Transformers

Natural Language Processing

Ника Зыкова, 2025/11/03

Организационное: ссылки

Гитхаб

Чат в –-

Организационное

Формула оценки:

Оценка = 0.6 * домашние работы (3 шт) + 0.4 * итоговый проект (защита на сессии)

План курса (может чуть-чуть поменяться):

1. Transformers
2. База про LLM: цикл обучения
3. Специальные языковые модели (MoE, reasoning, агенты)
4. Дообучение: оптимизации (адаптеры, квантизация)
5. Дообучение: подходы (SFT, RLHF, DPO)
6. История вопроса: более старые подходы и задачи
7. Интерпретация языковых моделей
8. *Дополнительная тема*

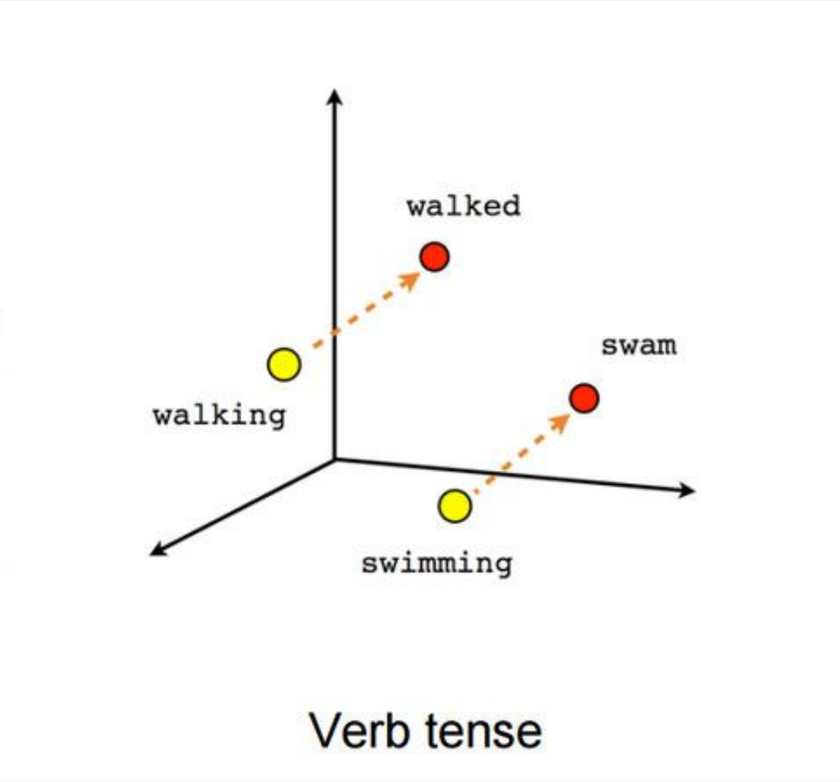
Курс в таком формате читается первый раз, не стесняйтесь задавать вопросы!

word2vec

В 2013 году был представлен word2vec, который дал нам принципиально новый способ векторизации текстов.

Однако, у него есть несколько недостатков:

- Плохо понимает разницу между синонимами и антонимами;
- Омонимы - это один и тот же вектор;
- Не учитывает порядок слов;
- Не учитывает синтаксис и сочетаемость слов;
- **Не учитывает контекст слов!**



https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.linkedin.com%2Fpulse%2Fword2vec-word-embeddings-used-nlp-vijay-ram&psig=AOvVaw2sEAHpli7v1qAqwCaIs2pJ&ust=1752431256077000&source=images&cd=vfe&opi=89978449&ved=0CBcQjhxqFwoTCOj8naj5t44DFQAAAAAdAAAAABAh

Память модели: RNN

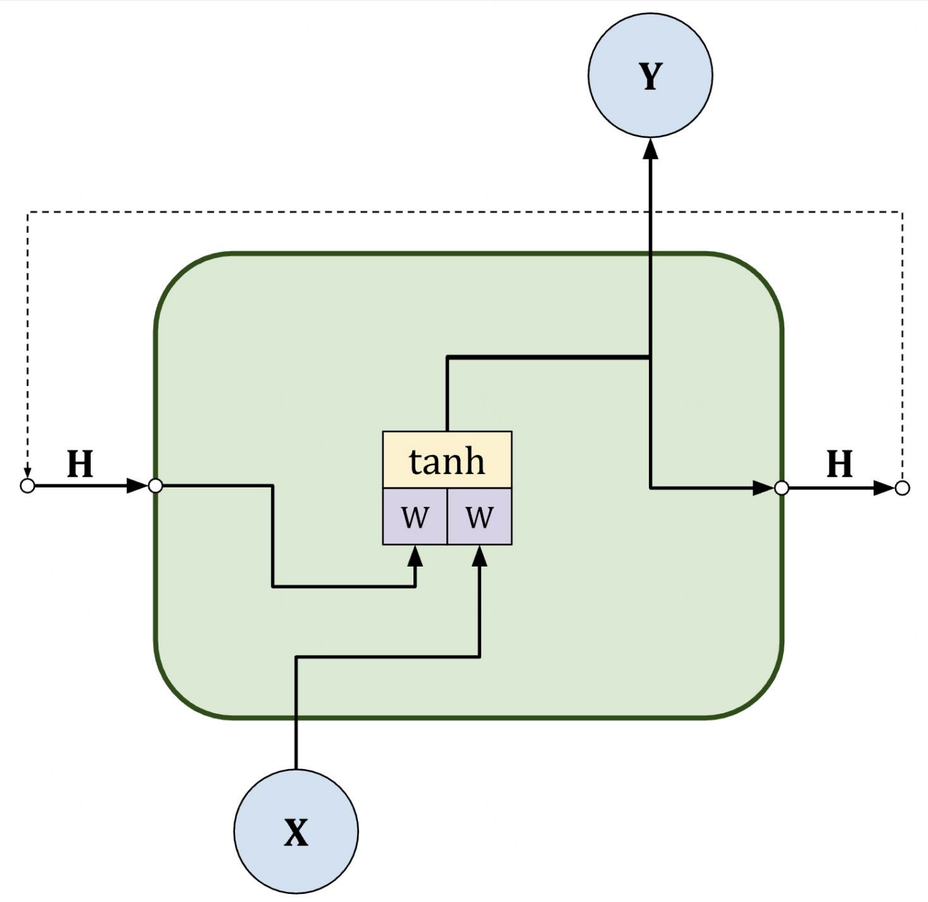
Идея следующая: давайте для векторизации текущего слова будем учитывать предыдущие.
Для этого введем скрытое состояние, которое будет отвечать за “память” модели:

$$H^{(t)} = \tanh\left(W^{hx} \cdot X^{(t)} + W^{hh} \cdot H^{(t-1)} + b_h\right)$$

$$Y^{(t)} = W^{yh} \cdot H^{(t)} + b_y$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

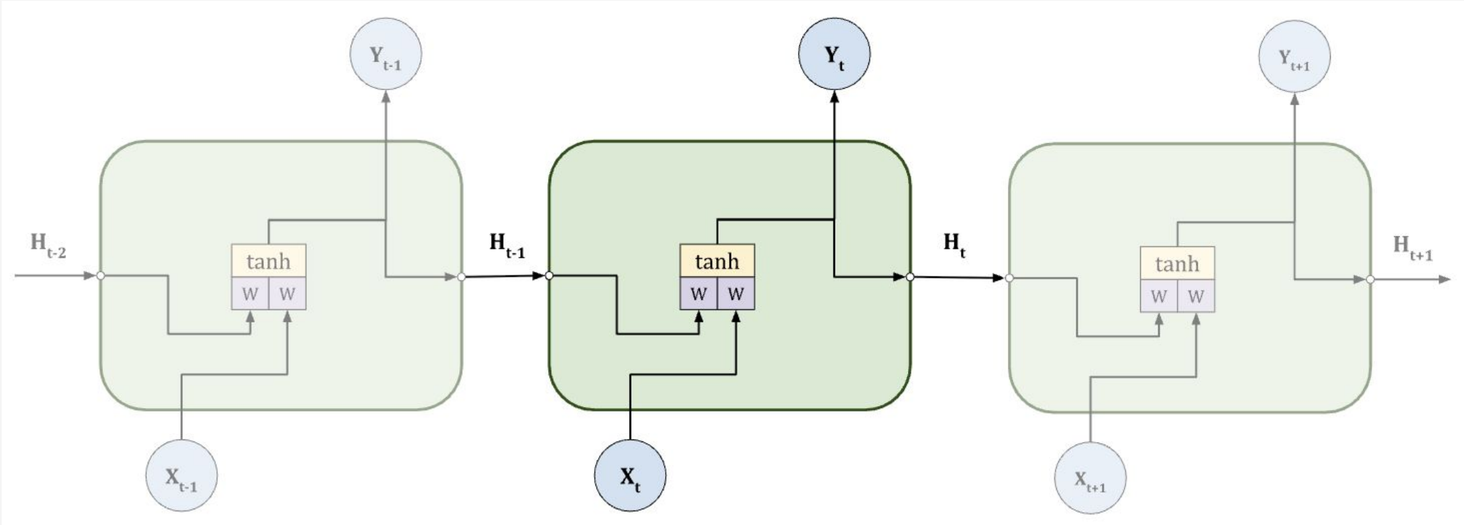
Проблема такого подхода в том, что модель не справляется с очень длинными текстами.



Для целого текста

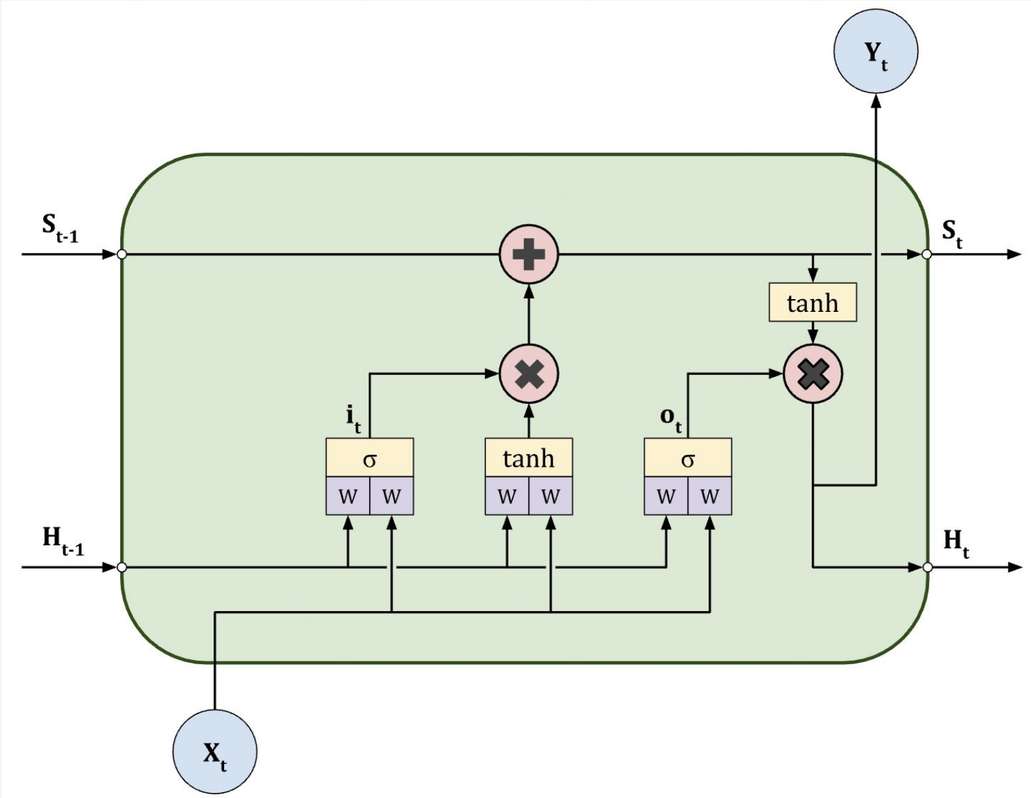
1. Задаем начальное скрытое состояние: нули, нормальное распределение или отдельный эмбеддинг;
2. Повторяем операцию RNN ячейки для каждого слова в тексте **последовательно.**

Почему последовательно и почему это плохо?

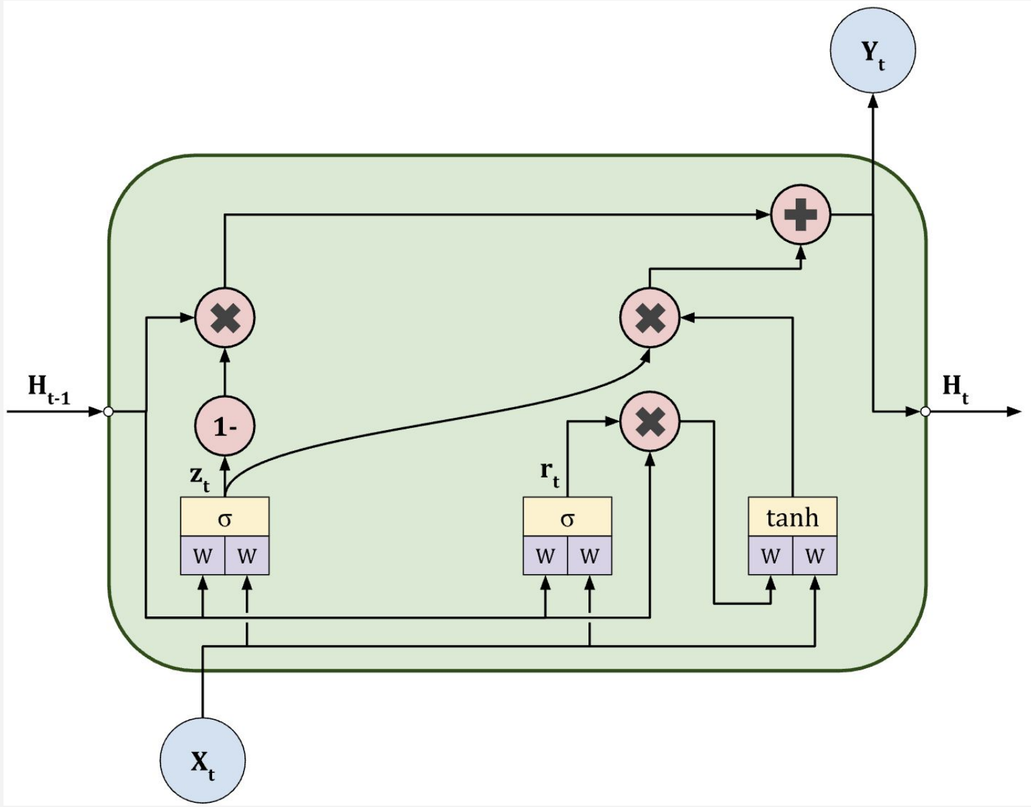


Забывание: LSTM и GRU

LSTM



GRU

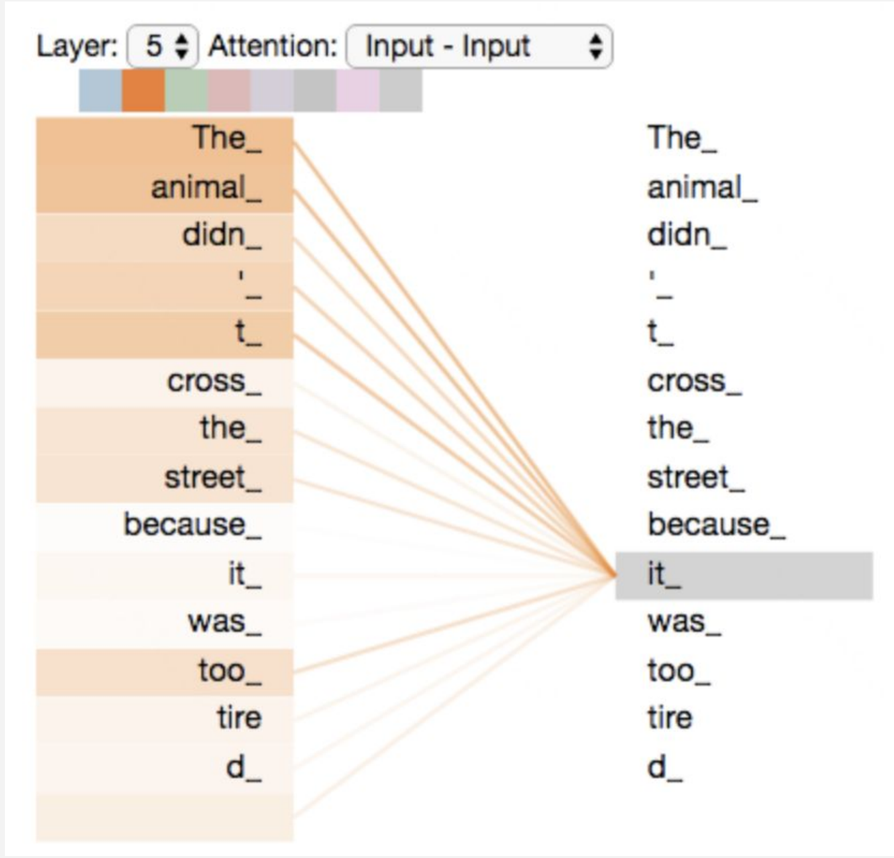


Attention is All You Need

У LSTM и GRU есть две большие проблемы:

- Они не умеет гибко учитывать контекст: предыдущий контекст либо важен целиком, либо нет. А слова находятся в более сложных отношениях обычно;
- Низкая производительность: операции со словами невозможно производить иначе, чем последовательно.

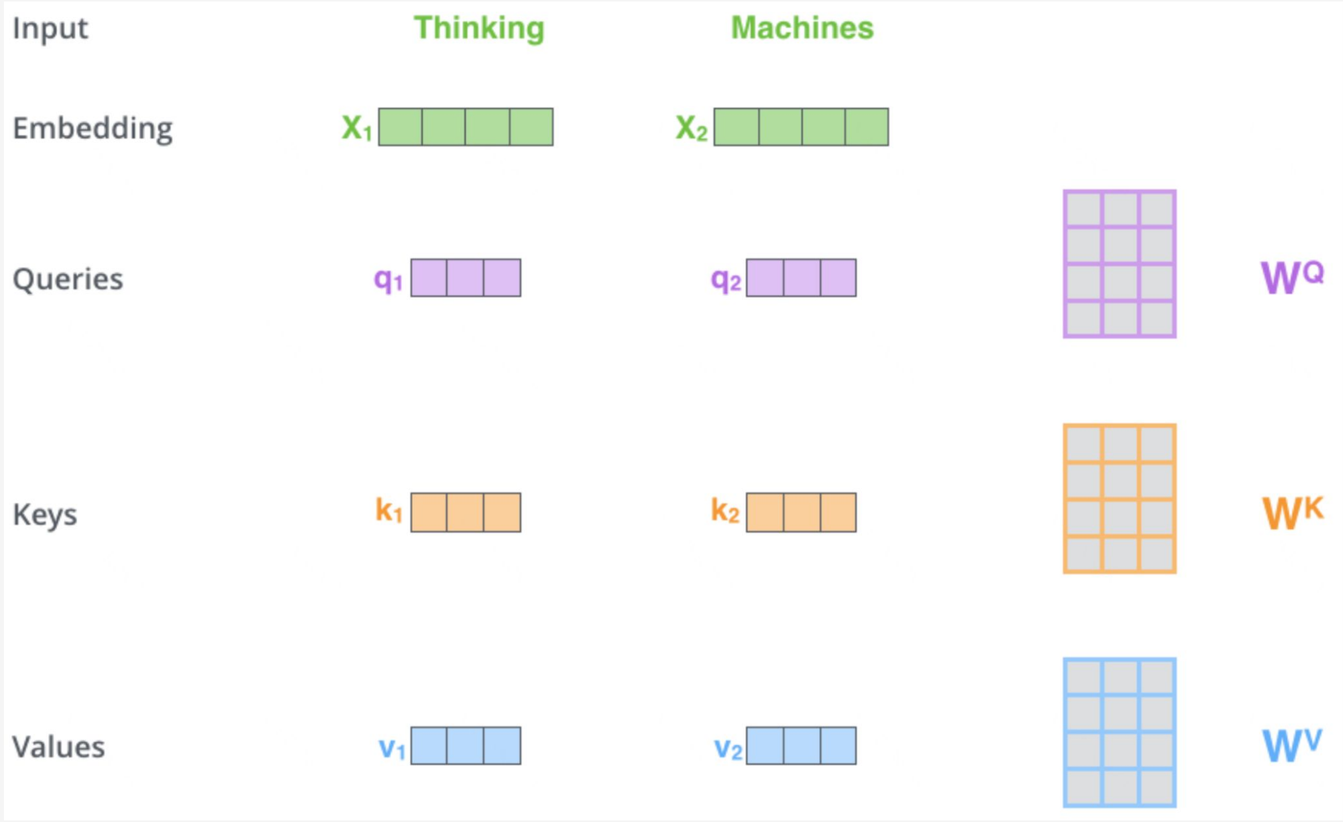
Механизм внимания, предложенный Google в 2017 году решил обе эти проблемы (но он не был первой попыткой в истории!)



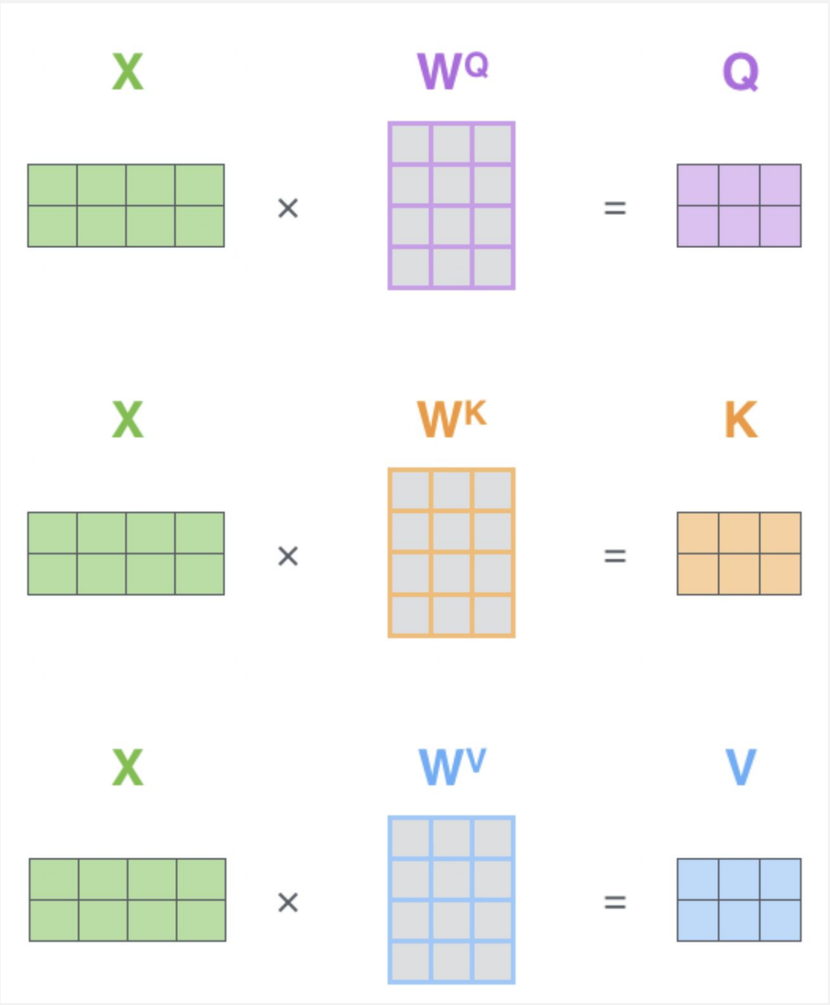
Attention подробнее

Для каждого исходного вектора мы хотим получить три других:

- Вектор запросов (Q): текущий элемент, для которого мы хотим вычислить внимание;
- Вектор ключей (K): опреляет, насколько текущий элемент связан с другими;
- Вектор значений (V): фактическая информация, которая будет передана по результату подсчета внимания.



Attention подробнее



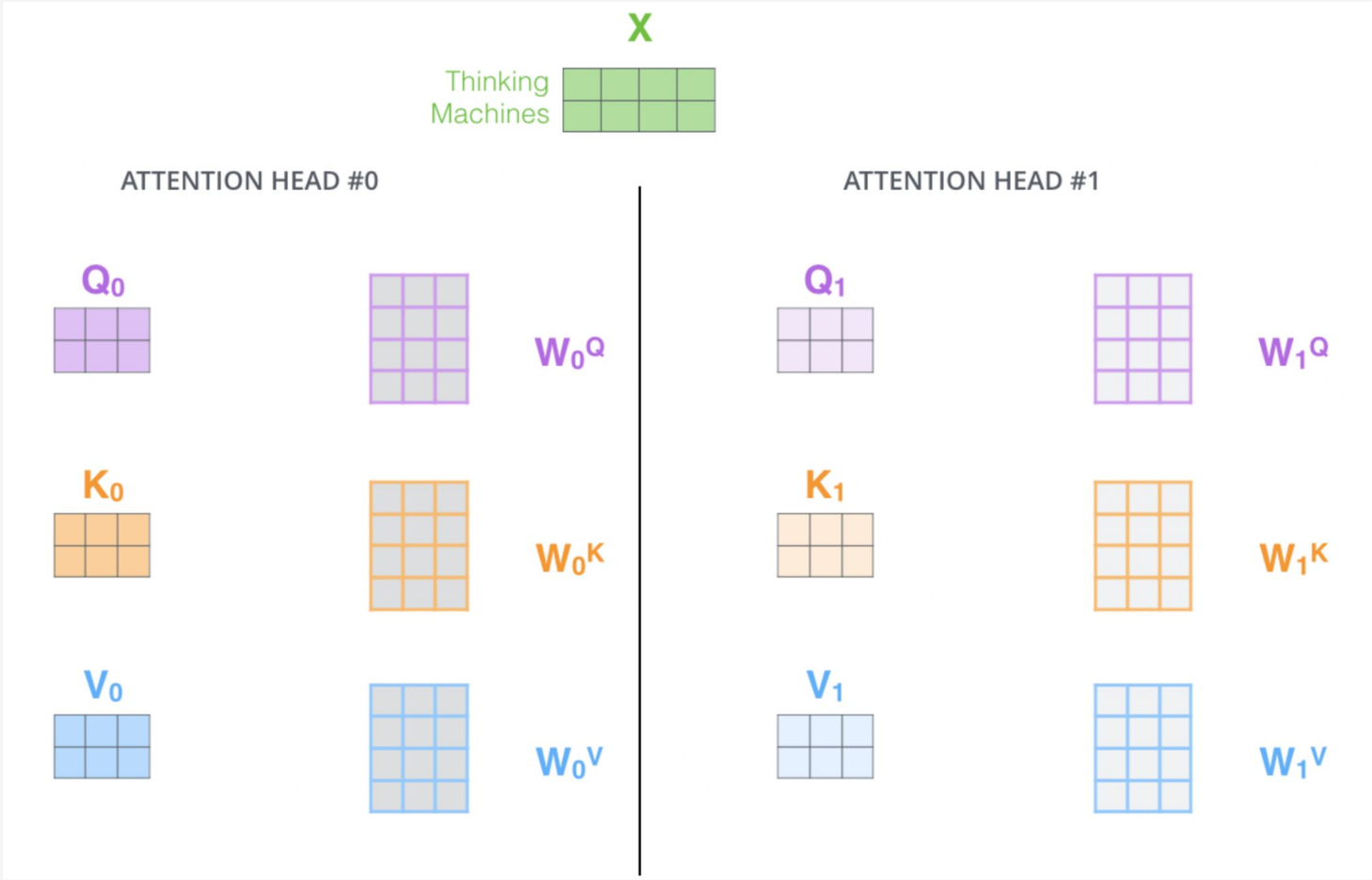
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

sqrt(d_k) - квадратный корень из размерности ключей помогает стабилизировать обучение

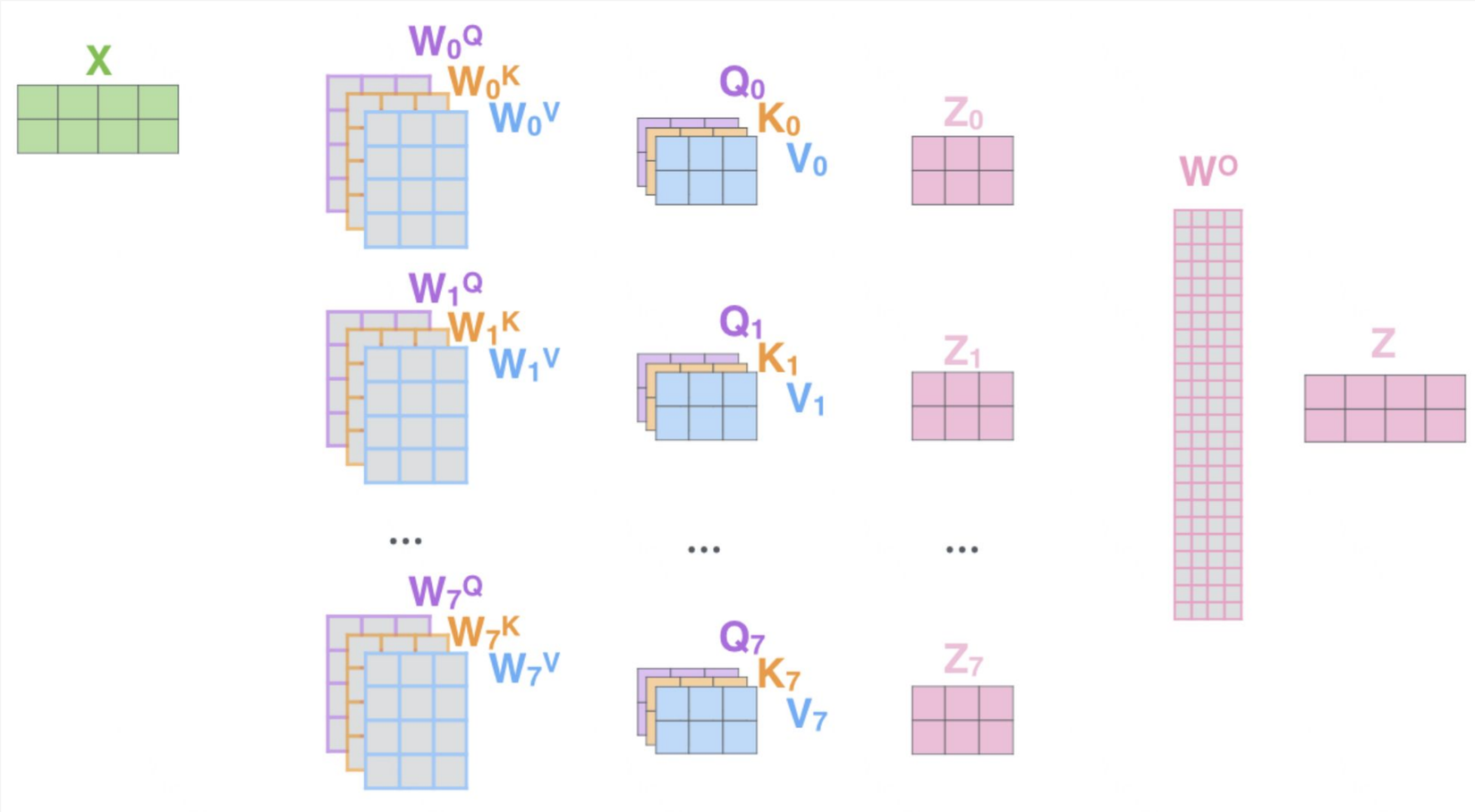
Attention подробнее

Обычно в механизме внимания делают несколько голов: параллельно считают одно и то же с разными матрицами весов, чтобы дать модели возможность учитывать разного уровня связи между словами.



Attention подробнее

В итоге это все складывается в параллельные операции с одним большим матричным умножением в конце.



Attention еще подробнее

У нас есть исходные эмбеддинги, где T - длина текста, d_{model} - размерность эмбеддинга:

$$X \in \mathbb{R}^{T \times d_{\text{model}}}$$

Считаем для них матрицы запросов, ключей и значений (i - номер головы):

$$Q_i = XW_Q, \quad K_i = XW_K, \quad V_i = XW_V$$

$$W_Q^i, W_K^i, W_V^i \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$$

$$d_{\text{head}} = \frac{d_{\text{model}}}{h}$$

Результирующие матрицы имеют размерность

$$Q^i, K^i, V^i \in \mathbb{R}^{\mathrm{T} \times d_{\text{head}}}$$

Attention еще подробнее

Считаем внимание для каждой головы:

$$\text{Attention}_i(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_{\text{head}}}}\right) V_i$$

Конкатенируем все головы в одну матрицу:

$$\text{Concat}(\text{head}_1, \ldots, \text{head}_h) \in \mathbb{R}^{T \times (h \cdot d_{\text{head}})} = \mathbb{R}^{T \times d_{\text{model}}}$$

Финальная линейная операция:

$$\text{Output} = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O$$

$$W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}, \text{Output} \in \mathbb{R}^{\text{T} \times d_{\text{model}}}$$

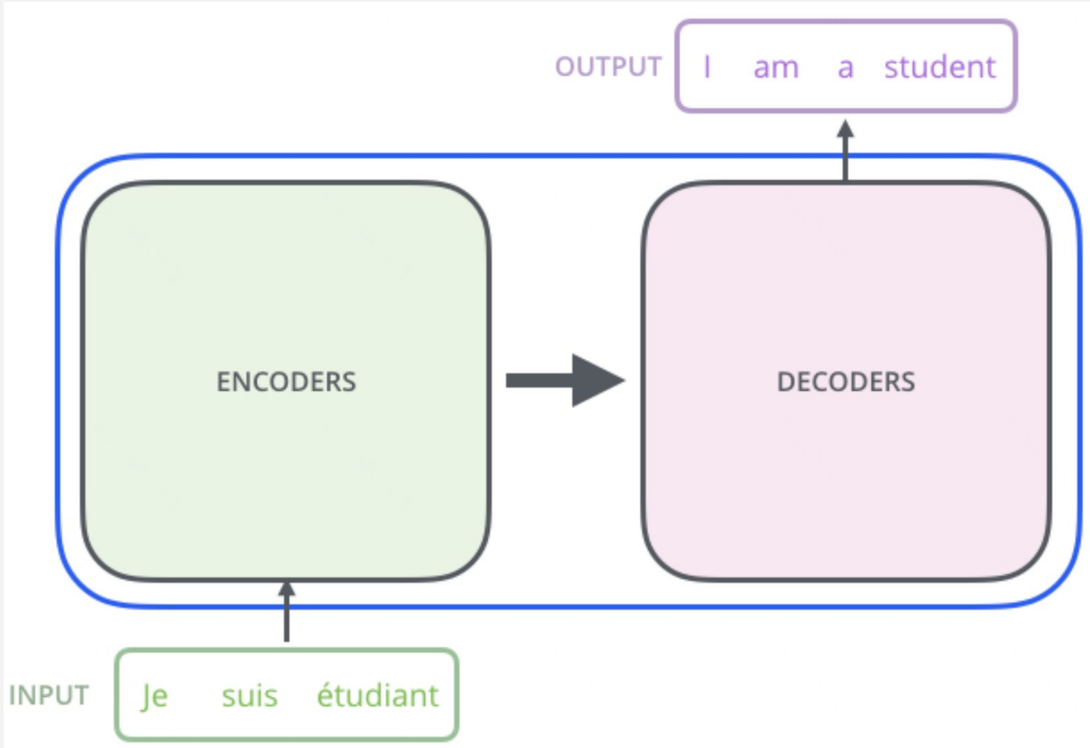
Как на входе!

Где это все в модели?

Полноценный трансформер состоит из двух частей:

- Энкодера: принимает на вход последовательность и максимально полно передает ее смысл в эмбеддингах;
- Декодера: генерирует выходную последовательность на основе того, что посчитал энкодер, и того, что уже было сгенерировано до этого.

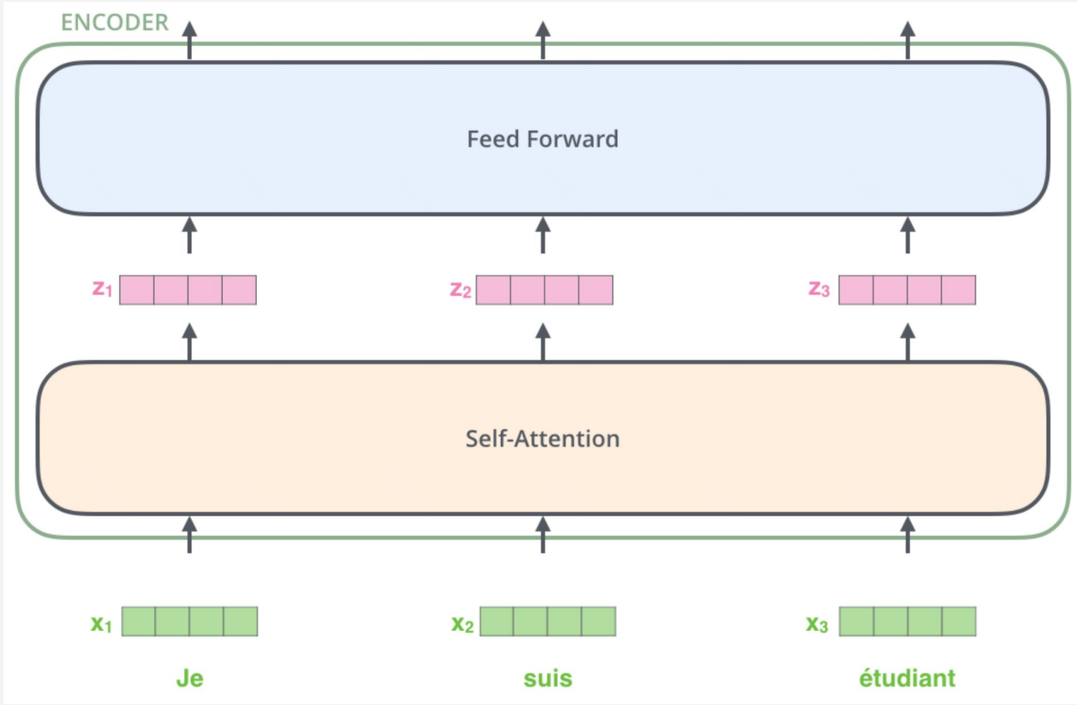
Помним, что изначально архитектуру придумывали для машинного перевода!



Где это все в модели?

Механизм внимания есть в энкодере и декодере: фактически, то, что мы с вами обсудили, и есть основное преобразование, которое делает энкодер, и называется **self-attention** .

В декодере дополнительно есть **cross-attention** - расчет внимания, где матрицы ключей и значений приходят из энкодера, а матрица запросов - из декодера (декодер спрашивает у энкодера, какие слова важны для генерации нового, и использует информацию из них).



Feed Forward

Два линейных преобразования с размерностью больше, чем у модели в целом:

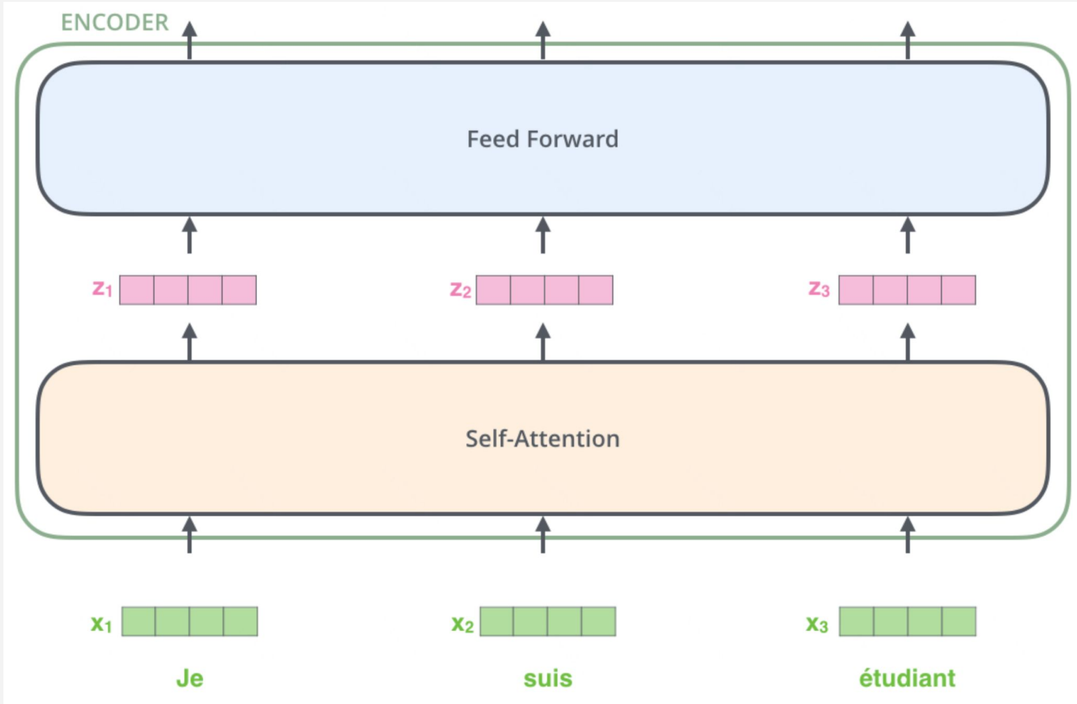
$$\mathrm{FFN}(x_t) = W_2\,\sigma(W_1 x_t + b_1) + b_2$$

Где

$$W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}, \quad W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$$
$$b_1 \in \mathbb{R}^{d_{\text{ff}}}, \quad b_2 \in \mathbb{R}^{d_{\text{model}}}$$

d_{ff} - размер скрытого слоя (обычно в 4 раза больше, чем d_{model})

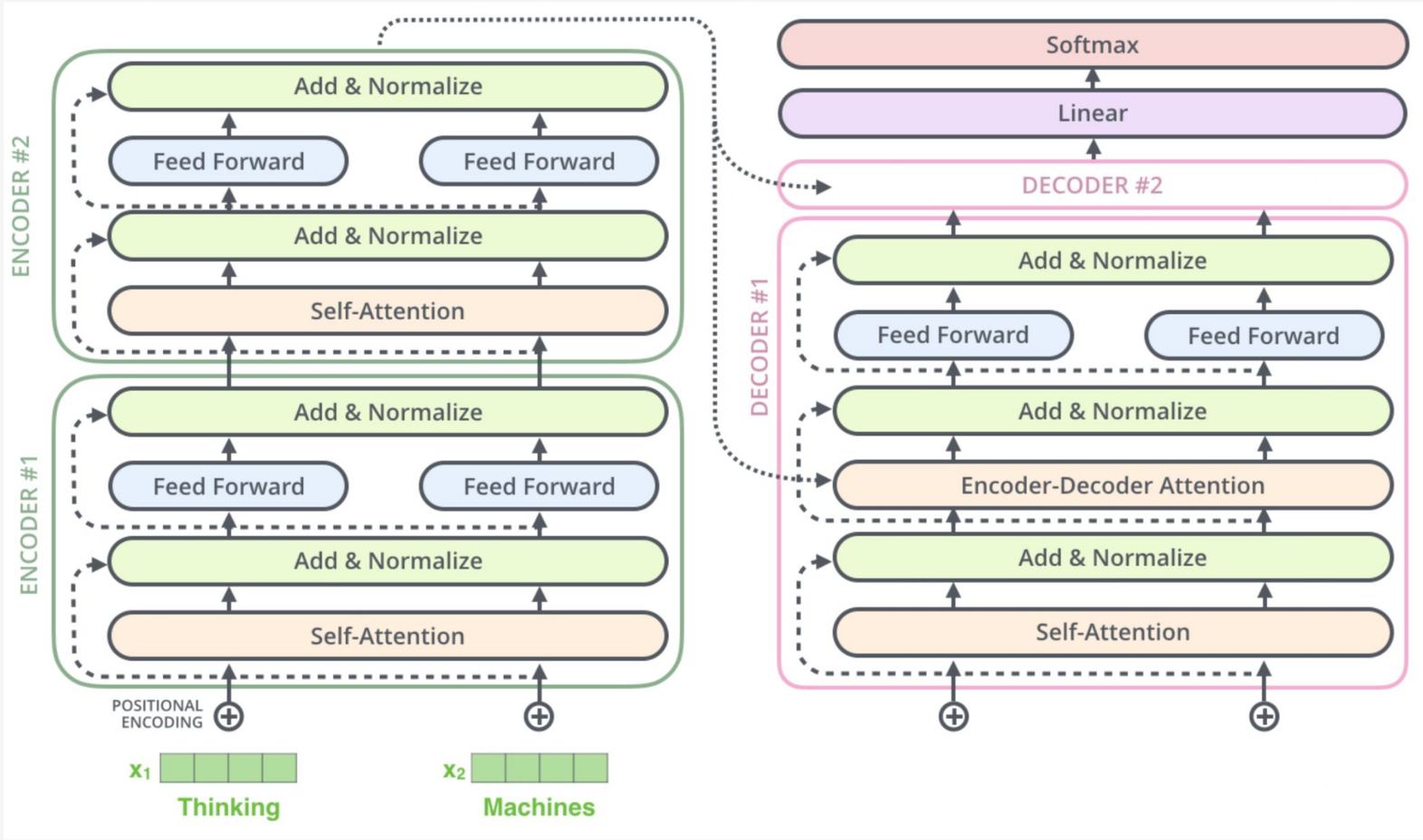
$\sigma(\cdot)$ - нелинейная функция (ReLU, GELU и т.п.)



Transformer целиком

Итоговый Трансформер состоит из последовательности преобразований с помощью механизма внимания.

И еще нескольких нюансов.



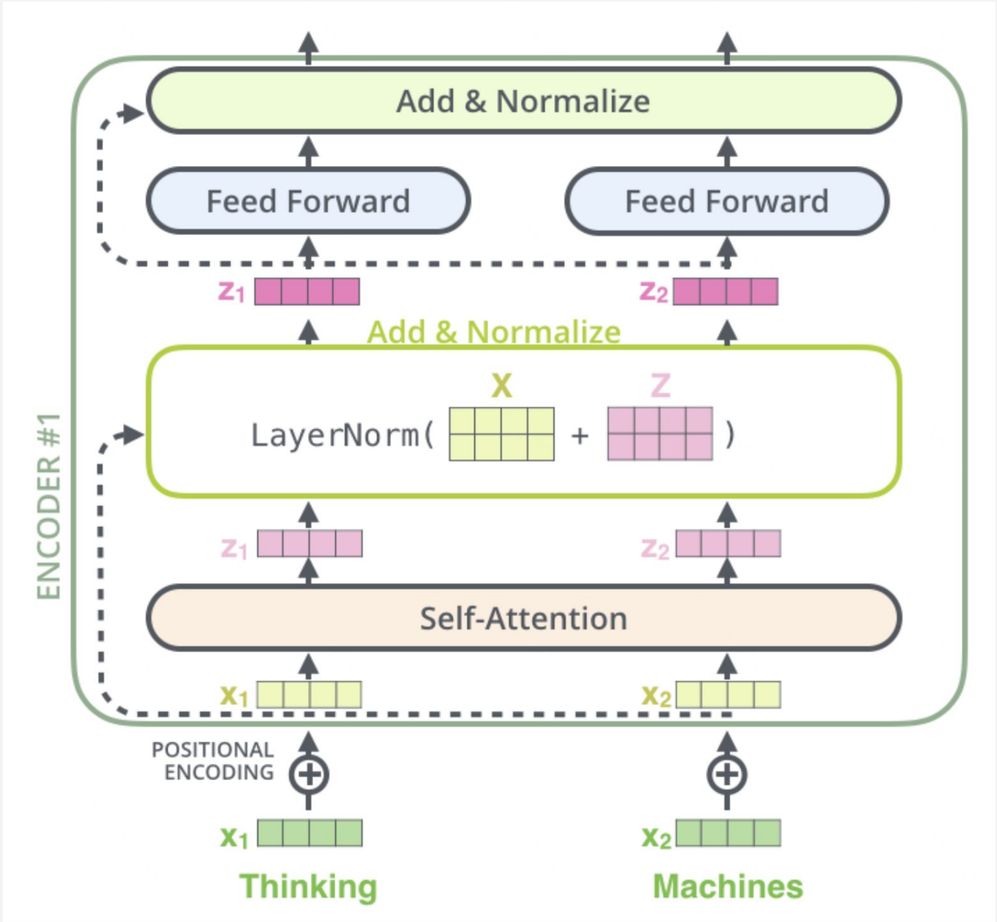
Нюансы: residuals

Чтобы избежать затухания градиента и стабилизировать обучение применяется прокидывание данных “мимо” внимания и FF-слоев.

Также в целях стабильности применяется нормализация:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \qquad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$$

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

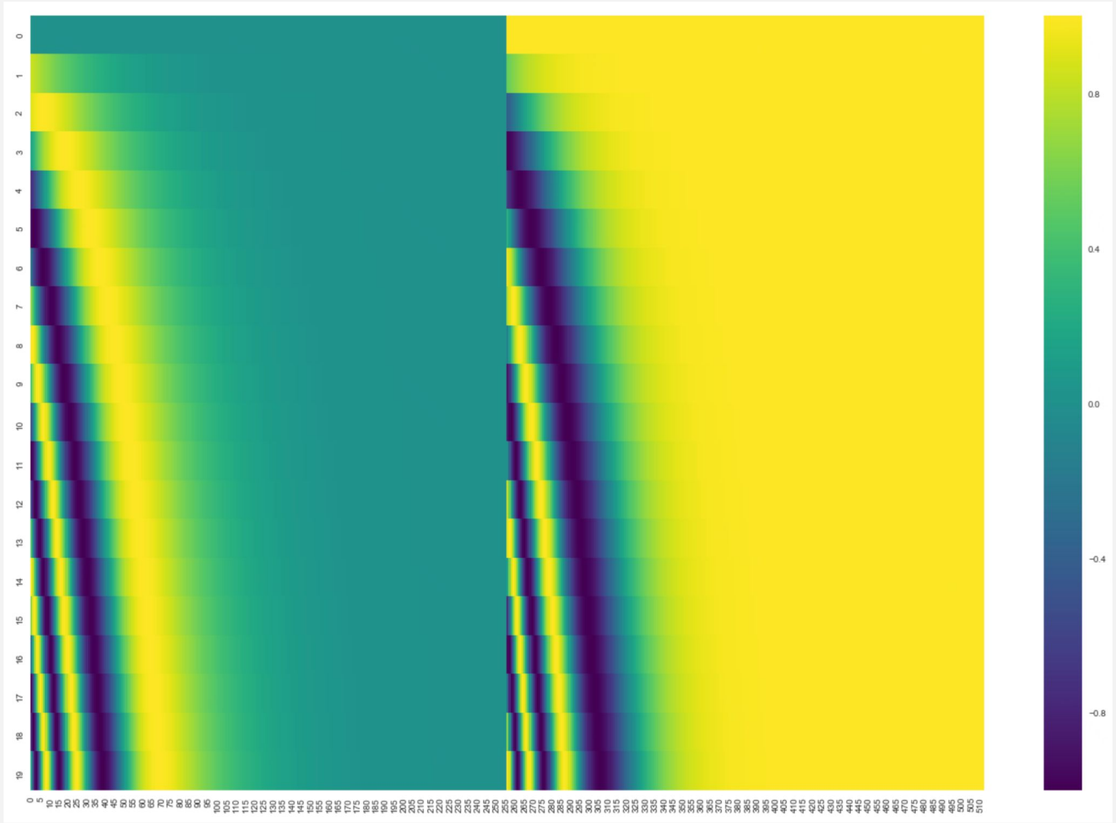


Нюансы: positions

Чтобы дать модели возможность учитывать порядок слов, были придуманы позиционные эмбеддинги, которые прибавляются в входным эмбеддингам в самом начале работы.

Они могут получатся двумя способами:

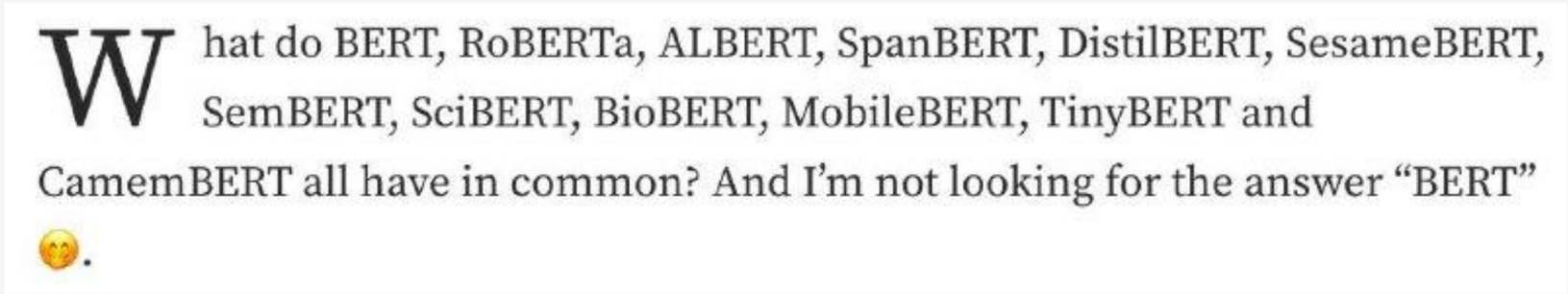
- Обучаемые: отдельная матрица эмбеддингов для позиций;
- Генерируемые: чаще всего с помощью синуса от функции от позиции.



В чем разница?

В разных моделях используют разные части архитектуры трансформера.

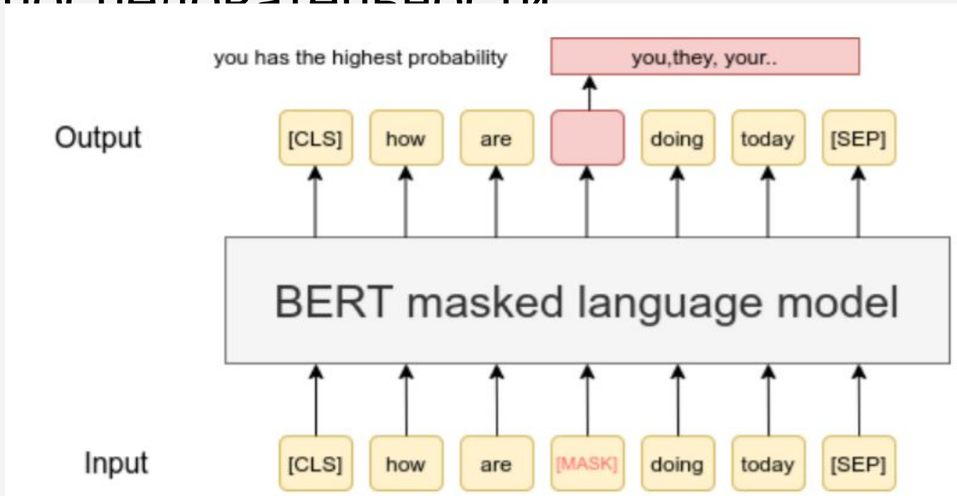
Encoder	Decoder	Encoder-Decoder
BERT	GPT1, 2, 3, 4	T5
RoBERTa	ChatGPT	Bart
ALBERT	PaLM	Pegasus
ELECTRA	LLaMA	ProphetNet



Encoder VS Decoder

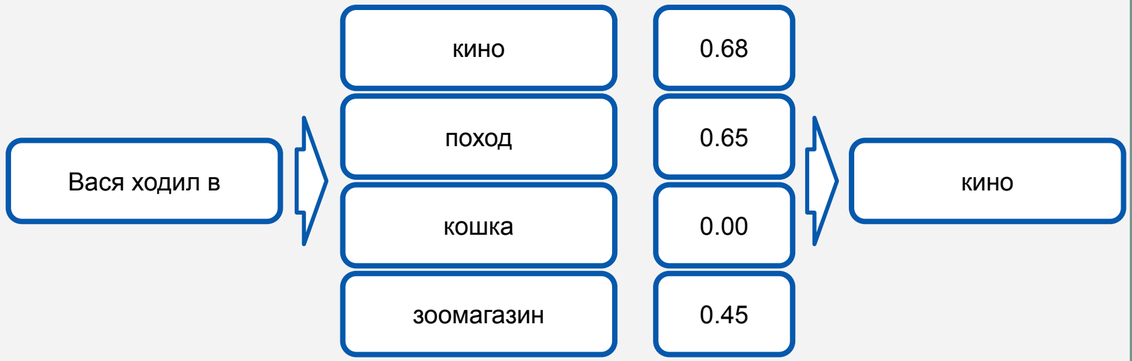
Encoder

- Слой эмбеддингов + позиционные эмбеддинги + несколько слоев энкодера;
- При векторизации конкретного слова смотрит на все слова в последовательности



Decoder

- Слой эмбеддингов + позиционные эмбеддинги + несколько слоев декодера;
- При векторизации смотрит только на уже сгенерированные слова;
- **Не содержит cross-attention** .



Почитать

- https://jalammar.github.io/illustrated-transformer/ - про Трансформеры
- http://vbystricky.ru/2021/05/rnn_lstm_gru_etc.html - про RNN, LSTM и Co
- https://arxiv.org/abs/1706.03762v7 - Attention Is All You Need