

Лекция 8. Рекуррентные нейронные сети. Трансформеры

Денис Деркач, Дмитрий Тарасов

Использовались слайды Антона Кленицкого, Лены Войта

24 марта 2025 года



LAMBDA • HSE

Напоминания



Представление текста

Первый шаг - токенизация (разбиение текста на отдельные слова - токены)

Дальше нужно перевести слова в числовой вид

Хотим представить слова в виде векторов небольшой размерности, которые отражают их смысл, чтобы

- ▶ Близкие по смыслу слова имели похожие вектора
- ▶ Разные по смыслу слова имели непохожие вектора

Основная идея:

Слова, которые часто встречаются в схожих контекстах, имеют похожее значение
(distributional hypothesis)

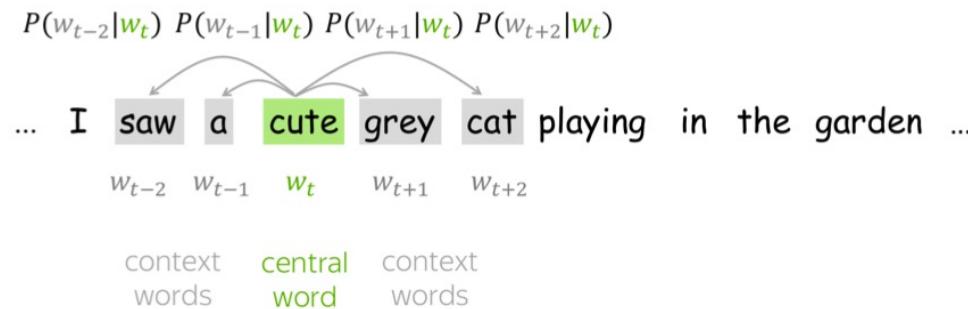
Word2Vec

Общая идея:

- ▶ Вектора слов, которые встречаются в одном контексте, должны быть близки

Реализация:

- ▶ Учимся предсказывать контекст (окружающие слова) по вектору данного слова



Word2Vec

Алгоритм:

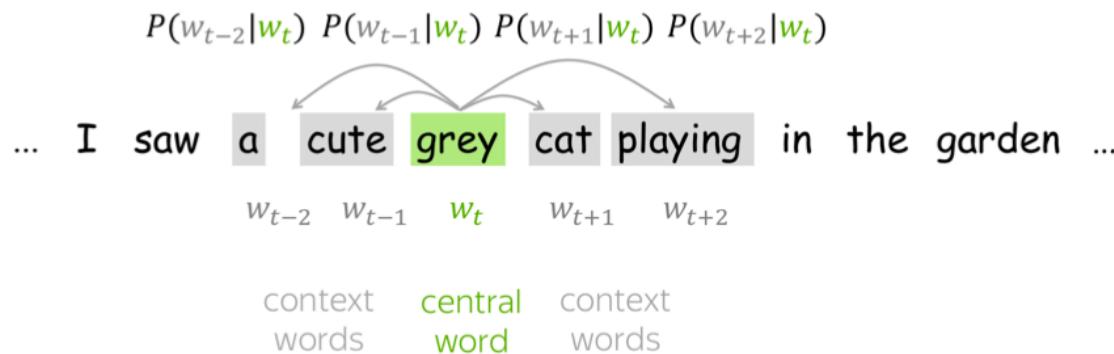
- ▶ Берем большой корпус текстов
- ▶ Проходим по текстам скользящим окном, смещаемся на одно слово на каждом шаге
- ▶ В каждом окне есть центральное слово и слова контекста (остальные слова в окне)
- ▶ Предсказываем вероятность окружающих слов на основе вектора центрального слова



Word2Vec – функция потерь

Функция потерь – отрицательное лог-правдоподобие:

$$L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j}|w_t, \theta)$$



Word2Vec – Обучение

Для каждого слова обучаются два вектора - v_w когда слово в центре и u_w когда слово в контексте.

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

o - outside word, c - central word

Обучаем стохастическим градиентным спуском:

$$L_{t,j}(\theta) = -\log P(o|c) = -u_o^T v_c + \log \sum_{w \in V} \exp(u_w^T v_c)$$

$$v_c = v_c - \alpha \frac{\partial L_{t,j}(\theta)}{\partial v_c}$$

$$u_w = u_w - \alpha \frac{\partial L_{t,j}(\theta)}{\partial u_w}$$

Увеличиваем близость между v_c и u_o и уменьшаем близость между v_c и всеми остальными u_w .

Word2Vec – Negative Sampling

$$L_{t,j}(\theta) = -\log P(o|c) = -u_o^T v_c + \log \sum_{w \in V} \exp(u_w^T v_c)$$

Словарь очень большой, на каждом шаге SGD обновляем вектора всех слов u_w - долго.

Выход - сэмплирование отрицательных примеров

$$\sum_{w \in V} \exp(u_w^T v_c) \rightarrow \sum_{w \in \{w_{i_1}, \dots, w_{i_K}\}} \exp(u_w^T v_c)$$

Методы построения эмбеддингов

- ▶ Word2Vec: используйте, когда семантические связи имеют решающее значение, и у вас большой набор данных.
- ▶ GloVe: подходит для разнообразных наборов данных и когда важен охват глобального контекста.
- ▶ FastText: выбирайте морфологически богатые языки или когда обработка слов, не входящих в словарный запас, имеет решающее значение.

Word embeddings + нейросети



Subword Tokenization

Subword tokenization - баланс между представлением на уровне символов и представлением на уровне слов

На уровне символов

- Слишком длинные последовательности
- Теряется семантика, смысл

На уровне слов

- Проблема out-of-vocabulary words
- Слишком большой словарь

Subword Tokenization

- Часто используемые слова рассматриваются как отдельные токены
- Редкие слова раскладываются на несколько осмысленных подслов

В результате

- Ограниченный размер словаря
- Осмысленные представления
- Способность обработать любое новое слово
- Один токенизатор для всех языков сразу

Byte Pair Encoding

- Берем большой корпус текстов
- Заранее определяем размер словаря (например, 50к)
- Начинаем с отдельных символов как токенов
- На каждой итерации сливаем два токена, которые имеют максимальную частоту совместной встречаемости
- Заканчиваем, когда достигнут размер словаря или максимальная частота равна 1

AABABCABBAABAC

AA - 2

AB - 4 AB = D

BA - 3

BC - 1

CA - 1

BB - 1

AC - 1

ADDcdbadac

AD - 2 AD = E

DD - 1

DC - 1

CD - 1

DB - 1

DA - 1

AC - 1

EDCdbeac

Рекуррентные нейронные сети



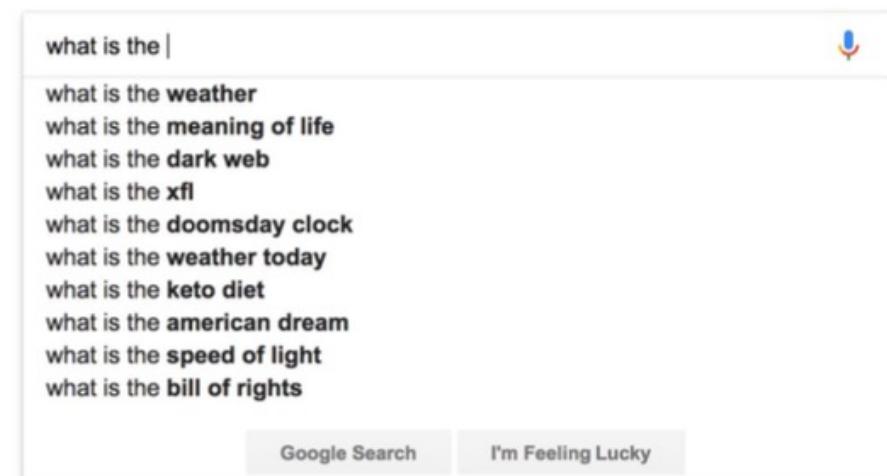
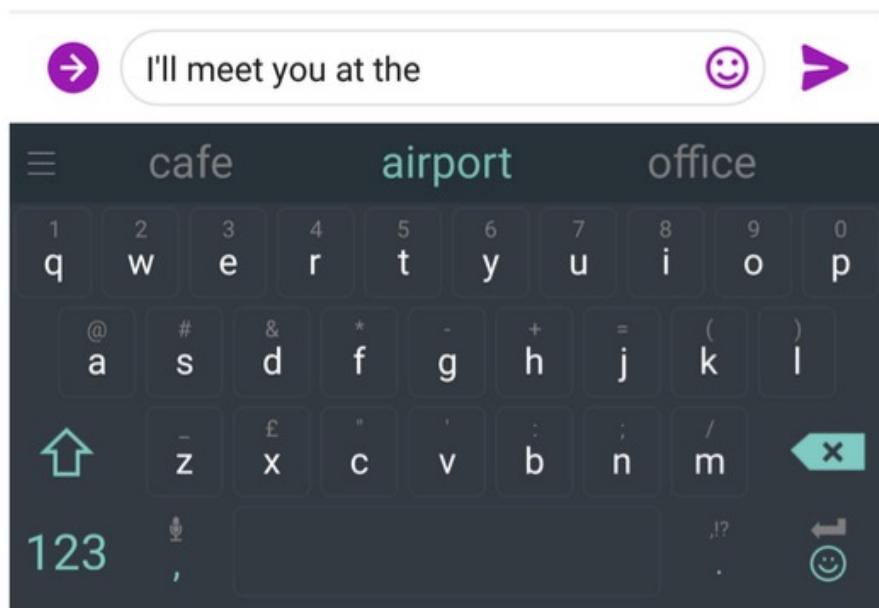
Типы данных

Часто нам встречаются последовательности данных

- ▶ Тексты: понимание речи, генерация текста
- ▶ Аудио
- ▶ Временные ряды
 - Анализ финансовых данных: фондовый рынок, сырьевые товары, Forex
 - Здравоохранение: частота пульса, уровень сахара (от медицинского оборудования и носимых устройств)
- ▶ Последовательности нуклеотидов в ДНК
- ▶ Пространственно-временные данные:
 - Самоуправление и отслеживание объектов
 - Тектоническая активность плит

Language model

Language model (языковая модель) – предсказание следующего слова по предыдущим



Language model

Формально - дана последовательность слов x_1, x_2, \dots, x_t ,
нужно вычислить вероятность следующего слова x_{t+1} :

$$P(x_{t+1}|x_1, \dots, x_t)$$

С помощью языковой модели можно назначить вероятность
любому отрывку текста:

$$\begin{aligned} P(x_1, \dots, x_t) &= P(x_1) \times P(x_2|x_1) \times \dots \times P(x_t|x_1, \dots, x_{t-1}) \\ &= \prod_{t=1}^T P(x_t|x_1, \dots, x_{t-1}) \end{aligned}$$

Language model

Классический вариант - подсчет n-грамм.

$$\begin{aligned} P(\text{books}|\text{the students opened their}) &= \\ &= \frac{P(\text{the students opened their books})}{P(\text{the students opened their})} \approx \\ &\approx \frac{\text{count}(\text{the students opened their books})}{\text{count}(\text{the students opened their})} \end{aligned}$$

Fixed-window neural net

output distribution

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

hidden layer

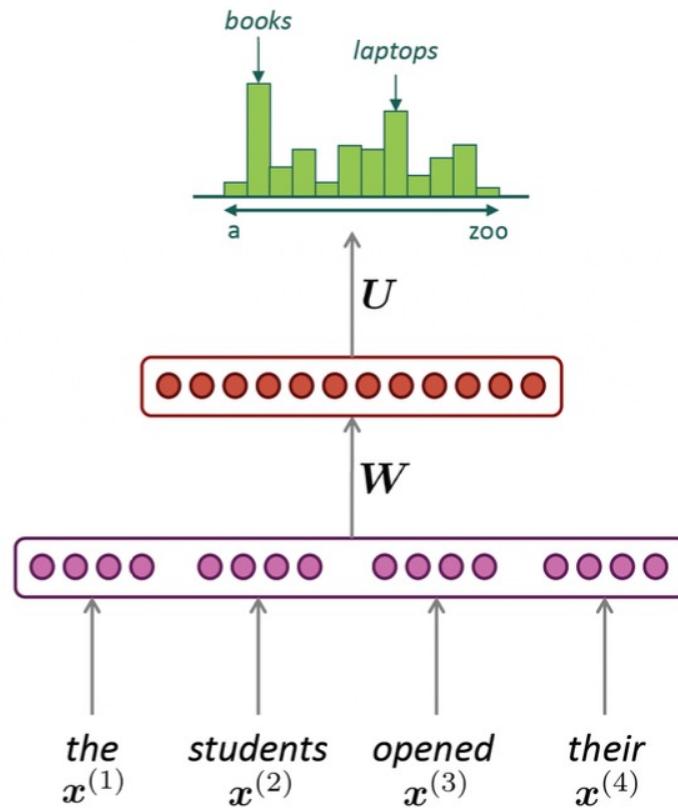
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

$$\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$

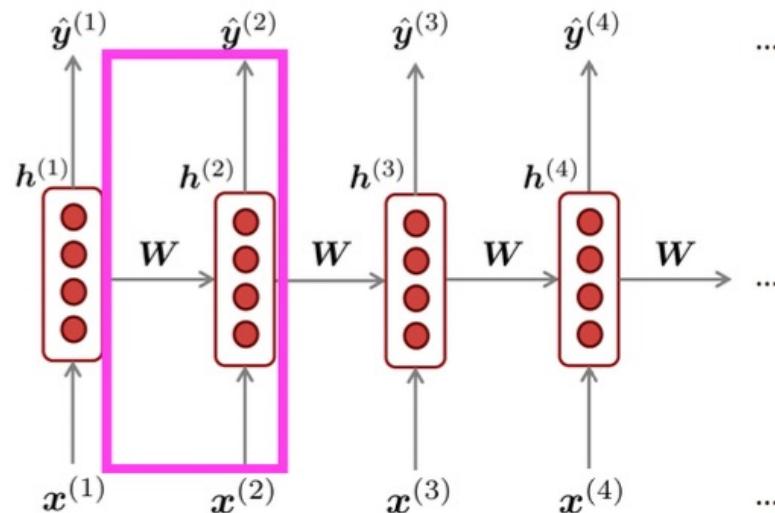


<https://web.stanford.edu/class/cs224n/>

Скрытые состояния

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



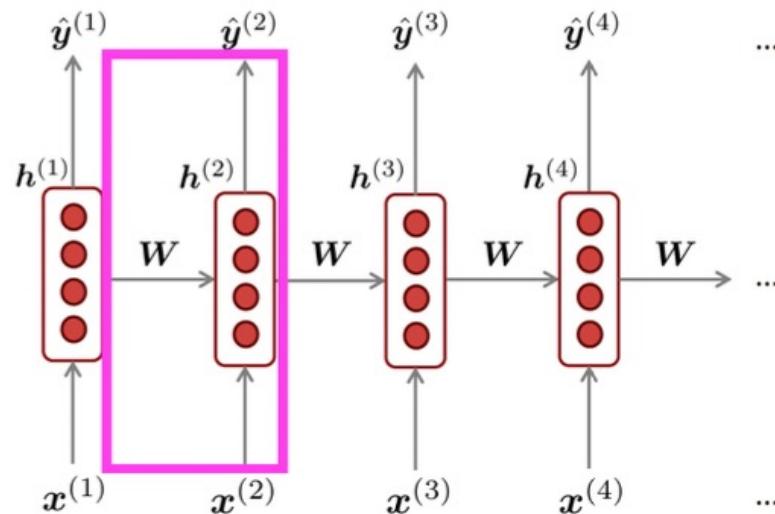
Скрытое состояния

Количество параметров не зависит от длины последовательности

$$h_0 \leftarrow init$$

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

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



RNN language model

A Simple RNN Language Model

output distribution

$$\hat{y}^{(t)} = \text{softmax}(\mathbf{U}h^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma(\mathbf{W}_h h^{(t-1)} + \mathbf{W}_e e^{(t)} + b_1)$$

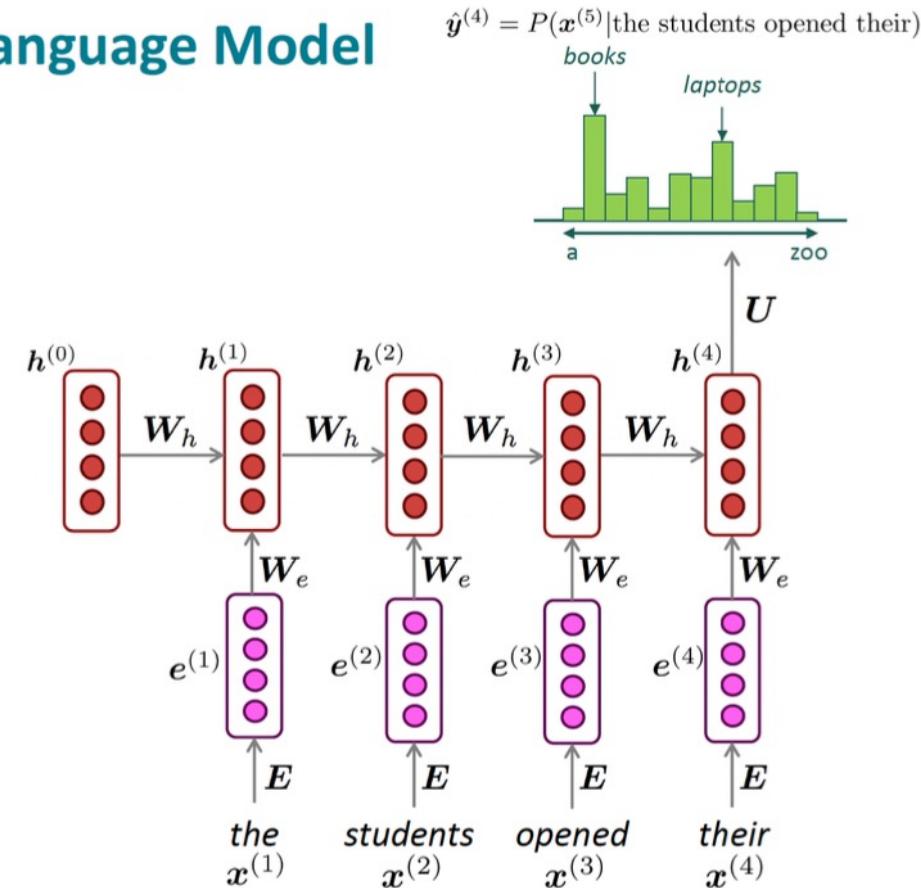
$h^{(0)}$ is the initial hidden state

word embeddings

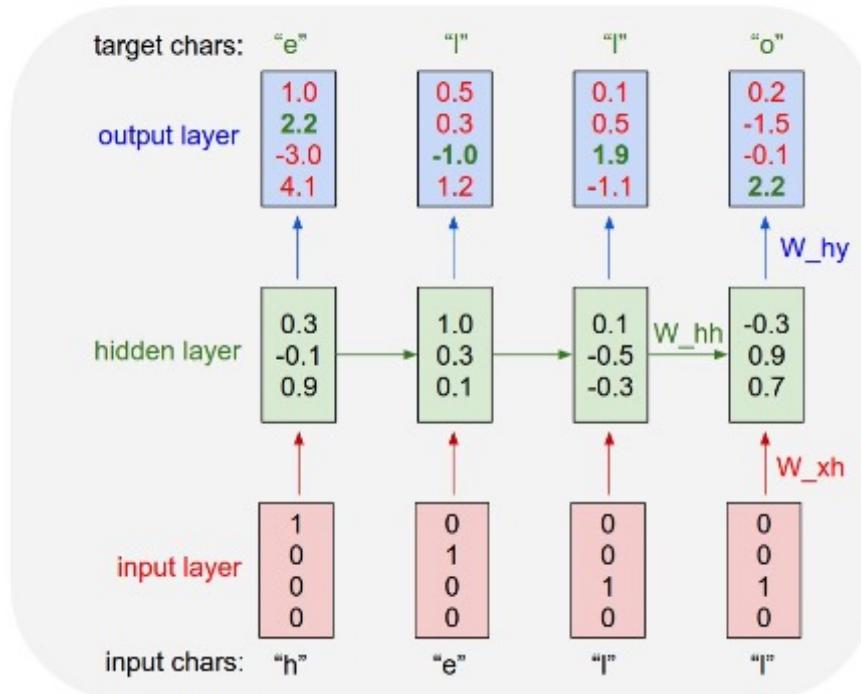
$$e^{(t)} = \mathbf{E}x^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



Простой RNN для генерации



› $xi \in \{h, e, l, o\}$. One can use one-hot encoding. We want to build "Hello".

› Autoregressive rule:

$$p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \dots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l).$$

› $p(x_2 = e|x_1 = h) = \text{softmax}(o_1);$

$$o_1 = W_{hy}h_1;$$

$$h_1 = \tanh(W_{hh}h_0 + W_{xh}x_1).$$

[A. Karpathy's blog](#)

Д. Деркач@DL08

Генерация текстов

Addition of LSTM layers creates a possibility to generate long texts.

Train 3-layer RNN with 512 hidden nodes on all the works of Shakespeare.
Then sample from the model:

KING LEAR: O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

Note: generation happens **character by character**. Needs to learn valid words, grammar, punctuation, etc.

Генерация LaTex кода

Proof. Omitted. \square

Lemma 0.1. Let \mathcal{C} be a set of the construction.

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. \square

Lemma 0.2. This is an integer \mathcal{Z} is injective.

Proof. See Spaces, Lemma ??.

Lemma 0.3. Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. \square

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccccc}
 S & \xrightarrow{\quad} & & & \\
 \downarrow & & & & \\
 \xi & \longrightarrow & \mathcal{O}_{X'} & & \\
 \text{gor}_s & & \uparrow & \searrow & \\
 & & =\alpha' & \longrightarrow & \\
 & & \downarrow & & \\
 & & =\alpha' & \longrightarrow & \alpha \\
 & & & & \\
 \text{Spec}(K_\psi) & & \text{Mor}_{\text{Sets}} & & d(\mathcal{O}_{X_{/\mathbb{A}}}, \mathcal{G}) \\
 & & & & \\
 & & & & X \\
 & & & & \downarrow \\
 & & & & d(\mathcal{O}_{X_{/\mathbb{A}}}, \mathcal{G})
 \end{array}$$

is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type f_{*} . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . \square

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??.
A reduced above we conclude that U is an open covering of \mathcal{C} . The functor \mathcal{F} is a “field”

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\bar{x}} \dashrightarrow (\mathcal{O}_{X_{\text{étale}}}) \longrightarrow \mathcal{O}_{X_{\ell}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_n}^{\overline{\nu}})$$

is an isomorphism of covering of \mathcal{O}_{X_i} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

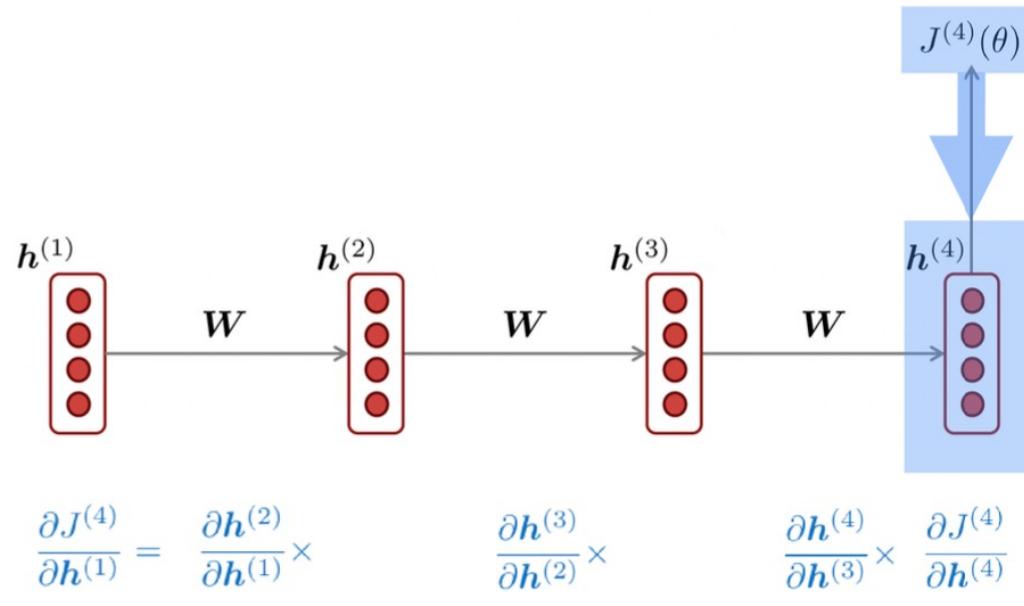
The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S .
If \mathcal{F} is a scheme theoretic image points. \square

If \mathcal{F} is a finite direct sum $\mathcal{O}_{X_{\lambda}}$ is a closed immersion, see Lemma ??.. This is a sequence of \mathcal{F} is a similar morphism.

Исчезающие и взрывающиеся градиенты

Многократное умножение на одну и ту же матрицу весов W

Норма градиента растет или убывает экспоненциально



Ограничение градиентов

- Exploding gradients => нестабильное обучение
- Gradient clipping - решение проблемы «взрывающихся» градиентов

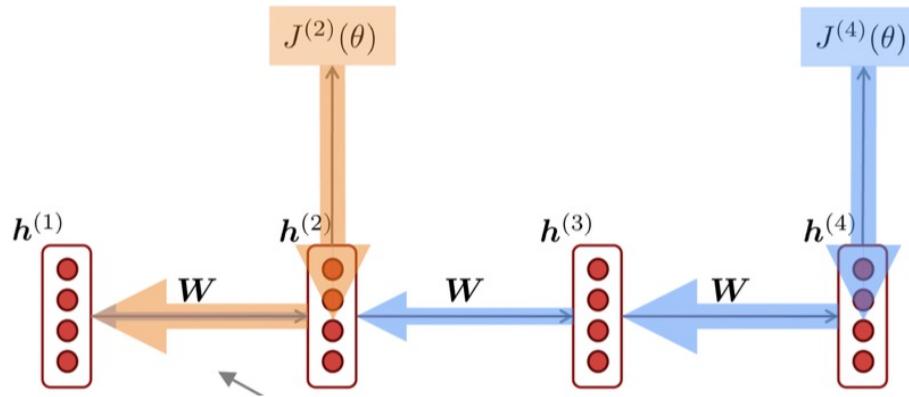
Просто обрезаем градиенты, ограничиваем их сверху,
например

$$\mathbf{g} = \nabla_{\theta} L(\theta)$$

Если $\|\mathbf{g}\| > threshold$, то

$$\mathbf{g} \leftarrow \frac{threshold}{\|\mathbf{g}\|} \mathbf{g}$$

Исчезающие градиенты



Обычная RNN могла бы описывать долгосрочные зависимости, на практике не получается ее обучить

- ▶ Сигнал от далеких шагов теряется, так как он гораздо меньше, чем сигнал от близких шагов
- ▶ Модель учитывает только краткосрочные эффекты, не долгосрочные

Рекуррентная нейронная сеть

Преимущества:

- ▶ Сохраняет контекст последовательности
- ▶ Хорошая производительность и гибкость

Недостатки:

- ▶ Двигается в последовательности в одном направлении (некоторые последовательности должны обрабатываться в обоих направлениях)
- ▶ Короткая память из-за проблем с исчезающими градиентами

Bi-RNN

Вычисляем у векторы:

$$\text{biRNN}^*(\mathbf{x}_{1:n}) = \mathbf{y}_{i:n} = \text{biRNN}(\mathbf{x}_{1:n}, 1), \dots, \text{biRNN}(\mathbf{x}_{1:n}, n).$$

Используется для
разметки текста и для
выделения признаков.

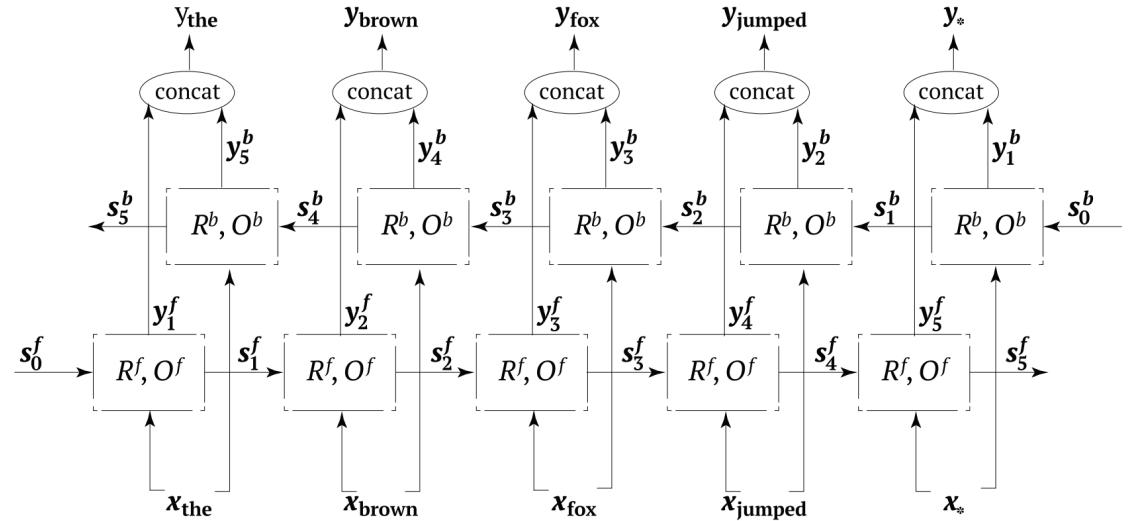


Рис. 14.6 ♦ Вычисление biRNN* для предложения «the brown fox jumped»

Goldberg, Yoav. Neural network methods for natural language processing

Рекуррентная нейронная сеть

Преимущества:

- ▶ Сохраняет контекст последовательности
- ▶ Хорошая производительность и гибкость

Недостатки:

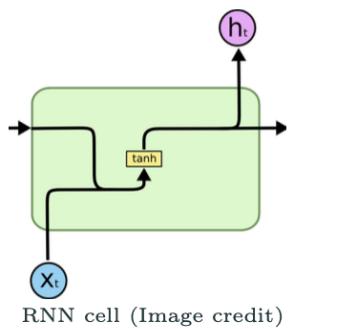
- ▶ ~~Двигается в последовательности в одном направлении (некоторые последовательности должны обрабатываться в обоих направлениях)~~
- ▶ Короткая память из-за проблем с исчезающими градиентами

LSTM/GRU

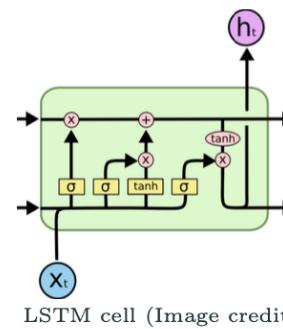


Архитектура долгой краткосрочной памяти

LSTM (Long Short-Term Memory) - решение проблемы «затухающих» градиентов:



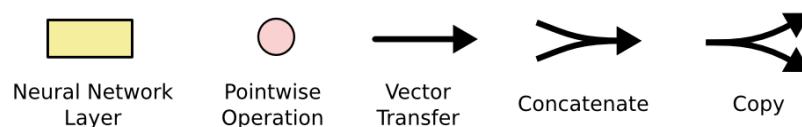
RNN cell (Image credit)



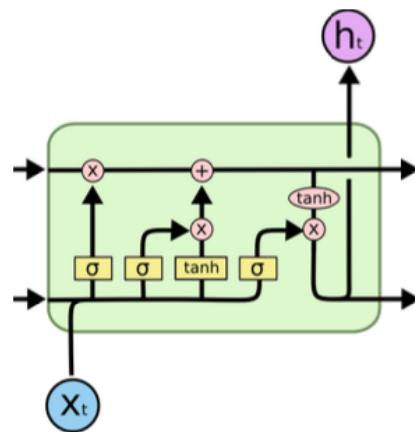
LSTM cell (Image credit)

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

- ▶ Добавляются ячейки памяти (cell state), в которую модель может записывать долгосрочную информацию
- ▶ Какую информацию считывать / записывать / удалять определяют гейты (gates)
- ▶ Гейты принимают значения от 0 до 1 и вычисляются динамически на основе контекста



LSTM



LSTM cell (Image credit)

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

forget gate

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

input gate

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

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

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

candidate cell state

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

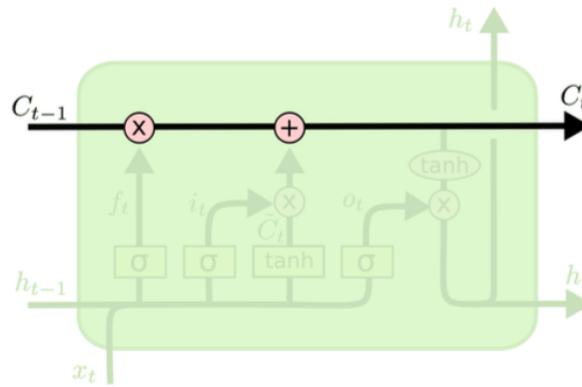
cell state

$$h_t = o_t \odot \tanh(c_t)$$

hidden state

LSTM

Почему это работает?



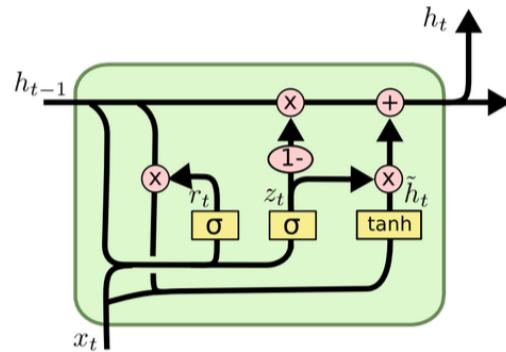
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

- Суть - чтобы градиент беспрепятственно протекал по сети (как обычно :)
- Если $f_t \approx 1$, то $c_t = c_{t-1} + i_t \odot \tilde{c}_t'$

Вентильный рекуррентный блок

GRU (Gated Recurrent Unit) - упрощенная версия



<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

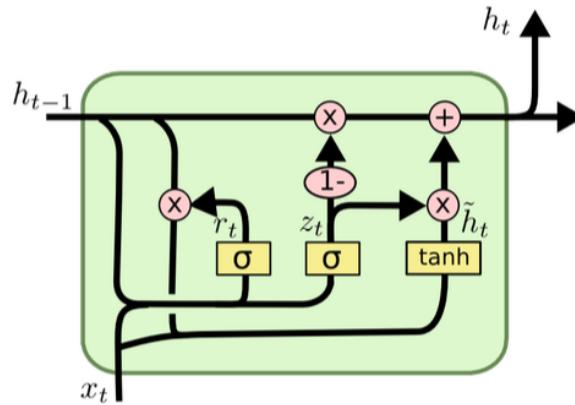
$$u_t = \sigma(W_u x_t + U_u h_{t-1} + b_u) \quad \text{update gate}$$

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

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

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

GRU



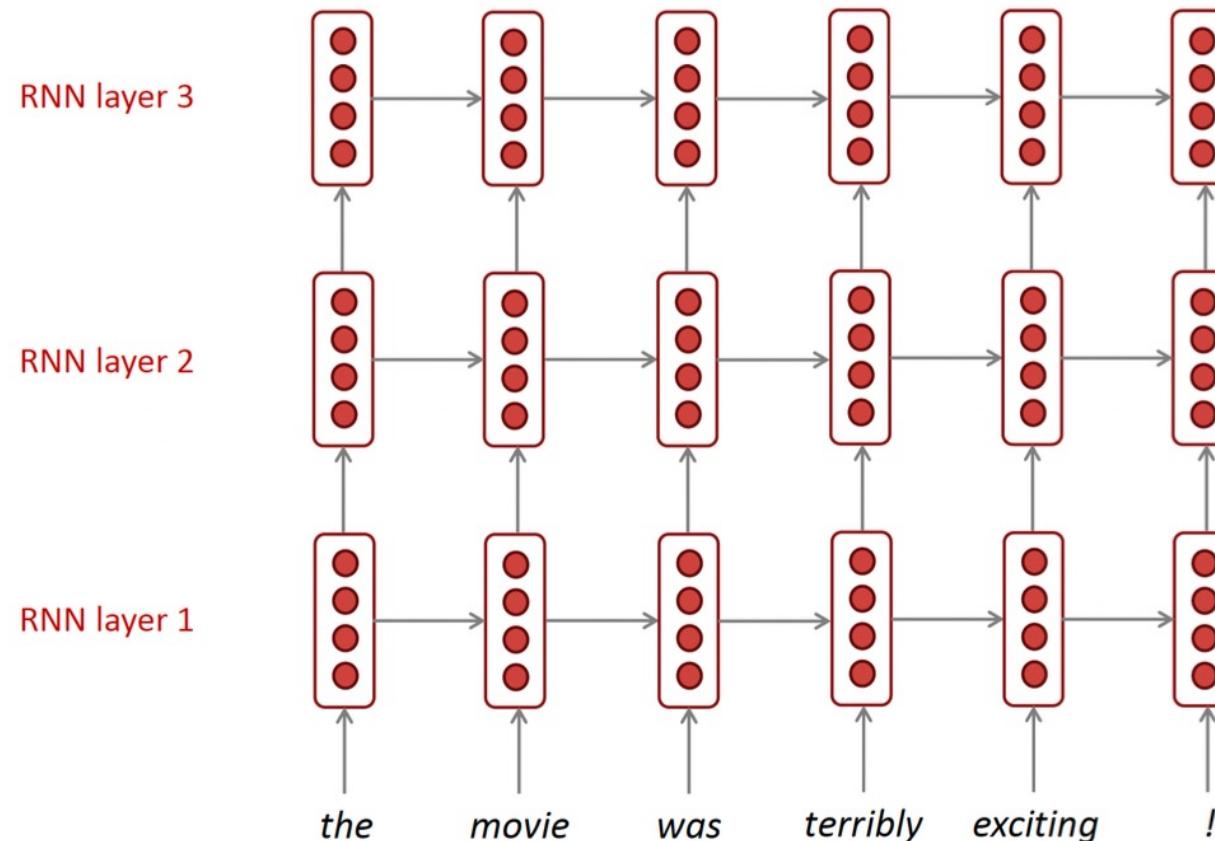
<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

- Тоже есть прямой путь для градиентов
- Меньше матриц, меньше весов
- Быстрее обучается
- Обычно не хуже или почти не хуже, чем LSTM

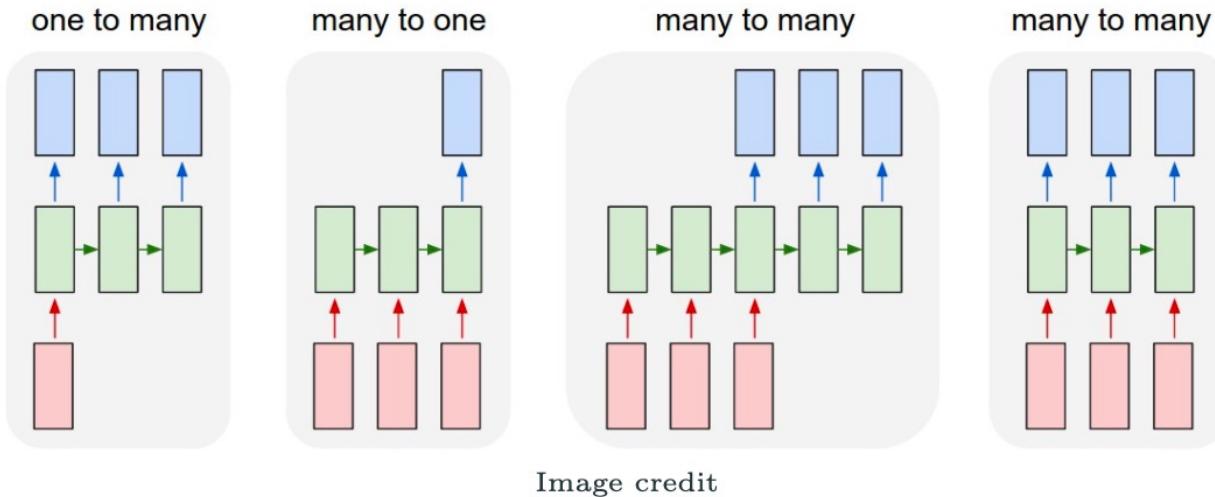
Некоторые применения



Multilayer (stacked) RNN



Приименение RNN



Примеры:

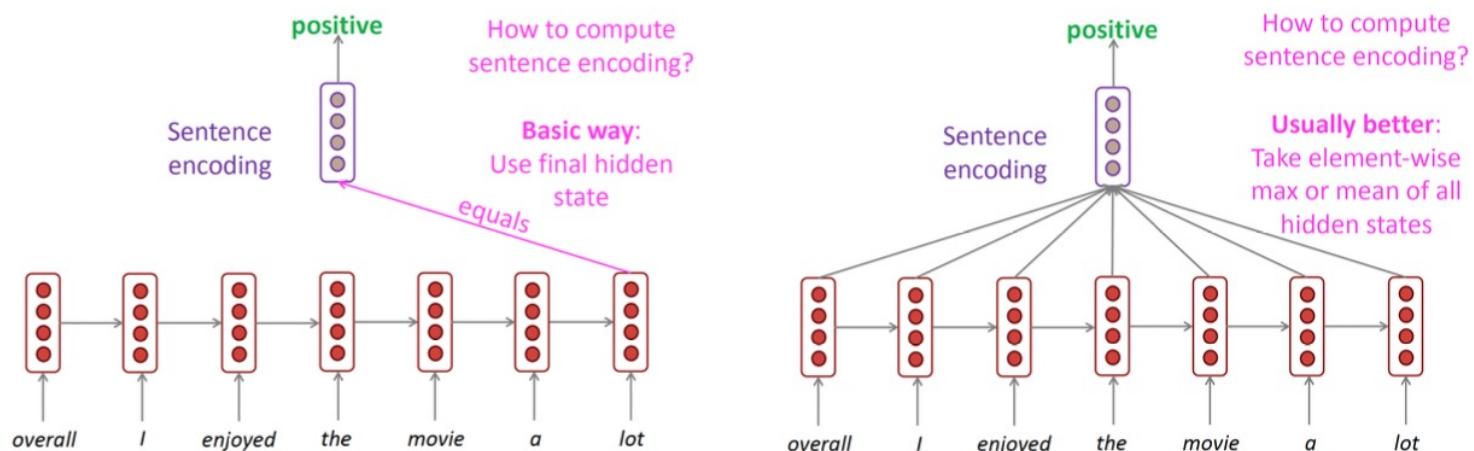
- one to many - описание изображения (image captioning)
- many to one - классификация текста
- many to many - машинный перевод
- synced many to many - LM, NER

Классификация всей последовательности

Нужен один выход на всю последовательность

Два варианта:

- Используем hidden state с последнего шага
- Агрегируем hidden states со всех шагов



Минибатчи для RNN

В батче могут быть последовательности разной длины. Что делать?

- Выравнивать по длине (дополнять зарезервированным под это символом) - padding
- Обычно также обрезают до фиксированной максимальной длины - truncation

[1, 5, 3]

[2, 4, 3, 6, 1]

[5, 2]

[1, 5, 3, 0, 0]

[2, 4, 3, 6, 1]

[5, 2, 0, 0, 0]

- Лучше брать в батч последовательности примерно равной длины

Take home

Преимущества:

- ▶ Универсальная архитектура для последовательностей
- ▶ Может обрабатывать последовательности любой длины
- ▶ Размер модели не увеличивается с увеличением длины

Но

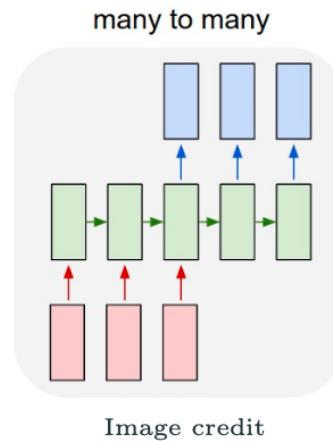
- ▶ Медленные вычисления, так как надо считать все
- ▶ последовательно и нельзя распараллелить
- ▶ LSTM/GRU также могут иметь проблемы с учетом долговременных зависимостей контекст последовательности

Sequence-to-sequence модели



Seq2seq

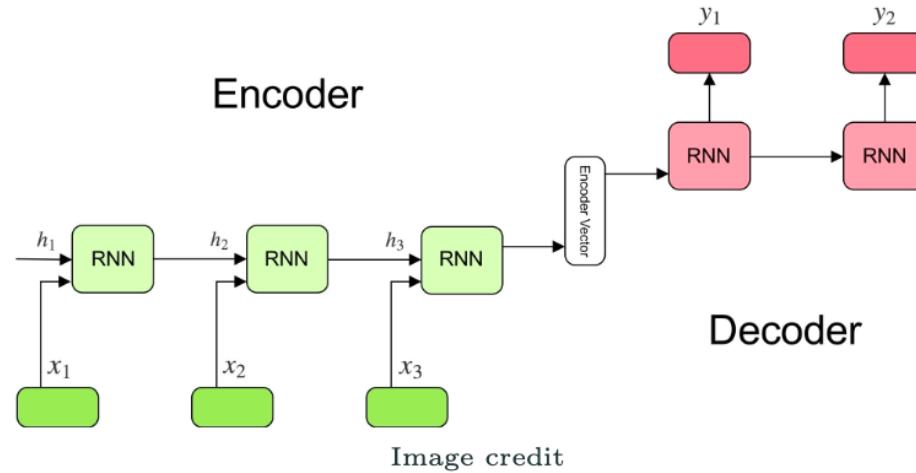
Преобразование последовательности произвольной длины на входе в последовательность произвольной длины на выходе



Например:

- machine translation
- summarization
- dialogue

Encoder-Decoder архитектура

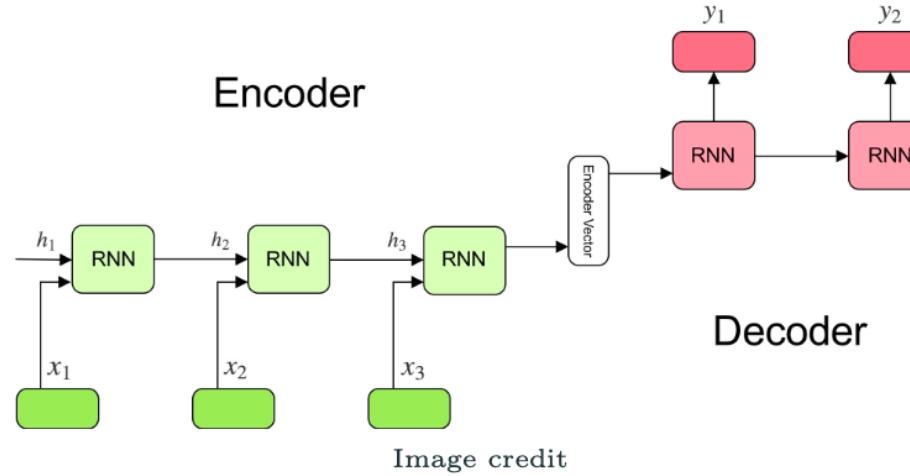


Encoder сворачивает входную последовательность в вектор контекста c (как правило, последнее скрытое состояние h_T):

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

$$c = h_T$$

Encoder-Decoder архитектура

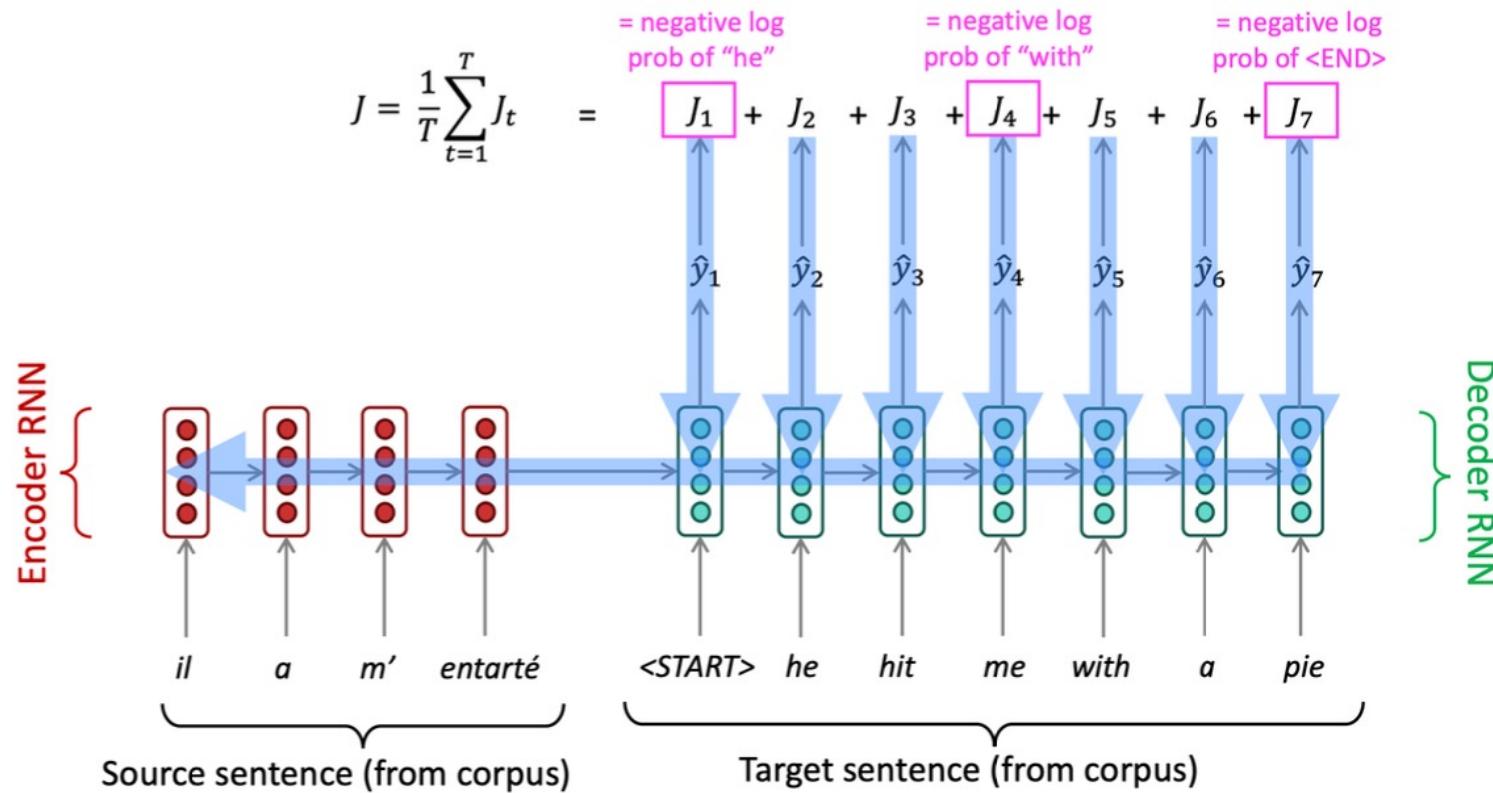


Decoder предсказывает следующий элемент на основе скрытого состояния s_t , предыдущего элемента y_{t-1} и вектора контекста c (condititonal generation):

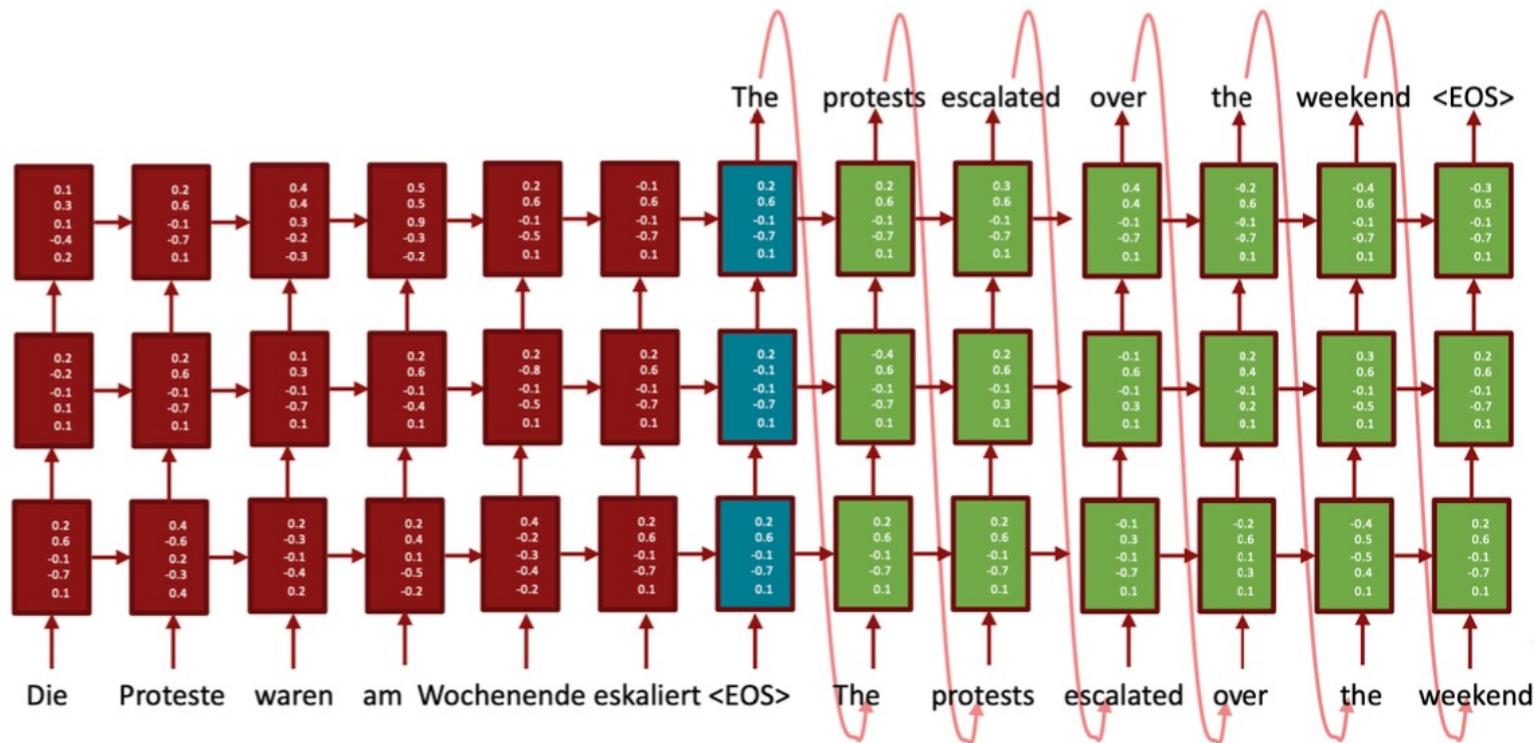
$$p(y_t|y_1, \dots, y_{t-1}, c) = f(y_{t-1}, s_t, c)$$

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

Тренировка

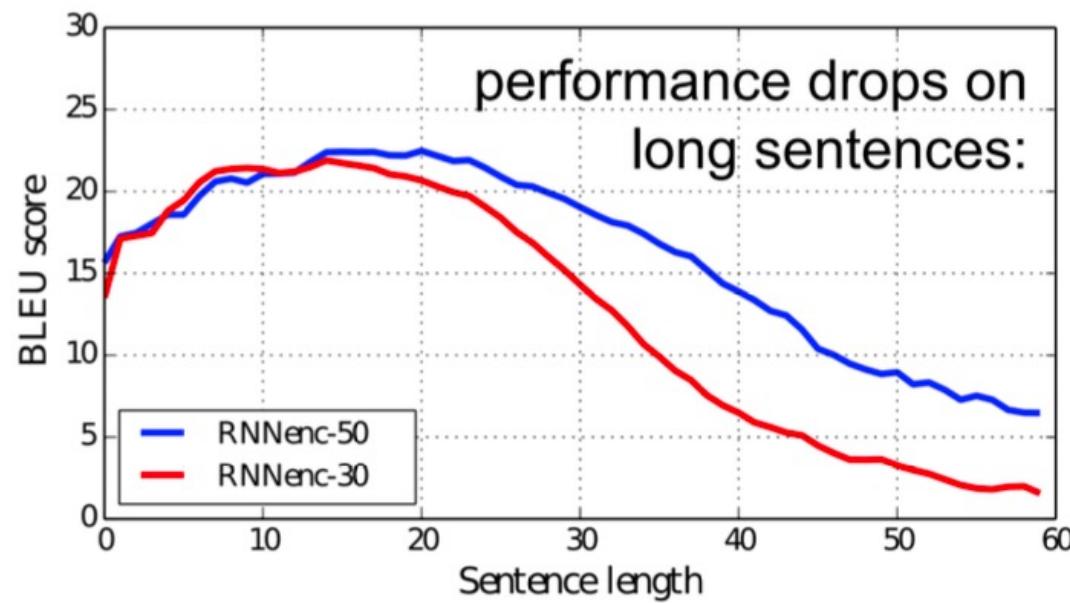


Практические применения



Проблемы подхода

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



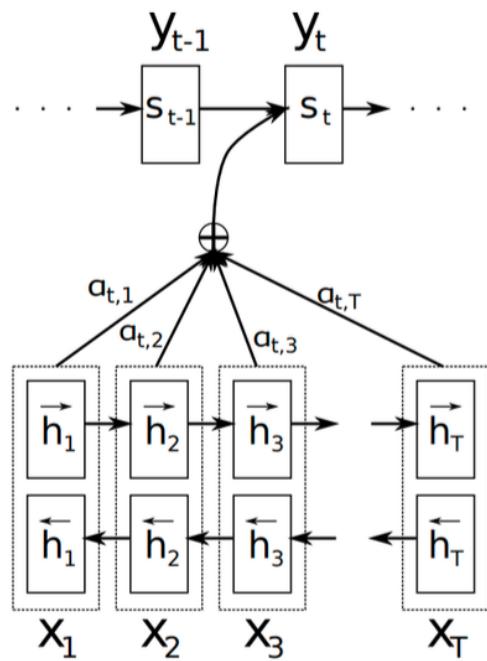
https://courses.grainger.illinois.edu/cs546/sp2018/Slides/Mar15_Bahdanau.pdf

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

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

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

Веса

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

Типы внимания

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

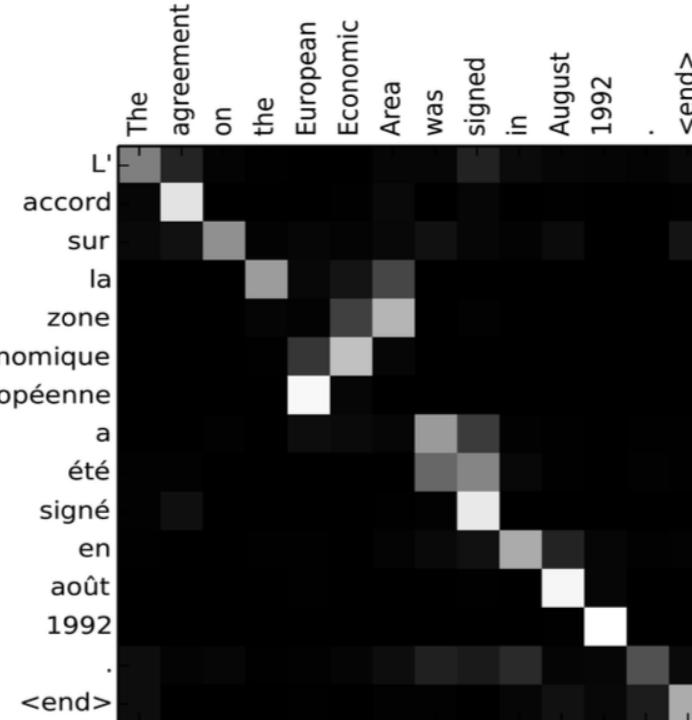
$$\text{score}(s_{t-1}, h_i) = s_{t-1}^T h_i$$

- Scaled dot-product

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

Интерпретация внимания

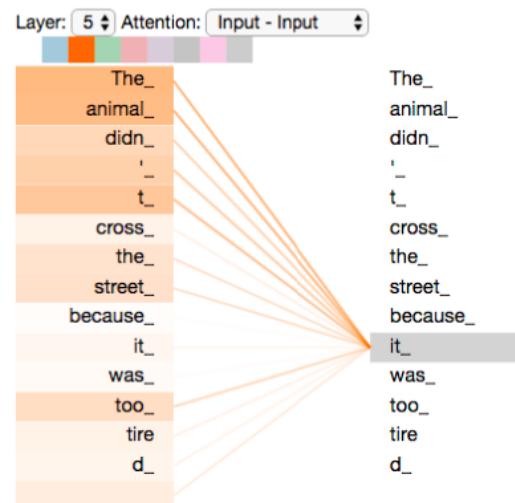
Матрица весов α_{ti} определяет alignment (выравнивание) элементов входной и выходной последовательностей



<https://arxiv.org/pdf/1409.0473>

Self-attention

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



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

Матричный вид Self-attention

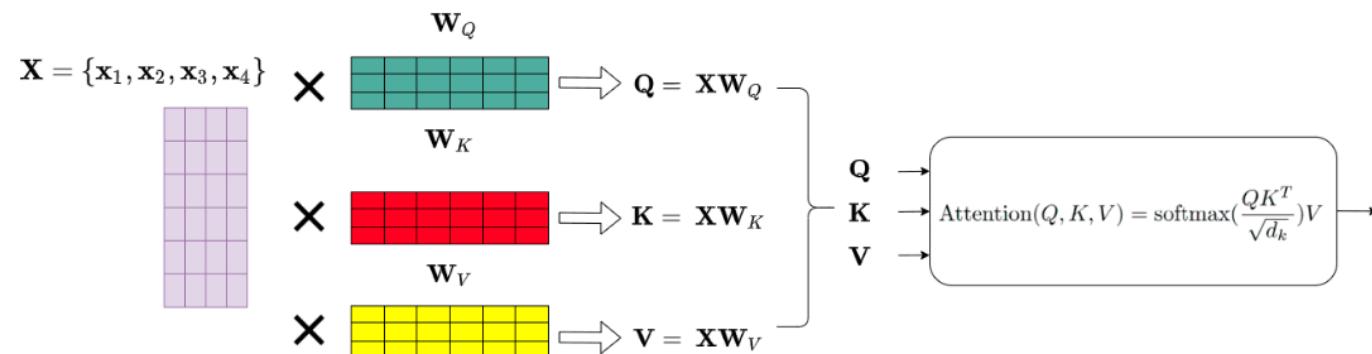
В матричном виде

$$Q = XW^Q$$

$$K = XW^K$$

$$V = XW^V$$

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

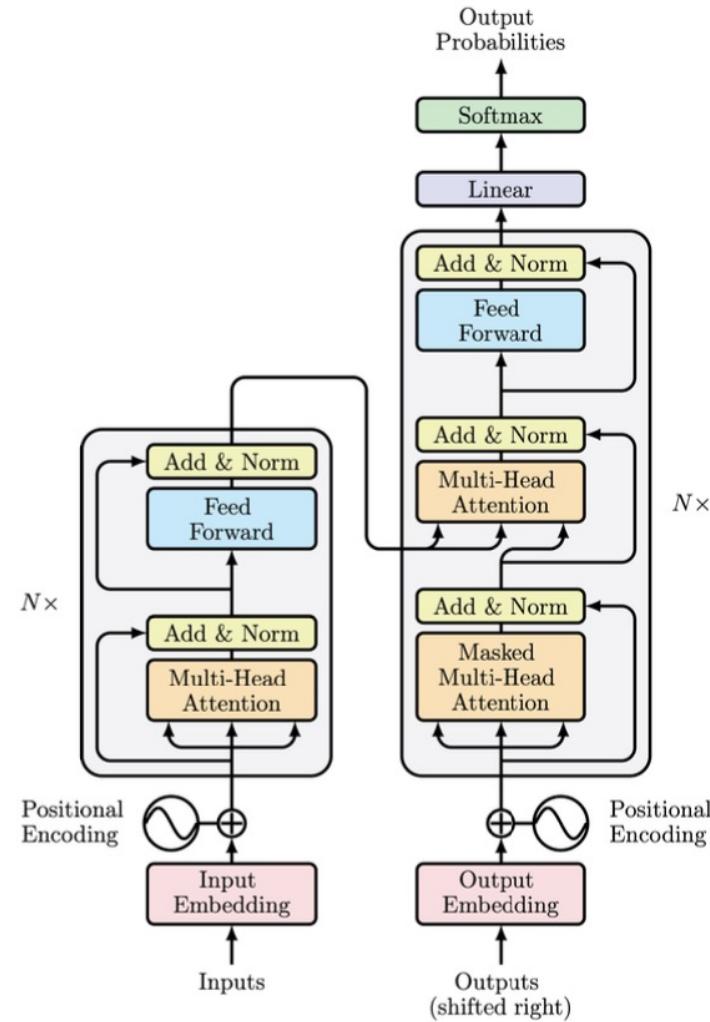


Трансформеры



Архитектура

- 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



Позиционное кодирование

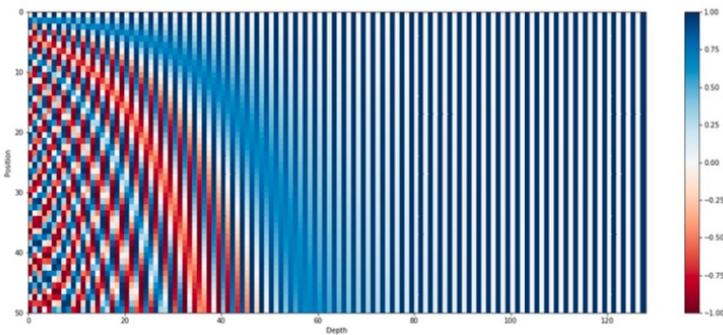
- Self-attention слой не знает о порядке элементов в последовательности!
- Даём эту информацию модели с помощью Positional encoding
- Прибавляем к эмбеддингам токенов позиционные эмбеддинги



Позиционное кодирование

Fixed positional encoding

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

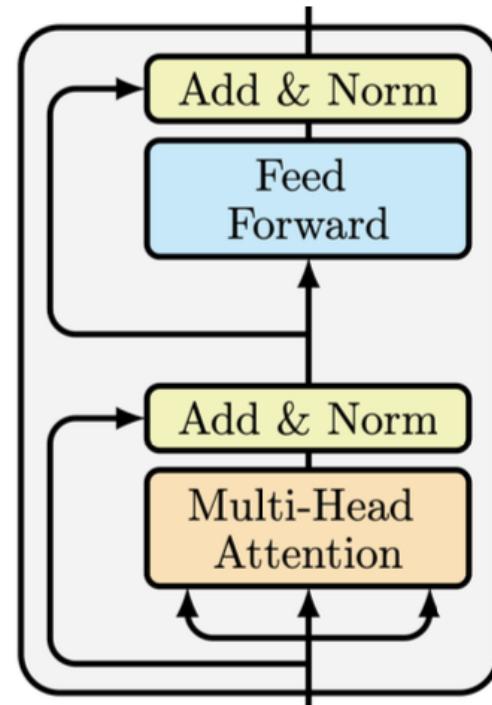


Learned positional encoding

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

Блок трансформера

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



Multi-head self-attention

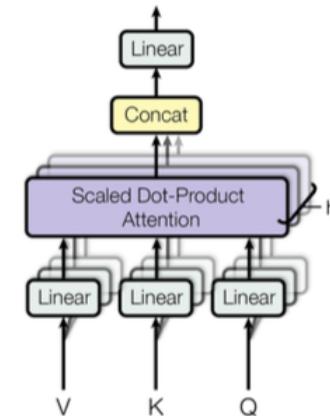
- Несколько независимых «голов» внимания, которые потом объединяются
- Чтобы не увеличивать количество вычислений, размерность каждой из h «голов» в h раз меньше размерности входа

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

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

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

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

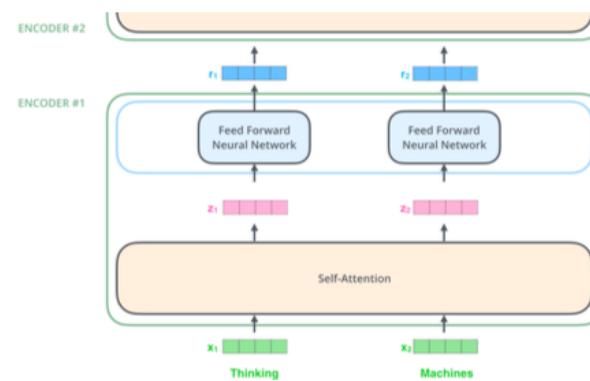


Pointwise feed-forward network

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

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

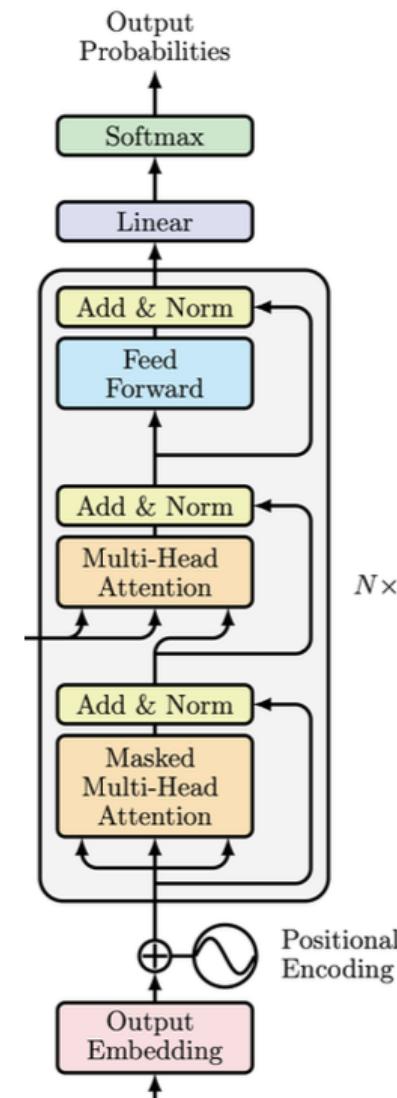
Один и те же веса применяются к каждому элементу последовательности независимо



Декодер

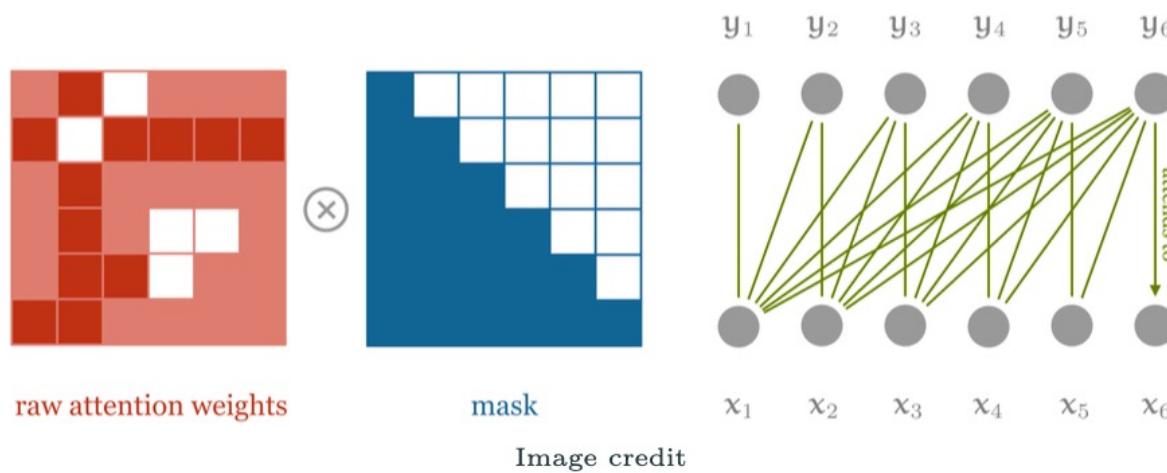
То же, что и в Encoder, плюс:

- 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



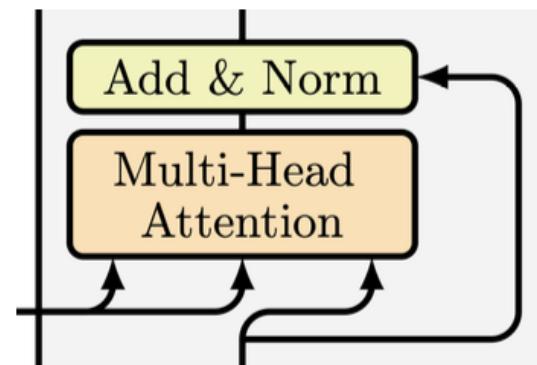
$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d}}\right)V$$

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

Cross-attention

Encoder - Decoder Attention

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



Summary

Pros

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

Cons

- Квадратичная зависимость self-attention от длины последовательности

