AI in Mathematics
Lecture 7
Deep Learning in Mathematics

Bar-Ilan University
Nebius Academy | Stevens Institute of Technology

May 6, 2025

About This Course

~~1 week: Intro~~

~~2 weeks: Classic ML~~

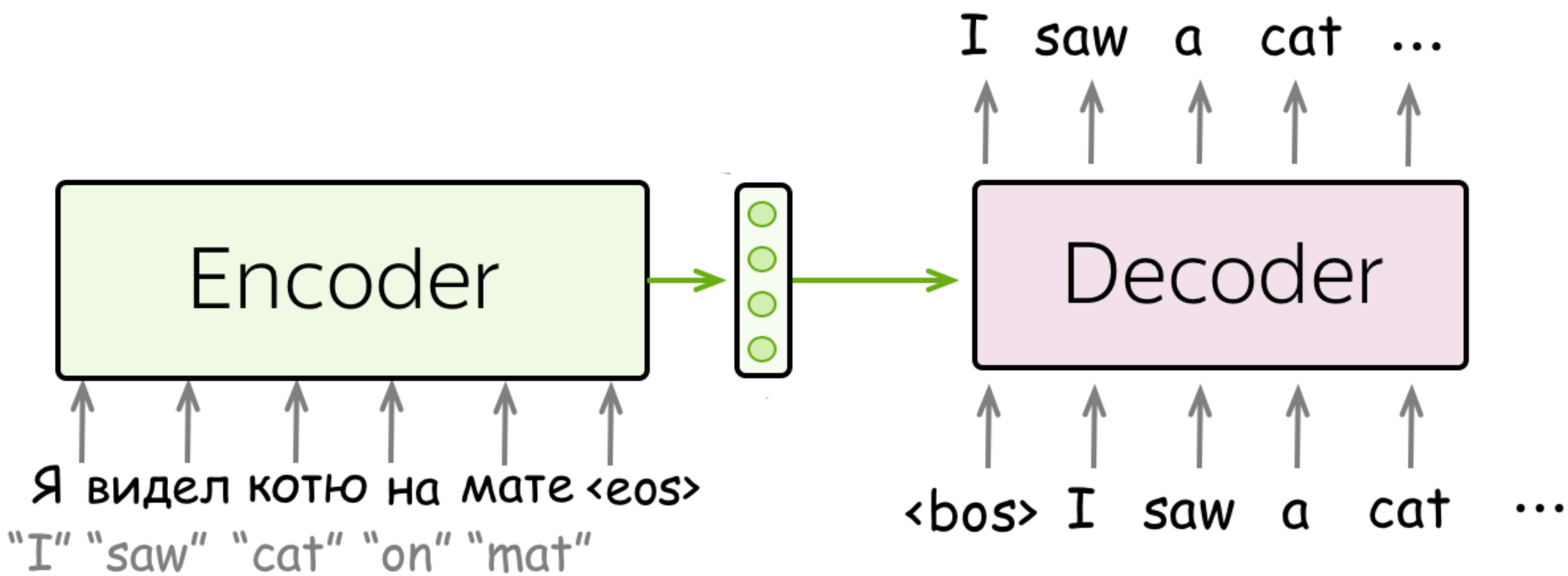
~~2 weeks: Deep Learning in Mathematics~~

4 weeks: Math as an NLP problem (LLMs etc.)

3 weeks: Reinforcement Learning (RL) in Math

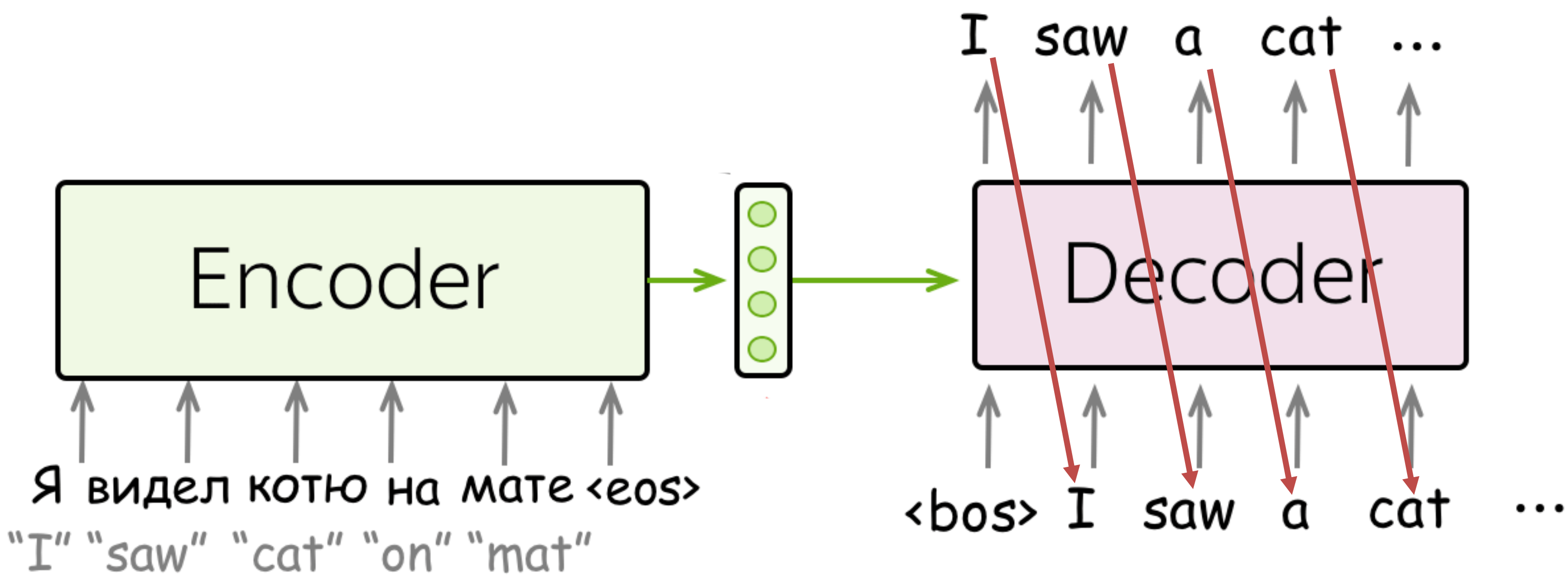
1 week: Advanced AI topics or Project Presentations

Seq2Seq

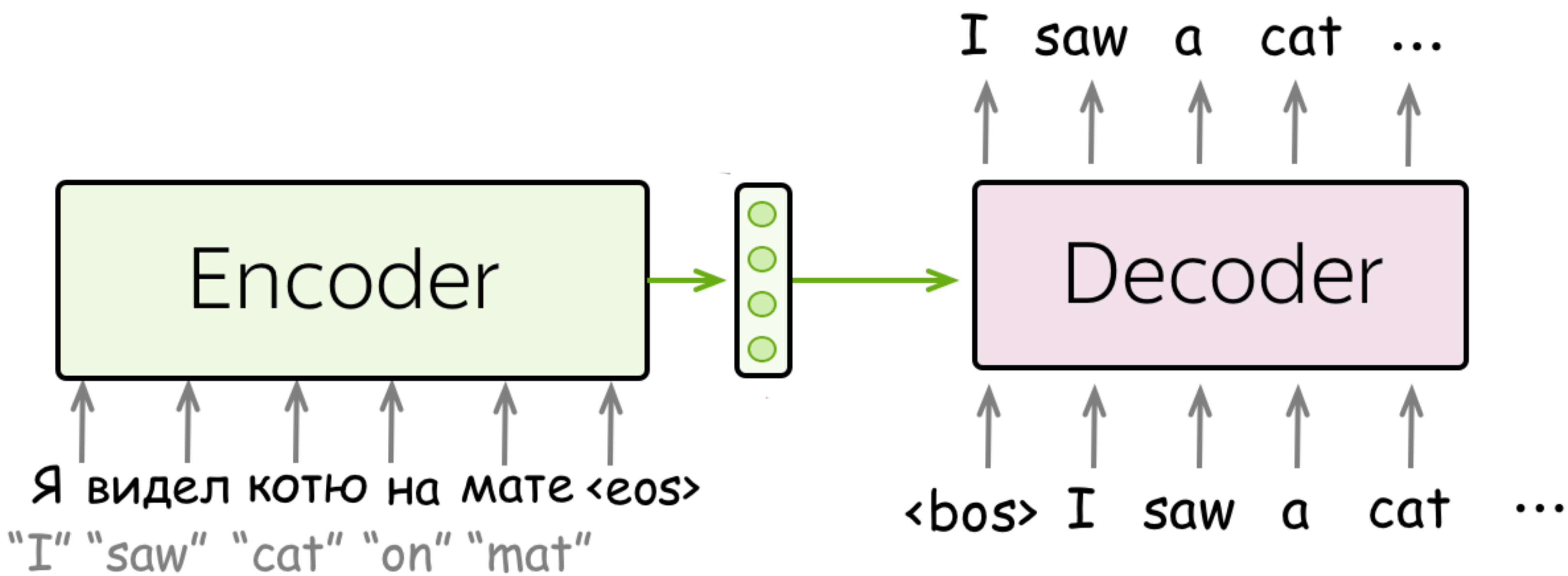


Pictures here and further from [NLP Course by Lena Voita]()

Seq2Seq

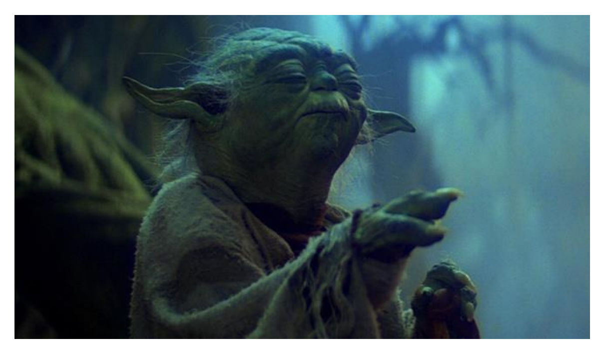

I saw a cat ...
Encoder
Decoder
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a cat ...

Seq2Seq

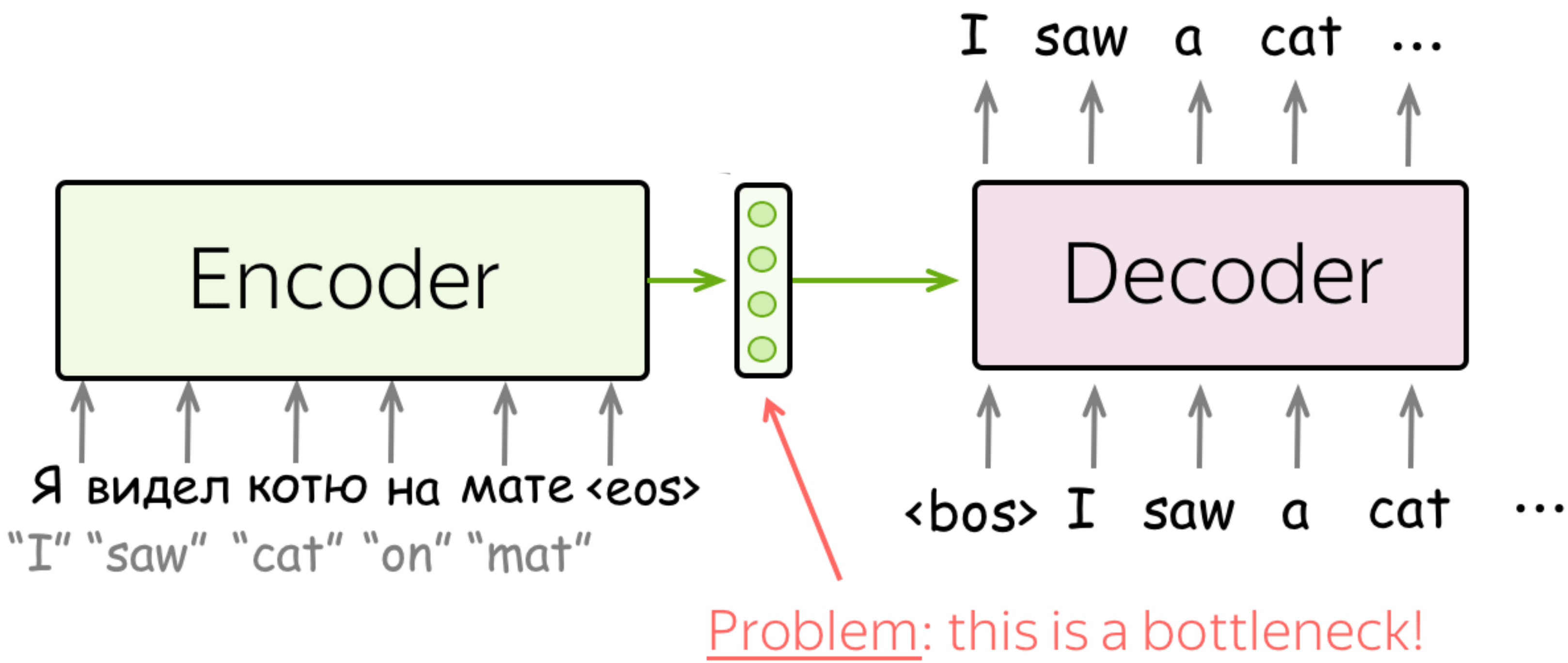


Teacher forcing during train:

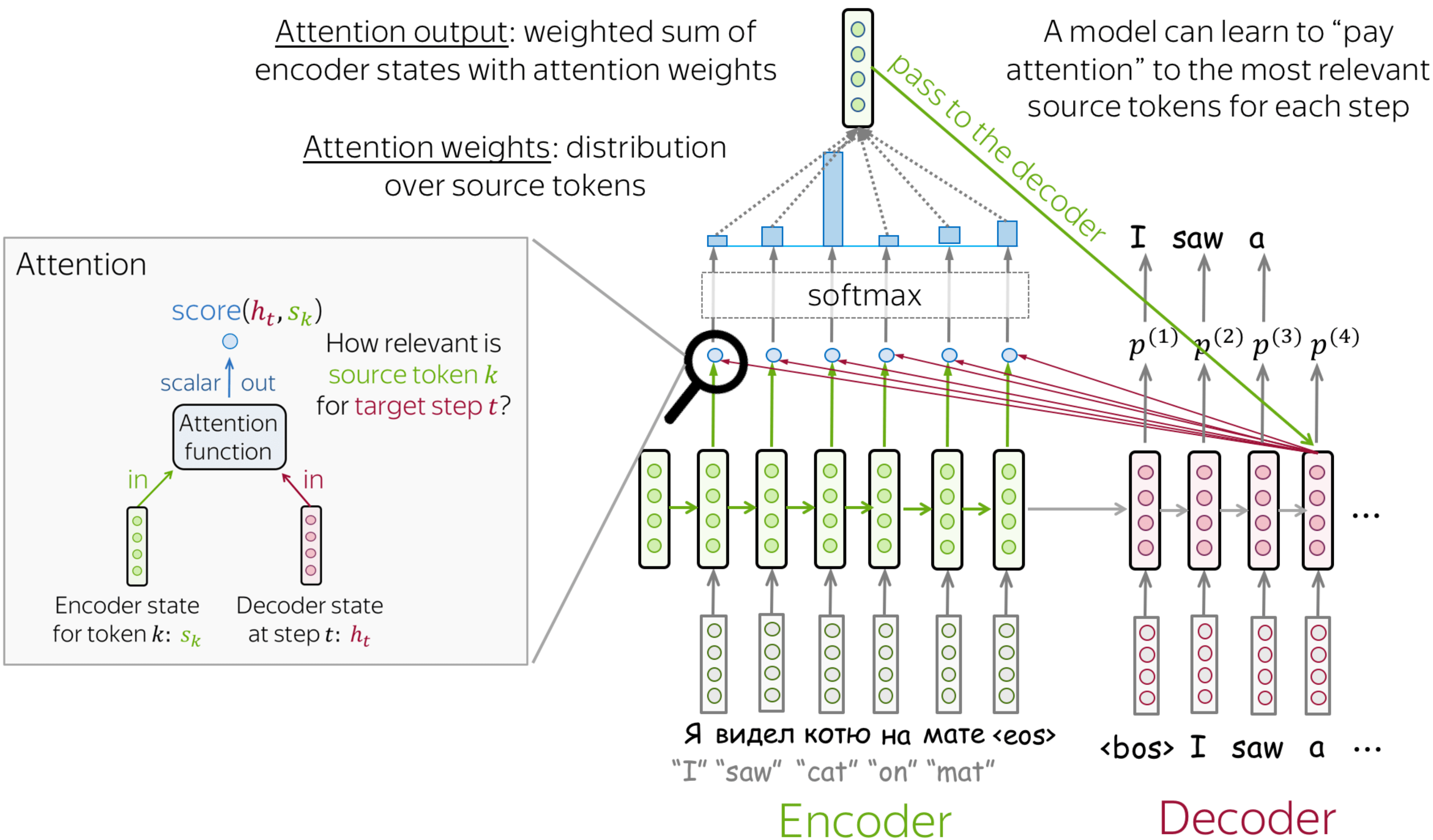
We provide ground-truth tokens on a train.



Seq2Seq


I saw a cat ...
Encoder
Decoder
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a cat ...
Problem: this is a bottleneck!

Attention!!!!

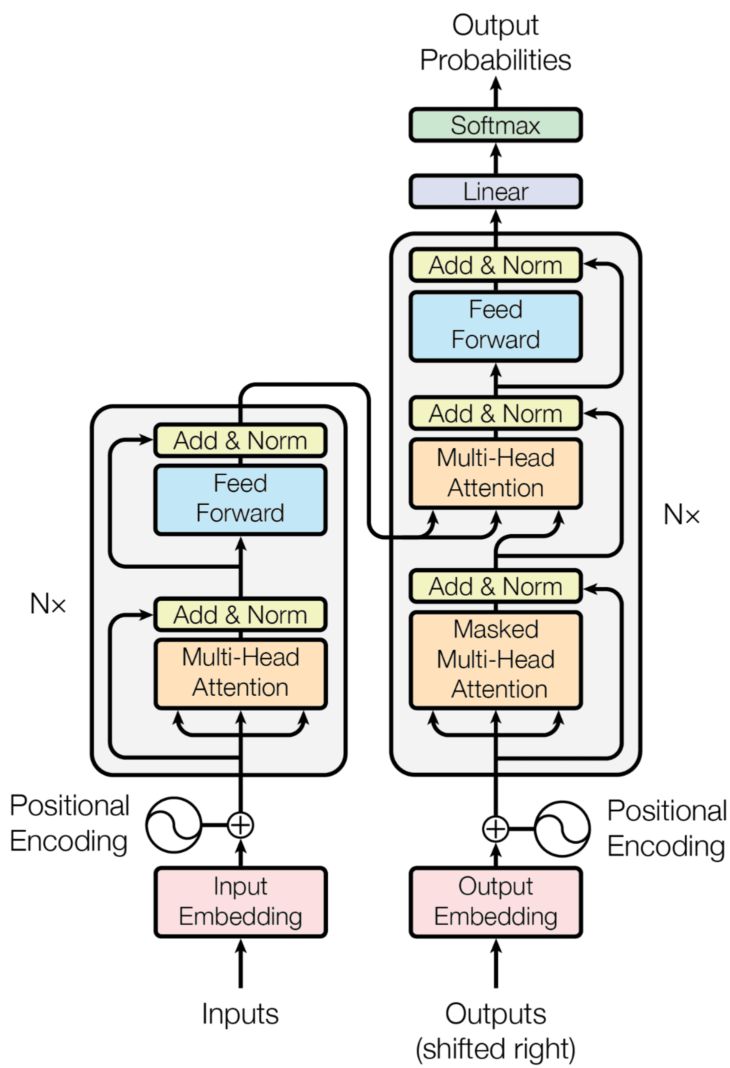


Attention output: weighted sum of encoder states with attention weights
Attention weights: distribution over source tokens
A model can learn to "pay attention" to the most relevant source tokens for each step
pass to the decoder
softmax
I saw a
$p^{(1)}$ $p^{(2)}$ $p^{(3)}$ $p^{(4)}$
Attention
score(h_t, s_k)
scalar out
Attention function
in
in
How relevant is source token k for target step t?
Encoder state for token k: s_k
Decoder state at step t: h_t
Я видел котю на мате <eos>
"I" "saw" "cat" "on" "mat"
<bos> I saw a …
Encoder
Decoder

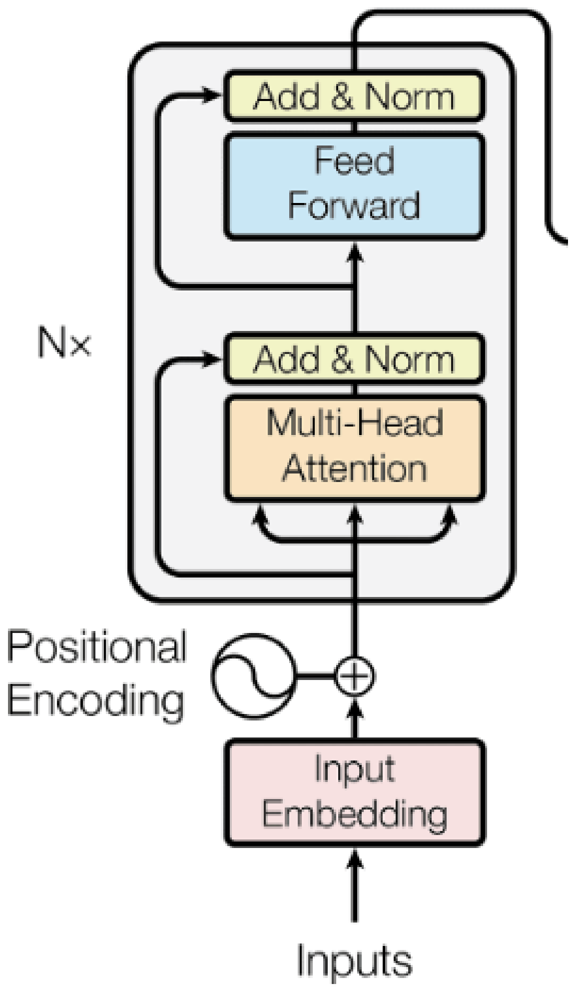
Structure

	Seq2Seq	Seq2Seq +Attention	Transformer
Encoder	RNN	RNN	Attention
Decoder	RNN	RNN	Attention
Interaction Encoder ↔ Decoder	Vector	N Vectors (Attention)	Attention

Transformer architecture



Encoder



N consequative blocks.
Multihead attention is concatination of attention outputs

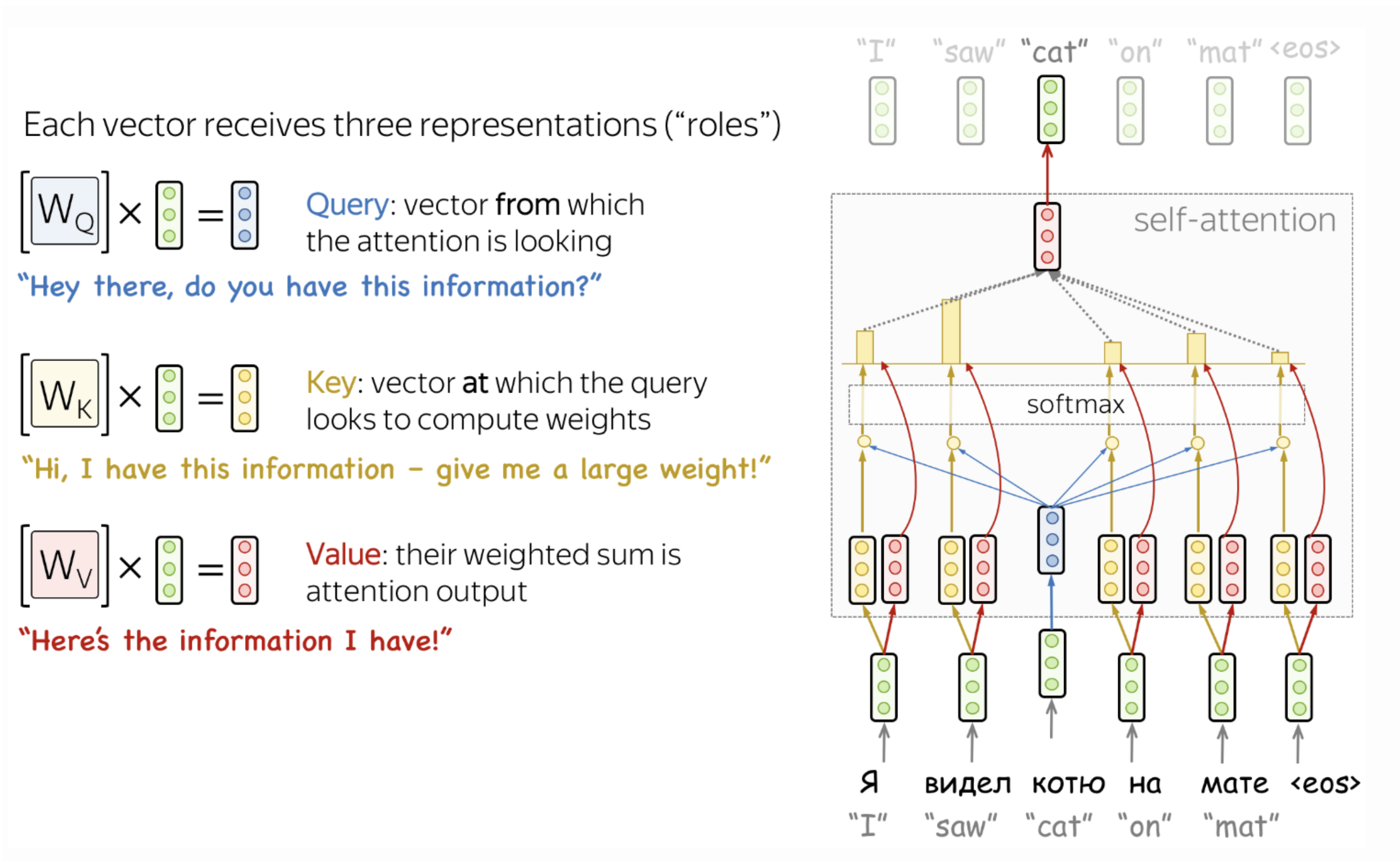
Add & Norm is a summation of previous information

Feed forward is a fully connected neural network.

Let’s take a closer look at multi-head attention.

QKV—Attention.

This version of attention won the hearts of all ML people.

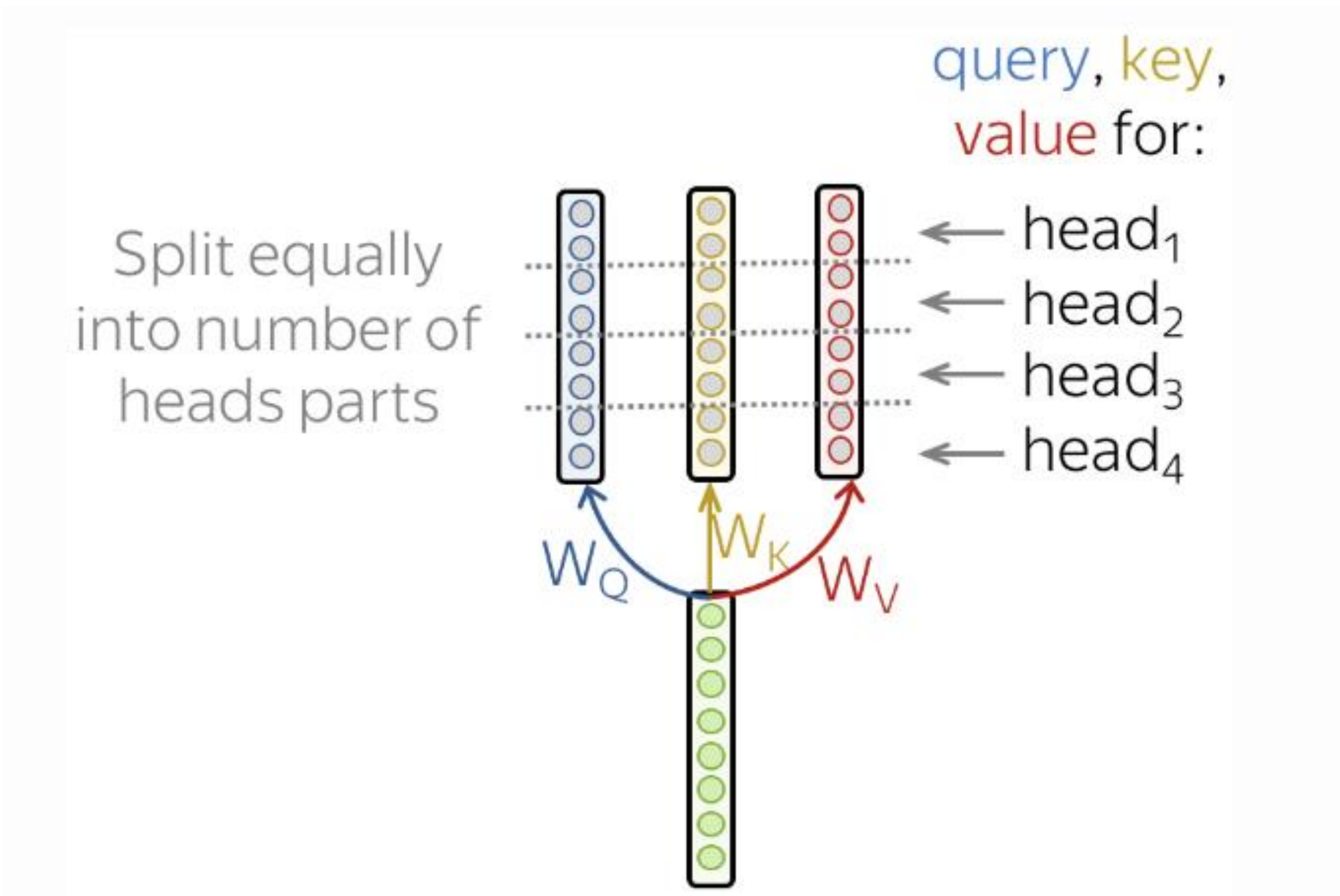


Attention

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

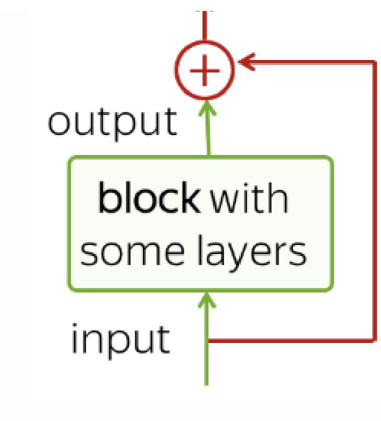
d_k is dimensionality of the key vectors

Attention

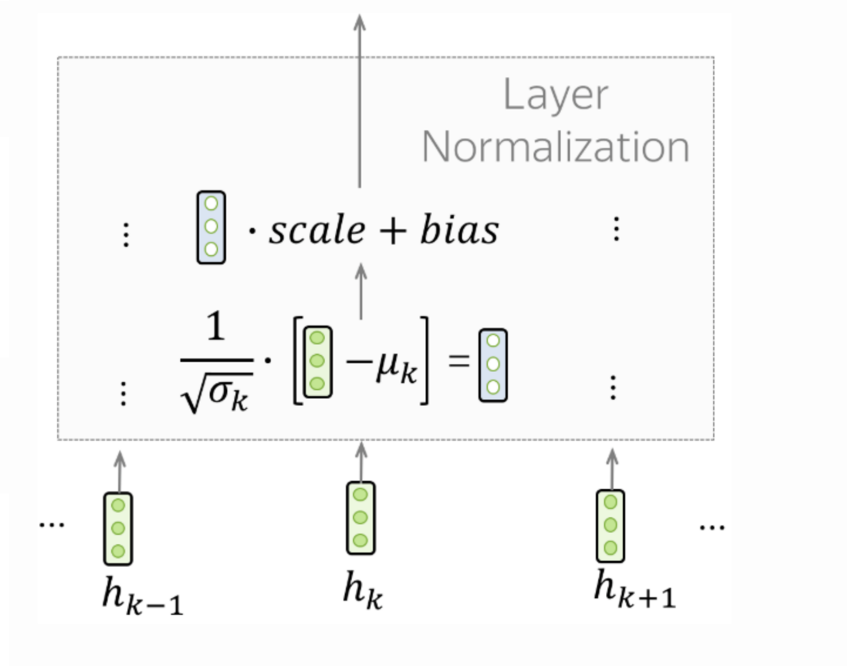

query, key,
value for:
Split equally
into number of
heads parts
head$_1$
head$_2$
head$_3$
head$_4$
W_Q
W_K
W_V

Add & Norm

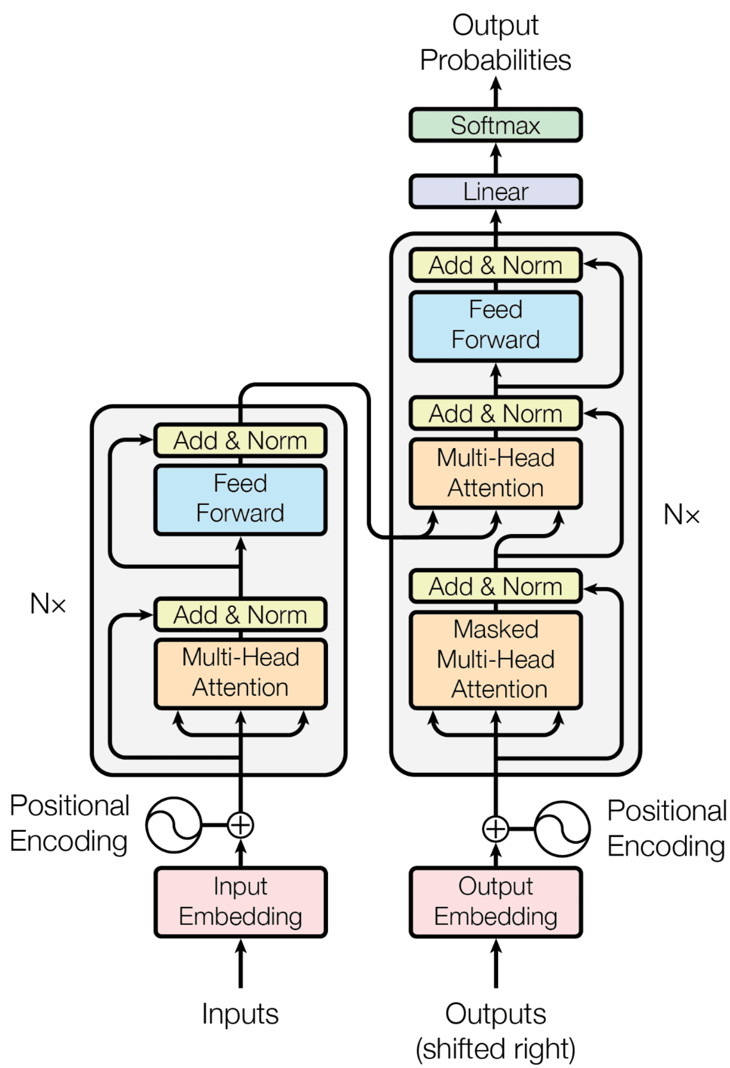
Residial connection:
Addresses
vanishing gradient.



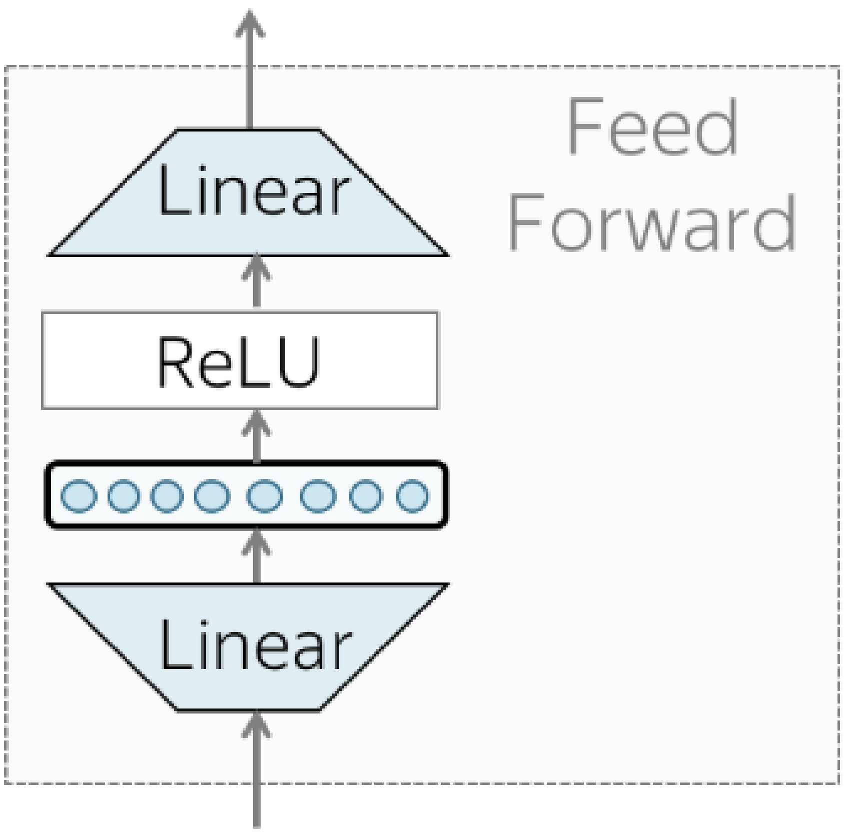
Layer normalization:
Stabilize learning



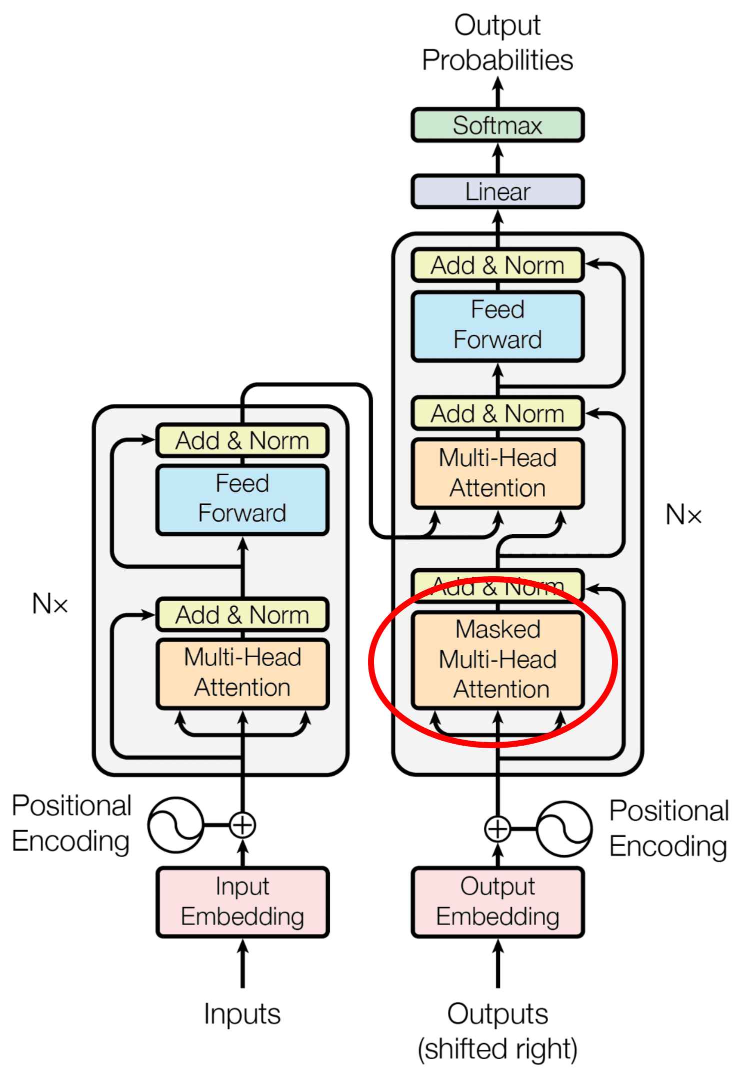
Transformer



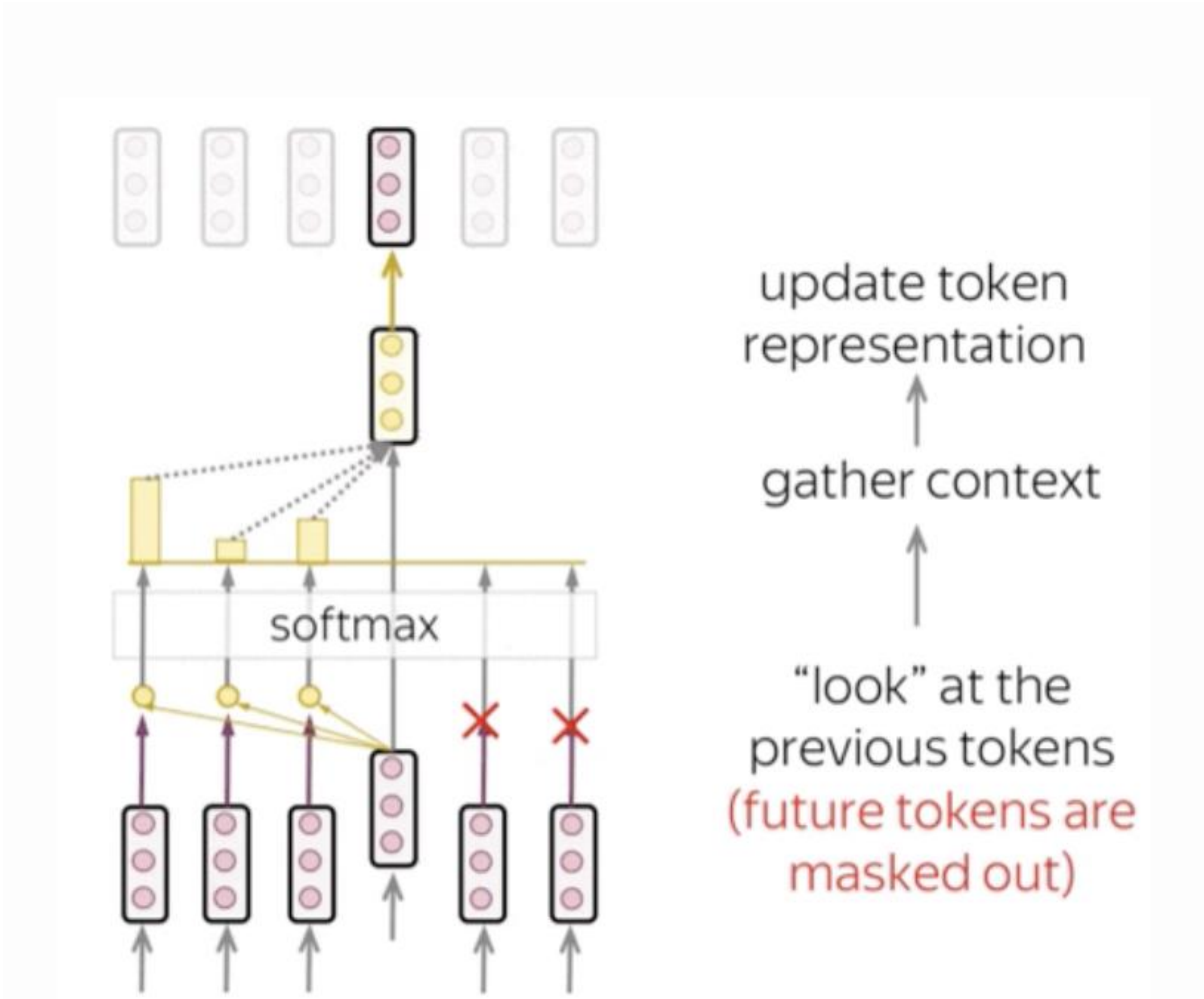
Feed Forward



Decoder


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

Masked Self-Attention



Masked attention

$$Attention(q, k, v) = softmax\left(\frac{qk^T}{\sqrt{d_k}} + \boldsymbol{M}\right)v$$

$$M = \begin{pmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{pmatrix}$$

Example for 3 tokens with Second and third masked.

M is a masked matrix, with values $\{0, -\infty\}$ used to **prevent attention** to certain positions.

Softmax turns $-\infty$ into **zero attention weight.**

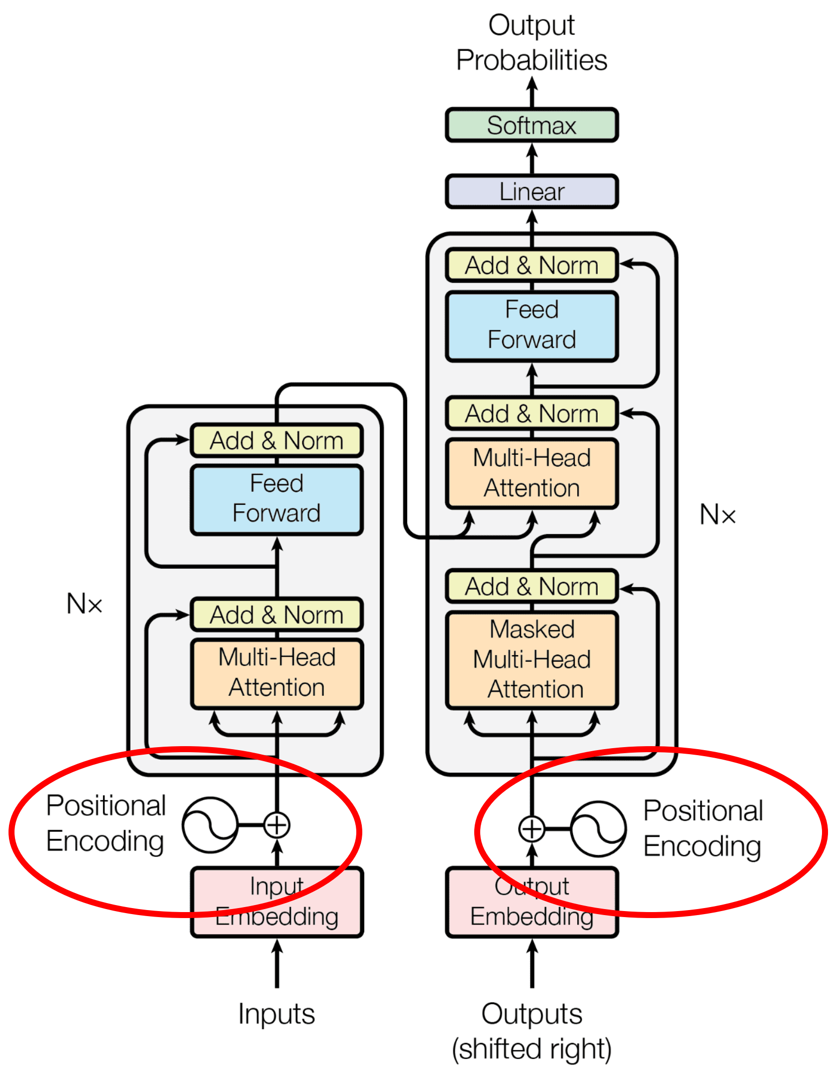
Masked Self-Attention

But why we need masked attention? How can we look in the future?

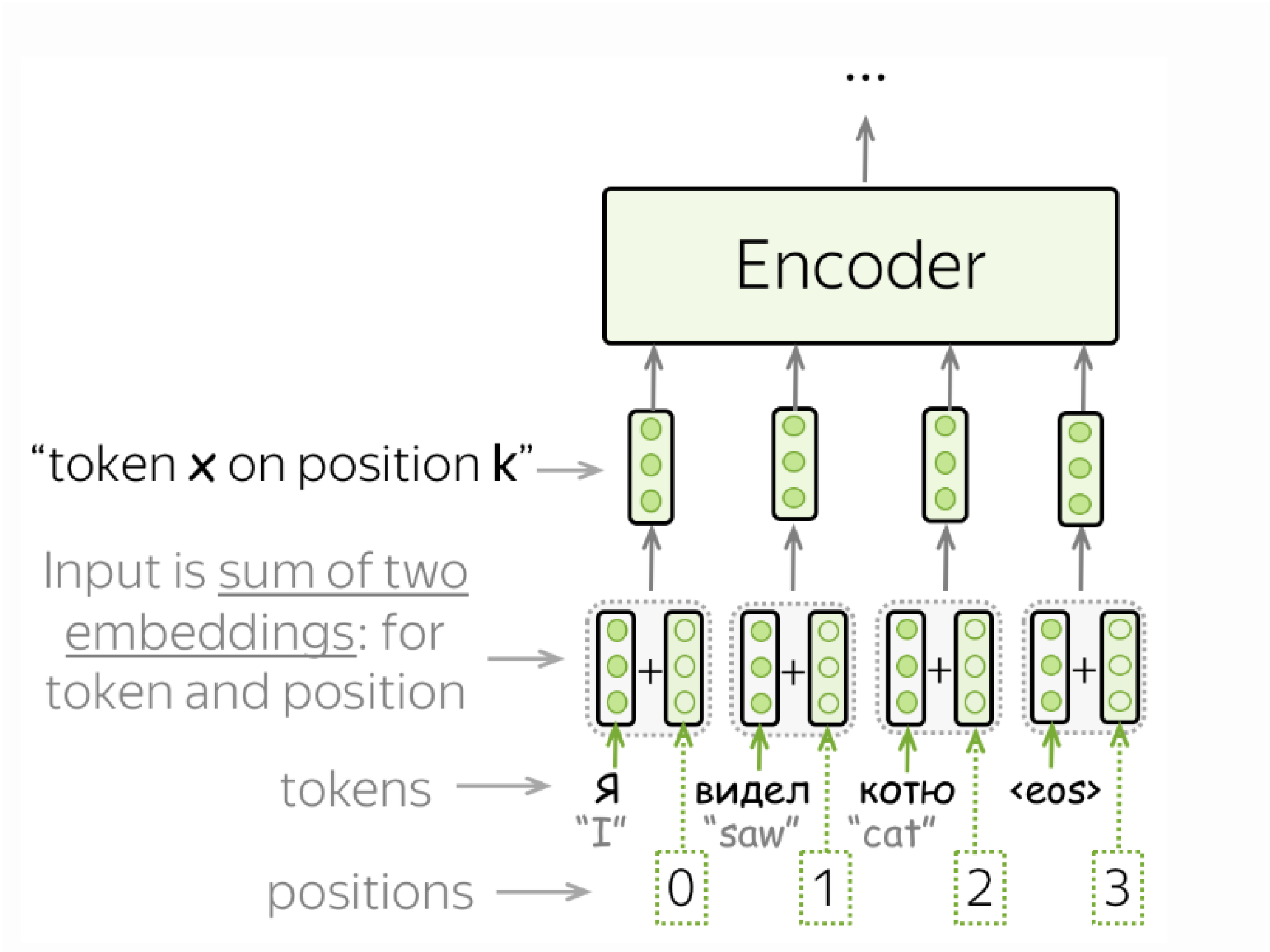
During training! Since we want to emitate a generation on a training set.

Don’t we have a simmilar problem in Encoder because all the words are considered in the same time?

Decoder


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Input
Embedding
Positional
Encoding
Output
Embedding
Inputs
Outputs
(shifted right)

Positional Encoding



Positional Encoding

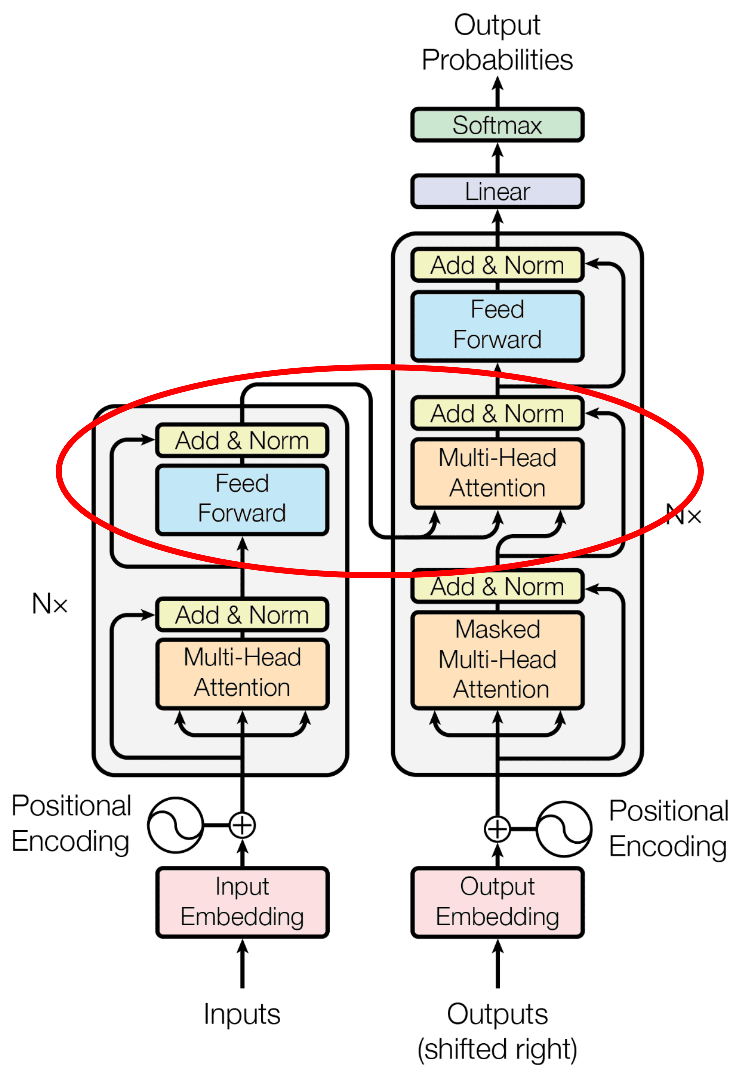
Originally used:

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

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

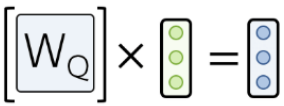
Currently in use are Rotary embeddings, and you can read about them in [this brilliant longread.]

Decoder


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)

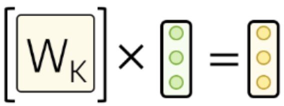
Attention

Each vector receives three representations (“roles”)



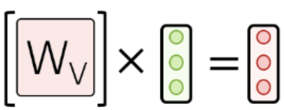
Query: vector **from** which the attention is looking

“Hey there, do you have this information?”



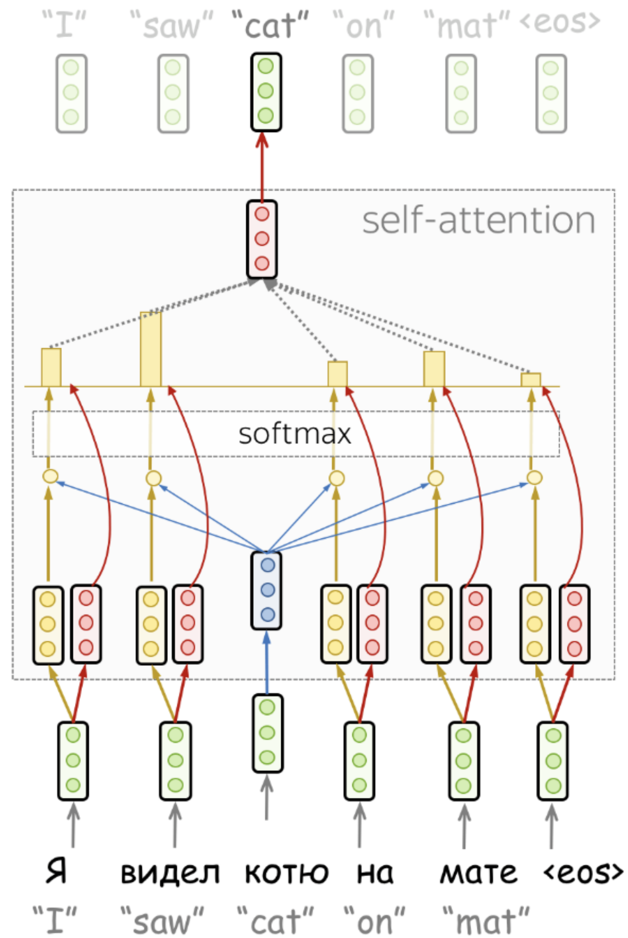
Key: vector **at** which the query looks to compute weights

“Hi, I have this information – give me a large weight!”

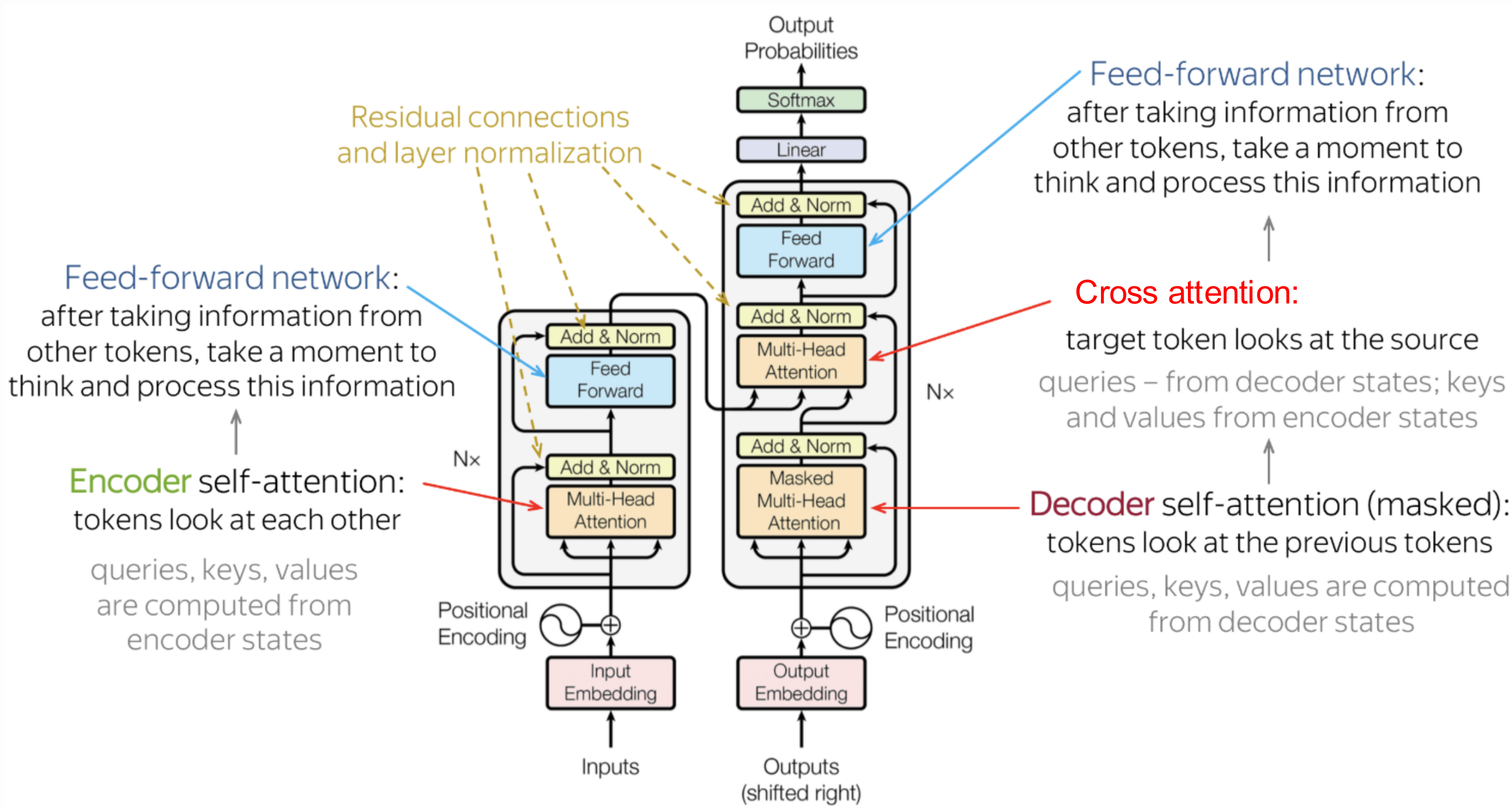


Value: their weighted sum is attention output

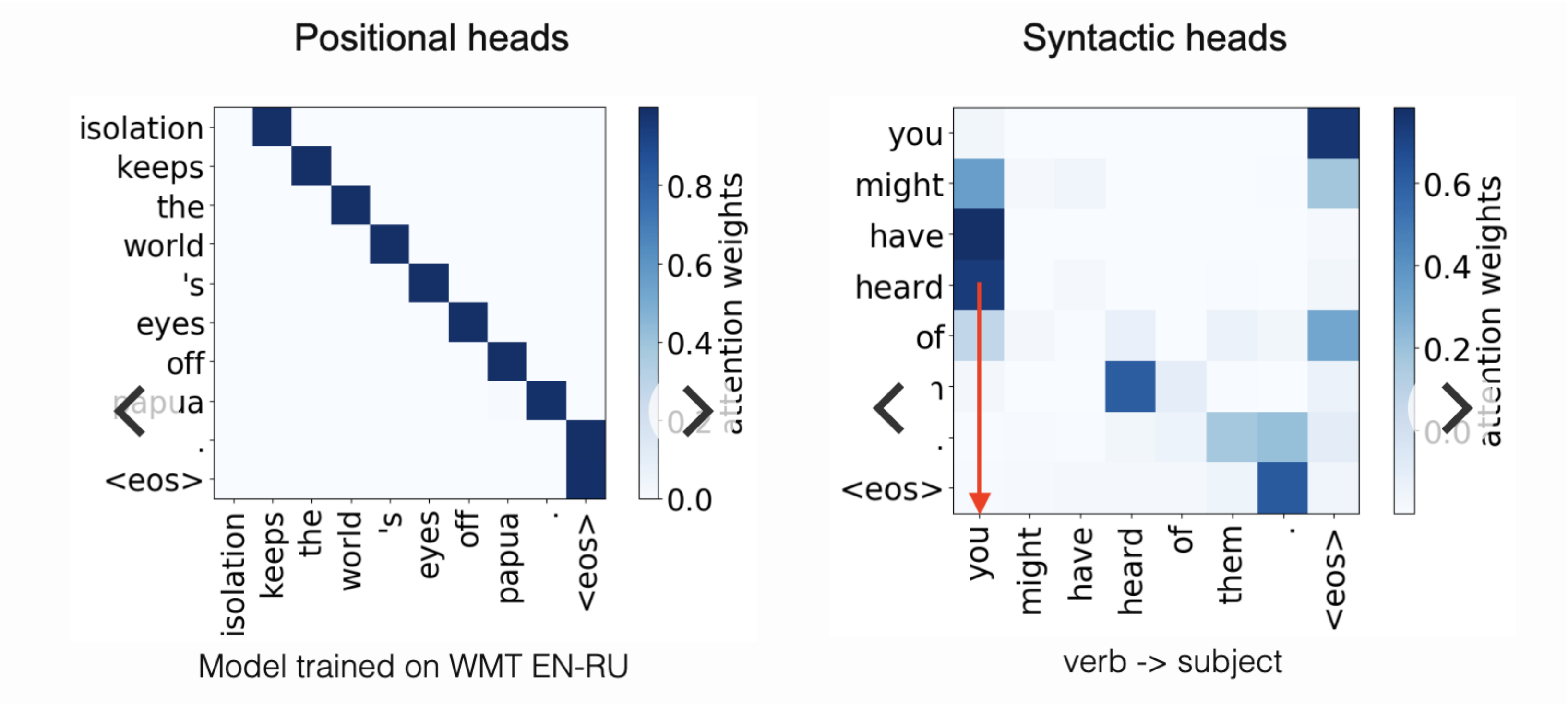
“Here's the information I have!”



Transformer



Interpretation of heads



Encoder-only vs Decoder-only Models

Encoder only

Example:
BERT(Bidirectional Encoder Representations from **Transformers**)

•Sentiment analysis
•Named entity recognition
•Question answering (extractive)
•Sentence similarity

Decoder only

Example:
GPT (Generative Pre-trained **Transformer**)

•Text generation
•Code completion
•Chatbots
•Story writing

BERT

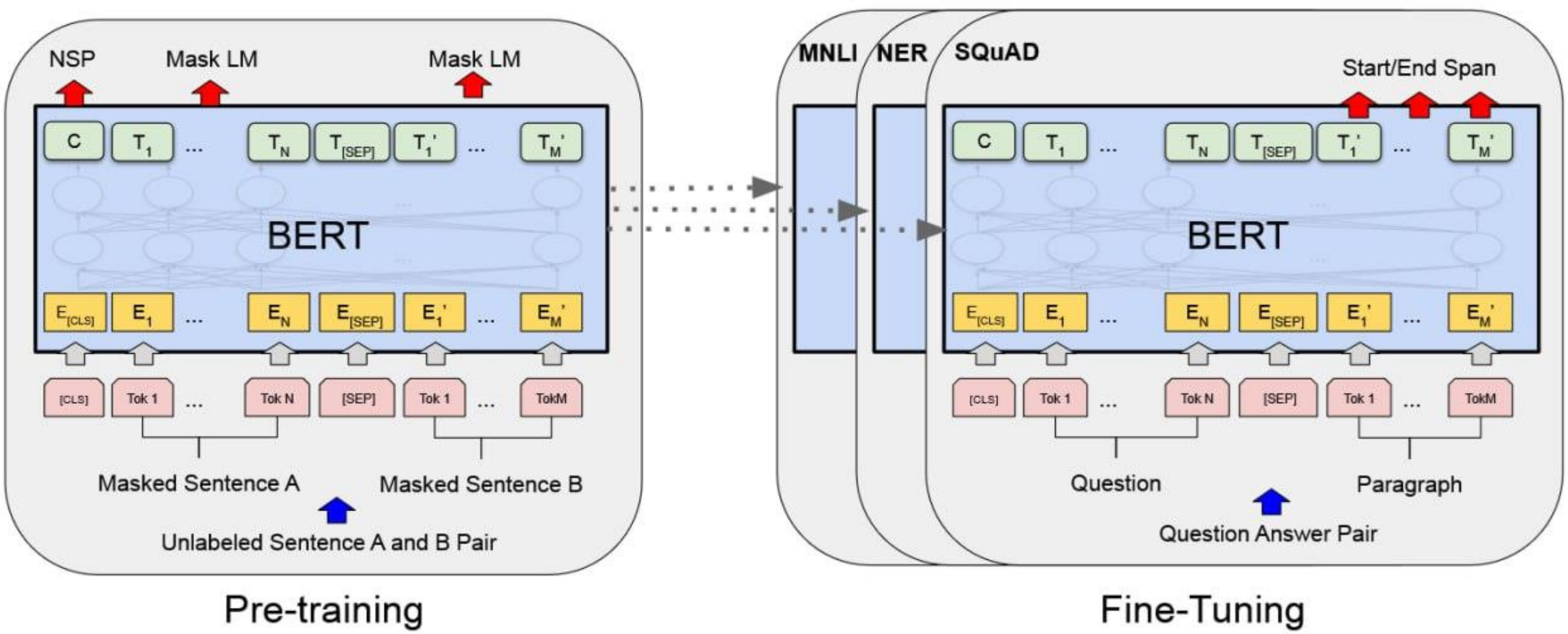


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

Lyapunov Functions

A Lyapunov function is a function associated with an ordinary differential equation (ODE):

$$\dot{x} = g(x), \quad x \in \mathbb{R}^n.$$

Definition: A function $V : \mathbb{R}^n \to \mathbb{R}$ is called a **Lyapunov function** for the system if:

- $V(x) > 0$ for all $x \neq 0$,
- $V(0) = 0$,
- $\dot{V}(x) = \langle \nabla V(x), \dot{x} \rangle \leq 0$.

Why are they important?

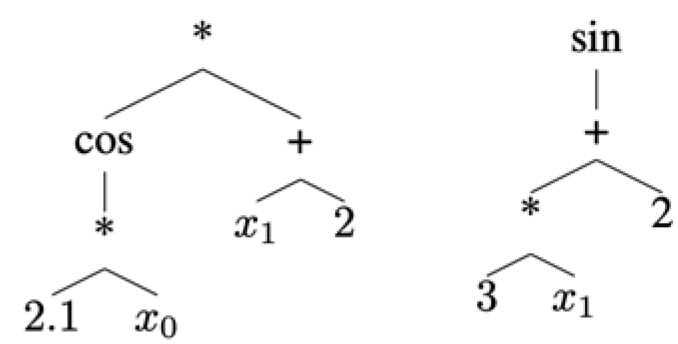
Lyapunov function $\Leftrightarrow$ Stable system

Lyapunov Functions

The problem of predicting Lyapunov function naturally states the question: How to represent functions as features or provide them as an answer?

$$\begin{cases} \dot{x}_0 = \cos(2.1x_0)(x_1+2) \\ \dot{x}_1 = \sin(3x_1+2) \end{cases}$$

is represented as



Dataset Generation

Generation of a dataset is important task:

For example for predicting the roots of polynomial, we can create a dataset of polynomials in different forms:

$$P(x) = 2x^5 - 30x^4 + 144x^3 - 240x^2 - 142x - 210$$

$$P(x) = 2(x^2 + 1)(x - 3)(x - 5)(x - 7)$$

What will be the difference in terms of teaching transformers?

Dataset Generation

Forward generation:

Generate systems and find Lyapunov function for them.

Hard to do.

Backward generation:

Start with generating a Laypunov function, generate a system with such Lyapunov function.

May reduce the problem.

Dataset Generation

For Lyapunov Functions we can use a backward approach:

1. Generate function V(x) in a generic way.
2. Create a system $\dot{x} = -\nabla V(x)$.

What are potential problems of such backward generation?

Dataset Creation

We will probably learn a different task: **integration**.
Authors address this problem by adding additional step.

1. Generate function V(x) in a generic way.

2. Create a system $\dot{x} = -\nabla V(x)$.

3. **Add noise to the system in such a way that the solution stays the same.**

Still we can not be sure that we won't solve subtask of our problem!

Results

Comparison with state-of-art:

Test sets	SOSTOOL `findlyap`	Existing AI methods Fossil 2	ANLC	LyzNet	Models PolyMixture	FBarr	FLyap	BPoly
FSOSTOOLS	-	32	30	46	**84**	80	53	54
FBarr	-	12	18	28	**89**	-	28	35
FLyap	-	42	32	66	83	**93**	-	73
BPoly	15	10	6	24	**99**	15	10	-

Table 5: **Performance comparison on different test sets**. Beam size 50. PolyMixture is BPoly + 300 FBarr.

Lyapunov functions

We train transformers with 8 layers, 10 attention heads and an embedding dimension of 640 (ablation studies on different model sizes can be found in Appendix C), on batches of 16 examples, using the Adam optimizer [Kingma and Ba, 2014] with a learning rate of 10−4, an initial linear warm-up phase of 10,000 optimization steps, and inverse square root scheduling.

Math Application

Can we train transformers to predict results of mathematical operations?

- matrix transposition: find M^T, a $n \times m$ matrix,
- matrix addition: find $M + N$, a $m \times n$ matrix,
- matrix-vector multiplication: find $M^T V$, in $\mathbb{R}^n$,
- matrix multiplication: find $M^T N$, a $n \times n$ matrix,
- eigenvalues: M symmetric, find its n (real) eigenvalues, sorted in descending order,
- eigenvectors: M symmetric, find D diagonal and Q orthogonal such that $QMQ^T = D$, set as a $(n+1) \times n$ matrix, with (sorted) eigenvalues in its first row,
- singular values: find the n eigenvalues of $M^T M$, sorted in descending order,
- singular value decomposition: find orthogonal U, V and diagonal S such that $S = UMV$, set as a $(m+n+1) \times min(m,n)$ matrix,
- inversion: M square and invertible, find its inverse P, such that $MP = PM = Id$.

It may look like an overkill, but this process can create a usefull intuition regarding subtasks: embedding, preprocessing. We will see an example on the seminar.