CMC MSU
Recent approaches to NLP with NNs
Assignment 3: Transliteration

Май 2021

В этом задании Вам предстоит решать задачу транслитерации имён людей с английского языка на русский. Под транслитерацией строки понимается запись этой строки при помощи алфавита другого языка с сохранением произношения, хотя и не всегда.

1 Теоретическая часть

Выполните следующие теоретические подзадачи:

- Основной компонентой архитектуры Transformer является слой MultiHeadAttention. Этот слой кратко описан в разделе 2.2, его логика определяется формулами 1 и 2. Опишите Ваши предположения: с какой целью перед применением $softmax$ элементы матрицы QK^T делятся на $\sqrt{d_k}$? ($d_k = d_{model}/h$ - размерность векторов-ключей, запросов и значений, d_{model} - размерность векторов на входе и выходе всех слоёв, h - число голов внимания). Какие могут возникнуть проблемы, затрудняющие обучение модели, если не делать деление на $\sqrt{d_k}$ и работать с большим значением d_{model}?

- Пусть на вход Transformer подаётся один пример - последовательность из n токенов. Перед обработкой очередным MultiHeadAttention (MHA) слоем она соответствует последовательности векторов $x_1 \dots x_n$. Выход с этого слоя - последовательность векторов $y_1 \dots y_n$, вектора можно упаковать в матрицу Y и обозначить преобразование в виде формулы $Y = \mathbf{MHA}(x_1 \dots x_n)$. Представим теперь, что имеется скрытый слой рекуррентной нейросети, который осуществляет аналогичное по типу входа и выхода преобразование $y_i = \mathbf{RecurrentLayer}(x_1 \dots x_i)$. Введем функцию $path(i, j)$, которая обозначает количество применений различных операций на пути создания вектора y_j из последовательности $x_i \dots x_j$. В терминологии *о-большого* $O(\cdot)$, зависящего от параметра n опишите, чему равняется $path(1, n)$ для рекуррентного слоя и для MultiHeadAttention слоя. (Например, $O(n \log(n)), O(n^2), O(1)$)

- В MultiHeadAttention слое Transformer на веса внимания накладывается маска, перед тем как рассчитывается взвешенная сумма векторов-значений. Опишите, какие типы масок используются?

2 Практическая часть

2.1 Ознакомление с базовым алгоритмом

Для задания в репозитории `https://github.com/nvanva/filimdb_evaluation` расположены набор данных и базовая реализация в translit_baseline.py. Базовый алгоритм основан на следующей идее: для осуществления транслитерации буквенные n-gram'ы с одного языка можно трансформировать на другой язык в n-gram'ы того же размера, используя самое частотное правило трансформации, выявленное по статистике на тренировочной выборке. Для тестировании реализации склонируйте репозиторий, разархивируйте датасеты и запустите оценочный скрипт. Для этого необходимо выполнить следующие команды:

1. git clone https://github.com/nvanva/filimdb_evaluation.git
2. cd filimdb_evaluation
3. ./init.sh
4. python evaluate_translit.py

2.2 Реализация алгоритма на основе Transformer

Для реализации Вашего алгоритма мы выкладываем шаблонный код, который необходимо дополнить. Файлы, которые будут использоваться:

- translit.py - основной файл, где Вам требуется внести изменения и который требуется загружать в систему проверки moodle.
- tests/test_translit_implementation.py - файл для тестирования модулей, которые Вам нужно будет реализовать в рамках этого задания.
- translit_utils/ - папка с двумя .py файлами, в которых реализована логика работы с данными и подсчет метрик.

Сначала Вам необходимо дополнить некоторые детали в коде архитектуры Transformer, реализовать начинку класса `LrScheduler`, который отвечает за обновление learning rate во время обучения. Далее требуется провести подбор гиперпараметров для модели по предложенному гайду.

Скрипт translit.py содержит следующие элементы:

- архитектуру модели Transformer в виде класса-наследника `torch.nn.Module`;
- сценарий обучения и тестирования модели в виде функций train и classify. Эти функции являются ключевыми для скрипта, который Вам потребуется загружать в moodle.

2.2.1 Positional Encoding

Как Вы помните, Transformer воспринимает на входе последовательность элементов как временной ряд. Поскольку Encoder внутри Transformer единовременно обрабатывает всю входную последовательность, информацию о позиции элемента требуется закодировать внутрь его embedding'а, поскольку далее внутри модели она никак иначе не идентифицируется. Для этого используется слой `PositionalEncoding`, который складывает embedding'и с вектором той же размерности. Матрицу этих векторов для каждой позиции временного ряда обозначим PE. Тогда компоненты матрицы определяются следующим образом:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

где pos - позиция во временном ряде, i - индекс компоненты соответствующего вектора, d_{model} - размерность каждого вектора. Таким образом, чётные компоненты представляют значения синуса, а нечётные - косинуса с разными аргументами.

В задании требуется реализовать эти формулы в виде кода внутри конструктора класса `PositionalEncoding` в основном файле translit.py, который впоследствии требуется загружать в систему moodle. После реализации запустите тест с помощью команды:

```
python -m tests.test_translit_implementation test_positional_encoding
```

Убедитесь, что отсутствует `AssertionError`!

2.2.2 MultiHeadAttention

Далее необходимо реализовать начинку метода `attention` в классе `MultiHeadAttention`. Сам `MultiHeadAttention` слой принимает на вход вектора запросов, ключей и значений для каждого шага последовательности в виде матриц `Q,K,V` соответственно. Каждый вектор-ключ, вектор-значение и вектор-запрос получаются в результате линейной проекции при помощи одной из трёх обучаемых матриц параметров вектора с предыдущего слоя. Эту семантику можно представить в виде формул:

$$\text{Attention}\ (Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

$$\begin{aligned}\text{MultiHead}\ (Q,K,V) =&\ \text{Concat}(\text{head}_1,\ldots,\ \text{head}_\text{h})\,W^O\\ \text{где head}_\text{i} =&\ \text{Attention}\ \left(QW_i^Q, KW_i^K, VW_i^V\right)\end{aligned} \tag{2}$$

h - число так называемых голов внимания, то есть параллельных подслоёв, реализующих Scaled Dot-Product Attention на векторе меньшей размерности ($d_k = d_q = d_v = d_{model}/h$). Требуемый по заданию метод `attention` соответствует Attention из формулы 1. Также логику работы `MultiHeadAttention` можно проследить на рисунке 1 (из оригинальной статьи [1]):

Внутри метода `attention` тажке предлагается применить *dropout* слой из конструктора MultiHeadAttention класса. Dropout слой следует применять непосредственно на веса внимания, т.е. результат *softmax* операции. Значение drop probability для этого dropout можно будет регулировать в функции `train` в `model_config['dropout']['attention']`.

Проверьте правильность реализации:

```
python -m tests.test_translit_implementation test_multi_head_attention
```

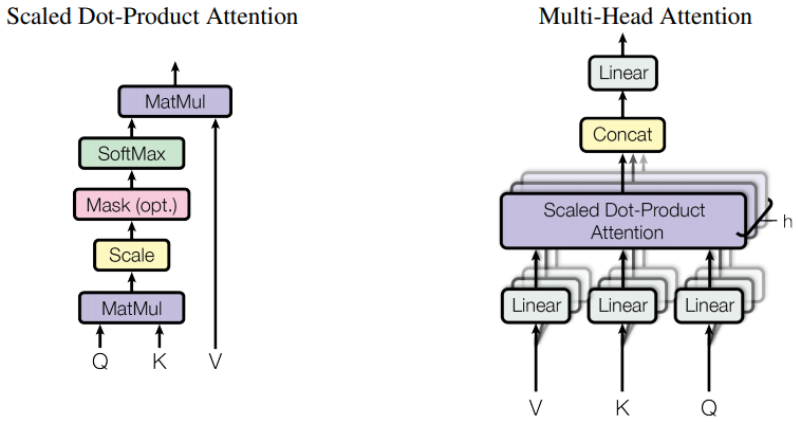

Рис. 1: Attention layer. Figure from [1].

2.2.3 LrScheduler

Последнее, чего не хватает для готовой модели - начинка класса LrScheduler, отвечающего за обновление learning rate после каждого шага оптимизатора по обновлению весов модели. Необходимо заполнить конструктор класса и метод `learning_rate`. Предлагается реализовать стратегию обновления learning rate (lr), которая описывается двумя стадиями:

- "warmup" стадия - lr поднимается линейно до определенного значения в течение фиксированного числа шагов (определяется как доля от числа всех шагов обучения - параметр `train_config['warmup_steps_part']` в функции train).
- "decrease" стадия - lr линейно опускается до нуля в течение оставшихся шагов обучения.

Вызов `learning_rate()` должен возвращать значение lr на очередном шаге, номер которого хранится в self.step. Конструктору класса передаются помимо warmup_steps_part пиковое значение lr в конце стадии "warmup" через lr_peak параметр и строковое название этой стратегии обновления lr - через type параметр. При желании Вы можете впоследствии попробовать другую стратегию обновления lr и вызывать её при помощи этого же self.type поля класса.

Проверка корректности реализации:

```
python -m tests.test_translit_implementation test_lr_scheduler
```

2.3 Подбор гиперпараметров

Реализация модели готова. Теперь требуется провести подбор оптимальных гиперпараметров.

В репозитории в директории TRANSLIT помимо train также расположена dev выборка и уменьшенные версии train и dev. Качество моделей с разными гиперпараметрами необходимо отслеживать на dev или на dev_small выборке (в целях экономии времени, поскольку генерация транслитераций - достаточно затратный по времени процесс, сравнимый с одной эпохой обучения). Скопируйте имеющийся код из `translit.py` в новый скрипт и реализуйте там необходимый evaluation code в функции train. Для генерации предсказаний можете использовать

функцию generate_predictions, для подсчёта метрики accuracy@1 - функцию `compute_metrics` из скрипта `metrics.py` в директории translit_utils.

Гиперпараметры хранятся внутри словаря `model_config` и `train_config` в функции train. Следующие гиперпараметры в `model_config` и `train_config` предлагается оставить без изменений:

- n_layers = 2
- n_heads = 2
- hidden_size = 128
- ff_hidden_size = 256
- warmup_steps_part = 0.1
- batch_size = 200

Варьировать предлагается значения dropout. В модели имеется 4 типа: `embedding dropout` применяется на embdedding'и перед отправкой их в первый слой Encoder или Decoder, `attention dropout` накладывается на веса внимания в MultiHeadAttention слое, `residual dropout` применяется на выходе из каждого подслоя (MultiHeadAttention или FeedForward) в слоях Encoder и Decoder и, наконец, `relu dropout` используется в FeedForward слое. Для всех 4-типов предлагается брать одно и то же значение и так перебрать 3 варианта: 0.1, 0.15, 0.2. Также предлагается попробовать 3 варианта пикового значения learning rate - `lr_peak` : 5e-4, 1e-3, 2e-3.

Учтите, если вы используете GPU, то обучение одной эпохи занимает около 1 минуты, при этом требуется до 1 GB видеопамяти. При использовании CPU скорость обучения замедляется примерно в 2 раза. Если возникнут проблемы с недостатком оперативной памяти/видеопамяти, уменьшайте batch size, но в таком случае оптимальная область значений learning rate изменится, и её требуется определять вновь. Для обучения модели с $batch\_size = 200$ потребуется от 300 эпох для достижения accuracy 0.66 на dev_small датасете.

3 Исследовательская часть

1. Предлагается реализовать дополнительный метод регуляризации - *label smoothing*. Для его использования иногда заменяют функции ошибки с CrossEntropy на Kullback-Leibler divergence, но здесь это не требуется. Теперь представим, что у нас есть на позиции t в последовательности токенов вектор предсказания из вероятностей для id каждого токена из словаря. Кросс-энтропия его сравнивает с ground truth one-hot представлением вида [0, ... 0, 1, 0, ..., 0]. А теперь представим, что мы немного "сгладили" значения в ground truth векторе и получили $[\alpha/|V|, ..., \alpha/|V|, 1 - \alpha + \alpha/|V|, \alpha/|V|, ... \alpha/|V|]$. α - параметр, вещественное значение от 0 до 1, $|V|$ - размер словаря - число компонент в ground truth векторе. Значения этого нового вектора всё ещё суммируются к 1. Считаем кросс-энтропию нашего вектора предсказаний и нового ground truth. Теперь, во-первых, cross-entropy никогда не достигнет 0-го значения, во-вторых, результат функции ошибки будет требовать от модели, как обычно, выдачи наибольшей по сравнению с другими компонентами вектора вероятности для правильного токена в словаре, но при этом не слишком большой, поскольку при приближении значения этой вероятности к 1 значении

функции ошибки увеличивается. С исследованиями применения label smoothing можно ознакомиться в статье [2].

Соответсвенно, чтобы встроить label smoothing в модель, необходимо провести описанное выше преобразование над ground truth векторами, а также реализовать подсчёт cross-entropy, поскольку используемый класс `torch.nn.CrossEntropy` не совсем подходит, так как для ground truth представления его `__call__` метод принимает id верного токена и строит one-hot вектор уже внутри. Тем не менее, можно реализовать требуемое на основе внутренней реализации этого класса CrossEntropyLoss.

Протестируйте различные значения α (Например, $0.05, 0.1, 0.2$). Опишите Ваши эксперименты и результаты.

Список литературы

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. ArXiv, abs/1706.03762, 2017.

[2] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? ArXiv, abs/1906.02629, 2019.