# Зачем нужна каузальная маска в трансформерах?Изображение выглядит как текст, диаграмма, снимок экрана, План Автоматически созданное описание

Перед нами оригинальная архитектура трансформера из статьи «Attention is all you need». Как мы видим, она состоит из различных блоков, основа которых – механизм внимания. Есть Multi-Head Attention (MHA) из encoder – блок так называемого самовнимания. Есть Multi-Head Attention из декодера, он называется Cross-Attention, разница между самовнимением и Cross-Attention в том, что в самовнимании и запросы, и ключи, и значения получаются на основе входящей последовательности эмбеддингов, а в Cross-Attention запросы используются из декодера, а ключи и значения из энкодера. Но помимо MHA есть Masked Multi-Head Attention (MMHA). Это аналог самовнимания из энкодера, но с той лишь разницей, что при обучении в этом блоке информации из будущих эмбеддингов запрещено влиять на прошлые эмбеддинги. Проявляется это в том, что после просчёта матрицы внимания, перед применением softmax, на местах, соответствующих влиянию эмбеддингов из будущего на эмбеддинги из прошлого, ставится -inf. Таким образом, после применения softmax, влияние этих эмбеддингов будет нулевым.

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Автоматически созданное описание

При использовании Masked MHA получается приятная особенность. Пусть мы обучаем модель и у нас есть вход: “Life is short eat deserts first”. Мы хотим, чтобы наша модель училась предсказывать следующее слово, то есть выполняла следующие предсказания:

“Life” -> is

“Life is” -> short

“Life is short” -> eat

“Life is short eat” -> deserts

“Life is short eat deserts” -> first.

Кажется, что нам нужно запустить модель 5 раз. Но давайте посмотрим на архитектуру GPT:

Изображение выглядит как текст, снимок экрана, линия, Шрифт

Автоматически созданное описание

Единственный блок, в котором происходит обмен информацией между эмбеддингами – Masked Multi Self Attention. Остальные блоки применяют преобразование к эмбеддингам поэлементно. В маскированном самовнимании из-за каузальности и замены на -inf, последующие слова влияют только на предыдущие:

Изображение выглядит как зарисовка, рисунок, дизайн

Автоматически созданное описание

Также матрицы внимания одинаковым образом умножаются на все элементы последовательности. Таким образом, нет смысла запускать нейронную сеть для предсказания следующего слова каждый раз.

Изображение выглядит как диаграмма, линия, График, План

Автоматически созданное описание

Например, на рисунке выходной эмбеддинг для элемента последовательности i будет одинаковым и если мы подадим на вход нейронной сети последовательность до i-го токена, и даже если подадим всю последовательность целиком. Так, получается, что для одного куска текста нужно прогнать его 1 раз при обучении.

Но что происходит при inference? Представим, что трансформер уже сгенерировал несколько токенов, допустим он уже сгенерировал “Мороз и солнце день …”. Давайте посмотрим, как будет происходить генерация следующего слова:

Изображение выглядит как диаграмма, линия, Технический чертеж, зарисовка

Автоматически созданное описание

Хотя мы и знаем всю сгенерированную последовательность, но все равно не позволяем слову “солнце” влиять на слово “мороз”. С одной стороны это кажется логичным, но с другой стороны – последовательность уже сгенерирована и что мешает модели “осмыслить” то, что она уже сгенерировала в будущем и внести правки в представления из прошлого? Мы не говорим о том, что стоит поменять выход модели, нет, сгенерированные токены мы не будет заново генерировать, но позволим влиять на будущие токены:

Изображение выглядит как диаграмма, зарисовка, Технический чертеж, линия

Автоматически созданное описание

Конечно, в таком случае при генерации придётся не сладко, потому что уже нельзя будет как раньше использовать кэшированные значения с предыдущего запуска, так как все эмбеддинги в промежуточных слоях изменятся. Также и при обучении теперь уже придётся делать отдельные запуски для каждой подпоследовательности:

Изображение выглядит как диаграмма, Технический чертеж, линия, План

Автоматически созданное описание

Раньше = = , но сейчас, очевидно, не так, и поэтому число вычислений значительно возрастёт. Например, длина входной последовательности была n\_. В исходном варианте число вычислений было (грубо говоря) O(n\_layers \* n \* n). А теперь станет;

O(n\_layers \* (1^2 + 2^2 + 3^2 + … n ^2)) = O(n\_layers \* n \* (n+1) \* (2n + 1) / 6), то есть сложность от длины последовательности с квадратичной станет – полиномиальной третьей степени. Но тем не менее, наличие маски в трансформере не обязательно и интересно понять, как изменится его работа, если её убрать. Есть надежда, что данный подход позволит увеличить качество на тех наборах данных и задачах, где данных немного и можно пожертвовать скоростью обучения и инференса.

# Эксперименты. Трансформер без маски и с ней.

Обучим трансформер с маской и без маски. Для того, чтобы обучение трансформера было таким же, как и в случае наличия маски, напишем вспомогательную функцию для того, чтобы можно было практически без изменений использовать обучение обычного трансформера с маской:

def get\_loss\_without\_mask(batch, model, criterion):  
 trg = batch[:, 1:].to(device)  
 n\_seq = batch.shape[1]  
 outputs = []  
 for i in range(1, n\_seq):  
 input = batch[:, :i]  
 output = model(input.to(device))  
 outputs.append(output[:, -1, :])  
 output = torch.stack(outputs, dim=1)  
 seq\_output = output.reshape(trg.shape[0], trg.shape[1], -1)  
 return criterion(seq\_output.transpose(1, 2), batch[:, 1:].to(device))

Этот код можно вставить в eval и train методы. Вот отличие этих двух подходов:

            if masked:

                assert model.use\_causal\_mask

                logits = model(src)

                loss = criterion(logits.transpose(1, 2), trg)

            else:

                assert not model.use\_causal\_mask

                loss = get\_loss\_without\_mask(batch, model, criterion)

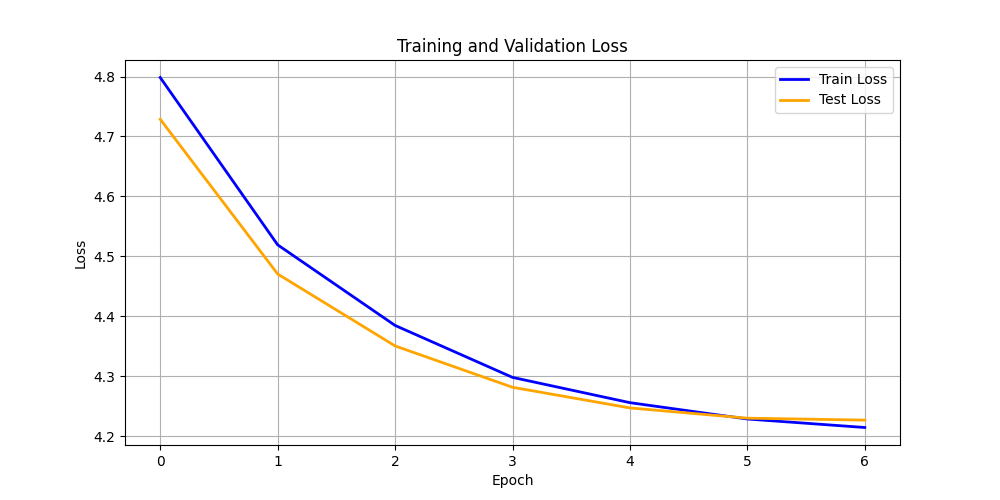
Мы выбрали следующие параметры модели в силу ограниченности ресурсов и сильной нагрузке на GPU при обучении без маски:

|  |  |
| --- | --- |
| Размер батча | 26 |
| Число эпох | 7 |
| Размер контекста | 128 |
| Размер словаря | 8000 |
| Число слоёв | 3 |
| Число голов | 4 |
| Промежуточная размерность в MLP | 1024 |
| Общее число параметров | 8446836 |

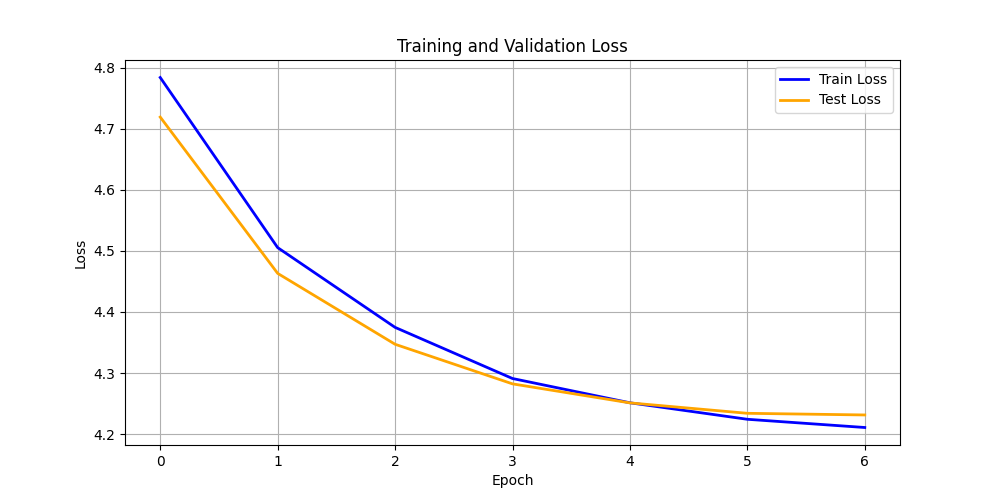
При обучении использовалась кросс-энтропия в качестве функции потерь. Также использовался косинусный планировщик. Вот сравнение этих двух моделей:

|  |  |  |
| --- | --- | --- |
|  | С маской | Без маски |
| Лосс (кросс-энтропия) | 4.227 | 4.231 |
| Время обучения (мин) | 20 | 559 |
| Используемая память GPU при обучении (Mb) | 3147 | 22855 |

Графики лоссов:



Лосс модели с маской



Лосс модели без маски

Как можно видеть, разницы особо никакой нет.

На вход модели подавалась последовательность и задачей было продлить эту последовательность. Генерация модели с маской:

***мои дядя самых честных правил****, - я должен был его любишь изъяснить. вот ты не то что на моем воспоминание, где жизнь, нет, и я сам был сам, как в уме моем кель*

***четыре года потратил деонардо на*** *меня, на меня не пролишь ; а он вам отвечал. я не люблю, потому что будет угодно! я не могу, что еи ни один только что мог бы*

***если сила плохих людеи******в том, что они вместе, то хорошим людям, чтобы стать силои, надо.*** *ну, сам, я им теперь говорю ; я верю!. все эти товары*

Генерация модели без маски:

***мои дядя самых честных правил,*** *а не совсем даже никто другои и не был убежден. " умнистныи, когда - то никогда, а по нашему, что на первому не скажет, что человек*

***четыре года потратил деонардо*** *на скашечныи чашу. - - - еи стансеи не выпуск в своеи, а в тоскную, которая может ехать, как бы в*

***если сила плохих людеи в том, что они вместе****, то хорошим людям, чтобы стать силои, надо, помогать в них, а в то есть ; иначе быть может, и сами собес*

В генерации использовалось сэмплирование из 100 самых вероятных токенов. Особо никаких различий в генерации той или иной модели замечено не было – они генерируют одинаково плохо.

# Выводы

В результате никаких особых отличий в качественном плане между моделью с маской и без маски обнаружено не было. Однако при обучении маска позволяет значительно экономить как память, так и время. Времени на обучение без маски потребовалось в 28 раз больше, памяти использовалось в 7 раз больше. Это связано с тем, что сложность при обучении модели без маски меняется с квадратичной на полиномиальную третьей степени. Конечно, возможно найти какой-либо другой подход, подобрать более эффективный код, оптимизировать что-либо, чтобы уменьшить разрыв по времени и памяти, указанный выше, но это всё все равно не будет иметь особого смысла так как модель с маской и без маски показала одинаковое качество.