UNC CHARLOTTE

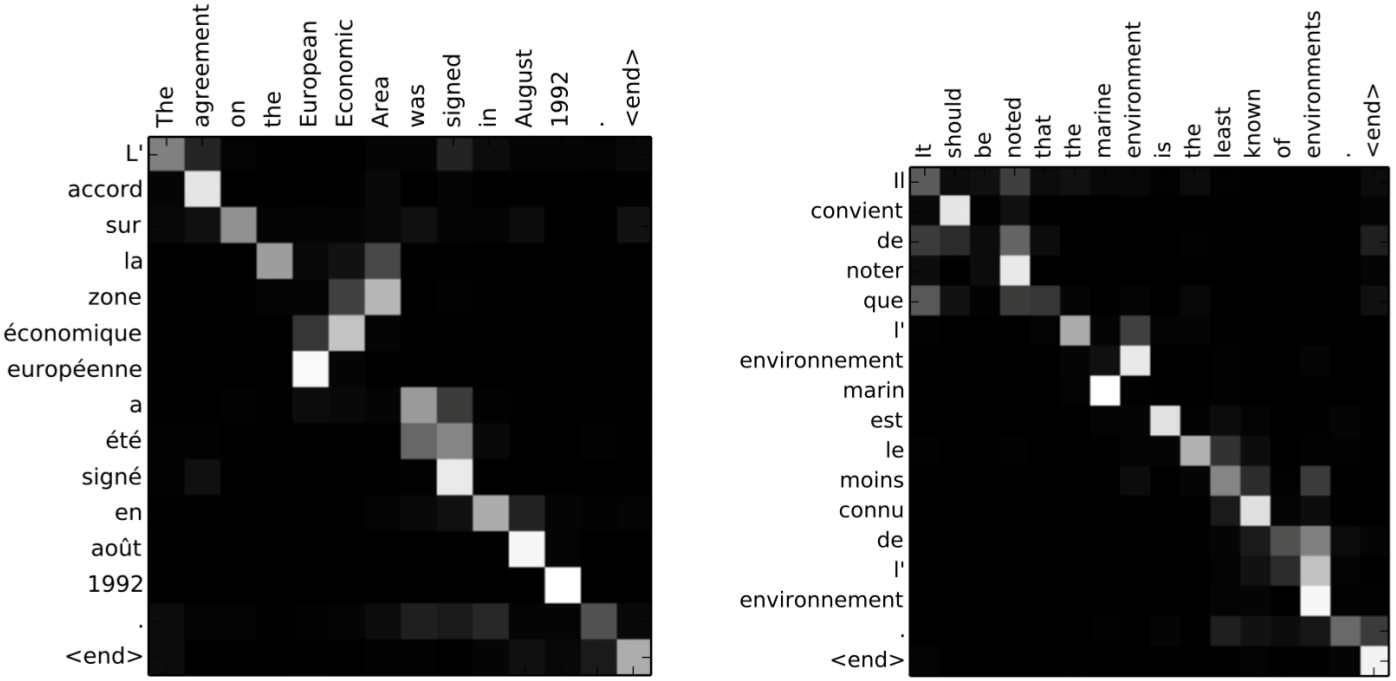
The WILLIAM STATES LEE COLLEGE *of* ENGINEERING

Introduction to ML
Lecture 11: Transformers

Hamed Tabkhi
Department of Electrical and Computer Engineering,
University of North Carolina Charlotte (UNCC)
htabkhiv@uncc.edu

Problem

- Remember the motivation for attention?
- At different steps, the decoder may need to focus on different source tokens, the ones which are more relevant at this step.
- Let's look at attention weights - which source words does the decoder use?



- From the examples, we see that attention learned (soft) alignment between source and target words - the decoder looks at those source tokens which it is translating at the current step.
- "Alignment" is a term from statistical machine translation, but in this part, its intuitive understanding as "what is translated to what" is enough.

Transformer: Attention is All You Need

- Transformer is a model introduced in the paper Attention is All You Need in 2017.

- It is based solely on attention mechanisms: i.e., without recurrence or convolutions. On top of higher translation quality, the model is faster to train by up to an order of magnitude.

- Currently, Transformers (with variations) are de-facto standard models not only in sequence to sequence tasks but also for language modeling and in pretraining settings, which we consider in the next lecture.

- **Transformer introduced a new modeling paradigm: in contrast to previous models where processing within encoder and decoder was done with recurrence or convolutions, Transformer operates using only attention.**

Transformer: Attention is All You Need

	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

Transformer: Attention is All You Need

Look at the illustration from the Google AI blog post introducing Transformer.

Transformer: Attention is All You Need

Without going into too many details, let's put into words what just saw in the illustration. We'll get something like the following:

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

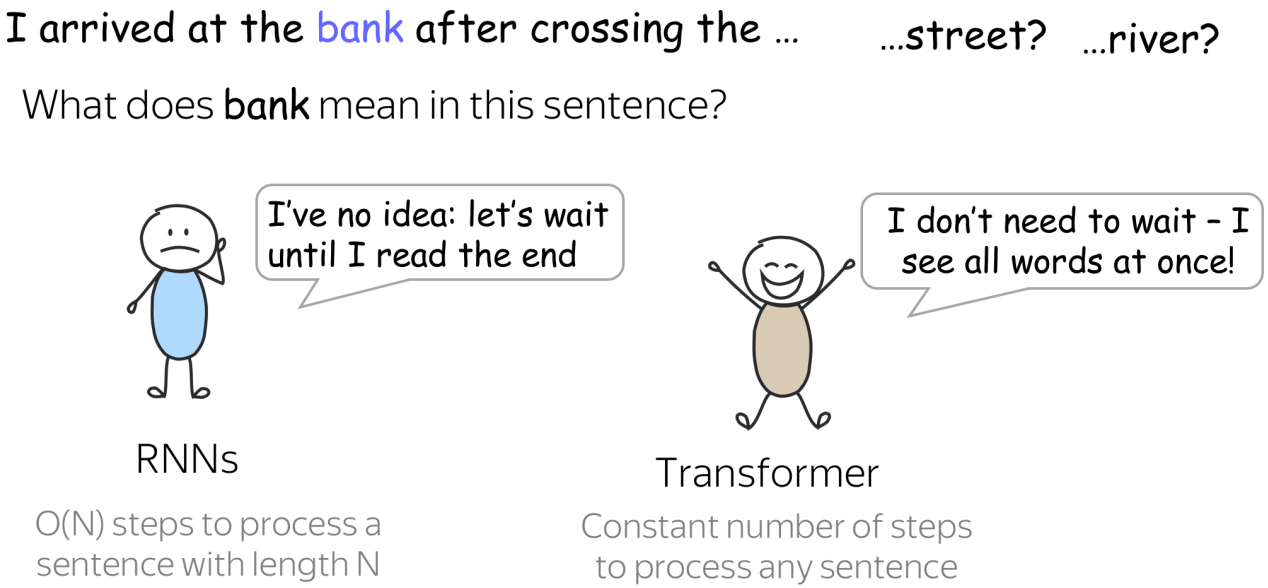
What they are doing:

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

repeat N times

Transformer: Attention is All You Need

- When encoding a sentence, RNNs won't understand what **bank** means until they read the whole sentence, and this can take a while for long sequences.
- In contrast, in Transformer's encoder tokens interact with each other all at once.
- Intuitively, Transformer's encoder can be thought of as a sequence of reasoning steps (layers). At each step, tokens look at each other (this is where we need attention - self-attention), exchange information and try to understand each other better in the context of the whole sentence. This happens in several layers (e.g., 6).
- In each decoder layer, tokens of the prefix also interact with each other via a self-attention mechanism, but additionally, they look at the encoder states (without this, no translation can happen, right?).



UNC CHARLOTTE

Self-Attention: the "Look at Each Other" Part

Self-attention is one of the key components of the model. The difference between attention and self-attention is that self-attention operates between representations of the same nature: e.g., all encoder states in some layer.

Decoder-encoder attention is looking

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

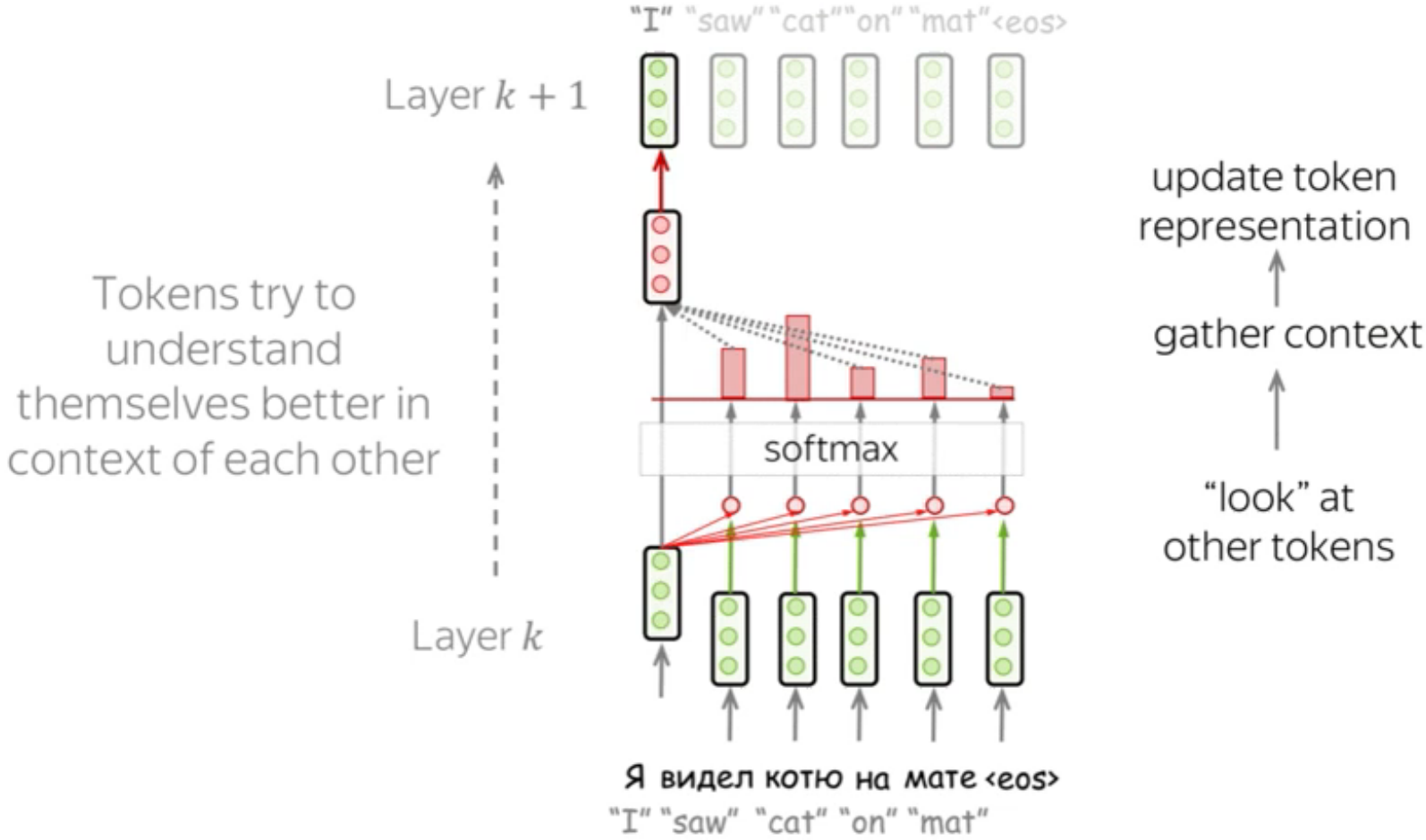
Self-attention is looking

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

Self-attention is the part of the model where tokens interact with each other. Each token "looks" at other tokens in the sentence with an attention mechanism, gathers context, and updates the previous representation of "self".

Self-Attention: the "Look at Each Other" Part

Look at the illustration.



Note that in practice, this happens in parallel.

UNC CHARLOTTE

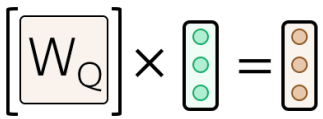
Query, Key, and Value in Self-Attention

- Formally, this intuition is implemented with a query-key-value attention. Each input token in self-attention receives three representations corresponding to the roles it can play:
  - query - asking for information;
  - key - saying that it has some information;
  - value - giving the information.

- The query is used when a token looks at others - it's seeking the information to understand itself better. The key is responding to a query's request: it is used to compute attention weights.

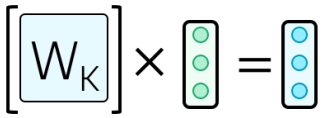
Query, Key, and Value in Self-Attention

Each vector receives three representations ("roles")



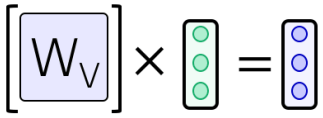
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



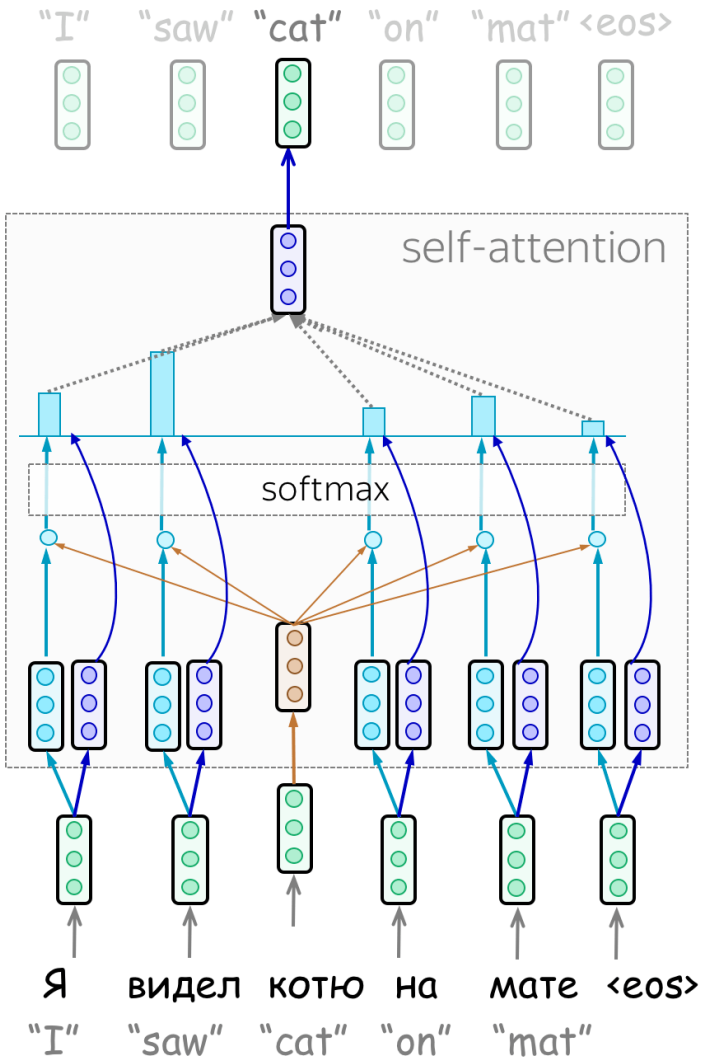
Key: vector **at** which the query looks to compute weights

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



Value: their weighted sum is attention output

"Here's the information I have!"



All these three sets of weight are differentiable (trainable through backpropagation)

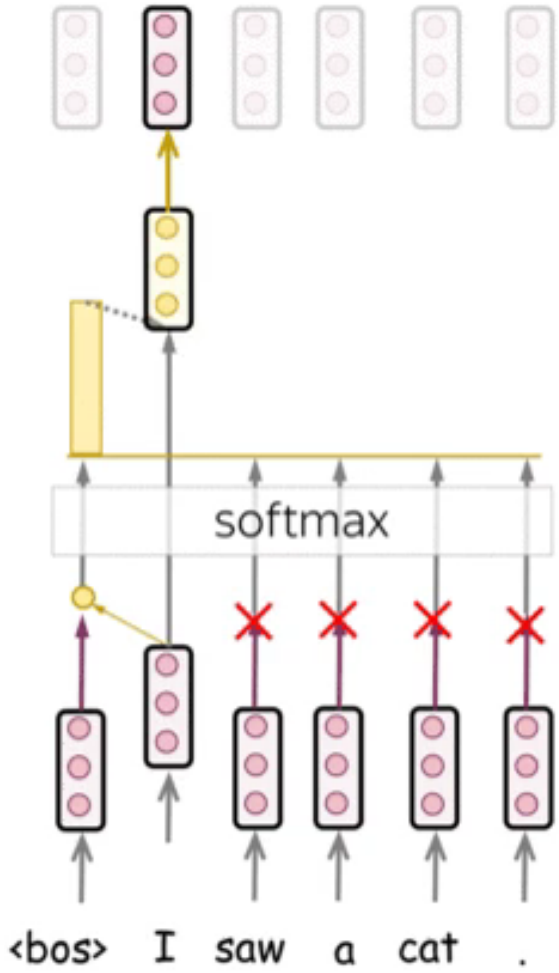
UNC CHARLOTTE

Masked Self-Attention: "Don't Look Ahead" for the Decoder

- In the decoder, there's also a self-attention mechanism: it is the one performing the "look at the previous tokens" function.
- In the decoder, self-attention is a bit different from the one in the encoder.
- While the encoder receives all tokens at once and the tokens can look at all tokens in the input sentence, in the decoder, we generate one token at a time: during generation, we don't know which tokens we'll generate in future.
- To forbid the decoder to look ahead, the model uses masked self-attention: future tokens are masked out.

Masked Self-Attention: "Don't Look Ahead" for the Decoder

Look at the illustration



update token representation

gather context

"look" at the previous tokens (future tokens are masked out)

UNC CHARLOTTE

Decoder look ahead during training

- During generation, it can't - we don't know what comes next.
- But in training, we use reference translations (which we know). Therefore, in training, we feed the whole target sentence to the decoder - without masks, the tokens would "see future", and this is not what we want.

- This is done for computational efficiency: the Transformer does not have a recurrence, so all tokens can be processed at once. This is one of the reasons it has become so popular for machine translation - it's much faster to train than the once dominant recurrent models.

- For recurrent models, one training step requires O(len(source) + len(target)) steps, but for Transformer, it's O(1), i.e. constant.

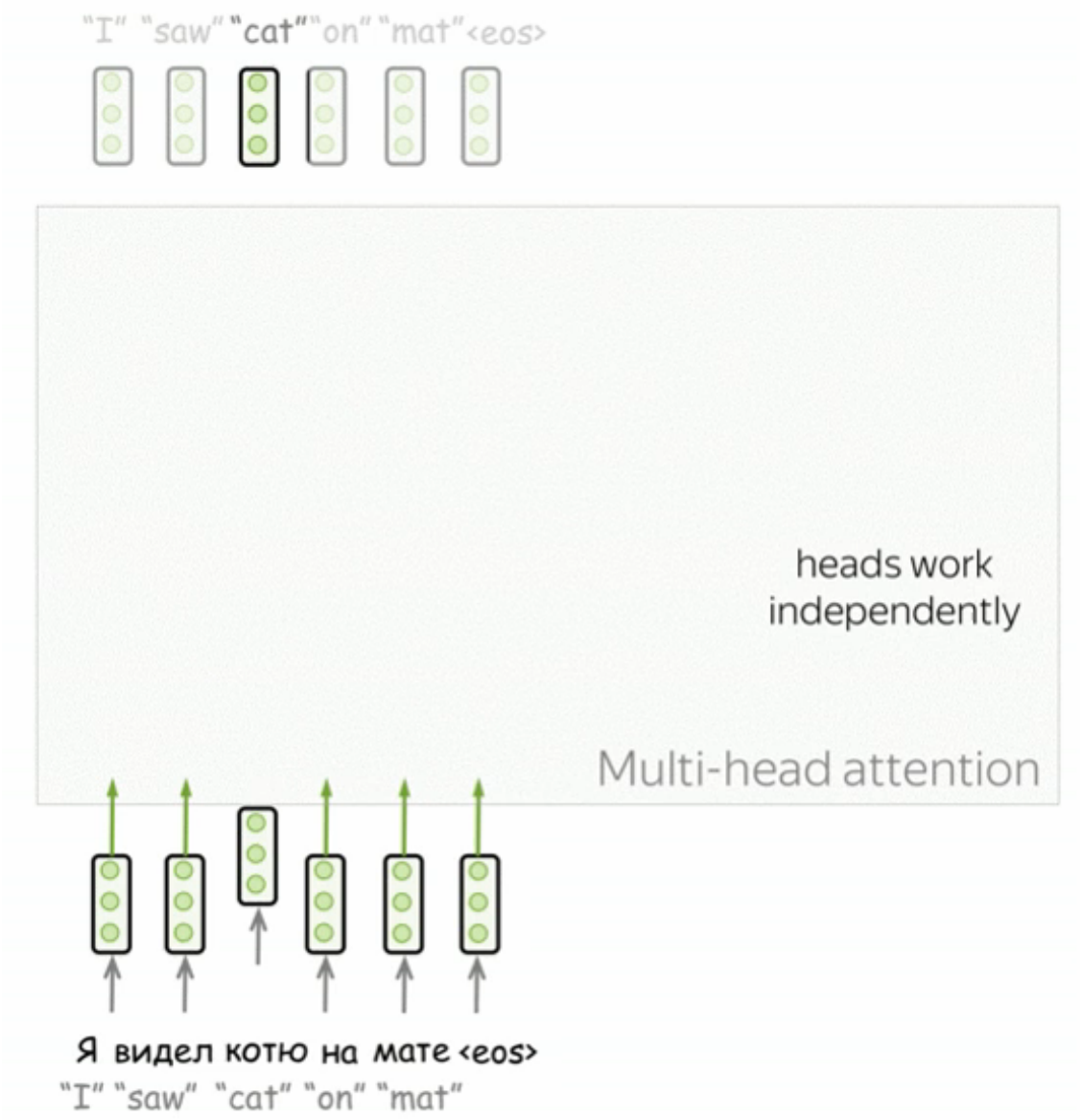
Multi-Head Attention: Independently Focus on Different Things

- Usually, understanding the role of a word in a sentence requires understanding how it is related to different parts of the sentence.
- This is important not only in processing source sentence but also in generating target.
- For example, in some languages, subjects define verb inflection (e.g., gender agreement), verbs define the case of their objects, and many more.

Each word is part of many relations.

- Therefore, we have to let the model focus on different things: this is the motivation behind Multi-Head Attention.
- Instead of having one attention mechanism, multi-head attention has several "heads" which work independently.

Multi-Head Attention

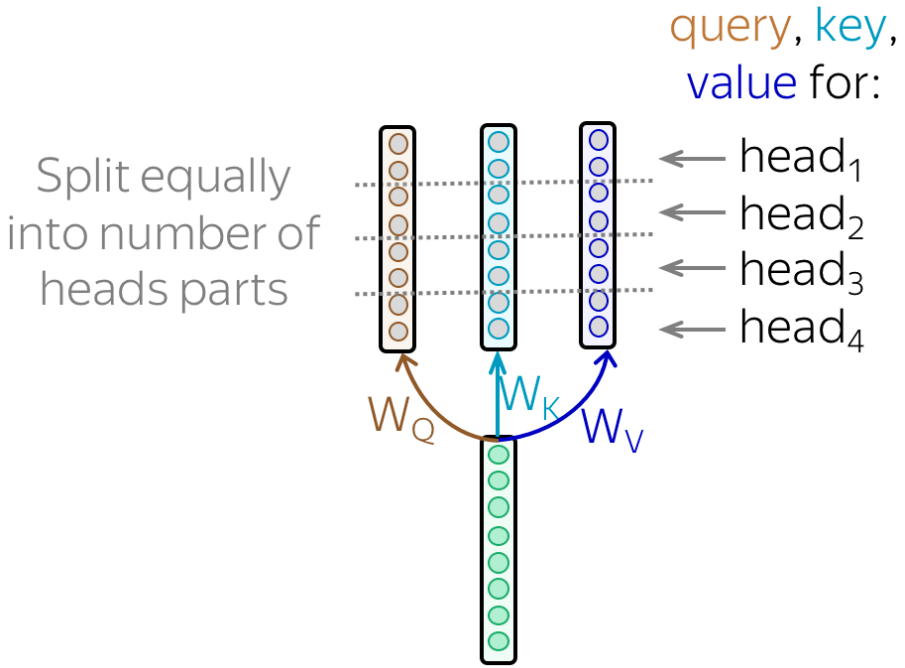


UNC CHARLOTTE

Multi-Head Attention:

- Formally, this is implemented as several attention mechanisms whose results are combined:

- In the implementation, you just split the queries, keys, and values you compute for a single-head attention into several parts. In this way, models with one attention head or several of them have the same size - multi-head attention does not increase model size.



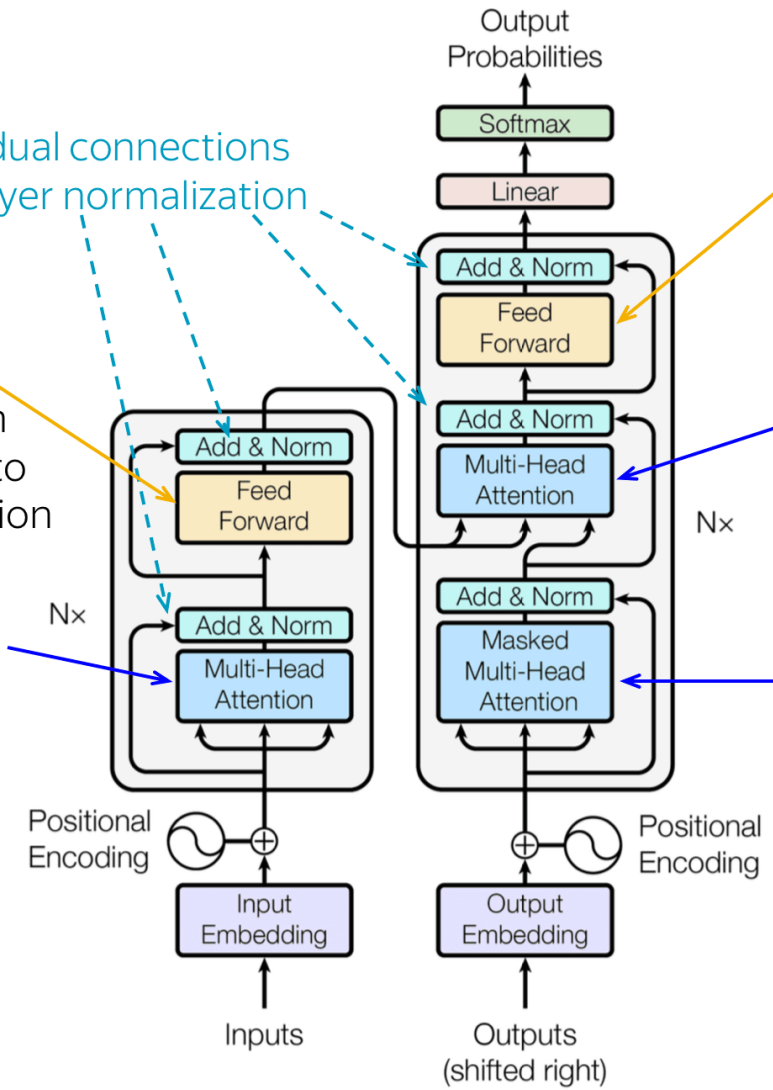
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_o,$$

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

UNC CHARLOTTE

Transformer: Model Architecture

Now, when we understand the main model components and the general idea, let's look at the whole model. The figure shows the model architecture from the original paper.



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

Transformer: Model Architecture

- Intuitively, the model does exactly what we discussed before: in the encoder, tokens communicate with each other and update their representations

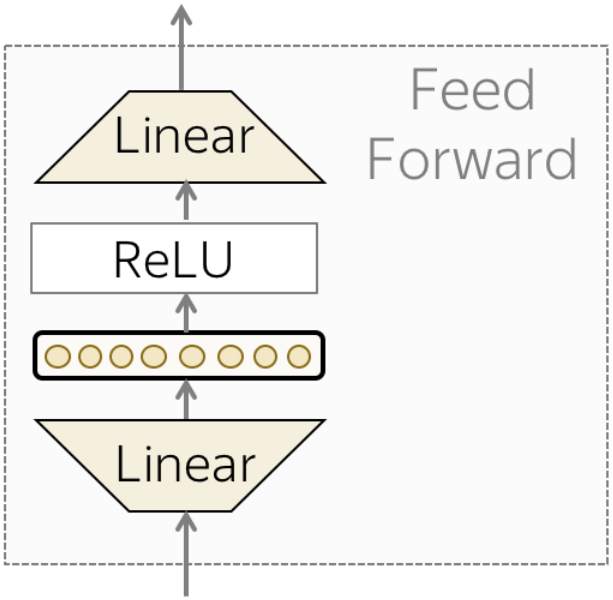
- In the decoder, a target token first looks at previously generated target tokens, then at the source, and finally updates its representation. This happens in several layers, usually 6.

Transformer: Model Architecture

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

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_n)W_o,$$

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

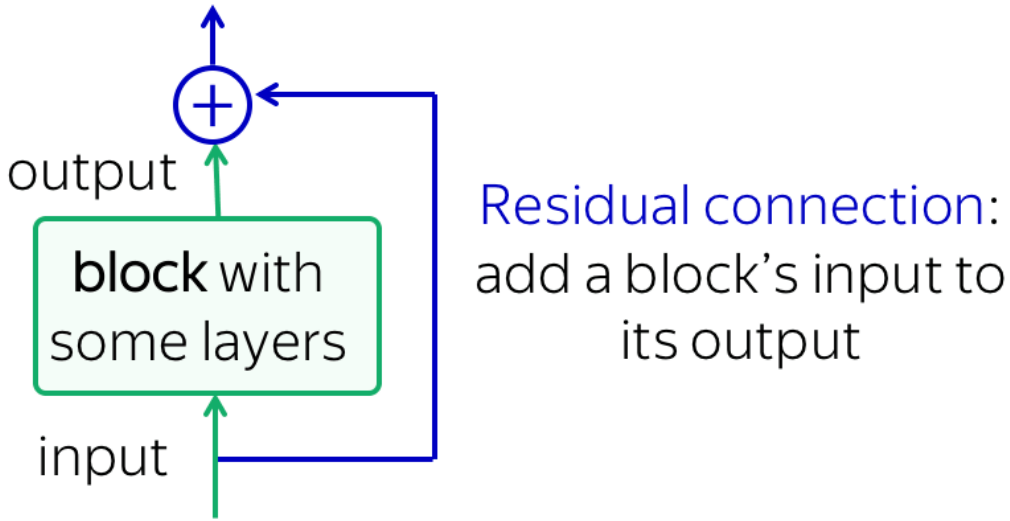


After looking at other tokens via an attention mechanism, a model uses an FFN block to process this new information (attention - "look at other tokens and gather information", FFN - "take a moment to think and process this information").

Residual connections

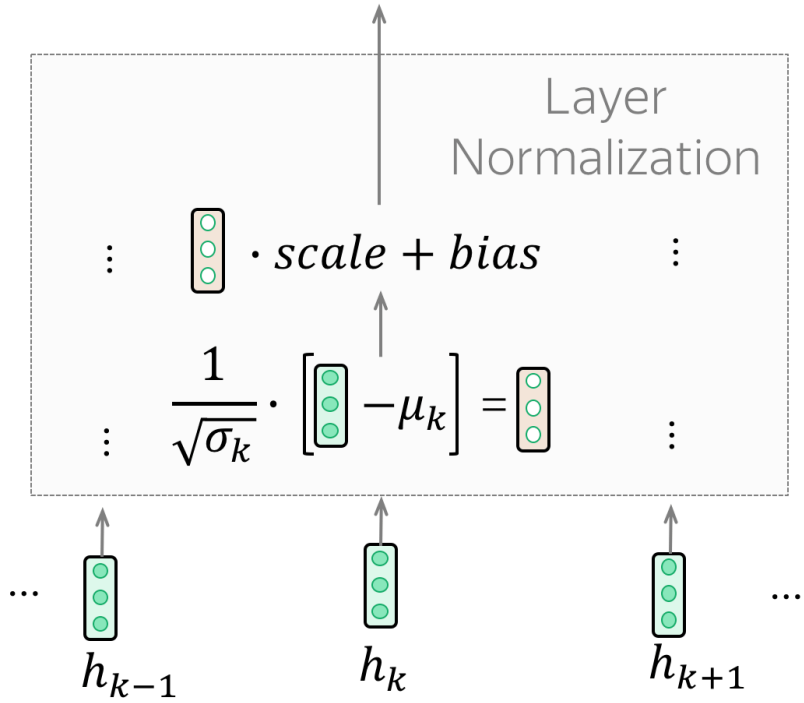
- We already saw residual connections. Residual connections are very simple (add a block's input to its output), but at the same time are very useful: they ease the gradient flow through a network and allow stacking a lot of layers.

- In the Transformer, residual connections are used after each attention and FFN block.
- On the illustration, residuals are shown as arrows coming around a block to the yellow "Add & Norm" layer.



Layer Normalization

- 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.
- In the Transformer, you have to normalize vector representation of each token.
- Additionally, here LayerNorm has trainable parameters, Scale and Bias, which are used after normalization to rescale layer's outputs (or the next layer's inputs).



UNC CHARLOTTE

Positional encoding

- Note that since Transformer does not contain recurrence or convolution, it does not know the order of input tokens.
- We have to let the model know the positions of the tokens explicitly.
- For this, we have two sets of embeddings:
  (1) tokens (as we always do)
  (2) positions (the new ones needed for this model)

- Then input representation of a token is the sum of two embeddings: (1) token and (2) positional.

- The positional embeddings can be learned, but the authors found that having fixed ones does not hurt the quality.

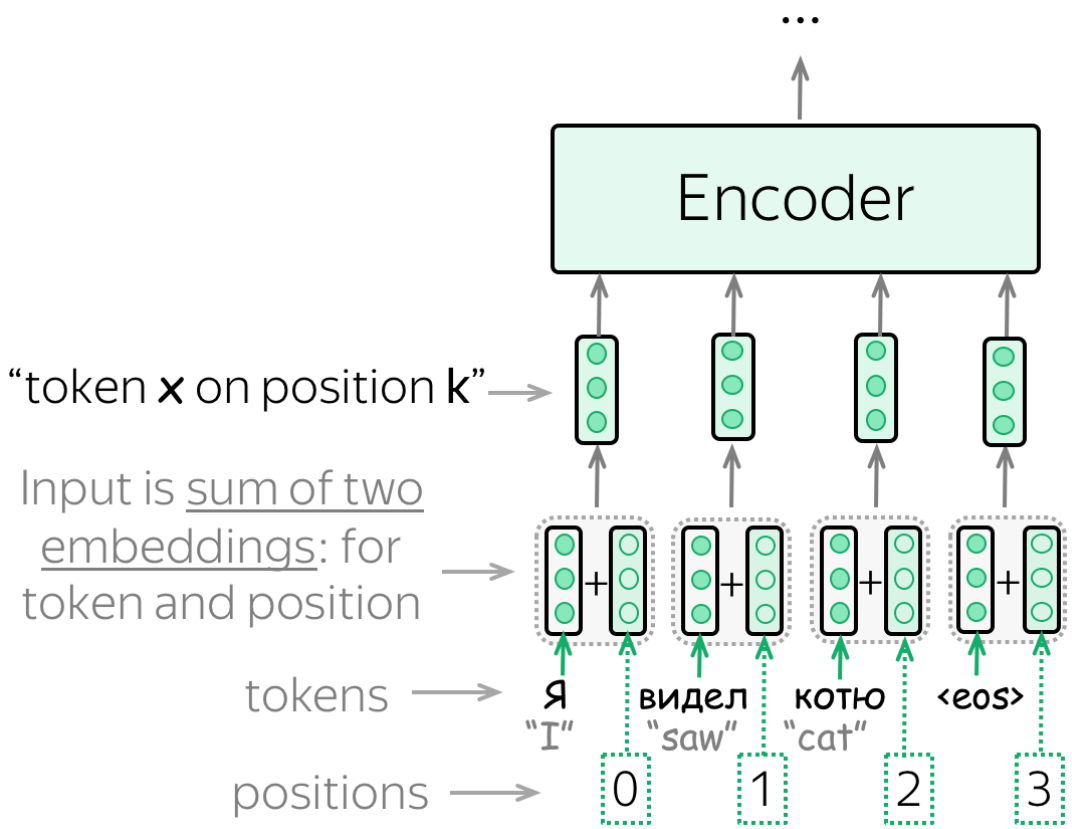
- The fixed positional encodings used in the Transformers.

Positional encoding

- Each dimension of the positional encoding corresponds to a sinusoid, and the wavelengths form a geometric progression from 2π to 10000 · 2π

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

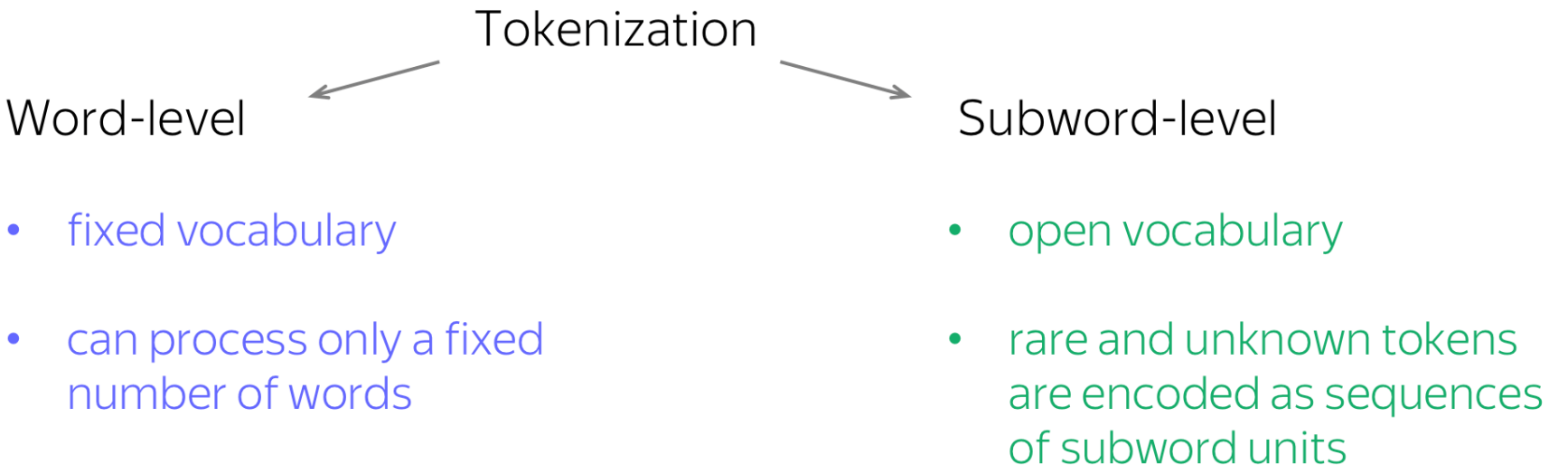
$$\mathrm{PE}_{pos,2i+1} = \cos(pos/10000^{2i/d_{model}}),$$



UNC CHARLOTTE

Subword Segmentation: Byte Pair Encoding

- As we know, a model has a predefined vocabulary of tokens.
- Those input tokens, which are not in the vocabulary, will be replaced with a special UNK ("unknown") token.
- Therefore, if you use the straightforward word-level tokenization (i.e., your tokens are words), you will be able to process a fixed number of words.
- This is the fixed vocabulary problem: you will be getting lot's of unknown tokens, and your model won't translate them properly.



Subword Segmentation: Byte Pair Encoding

- How can we represent all words, even those we haven't seen in the training data?
- Well, even if you are not familiar with a word, you are familiar with the parts it consists of - subwords (in the worst case, symbols).

- Then why don't we split the rare and unknown words into smaller parts?
- This is exactly what was proposed in the paper Neural Machine Translation of Rare Words with Subword Units by Rico Sennrich, Barry Haddow and Alexandra Birch.
  - They introduced the de-facto standard subword segmentation, Byte Pair Encoding (BPE).
  - BPE keeps frequent words intact and splits rare and unknown ones into smaller known parts.