Търсене и извличане на информация. Приложение на дълбоко машинно обучение

Зимен семестър 2021/2022

Курсов проект

Невронен машинен превод

Факултет по математика и информатика
специалност „Софтуерно инженерство“

Име, презиме, фамилия	Факултетен номер
Павел Светозаров Сарлов	62393

СЪДЪРЖАНИЕ

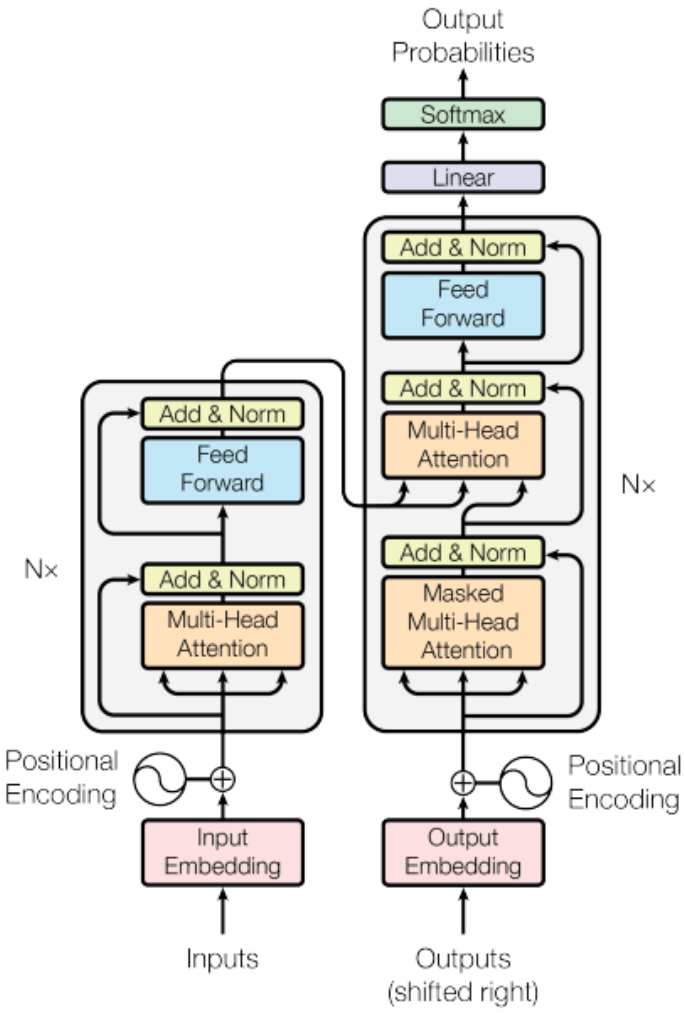
1. Въведение

Тещият документ е предназначен да опише моделът, имплементиран при разработката на курсов проект на тема „Невронен машинен превод“. Важно е да се отбележи, че кодът по проекта е в голяма част идентичен с предоставения такъв от [4], където е имплементирана архитектурата *Tranformer*, описана в [1], с някои минимални разлики. Тъй като това са сравнително нови и трудни за мен неща съм се фокусирал главно върху имплементирането на нещо, което да дава някакъв резултат, вместо да се стремя към креативност.

2. Архитектура

За реализиране на задачата бяха разгледани имплементации на невронен машинен превод с рекурентни невронни мрежи, конволюционни невронни мрежи, архитектура *Transformer* и т.н. От изброените спрямо време за обучение и резултат най-добре се справи архитектурата *Transformer*. В текущата секция накратко ще разгледаме отделните модули на самата архитектура.

Подобно на конволюционния модел, при *Tranformer* не присъства рекурентност. Също така не използва конволюционни слоеве. Вместо това е изграден изцяло от линейни слоеве,



Фигура 1: Схема на Transformer архитектура за машинен превод. (източник: [1])

механизми за „внимание“ и тяхната нормализация. Както и при другите архитектури, тук също имаме encoder-decoder структура.

2.1. Encoder

Encoder-ът приема вход поредица от думи, която първо преминава през стандартен слой за влагане, след което, тъй като липсва рекурентност и моделът няма как да знае подредбата на думите в поредицата, се използва и втори слой за позиционно влагане. За позиционното кодиране съм използвал размер от 1000 думи, което е зададено и като лимит при превод на изречение, т.е. моделът приема изречения с максимална дължина 1000 думи. Освен поредицата от думи, като вход се подава и маска, която на позициите където има *padToken* има 0 и 1 иначе. Тя се използва в механизма за „внимание“, който указва на модела да не обръща внимание на *padToken*-и, които сами по себе си не носят полезна информация.

Позиционните влагания и влаганията на думите се сумират, при което се получава вектор, съдържащ информация за самата дума и нейната позиция в изречението, като преди сумирането влаганията на думите се умножават по коефициент за скалиране $\sqrt{hidden}$, където *hidden* е размера на скрития вектор. Казват, че това би трябвало да намали дисперсията на влаганията и да улесни обучението на модела. [1]

След сумиране на влаганията и прилагане на *dropout* върху тях, резултатът се подава на *N*-те слоя на *Encoder*-а (*EncoderLayer*). За моя модел съм използва 3 слоя (в имплементацията на [1] използват 6 слоя, но при мен нещо моделът не се държеше адекватно при трениране с повече от 3 слоя, може би параметрите не бяха подходящи или просто му е нужно повече време, но тъй като съм притиснат откъм време и ресурси, реших да не се заигравам и го оставих с най-удовлетворителния резултат). *Encoder* слоевете са главната част от кодирането, при тях се прилага механизъм за „внимание“ (*multi-head attention*) върху входната поредица и позиционно пренасочване (*position-wise feedforward*) с прилагане на *dropout* след всеки от тях преди нормализация.

2.2. Decoder

Decoder-ът е подобен на *encoder*-а, с разликата че има два слоя за „внимание“: единия за изхода от *encoder*-а, другия за целовата поредица. Затова като вход приема изхода от *encoder*-а и целовата поредица, както и техните съответстващи маски. За маскирането за използвани функциите *make_src_mask* и *make_trg_mask*, предоставени от [4].

Тъй като обработката върху думите от целовата поредица става паралелно има нужда от метод за предотвратяване на „маменето“ от страна на *decoder*-а да гледа каква е следващата дума в целовата поредица и да я подава като изход. В това се изразява и самото маскиране на целовата поредица.

В слоевете на *decoder*-а няма въведение на нещо различно от споменатото при *encoder*-а (с изключение на гореупоменатите неща).

2.3. Multi-Head Attention

Механизмът за „внимание“ беше най-сложната част от цялата архитектура на *Transformer*. Смятам да не влизам в детайли за това как работи, хората, навлезли надълбоко в тези неща, вече са го направили и не мисля, че ще го обясня по-добре от тях. Кодът за модула е взет от [4]. В главния документ [1] и в *PyTorch* имплементацията [4] има информация за модула. Джей Аламмар предоставя доста добро описание на цялата архитектура и детайлно описание на *Multi-Head Attention* механизма в своя блог [3].

2.4. Position-wise Feedforward

Състои се от два линейни слоя, които преобразуват последното измерение на подадения тензор, т.е. прилагат се за всяка позиция от входната поредица, откъдето идва и самото име. Между двата слоя е използвана ReLU функция за активация и е приложен dropout. В модела слоят е представен като последователност от изброените стъпки. По-чиста имплементация би било просто да се отдели в друг модул, подобно на имплементацията в [4].

2.5. NMTmodel

Този модул обединява encoder-а и decoder-а в една обща структура. Приема за вход два листа от изречения (един на входния език, другия – на изходния) с големина *batch_size*. Двете партиди се допълват с *padToken*-и, след това се създават маските им, като на изходящата партида взимаме без последния елемент или *endToken*-а, тъй като очакваме моделът да го предвиди. Накрая данните се подават на *encoder*-а и *decoder*-а, резултатите се нагласят и се изчислява крос-ентропията, която се връща като резултат.

Модулът предоставя и интерфейс за превод на изречение. Нещото, което напълно съм имплементирал сам, е превод чрез търсене по метода на лъча. Това е включено като функционалност към вече предоставената *translate* такава, като след името на резултатния корпус се добави ключовата дума *beam*, т.е.:

- python run.py translate <sourceCorpus> <resultCorpus> beam

Интерфейса за превод на изречение предоставя възможността да се определи дали да се използва *greedy* или *beam* метод, както и колко да е широк лъча и колко да е наказанието *alpha* при нормиране на сумираните логаритми от условните вероятности. Тези стойности могат да се променят във файла с параметрите.

3. Обучение

Обучението на модела беше извършено в средата на *Google Colab*. За обучение на модела бяха използвани следните параметри:

- размер на скритите вектори (*enc_hid_size, dec_hid_size*) – за да може тренирането да се извърши с предостаните ни ресурси е използван размер 128;
- размер на позиционните вектори (*enc_posf_size, dec_posf_size*) – по същата логика като за скритите вектори е използван размер 256;
- *dropout* (*enc_dropout, dec_dropout*) – в началото 0.2, след което го промених на 0.1, тъй като дава по-добри резултати;
- слоеве за *encoder/decoder* (*enc_layers, dec_layers*) – в основния документ [1] са използвали 6 слоя, това значително увеличава времето за обучение, а и при мен моделът се държеше доста странно с повече от 3 слоя;
- глави на механизма за „внимание“ (*enc_heads, dec_heads*) – използвал съм броя, който е посочен и в главния документ [1];
- лимит на входната поредица (*limit*) – 1000;
- степен на обучение (*learning_rate*) – първоначално използвах 0.001 като загрявка на модела, след което намалих на 0.0005, тъй като в [4] споменават, че е добре да се използва по-ниска от степента по подразбиране на *Adam* оптимизатора;

Останалите параметри са оставени така, както са получени. Цялата продължителност на обучение на модела е около 25-30 епохи (тъй като *Colab* достигаше лимита си на няколко пъти по време на обучение съм изгубил точния брой).

4. Резултати

За ширина на лъча (*beam_width*) е използвана 3 и наказание (*alpha)* равно на 0.7. Преводът чрез този метод се оказа доста по-бавен спрямо *greedy* алтернативата (особено с по-голяма широчина). Като компенсация обаче добавя споменатите 1-2 точки *BLEU* резултат. Производителността може и да се дължи на лоша имплементация.

Резултатните перплексия и *BLEU score* за корпусите *dev* и *test* са представени в следната таблица:

Корпус	**Перплексия~**	***BLEU~ (greedy)***	***BLEU~ (beam)***
dev	4.54	38.44	39.66
test	4.91	36.42	37.92

5. Източници

[1] Attention is All you Need (acm.org)

[2] The Annotated Transformer (harvard.edu)

[3] The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. (jalammar.github.io)

[4] bentrevett/pytorch-seq2seq (github.com)