RNN, ВНИМАНИЕ, ТРАНСФОРМЕР

Сергей Николенко

ИТМО — Санкт-Петербург

19 октября 2026 г.





Random facts:

- 19 октября 202 г. до н.э. Сципион Африканский разбил Ганнибала при Заме, а 19 октября 439 г. Карфаген у римлян отобрали вандалы Гейзериха
- 19 октября 1453 г. капитуляция англичан в Бордо положила конец Столетней войне, а 19 октября 1812 г. Наполеон покинул Москву
- 19 октября 1781 г. в Йорктауне капитулировал Корнуоллис, и это фактически решило исход Войны за независимость США
- 19 октября 1900 г. Макс Планк вывел свою формулу для излучения абсолютно чёрного тела
- 19 октября 1987 г., в «чёрный понедельник», индекс Доу-Джонса упал за день на 22.6%; это рекорд до сих пор
- 19 октября 2002 г. Владимир Крамник свёл вничью восьмую, решающую партию матча против Deep Fritz, и весь матч закончился вничью

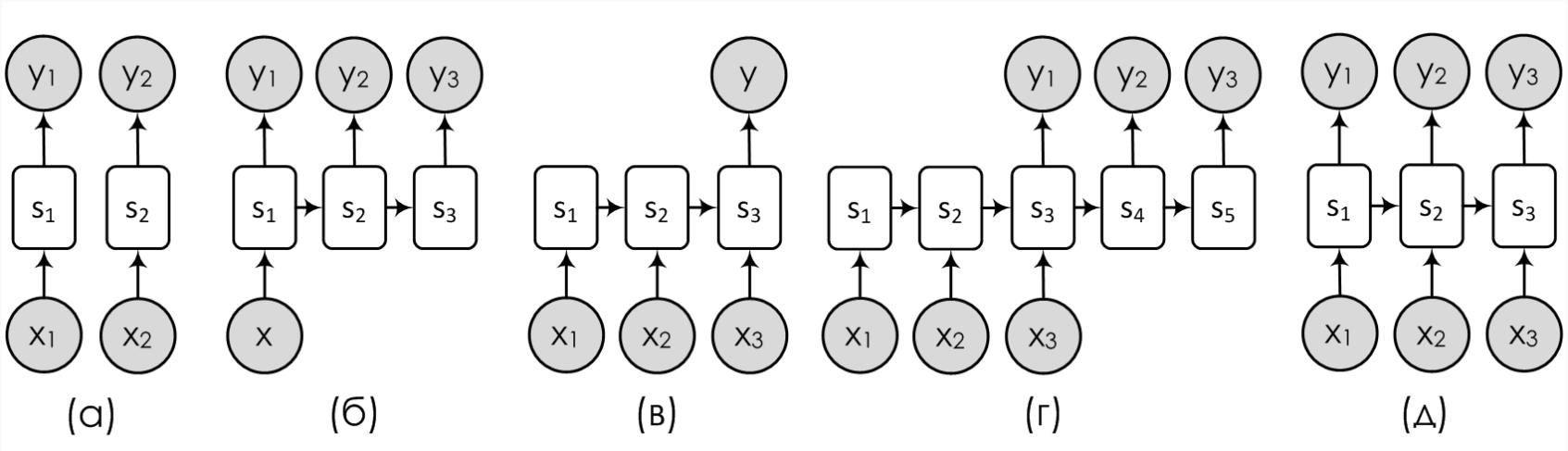
Краткое резюме предыдущей серии

- В прошлый раз мы закончили с тем, как обучать сети (инициализация, нормализация по мини-батчам), и прошли свёрточные сети от неокогнитрона и LeNet до ResNet и EfficientNet.
- Свёртка — операция *локальная*: чтобы связать далёкие части входа, приходится складывать много слоёв.
- Сегодня — последовательности: рекуррентные сети и их беды с градиентами, LSTM и GRU, потом *внимание* — сначала поверх рекуррентных сетей, а потом вместо них: *трансформер*.

Рекуррентные нейронные сети

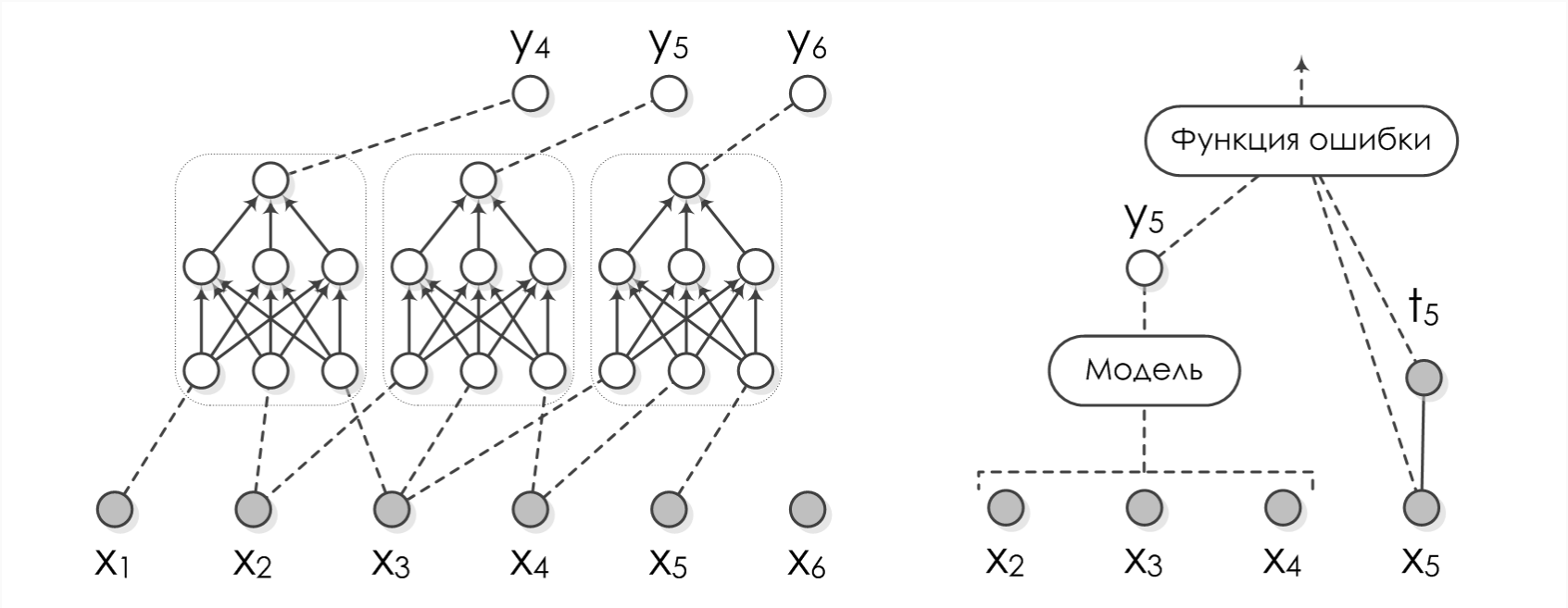
Последовательности

- Последовательности: текст, временные ряды, речь, музыка...
- Есть разные виды задач, основанных на последовательностях:



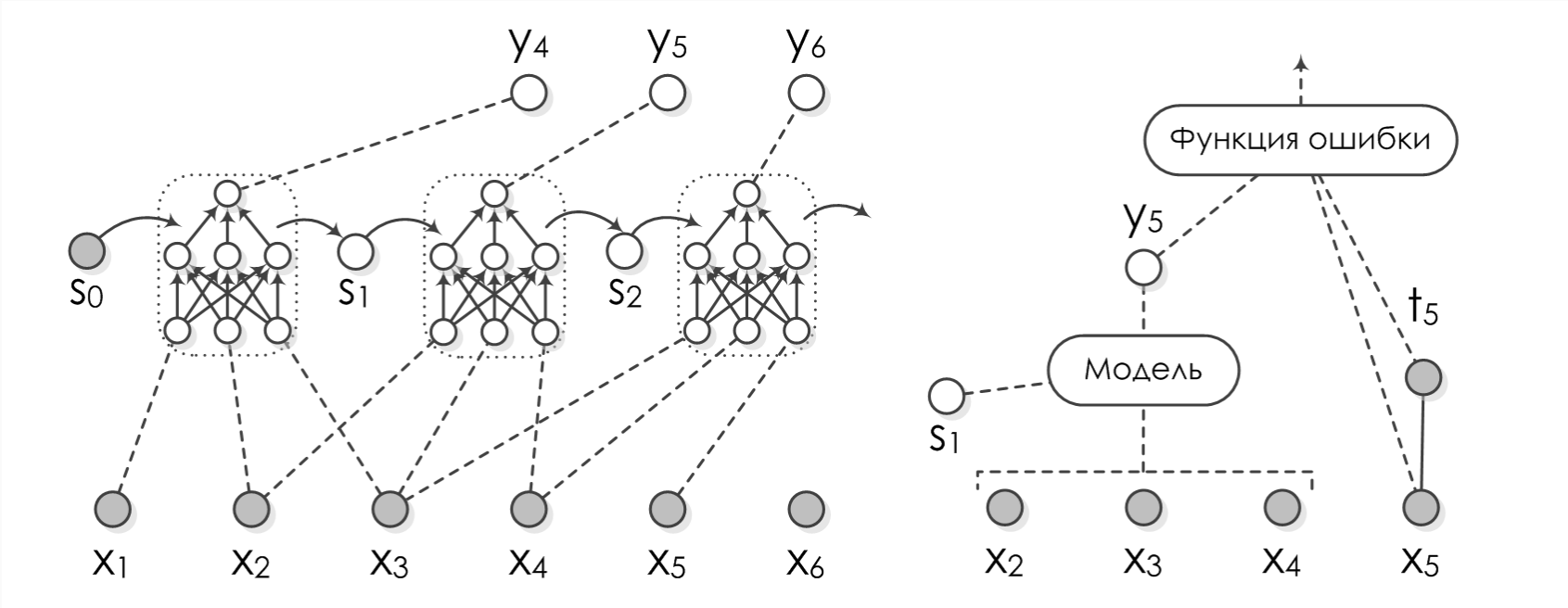
Последовательности

- Как применить к последовательности нейронную сеть?
- Можно использовать скользящее окно:



Последовательности

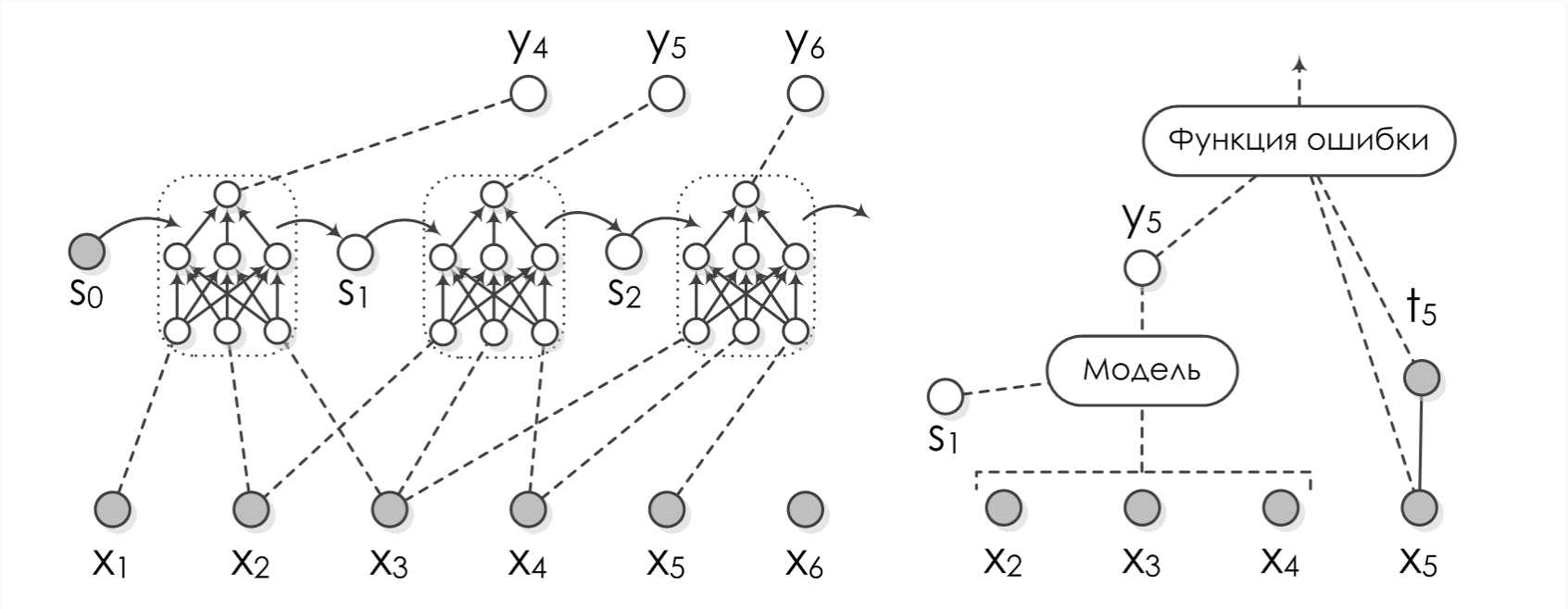
- ...но ещё лучше будет сохранять какое-нибудь скрытое состояние и обновлять его каждый раз.
- Это в точности идея *рекуррентных нейронных сетей* (recurrent neural networks, RNN).



Последовательности

- Но как теперь делать backpropagation? Получается, что в графе вычислений теперь циклы:

$$s_i = h(x_i, x_{i+1}, x_{i+2}, s_{i-1}).$$



- Это же ужасно, и всё сломалось?..

Последовательности

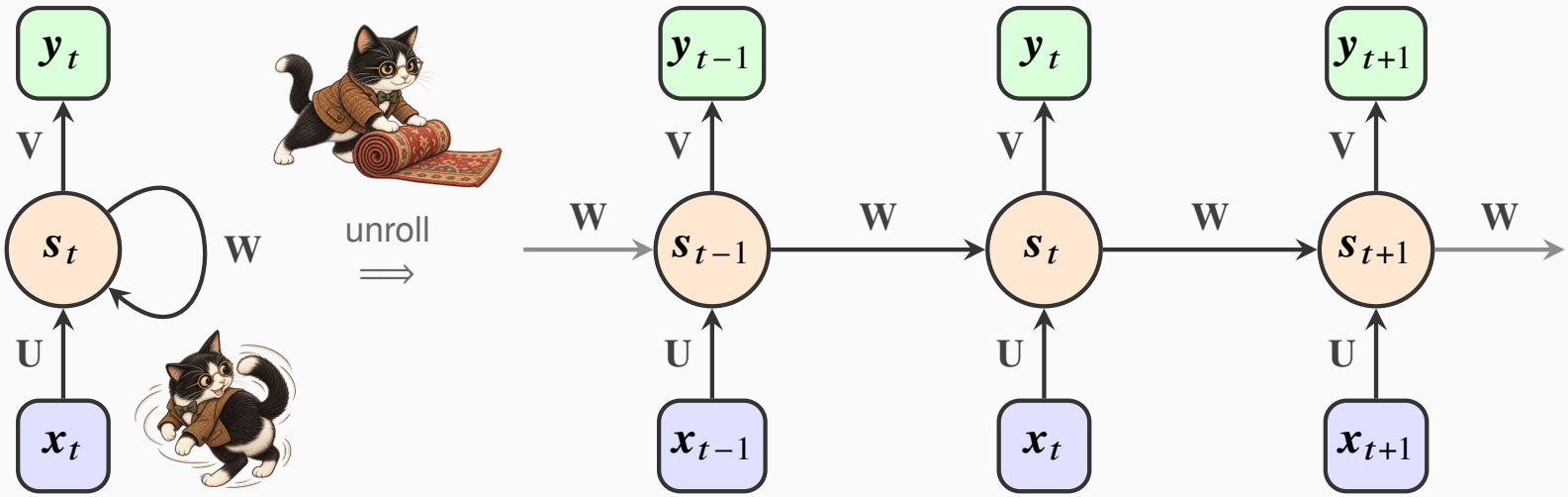
- ...да нет, конечно. Можно “развернуть” циклы обратно:

$$y_6 = f(x_3, x_4, x_5, s_2) = f(x_3, x_4, x_5, h(x_2, x_3, x_4, s_1)) = \\ = f(x_3, x_4, x_5, h(x_2, x_3, x_4, h(x_1, x_2, x_3, s_0))).$$

- Так что формально проблемы нет.
- Но масса проблем в реальности: получается, что рекуррентная сеть – это такая *очень* глубокая сеть с кучей общих весов...

Простая RNN

- “Простая” RNN, свёрнутая и развёрнутая во времени; веса W, U, V общие для всех шагов:



- Обучение — backpropagation through time (BPTT): развернули, и получилась очень глубокая сеть с общими весами; градиент по общей матрице — сумма градиентов по всем её копиям.
- Для длинных последовательностей — truncated BPTT: состояние несём через всю последовательность, а градиент пропускаем только через последние несколько десятков шагов.

Простая RNN

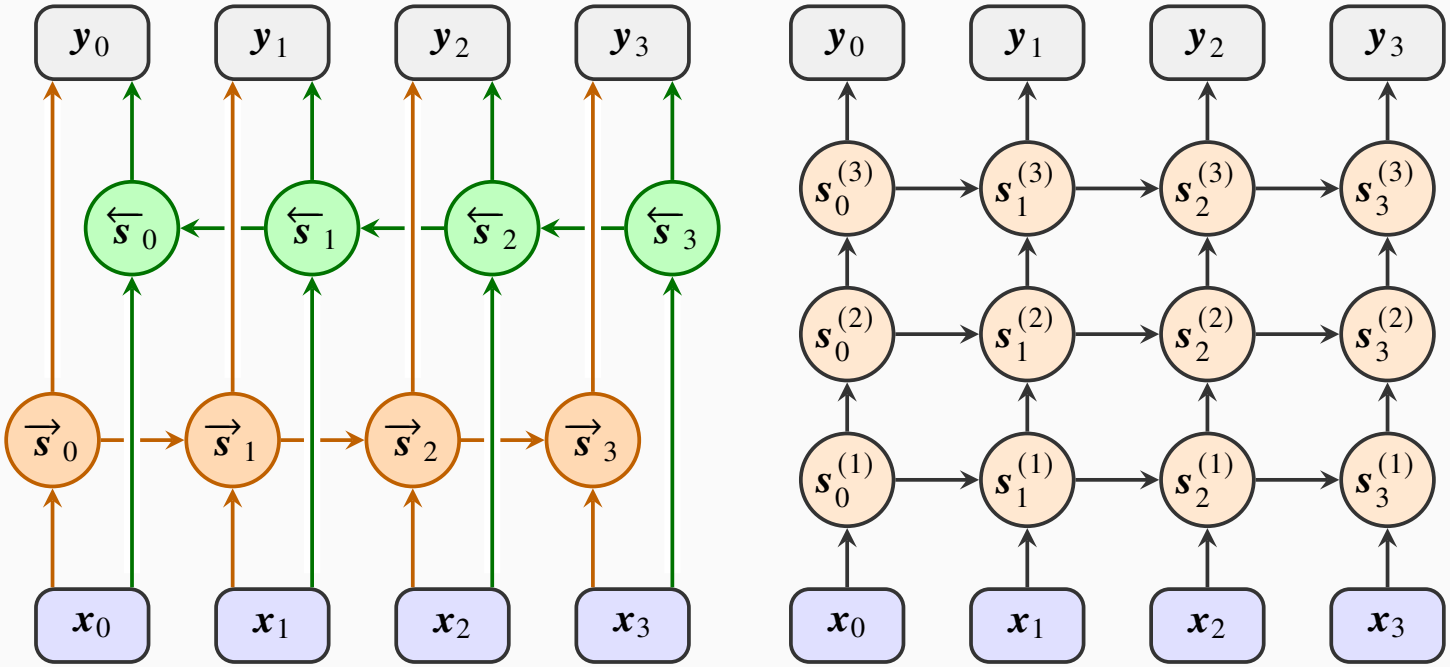
- Формально:

$$\begin{aligned}
\mathbf{a}_t &= \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t, \\
\mathbf{s}_t &= f(\mathbf{a}_t), \\
\mathbf{o}_t &= \mathbf{c} + V\mathbf{s}_t, \\
\mathbf{y}_t &= h(\mathbf{o}_t),
\end{aligned}$$

где f – рекуррентная нелинейность, h – функция выхода.

Двунаправленная RNN

- Иногда нужен контекст с обеих сторон — двунаправленная RNN (слева); а ещё рекуррентные слои можно складывать в стопку — глубокая RNN (справа):



Двунаправленная RNN

- Формально:

$$\begin{aligned}
\mathbf{s}_t &= \sigma\left(\mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t\right), \\
\mathbf{s}'_t &= \sigma\left(\mathbf{b}' + W'\mathbf{s}'_{t+1} + U'\mathbf{x}_t\right), \\
\mathbf{o}_t &= \mathbf{c} + V\mathbf{s}_t + V'\mathbf{s}'_t, \\
\mathbf{y}_t &= h\left(\mathbf{o}_t\right).
\end{aligned}$$

- И это, конечно, обобщается на любой другой тип конструкций.

Взрывающиеся и затухающие градиенты

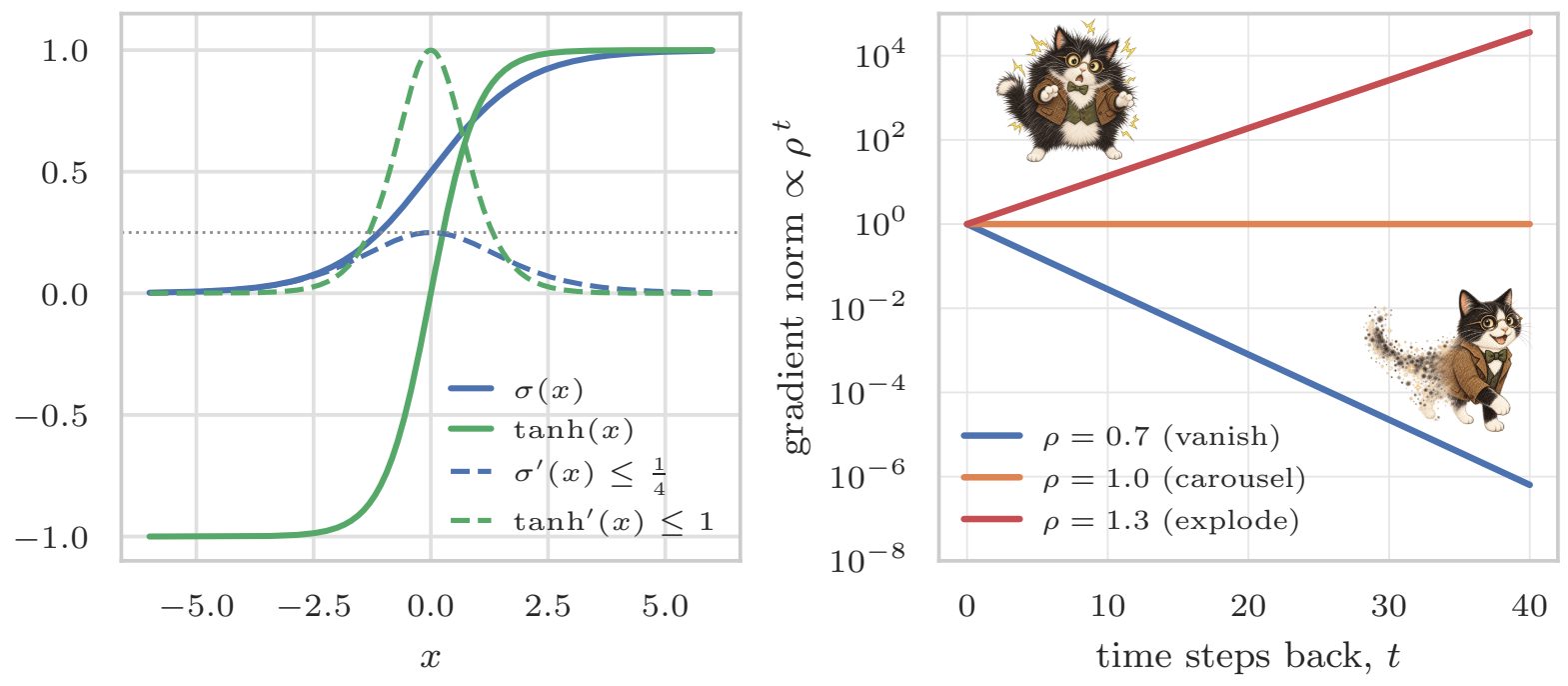
- Две проблемы:
  - взрывающиеся градиенты (exploding gradients);
  - затухающие градиенты (vanishing gradients).
- Градиент по раннему состоянию — длинное произведение якобианов:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_T} \prod_{k=T-1}^{t} \frac{\partial \mathbf{s}_{k+1}}{\partial \mathbf{s}_k}, \qquad \frac{\partial \mathbf{s}_{k+1}}{\partial \mathbf{s}_k} = \operatorname{diag}\left(f'(\mathbf{a}_{k+1})\right) W.$$

- Каждый раз умножаем на одну и ту же W, и норма градиента может экспоненциально расти или убывать.

Взрывающиеся и затухающие градиенты

- Сама нелинейность градиент не усиливает: $\sigma' \le \frac{1}{4}$, $\tanh' \le 1$ (слева).
- А в упрощённой модели с одним множителем ρ на шаг градиент ведёт себя как ρ^t (справа):



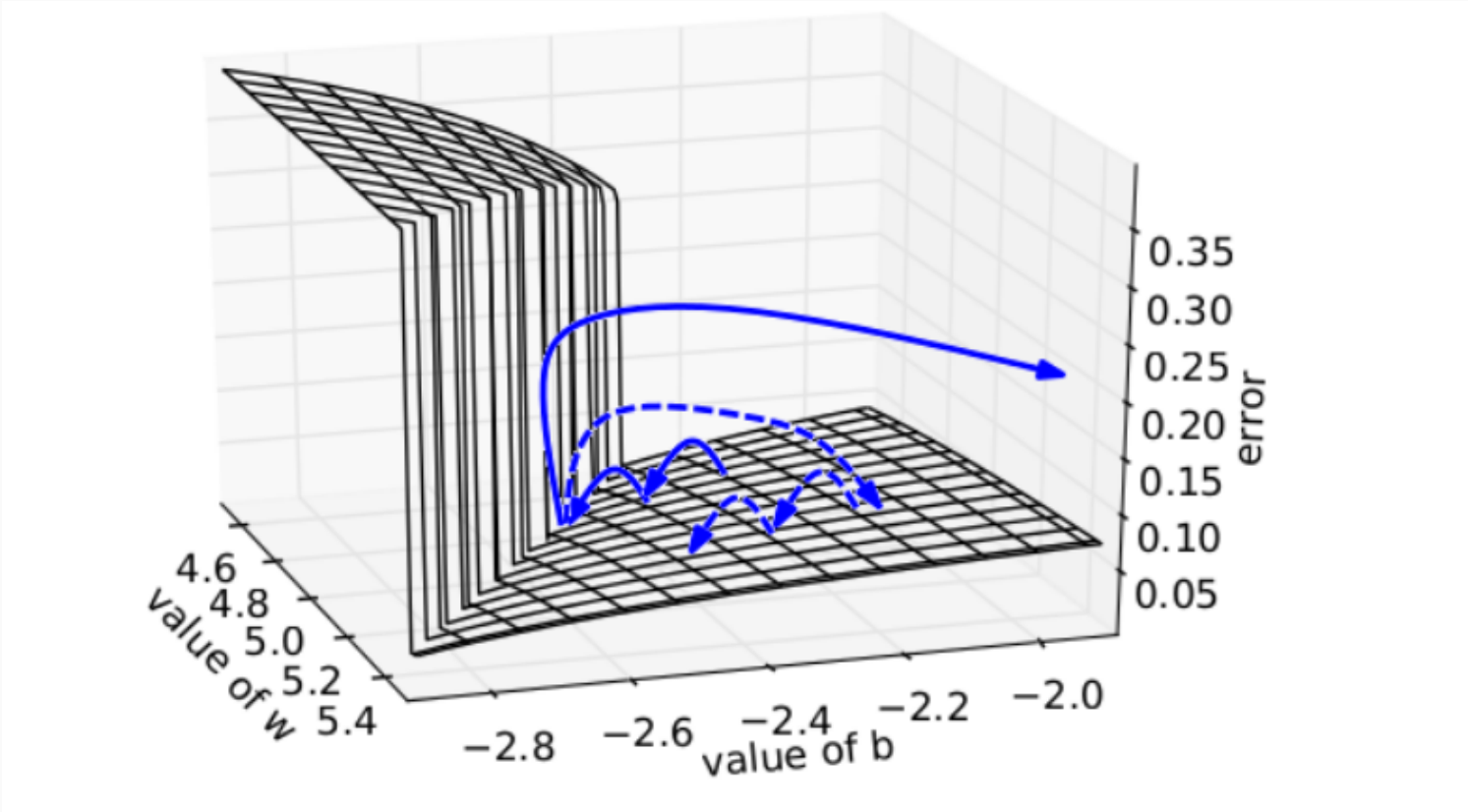
- Устойчиво только при $\rho = 1$; запомним это.

Взрывающиеся и затухающие градиенты

- Взрывающиеся градиенты неприятны, но лечатся просто: обрезать градиенты, ограничить сверху, чтобы не росли.
- Два варианта – ограничить общую норму или каждое значение:
  - `sgd = optimizers.SGD(lr=0.01, clipnorm=1.)`
  - `sgd = optimizers.SGD(lr=0.01, clipvalue=.05)`

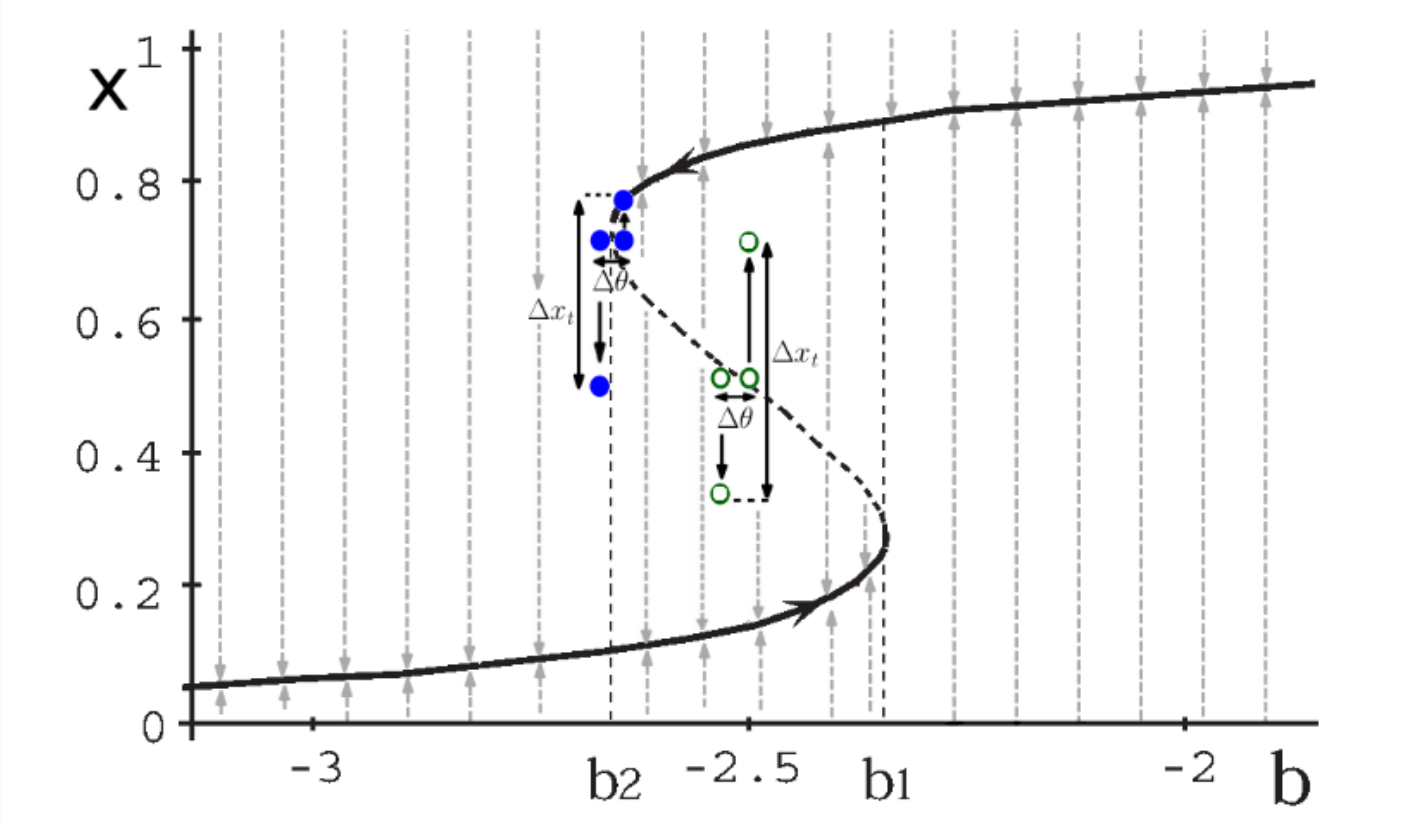
Взрывающиеся и затухающие градиенты

- (Pascanu et al., 2013) – вот что будет происходить:



Взрывающиеся и затухающие градиенты

- Там же объясняется, откуда возьмутся такие перепады: есть точки бифуркации у RNN.



Взрывающиеся и затухающие градиенты

- А вот затухающие градиенты гораздо хуже, потому что нет никакого сообщения об ошибке.
- Градиенты выглядят нормально, просто определяются последними несколькими элементами последовательности.
- Сеть не выучивает дальние зависимости: в тексте соседние слова на предсказание влияют, а связь через абзац выучивается слабо или никак.
- Память у сети есть, просто очень короткая.

Карусель константной ошибки: LSTM и GRU

LSTM

- Затухающие градиенты: надо каждый раз умножать на W.
- Поэтому не получается долгосрочную память реализовать.



- А хочется. Что делать?..

LSTM

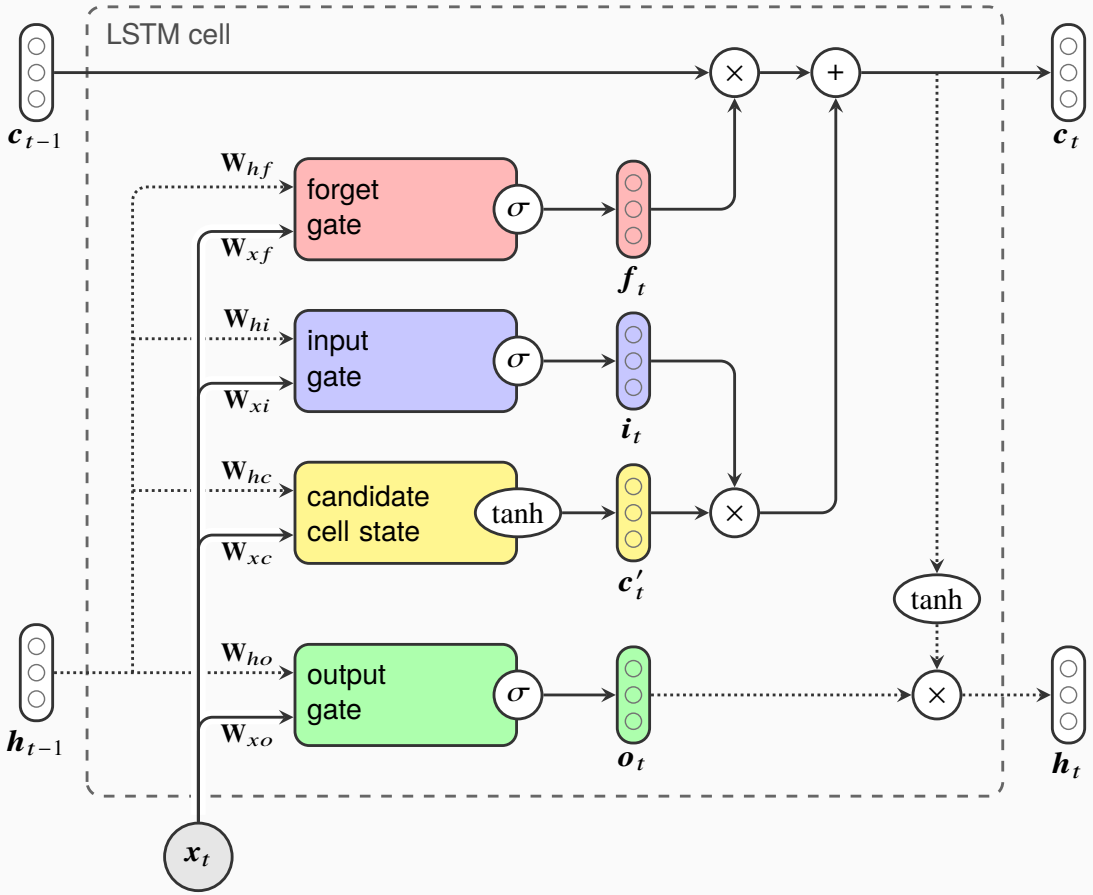
- Базовую идею мы уже видели в ResNet: надо сделать так, чтобы градиент проходил.
- В RNN это называется «карусель константной ошибки» (constant error carousel).



- Идея из середины 1990-х (Шмидхубер): давайте составлять RNN из более сложных частей, в которых будет прямой путь для градиентов, и память будет контролироваться явно.

LSTM

- LSTM (long short-term memory). “Ванильный” LSTM: $\mathbf{c}_t$ – состояние ячейки памяти, $\mathbf{h}_t$ – скрытое состояние.
- Input gate и forget gate определяют, надо ли менять $\mathbf{c}_t$ на нового кандидата в состояния ячейки.



LSTM

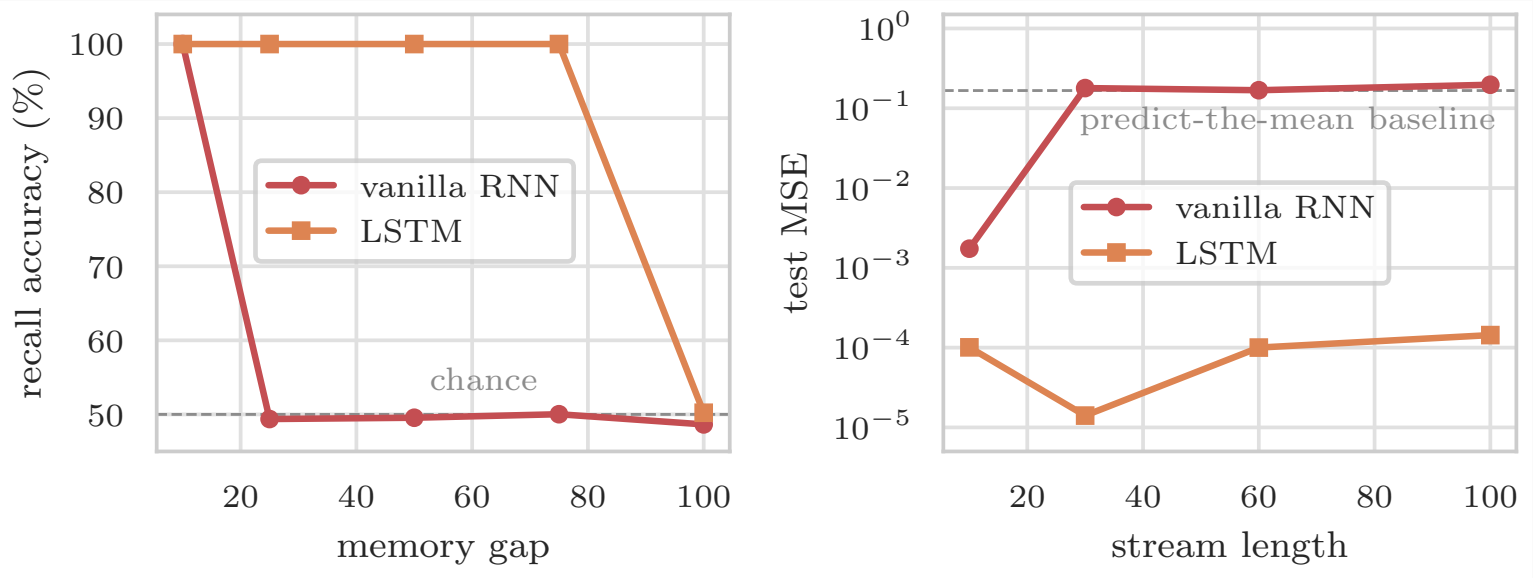
- Формально:

$$
\begin{aligned}
\mathbf{c}'_t &= \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_{c'}\right) && \textit{candidate cell state} \\
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i\right) && \textit{input gate} \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f\right) && \textit{forget gate} \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o\right) && \textit{output gate} \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t, && \textit{cell state} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \textit{block output}
\end{aligned}
$$

- Так что LSTM может контролировать состояние ячейки при помощи скрытого состояния и весов.
- Например, если $\mathbf{f}_t = 1$ (ничего не забываем), то получится карусель константной ошибки: $\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t$, и $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = 1$.
- Важно инициализировать $\mathbf{b}_f$ положительным (1–2), чтобы поначалу было $\mathbf{f}_t \approx 1$; иначе $\mathbf{f}_t \approx \frac{1}{2}$, и память затухает вдвое на каждом шаге.

LSTM

- Если $\mathbf{f}_t \equiv f$, то у памяти период полураспада $t_{1/2} = \frac{\log(1/2)}{\log f}$: при $f = 0.9$ это примерно 6.6 шага, при $f = 0.99 - 69$, при $f = 0.999 - 693$.
- Эксперимент: запомнить один бит из начала последовательности (слева) и сложить два помеченных числа из длинного потока, adding problem (справа):



- Простая RNN держит около 10 шагов, LSTM — около ста. Но только с положительным $\mathbf{b}_f$: с нулевым LSTM работала хуже простой RNN!

LSTM

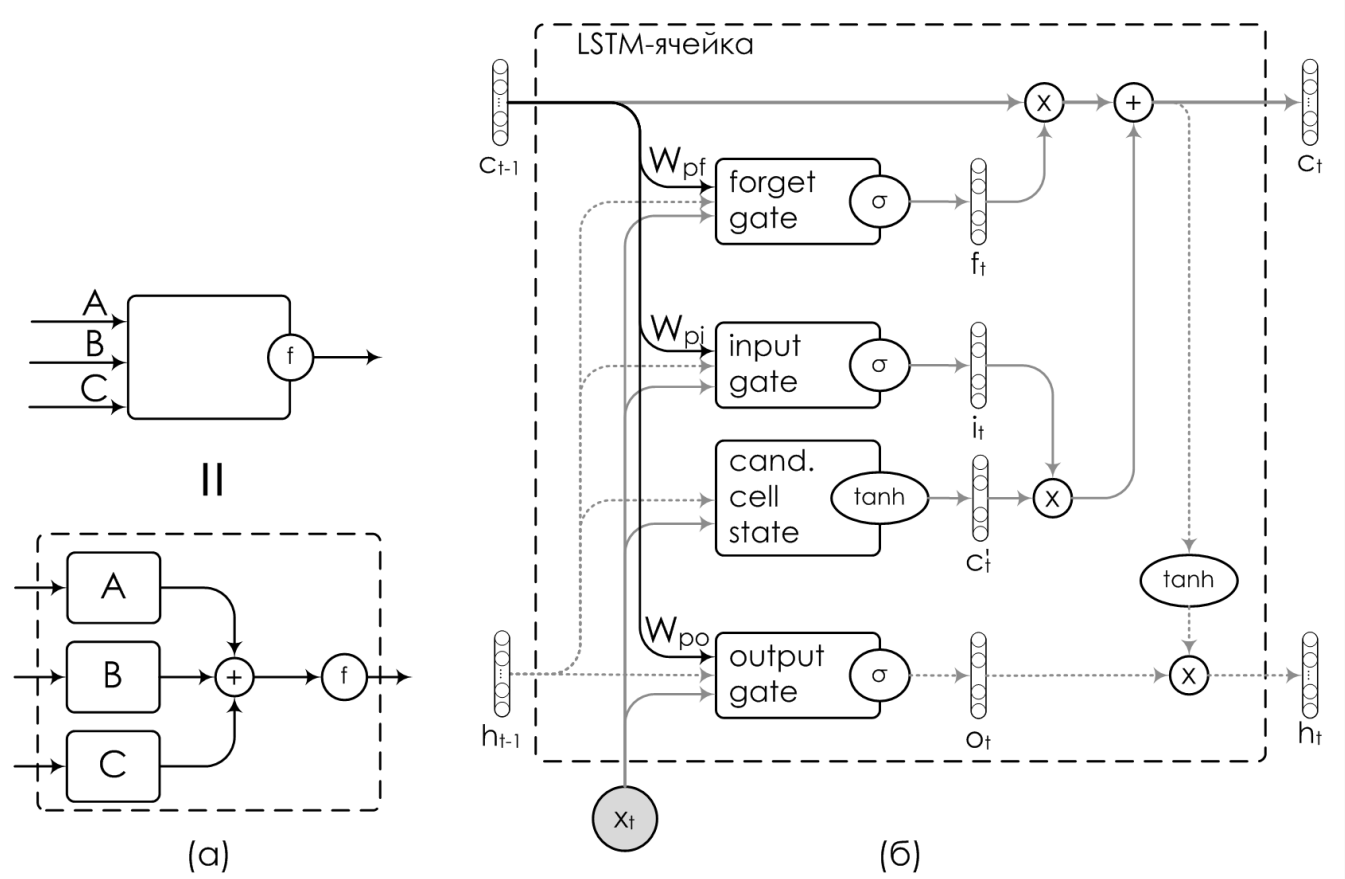
- LSTM был создан в середине 1990-х (Hochreiter and Schmidhuber, 1995; 1997).
- В полностью современной форме в (Gers, Schmidhuber, 2000).
- Проблема: хотим управлять $\mathbf{c}$, но гейты его не получают! Они видят только $\mathbf{h}_{t-1}$, а это

$$\mathbf{h}_{t-1} = \mathbf{o}_{t-1} \odot \tanh(\mathbf{c}_{t-1}).$$

- Так что если output gate закрыт, то поведение LSTM вообще от состояния ячейки не зависит.
- Нехорошо. Что делать?..

LSTM

- ...конечно, добавить ещё несколько матриц! (peepholes)



LSTM

- Формально:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{pi}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{pf}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{po}\mathbf{c}_{t-1} + \mathbf{b}_o\right)
\end{aligned}$$

- Видно, что тут есть огромное поле для вариантов LSTM: можно удалить любой гейт, любую замочную скважину, поменять функции активации...
- Как выбрать?

LSTM

- «LSTM: a Search Space Odyssey» (Greff et al., 2015).
- Большое экспериментальное сравнение.
- В честности, некоторые куда более простые архитектуры (без одного из гейтов!) не сильно проигрывали «ванильному» LSTM.
- И это приводит нас к...

Неразумная эффективность RNN

- Karpathy, «The Unreasonable Effectiveness of Recurrent Neural Networks» (2015): символьная LSTM-модель языка, читает и предсказывает по одной букве.
- Шекспир — пьеса с ремарками, алгебраическая геометрия — леммы в LaTeX, ядро Linux — правдоподобный C.
- В миниатюре: двухслойная LSTM на Tiny Shakespeare после N шагов обучения:

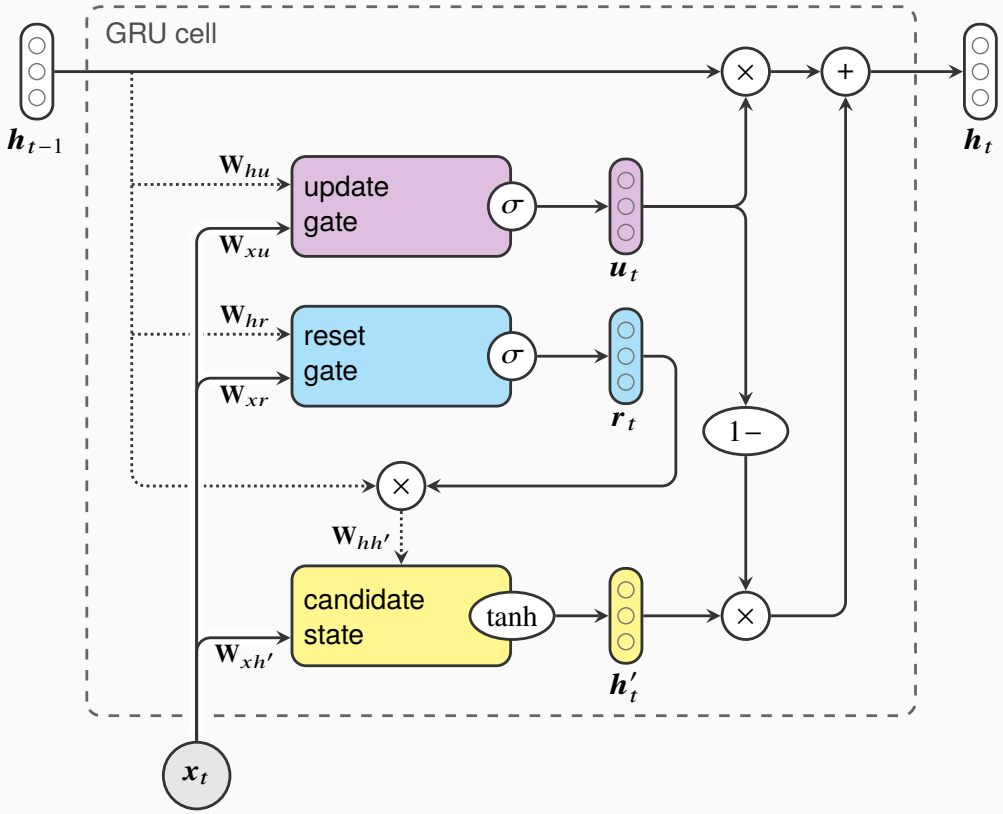
0	`mA;pcPcKN-UkccQ:Ae3xnEgbEo?dL.vDQdFaKSCljO`
500	*GOLUCESTER: Part in good not face a speat true him. That your forthard being midiles.*
2000	*Whose poor cousin of Buckingham, I would be my soul, which bring thy horses of respect was there.*
6000	*The king sound our stalk, and bid his countenance … Why should the head, sir, I thank your grace.*

- Отдельные ячейки выучивают позицию в строке, «внутри кавычек», глубину скобок.

GRU



- ...Gated Recurrent Units (GRU; Cho et al., 2014).
- В GRU тоже есть прямой путь для градиентов, но проще.



GRU

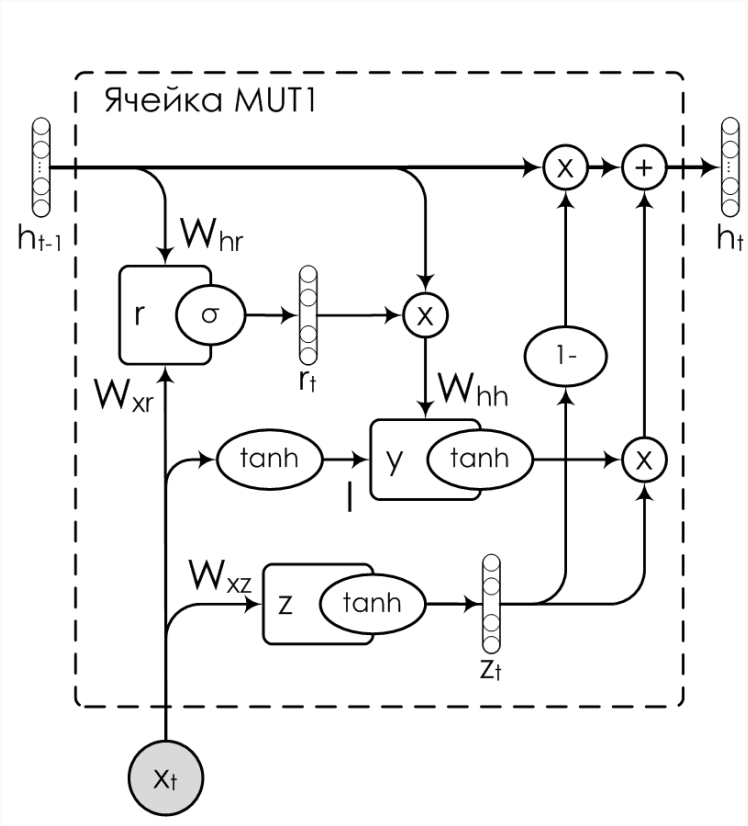
- Формально:

$$
\begin{aligned}
\mathbf{u}_t &= \sigma(W_{xu}\mathbf{x}_t + W_{hu}\mathbf{h}_{t-1} + \mathbf{b}_u) \\
\mathbf{r}_t &= \sigma(W_{xr}\mathbf{x}_t + W_{hr}\mathbf{h}_{t-1} + \mathbf{b}_r) \\
\mathbf{h}'_t &= \tanh(W_{xh'}\mathbf{x}_t + W_{hh'}(\mathbf{r}_t \odot \mathbf{h}_{t-1})) \\
\mathbf{h}_t &= (1 - \mathbf{u}_t) \odot \mathbf{h}'_t + \mathbf{u}_t \odot \mathbf{h}_{t-1}
\end{aligned}
$$

- Теперь есть update gate и reset gate, нет разницы между $\mathbf{c}_t$ и $\mathbf{h}_t$.
- Меньше матриц (6, а не 8 или 11 с замочными скважинами), меньше весов, но только чуть хуже LSTM работает.
- Так что можно больше GRU поместить, и сеть станет лучше.

GRU

- Другие варианты тоже есть.
- (Józefowicz, Zaremba, Sutskever, 2015): огромное сравнение, выращивали архитектуры эволюционными методами.
- Три новых интересных архитектуры; например:



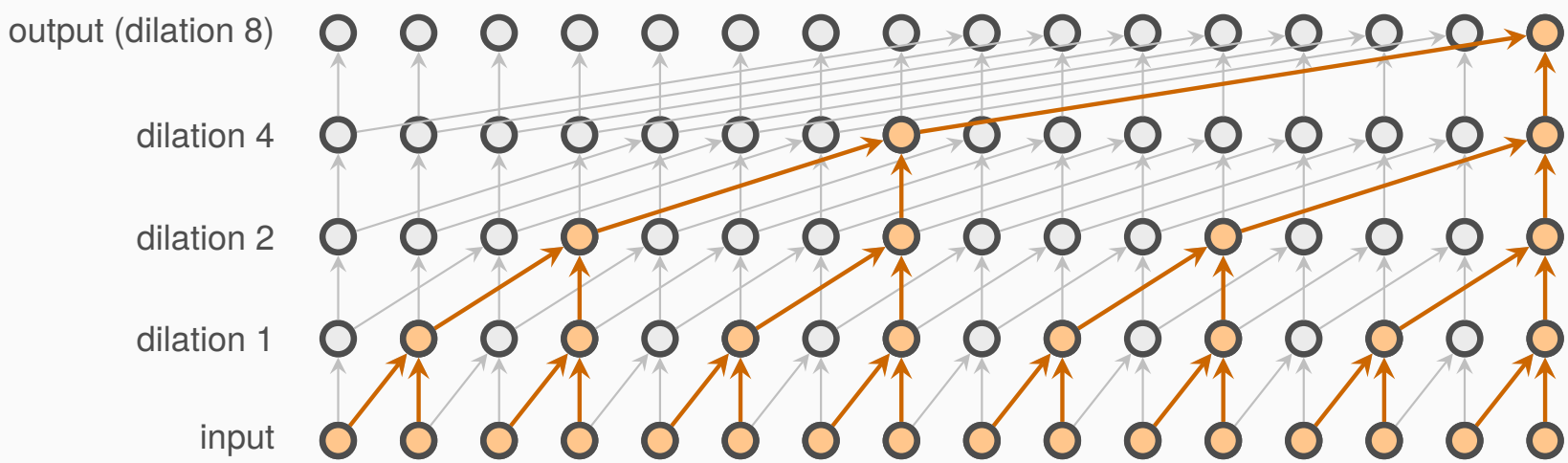
GRU

- Так кто же победил? В середине 2010-х GRU был очень популярен (внимание Bahdanau et al. — на GRU), но надёжным выбором по умолчанию остался LSTM: seq2seq, GNMT, Show-Attend-Tell, а позже AlphaStar и OpenAI Five.
- Есть и теоретическая причина (Weiss et al., 2018): с отдельной ячейкой $\mathbf{c}_t$ LSTM может работать как неограниченный счётчик и распознавать $a^n b^n$, а GRU, как и простая RNN, в конечной точности — нет.
- Итого:

Ячейка	Гейты	Матриц	Путь для памяти
Простая RNN	—	2	нет
LSTM	input, forget, output	8 (11)	сумма в $\mathbf{c}_t$
GRU	update, reset	6	смесь в $\mathbf{h}_t$

Последовательности без рекуррентности

- Рекуррентность — не единственный способ: одномерная *каузальная* свёртка по времени тоже разделяет веса между позициями и смотрит только в прошлое.
- Проблема — контекст. Решение — dilated convolutions, рецептивное поле растёт экспоненциально с глубиной:



- WaveNet (2016) — звук по одному отсчёту; ByteNet и ConvS2S — перевод; TCN (Bai et al., 2018) часто обгоняет LSTM и GRU.

Последовательности без рекуррентности

- Главное преимущество свёрток: все позиции считаются *параллельно*.
- А RNN последовательна по своей природе: T шагов один за другим, и на GPU, которые любят тысячи независимых умножений, это очень плохо.
- Частичный выход: пусть гейты и кандидат считаются только по входу, тогда рекуррентность линейна по состоянию и дешёва:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + (1 - \mathbf{f}_t) \odot \mathbf{z}_t$$

  (QRNN, SRU, а в 2024 году — minGRU и minLSTM); такие рекуррентности даже параллелятся по времени (parallel scan).
- Но в итоге последовательная природа RNN подготовила почву для архитектуры, которая смотрит на всю последовательность сразу. Первый её ингредиент — *внимание*.

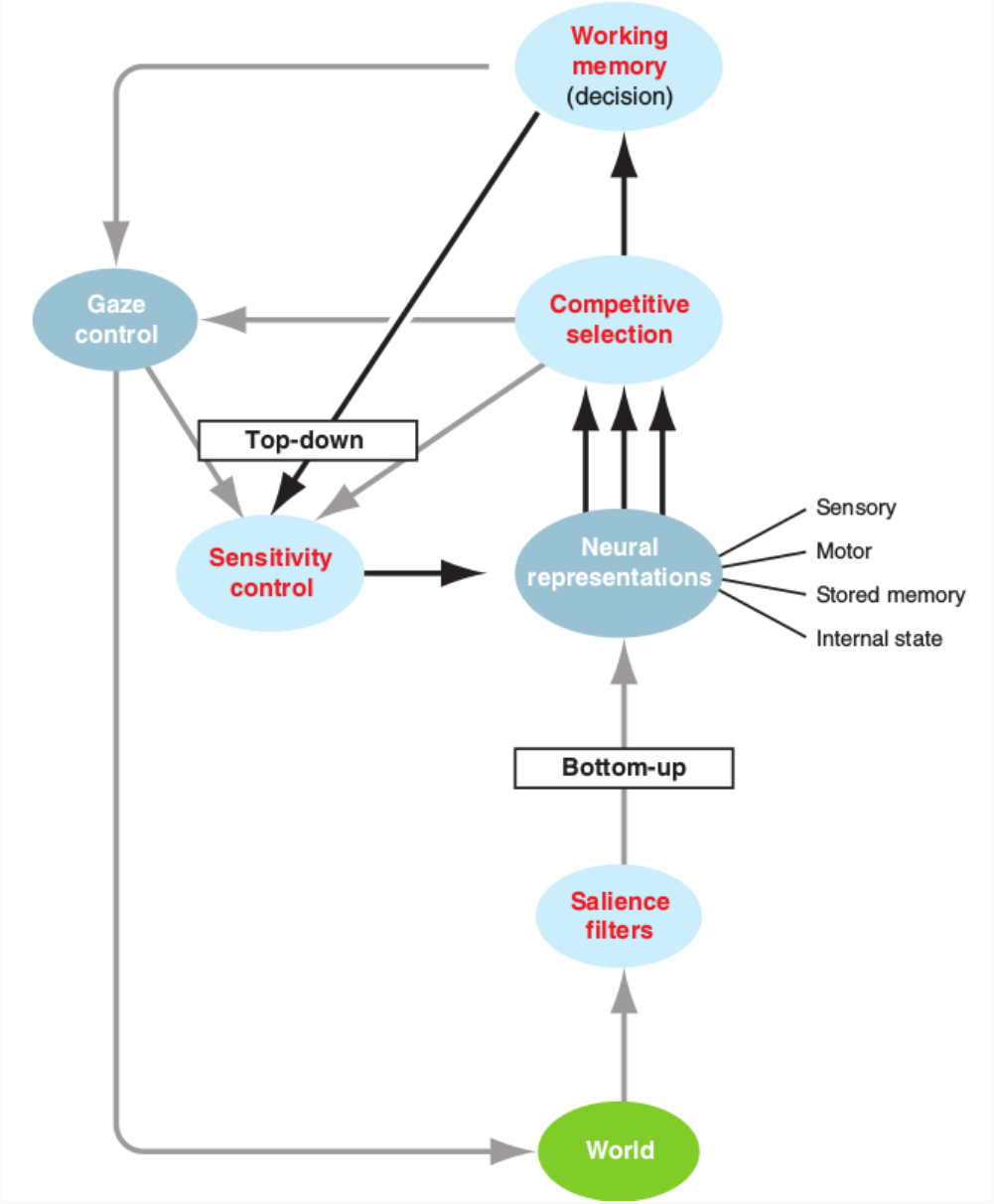
Что такое внимание?

Внимание

- Вы же сейчас *внимательно* меня слушаете, правильно?
- А что это значит?..
- Изображение с сетчатки тоже проходит через CNN, но потом мы часть его замечаем, а часть не особенно. Что это значит?
- Оказывается, что это довольно сложный вопрос.
- А.Р. Лурия: внимание, память и активация коры.

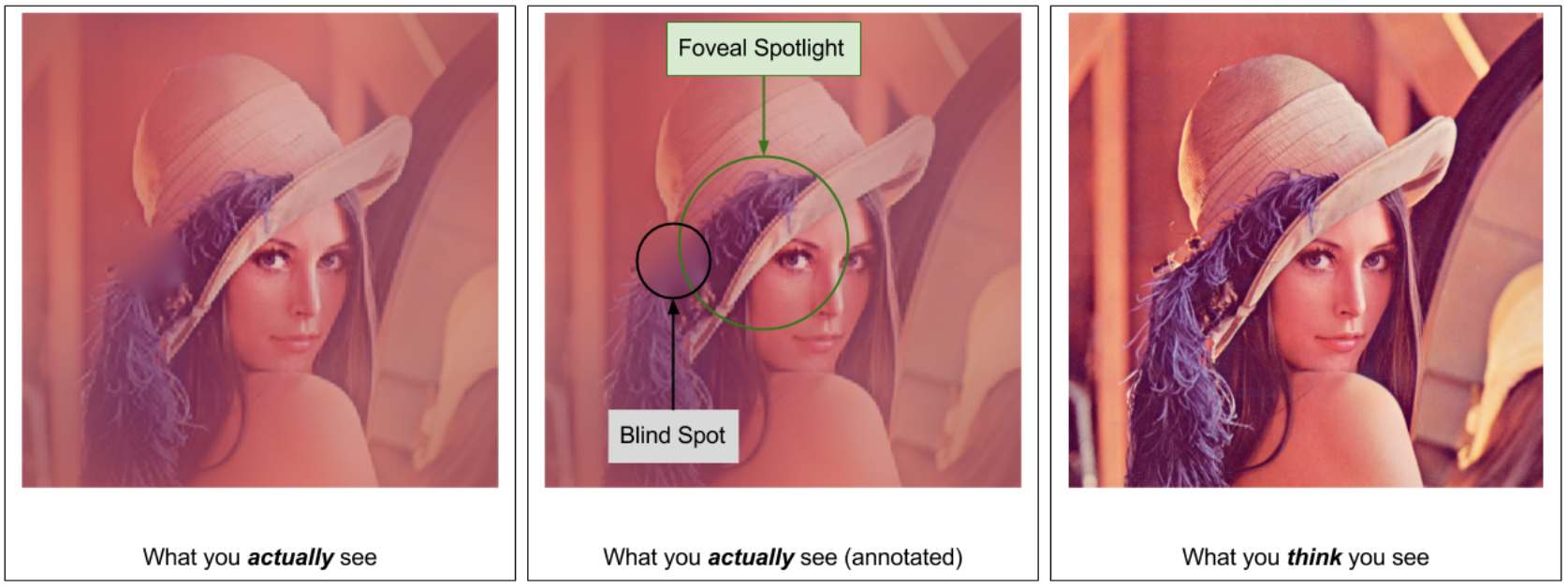
Внимание

- Дело во взаимодействии с рабочей памятью (Knudsen, 2007):



Внимание

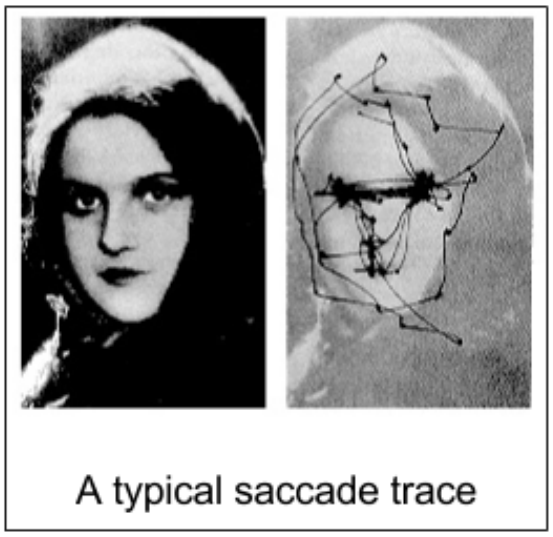
- Как это реализовать в нейронной сети? Особенно «сознательное» внимание.
- Но и «бессознательное» тоже; например, с картинками: мы же на самом деле мало чего видим в каждый момент времени.
- Центральная ямка сетчатки (fovea):



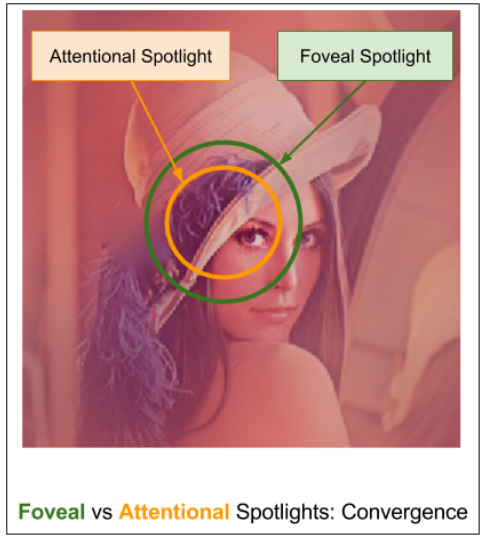
- Simons, Chabris, 1999: `https://www.youtube.com/watch?v=vJG698U2Mvo`

Внимание

- Мы делаем *саккады*, причём это тоже не всегда помогает:



A typical saccade trace

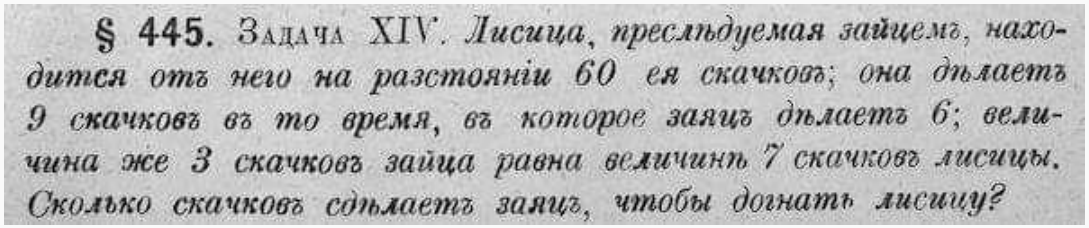


Foveal vs Attentional Spotlights: Convergence



Foveal vs Attentional Spotlights: Divergence

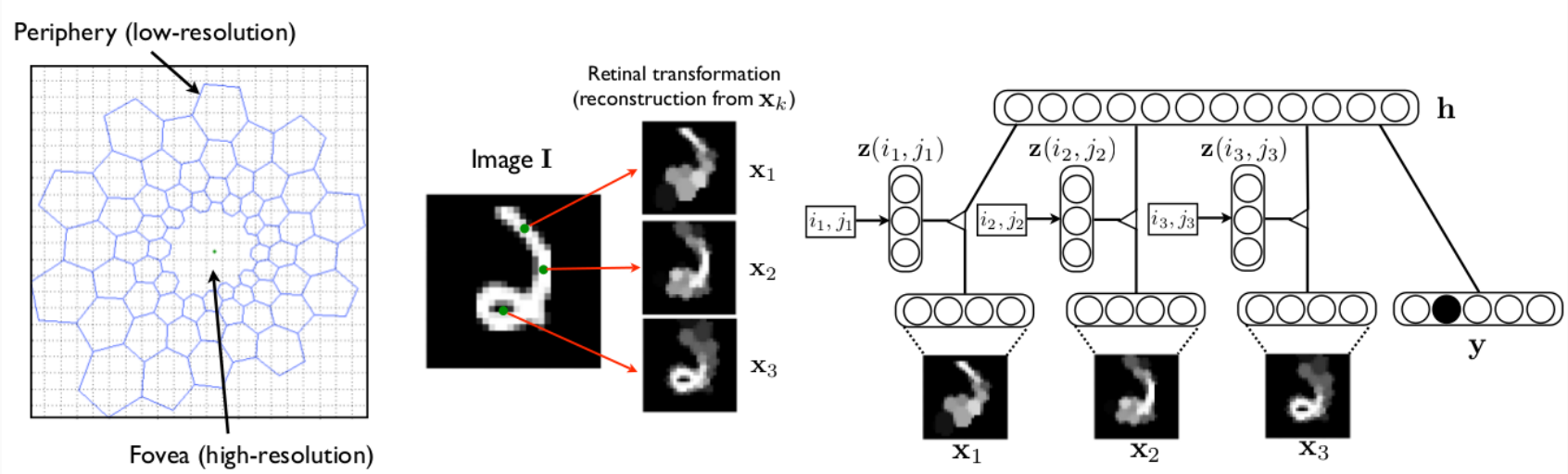
- Или вот не совсем зрительный пример:



§ 445. Задача XIV. *Лисица, преслѣдуемая зайцемъ, находится отъ него на разстояніи 60 ея скачковъ; она дѣлаетъ 9 скачковъ въ то время, въ которое заяцъ дѣлаетъ 6; величина же 3 скачковъ зайца равна величинѣ 7 скачковъ лисицы. Сколько скачковъ сдѣлаетъ заяцъ, чтобы догнать лисицу?*

Foveal glimpses

- В нейросети мы тоже хотим осознанно понимать, «на что» смотреть.
- Одна из первых работ (Larochelle, Hinton, 2010):

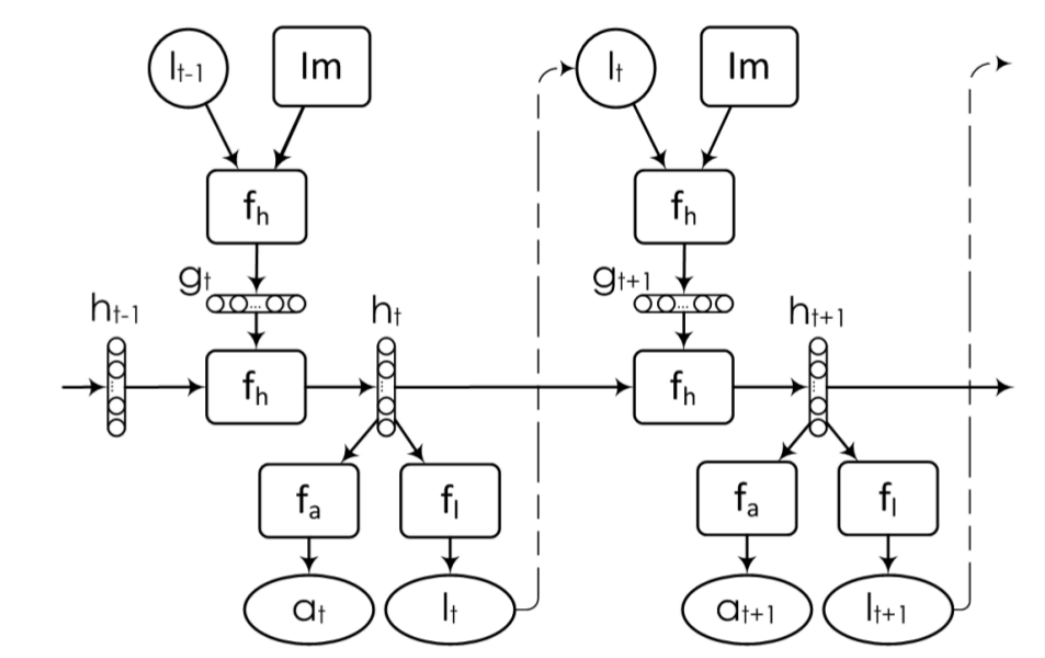


- Пытаются моделировать положения фиксаций и строить последовательность при помощи RBM.
- Последовательность – значит...

Рекуррентные модели зрительного внимания

Recurrent visual attention

- По-настоящему всё появилось в (Mnih et al., 2014), «Recurrent Models of Visual Attention»:
  - из предыдущего $\mathbf{h}_{t-1}$ и положения $_t$ для нового «взгляда» f_g делает $\mathbf{g}_t$, вход для шага t;
  - из $\mathbf{h}_{t-1}$ и $\mathbf{g}_t$ функцией f_h получается $\mathbf{h}_t$;
  - из него – «действие» $a_t = f_a(\mathbf{h}_t)$ и положение следующего «взгляда» $_{t+1} = f_l(\mathbf{h}_t)$.



Recurrent visual attention

- Давайте разберёмся в модели формально:

$$\begin{aligned}
\mathbf{g}_t &= f_g(\mathbf{x}_t, \mathbf{l}_{t-1}; \theta_g), \\
\mathbf{h}_t &= f_h(\mathbf{h}_{t-1}, \mathbf{g}_t; \theta_h), \\
\mathbf{l}_t &\sim p(\cdot \mid f_l(\mathbf{h}_t; \theta_l)), \\
a_t &\sim p(\cdot \mid f_a(\mathbf{h}_t; \theta_a)).
\end{aligned}$$

- После очередного действия получается новое наблюдение $\mathbf{x}_{t+1}$ и награда r_t, которая будет скорее всего в конце, после всех шагов, за правильную классификацию.
- Что это напоминает?..

Recurrent visual attention

- ...о да, это reinforcement learning!
- Выучить надо стохастическую стратегию $\pi((\mathbf{l}_t, a_t) \mid \mathbf{s}_{1:t}; \theta)$, которая по истории будет выдавать следующее действие.
- У нас π задаётся через RNN, а оптимизировать надо

$$J(\theta) = \mathbb{E}_{p(\mathbf{s}_{1:T};\theta)}[R] = \mathbb{E}_{p(\mathbf{s}_{1:T};\theta)}\left[\sum_{t=1}^{T} r_t\right].$$

- Выглядит очень сложно – ожидание по последовательностям действий, т.е. по пространству большой размерности.
- Но есть методы – давайте сделаем preview тому, что потом будет в reinforcement learning.

Recurrent visual attention

- (Williams, 1992): алгоритм REINFORCE, в котором доказывается и используется выборочная оценка этого ожидания. Давайте выведем, а потом применим к нашему случаю...
- Нам надо оптимизировать

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^{T} r_t(s_t, a_t) \right] \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} r(s_t^{(i)}, a_t^{(i)}),$$

  где мы взяли M примеров траекторий τ.
- Определим $r(\tau) = \sum_t r(s_t, a_t)$. Тогда

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [r(\tau)] = \int \pi_\theta(\tau) r(\tau) \mathrm{d}\tau,$$

  т.е. тот самый страшный интеграл по траекториям.
- Но оказывается, что можно продифференцировать по θ...

Recurrent visual attention

- Продифференцируем по θ:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int \nabla_\theta \pi_\theta(\tau) r(\tau) \mathrm{d}\tau \\
&= \int \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} r(\tau) \mathrm{d}\tau \\
&= \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) \mathrm{d}\tau \\
&= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[\nabla_\theta \log \pi_\theta(\tau) r(\tau) \right] \\
&\approx \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta \log \pi_\theta(\tau^{(i)}) r^{(i)}(\tau),
\end{aligned}$$

если приблизить выборкой; но сначала давайте ещё посмотрим на $\pi_\theta(\tau)$...

Recurrent visual attention

- Вероятность определяется как

$$\pi_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t).$$

- Берём логарифм, а потом заметим, что от θ зависят только действия:

$$\nabla_\theta \log \pi_\theta(\tau) =$$

$$= \nabla_\theta \left(\log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=1}^{T} \log p(s_{t+1} \mid s_t, a_t) \right) =$$

$$= \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

Recurrent visual attention

- Итого получается вполне tractable градиент:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[r(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]$$
$$\approx \frac{1}{M} \sum_{i=1}^{M} r(\tau^{(i)}) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}).$$

- У нас тоже можно считать, что награда R даётся целиком:

$$\nabla_\theta J(\theta) \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi \left(\mathbf{l}_t^{(i)}, a_t^{(i)} \mid \mathbf{s}_{1:t}^{(i)}; \theta \right) R^{(i)}.$$

- Т.е. надо уметь считать $\log \pi \left(\mathbf{l}_t^{(i)}, a_t^{(i)} \mid \mathbf{s}_{1:t}^{(i)}; \theta \right)$, но в случае RNN это просто градиент сети, который можно посчитать через backpropagation.
- Ещё можно сделать частично supervised loss на последнем шаге, где мы знаем классификацию.

Recurrent visual attention

- Результаты:



(a) Translated MNIST inputs.



(b) Cluttered Translated MNIST inputs.

(a) 28x28 MNIST

Model	Error
FC, 2 layers (256 hiddens each)	1.69%
Convolutional, 2 layers	1.21%
RAM, 2 glimpses, 8×8, 1 scale	3.79%
RAM, 3 glimpses, 8×8, 1 scale	1.51%
RAM, 4 glimpses, 8×8, 1 scale	1.54%
RAM, 5 glimpses, 8×8, 1 scale	1.34%
RAM, 6 glimpses, 8×8, 1 scale	1.12%
RAM, 7 glimpses, 8×8, 1 scale	**1.07%**

(b) 60x60 Translated MNIST

Model	Error
FC, 2 layers (64 hiddens each)	6.42%
FC, 2 layers (256 hiddens each)	2.63%
Convolutional, 2 layers	1.62%
RAM, 4 glimpses, 12×12, 3 scales	1.54%
RAM, 6 glimpses, 12×12, 3 scales	**1.22%**
RAM, 8 glimpses, 12×12, 3 scales	**1.2%**

Recurrent visual attention

- Результаты:

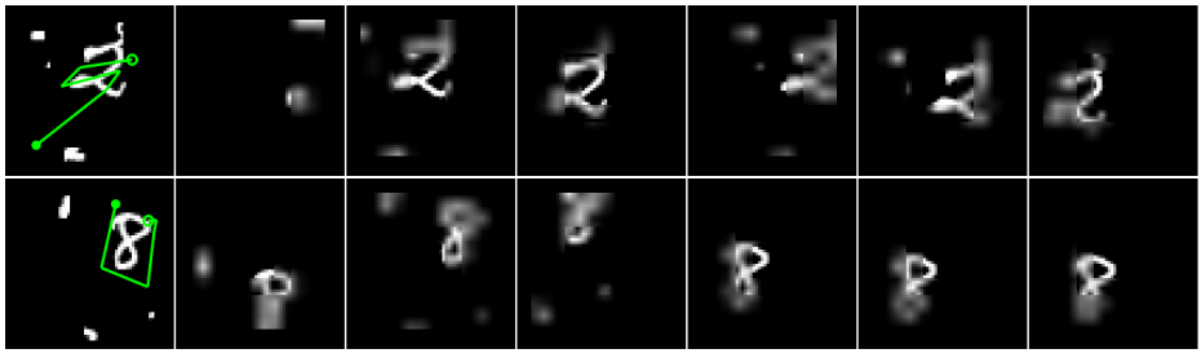
(a) 60x60 Cluttered Translated MNIST

Model	Error
FC, 2 layers (64 hiddens each)	28.58%
FC, 2 layers (256 hiddens each)	11.96%
Convolutional, 2 layers	8.09%
RAM, 4 glimpses, 12 × 12, 3 scales	4.96%
RAM, 6 glimpses, 12 × 12, 3 scales	4.08%
RAM, 8 glimpses, 12 × 12, 3 scales	4.04%
RAM, 8 random glimpses	14.4%

(b) 100x100 Cluttered Translated MNIST

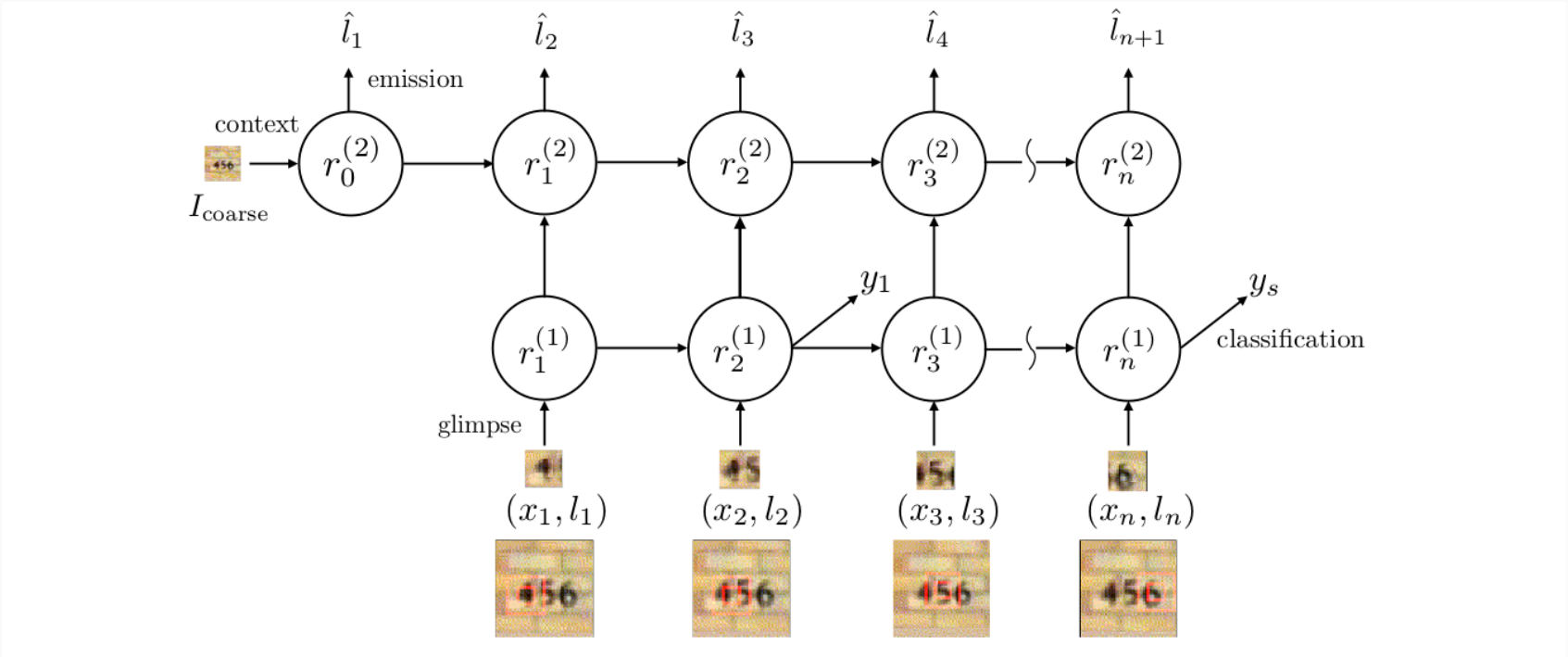
Model	Error
Convolutional, 2 layers	14.35%
RAM, 4 glimpses, 12 × 12, 4 scales	9.41%
RAM, 6 glimpses, 12 × 12, 4 scales	8.31%
RAM, 8 glimpses, 12 × 12, 4 scales	8.11%
RAM, 8 random glimpses	28.4%

- А вот как внимание гуляет по картинке:



Recurrent visual attention

- В следующей работе (Ba et al., 2015) сделали глубокую модель:



- Кстати, обучали по-другому, вариационными методами. Как это?..

Recurrent visual attention

- Нам нужно классифицировать, т.е. $p(y \mid \mathbf{x}, \theta)$ максимизировать.
- Маргинализуем по положениям glimpses:

$$\log p(y \mid \mathbf{x}, \theta) = \log \sum_l p(l \mid \mathbf{x}, \theta) p(y \mid l, \mathbf{x}, \theta).$$

- Запишем вариационную нижнюю оценку свободной энергии (как её получить?):

$$\begin{aligned} \log \sum_l p(l \mid \mathbf{x}, \theta) p(y \mid l, \mathbf{x}, \theta) &\geq \sum_l p(l \mid \mathbf{x}, \theta) \log p(y, l \mid \mathbf{x}, \theta) + H[l] \\ &= \sum_l p(l \mid \mathbf{x}, \theta) \log p(y \mid l, \mathbf{x}, \theta). \end{aligned}$$

Recurrent visual attention

- И теперь можно брать производные:

$$\frac{\partial J}{\partial \theta} = \sum_l p(l \mid \mathbf{x}, \theta) \frac{\partial \log p(y \mid l, \mathbf{x}, \theta)}{\partial \theta} + \sum_l \log p(y \mid l, \mathbf{x}, \theta) \frac{\partial p(l \mid \mathbf{x}, \theta)}{\partial \theta}$$

$$= \sum_l p(l \mid \mathbf{x}, \theta) \left[\frac{\partial \log p(y \mid l, \mathbf{x}, \theta)}{\partial \theta} + \log p(y \mid l, \mathbf{x}, \theta) \frac{\partial \log p(l \mid \mathbf{x}, \theta)}{\partial \theta} \right].$$

- А эту сумму уже будем приближать выборкой:

$$\frac{\partial J}{\partial \theta} \approx \frac{1}{M} \sum_{i=1}^{M} \left[\frac{\partial \log p(y \mid l^{(i)}, \mathbf{x}, \theta)}{\partial \theta} + \log p(y \mid l^{(i)}, \mathbf{x}, \theta) \frac{\partial \log p(l^{(i)} \mid \mathbf{x}, \theta)}{\partial \theta} \right],$$

$$\text{где} \quad l^{(i)} \sim p(l_n \mid \mathbf{x}, \theta) = N(l_n \mid \hat{l}_n, \Sigma).$$

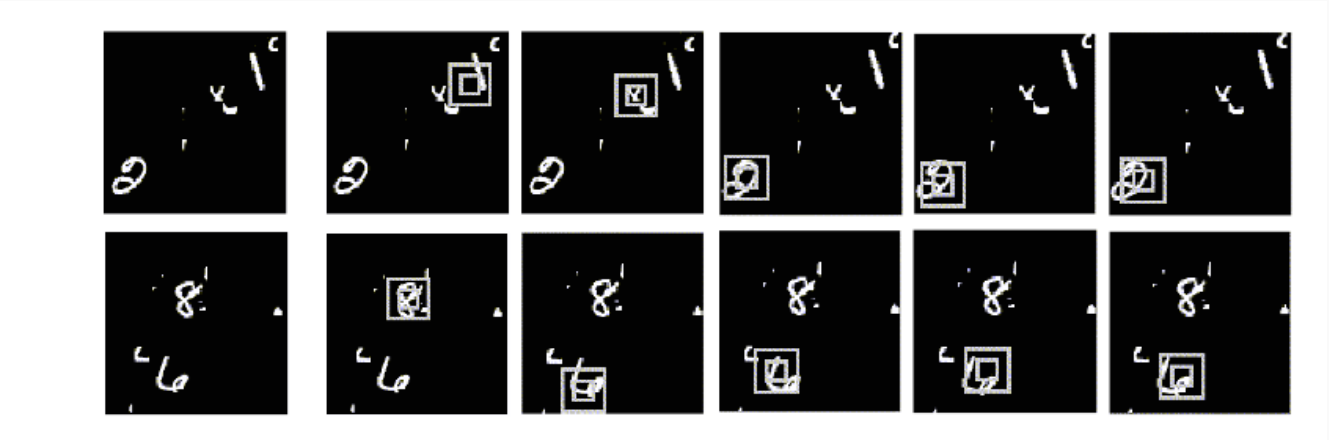
- И это уже алгоритм: сэмплируем glimpses, потом используем их в backpropagation.
- Как и в (Mnih et al., 2014), надо бы уменьшить дисперсию; для этого вычитают baseline, пока не будем углубляться.

Recurrent visual attention

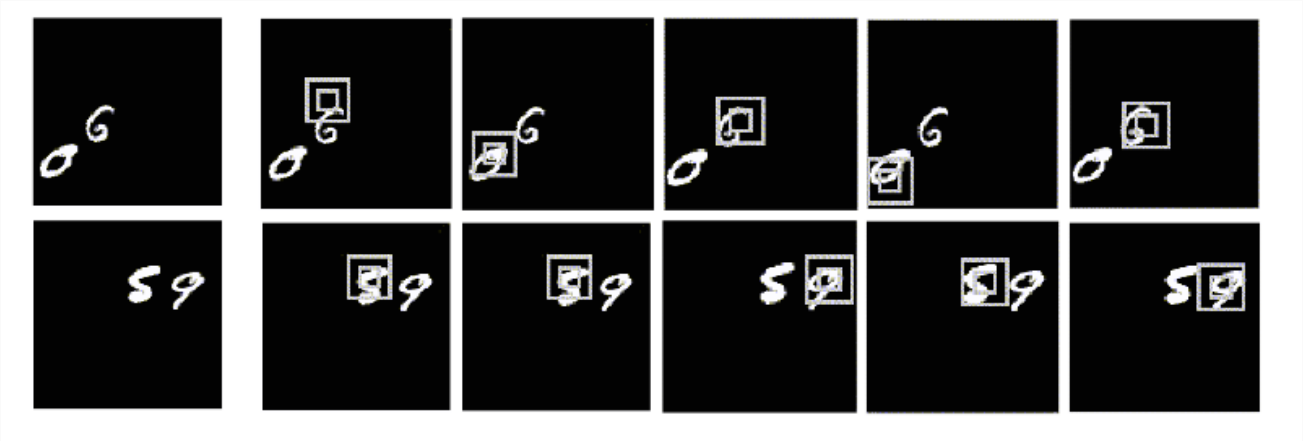
- Получился интересный результат – мы увидели, что примерно один и тот же алгоритм может получиться с двух разных сторон:
  - из обучения с подкреплением через REINFORCE;
  - из вариационной оценки собственно целевой функции.
- Важный гиперпараметр – размер glimpse, т.е. как переводить единицы измерений в системе координат glimpses в пиксели.
- То же самое легко расширить на последовательную классификацию нескольких объектов – просто фиксированное число glimpses на объект, потом классифицируем, плюс терминальное действие в конце всего.

Recurrent visual attention

- Вот как это работает на распознавании двух цифр:



- Любопытно, что на сложении по-другому:



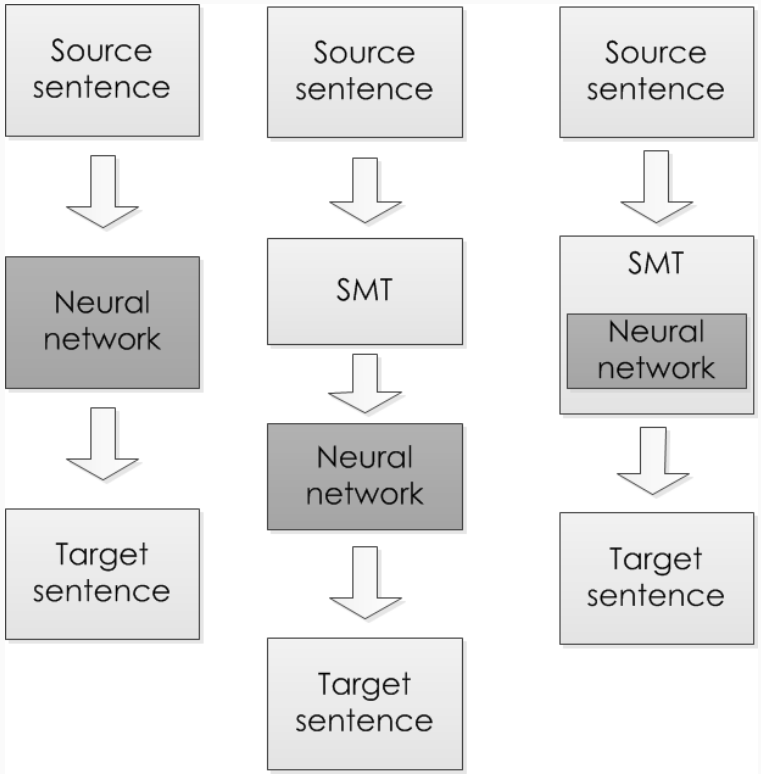
Машинный перевод: encoder-decoder и внимание

Машинный перевод

- Перевод – очень хорошая задача:
  - очевидно очень практическая;
  - очевидно очень высокоуровневая, требует понимания;
  - считается довольно неплохо квантифицируемой (BLEU, TER – хотя см. выше);
  - имеет большие доступные датасеты параллельных переводов.

Машинный перевод

- Статистический машинный перевод (statistical machine translation, SMT): моделируем условную вероятность $p(y \mid x)$ перевода y при условии исходного текста x.
- Классический SMT: моделируем $\log p(y \mid x)$ линейной комбинацией признаков, строим признаки.



Машинный перевод

- Нам больше интересно моделирование sequence-to-sequence:
  - RNN естественным образом моделирует последовательность $X = (x_1, x_2, \dots, x_T)$ как $p(x_1)$, $p(x_2 \mid x_1)$, ..., $p(x_T \mid x_{<T}) = p(x_T \mid x_{T-1}, \dots, x_1)$, и теперь $p(X)$ – это просто

$$p(X) = p(x_1)p(x_2 \mid x_1) \dots p(x_k \mid x_{<k}) \dots p(x_T \mid x_{<T});$$

  - так RNN и в языковых моделях используются;
  - предсказываем следующее слово на основе скрытого состояния и предыдущего слова;
- Как применить эту идею к переводу?

Метрики качества для sequence-to-sequence моделей

- Дальше будет самое интересное: машинный перевод, диалоговые модели, ответ на вопросы.
- Но как мы будем оценивать NLP-модели, которые генерируют текст?
- Есть метрики качества, которые сравнивают результат с правильными ответами:
  - BLEU (Bilingual Evaluation Understudy): перевзвешенная precision (в т.ч. для нескольких правильных ответов);
  - METEOR: гармоническое среднее precision и recall по униграммам;
  - TER (Translation Edit Rate): число исправлений между выходом и правильным ответом, делённое на среднее число слов;
  - LEPOR: комбинируем базовые факторы и метрики с настраиваемыми параметрами.
- Есть ещё куча метрик, связанных с представлениями слов и предложений (хотим поближе к правильному ответу).
- Одна только проблема...

Метрики качества для sequence-to-sequence моделей

- ...всё это вообще не работает.

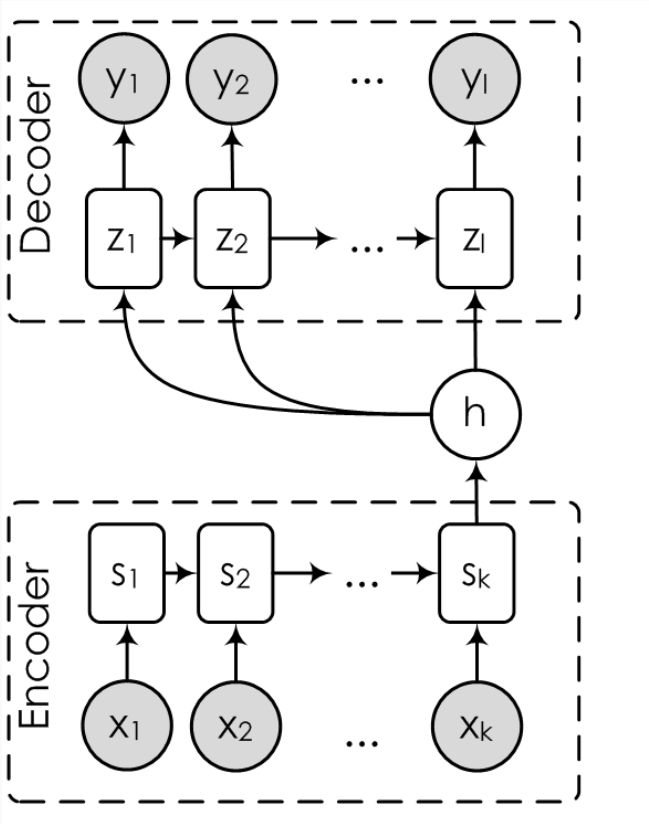
	Twitter				Ubuntu			
Metric	**Spearman**	p-value	**Pearson**	p-value	**Spearman**	p-value	**Pearson**	p-value
Greedy	0.2119	0.034	0.1994	0.047	0.05276	0.6	0.02049	0.84
Average	0.2259	0.024	0.1971	0.049	-0.1387	0.17	-0.1631	0.10
Extrema	0.2103	0.036	0.1842	0.067	0.09243	0.36	-0.002903	0.98
METEOR	0.1887	0.06	0.1927	0.055	0.06314	0.53	0.1419	0.16
BLEU-1	0.1665	0.098	0.1288	0.2	-0.02552	0.8	0.01929	0.85
BLEU-2	0.3576	< 0.01	0.3874	< 0.01	0.03819	0.71	0.0586	0.56
BLEU-3	0.3423	< 0.01	0.1443	0.15	0.0878	0.38	0.1116	0.27
BLEU-4	0.3417	< 0.01	0.1392	0.17	0.1218	0.23	0.1132	0.26
ROUGE	0.1235	0.22	0.09714	0.34	0.05405	0.5933	0.06401	0.53
Human	0.9476	< 0.01	1.0	0.0	0.9550	< 0.01	1.0	0.0

Table 3: Correlation between each metric and human judgements for each response. Correlations shown in the human row result from randomly dividing human judges into two groups.

- Тут нужно что-то новое. И пока не совсем ясно, что именно.

Encoder-decoder архитектуры

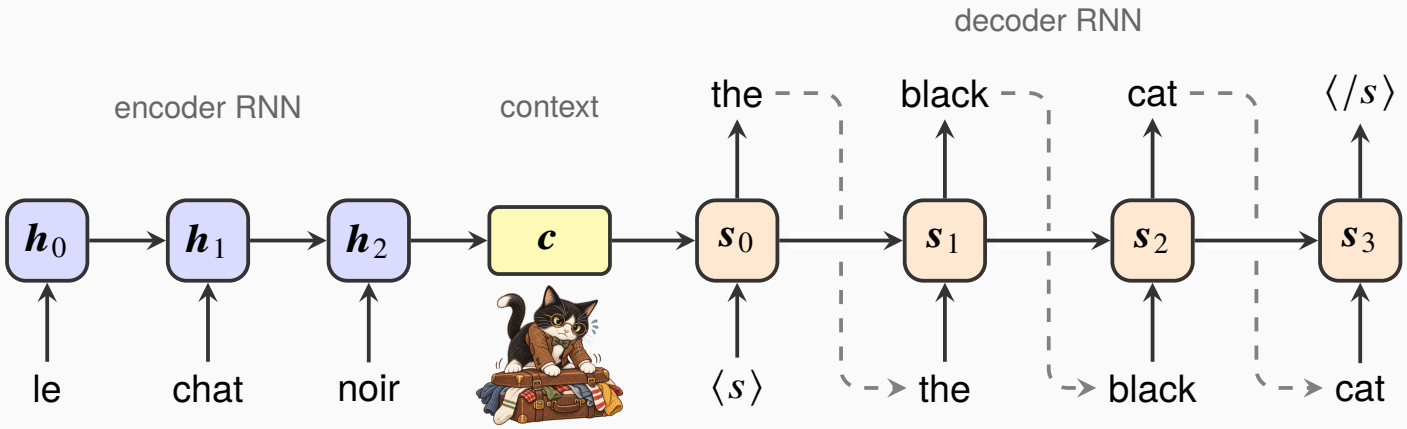
- Encoder-decoder архитектуры (Sutskever et al., 2014; Cho et al., 2014):



- Сначала кодируем, потом декодируем обратно.

Encoder-decoder архитектуры

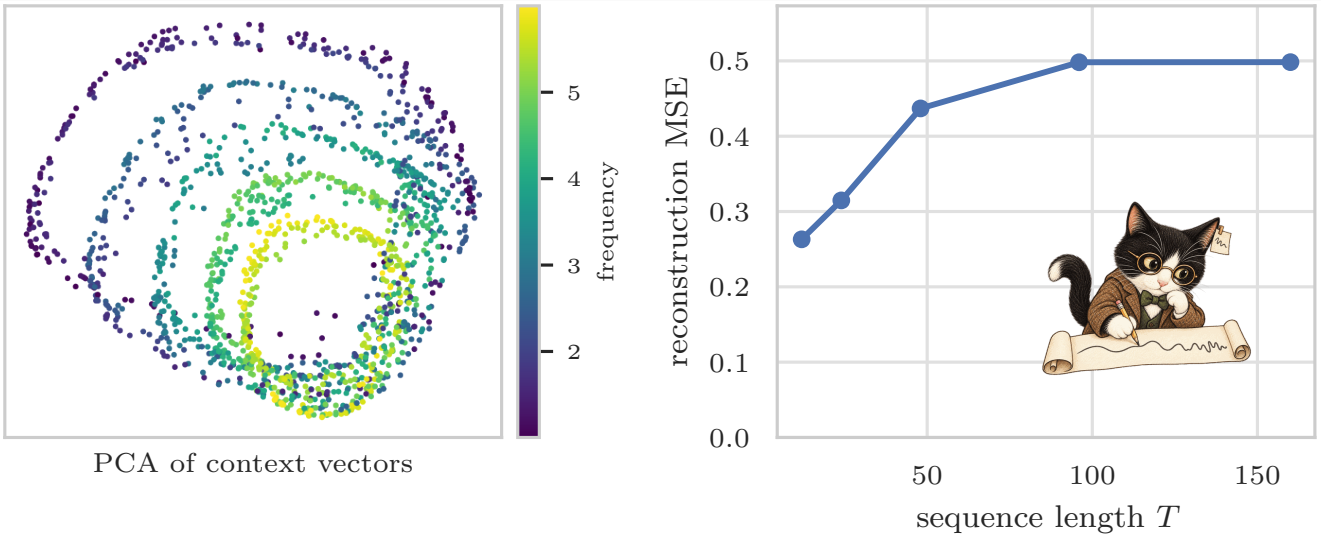
- Так же может работать и с переводом:



- Проблема: надо сжимать всё предложение в один вектор.
- С длинными участками текста это вообще перестаёт работать.

Encoder-decoder архитектуры

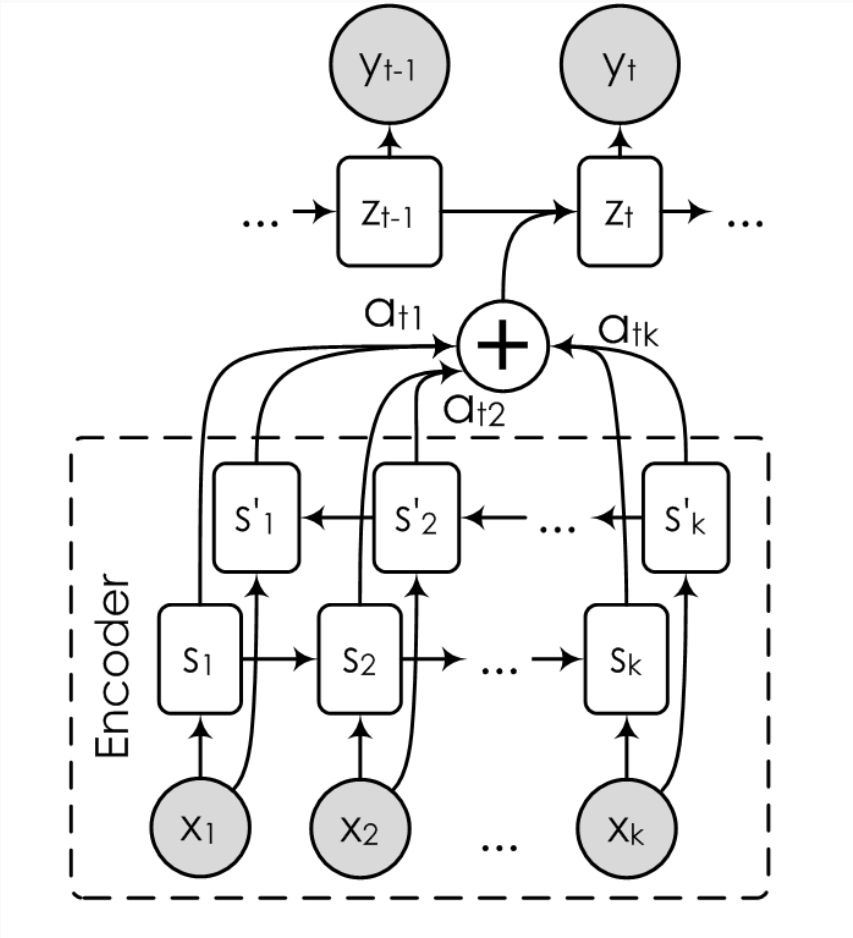
- Эксперимент: LSTM-автокодировщик сжимает отрезок синусоиды в один вектор контекста и должен нарисовать её обратно.
- Векторы контекста упорядочены по частоте (слева), но при фиксированном размере контекста с ростом длины входа реконструкция всё хуже (справа):



- Заплатки: вход задом наперёд (Sutskever et al., 2014), двунаправленный кодировщик.

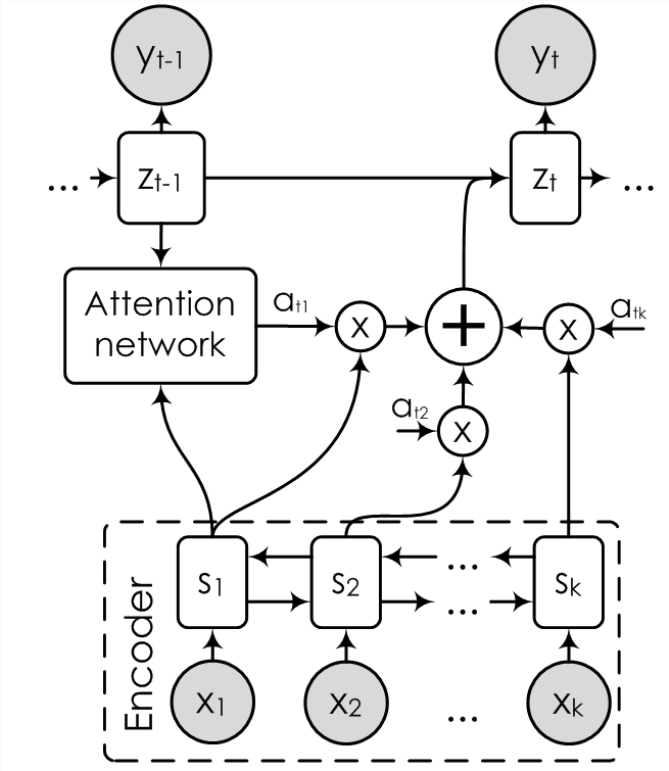
Внимание в нейронных сетях

- Решение: давайте обучим специальные веса, показывающие, насколько та или иная часть входа важна для текущего выхода.
- Прямое применение – двунаправленный LSTM плюс внимание (Bahdanau et al. 2014):



Внимание в нейронных сетях

- *Мягкое внимание* (soft attention) (Luong et al. 2015a; 2015b; Jean et al. 2015):
  - encoder – двунаправленная RNN, есть оба контекста;
  - сеть внимания выдаёт оценку релевантности – надо ли переводить это слово прямо сейчас?



Внимание в нейронных сетях

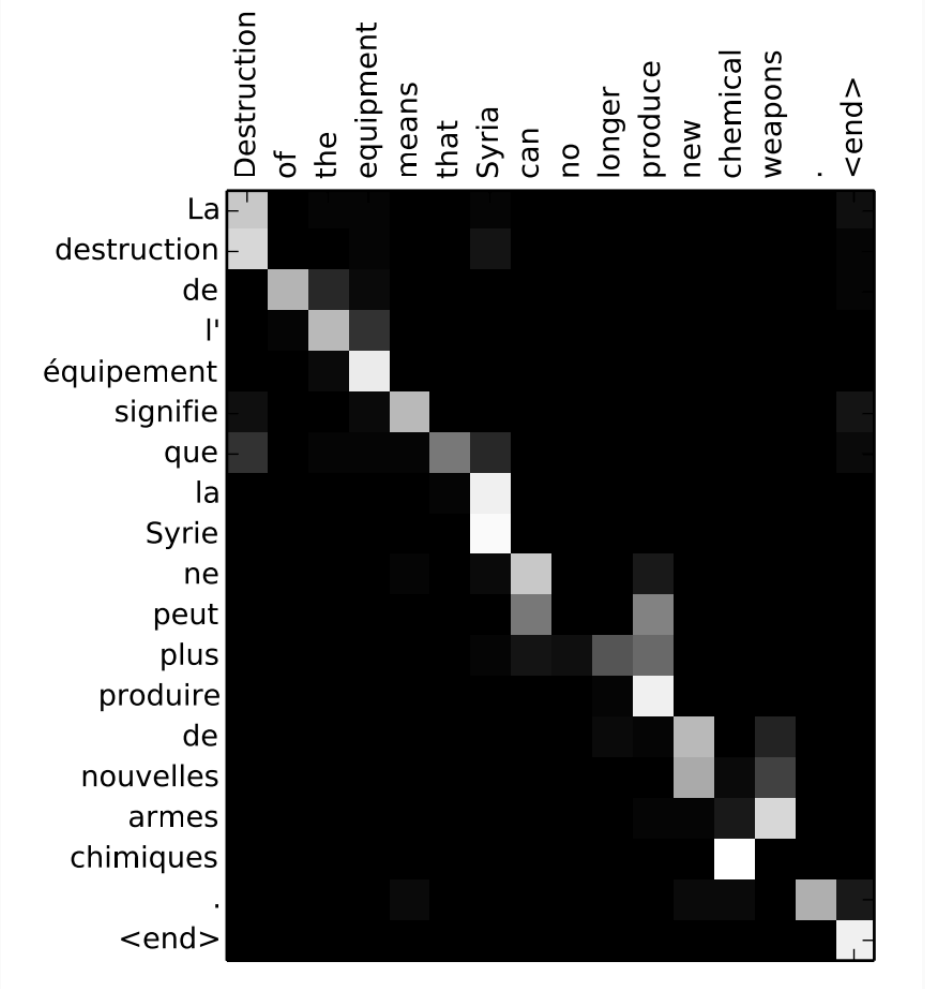
- Формально очень просто: считаем веса внимания α_{tj} и перевзвешиваем состояния кодировщика:

$$e_{tj} = a(\mathbf{s}_{t-1}, \mathbf{h}_j), \quad \alpha_{tj} = \operatorname{softmax}(e_{tj}; e_{t*}),$$
$$\mathbf{c}_t = \sum_j \alpha_{tj}\mathbf{h}_j, \text{ и теперь } \mathbf{s}_t = f(\mathbf{s}_{t-1}, y_{t-1}, \mathbf{c}_t).$$

- Функция внимания a бывает разной:
  - аддитивная (Bahdanau): $a(\mathbf{s}, \mathbf{h}) = \mathbf{v}^\top \tanh(W_1\mathbf{s} + W_2\mathbf{h})$;
  - мультипликативная: $a(\mathbf{s}, \mathbf{h}) = \mathbf{s}^\top\mathbf{h}$ или $\mathbf{s}^\top W\mathbf{h}$;
  - scaled dot-product: $a(\mathbf{s}, \mathbf{h}) = \frac{1}{\sqrt{d}}\mathbf{s}^\top\mathbf{h}$ — запомним её, она нам скоро понадобится.
- По сути это мягкий поиск: запрос (состояние декодера) сравниваем с ключами (состояниями кодировщика) и достаём смесь значений.

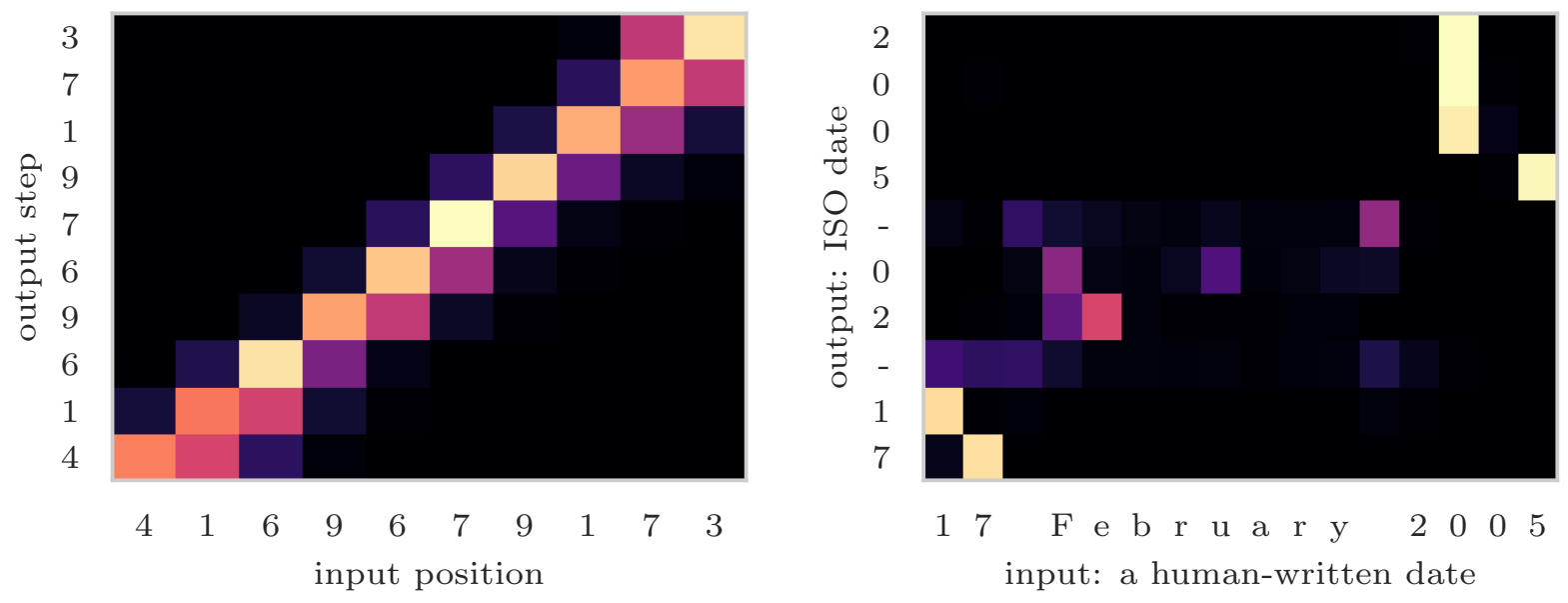
Внимание в нейронных сетях

- В результате можно визуализировать, на что смотрит сеть:



Внимание в нейронных сетях

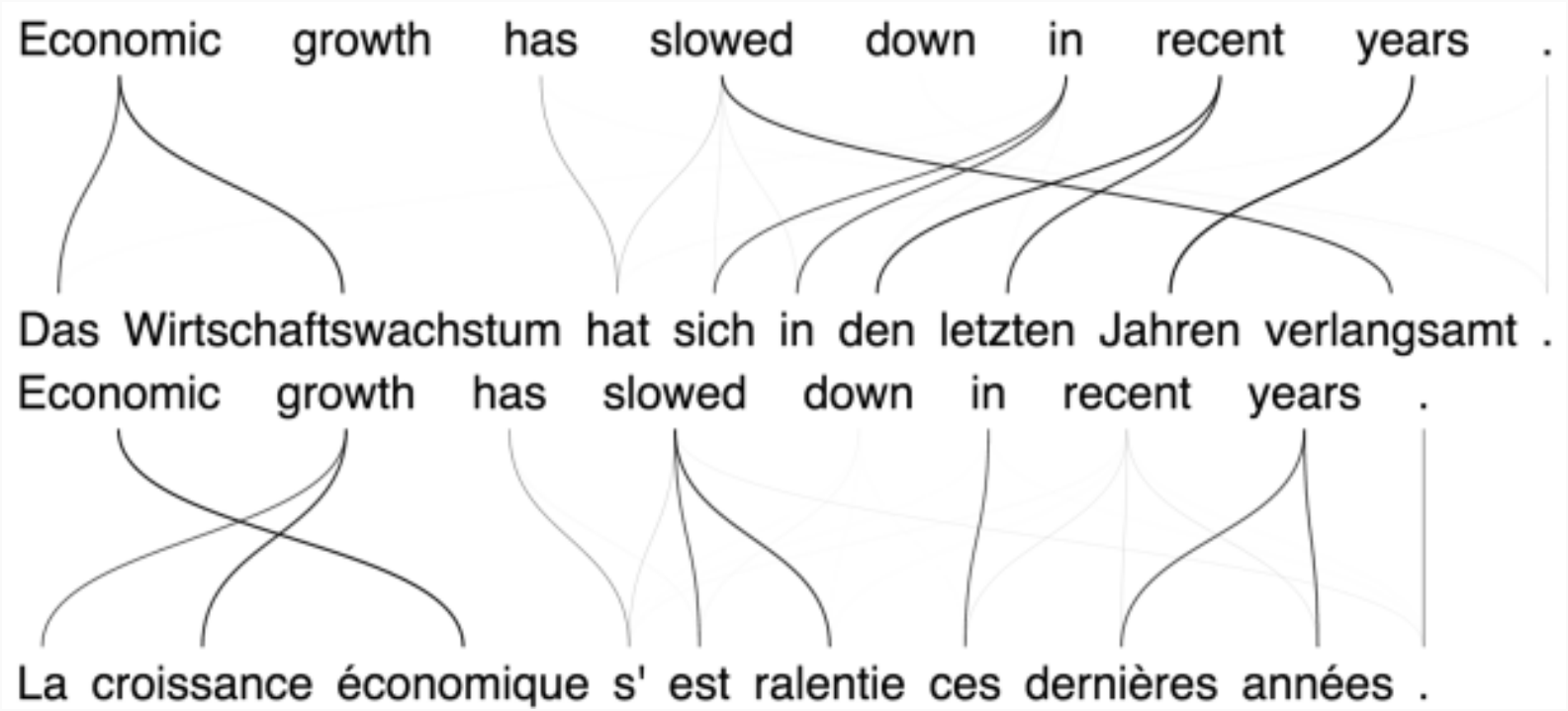
- Попробуем сами: маленький encoder-decoder с вниманием Bahdanau.
- Развернуть последовательность цифр — чистая антидиагональ (слева); привести дату к формату ISO — выравнивание зависит от содержания и скачет (справа):



- Выравнивания никто не подсказывал: модель знала только, правильный ли у неё ответ.

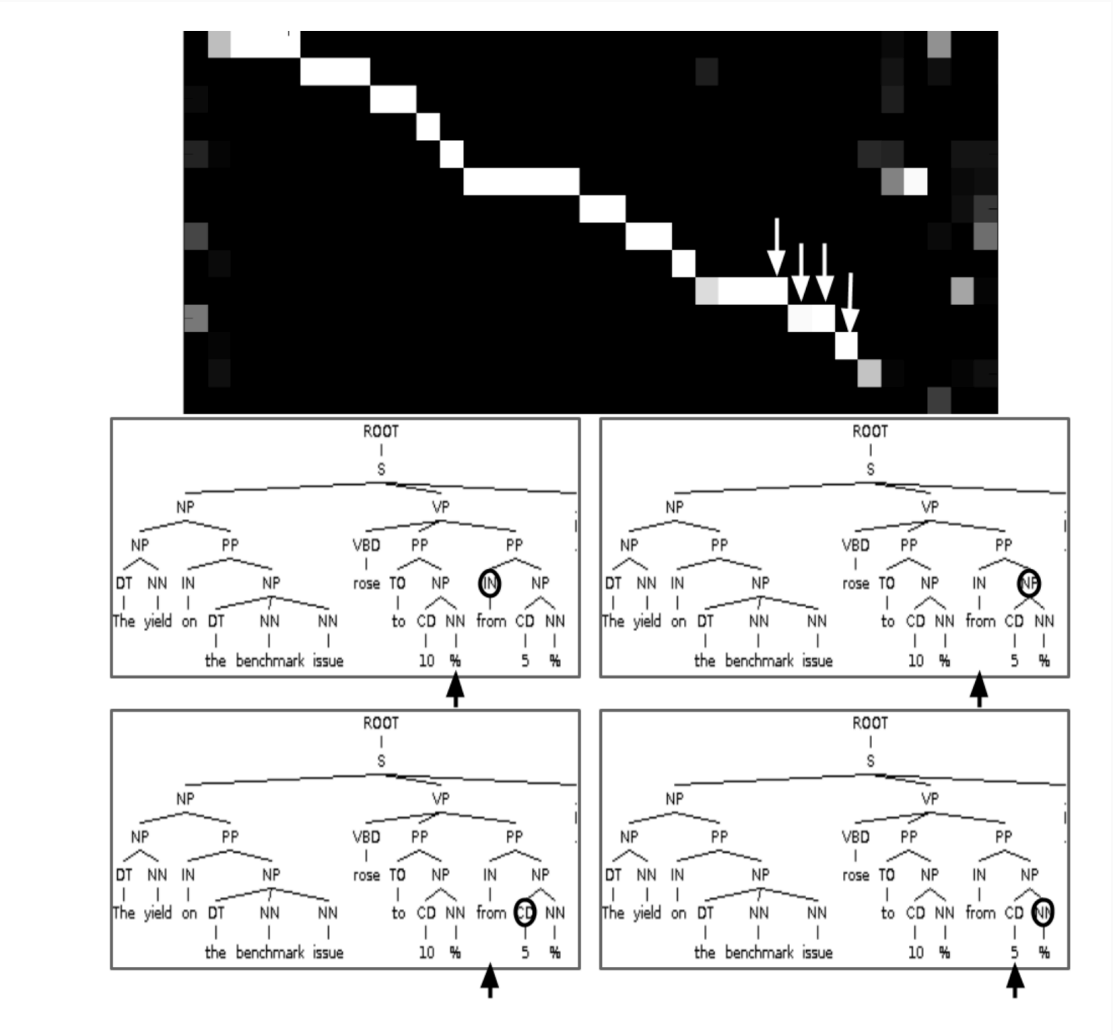
Внимание в нейронных сетях

- Получается гораздо лучше порядок слов:



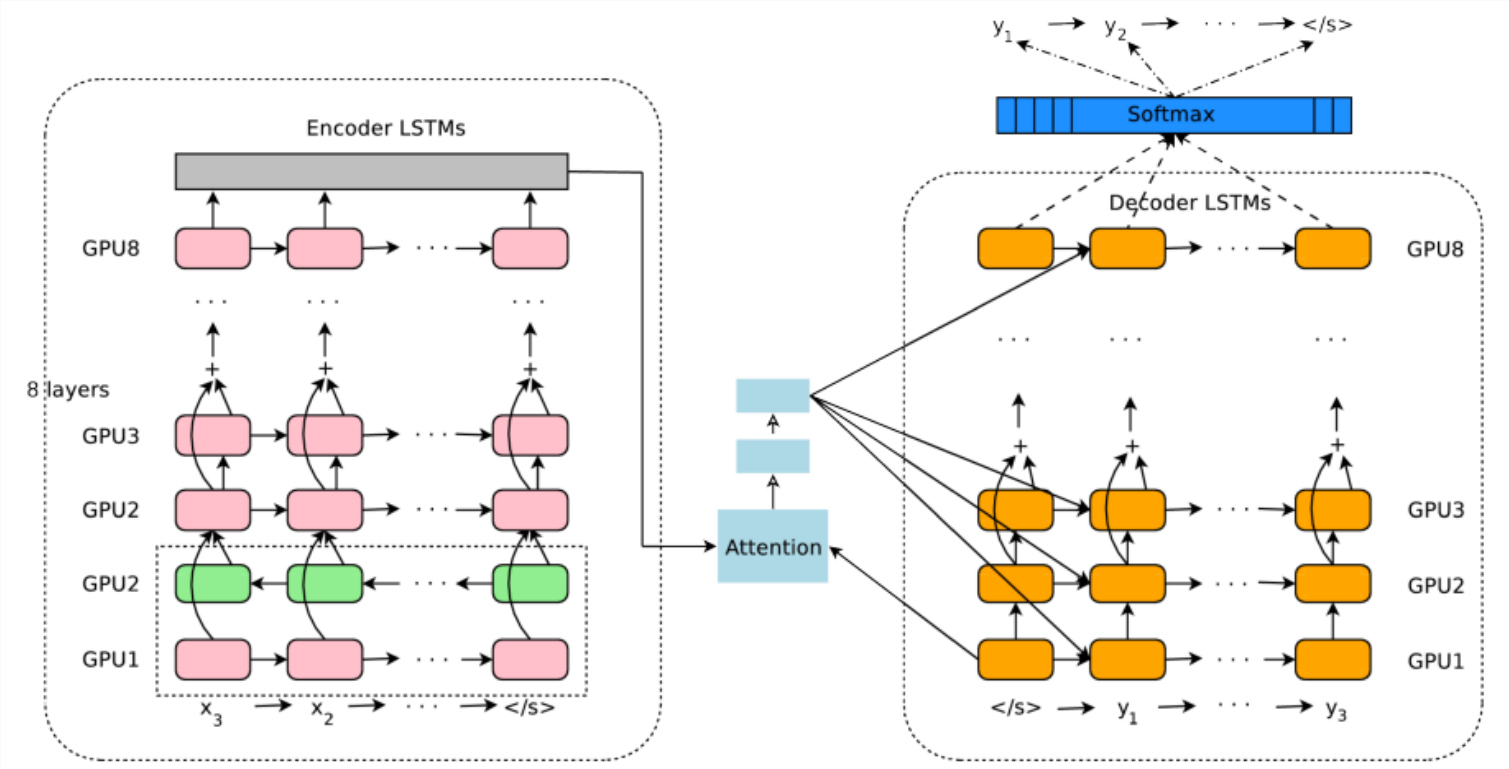
Внимание в нейронных сетях

- Другая необычная работа – «Grammar as a Foreign Language» (Vinyals et al., 2015)



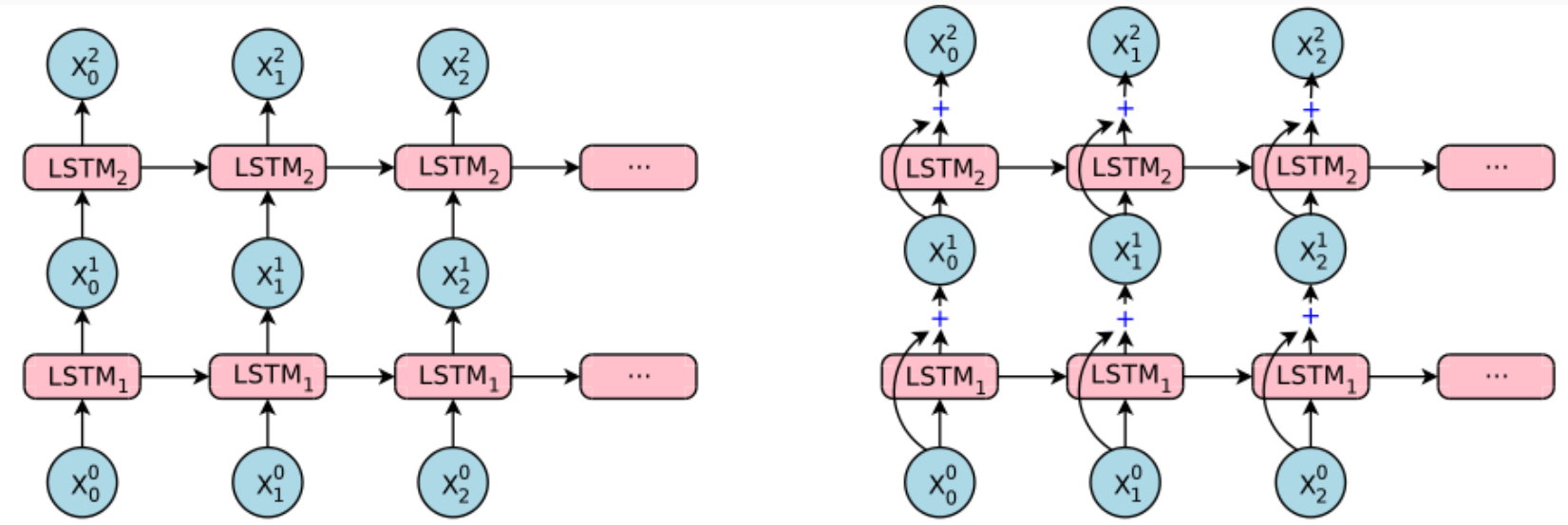
Google Translate

- Сентябрь 2016: Wu et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*:
  - как на самом деле работает Google Translate;
  - базовая архитектура та же самая: encoder, decoder, attention;
  - RNN глубокие, по 8 уровней в encoder и decoder:



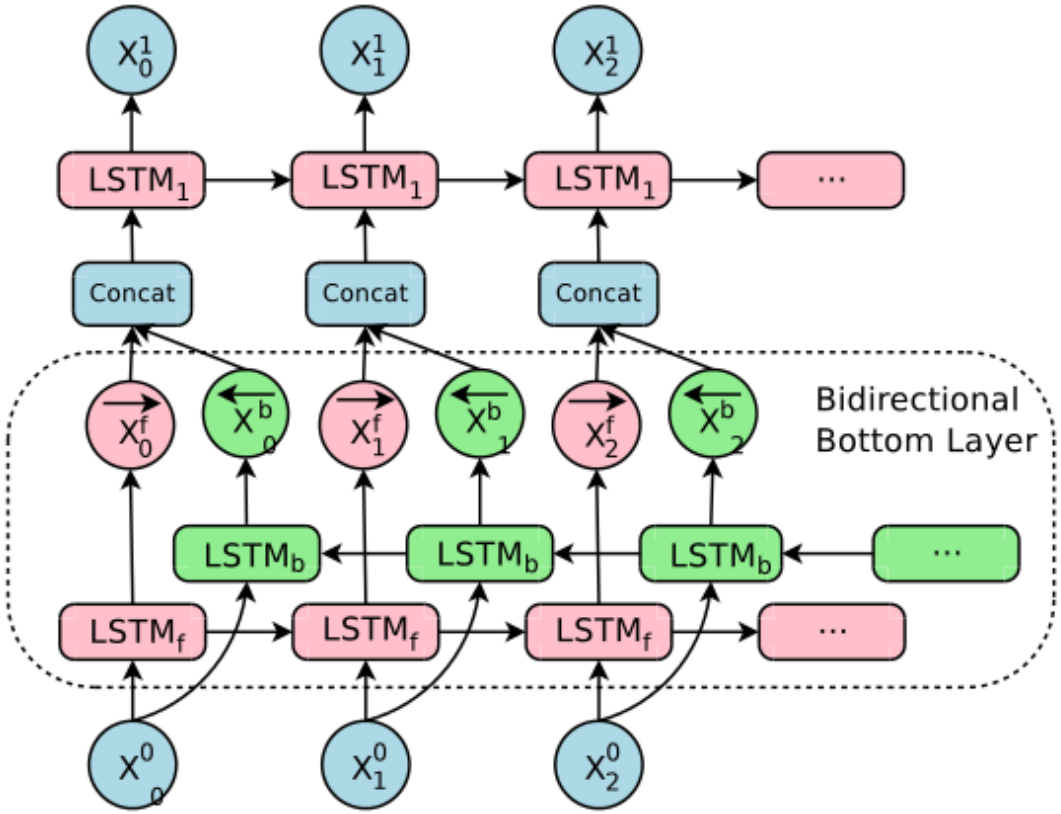
Google Translate

- Сентябрь 2016: Wu et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*:
  - просто stacked LSTM перестают работать далее 4-5 уровней;
  - поэтому добавляют остаточные связи, как в ResNet:



Google Translate

- Сентябрь 2016: Wu et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*:
  - нижний уровень, естественно, двунаправленный:



Google Translate

- Сентябрь 2016: Wu et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*:
- в GNMT ещё две идеи о сегментации слов:
  - *wordpiece model*: разбить слова на кусочки (отдельной моделью); пример из статьи:

    `Jet makers feud over seat width with big orders at stake`

    превращается в

    `_J et _makers _fe ud _over _seat _width _with _big _orders _at _stake`

  - *mixed word/character model*: конвертировать слова, не попадающие в словарь, в последовательность букв-токенов; пример из статьи:

    `Miki` превращается в `<B>M <M>i <M>k <E>i`

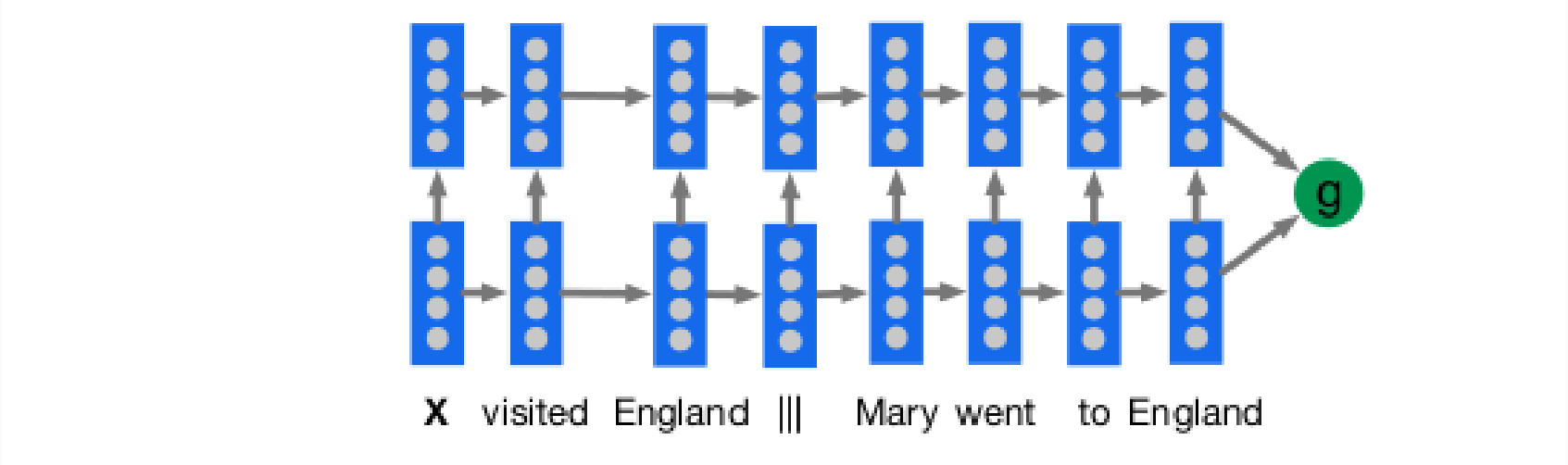
Teaching machines to read

- (Hermann et al., 2015): «Teaching machines to read and comprehend» (Google DeepMind)
- Предлагают новый способ построить датасет для понимания, автоматически создавая тройки (context, query, answer) из текстов новостей и т.п.

Original Version	Anonymised Version
Context	
The BBC producer allegedly struck by Jeremy Clarkson will not press charges against the "Top Gear" host, his lawyer said Friday. Clarkson, who hosted one of the most-watched television shows in the world, was dropped by the BBC Wednesday after an internal investigation by the British broadcaster found he had subjected producer Oisin Tymon "to an unprovoked physical and verbal attack." ...	the *ent381* producer allegedly struck by *ent212* will not press charges against the " *ent153* " host , his lawyer said friday . *ent212* , who hosted one of the most - watched television shows in the world , was dropped by the *ent381* wednesday after an internal investigation by the *ent180* broadcaster found he had subjected producer *ent193* " to an unprovoked physical and verbal attack . " ...
Query	
Producer **X** will not press charges against Jeremy Clarkson, his lawyer says.	producer **X** will not press charges against *ent212* , his lawyer says .
Answer	
Oisin Tymon	*ent193*

Teaching machines to read

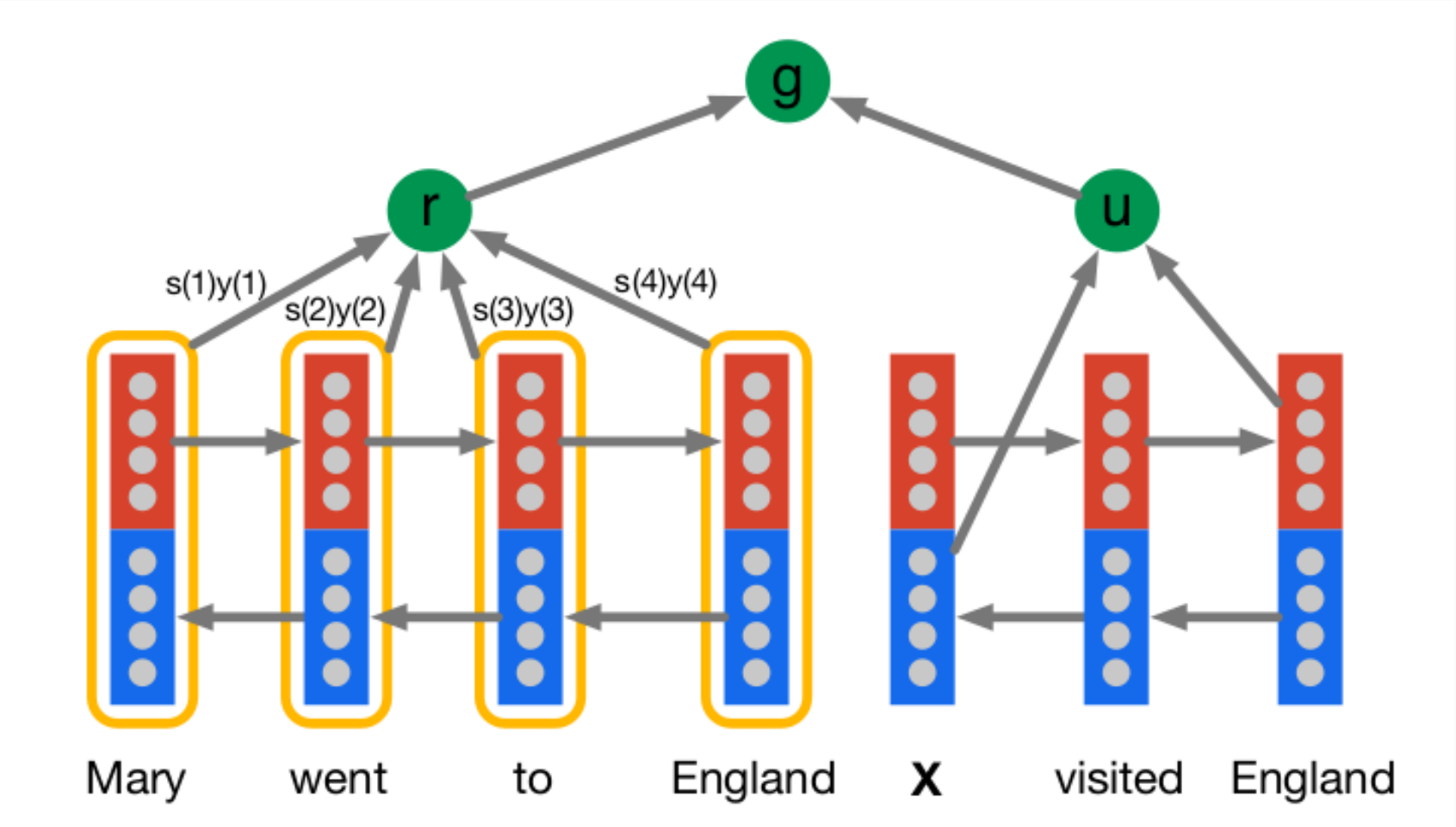
- Базовая модель – глубокий LSTM:



- Но так, конечно, совсем плохо работает.

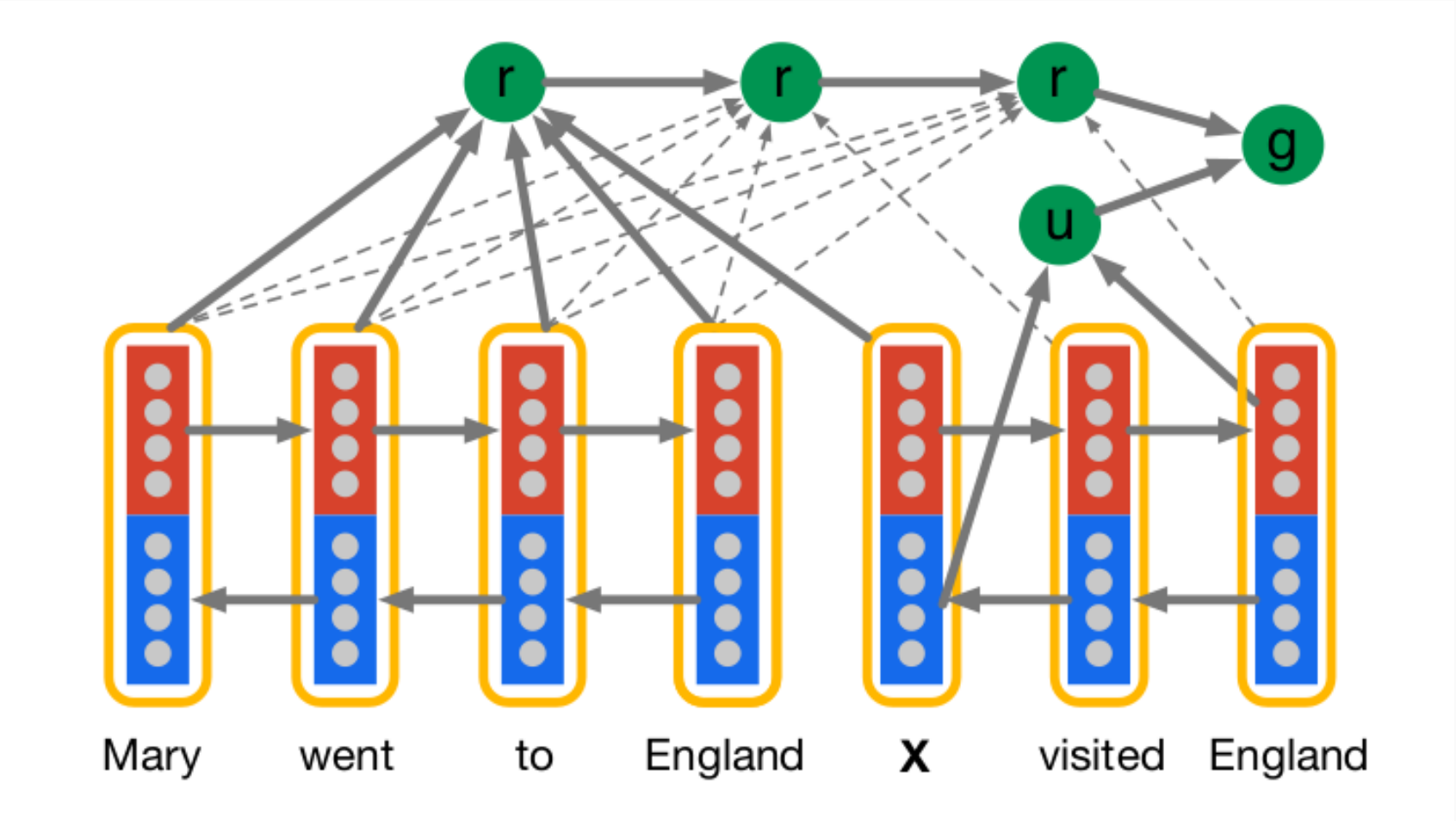
Teaching machines to read

- Attentive Reader: обучаемся, на какую часть документа смотреть



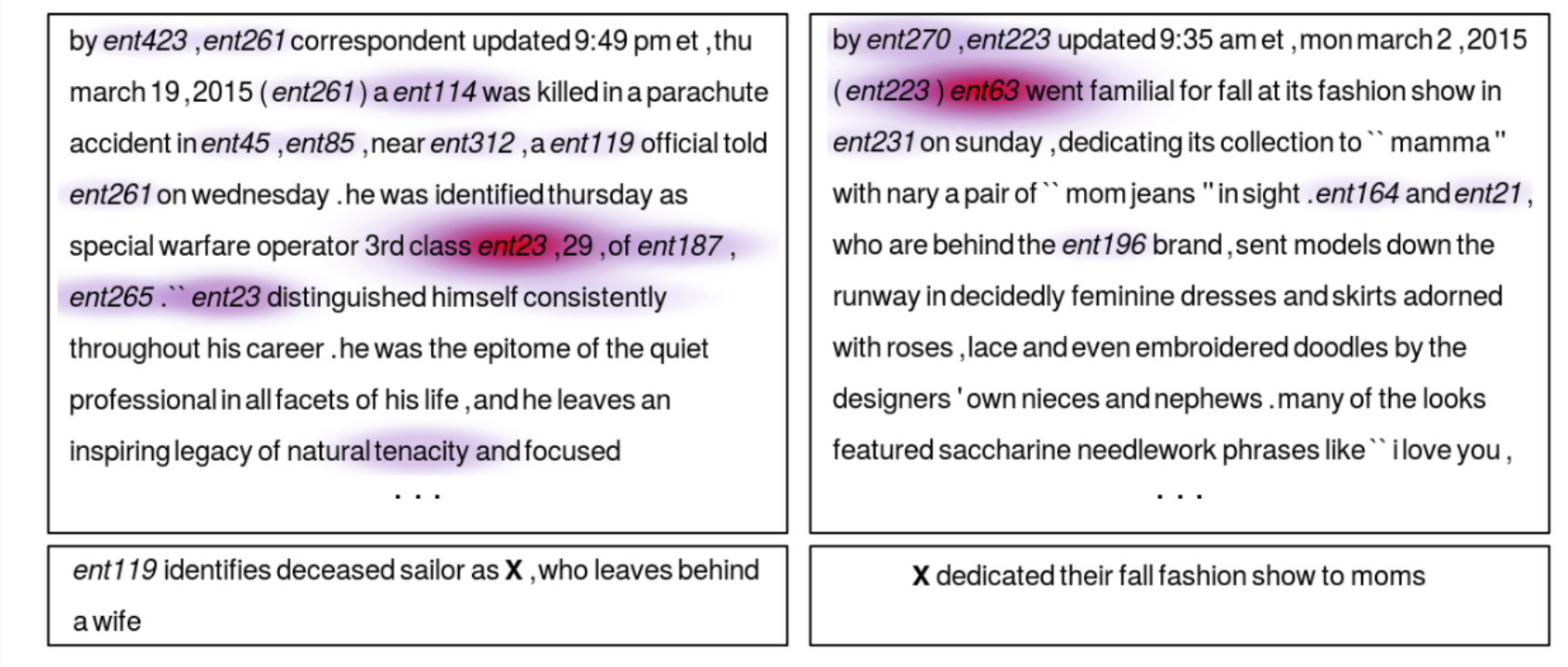
Teaching machines to read

- Impatient Reader: можем перечитывать нужные части документа по мере прочтения запроса



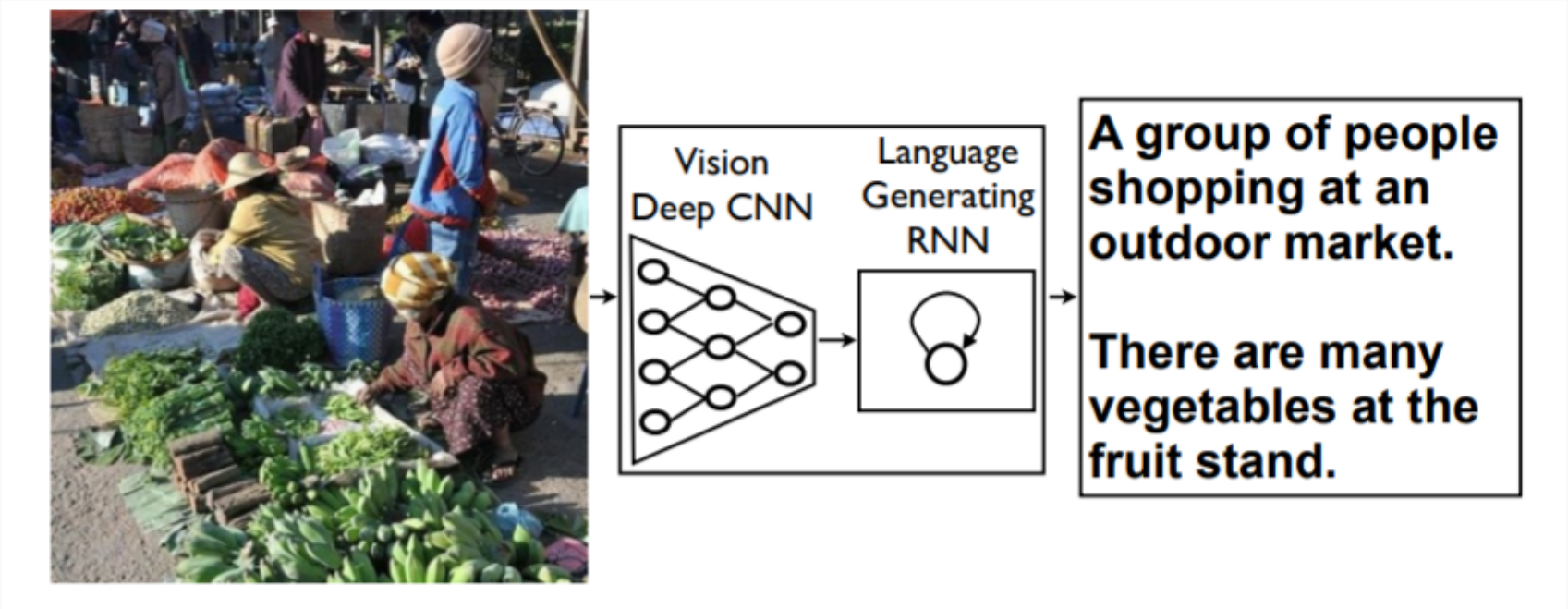
Teaching machines to read

- Получаются разумные карты внимания:



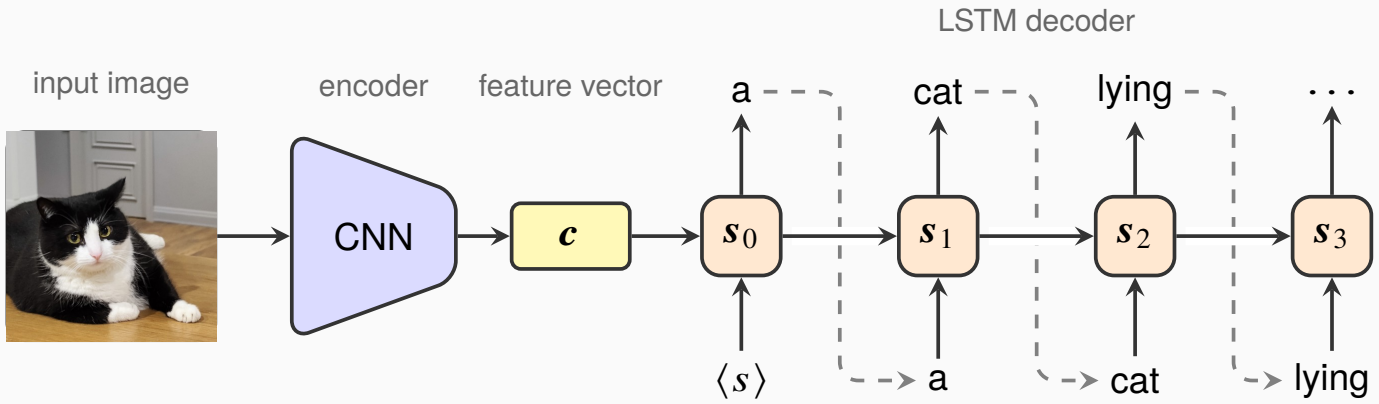
Show, Attend, and Tell

- Теперь давайте про подписи к картинкам.
- Сначала было «Show and Tell» (Vinyals et al., 2015):



Show, Attend, and Tell

- Довольно прямолинейная архитектура:
  - целевая функция – это просто $\sum_{(I,S)} \log p(S \mid I; \theta)$, где I – картинка, S – описание;
  - раскладываем и моделируем $p(S_t \mid I, S_0, \ldots, S_{t-1})$ рекуррентной сетью с LSTM;
  - а CNN используем, чтобы извлечь признаки.



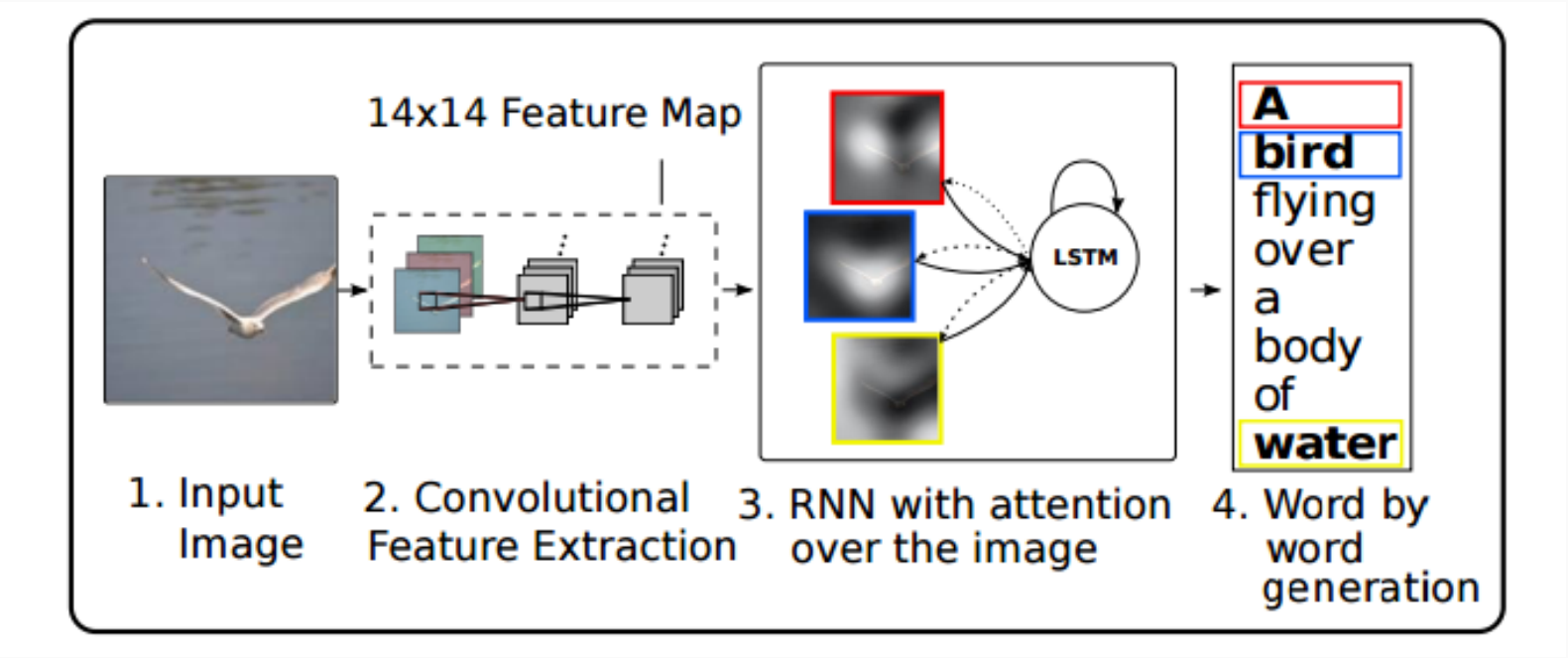
Show, Attend, and Tell

- Получалось хорошо, но можно лучше:



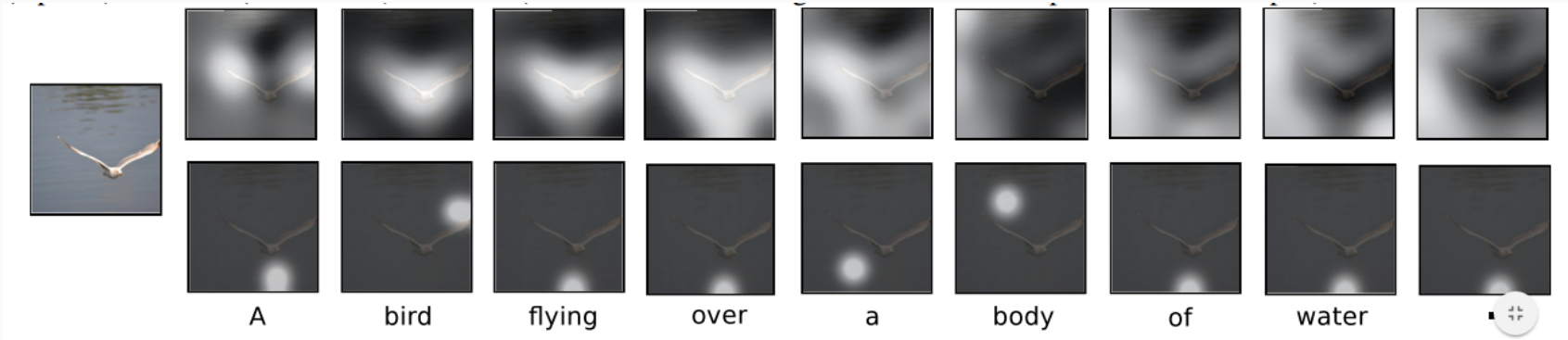
Show, Attend, and Tell

- Из этого появилось «Show, Attend, and Tell» (Xu et al., 2015)



Show, Attend, and Tell

- Soft attention vs. hard attention (стохастически выбираем однозначный кусок картинки).



- Soft attention – строим аннотацию с весами

$$\phi(\{\mathbf{a}\}_i, \{\alpha_i\}) = \sum_{i=1}^{L} \alpha_{t,i} \mathbf{a}_i.$$

Show, Attend, and Tell

- Hard attention обучается максимизацией вариационной нижней оценки

$$L_s = \sum_s p(s \mid \mathbf{a}) \log p(\mathbf{y} \mid s, \mathbf{a}) \leq \log \sum_s p(s \mid \mathbf{a}) p(\mathbf{y} \mid s, \mathbf{a}) = \log p(\mathbf{y} \mid \mathbf{a}).$$

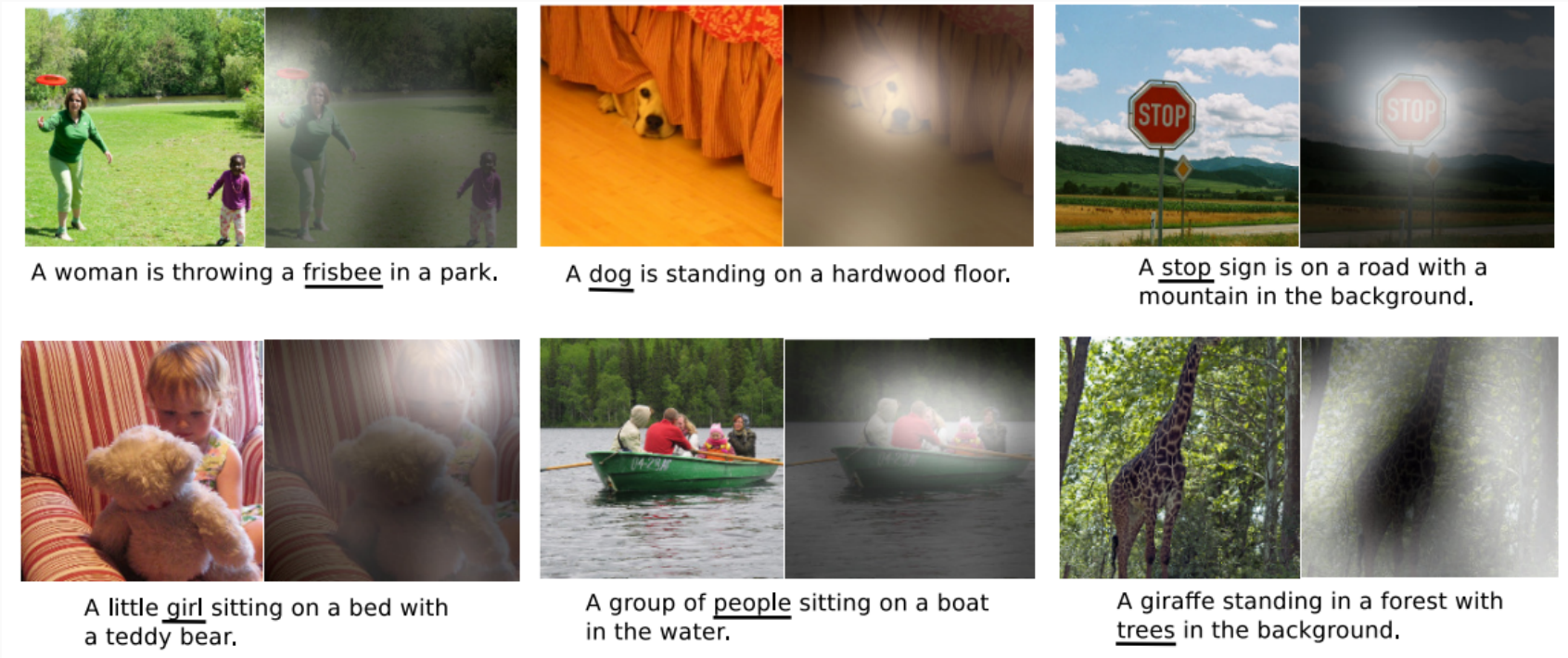
- От L_s можно брать производные:

$$\frac{\partial L_s}{\partial W} = \sum_s p(s \mid \mathbf{a}) \left[\frac{\partial \log p(\mathbf{y} \mid s, \mathbf{a})}{\partial W} + \log p(\mathbf{y} \mid s, \mathbf{a}) \frac{\partial \log p(s \mid \mathbf{a})}{\partial W} \right].$$

- И дальше сэмплируем s_t с вероятностями α_i и приближаем ожидание выборкой.
- Опять те же трюки, вычитаем baseline, всё такое.

Show, Attend, and Tell

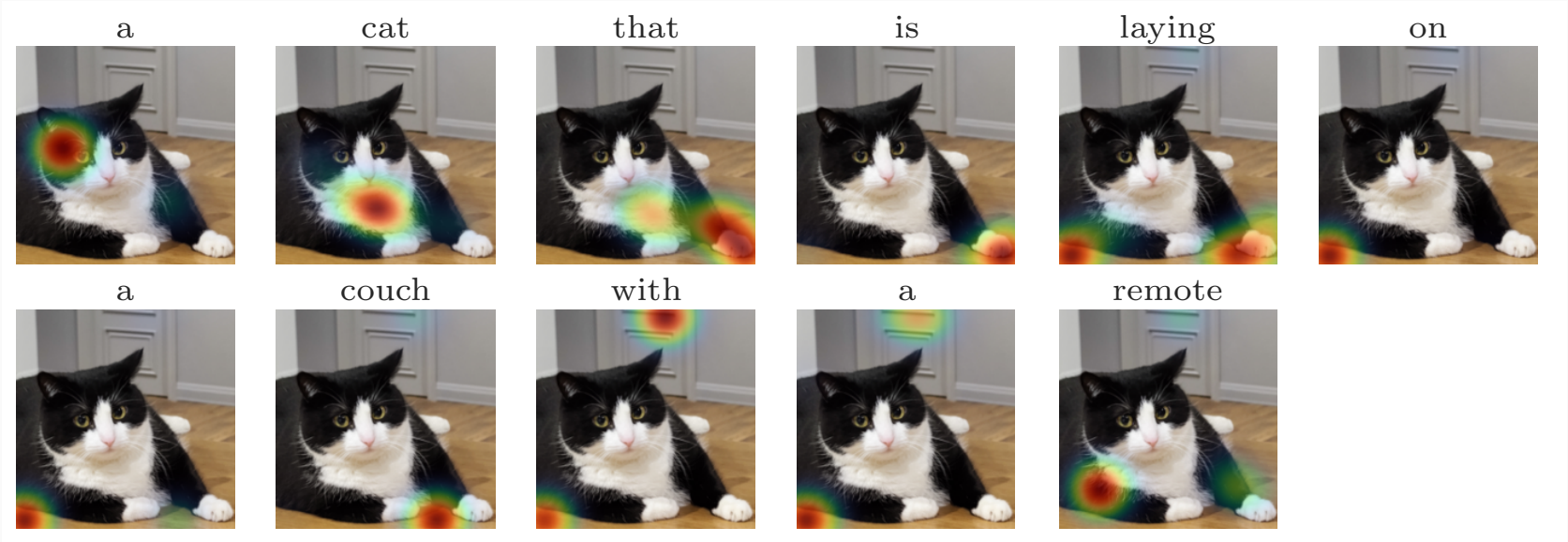
- Часто получаются очень хорошие результаты:



- А когда плохие, можно посмотреть почему.

Show, Attend, and Tell

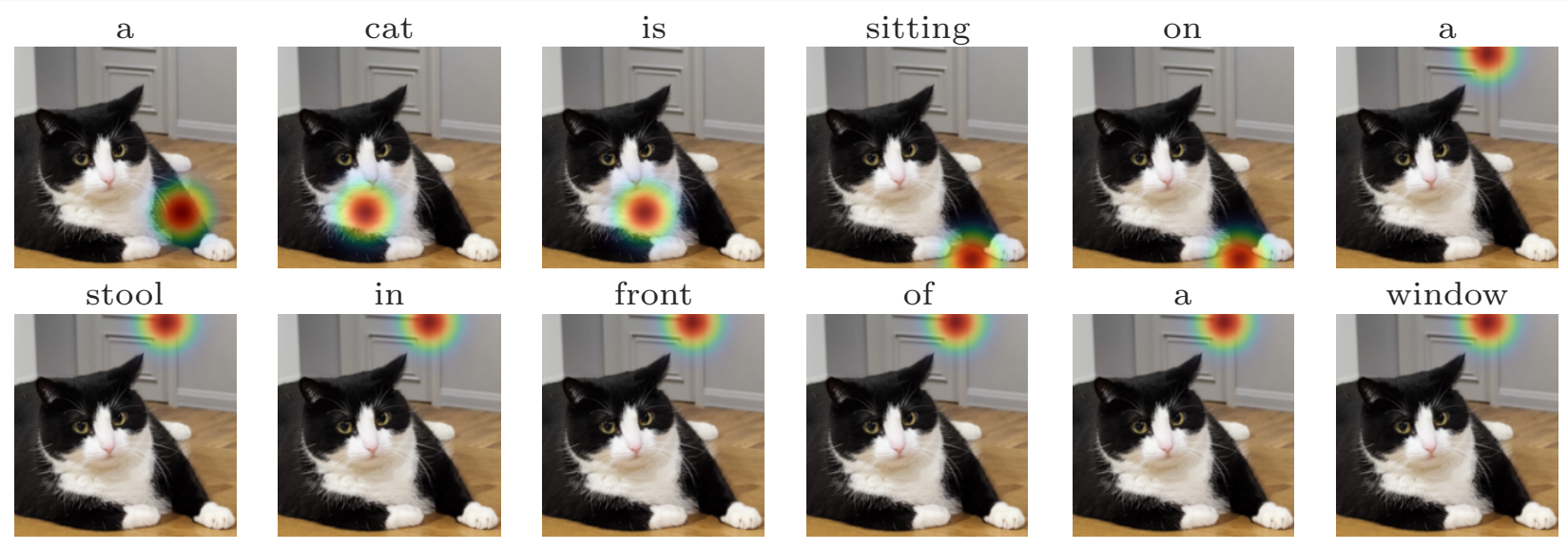
- Обучим Show, Attend and Tell сами (замороженный VGG-16, LSTM с аддитивным вниманием, часть MS-COCO) и покажем ему моего кота.
- Soft attention: где декодер смотрит при каждом слове:



- Кота модель видит, а «couch» и «remote» выдумывает; на слове «cat» внимание собирается на животном.

Show, Attend, and Tell

- Hard attention, обучение через REINFORCE: на каждом шаге одно выбранное место.



- Подпись не лучше, чем у soft attention (оптимизировать труднее), но внимание заметно сосредоточеннее.

Show, Attend, and Tell

- Show and Tell (ST) против Show, Attend and Tell (SAT) на одних и тех же данных:



ST: a black and white cat sitting in front of a white sink
SAT: a cat laying on a couch with a remote



ST: a cat laying on top of a white chair with a hat
SAT: a cat laying on a wooden floor with a pair of shoes



ST: a cat laying on top of a laptop on a bed
SAT: a cat laying on a wooden floor next to a piece of luggage

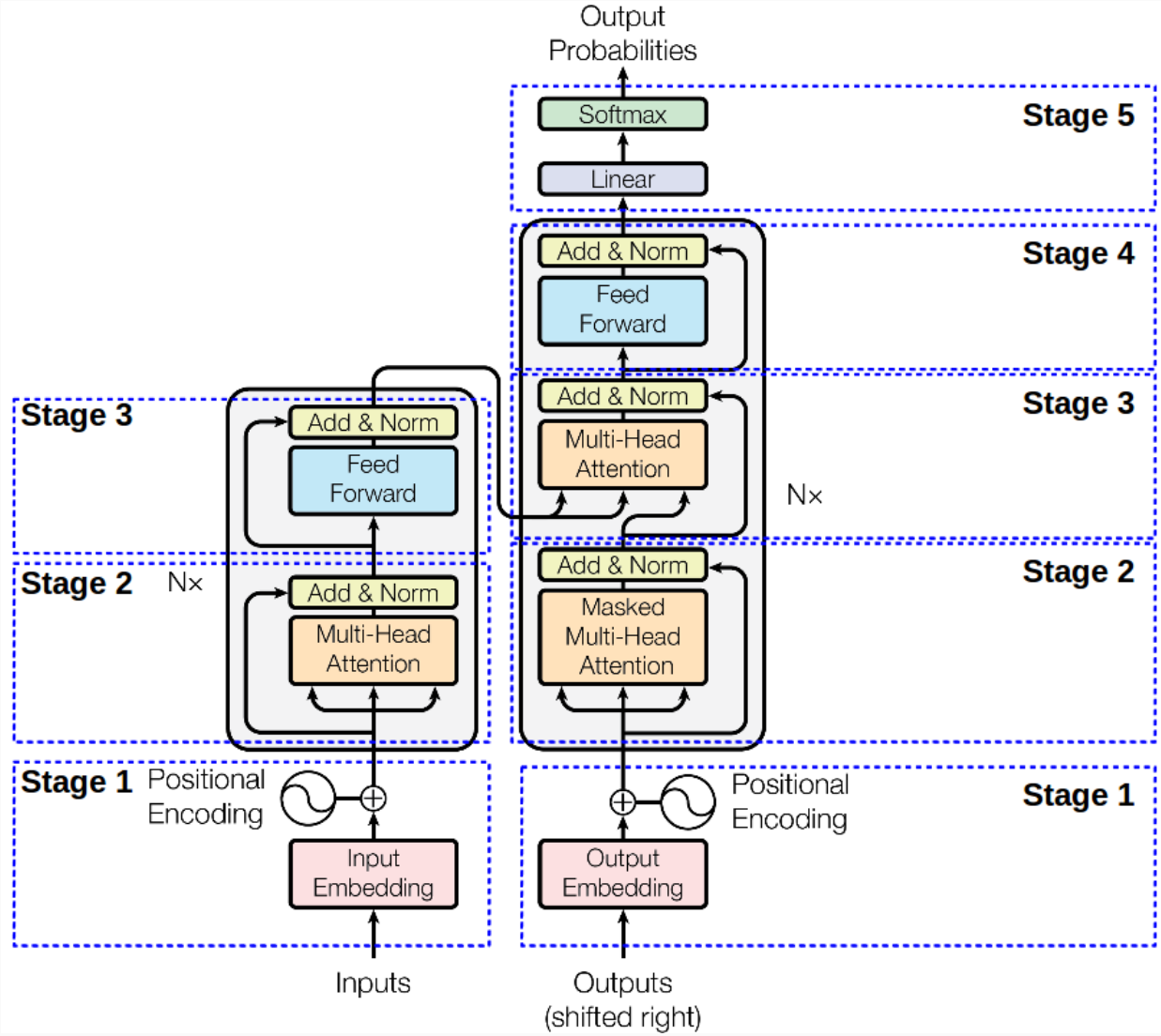
Самовнимание и Transformer

Transformer

- Мы изучали перевод на рекуррентных сетях и дошли до архитектуры Google NMT
- Но в 2017 году оказалось, что всё может быть ещё проще и интереснее
- Google: «Attention is all you need» (Vaswani et al., 2017)
- Основная идея – self-attention; оказывается очень плодотворной для всевозможных seq2seq задач
- Главная мотивация – попробовать всё-таки уйти от кодирования вектором постоянной длины

Transformer

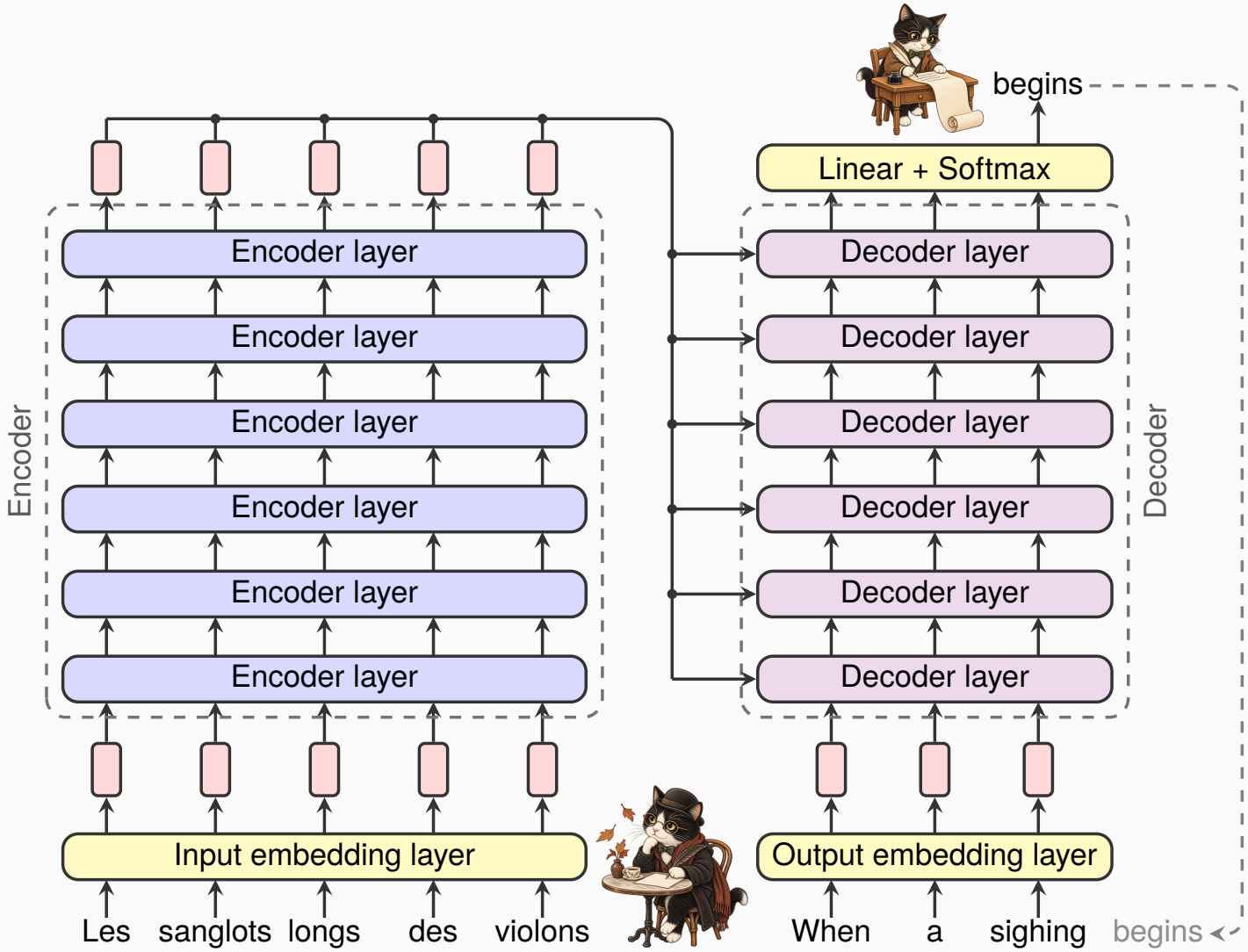
- Общая схема:



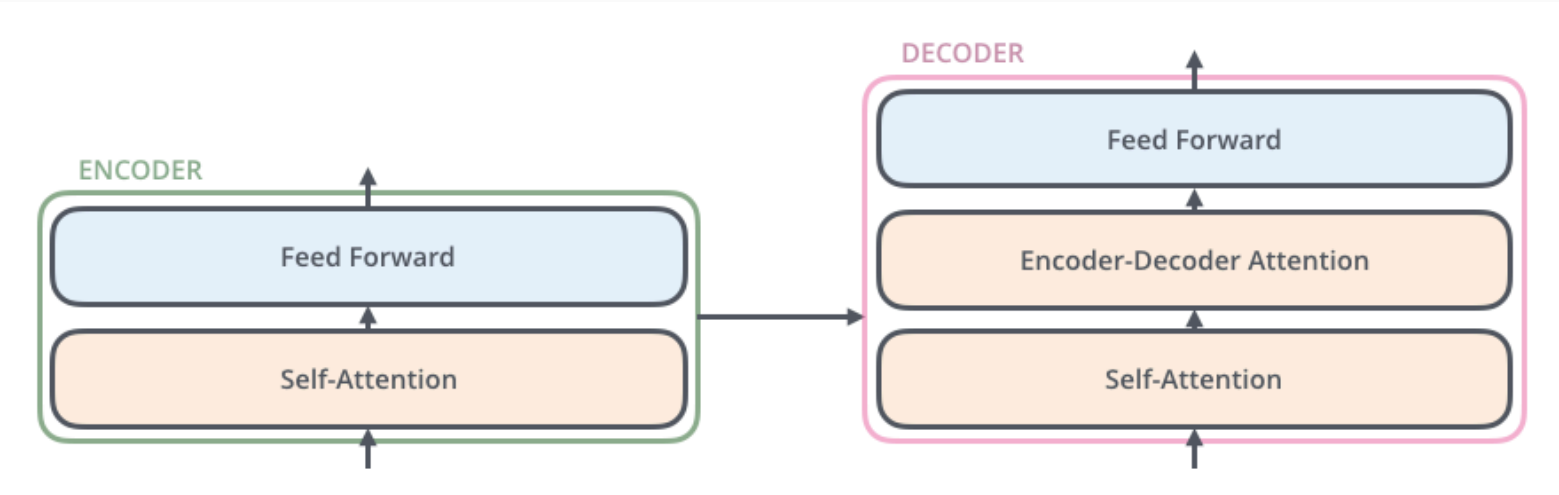
- Теперь подробнее...

Transformer

- Суть, как и раньше, – encoder-decoder:

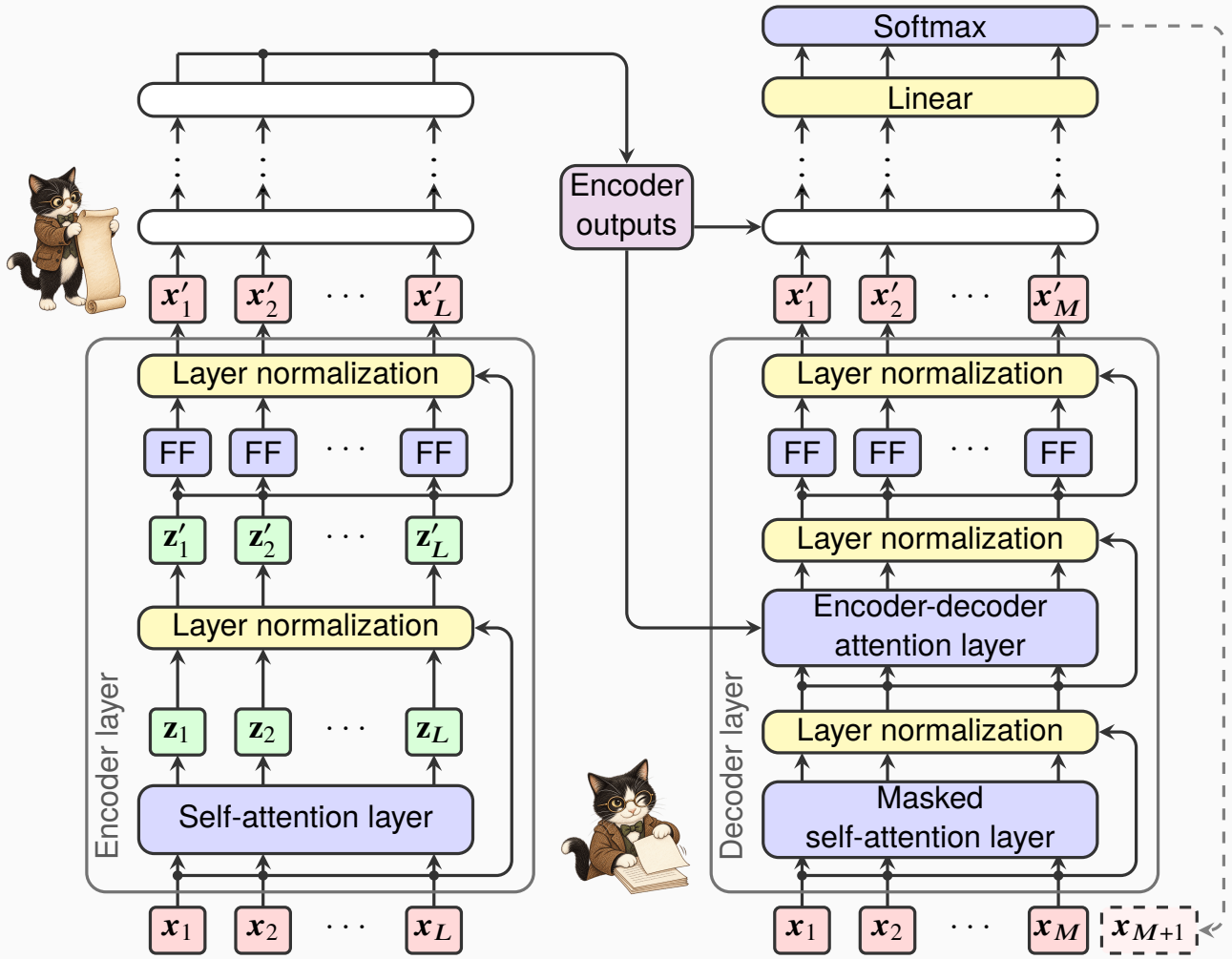


- В каждом слое – слой self-attention, а потом feedforward layer, который независимо применяется к каждой позиции входа. У декодера ещё есть attention между ними:



Transformer

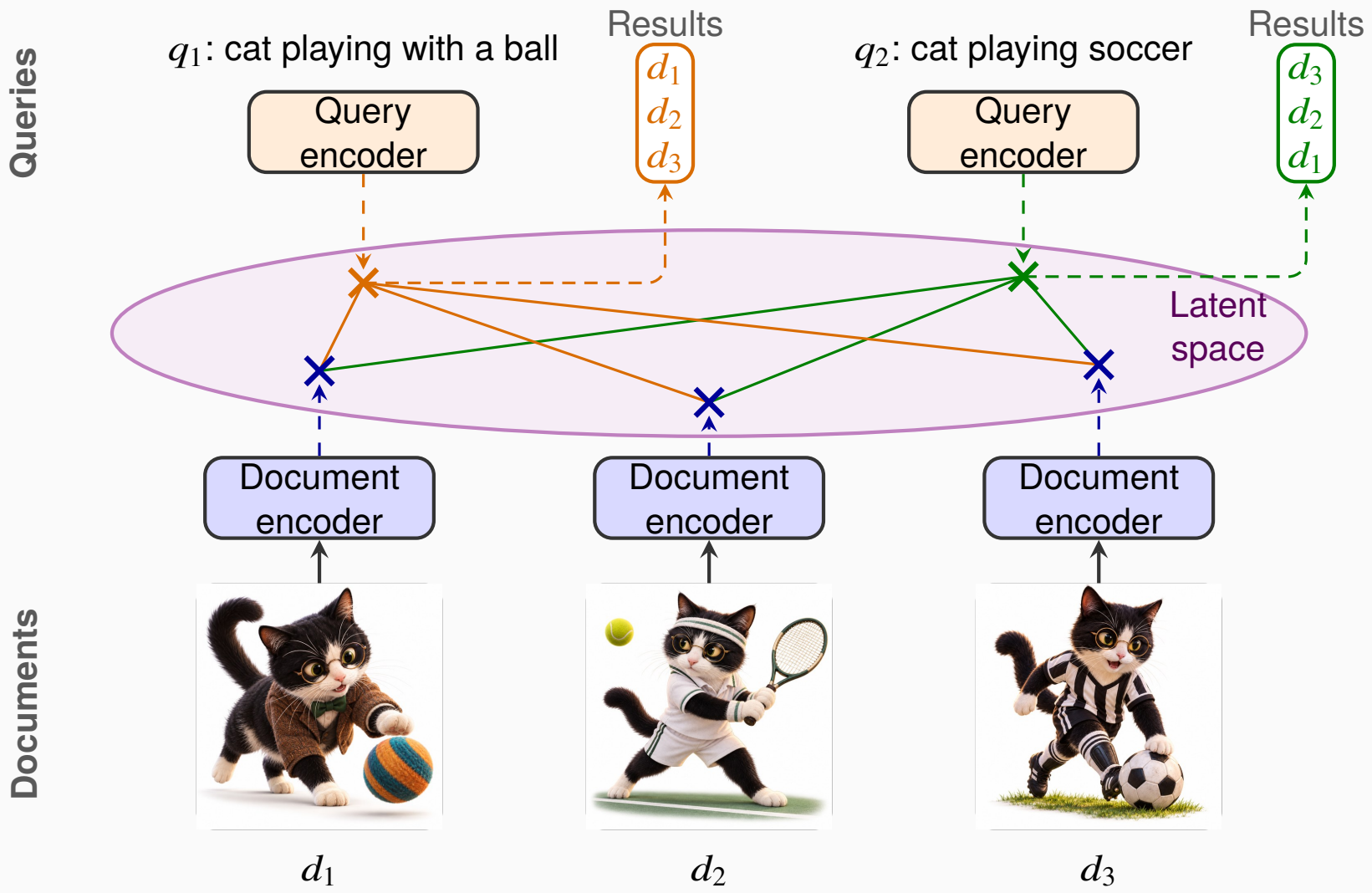
- Слова, естественно, представляются векторами, в feedforward слое всё параллельно:



- Но что же это такое – self-attention?

Self-attention

- Чтобы понять самовнимание (self-attention), начнём с информационного поиска (information retrieval)



Self-attention

- Запросы и документы отображаются в одно и то же латентное пространство (но разными кодировщиками — это могут быть вообще объекты разной природы)
- Запрос идёт через query encoder
- Документы (на иллюстрации это картинки) — через другой encoder, но в то же пространство
- Чтобы найти самые релевантные документы, мы ищем ближайших соседей в латентном пространстве среди документов; часто предполагают, что латентное пространство линейно, и расстояние там — это просто скалярное произведение $\mathrm{dist}(q, d) = \mathrm{Enc}_q(q)^\top \mathrm{Enc}_d(d)$.

Self-attention

- В самовнимании эта интуиция используется очень абстрактно. Слой получает на вход последовательность векторов $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L$, т.е. матрицу $X \in \mathbb{R}^{d \times L}$.
- Запросы, ключи и документы берутся из самих $\mathbf{x}_i$:
  - умножая $\mathbf{x}_1$ на матрицу весов W^Q, мы получаем вектор *запроса* $\mathbf{q}_1 = W^Q \mathbf{x}_1$; обратите внимание, что размерность q векторов запросов, $\mathbf{q}_i \in \mathbb{R}^q$, может отличаться (обычно меньше) от входной размерности d, $\mathbf{x}_i \in \mathbb{R}^d$;
  - умножая $\mathbf{x}_i$ на матрицу весов W^K, мы получаем векторы *ключей* $\mathbf{k}_i = W^K \mathbf{x}_i$ для $i = 1, \dots, L$; поскольку мы хотим, чтобы запросы и ключи находились в одном латентном пространстве, ключи имеют ту же размерность, что и запросы, $\mathbf{k}_i \in \mathbb{R}^q$, таким образом $W^K \in \mathbb{R}^{q \times d}$;
  - третья матрица весов W^V даёт векторы *значений* $\mathbf{v}_i = W^V \mathbf{x}_i$ для $i = 1, \dots, L$; это те *документы*, которые мы будем «извлекать» с помощью ключей $\mathbf{k}_i$; формально мы имеем другую размерность v для значений, $\mathbf{v}_i \in \mathbb{R}^v$ и $W^V \in \mathbb{R}^{v \times d}$; на практике обычно $v = q$.

Self-attention

- Затем мы выполняем поиск, вычисляя оценки внимания как скалярные произведения между запросами и документами
- Нормализуем $\mathbf{q}_i^\top \mathbf{k}_j$, деля на $\sqrt{q}$, и получаем оценки через softmax, чтобы преобразовать их в вероятности:

$$\alpha_{ij} = \mathrm{softmax}\left(\frac{1}{\sqrt{q}}\mathbf{q}_i K^\top\right)_j,$$

где $K \in \mathbb{R}^{q \times L}$ — это все ключи, объединенные в матрицу, $K = W^K X$.

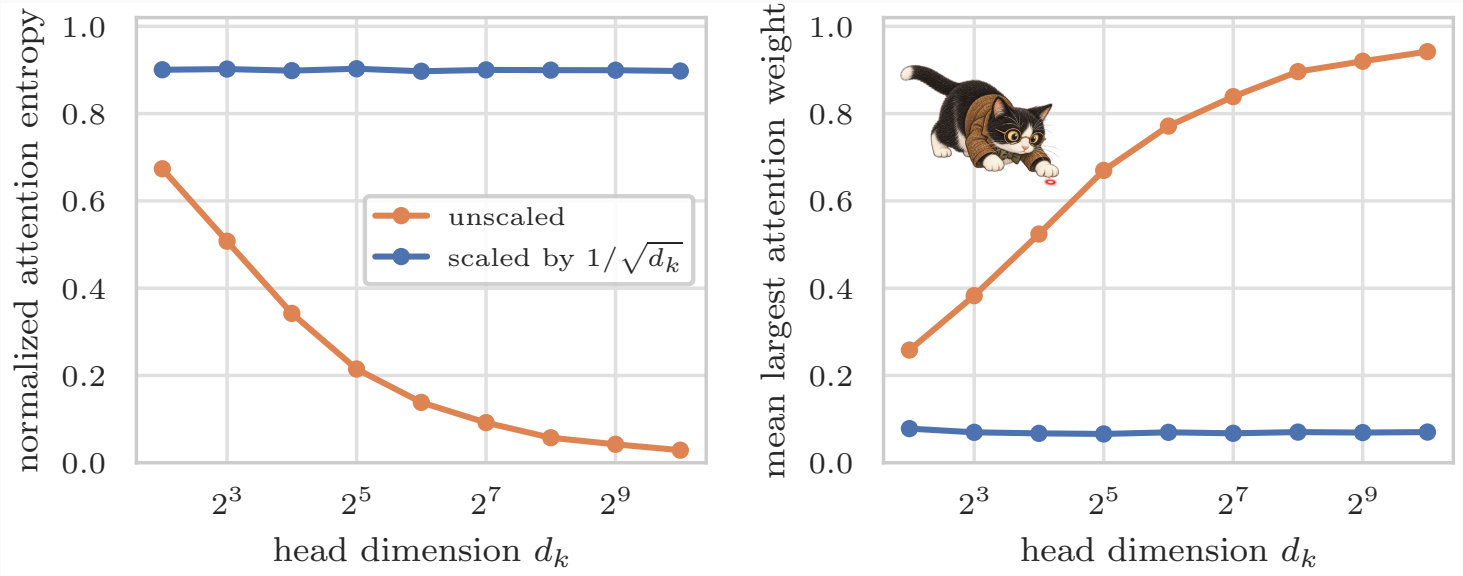
- Затем используем результат в качестве коэффициентов для выпуклой комбинации *значений* $\mathbf{v}_j$:

$$\mathbf{z}_i = \mathrm{softmax}\left(\frac{1}{\sqrt{q}}\mathbf{q}_i K^\top\right) V,$$

где $V \in \mathbb{R}^{q \times L}$ — значения (values), $V = W^V X$.

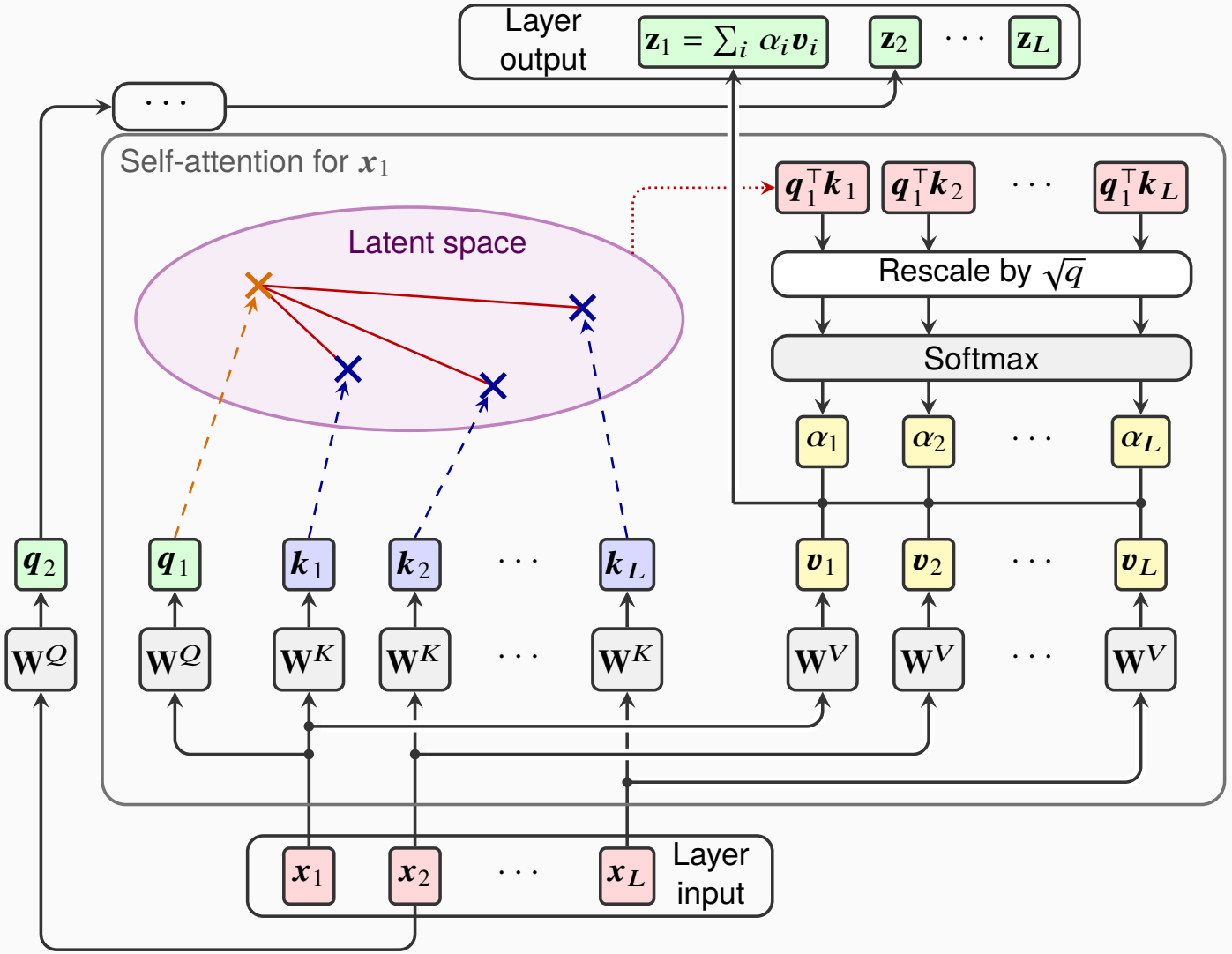
Self-attention

- Зачем делить на $\sqrt{q}$? Скалярное произведение $\mathbf{q}_i^\top \mathbf{k}_j = \sum_{r=1}^{q} q_{ir} k_{jr}$ — сумма q примерно независимых слагаемых с дисперсией около 1, так что оно порядка $\sqrt{q}$.
- Без нормировки softmax насыщается, и внимание превращается в почти детерминированный поиск; слева — нормированная энтропия весов, справа — самый большой вес:



Self-attention

- Общая структура слоя самовнимания (self-attention):



Self-attention

- И это вся интуиция! Мы можем объединить вычисления каждого $\mathbf{z}_i$ в одну формулу в матричном виде:

$$Z = \mathrm{softmax}\left(\frac{1}{\sqrt{q}} QK^\top\right) V.$$

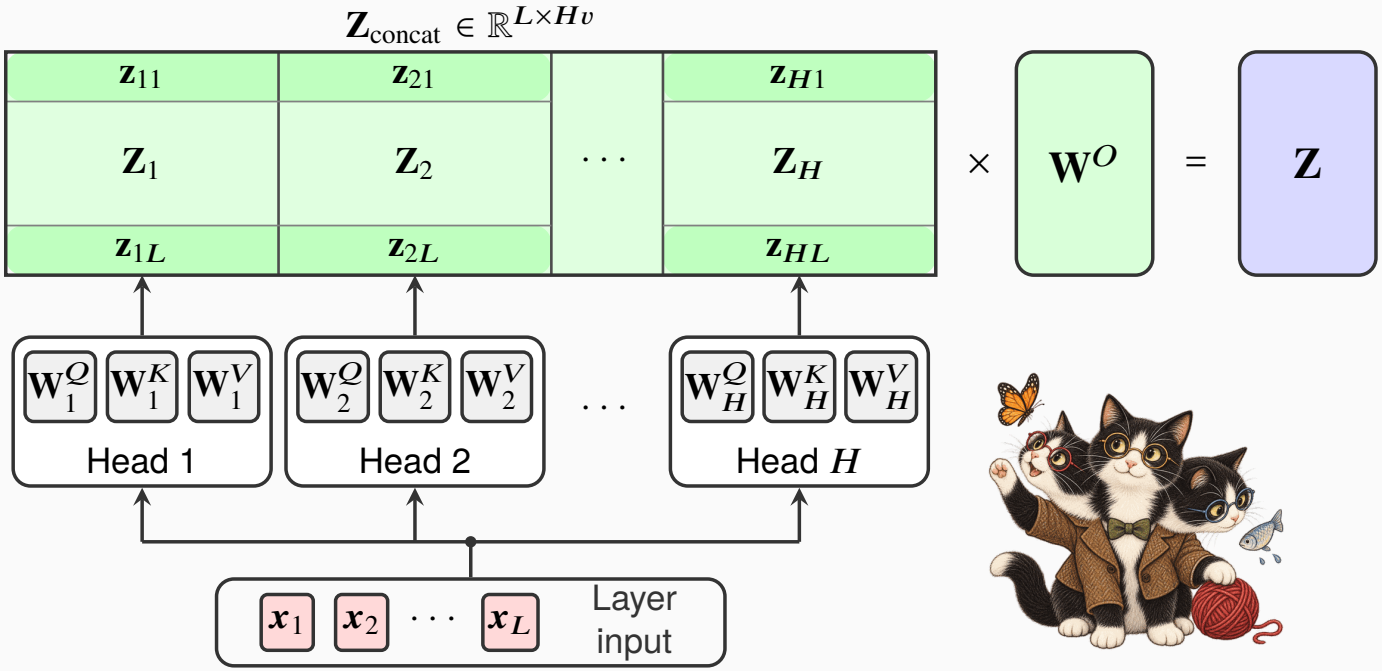
- Но это только один способ «смотреть» на входные векторы; то, что мы сейчас определили, — это не полный слой самовнимания, а только одна *голова* (self-attention head)
- Можно распараллелить эти вычисления вдоль H различных голов, используя разные матрицы весов $W_1^Q, \dots W_H^Q$, $W_1^K, \dots W_H^K$ и $W_1^V, \dots W_H^V$. Это *многоголовое внимание* (multi-head attention)

Self-attention

- После H параллельных голов мы получаем матрицы выходов $Z_1, Z_2, \ldots, Z_H$ размерности $v \times L$, объединяем их все в $Z_{\text{concat}} \in \mathbb{R}^{L \times Hv}$ и добавляем ещё одну матрицу весов $W^O \in \mathbb{R}^{Hv \times d}$, чтобы сжать результат обратно до необходимой размерности:
$$Z = \left(Z_{\text{concat}} W^O\right)^\top.$$
- С помощью W^O слой самовнимания может комбинировать представления, полученные из различных голов внимания; это также важный способ добавить гибкость и выразительность в архитектуру.

Self-attention

- Многоголовое внимание:



- Это самая важная часть, но это ещё не всё.

Два вида смешивания: токены и каналы

- Блок трансформера удобно понимать как чередование двух операций:
  - *внимание смешивает токены*: каждая позиция собирает информацию из других;
  - *FFN смешивает каналы*: каждая позиция обрабатывается отдельно, $d \to 4d \to d$.
- С нормализацией N и остаточными связями:

$$A = X + \mathrm{MHA}(N(X)), \qquad Y = A + \mathrm{FFN}(N(A)).$$

- Одного внимания мало: без остаточных связей и FFN стопка слоёв самовнимания вырождается в матрицу ранга 1, причём дважды экспоненциально быстро (Dong et al., 2021).

Два вида смешивания: токены и каналы

- Где параметры? При $d_{\text{ff}} = 4d$ у FFN около $8d^2$ весов, у W^Q, W^K, W^V, W^O — около $4d^2$; большая часть модели — в FFN.
- Где вычисления? Проекции — $4Ld^2$, $QK^\top$ и умножение на V — $2L^2d$, FFN — $8Ld^2$; квадратичная часть доминирует, только когда L в несколько раз больше d.
- А вот с памятью хуже: наивная реализация хранит HL^2 весов внимания (к этому мы ещё вернёмся).
- Удобный взгляд — *residual stream*: каждый слой читает из общего потока и пишет в него. Голова внимания — маленькая ассоциативная память: запросы и ключи решают, *откуда* читать, значения — *что* скопировать.

Декодер

- Декодер включает два дополнительных типа слоёв: *маскированное самовнимание* (masked self-attention) и *encoder-decoder attention*
- Маскированное внимание в основном означает, что, поскольку декодер работает авторегрессионно, он не должен смотреть на токены, которые ещё не породил
- Для этого проще всего ввести всю последовательность и замаскировать будущие позиции внутри слоёв самовнимания
- Формально это означает, что мы задаем аргументы softmax как $-\infty$ для будущих токенов, что означает, что их веса внимания всегда будут нулевыми

Декодер

- Encoder-decoder attention — это вариация механизма самовнимания, которая опирается на результаты кодировщика
- Звучит сложно... но на самом деле это почти тривиальное изменение! Мы используем разные векторы в качестве входных данных в том же самовнимании:
  - для создания запросов используем векторы из предыдущего слоя, то есть текущие представления уже порождённых выходных токенов;
  - но для «документов» в задаче «поиска», то есть для векторов ключей и значений, мы используем векторы из энкодера
- Неформально это значит, что мы выполняем «поиск» на выходе энкодера с запросами, состоящими из уже порождённых токенов
- Формально все размерности хорошо совпадают: в аргументе softmax есть L элементов для каждого вектора, но число запросов и, следовательно, число выходов совпадает с числом входов

Декодер

- Обучается декодер параллельно, хотя модель авторегрессионная: подаём сдвинутую последовательность $\langle \mathrm{BOS} \rangle, y_1, \ldots, y_{T-1}$ и предсказываем $y_1, \ldots, y_T$ сразу во всех позициях (teacher forcing).
- Маскированное внимание — одна треугольная матрица:

$$\mathrm{Attn}_{\mathrm{causal}}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{q}} + M\right) V,$$

$$M_{ij} = 0 \text{ при } j \le i, \qquad M_{ij} = -\infty \text{ при } j > i.$$

- А порождение — честно по одному токену; ключи и значения префикса хранятся в *KV-кэше*, чтобы не пересчитывать их на каждом шаге.
- Тонкость: при обучении префикс написан человеком, а при порождении — самой моделью, и ранняя ошибка остаётся в контексте (exposure bias).

Токенизация

- Мы используем вложения (embeddings), но как мы делим текст на токены?
- *Byte-pair encoding* (BPE): старая идея из сжатия данных (Gage, 1994), которую для перевода приспособили Sennrich et al. (2016): жадно склеиваем самую частую пару символов
- Давайте начнем с построения словаря:

$$\{\text{cat} : 10, \text{pet} : 12, \text{mat} : 5, \text{rat} : 8, \text{eats} : 4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4



Токенизация

- Затем разделим его на символы и подсчитаем пары символов:

$$\{\text{cat} : 10, \text{pet} : 12, \text{mat} : 5, \text{rat} : 8, \text{eats} : 4\},$$

$$\{c : 10, a : 27, t : 39, p : 12, e : 16, m : 5, r : 8, s : 4\},$$

$$\{ca : 10, at : 27, pe : 12, et : 12, ma : 5, ra : 8, ea : 4, ts : 4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4



Токенизация

- Теперь возьмем наиболее частую пару («*at*») и перекодируем её в новый символ (Z), который добавляется в словарь:

$$\{\mathrm{cZ}:10, \mathrm{pet}:12, \mathrm{mZ}:5, \mathrm{rZ}:8, \mathrm{eZs}:4\},$$
$$\{c:10, Z:27, t:12, p:12, e:16, m:5, r:8, s:4\},$$
$$\{cZ:10, pe:12, et:12, mZ:5, rZ:8, eZ:4, Zs:4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4



Токенизация

- Можно теперь выбрать самую частую пару — *pe* или *et* — и заменить её другим новым символом *Y*:

$$\{\mathrm{cZ}:10,\mathrm{Yt}:12,\mathrm{mZ}:5,\mathrm{rZ}:8,\mathrm{eZs}:4\},$$
$$\{c:10,Z:27,Y:12,t:12,e:4,m:5,r:8,s:4\},$$
$$\{cZ:10,Yt:12,mZ:5,rZ:8,eZ:4,Zs:4\}.$$

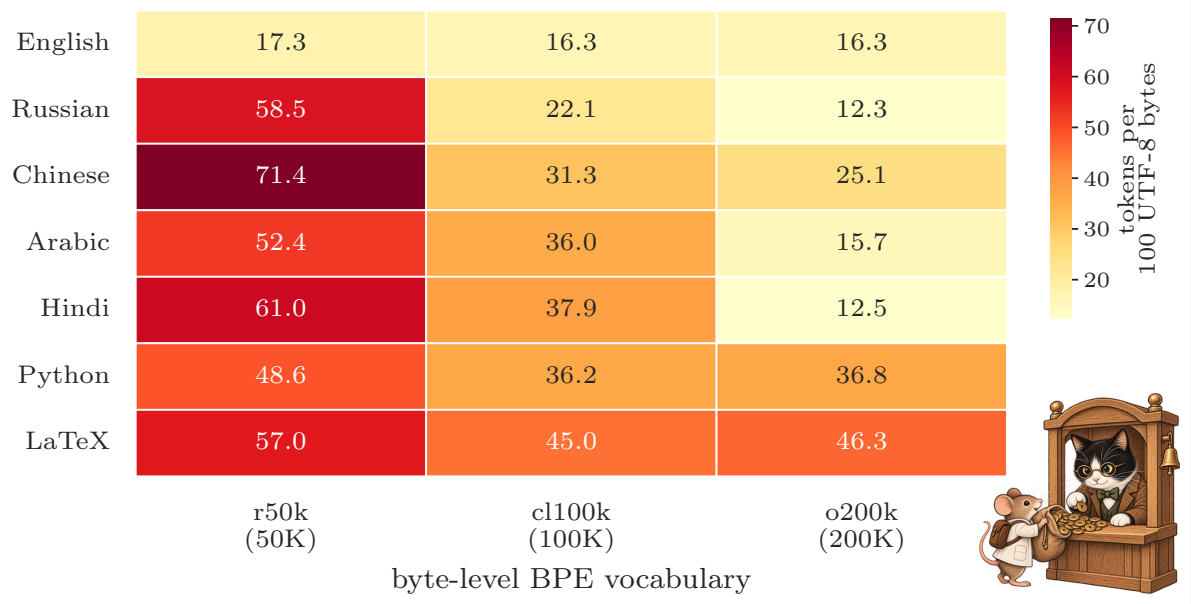
- И так далее, пока не достигнем требуемого размера словаря

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4



Токенизация

- Варианты: byte-level BPE (GPT-2), WordPiece (BERT), Unigram, SentencePiece.
- Токенизатор — часть архитектуры: чем больше словарь, тем короче последовательности, но больше таблица вложений и softmax на выходе.
- И налог на разные языки неравномерный; токенов на 100 байт UTF-8 в трёх поколениях BPE из tiktoken:



Positional encodings

- В трансформере каждый входной токен может “посмотреть” на любой другой *напрямую*, через любое число других токенов
- Это большое преимущество по сравнению с RNN!
- Но это значит, что у нас фиксированное ограниченное *context window* и, главное, поскольку веса внимания покрывают все токены равномерно, мы *теряем последовательность*: слой self-attention “не знает”, что некоторые токены стоят рядом, а некоторые далеко друг от друга
- Нужно как-то дать понять трансформеру, что у последовательности есть порядок, и это делается через *positional encodings*
- Формально: $\mathrm{Attn}(PX) = P\,\mathrm{Attn}(X)$, вход для самовнимания — множество

Positional encodings

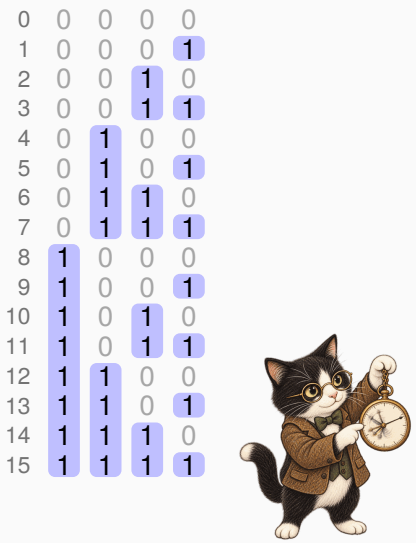
- Positional encodings — векторы, добавляемые ко вложениям, которые отражают порядковый номер токена
- Как его закодировать? Можно просто число увеличивать, но это не так просто:
  - если использовать увеличивающуюся последовательность вроде $1, 2, 3, \ldots$, она не обобщится на последовательности длиннее, чем часто бывало в датасете, да и вообще поведение будет плохо определено для больших значений;
  - если уложить в интервал вроде $[0, 1]$, то positional encoding не будет знать, сколько на самом деле слов между токенами: расстояние от 0 до $\frac{1}{2}$ может быть 1 токен или 100.

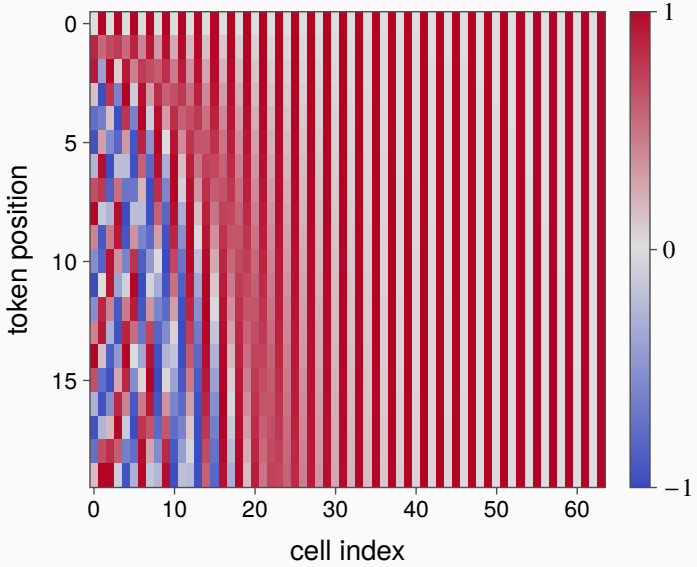
Positional encodings

- Так что мы используем нечто вроде позиционной системы счисления, только непрерывную:

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

где pos — позиция токена, d — размерность вложения.

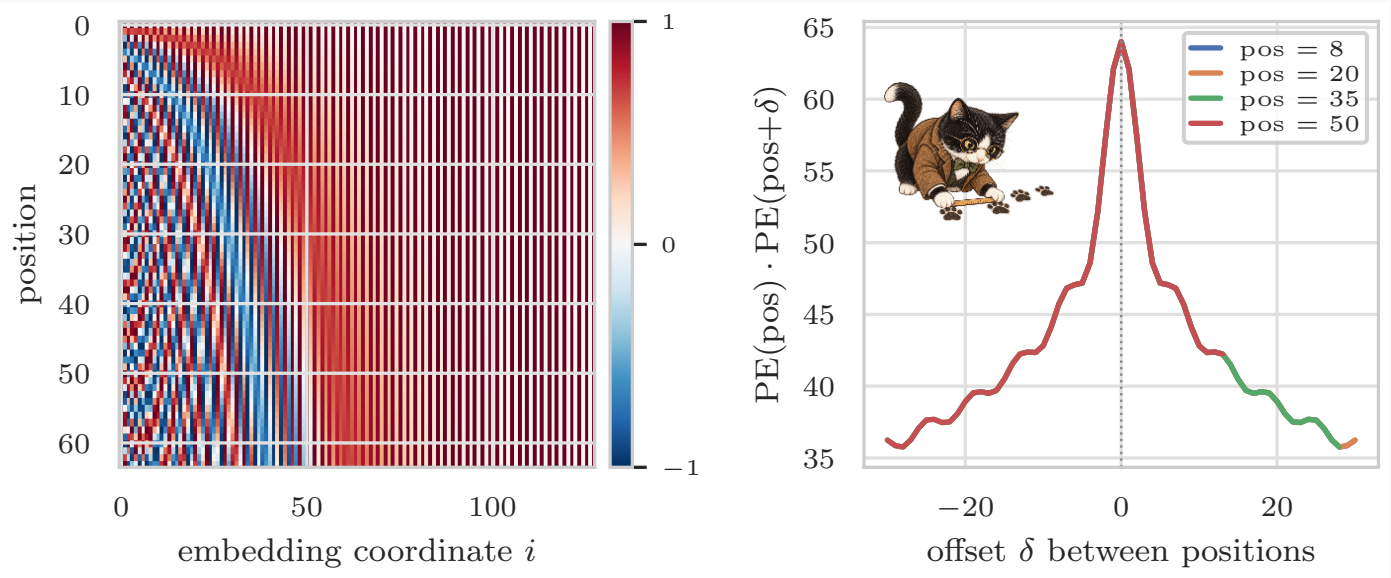




Positional encodings

- Главное свойство: $\mathrm{PE}(p)^\top \mathrm{PE}(p+\delta)$ зависит только от сдвига δ, ведь

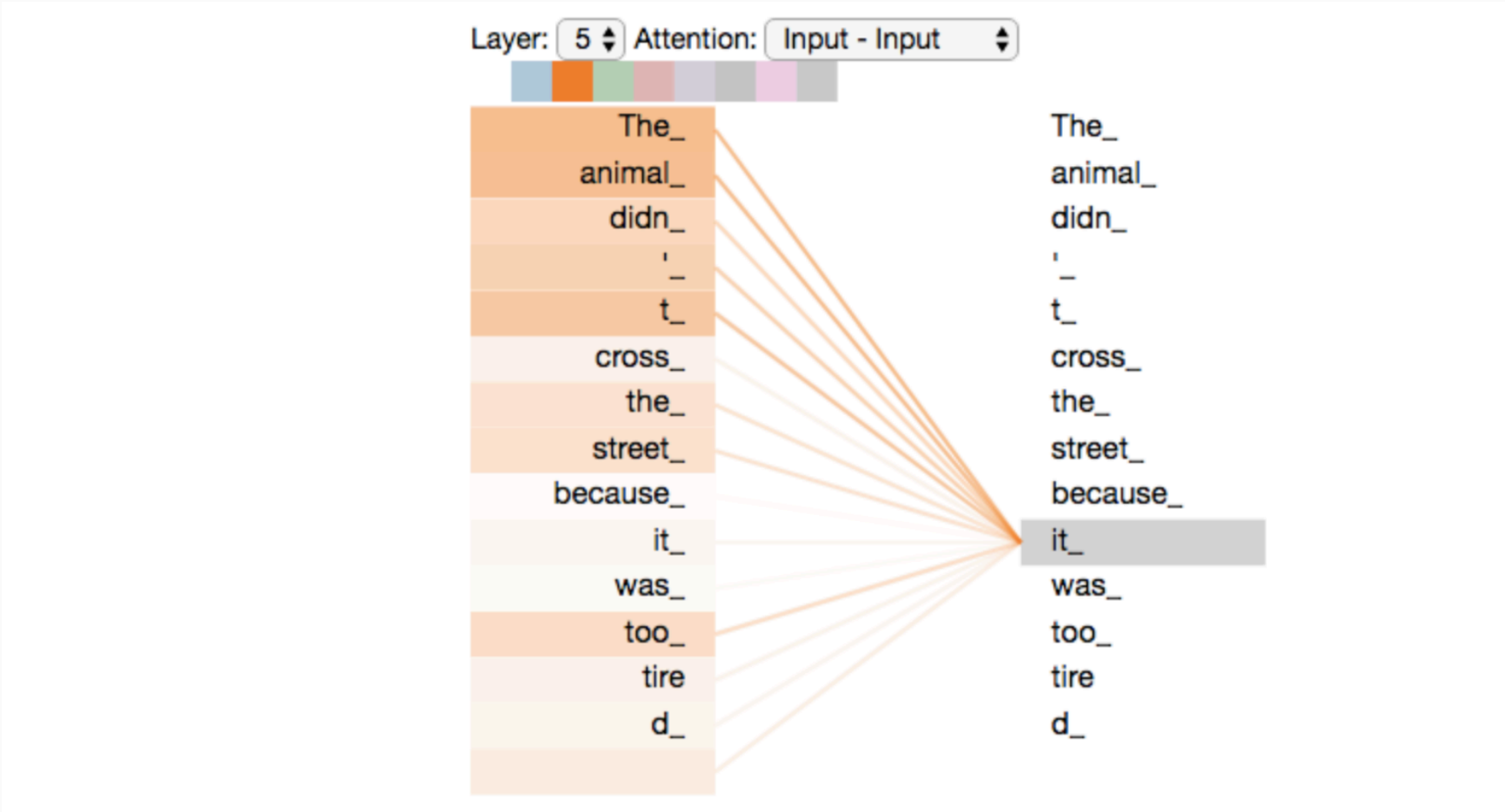
$$\sin(p\omega_i)\sin((p+\delta)\omega_i) + \cos(p\omega_i)\cos((p+\delta)\omega_i) = \cos(\delta\omega_i).$$



- Позиционные кодировки не конкатенируют, а *складывают* с вложениями токенов; выученные позиционные векторы работают почти так же (Vaswani et al., 2017).

Результаты

- В результате трансформер обучает веса того, как одни слова ”участвуют в обработке” других слов:



Результаты

- Бонус от self-attention – во-первых, вычислительный, во-вторых, сокращает пути между словами, в-третьих, потенциальная интерпретируемость:

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Результаты

- Работает лучше, обучается в сто раз быстрее:

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Спасибо за внимание!



