Трансформеры, модель BERT: архитектура и применение

Введение

[2]
Для начала вспомним, что основным подходом для работы с последовательностями до 2017 года являлось использование рекуррентных нейронных сетей. Однако у такого подхода есть несколько известных минусов:

1. Во-первых, RNN содержат всю информацию о последовательности в скрытом состоянии, которое обновляется с каждым шагом. Если модели необходимо «вспомнить» что-то, что было сотни шагов назад, то эту информацию необходимо хранить внутри скрытого состояния и не заменять чем-то новым. Следовательно, придется иметь либо очень большое скрытое состояние, либо мириться с потерей информации.
2. Во-вторых, обучение рекуррентных сетей сложно распараллелить: чтобы получить скрытое состояние RNN-слоя для шага i + 1, вам необходимо вычислить состояние для шага i. Таким образом, обработка батча примеров длиной 1000 должна потребовать последовательных операций, что занимает много времени.

Таким образом, обе эти проблемы затрудняют применение RNN к по-настоящему длинным последовательностям: даже если вы дождетесь конца обучения, ваша модель по своей конструкции будет так или иначе терять информацию о том, что было в начале текста. Хочется иметь способ «читать» последовательность так, чтобы в каждый момент времени можно было обратиться к произвольному моменту из прошлого.

[3]
Пример:
Предположим, что при переводе с английского на русский мы столкнулись со следующими предложениями:
The animal didn't cross the street because it was too tired.
The animal didn't cross the street because it was too wide.
Переводчику не очевидно, как нужно перевести слово it – как “оно” или “она”.
Для решение подобной задачи в 2015 году в рекуррентные сети был добавлен механизм внимания.

[4]

Сразу хотел бы сказать, что в рамках своего доклада я буду рассматривать архитектуры механизма внимания и трансформера на примере задачи машинного перевода.
Главной идеей механизма внимания является то, что он позволяет для каждого выхода нейронной сети иметь вектор *c* входного контекста. При синтезе нового слова это дает возможность “посмотреть” сразу на все предложение.

Здесь представлена схема работы механизма внимания. Так, желтым цветом обозначен энкодер, синим декодер, а черными овалами – функция внимания. Attention, используя состояния энкодера и декодера, выдает значения, показывающие, насколько важно посмотреть на каждое слово входной последовательности для получения текущего слова в декодере. После нормировки данных значений функцией softmax, мы получаем распределение важности каждого слова для вывода текущего. В итоге для каждого выходного слова мы получаем вектор контекста c. Таким образом, при работе с данным словом мы сможем учитывать не только его конкретный перевод, но еще и его контекст для более точного перевода.

[5]
Теперь предлагается посмотреть на пример работы механизма внимания при переводе предложения с французского языка на английский. Чем светлее квадратик на пересечении двух слов, тем важнее было обратить внимание на конкретное слово во французском языке для получения перевода. Так, можем заметить, что attention учел различие в порядке слов в французском и английском языках.

[6]
Новая проблема: пусть нам нужно перевести длинное предложение:
A man in a nice cowboy hat and a parrot on his shoulder has entered the bar.
Человек в красивой ковбойской шляпе и попугаем на плече … [зашел? зашла? зашло? залетело?]
На предыдущих слайдах мы рассматривали механизм внимания для определения более корректного смысла на русском языке из предложения на английском. Теперь же для верного перевода на русский язык нам нужно смотреть не только на контекст на английском, но и на русском.
С этой задачей помогает справиться механизм self-attention. Его принцип аналогичен attention, только в векторе контекста с учитываются слова той же последовательности, откуда мы берем слово.

[7]
Если в модели несколько механизмов внимания и само-внимания, она называется multi-head attention. При этом каждый из них может начать выделять определенную информацию, например, согласование по падежу/лицу и т. д.

[8]
[9]
Пришло время перейти к трансформерам. Трансформер - нейросетевая архитектура на основе внимания и полносвязных слоев. Эта технология была описана в статье “Attention is all you need” исследователями из Google в 2017-м году. Модель поистине произвела революцию в NLP, так как ее внедрение привело к значительно лучшим

результатам качества и скорости в задачах обработки текста, например, в машинном переводе. Кстати, изначально трансформер позиционировался именно как решение задачи машинного перевода, однако благодаря своей универсальности и масштабируемости этот механизм оказался применим к множеству задач помимо обработки естественного языка.
Данная схема была представлена в оригинальной статье, однако удобнее рассматривать более наглядное ее представление.

[10]

[11]
Глобально трансформер состоит из энкодера и декодера. Напомню, что энкодер извлекает информацию из входящей последовательности, а декодер использует извлеченную информацию для генерации элементов последовательности на выходе (например, текста на другом языке).

[12]
Кодирующий элемент представляет собой набор одинаковых энкодеров, соединенных последовательно. В работе “Attention is all you need” их было 6. Таким образом, входная последовательность проходит через 6 этапов кодирования, после чего результат подается в каждый из 6 декодеров. Здесь 6 это лишь гиперпараметр, который как мы увидим далее меняется в зависимости от решаемой задачи.

[13]
Каждый энкодер можно разбить на 2 слоя, первый называется self-attention. Сначала данные попадают внутрь него, что позволяет выявить отношения между словами. Второй уровень это полносвязная сеть, позволяющая сделать обучаемое преобразование выходных данных со слоя внимания, чтобы они стали более информативными.
Декодер также содержит слои self-attention и feedforward, но между ними есть слой внимания, который позволяет декодеру сфокусироваться на релевантных частях входящего предложения

[14]
Первый этап – вычислить матрицы запроса, ключа и значения. Это делается с помощью формирования из эмбеддингов матрицы X и ее умножения на матрицы весов, которые мы обучили (WQ, WK, WV).
Что представляют собой матрицы «запроса», «ключа» и «значения»?
Это абстракции, которые оказываются весьма полезны для понимания и вычисления внимания.
Умножение x1 на матрицу весов WQ производит q1 – вектор «запроса», относящийся к этому слову. В итоге мы создаем проекции «запроса», «ключа» и «значения» для всех слов во входящем предложении.

Второй этап вычисления внутреннего внимания – получение коэффициента (score). Допустим, мы подсчитываем внутреннее внимание для первого слова в

последовательности. Нам нужно оценить каждое слово во входящем предложении по отношению к данному слову. Коэффициент определяет, насколько нужно сфокусироваться на других частях входящего предложения во время кодирования слова в конкретной позиции. Коэффициент подсчитывается с помощью произведения матрицы запроса и матрицы ключа.

[15]
Третий и четвертый этапы – разделить эти коэффициенты нормировочную константу, а затем пропустить результат через функцию софтмакс (softmax). Данная функция нормализует коэффициенты так, чтобы они были положительными и в сумме давали 1. Полученный софтмакс-коэффициент (softmax score) определяет, в какой мере каждое из слов предложения будет выражено в определенной позиции.
Пятый этап – умножить матрицу значения на софтмакс-коэффициент (перед их сложением). Интуиция здесь следующая: нужно держать без изменений значения слов, на которых мы фокусируемся, и отвести на второй план нерелевантные слова. На этом завершается вычисление внутреннего внимания. В результате мы получаем вектор, который можем передавать дальше в нейронную сеть прямого распространения.

[16]
В силу множественного внимания у нас есть не один, а множество наборов матриц запроса/ключа/значения. Трансформер использует 8 «голов» внимания, так что в итоге у нас получается 8 наборов для каждого энкодера/декодера. В результате получим 8 разных Z матриц, но слой сети прямого распространения не ожидает, что к нему поступит 8 матриц – он ждет всего одну, в которую нам и необходимо сжать полученные Z матрицы. Как это сделать?

[18]
Конкатенировать и затем умножить их на дополнительные веса матрицы WO.

[19]
Попробуем посмотреть на все в одном месте.

[20]
Энкодер:

1. Для учета порядка слов входной последовательности в Трансформеры добавляют вектор в каждый входящий эмбеддинг. Эти векторы имеют определенный шаблон, который модель запоминает и который помогает определить позицию каждого слова или расстояние между разными словами в предложении.
2. Multi-head self-attention обнаруживает связи между словами.
3. Нормирование уровня (Add & Norm) нужно для приведения векторов к одной размерности. А сквозная связь предотвращает затухание градиента.
4. Полносвязная сеть позволяет сделать обучаемое преобразование выходных данных со слоя внимания, чтобы они стали более информативными.

Декодер:

1. В декодере слой внутреннего внимания может фокусироваться только на предыдущих позициях в выходном предложении. Это делается с помощью маскировки всех позиций после текущей (устанавливая их в –inf) перед этапом софтмакс в вычислении внутреннего внимания.
   Маскирование запрещает “подглядывать” в те слова, которые еще не сгенерированы. Для чего это нужно? Это нужно для того, чтобы уровень masked multi-head attention правильно указал, на какие слова уже сгенерированной последовательности нужно обратить внимание, чтобы синтезировать следующее слово, и не указал на слова “из будущего”, так как мы хотим составлять выходное предложение последовательно.
2. Multi-head attention работает почти так же, как multi-head self-attention, только у него слегка меняются входные данные: матрицы ключей и значений поступают от энкодера, а запрос от вышестоящего слоя внимания.

Финал:
Стек декодеров на выходе возвращает вектор чисел с плавающей точкой. Как можно получить из этого вектора слово? За это отвечает линейный слой и следующий за ним слой софтмакс.
Линейный слой – это простая полносвязная нейронная сеть, которая переводит вектор, созданный стеком декодеров, в значительно больший вектор, называемый логит вектором (logits vector). Таким образом мы интерпретируем выход нашей модели с помощью линейного слоя.
Слой софтмакс переводит этот показатель в вероятности. Выбирается ячейка с наиболее высокой вероятностью и на выход подается соответствующее слово.

[21]
Еще одна иллюстриция.

[22]
[23]
Компания Google разработала языковую модель BERT в 2018 году. Она состоит из простого набора блоков-трансформеров, который был предварительно обучен на большом корпусе текстов общего характера.
Как понятно из названия, модель Bidirectional Encoder Representations from Transformers (или BERT) отличается двунаправленностью внимания: это значит, что при обработке входной последовательности все токены могут использовать информацию друг о друге. Это делает такую архитектуру более удобной для задач, где нужно сделать предсказание относительно всего входа целиком без генерации.
БЕРТ был специально обучен на текстах с Википедии (~2,5 миллиарда слов) и Google BooksCorpus (~800 миллионов слов).

[24]
Архитектура transformer содержит кодер и декодер. Архитектура BERT отличается от структуры традиционного трансформера тем, что модель собирает кодеры друг за другом.

[25]
Исследователи в области компьютерного зрения неоднократно демонстрировали пользу трансферного обучения – предварительного обучения модели нейронной сети на хорошо известной задаче, с последующим дообучением с использованием уже обученной нейронной сети в качестве основы для новой модели с конкретной направленностью. В последние годы исследователи пришли к выводу, что подобная техника может быть чрезвычайно полезна и во многих задачах обработки естественного языка.

BERT обучается одновременно на двух задачах — предсказание следующего предложения и генерация пропущенного токена.
На вход BERT подаются токенизированные пары предложений, в которых некоторые токены скрыты. Таким образом сеть обучается глубокому двунаправленному представлению языка, учится понимать контекст предложения.
Задача же предсказания следующего предложения есть задача бинарной классификации — является ли второе предложение продолжением первого. Благодаря ей сеть можно обучить различать наличие связи между предложениями в тексте.

[26]
В рамках процесса обучения BERT модель в качестве входных данных получает пары фраз, на которых она учится предсказывать, является ли вторая фраза в паре следующей после первой в исходном тексте. Во время обучения 50% входных данных представляют собой пары, в которых вторая фраза действительно является следующей фразой в исходном тексте, а в остальных 50% в качестве второй фразы выбирается случайная фраза из того же текста.

Чтобы помочь модели различить две фразы в процессе обучения, перед входом в модель входные данные обрабатываются следующим образом:

1. В начало первой фразы вставляется токен [CLS]. В конец каждой из фраз вставляется токен [SEP].
2. К каждому токену добавляется эмбеддинг фразы, обозначающий Фразу A или Фразу B. Эмбеддинги фраз по своей концепции аналогичны эмбеддингам токенов со словарем из двух элементов.
3. К каждому токену добавляется позиционный эмбеддинг, чтобы указать его положение в последовательности.

Чтобы предсказать, действительно ли вторая фраза связана с первой, выполняются следующие шаги:

1. Вся входная последовательность проходит через модель-трансформер.
2. Выход токена [CLS] преобразуется в вектор размерности 2×1 с помощью простого слоя классификации.
3. Вычисление вероятности с помощью softmax.

[27]
BERT можно использовать для самых разных языковых задач:

1. В 2019 году компания Google объявила об использовании BERT для анализа англоязычных поисковых запросов. В конце того же года также было начато использование модели в алгоритме поиска на других языках.
2. Задачи классификации, такие как, например, анализ тональности, выполняются аналогично классификации “следующей фразы”, добавляя слой классификации поверх выходных данных трансформера для токена [CLS].
3. В задачах формирования ответов на вопросы программа получает вопрос относительно текстовой последовательности и должна отметить ответ.
4. При распознавании именованных объектов программа получает текстовую последовательность и должна помечать различные типы объектов (человек, организация, дата и т. д.), которые появляются в тексте. Используя BERT, модель можно обучить, пропуская выходной вектор каждого токена через классификационный слой, который прогнозирует метку.