

Лекция 9. Механизм внимания. Трансформеры.

Глубинное обучение

Антон Кленицкий

Recap

Simple RNN

Хотим, чтобы модель обрабатывала последовательности любой длины

- Сохраняем вектор скрытого состояния (hidden state), передаем его на следующий шаг
- Weights sharing - применение одних и тех же весов на каждом шаге

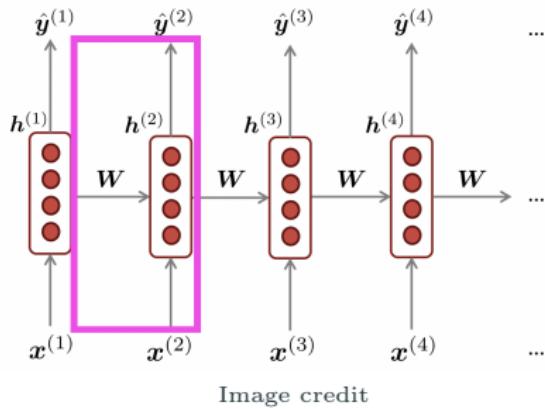


Image credit

Simple RNN

Количество параметров не зависит от длины последовательности

$$h_0 \leftarrow init$$

$$h_t = f_1(W_h h_{t-1} + W_x x_t + b_h)$$

$$y_t = f_2(U h_t + b_y)$$

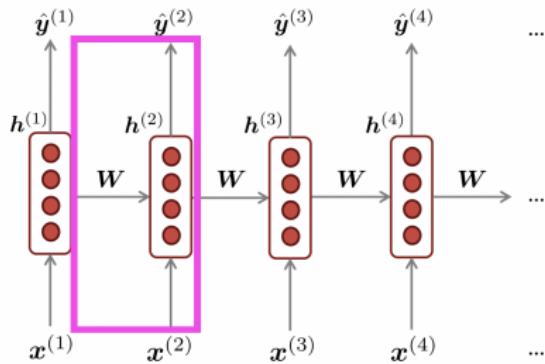


Image credit

RNN language model

A Simple RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}\mathbf{h}^{(t)} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

$$\hat{y}^{(4)} = P(\mathbf{x}^{(5)} | \text{the students opened their})$$



hidden states

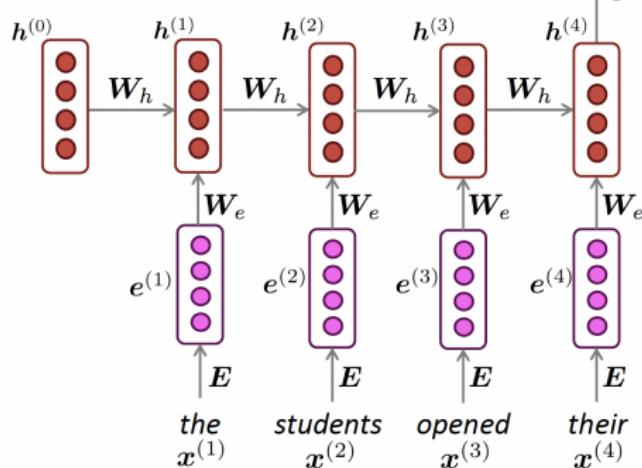
$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1)$$

$\mathbf{h}^{(0)}$ is the initial hidden state

word embeddings

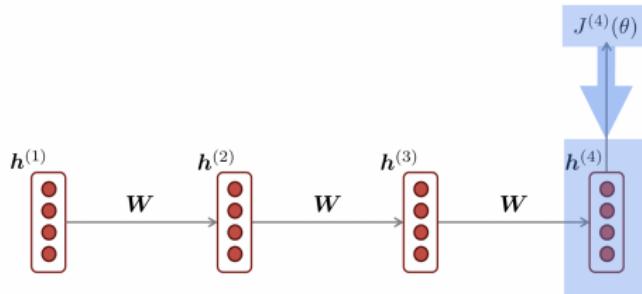
$$\mathbf{e}^{(t)} = \mathbf{E}\mathbf{x}^{(t)}$$

words / one-hot vectors
 $\mathbf{x}^{(t)} \in \mathbb{R}^{|V|}$



Vanishing / exploding gradients

Многократное умножение на **одну и ту же** матрицу весов
 $W \Rightarrow$ норма градиента растет или убывает экспоненциально



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

Image credit

- Exploding gradients \Rightarrow нестабильное обучение
Решение - gradient clipping
- Vanishing gradients \Rightarrow модель не способна выучить долгосрочные зависимости

LSTM (Long Short-Term Memory)

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad \text{forget gate}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad \text{input gate}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad \text{output gate}$$

- forget gate - какую часть забыть из предыдущего cell state
- input gate - какую часть нового cell state использовать
- output gate - какую часть cell state записать в hidden state

$$c'_t = \tanh(W_c x_t + U_c h_{t-1} + b_{c'}) \quad \text{candidate cell state}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot c'_t \quad \text{cell state}$$

$$h_t = o_t \odot \tanh(c_t) \quad \text{hidden state}$$

GRU

GRU (Gated Recurrent Unit) - упрощенная версия

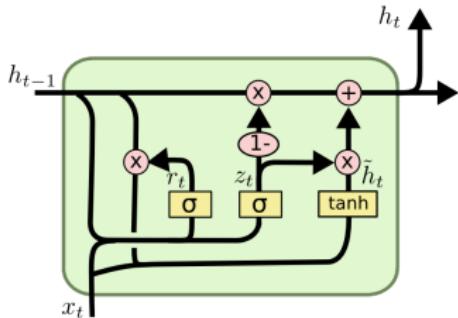


Image credit

$$u_t = \sigma(W_u x_t + U_u h_{t-1} + b_u) \quad \text{update gate}$$

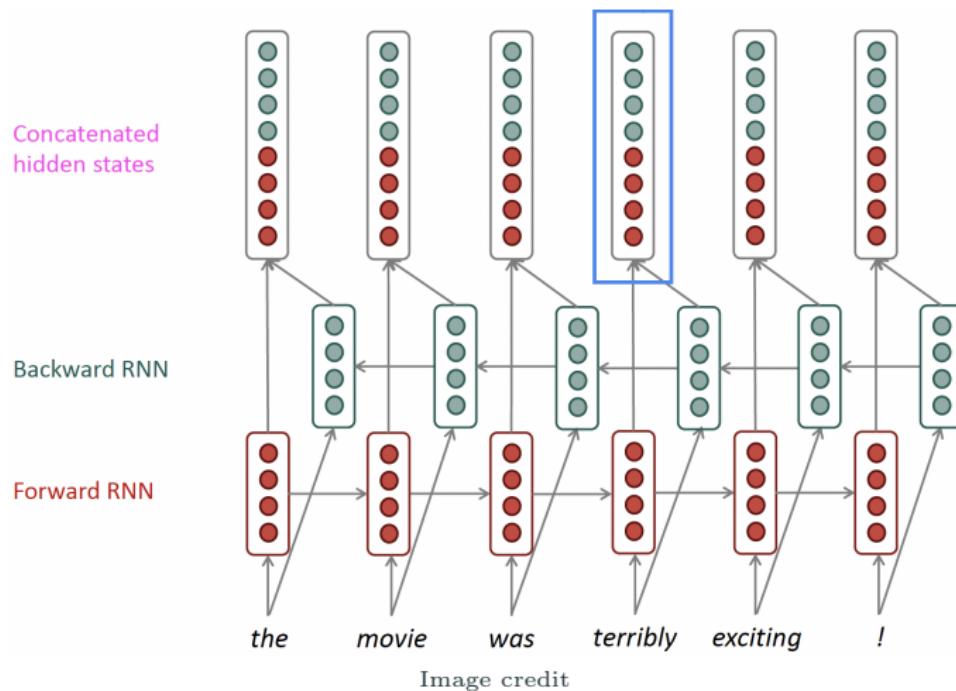
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad \text{reset gate}$$

$$h'_t = \tanh(W_{h'} x_t + W_h(r_t \odot h_{t-1})) \quad \text{hidden state candidate}$$

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot h'_t \quad \text{hidden state}$$

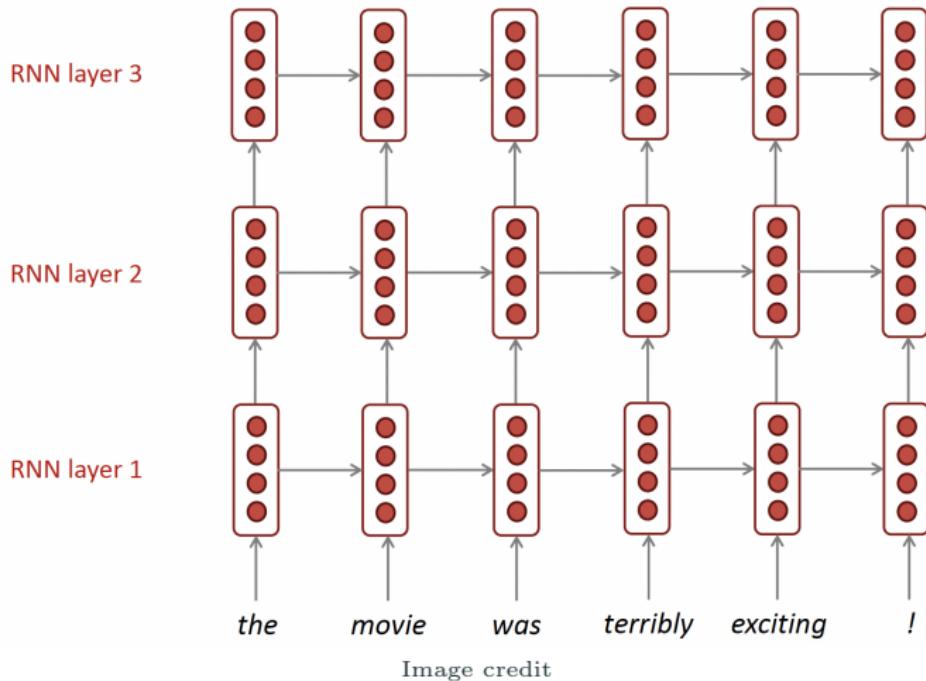
Bidirectional RNN

- Bidirectional RNN - двухнаправленная RNN
- Учет контекста с обеих сторон

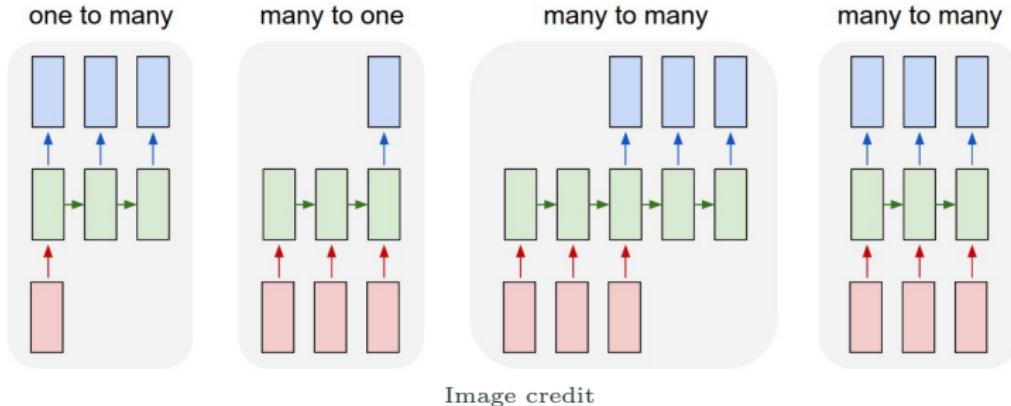


Multilayer (stacked) RNN

Актуальны skip connections, layer normalization



Применение RNN



Примеры:

- one to many - описание изображения (image captioning)
- many to one - классификация текста
- many to many - машинный перевод
- synced many to many - LM, NER

Sequence-to-sequence models

Seq2seq

Преобразование последовательности произвольной длины на входе в последовательность произвольной длины на выходе

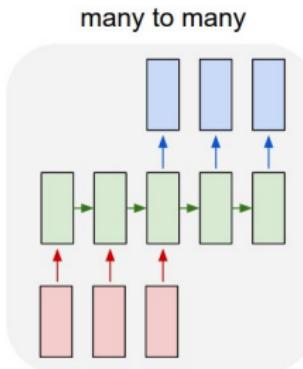
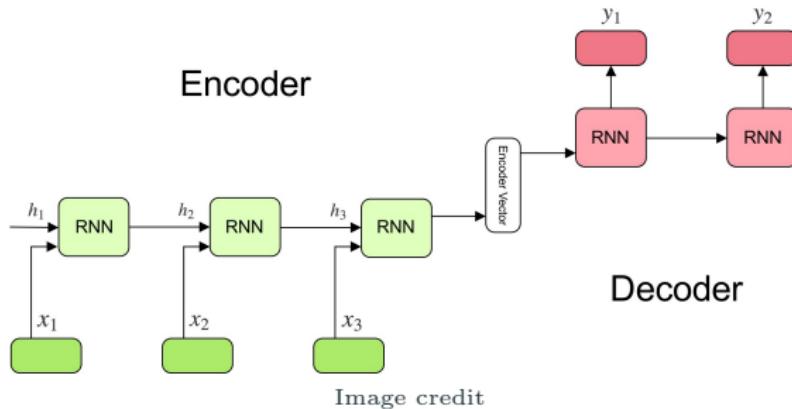


Image credit

Например:

- machine translation
- summarization
- dialogue

Encoder-Decoder architecture

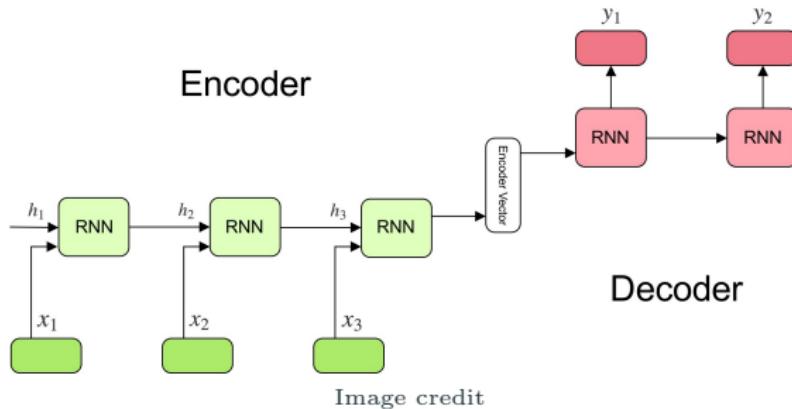


Encoder сворачивает входную последовательность в вектор контекста c (как правило, последнее скрытое состояние h_T):

$$h_t = f(x_t, h_{t-1})$$

$$c = h_T$$

Encoder-Decoder architecture



Decoder предсказывает следующий элемент на основе скрытого состояния s_t , предыдущего элемента y_{t-1} и вектора контекста c (condititonal generation):

$$p(y_t | y_1, \dots, y_{t-1}, c) = f(y_{t-1}, s_t, c)$$

$$s_t = f(y_{t-1}, s_{t-1}, c)$$

Encoder-Decoder training

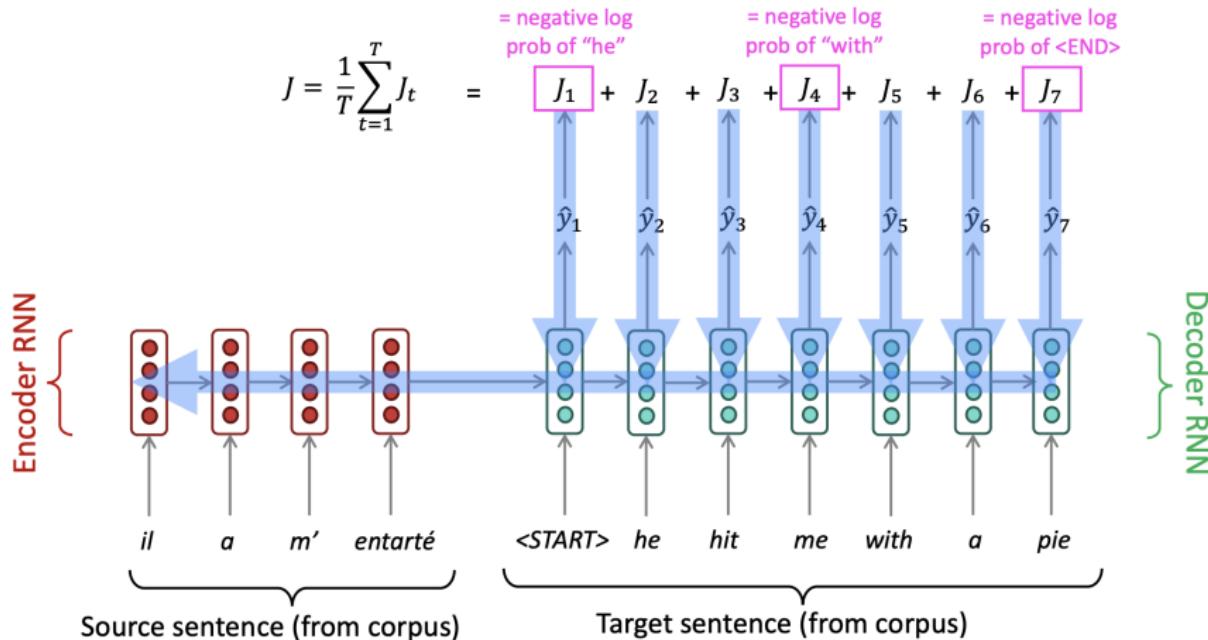


Image credit

Multilayer Encoder-Decoder

На практике обычно многослойный вариант

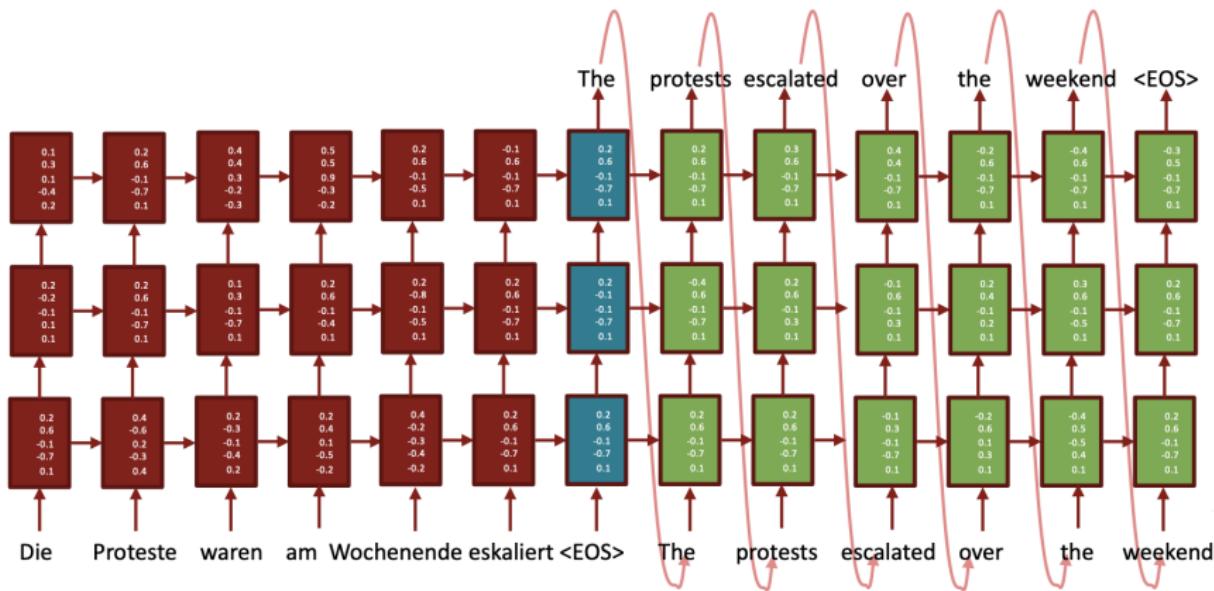


Image credit

Проблема такого подхода

- Informational bottleneck - в векторе контекста фиксированной длины должна содержаться вся информация о входной последовательности
- Модель плохо работает для длинных последовательностей

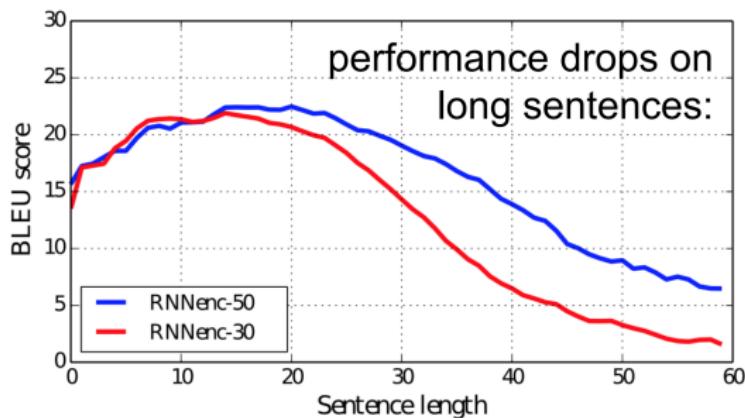
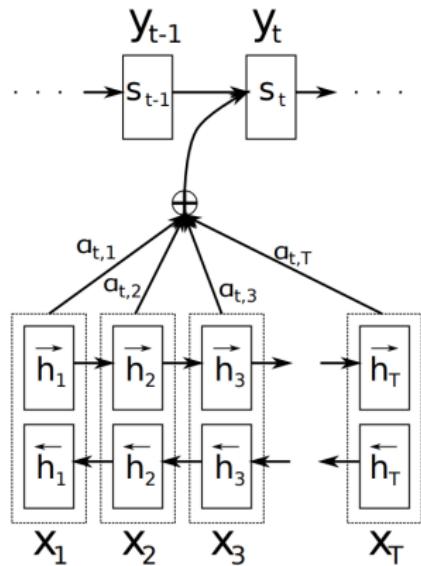


Image credit

Attention

Attention

Bahdanau, D., Cho, K., & Bengio, Y. (2015) Neural machine translation by jointly learning to align and translate.



- Умный pooling
- Вектор контекста c_t свой на каждом шаге декодера
- c_t - сумма h_i со всех шагов энкодера с обучаемыми весами
- Внимание выбирает релевантные элементы во входной последовательности

Attention

Скрытое состояние декодера

$$s_t = f(y_{t-1}, s_{t-1}, c_t)$$

Attention

Скрытое состояние декодера

$$s_t = f(y_{t-1}, s_{t-1}, c_t)$$

Вектор контекста - взвешенная сумма всех h_i энкодера

$$c_t = \sum_{i=1}^n \alpha_{ti} h_i$$

Attention

Скрытое состояние декодера

$$s_t = f(y_{t-1}, s_{t-1}, c_t)$$

Вектор контекста - взвешенная сумма всех h_i энкодера

$$c_t = \sum_{i=1}^n \alpha_{ti} h_i$$

Alignment score

$$\text{score}(s_{t-1}, h_i) = v^T \tanh(W_1 s_{t-1} + W_2 h_i)$$

v, W_1, W_2 - обучаемые параметры

Беса

$$\alpha_{ti} = \text{softmax}(\text{score}(s_{t-1}, h_i)) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{j=1}^n \exp(\text{score}(s_{t-1}, h_j))}$$

Виды attention

Можно по-разному считать $\text{score}(s_{t-1}, h_i)$

- Additive attention

$$\text{score}(s_{t-1}, h_i) = v^T \tanh(W_1 s_{t-1} + W_2 h_i)$$

- Multiplicative attention

$$\text{score}(s_{t-1}, h_i) = s_{t-1}^T W h_i$$

- Dot-product

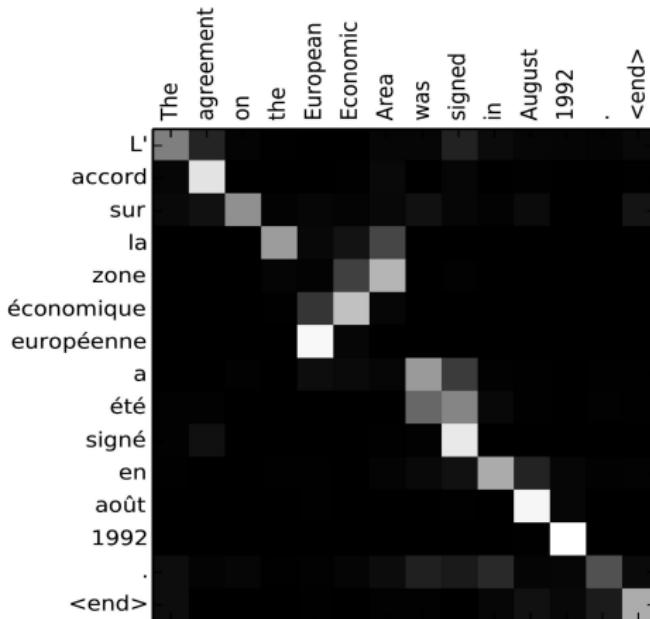
$$\text{score}(s_{t-1}, h_i) = s_{t-1}^T h_i$$

- Scaled dot-product

$$\text{score}(s_{t-1}, h_i) = s_{t-1}^T h_i / \sqrt{d}$$

Интерпретация attention

Матрица весов α_{ti} определяет alignment (выравнивание) элементов входной и выходной последовательностей



Alignment matrix (Image credit)

Self-attention

- Attention позволяет получить представление входной последовательности
- как и RNN
- Почему бы не отказаться от рекуррентности полностью?

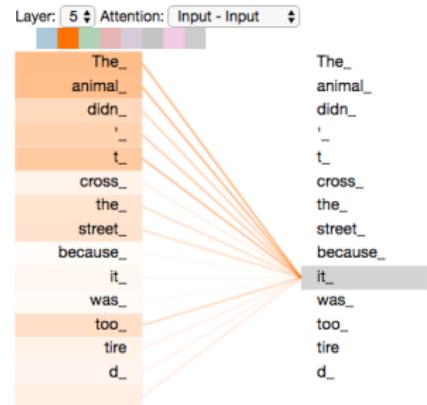


Image credit

Query, Key, Value

Dot-product attention

$$\text{output}_i = \sum_j \text{softmax}(s_{i-1}^T h_j) h_j$$

Можно переписать в виде

$$\text{output}_i = \sum_j \text{softmax}(q_i^T k_j) v_j$$

где

$$q_i \equiv s_{i-1} \quad \text{query}$$

$$k_i \equiv h_i \quad \text{key}$$

$$v_i \equiv h_i \quad \text{value}$$

Query, key, value - терминология из information retrieval

Self-attention

x_i - элементы последовательности с размерностью d

Self-attention

x_i - элементы последовательности с размерностью d

$$q_i = W_q x_i \quad \text{query}$$

$$k_i = W_k x_i \quad \text{key}$$

$$v_i = W_v x_i \quad \text{value}$$

Добавили матрицы $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ для большей выразительности

Self-attention

x_i - элементы последовательности с размерностью d

$$q_i = W_q x_i \quad \text{query}$$

$$k_i = W_k x_i \quad \text{key}$$

$$v_i = W_v x_i \quad \text{value}$$

Добавили матрицы $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ для большей выразительности

$$y_i = \sum_j \text{softmax}\left(\frac{q_i^T k_j}{\sqrt{d}}\right) v_j$$

Делим на \sqrt{d} , чтобы лучше обучалось, градиенты софтмакса не были слишком маленькими

Self-attention

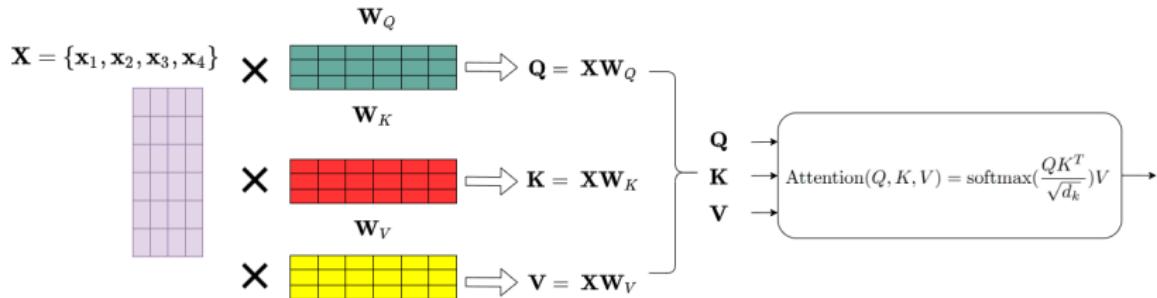
В матричном виде

$$Q = XW^Q$$

$$K = XW^K$$

$$V = XW^V$$

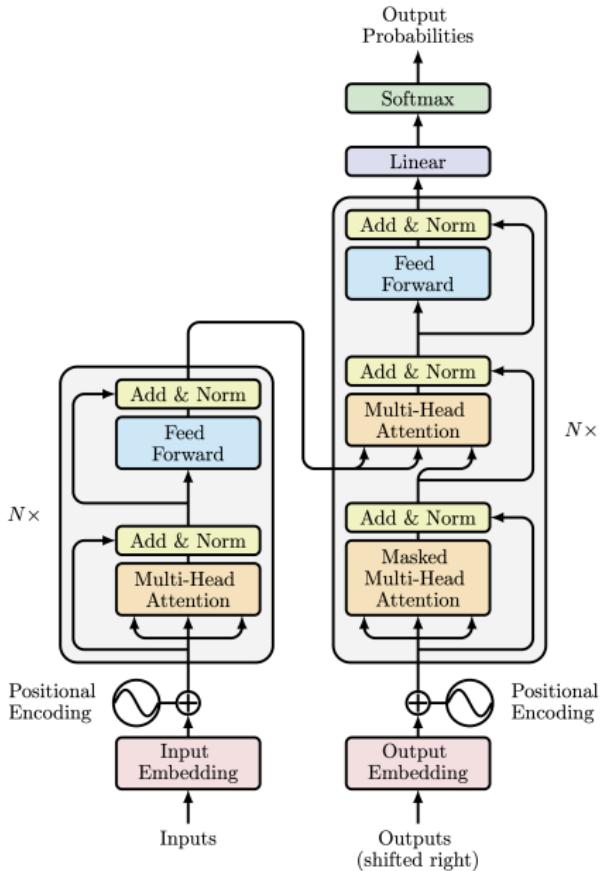
$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d}} \right) V$$



Transformers

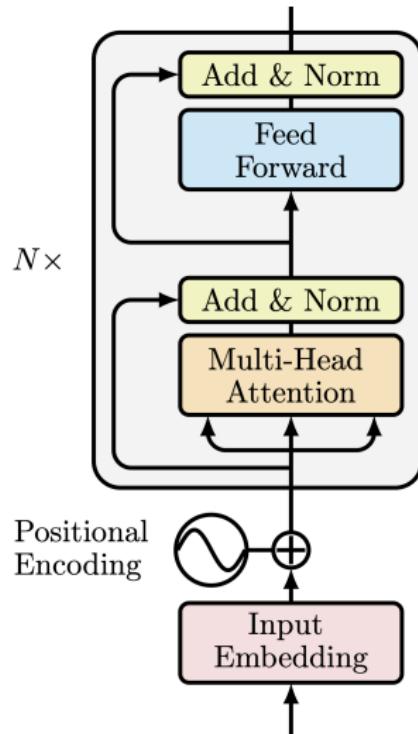
Transformer

- Vaswani A. et al. (2017)
Attention is all you need.
- Encoder-Decoder
архитектура
- Изначально придумали
для machine translation
- Трансформеры стали
универсальной
архитектурой для
обработки
последовательностей



Transformer Encoder

- Embedding layer
- Positional encoding
- N Transformer blocks



Positional encoding

- Self-attention слой не знает о порядке элементов в последовательности!
- Даем эту информацию модели с помощью Positional encoding
- Прибавляем к эмбеддингам токенов позиционные эмбеддинги



Image credit

Positional encoding

Fixed positional encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

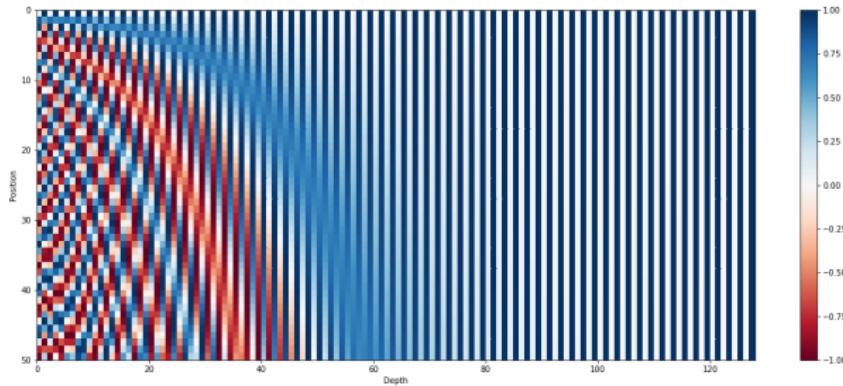


Image credit

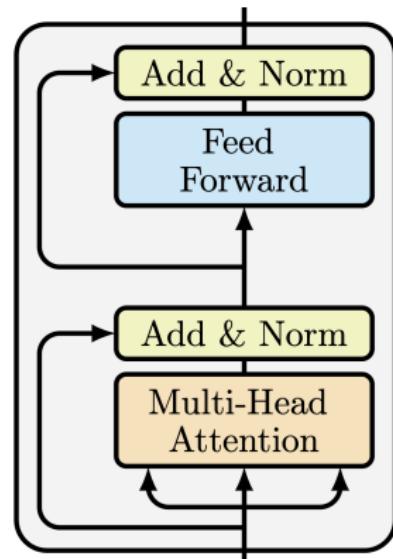
Positional encoding

Learned positional encoding

- Можно обучать позиционные эмбеддинги вместе с моделью
- Рассматриваем позицию как категориальную переменную (от 0 до $t - 1$)
- Учим дополнительный слой эмбеддингов для этой переменной

Transformer block

- Multi-head self-attention layer
- Pointwise feed-forward layer
- Add & Norm - residual connections + layer normalization



Multi-head self-attention

- Несколько независимых «голов» внимания, которые потом объединяются
- Чтобы не увеличивать количество вычислений, размерность каждой из h «голов» в h раз меньше размерности входа

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d/h}}\right)V$$

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

$$W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d/h}$$

$$\text{MultiHead}(Q, K, V) = [\text{head}_1; \dots; \text{head}_h]W^O$$

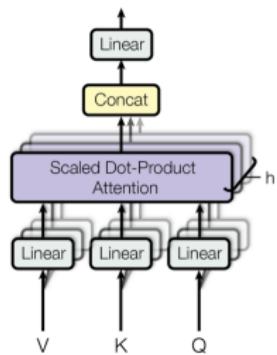


Image credit

Pointwise feed-forward network

- Self-attention - линейная комбинация values
- Нужна нелинейность!
- Для этого добавляем Pointwise feed-forward network

$$FFN(x) = W_2 \max(0, W_1 x + b_1) + b_2$$

Один и те же веса применяются к каждому элементу последовательности независимо

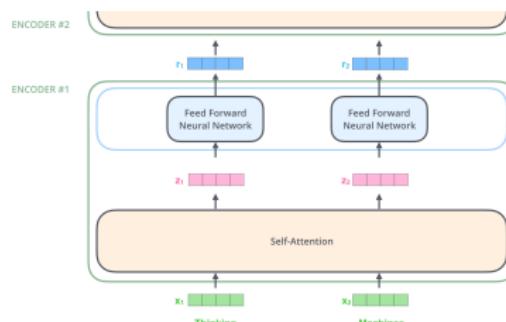
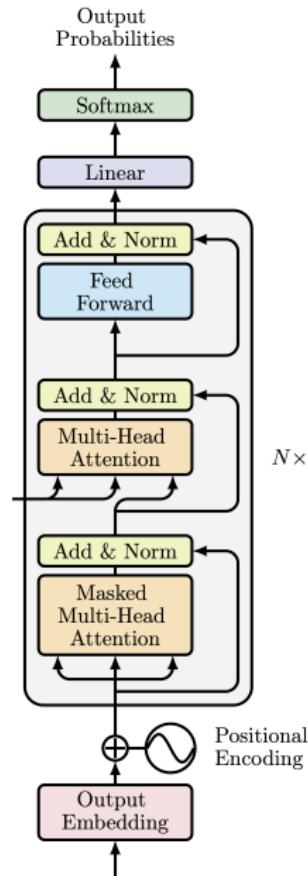


Image credit

Transformer Decoder

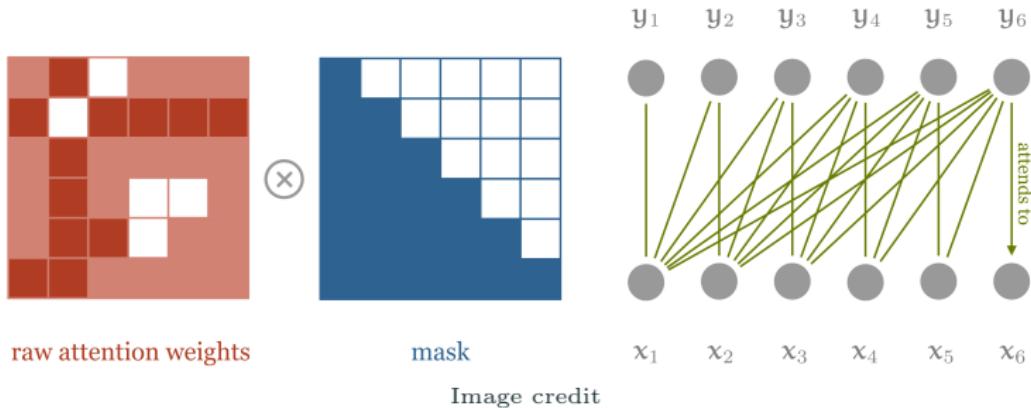
То же, что и в Encoder, плюс:

- Masked self-attention
- Дополнительный слой с Cross-attention
- Linear + Softmax на выходе



Masked Self-Attention

Нужно, чтобы декодер смотрел только на предыдущие элементы, не заглядывал в будущее



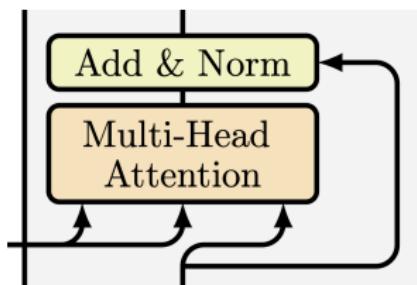
$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d}}\right)V$$

$$M_{ij} = \begin{cases} 0, & j \leq i \\ -\infty, & j > i \end{cases}$$

Cross-attention

Encoder - Decoder Attention

- Keys, values - выходы из Encoder'a
- Queries - выходы предыдущего слоя Decoder'a



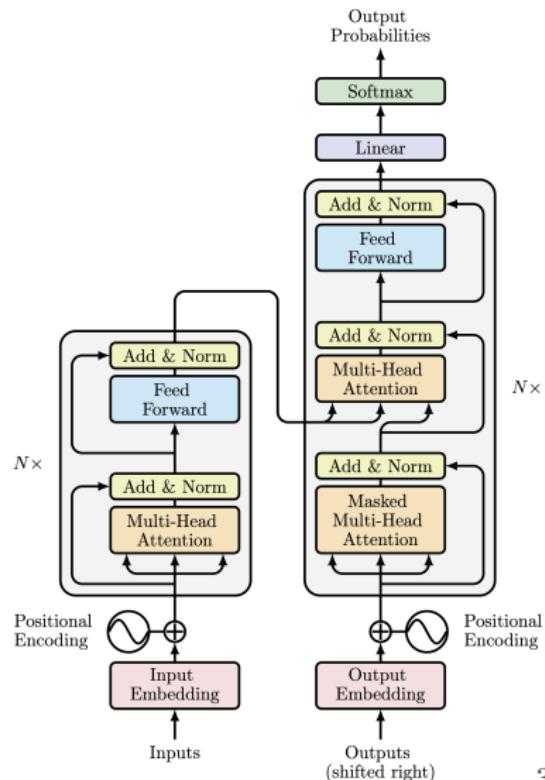
Transformer summary

Pros

- Self-attention может улавливать долгосрочные зависимости
- Параллелизуется, в отличие от RNN

Cons

- Квадратичная зависимость self-attention от длины последовательности



Attention

- <https://lilianweng.github.io/posts/2018-06-24-attention/>
- <https://theaisummer.com/attention/>

Transformer

- The Illustrated Transformer
- The Annotated Transformer
- <https://theaisummer.com/transformer/>
- <https://peterbloem.nl/blog/transformers>

В следующий раз

- Pretrained transformers in NLP
- BERT
- GPT
- and others..