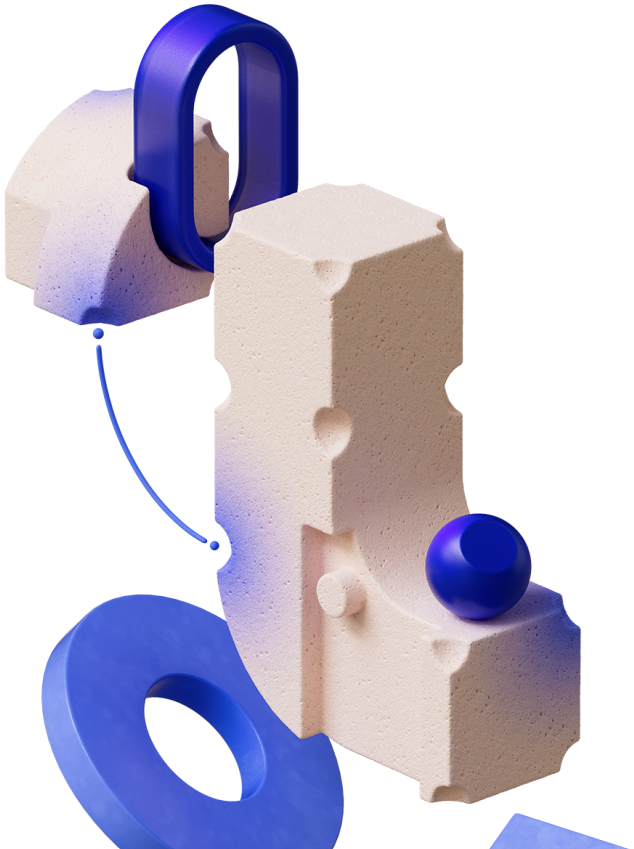
Google DeepMind



Transformers 101

Nikita Saxena
Research Engineer

Google DeepMind

What all can transformers do? Let's try them out.

Colab Notebook

tme: 7 mins

Agenda

Tokenizers and Word Embeddings

1

01 Tokenizer

Tokenization

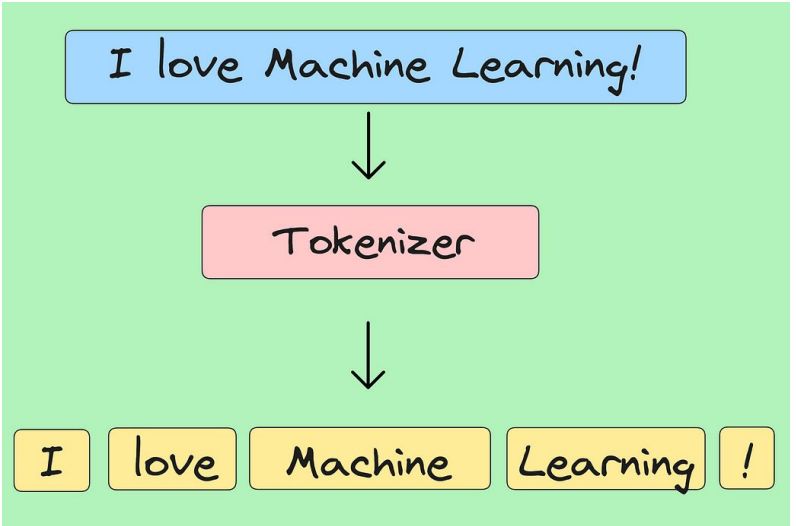
Breaking raw text into smaller units called "tokens.

Why

LLMs operate on numbers, not raw characters or words directly.

Types of Tokens

- Words (e.g., "cat", "sat", "on")
- Characters (e.g., 'c', 'a', 't')
- Sub-words (Common in LLMs) Balance between words and characters.
    - "tokenization" -> ["token", "ization"]
    - "unhappily" -> ["un", "happ", "ily"]



The output of the tokenziation is a sequence of tokens, which are then mapped to a unique integer ID from a fixed vocabulary.

Sub-word tokenization is the most popular - some algorithms are ***Byte Pair Encoding (BPE), WordPiece, SentencePiece.***

Google DeepMind

Let's try out tokenizers of different models!

Colab Notebook

tme: 5 mins

02

Word Embeddings

Word Embeddings

From Token IDs to Meaning

What?

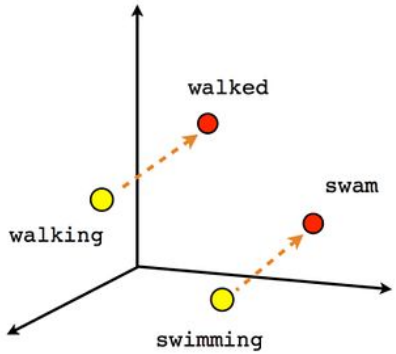
Vector representations of tokens in a *high-dimensional (e.g., 300, 768, or 4096)* space.

How?

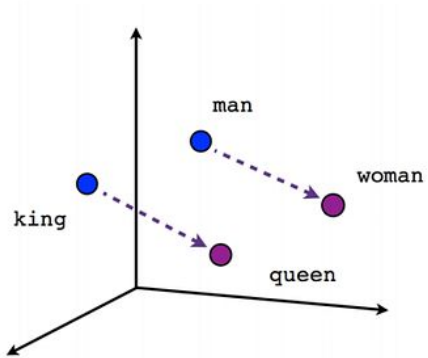
- Learned from large text corpora.
- *Classic Techniques*: Word2Vec, GloVe.
- *In Modern LLMs:* Learnt via their architecture during pretraining.

Why?

Words appearing in similar contexts tend to have similar meanings.

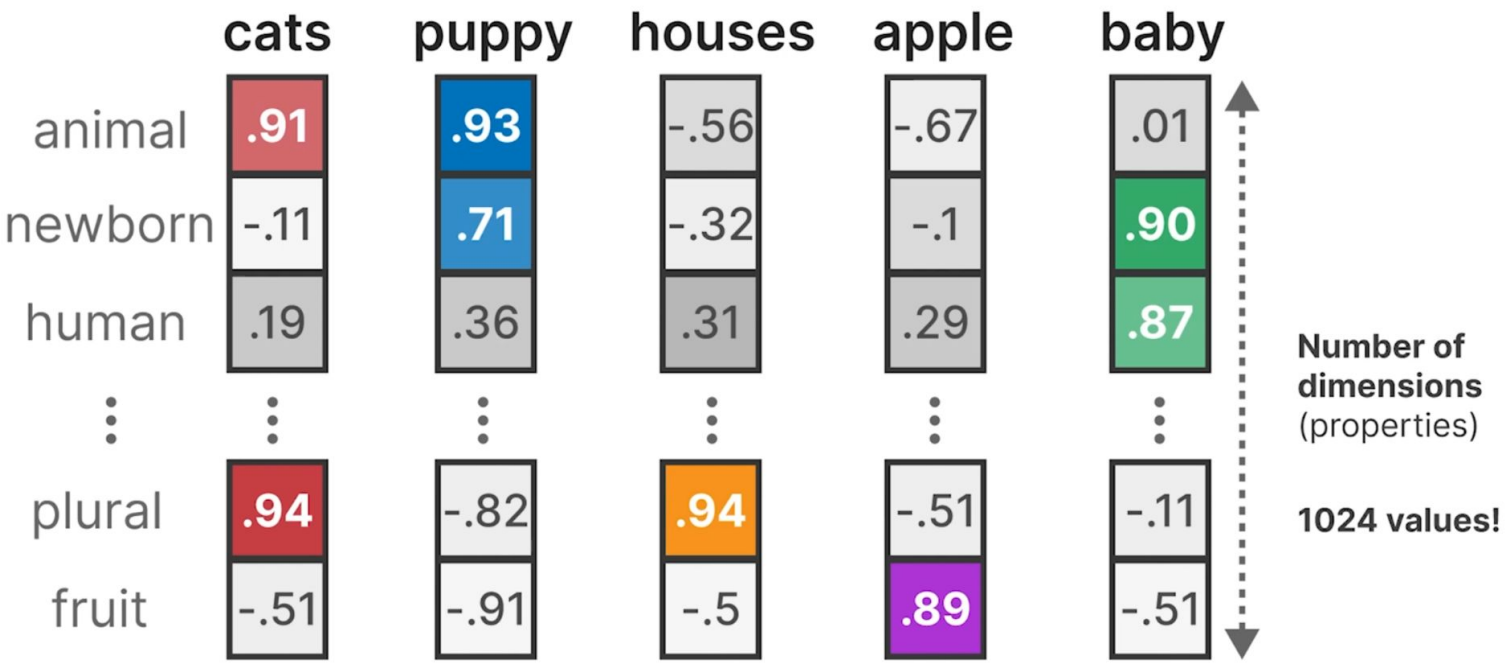


Verb tense



Male-Female

Word Embeddings



Encoder-Decoder Network

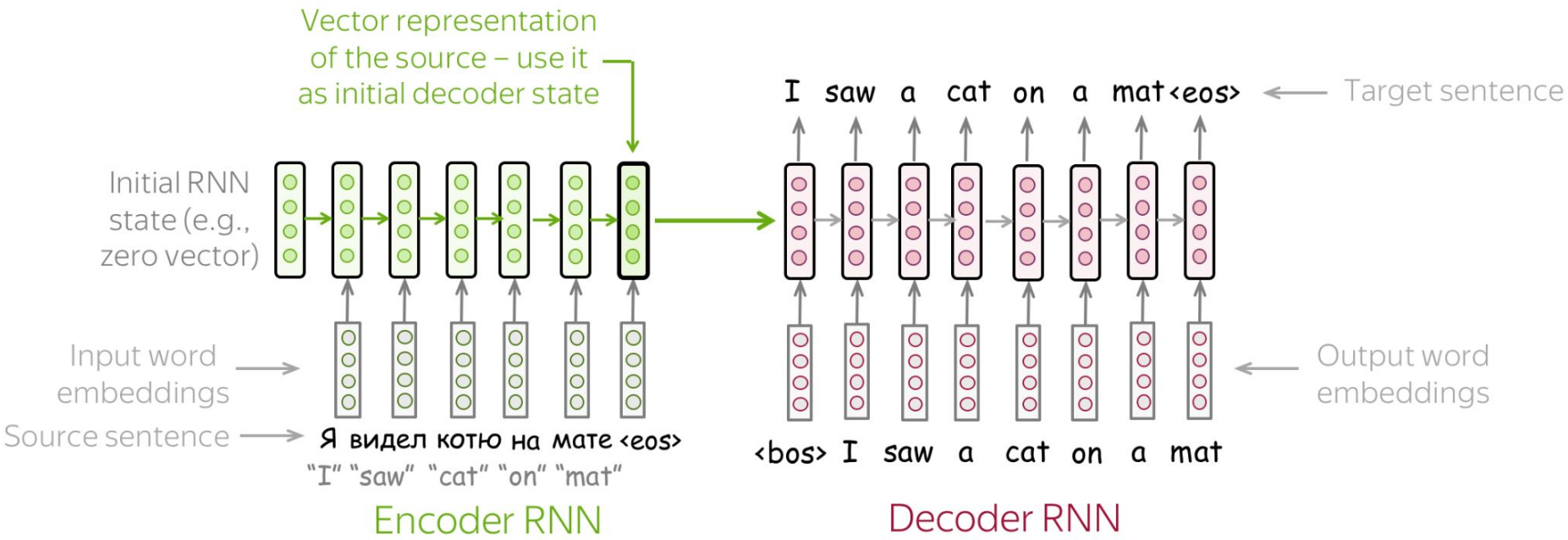
I. Simple RNN Model
II. Training Objective
III. The Bottleneck

2

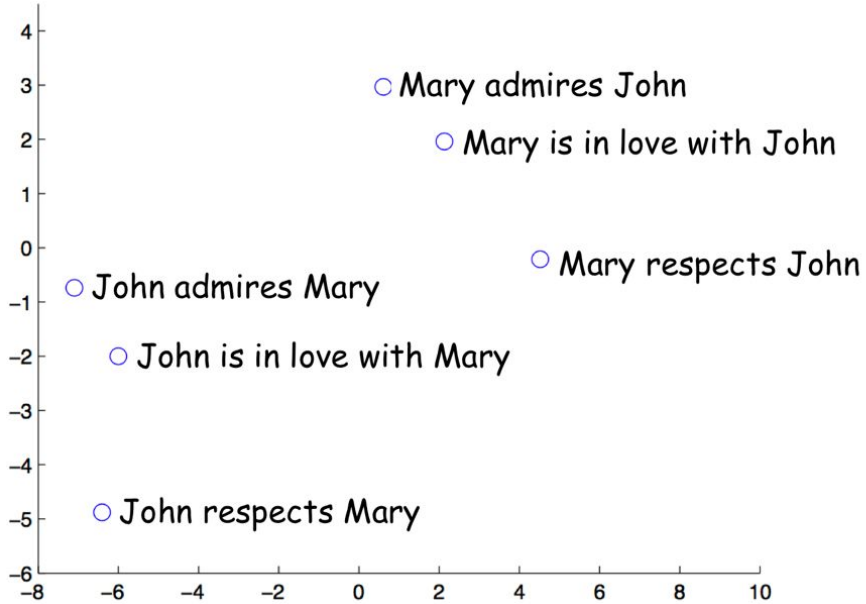
01

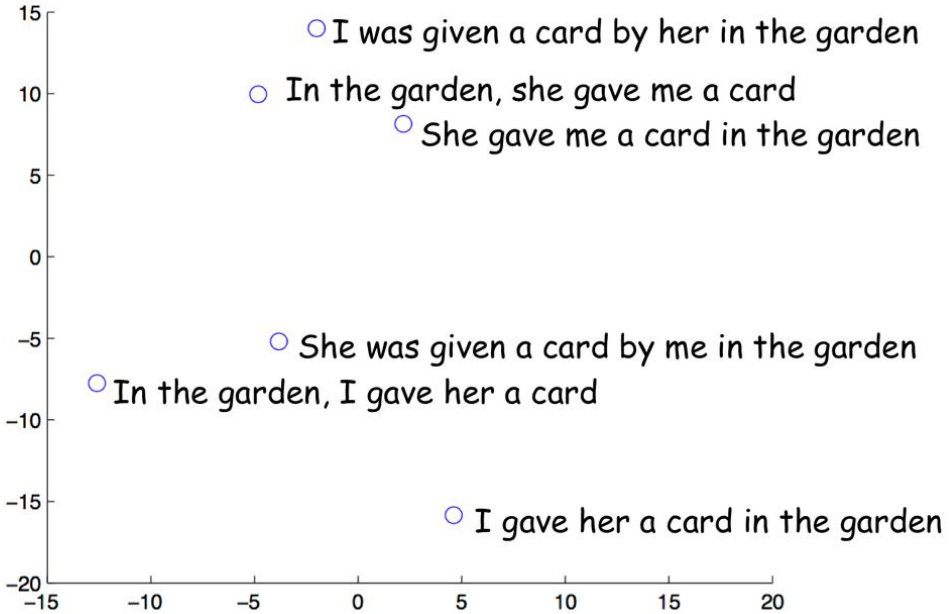
Simple RNN Model

Simplest Model: Two RNNs for Encoder and Decoder



Simplest Model: Two RNNs for Encoder and Decoder





02

Training Objective

Training Objective: Cross Entropy Loss

Formally, let's assume we have a training instance with the source $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$ and the target $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$. Then at the timestep $\boldsymbol{t}$, a model predicts a probability distribution $\boldsymbol{p}^{(t)} = \boldsymbol{p}(*|\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{t-1}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$. The target at this step is $\boldsymbol{p}^* = \textbf{one-hot}(\boldsymbol{y}_t)$, i.e., we want a model to assign probability 1 to the correct token, $\boldsymbol{y}_t$, and zero to the rest.

The standard loss function is the **cross-entropy loss**. Cross-entropy loss for the target distribution $\boldsymbol{p}^*$ and the predicted distribution $\boldsymbol{p}$ is

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

Since only one of $\boldsymbol{p}_i^*$ is non-zero (for the correct token $\boldsymbol{y}_t$), we will get

$$\boldsymbol{Loss}(p^*, p) = -\log(p_{y_t}) = -\log(p(y_t|y_{<t}, x)).$$

At each step, we maximize the probability a model assigns to the correct token.

Source sequence:

Я видел котю на мате <eos>

"I" "saw" "cat" "on" "mat"

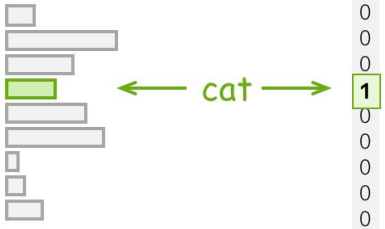
Target sequence: ← one training example

I saw a cat on a mat <eos> ← one step for this example

previous tokens

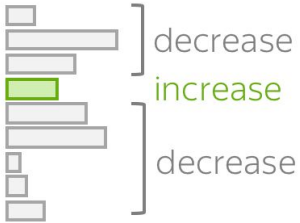
we want the model to predict this

Model prediction: p(∗ |I saw a, Я ... <eos>)

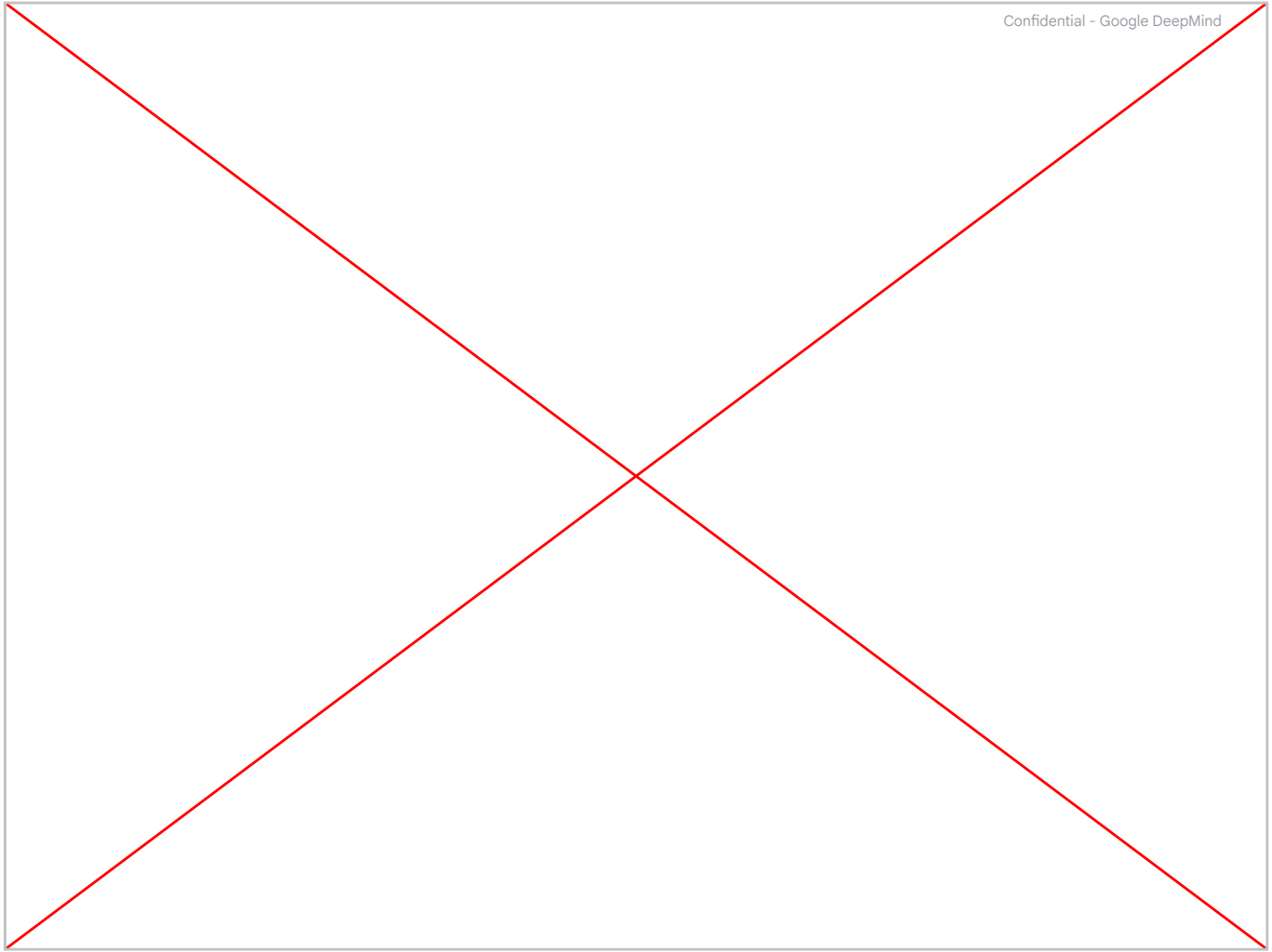


Target

Loss = -log (p(cat)) → min



Illustrating the Training Process

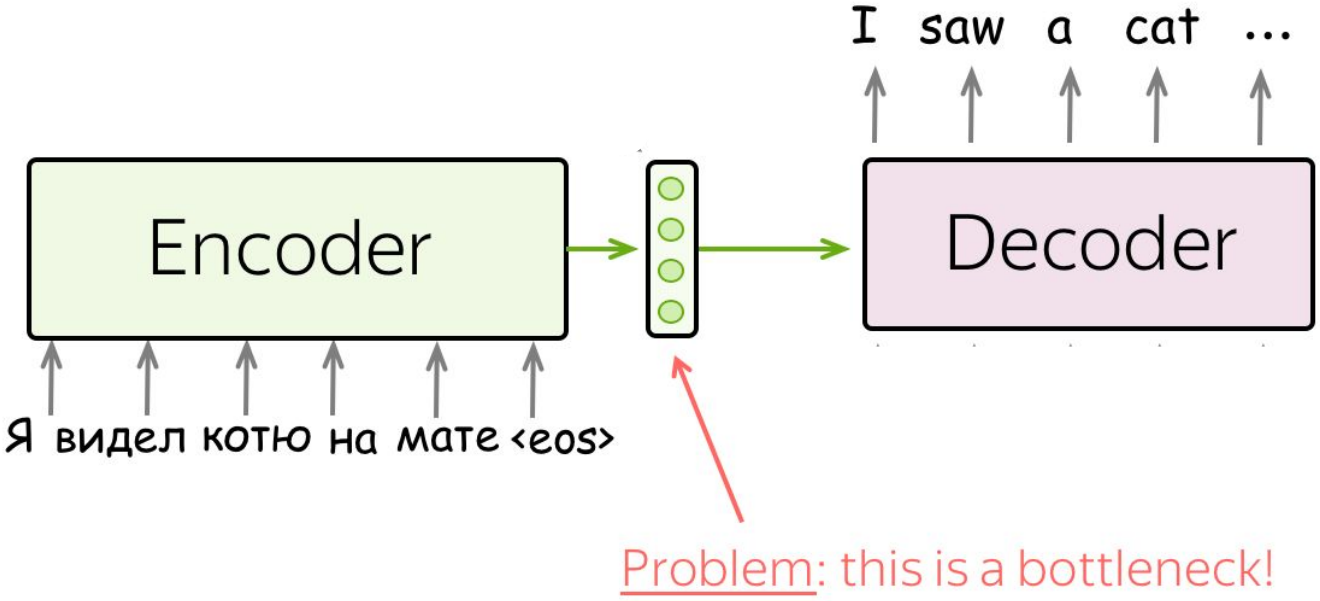


03

The Bottleneck

Imagine the whole universe - try to visualize everything you can find there and how you can describe it in words.

Then imagine all of it is compressed into a single vector of size e.g. 512.
Do you feel that the universe is still ok?

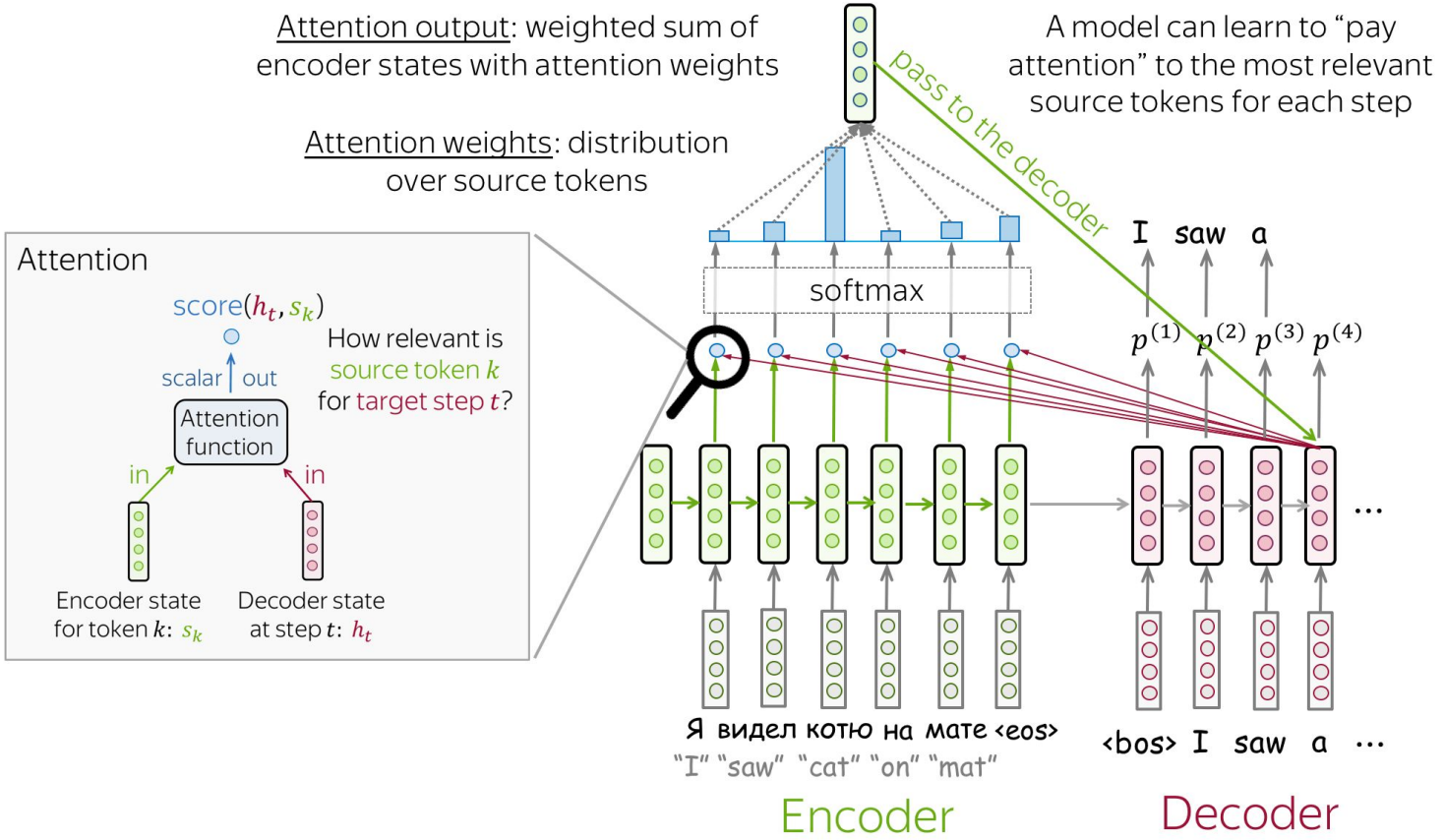


Attention

I. Overview
II. Step by Step Visualization
III. Bahdanau Attention
IV. Attention and Alignment
V. Visualization

01 Overview

Overview



Overview

Attention output

$$c^{(t)} = a_1^{(t)} s_1 + a_2^{(t)} s_2 + \cdots + a_m^{(t)} s_m = \sum_{k=1}^{m} a_k^{(t)} s_k$$

"source context for decoder step t"

(weighted sum)

Attention weights

$$a_k^{(t)} = \frac{\exp(\text{score}(h_t, s_k))}{\sum_{i=1}^{m} \exp(\text{score}(h_t, s_i))}, \text{k} = 1..\text{m}$$

"attention weight for source token k at decoder step t"

(softmax)

Attention scores

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

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

Attention input

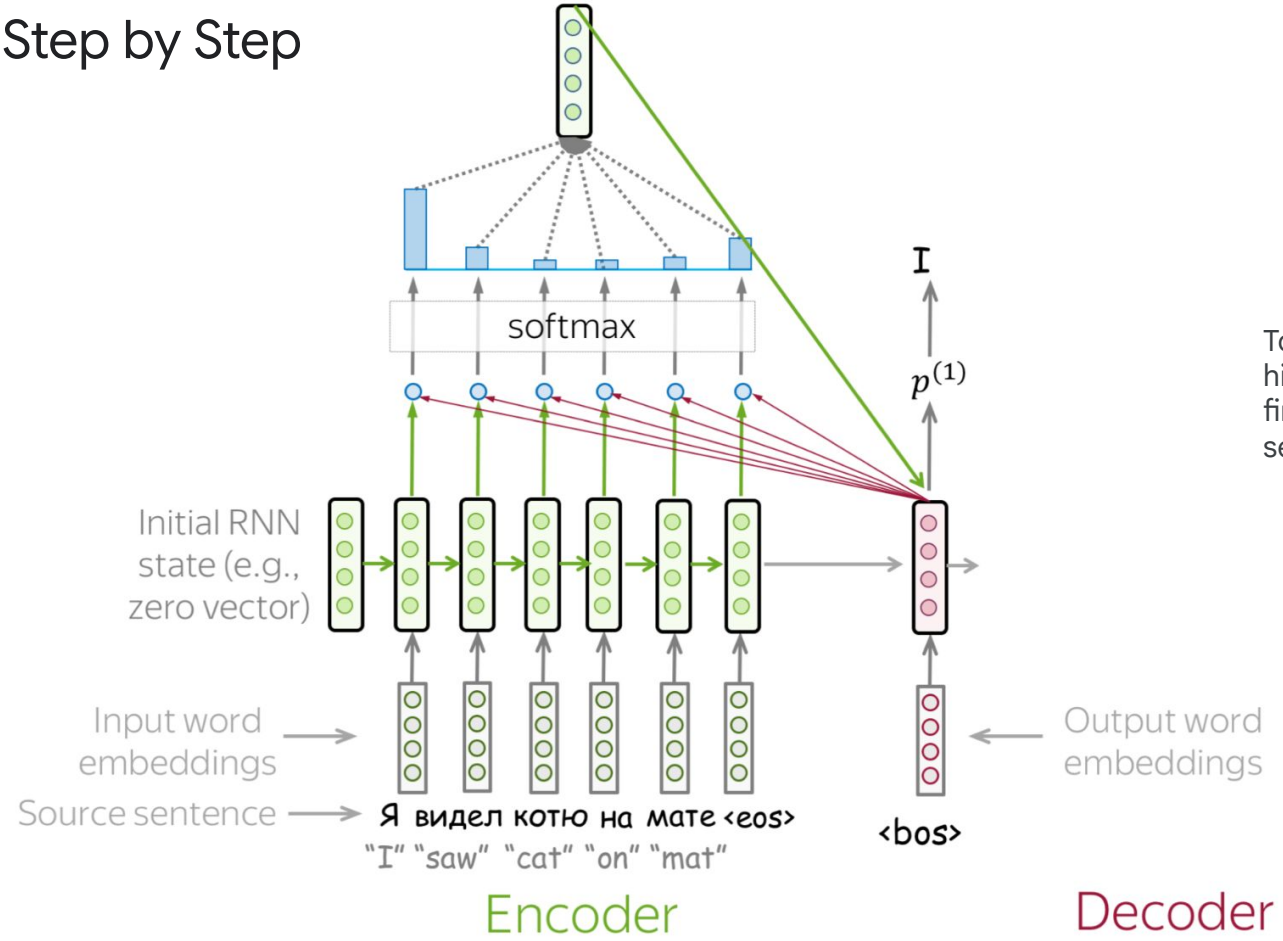
$s_1, s_2, \ldots, s_m$ all encoder states

h_t one decoder state

02

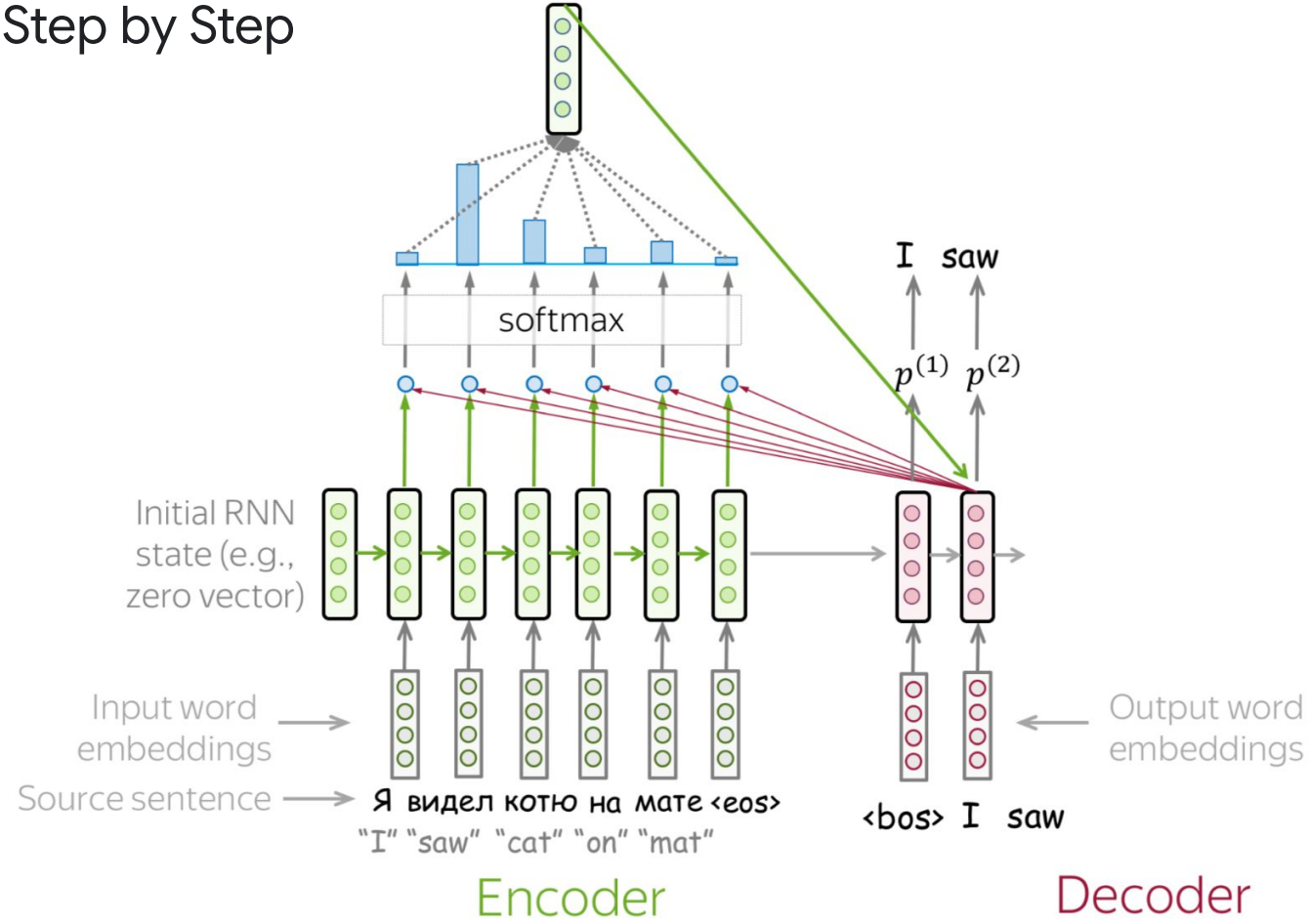
Step by Step Visualization

Step by Step

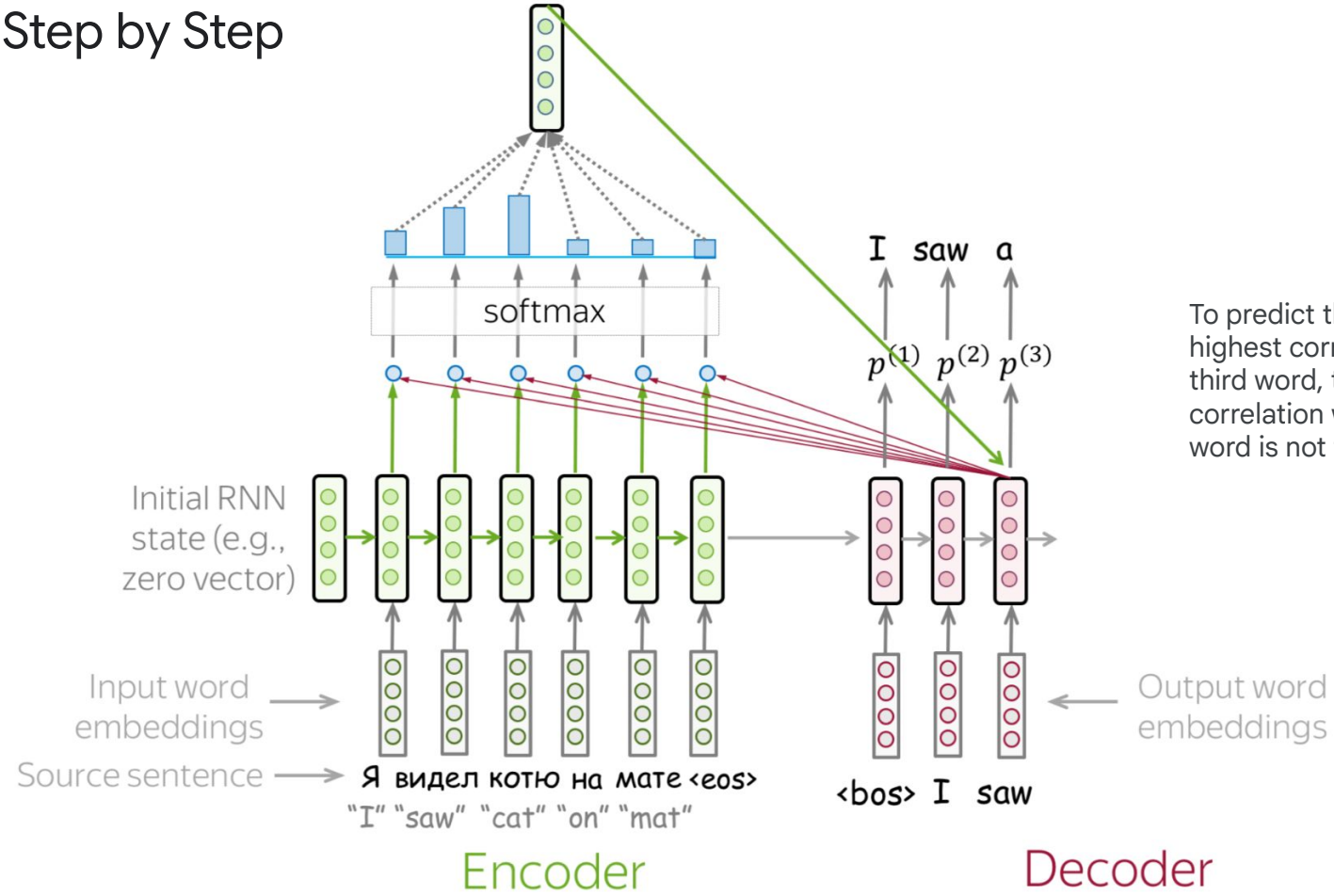


To predict the word “I” the highest correlation is with the first word in the source sentence.

Step by Step

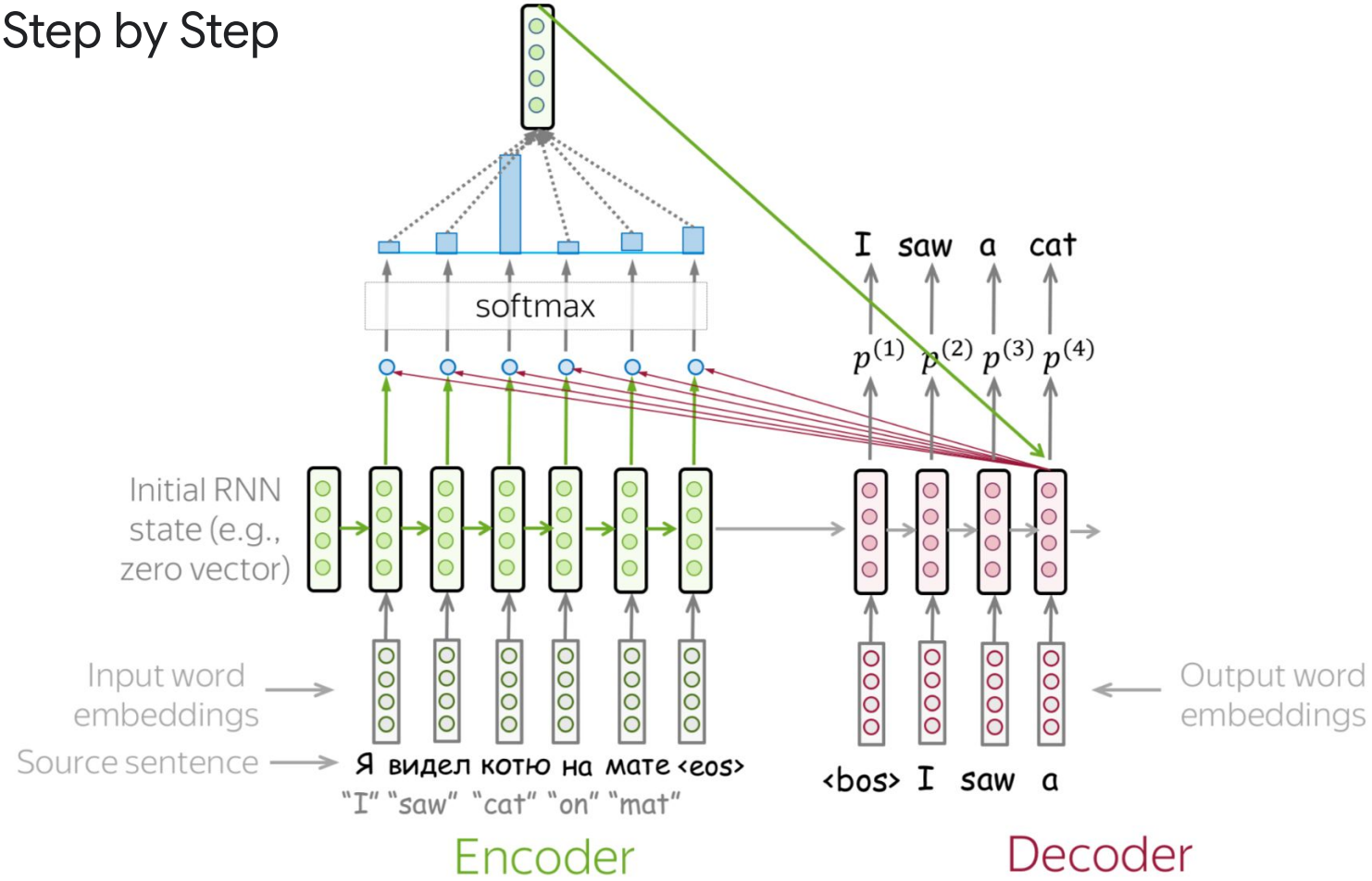


Step by Step

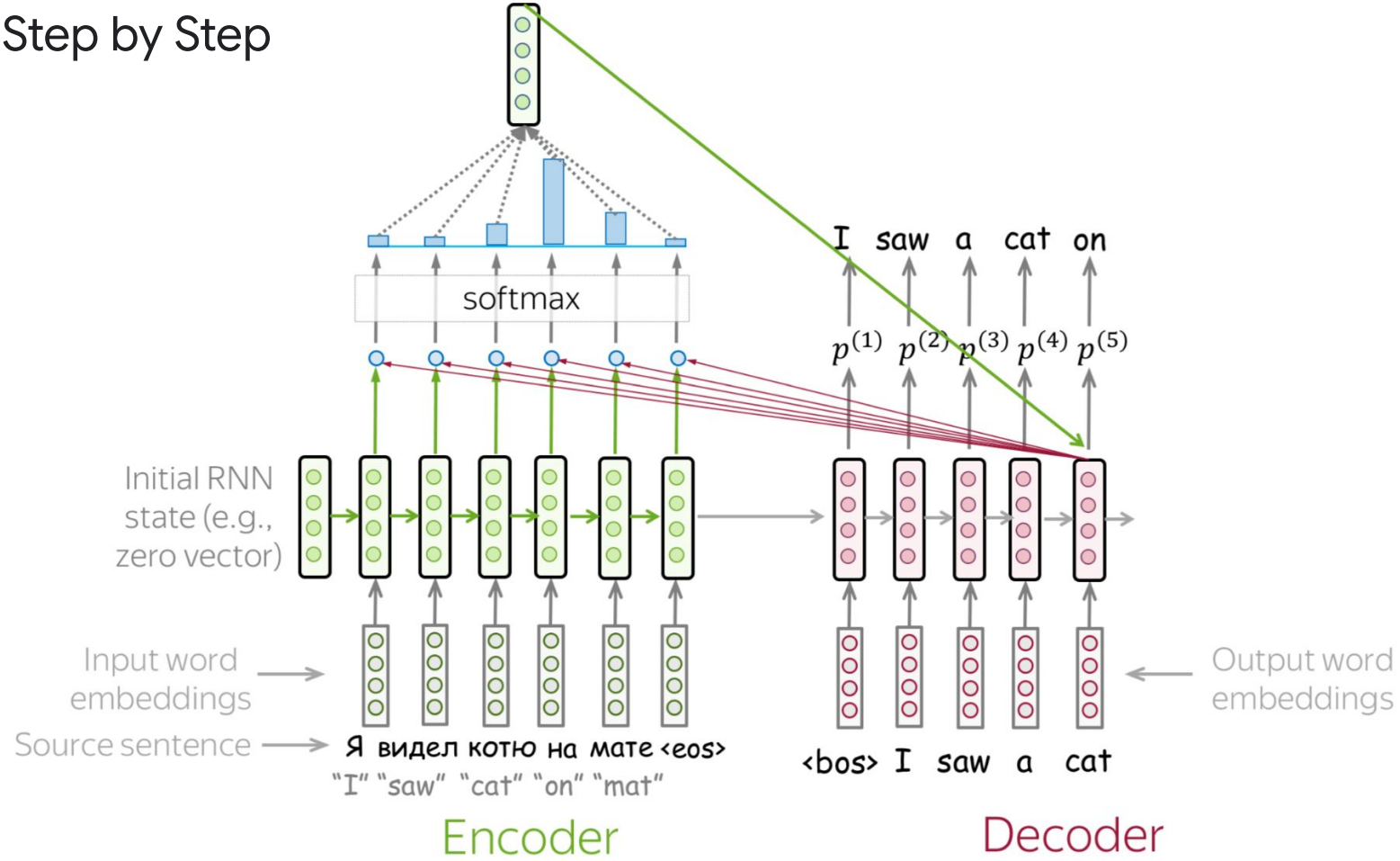


To predict the word “a” the highest correlation is with the third word, though the correlation with the second word is not too low.

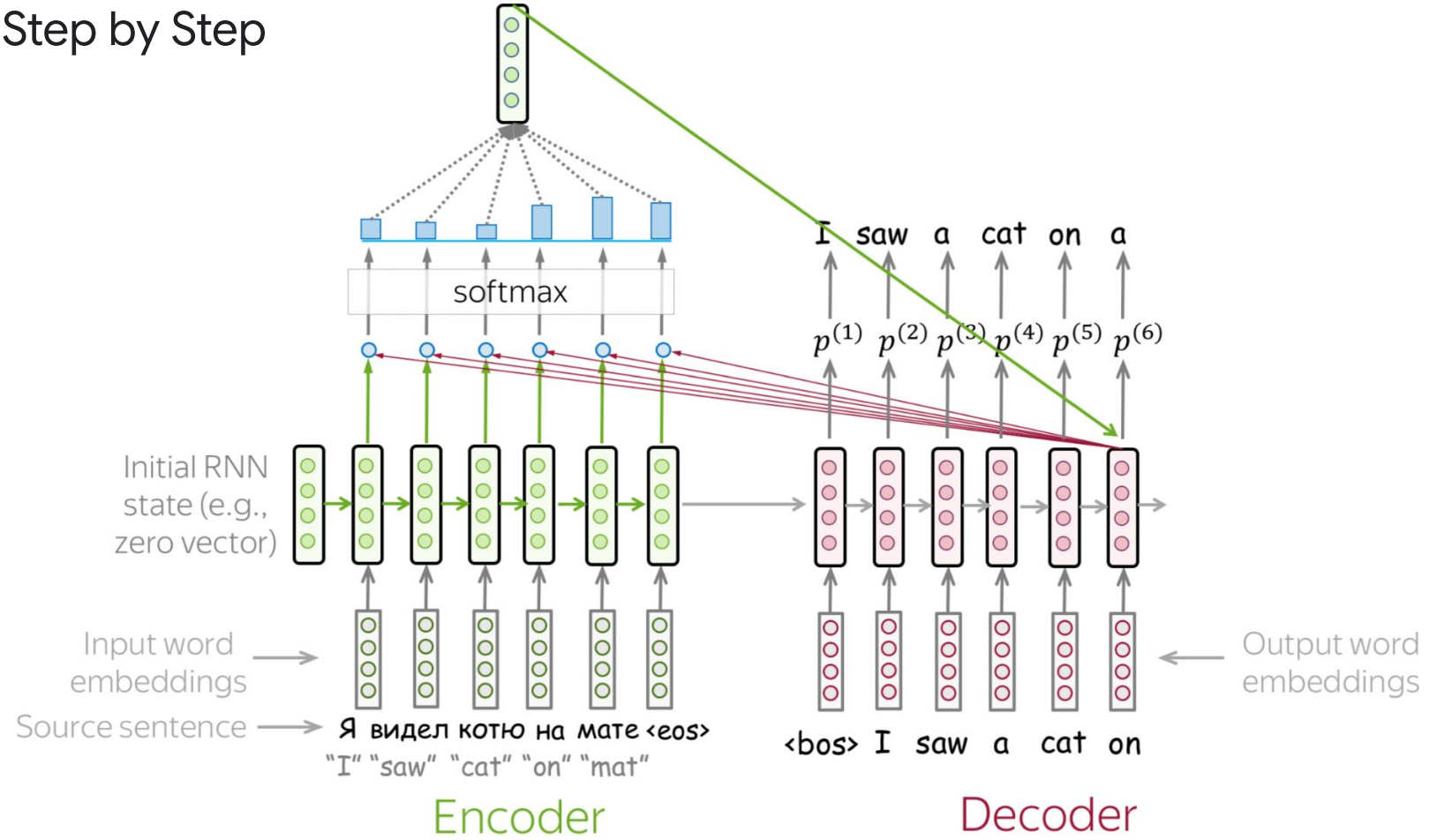
Step by Step



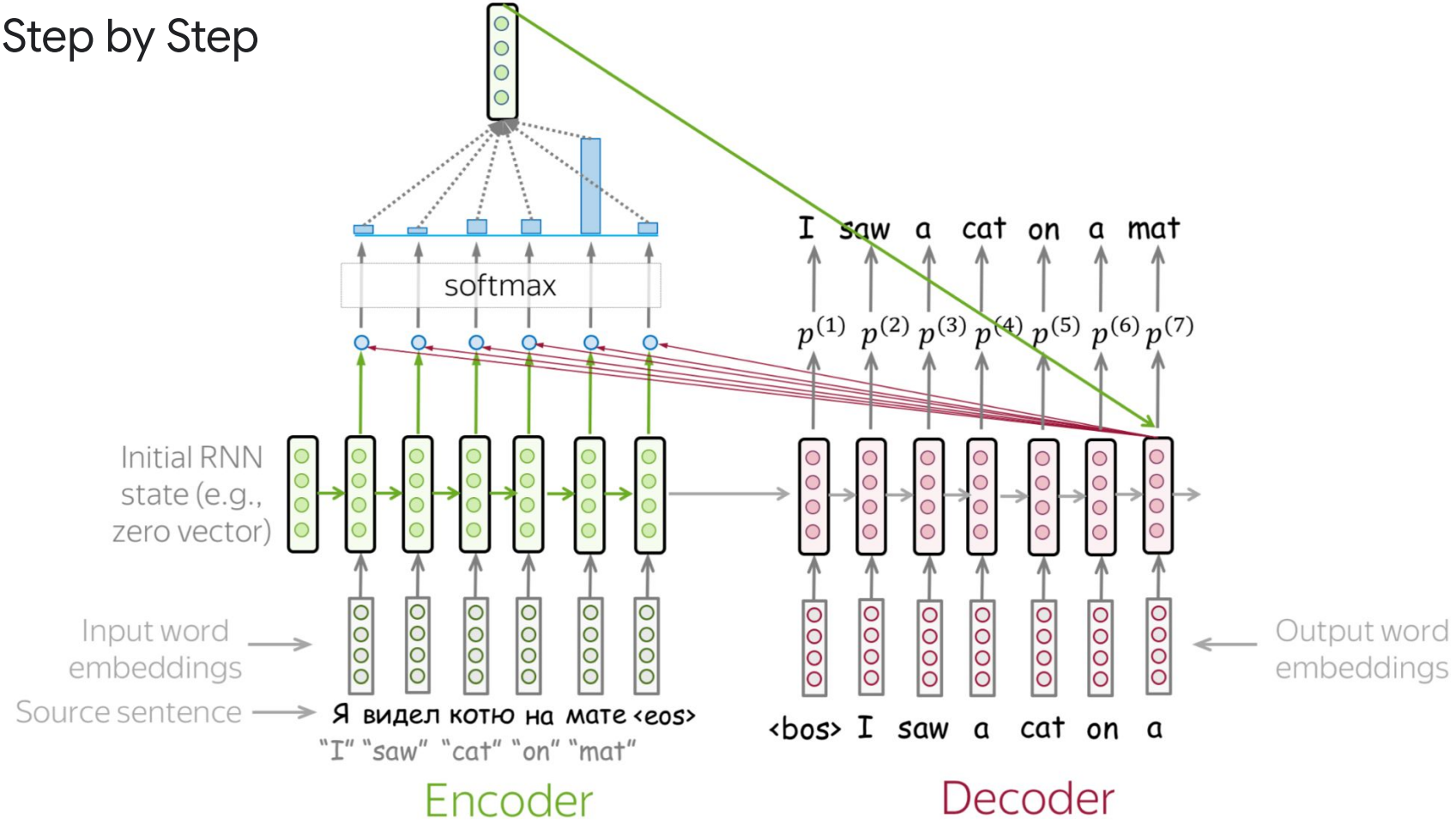
Step by Step



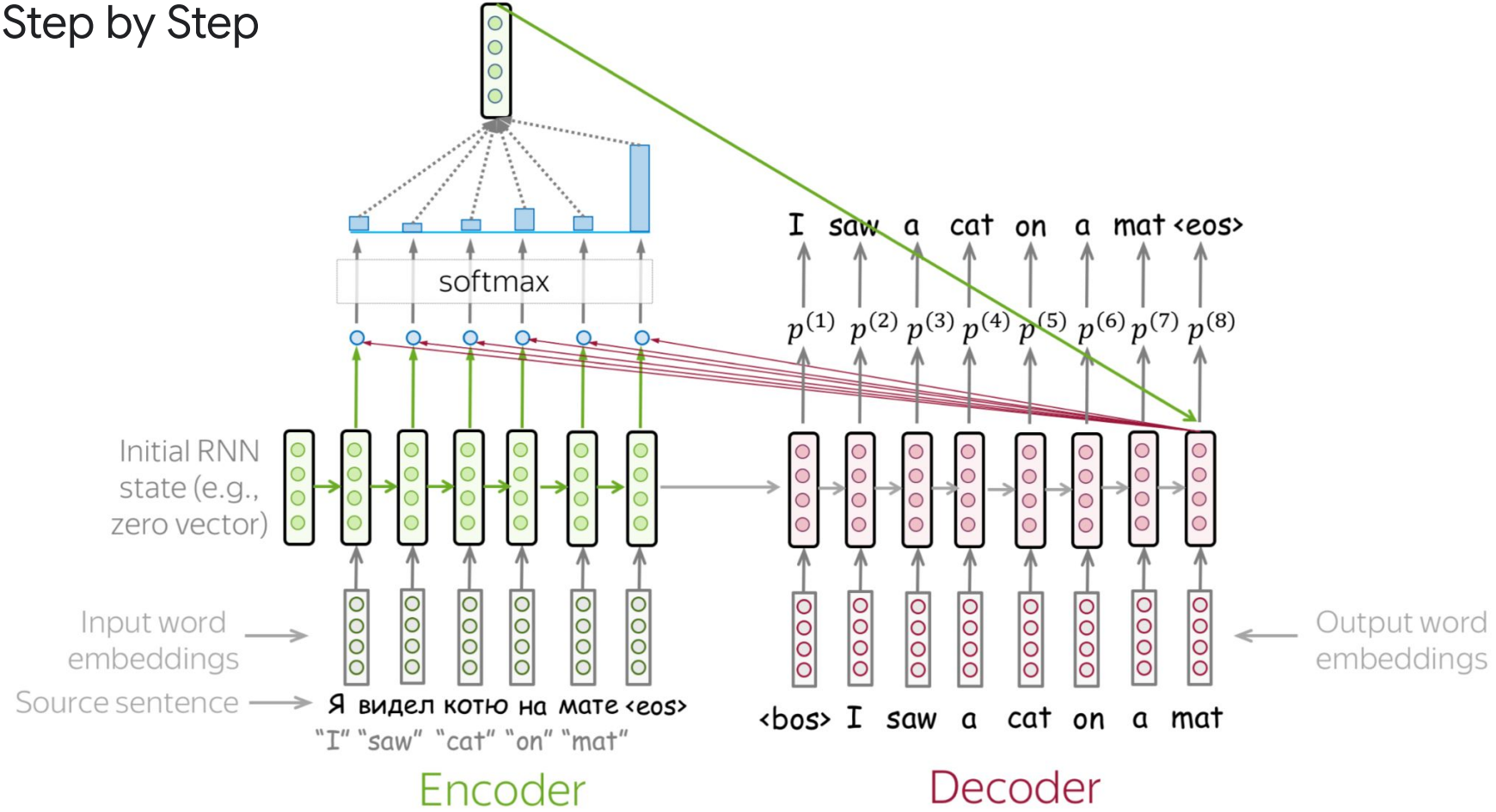
Step by Step



Step by Step



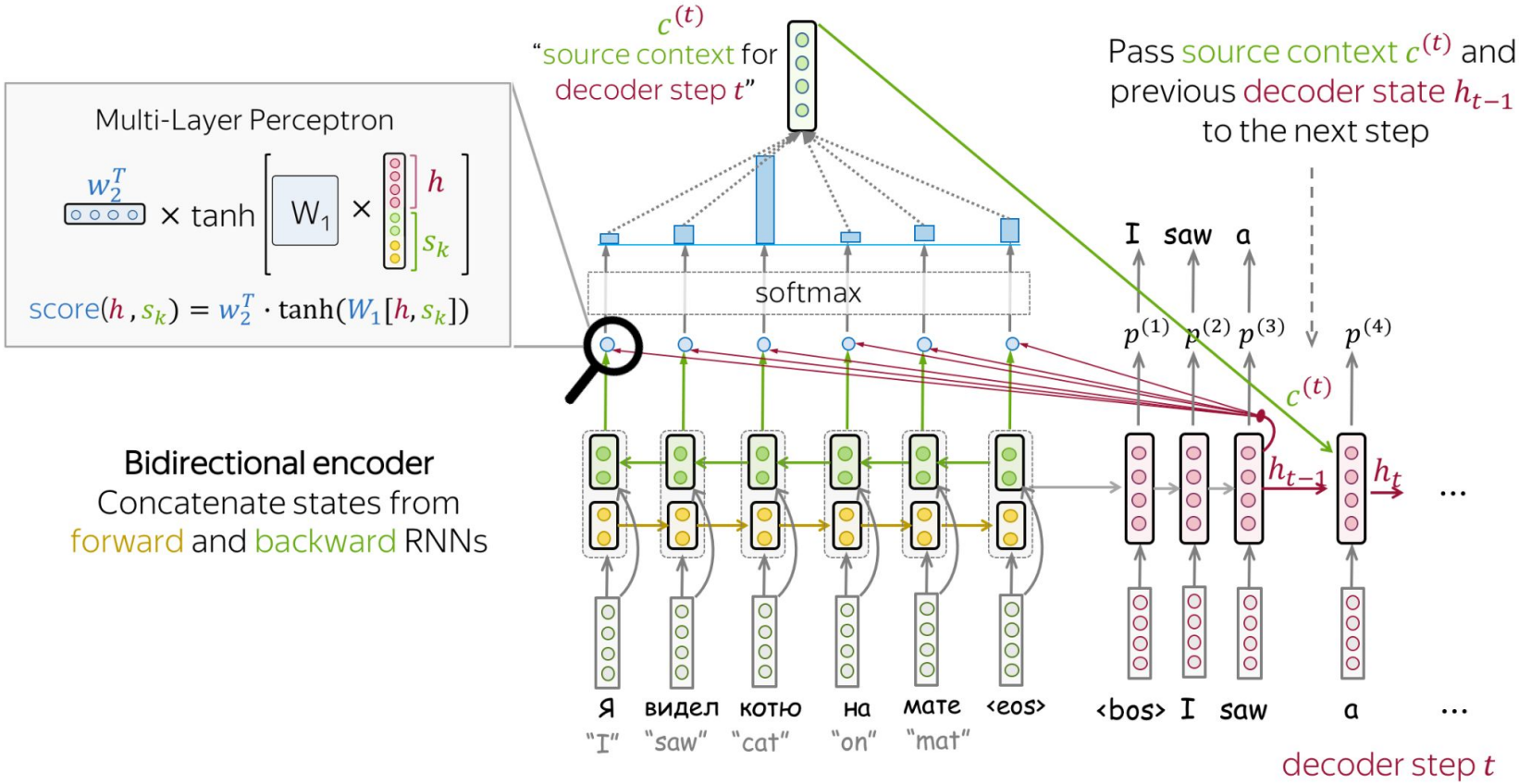
Step by Step



03

Bahdanau Attention

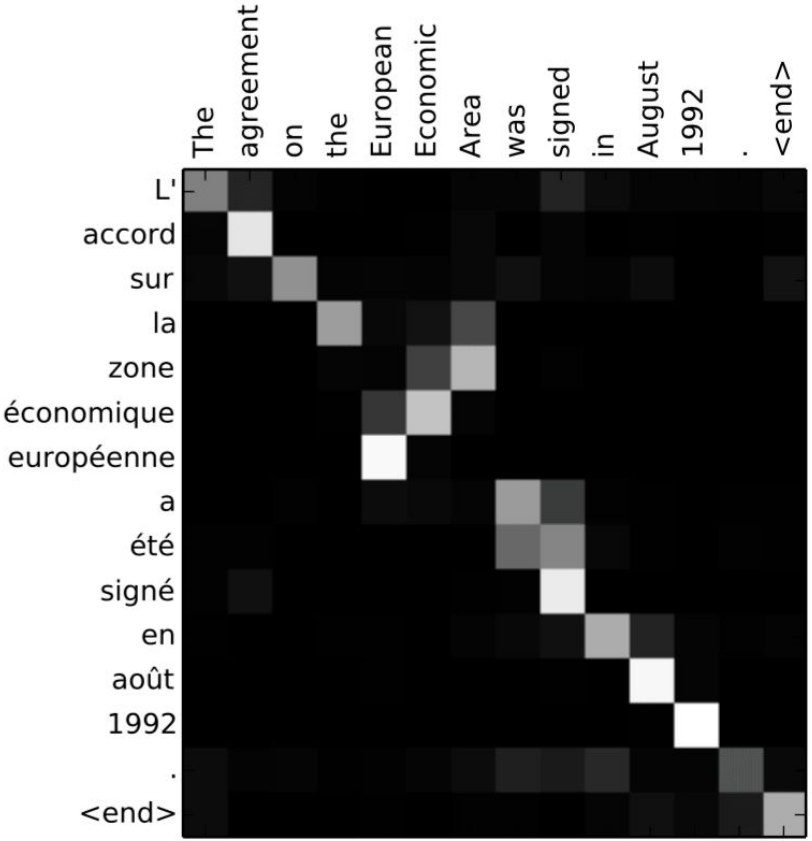
Bahdanau attention

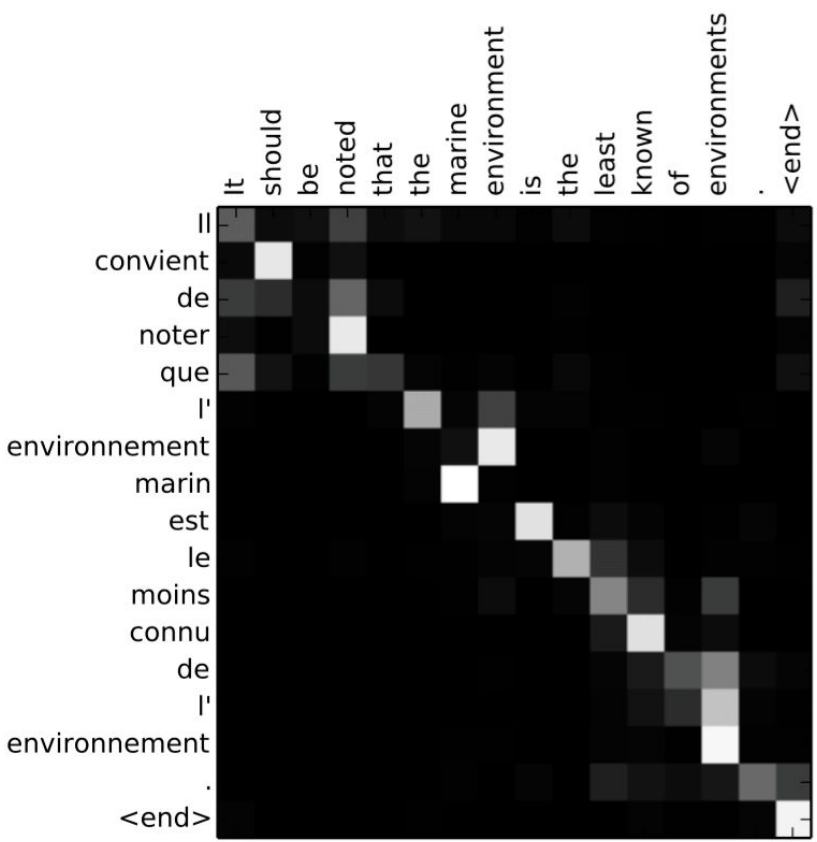


03

Attention & Alignment

Attention Learns (nearly) Alignment





Google DeepMind

Quick Visualization to Recap What we Have Learnt.

https://www.youtube.com/watch?v=eMIx5fFNoYc

Transformers

I. Overview
II. Self-Attention
III. Multi-Head Attention
IV. Model Architecture
    A. Positional Encoding
    B. Feed Forward Network
    C. Residual And LayerNorm

01 Overview

Overview

	Seq2seq without attention	Seq2seq with attention	Transformer
processing within encoder	RNN/CNN	RNN/CNN	attention
processing within decoder	RNN/CNN	RNN/CNN	attention
decoder-encoder interaction	static fixed-sized vector	attention	attention

Overview

Unfilled Circles

Initial representations (**embeddings**) for each word.

Filled Circles

New representation per word informed by the entire context (**self-attention**).

Overview

Encoder

Who is doing:

- all source tokens

What they are doing:

- look at each other
- update representations

repeat N times

Decoder

Who is doing:

- target token at the current step

What they are doing:

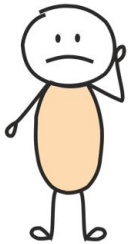
- looks at previous target tokens
- looks at source representations
- update representation

repeat N times

Overview

I arrived at the bank after crossing the ... ...street? ...river?

What does **bank** mean in this sentence?



I've no idea: let's wait until I read the end

RNNs

O(N) steps to process a sentence with length N

I don't need to wait - I see all words at once!

Transformer

Constant number of steps to process any sentence

02

Self Attention

Massa aenean et sagittis ultrices vestibulum in sed tincidunt a scelerisque bibendum a imperdiet aliquam fringilla senectus habitasse suspendisse turpis.

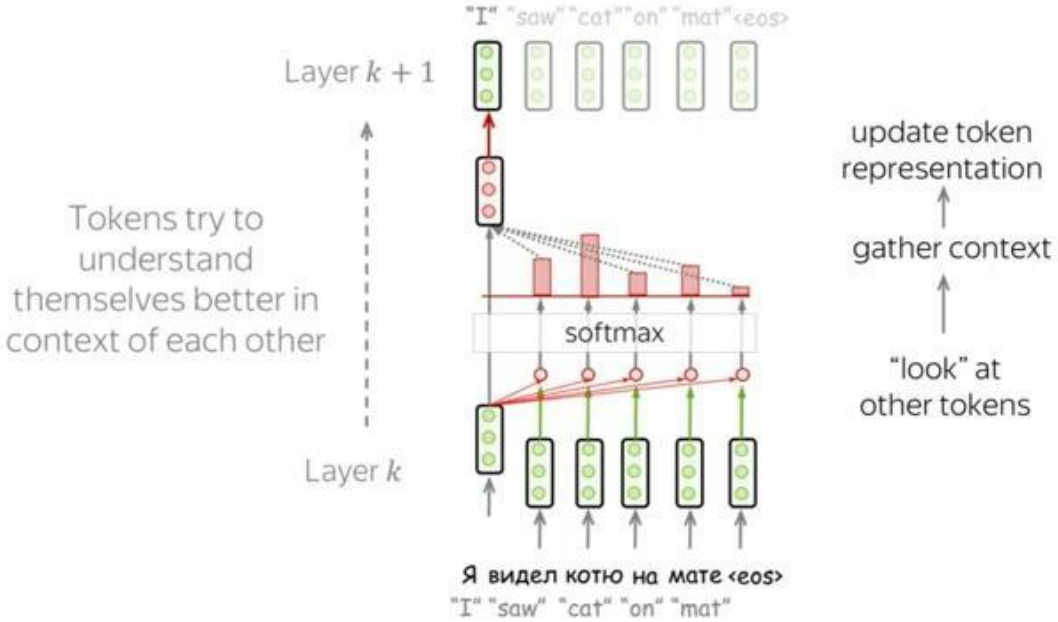
Overview

Decoder-encoder attention is looking

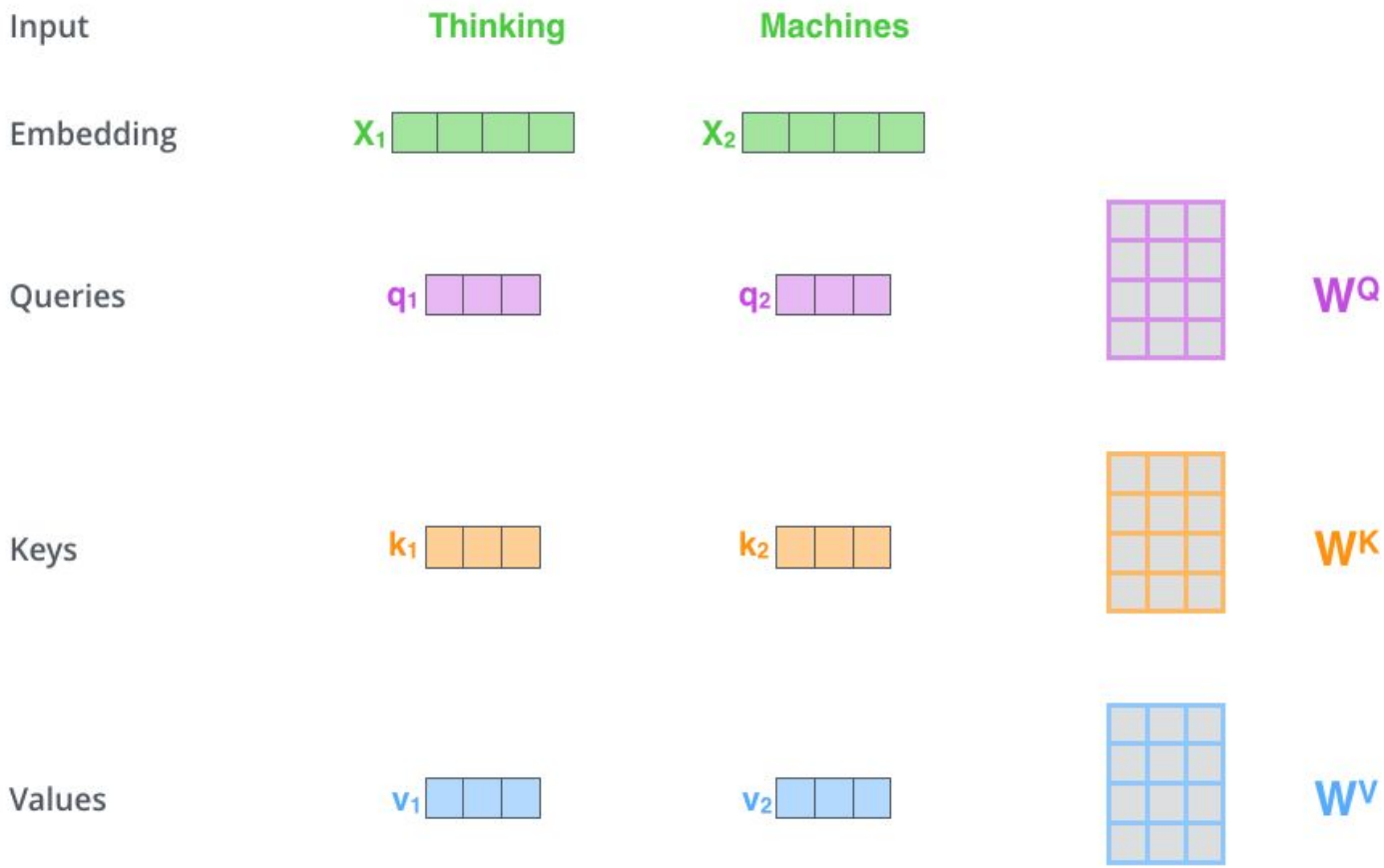
- **from**: one current decoder state
- **at**: all encoder states

Self-attention is looking

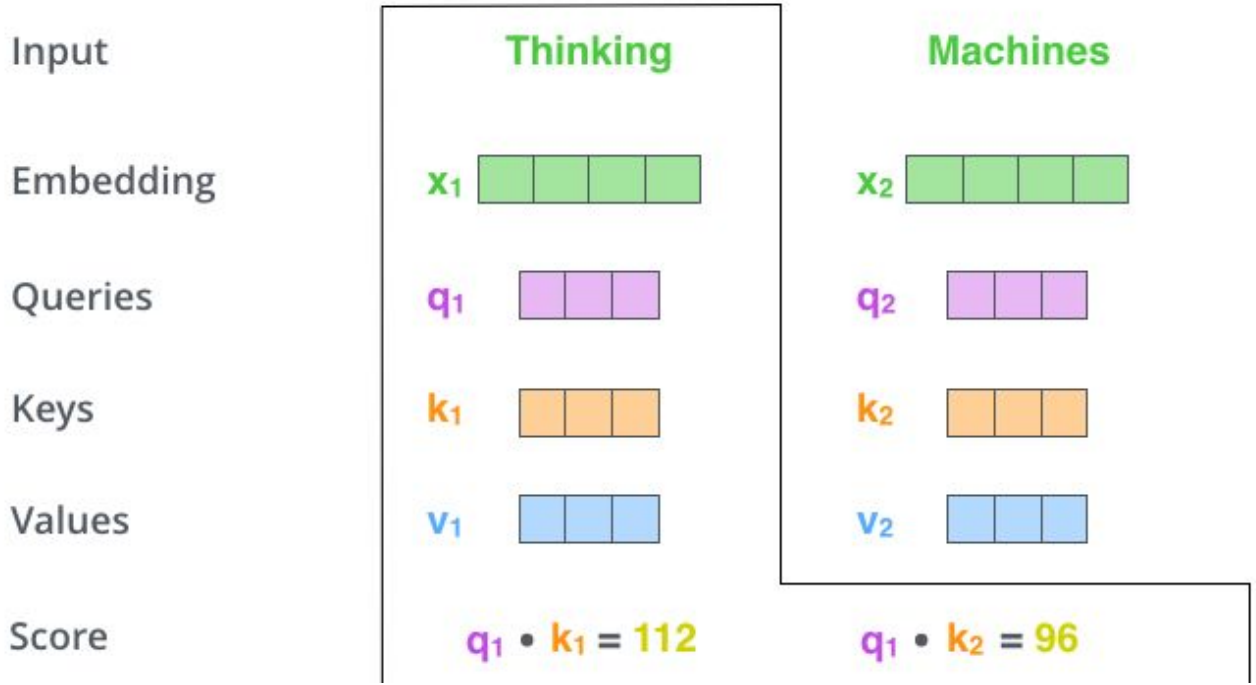
- **from**: each state from a set of states
- **at**: all other states in the same set



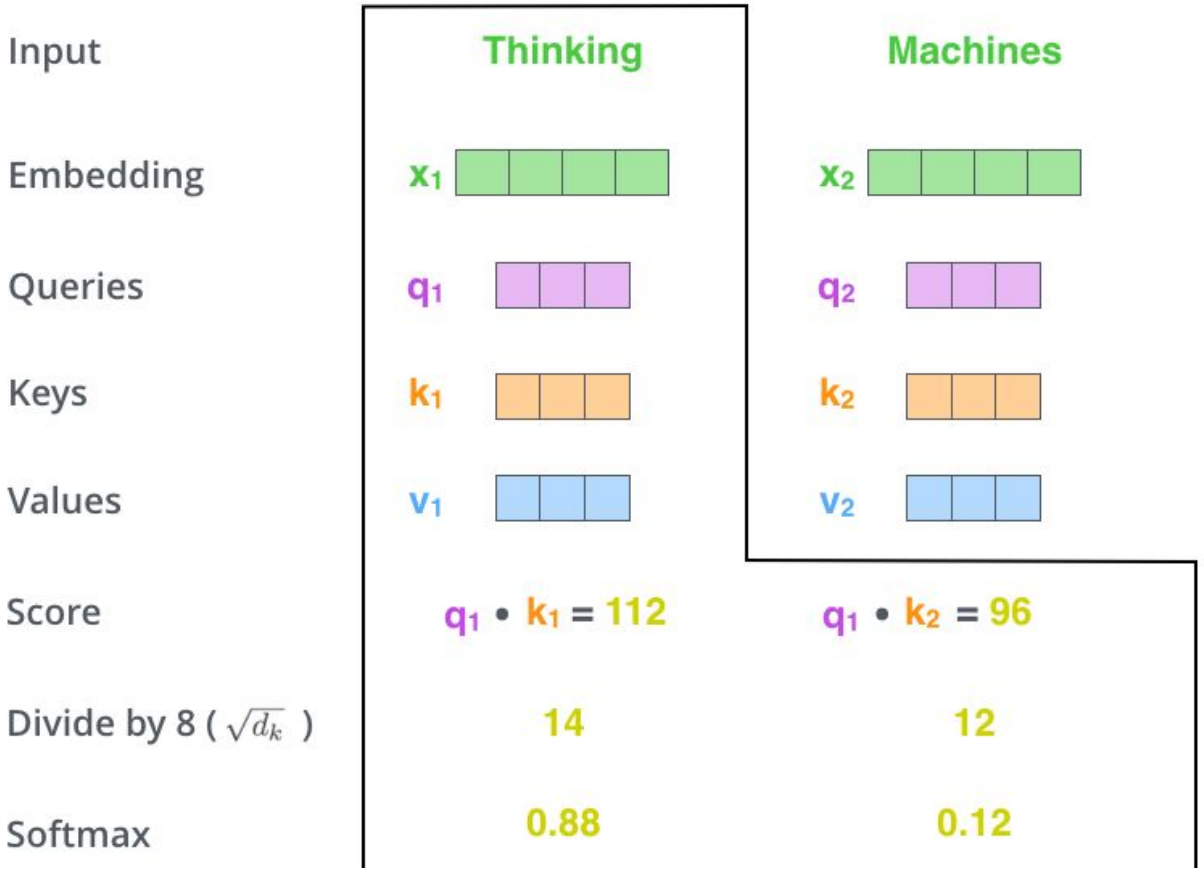
Step 0: Initialization



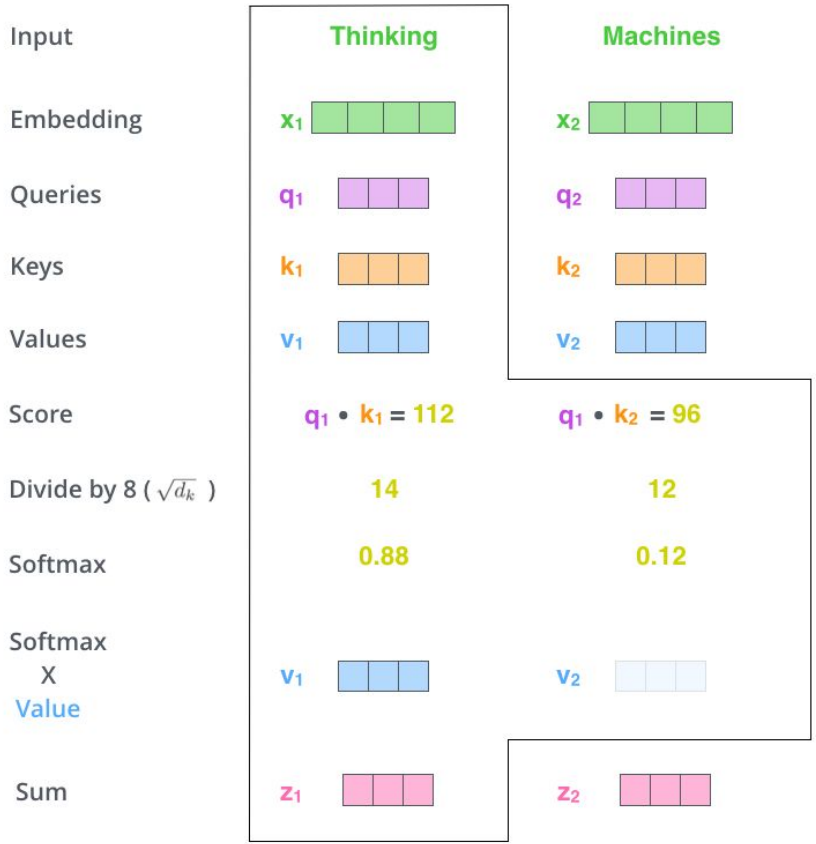
Step 1: Calculate the Score



Step 2:
Apply a transformation to the score



Step 3: Multiply by Value. Sum Up.



Summing Up

Each vector receives three representations (“roles”)

$W_Q \times$ = Query: vector **from** which the attention is looking

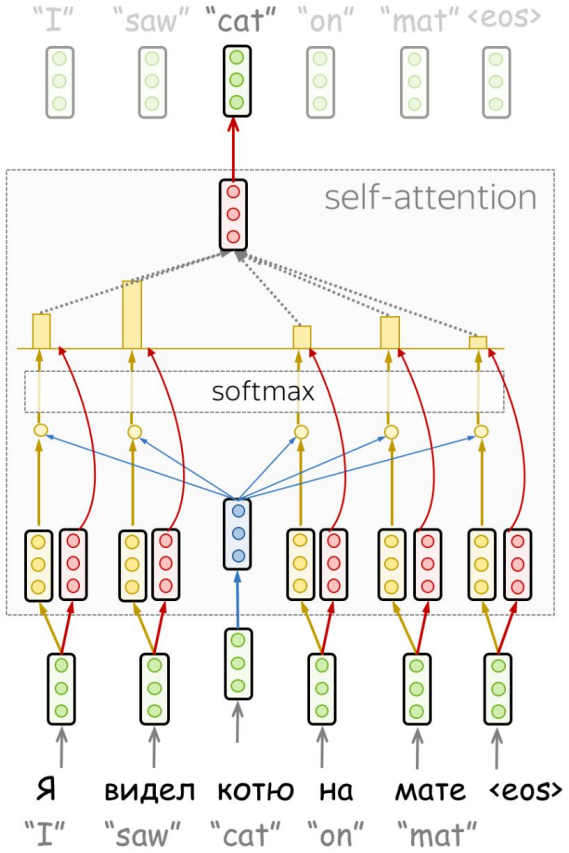
“Hey there, do you have this information?”

$W_K \times$ = Key: vector **at** which the query looks to compute weights

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

$W_V \times$ = Value: their weighted sum is attention output

“Here’s the information I have!”



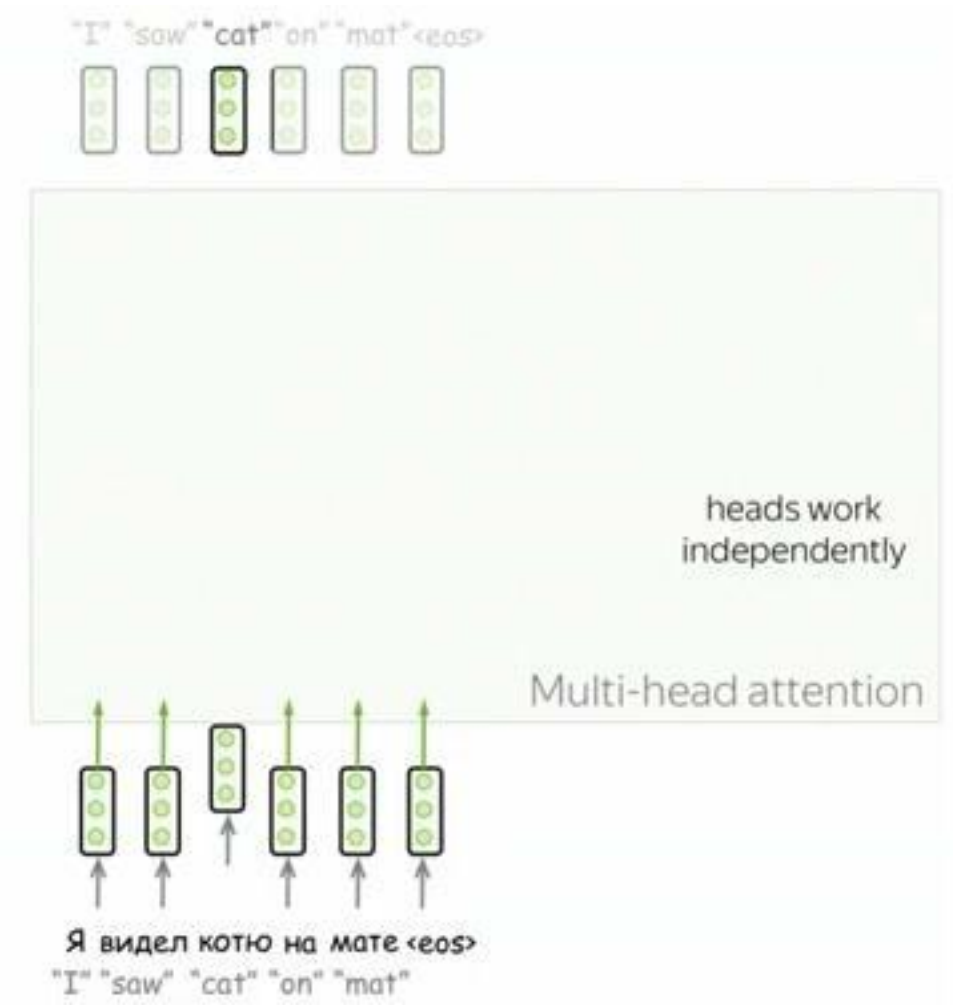
03

Multi Headed Attention

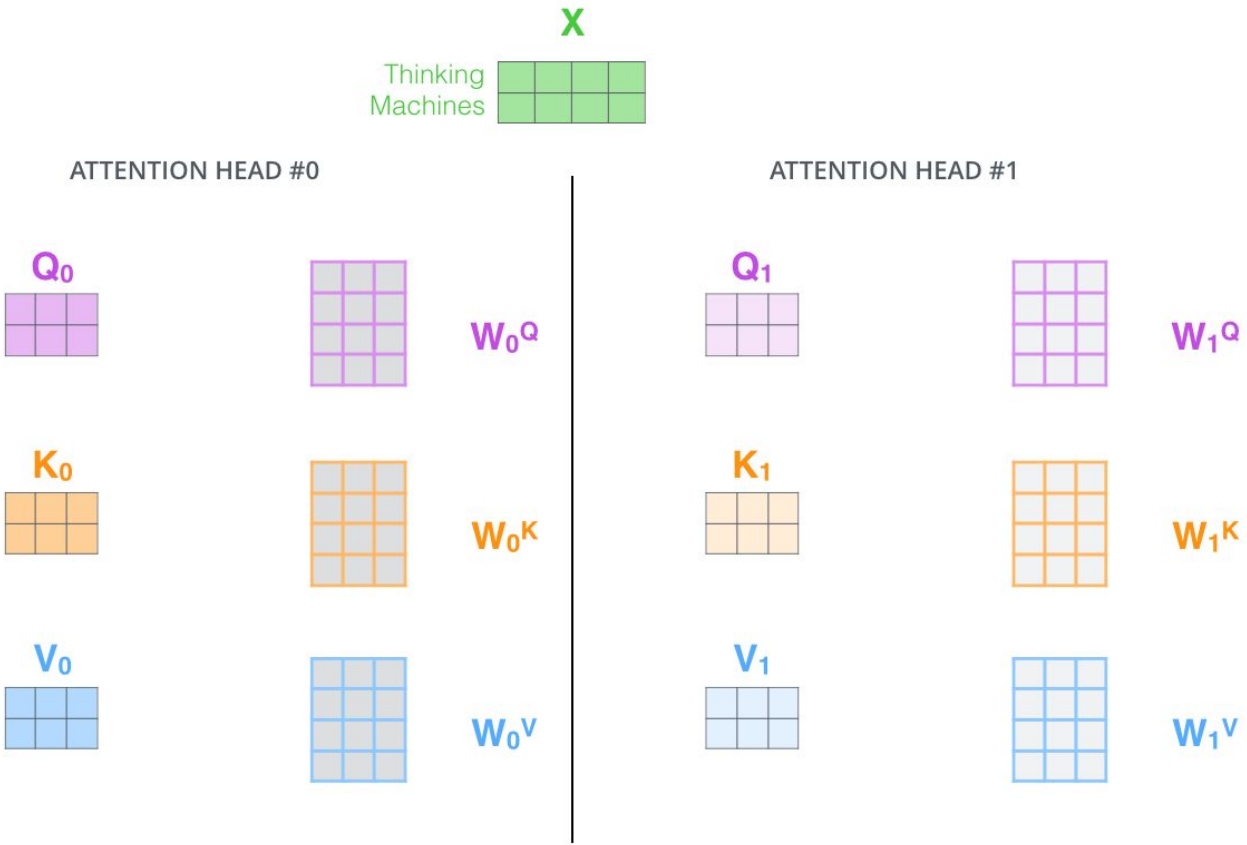
Let's quickly look at the difference between multi-head and self-head attention

video

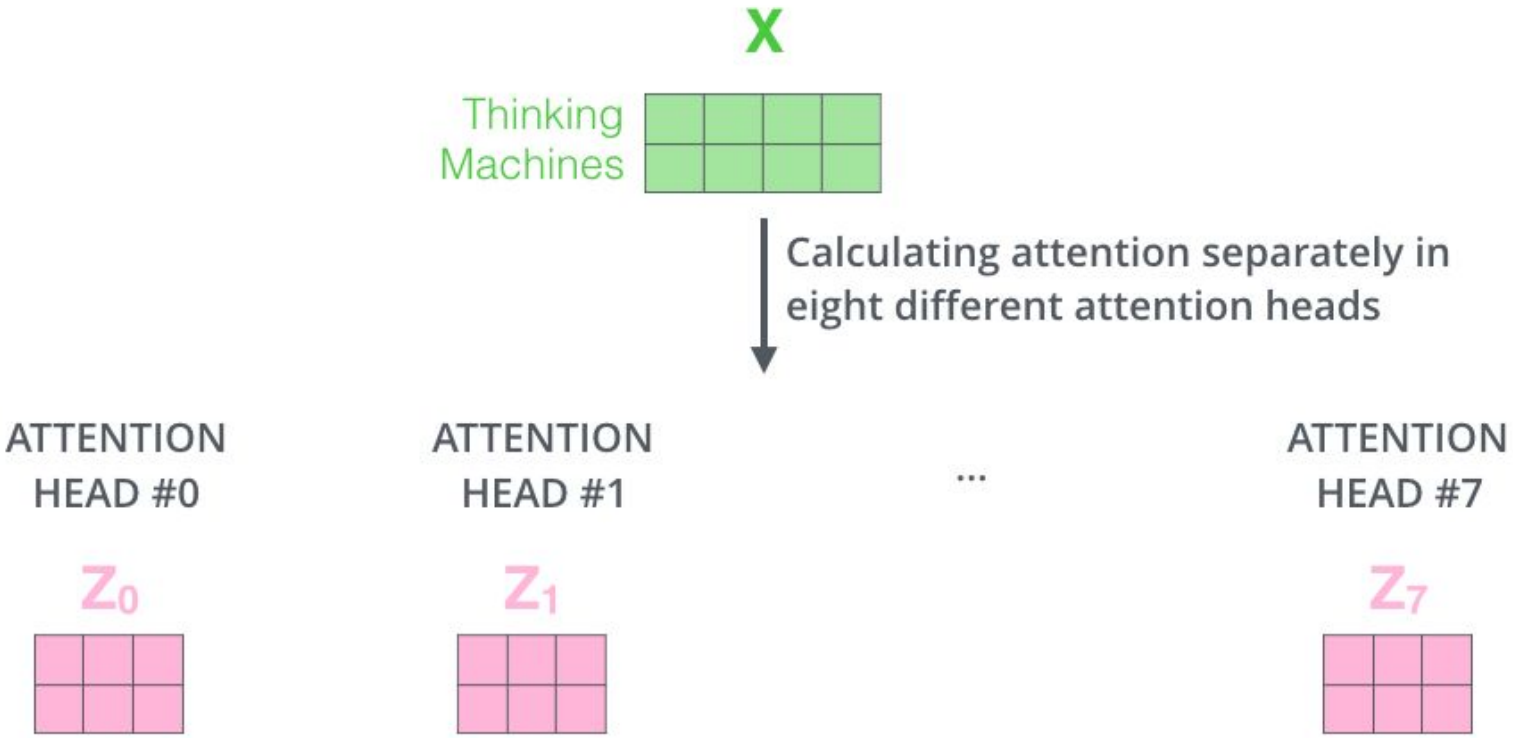
Multi-Headed Attention



Multi-Headed Attention



Multi-Headed Attention

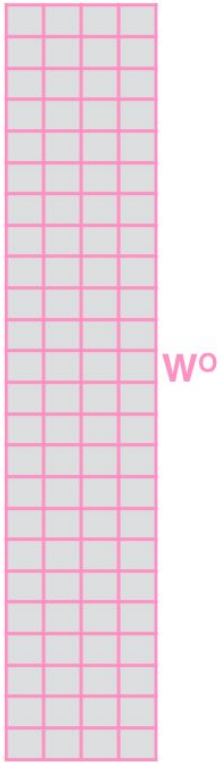


Multi-Headed Attention

1) Concatenate all the attention heads

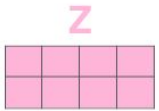
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Multi-Headed Attention

1) This is our input sentence*

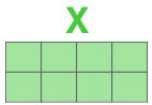
2) We embed each word*

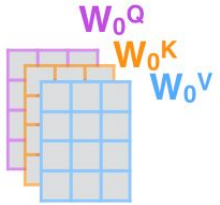
3) Split into 8 heads. We multiply X or R with weight matrices

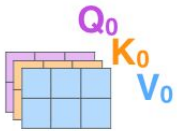
4) Calculate attention using the resulting Q/K/V matrices

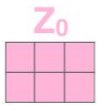
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

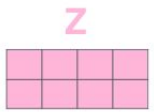
Thinking Machines

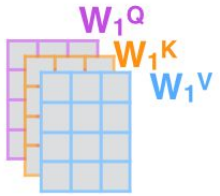


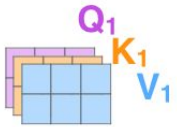


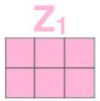




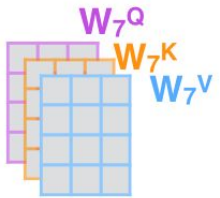


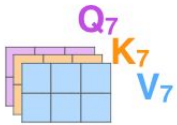


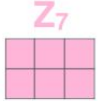




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



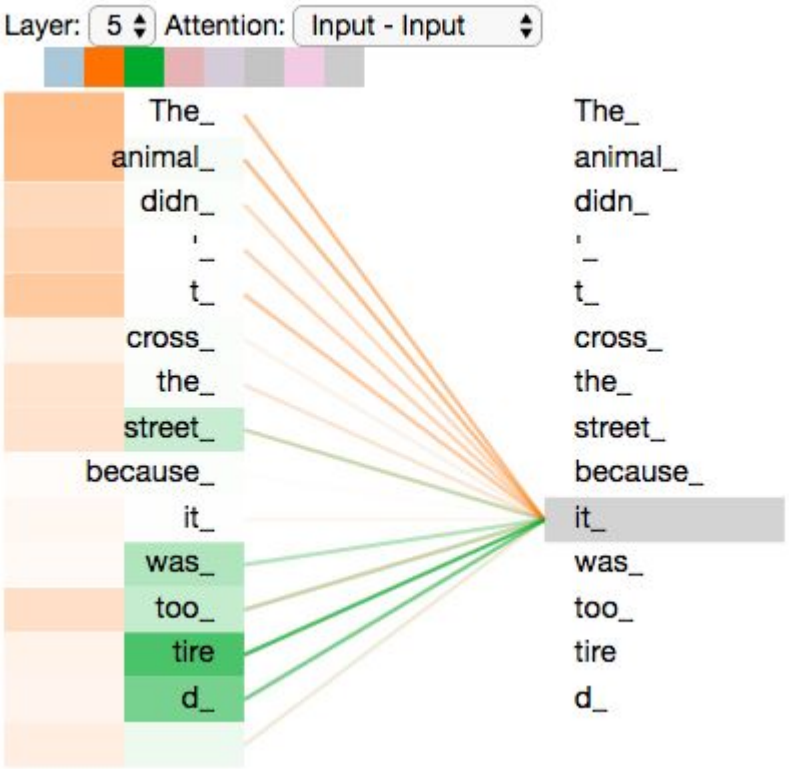




Multi-Headed Attention

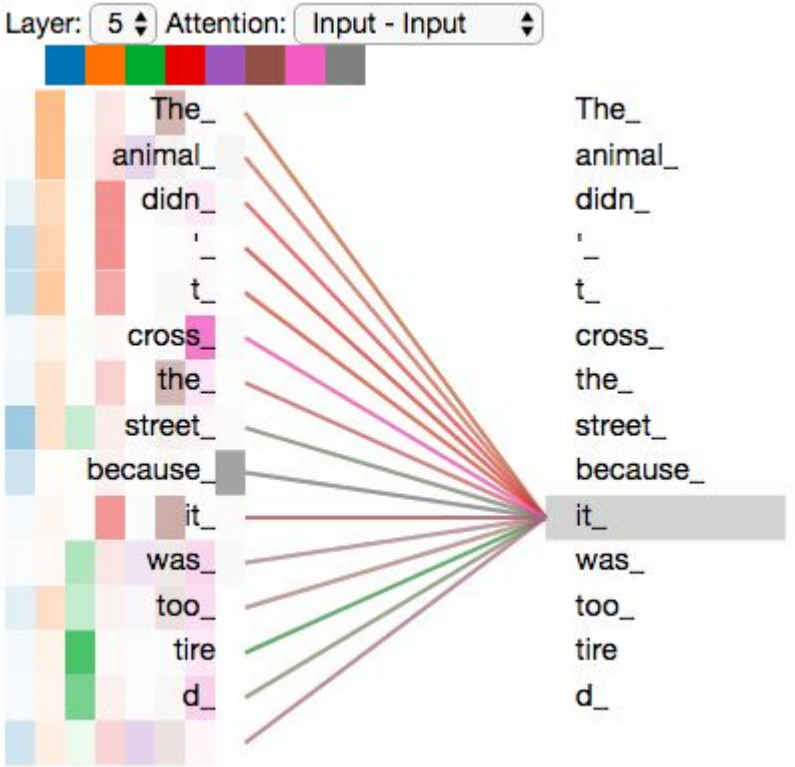
As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired".

In a sense, the **model's representation of the word "it" bakes in some of the representation of both "animal" and "tired"**.



Multi-Headed Attention

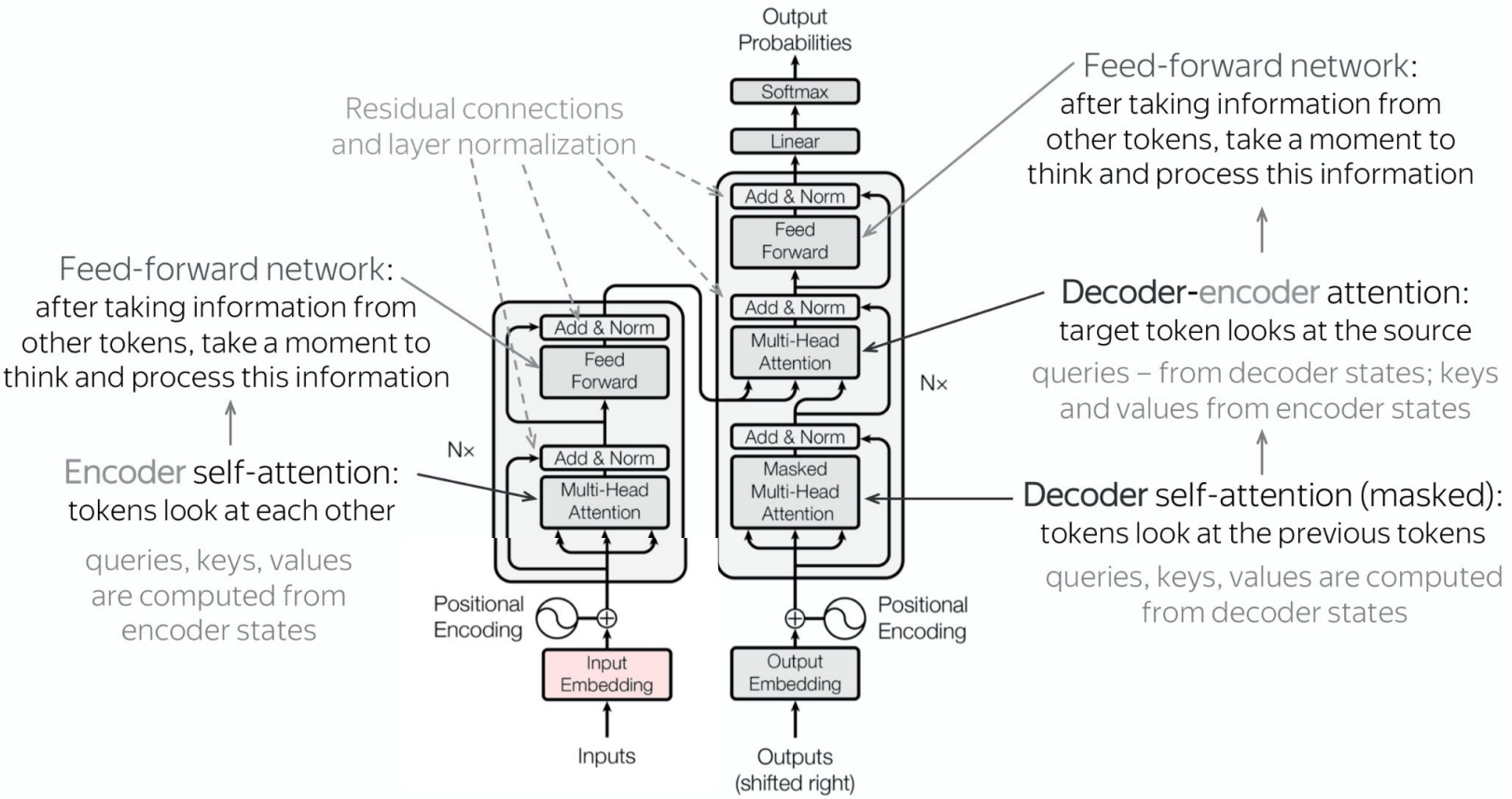
If we add all the attention heads to the picture, however, things can be harder to interpret:



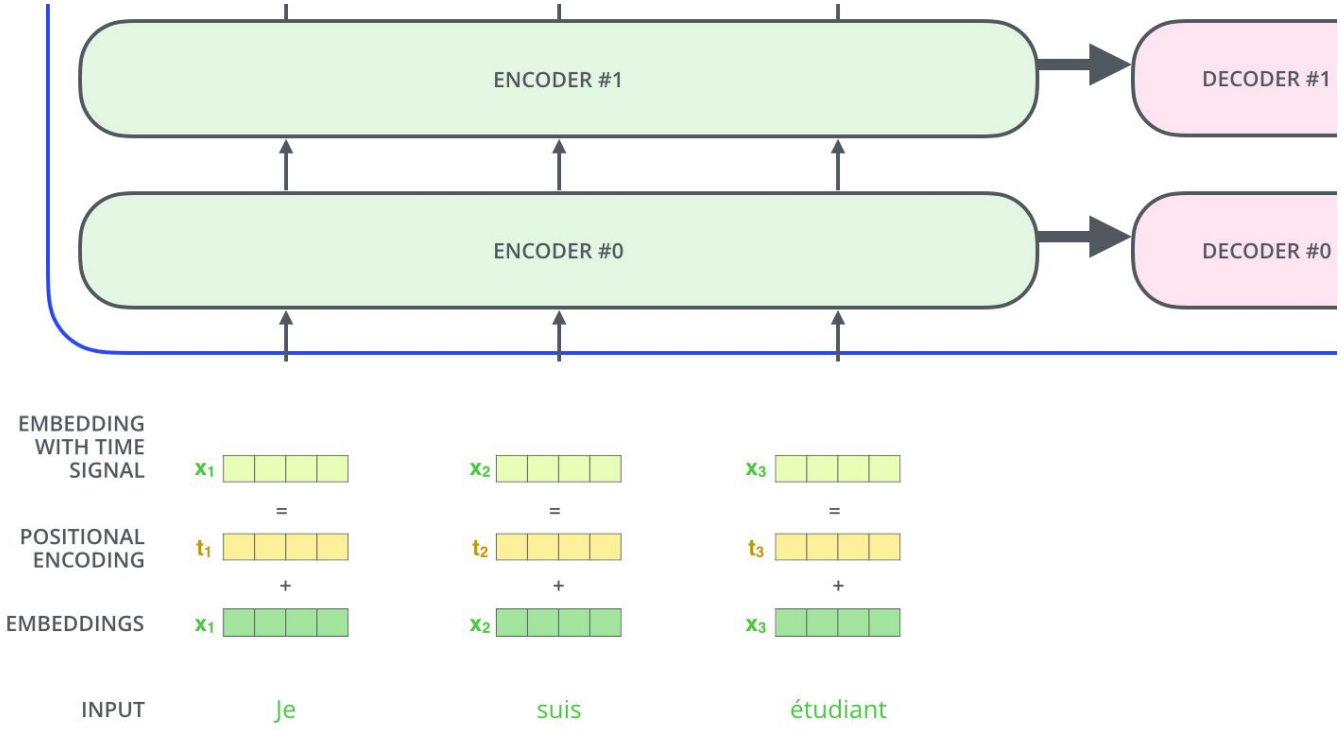
04

Model Architecture

Model Architecture



Positional Encoding



Positional Encoding

Why do we need positional embeddings?
Let's watch a short video.

Positional Encoding Options

Range Based Assignment

Assign a number to each time-step **within the [0, 1] range** in which 0 means the first word and 1 is the last time-step.

Cons:

- We can't figure out how many words are present within a specific range.
- In other words, time-step delta doesn't have consistent meaning across different sentences.

Linear Assignment

Assign a number to each time-step **linearly**. That is, the first word is given "1", the second word is given "2", and so on.

Cons:

- The values could get quite large.
- Our model can face sentences longer than the ones in training.
- Our model may not see any sample with one specific length which would hurt generalization of our model.

Positional Encoding Criteria

Ideally, the following criteria should be satisfied:

- ➔ **Unique encoding** for each time-step (word's position in a sentence)
- ➔ **Consistent distance** between any two time-steps across sentences with different lengths.
- ➔ **Generalize to longer sentences** without any efforts. Its values should be bounded.
- ➔ **Deterministic**.

Positional Encoding: Proposed Method

d-dimensional vector that contains information about a specific position in a sentence.

Let t be the desired position in an input sentence, $\overrightarrow{p_t} \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension (where $d \equiv_2 0$)
Then $f : \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector $\overrightarrow{p_t}$ and it is defined as follows:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

As it can be derived from the function definition, the frequencies are decreasing along the vector dimension. Thus it forms a geometric progression from 2π to $10000 \cdot 2\pi$ on the wavelengths.

Positional Encoding: Proposed Method

You can also imagine the positional embedding $\overrightarrow{p_t}$ as a vector containing pairs of sines and cosines for each frequency (Note that d is divisble by 2):

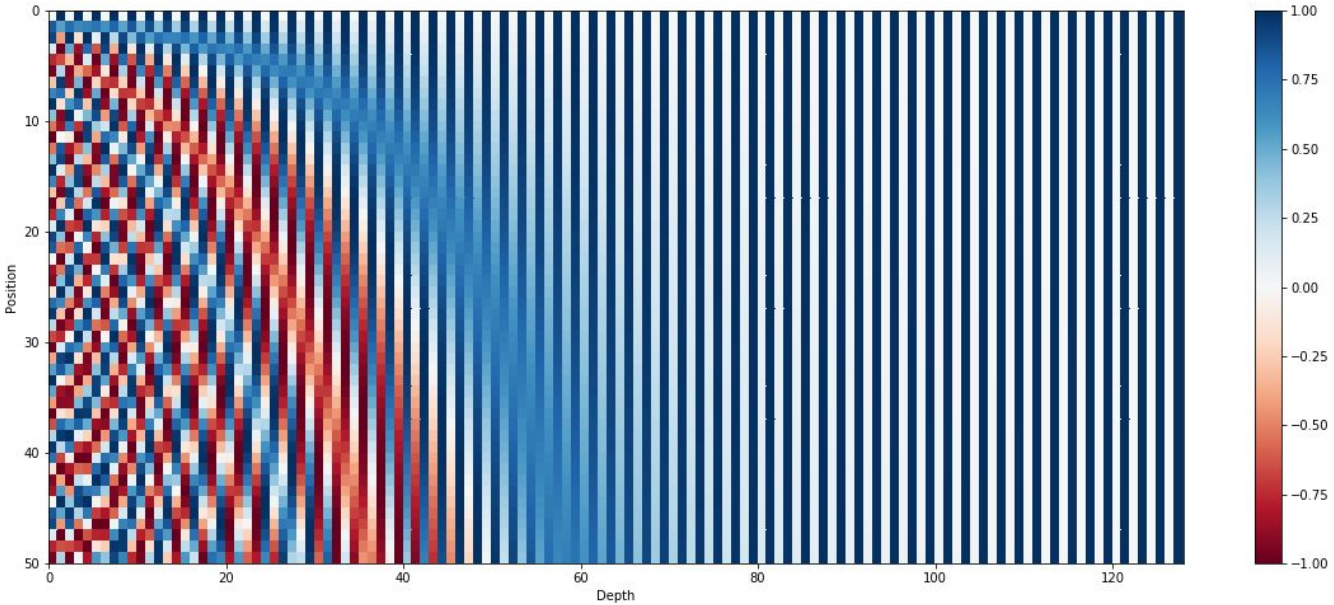
$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

Positional Encoding: The Intuition

Suppose you want to represent a number in binary format, how will that be?

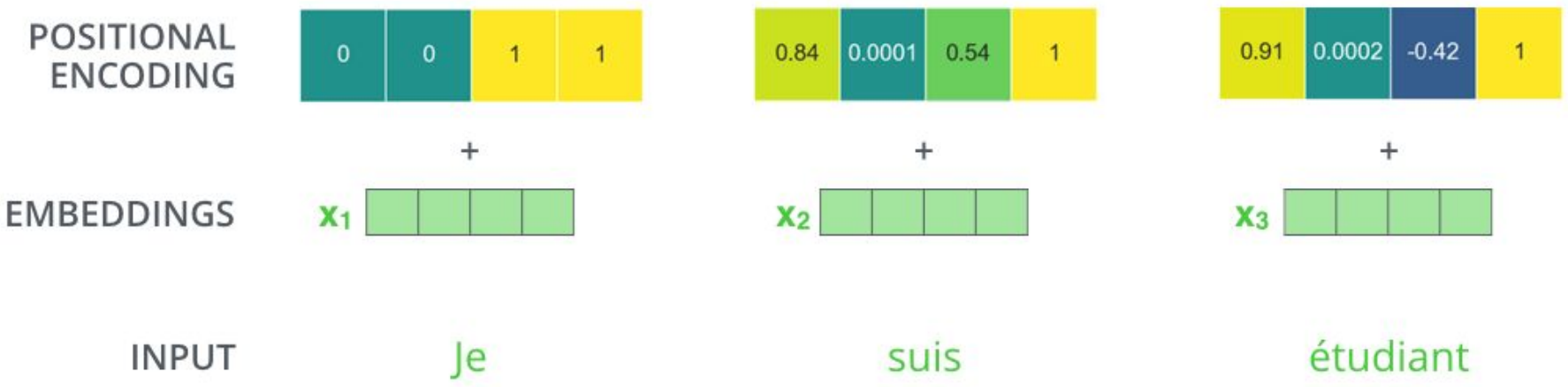
0 :	0	0	0	0	
1 :	0	0	0	1	
2 :	0	0	1	0	
3 :	0	0	1	1	
4 :	0	1	0	0	
5 :	0	1	0	1	
6 :	0	1	1	0	
7 :	0	1	1	1	
8 :	1	0	0	0	
9 :	1	0	0	1	
10 :	1	0	1	0	
11 :	1	0	1	1	
12 :	1	1	0	0	
13 :	1	1	0	1	
14 :	1	1	1	0	
15 :	1	1	1	1	

Positional Encoding: **The Intuition**



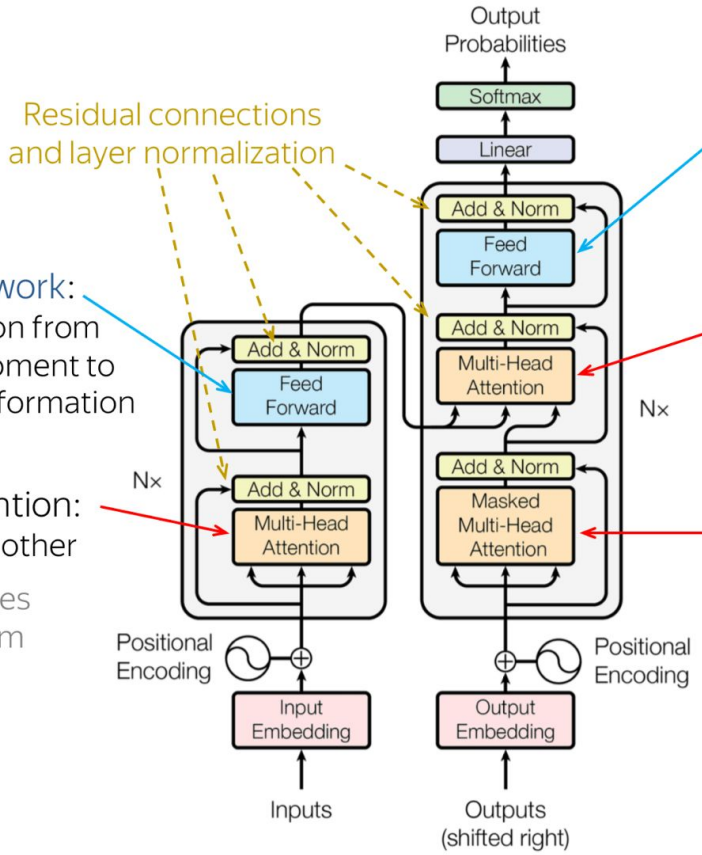
The 128-dimensional positional encoding for a sentence with the maximum length of 50. Each row represents the embedding vector p_t

Positional Encoding:
Visualization



Assumed the embedding has a dimensionality of 4, this is how the actual positional encodings would look like

Model Architecture



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

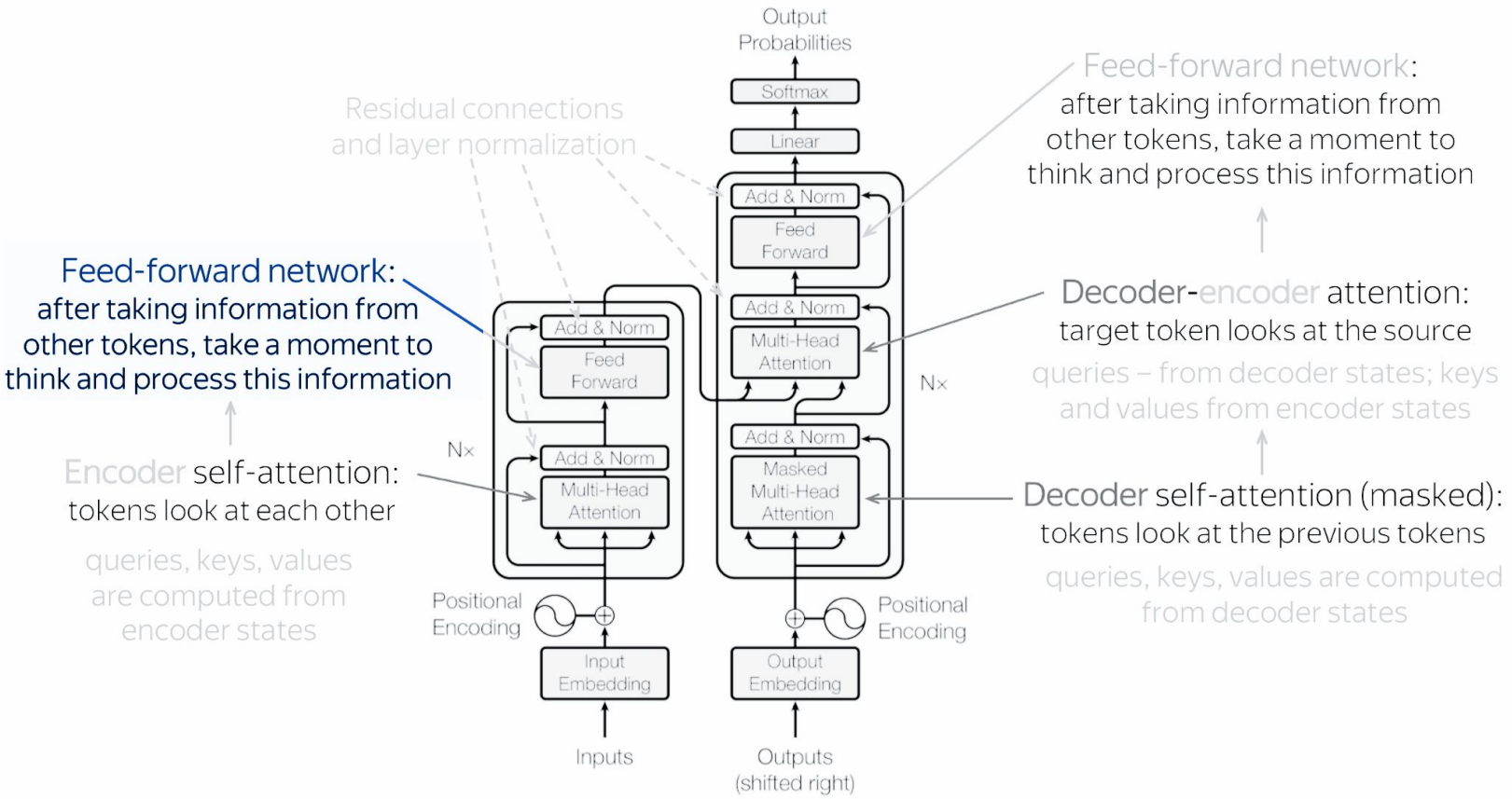
Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

Model Architecture

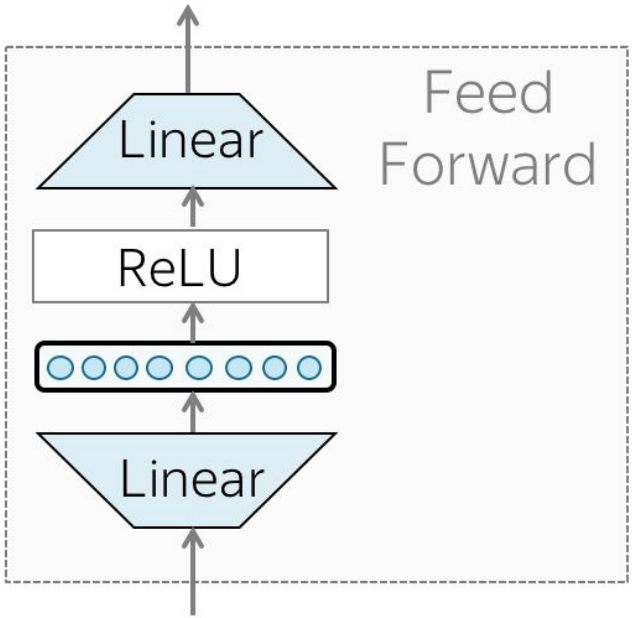


Feed Forward Network

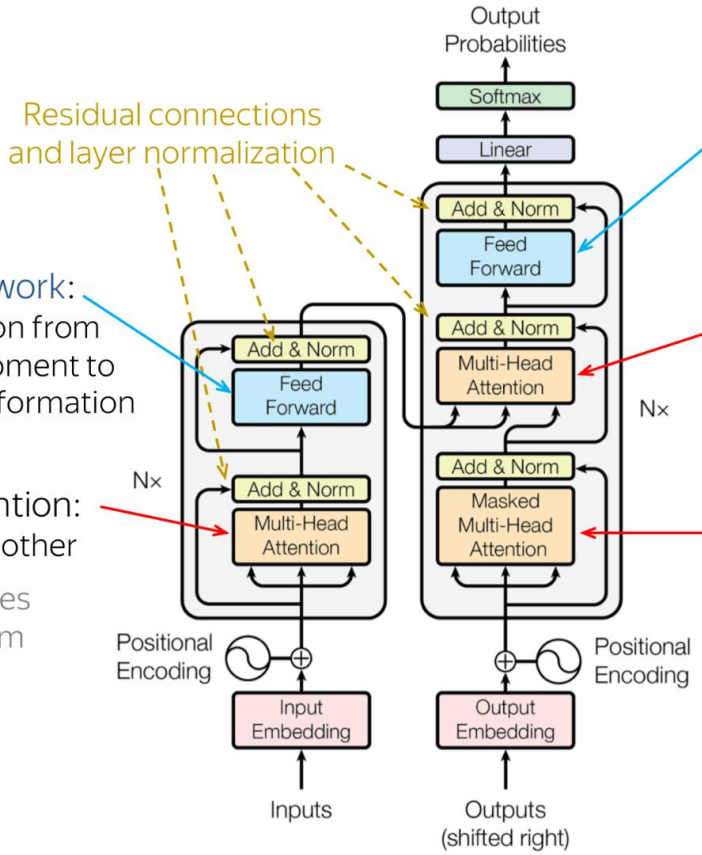
In addition to attention, each layer has a feed-forward network block: two linear layers with ReLU non-linearity between them:

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

After looking at other tokens via an attention mechanism, a model uses an FFN block to process this new information (attention - "look at other tokens and gather information", FFN - "take a moment to think and process this information").



Model Architecture



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

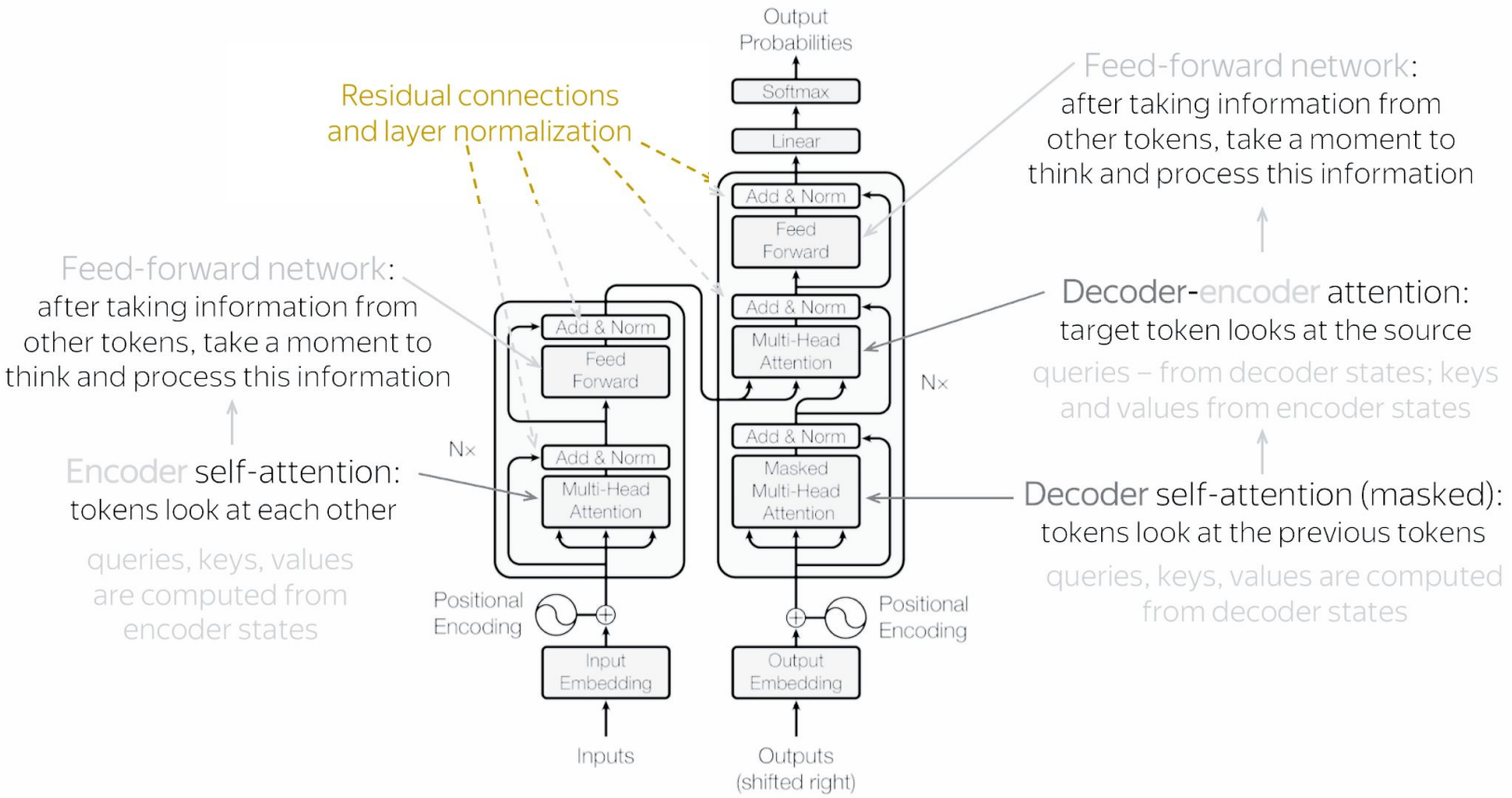
Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

Decoder self-attention (masked): tokens look at the previous tokens

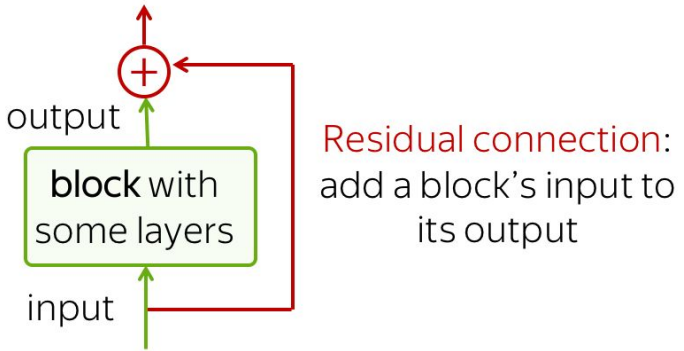
queries, keys, values are computed from decoder states

Model Architecture



Residual Connections

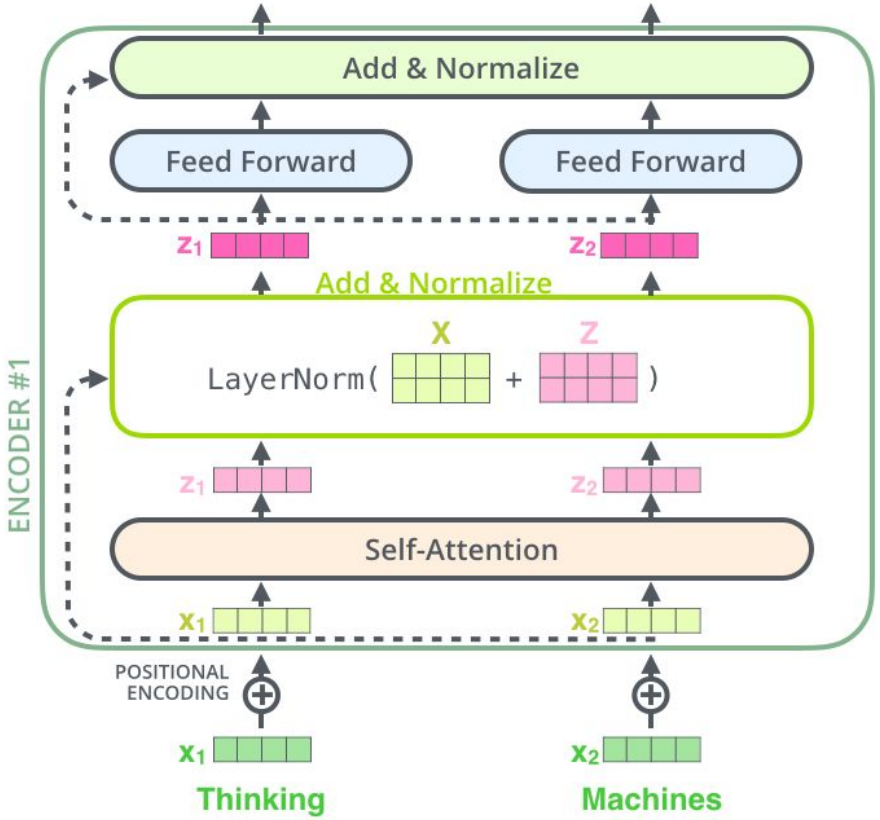
Residual connections are very simple (add a block's input to its output), but at the same time are very useful: they ease the gradient flow through a network and allow stacking a lot of layers.



Residual Connections

In the Transformer, residual connections are used after each **attention** and **FFN** block.

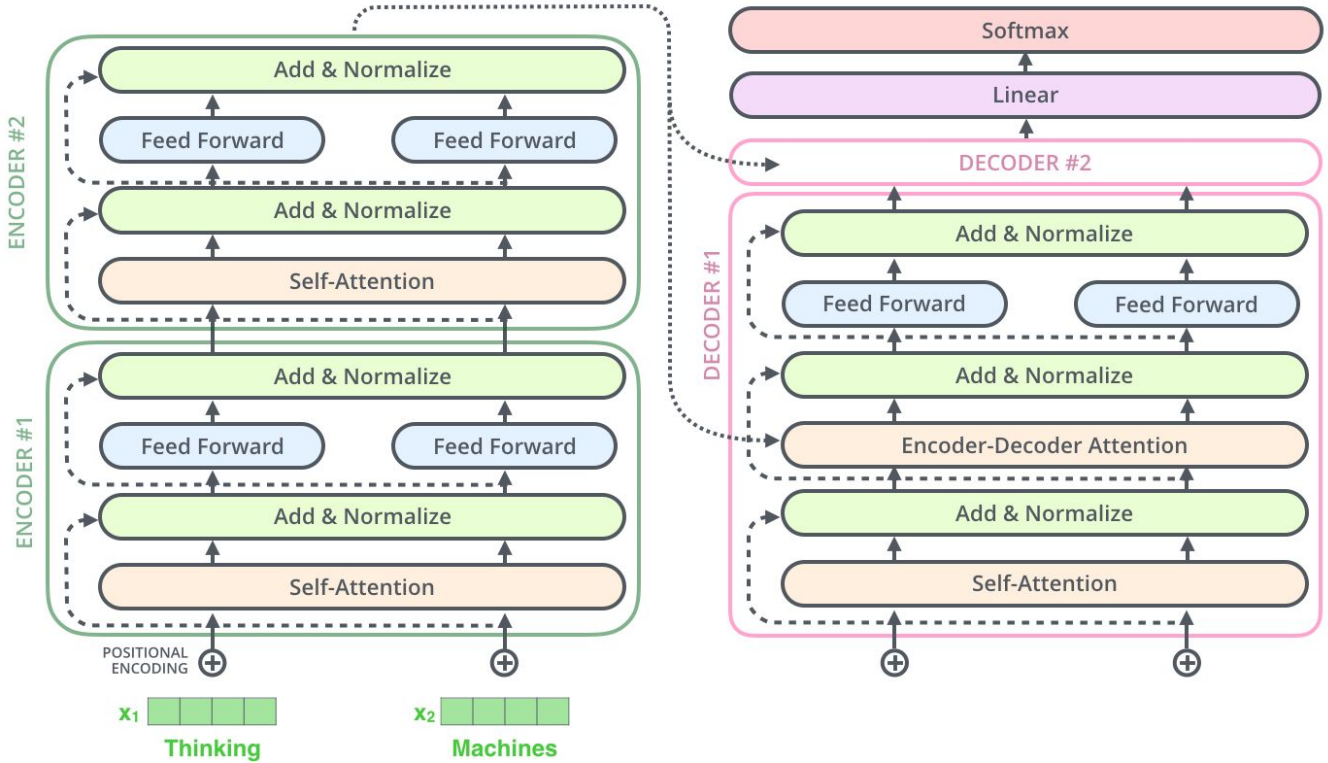
Residuals are shown as arrows coming around a block to the **“Add & Normalize"** layer.
In the **“Add & Normalize"** part, the "Add" part stands for the residual connection.



Residual Connections

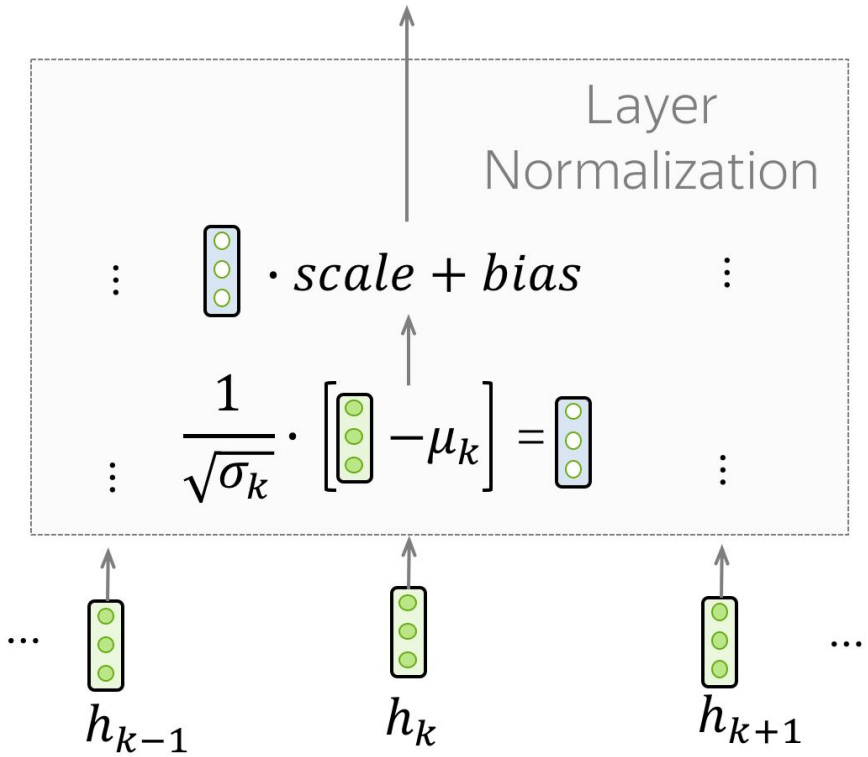
This goes for the sub-layers of the decoder as well.

If we're to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:



Layer Norm

- The "Norm" part in the "Add & Norm" layer denotes Layer Normalization.
- It independently normalizes vector representation of each example in batch - this is done to control "flow" to the next layer.
- Layer normalization improves convergence stability and sometimes even quality.



Google DeepMind

Summing up what we have learnt

[Video]

tme: 2.5h

Google DeepMind

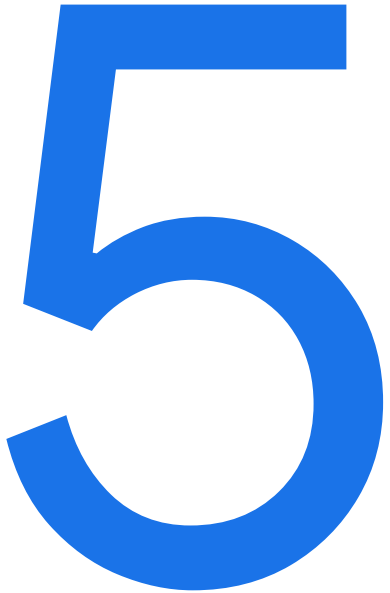
Let us build our own transformer.

[Colab Notebook]

tme: 2.5h

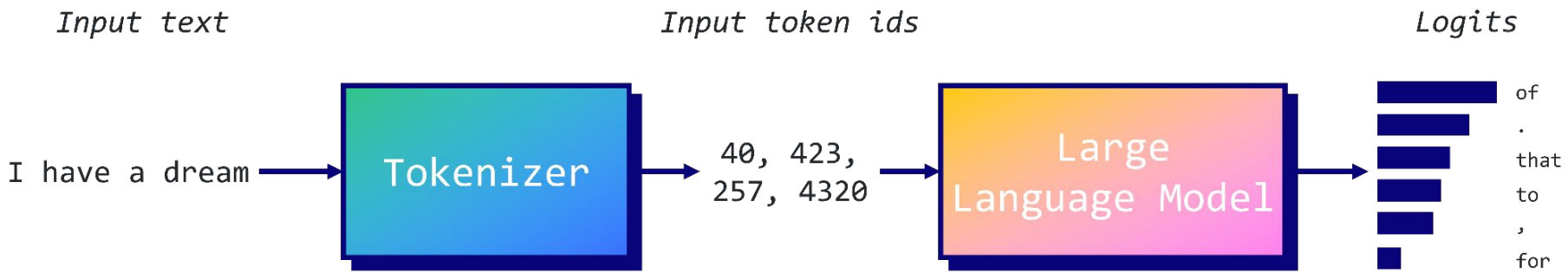
Sampling Strategies

I. Overview
II. Greedy and Beam Sampling
III. Top-k and Top-p Sampling



01 Overview

LLMs calculate logits, which are scores assigned to every possible token in their vocabulary.



This essentially represents a ranked list of potential next tokens in the sequence.

Autoregressive models like GPT predict the next token in a sequence based on the preceding tokens. Consider a sequence of tokens $w = (w_1, w_2, \ldots, w_t)$. The joint probability of this sequence $P(w)$ can be broken down as:

$$
\begin{aligned}
P(w) &= P(w_1, w_2, \ldots, w_t) && (1)\\
&= P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)\ldots P(w_t|w_1, \ldots, w_{t-1}) && (2)\\
&= \prod_{i=1}^{t} P(w_i|w_1, \ldots, w_{i-1}). && (3)
\end{aligned}
$$

For each token w_i in the sequence, $P(w_i|w_1, \ldots, w_{i-1})$ represents the conditional probability of w_i given all the preceding tokens $(w_1, \ldots, w_{i-1})$. GPT-2 calculates this conditional probability for each of the 50,257 tokens in its vocabulary.

This leads to the question: how do we use these probabilities to generate text? This is where decoding strategies, such as greedy search and beam search, come into play.

01

Greedy and Beam Search

Greedy Search

Definition: Takes the most probable token at each step as the next token in the sequence.

Step by Step Example

Input: "I have a dream" → Most likely token: " of"
Input: "I have a dream of" → Most likely token: " being"
Input: "I have a dream of being" → Most likely token: " a"
Input: "I have a dream of being a" → Most likely token: " doctor"
Input: "I have a dream of being a doctor" → Most likely token: "."

Con

- ➔ Considers the most probable token at each step without considering the overall effect on the sequence.
- ➔ Can miss out on better sequences that might have appeared with slightly less probable next tokens.

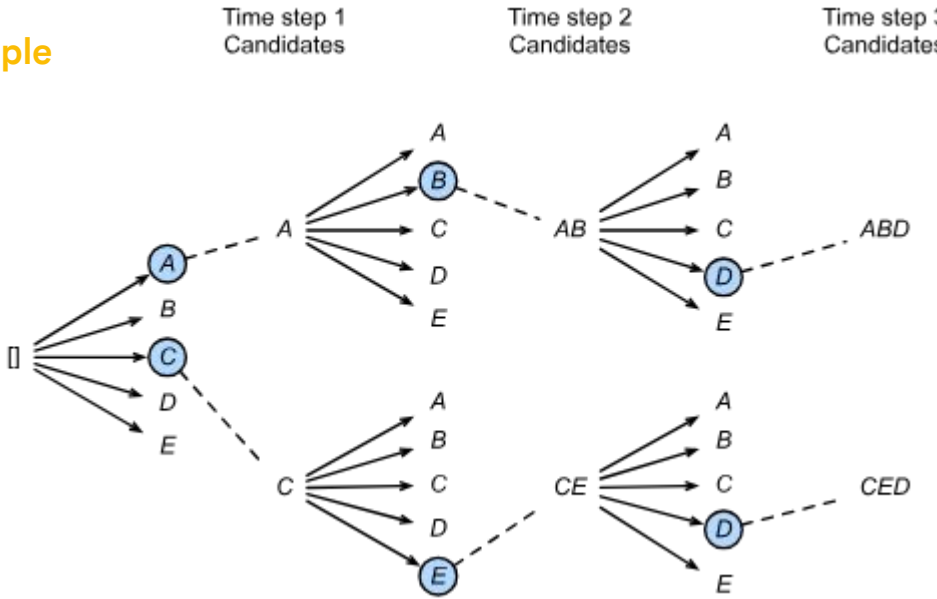
Pro

Fast and efficient as it doesn't need to keep track of multiple sequences.

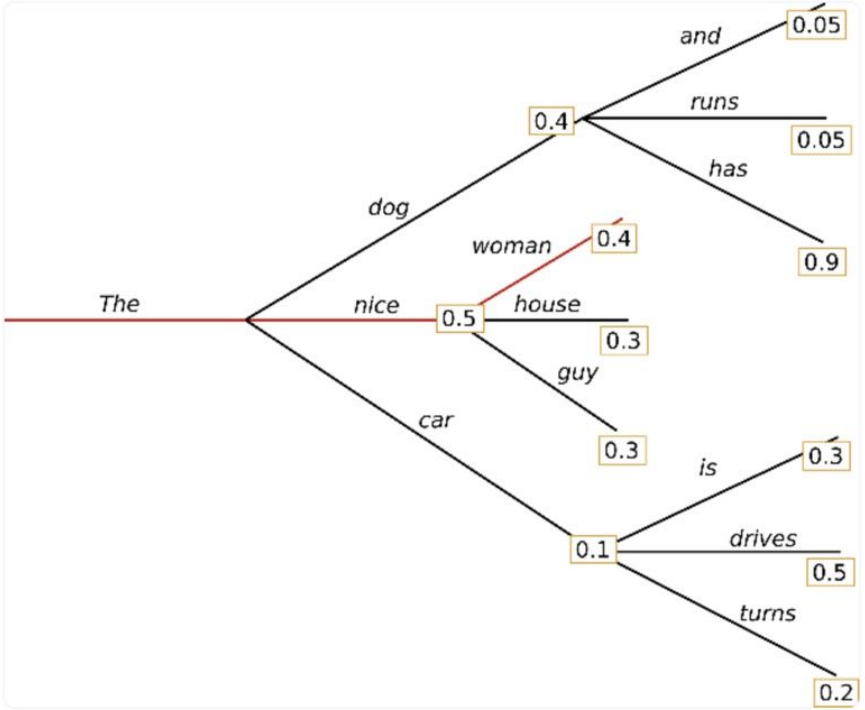
Beam Search

Definition: At each step, takes into account the ***n* most likely tokens**, where *n* represents the number of beams. This is repeated until a predefined maximum length is reached or an end-of-sequence token appears.
At this point, the **sequence (or "beam") with the highest overall score** is chosen as the output.

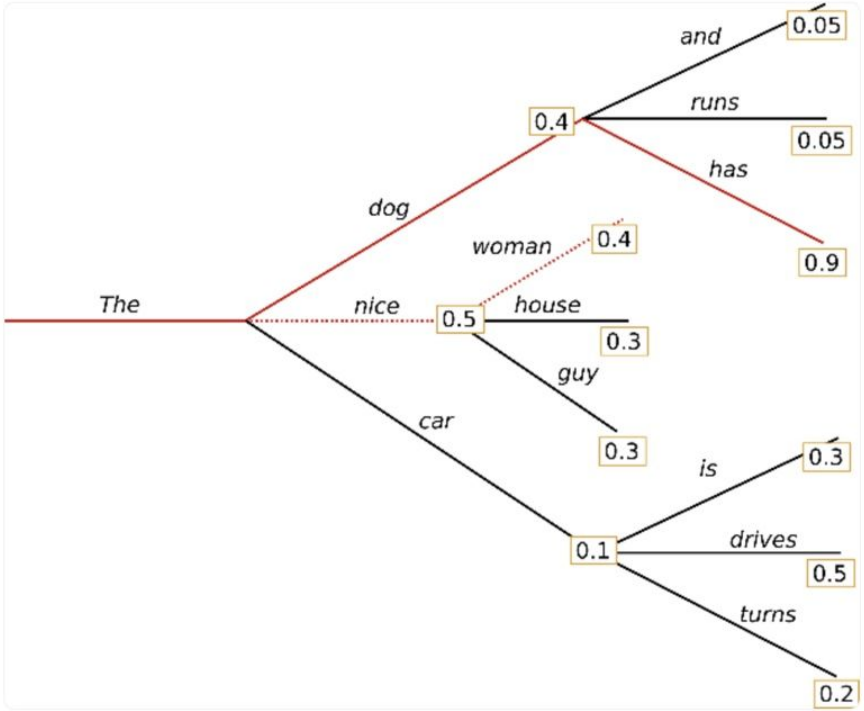
Step by Step Example



Greedy Search



Beam Search



02

top-k and top-p Sampling

Top-k Sampling

Definition: Select a token randomly from the k most likely options.

Example

Suppose we have $k=3$ and four tokens: *A*, *B*, *C*, and *D*, with respective probabilities:

$P(A)=30,\ P(B)=15,\ P(C)=5,\ and\ P(D)=1.$

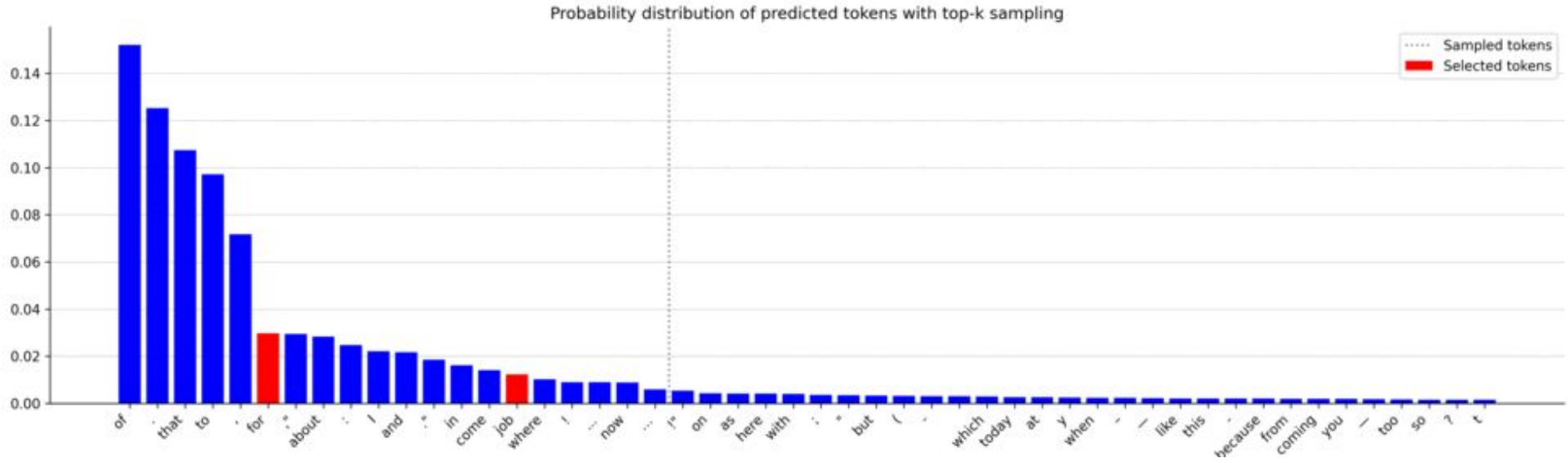
In top-k ($k=3$) sampling, token D is disregarded.

The algorithm will output A 60% of the time, B 30% of the time, and C 10% of the time.

This approach ensures that we **prioritize the most probable tokens while introducing an element of randomness** in the selection process

Top-k Sampling: Probability Distribution

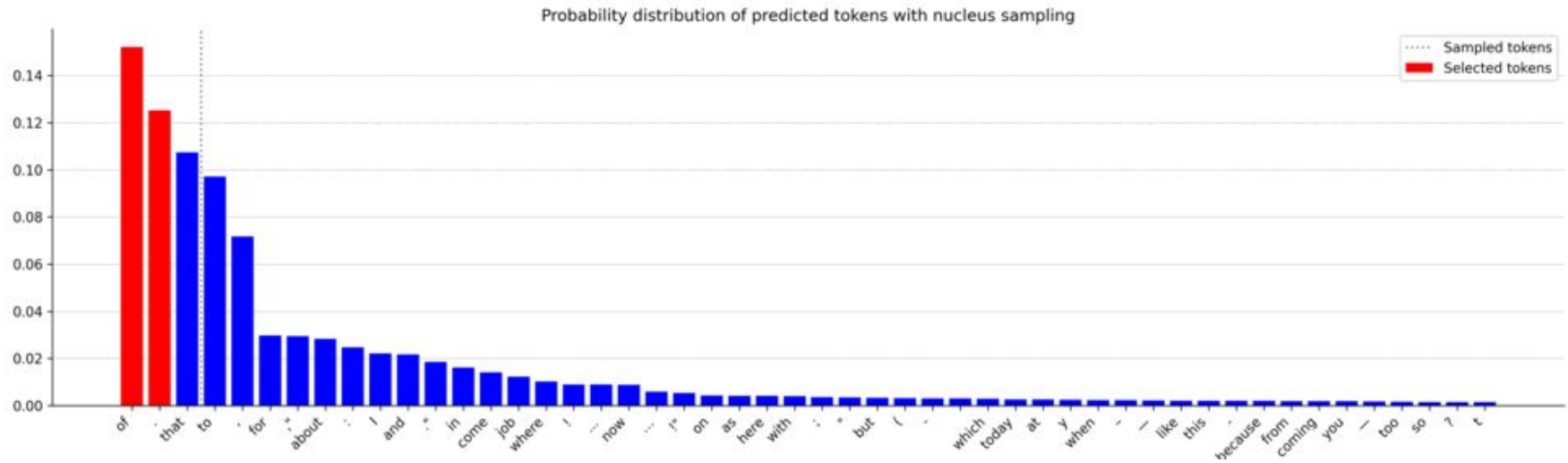
While the most probable tokens are selected (in red) most of the time, it also allows less likely tokens to be chosen. This allows steer a sequence towards a less predictable but more natural-sounding sentence.



Nucleus (top-p) Sampling

Chooses a cutoff value p such that the sum of the probabilities of the selected tokens exceeds p.
This forms a “nucleus” of tokens from which to randomly choose the next token.

In other words, the model examines its top probable tokens in descending order and keeps adding them to the list until the total probability surpasses the threshold p.



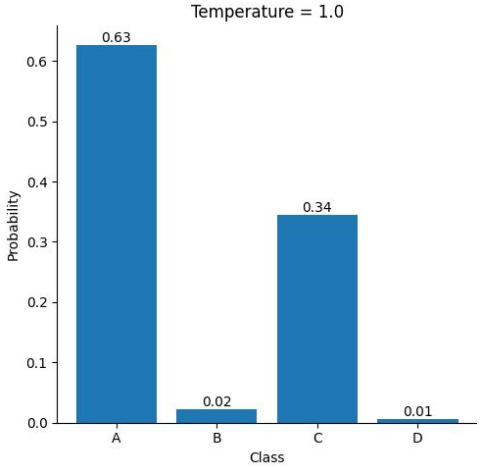
Introducing Randomness: Temperature

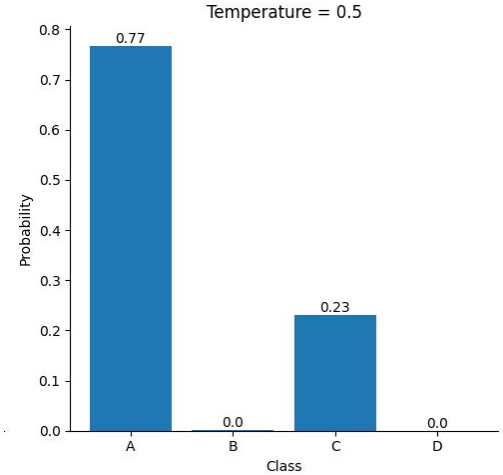
Temperature, T

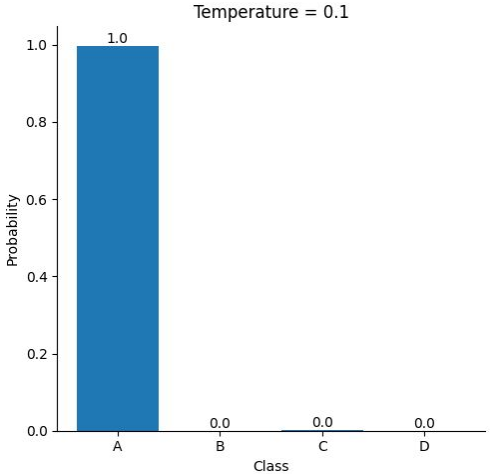
- ranges from *0 to 1*,
- affects the probabilities generated by the softmax,
- making **the most likely tokens more influential.**

In practice, it simply consists of dividing the input logits by a value we call temperature:

$$\text{softmax}(x_i) = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}}$$







Google DeepMind

Now that we are familiar with sampling strategies, let us try them out.

[Colab Notebook]

References

1. [1706.03762] Attention Is All You Need
2. [2304.10557] An Introduction to Transformers
3. [2006.16362] Multi-Head Attention: Collaborate Instead of Concatenate
4. Additional Blogs:
    a. Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog
    b. Transformer: A Novel Neural Network Architecture for Language Understanding
    c. The Illustrated Transformer – Jay Alammar
    d. Seq2seq and Attention
    e. Decoding Strategies in Large Language Models – Maxime Labonne
    f. Andrej Karpathy Lectures: llmintro.pdf

Thank you.

Nikita Saxena
Research Engineer

Google DeepMind