Foundations for Natural Language Processing

Transformer (part 2)

Ivan Titov
(with graphics/materials from Elena Voita)

School of informatics

Recap: 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

Recap: Encoder-decoder attention vs self- attention

Decoder-encoder attention is looking

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

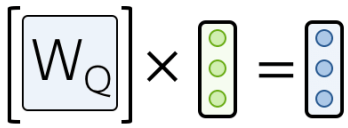
aka "cross-attention"

Self-attention is looking

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

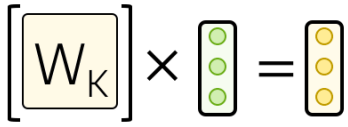
Recap: QKV Attention

Each vector receives three representations ("roles")



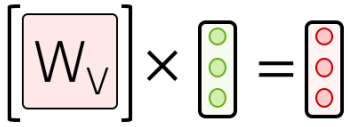
Query: vector **from** which the attention is looking

"Hey there, do you have this information?"



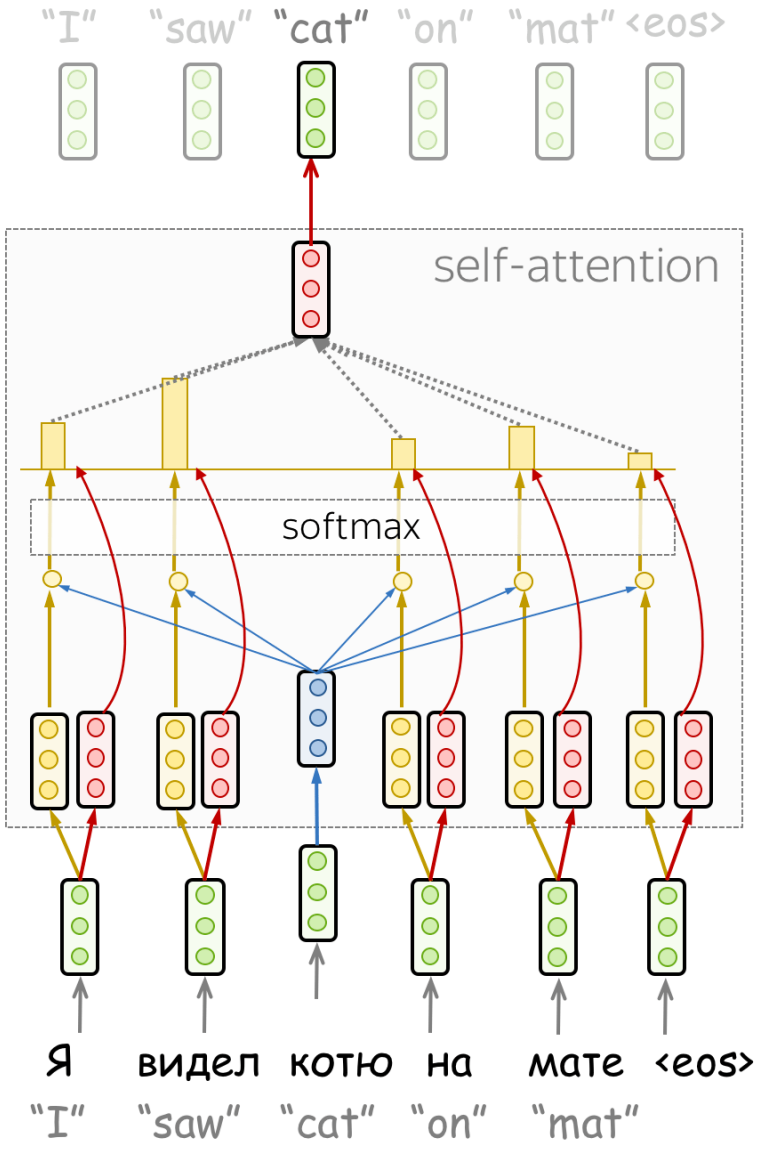
Key: vector **at** which the query looks to compute weights

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

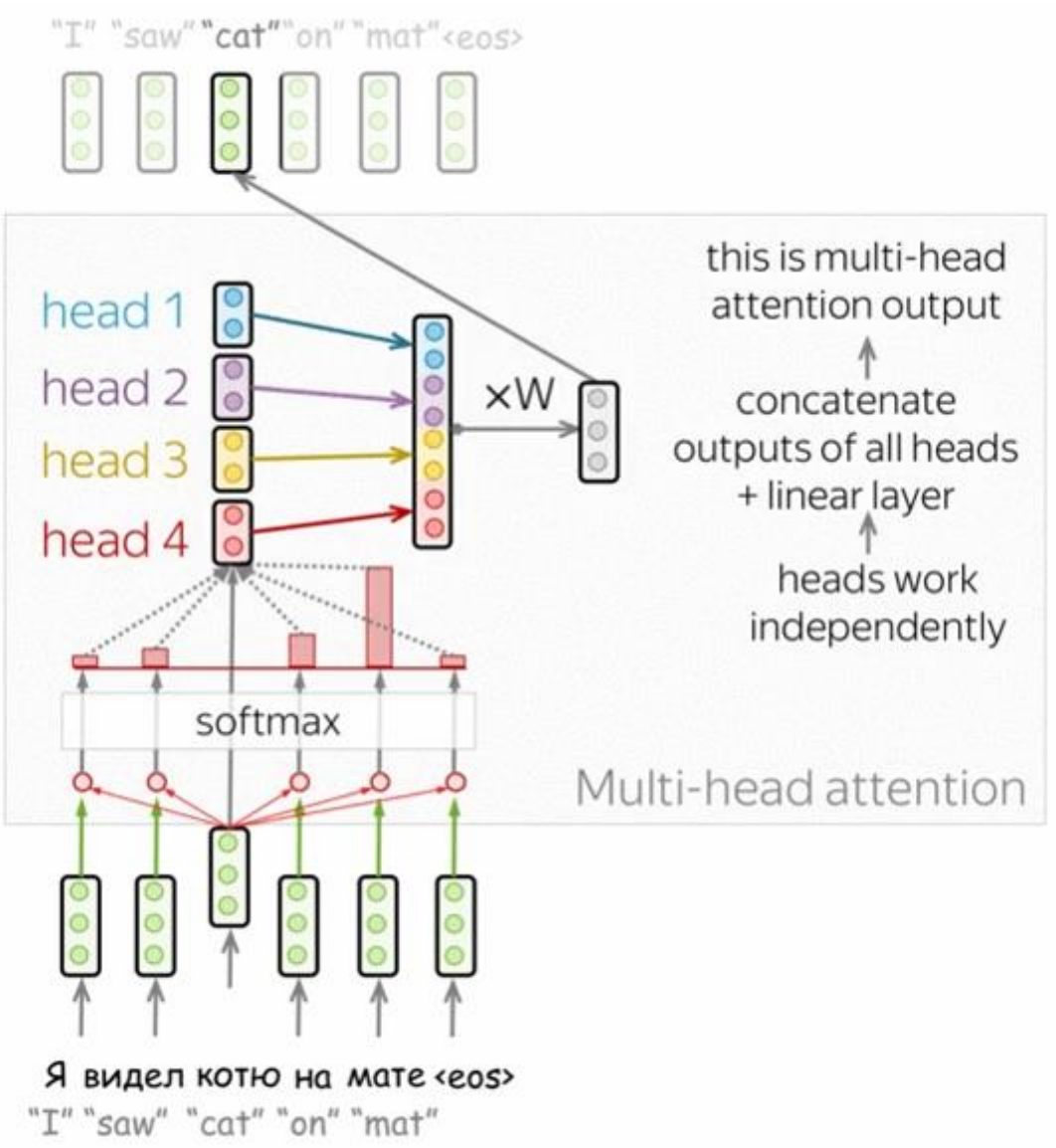


Value: their weighted sum is attention output

"Here's the information I have!"

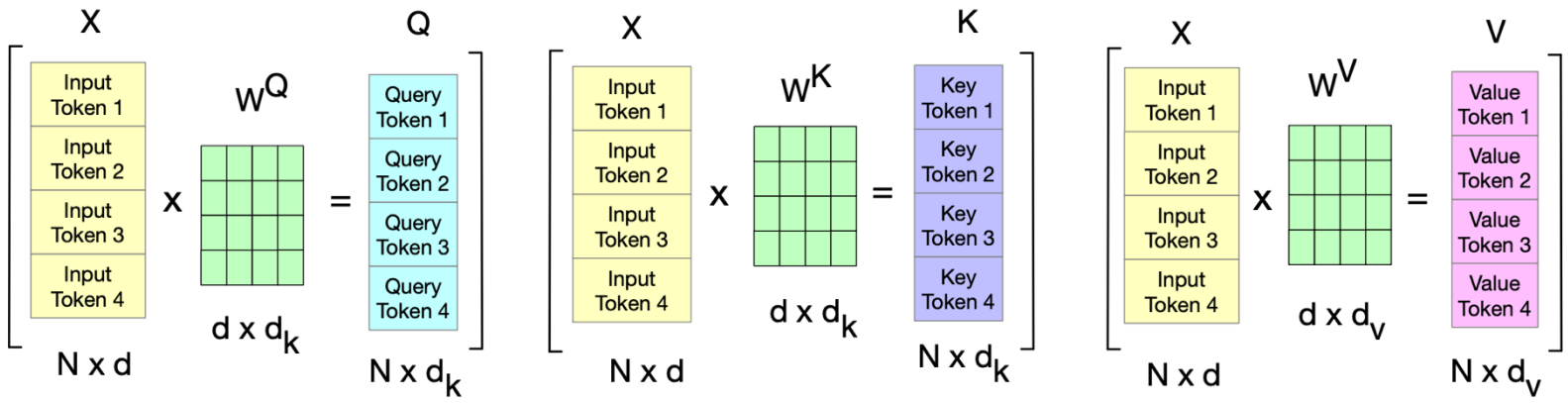


Recap: Multi-Head Attention



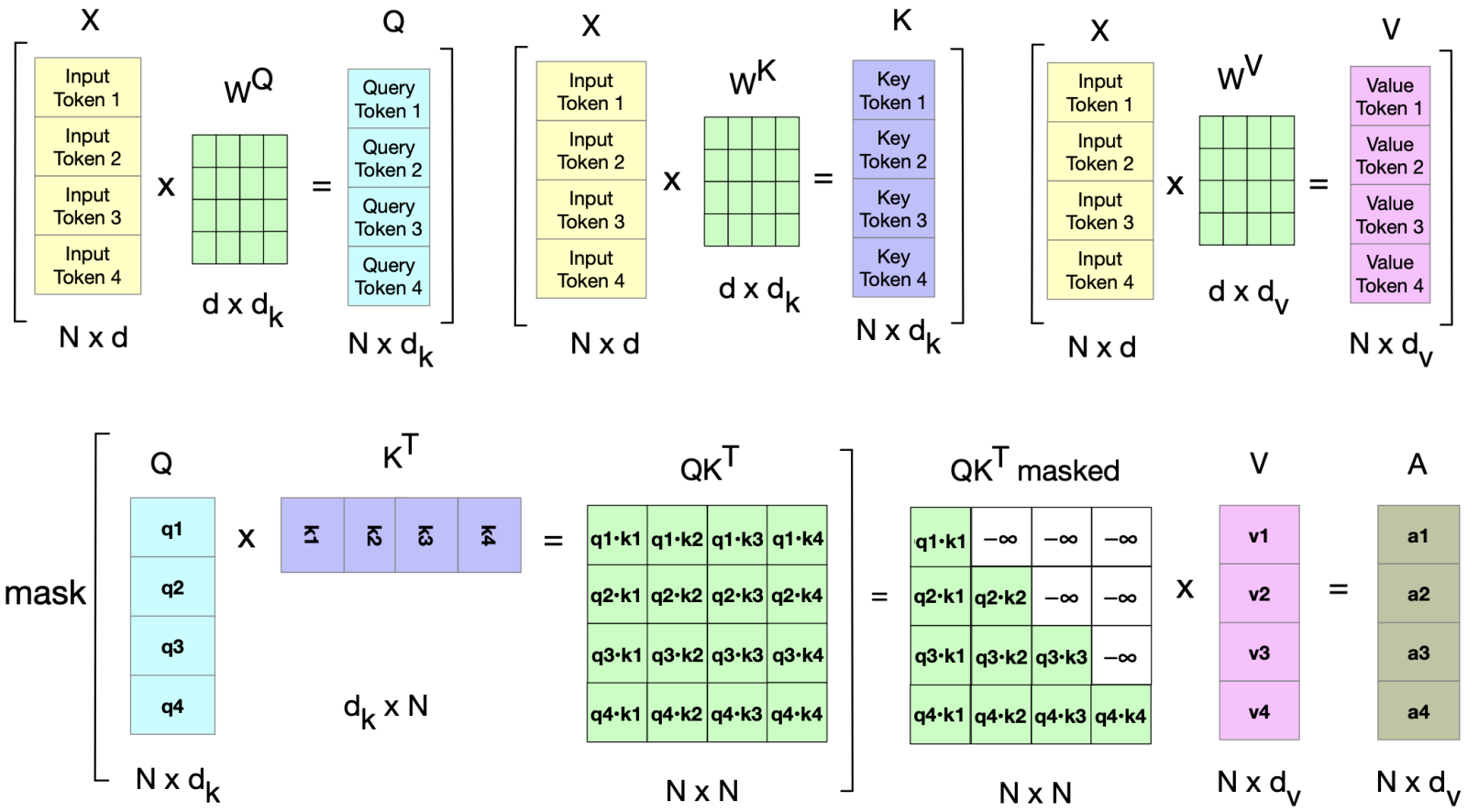
Each heads performs independent QKV attention (with their own head-specific parameters, i.e. W_k, W_q, W_v matrices)

Parallel computation for one head



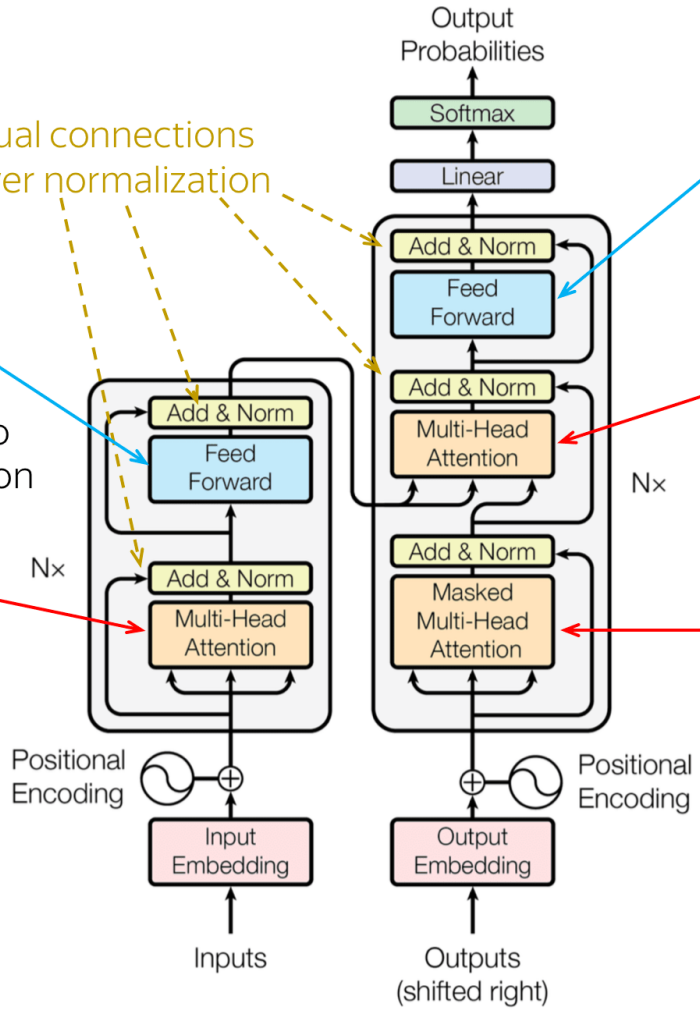
From Jurafsky and Martin

Parallel computation for one head



From Jurafsky and Martin

Transformer architecture: encoder-decoder



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

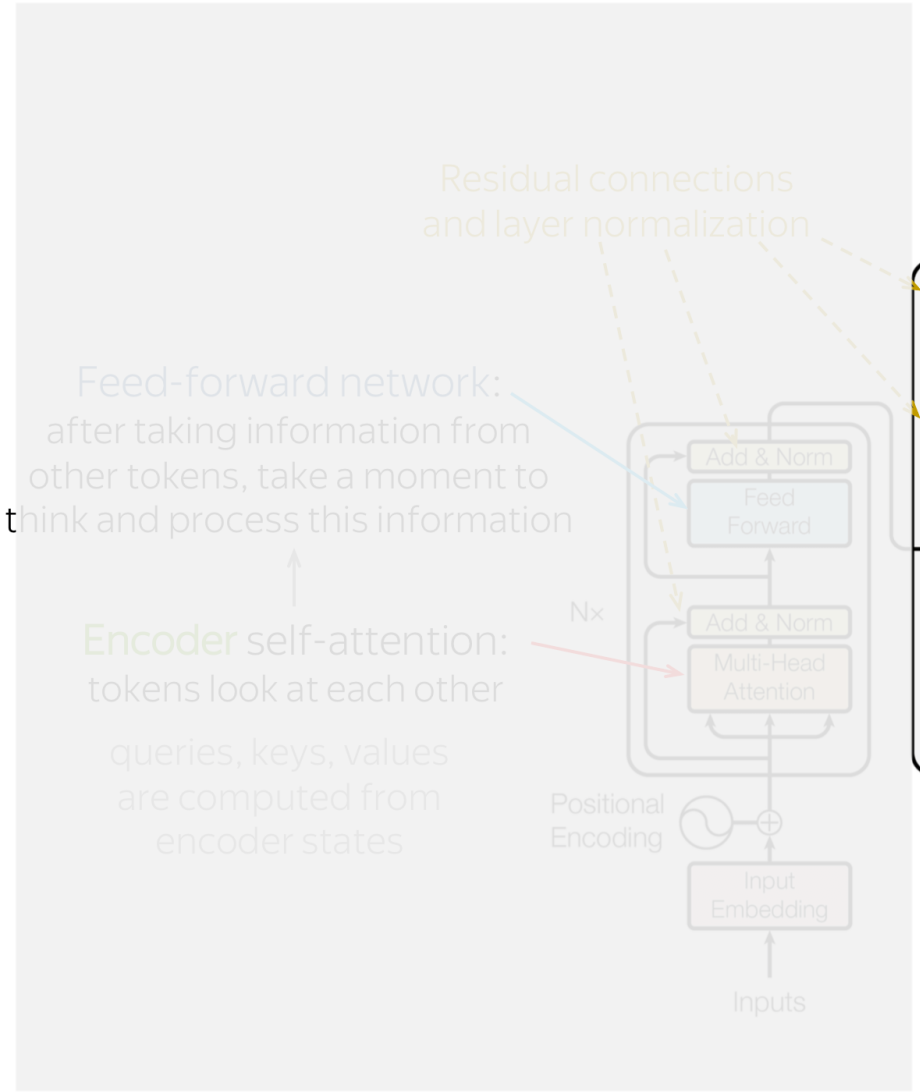
Decoder-encoder attention: target token looks at the source

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 architecture: language model



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

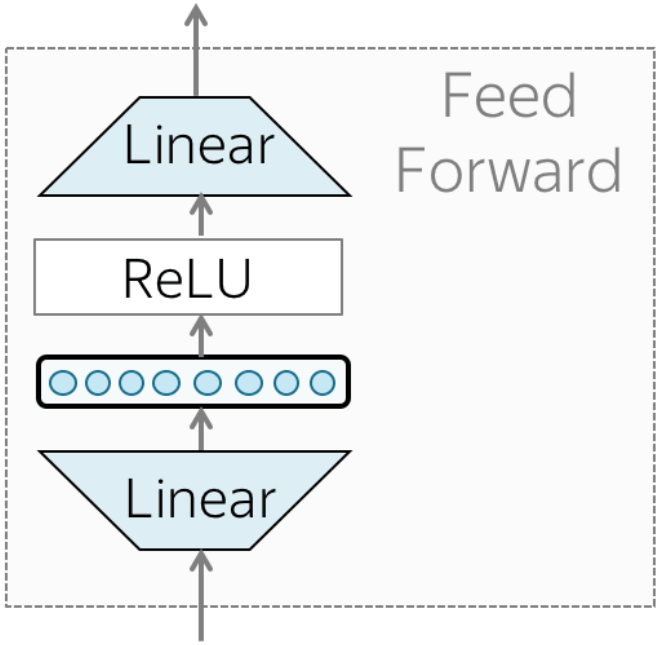
Decoder-encoder attention: target token looks at the source queries – from decoder states; keys and values from encoder states

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

queries, keys, values are computed from decoder states

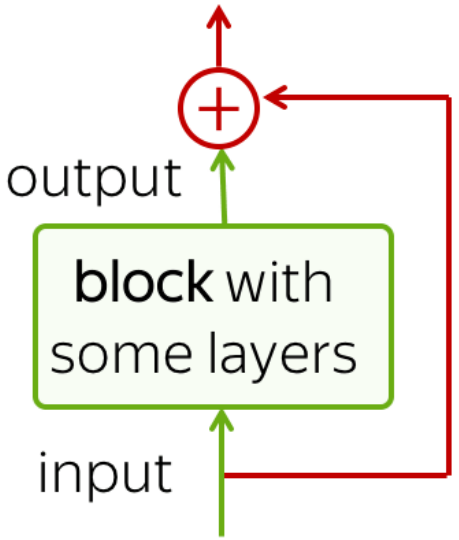
aka decoder-only model

Feedforward blocks



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

Residual connections (Recap from 2 weeks ago)



Residual connection: add a block's input to its output

Recall, we considered them earlier

Enable learning of deep architectures (i.e. many layers)

With residuals, non-adjacent modules 'communicate' between each other through the 'residual channel' or a module can directly send information to the top level

One layer – *residual stream* for a token

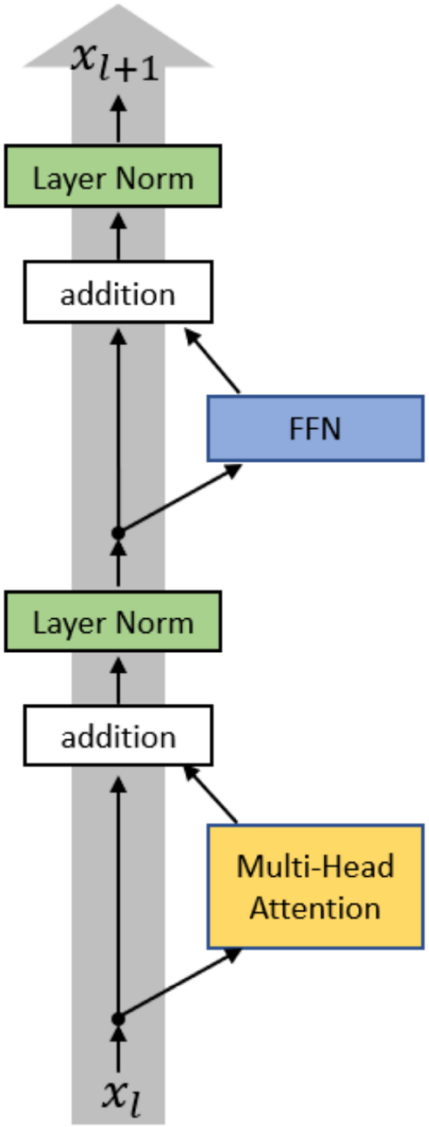
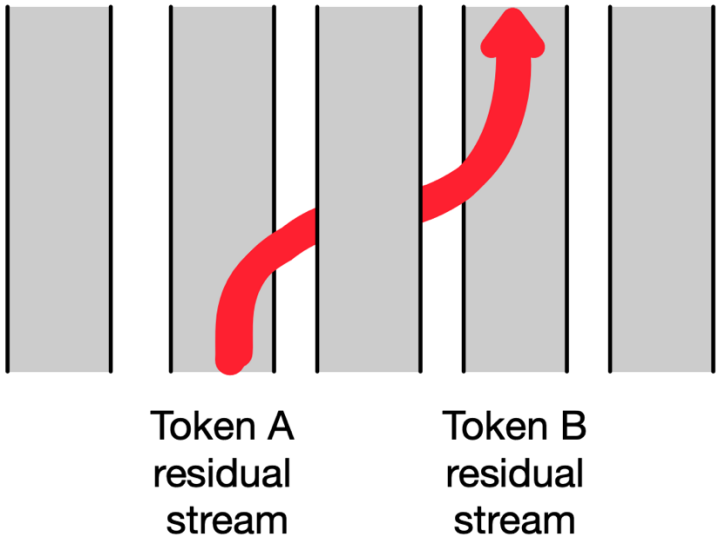


Image from Xiong et al., 2020

Residual streams

Heads move information across token-specific residual streams

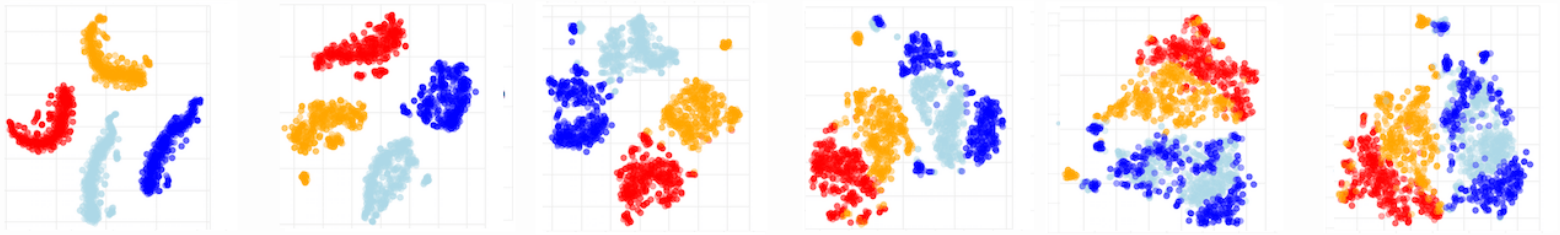


From Jurafsky & Martin

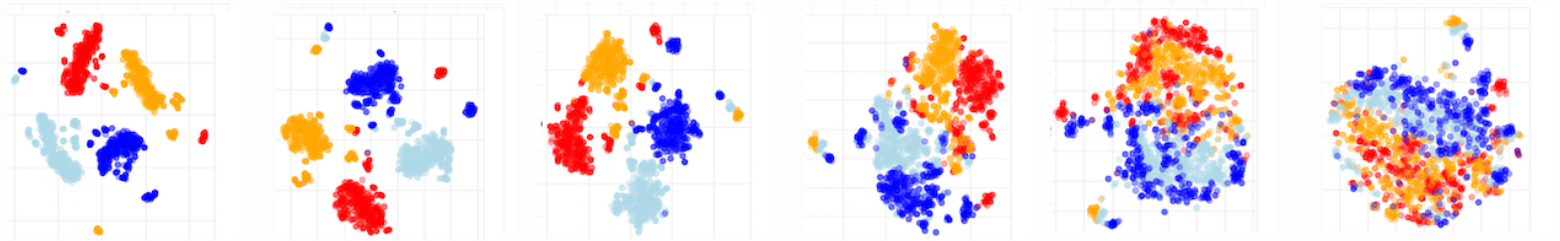
Representation of a token across layers

Representations of tokens is, are, were, was from a lot of sentences and visualized their *t-sne* projections (on the x-axis are layers).

Encoder in an MT model (Eng to Rus)



Transformer LM

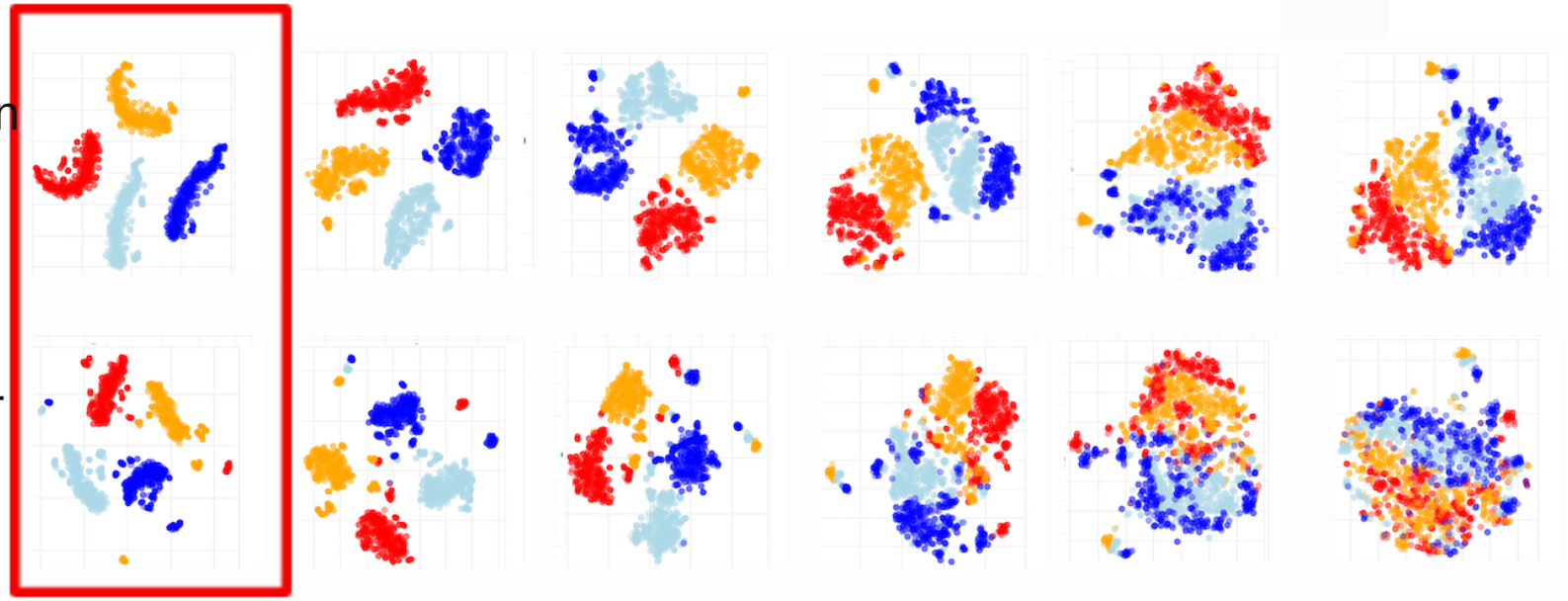


Representation of a token across layers

Representations of tokens is, are, were, was from a lot of sentences and visualized their *t-sne* projections (on the x-axis are layers).

Encoder in an MT model (Eng to Rus)

Transformer LM



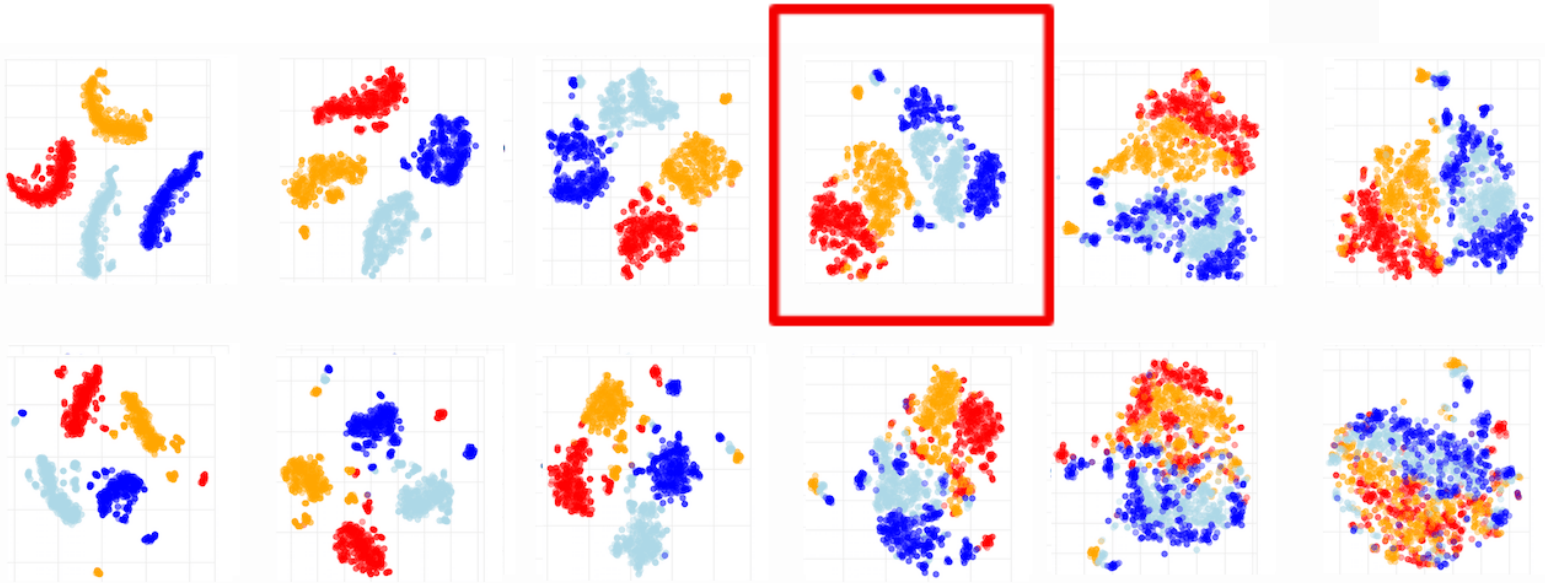
Representations of different token types in bottom layer are very distinct

Representation of a token across layers

Representations of tokens is, are, were, was from a lot of sentences and visualized their *t-sne* projections (on the x-axis are layers).

Encoder in an MT model (Eng to Rus)

Transformer LM



Any thoughts why?

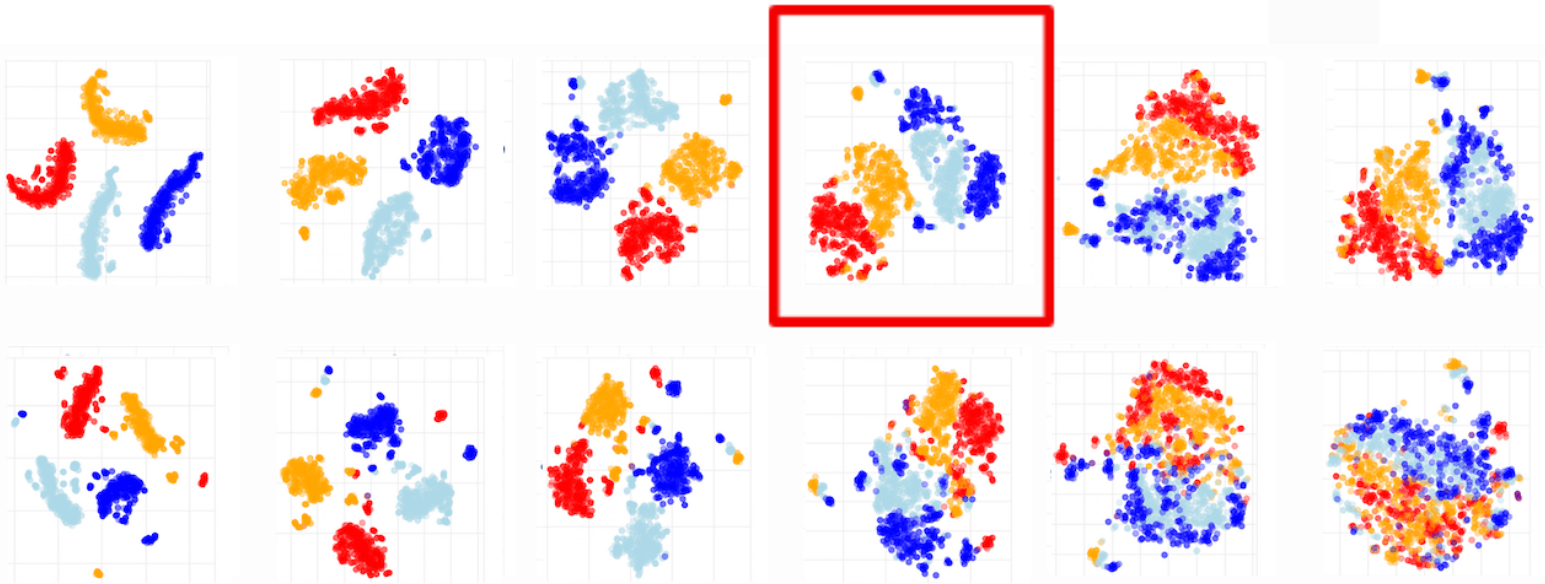
In MT encoder distinct 'clusters' for was appear in higher layers

Representation of a token across layers

Representations of tokens is, are, were, was from a lot of sentences and visualized their *t-sne* projections (on the x-axis are layers).

Encoder in an MT model (Eng to Rus)

Transformer LM



Any thoughts why?

The encoder refines the token representation to provide information for translation/decode. E.g., " was may be translated differently depending on the grammatical gender of its subject and other contextual factors.

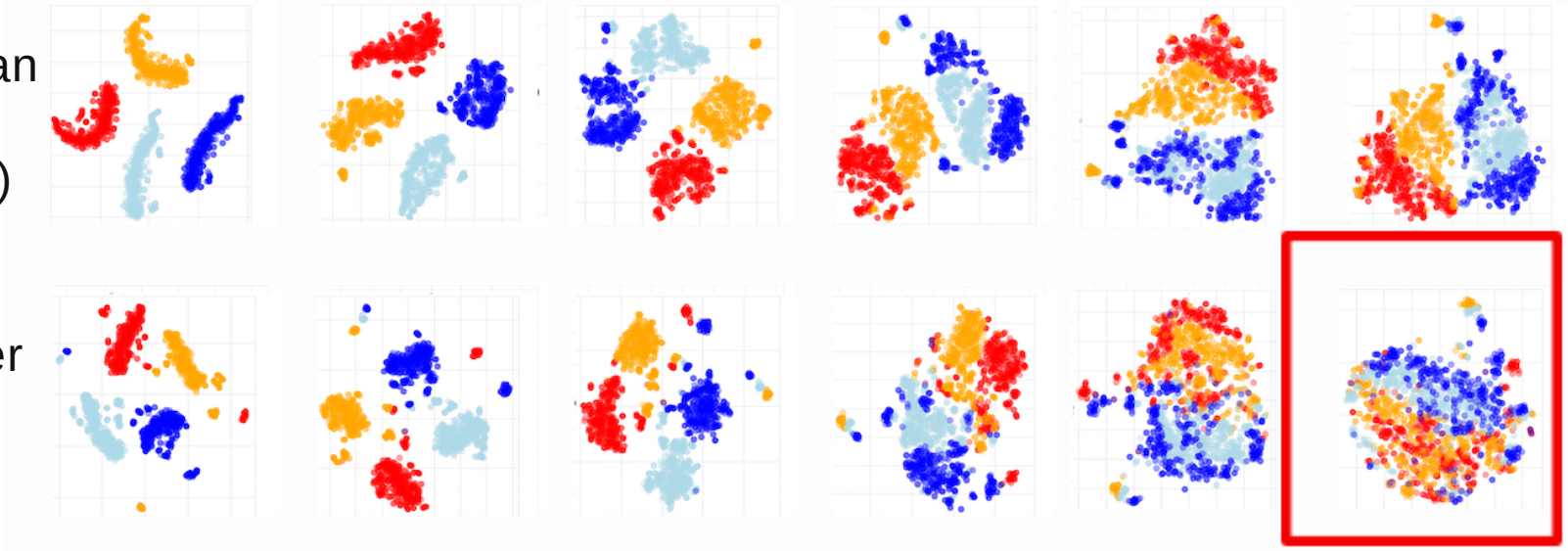
In MT encoder distinct 'clusters' for was appear in higher layers

Representation of a token across layers

Representations of tokens is, are, were, was from a lot of sentences and visualized their *t-sne* projections (on the x-axis are layers).

Encoder in an MT model (Eng to Rus)

Transformer LM



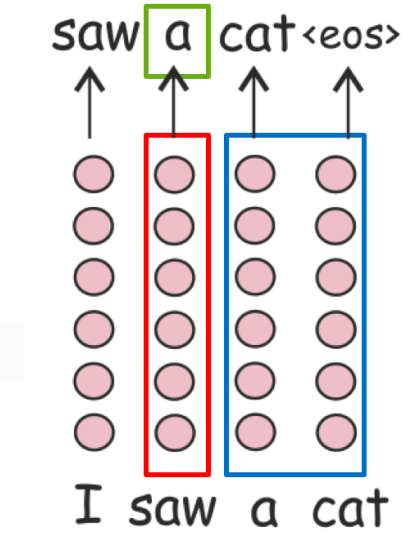
Any thoughts why?

In LM, the tokens get 'mixed' in higher layers

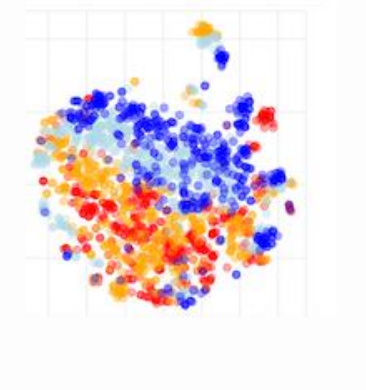
Representation of a token across layers

The information of a token is refined due to the two roles it plays:

- predicting the output label from a current token representation;
- maintaining information necessary to build representations of other tokens.

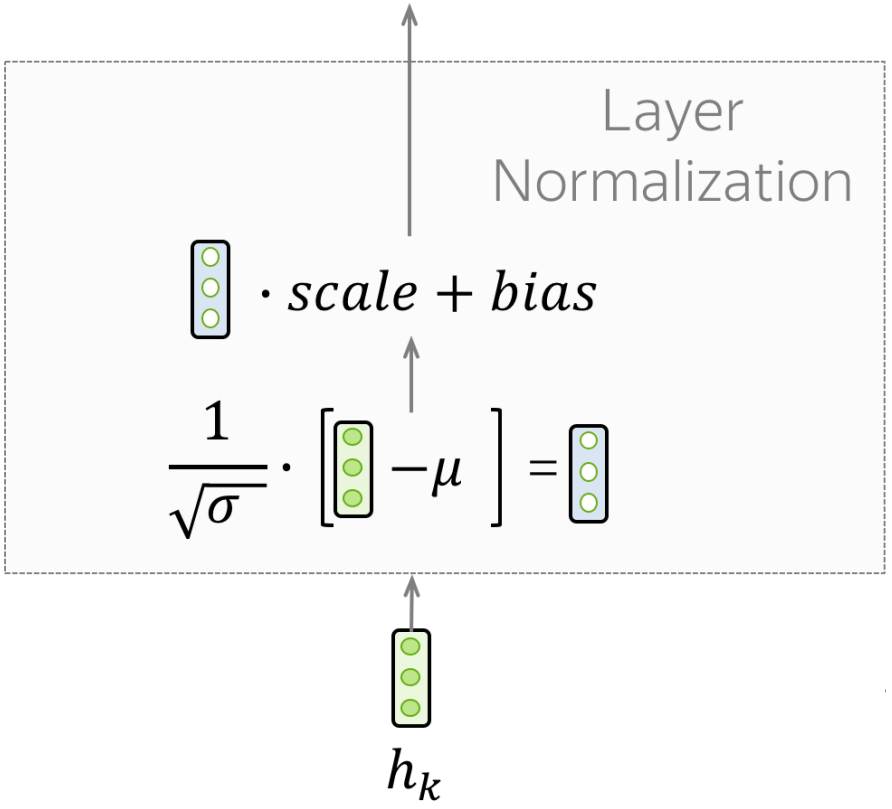


So, in top layers of a LM, the token representation focuses primarily on what the next token is likely to be rather than what the current token is



Layer Normalization

LayerNorm improves training stability (but there are alternatives to LayerNorm)



μ - is the mean of a token representation h_k (across its dimension)

σ - is the standard deviation, computed analogously

scale and *bias* are trainable parameters

Dropout

- A *regularization* technique that randomly drops units (neurons) during training.
- Helps prevent overfitting by reducing co-adaptation of neurons.
- Active only during training; during inference, all neurons are used *with scaled weights*.

Dropout: formally

Each neuron is retained with prob *p*, otherwise it is set to 0.

$$\tilde{h}_k = h_k \cdot m_k, \quad m_k \sim \text{Bernoulli}(p)$$

where:

h_k is the input activation

m_k is a binary mask sampled from a Bernoulli distribution with

$\tilde{h}_k$ is the resulting activation after dropout

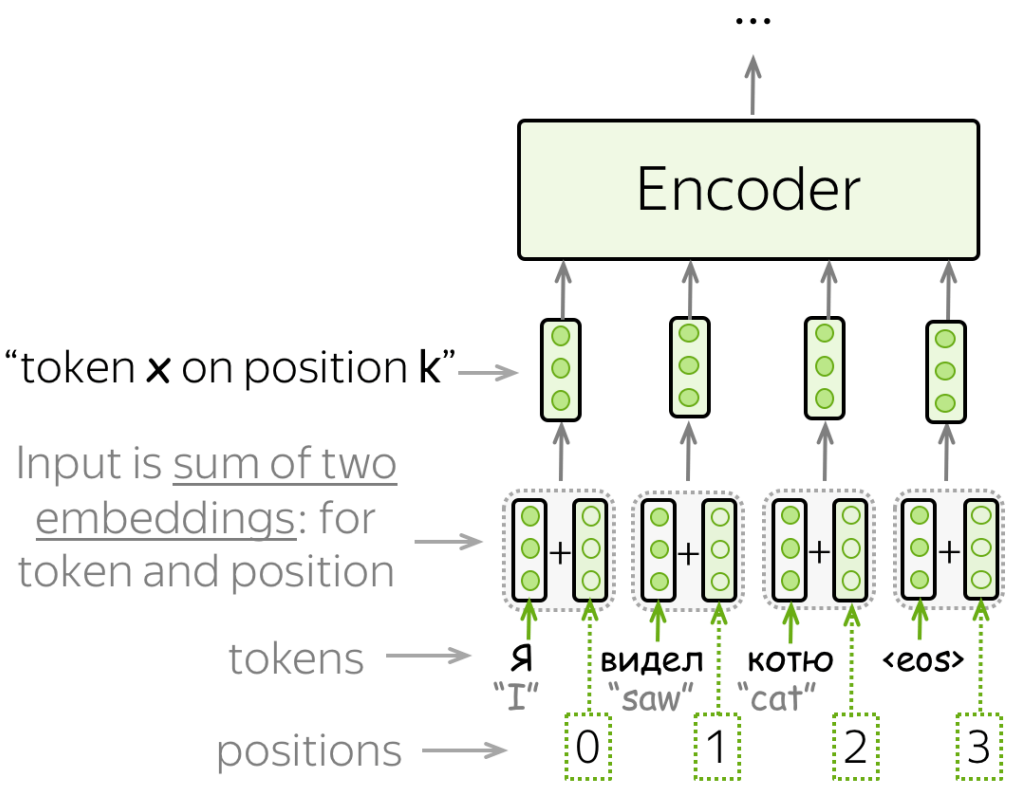
At test time, the layer is rescaled to have the same magnitude:

$$\tilde{h}_k = h_k \cdot p$$

In Transformer, Dropout can be used after MLP and Attention Layers

Positional Encoding

Transformers (unlike RNNs) do not have a notion of order of the tokens. To incorporate information about the order, we use '*position embeddings*'



How to represent position as vectors?

Learnable position embeddings: similarly to word vectors, a unique vector is learned for each position.

Issues with this approach?

How to represent position as vectors?

Learnable position embeddings: similarly to word vectors, a unique vector is learned for each position.

Issues with this approach?

- **Poor generalization to long sequences**:
  - Embeddings for high positions are oundertrained, leading to poor performance on long contexts.

- **Not suitable for long-context Transformers**:
  - modern applications require handling thousands of tokens.

- **Incorrect inductive biases**:
  - In NLP, **relative position** matters more than absolute one.
  - *E.g.*: The model should behave for token pairs (**20, 25)** and (**140, 145)** but completely different emb parameters determine the relations

How to represent position as vectors?

Fixed periodic functions to compute position embeddings

For example, the original paper, they proposed the sinusoid waves

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

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

i is the index of the embedding (runs from *0* to *2 i + 2)*

The intuition

Suppose you want to represent a number in binary format

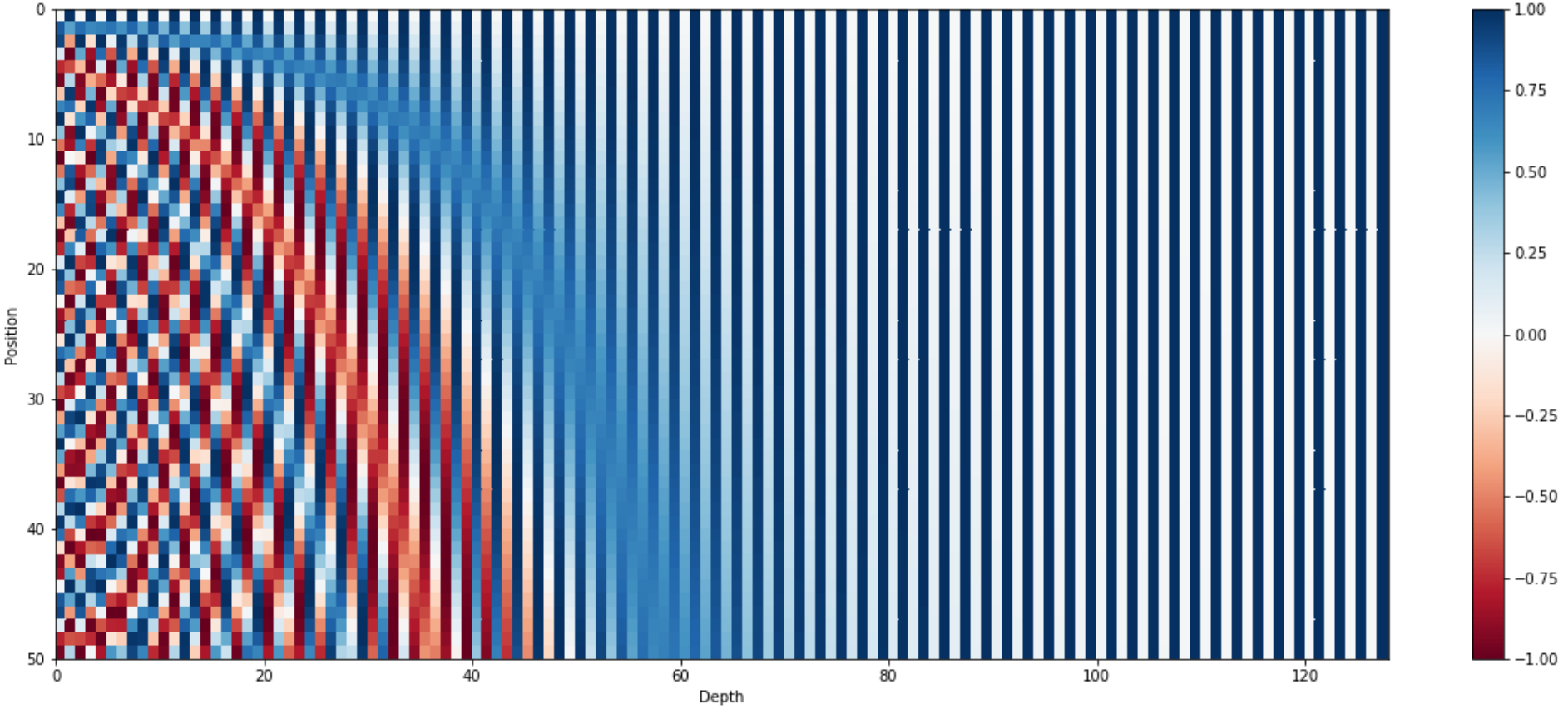
0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

You can spot the rate of change between different bits. The least significant bit is alternating on every number, the second-lowest bit is alternating (‘rotating’) on every two numbers, and so on.

The sinusoidal waves are just continuous counterpart

From https://kazemnejad.com/

The sinusoidal waves



From https://kazemnejad.com/

Why sinusoidal waves?

Better generalization to long sequences

- Unlike learned embeddings, they extend naturally to positions beyond the training range.

Encourage relative positioning (but does not do it so great)

- the difference between positions is easily computable for a pair of tokens due to the periodic nature of sine and cosine.
- the relative position between tokens **20** ↔ **25** and **140** ↔ **145** is encoded in a similar way (kind of), helping the model generalize better.

No extra parameters to learn and store

- These embeddings do not require additional training

More modern approaches

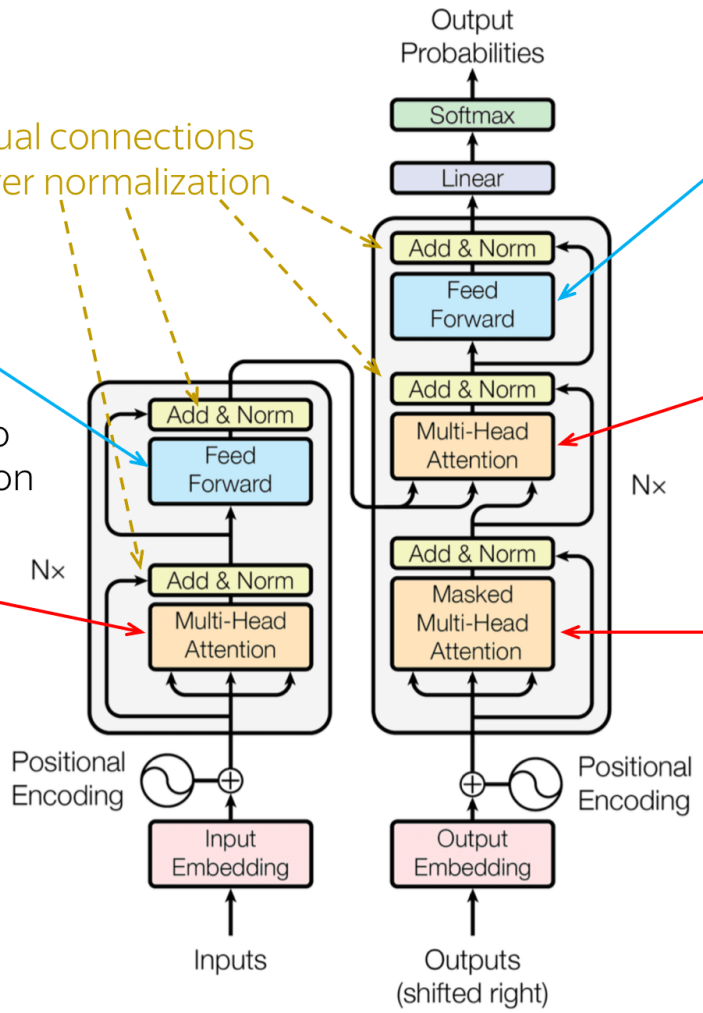
Recent generations of Transformers continue to use periodic functions but in different ways.

Rotary Positional Embeddings (RoPE) encode positional information by **rotating token embeddings** rather than adding positional vectors. So they also more explicit in encoding only the relational informatin (i.e. the distance between tokens)

RoPE's self-attention computation is function $f(x_i, i, x_j, j)$ such that:

$$f(x_i, i, x_j, j) = g(x_i, x_j, i - j),$$

We should now know every block!



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

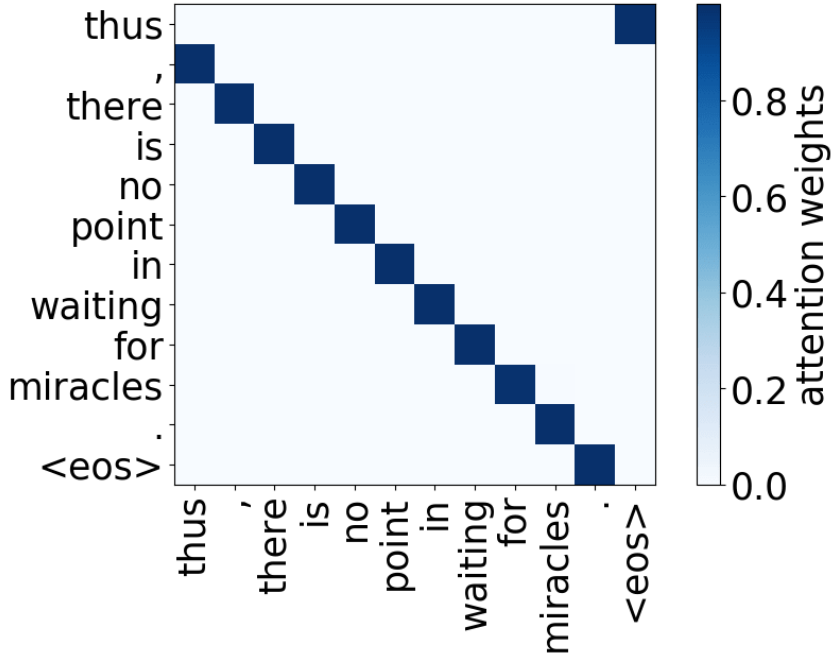
Decoder self-attention (masked):
tokens look at the previous tokens

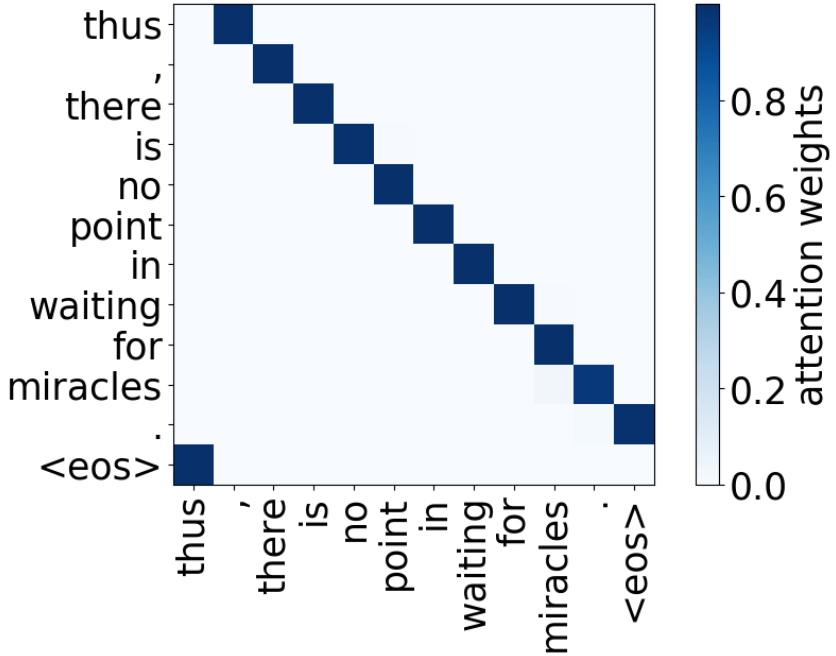
queries, keys, values are computed from decoder states

Interpretability

Recall, individual heads focus on different tokens (have their individual attention modules)

It turns out many heads have interpretable roles, the are *Position Heads* which look into adjacent positions



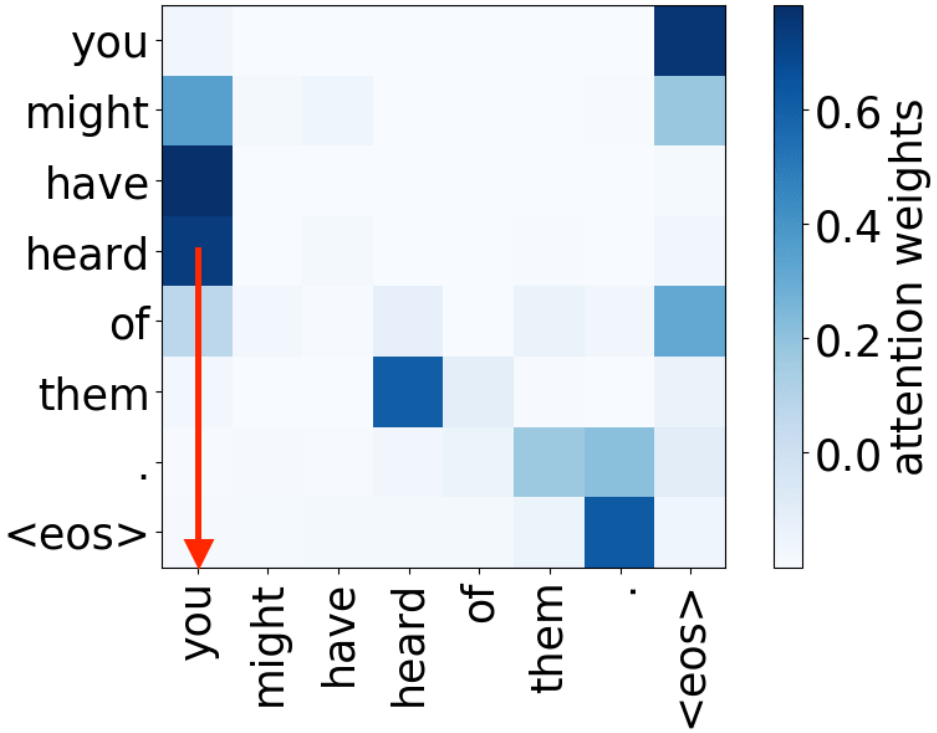


Encoder of a machine translation Transformer model

Voita et al., ACL 2019

Interpretability

Syntactic heads:



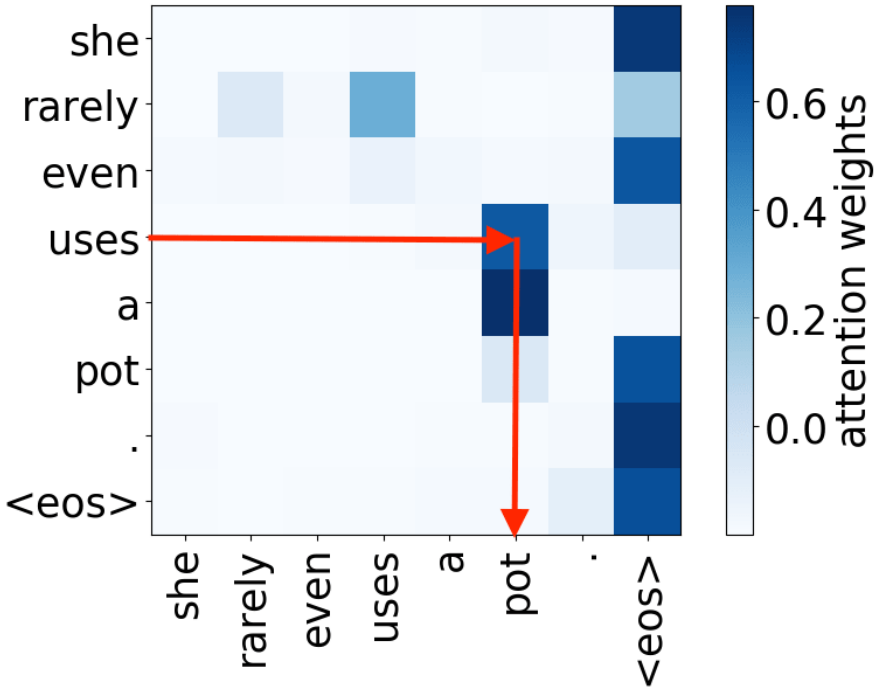
Encoder of a machine translation Transformer model

verb->subject

Voita et al., ACL 2019

Interpretability

Syntactic heads:



verb->object

Encoder of a machine translation Transformer model

Voita et al., ACL 2019

Heads roles in Encoder

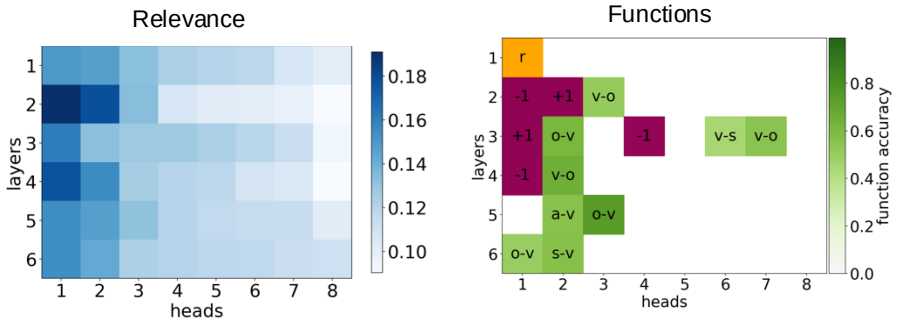
Relevance is estimated importance of the head, the heads in this plots are sorted according to their estimated importance (in each row)

positional

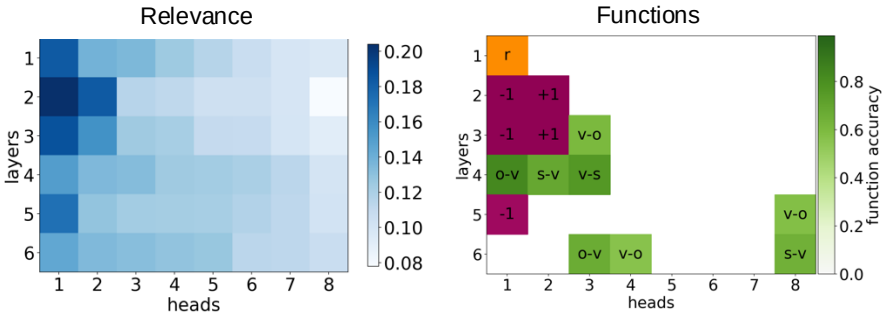
syntactic

rare tokens

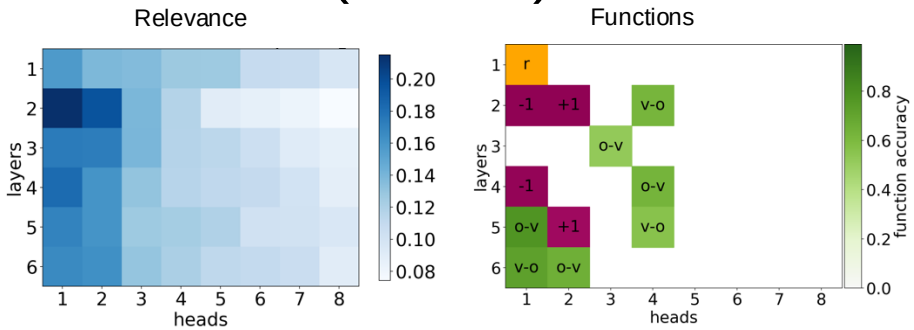
(EN-DE)



(EN-RU)



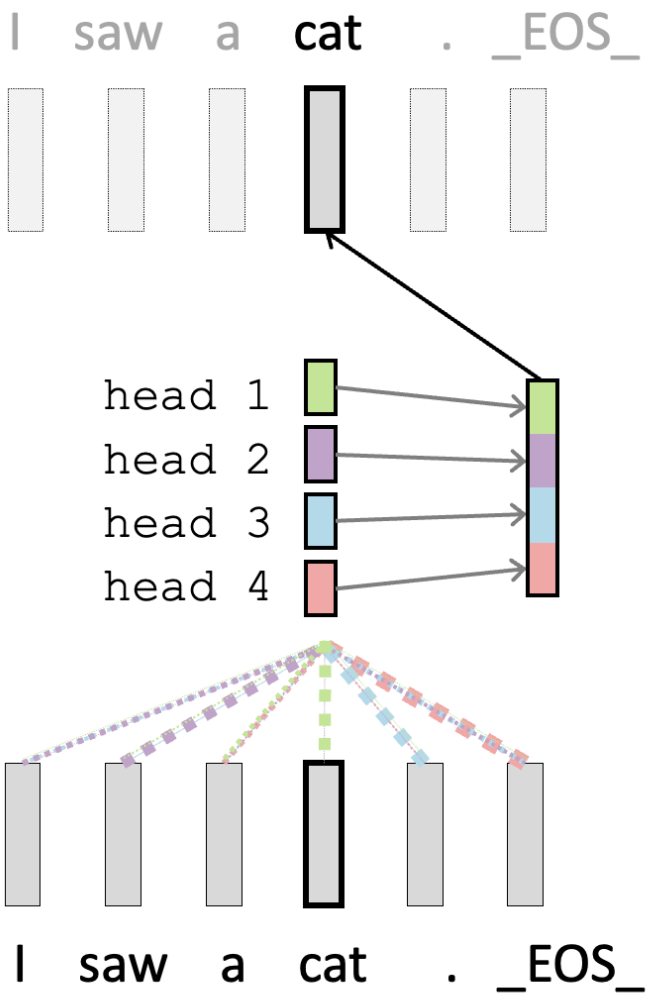
(EN-FR)



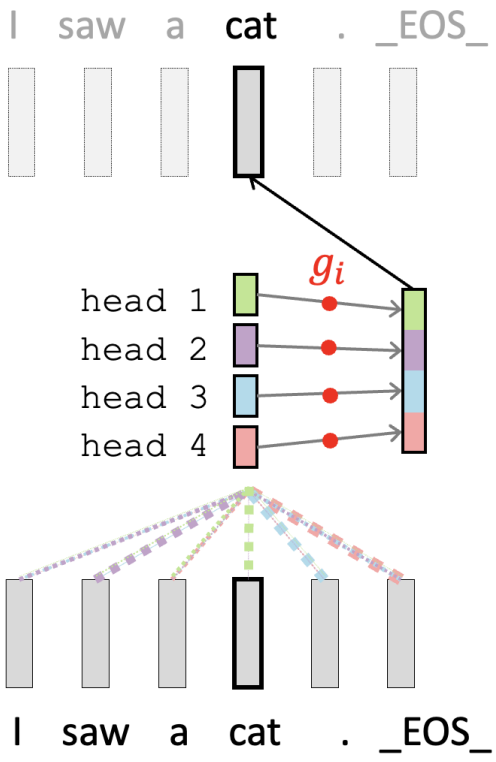
The important heads in MT encoders are specialized.

Do we need all the rest?

Original model



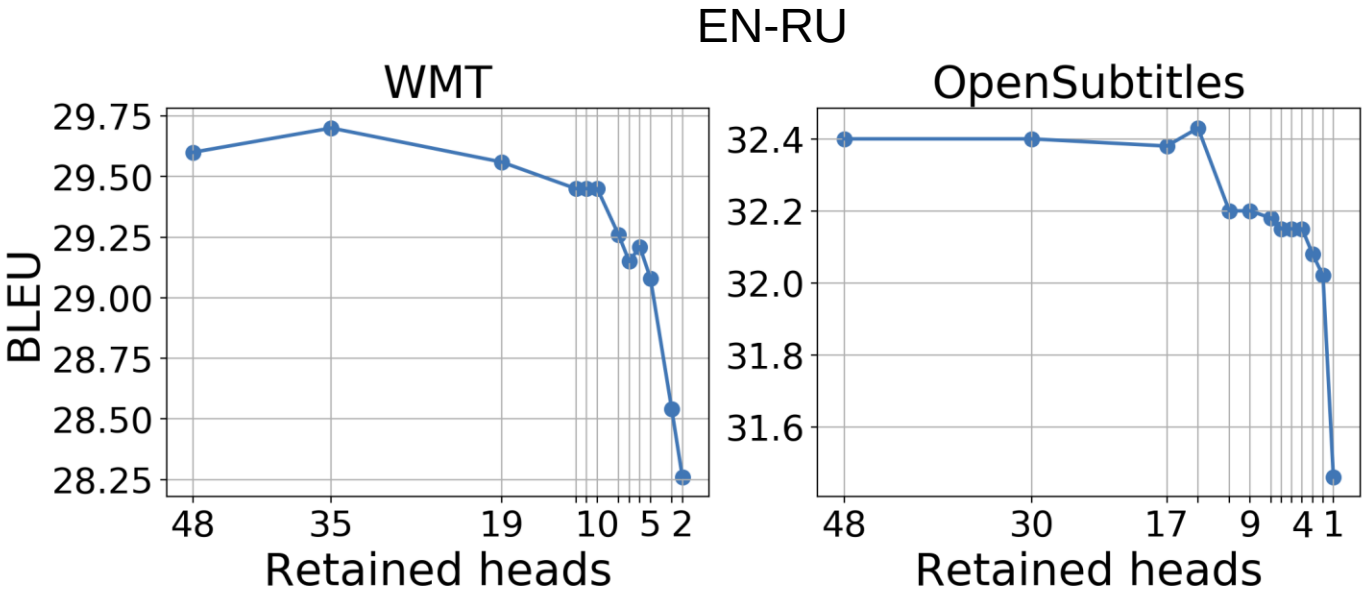
Our modification: pruning heads



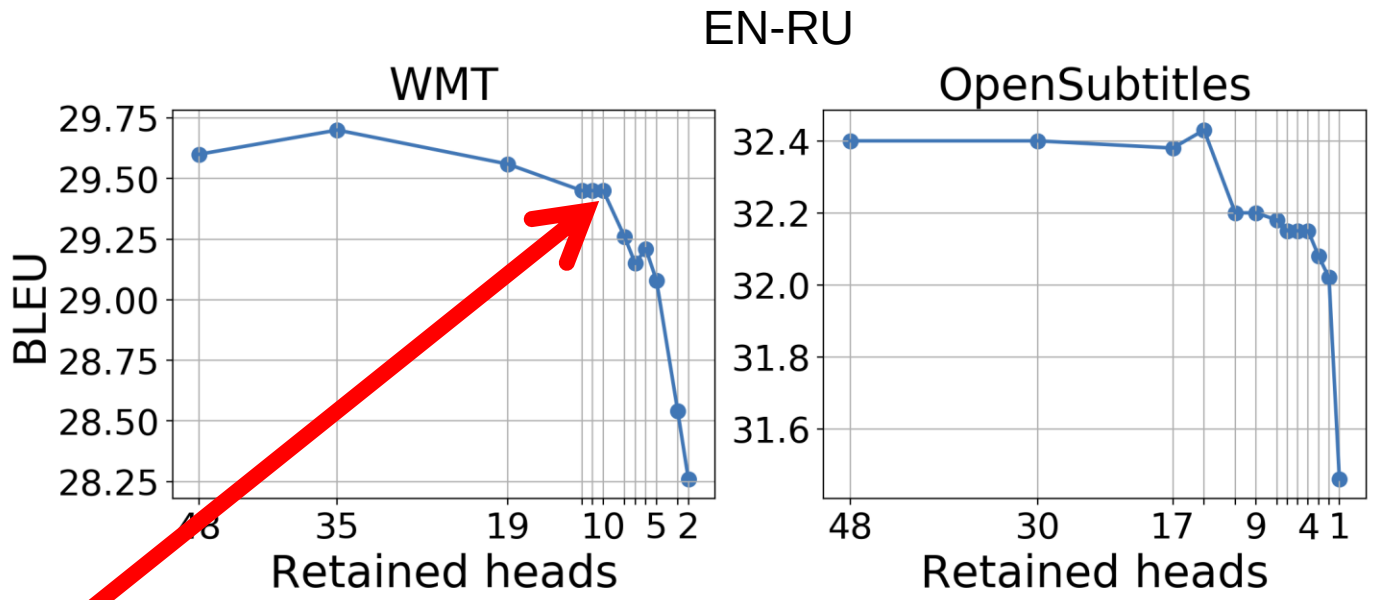
Multiply output of each head i by a scalar gate g_i

Now optimize only g_i with original cross-entropy loss on translation data + a loss pushing as many gi to exactly 0.

Pruned heads and BLEU score

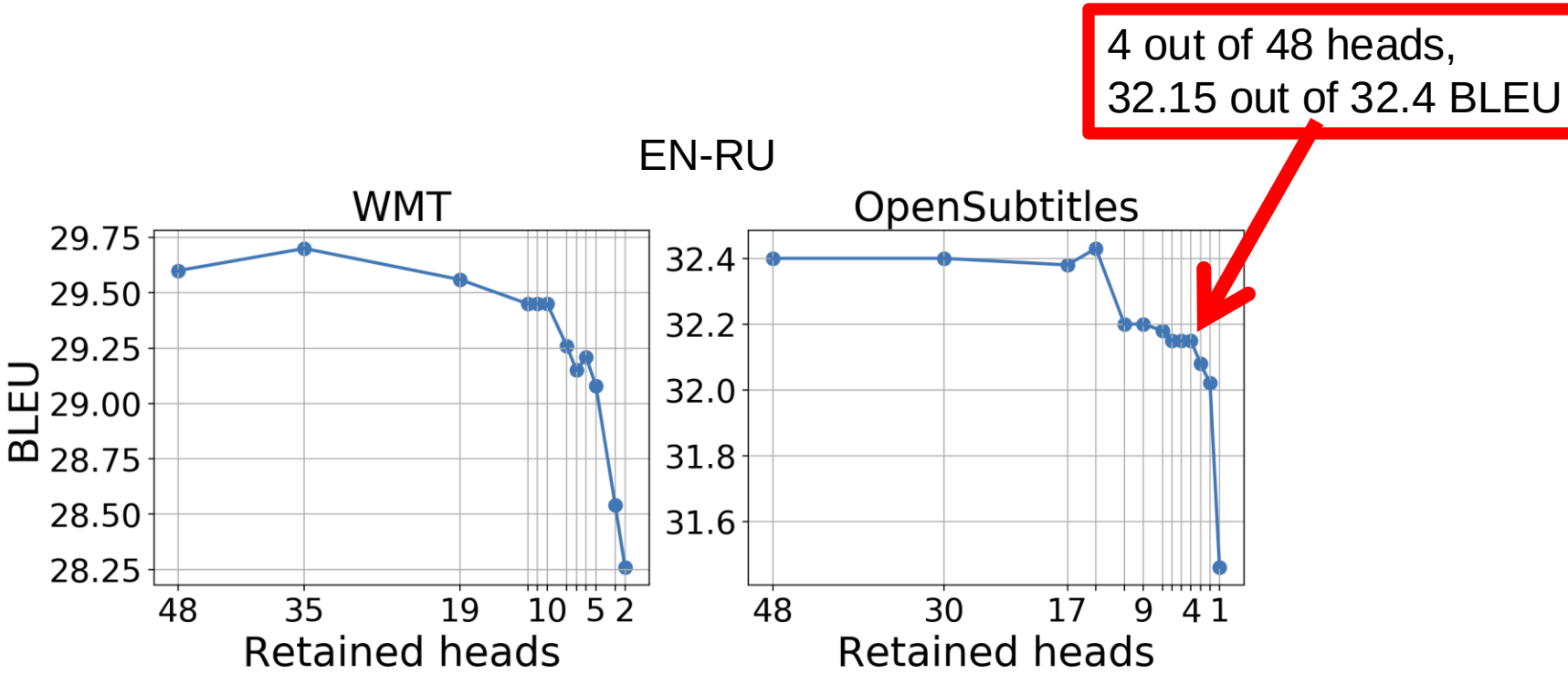


Pruned heads and BLEU score

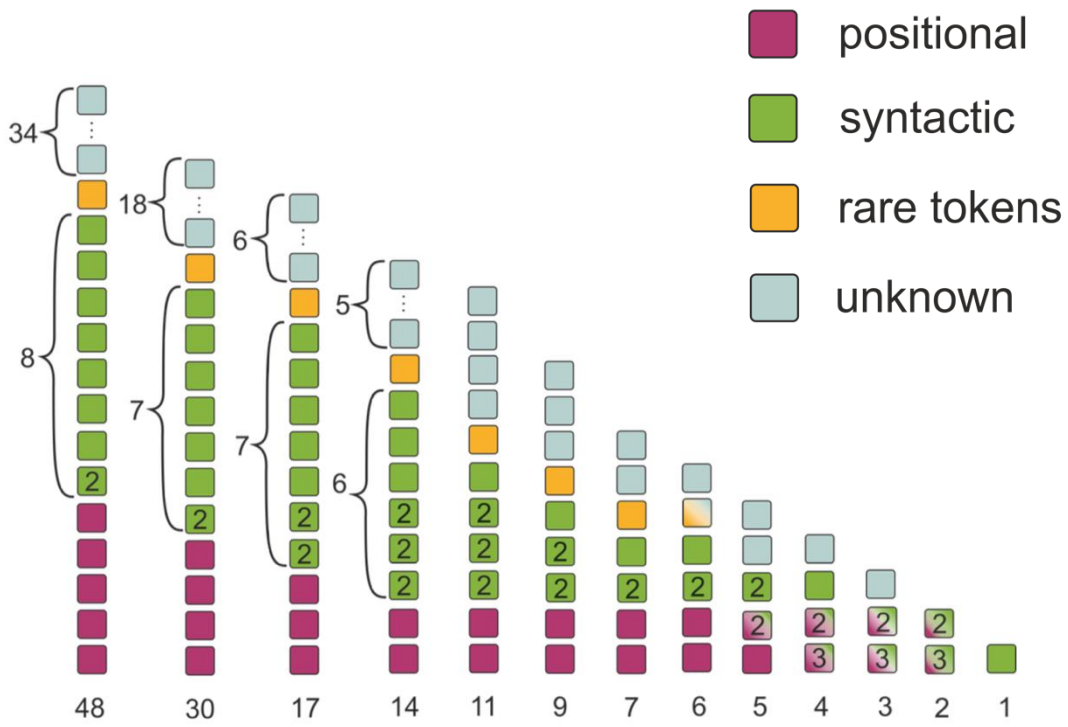


10 out of 48 heads,
29.45 out of 29.6 BLEU

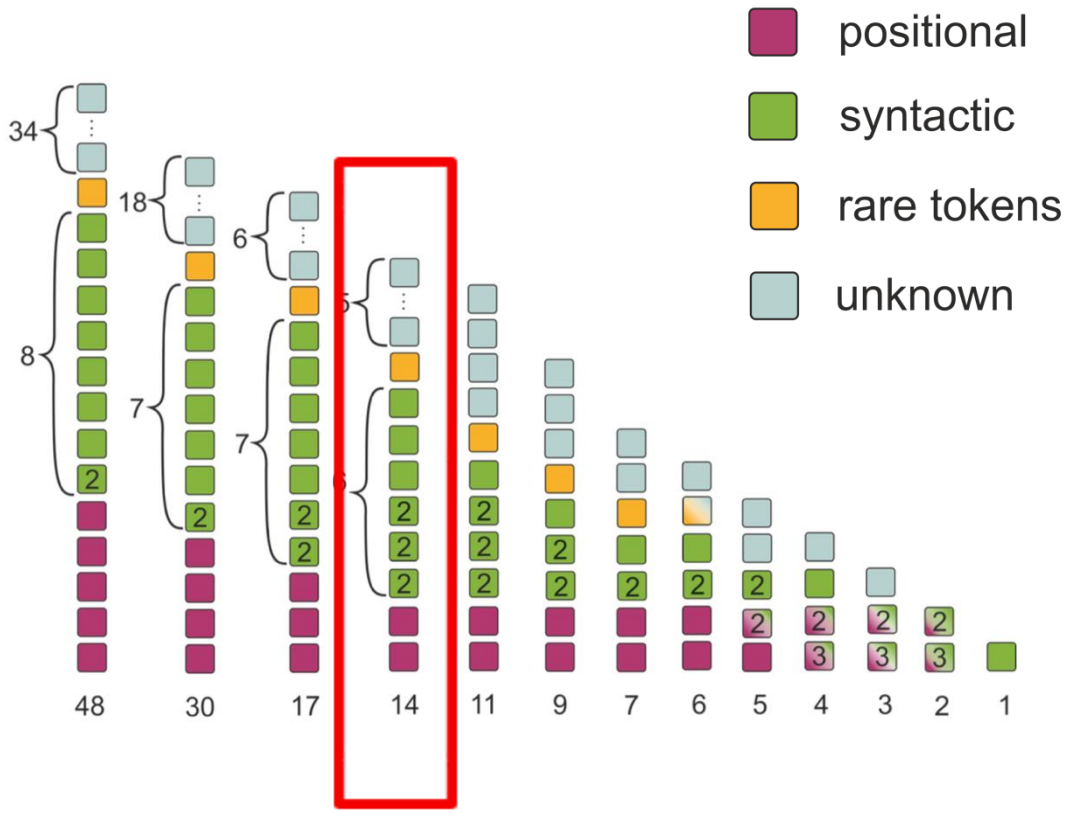
Pruned heads and BLEU score



What are the functions of alive heads?



What are the functions of alive heads?

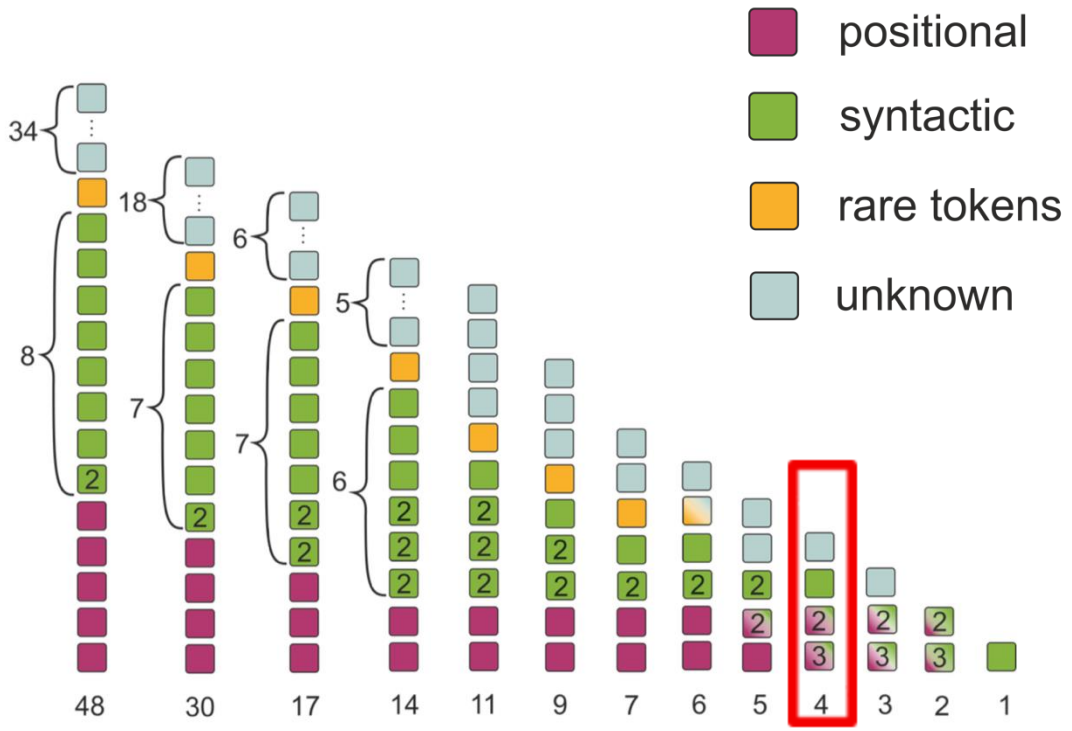


2/3 of heads are pruned, but all the functions are still alive

What are the functions of alive heads?

Heads take several functions

What are the functions of alive heads?



Functions “drift” to other heads

Key findings were

- Only a small subset of encoder heads are important for translation;

- Many important heads have one or more specialized and interpretable functions in the model;

In Large Language Models, things are a bit less crisp, but there are still some components which are interpretable:

- Heads retrieving relevant information from the context
- Heads tracking entities across texts

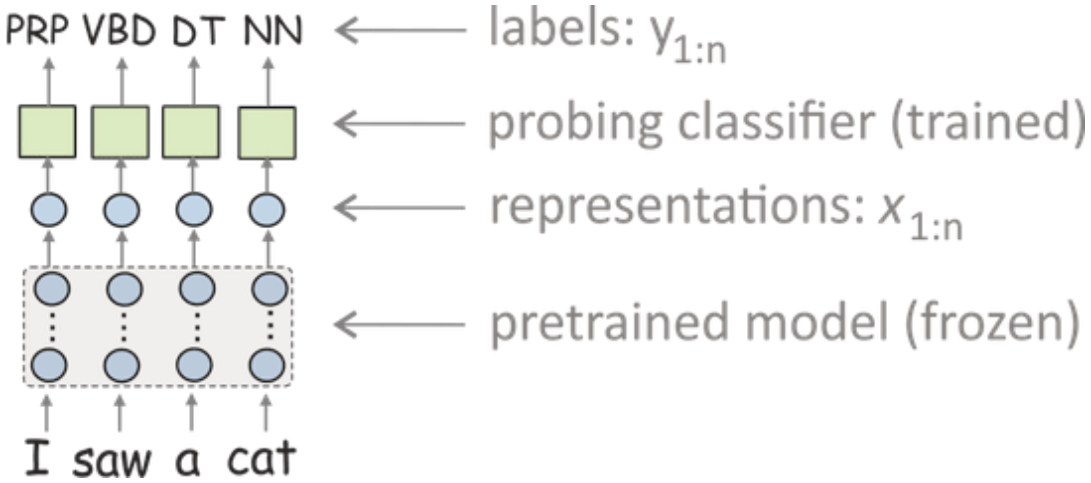
- …

Interpretability: model agnostic approaches

You can ask: what a certain model component represent?

Diagnostic classifiers (aka probing):

- feed data to a network and get vector representations of this data,
- train a classifier to predict some (linguistic) labels from these representations (the model itself is frozen),
- use the classifier's accuracy as a measure of how well representations encode labels.

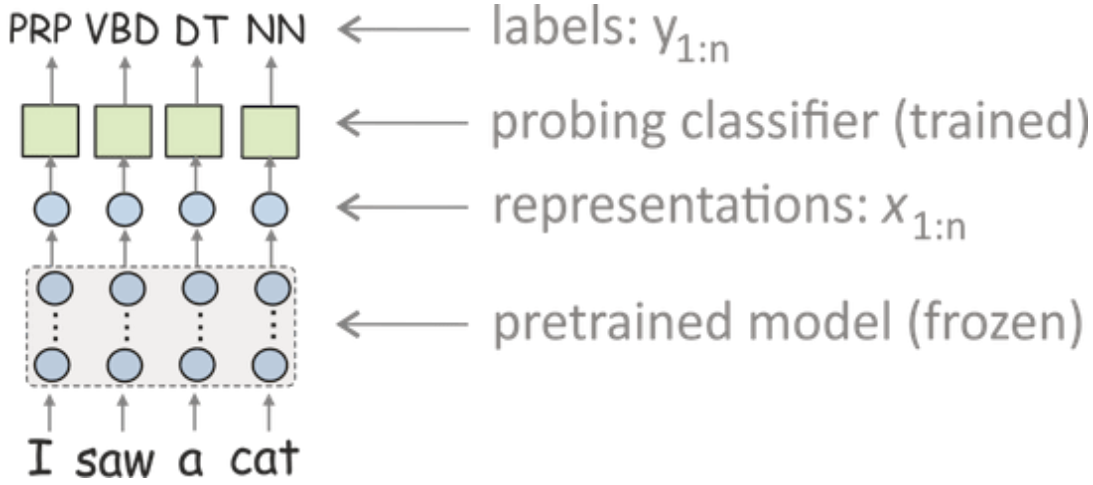


Interpretability: model agnostic approaches

Issues with this idea, and modifications of basic probing:

- Is the classifier learning the linguistic phenomenon, or is it the model that encodes it? (Voita and Titov, 2020)

- Even if the phenomenon is encoded, does the model actually use it? Can we intervene in the representation to influence what is generated? (Vig et al, 2020)



Take-aways / notes

- We have a solid understanding of the Transformer model (well, almost! 😊)
- Attention is the key component (make sure you understand it!), but other elements – residuals, MLPs, position embeddings, layer normalization and even droput -- also play important roles.
- Attention heads can sometimes learn specialized, interpretable functions.
- As models become more complex and successful, the demand for explainable AI continues to grow.
- In many upcoming lectures, the Transformer will serve as the backbone, but the focus will shift to its applications, training strategies, and more.
- Lab 3 deals with Transformer, you will be training it from scratch