

Методы семплирования токенов из распределения

(не успели в прошлый раз)

Методы семплирования токенов

Генеративная модель выдает распределение вероятностей токенов



Как выбрать токен из этого распределения?

Жадное семплирование

Greedy Sampling

Можно всегда выбирать токен с максимальной вероятностью

$$y_t = \operatorname{argmax}_{y \in V} p(y | y_{<t})$$

Плюсы:

- Максимизируем вероятность текста

Минусы:

- Теряем разнообразие текста

Жадное семплирование

Greedy Sampling


Можно всегда выбирать токен с максимальной вероятностью

$$y_t = \operatorname{argmax}_{y \in V} p(y | y_{<t})$$

Плюсы:

- Максимизируем вероятность текста

Минусы:

- Теряем разнообразие текста
 - Плохо, если генерация **безусловная**
 - Не страшно, если **условная** (seq2seq)
- 

Beam Search

Лучевой поиск

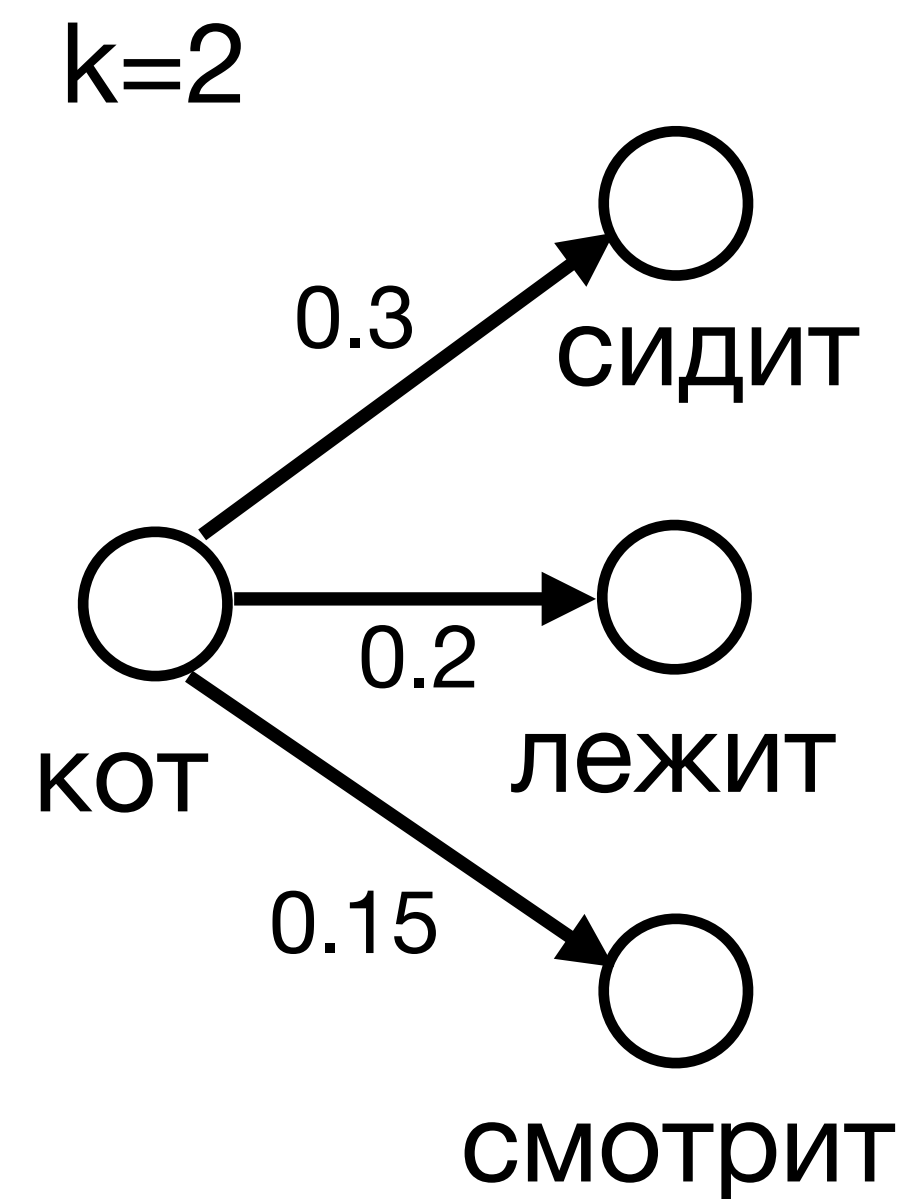
- Попробуем максимизировать вероятность текста еще больше
- При жадном семплировании мы **не учитываем** влияние токена на следующие за ним
- Будем строить предсказания на несколько токенов вперед, а затем выбирать токен с наибольшей совокупной вероятностью

Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

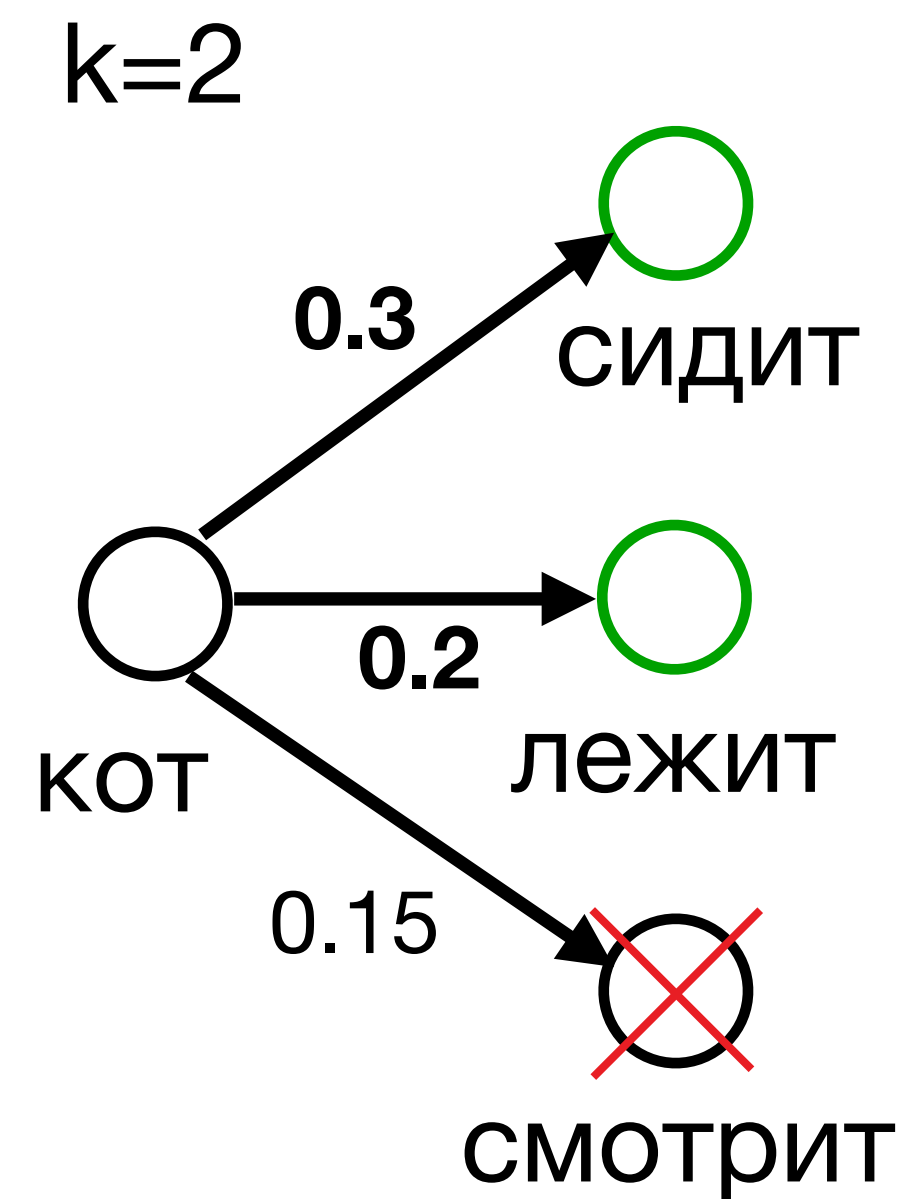


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

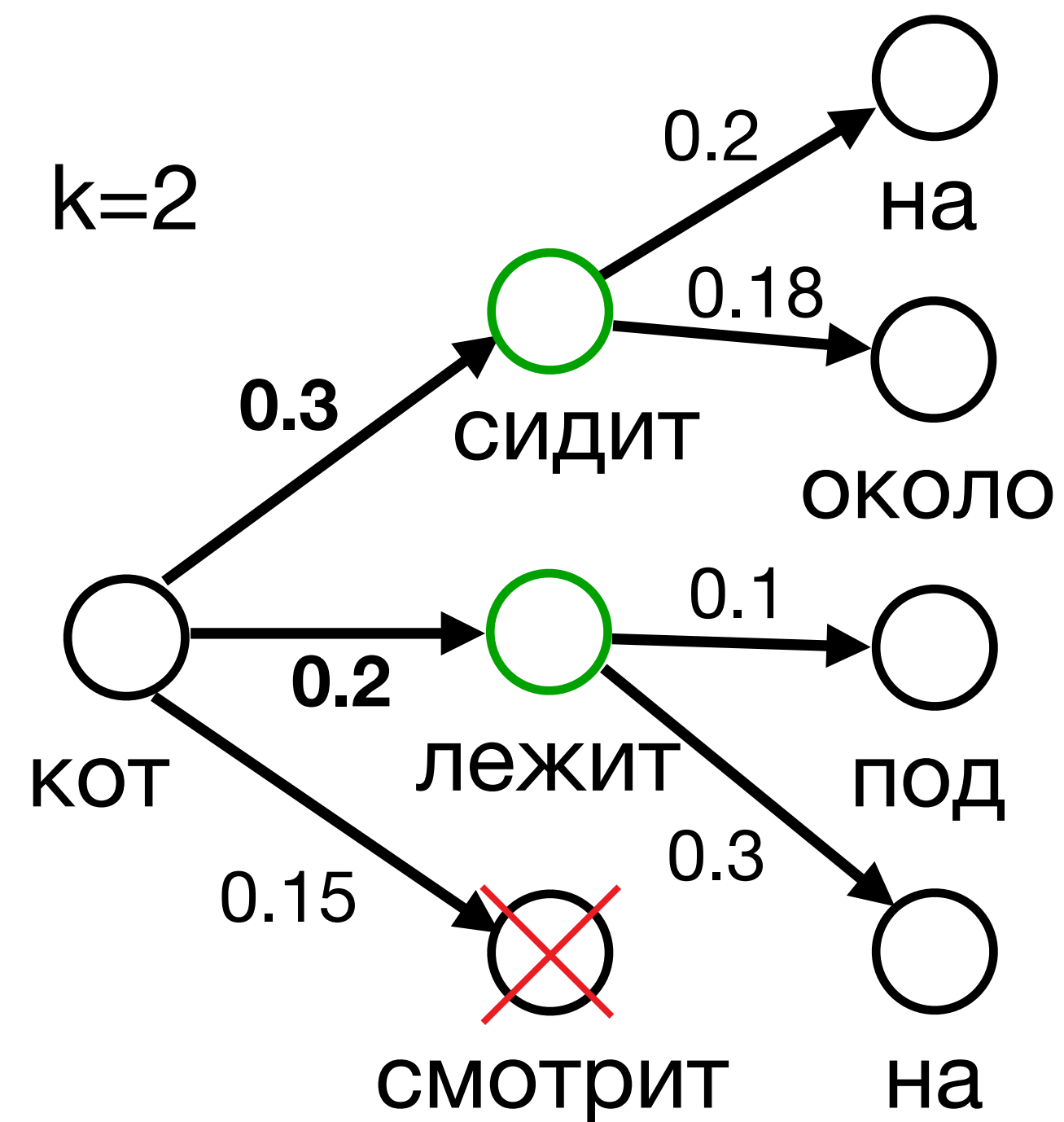


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

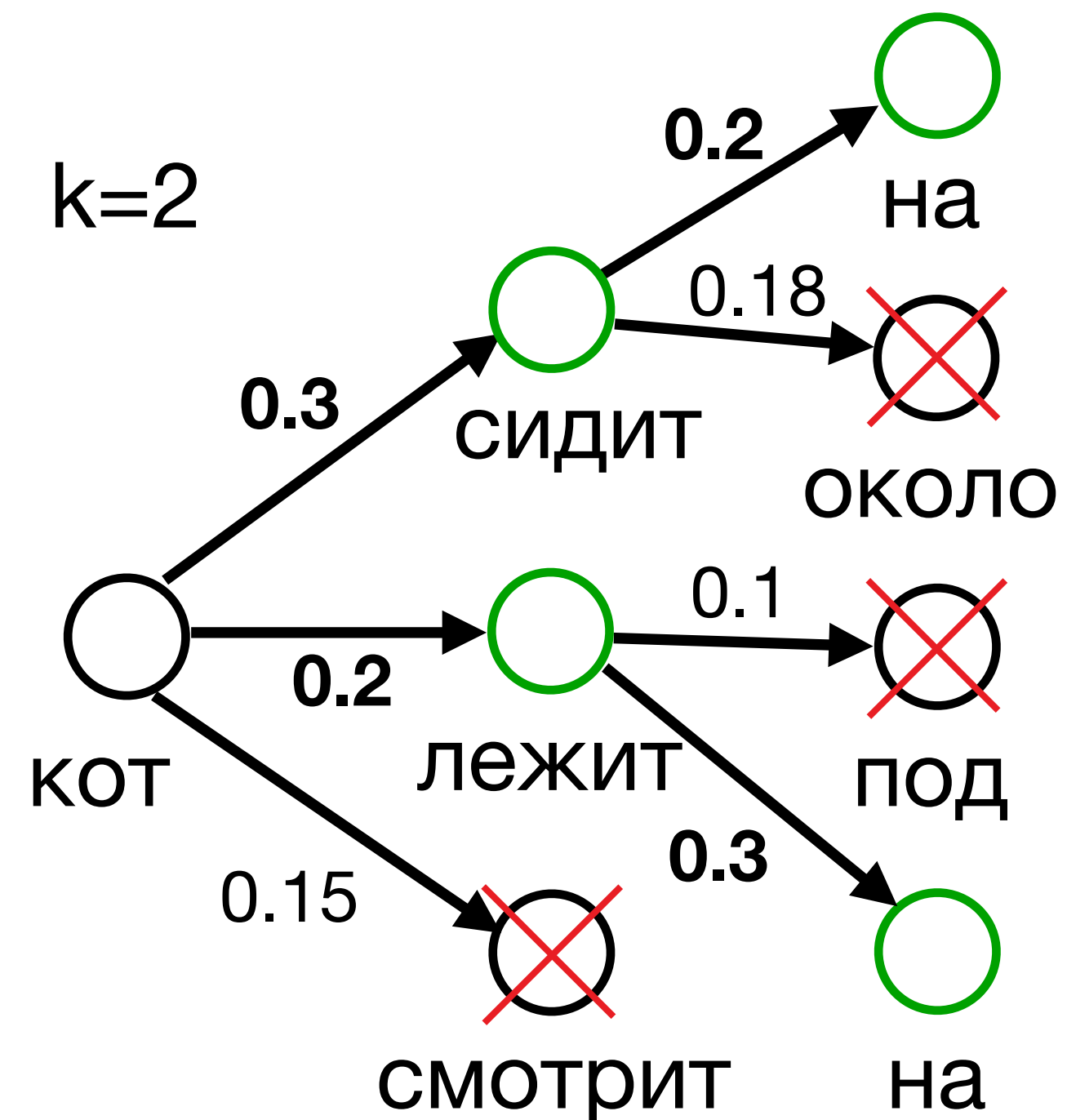


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

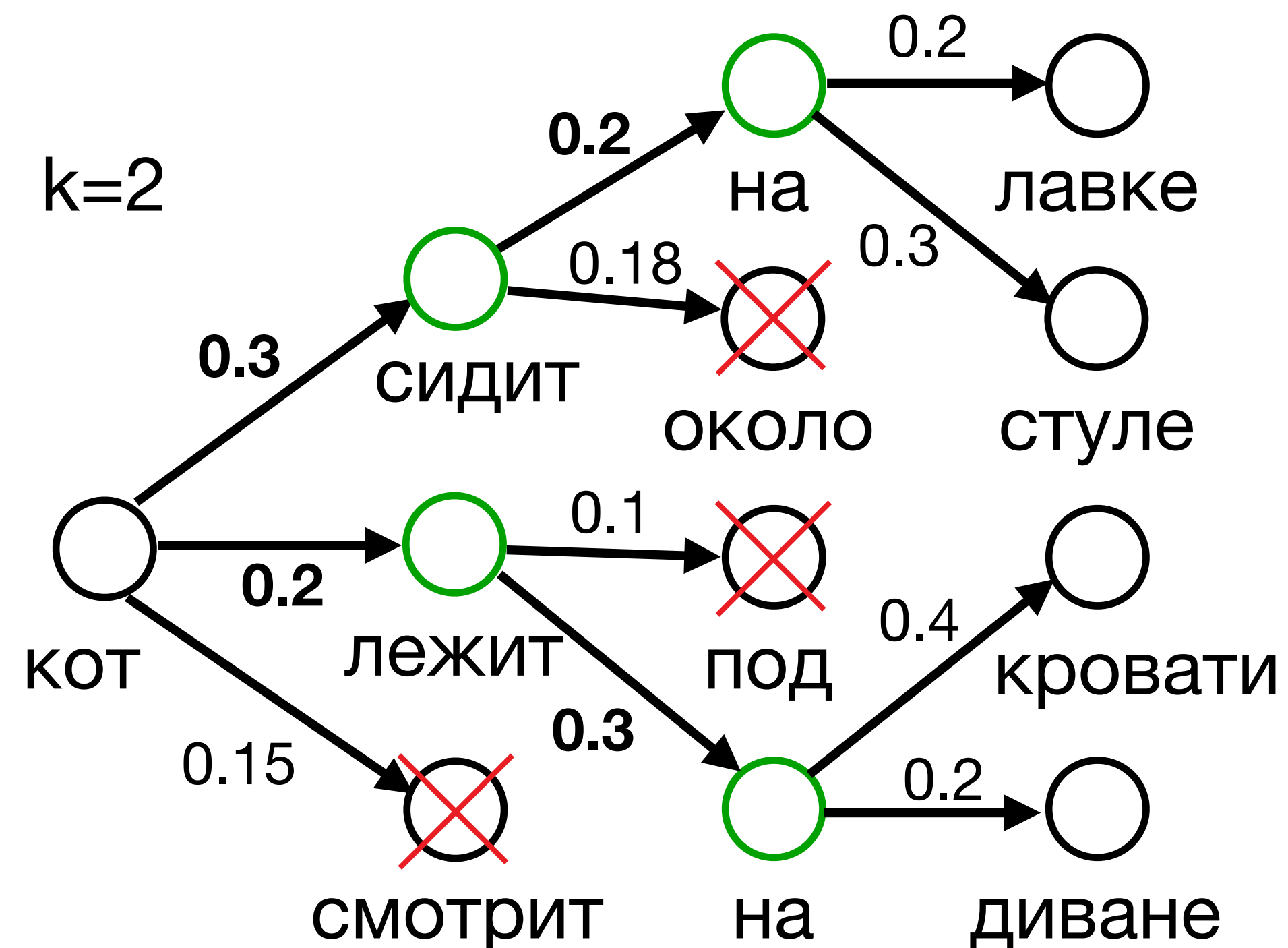


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

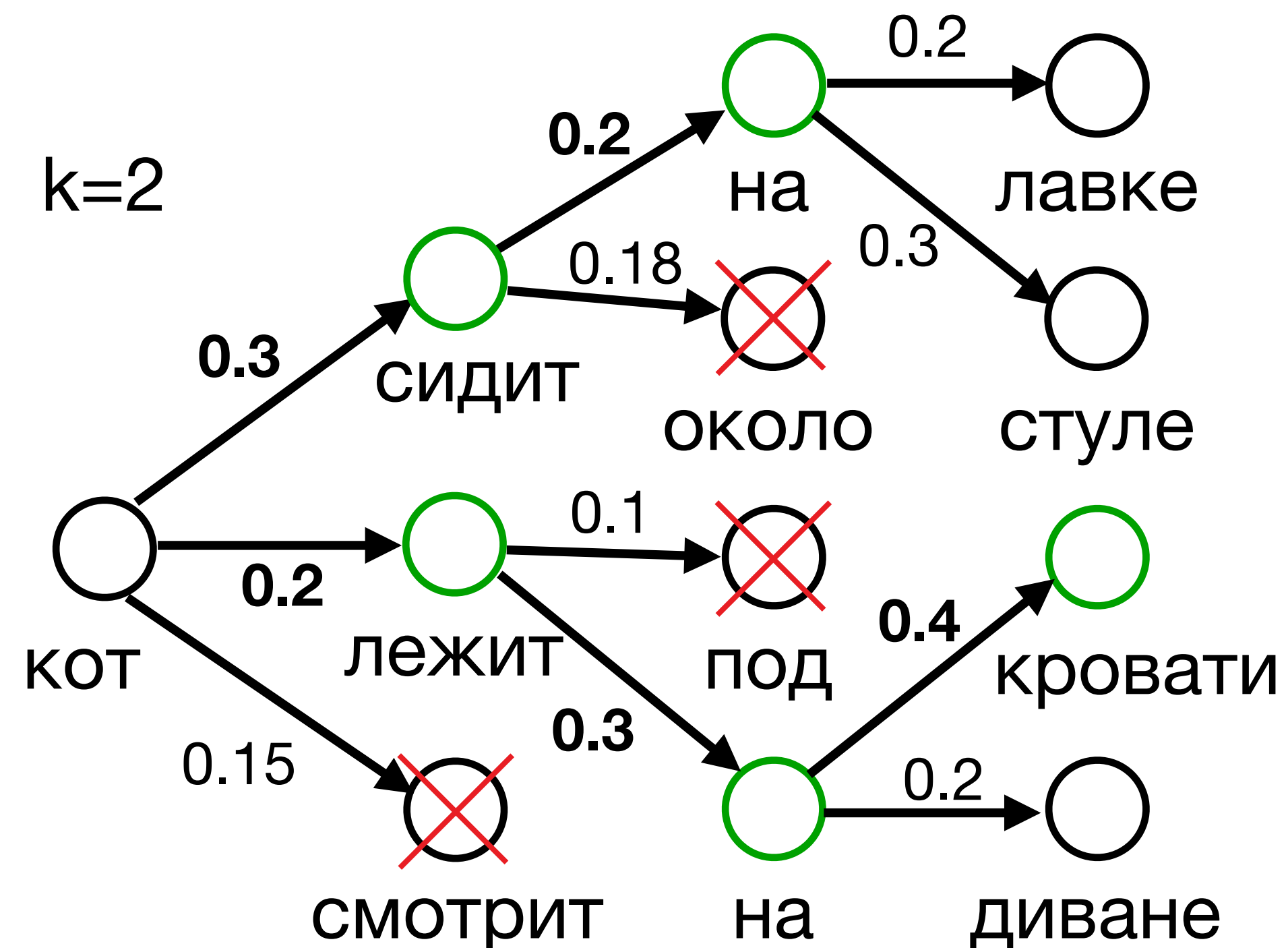


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных



Траектория с "лежит" имеет максимальную вероятность

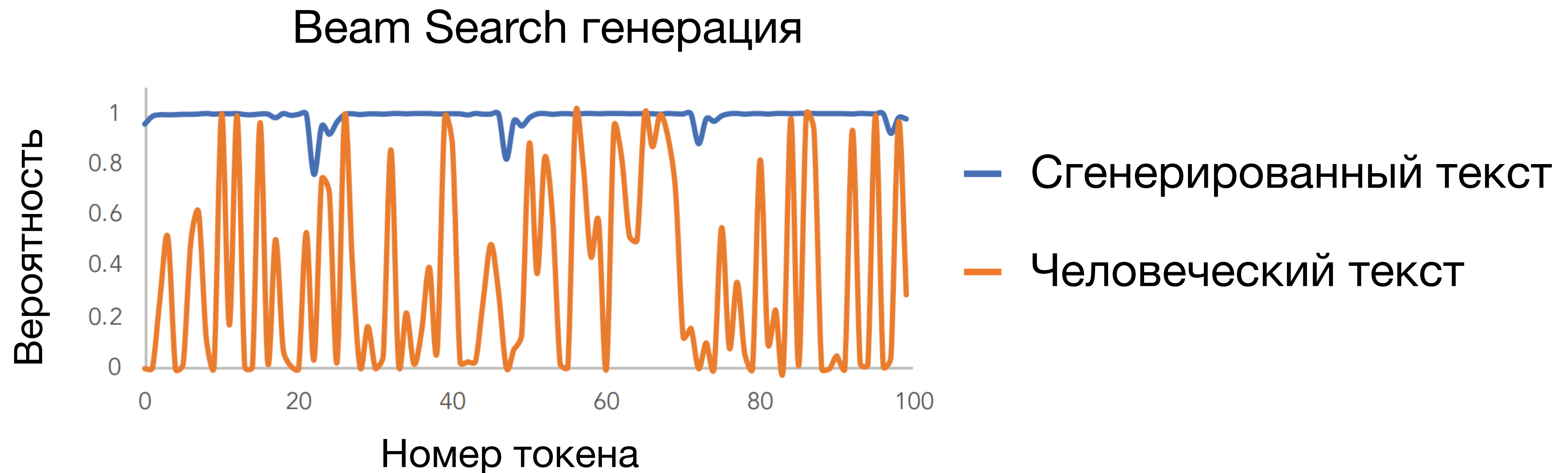
=> выбираем "лежит"

Beam Search vs Жадное семплирование

- Beam Search работает лучше для всех seq2seq задач
- Beam Search работает гораздо медленнее (зависит от k)
- Популярные значения k – 3 или 5

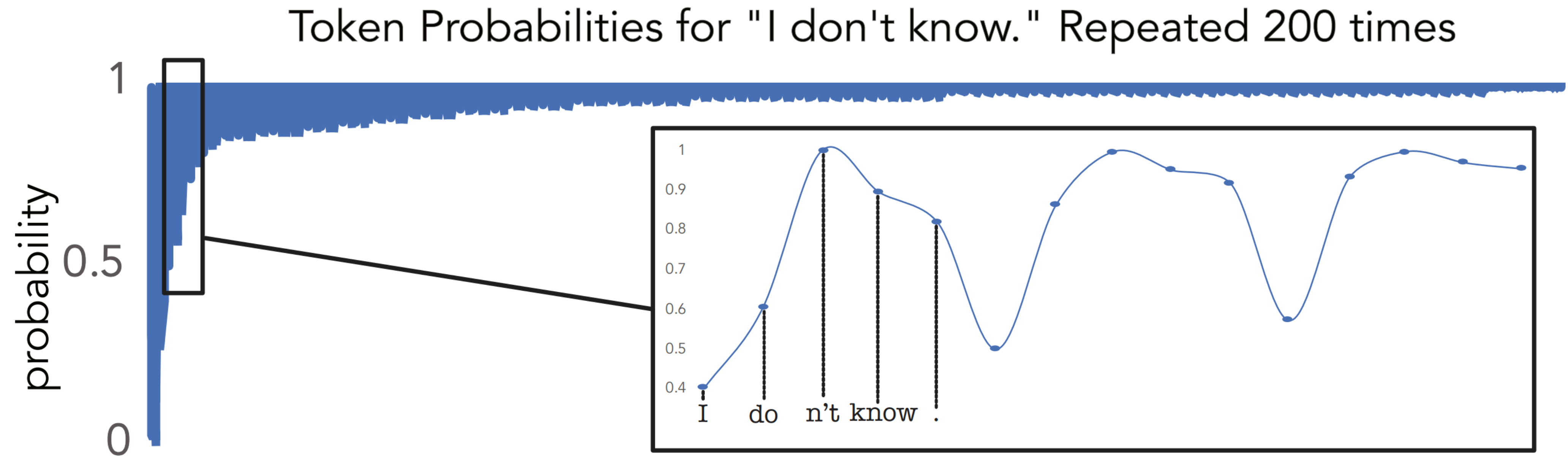
Безусловная генерация

Оба метода уменьшают разнообразие безусловной генерации!



Безусловная генерация

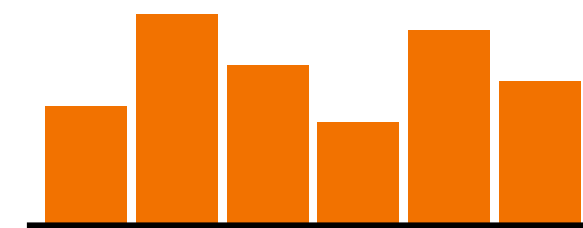
Оба метода поощряют повторения!



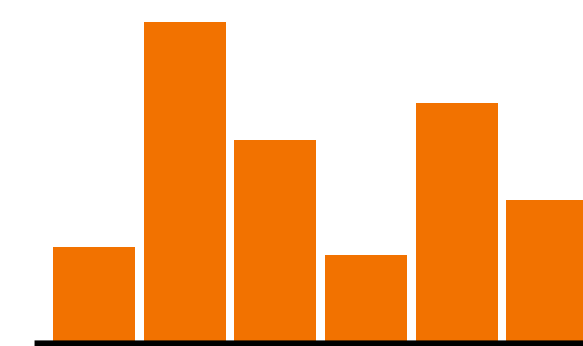
Семплирование с температурой

- При обычном семплировании с вероятностями вероятностная масса случайных токенов слишком велика
- Сделаем распределение более вырожденным, добавив температуру $\tau \in [0,1]$

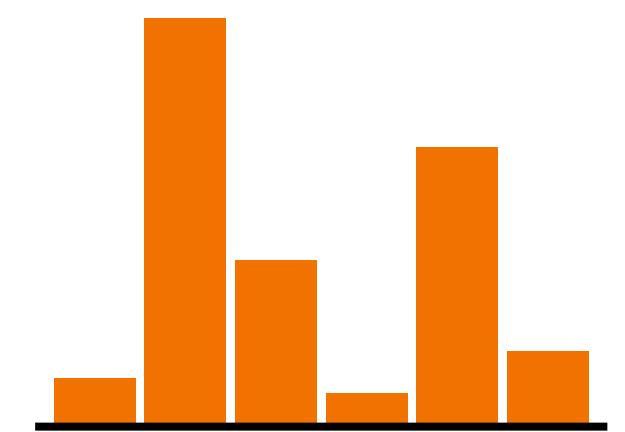
$$p_{\tau}(y | y_{<t}) = \frac{\exp \frac{l(y | y_{<t})}{\tau}}{\sum_{w \in V} \exp \frac{l(w | y_{<t})}{\tau}}$$



$p_2(y_t | y_{<t})$



$p(y_t | y_{<t})$

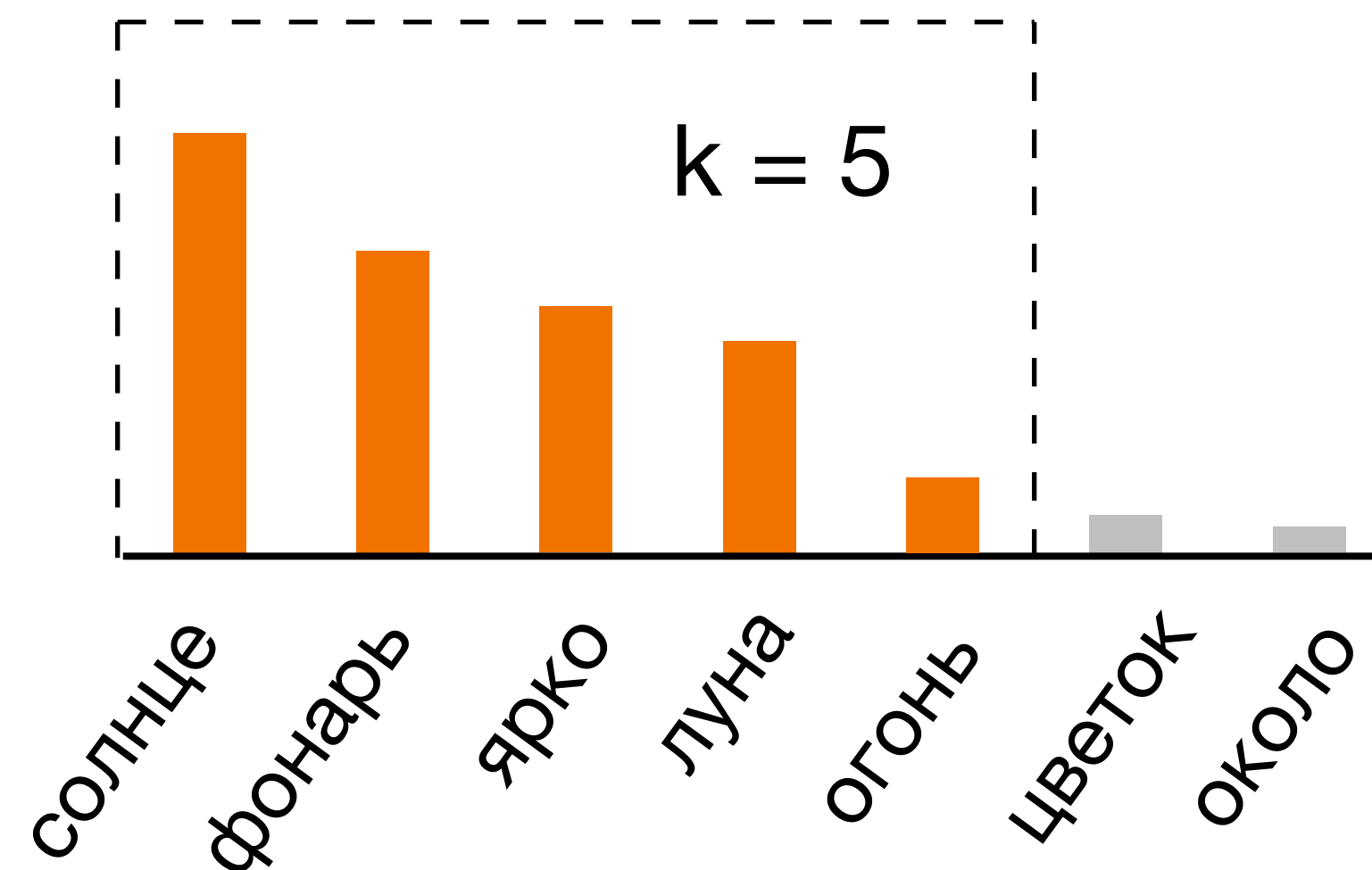


$p_{0.5}(y_t | y_{<t})$

Top-k семплирование

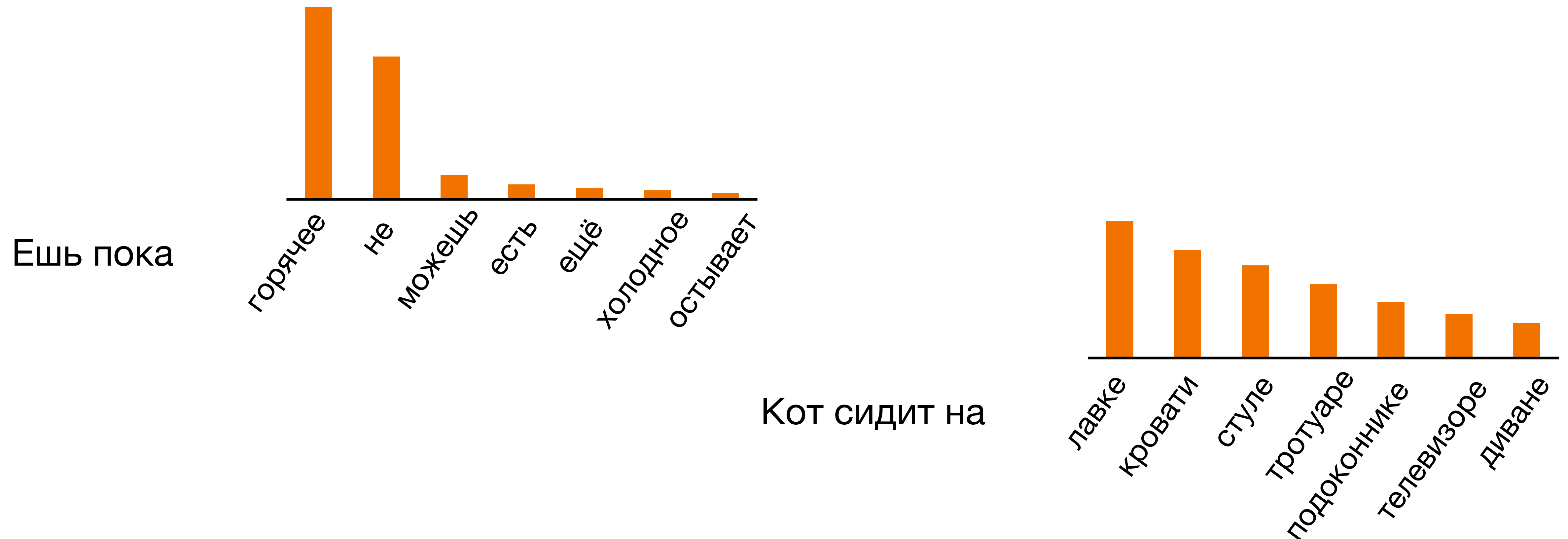
- Даже с температурой остается ненулевая генерация выдать случайный токен
- Оставим только k самых вероятных токенов и будем семплировать из них

На улице светит



Top-k семплирование

- Не во всех ситуациях самых вероятных токенов одинаковое число
- Из-за этого невозможно подобрать идеальное k



Тор-р семплирование

Nucleus sampling

- Будем выбирать из минимального числа токенов, суммарная вероятность которых больше p .

$$\sum_{w \in V^{(p)}} p(w | y_{<t}) \geq p$$

$$|V^{(p)}| \rightarrow \min$$

Популярное значение для p – 0.9 или 0.95

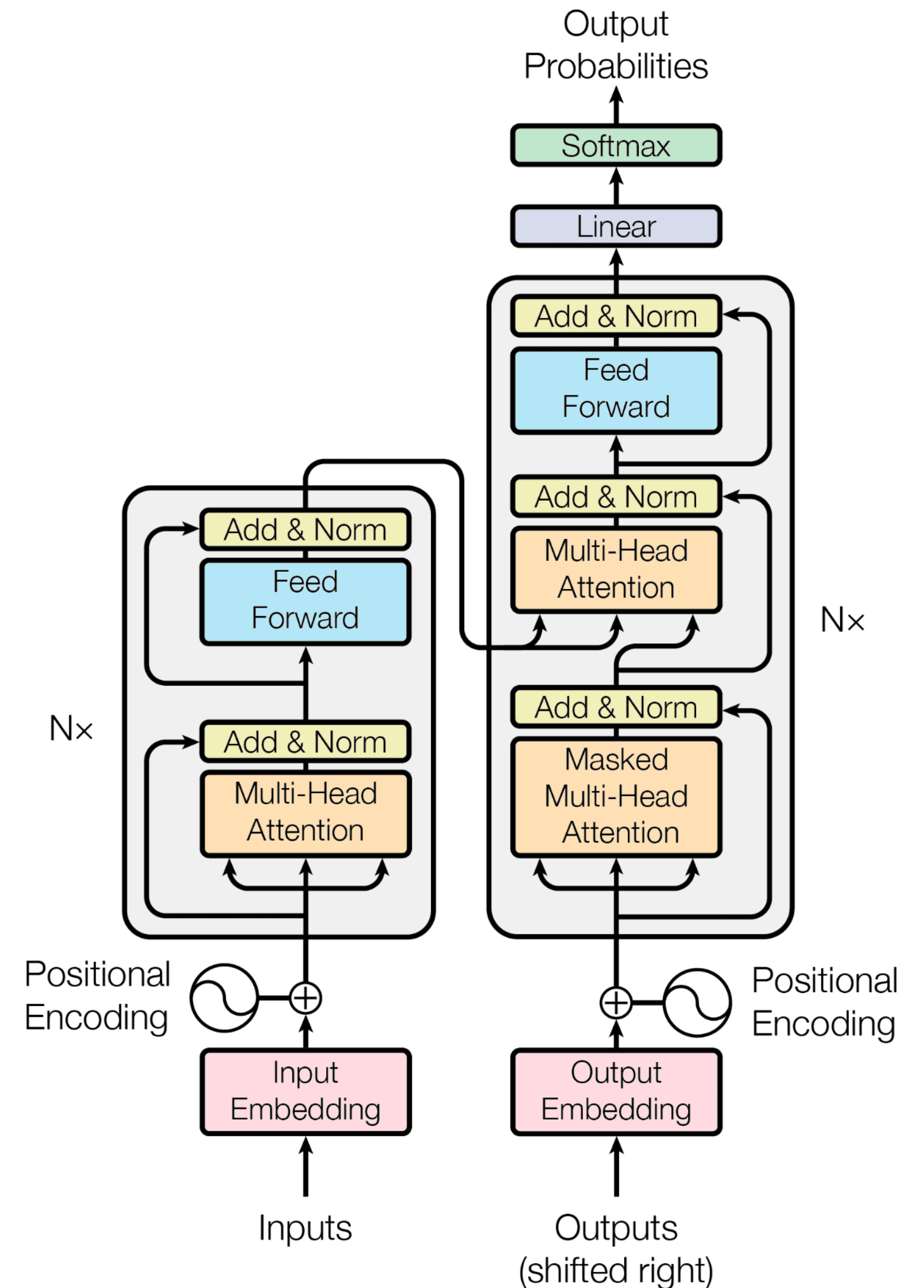
BERT и GPT

План

- **BERT**
 - И его модификации
- **GPT**
 - И почему она так хороша
- Трюки для обучения трансформерных моделей

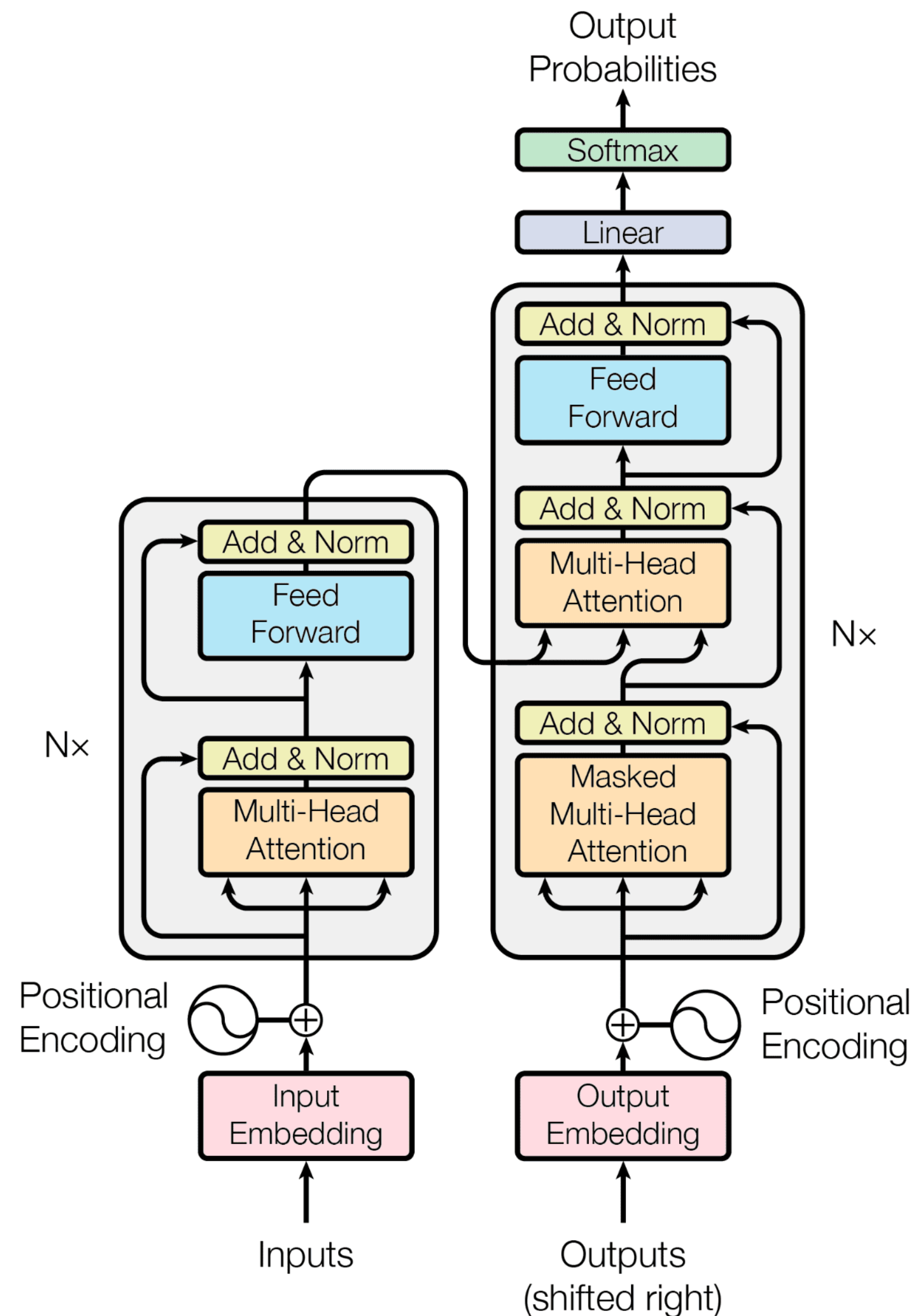
Трансформер

- Модель для **Seq2seq**, состоящая из **Encoder** и **Decoder**
- **Encoder** извлекает информацию из текста
- **Decoder** генерирует новый текст на основе этой информации
- Хорошо работает благодаря механизму **ВНИМАНИЯ**



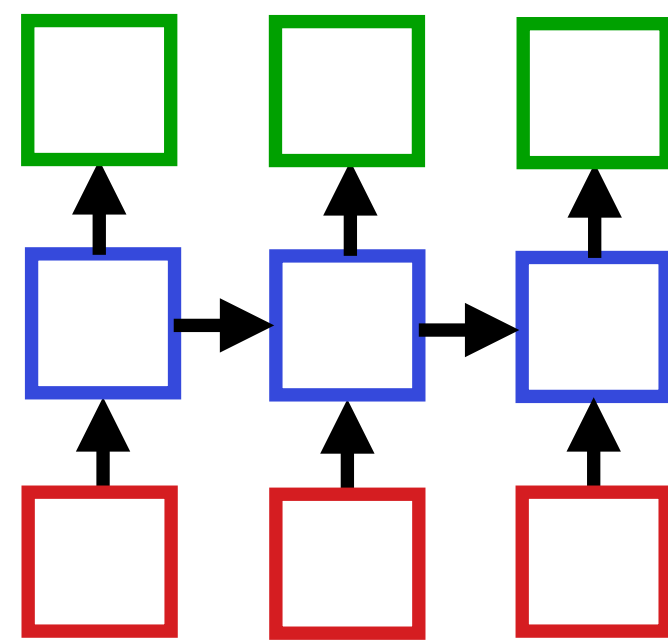
Трансформер

Можно ли адаптировать эту модель под другие виды задач?



Какие бывают задачи?

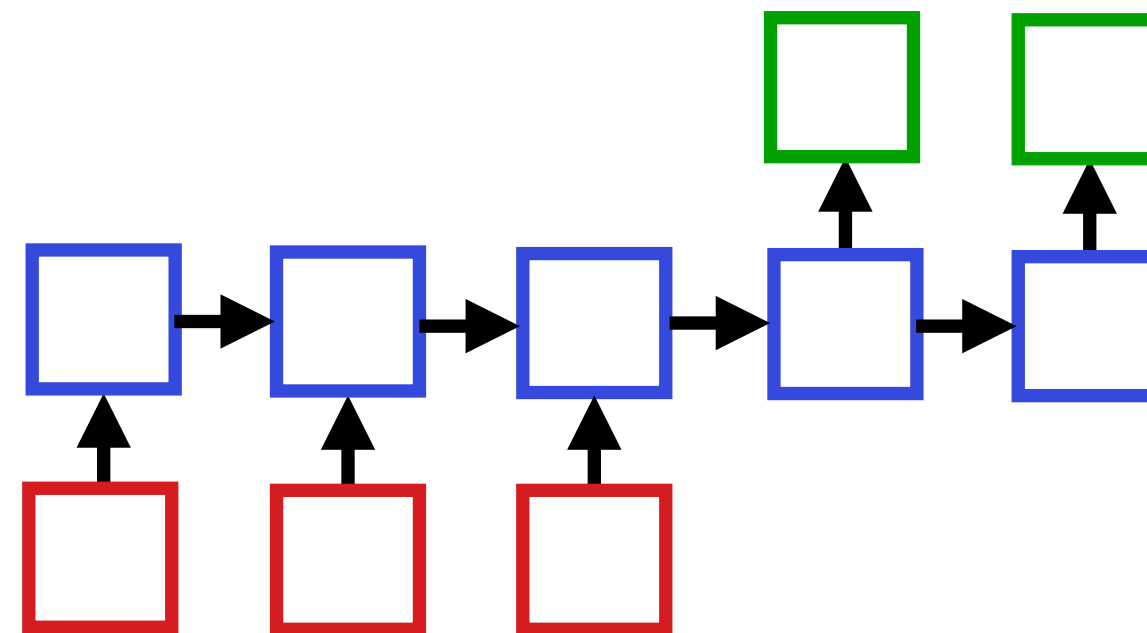
many-to-many
одинаковая длина



Классификация токенов:

- Named Entity Recognition
- Part-of-speech tagging

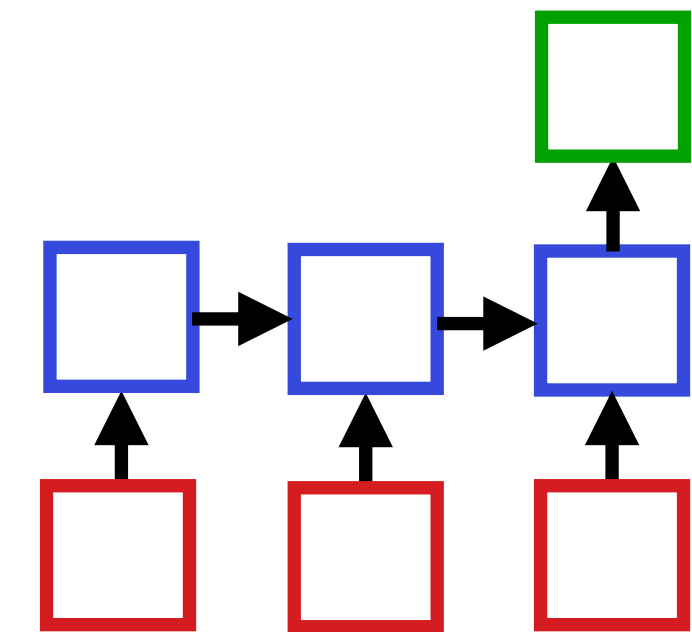
many-to-many
разная длина



Seq2seq:

- Машинный перевод
- Суммаризация
- Перенос стиля

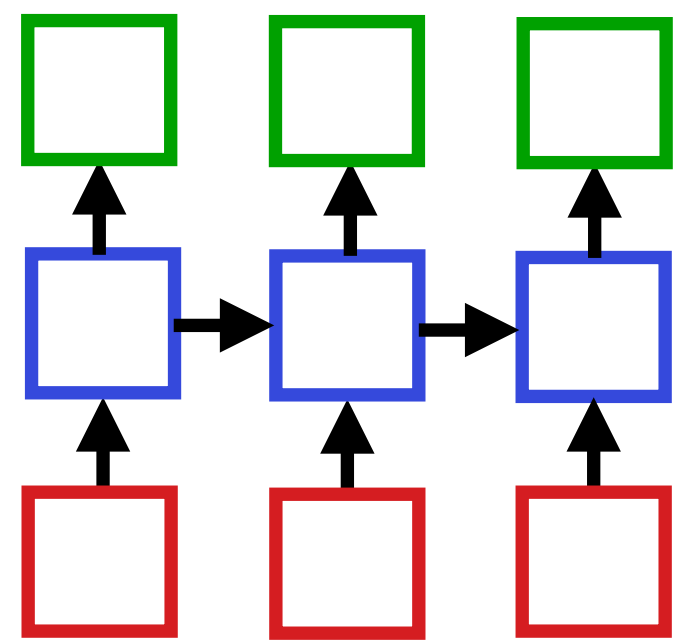
many-to-one



- Классификация
- Регрессия

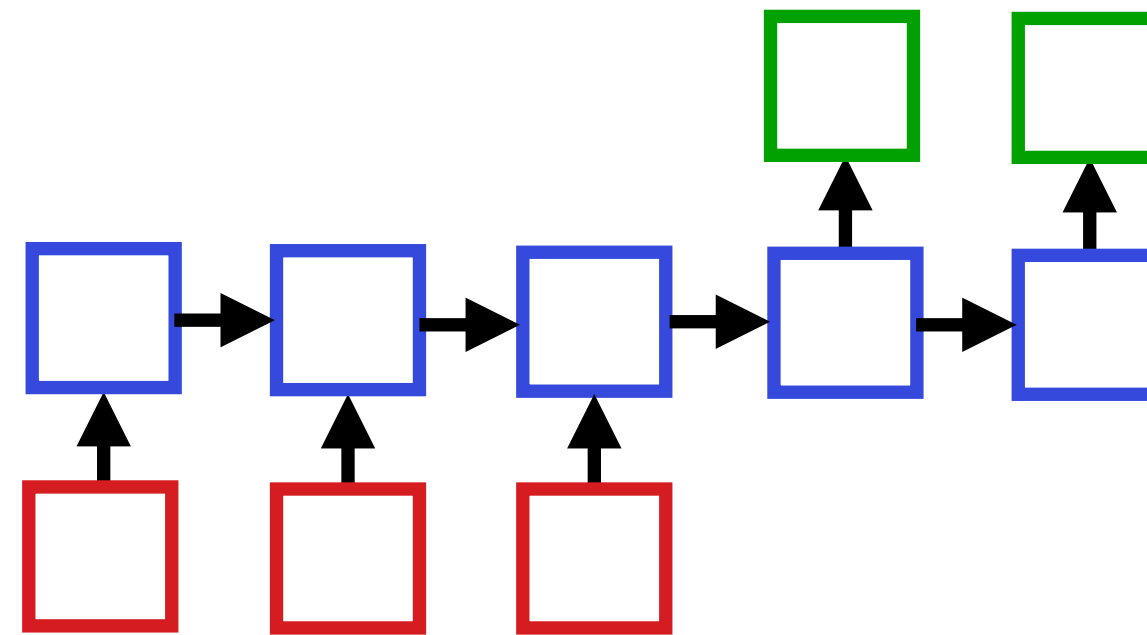
Какие бывают задачи?

many-to-many
одинаковая длина



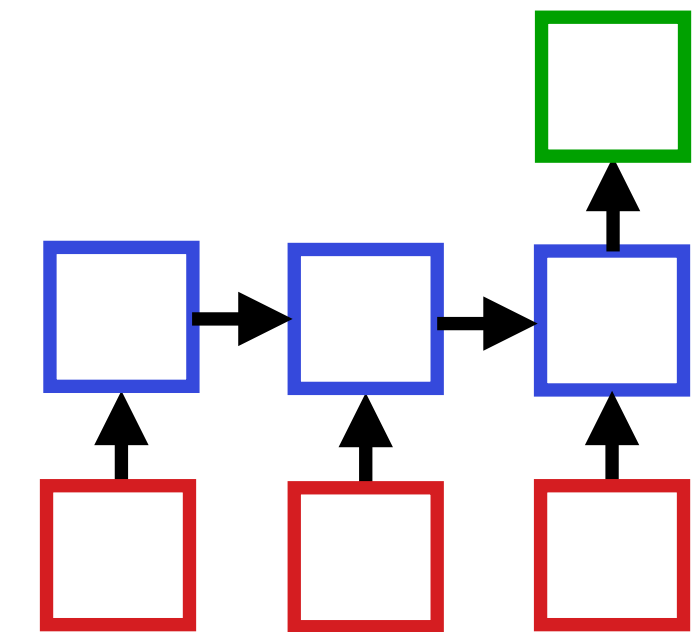
- Классификация токенов:
- Named Entity Recognition
 - Part-of-speech tagging

many-to-many
разная длина



- Seq2seq:
- Машинный перевод
 - Суммаризация
 - Перенос стиля

many-to-one



- Классификация
- Регрессия

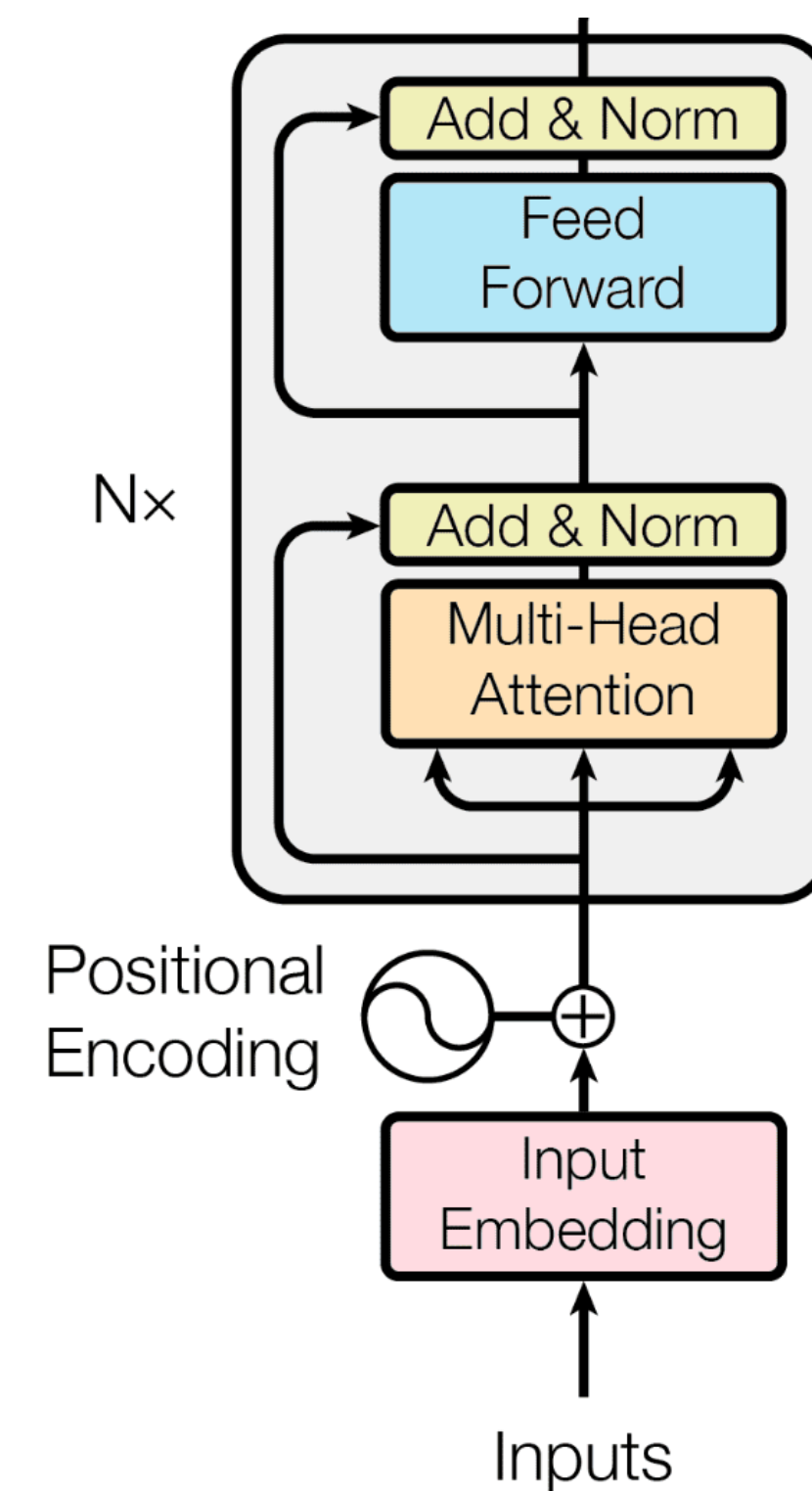
Не нужно генерировать текст
Можем взять отдельно **Encoder!**

Нужно генерировать текст

BERT (2018)

Bidirectional Encoder Representations

- Модель на основе **Encoder**'а Трансформера
- В основном используется для решения произвольных задач классификации
- Самое важное – способ обучения BERT



BERT: Мотивация

- Трансформерные модели могут быть огромными
- Базовый BERT содержит 110M параметров
- Для обучения такой модели требуется очень много данных
- В задачах классификации размеченных данных обычно недостаточно

BERT: Мотивация

- Трансформерные модели могут быть огромными
- Базовый BERT содержит 110M параметров
- Для обучения такой модели требуется очень много данных
- В задачах классификации размеченных данных обычно недостаточно

Предобучим модель понимать язык на **неразмеченных** данных!

BERT: Обучение

Для предобучения любой модели без разметки необходимо создать **искусственную** задачу

BERT предобучается одновременно решая две задачи:

- Masked Language Modelling (**MLM**)
- Next Sentence Prediction (**NSP**)

Masked Language Modelling (MLM)

Идея: Маскируем несколько случайных токенов и просим модель их восстановить. Модель учится понимать, что значат слова и как они связаны между собой.

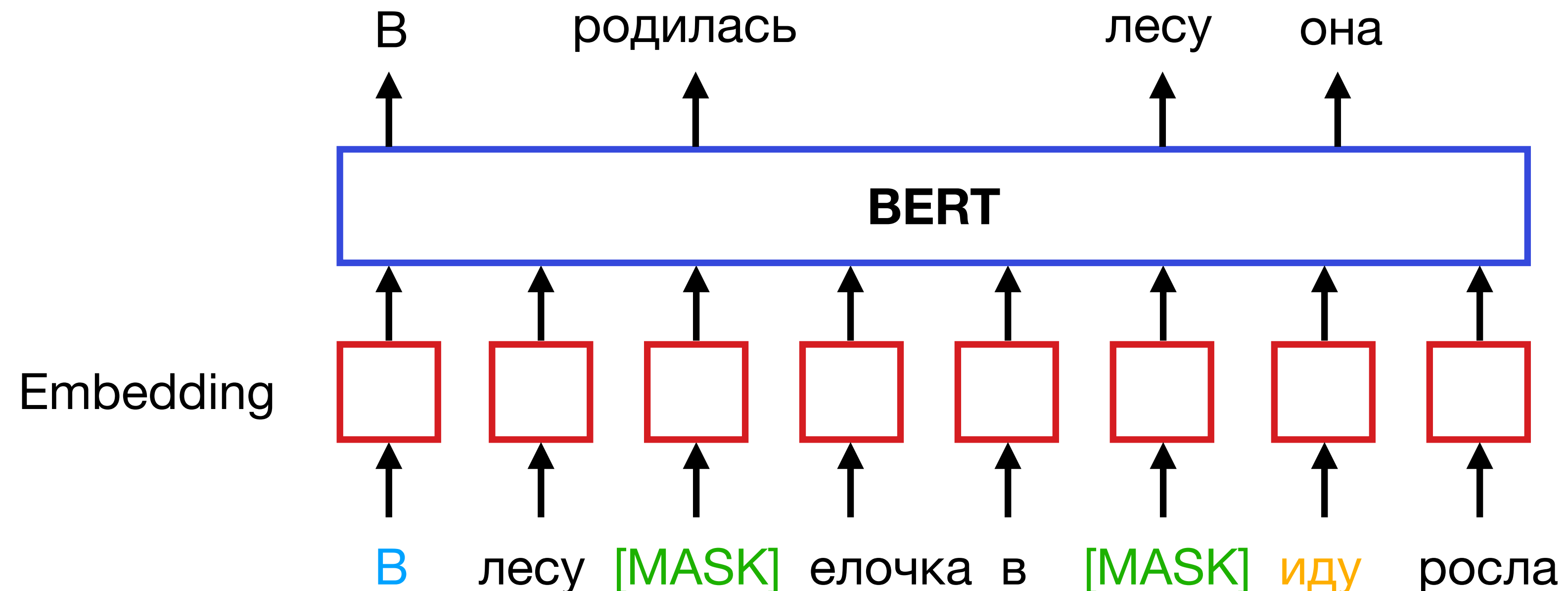
Алгоритм обучения:

- Случайно выбираем 15% токенов. Из них
 - 80% маскируем
 - 10% заменяем случайными токенами (чтобы модель не переобучалась под [MASK])
 - 10% оставляем без изменений (чтобы модель не считала все токены неправильными)
- Считаем ошибку для всех выбранных токенов

Masked Language Modelling (MLM)

Алгоритм обучения:

- Случайно выбираем 15% токенов. Из них
 - 80% маскируем
 - 10% заменяем случайными токенами
 - 10% оставляем без изменений
- Считаем ошибку для всех выбранных токенов

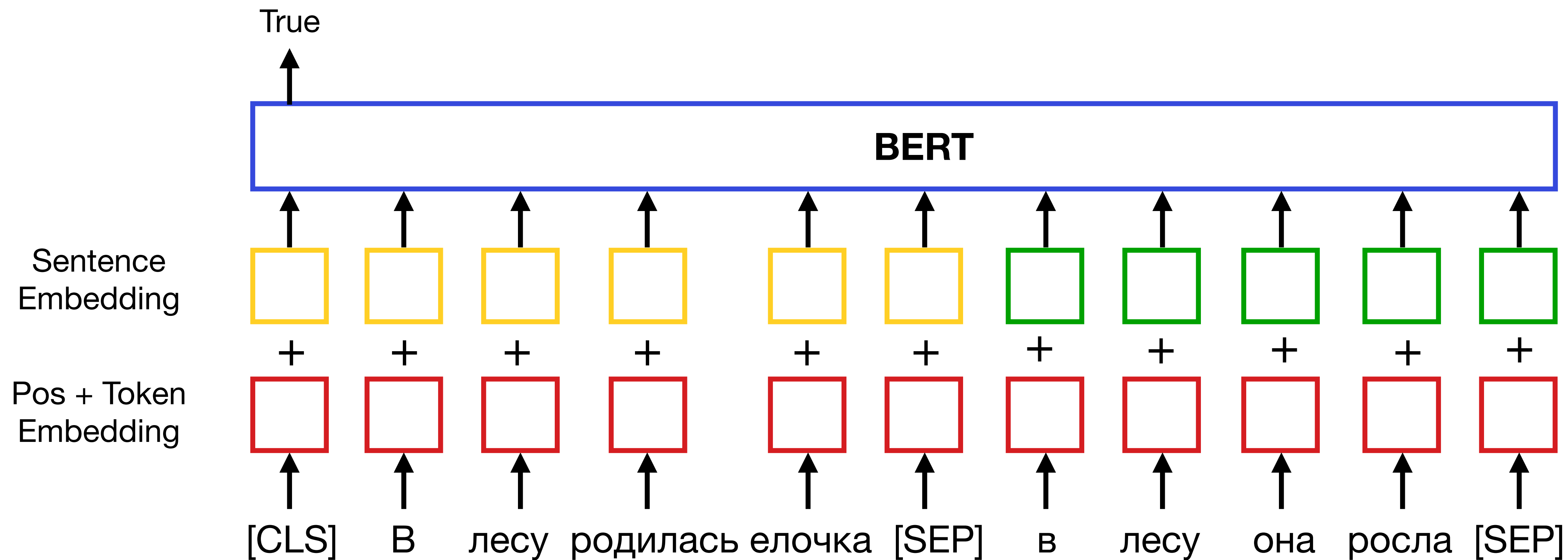


Next Sentence Prediction (NSP)

Идея: Хотим научить модель извлекать информацию из текста целиком. Полезно для классификации текста.

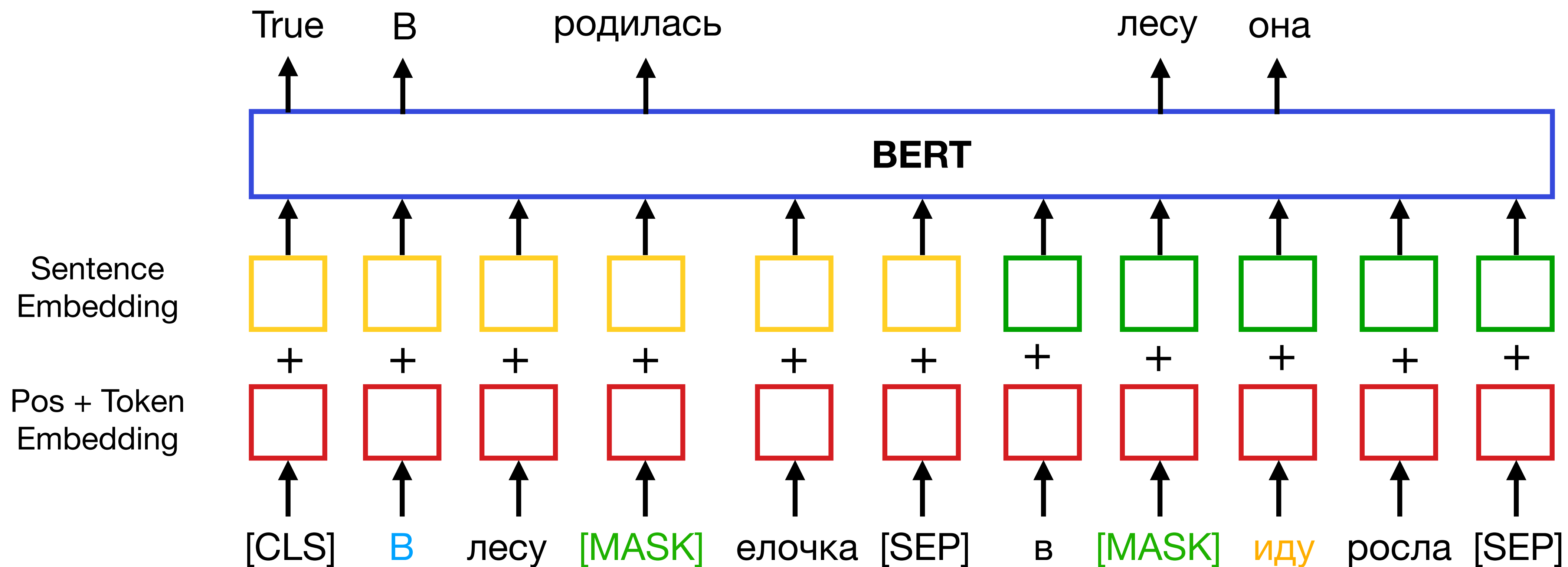
- Подаем в BERT два склеенных текста
- В половине случаев второй текст является продолжением первого, а в другой половине он случайный
- Добавляем [CLS] токен в начало и [SEP] в конец каждого текста
- Задача модели – по векторному представлению [CLS] предсказать, связаны ли тексты

Next Sentence Prediction (NSP)



Комбинация двух задач

BERT учится решать две задачи одновременно.



BERT: Результат

- Использование предобучения позволило обучить модель хорошо понимать текст
- На момент создания BERT обгонял ближайшего конкурента на 7% на бенчмарке GLEU
~~А так же уволил с работы половину ресерчеров~~

Датасеты (16 гб):

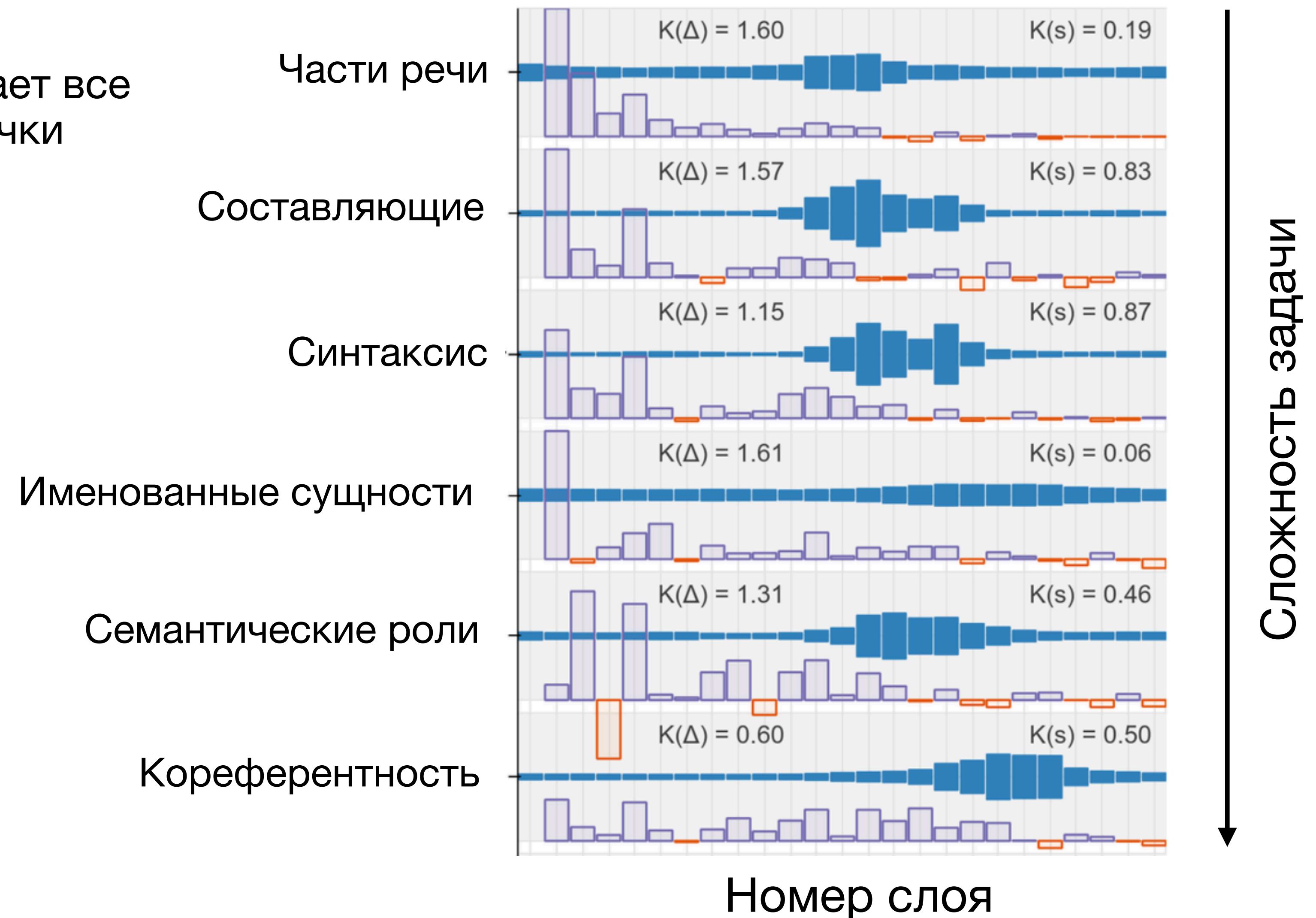
- BooksCorpus (800M слов)
- English Wikipedia (2500M слов)

Время обучения:

- 4 дня на 16 TPU

Как BERT извлекает признаки?

BERT на каждом слое извлекает все более сложные признаки с точки зрения лингвистики



Модификации BERT

Создание BERT стало революционным.

Появилась модель, которая хорошо решает ВСЕ задачи классификации.

Исследователи принялись изучать BERT и предлагать модификации.

Примеры:

- RoBERTa
- ALBERT
- DistilBERT
- DeBERTa
- ELECTRA

RoBERTa (2019)

Robustly optimized BERT approach

Оказалось, что BERT обучен очень неоптимально

RoBERTa учится

- Дольше (в 5 раз)
- На большем объеме данных (в 10 раз больше)
- С большим размером батча (8к токенов)
- С **динамическим** маскированием

Перед началом обучения BERT для каждого текста генерировалось 10 разных вариантов маскирования

RoBERTa маскирует текст случайно на каждой итерации

ALBERT (2019)

A Little BERT

- С увеличением размера модели качество растет
- Однако так же растет сложность вычислений и затраты по памяти
- Можем уменьшить число параметров двумя трюками:
 - Факторизуем матрицу эмбеддингов, так как она очень большая
 - Будем делить одни параметры между несколькими слоями (**parameter sharing**)
- Задача NSP заменяется на Sentence Order Prediction (SOP). Два предложения из одного текста идут либо в правильном порядке, либо в обратном

ALBERT: Факторизация матрицы

- Матрица эмбеддингов E имеет размер $|V| \times D$, где $|V|$ – размер словаря, а D – размер скрытого пространства.
- Чтобы уменьшить число параметров, приблизим E произведением двух матриц $E = \underbrace{A}_{|V| \times H} \times \underbrace{B}_{H \times D}$
- При этом $H \ll D$ (низкоранговое приближение)

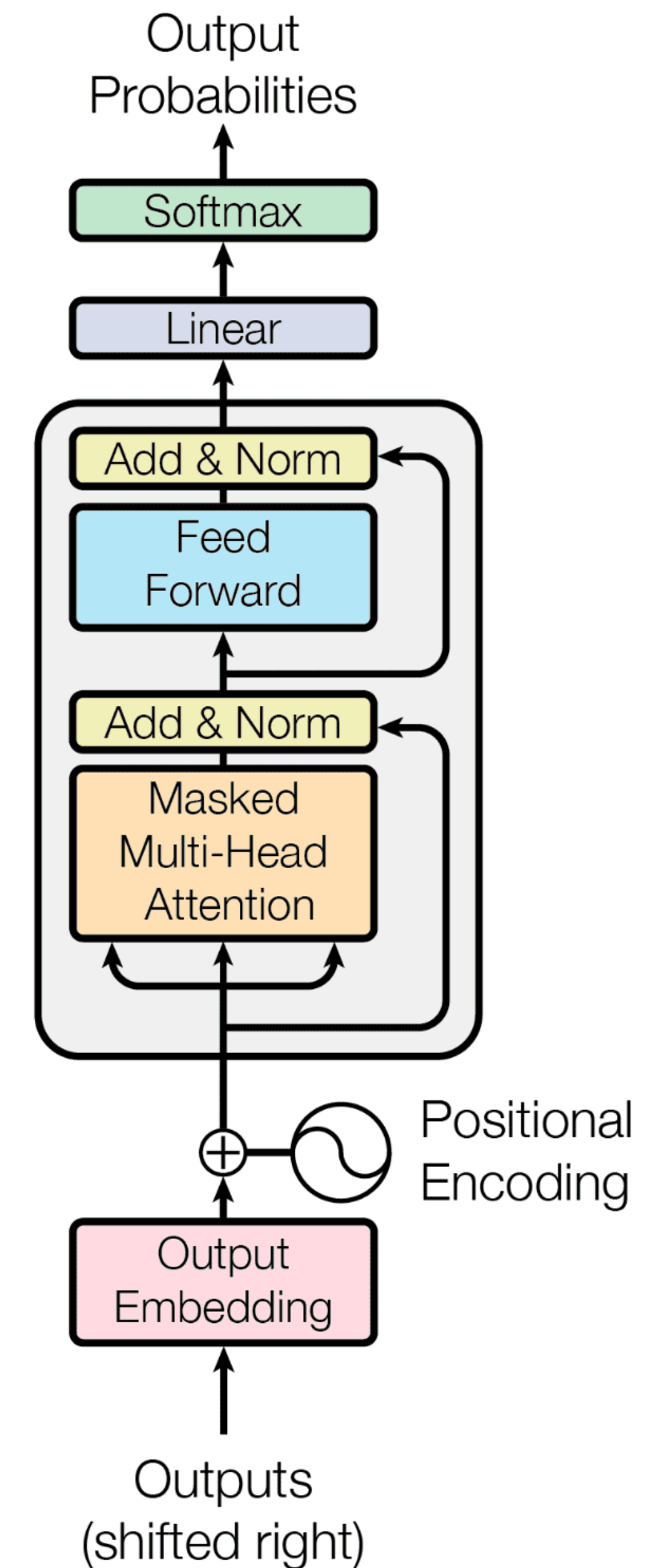
ALBERT: Parameter sharing

- Оказывается, что если применить один и тот же слой несколько раз, информации извлечется больше, чем когда он применяется один раз
- Используем эту идею и разделим параметры одного слоя между всем слоями модели
- Если в модели L слоев, то затраты по памяти уменьшатся в L раз
- Удалось уменьшить число параметров в 10 раз без потери в качестве

GPT (2018)

Generative Pre-Training

- Второй класс моделей на основе Трансформера, способных предобучаться
- Применяется для **генерации** текста
- Использует **декодер** Трансформера без Cross Attention



GPT: Обучение

- GPT учится **языковому моделированию** (безусловная генерация текста)
- Благодаря способности Трансформера масштабироваться, GPT запоминает очень много информации в процессе обучения

Из-за этого GPT отлично дообучается на частные задачи

- Любую задачу NLP можно свести к **генерации текста**

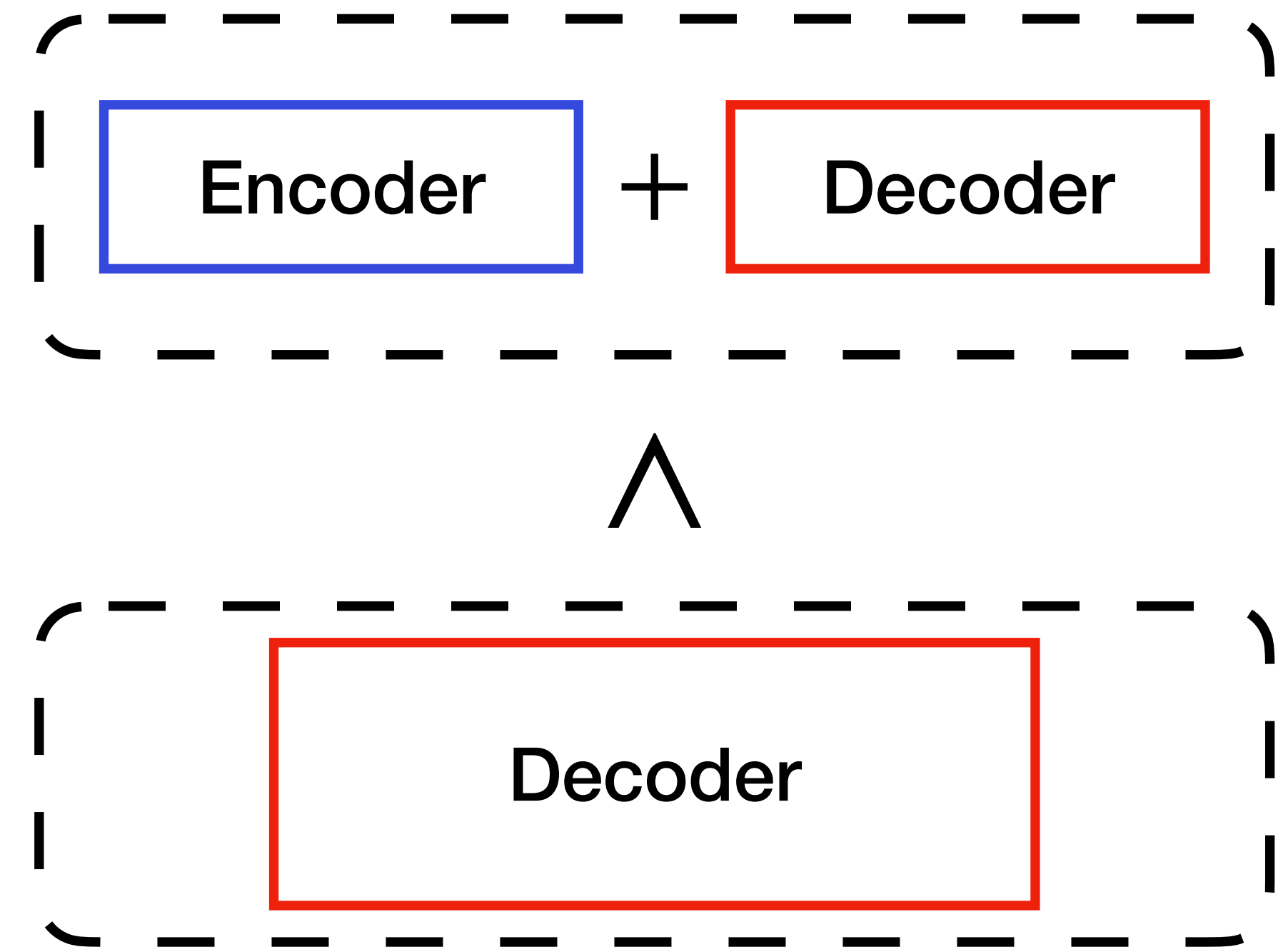
	Параметры	Данные
GPT-1	117M	5 гб
GPT-2	1.5B	40 гб
GPT-3	175B	45 тб

GPT обучается решать разные задачи

Задача	Пример текста, обучающий этой задаче
Грамматика	В свободное время я люблю (читать, табуретка)
Лексическая семантика	Я пошел в магазин, чтобы купить манго, апельсин и (яблоки, енота)
Знания о мире	Столица Франции – (Париж, Вена)
Классификация тональности	Я в восторге от декораций и игры актеров, спектакль был (хорошим, плохим)
Перевод	"Стол" по-английски будет ("table", "apple")
Пространственное мышление	Леша сидел на диване в гостиной, рядом с ним сидел Саша. Через 15 минут Саша встал и вышел из (гостиной, кухни)
Математика	Если прибавить 4 к 3, то будет (7, 8)

GPT > Трансформер

- Оказалось, что модели вида GPT работают лучше даже для многих seq2seq задач, чем **encoder-decoder** Трансформеры при одинаковом размере
- Объясняется это тем, что лучше выучить одну большую **decoder**-модель, чем разделить веса на две модели

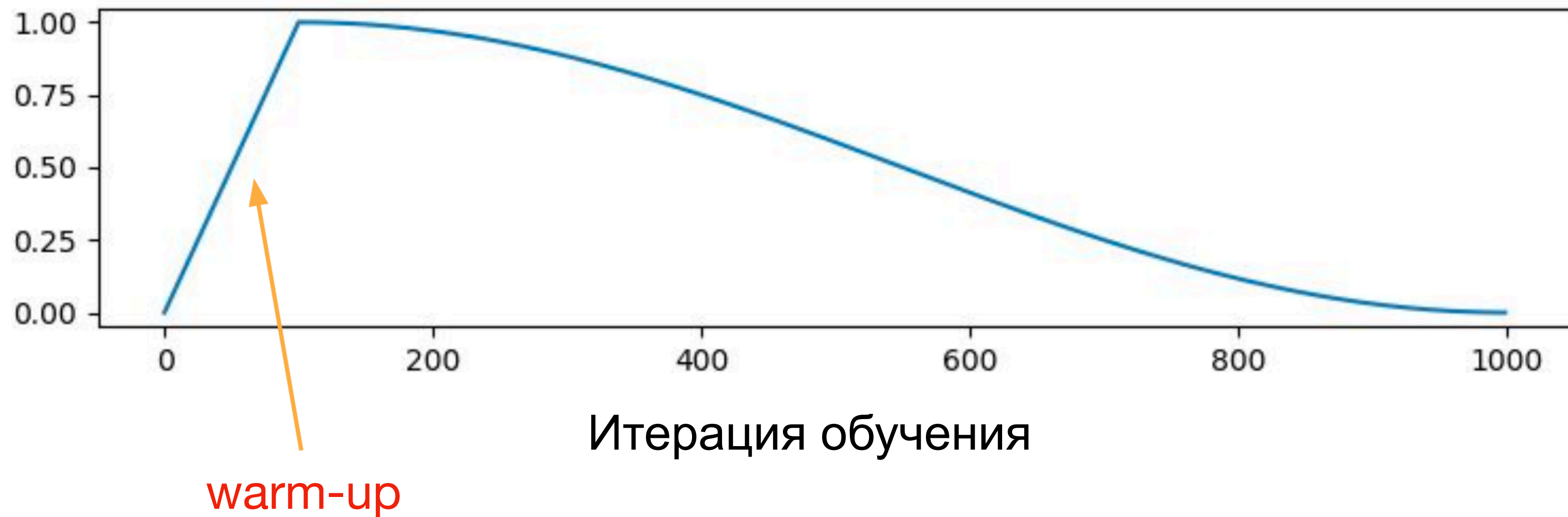


Трюки для обучения Трансформеров

Warm-up

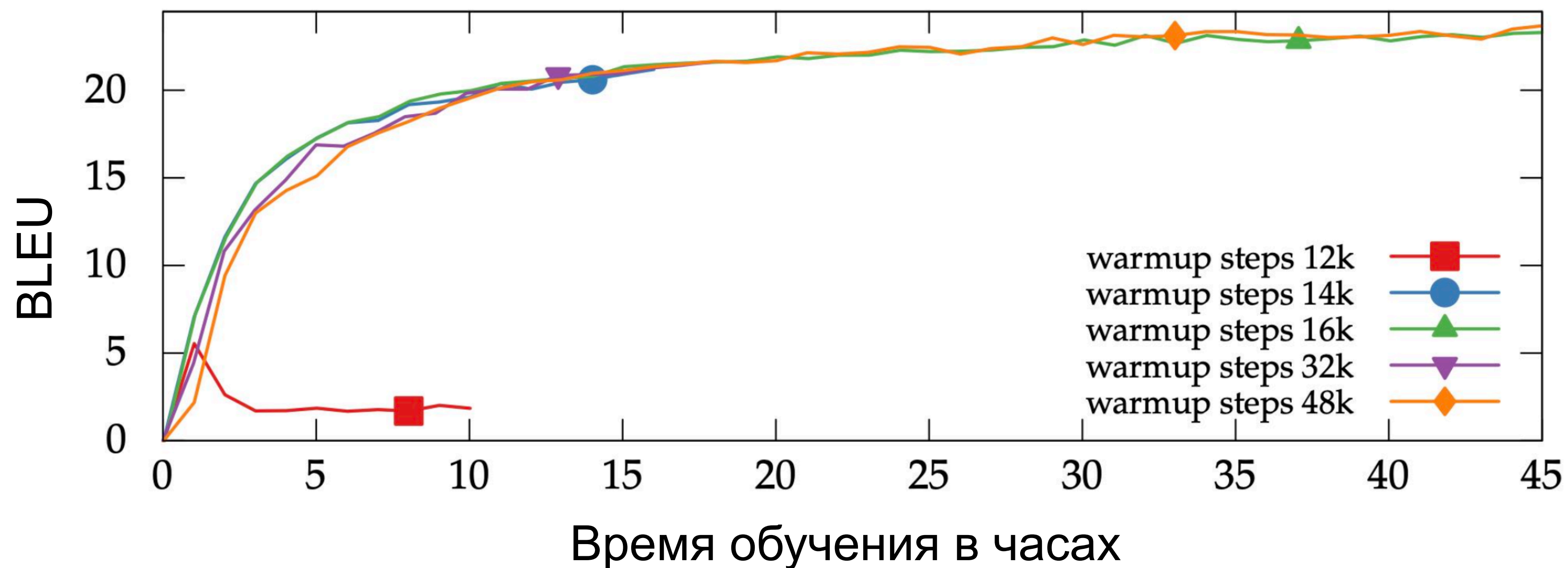
Постепенное увеличение скорости обучения на ранних шагах оптимизации.

График изменения Learning rate



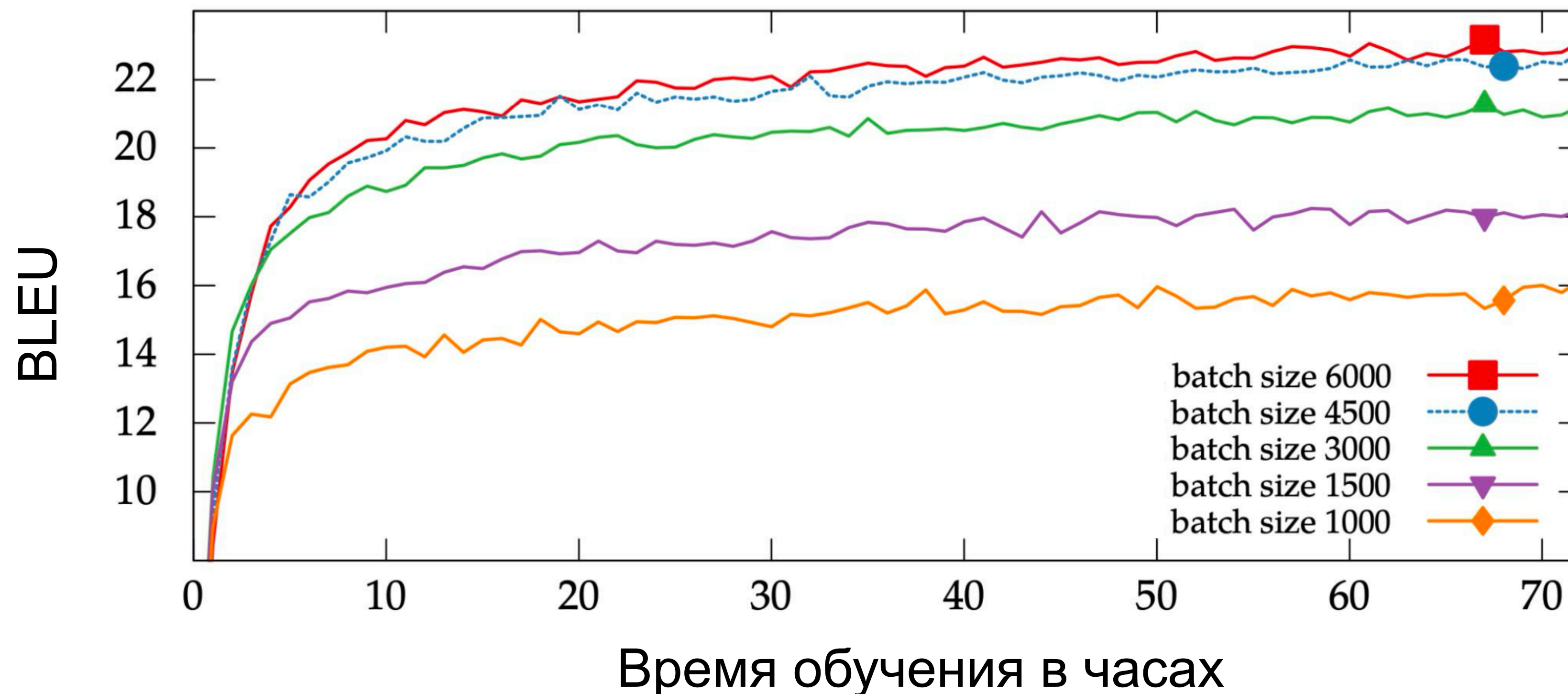
Warm-up

- Трансформеры не обучаются без warm-up!
- Чем глубже Трансформер, тем нужнее warm-up
- Без warm-up градиенты начинают затухать



Размер батча

- Обычно с небольшим размером батча трансформер хуже учится
- Но не всегда!
- Оптимальный размер батча зависит от размера модели



Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)
- Gradient clipping
- Аккумуляция градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных

Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)
- Gradient clipping
- Аккумуляция градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных

Из-за self-attention разные параметры получают разные по норме градиенты.

Adam настраивает шаг обучения для каждого параметра отдельно

Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)

- Gradient clipping

Защищает от взрыва градиентов

- Аккумуляция градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных

Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)
- Gradient clipping
- Аккумуляирование градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных



Полезно, если большой батч не
влезает в память

Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)
- Gradient clipping
- Аккумуляция градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных

Редко применяется, но
позволяет ускорить обучение

Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)
- Gradient clipping
- Аккумуляция градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных



Уменьшает затраты по памяти

Дополнительные трюки

- SGD работает гораздо хуже, чем Adam (AdamW, LAMB, AdaFactor, ...)
- Gradient clipping
- Аккумуляция градиентов
- Группировка текстов по длине в батчах
- Mixed-precision
- Очистка данных

Данные должны максимально соответствовать downstream задаче

Чистые данные избавлены от

- Жаргонных текстов
- Ложной информации
- Специальных символов разметки