СОДЕРЖАНИЕ

1 Трансформеры

Теперь, когда мы познакомились с механизмом внимания, напрашивается логичный вопрос: нужны ли нам вообще теперь LSTM? Как оказалось - нет. В своей фундаментальной работе ”Attention Is All You Need” [3], Vaswani et al. представили новую архитекутуру нейронной сети, **трансформер**, основанный исключительно на механизмах внимания, полностью отказываясь от рекуррентности и сверток.

С момента своей первой публикации, трансформер обошел большинство моделей глубого обучение, прежде считавшиееся лучшими в своей сфере. Нынче архитекутура трансформера считается лидирующей во многих областях глубого обучения и потому представляет для нас особый интерес.

1.1 Архитектура «с высока»

Ключевая проблема рекуррентных нейронных сетей, LSTMs и GRU считавшихся передовыми подходами для моделирования последовательностей на момент выхода статьи [3] кроется в их последовательной природе. Обобщая, они генерируют последовательность скрытых состояний h_t, как функцию предыдщего скрытых состояний h_{t-1} и входа для позиции t. Подобная последовательная природа исключает распараллеливание внутри обучающих примеров, что становится критичным при бОльших длинах последовательностей. Подобное ограничение и послужило вдохновением для трансформера - архитектуры модели, которая отказывается от рекуррентности и полностью полагается на механизм внимания для установления глобальных зависимостей между входом и выходом.

С высока, архитектура классического трансформера, представленного в ”Attention Is All You Need” [3] представлена на рис. 1.1.

Представленная архитектура адаптирована для модели кодировщика-декодера (см. главу todo), однако трансформер может состоять как только из левой части, т.е. кодировщика, так и правой, т.е. декодера, как мы это позже увидим в моделях BERT и GPT (см. главы todo и todo). Архитектура трансформера определяется общим механизмом, включающим в себя

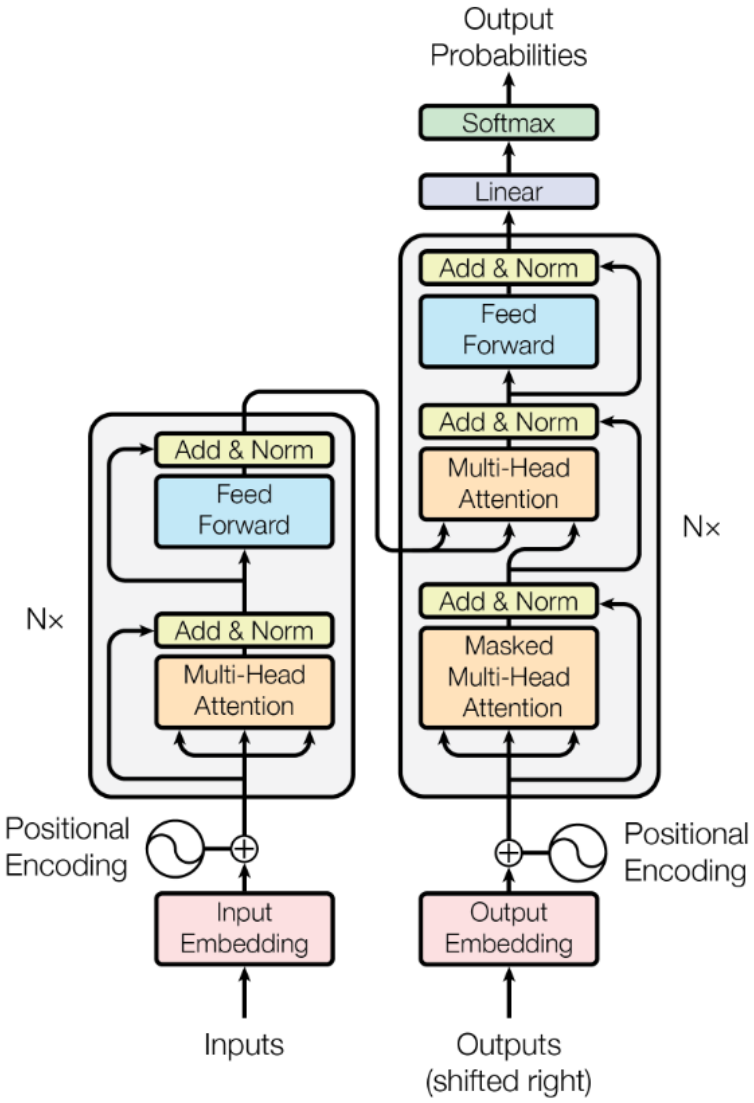

Рис. 1.1: Архитектура модели Трансформер. (Изначально, в своей статье "Attention Is All You Need модель трансформера была спроектирована для задачи машинного перевода, из-за чего на диаграмме изображены слои эмбеддингов, однако мы на них не будем акцентировать внимания. Нас сейчас больше интересует архитекутра самой модели, ведь поняв ее, мы сможем легко обобщить архитектуру трансформера для решения проблемы прогнозирования временных рядов).

внимание и прямое распространение (attention-plus-feed-forward).

1.2 Ключевые компоненты

В отличии от ранее расмотренных LSTM, трансформер имеет гораздо более структурированную архитектуру и можно представить, что мы собираем его из нескольких строительных блоков.

1.2.1 Внимание

Функцию внимания можно описать как отображение **запроса (query)** и набора **пар ключ-значение (key-value pairs)** в **выход (output)**, где запрос, ключи, значения и выход - векторы. Выход находится как взвешенная сумма значений, где вес, присваивающийся каждому значению вычисляется функцией совместимости запроса с соответствующим ключом [3].

Одним известным подходом к вычислению совместимости (сходства) между двумя значениями является **косинусово сходство (cosine similarity)**:

$$\cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{||\boldsymbol{A}||||\boldsymbol{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Однако, чаще сходство в контексте механизма внимания рассчитывают несколько иначе - берут только числитель косинусова сходства, т.е. скалярное произведение $\boldsymbol{A} \cdot \boldsymbol{B}$. Т.к. знаменатель всего навсего нормирует значения, чтобы они были в диапазоне от -1 до 1, что в данном случае нам не особо нужно.

В своей работе, Vaswani et al. рассчитывают внимание следующим образом:

$$\text{Attention}(Q,\, K,\, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V,$$

где Q, K, V - матрицы запросов, ключей и значений соответственно (результат также представляет из себя матрицу).

Здесь, мы как раз и видим, что сходство между запросом и ключом находится из скалярного произведения (плюс масштабирующий коэффициент $\frac{1}{\sqrt{d_k}}$, который на практике помагает численной устойчивости). Затем применяется функция sofmax для получения весов для значений.

В контексте модели кодировщика-декодера, обычно механизм внимания разделяют на два: **self-attention** и **cross-attention**:

- **self-attention**: значит, что мы рассчитываем сходство между элементами *внутри* последовательности (например при обработке последовательности слов в предложении, self-attention как бы закодирует контекст, т.е. то, как слова влияют друг на друга).

- **cross-attention**: значит, что мы рассчитываем сходство между эле-

ментами *между* последовательностями (например в задаче машинного перевода, cross-attention закодирует сходство между словами в предложении на одном языке и его переводе на другом языке).

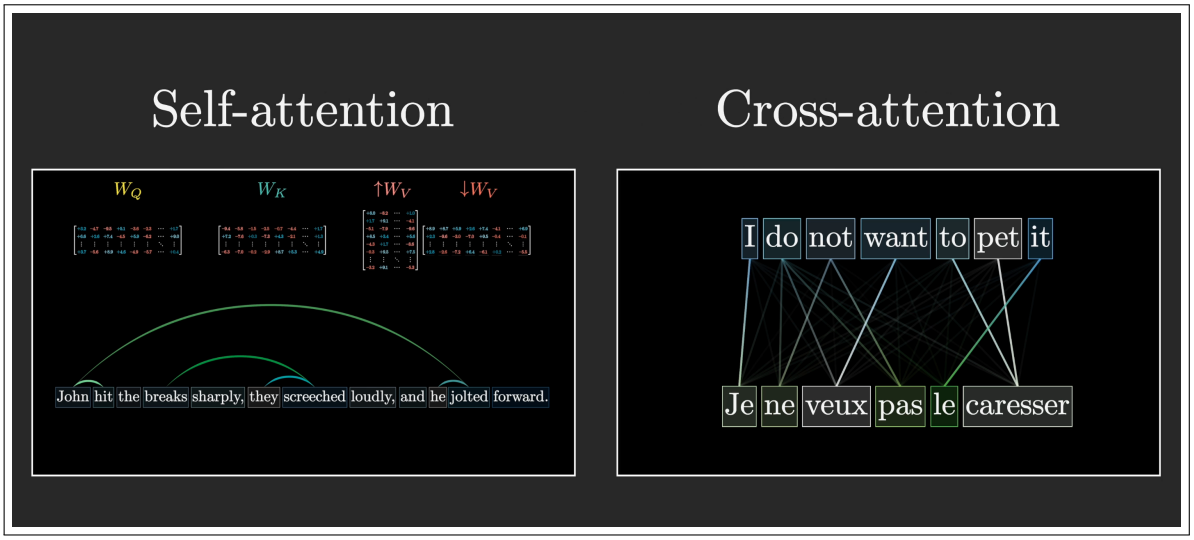

Рис. 1.2: Визуализация self-attention и cross-attention в контексе задачи NLP [1].

1.2.1.1 Multi-Head Внимание

В механзиме self-attention, описанном выше, всего одна матрица, которая описывает сходство между позициями в последовательности. Однако это может превратиться в bottleneck архитектуры - было бы полезно, чтобы пары точек были похожи в одних измерениях и отличались в других.

Оказалось выгодным параллельно рассчитывать сразу H наборов self-attention (H зовутся головами, от англ. Heads) и затем проецировать результаты в единственную матрицу, используемую в дальнейших вычислениях. Такого рода обобщение называется **multi-head self-attention** [2] (рис. 1.3).

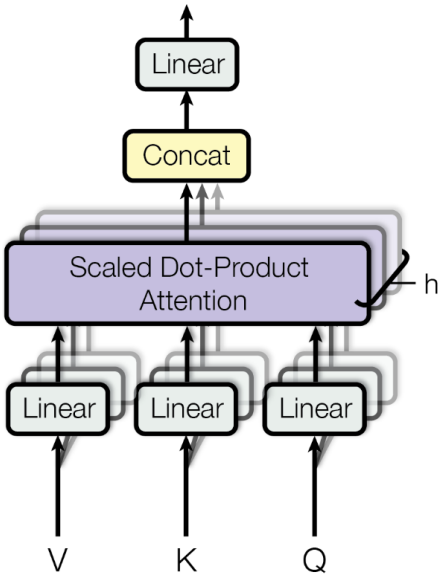

Рис. 1.3: Multi-Head Attention [3].

Multi-head attention позволяет модели совместно воспринимать информацию из разных подпространств представлений в разных позициях. При использовании одной головы внимания усреднение препятствует этому.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)\boldsymbol{W}^O,$$
$$\text{где head}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V),$$

где проекции - матрицы параметров $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

1.2.1.2 Авторегрессионное маскирование

Правая часть трансформера (рис. 1.1), представляющая из себя декодер также является моделью авторегрессии, тк от нее требуется предсказать следущее (в данном случае слово) w_n при заданных предыдщих словах $w_{1:n-1}$, т.е. вернуть $p(w_n = w|w_{1:n-1})$. Для успешного выполнения данной операции, нам потребуется немного модифицировать наш механизм self-attention. Ранее мы рассматривали self-attention как нечто, что собирает контекст для слова как из предыдщих слов в предложении, так и из последующих, однако в случае предсказания новых слов у нас нет последующих. Таким образом для корректной работы нашего механизма внимания от нас требуется выполнить операцию, известную как **маскирование (masking)**. Грубо говоря, мы зануляем веса в матрице, соответствующие

словам, стоящим после рассматриваемого (формально говоря мы присваиваем не 0, а $-\infty$, чтобы после применения операции softmax получить валидное распределение).

1.2.2 Позиционное кодирование

Поскольку наша модель не содержит ни рекуррентности, ни свертки, для того чтобы модель могла пользоваться информацией о порядке последовательности, мы должны добавить информацию об относительном или абсолютном положении элементов в последовательности. Данного эффекта можно добиться с помощью **позиционного кодирования (positional encoding)**. Существует множество различных видов позиционного кодирования, один из них заключается в том, чтобы напрямую включить информацию о положении элемента в последовательности в его векторное представление. Информация о положении элемента может быть как фиксированной, например добавление вектора синусоид различных частот и фаз, чтобы закодировать положение слова в предложении (подобный подход как раз и применяется в работе ”Attention Is All You Need” [3]), или она может быть свободным обучаемым параметром, как это обычно делается в трансформерах для изображений. Также существуют подходы для включения информации об относительном расстояним между парами элементов, модифицируя механизм self-attention, что наблюдается в эквивариантных трансформерах [2].

1.2.3 Feed-Forward & Layer Norm & Residual connections

Feed Forward Layers. Помимо всего рассмотренного, модель трансформера, представленная Vaswani et al. [3], также включает в себя, так называемые, **слои прямого распространения (Feed Forward Layers)**, которые представляют из себя просто многослойные перцептроны (см. главу todo), состоящие из двух линейных преобразований с функцией активации ReLU между ними:

$$\text{FFN}(x) = \max(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2$$

Layer Norm. Плюс, для стабилизации обучения применяется нор-

мировка. Опять же, существует множество различных выборов для расчета нормировки, но стандартным подходом считается **LayerNorm**, который нормализует каждый токен по-отдельности, вычитая среднее и деля на среднеквадратическое отклонение (что также известно как стандартизация),

$$\bar{\boldsymbol{x}}_{d,n} = \frac{1}{\sqrt{\mathbb{V}[\boldsymbol{x}_n]}}(\boldsymbol{x}_{d,n} - \text{mean}(\boldsymbol{x}_n))\gamma_d + \beta_d = \text{LayerNorm}(\boldsymbol{X})_{d,n},$$

где $\text{mean}(\boldsymbol{x}_n) = \frac{1}{D}\sum_{d=1}^{D} \boldsymbol{x}_{d,n}$ и $\mathbb{V}(\boldsymbol{x}_n) = \frac{1}{D}\sum_{d=1}^{D}(x_{d,n} - \text{mean}(\boldsymbol{x}_n))^2$. Параметры γ_d и β_d - обучаемые параметры масштаба и сдвига.

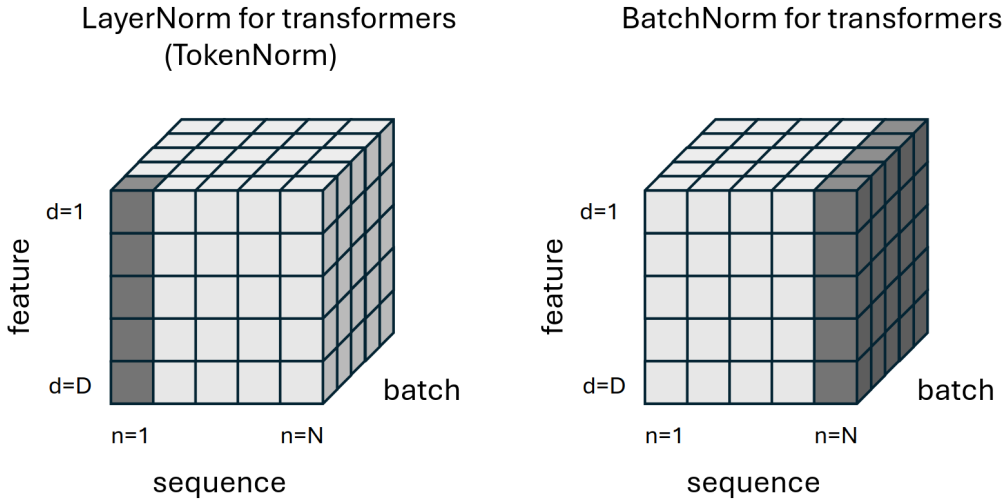

Рис. 1.4: Трансформеры производят layer normalisation (слева), которая нормализует среднее и среднеквадратическое отклонение каждого индивидуального токена в каждой последовательности в бэтче. В свою очередь, batch normalisation (справа), которая нормализует по размерности признака *и* бэтча вместе, на практике показывает себя куда менее стабильной. Разумеется существуют и другие разновидности нормализации. (На данном рисунке автор, во избежание путаницы называет LayerNorm, в применении к трансформерам, TokenNorm, т.к. LayerNorm, в применении к CNN действует иначе) [2].

Residual connections. В машинном обучении широко используют, так называемые, **остаточные соединения (residual connections)**, т.к. они упрощают инициализацию имеют разумный индуктивный уклон в сторону простых функций и стабилизируют обучение. Вместо того чтобы напрямую указывать функцию $x^{(m)} = f_\theta(x^{(m-1)})$, идея заключается в том, чтобы параметризовать ее как тождественное преобразование и остаточный член:

$$x^{(m)} = x^{(m-1)} + \text{res}_\theta(x^{(m-1)}).$$

Равнозначно это можно рассматривать как моделирование разностей между представлениями $x^{(m)} - x^{(m-1)} = \text{res}_\theta(x^{(m-1)})$ и оно будет хорошо работать когда моделируемая функция близка к тождественной [2]. (Схематично, residual connections представлены на рис. 1.1 в виде, огибающих блоки, стрелочек, ведущих изначальные (до преобразования) величины в блок сложения и нормировки (с преобразованными этими же величинами)).

1.3 Преимущества при моделировании временных рядов

У трансформеров есть ряд преимуществ при моделировании временных рядов.

- **Распараллеливание.** Благодаря своей природе, трансформеры очень хорошо поддаются распараллеливанию (в отличии от рекуррентных нейронных сетей), из-за чего мы можем обрабатывать гораздо бОльшие объемы данных за то же время.

- **Долгосрочные зависимости.** Одна из ключевых проблем классических рекуррентных нейронных сетей заключается в запоминании долгосроных последовательностей из-за проблемы затухающего градиента. Хорошей попыткой решения данной проблемы послужили LSTM, которые сильно продвинули RNN вперед, однако и у тех, при обработке достаточно длинных последовательностей, возникают проблемы с долгосрочными зависиимостями. Механизм внимания полность решает данную проблемы, т.к. в силу отсутствия рекуррентных связей допускает, вообще говоря, бесконечно долгие зависимости.

- **Гибкость.** Трансформеры естественным образом обобщаются на многомерные временные ряды, а также данные с нерегулярной частотой измерений или пропущенными значениями, что делает их универсальным инструментом для анализа временных данных.

1.4 Modifications

1.4.1 BERT

1.4.2 GPT

1.4.3 TFT

1.4.4 Informer

1.5 State-of-the-art transformers for time series forecasting

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Grant Sanderson. Attention in transformers, step-by-step | dl6. https://www.youtube.com/watch?v=eMlx5fFNoYc&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=7, 2023. YouTube video, 3Blue1Brown.

[2] Richard E. Turner. An introduction to transformers. Lecture or Technical Report, 2023. Department of Engineering, University of Cambridge and Microsoft Research, Cambridge, UK.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems (NeurIPS), volume 30, 2017.