Intro to seq2seq

Машинное обучение в диалоговых системах
2 апреля 2019

Cheskidova E - cheskidova@phystech.edu

Outline

1. Seq2seq architectures with rnn
    a. Applications in NLP
    b. Simple encoder, simple decoder
    c. Decoder with attention

2. Transformer
    a. Encoder
        i. Multi-head self-attention
        ii. LayerNorm & residual connections
        iii. Position-wise feed-forward
        iv. Positional Encoding
    b. Decoder
        i. Multi-head attention with encoder outputs
        ii. Masking

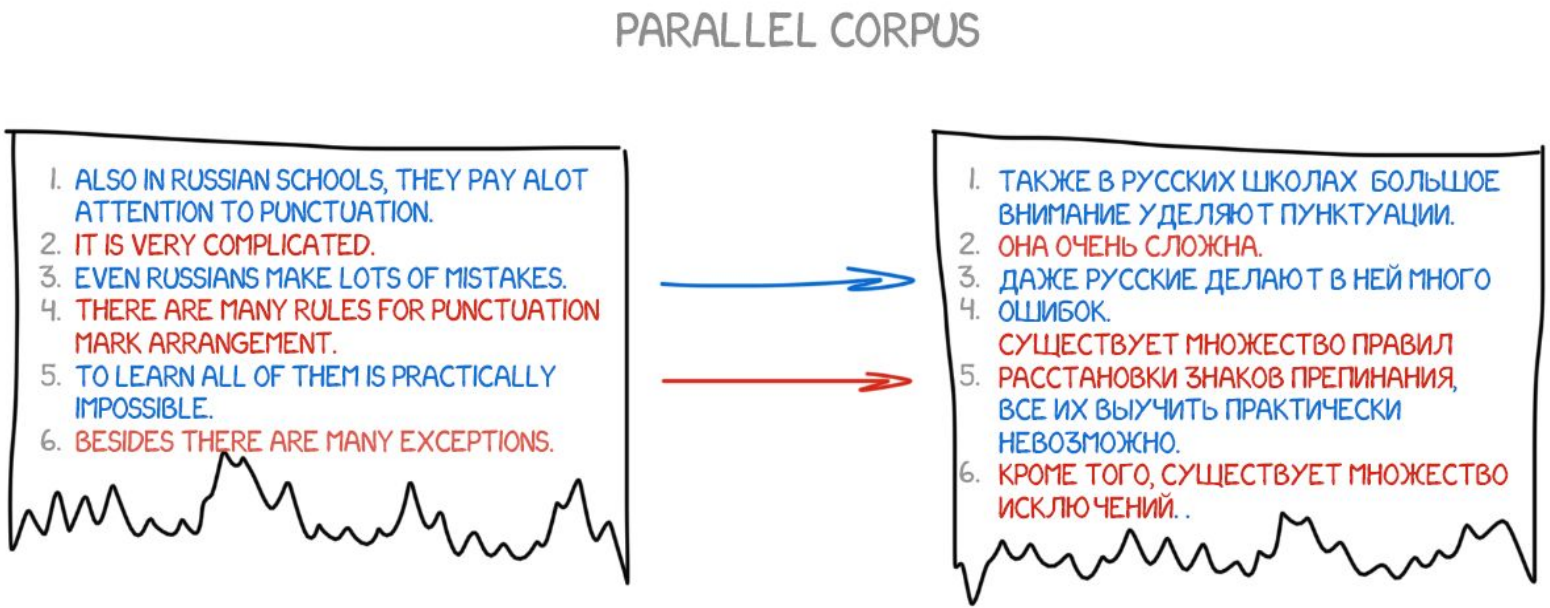
3. Optional Part

Outline

1. Seq2seq architectures with rnn
    a. Applications in NLP
    b. Simple encoder, simple decoder
    c. Decoder with attention

2. Transformer
    a. Encoder
        i. Multi-head self-attention
        ii. LayerNorm & residual connections
        iii. Position-wise feed-forward
        iv. Positional Encoding
    b. Decoder
        i. Multi-head attention with encoder outputs
        ii. Masking

Machine translation



Generative chatbot

INPUT: *Hi!*

OUTPUT: *Hi!*

INPUT: *How are you?*

OUTPUT: *I am good. Thanks for asking.*

INPUT: *You're asking me out. That's so cute. What's your name again?*

OUTPUT: *Forget it.*

INPUT: *No, no, it's my fault—we didn't have a proper introduction*

OUTPUT: *Cameron.*

INPUT: *The thing is, Cameron—I'm at the mercy of a particularly hideous breed of loser. My sister. I can't date until she does.*

OUTPUT: *Seems like she could get a date easy enough…*

Other applications

Music Generator

Random seed = 5 → Sequence Model →

Text Captioning

Random seed = 10 → Sequence Model → *Neuron is a cell that can transmit electrical signals. They can also refer to units in ANN model.....*

Image Generator

Random seed = 15 → Sequence Model →

Outline

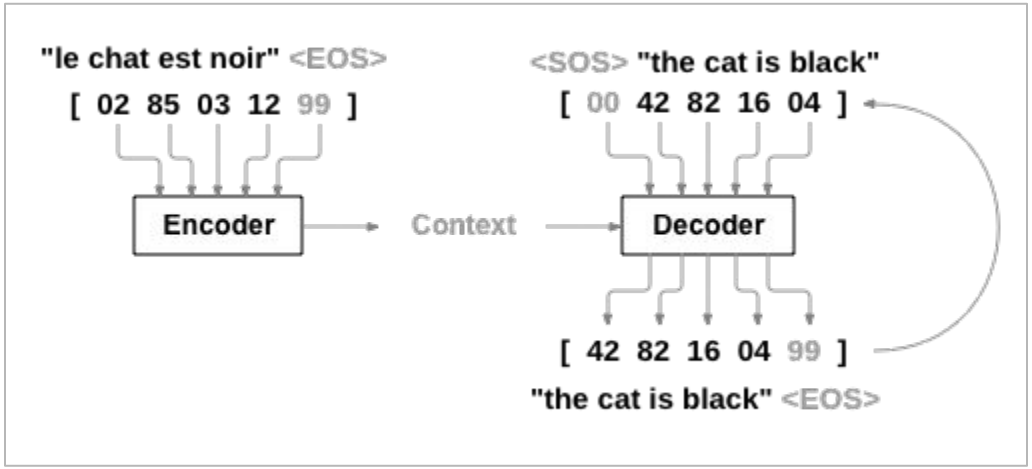
1. Seq2seq architectures with rnn
   a. Applications
   b. Simple encoder, simple decoder
   c. Decoder with attention

2. Transformer
   a. Encoder
      i. Multi-head self-attention
      ii. LayerNorm & residual connections
      iii. Position-wise feed-forward
      iv. Positional Encoding
   b. Decoder
      i. Multi-head attention with encoder outputs
      ii. Masking

Seq2seq architectures with rnn

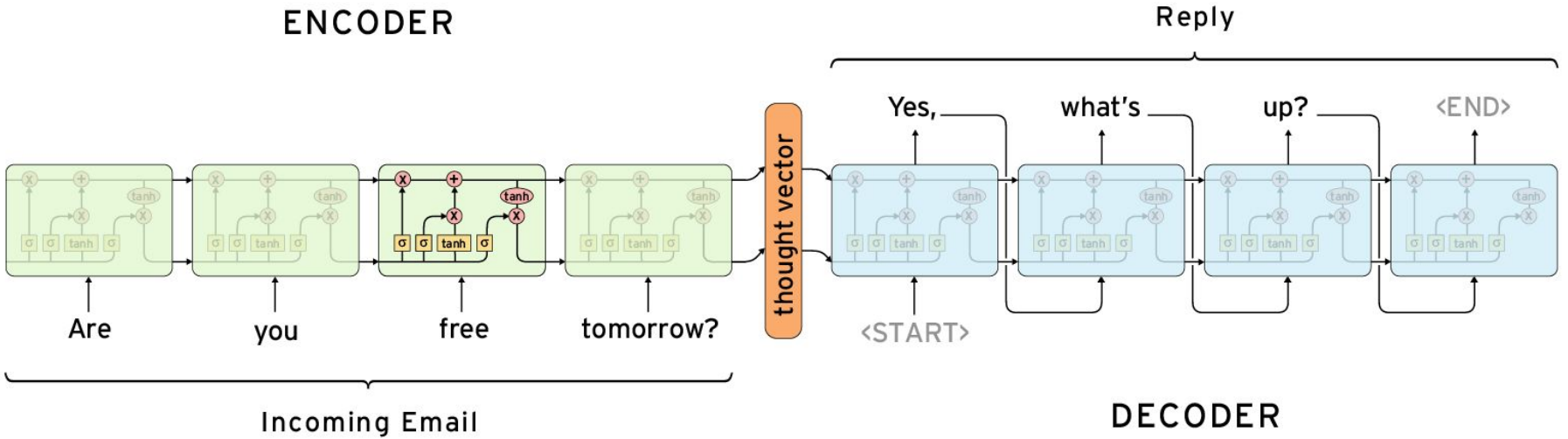
Simple encoder, simple decoder

- How to transmit information from encoder to decoder?
    - Initialize hidden state of **decoder** with last hidden state of **encoder**
    - Concatenate last hidden state of **encoder** to each input of **decoder**



pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

Simple encoder-decoder



Training

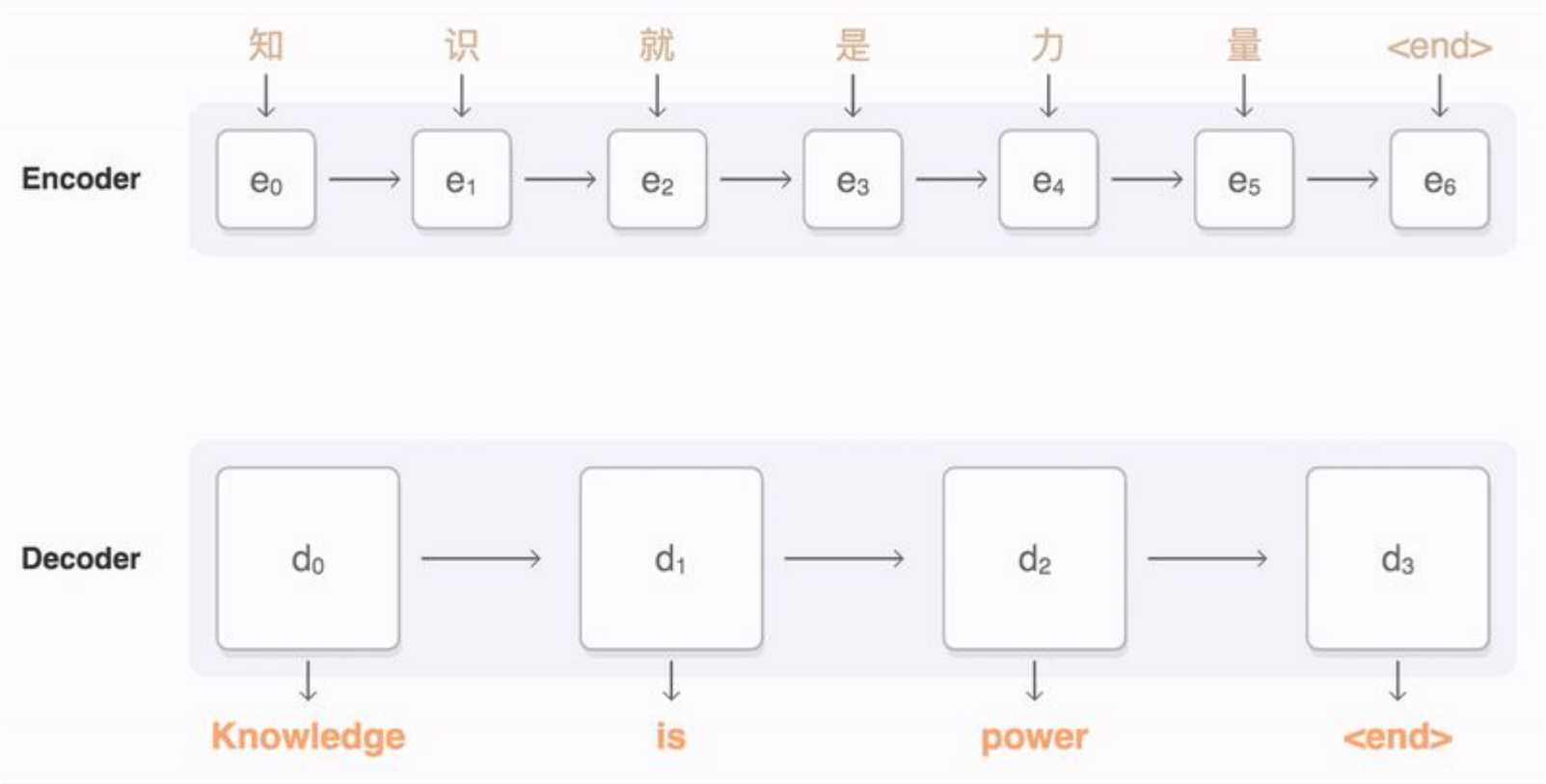
1. **X** := last **hidden state of encoder**
2. Init **decoder hidden state** with **X**
3. Input <START> token to decoder to start decoding
4. Calculate loss for the decoder output and right output (e.g. cross-entropy)
5. If (**teacher_forcing**):

    feeding ***right_prev_token*** into decoder

    else:

    feeding ***decoder_last_output*** into decoder

Attention in seq2seq

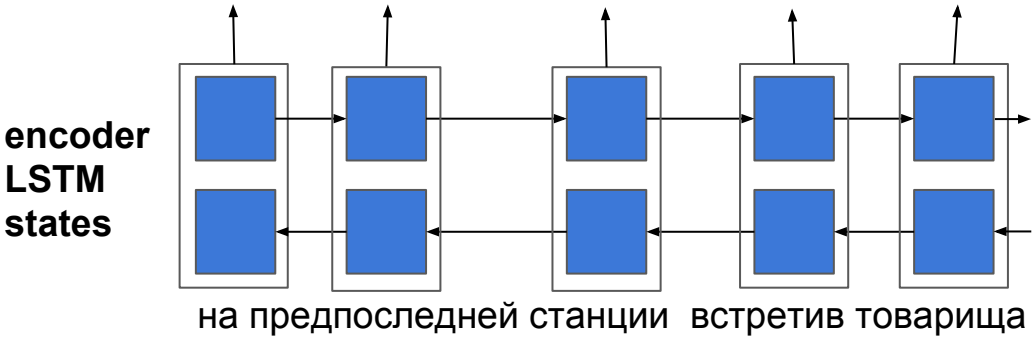


Outline

1. Seq2seq architectures with rnn
    a. Applications
    b. Simple encoder, simple decoder
    c. Decoder with attention

2. Transformer
    a. Encoder
        i. Multi-head self-attention
        ii. LayerNorm & residual connections
        iii. Position-wise feed-forward
        iv. Positional Encoding
    b. Decoder
        i. Multi-head attention with encoder outputs
        ii. Masking

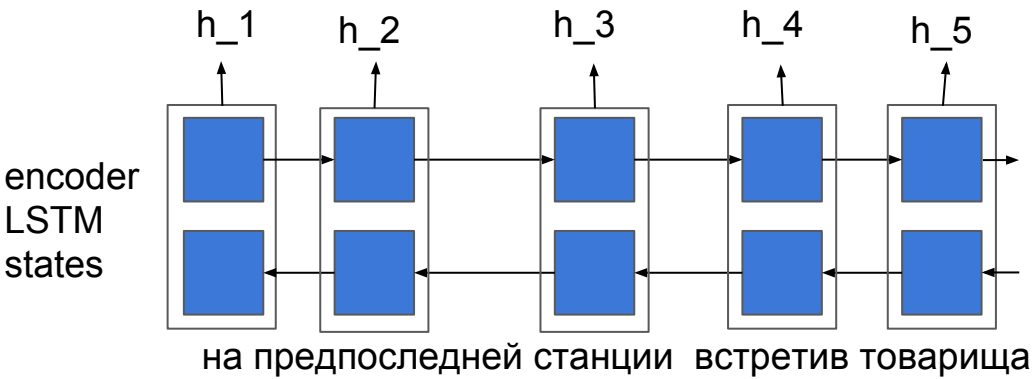
Attention for neural machine translation



Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014

Slide credit: Michael Figurnov, DeepBayes school

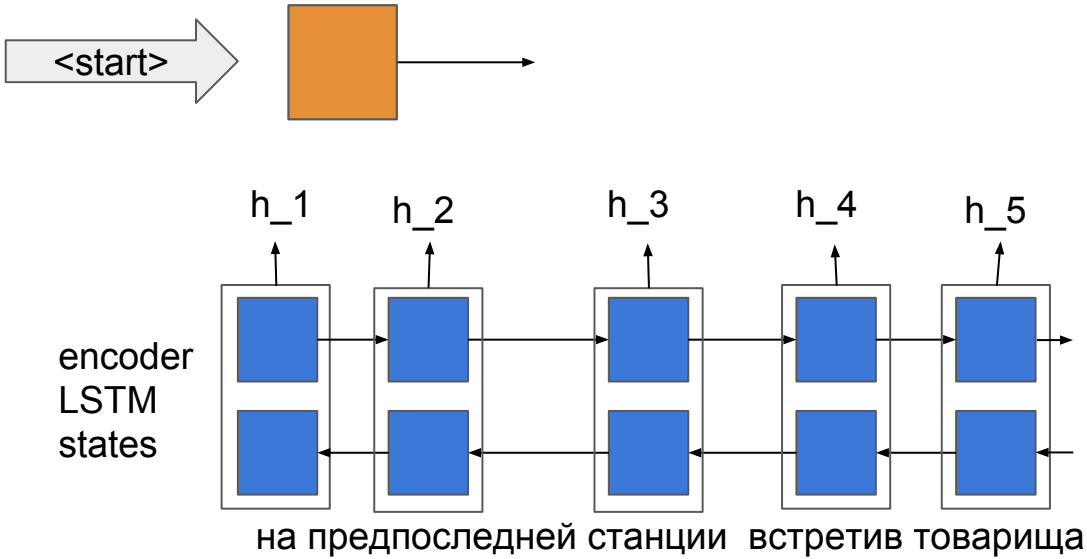
Attention for neural machine translation



Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014

Slide credit: Michael Figurnov, DeepBayes school

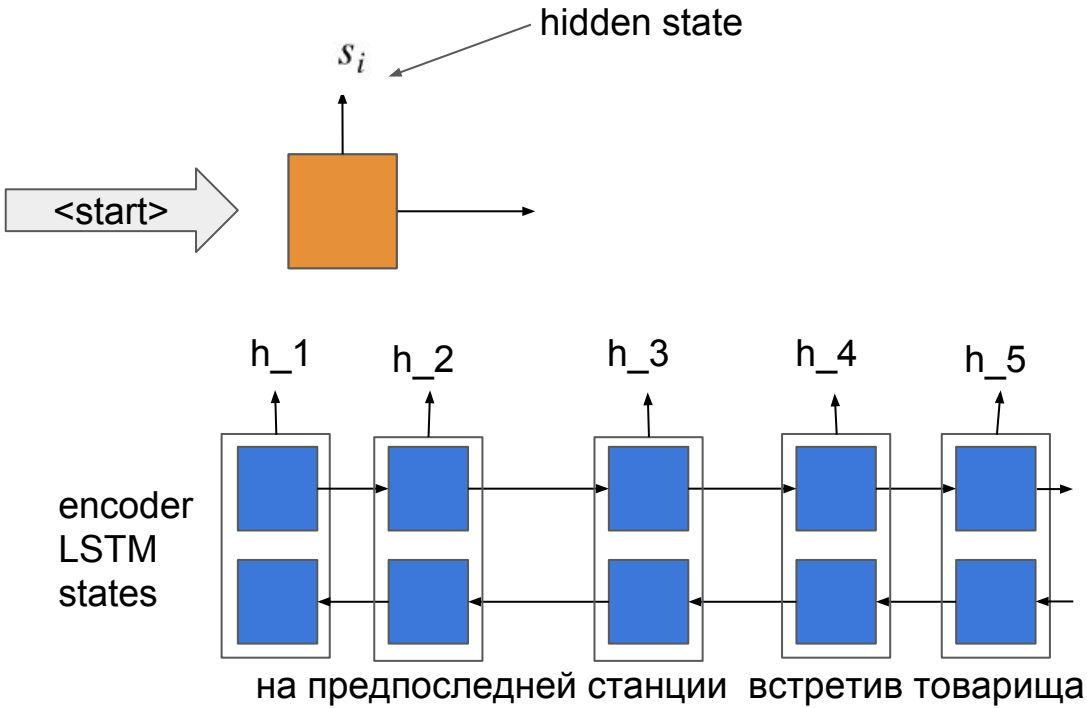
Attention for neural machine translation



Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014

Slide credit: Michael Figurnov, DeepBayes school

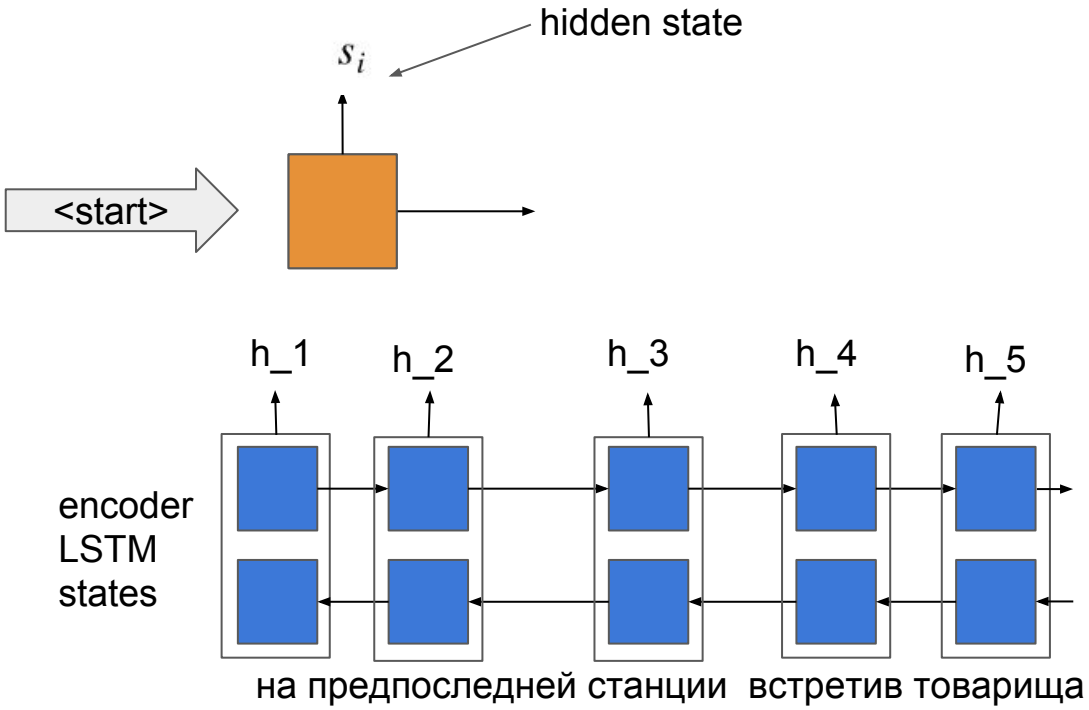
Attention for neural machine translation



Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014

Slide credit: Michael Figurnov, DeepBayes school

Attention for neural machine translation



Energy

$$e_{ij} = a(s_{i-1}, h_j)$$

a - some function

Attention weights

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

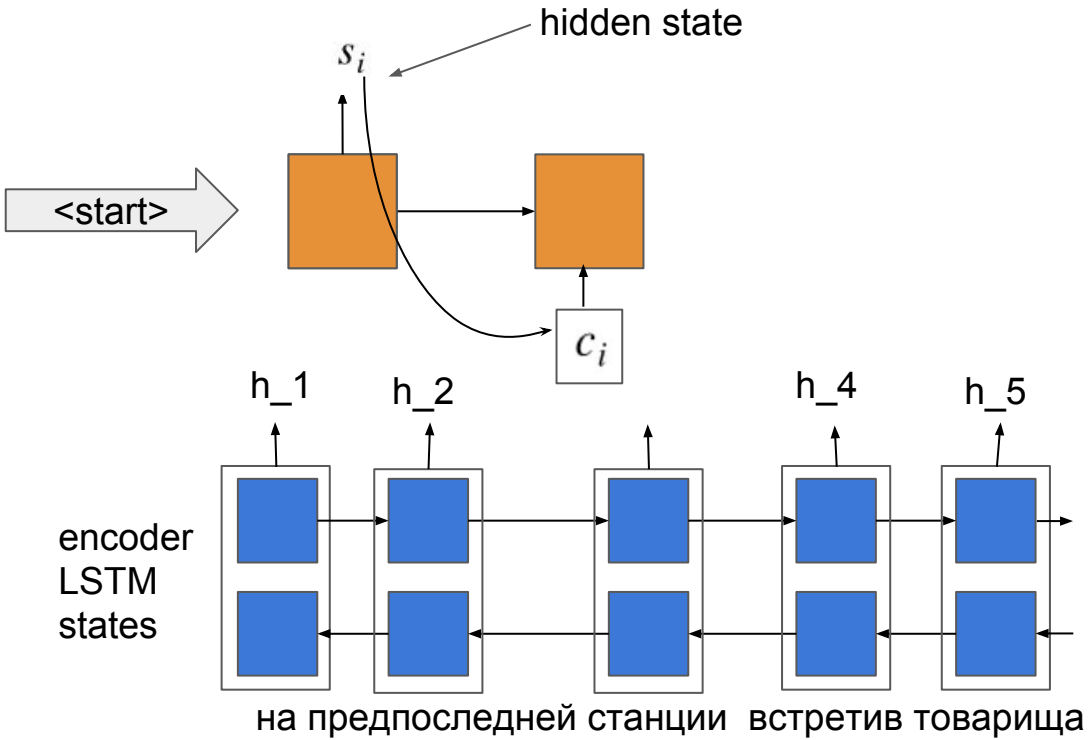
Context vector

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j$$

Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014

Slide credit: Michael Figurnov, DeepBayes school

Attention for neural machine translation



Energy

$$e_{ij} = a(s_{i-1}, h_j)$$

a - some function

Attention weights

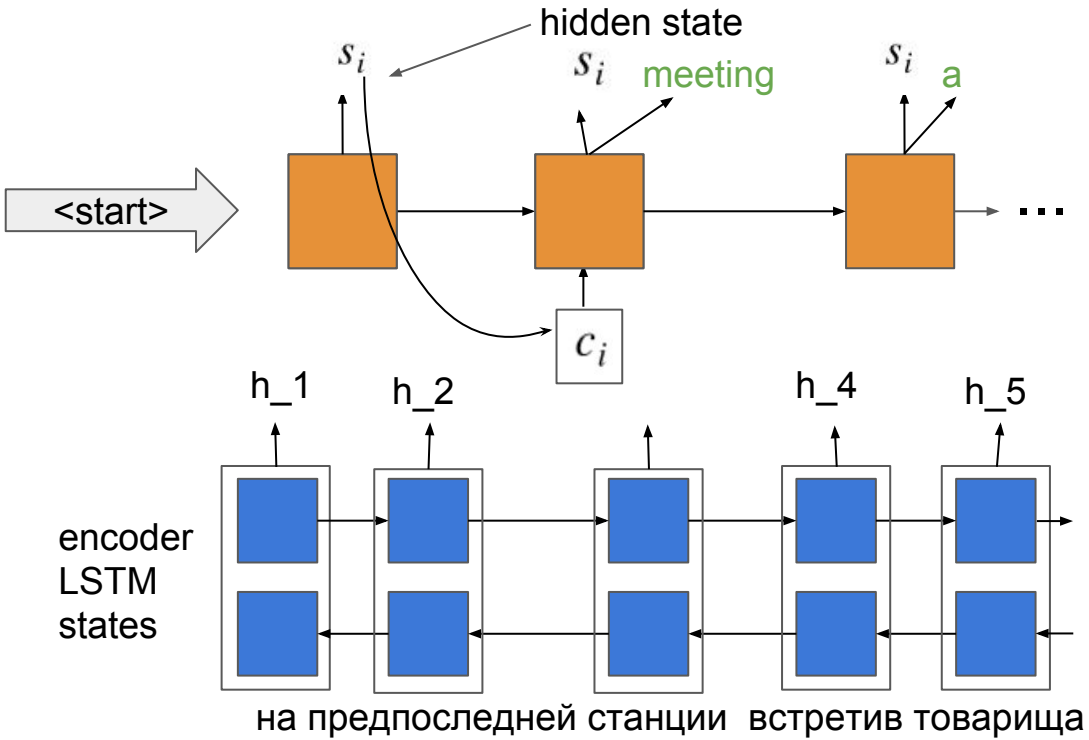
$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

Context vector

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j$$

Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014
Slide credit: Michael Figurnov, DeepBayes school

Attention for neural machine translation



Energy

$$e_{ij} = a(s_{i-1}, h_j)$$

a - some function

Attention weights

$$a_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

Context vector

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j$$

Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014
Slide credit: Michael Figurnov, DeepBayes school

Many ways to compute the energy

$$\text{score}(h, f) = \text{NN}(h, f) = w_3^T \tanh(W_1 h + W_2 f)$$ [1]

additive attention
dim(*h*) != dim(*f*)

$$\text{score}(h, f) = h^T f$$ [2]

$$\text{score}(h, f) = h^T f / \sqrt{d}, \; d = \dim(h)$$ [3]

multiplicative attention
dim(*h*) = dim(*f*)

[1] Bahdanau et al. "Neural Machine Translation by Jointly Learning to Align and Translate", 2014
[2] Luong et al. "Effective approaches to attention-based neural machine translation", 2015
[3] Vaswani et al. "Attention Is All You Need", 2017

Slide credit: Michael Figurnov, DeepBayes school

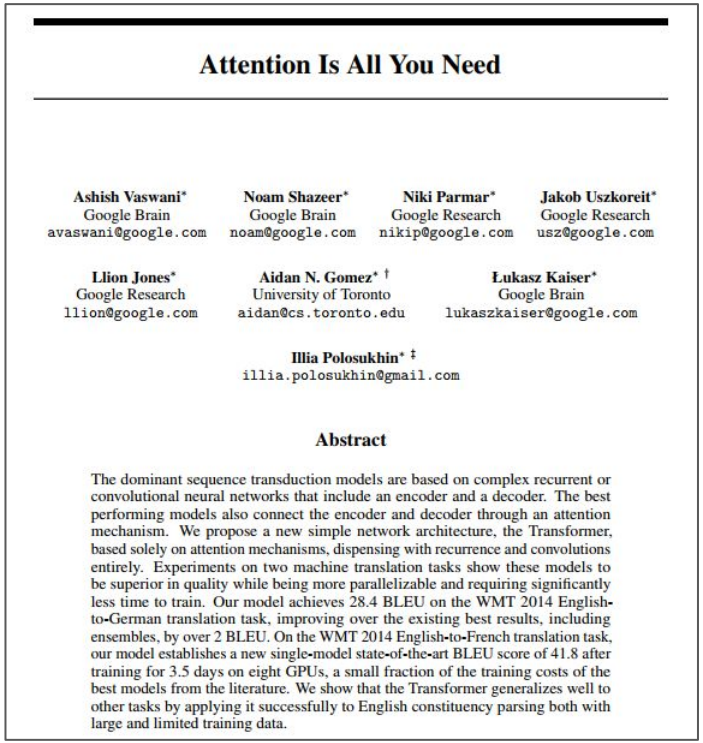
Outline

1. Seq2seq architectures with rnn
    a. Applications
    b. Simple encoder, simple decoder
    c. Decoder with attention

2. Transformer
    a. Encoder
        i. Multi-head self-attention
        ii. LayerNorm & residual connections
        iii. Position-wise feed-forward
        iv. Positional Encoding
    b. Decoder
        i. Multi-head attention with encoder outputs
        ii. Masking

Transformer

NIPS 2017



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

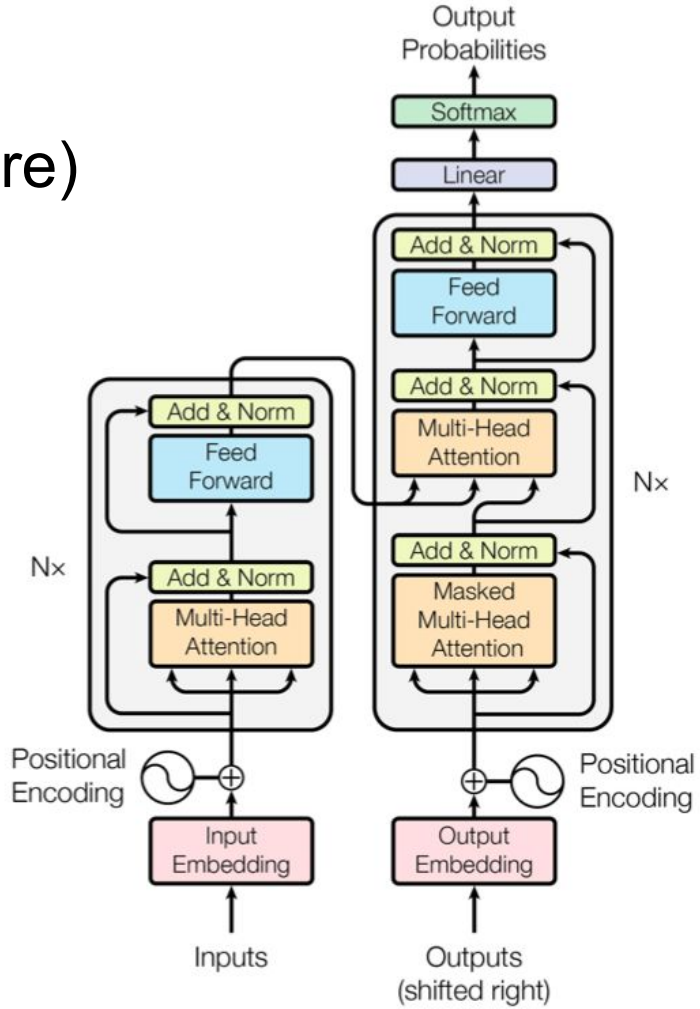
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

link: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

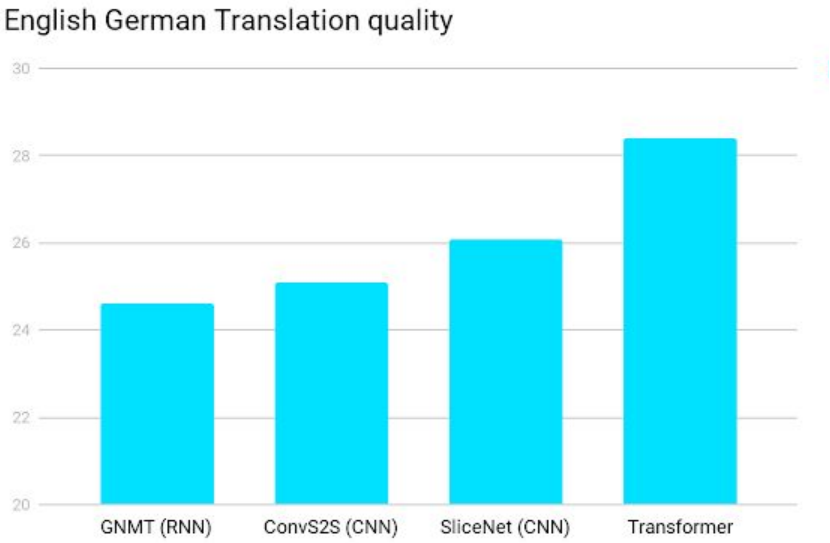
Transformer
(model architecture)

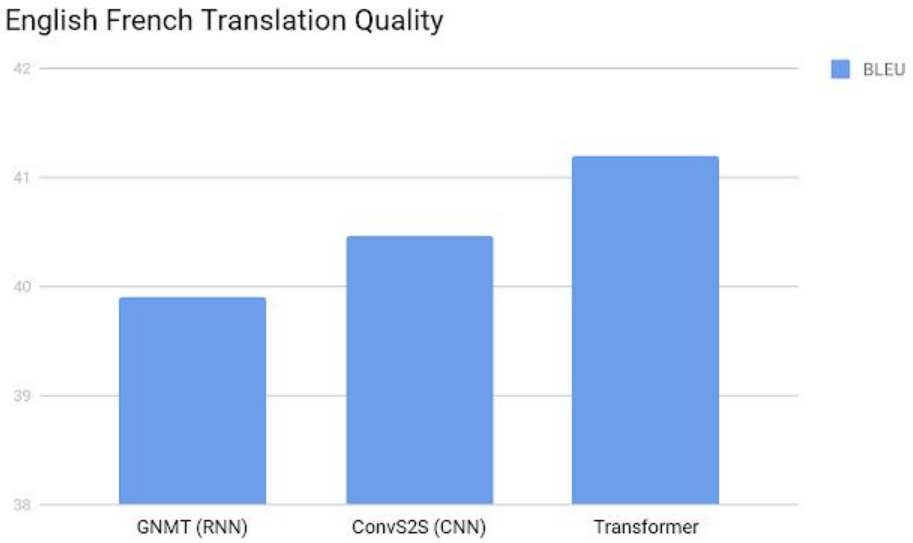
N = 6



Transformer

- Better BLUE score





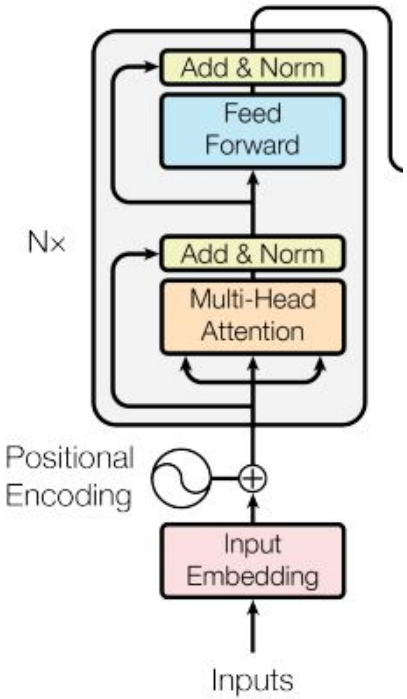
https://research.googleblog.com/2017/08/transformer-novel-neural-network.html

GLUE Benchmarks

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX
1	GLUE Human Baselines	GLUE Human Baselines		87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2	93.6	95.9	-
2	Microsoft D365 AI & MSR AI	MT-DNN++ (BigBird)		83.8	65.4	95.6	91.1/88.2	89.6/89.0	72.7/89.6	87.9	87.4	95.8	85.1	65.1	41.9
3	王玮	ALICE large (Alibaba DAMO NLP)		83.3	63.5	95.2	91.8/89.0	89.8/88.8	74.0/90.4	87.9	87.4	95.7	80.9	65.1	40.7
4	Stanford Hazy Research	Snorkel MeTaL		83.2	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9	87.6	87.2	93.9	80.9	65.1	39.9
5	Anonymous Anonymous	BERT + BAM		82.3	61.5	95.2	91.3/88.3	88.6/87.9	72.5/89.7	86.6	85.8	93.1	80.4	65.1	40.7
6	张倬胜	SemBERT		82.0	62.3	94.6	90.6/87.3	87.8/86.7	72.8/89.8	87.2	86.3	93.2	78.5	65.1	40.5
7	Jason Phang	BERT on STILTs		82.0	62.1	94.3	90.2/86.6	88.7/88.3	71.9/89.4	86.4	85.6	92.7	80.1	65.1	28.3
8	Jacob Devlin	BERT: 24-layers, 16-heads, 1024-hic		80.5	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7	85.9	92.7	70.1	65.1	39.6
9	Neil Houlsby	BERT + Single-task Adapters		80.2	59.2	94.3	88.7/84.3	87.3/86.1	71.5/89.4	85.4	85.0	92.4	71.6	65.1	9.2
10	Alec Radford	Singletask Pretrain Transformer		72.8	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1	81.4	87.4	56.0	53.4	29.8
11	GLUE Baselines	BiLSTM+ELMo+Attn		70.0	33.6	90.4	84.4/78.0	74.2/72.3	63.1/84.3	74.1	74.5	79.8	58.9	65.1	21.7
		BiLSTM+ELMo		67.7	32.1	89.3	84.7/78.0	70.3/67.8	61.1/82.6	67.2	67.9	75.5	57.4	65.1	21.3
		Single Task BiLSTM+ELMo+Attn		66.5	35.0	90.2	80.2/68.8	55.5/52.5	66.1/86.5	76.9	76.7	76.7	50.3	65.1	27.9
		Single Task BiLSTM+ELMo		66.4	35.0	90.2	80.8/69.0	64.0/60.2	65.6/85.7	72.9	73.4	71.7	50.1	65.1	19.5
		GenSen		66.1	7.7	83.1	83.0/76.6	79.3/79.2	59.8/82.9	71.4	71.3	78.6	59.2	65.1	20.6
		BiLSTM+Attn		65.6	18.6	83.0	83.9/76.2	72.8/70.5	60.1/82.4	67.6	68.3	74.3	58.4	65.1	17.8
		BiLSTM		64.2	11.6	82.8	81.8/74.3	70.3/67.8	62.5/84.2	65.6	66.1	74.6	57.4	65.1	20.3

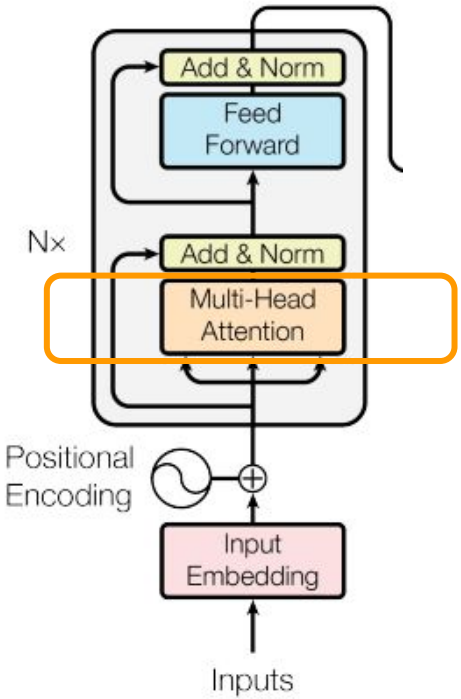
https://gluebenchmark.com/leaderboard

Encoder



- Multi-head self-attention
- LayerNorm & Residual connections
- Position-wise feed-forward
- Positional Encoding

Encoder



- Multi-head self-attention
- LayerNorm & Residual connections
- Position-wise feed-forward
- Positional Encoding

Multi-head self-attention

How does one head work?

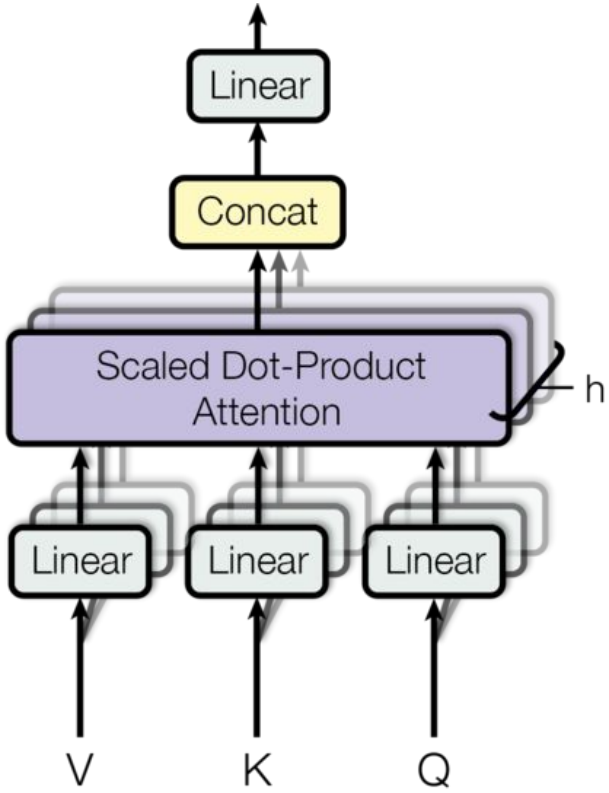
1. Value (V), Key (K), Query (Q) -- наборы векторов слов.
   (Value & Key одни и те же входные вектора, но подаются они на разные входы независимо)

2. Преобразуем входные V, K, Q каждый своим отдельным обучаемым линейным преобразованием.

3. Считаем скалярное произведение каждого вектора из Q с каждым вектором из K *(т.е. величину схожести каждого слова с каждым)*. Полученные произведения делим на корень из размерности вектора.

4. Для конкретного вектора q из Q мы получим преобразованный вектор, путем сложения всех векторов из V, с нормализованными весами, полученными в п. 3 с этим самым вектором q.

5. Возвращаем набор преобразованных векторов q.



Multi-head self-attention

Intuition

Keys = Values = Queries =
embeddings(["In", "my", "humble", "opinion"])

q := In;
q_out := my * (In, my) + humble * (In, humble) + opinion * (In, opinion)

Multi-head self-attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

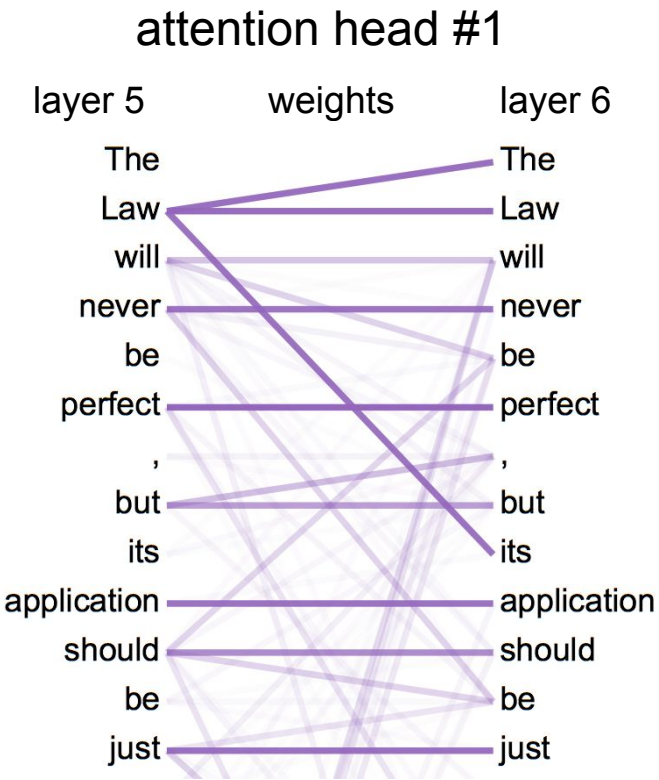
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O$$

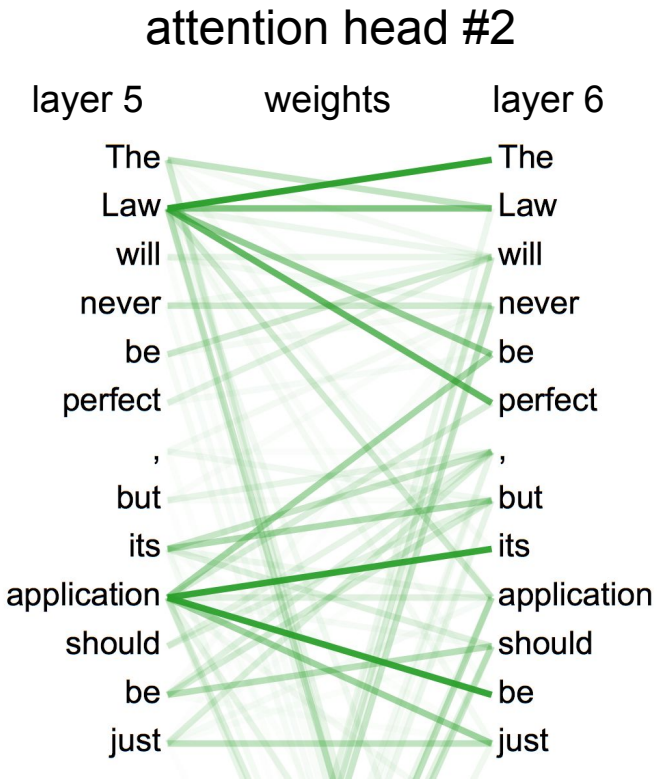
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

Dimension of the returned single matrix after applying Multi-head sublayer is the same as the dimension of any its input.

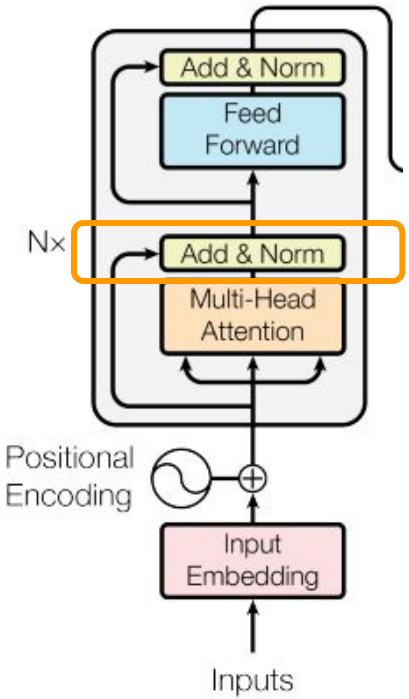
What do the self-attention heads look at?





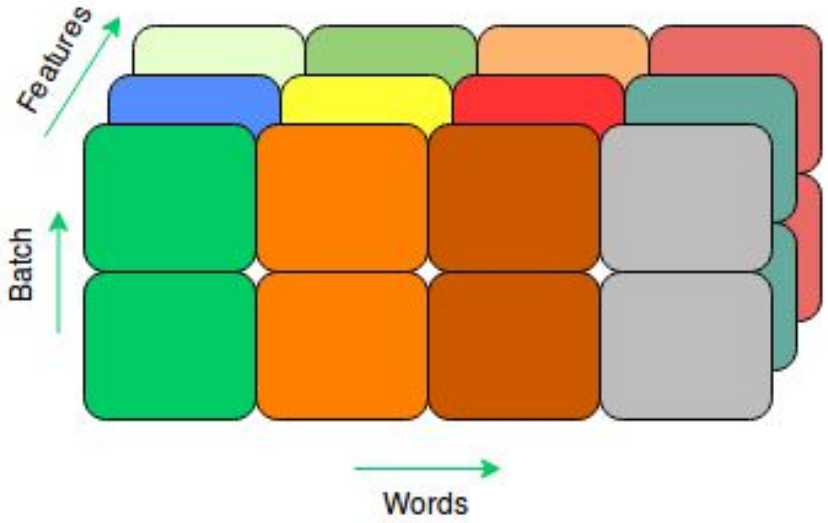
Slide credit: Michael Figurnov, DeepBayes school

Encoder

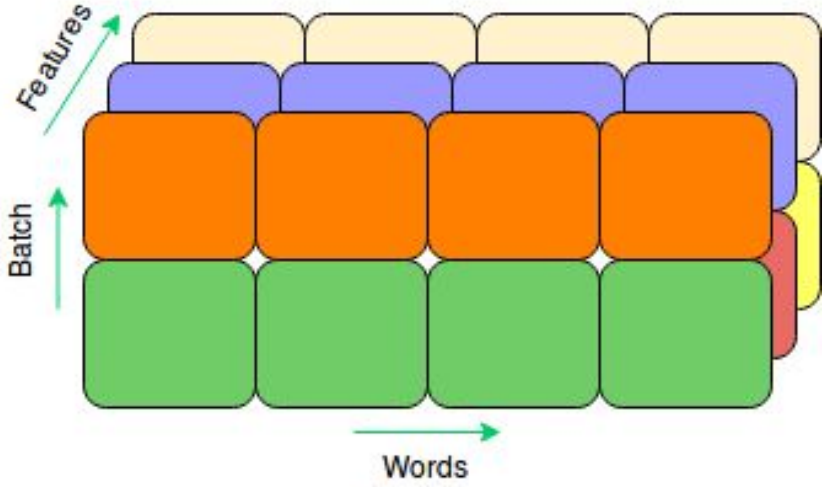


- Multi-head self-attention
- LayerNorm & Residual connections
- Position-wise feed-forward
- Positional Encoding

LayerNorm & Residual connections



Batch Norm



Layer Norm

LayerNorm & Residual connections

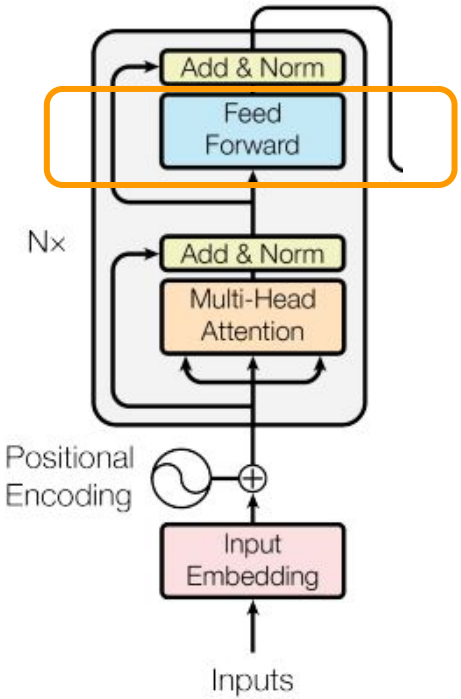
```
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

Residual connection

http://nlp.seas.harvard.edu/2018/04/03/attention.html

Encoder



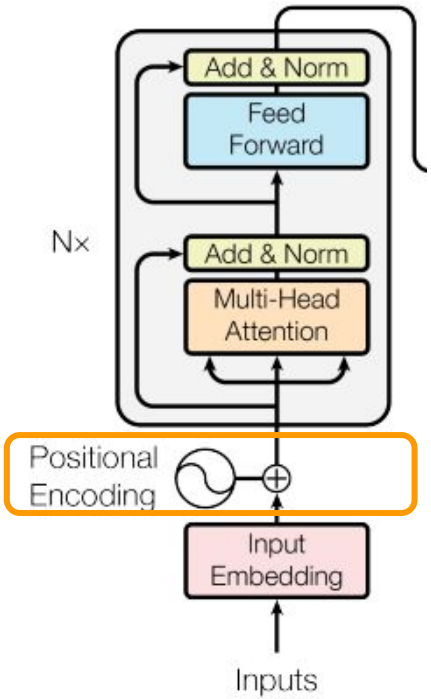
- Multi-head self-attention
- LayerNorm & Residual connections
- Position-wise feed-forward
- Positional Encoding

Position-wise feed-forward

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

- ReLu + two dense layers
- dim(input) = dim(output)

Encoder



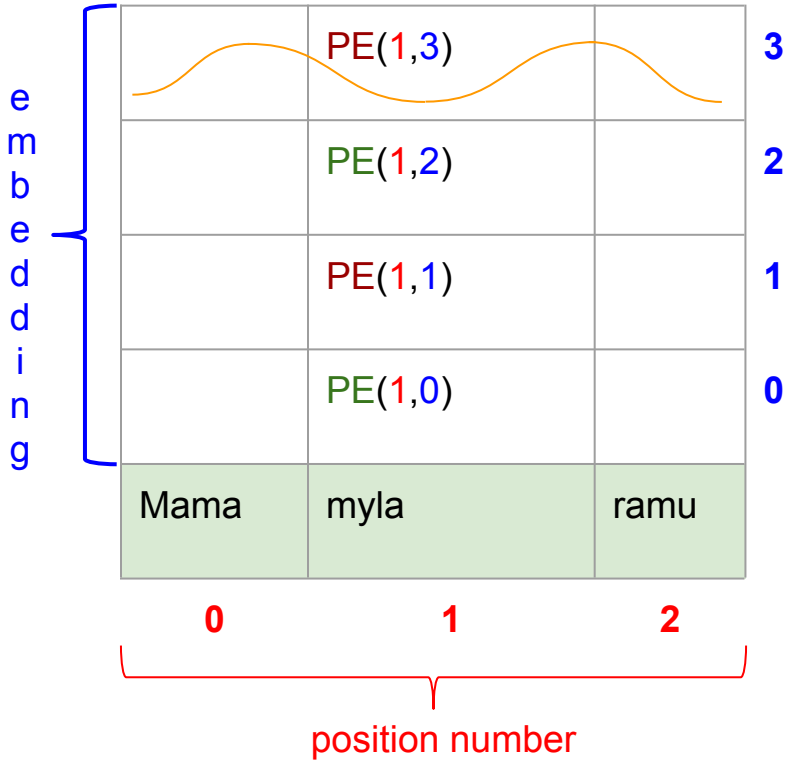
- Multi-head self-attention
- LayerNorm & Residual connections
- Position-wise feed-forward
- Positional Encoding

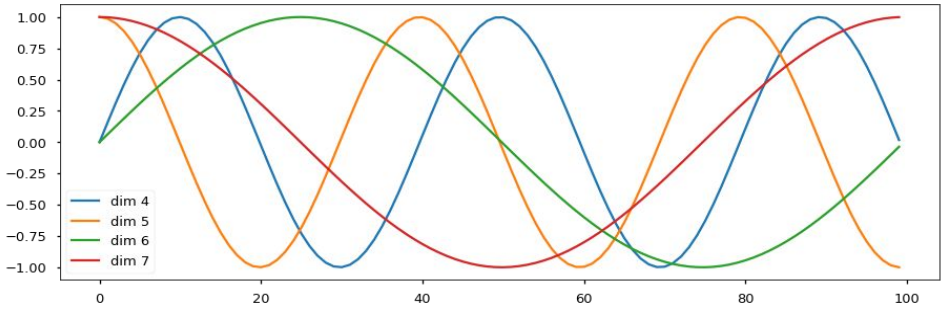
Positional encoding

- Learning embeddings for each position
  - (-) Bad embeddings for high-numbered positions
  - (-) More parameters needed
  - (-) Length restrictions
- Using sinusoids to encode position
  - **adding position vector to embedding vector**
  - concatenating position vector to embedding vector

Same scores with these two approaches

Positional encoding

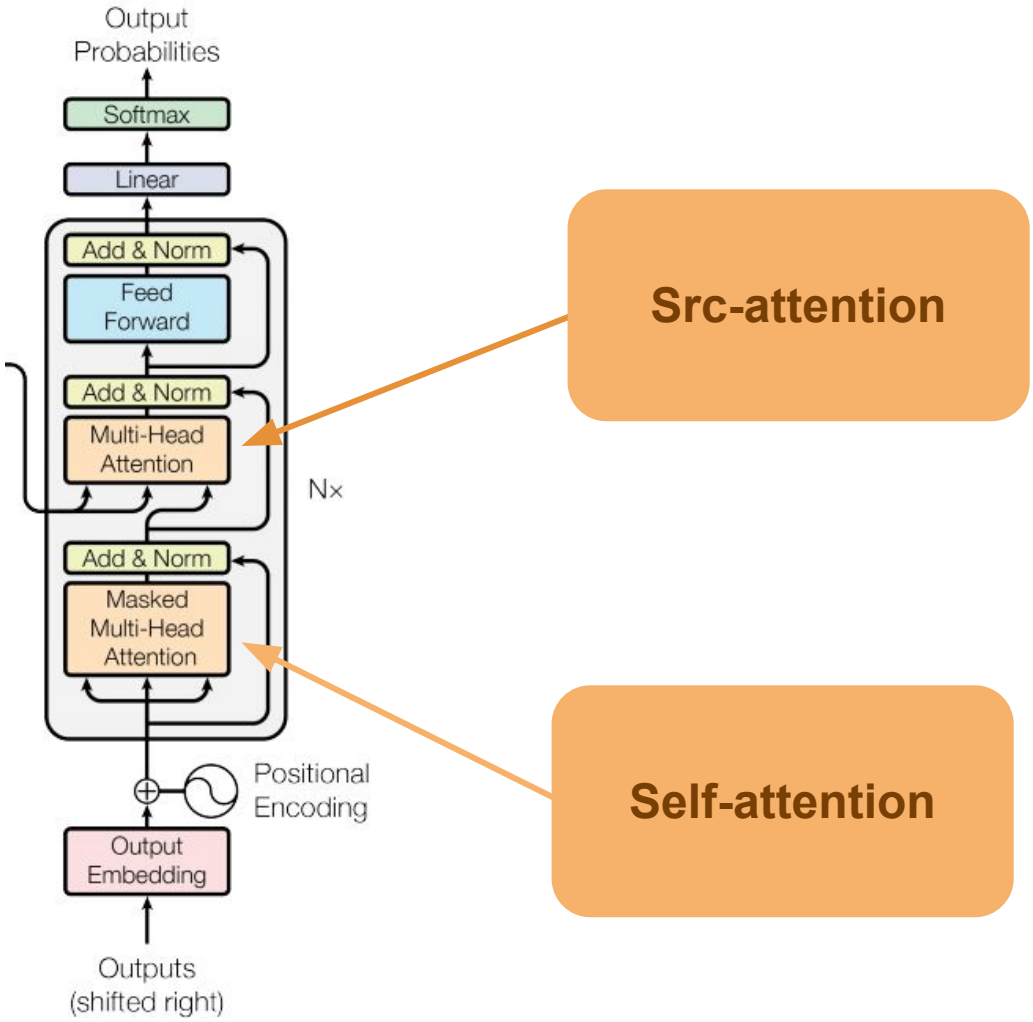




$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

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

Decoder


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

Decoder

```
class DecoderLayer(nn.Module):
    "Decoder is made of self-attn, src-attn, and feed forward (defined below)"
    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn = src_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        "Follow Figure 1 (right) for connections."
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)
```

http://nlp.seas.harvard.edu/2018/04/03/attention.html

https://research.googleblog.com/2017/08/transformer-novel-neural-network.html

Optional part

- Regularization
  (Label smoothing)
- BPE —
  Byte Pair Encoding for NMT
- Beam Search

Regularization

Label smoothing (LRS)

$$q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$$

- Replace true label distribution $q(k|x) = \delta_{k,y}$ with $q'(k|x)$
- $u(k)$ - independent from x distribution (e.g. uniform)
- ϵ - just a small real number
- $\delta_{k,y}$ - Dirac delta; equals 1 if k = y else equals 0;

This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

Szegedy et. al, “Rethinking the Inception Architecture for Computer Vision”, 2015
https://arxiv.org/pdf/1512.00567.pdf

BPE

**** Byte Pair Encoding (BPE)*** *(Gage, 1994) is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. We adapt this algorithm for word segmentation. Instead of merging frequent pairs of bytes, we merge characters or character sequences.*

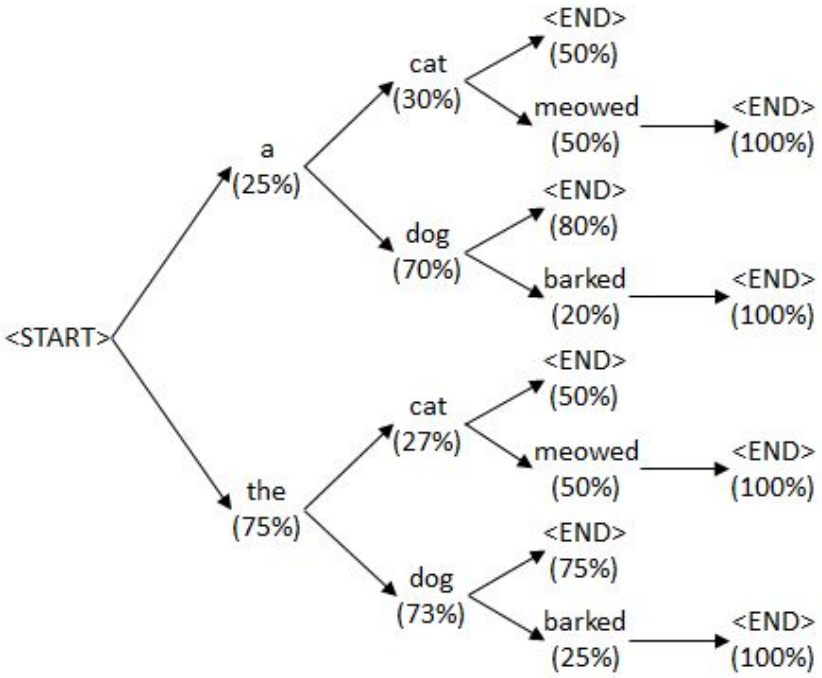
- *Official implementation: https://github.com/rsennrich/subword-nmt*

Example

- Input:
  - `"It is the case of Alexander Nikitin."`
- After applying BPE:
  - `It is the case of Alexander Ni@@ ki@@ tin .`

Rico Sennrich et al, "Neural Machine Translation of Rare Words with Subword Units", 2016
https://arxiv.org/pdf/1508.07909.pdf

Beam Search



PyTorch OpenNMT implementation:
https://github.com/OpenNMT/OpenNMT-py/blob/master/onmt/translate/Beam.py


вопросы