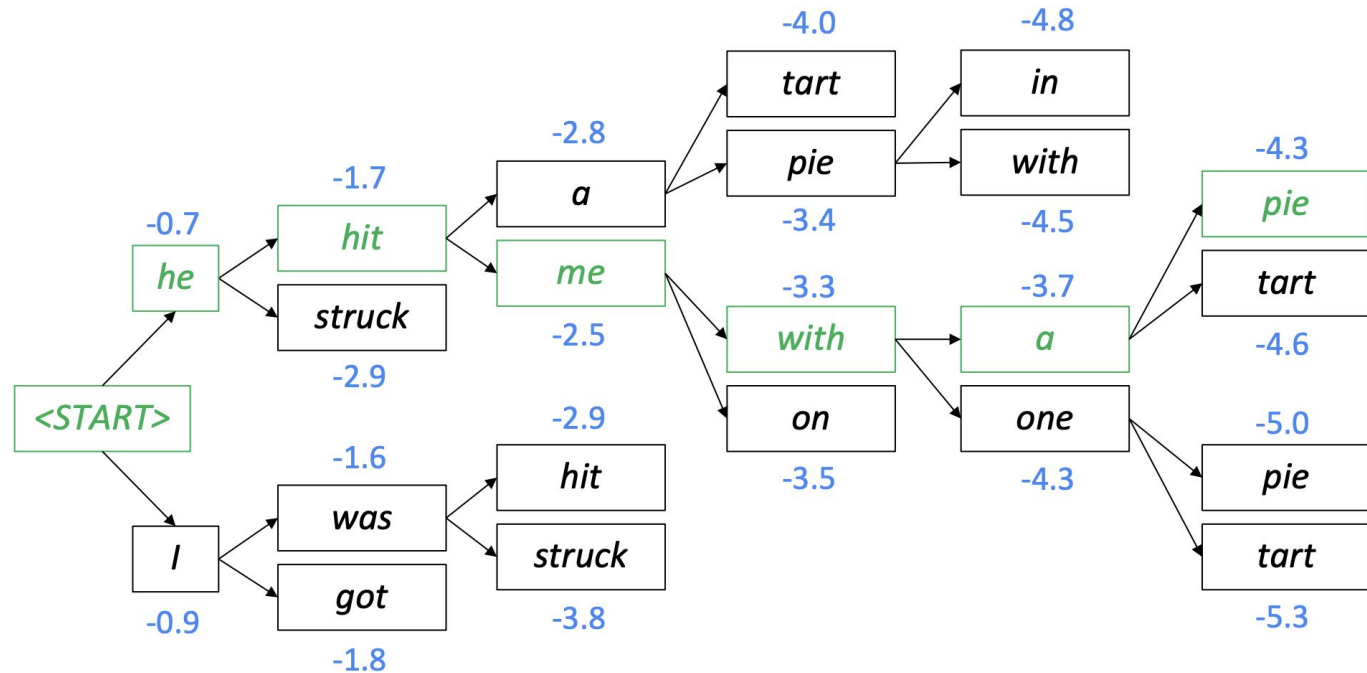
Keep
probable
translations
(hypotheses)

•

(in practice 4 to 10)

Example:

beam size $k = 2$



$$\text{score}(y_1, \dots, y_t) = \sum_{i=1}^t \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$$

Machine translation. Evaluation

BLEU (Bilingual Evaluation Understudy)

BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:

- n-gram precision (usually for 1, 2, 3 and 4-grams)
 - Plus a penalty for too-short system translations
- $$\frac{\text{(Sum over unique n-gram counts in the candidate)}}{\text{(total \# of words in candidate)}}$$

Limitations: semantics, the order of the n-grams in the sentence.

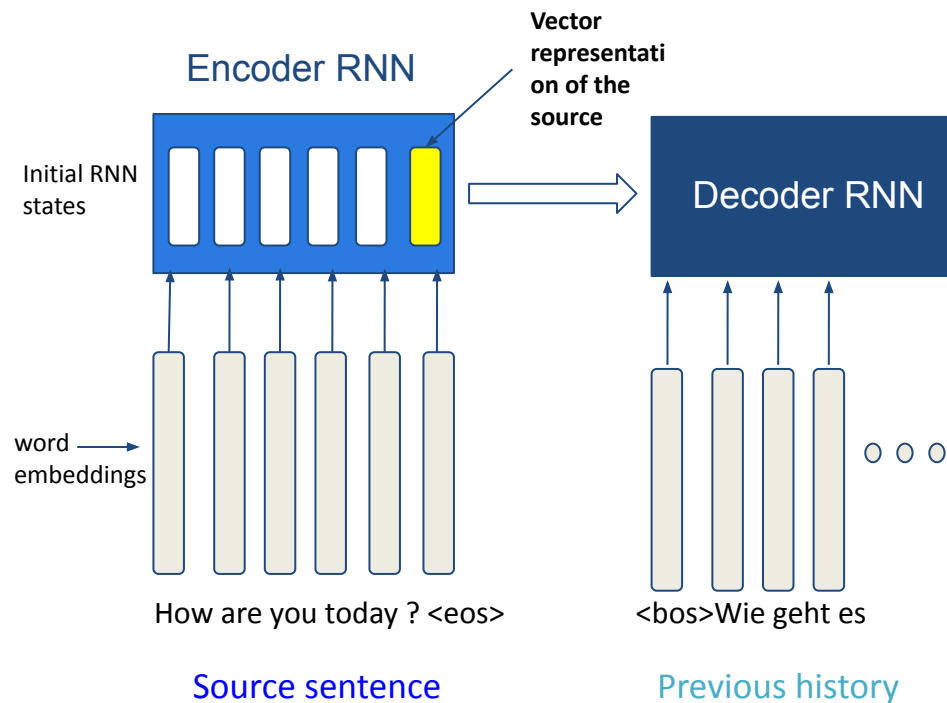
ROUGE (Recall Oriented Understudy for Gisting Evaluation)

[check quality of machine text for machine translation or summarization]

calculates precision and recall for machine texts by counting the n-gram overlap between the machine texts and a reference text

Limitations: meaning (does not understand concepts and themes)

Machine translation. Limitations



- for the encoder, it is hard to compress the sentence;
- for the decoder, different information may be relevant at different steps;
- how far we can see in the past is finite. RNN's tend to forget information from timesteps that are far behind.

The intermediate representation z cannot encode information from all the input timesteps
bottleneck problem.

The stacked RNN layers usually create **vanishing gradient problem**

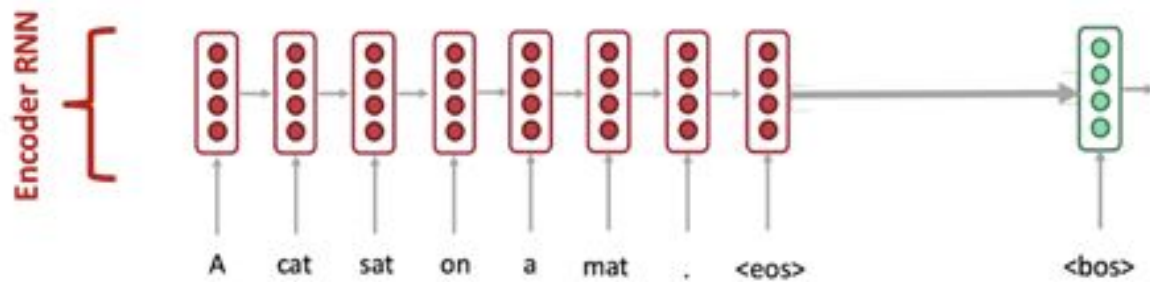
Attention:

At different steps, let a model "focus" on different parts of the source tokens
(more relevant ones).

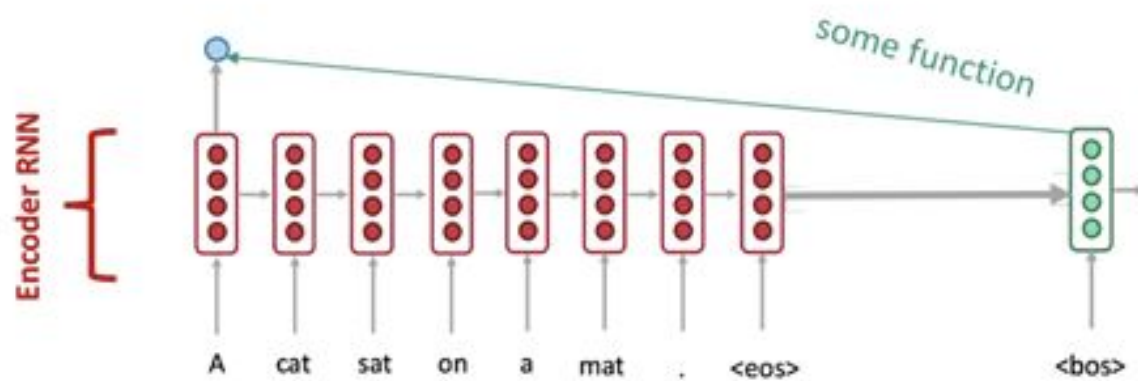
Core idea:

on each step of the decoder, use direct connection to the encoder to focus on a
particular part of the source sequence

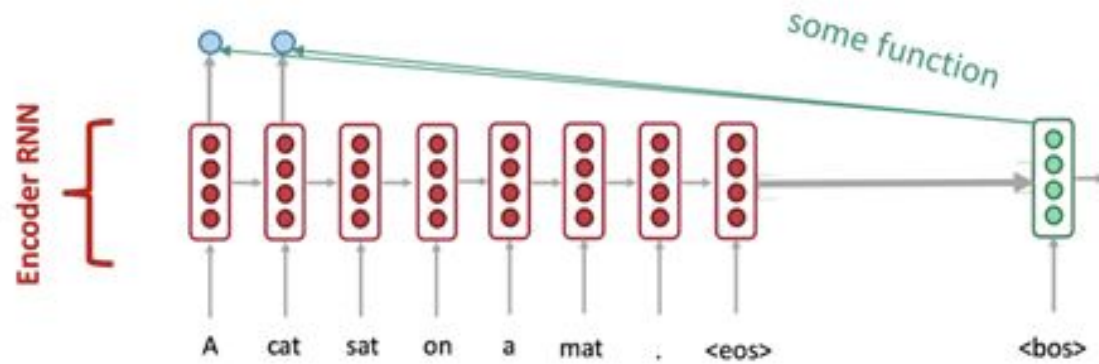
Attention



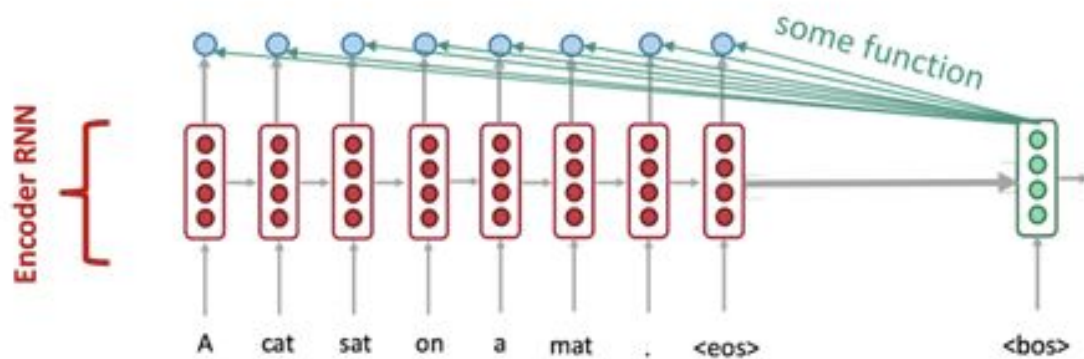
Attention



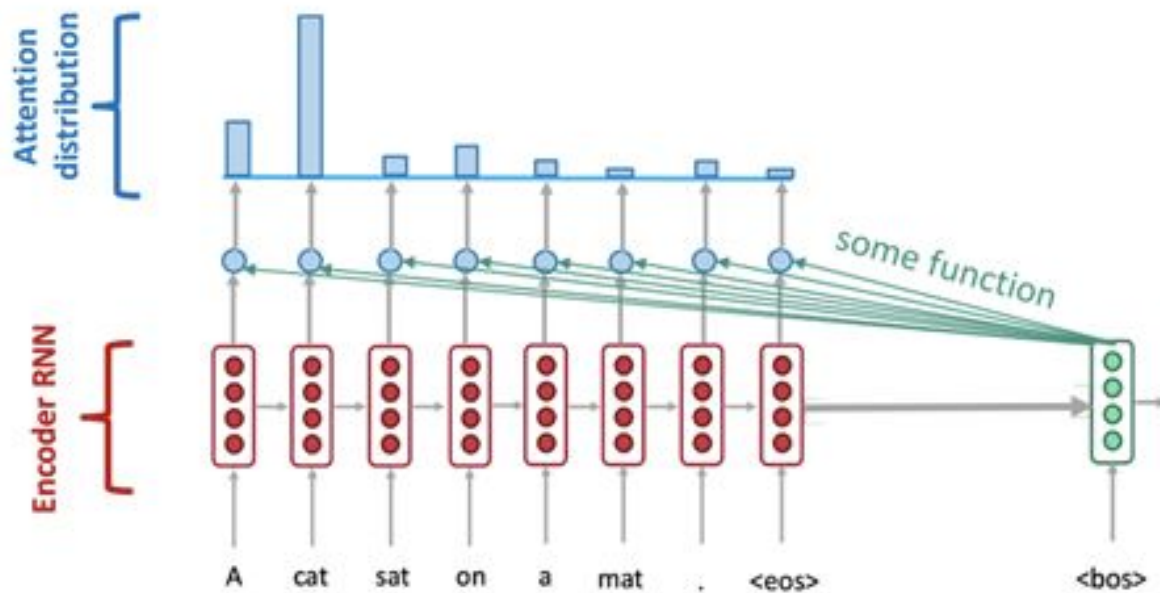
Attention



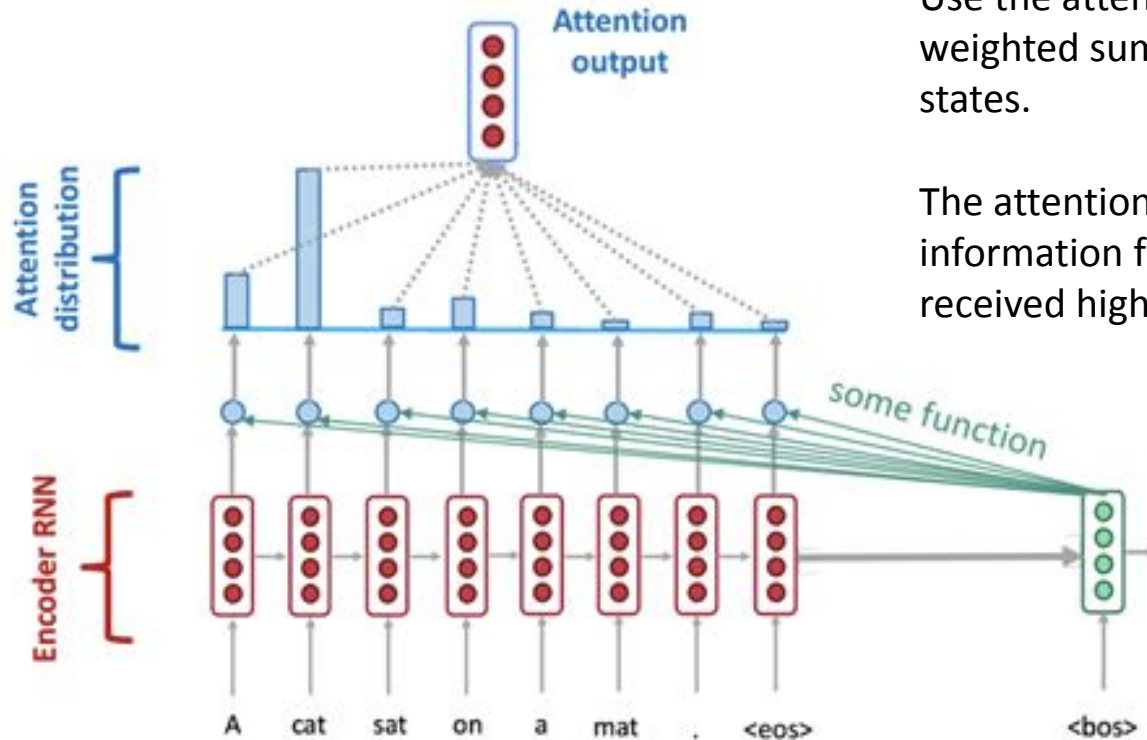
Attention



Attention



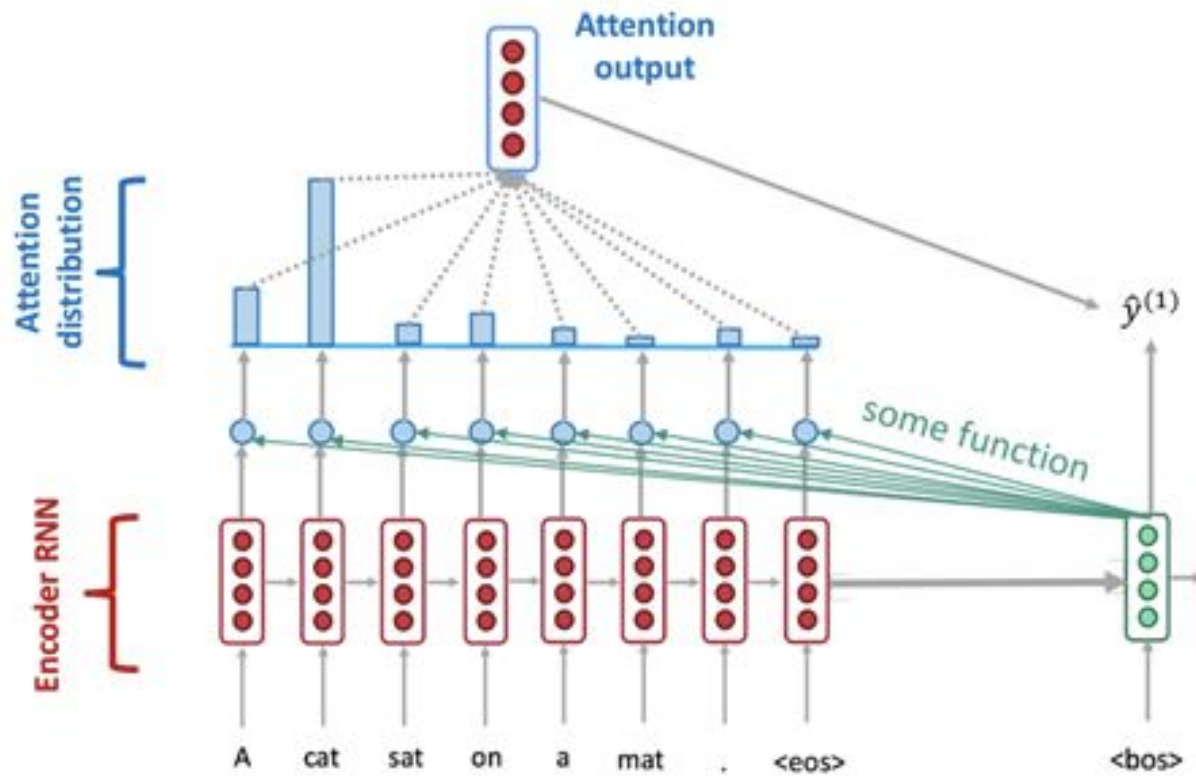
Attention



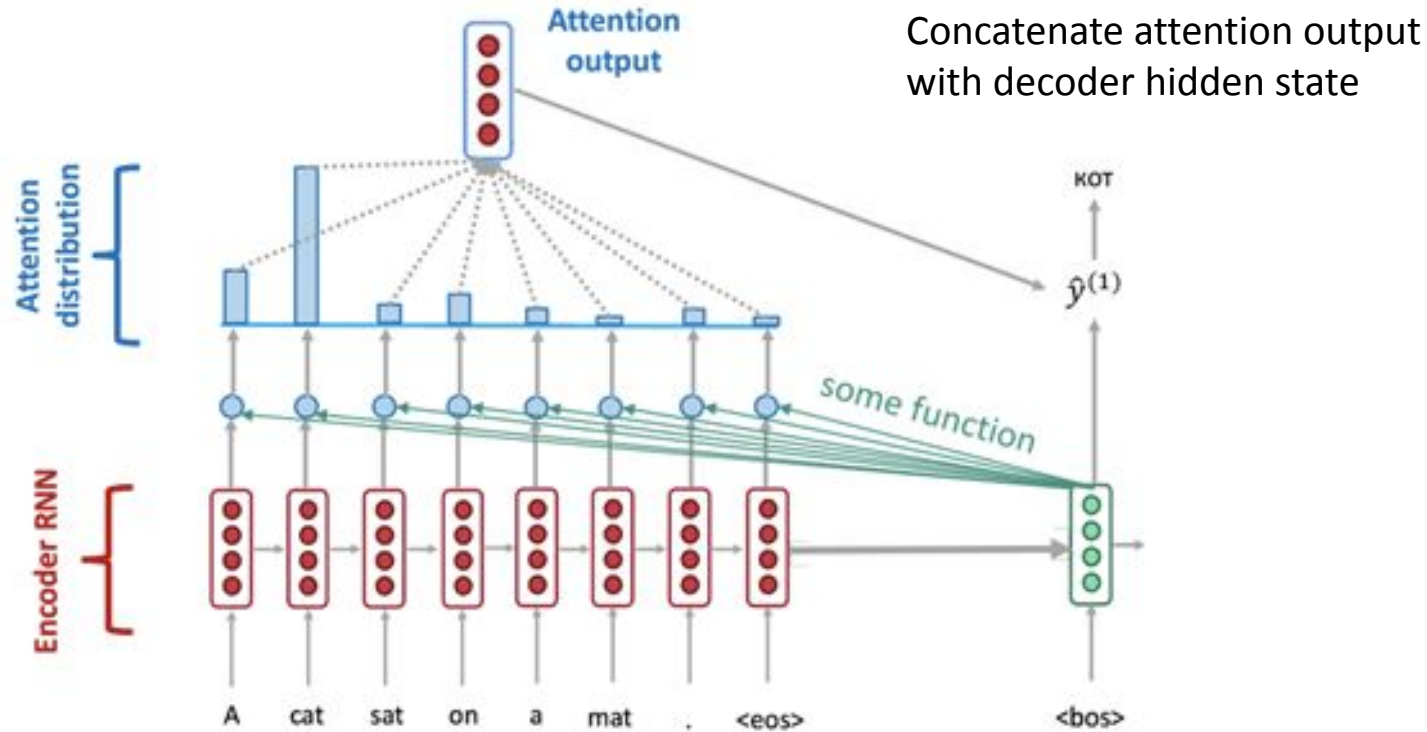
Use the attention distribution to take a weighted sum of the encoder hidden states.

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

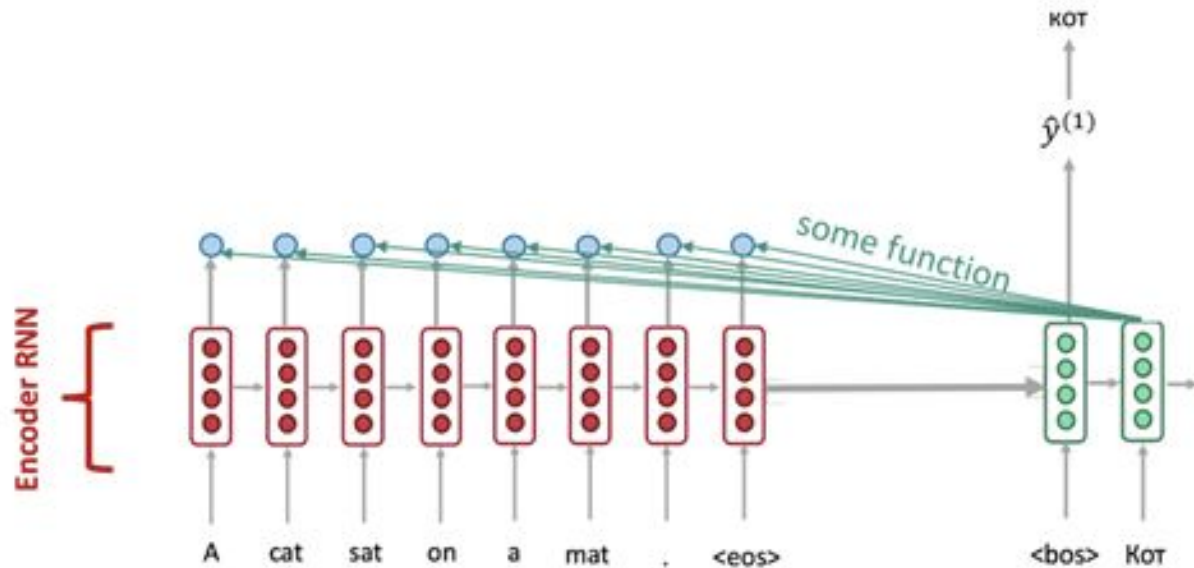
Attention



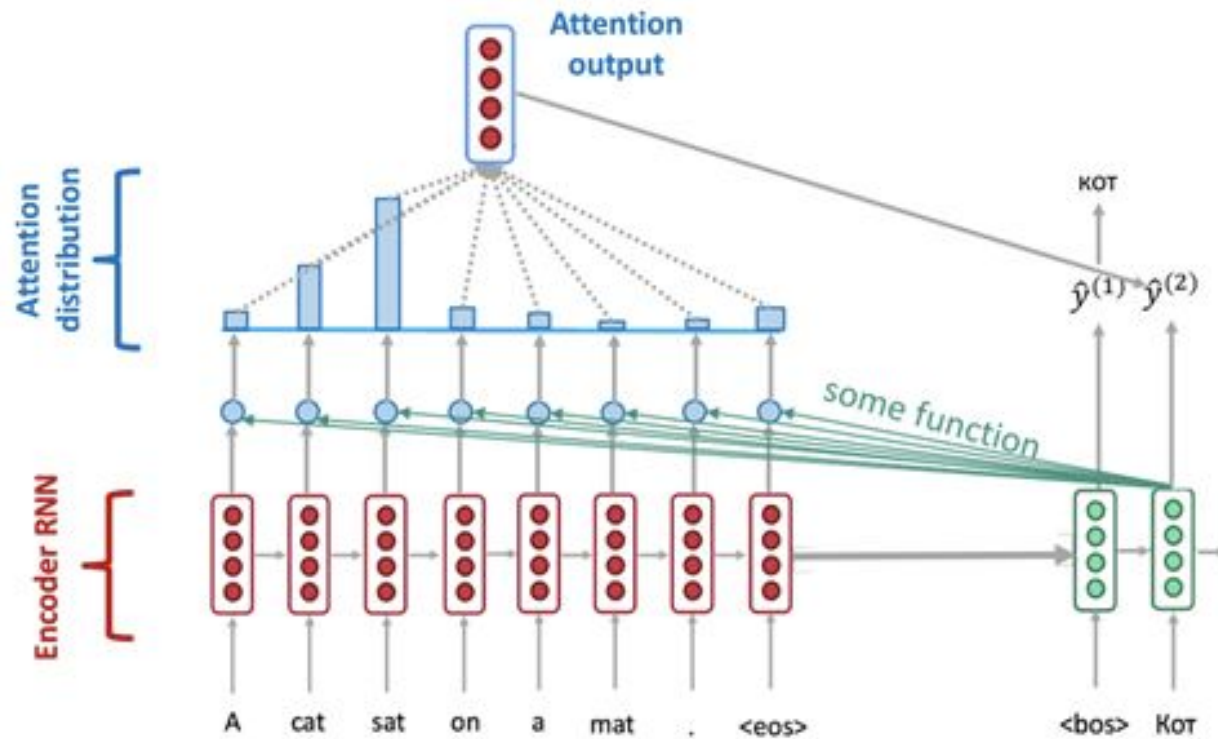
Attention



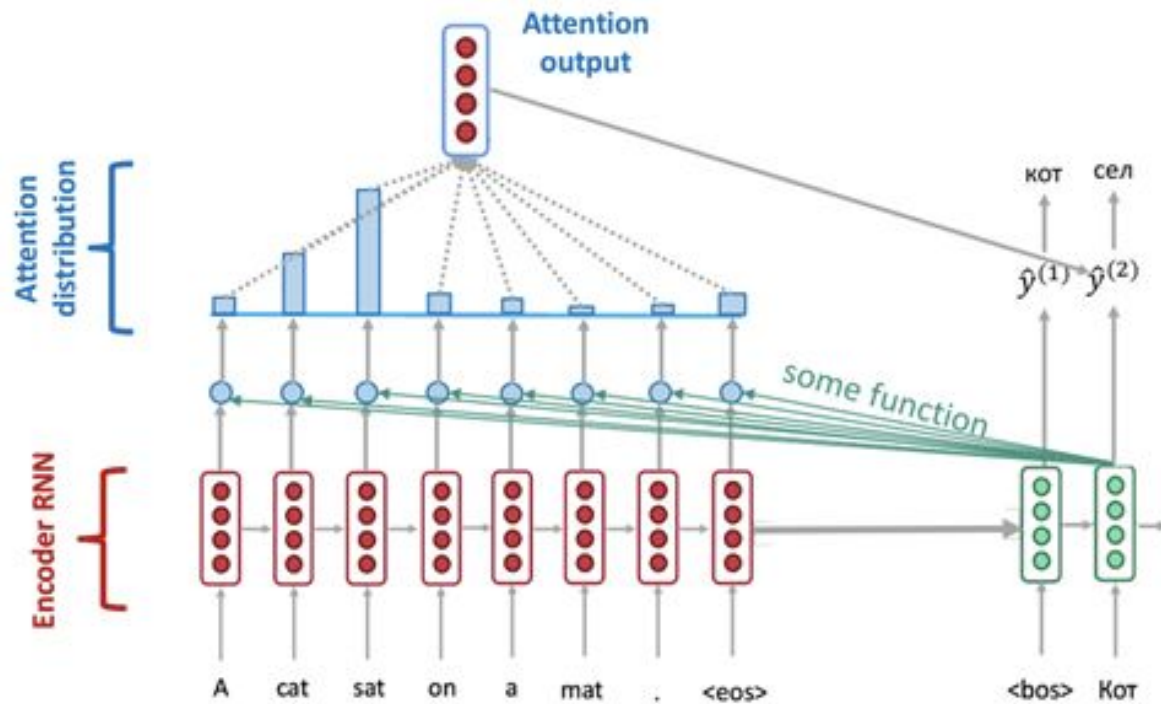
Attention



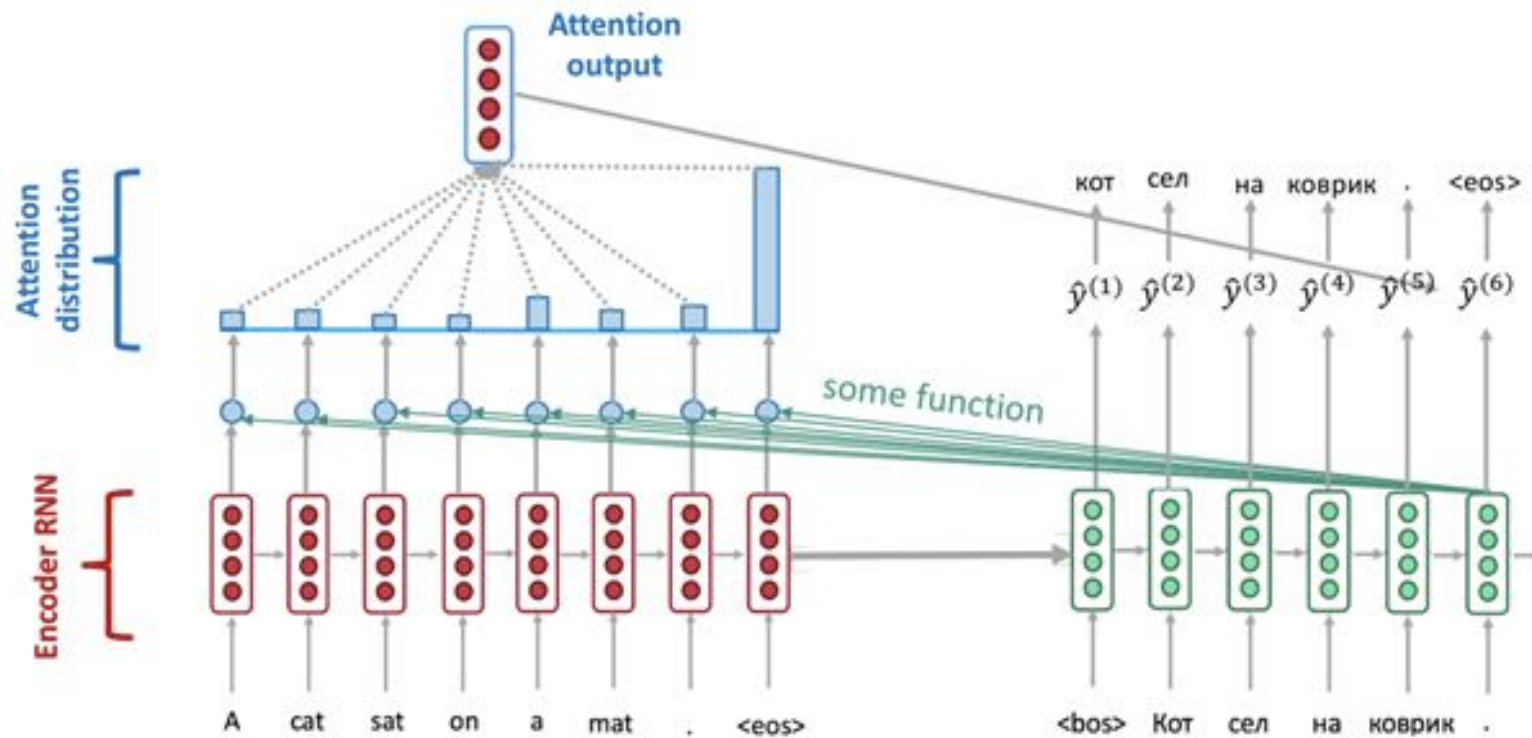
Attention



Attention



Attention



Attention

At each decoder step, attention:

- receives attention input
- computes attention scores
- computes attention weights:
a probability distribution -
softmax applied to attention
scores;
- computes attention output:
the weighted sum of encoder
states with attention weights.

Attention output

↑
(weighted
sum)

Attention weights

↑
(softmax)

Attention scores

↑

Attention input

$$\underset{\substack{\uparrow \\ \text{"source context for decoder step } t"}}}{c^{(t)}} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \dots + a_m^{(t)} s_m = \sum_{k=1}^m a_k^{(t)} s_k$$

$$\underset{\substack{\uparrow \\ \text{"attention weight for source token } k \text{ at decoder step } t"}}}{a_k^{(t)}} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^m \exp(\text{score}(h_t, s_i))}, k = 1..m$$

$$\text{score}(h_t, s_k), k = 1..m$$

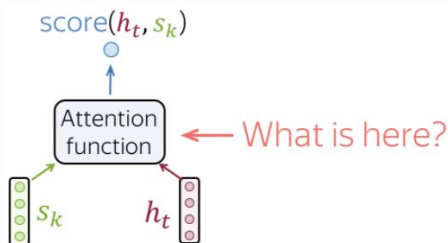
↑
"How relevant is source token k for target step t ?"

s_1, s_2, \dots, s_m
all encoder states

h_t
one decoder state

Attention variants

There are several ways to compute attention score:



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

- dot-product - the simplest method;
- general or bilinear function (aka "Luong attention") from [Effective Approaches to Attention-based Neural Machine Translation](#);
- additive or multi-layer perceptron (aka "[Bahdanau attention](#)");
- actually any function you want =).

Early attention models

Bahdanau attention (paper

Neural Machine Translation by

Jointly Learning to Align and

Translate by Dzmitry Bahdanau,

KyungHyun Cho and Yoshua

Bengio);

Luong attention (the paper

Effective Approaches to

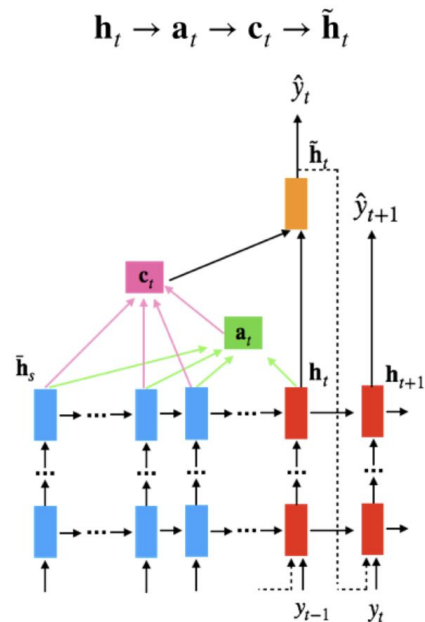
Attention-based Neural

Machine Translation by

Minh-Thang Luong, Hieu Pham,

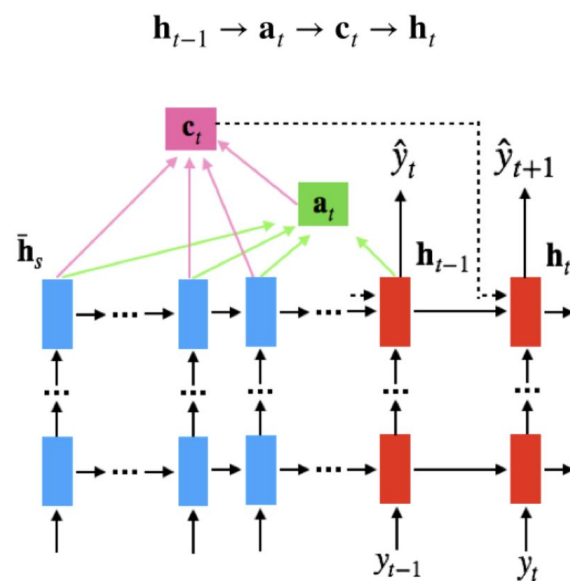
Christopher D. Manning.

Luong Attention Mechanism



$$\begin{aligned} \mathbf{a}_t(s) &= \text{align}(\mathbf{h}_t, \bar{\mathbf{h}}_s) \\ \mathbf{c}_t &= \sum \mathbf{a}_t \mathbf{h}_s \\ \tilde{\mathbf{h}}_t &= \tanh(W_c[\mathbf{c}_t; \mathbf{h}_t]) \end{aligned}$$

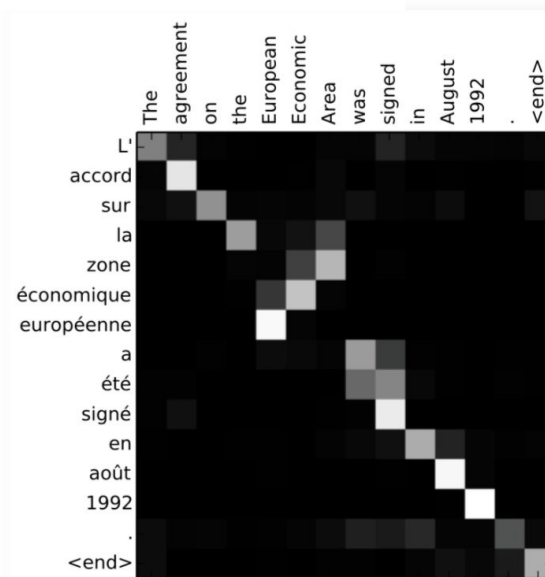
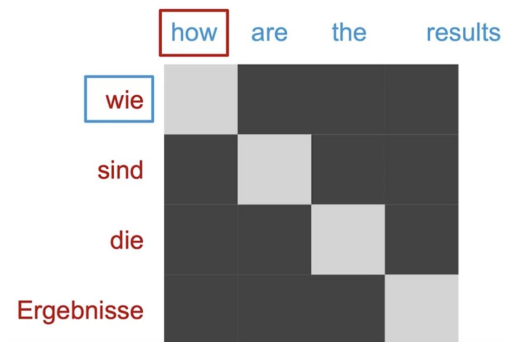
Bahdanau Attention Mechanism



$$\begin{aligned} \mathbf{a}_t(s) &= \text{align}(\mathbf{h}_{t-1}, \bar{\mathbf{h}}_s) \\ \mathbf{c}_t &= \sum \mathbf{a}_t \mathbf{h}_s \\ \mathbf{h}_t &= \text{RNN}(\mathbf{h}_{t-1}^{l-1}, [\mathbf{c}_t; \mathbf{h}_{t-1}]) \end{aligned}$$

Attention is good

- Attention significantly improves NMT performance
 - 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
 - attention distribution shows what the decoder was focusing on
 - Alignment



Transformer

Attention is all you need =)

2017

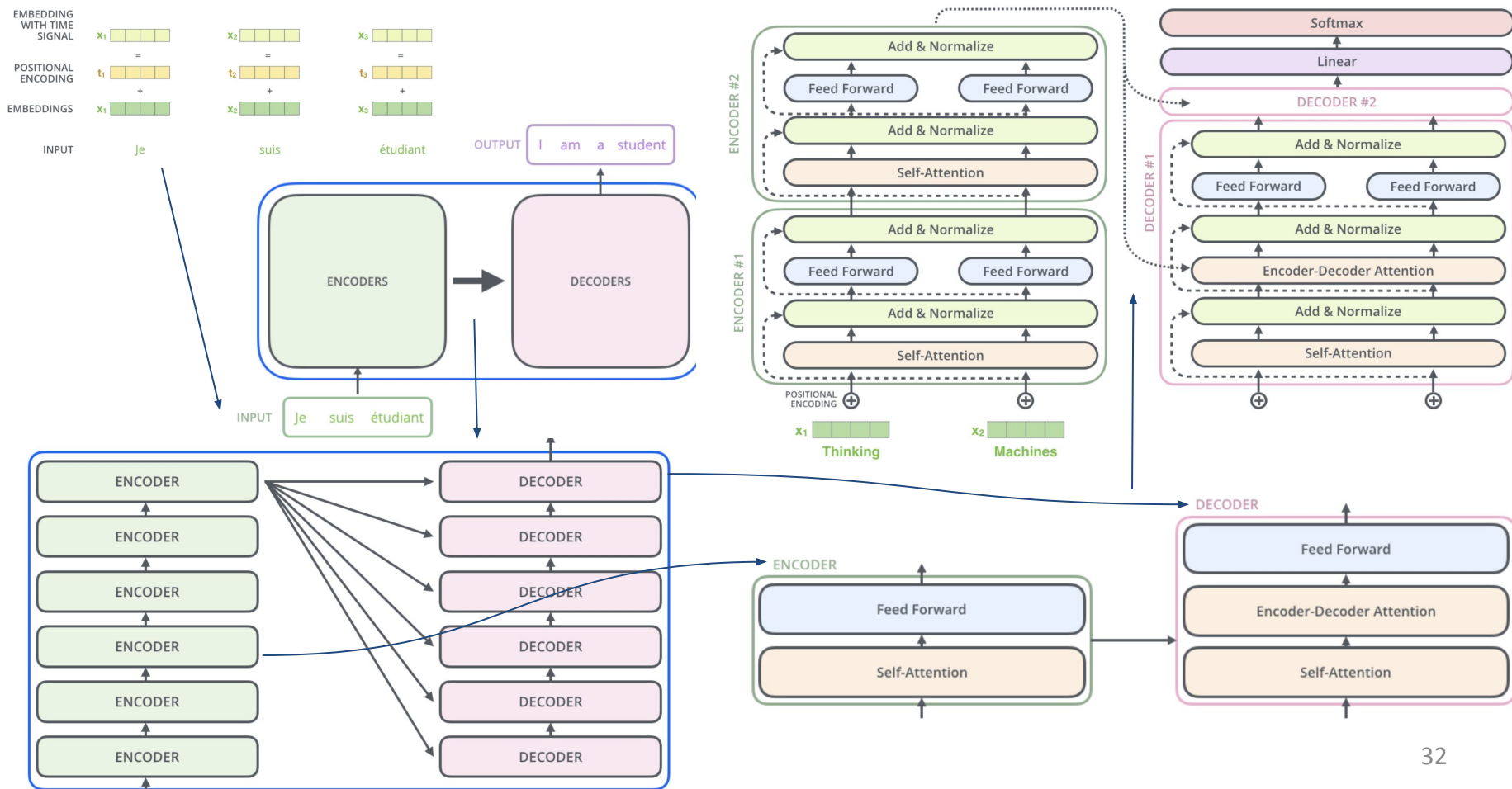
Previously:

- RNN encoder + RNN decoder, interaction
via fix-sized vector
- RNN encoder + RNN decoder, interaction
via fix-sized vector

NOW:

attention + attention+ attention

Transformers. High-level



Transformers. Self-attention

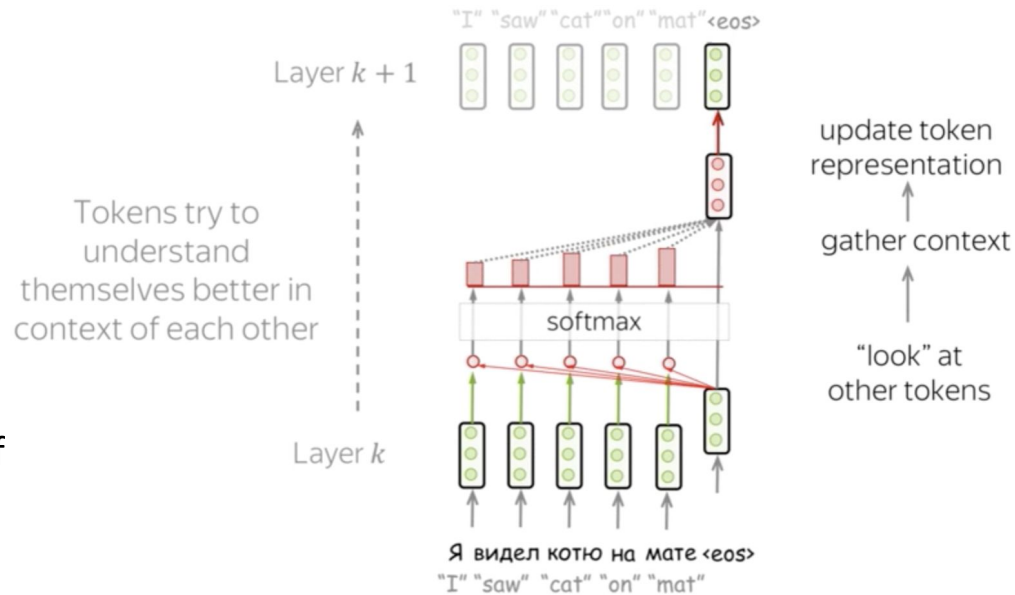
Previously - one decoder state looked at all encoder states

NOW - each state looks at each other states

Self-attention:

- tokens interact with each other
- each token "looks" at other tokens
- gathers context
- updates the previous representation of

In Parallel!



Transformers. Self-attention

Query, Key and Value vectors:

Each vector (512 dim) receives three representations:

- query - asking for information;
- key - saying that it has some information;
- value - giving the information.

These matrices allow different aspects of the x vectors to be used/emphasized in each of the three roles

Attention matches the key and query by assigning a value to the place the key is most likely to be.

$$\begin{matrix} \textcolor{green}{x} & & \textcolor{violet}{W}^Q & & \textcolor{violet}{Q} \\ \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} & \times & \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} & = & \begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix} \end{matrix}$$

$$\begin{matrix} \textcolor{green}{x} & & \textcolor{brown}{W}^K & & \textcolor{brown}{K} \\ \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} & \times & \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} & = & \begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix} \end{matrix}$$

$$\begin{matrix} \textcolor{green}{x} & & \textcolor{blue}{W}^V & & \textcolor{blue}{V} \\ \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} & \times & \begin{matrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{matrix} & = & \begin{matrix} \square & \square & \square \\ \square & \square & \square \end{matrix} \end{matrix}$$



Attention weights

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

Transformers. Masked self-attention

Decoder has different self-attention =>

Masked self-attention

- we generate one token at a time (we don't have all source tokens at once like in encoder):
during generation, we don't know which tokens we'll generate in future.
- to enable parallelization we forbid the decoder to look ahead - future tokens are masked out
(setting them to **-inf**) before the softmax step in the self-attention calculation

Transformers. Multi-head attention

- We need to know different relationships between tokens in a sentence: syntactic relationships, lexical preferences, order, grammar issues like case or gender agreement.

- Instead of having one attention mechanism, multi-head attention has several "heads" which work independently and focus on different things.

- Heads work independently

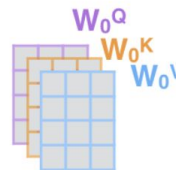
1) This is our input sentence*

Thinking
Machines

2) We embed each word*



3) Split into 8 heads. We multiply X or R with weight matrices



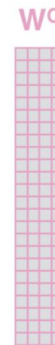
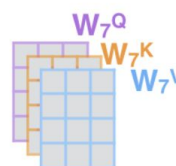
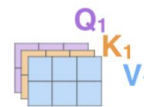
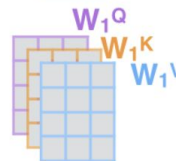
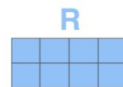
4) Calculate attention using the resulting $Q/K/V$ matrices



5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

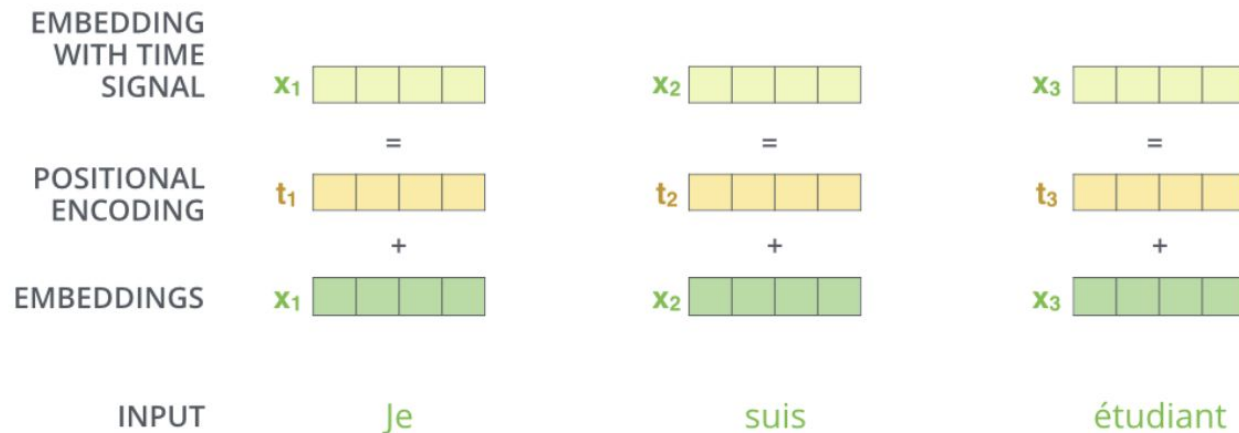


$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^O$$

$$\text{where head}_i = \text{Attention}(\mathbf{Q} \mathbf{W}_i^Q, \mathbf{K} \mathbf{W}_i^K, \mathbf{V} \mathbf{W}_i^V)$$

Transformers. Positional encoding

- positional encoding provides *order information* to the model



The fixed positional encodings

used in the Transformer

$$\text{PE}(i, \delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

Transformers. Extra

- **Feed-forward blocks**

each layer has a feed forward network block:
two linear layers with ReLU non-linearity between them

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2.$$

- **Residual connection (train better)**

Residual connections => add a block's input to its output

They ease the gradient flow through a network and allow stacking a lot of layers

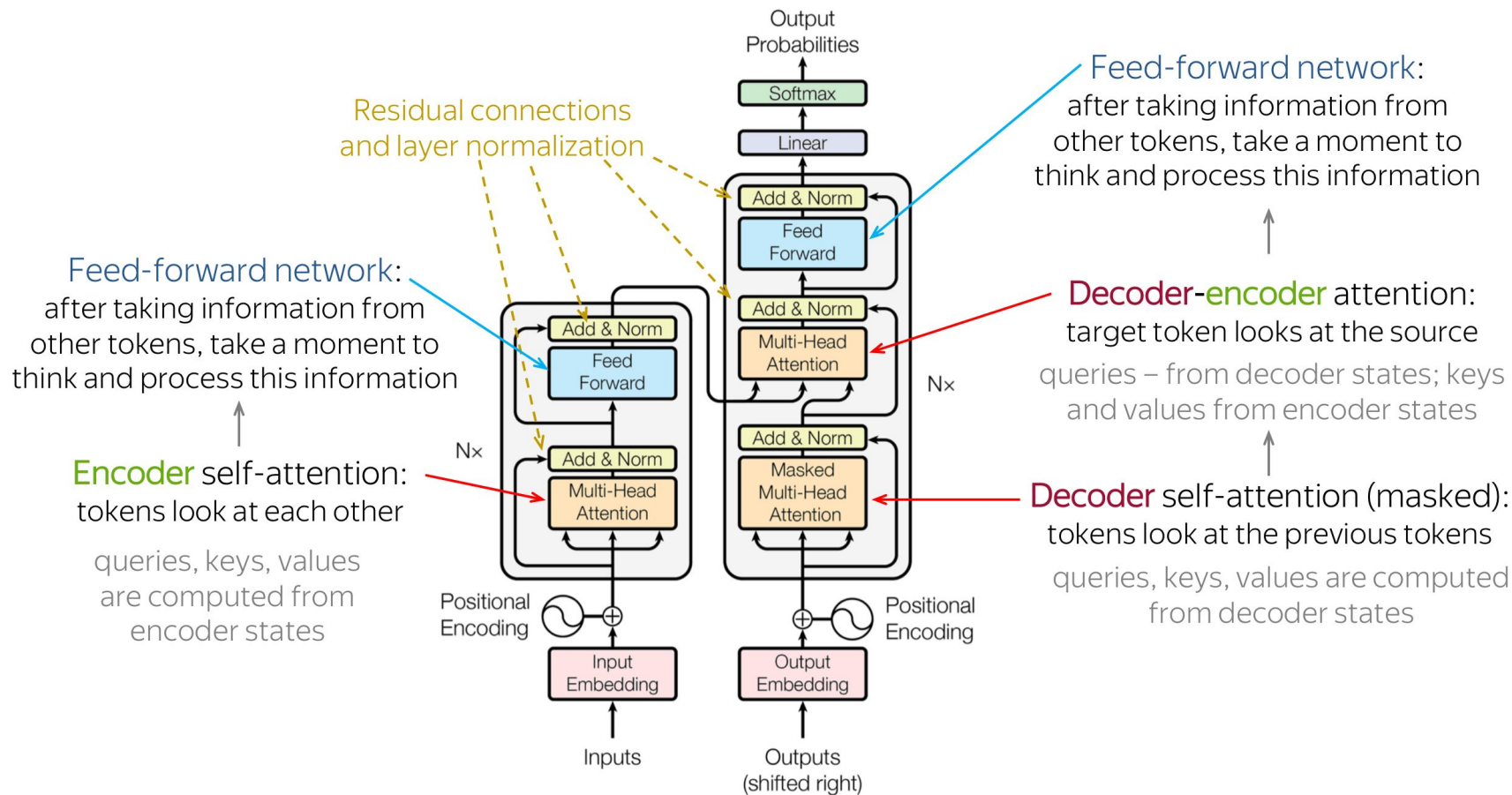
* In the Transformer in the "Add & Norm" part, the "Add" part stands for the residual connection.

- **Layer Normalization (train faster)**

Normalizes vector representation of each example in batch. Improves convergence stability

Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer

Transformers. One more time



Transformers

Why are transformers good?

- No recurrence, distributed and independent representations at each block
(all tokens can be processed at once)
- No long relations: for recurrent models - one training step requires $O(\text{len}(\text{source}) + \text{len}(\text{target}))$ steps, for Transformer, it's $O(1)$, since all words interact at every layer.
- Fast learning: encoder and decoder can be parallel
- Self-Attention: model does not need to remember too much
- Multi-head attention allows to pay attention to different aspects
- The meaning heavily depends on the context

References

Transformers

[Attention is all you need](#)

[Transformer survey](#)

High-level

Jay Alammar:

- Transformers - <http://jalammar.github.io/illustrated-transformer/>
- Seq2seq with attention - <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>

AI Summer <https://theaisummer.com/transformer/>

Deeper

- Stanford Lectures https://www.youtube.com/watch?v=lxQtK2SjWWM&ab_channel=StanfordUniversitySchoolofEngineering
- Lena Voita https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- <https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html>

With code:

- <https://nlp.seas.harvard.edu/2018/04/03/attention.html>
- <https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/text/transformer.ipynb#scrollTo=1kLCla68EIoE>