Внимание - это всё, что нужно

Авторы: Ашиш Васвани*, Ноам Шазир*, Ники Пармар*, Якоб Ушкорайт*, Ллион Джонс*, Эйдан Н. Гомес*†, Лукаш Кайзер*, Иллия Полосухин*‡

*Google Brain, Google Research, †Университет Торонто

*Равный вклад. Порядок перечисления случайный.

Аннотация

Доминирующие модели преобразования последовательностей основаны на сложных рекуррентных или свёрточных нейронных сетях, включающих кодировщик и декодировщик. Лучшие модели также соединяют кодировщик и декодировщик через механизм внимания. Мы предлагаем новую простую сетевую архитектуру - Трансформер, основанную исключительно на механизмах внимания и полностью отказывающуюся от рекуррентности и свёрток. Эксперименты на двух задачах машинного перевода показывают превосходное качество этих моделей при большей параллелизуемости и значительно меньшем времени обучения. Наша модель достигает 28.4 BLEU на задаче перевода WMT 2014 с английского на немецкий, улучшая существующие лучшие результаты, включая ансамбли, более чем на 2 BLEU. На задаче перевода WMT 2014 с английского на французский наша модель устанавливает новый рекорд BLEU для одиночной модели - 41.8 после обучения в течение 3.5 дней на восьми GPU, что составляет малую долю стоимости обучения лучших моделей из литературы. Мы показываем, что Трансформер хорошо обобщается на другие задачи, успешно применив его к синтаксическому разбору английского языка как с большими, так и с ограниченными обучающими данными.

---

1. Введение

Рекуррентные нейронные сети, в частности сети с долгой краткосрочной памятью (LSTM) и управляемые рекуррентные нейронные сети, прочно утвердились как современные подходы в моделировании последовательностей и задачах преобразования, таких как языковое моделирование и машинный перевод. Многочисленные усилия продолжают раздвигать границы рекуррентных языковых моделей и архитектур кодировщик-декодировщик.

Рекуррентные модели обычно выполняют вычисления вдоль символьных позиций входных и выходных последовательностей. Выравнивая позиции по шагам вычислительного времени, они генерируют последовательность скрытых состояний h_t как функцию предыдущего скрытого состояния h_(t-1) и входа для позиции t. Эта внутренне последовательная природа исключает параллелизацию в пределах обучающих примеров, что становится критичным при больших длинах последовательностей, поскольку ограничения памяти лимитируют пакетную обработку примеров. Недавние работы достигли значительных улучшений в вычислительной эффективности за счёт трюков факторизации и условных

вычислений, одновременно улучшая производительность модели в последнем случае. Однако фундаментальное ограничение последовательных вычислений остаётся.

Механизмы внимания стали неотъемлемой частью привлекательных моделей моделирования последовательностей и преобразования в различных задачах, позволяя моделировать зависимости независимо от их расстояния во входных или выходных последовательностях. Однако за редким исключением такие механизмы внимания используются в сочетании с рекуррентной сетью.

В этой работе мы предлагаем Трансформер - архитектуру модели, избегающую рекуррентности и вместо этого полностью полагающуюся на механизм внимания для извлечения глобальных зависимостей между входом и выходом. Трансформер позволяет значительно большую параллелизацию и может достичь нового уровня качества перевода после обучения всего лишь двенадцать часов на восьми GPU P100.

---

2. Предпосылки

Цель сокращения последовательных вычислений также лежит в основе Extended Neural GPU, ByteNet и ConvS2S, все из которых используют свёрточные нейронные сети в качестве базового строительного блока, вычисляя скрытые представления параллельно для всех входных и выходных позиций. В этих моделях количество операций, требуемых для связывания сигналов из двух произвольных входных или выходных позиций, растёт с расстоянием между позициями - линейно для ConvS2S и логарифмически для ByteNet. Это затрудняет изучение зависимостей между удалёнными позициями. В Трансформере это сводится к постоянному числу операций, хотя и за счёт снижения эффективного разрешения из-за усреднения взвешенных по вниманию позиций - эффект, который мы компенсируем Многоголовым Вниманием, описанным в разделе 3.2.

Самовнимание, иногда называемое интра-вниманием, является механизмом внимания, связывающим различные позиции одной последовательности для вычисления представления последовательности. Самовнимание успешно использовалось в различных задачах, включая понимание прочитанного, абстрактное суммирование, текстовую импликацию и изучение независимых от задачи представлений предложений.

Сквозные сети памяти основаны на рекуррентном механизме внимания вместо последовательно выровненной рекуррентности и показали хорошие результаты на простых языковых задачах ответов на вопросы и языкового моделирования.

Насколько нам известно, однако, Трансформер является первой моделью преобразования, полностью полагающейся на самовнимание для вычисления представлений своего входа и выхода без использования последовательно выровненных RNN или свёрток. В следующих разделах мы опишем Трансформер, обоснуем самовнимание и обсудим его преимущества над такими моделями.

---

3. Архитектура модели

Большинство конкурентоспособных нейронных моделей преобразования последовательностей имеют структуру кодировщик-декодировщик. Здесь кодировщик отображает входную последовательность символьных представлений (x_1, ..., x_n) в последовательность непрерывных представлений z = (z_1, ..., z_n). По заданному z декодировщик затем генерирует выходную последовательность (y_1, ..., y_m) символов по одному элементу за раз. На каждом шаге модель является авторегрессивной, использующей ранее сгенерированные символы в качестве дополнительного входа при генерации следующего.

![Рисунок 1: Трансформер - архитектура модели](Примечание: оригинальная схема архитектуры)

Трансформер следует этой общей архитектуре, используя наложенные слои самовнимания и поточечные, полносвязные слои как для кодировщика, так и для декодировщика, показанные в левой и правой половинах Рисунка 1 соответственно.

3.1 Стеки кодировщика и декодировщика

Кодировщик: Кодировщик состоит из стека из N = 6 идентичных слоёв. Каждый слой имеет два подслоя. Первый - это механизм многоголового самовнимания, а второй - простая поточечная полносвязная прямая сеть. Мы применяем остаточное соединение вокруг каждого из двух подслоёв с последующей нормализацией слоя. То есть выход каждого подслоя - LayerNorm(x + Sublayer(x)), где Sublayer(x) - функция, реализуемая самим подслоем. Чтобы облегчить эти остаточные соединения, все подслои в модели, а также слои эмбеддинга, производят выходы размерности d_model = 512.

Декодировщик: Декодировщик также состоит из стека из N = 6 идентичных слоёв. В дополнение к двум подслоям в каждом слое кодировщика, декодировщик вставляет третий подслой, который выполняет многоголовое внимание к выходу стека кодировщика. Подобно кодировщику, мы применяем остаточные соединения вокруг каждого из подслоёв с последующей нормализацией слоя. Мы также модифицируем подслой самовнимания в стеке декодировщика, чтобы предотвратить обращение позиций к последующим позициям. Это маскирование в сочетании с тем фактом, что выходные эмбеддинги сдвинуты на одну позицию, гарантирует, что предсказания для позиции i могут зависеть только от известных выходов в позициях меньше i.

3.2 Внимание

Функцию внимания можно описать как отображение запроса и набора пар ключ-значение в выход, где запрос, ключи, значения и выход - все векторы. Выход вычисляется как взвешенная сумма значений, где вес, присваиваемый каждому значению, вычисляется функцией совместимости запроса с соответствующим ключом.

3.2.1 Масштабированное скалярное внимание

Мы называем наше конкретное внимание "Масштабированным скалярным вниманием" (Рисунок 2). Вход состоит из запросов и ключей размерности d_k и значений размерности d_v. Мы вычисляем

скалярные произведения запроса со всеми ключами, делим каждое на √d_k и применяем функцию softmax для получения весов на значениях.

На практике мы вычисляем функцию внимания для набора запросов одновременно, упакованных вместе в матрицу Q. Ключи и значения также упакованы в матрицы K и V. Мы вычисляем матрицу выходов как:

Attention(Q, K, V) = softmax(QK^T / √d_k)V

Две наиболее часто используемые функции внимания - это аддитивное внимание и скалярное (мультипликативное) внимание. Скалярное внимание идентично нашему алгоритму, за исключением масштабирующего фактора 1/√d_k. Аддитивное внимание вычисляет функцию совместимости, используя прямую сеть с одним скрытым слоем. Хотя оба похожи по теоретической сложности, скалярное внимание намного быстрее и более эффективно по памяти на практике, поскольку может быть реализовано с использованием высоко оптимизированного кода матричного умножения.

Для малых значений d_k оба механизма работают одинаково, аддитивное внимание превосходит скалярное внимание без масштабирования для больших значений d_k. Мы подозреваем, что для больших значений d_k скалярные произведения растут по величине, проталкивая функцию softmax в области, где она имеет чрезвычайно малые градиенты. Чтобы противодействовать этому эффекту, мы масштабируем скалярные произведения на 1/√d_k.

3.2.2 Многоголовое внимание

Вместо выполнения одной функции внимания с d_model-мерными ключами, значениями и запросами, мы обнаружили, что полезно линейно проецировать запросы, ключи и значения h раз с различными изученными линейными проекциями в размерности d_k, d_k и d_v соответственно. На каждой из этих спроецированных версий запросов, ключей и значений мы затем выполняем функцию внимания параллельно, получая d_v-мерные выходные значения. Они конкатенируются и снова проецируются, в результате получаются финальные значения, как показано на Рисунке 2.

Многоголовое внимание позволяет модели совместно обращать внимание на информацию из различных подпространств представлений в различных позициях. С одной головой внимания усреднение подавляет это.

MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O

где **head_i = Attention(QW^Q_i, KW^K_i, VW^V_i)**

В этой работе мы используем h = 8 параллельных слоёв внимания или голов. Для каждой из них мы используем d_k = d_v = d_model/h = 64. Из-за уменьшенной размерности каждой головы общая вычислительная стоимость подобна стоимости одноголового внимания с полной размерностью.

3.2.3 Применения внимания в нашей модели

Трансформер использует многоголовое внимание тремя различными способами:

- В слоях "внимания кодировщик-декодировщик" запросы приходят из предыдущего слоя декодировщика, а ключи памяти и значения приходят из выхода кодировщика. Это позволяет каждой позиции в декодировщике обращать внимание на все позиции во входной последовательности. Это имитирует типичные механизмы внимания кодировщик-декодировщик в моделях последовательность-к-последовательности.
- Кодировщик содержит слои самовнимания. В слое самовнимания все ключи, значения и запросы приходят из одного места, в данном случае из выхода предыдущего слоя в кодировщике. Каждая позиция в кодировщике может обращать внимание на все позиции в предыдущем слое кодировщика.
- Аналогично, слои самовнимания в декодировщике позволяют каждой позиции в декодировщике обращать внимание на все позиции в декодировщике вплоть до этой позиции включительно. Нам нужно предотвратить поток информации влево в декодировщике для сохранения авторегрессивного свойства. Мы реализуем это внутри масштабированного скалярного внимания, маскируя (устанавливая в $-\infty$) все значения на входе softmax, которые соответствуют недопустимым соединениям.

3.3 Поточечные прямые сети

В дополнение к подслоям внимания, каждый из слоёв в нашем кодировщике и декодировщике содержит полносвязную прямую сеть, которая применяется к каждой позиции отдельно и идентично. Она состоит из двух линейных преобразований с активацией ReLU между ними.

FFN(x) = max(0, xW_1 + b_1)W_2 + b_2

Хотя линейные преобразования одинаковы в различных позициях, они используют разные параметры от слоя к слою. Другой способ описания этого - две свёртки с размером ядра 1. Размерность входа и выхода - d_model = 512, а внутренний слой имеет размерность d_ff = 2048.

3.4 Эмбеддинги и Softmax

Подобно другим моделям преобразования последовательностей, мы используем изученные эмбеддинги для преобразования входных токенов и выходных токенов в векторы размерности d_model. Мы также используем обычное изученное линейное преобразование и функцию softmax для преобразования выхода декодировщика в вероятности предсказания следующего токена. В нашей модели мы разделяем одну и ту же весовую матрицу между двумя слоями эмбеддинга и предварительным softmax линейным преобразованием. В слоях эмбеддинга мы умножаем эти веса на $\sqrt{d\_model}$.

3.5 Позиционное кодирование

Поскольку наша модель не содержит рекуррентности и свёрток, для того чтобы модель могла использовать порядок последовательности, мы должны внедрить некоторую информацию об относительной или абсолютной позиции токенов в последовательности. Для этого мы добавляем "позиционные кодирования" к входным эмбеддингам в нижних частях стеков кодировщика и декодировщика. Позиционные кодирования имеют ту же размерность d_model, что и эмбеддинги, так

что эти два могут суммироваться. Существует много вариантов позиционных кодирований - изученные и фиксированные.

В этой работе мы используем синусные и косинусные функции различных частот:

PE_(pos,2i) = sin(pos/10000^(2i/d_model)) PE_(pos,2i+1) = cos(pos/10000^(2i/d_model))

где pos - позиция, а i - размерность. То есть каждая размерность позиционного кодирования соответствует синусоиде. Длины волн образуют геометрическую прогрессию от 2π до 10000·2π. Мы выбрали эту функцию, потому что предположили, что она позволит модели легко научиться обращать внимание по относительным позициям, поскольку для любого фиксированного смещения k, PE_(pos+k) может быть представлено как линейная функция от PE_pos.

Мы также экспериментировали с использованием изученных позиционных эмбеддингов и обнаружили, что две версии дают почти идентичные результаты (см. Таблицу 3, строка (E)). Мы выбрали синусоидальную версию, потому что она может позволить модели экстраполировать на длины последовательностей, превышающие встреченные во время обучения.

---

4. Почему самовнимание

В этом разделе мы сравниваем различные аспекты слоёв самовнимания с рекуррентными и свёрточными слоями, обычно используемыми для отображения одной последовательности переменной длины символьных представлений (x_1, ..., x_n) в другую последовательность равной длины (z_1, ..., z_n), где x_i, z_i ∈ R^d, например, скрытый слой в типичном кодировщике или декодировщике преобразования последовательностей. Обосновывая наше использование самовнимания, мы рассматриваем три требования.

Первое - общая вычислительная сложность на слой. Второе - объём вычислений, который может быть распараллелен, измеряемый минимальным числом последовательных операций, требуемых.

Третье - длина пути между зависимостями дальнего действия в сети. Изучение зависимостей дальнего действия является ключевым вызовом во многих задачах преобразования последовательностей. Один ключевой фактор, влияющий на способность изучать такие зависимости - длина путей, которые должны пройти прямые и обратные сигналы в сети. Чем короче эти пути между любой комбинацией позиций во входных и выходных последовательностях, тем легче изучать зависимости дальнего действия. Следовательно, мы также сравниваем максимальную длину пути между любыми двумя входными и выходными позициями в сетях, состоящих из различных типов слоёв.

Таблица 1: Максимальные длины путей, сложность на слой и минимальное число последовательных операций

Тип слоя	Сложность на слой	Последовательные операции	Максимальная длина пути
Самовнимание	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Рекуррентный	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Свёрточный	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Самовнимание (ограниченное)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Как отмечено в Таблице 1, слой самовнимания соединяет все позиции постоянным числом последовательно выполняемых операций, тогда как рекуррентный слой требует O(n) последовательных операций. С точки зрения вычислительной сложности, слои самовнимания быстрее рекуррентных слоёв, когда длина последовательности n меньше размерности представления d, что чаще всего имеет место с представлениями предложений, используемыми современными моделями в машинном переводе.

Одиночный свёрточный слой с шириной ядра $k < n$ не соединяет все пары входных и выходных позиций. Для этого требуется стек из O(n/k) свёрточных слоёв в случае смежных ядер или $O(\log_k(n))$ в случае расширенных свёрток, увеличивая длину самых длинных путей между любыми двумя позициями в сети. Свёрточные слои обычно более дорогостоящи, чем рекуррентные слои, в k раз. Однако разделимые свёртки значительно снижают сложность. Даже при $k = n$, однако, сложность разделимой свёртки равна комбинации слоя самовнимания и поточечного прямого слоя - подход, который мы используем в нашей модели.

Как дополнительное преимущество, самовнимание может давать более интерпретируемые модели. Мы исследуем распределения внимания из наших моделей и представляем и обсуждаем примеры в приложении. Не только отдельные головы внимания явно учатся выполнять различные задачи, многие, по-видимому, демонстрируют поведение, связанное с синтаксической и семантической структурой предложений.

---

5. Обучение

Этот раздел описывает режим обучения для наших моделей.

5.1 Обучающие данные и пакетирование

Мы обучались на стандартном наборе данных WMT 2014 английский-немецкий, состоящем примерно из 4.5 миллионов пар предложений. Предложения были закодированы с использованием кодирования парами байтов, которое имеет общий словарь источник-цель примерно из 37000 токенов. Для английского-французского мы использовали значительно больший набор данных WMT 2014 английский-французский, состоящий из 36 миллионов предложений, и разделили токены на словарь из 32000 словочастей. Пары предложений были сгруппированы в пакеты по приблизительной длине

последовательности. Каждый обучающий пакет содержал набор пар предложений, содержащих приблизительно 25000 исходных токенов и 25000 целевых токенов.

5.2 Аппаратное обеспечение и расписание

Мы обучали наши модели на одной машине с 8 GPU NVIDIA P100. Для наших базовых моделей, использующих гиперпараметры, описанные в статье, каждый обучающий шаг занимал около 0.4 секунды. Мы обучали базовые модели в течение 100,000 шагов или 12 часов. Для наших больших моделей время шага составляло 1.0 секунду. Большие модели обучались в течение 300,000 шагов (3.5 дня).

5.3 Оптимизатор

Мы использовали оптимизатор Adam с $\beta_1 = 0.9$, $\beta_2 = 0.98$ и $\epsilon = 10^{-9}$. Мы варьировали скорость обучения в ходе обучения согласно формуле:

$$lrate = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

Это соответствует линейному увеличению скорости обучения для первых warmup_steps обучающих шагов и её уменьшению после этого пропорционально обратному квадратному корню из номера шага. Мы использовали warmup_steps = 4000.

5.4 Регуляризация

Мы применяем три типа регуляризации во время обучения:

Остаточный Dropout: Мы применяем dropout к выходу каждого подслоя, перед тем как он добавляется к входу подслоя и нормализуется. Кроме того, мы применяем dropout к суммам эмбеддингов и позиционных кодирований как в стеках кодировщика, так и декодировщика. Для базовой модели мы используем скорость $P_{drop} = 0.1$.

Сглаживание меток: Во время обучения мы применяли сглаживание меток со значением $\epsilon_{ls} = 0.1$. Это ухудшает перплексию, поскольку модель учится быть более неуверенной, но улучшает точность и оценку BLEU.

---

6. Результаты

6.1 Машинный перевод

На задаче перевода WMT 2014 с английского на немецкий большая модель трансформер (Transformer (big) в Таблице 2) превосходит лучшие ранее опубликованные модели (включая ансамбли) более чем на 2.0 BLEU, устанавливая новый рекорд BLEU 28.4. Конфигурация этой модели перечислена в нижней строке Таблицы 3. Обучение заняло 3.5 дня на 8 GPU P100. Даже наша базовая модель превосходит все ранее опубликованные модели и ансамбли при малой доле стоимости обучения любой из конкурентных моделей.

На задаче перевода WMT 2014 с английского на французский наша большая модель достигает оценки BLEU 41.0, превосходя все ранее опубликованные одиночные модели при менее чем 1/4 стоимости обучения предыдущей современной модели. Модель Transformer (big), обученная для английского-французского, использовала скорость dropout P_drop = 0.1 вместо 0.3.

Для базовых моделей мы использовали одиночную модель, полученную усреднением последних 5 контрольных точек, которые записывались с 10-минутными интервалами. Для больших моделей мы усредняли последние 20 контрольных точек. Мы использовали поиск по лучу с размером луча 4 и штрафом за длину $\alpha = 0.6$. Эти гиперпараметры были выбраны после экспериментирования на наборе для разработки. Мы устанавливали максимальную длину выхода во время вывода как длина входа + 50, но завершали досрочно, когда возможно.

Таблица 2: Transformer достигает лучших оценок BLEU, чем предыдущие современные модели

Модель	**BLEU**	**Стоимость обучения (FLOPs)**
	EN-DE	EN-FR
ByteNet	23.75	-
Deep-Att + PosUnk	-	39.2
GNMT + RL	24.6	39.92
ConvS2S	25.16	40.46
MoE	26.03	40.56
Deep-Att + PosUnk Ensemble	-	40.4
GNMT + RL Ensemble	26.30	41.16
ConvS2S Ensemble	26.36	41.29
Transformer (базовая)	**27.3**	**38.1**
Transformer (большая)	**28.4**	**41.8**

Таблица 2 суммирует наши результаты и сравнивает качество нашего перевода и стоимость обучения с другими архитектурами моделей из литературы. Мы оцениваем количество операций с плавающей точкой, используемых для обучения модели, умножая время обучения, количество использованных GPU и оценку устойчивой производительности с плавающей точкой одинарной точности каждого GPU.

6.2 Вариации модели

Чтобы оценить важность различных компонентов Трансформера, мы варьировали нашу базовую модель различными способами, измеряя изменение производительности на переводе с английского на немецкий на наборе для разработки newstest2013. Мы использовали поиск по лучу, как описано в предыдущем разделе, но без усреднения контрольных точек. Мы представляем эти результаты в Таблице 3.

Таблица 3: Вариации архитектуры Transformer

	N	d_model	d_ff	h	d_k	d_v	P_drop	ϵ_ls	шагов	PPL	BLEU	параметры
база	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65M
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58M
					32					5.01	25.4	60M
(C)	2									6.11	23.7	36M
	4									5.19	25.3	50M
	8									4.88	25.5	80M
		256			32	32				5.75	24.5	28M
		1024			128	128				4.66	26.0	168M
			1024							5.12	25.4	53M
			4096							4.75	26.2	90M
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	позиционный эмбеддинг вместо синусоид									4.92	25.7	
большая	6	1024	4096	16			0.3		300K	4.33	26.4	213M

В строках (A) Таблицы 3 мы варьируем количество голов внимания и размерности ключа и значения внимания, сохраняя объём вычислений постоянным, как описано в Разделе 3.2.2. Хотя одноголовое внимание на 0.9 BLEU хуже лучшей настройки, качество также падает при слишком большом количестве голов.

В строках (B) Таблицы 3 мы наблюдаем, что уменьшение размера ключа внимания d_k вредит качеству модели. Это предполагает, что определение совместимости не является простым и что более сложная функция совместимости, чем скалярное произведение, может быть полезной. Мы далее наблюдаем в строках (C) и (D), что, как ожидалось, большие модели лучше, и dropout очень помогает избежать переобучения. В строке (E) мы заменяем наше синусоидальное позиционное кодирование на изученные позиционные эмбеддинги и наблюдаем почти идентичные результаты с базовой моделью.

6.3 Синтаксический разбор английского языка

Чтобы оценить, может ли Трансформер обобщаться на другие задачи, мы провели эксперименты на синтаксическом разборе английского языка. Эта задача представляет специфические вызовы: выход подчинён сильным структурным ограничениям и значительно длиннее входа. Кроме того, модели RNN последовательность-к-последовательности не смогли достичь современных результатов в режимах с малым количеством данных.

Мы обучили 4-слойный трансформер с d_model = 1024 на части Wall Street Journal (WSJ) Penn Treebank, примерно 40K обучающих предложений. Мы также обучили его в полу-контролируемой настройке, используя более крупные корпуса с высокой достоверностью и BerkleyParser с приблизительно 17M предложений. Мы использовали словарь из 16K токенов только для настройки WSJ и словарь из 32K токенов для полу-контролируемой настройки.

Мы провели лишь небольшое количество экспериментов для выбора dropout, как для внимания, так и для остаточных соединений (раздел 5.4), скоростей обучения и размера луча на наборе для разработки Раздела 22, все остальные параметры остались неизменными от базовой модели перевода с английского на немецкий. Во время вывода мы увеличили максимальную длину выхода до длина входа + 300. Мы использовали размер луча 21 и α = 0.3 как для настройки только WSJ, так и для полу-контролируемой настройки.

Таблица 4: Transformer хорошо обобщается на синтаксический разбор английского языка

Парсер	**Обучение**	**WSJ 23 F1**
Vinyals & Kaiser et al. (2014)	только WSJ, дискриминативный	88.3
Petrov et al. (2006)	только WSJ, дискриминативный	90.4
Zhu et al. (2013)	только WSJ, дискриминативный	90.4
Dyer et al. (2016)	только WSJ, дискриминативный	91.7
Transformer (4 слоя)	**только WSJ, дискриминативный**	**91.3**
Zhu et al. (2013)	полу-контролируемый	91.3
Huang & Harper (2009)	полу-контролируемый	91.3
McClosky et al. (2006)	полу-контролируемый	92.1
Vinyals & Kaiser et al. (2014)	полу-контролируемый	92.1
Transformer (4 слоя)	**полу-контролируемый**	**92.7**
Luong et al. (2015)	многозадачный	93.0
Dyer et al. (2016)	генеративный	93.3

Наши результаты в Таблице 4 показывают, что несмотря на отсутствие специфической для задачи настройки, наша модель работает удивительно хорошо, давая лучшие результаты, чем все ранее опубликованные модели, за исключением Recurrent Neural Network Grammar.

В отличие от моделей RNN последовательность-к-последовательности, Трансформер превосходит BerkeleyParser даже при обучении только на обучающем наборе WSJ из 40K предложений.

---

7. Заключение

В этой работе мы представили Трансформер - первую модель преобразования последовательностей, основанную полностью на внимании, заменяющую рекуррентные слои, наиболее часто используемые в архитектурах кодировщик-декодировщик, многоголовым самовниманием.

Для задач перевода Трансформер может обучаться значительно быстрее, чем архитектуры, основанные на рекуррентных или свёрточных слоях. Как на задаче перевода WMT 2014 с английского на немецкий, так и на задаче WMT 2014 с английского на французский мы достигаем нового рекордного уровня. В первой задаче наша лучшая модель превосходит даже все ранее опубликованные ансамбли.

Мы с энтузиазмом смотрим на будущее моделей на основе внимания и планируем применить их к другим задачам. Мы планируем расширить Трансформер на проблемы, включающие входные и выходные модальности, отличные от текста, и исследовать локальные, ограниченные механизмы внимания для эффективной обработки больших входов и выходов, таких как изображения, аудио и видео. Сделать генерацию менее последовательной - ещё одна из наших исследовательских целей.

Код, который мы использовали для обучения и оценки наших моделей, доступен на https://github.com/tensorflow/tensor2tensor.

Благодарности

Мы благодарны Налу Калчбреннеру и Стефану Гоувсу за их плодотворные комментарии, исправления и вдохновение.

Визуализации внимания

Примечание к рисункам:

Оригинальная статья содержит следующие визуализации механизмов внимания:

Рисунок 3: Пример механизма внимания, следующего за зависимостями дальнего действия в самовнимании кодировщика в слое 5 из 6. Многие из голов внимания обращают внимание на удалённую зависимость глагола 'making', завершая фразу 'making...more difficult'. Внимание здесь показано только для слова 'making'. Разные цвета представляют разные головы.

Рисунок 4: Две головы внимания, также в слое 5 из 6, по-видимому, участвующие в разрешении анафоры. Вверху: Полное внимание для головы 5. Внизу: Изолированное внимание только от слова 'its' для голов внимания 5 и 6. Обратите внимание, что внимание очень острое для этого слова.

Рисунок 5: Многие из голов внимания демонстрируют поведение, которое кажется связанным со структурой предложения. Мы даём два таких примера выше, из двух разных голов из самовнимания кодировщика на слое 5 из 6. Головы явно научились выполнять различные задачи.

Список литературы

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[3] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. CoRR, abs/1703.03906, 2017.

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. arXiv preprint arXiv:1601.06733, 2016.

[5] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. CoRR, abs/1406.1078, 2014.

[6] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357, 2016.

[7] Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555, 2014.

[8] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In Proc. of NAACL, 2016.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. arXiv preprint arXiv:1705.03122v2, 2017.

[10] Alex Graves. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.

[Остальные ссылки продолжаются до [40] в оригинальном формате...]

Примечание: Эта статья - перевод знаменитой работы "Attention Is All You Need" (2017), которая представила архитектуру Transformer, революционизировавшую область обработки естественного языка и ставшую основой для современных моделей, таких как BERT, GPT и многих других.

Оригинальная публикация: 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. **arXiv:** 1706.03762v7 [cs.CL] 2 Aug 2023