Seq2seq and Attention

Vlad Shakhuro

7 November 2025

Outline

1. Seq2seq basics
2. Attention
3. Transformer
4. Byte-pair encoding
5. Attention heads analysis

Translation

Human translation

$$y^* = \arg\max_y p(y \mid x)$$

Machine translation

$$y' = \arg\max_y p(y \mid x, \theta)$$

Questions to answer:

- **modeling**
  How does the model for $p(y \mid x, \theta)$ look like?
- **learning**
  How to find θ?
- **search**
  How to find the argmax?

Conditional language models

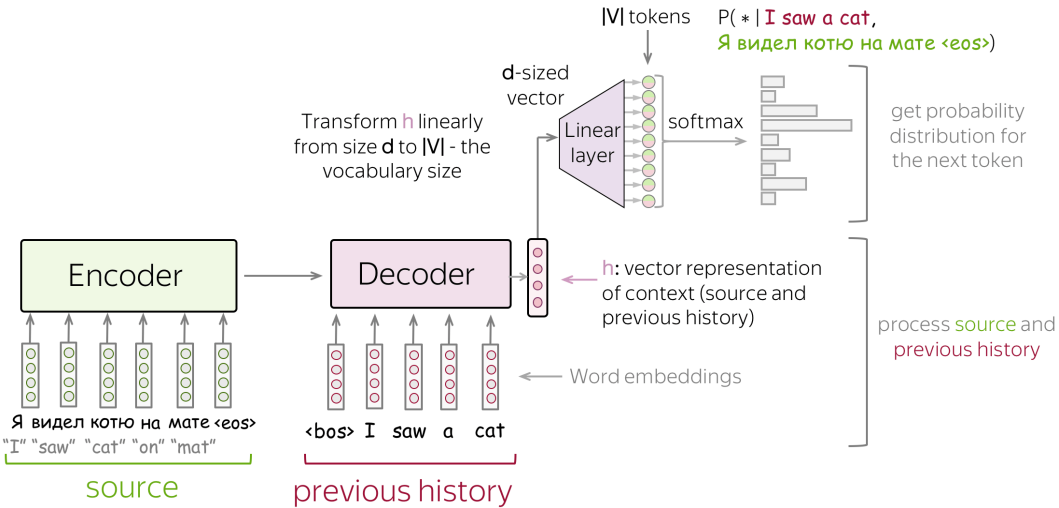
Language models:

$$p(y_1, y_2, \ldots, y_n) = \prod_{t=1}^{n} p(y_t | y_{<t})$$

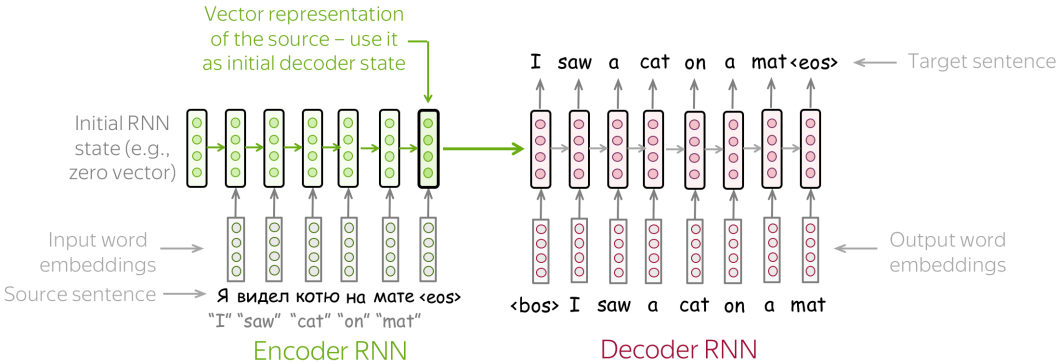
Conditional language models:

$$p(y_1, y_2, \ldots, y_n | \boldsymbol{x}) = \prod_{t=1}^{n} p(y_t | y_{<t}, \boldsymbol{x})$$

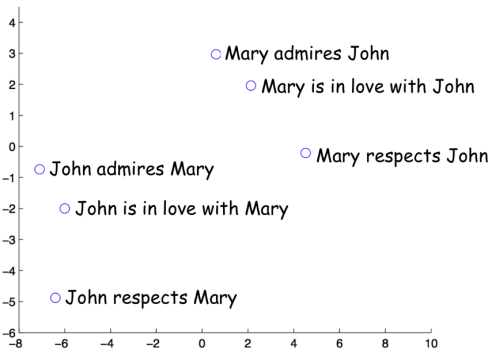
Encoder-decoder architecture

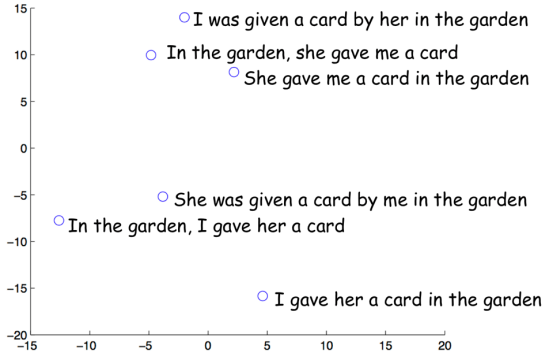


RNN encoder and decoder



What encoder state represents?





Training with cross-entropy and teacher forcing

Source sequence:

Я видел котю на мате <eos>

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

Target sequence:

I saw a cat on a mat <eos>

one training example

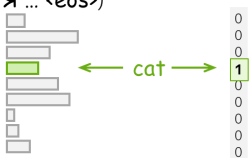
one step for this example

previous tokens

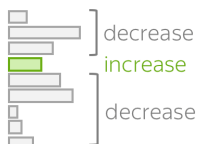
we want the model to predict this

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

Target



Loss = -log (p(cat)) → min



Inference: greedy decoding

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

Greedy decoding: at each step pick token with highest probability. *What are the limitations of this method?*

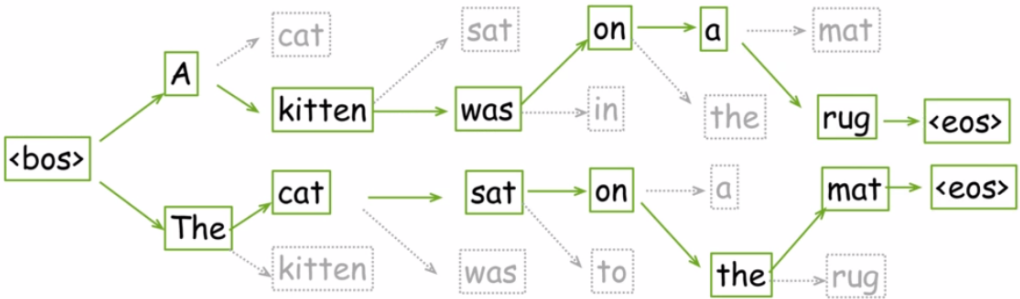
Inference: greedy decoding

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

Greedy decoding: at each step pick token with highest probability. *What are the limitations of this method?*

$$\arg\max_y \prod_{t=1}^{n} p(y_t \mid y_{<t}, x) \neq \prod_{t=1}^{n} \arg\max_{y_t} p(y_t \mid y_{<t}, x)$$

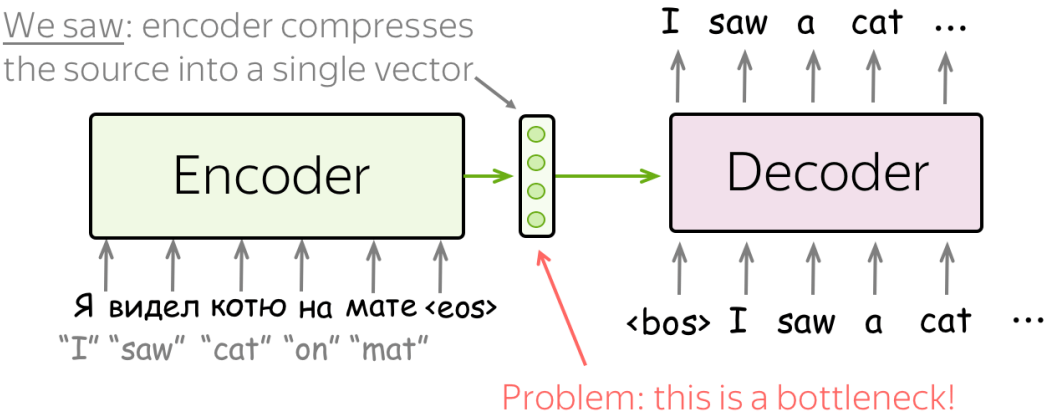
Inference: beam search



Outline

1. Seq2seq basics
2. Attention
3. Transformer
4. Byte-pair encoding
5. Attention heads analysis

The problem of a fixed representation

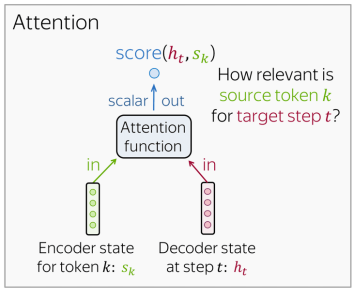


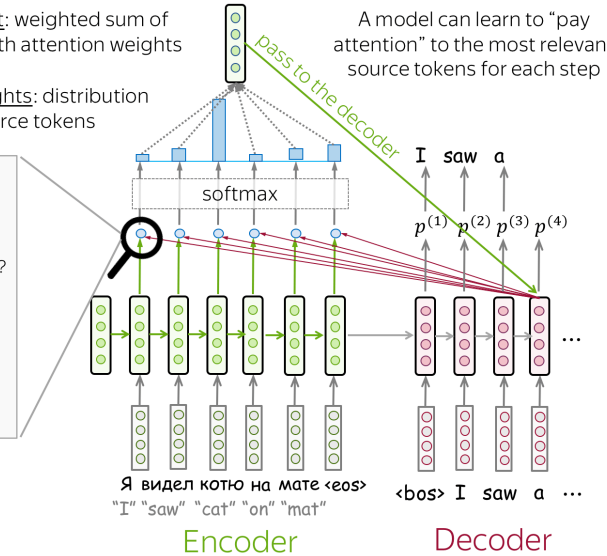
Attention overview

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





Attention computation

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?"

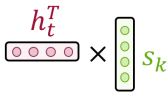
Attention input

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

h_t one decoder state

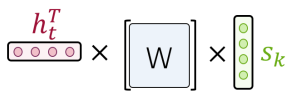
Score computation

Dot-product



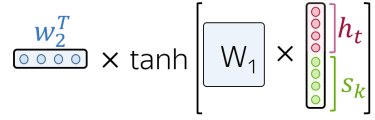
$\text{score}(h_t, s_k) = h_t^T \; s_k$

Bilinear



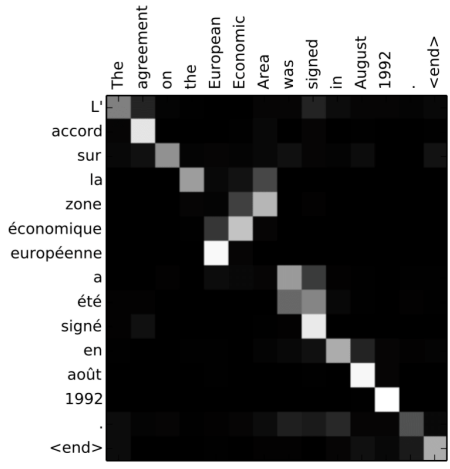
$\text{score}(h_t, s_k) = h_t^T \; W s_k$

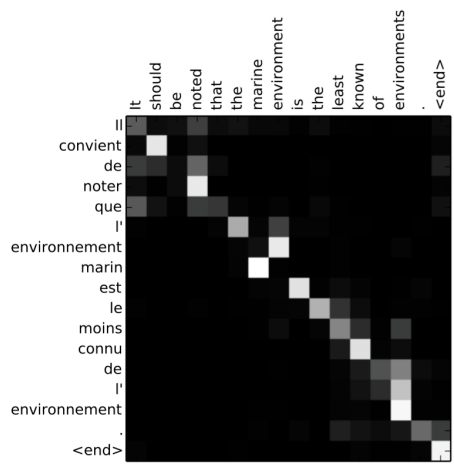
Multi-Layer Perceptron



$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$

Attention learns alignment





Outline

1. Seq2seq basics
2. Attention
3. Transformer
4. Byte-pair encoding
5. Attention heads analysis

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

Attention is all you need

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

Why attention can be better than RNN?

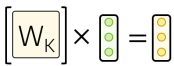
I arrived at the **bank** after crossing the … …street? …river?

Self-attention

Each vector receives three representations (“roles”)

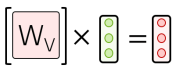
Query: vector **from** which the attention is looking

“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!”



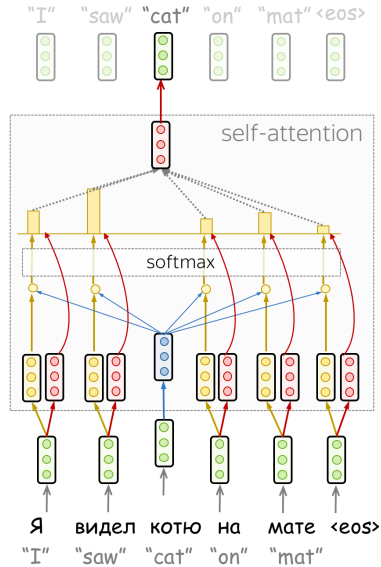
Value: their weighted sum is attention output

“Here’s the information I have!”

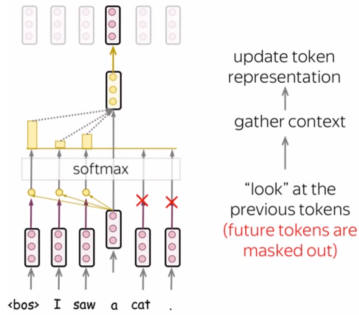
$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from (q) to (k, v)

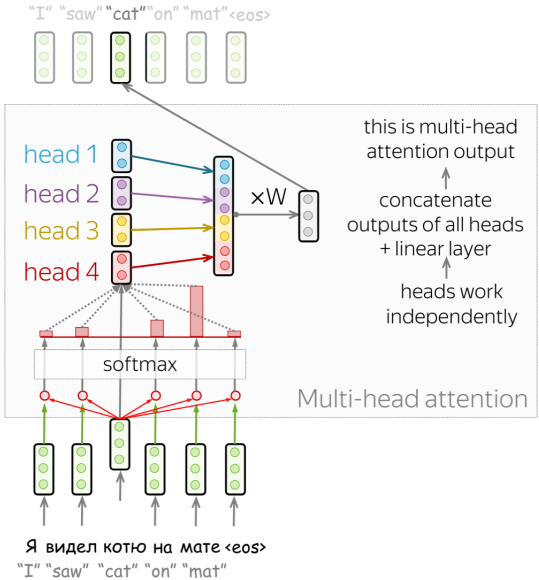
vector dimensionality of K, V

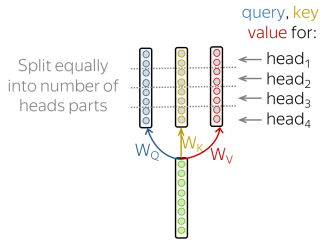


Masked self-attention

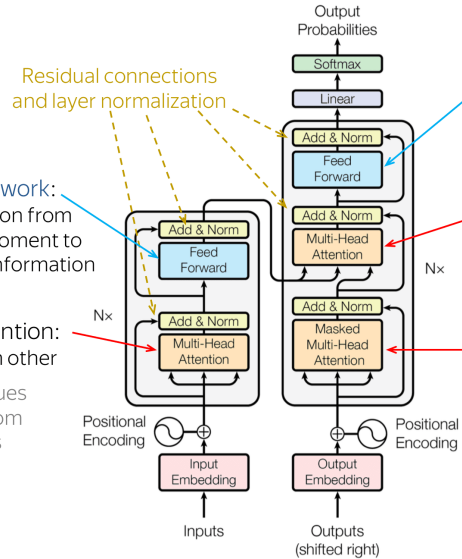


Multi-head self-attention





Transformer 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

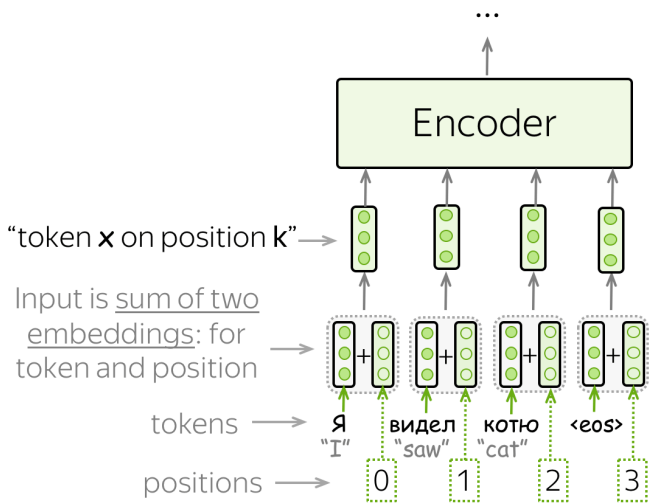
Decoder-encoder attention: target token looks at the source

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

Positional embeddings



Outline

1. Seq2seq basics
2. Attention
3. Transformer
4. Byte-pair encoding
5. Attention heads analysis

Subword segmentation

Byte-pair encoding

```
(hug, 10), (pug, 5), (pun, 12), (bun, 4), (hugs, 5)

V: [b, g, h, n, p, s, u]
C: (h-u-g,10), (p-u-g,5), (p-u-n,12), (b-u-n,4), (h-u-g-s,5)
```

Byte-pair encoding

(hug, 10), (pug, 5), (pun, 12), (bun, 4), (hugs, 5)

V: [b, g, h, n, p, s, u]
C: (h-u-g,10), (p-u-g,5), (p-u-n,12), (b-u-n,4), (h-u-g-s,5)

V: [b, g, h, n, p, s, u, ug]
C: (h-ug,10), (p-ug,5), (p-u-n,12), (b-u-n,4), (h-ug-s,5)

Byte-pair encoding

(hug, 10), (pug, 5), (pun, 12), (bun, 4), (hugs, 5)

V: [b, g, h, n, p, s, u]
C: (h-u-g,10), (p-u-g,5), (p-u-n,12), (b-u-n,4), (h-u-g-s,5)

V: [b, g, h, n, p, s, u, ug]
C: (h-ug,10), (p-ug,5), (p-u-n,12), (b-u-n,4), (h-ug-s,5)

V: [b, g, h, n, p, s, u, ug, un]
C: (h-ug,10), (p-ug,5), (p-un,12), (b-un,4), (h-ug-s,5)

Byte-pair encoding

(hug, 10), (pug, 5), (pun, 12), (bun, 4), (hugs, 5)

V: [b, g, h, n, p, s, u]
C: (h-u-g,10), (p-u-g,5), (p-u-n,12), (b-u-n,4), (h-u-g-s,5)

V: [b, g, h, n, p, s, u, ug]
C: (h-ug,10), (p-ug,5), (p-u-n,12), (b-u-n,4), (h-ug-s,5)

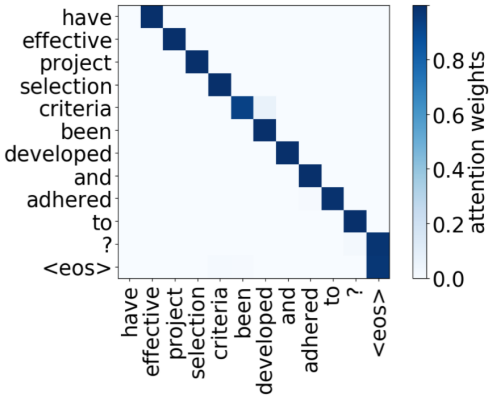
V: [b, g, h, n, p, s, u, ug, un]
C: (h-ug,10), (p-ug,5), (p-un,12), (b-un,4), (h-ug-s,5)

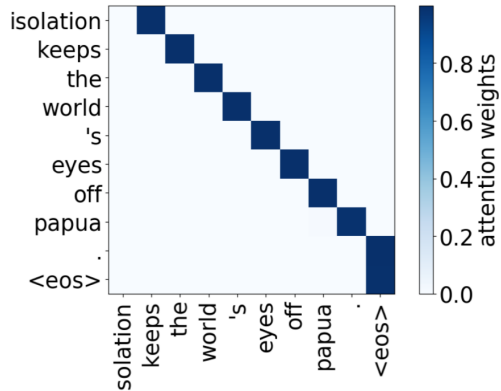
V: [b, g, h, n, p, s, u, ug, un, hug]
C: (hug,10), (p-ug,5), (p-un,12), (b-un,4), (hug-s,5)

Outline

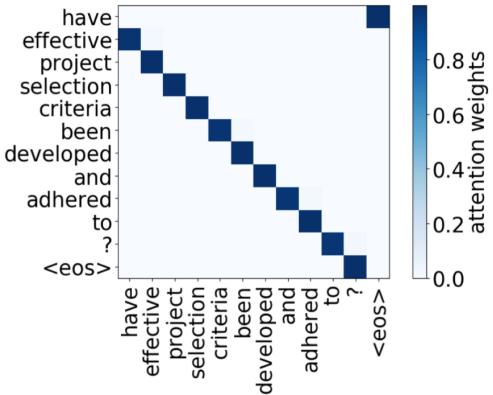
1. Seq2seq basics
2. Attention
3. Transformer
4. Byte-pair encoding
5. Attention heads analysis

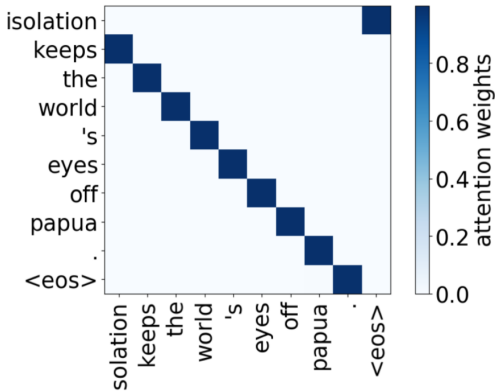
Positional heads: attention to neighbours





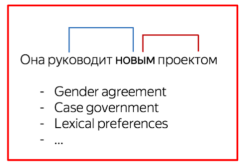
Positional heads: attention to neighbours

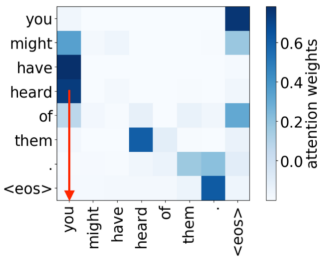


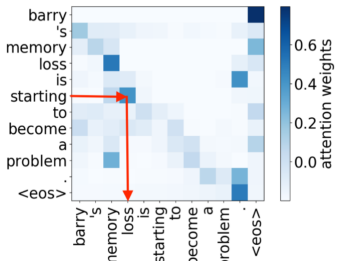


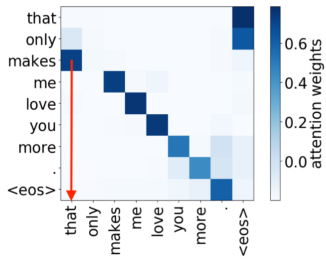
Syntactic heads: track dependencies

- verb->subject

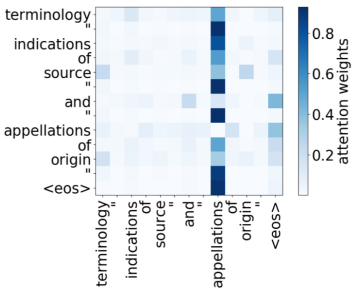


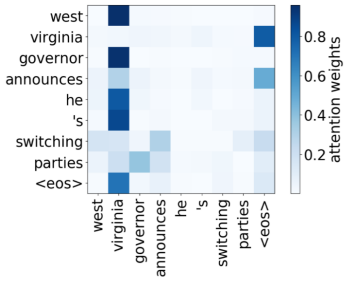


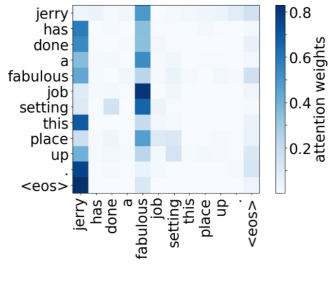




Attention to rare tokens







Conclusion

We reviewed following topics:

- seq2seq task basics: model architecture, training, inference
- attention basics
- multi-head self-attention
- transformer architecture