

LLM-инженер

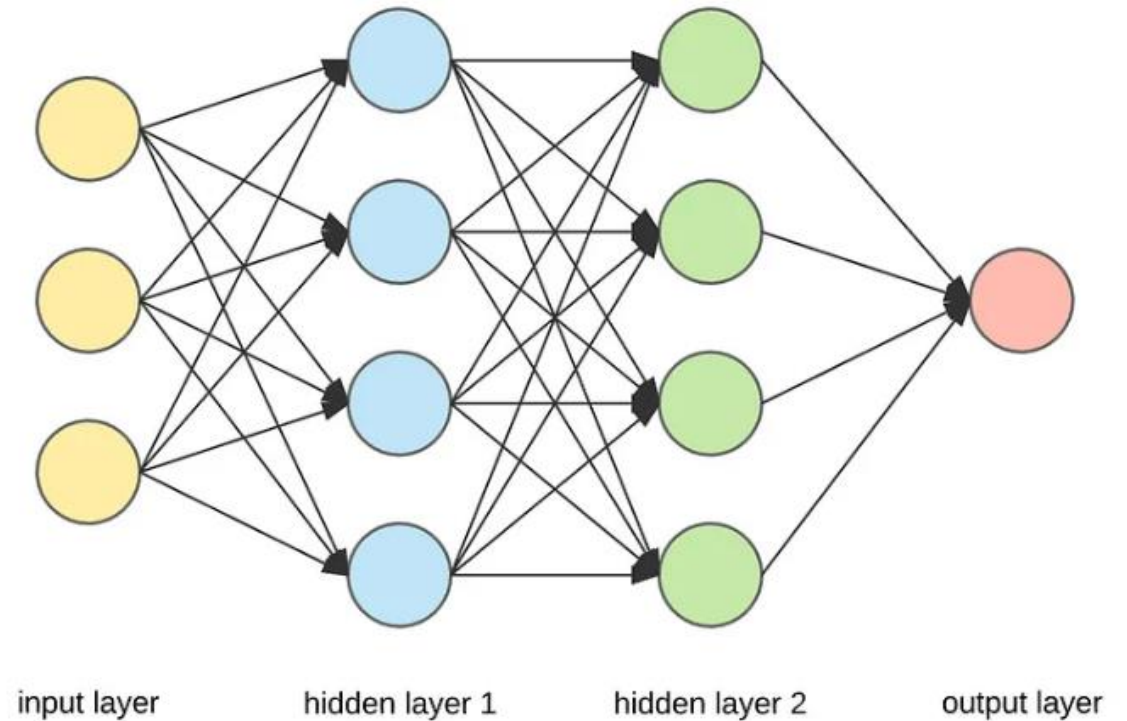
online course ✨

модуль #1

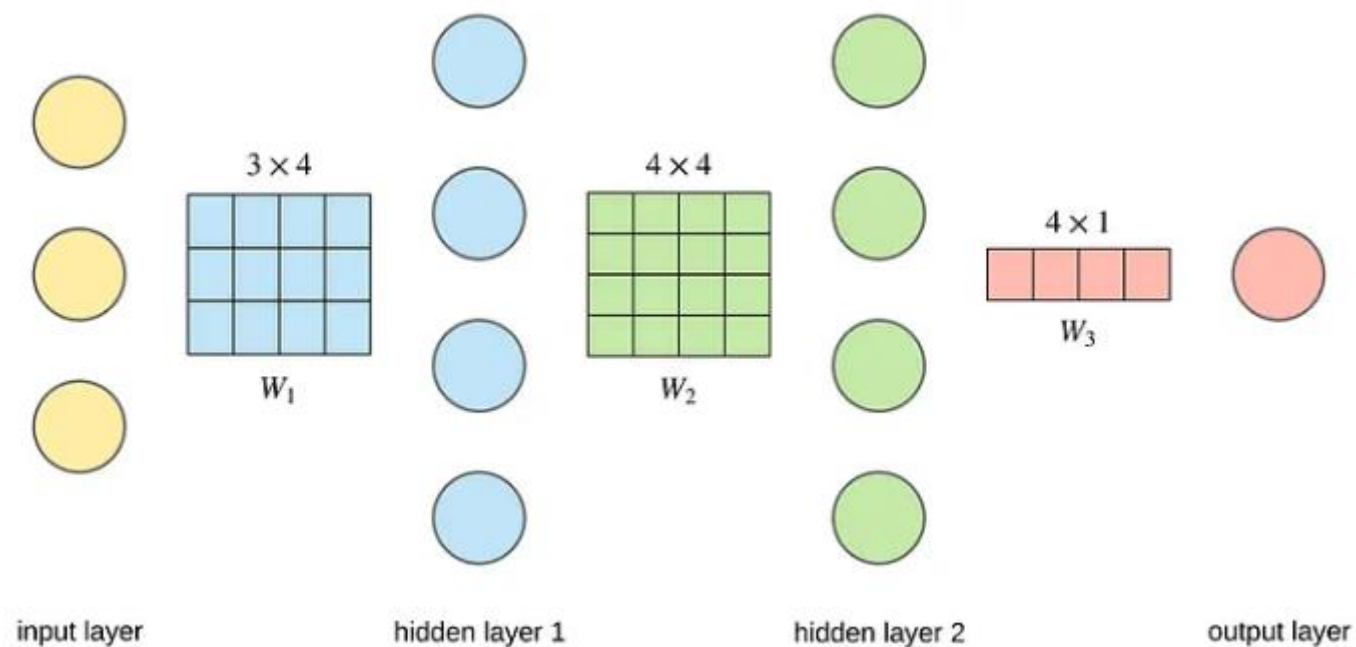
лекция: attention & трансформеры

Нейронные сети

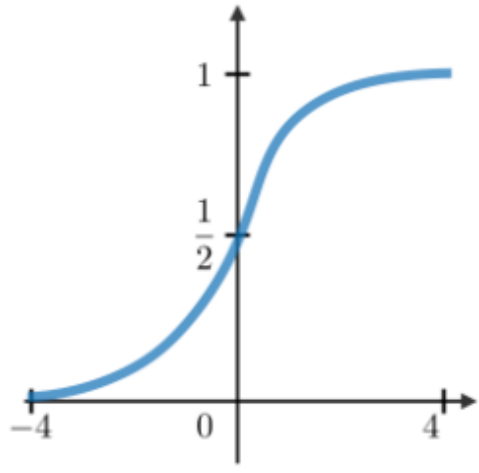
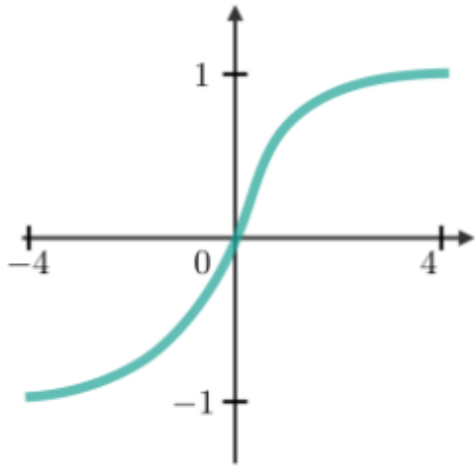
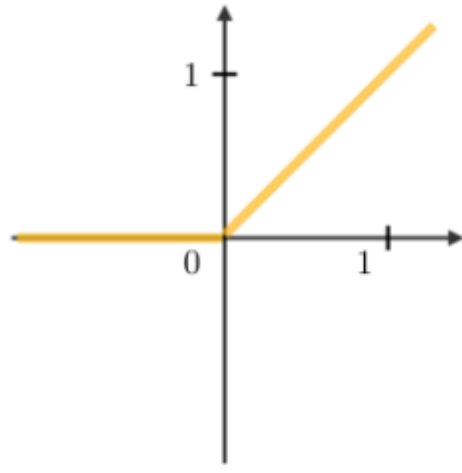
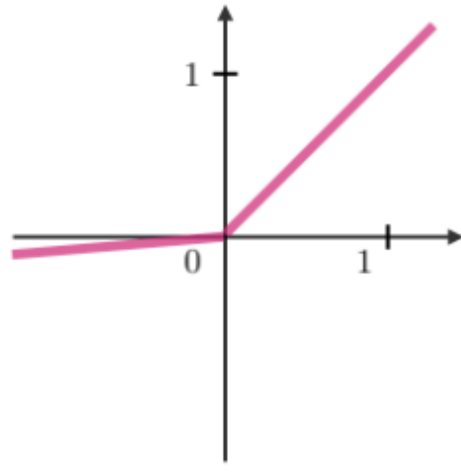
Искусственная нейронная сеть (ИНС) — упрощенная модель биологической нейронной сети, представляющая собой совокупность искусственных нейронов, взаимодействующих между собой.



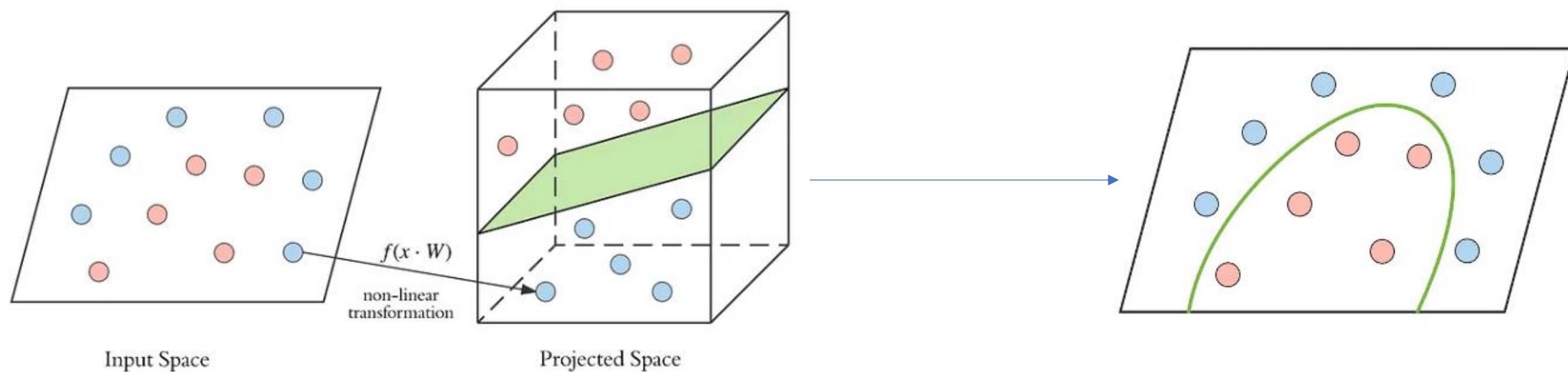
Матричный вид структуры ИНС



Функции активации

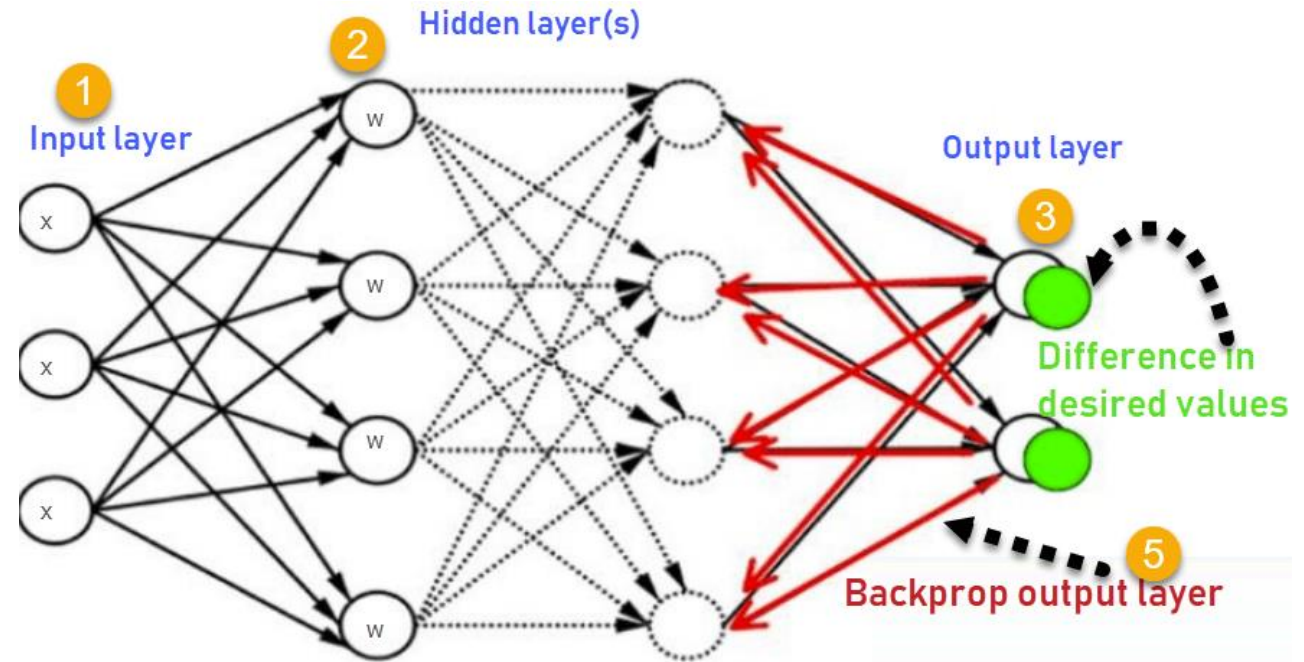
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

Нелинейность функций активации наглядно



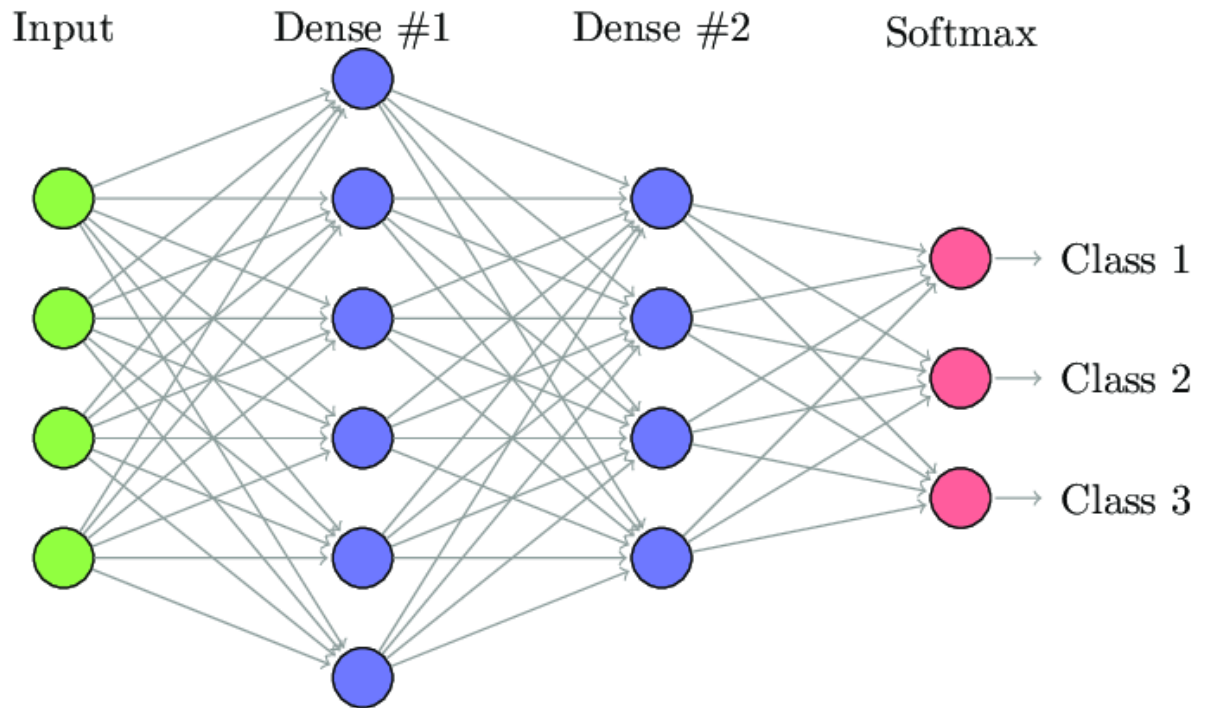
Обратное распространение ошибки

- На каждой итерации происходит два прохода сети — прямой и обратный.
- На прямом входной вектор распространяется от входов сети к ее выходам и формирует некоторый выходной вектор, соответствующий текущему (фактическому) состоянию весов.
- Затем вычисляется ошибка нейронной сети как разность между фактическим и целевым значениями.
- На обратном проходе эта ошибка распространяется от выхода сети к ее входам, и производится коррекция весов нейронов



Архитектуры ИНС – полносвязная сеть (FNN)

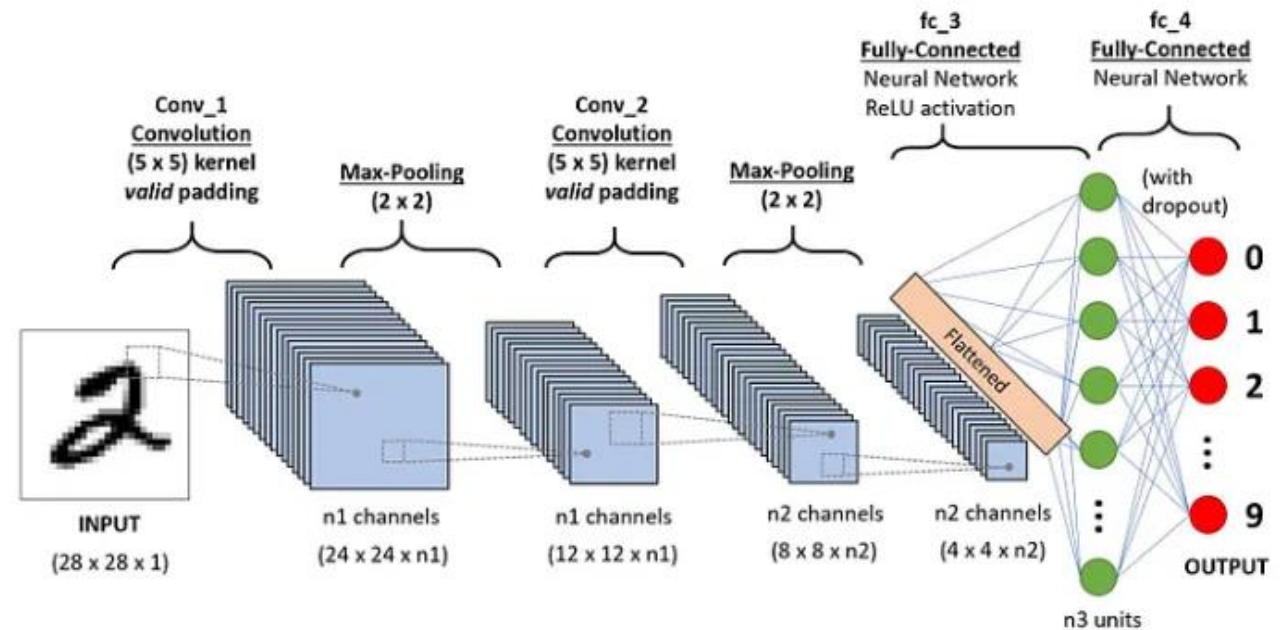
- Каждый нейрон связан со всеми нейронами предыдущего слоя
- У такой сети много параметров
- Трудно обучать



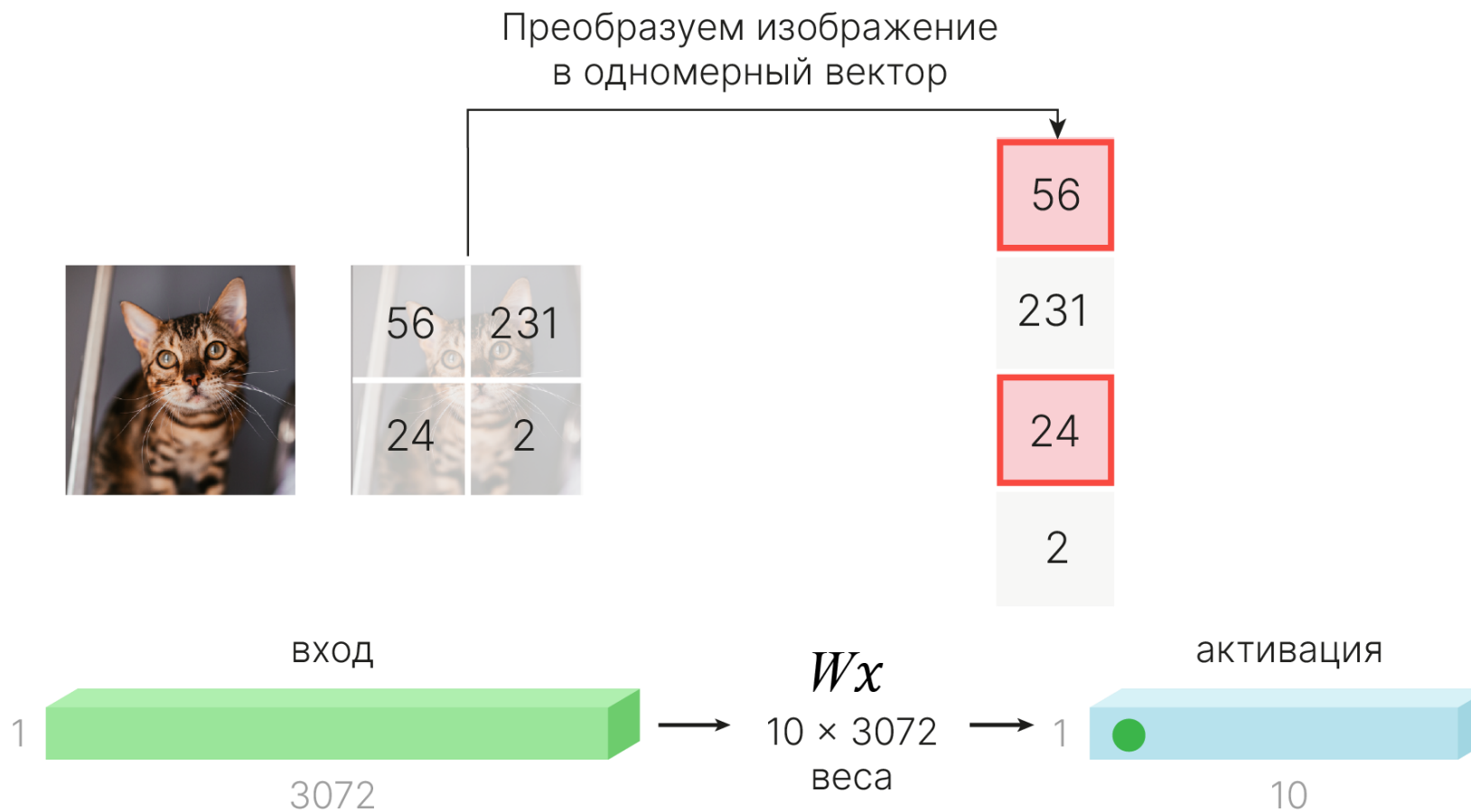
Архитектуры ИНС – сверточная сеть (CNN)

Внутри сверточной сети (CNN):

- Сверточные слои;
- Слои субдискретизации (Subsampling или Pooling), которые уменьшают размер изображения;
- Полносвязные слои.

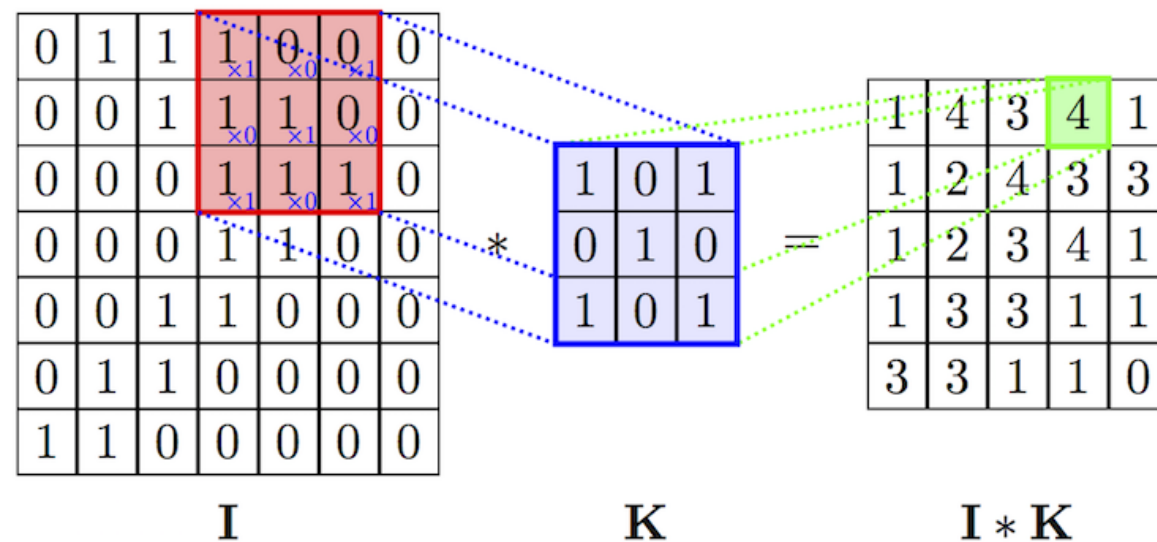


Можно ли без CNN?



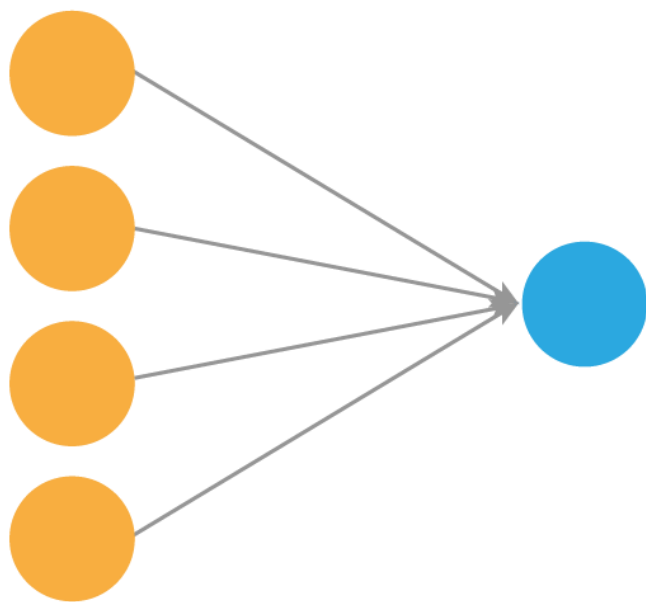
Операция свертки

Слой свёртки — основной блок CNN. Он включает в себя для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам (суммируя результаты поэлементного произведения для каждого фрагмента). Весовые коэффициенты ядра свёртки (небольшой матрицы) неизвестны и устанавливаются в процессе обучения.



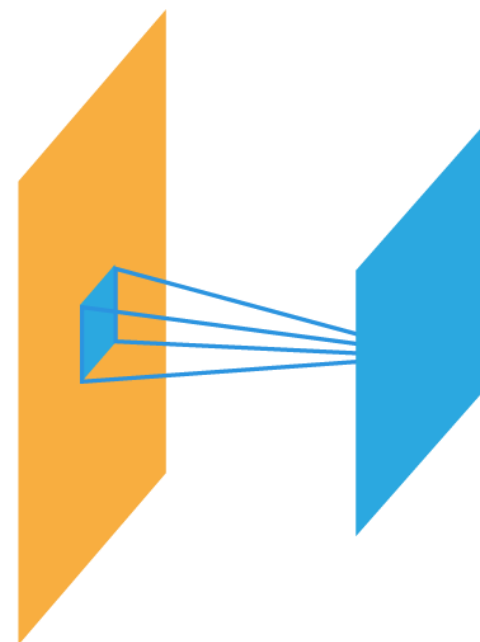
Линейный слой VS Сверточный

Полносвязный слой



$$\mathbb{R}^h \rightarrow \mathbb{R}^l$$

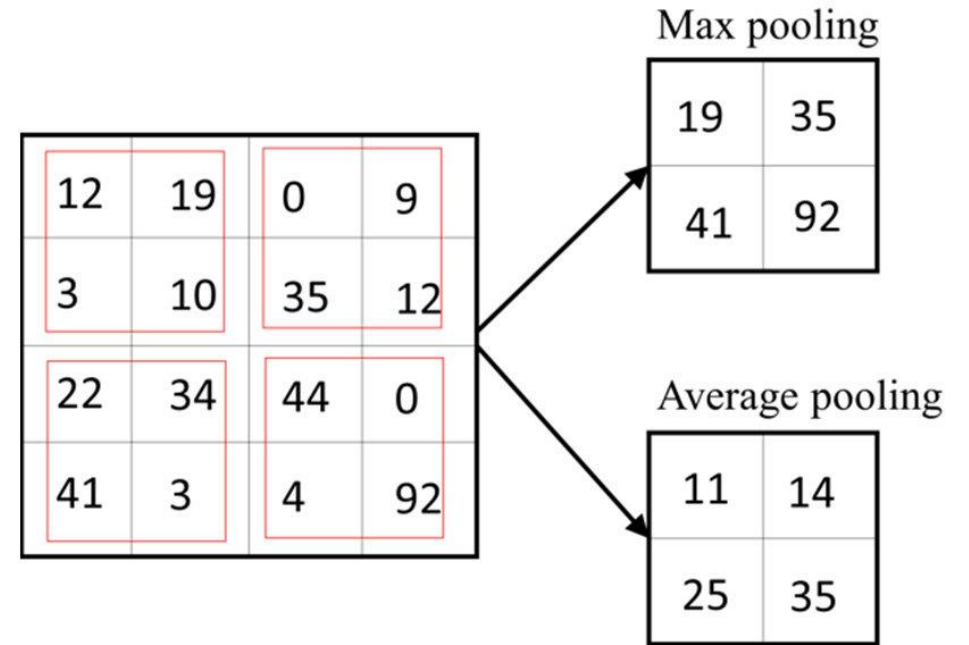
Свёрточный слой



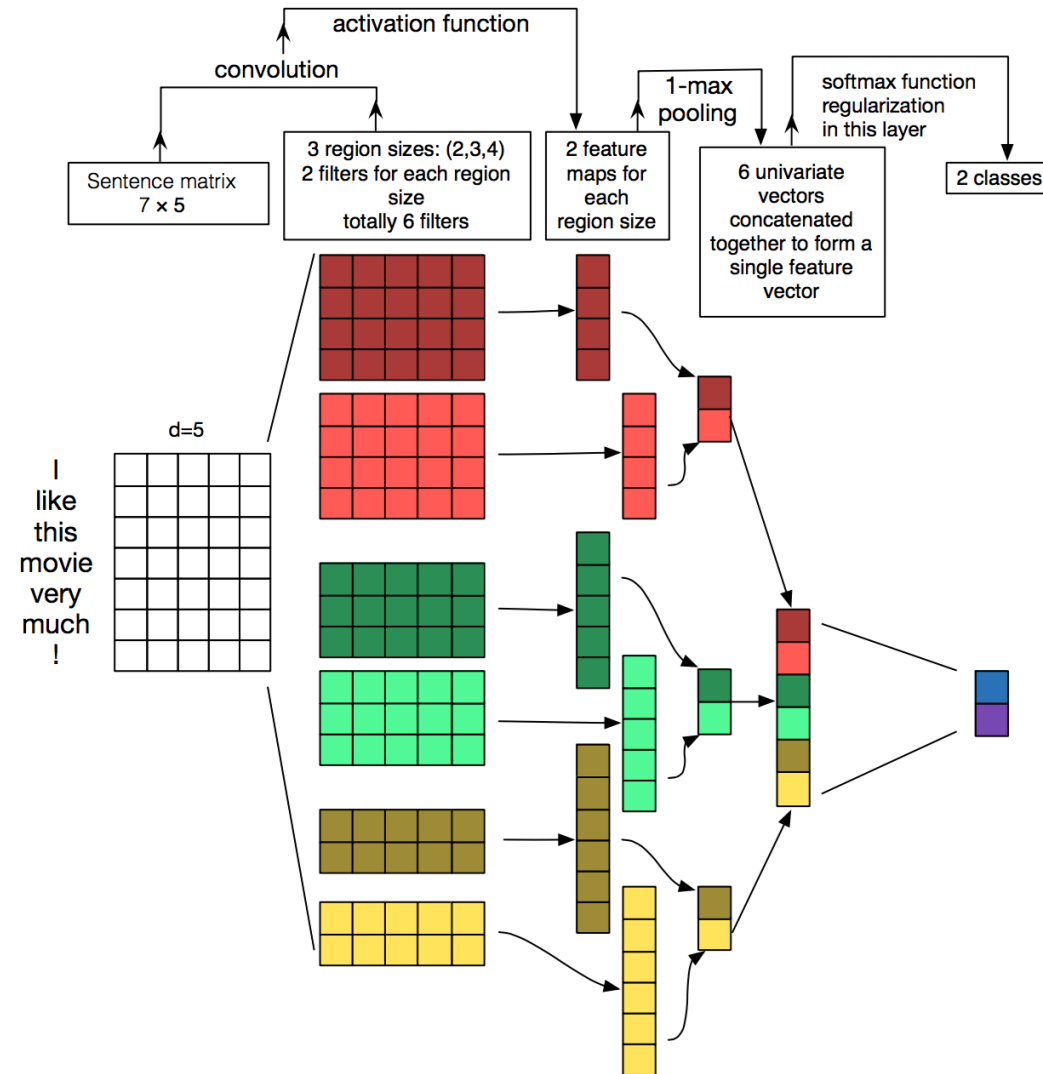
$$\mathbb{R}^h \times \mathbb{R}^w \rightarrow \mathbb{R}^{h'} \times \mathbb{R}^{w'}$$

Операция пулинга (pooling)

- Слой пулинга — представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей уплотняется до одного пикселя, проходя нелинейное преобразование.
- Операция пулинга позволяет существенно уменьшить пространственный объём изображения.
- Пулинг интерпретируется так: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться.

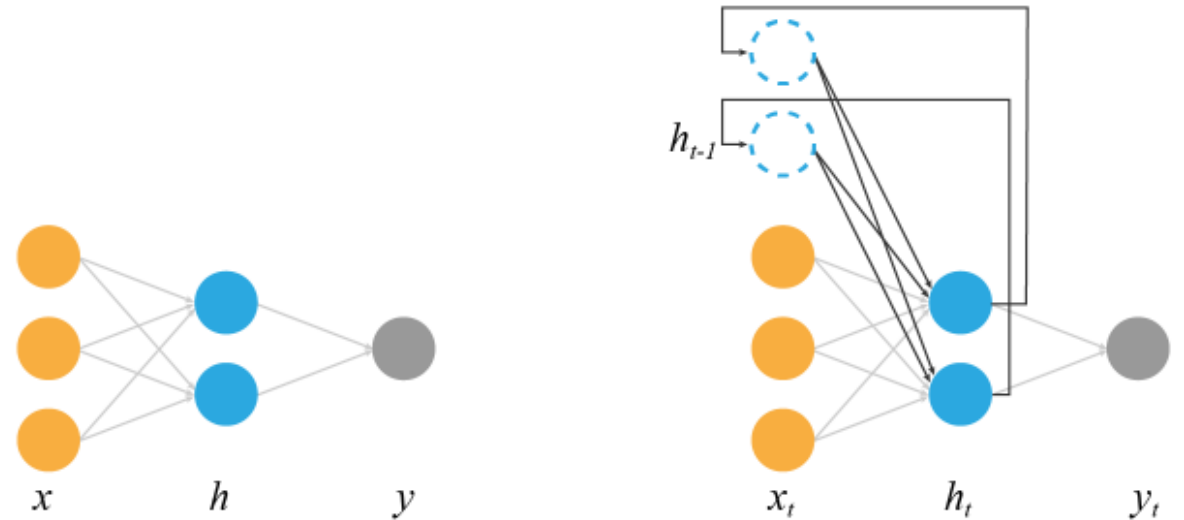


CNN B NLP



Архитектуры ИНС – рекуррентная сеть (RNN)

- Могут учесть порядок во времени – работать с последовательными данными
- Обладают памятью за счет того, что содержимое слоя передается нейросети обратно, это как бы ее память.



```
nn.Linear(in_features=3, out_features=2) nn.RNN(input_size=3, hidden_size=2)
```

$$h = f_{\text{act}}(W_{xh}x)$$

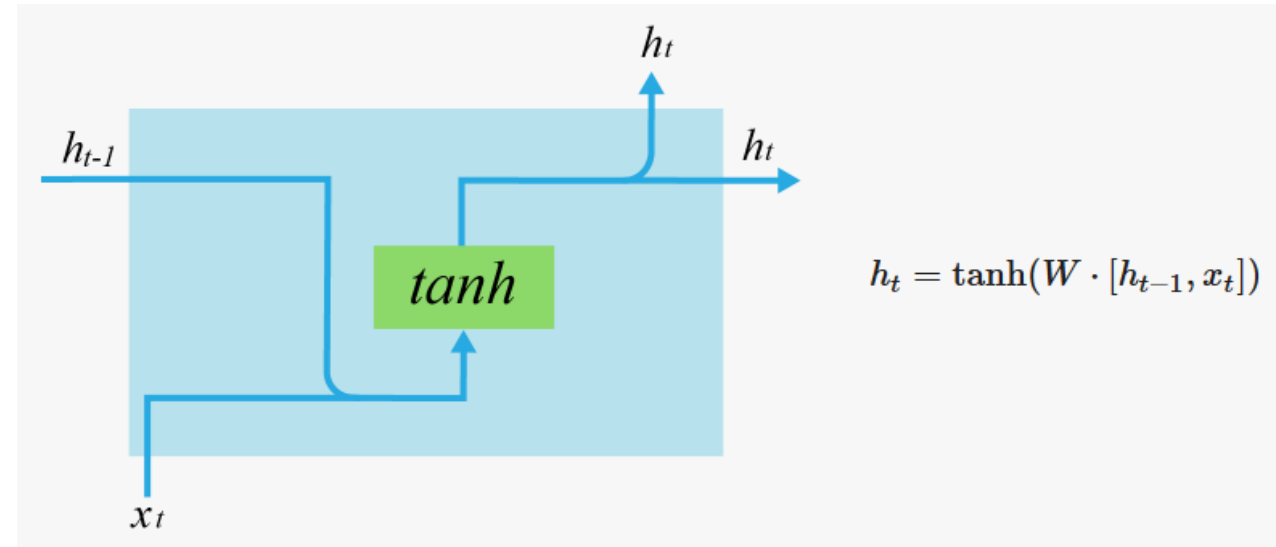
$$h_t = f_{\text{act}}(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y = f_{\text{act}}(W_{hy}h)$$

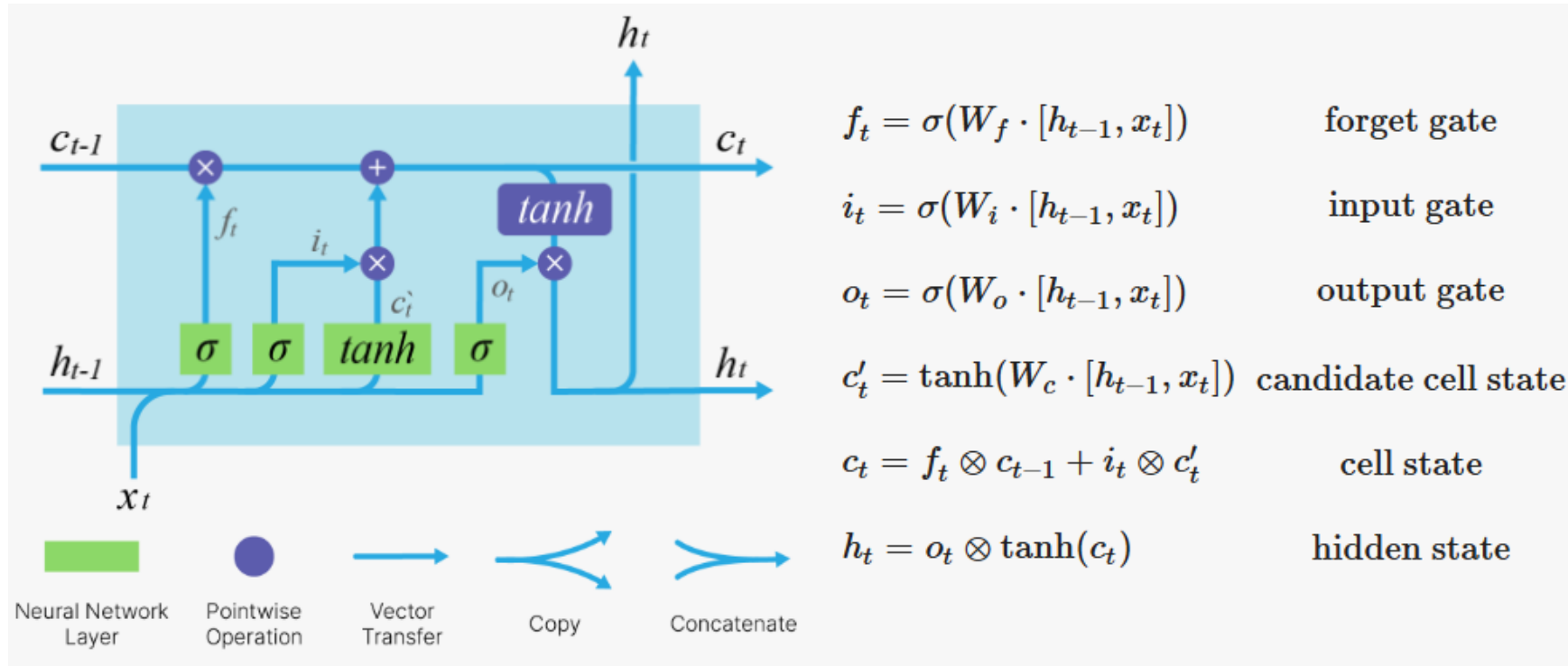
$$y_t = f_{\text{act}}(W_{hy}h_t)$$

Обычная RNN

- Любая рекуррентная нейронная сеть имеет форму цепочки повторяющихся модулей нейронной сети. В обычной RNN структура одного такого модуля очень проста, например, он может представлять собой один слой с функцией активации \tanh (гиперболический тангенс).
- По мере роста расстояния между модулями RNN теряют способность связывать информацию



LSTM - Long short-term memory



- Как только мы начинаем говорить про глубокие нейросети – от градиентов, которые с конца должны пройти в самое начало ничего не остается. Для решения проблемы придумали LSTM или GRU- это ячейки памяти.
- Интуиция - давайте возьмём RNN со skip-connections и попробуем улучшить его так, чтобы внутреннее состояние можно было сильно менять, при этом не сталкиваясь с проблемой взрывающихся и затухающих градиентов. В целом, хотим чтобы h_t был по типу памяти слоя. Хорошо бы, чтобы память можно было обновлять, обнулять, добавлять в неё что-то.

GRU - Gated Recurrent Unit

Более легковесная версия LSTM – GRU (Gated Recurrent Unit). Так же использует гейты и имеет похожую структуру, но имеет несколько отличий:

- Имеет только одно внутреннее состояние h_t
- Input и Output gates объединены в один гейт

Reset gate – какие части h_{t-1} должны быть использованы для подсчёта прибавки к памяти:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

Update gate – какие части h_{t-1} должны измениться:

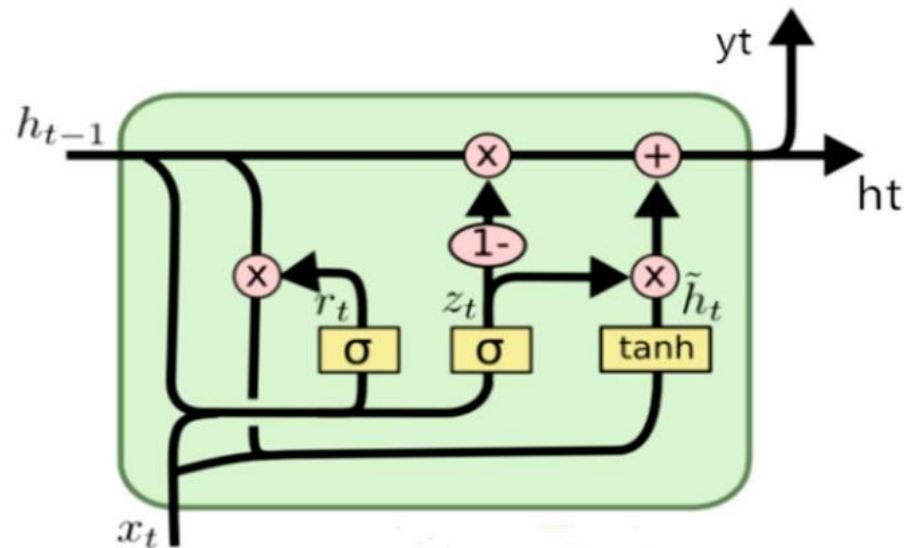
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

Прибавка к памяти:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t])$$

Обновление памяти:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

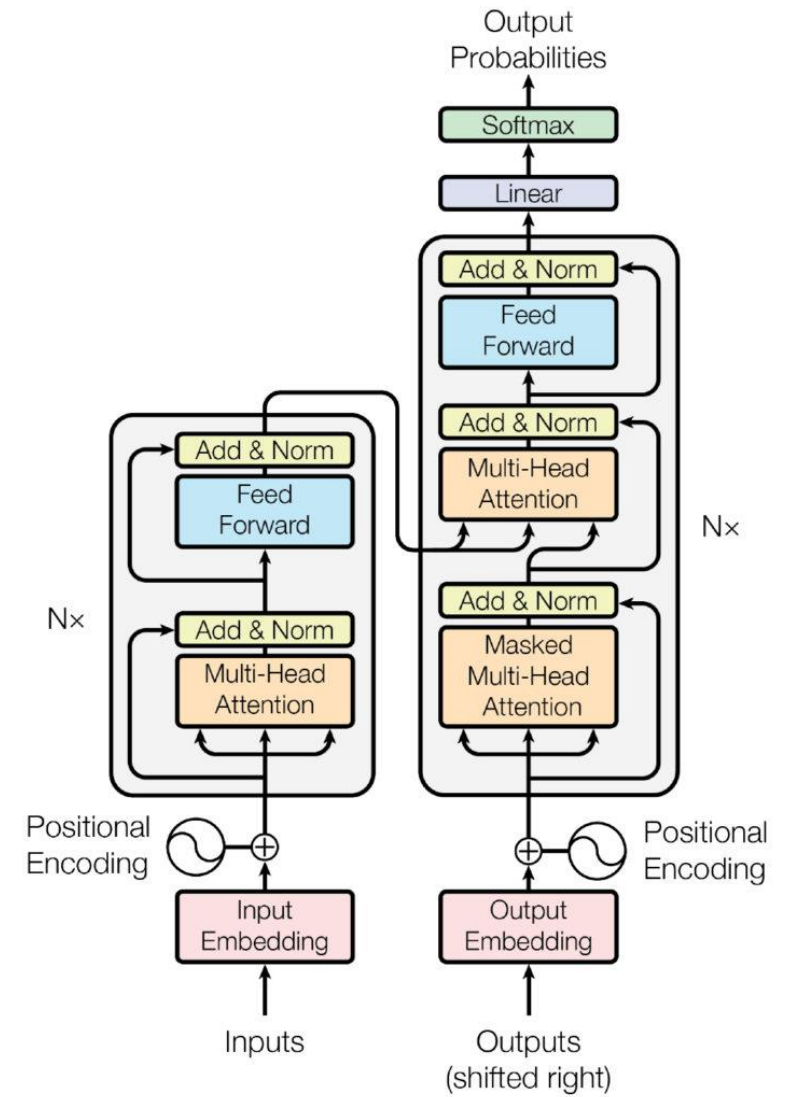


LSTM vs GRU

- У GRU меньше параметров и она более вычислительно эффективна
- Много данных → LSTM обычно показывает результаты чуть лучше
- Мало данных → GRU обычно показывает результаты чуть лучше
- Однозначно сказать нельзя кто лучше

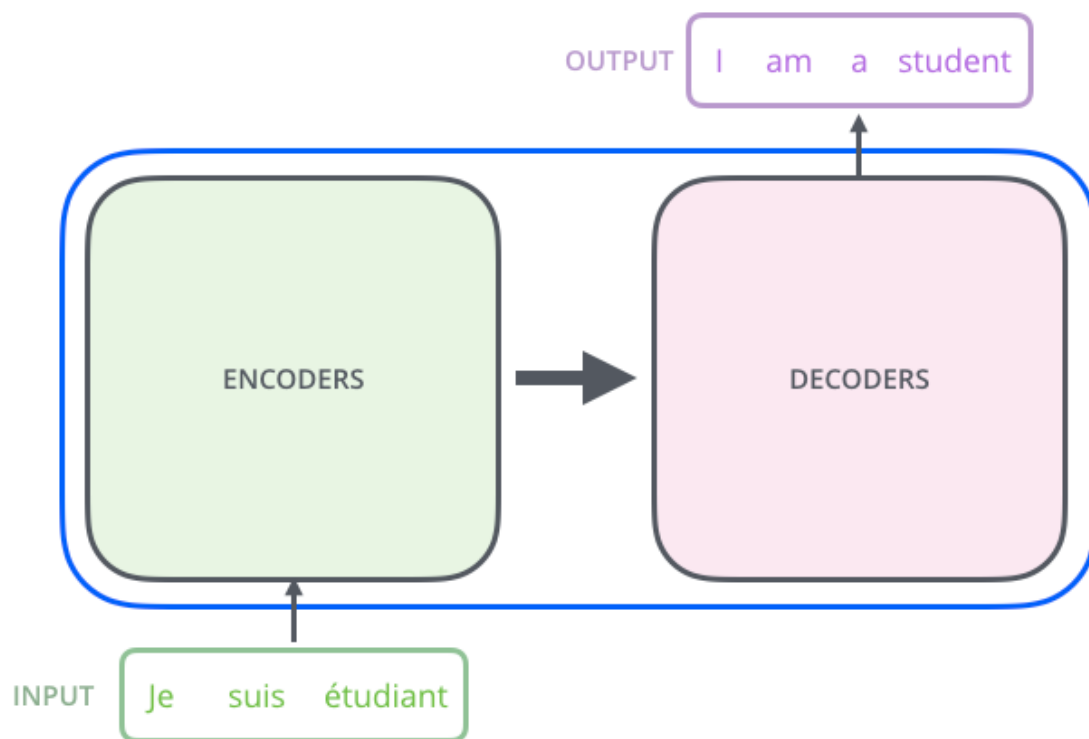
Трансформеры

- Трансформеры были представлены в 2017 году компанией Google в статье «Attention Is All You Need».
- По аналогии с рекуррентными нейронными сетями (РНС) трансформеры предназначены для обработки последовательностей, таких как текст на естественном языке, и решения таких задач как машинный перевод и автоматическое реферирование. В отличие от РНС, трансформеры не требуют обработки последовательностей по порядку. Например, если входные данные — это текст, то трансформеру не требуется обрабатывать конец текста после обработки его начала.
- В последнее время трансформеры значительно превзошли другие нейросетевые архитектуры и сейчас являются State-Of-The-Art моделями.



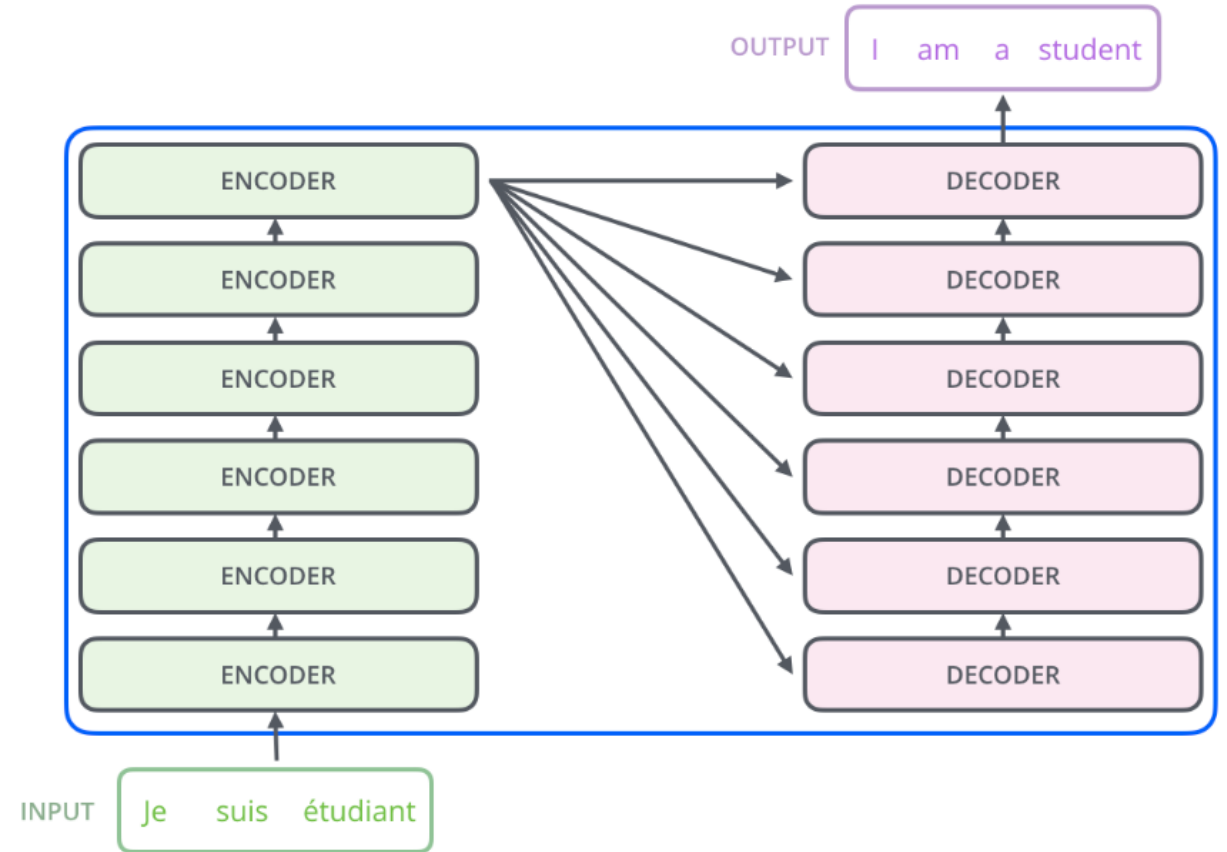
Трансформеры – Encoder-Decoder

Трансформеры – seq2seq модель с архитектурой Encoder-Decoder



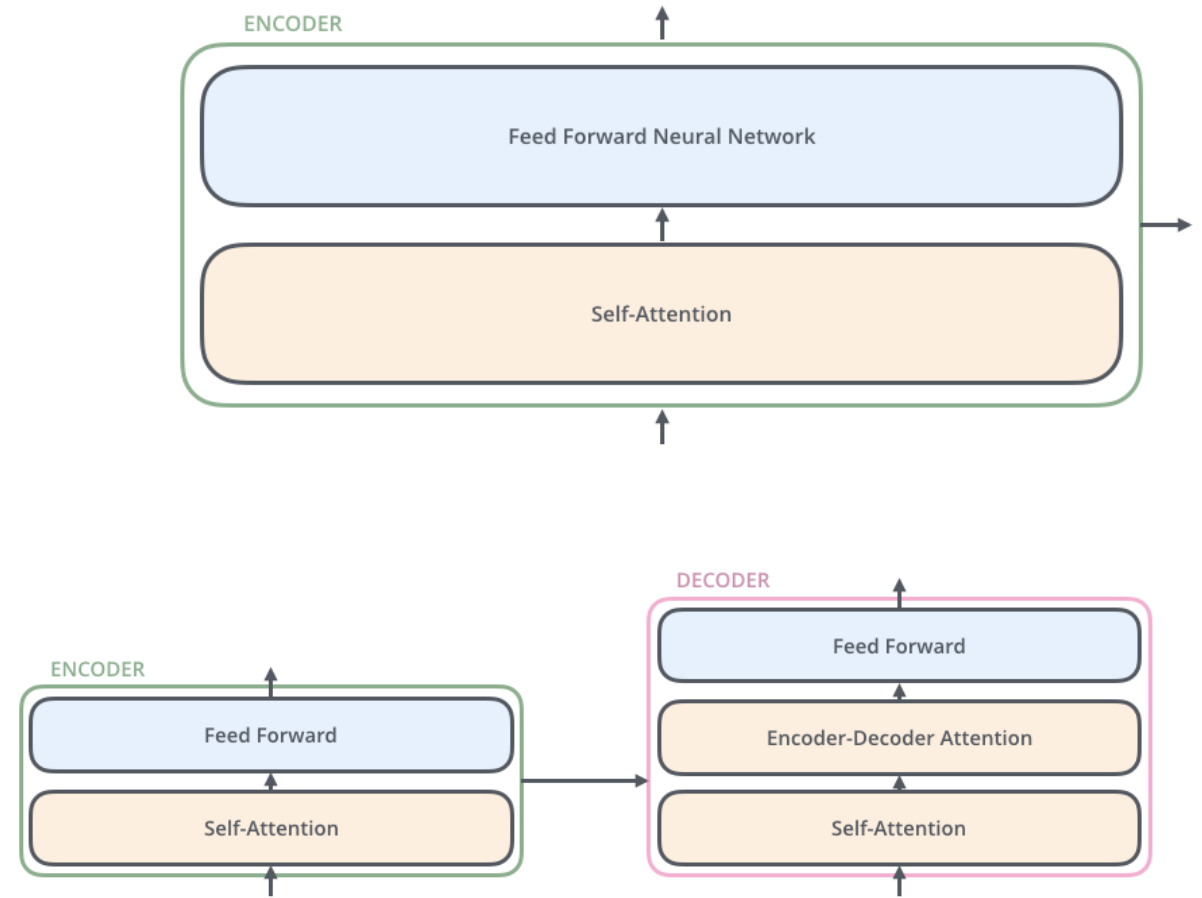
Трансформеры – внутри Encoder-Decoder

- Encoder состоит из нескольких слоев, каждый из которых является encoder'ом.
- Decoder также состоит из нескольких слоев, и на каждый слой Decoder'а подается одно и то же состояние Encoder'а

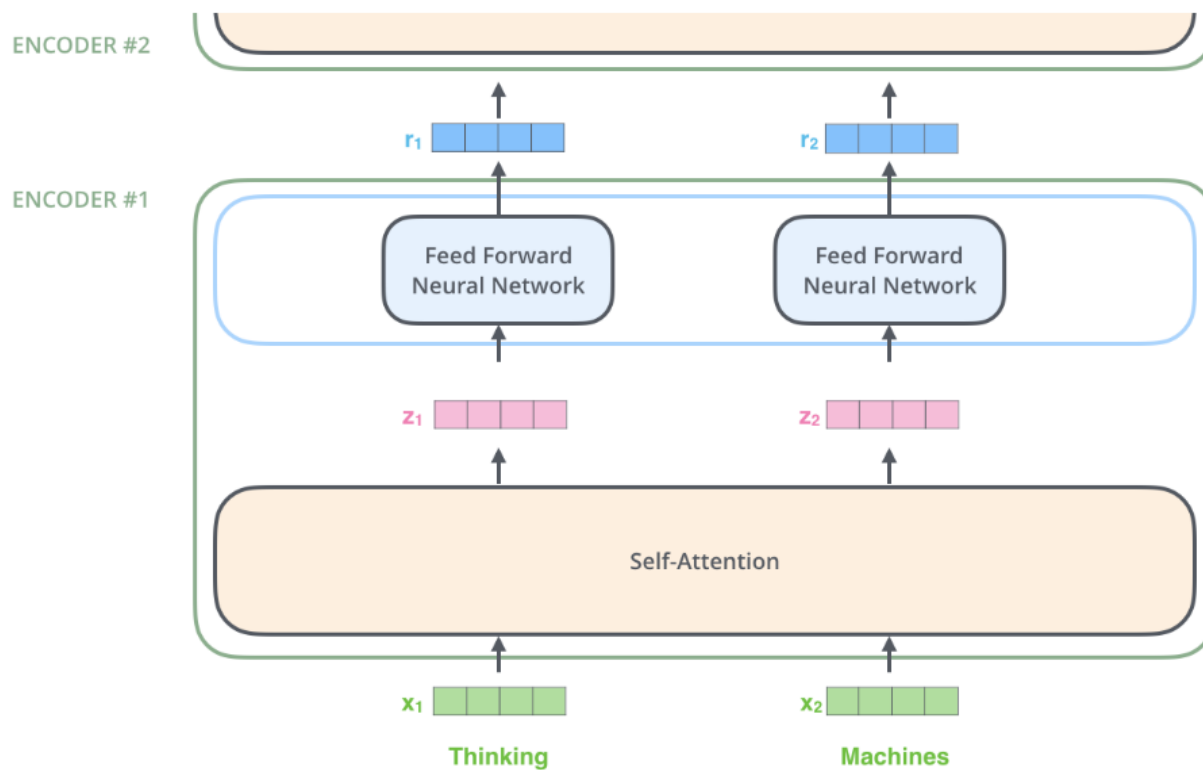


Трансформеры – внутри Encoder-Decoder

- Все Encoder'ы состоят из двух частей – механизма Self-Attention и полносвязного слоя (FFNN)
- Decoder'ы имеют еще одну часть – Encoder-Decoder Attention

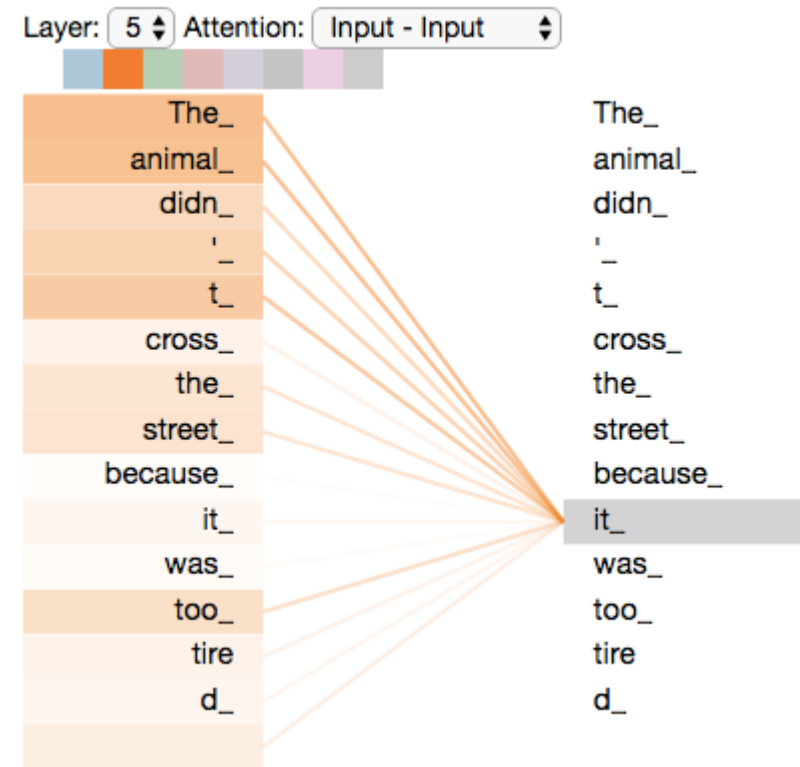


Трансформеры – внутри Encoder-Decoder



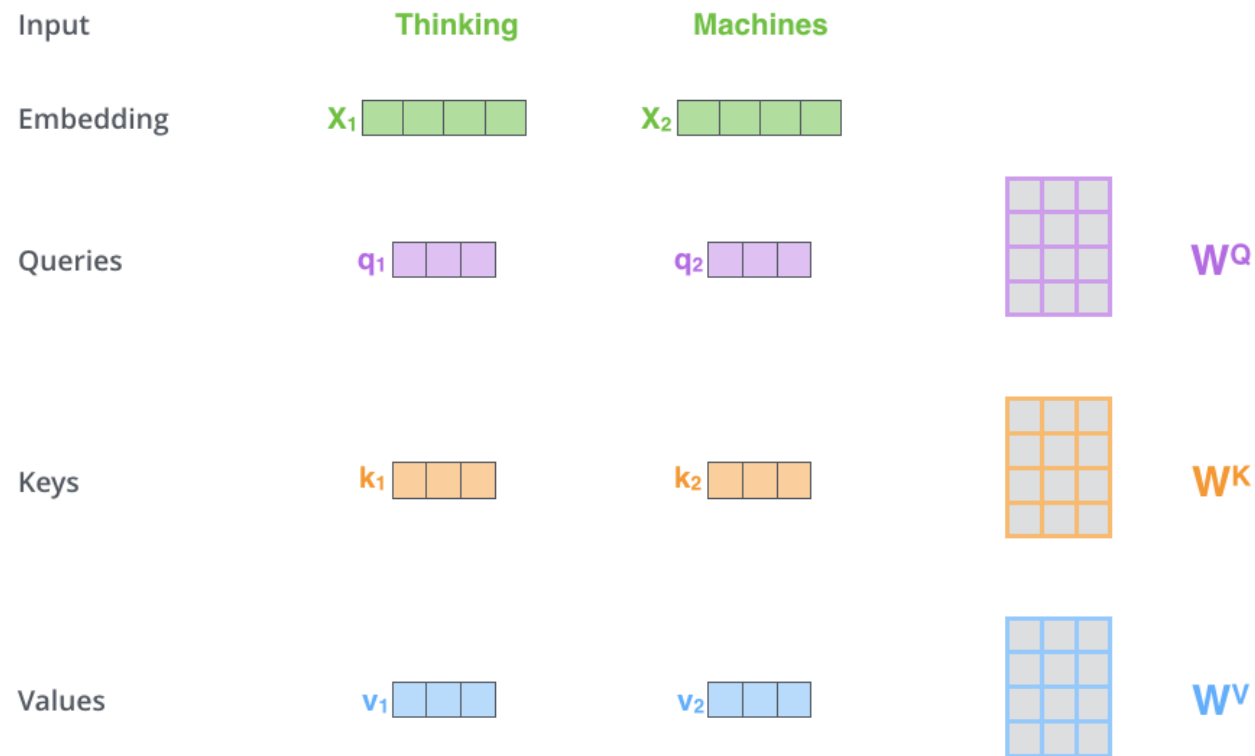
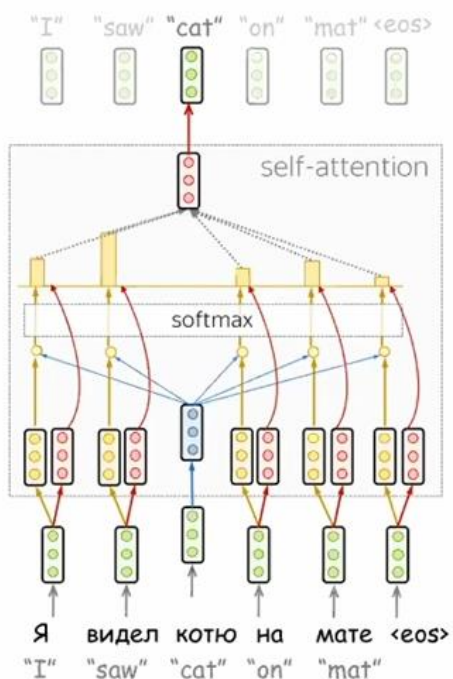
Трансформеры – Self-Attention

- По мере обработки каждого слова, self-attention позволяет модели искать в других позициях входной последовательности «подсказки», которые могут улучшить кодирование этого слова.
- Фактически Self-Attention – способ встроить «понимание» других релевантных слов в то, которое обрабатывается на текущий момент.



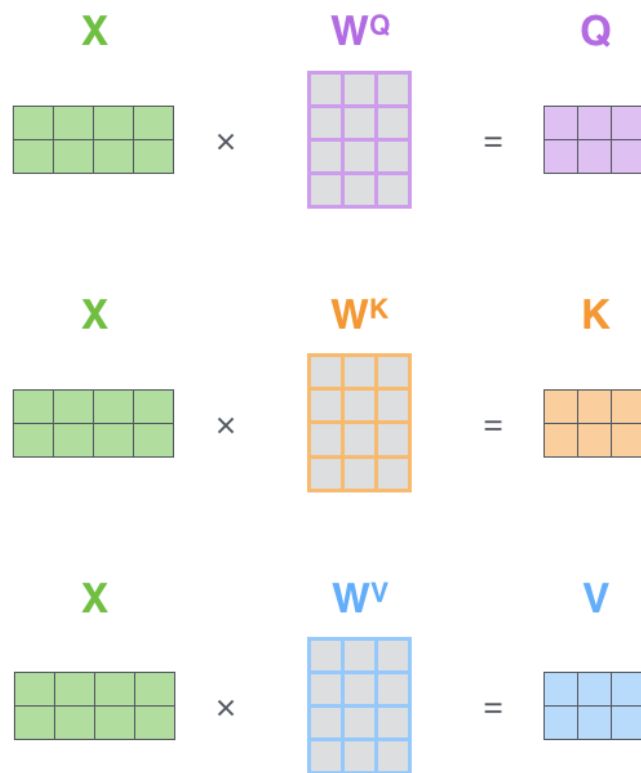
Трансформеры – Self-Attention

- Первый шаг – создание векторов q , k , v путем умножения входного эмбединга на три матрицы, которые мы обучали в процессе.



Трансформеры – Self-Attention

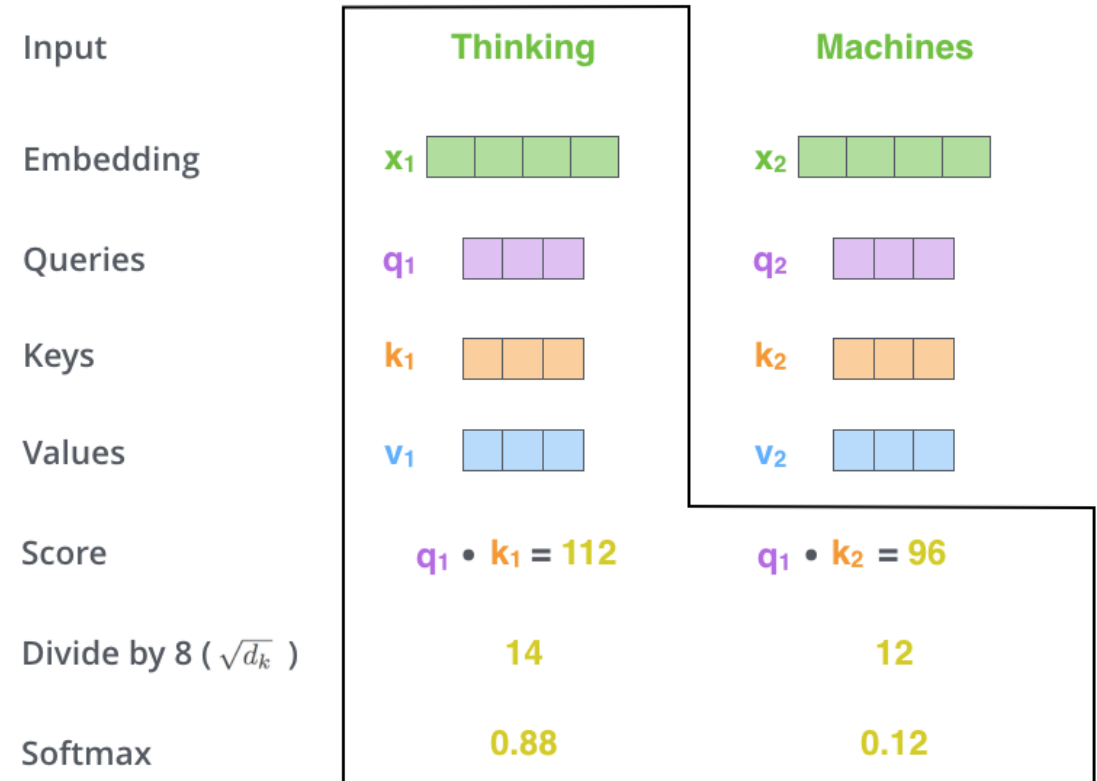
- В итоге получаем матрицы запросов (query), ключей (keys), значений (values)



Трансформеры – Self-Attention

Дальнейшие шаги:

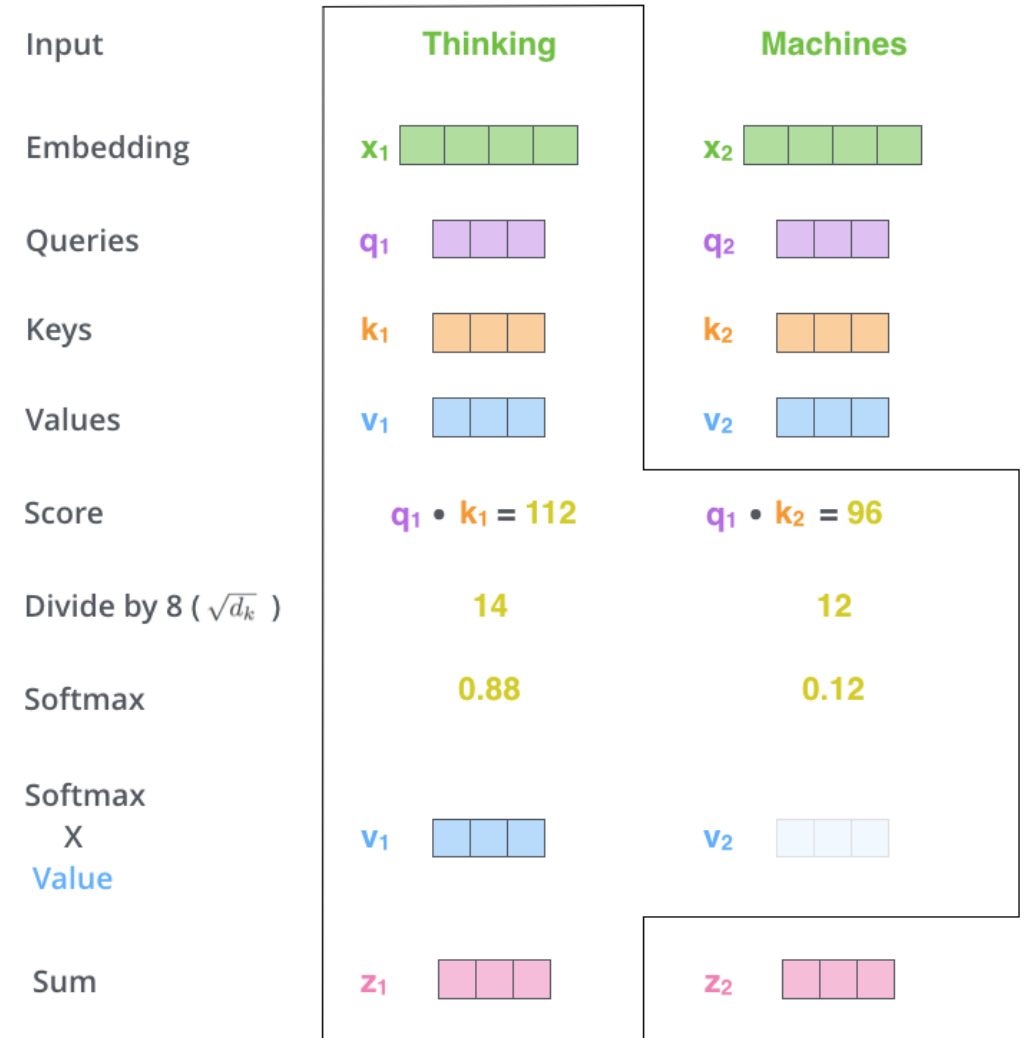
- Умножаем вектор **q** на вектора **k** получая оценки того, сколько внимания нужно уделять другим частям входного предложения, когда мы кодируем слово в определенной позиции.
- Делим полученные оценки на 8 (квадратный корень из размерности вектора)
- Применяем Softmax



Трансформеры – Self-Attention

Затем:

- Умножаем полученные значения на вектора \mathbf{v}
- Складываем все вместе, получая выходной вектор



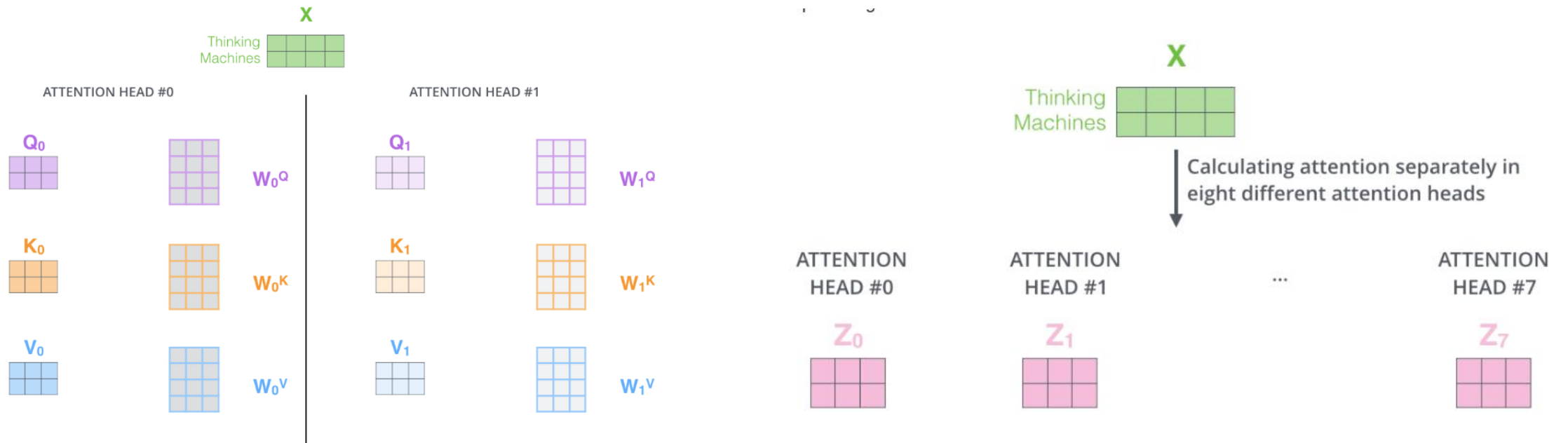
Трансформеры – Self-Attention

В матричной форме это выглядит как:

$$\text{softmax}\left(\frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \end{matrix}\right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$
$$= \begin{matrix} \text{Z} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

Трансформеры – Multi-Head Attention

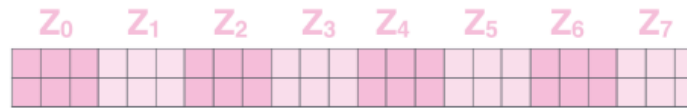
В Multi-Head Attention мы поддерживаем отдельные весовые матрицы Q/K/V для каждой головы (head), что приводит к различным матрицам Q/K/V.



Трансформеры – Multi-Head Attention

После вычисления векторов в Multi-Head Attention они конкатенируются, умножаются на матрицу весов W

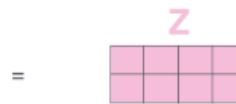
1) Concatenate all the attention heads



2) Multiply with a weight matrix W^O that was trained jointly with the model

\times

3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

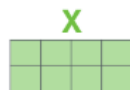


Трансформеры – Multi-Head Attention

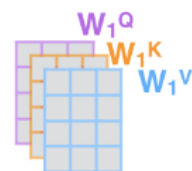
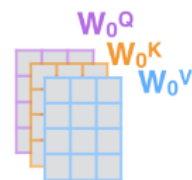
Соберем все вместе:

- 1) This is our input sentence*
- 2) We embed each word*
- 3) Split into 8 heads. We multiply X or R with weight matrices
- 4) Calculate attention using the resulting $Q/K/V$ matrices
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

Thinking
Machines



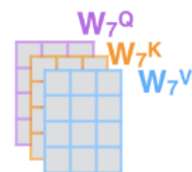
* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one



...

...

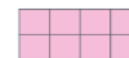
...



W^O

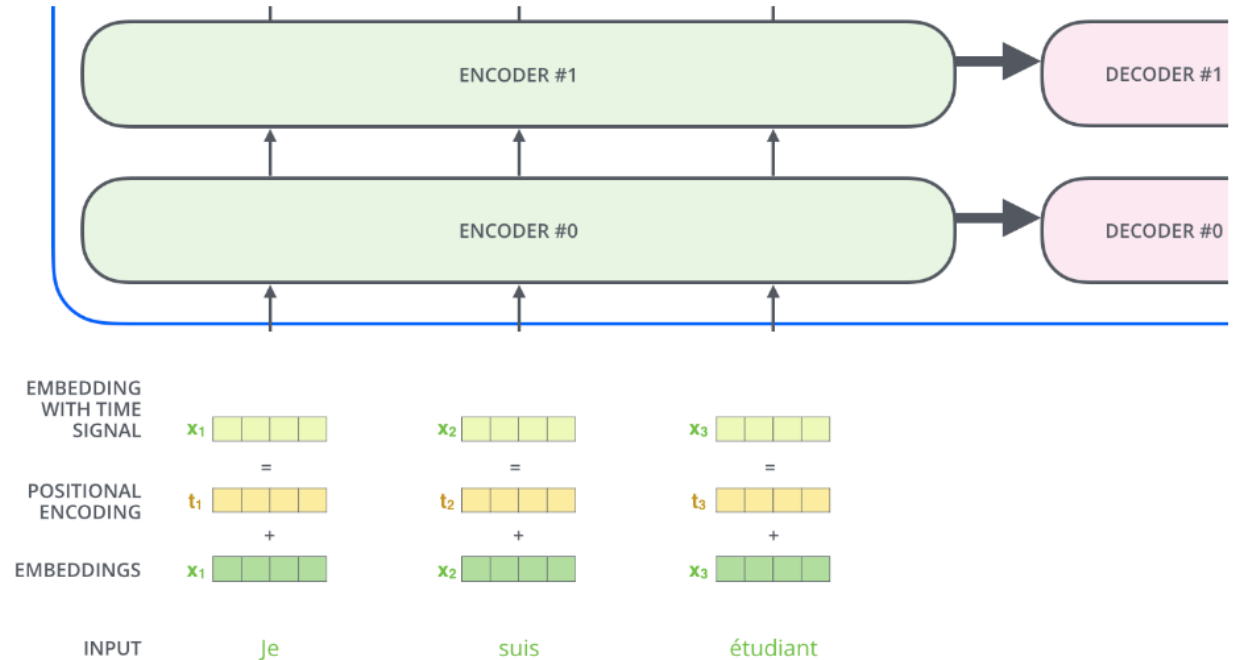


Z



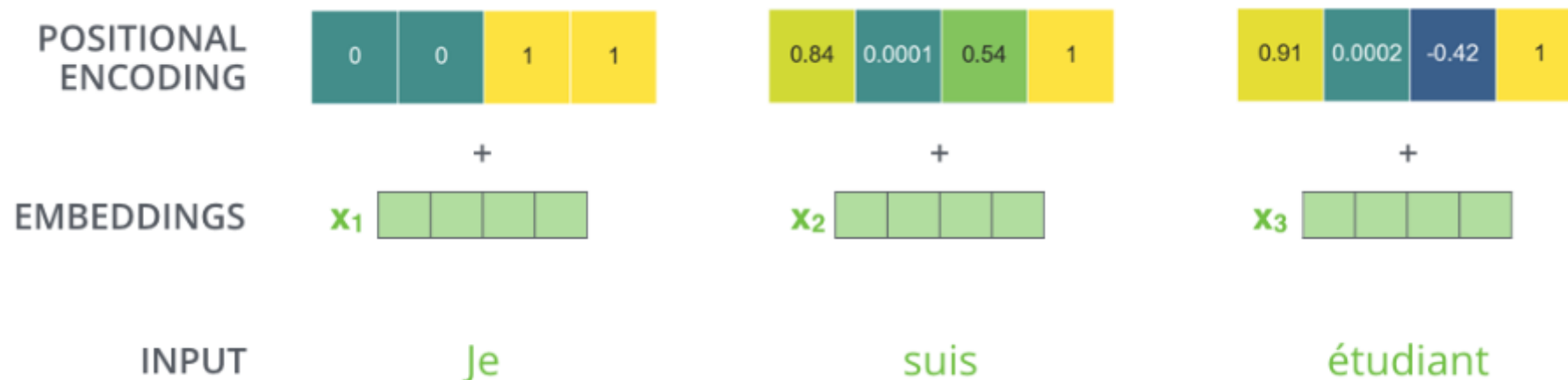
Трансформеры – Positional Encoding

- Сейчас в описанном механизме не учитывается порядок слов.
- Чтобы решить эту проблему, к основному эмбедингу добавляется positional эмбединг.



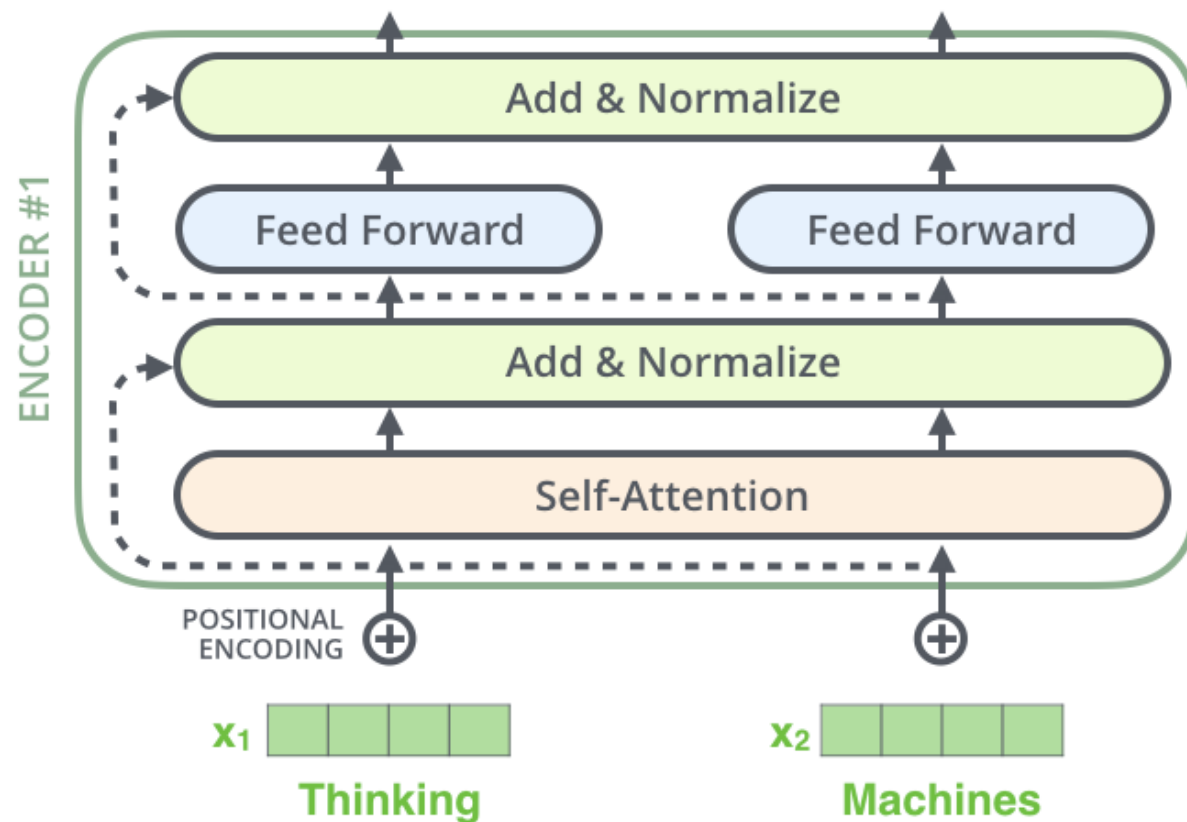
Трансформеры – Positional Encoding

Интуиция здесь такова, что добавление векторов к эмбедингам обеспечивает значимые расстояния между векторами эмбедингов после их проецирования в векторы Q/K/V и во время скалярного произведения в Attention.

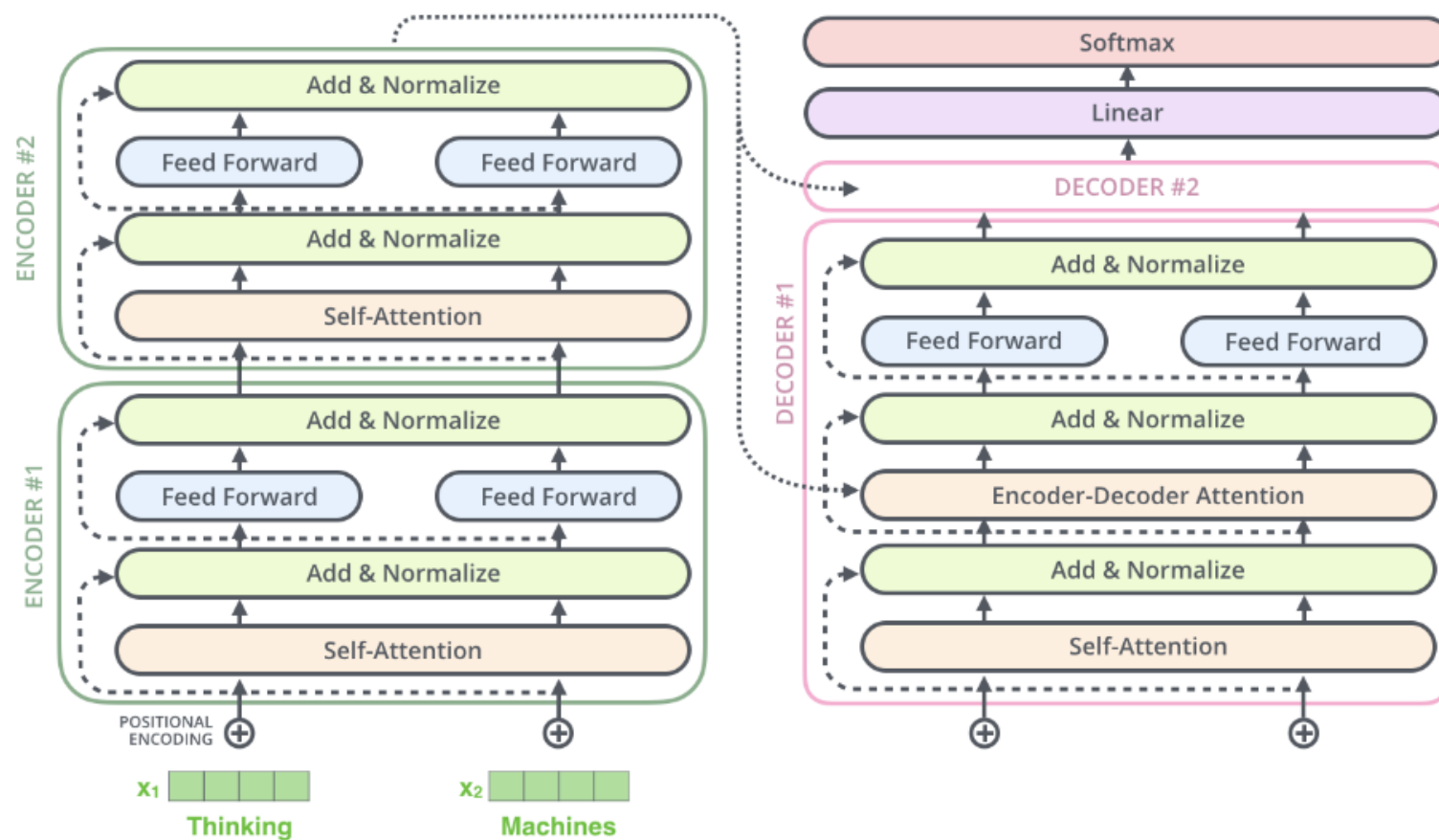


Трансформеры – Residuals & Normalization

Каждый подуровень (self-attention, ffnn) в каждом Encoder'е имеет residual-соединение, и за ним следует шаг нормализации.

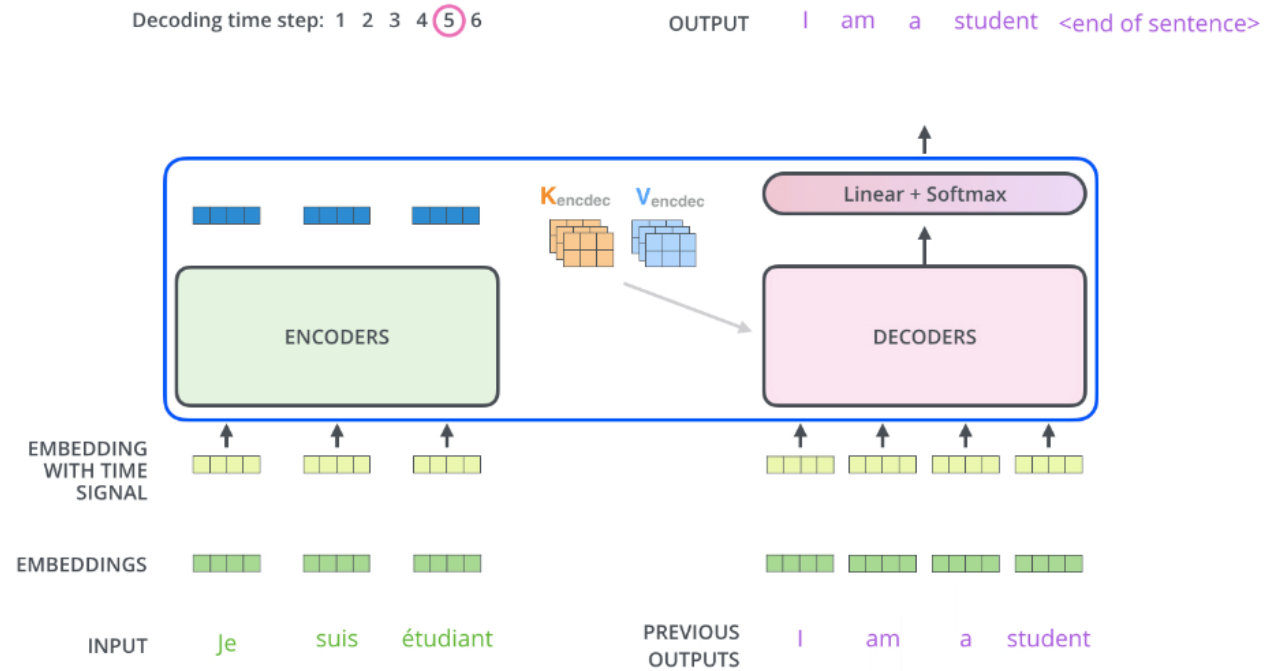


Трансформеры – Residuals & Normalization



Трансформеры – Decoder

Вектора K и V используются Decoder'ами в Encoder-Decoder Attention слоях, которые помогают Decoder'у сосредоточиться на соответствующих местах входной последовательности



Трансформеры – Linear & Softmax

Which word in our vocabulary
is associated with this index?

am

Get the index of the cell
with the highest value
(argmax)

5

log_probs



Softmax

logits



Linear

Decoder stack output

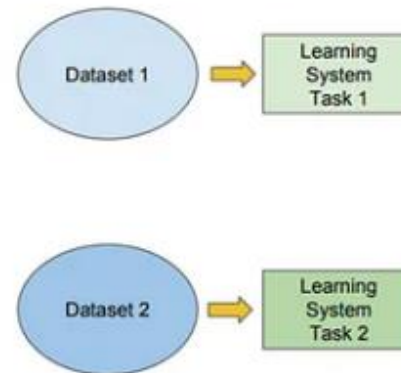


Трасферное обучение (transfer learning)

- Трансферное обучение — подраздел ML. Его цель — применение знаний, полученных благодаря решению одной задачи, к другой, но схожей задаче.
- При переносе обучения мы начинаем с признаков, усвоенных при решении другой задачи, вместо того чтобы тренировать модели с нуля.
- Трансферное обучение предполагает использование предобученных моделей (созданных и натренированных на большом наборе общедоступных данных).

Traditional ML

- Isolated, single task learning:
 - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
 - Learning process can be faster, more accurate and/or need less training data

