

Машинное обучение (Machine Learning)

Attention

Уткин Л.В.

Санкт-Петербургский политехнический университет Петра Великого





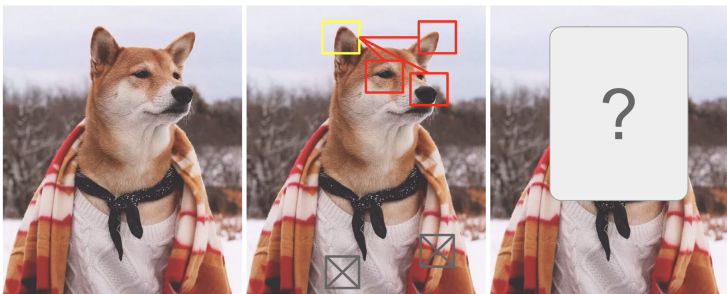
Attention

Attention - основные понятия



Мотивация

- Внимание в некоторой степени мотивируется тем, как мы обращаем визуальное внимание на различные области изображения или сопоставляем слова в одном предложении



<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html>

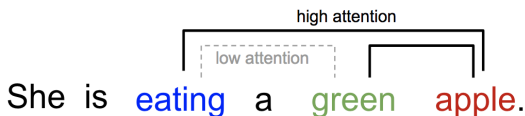


Мотивация

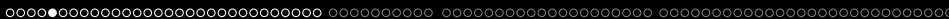
- Внимание человека фокусируется на определенной области с “большим разрешением” (взгляд на острое ухо в желтой рамке), воспринимая окружающее изображение в “малом разрешении” (снежный фон и одежда), а затем отрегулирует фокус или сделает вывод соответствующим образом.
- При имеющемся небольшом участке изображения, остальные части дают подсказки, что там должно изображаться. Ожидаем увидеть заостренное ухо в желтой рамке, потому что мы видели нос собаки, другое острое ухо справа и глаза Сибы (в красных рамках).
- Тем не менее, свитер и одеяло в нижней части не будут так полезны, как эти черты собаки.



Мотивация

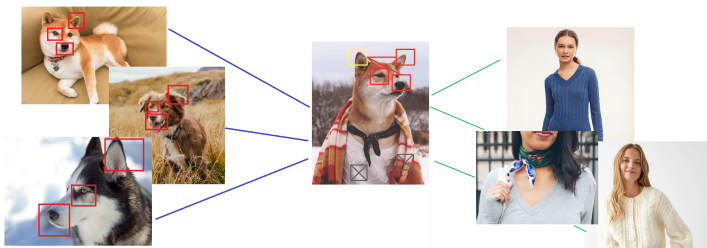


- Когда мы видим «едят», мы ожидаем, что очень скоро встретимся со словом «еда».
- Цветом выделены слова описывающие еду, но, вероятно, не так, как «еда» напрямую.



Мотивация

- Attention можно интерпретировать как вектор весов важности (attention vector)
- Чтобы предсказать или сделать вывод об одном элементе, таком как пиксель в изображении или слово в предложении, оцениваем, используя этот вектор, насколько сильно он соотносится (коррелирует) с другими элементами



Attention is all you need

- **NLP:** A. Vaswani et al. Attention is all you need // Advances in neural information processing systems, 2017
- **Computer vision:**
 - **Vision Transformer (ViT):** A. Dosovitskiy et al. in An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // arXiv:2010.11929, 2020
 - **Performer:** K. Choromanski et al. Rethinking Attention with **Performers** // arXiv:2009.14794, 2020
- **Audio:** Reformer: H.R. Ihm et al. **Reformer-TTS:** Neural Speech Synthesis with Reformer Network // Proceedings of Interspeech, 2012–2016. 2020.
- L. Madaan et al. **Treeformer:** Dense Gradient Trees for Efficient Attention Computation // arXiv:2208.0901, 2022
- **etc. etc.**



Регрессия Надарая-Уотсона (1)

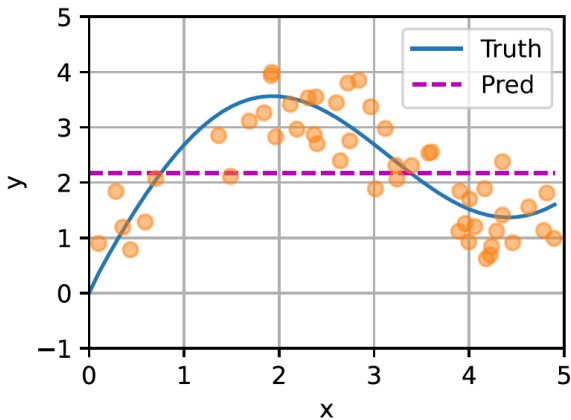
- Обучающая выборка: n примеров
 $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i = (x_{i1}, \dots, x_{im}) \in \mathbb{R}^m$,
 $y_i \in \mathbb{R}$
- Регрессионная модель $f : \mathbb{R}^m \rightarrow \mathbb{R}$, прогнозирует предсказание $y = f(\mathbf{x})$ для нового примера \mathbf{x}
- Самая простая оценка:

$$\tilde{y} = f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n y_i$$



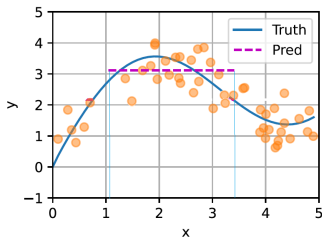
Регрессия Надарая-Уотсона (2)

$$\tilde{y} = f(x) = \frac{1}{n} \sum_{i=1}^n y_i$$



Другое простое решение с k ближ. соседями

$$\tilde{y} = f(x) = k^{-1} \sum_{i=1}^k y_i : k \text{ ближайших точек к } x$$



- k ближайших примеров к x имеют веса $1/k$, другие примеры имеют нулевые веса: k примеров более важны, так как они ближе к x

Обобщение: регрессия Надарая-Уотсона

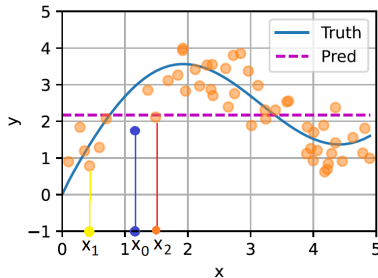
- E.A. Nadaraya. On estimating regression. Theory of Probability & Its Applications, 9(1):141–142, 1964
- G.S. Watson. Smooth regression analysis. Sankhya: The Indian Journal of Statistics, Series A, 359–372, 1964

$$\tilde{y} = f(\mathbf{x}) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i) \cdot y_i$$

- $\alpha(\mathbf{x}, \mathbf{x}_i)$ - вес внимания (attention weight)
характеризует насколько пример \mathbf{x}_i близок к \mathbf{x} (по расстоянию)
- Для оценки $f(\mathbf{x})$, метки y_i из датасета взвешиваются в соответствии с тем, насколько вектор признаков \mathbf{x}_i близок к \mathbf{x}
- Ближе \mathbf{x}_i к \mathbf{x} , больше вес y_i



Смысл весов точек



- Вектор x_2 ближе к x_0 , чем к x_1
- Следовательно, вес x_2 должен быть больше, чем вес x_1

Регрессия Надарая-Уотсона

$$\tilde{y} = f(\mathbf{x}) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i) \cdot y_i$$

- $\alpha(\mathbf{x}, \mathbf{x}_i)$: вес внимания (**attention weight**) характеризует как близко пример \mathbf{x}_i к \mathbf{x} (по расстоянию)
- Ядро $K(\mathbf{x}, \mathbf{x}_i)$ - мера близости

$$\alpha(\mathbf{x}, \mathbf{x}_i) = \frac{K(\mathbf{x}, \mathbf{x}_i)}{\sum_{j=1}^n K(\mathbf{x}, \mathbf{x}_j)}$$

- Элементы внимания: \mathbf{x} - **query**, \mathbf{x}_i - **keys**, y_i - **values**
- Веса всех пар (\mathbf{x}_i, y_i) - распределение вероятностей: они неотрицательные и в сумме равны 1



Вес внимания и гауссово ядро (1)

- Гауссово ядро:

$$K(\mathbf{x}, \mathbf{x}_i) = \exp\left(-\|\mathbf{x} - \mathbf{x}_i\|^2 / \sigma\right)$$

- Вес внимания:

$$\alpha(\mathbf{x}, \mathbf{x}_i) = \frac{K(\mathbf{x}, \mathbf{x}_i)}{\sum_{j=1}^n K(\mathbf{x}, \mathbf{x}_j)} = \text{softmax}\left(-\|\mathbf{x} - \mathbf{x}_i\|^2 / \sigma\right)$$

- Регрессия Надарая-Уотсона:

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i) \cdot y_i = \sum_{i=1}^n \text{softmax}\left(-\|\mathbf{x} - \mathbf{x}_i\|^2 / \sigma\right) y_i$$



Весы внимания и гауссово ядро (2)

- Регрессия Надарая-Уотсона:

$$f(\mathbf{x}) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i) \cdot y_i = \sum_{i=1}^n \text{softmax} \left(- \|\mathbf{x} - \mathbf{x}_i\|^2 / \sigma \right) y_i$$

- Предполагая, что keys и query нормализованы ($\|\mathbf{x}\|_2 = \|\mathbf{x}_i\|_2 = 1$), тогда

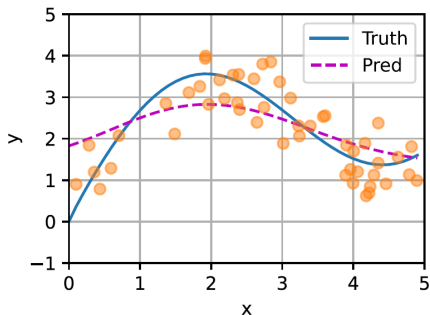
$$\|\mathbf{x} - \mathbf{x}_i\|_2^2 = 2(1 - \mathbf{x}^T \mathbf{x}_i),$$

$$f(\mathbf{x}) = \sum_{i=1}^n \text{softmax} \left(\frac{\mathbf{x}^T \mathbf{x}_i}{\sigma} \right) y_i.$$



Непараметрическая модель внимания

- Если σ задано, например 1, то получаем непараметрическую модель внимания



Регрессия с обучаемым параметром (1)

- Пусть σ - параметр обучения

$$\tilde{y} = f(\mathbf{x}) = \sum_{i=1}^n \text{softmax} \left(- \|\mathbf{x} - \mathbf{x}_i\|^2 / \sigma \right) y_i$$

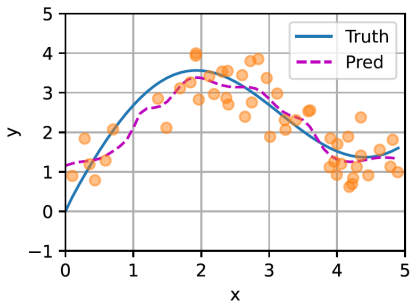
- Обучение - стандартная функция потерь L_2 :

$$\sum_{k=1}^n (y_k - f(\mathbf{x}_k))^2 \rightarrow \min_{\sigma}$$

или

$$\sum_{k=1}^n \left(y_k - \sum_{i=1}^n \text{softmax} \left(- \|\mathbf{x}_k - \mathbf{x}_i\|^2 / \sigma \right) y_i \right)^2 \rightarrow \min_{\sigma}$$

Регрессия с обучаемым параметром (2)



Больше обучаемых параметров, лучше предсказание модели с вниманием, но при условии достаточного числа обучающих данных.



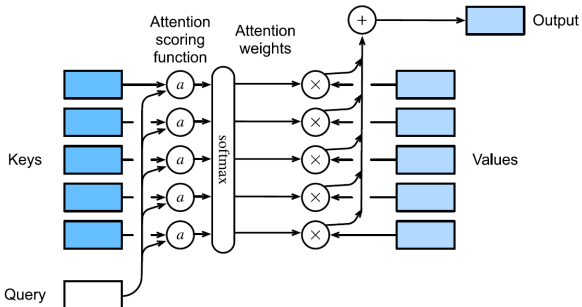
Регрессия Надарая-Уотсона (итог)

- Ядерная регрессия Надарая-Уотсона — пример машинного обучения с механизмом внимания
- Модель внимания в соответствии с регрессией Надарая-Ватсона - средневзвешенное значение меток обучающей выборки
- Вес внимания присваивается примеру (value, y_i) на основе запроса (query, x) и ключа (key, x_i), связанного с примером
- Модель внимания может быть непараметрической или параметрической



Модель внимания в более общем виде

- $a(x, x_i; \sigma) = \exp\left(-\|x - x_i\|^2 / 2\sigma\right)$ - скоринговая функция внимания (функция оценки)



Модель внимания в более общем виде

- Имеются вектор query $\mathbf{q} \in \mathbb{R}^q$, пары векторов key-value $(\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_n, \mathbf{v}_n)$, $\mathbf{k}_i \in \mathbb{R}^k$, $\mathbf{v}_i \in \mathbb{R}^v$,
- Модель внимания (пулинг внимания) f :

$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_n, \mathbf{v}_n)) = \sum_{i=1}^n \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i \in \mathbb{R}^v$$

где

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^n \exp(a(\mathbf{q}, \mathbf{k}_j))} \in \mathbb{R}$$

- Выбор скоринговой функции a определяет вид модели внимания.

Additive attention (Bahdanau et al.)

- Скоринговая функция a :

$$a(\mathbf{q}, \mathbf{k}) = \mathbf{w}_v^T \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R}$$

где $\mathbf{W}_q \in \mathbb{R}^{h \times q}$, $\mathbf{W}_k \in \mathbb{R}^{h \times k}$, $\mathbf{w}_v \in \mathbb{R}^h$ - параметры обучения

- запрос и ключ (query и key) конкатенируются и передаются в нейронную сеть с одним скрытым слоем, количество скрытых нейронов которого равно гиперпараметру h .



Scaled Dot-Product attention (Thang Luong et al.)

- Более эффективная с вычислительной точки зрения скоринговая функция - скалярное произведение.
- Но операция скалярного произведения требует, чтобы и запрос, и ключ имели одинаковую длину вектора, скажем, d .
- Предположим, что все элементы запроса и ключа являются независимыми случайными величинами с нулевым средним и единичной дисперсией, тогда скалярное произведение обоих векторов имеет нулевое среднее значение и дисперсию d .



Scaled Dot-Product attention

- Чтобы гарантировать, что дисперсия скалярного произведения по-прежнему остается единицей независимо от длины вектора, масштабированная скоринговая функция имеет вид:

$$\alpha(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k} / \sqrt{d}$$

- Scaled Dot-Product attention для n запросов:

$$\text{softmax} \left(\frac{\mathbf{QK}^T}{\sqrt{d}} \right) \mathbf{V} \in \mathbb{R}^{n \times n}, \mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{m \times d}, \mathbf{V} \in \mathbb{R}^{m \times v}$$

- где $\mathbf{Q} = \mathbf{Q}_0 \mathbf{W}_q \in \mathbb{R}^{n \times n}$, $\mathbf{K} = \mathbf{K}_0 \mathbf{W}_k \in \mathbb{R}^{m \times d}$,
 $\mathbf{V} = \mathbf{V}_0 \mathbf{W}_v \in \mathbb{R}^{m \times v}$



Модели внимания

- Можно вычислить результат модели внимания как средневзвешенное значение, где разные варианты скоринговых функций приводят к разному поведению модели внимания.
- Когда запросы и ключи являются векторами разной длины, можно использовать функцию Additive attention.
- Когда запросы и ключи одинаковы, масштабированная скоринговая функция скалярного произведения более эффективна в вычислительном отношении.



Модель внимания для временного ряда

- Модель внимания объединяет временные характеристики с использованием динамически генерируемых весов, позволяя напрямую фокусироваться на важных временных моментах времени в прошлом, даже если они очень далеки в скользящем окне.
- Модель внимания имеет вид:

$$\tilde{\mathbf{h}}_t = \sum_{\tau=0}^k \alpha(t, \tau) \mathbf{h}_{t-\tau}$$

где $\mathbf{h}_{t-\tau}$ - промежуточный вектор признаков,
 $\alpha(t, \tau) \in [0, 1]$ - вес внимания для $t - \tau$ момента времени, $\tilde{\mathbf{h}}_t$ - выходной вектор



Self-attention - предварительно

- “Self-attention is an attention relating different positions of a single sequence in order to compute a representation of the sequence.” (*A. Vaswani et al. Attention is All You Need*)
- **Цель** в NLP: Он изменяет стандартную структуру LSTM, заменяя ячейку памяти сетью памяти. Это связано с тем, что сети памяти имеют набор key-векторов и набор value-векторов, тогда как LSTM поддерживают скрытый вектор и вектор памяти.



Self-attention

- Дана последовательность токенов $\mathbf{x}_1, \dots, \mathbf{x}_n$, где $\mathbf{x}_i \in \mathbb{R}^d$. Ее self-attention - последовательность такой же длины $\mathbf{x}_1^*, \dots, \mathbf{x}_n^*$, где

$$\mathbf{x}_i^* = f(\mathbf{x}_i, (\mathbf{x}_i, \mathbf{x}_i), \dots, (\mathbf{x}_n, \mathbf{x}_n)) \in \mathbb{R}^d$$

- Self-attention как non-local means denoising:

$$\mathbf{x}_i^* = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i) \mathbf{x}_i = \sum_{i=1}^n \text{softmax} \left(\frac{-\|\mathbf{x} - \mathbf{x}_i\|^2}{\tau} \right) \mathbf{x}_i$$

- $\mathbf{q}_i = \mathbf{k}_i = \mathbf{v}_i = \mathbf{y}_i$: query=keys=values

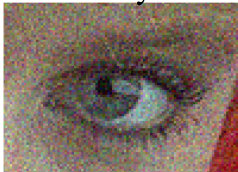
Non-local means denoising

- Self-attention как метод сглаживания (non-local means denoising):

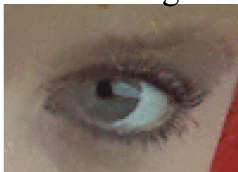
$$\mathbf{x}^* = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i) \mathbf{x}_i = \sum_{i=1}^n \text{softmax} \left(\frac{-\|\mathbf{x} - \mathbf{x}_i\|^2}{\sigma} \right) \mathbf{x}_i$$

- Удаление шума путем вычисления средней интенсивности каждого пиксела по соседним пикселам

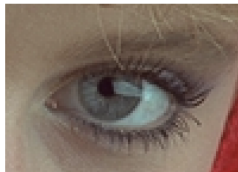
Noisy



Denoising



Ground Truth





Multi-head attention (многомерное внимание)

- Пусть σ - гиперпараметр

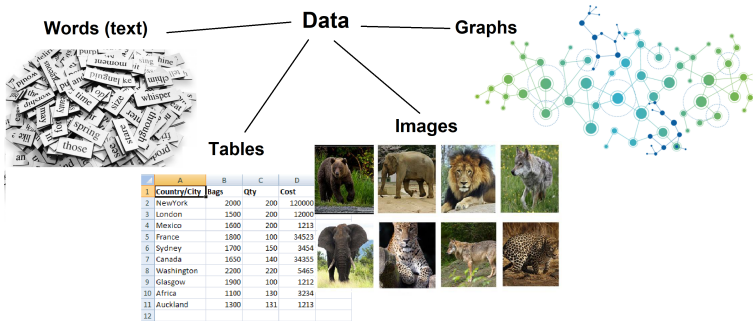
$$f(\mathbf{x}) = \sum_{i=1}^n \text{softmax} \left(- \|\mathbf{x} - \mathbf{x}_i\|^2 / \sigma \right) y_i$$

- Возьмем $\sigma_1, \sigma_2, \dots, \sigma_s$ вместо σ и получим $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_s(\mathbf{x})$
- Можно конкатенировать $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_s(\mathbf{x})$ или усреднить

RF-based attention

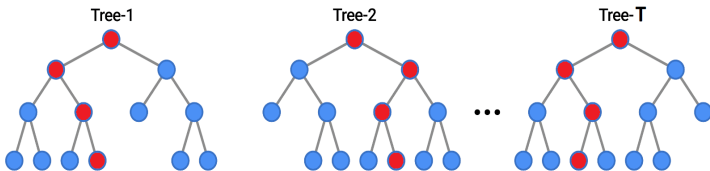
Attention и случайный лес

Data





Random forest

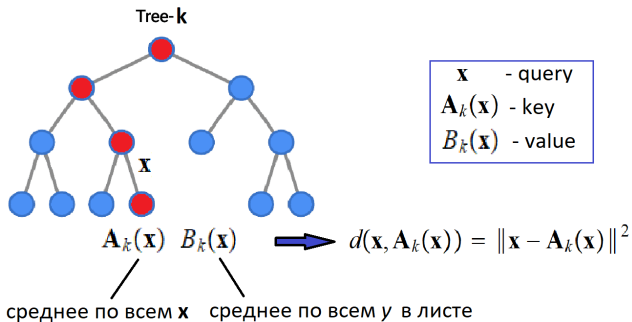




Случайный лес и внимание (идеи)

- Представить деревья как keys и values в терминах регрессии Надарая-Уотсона, т.е. адаптировать регрессию Надарая-Уотсона к лесу
- Назначить обучаемые (параметрические) веса внимания каждому дереву, так чтобы они зависели от каждого примера и от дерева, а также учились на всех примерах.
- Предложить способ связи весов внимания с деревьями и обучающими (тестовыми) примерами

Query, keys, values



Обучение параметров веса внимания

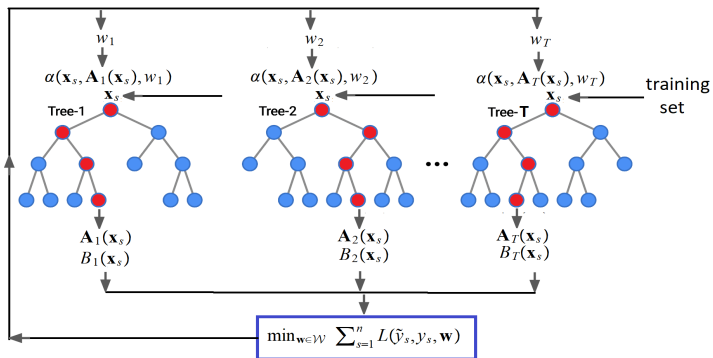
- Регрессия Надарая-Уотсона с query \mathbf{x} , keys $\mathbf{A}_k(\mathbf{x})$, values $B_k(\mathbf{x}) = \tilde{y}_k$:

$$\tilde{y} = \sum_{k=1}^T \alpha(\mathbf{x}, \mathbf{A}_k(\mathbf{x}), \mathbf{w}) \cdot \tilde{y}_k$$

- $\alpha(\mathbf{x}, \mathbf{A}_k(\mathbf{x}), \mathbf{w})$ - вес внимания с обучаемыми параметрами \mathbf{w}
-

$$\begin{aligned} \mathbf{w}_{opt} &= \arg \min_{\mathbf{w} \in \mathcal{W}} \sum_{s=1}^n L(\tilde{y}_s, y_s, \mathbf{w}) \\ &= \arg \min_{\mathbf{w} \in \mathcal{W}} \sum_{s=1}^n \left(y_s - \sum_{k=1}^T \alpha(\mathbf{x}_s, \mathbf{A}_k(\mathbf{x}_s), \mathbf{w}) \cdot \tilde{y}_k \right)^2 \end{aligned}$$

Иллюстрация процесса обучения



Self-attention (самовнимание)

- Возвращаемся к Н-У регрессии

$$\tilde{y} = \sum_{k=1}^T \alpha(\mathbf{x}, \mathbf{A}_k(\mathbf{x}), \mathbf{w}) \cdot \tilde{y}_k(\mathbf{x})$$

- Рассмотрим операцию самовнимания для $\tilde{y}_k(\mathbf{x})$ с обучаемыми параметрами $\mathbf{v} = (v_1, \dots, v_T)$:

$$y_k^*(\mathbf{x}) = \sum_{i=1}^T \beta(\tilde{y}_k, \tilde{y}_i, \mathbf{v}) \cdot \tilde{y}_i(\mathbf{x})$$

- Отсюда

$$\tilde{y} = \sum_{i=1}^T \alpha(\mathbf{x}, \mathbf{A}_i(\mathbf{x}), \mathbf{w}) \cdot \sum_{k=1}^T \beta(\tilde{y}_i, \tilde{y}_k, \mathbf{v}) \cdot \tilde{y}_k(\mathbf{x})$$

Multi-head self-attention (многомерное самовнимание)

- Самовнимание самовнимания:

$$\begin{aligned}\tilde{y} &= \sum_{i=1}^T \sum_{k(1)=1}^T \sum_{k(2)=1}^T \cdots \sum_{k(t)=1}^T \alpha(\mathbf{x}, \mathbf{A}_i(\mathbf{x}), \mathbf{w}) \\ &\times \beta_1(\tilde{y}_i, \tilde{y}_{k(1)}, \mathbf{v}^{(1)}) \times \beta_2(\tilde{y}_{k(1)}, \tilde{y}_{k(2)}, \mathbf{v}^{(2)}) \cdots \\ &\times \beta_t(\tilde{y}_{k(t-1)}, \tilde{y}_{k(t)}, \mathbf{v}^{(t)}) \cdot \tilde{y}_{k(t)}\end{aligned}$$



NLP

Attention и NLP

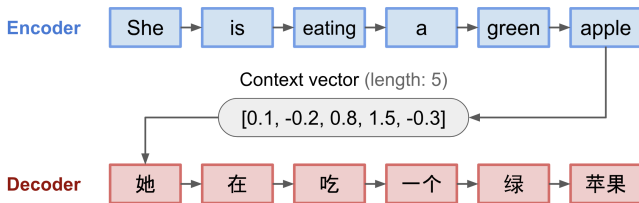
Материалы по NLP и трансформерам являются компиляцией и заимствованием материалов из замечательного курса и презентации К.В. Воронцова.



Что не так с моделью Seq2Seq

- Модель seq2seq разработана для NLP
- seq2seq преобразует входную последовательность (источник) в новую (цель), и обе последовательности могут иметь произвольную длину
- Примеры - машинный перевод, генерация диалоговых вопросов и ответов и т.д.
- seq2seq имеет архитектуру кодер-декодер:
 - Кодер обрабатывает входную последовательность и сжимает информацию в контекстный вектор (embedding) фиксированной длины
 - Декодер инициализируется контекстным вектором, чтобы выдать преобразованный вывод
- Кодер и декодер являются РНН, использует LSTM

Что не так с моделью Seq2Seq



- Недостаток - невозможность запоминания длинных предложений, он забывает первую часть, когда завершает обработку всего ввода
- Attention пытается решить эту проблему

Зачем Attention

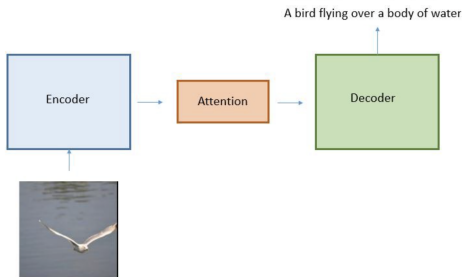
- Attention был создан, чтобы помочь запомнить длинные исходные предложения в машинном переводе (NMT)
- Вместо создания одного вектора контекста из последнего скрытого состояния кодера, идея - создание shortcuts между вектором контекста и всем исходным вводом
- Вес этих shortcuts настраивается для каждого элемента вывода



Attention для перевода

- Хотя контекстный вектор имеет доступ ко всей входной последовательности, необходимо беспокоиться о забывании
- Выравнивание между источником и целью обучается и контролируется контекстным вектором
- Контекстный вектор использует три фрагмента информации:
 - 1 кодер скрытых состояний
 - 2 декодер скрытых состояний
 - 3 выравнивание между источником и целью

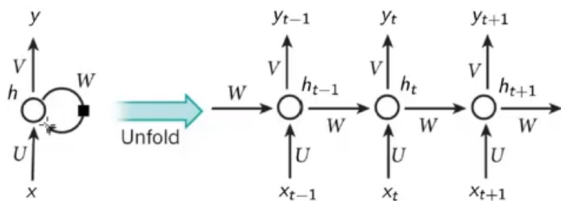
Attention для почти перевода





RNN

- x_t - входной вектор в момент $t = 1, \dots, T$
- h_t - вектор скрытого состояния в момент t
- y_t - выходной вектор
- $h_t = \sigma_h(Ux_t + Wh_{t-1}); y_t = \sigma_y(Vh_t)$





RNN (обучение)

- x_t - входной вектор в момент $t = 1, \dots, T$
- h_t - вектор скрытого состояния в момент t
- y_t - выходной вектор
- $h_t = \sigma_h(Ux_t + Wh_{t-1}); y_t = \sigma_y(Vh_t)$

Обучение RNN: $\sum_{t=0}^T L_t(U, V, W) \rightarrow \min_{U, V, W}$

- длины входного и выходного сигнала должны совпадать, невозможно загадывание вперед



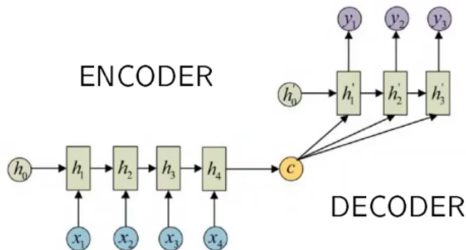
RNN: Seq2Seq (более формально)

$X = (x_1, \dots, x_n)$ - входная последовательность

$Y = (y_1, \dots, y_m)$ - выходная последовательность

$c \equiv h_n$ кодирует всю информацию про X для синтеза Y

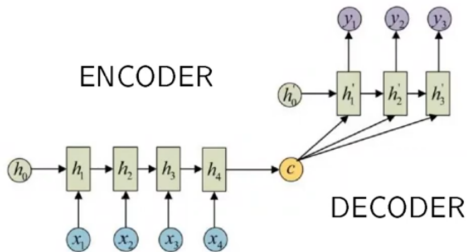
$h_i = f_{in}(x_i, h_{i-1}); h'_t = f_{out}(h'_{t-1}, y_{t-1}, c); y_t = f_y(h'_t, y_{t-1})$





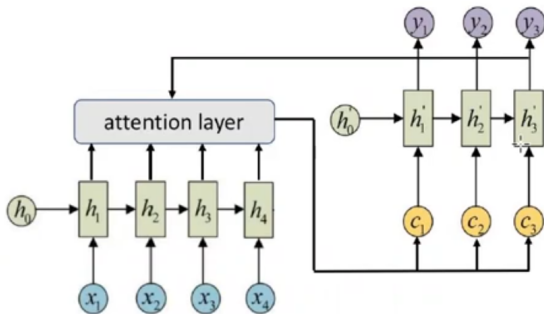
RNN: Seq2Seq (более формально)

- h_n лучше помнит конец последовательности, чем начало
- чем больше n , тем труднее упаковать информацию в c
- придется контролировать затухание градиента
- RNN трудно распараллеливать



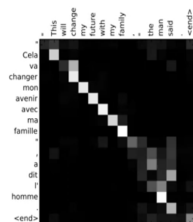
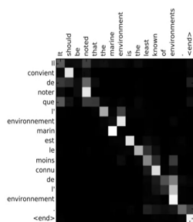
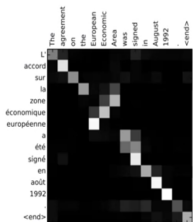
Attention и RNN

- можно отказаться от рекуррентности как по h_i , так и по h'_t
- можно вводить обучаемые параметры в a и c



Attention - интерпретируемость

При обработке конкретной последовательности X визуализация матрицы α_{ti} показывает, на какие слова x_i модель обращает внимание, генерируя слово перевода y_t

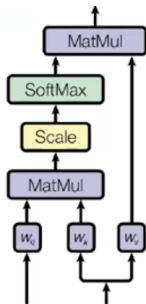


Функции сходства (1)

- $a(h, h') = h^T h'$ - скалярное произведение
- $a(h, h') = \exp(h^T h')$ - норм превращается в SoftMax
- $a(h, h') = h^T W h'$ - с матрицей обучаемых параметров W
- $a(h, h') = w^T \text{th}(Uh + Vh')$ - аддитивное внимание с w, U, V

Функции сходства (2)

- Линейное преобразование векторов query, key, value:
$$a(h_i, h'_{t-1}) = (W_k h_i)^T (W_q h'_{t-1}) / \sqrt{d};$$
$$\alpha_{ti} = \text{SoftMax}_i a(h_i, h'_{t-1}); c_t = \sum_i \alpha_{ti} W_v h_i$$
- $W_{q, d \times \dim(h')}$, $W_{k, d \times \dim(h)}$, $W_{v, d \times \dim(h)}$ - матрицы весов линейных нейронов (обучаемые линейные преобразования в пространство размерности d); часто $W_k = W_v$ для простоты



Attention - веса внимания - 2

- q - вектор-запрос, для которого вычисляется контекст
- $K = (k_1, \dots, k_n)$ - векторы-ключи, сравниваемые с запросом
- $V = (v_1, \dots, v_n)$ - векторы-значения, образующие контекст
- $a(k_i, q)$ - оценка сходства ключа k_i запросу q
- c - искомый вектор контекста, релевантный запросу

Attention - веса внимания -2

- *Модель внимания* - это 3-х-слойная сеть, вычисляющая выпуклую комбинацию значений v_i , релевантных запросу q

$$c = \text{Attn}(q, K, V) = \sum_i v_i \text{SoftMax}_i(k_i, q)$$

$c_t = \text{Attn}(W_q h'_{t-1}, W_k H, W_v H)$ - пример с пред. слайда, $H = (h_1, \dots, h_n)$ - входные векторы, h'_{t-1} - выходной

- *Самовнимание* (self-attention):
 $c_t = \text{Attn}(W_q h_i, W_k H, W_v H)$ - частный случай, когда $h_i \in H$

Многомерное внимание (multi-head attention)

Идея: J разных моделей внимания совместно обучаются выделять различные аспекты информации (части речи, синтаксис, и т.д.):

$$c^j = \text{Attn}(W_q^j q, W_k^j H, W_v^j H), \quad j = 1, \dots, J$$

Варианты агрегирования выходного вектора:

- $c = J^{-1} \sum_{j=1}^J c^j$ - усреднение
- $c = [c^1, \dots, c^J]$ - конкатенация
- $c = [c^1, \dots, c^J] W$ - возврат к нужной разм-ти

Регуляризация: чтобы аспекты внимания были максимально различны, строки $J \times n$ матриц A , $\alpha_{ji} = \text{SoftMax}_i a(W_k^j h_i, W_q^j q)$ декоррелируются ($\alpha_j^T \alpha_j \rightarrow 1$):

$$\|AA^T - I\|^2 \rightarrow \min_{\{W_k^j, W_q^j\}}$$

Иерархическое внимание (hierarchical attention - 1)

Вложенная структура: слова \in предложения \in документы

x_{it} - слова $t = 1, \dots, T_i$ в предложениях $i = 1, \dots, L$

Сеть первого (нижнего) уровня, обучение эмбедингов

s_i :

$h_{it} = BidirGRU(W_0 x_{it})$ - GRU для векторизации слов

$h_{it} = th(W_1 h_{it} + b_1)$ - обучаемое преобразование Key

$s_i = \sum_t h_{it} \text{SoftMax}_t(u_{it}^T q_1)$ - эмбединг предложения, Query

q_1

Иерархическое внимание (hierarchical attention - 2)

Сеть второго (верхнего) уровня, обучение эмбедингов v :

$h_i = BidirGRU(s_i)$ - GRU для векторизации предложений

$u_i = th(W_2 h_i + b_2)$ - обучаемое преобразование Key

$v = \sum_i h_i \text{SoftMax}_t(u_i^T q_2)$ - эмбединг предложения, Query q_2

Максимизация правдоподобия для классификации документов:

$$\sum_d \sum_y \ln(\text{SoftMax}_y(W_y v + b_y)) \rightarrow \max$$

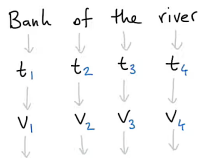
Transformer

Transformer

Self-attention (1)

Bank of the river
↓ ↓ ↓ ↓
 t_1 t_2 t_3 t_4
↓ ↓ ↓ ↓

Self-attention (2)



Self-attention (3-1)

Bank of the river.

t_1 t_2 t_3 t_4

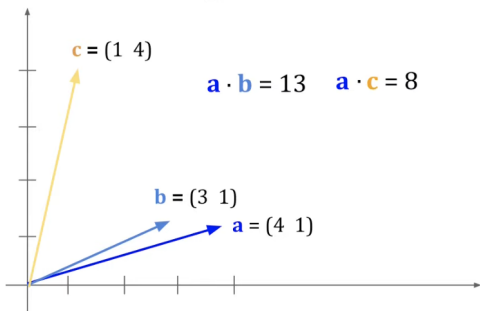
v_1 v_2 v_3 v_4

ATTENTION

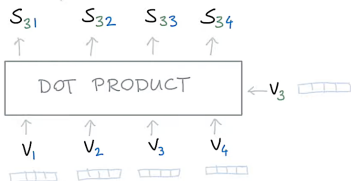
y_1 y_2 y_3 y_4

Self-attention (3-2)

Vector dot product

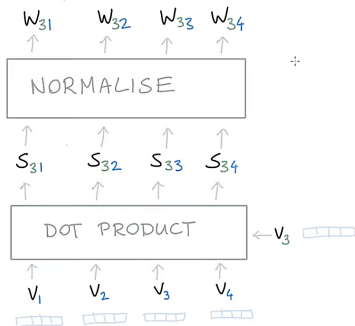


Self-attention (4)

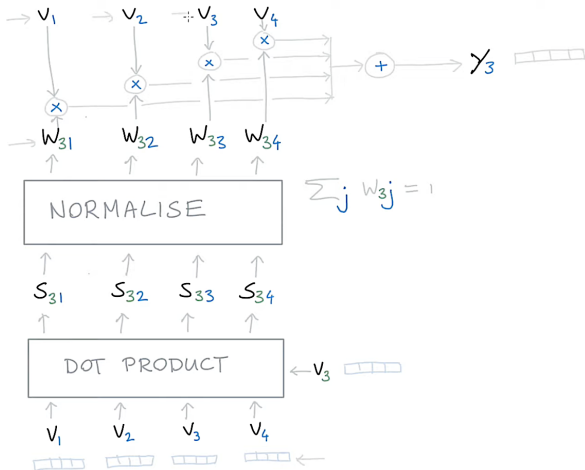


$$\begin{bmatrix} a & b & c \\ a & b & c \end{bmatrix}$$
$$S_{34} = aa + bb + cc$$

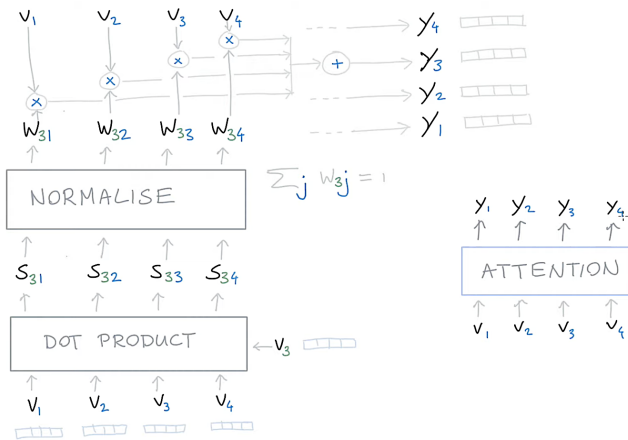
Self-attention (5)



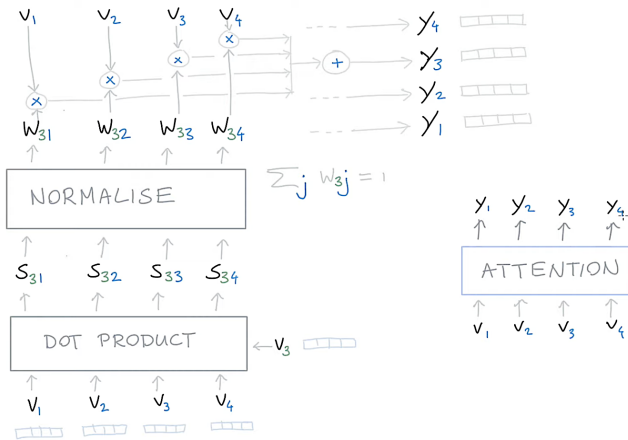
Self-attention (6)



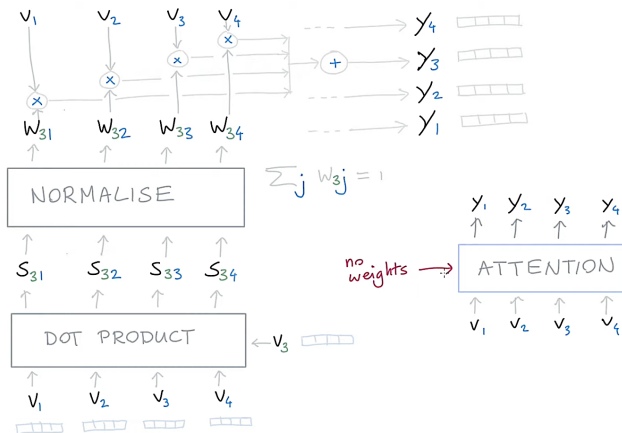
Self-attention (7)



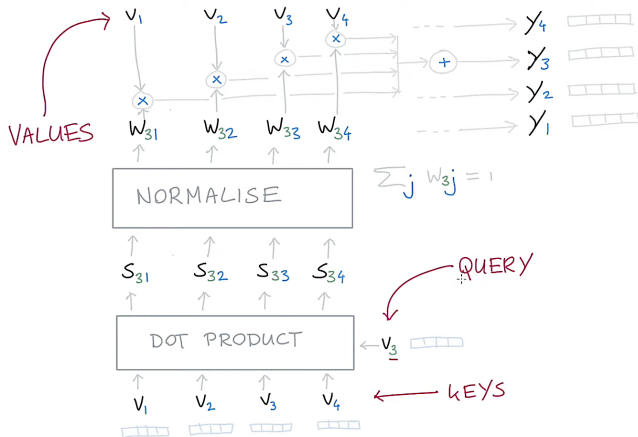
Self-attention (8)



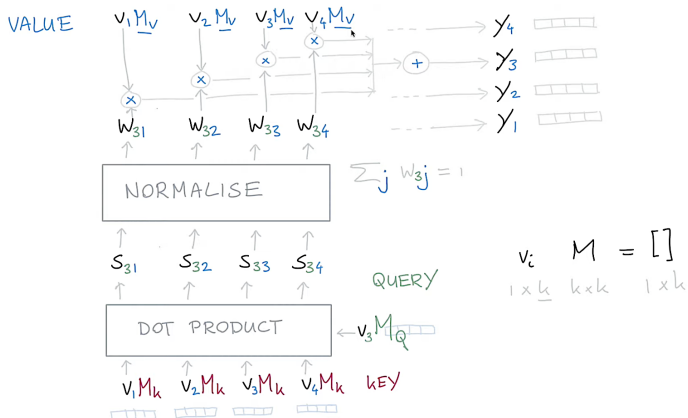
Self-attention (9)



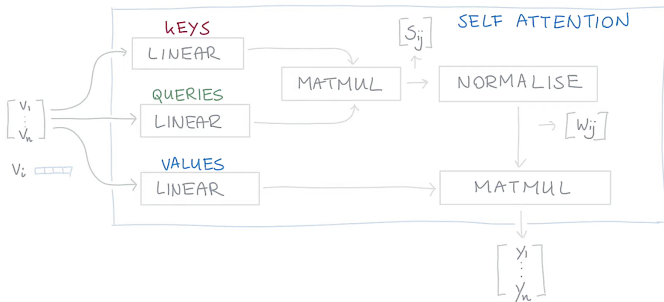
Self-attention (10)



Self-attention (11)



Self-attention (12)





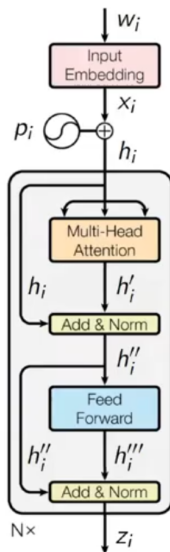
Трансформер для машинного перевода

Трансформер - это нейросетевая архитектура на основе моделей внимания и полносвязанных слоев, без RNN

Схема преобразований данных в машинном переводе:

- $S = (w_1, \dots, w_n)$ - слова предложения на входном языке
↓ *обучаемая или предобученная векторизация слов*
- $X = (x_1, \dots, x_n)$ - эмбединги слов входного предложения
↓ *трансформер-кодировщик*
- $Z = (z_1, \dots, z_n)$ - контекстные эмбединги слов
↓ *трансформер-декодировщик*
- $Y = (y_1, \dots, y_m)$ - эмбединги слов выходного предложения
↓ *генерация слов из построенной языковой модели*
- $\tilde{S} = (\tilde{w}_1, \dots, \tilde{w}_m)$ - слова предложения на выходном языке

Архитектура трансформера - кодировщика (1)





Архитектура трансформера - кодировщика (2)

- 1 Добавляются позиционные векторы p_i : $h_i = x_i + p_i$,
 $H = (h_1, \dots, h_n)$
- 2 Многомерное самовнимание: $h_i^j =$
 $\text{Attn}(W_q^j h_i, W_k^j H, W_v^j H)$
- 3 Конкатенация: $h_i' = \text{MH}_j(h_i^j) \equiv [h_i^{j1}, \dots, h_i^{jJ}]$
- 4 Сквозная связь + нормировка уровня: $h_i'' =$
 $\text{LN}(h_i' + h_i; \mu_1, \sigma_1)$
- 5 Полносвязная 2х-слойная сеть FFN:
 $h_i''' = W_2 \text{ReLU}(W_1 h_i'' + b_1) + b_2$
- 6 Сквозная связь + нормировка уровня: $z_i =$
 $\text{LN}(h_i''' + h_i''; \mu_2, \sigma_2)$



Замечания - 1

- вычисления параллельны по элементам последовательности $(x_1, \dots, x_n) \rightarrow (z_1, \dots, z_n)$, что было невозможно в RNN
- $N = 6$ блоков $h_i \rightarrow \square \rightarrow z_i$ соединяются последовательно
- возможно использование предобученных эмбедингов x_i



Замечания - 2

- возможно обучение эмбедингов $x_i \in \mathbb{R}^d$ слов $w_i \in V$:
 $x_i = u_{w_i}$ или в матричной записи $X_{d \times n} = U \cdot B_{d \times |V| \times n}$,
где
 - V - словарь слов входных последовательностей,
 - U - матрица обучаемых векторных представлений слов,
 - $b_{vi} = [w_i = v]$ - матрица one-hot кодирования
- нормировка уровня (Layer Normalization), $x, \mu, \sigma \in \mathbb{R}^d$:

$$LN_s(x; \mu, \sigma) = \sigma_s \frac{x_s - \bar{x}}{\sigma_x} + \mu_s, \quad s = 1, \dots, d,$$

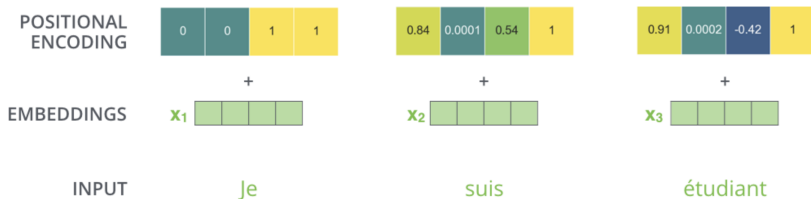
$$\bar{x} = \frac{1}{d} \sum_s x_s, \quad \sigma_x^2 = \frac{1}{d} \sum_s (x_s - \bar{x})^2 - \text{среднее и дис-я } x$$

Позиционное кодирование (1)

- Как учитывать порядок слов во входящем предложении?
- Позиционная информация кодируется векторами, по одному на каждую абсолютную позицию в обрабатываемом контексте
- Вектор позиции прибавляется к смысловому вектору токена (эмбеддингу) на входе модели, после чего результат подаётся в основную часть нейросети

Позиционное кодирование (2)

- Слово (эмбединг) $x_j \in \mathbb{R}^d$: $x_{p_j} = x_j + \hat{p}_j$



Позиционное кодирование (3)

- Сформировать вектор $\hat{p}_t \in \mathbb{R}_x^d$ для позиции t можно разными способами:
 - Учить такие векторы вместе с остальными частями сети: это увеличивает число параметров; модель не может обрабатывать последовательности с длиной, большей её контекста на обучении
 - Использование синусоидальных (тригонометрических) векторов. Элементы \hat{p}_t^i вектора определяются формулой:

$$\hat{p}_t^i = \begin{cases} \sin(w_k t), & i = 2k \\ \cos(w_k t) & i = 2k + 1 \end{cases}, \quad w_k = \frac{1}{10000^{2k/d_x}}$$

- k - номер вхождения \hat{p}_t^i



Позиционное кодирование (4)

- Идея позиционного кодирования состоит в том, чтобы различать разные позиции с помощью непрерывных систем.
- Здесь используем функции синуса и косинуса с разными частотами.

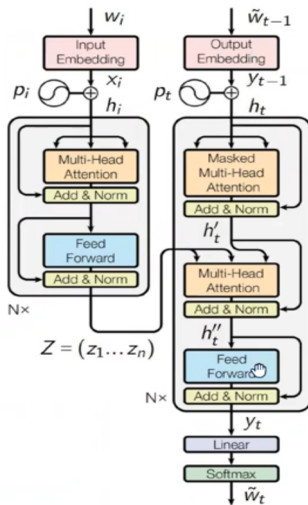
Позиционное кодирование (5)

- Относительные методы - кодируется не одиночный индекс, а пара позиций на разных расстояниях друг от друга. Относительное кодирование производится не путём сложения векторов на входе, а через модификацию подсчёта внимания.
- Предлагается для каждого расстояния $i - j$ обучать два вектора $a_{ij}^K, a_{ij}^V \in \mathbb{R}^d$ и использовать их в обновлённых формулах подсчёта self-attention:

$$e_{ij} = \frac{(x_i W^Q) (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}, \quad z_i = \sum_{j=1}^n \alpha_{ij} (x_{ij} W^V + a_{ij}^V).$$

- Информация о позиции вкладывается в векторы ключей и значений

Архитектура трансформера - декодировщика (1)



Архитектура трансформера - декодировщика (2)

Авторегрессионный синтез последовательности:

$y_0 = \langle \text{BOS} \rangle$ - эмбединг символа начала

- 1 Маскирование "данных из будущего" $h_t = y_{t-1} + p_t$,
 $H_t = (h_1, \dots, h_t)$
- 2 Многомерное самовнимание:
 $h'_i = LN \circ MH_j \circ \text{Attn}(W_q^j h_t, W_k^j H_t, W_v^j H_t)$
- 3 Многомерное внимание на кодировку Z :
 $h''_i = LN \circ MH_j \circ \text{Attn}(\tilde{W}_q^j h'_t, \tilde{W}_k^j Z, \tilde{W}_v^j Z)$
- 4 Полносвязная 2x-слойная сеть: $y_t = LN \circ FFN(h''_t)$
- 5 Линейный предсказательный слой:
 $p(\tilde{w}|t) = \text{SoftMax}(W_v y_t + b_v)$

Генерация $\tilde{w}_t = \arg \max_{\tilde{w}} p(\tilde{w}|t)$ пока $\tilde{w}_t \neq \langle \text{EOS} \rangle$

Критерии обучения и валидации для машинного перевода - 1

Критерий для обучения параметров нейронной сети W по обучающей выборке предложений S с переводом \tilde{S} :

$$\sum_{(S, \tilde{S})} \sum_{\tilde{w}_t \in \tilde{S}} \ln p(\tilde{w}_t | t, S, W) \rightarrow \max_W$$

Критерии обучения и валидации для машинного перевода - 2

Критерии оценивания моделей (недифференцируемые) по выборке пар предложений “перевод S , эталон S_0 ”:

- BiLingual Evaluation Understudy:

$$\text{BLEU} = \min \left(1, \frac{\sum \text{len}(S)}{\sum \text{len}(S_0)} \right) \times \text{mean}_{(S, S_0)} \left(\prod_{n=1}^4 \frac{\#n\text{-грамм из } S, \text{ входящих в } S_0}{\#n\text{-грамм в } S} \right)^{\frac{1}{4}}$$

- Word Error Rate:

$$\text{WER} = \text{mean}_{(S, S_0)} \left(\prod_{n=1}^4 \frac{\#вставок + \#удалений + \#замен}{\text{len}(S)} \right)$$

BERT (Bidirectional Encoder Representations from Transformer)

Трансформер BERT - это кодировщик без декодировщика, предобучаемый для решения широкого класса задач NLP

Схема преобразований данных:

- $S = (w_1, \dots, w_n)$ - токены предложения на входном языке
↓ *обучение эмбедингов вместе с трансформером*
- $X = (x_1, \dots, x_n)$ - эмбединги токенов входного предложения
↓ *трансформер кодировщика*
- $Z = (z_1, \dots, z_n)$ - трансформированные эмбединги
↓ *дообучение на конкретную задачу*
- Y - выходной текст/разметка/классификация и т.д.

Критерии MLM (masked language modeling) для обучения BERT

Критерий маскированного языкового моделирования MLM строится автоматически по текстам (self-supervised learning)

$$\sum_S \sum_{i \in M(S)} \ln p(w_i | i, S, W) \rightarrow \max_W$$

где $M(S)$ - подмножество маскированных токенов из S ,

$$p(w | i, S, W) = \text{SoftMax}_{w \in V} (W_z z_i(S, W_T) + b_z)$$

- языковая модель, предсказывающая i -ый токен предложения S ; $z_i(S, W_T)$ - контекстный эмбединг i -го токена предложения S на выходе Трансформера с параметрами W_T ; W - все параметры Трансформера и языковой модели.

Критерии NSP (next sentence prediction) для обучения BERT

Критерий предсказания связи между предложениями NSP строится автоматически по текстам (self-supervised learning):

$$\sum_{(S,S')} \ln p(y_{SS'} | S, S', W) \rightarrow \max_W$$

где $y_{SS'} = [\text{за } S \text{ следует } S']$ - классификация пары предложений,

$$p(y_{SS'} | S, S', W) = \text{SoftMax}_{y \in \{0,1\}}(W_y \text{th}(W_z z_0(S, S', W_T) + b_s) + b_y)$$

- вероятностная модель бинарной классификации пар (S, S') ; $z_0(S, S', W_T)$ - контекстный эмбединг токена $\langle CLS \rangle$ для пары предложений, записанной в виде $\langle CLS \rangle S \langle SEP \rangle S' \langle SEP \rangle$



Замечания по трансформерам

- **Fine-tuning:** для дообучения на задаче задается модель $f(Z(S, W_T), W_f)$, выборка $\{S\}$ и критерий $L(S, f) \rightarrow \max$
- **Multi-task learning:** для дообучения на наборе задач $\{t\}$ задаются модели $f(Z(S, W_T), W_t)$, выборки $\{S\}_t$ и сумма критериев $\sum_t \lambda_t \sum_S L_t(S, f_t) \rightarrow \max$
- GLUE, SuperGLUE, Russian SuperGLUE - наборы текстовых задач на понимание естественного языка
- Трансформеры обычно строятся не на словах, а на токенах, получаемых BPE (Byte-Pair Encoding) или WordPiece
- BERT_{BASE}, GPT1: $N = 12, d = 768, J = 8$ весов 65M
- BERT_{LARGE}: $N = 24, d = 1024, J = 16$ весов 340M



Еще замечания

- Модели внимания сначала встраивались в RNN или CNN, но оказалось, что они самодостаточны
- Модель внимания работает точнее и быстрее RNN
- Легко предобучается и используется для многих задач
- Легко обобщается на тексты, графы, изображения
- Доказано, что модель внимания multi-head self-attention (MHSA) эквивалентна сверточной сети [Cordonnier, 2020 On the relationship between self-attention and convolutional layers]
- Модель внимания лежит в основе трансформера

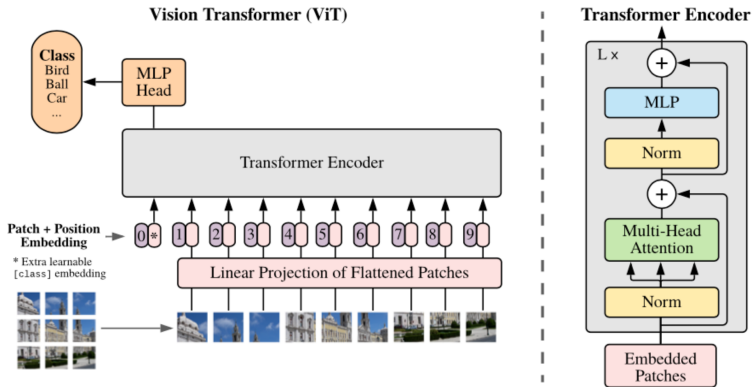


ViT

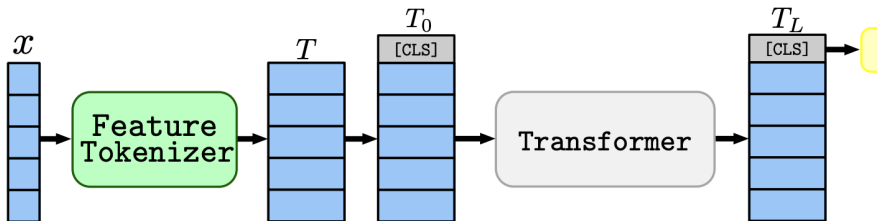
- Общая структура ViT состоит из шагов:
- Разделение изображения на фрагменты (фиксированных размеров).
- Выравнивание фрагментов изображения.
- Создание линейных эмбеддингов меньшей размерности из этих фрагментов.
- Включение позиционных эмбеддингов.
- подача последовательности на вход кодера трансформера.

- Обучение ViT на размеченных изображениях.
- Точная настройка для классификации изображений.

ViT

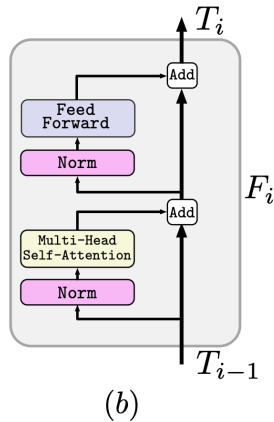
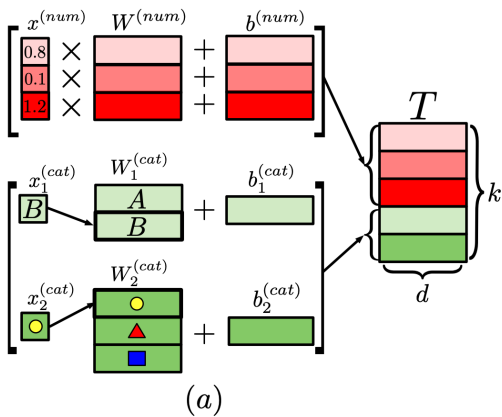


Tabular data (1)



Gorishniy, Y., Rubachev, I., Khruikov, V. and Babenko, A., 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34, pp.18932-18943.

Tabular data (2)



Tabular data (3)

- Модуль **tokenizer** преобразует входные объекты x в эмбединги $T \in \mathbf{R}^{k \times d}$
- Эмбединги для признака x_j вычисляется как:

$$T_j = b_j + f_j(x_j) \in \mathbf{R}^d, \quad f_j : \mathbf{X}_j \rightarrow \mathbf{R}^d$$

- b_j смещение j -го признака, $f_j^{(num)}$ реализуется как поэлементное умножение с $W_j^{(num)}$
- $f_j^{(cat)}$ реализуется как таблица $W_j^{(cat)} \in \mathbf{R}^{S_j \times d}$ для категориальных признаков
- В целом

$$T_j^{(num)} = b_j^{(num)} + x_j^{(num)} \cdot W_j^{(num)} \in \mathbf{R}^d$$

$$T_j^{(cat)} = b_j^{(cat)} + e_j^T W_j^{(cat)} \in \mathbf{R}^d$$

$$T = \text{stack} \left[T_1^{(num)}, \dots, T_{k^{(num)}}^{(num)}, T_1^{(cat)}, \dots, T_{k^{(cat)}}^{(cat)} \right] \in \mathbf{R}^{k \times d}$$

Tabular data (4)

- **Pre-processing** для регрессии:

$$y_{new} = \frac{y_{old} - \text{mean}(y_{train})}{\text{std}(y_{train})}$$

- **Transformer.** Эмбединг [CLS] токена (или “токен классификации”, или “classification token”, или “output token”) добавляется к T .
- L слоев трансформера F_1, \dots, F_L :

$$T_0 = \text{stack}([CLS], T), \quad T_i = F_i(T_{i-1})$$

- **Prediction.** Последнее представление CLS токена используется для предсказания:

$$\hat{y} = \text{Linear}(\text{ReLU}(\text{LayerNorm}(T_L^{[CLS]})))$$

Вопросы

?