Имена: Хиндату Сани
ФН: 81948

Невронен машинен превод

Курсова работа по
„Търсене и извличане на информация. Приложение на дълбоко машинно обучение“
Зимен семестър 2022/2023

Описание на архитектурата

Моделът използва *transformer encoder-decoder* архитектура с позиционно кодиране на входа и изхода, представена в [1].

Влагане на думите

Големината на речниците задава входния размер на влагането. За разлика от [1], тук се използват две отделни влагания за български и английски. Размерността на влагането d_{model} е размерността на модела.

Позиционно кодиране

При *transformer* архитектурите се губи информацията за последователността на думите. Решението е за всяко векторно влагане на думи да се добави позиционен вектор с тази информация.

Позиционното кодиране би трябвало да е уникално за всяка позиция, да не зависи от дължината на изречението и да генерализира добре за дълги изречения. В класът `PositionalEncoding` се използва синусоидалното кодиране, описано в [1].

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

където pos е позицията на думата в изречението, а i е позицията на елемент във векторното ѝ влагане.

Многоглаво внимание

Класът `MultiHeadAttention` реализира многоглаво внимание с h глави и опция за маскиране, описано в [1]. Маскират се стойностите с $-\inf$. Маската M се отнася към изчисляването на скалираното скаларно внимание:

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

Многото глави позволяват на модела да отдели внимание към различни представяния. Вместо отделни едноглави внимания и конкатенацията им, се използват матрици $W_K \in^{d_k \times hd_{model}}, W_Q \in^{d_k \times hd_{model}}, W_V \in^{d_v \times hd_{model}}, W_O \in^{hd_v \times d_{model}}$, които се прилагат към входните данни $queries, keys, values$ (заявки, ключове, стойности).

Размерностите d_k и d_v са хипер параметри, но в моделът, както в [1], е наложено ограничението да са равни и двете на d_{model}/h.

Transformer блок

Класът `TransformerCell` реализира transformer блок. Един блок се състои от два подслоя - първият е многоглаво внимание, а вторият е невронна мрежа с *ReLU* активационна функция. Изходът на подслоевете е с размер d_{model}. Изходът от подслоя се събира с входа към него (към който се прилага $dropout$ *dropout*) и към този сбор се прилага нормализация на слой.

В извикването на многоглавото внимание се маскират позициите с думи за попълване. Многоглавото внимание се извиква с едни и същи ключове, стойности и заявки - вложените български думи.

Невронната мрежа е с един скрит слой с размерност d_{ff}.

Encoder

Encoder-ът, реализиран в класа `Encoder`, се състои от n на брой transformer блокове, като изходът на всеки е входът на следващия. Преди блоковете, на входните данни се прилага $dropout$ dropout слой.

Decoder блок

Класът `DecoderCell` реализира *decoder* блок - слой от маскирано многоглаво внимание с нормализация на слой след добавяне на остатъчна връзка и обикновен *transformer* блок.

В извикването на многоглавото маскирано внимание се маскират позициите с думи за попълване и допълнително се маскират думите по-нататък в изречението, за да не “преписва” моделът. Ключовете, стойностите и заявките са вложените английски думи.

А при извикването на многоглавите внимания в transformer блок частта, ключовете и заявките са изходът на *encoder*-а, а стойностите - изходът от предходния подслой.

Decoder

Decoder-ът се състои от n на брой decoder блокове, като изходът на всеки е входът на следващия. Преди блоковете, на входните данни се прилага $dropout$ *dropout* слой.

Обучение на модела

Използва се предоставения оптимизатор *Adam* и функцията за обучение в `run.py` с допълнителни параметри. Моделът се обучава приблизително 20 епохи с партиди от 64.

Крос-ентропията се изчислява с изглаждане на етикетите (*label smoothing*) $\epsilon_{ls} = 0.2$, описано в [3], и игнориране на думите за попълване. При изглаждането на етикетите вместо за действително разпределение да се използва

$$p(y|x, y_1, \ldots, y_{n-1}) = \begin{cases} 1 & y = y_n \\ 0 & y \neq y_n \end{cases}$$

се използва

$$p'(y|x, y_1, \ldots, y_{n-1}) = \begin{cases} 1 - \epsilon + \epsilon u(y) & y = y_n \\ \epsilon u(y) & y \neq y_n \end{cases}$$

където $u(y) = 1/K$ е фиксирано равномерно разпределение.

Настройка на параметрите

Transformer

Хипер параметрите са d_{model} (размерността на модела), h (брой глави на многоглавото внимание), n (брой блокове на encoder и decoder), d_{ff} (размера на скрития слой в НМ), $dropout$ (вероятността за dropout). Тези хиперпараметри са инициализирани според препоръките в [5]:

$$d_{model} = 512$$
$$h = 4$$
$$d_{ff} = 512$$
$$dropout = 0.2$$
$$n = 5$$

Оптимизатор

На *Adam* оптимизатора бяха добавени следните параметри както в [1]:

$$\beta_1 = 0.9$$
$$\beta_2 = 0.98$$
$$\epsilon = 10^{-9}$$

Добавени функции, файлове и промени

`transformer.py`

Съдържа класовете за компонентите на transformer-а:

- `PositionalEncoding` — позиционното кодиране, взет от [4]
- `MultiHeadAttention` — многоглавото внимание
- `TransformerCell` — transformer блок
- `DecoderCell` — decoder блок
- `Encoder`
- `Decoder`

`run.py`

Вместо извикването на модела да връща крос-ентропията, то връща предвидените индекси. Изчисляването на крос-ентропията става в `run.py` при трениране и във функцията `perplexity()` като думите за попълване се игнорират.

`model.py`

Функцията за превод `translateSentence()` е преместена извън класа на модела и приема модела като параметър. Тя връща низ, завършващ с нов ред.

Превод на изречения

Преводът се извършва от функцията `translateSentence()` във файла `model.py`. За намирането на най-добрия кандидат се използва търсене в лъч с ограничение β с евристика скоростта на крос-ентропия, описана в [2].

Използва се $\beta = 1$. Превод на корпус се извършва чрез командата

```
python run.py translate “corpus_to_translate” “result_corpus”
```

Резултати върху тестовия корпус

Резултатът върху тестовия корпус се съдържа във файла `result.en`.

Перплексията е 694.9. *BLEU* е 33.26.

Източници

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). *Attention is all you need.* Retrieved from http://arxiv.org/abs/1706.03762

[2] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). *Sequence to sequence learning with Neural Networks*. Retrieved from http://arxiv.org/abs/1409.3215

[3] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). *Rethinking the inception architecture for computer vision*. Retrieved from http://arxiv.org/abs/1512.00567

[4] *Language Modeling with nn.Transformer and TorchText — PyTorch Tutorials 1.13.1+cu117 documentation.* (n.d.). https://pytorch.org/tutorials/beginner/transformer_tutorial.html

[5] Araabi, A., & Monz, C. (2020). *Optimizing Transformer for low-resource neural machine translation*. Retrieved from http://arxiv.org/abs/2011.02266