Quora question pairs challenge

Dmitry Salnikov

November 2019

Данный документ описывает модели на основе [1] и их применение к детектированию парафраз, а также применение этих моделей в соревновании на kaggle Quora question pairs detection. Еще здесь будут идеи, потенциально способные улучшить качество модели.

1 Описание моделей

В данной секции описываются модели и детали их реализации. Указанные значения на валидации считаются на случайном разбиении датасета. Размер валидационной выборки 10000 точек.

1.1 Модель 1

Модель 1 представляет собой полносвязную нейронную сеть, использующую скрытые состояния BERT(используется модель на bert-base-uncased из huggingface, предобученная на строчных английских текстах) как входные фичи. Конкретнее, используется 2-й с конца скрытый слой(Это лучше работает, смотрите, например). Для каждого из двух входных вопросов получается векторное представления, они собираются в один вектор, который потом передается на вход полносвязной сети.

Обозначим batch_size - размер входного батча. Пусть вопрос 1 кодируется последовательностью seq_1, вопрос 2 - последовательностью seq_2 . На выходе из BERT получаем два вектора:

- vec 1 : batch_size $\times$ len(seq_1) $\times$ 768
- vec 2 : batch_size $\times$ len(seq_2) $\times$ 768

Мы хотим получить из этого вектор размерности 768$\times$2. Для этого можно использовать несколько различных стратегий:

- усреднить по размерности 1 (a.k.a average pooling)
- VectorAttention
- NNAttention
- Seq2SeqAttention

Опишем эти подходы более подробно:

1.1.1 VectorAttention

Вместо того чтобы усреднять с равномерными весами, можно попробовать их обучить. Данный слой представляет собой линейный слой, осуществляющий линейное преобразование $768 \to 1$. Таким образом, каждому вектору из последовательности сопоставляется некоторое число. Потом это все пропускается через softmax, чтобы суммировалось к 1. Суммируя входные последовательности с полученными весами получаем искомое представление.

1.1.2 NNAttention

Этот слой обобщает предыдущий подход, но вместо линейного слоя использует полносвязную сеть и tanh активацией. Полученные веса все так же пропускаются через softmax. Последовательности суммируются с полученными весами

1.1.3 Seq2SeqAttention

Заметим, что предыдущие слои никак не используют информацию о векторах другого предложения при рассчете весов. Данный слой пытается это исправить. Идея похожа на то что происходит при рассчете весов self-attention в трансформере. Обучается матрица, которая линейно проецирует данный вектор на вектор некоторой заданной размерности. Затем считаются попарные скалярные произведения, которые потом преобразуются в веса. Все это можно эффективно реализовать в виде перемножений матриц. Также заметим, что данная конструкция симметрична относительно перестановки предложений местами, что логично в данной задаче.

Эти слои реализованы в `layers.py`. Сама модель в `models.py`

Изложенные подходы ускоряют сходимость в смысле количества шагов, а также улучшают logloss на валидационной выборке. Эмпирически на валидации работает вот так: VectorAttention $<$ NNA $<$ Seq2SeqAttention.

В данной модели BERT никак не обучается, но реализация легко позволят включить это при надобности. Сеть-классификатор в данном случае имеет следующую архитектуру 1

Применение данного подхода дает логлосс примерно 0.35-0.4 на валидации. Графики обучения приведены на 2, 3.

1.2 Модель 2

В отличие от первой модели, вторая использует только CLS токен и линейный слой над ним. Также, веса BERTa не фиксируются. Используется модель bert-base-cased-finetuned-mrpc. Это стандартный BERT [1] для классификации, дополнительно обученный на Microsoft Research Paraphrase Corpus. Так как задача схожая, имеет смысл посмотреть на значения метрик до обучения. Получается примерно 0.6 по accuracy и f1. Если аккуратно дообучить эту модель на датасете с вопросами, получим accuracy $>$ 0.9 и f1 $>$ 0.9 на валидации, что является очень неплохим результатом. График обучения на 4 .

2 Применение моделей к решению контеста

В решении не используется никаких ручных фичей, только описанные выше. Обе модели обучаются на стандартном датасете без аугментации. Возможный способ аугментации будет описан ниже. Тренировочная выборка разбивается случайно на обучающую и валидационную выборку. Размер валидационной выборки 10000, все остальное обучающая. Обучение моделей занимает несколько часов на GTX1080(в сумме примерно 15 часов для первой, 5 часов для второй). Код для обучения каждой из моделей есть в соответствующем ноутбуке(`model1.ipynb` и `model2.ipynb` соответственно).

Составление предсказаний для одной модели занимает около 4 часов на GTX1080 в силу большого размера тестовой выборки.

2.1 Описание и обработка датасета

Датасет состоит из набора пар вопросов и лейблов. Доля положительных примеров составляет 0.36. Это не является сильным дизбалансом для данной задачи(почему будет описано ниже). Заметим, что отношение 'похожие вопросы', является отношением эквивалентности. Соответственно, лейбл для любой пары из тренировочной выборки может быть получен

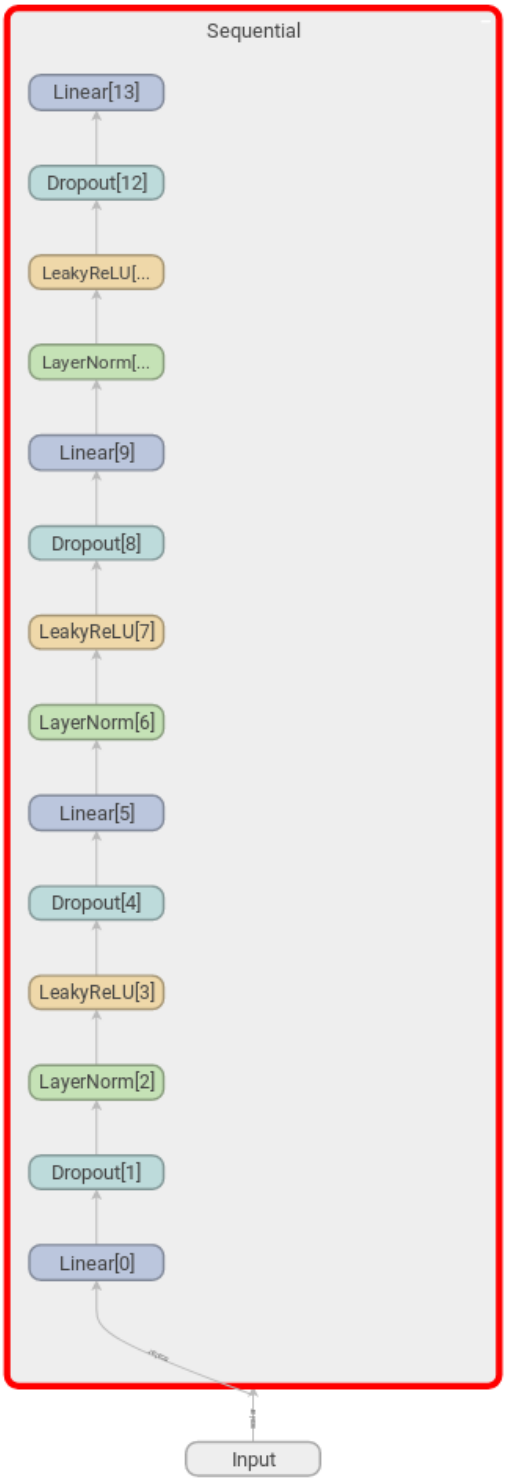

Figure 1: Classificaiton Network

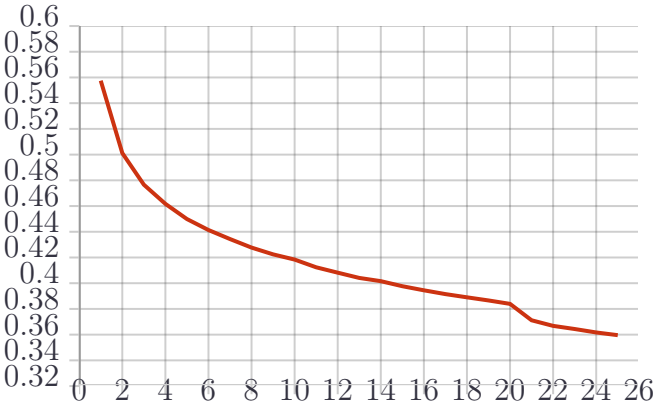

Figure 2: train loss

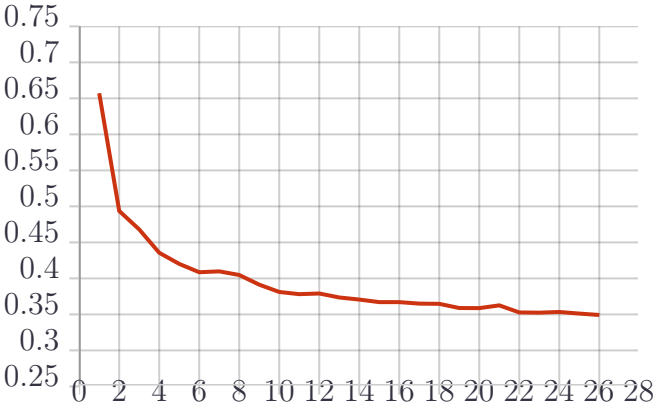

Figure 3: val loss

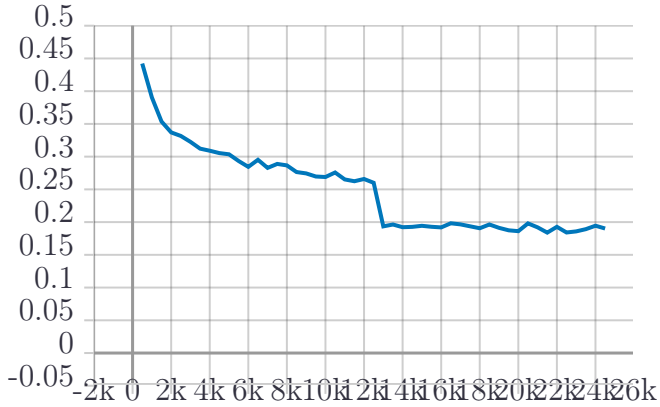

Figure 4: train loss

с помощью проверки принадлежности к компоненте связности графа (ребро=вопросы похожи). Так как далеко не все пары вершин есть в тренировочной выборке, можно получить гораздо больше данных и таким образом получить желаемый баланс классов. Еще в датасете есть несколько NaN значений. Из тренировочной выборки они выкидываются, для тестовой в них проставляется 0. Для ускорения работы 2й модели, для тренировочных векторов предпросчитываются векторы и маски токенизации.

2.2 Обучение моделей

Модель 1 обучается с помощью Adam и мультипликативным уменьшением шага начиная с некоторой итерации(подробнее `model1.ipynb`)

Модель 2 обучается с помощью AdamW и линейным уменьшением шага (подробнее `model2.ipynb`)

2.3 Kaggle magic™

Первая модель на лидерборде дает 0.35 - схоже с валидацией Вторая модель несмотря на гораздо лучший скор на валидации получает только 0.30 Взвешенная комбинация этих моделей получает около 0.26. Почему так происходит? После долгого курения форумов, выяснилось что в тестовой выборке другое распределение лейблов(0.17 положительных примеров). Не очень понятно, зачем составители соревнования так сделали, поскольку это ломает вообще все статистические методы. Получается, для получения хорошего скора в смысле логлосса, нам нужно переобучиться на предполагаемый баланс классов в лидерборде. Это можно сделать, например, с помощью аугментации датасета негативными примерами(получить много таких можно семплируя точки из разных компонент связности в графе). Однако, для этого нужно обучать все модели заново и пересчитывать предсказания для теста(потенциально это можно сделать, но хочется быстрее). Ответ нашелся в глубине форумов и состоит в применении некоторой функции, которая нелинейно преобразует предсказания модели с учетом изменений в процентах положительных примеров. Эта функция реализована в `utils.merge_submissions_to_replicate`. Смотрите также.
В конечном итоге получаем:

- public 0.24194
- private 0.24918

References

[1] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).