

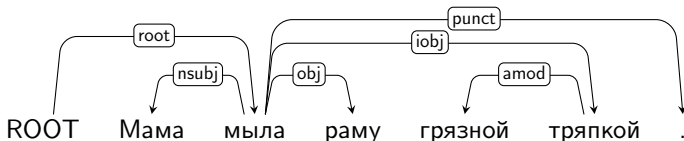
## Механизм внимания.

Алексей Андреевич Сорокин  
Yandex Research,  
МГУ, отделение теоретической и прикладной лингвистики.

Школа РАИИ 2021  
лекция 4.

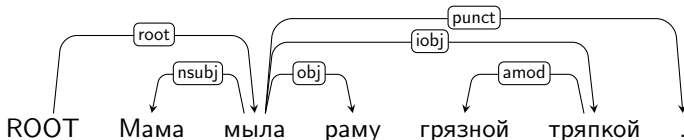
# Постановка задачи

- Нужно по предложению восстановить дерево зависимостей:



# Постановка задачи

- Нужно по предложению восстановить дерево зависимостей:



- Дерево кодируется таблицей:

Мама	2	nsbj
мыла	0	root
раму	1	obj
грязной	5	amod
тряпкой	2	iobj
.	2	punct

## Графовый подход: алгоритм решения

- Для восстановления дерева нужно восстановить родителя и тип зависимости каждого слова.
- Тип зависимости сложно восстановить, не зная родителя, поэтому вначале нужно построить дерево.
- После этого это обычная разметка последовательности.

## Графовый подход: алгоритм решения

- Для восстановления дерева нужно восстановить родителя и тип зависимости каждого слова.
- Тип зависимости сложно восстановить, не зная родителя, поэтому вначале нужно построить дерево.
- После этого это обычная разметка последовательности.
- Будем восстанавливать дерево, предсказывая  $p_{ij}$  – вероятность, что  $i$ -ое слово зависит от  $j$ -го.
- Для каждого  $i$  ответ – распределение вероятностей.

# Графовый подход: вычисление вероятностей

- Пусть для каждого слова уже вычислены выходы энкодера  $h_i$ .
- Вычислим их вектора как родителя и как ребёнка:

$$\begin{aligned}h_i^{head} &= g(W^{head} h_i + b^{head}) \\h_i^{dep} &= g(W^{dep} h_i + b^{dep})\end{aligned}$$

# Графовый подход: вычисление вероятностей

- Пусть для каждого слова уже вычислены выходы энкодера  $h_i$ .
- Вычислим их вектора как родителя и как ребёнка:

$$\begin{aligned}h_i^{head} &= g(W^{head} h_i + b^{head}) \\ h_i^{dep} &= g(W^{dep} h_i + b^{dep})\end{aligned}$$

- Вычислим меры близости между векторами с помощью би-аффинной сети:

$$e_{ij} = h_i^{dep^T} U h_j^{head} + (w^{dep}, h_i^{dep}) + (w^{head}, h_j^{head})$$

# Графовый подход: вычисление вероятностей

- Пусть для каждого слова уже вычислены выходы энкодера  $h_i$ .
- Вычислим их вектора как родителя и как ребёнка:

$$\begin{aligned}h_i^{head} &= g(W^{head} h_i + b^{head}) \\ h_i^{dep} &= g(W^{dep} h_i + b^{dep})\end{aligned}$$

- Вычислим меры близости между векторами с помощью би-аффинной сети:

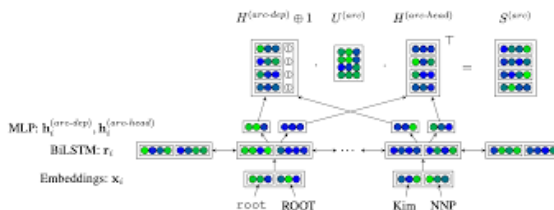
$$e_{ij} = h_i^{dep^T} U h_j^{head} + (w^{dep}, h_i^{dep}) + (w^{head}, h_j^{head})$$

- Переведём близости в вероятности:

$$[p_{i0}, \dots, p_{in}] = \text{softmax}([e_{i0}, \dots, e_{in}])$$



# Графовый подход: иллюстрация



## Графовый подход: определение вершины

- Для каждой пары позиций  $i, j$  определим вероятность, что  $w_j$  – родитель  $w_i$ :

	ROOT	мама	мыла	раму	грязной	тряпкой
мама	0.2	0	0.8	0	0	0
мыла	0.7	0.2	0	0.1	0	0
раму	0.0	0.01	0.99	0	0	0
грязной	0.0	0.0	0.0	0.1	0	0.9
тряпкой	0.0	0.0	0.9	0.1	0	0

# Графовый подход: определение вершины

- Выберем вершины с наибольшей вероятностью:

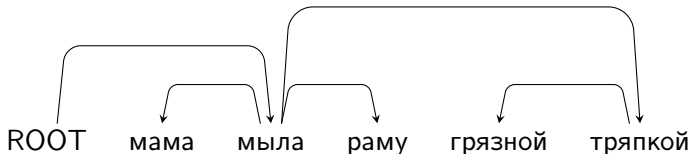
	ROOT	мама	мыла	раму	грязной	тряпкой
мама	0.2	0	0.8	0	0	0
мыла	0.7	0.2	0	0.1	0	0
раму	0.0	0.01	0.99	0	0	0
грязной	0.0	0.0	0.0	0.1	0	0.9
тряпкой	0.0	0.0	0.9	0.1	0	0

## Графовый подход: определение вершины

- Выберем вершины с наибольшей вероятностью:

	ROOT	мама	мыла	раму	грязной	тряпкой
мама	0.2	0	0.8	0	0	0
мыла	0.7	0.2	0	0.1	0	0
раму	0.0	0.01	0.99	0	0	0
грязной	0.0	0.0	0.0	0.1	0	0.9
тряпкой	0.0	0.0	0.9	0.1	0	0

- В данном случае получим дерево:



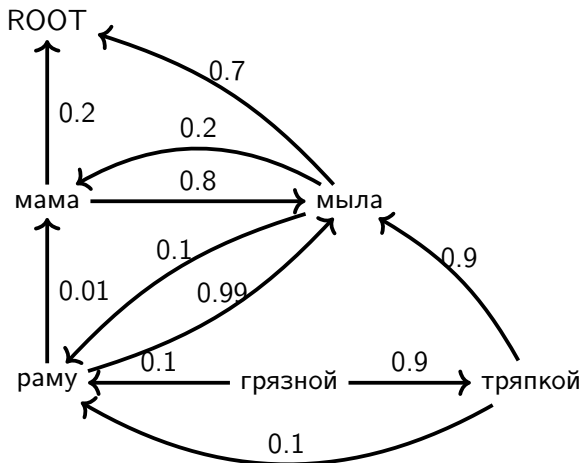
## Графовый подход: построение дерева

- Жадный алгоритм построения наилучшего дерева – выбрать для каждой вершины наиболее вероятного предка.
- Проблема: результат – необязательно дерево (могут получиться циклы).

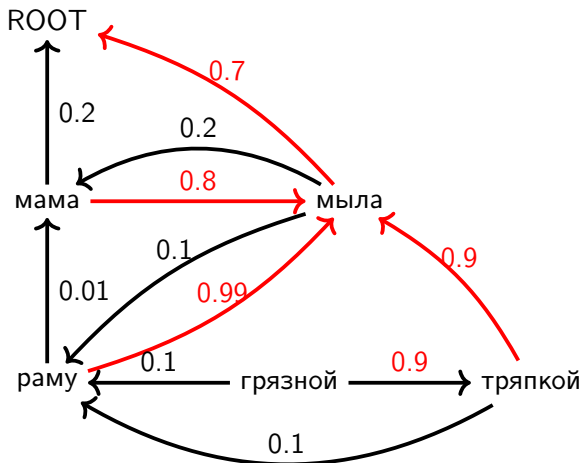
## Графовый подход: построение дерева

- Жадный алгоритм построения наилучшего дерева – выбрать для каждой вершины наиболее вероятного предка.
- Проблема: результат – необязательно дерево (могут получиться циклы).
- На самом деле нужно построить *максимальное остовное дерево* – дерево максимальной стоимости, содержащее все вершины.
- В данном случае стоимость равна вероятности, стоимость дерева – произведение стоимостей рёбер.
- При переходе к логарифмам стандартная ситуация: стоимость равна сумме стоимостей рёбер.

## Графовый подход: иллюстрация

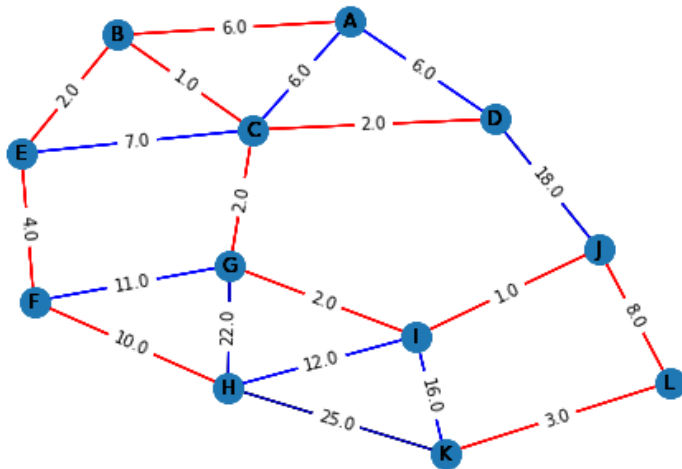


## Графовый подход: иллюстрация





# Минимальное остовное дерево



## Минимальное остовное дерево: алгоритмы

- В неориентированном случае минимальное остовное дерево строится алгоритмом Краскала:
  - Начать с пустого дерева.

# Минимальное остовное дерево: алгоритмы

- В неориентированном случае минимальное остовное дерево строится алгоритмом Краскала:
  - Начать с пустого дерева.
  - На каждом шаге добавлять минимальное ребро, не приводящее к циклу.

## Минимальное остовное дерево: алгоритмы

- В неориентированном случае минимальное остовное дерево строится алгоритмом Краскала:
  - Начать с пустого дерева.
  - На каждом шаге добавлять минимальное ребро, не приводящее к циклу.
- Ориентированное дерево – алгоритм Чу-Лю-Эдмондса.
  - Для каждой вершины выбрать самое полезное исходящее ребро.
  - Если нет циклов – это ответ.

# Минимальное остовное дерево: алгоритмы

- В неориентированном случае минимальное остовное дерево строится алгоритмом Краскала:
  - Начать с пустого дерева.
  - На каждом шаге добавлять минимальное ребро, не приводящее к циклу.
- Ориентированное дерево – алгоритм Чу-Лю-Эдмондса.
  - Для каждой вершины выбрать самое полезное исходящее ребро.
  - Если нет циклов – это ответ.
  - Если есть – попытаться в каждом цикле его разорвать с минимальной потерей в стоимости.

# Нейронный машинный перевод

- Нейронный машинный перевод — задача условного порождения текста.
- Обычная вероятностная модель порождает текст на основе предыдущих слов.

$$w_i \sim p(w|h_{i-1})$$

$h_{i-1}$  — состояние языковой модели после  $i - 1$  слова

# Нейронный машинный перевод

- Нейронный машинный перевод — задача условного порождения текста.
- Обычная вероятностная модель порождает текст на основе предыдущих слов.

$$w_i \sim p(w|h_{i-1})$$

$h_{i-1}$  — состояние языковой модели после  $i - 1$  слова

- Условная вероятностная модель:

$$w_i \sim p(w|h_{i-1}, c)$$

$c$  — глобальный контекст

- Основная проблема: как вычислять  $c$ .

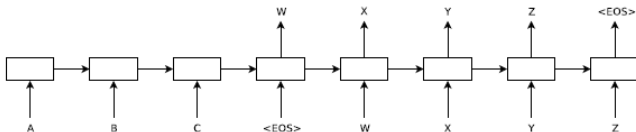
# Нейронный машинный перевод

- Базовая модель: кодировщик-декодировщик (encoder-decoder):

$$c = LSTM(x_1, \dots, x_m),$$

$x_1 \dots x_m$  — исходное предложение

- Вектор  $c$  запоминает всю информацию об исходном предложении в одном векторе.





# Нейронный машинный перевод

- Обычно на каждый шаг декодера явно подаётся предыдущее слово:

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

- Как и энкодер, так и декодер включают в себя несколько слоёв.
- Входом следующего служит выход предыдущего.

# Вектора предложений



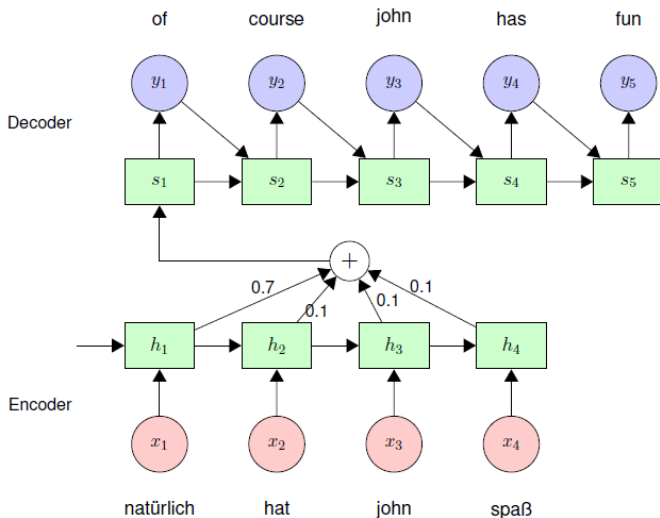
## Механизм внимания: мотивация

- При сжатии всего предложения в один вектор часть информации неминуемо теряется.
- На разных этапах порождения выходного предложения полезна разная информация об исходных словах.

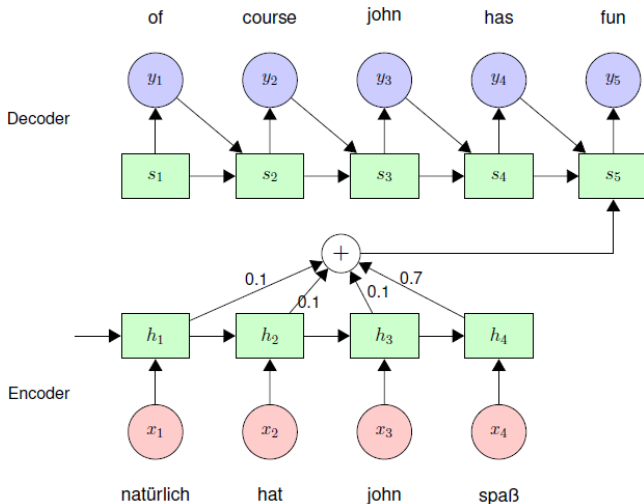
## Механизм внимания: мотивация

- При сжатии всего предложения в один вектор часть информации неминуемо теряется.
- На разных этапах порождения выходного предложения полезна разная информация об исходных словах.
- Выход: сделать контекстный вектор зависящим от позиции в порождаемом предложении.

# Механизм внимания: иллюстрация



# Механизм внимания: иллюстрация



# Механизм внимания: реализация

- Каждое следующее слово порождается отдельным вектором контекста:

$$w_{i+1} \sim p(h_i, c_i)$$

$h_i$  – состояние декодера после  $i$  слов,  
 $c_i$  – исходный контекст после  $i$  слов.

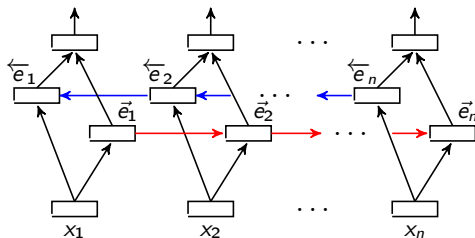
- Вектор контекста – сумма векторов контекста для отдельных позиций:

$$c_i = \sum_j \alpha_{ij} e_j,$$

$e_j$  – вектор контекста в позиции  $j$ ,  
 $\alpha_{ij}$  – мера влияния  $e_j$  на  $c_i$ .

# Механизм внимания: реализация

- Как считать  $\alpha_{ij}$  и вектора  $e_j$ ?
- $e_j$  вычисляется с помощью двунаправленной рекуррентной сети (конкатенация  $\vec{e}_j$  и  $\leftarrow e_j$ ):





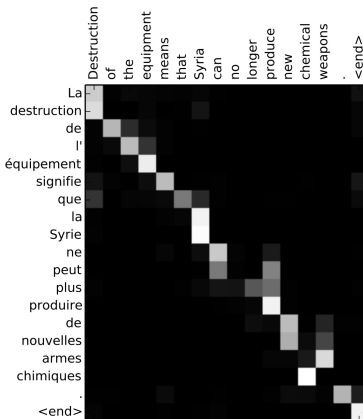
# Механизм внимания: реализация

- $\alpha_{ij}$  можно считать разными способами.
- Bahdanau et al., 2015, Jointly learning to align and translate использовалось:

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(u_{ij}), \\ u_{ij} &= a(h_{i-1}, e_j), \\ h_{i-1} &- \text{состояние декодера на предыдущем шаге.}\end{aligned}$$

# Механизм внимания: интерпретация

- Механизм внимания показывает, какие слова исходного текста влияют на слова сгенерированного текста.



# Механизм внимания: вариации

- Механизм внимания показывает, какие слова исходного текста влияют на слова сгенерированного текста.
- Конкретные формулы могут отличаться.
- Задача формулы — вычислить числа  $u_{ij}$ , показывающие связь исходного вектора  $e_j$  с вектором контекста  $h_i$  в переводном тексте
- Далее эти формулы будут переведены в вероятности.

$$\alpha_{ij} = \text{softmax}(u_{ij})$$

## Механизм внимания: вариации

- Luong et al., Effective Approaches to Attention-based Neural Machine Translation: 3 формулы для механизма внимания.

$$u_{ij} = \langle h_i, e_j \rangle \quad (\text{скалярное произведение}),$$

$$u_{ij} = h_i^T W_a e_j \quad (\text{скалярное произведение} \\ \text{с обученной матрицей}),$$

$$u_{ij} = v_a^T \tanh(W_a[h_i, e_j]) \quad (\text{однослойная сеть})$$

- Лучше работает второй подход, но он требует больше ресурсов, чем первый.

## Механизм внимания: вариации

- Luong et al., Effective Approaches to Attention-based Neural Machine Translation: 3 формулы для механизма внимания.

$$u_{ij} = \langle h_i, e_j \rangle \quad (\text{скалярное произведение}),$$

$$u_{ij} = h_i^T W_a e_j \quad (\text{скалярное произведение} \\ \text{с обученной матрицей}),$$

$$u_{ij} = v_a^T \tanh(W_a[h_i, e_j]) \quad (\text{однослойная сеть})$$

- Лучше работает второй подход, но он требует больше ресурсов, чем первый.
- Третий подход (Bahdanau et al., 2015) проигрывает первым двум.

## Механизм внимания: вариации

- Внимание можно использовать и для классификации.
- Самый простой способ получить вектор для предложения — взять взвешенную сумму векторов для его слов.
- Именно это делает tf-idf.

# Механизм внимания: вариации

- Внимание можно использовать и для классификации.
- Самый простой способ получить вектор для предложения — взять взвешенную сумму векторов для его слов.
- Именно это делает tf-idf.
- Также применялись веса, зависящие от частей речи.
- Можно настроить веса под задачу:

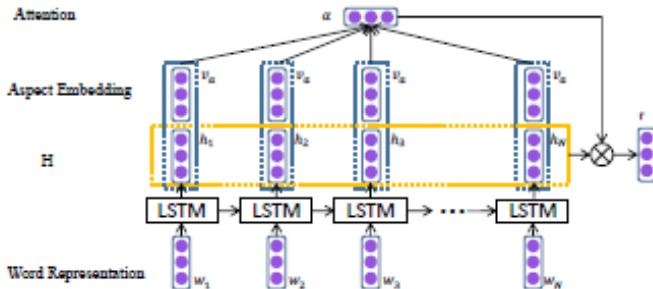
$$s = \sum_j \alpha_j x_j \quad x_j \text{ — представление } i\text{-го слова,}$$
$$\alpha_j = \frac{\exp(e_j)}{\sum_j \exp(e_j)}$$

$$e_j = \langle v_a, h_i \rangle,$$

$$h_i = f(x_1, \dots, x_n)$$

$f$  — свёрточная или рекуррентная сеть для контекста

# Простой механизм внимания: иллюстрация



[Wang et al., 2016. Attention-based LSTM for Aspect-level Sentiment Classification]



## Простой механизм внимания для анализа тональности

- Механизм внимания работает с обучаемым представлением аспекта  $v_a$  (для ресторана стоимость, обслуживание и т.д.) и выдачей энкодера, не зависящей от аспекта.
- Это означает, что модель можно обучать одновременно на несколько аспектов.

# Механизм внимания: переобозначение

- Текущая формула (переобозначения):

$$h = \sum_i \alpha_i h_i^{value},$$

$$\alpha_i \sim \exp(\langle h_i^{key}, s \rangle),$$

$s$  – глобальный вектор “запроса” (query),

$h_i^{value}$  – “эмбеddинг-значение” (value),

$h_i^{key}$  – “эмбеddинг-ключ” (key).

- Откуда взять  $h_i^{value}$ ,  $h_i^{key}$ .

# Механизм внимания: переобозначение

- Текущая формула (переобозначения):

$$h = \sum_i \alpha_i h_i^{value},$$

$$\alpha_i \sim \exp(\langle h_i^{key}, s \rangle),$$

$s$  – глобальный вектор “запроса” (query),

$h_i^{value}$  – “эмбеddинг-значение” (value),

$h_i^{key}$  – “эмбеddинг-ключ” (key).

- Откуда взять  $h_i^{value}$ ,  $h_i^{key}$ .
- Проще всего вставить один слой персептрона:

$$\begin{aligned} h_i^{value} &= g(W^{value} h_i), \\ h_i^{key} &= g(W^{key} h_i) \end{aligned}$$

# Механизм внимания: матричный вид

- Всё можно переписать в матричном виде:

$$\begin{aligned}h &= A_{1 \times L} V_{L \times d}, \\A &= \text{softmax}(q_{1 \times d} K_{L \times d}^T), \\V &= g(H_{L \times d} W_{d \times d}^{\text{value}}), \\K &= g(H_{L \times d} W_{d \times d}^{\text{key}})\end{aligned}$$

- Финальная формула:

$$h = \text{softmax}(QK^T)V$$

## Механизм внимания: матричный вид

- Всё можно переписать в матричном виде:

$$\begin{aligned}h &= A_{1 \times L} V_{L \times d}, \\A &= \text{softmax}(q_{1 \times d} K_{L \times d}^T), \\V &= g(H_{L \times d} W_{d \times d}^{\text{value}}), \\K &= g(H_{L \times d} W_{d \times d}^{\text{key}})\end{aligned}$$

- Финальная формула:

$$h = \text{softmax}(QK^T)V$$

- На практике добавляют нормализующий множитель:

$$h = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

- Без него обучение более нестабильное.

## Механизм самовнимания : матричный вид

- Механизм внимания используется, чтобы посчитать состояние для всего предложения с учётом всех слов.
- А что если так же считать новые состояния для всех слов?
- В этом случае даже удалённые слова будут влиять на текущий вектор (проблема для рекуррентных сетей).

# Механизм самовнимания : матричный вид

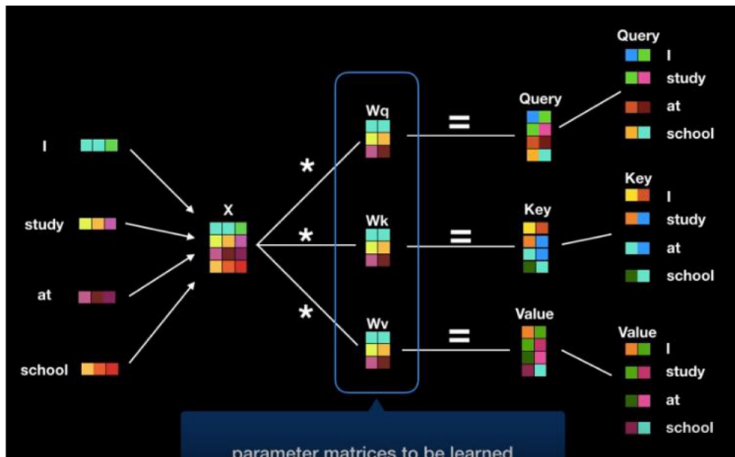
- Механизм внимания используется, чтобы посчитать состояние для всего предложения с учётом всех слов.
- А что если так же считать новые состояния для всех слов?
- В этом случае даже удалённые слова будут влиять на текущий вектор (проблема для рекуррентных сетей).
- Достаточно сделать “запрос”  $Q$  матрицей:

$$Q_{L \times d} = g(H_{L \times d} W_{d \times d}^{query})$$

- В итоге получаем:

































































$$\begin{aligned} H' &= A_{L \times L} V_{L \times d}, \\ A &= \text{softmax}(Q_{L \times d} K_{L \times d}^T), \\ Q &= g(H_{L \times d} W_{d \times d}^{query}), \\ V &= g(H_{L \times d} W_{d \times d}^{value}), \\ K &= g(H_{L \times d} W_{d \times d}^{key}) \end{aligned}$$

# Механизм самовнимания: иллюстрация



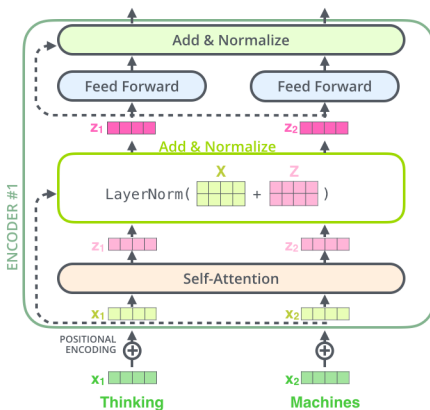


# Механизм самовнимания: иллюстрация

	Query * Key <sup>T</sup>	Score	Softmax	Value	Softmax * Value	Σ Softmax * Value (Attention layer output)
<b>I</b>	I * I  *  = 130		0.92	 I		}
	I * study  *  = 50		0.05	 study		
	I * at  *  = 20		0.02	 at		
	I * school  *  = 10		0.01	 school		
<b>study</b>	study * I  *  = 30		0.02			}
	study * study  *  = 110		0.70			
	study * at  *  = 20		0.03			
	study * school  *  = 70		0.25			
<b>at</b>	at * I  *  = 30		0.03			}
	at * study  *  = 50		0.10			
	at * at  *  = 90		0.80			
	at * school  *  = 40		0.07			
<b>school</b>	school * I  *  = 30		0.01			}
	school * study  *  = 80		0.27			
	school * at  *  = 23		0.02			
	school * school  *  = 160		0.70			

# Механизм самовнимания: трансформеры

- Механизм внимания – это один слой трансформерной архитектуры.
- Между такими слоями вставляются residual-переходы полностью связанные подслои и слой-нормализация (LayerNorm). Параллельно с трансформерным блоком вставляется residual-переход.



## Механизм самовнимания: множественное внимание

- Может возникнуть потребность проявлять внимание с точки зрения разных аспектов и к разным словам.

Вася съел большую банку варенья.

банку → большую      морфология

банку → съел            семантика

# Механизм самовнимания: множественное внимание

- Может возникнуть потребность проявлять внимание с точки зрения разных аспектов и к разным словам.

Вася съел большую банку варенья.

банку → большую      морфология

банку → съел            семантика

- Это делается с помощью множественного внимания (multi-head attention):

$$H'_i = \text{Attention}(HW_{q,i}, HW_{k,i}, HW_{v,i})$$

$$H' = \text{Concat}(H'_1, \dots, H'_m)$$

$$W_{*,i} \in \mathbb{R}^{D \times \frac{D}{m}}$$

- Все эти операции можно делать параллельно.

# Механизм самовнимания: множественное внимание



## Механизм самовнимания: энкодер-декодер

- В энкодере механизм внимания нужен, чтобы пересчитывать состояния модели, кодирующие исходное предложение.
- В декодере нужно учитывать не только исходное предложение, но и сгенерированную часть результата.

# Механизм самовнимания: энкодер-декодер

- В энкодере механизм внимания нужен, чтобы пересчитывать состояния модели, кодирующие исходное предложение.
- В декодере нужно учитывать не только исходное предложение, но и сгенерированную часть результата.
- Как следствие, есть два подслоя внимания:
  - Внимание состояний энкодера к состояниям декодера  $\alpha_{ij}$ :

$i$  – позиция в генерируемом тексте,  
 $j$  – позиция в исходном тексте.

# Механизм самовнимания: энкодер-декодер

- В энкодере механизм внимания нужен, чтобы пересчитывать состояния модели, кодирующие исходное предложение.
- В декодере нужно учитывать не только исходное предложение, но и сгенерированную часть результата.
- Как следствие, есть два подслоя внимания:

- Внимание состояний энкодера к состояниям декодера  $\alpha_{ij}$ :

$i$  – позиция в генерируемом тексте,

$j$  – позиция в исходном тексте.

- Внимание состояний энкодера к состояниям декодера  $\beta_{ij}$ :

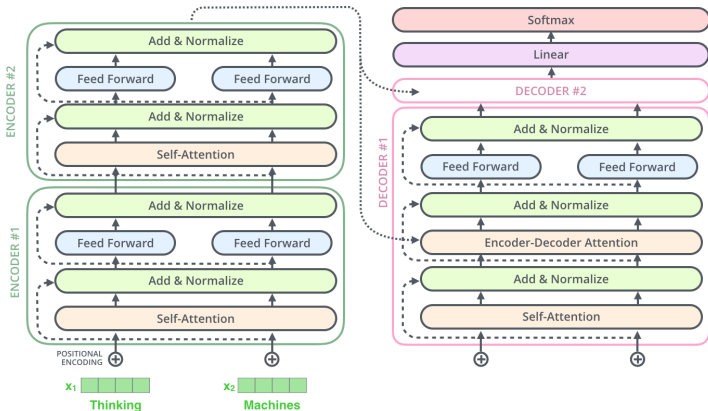
$i$  – позиция в генерируемом тексте,

$j$  – позиция в генерируемом тексте,  $j < i$ .

- При обучении считается  $\beta_{ij} = 0$  при  $j \geq i$ , чтобы модель не заглядывала в будущее.



# Трансформеры: энкодер-декодер



# Трансформеры: скорость

- Скорость передачи информации на  $m$  шагов:
  - Свёрточные сети:  $O(\frac{m}{w})$ .
  - Рекуррентные сети:  $O(m)$ .
  - Трансформеры:  $O(1)$ .
- Затраты памяти на последовательность длины  $L$ :
  - Свёрточные сети:  $O(wd^2L)$ .
  - Рекуррентные сети:  $O(d^2L)$ .
  - Трансформеры:  $O(L^2d)$ .

# Трансформеры: позиционное кодирование

- Пока механизм внимания никак не различает одинаковые вектора, стоящие в разных местах.
- Чтобы это исправить, конкатенируют вектора слов с позиционными эмбедингами:

$$x_i = [emb(word_i), x_{pos}(i)]$$

# Трансформеры: позиционное кодирование

- Пока механизм внимания никак не различает одинаковые вектора, стоящие в разных местах.
- Чтобы это исправить, конкатенируют вектора слов с позиционными эмбедингами:

$$x_i = [emb(word_i), x_{pos}(i)]$$

- $x_{pos}$  иногда задают явно (Vaswani et al., 2017):

$$\begin{aligned}(x_{pos}(i))_{2j} &= \sin \frac{i}{10000^{j/d}}, \\ (x_{pos}(i))_{2j+1} &= \cos \frac{i}{10000^{j/d}}.\end{aligned}$$

- В более поздних подходах их сделали обучаемыми:
  - Это дополнительная матрица размера  $\max\_length \times d$ .
  - Последовательности длиннее  $\max\_length$  не обрабатываются.