Отчет о проделанной работе в большом домашнем задании по Глубинному Обучению:

Выполнил:

студент группы БПМИ211 Черномордин Родион Романович

Москва 2023

1 Предисловие

в моем отчете я буду идти вместе с тобой по моему блокноту. Все решения я проводил на размере train выборки 0.5 от общего числа (если я же использовал полный датасет - я об этом напишу). Спасибо за твои советы к прошлому бдз, почти все их них я опробовал на этой задаче. Если в прошлой работе блистала моя неопытность, то в этой звездой шоу станет невнимательность. Весь код для финала я раскидал по файлам, как и обещал.

2 Решение

2.1 Первые шаги

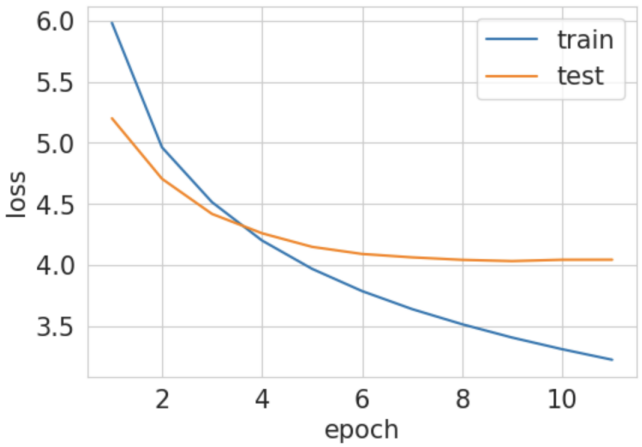
Не будет секретом, что все студенты использовали уже готовый трансформер с сайта pytorch [1], в котором большая и важная часть была реализована. Пропустим ту часть, где я пытался совместить самописный код с кодом с сайта. Важное стоит отметить, что дало мне большой буст - в самом начале я по привычке сделал Dataset для всех данных, в котором определил функции так, чтобы они дополняли все мои предложения отступами до максимальной возможной длины. Так во время генерации перевода с немецкого на английский для test выборки, я делал это по одному предложению (batch size = 1) и подавал на вход предложения + сколько-то клеточек отступа, модель ориентировалась на эту длину и выдавала везде дичь на BLEU=5-6. Когда я это заметил, то удалил эту механику, точнее я ее перенес в функцию Dataloader, внутри которого использовал дополнения паддингами для RNN.

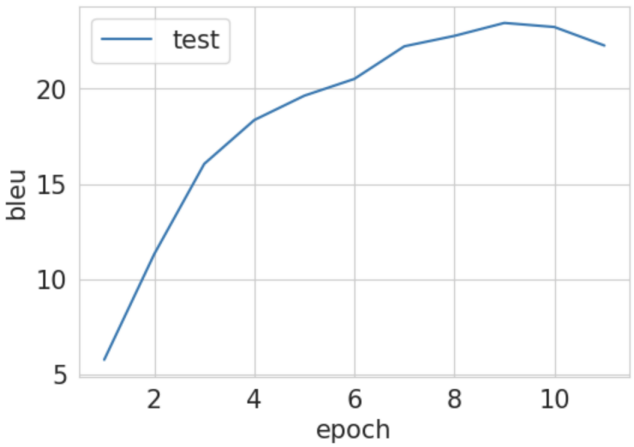
До этого я проводил небольшие тесты, модифицировал код (так можно будет увидеть у меня на графиках loss = 5 и в то же время loss = 0.03). Самое важное к чему я пришел за это время (и это же есть в статье на pytorch) - нужна инициализация весов, label smoothing = 0.1 (спасибо, что его посоветовал), в качестве метода сходимости я буду использовать Adam (так как и на лекции говорили, что с Трансформерами Adam или AdamW самые крутые), для него использовал гиперпараметры со статьи, а начальные параметры я использовал такие:

d_model = emb size	**количество enc слоев**	**количество dec слоев**
512	3	3

количество голов	**размер feedforward**	**p dropout**
8	512	0.1

Вот такое качество выдавало на половине train выборки:

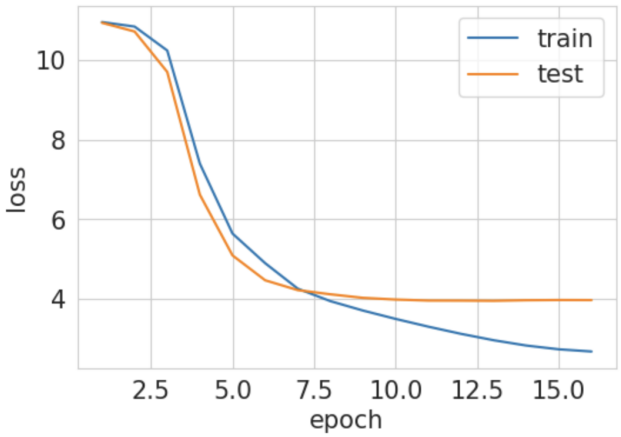


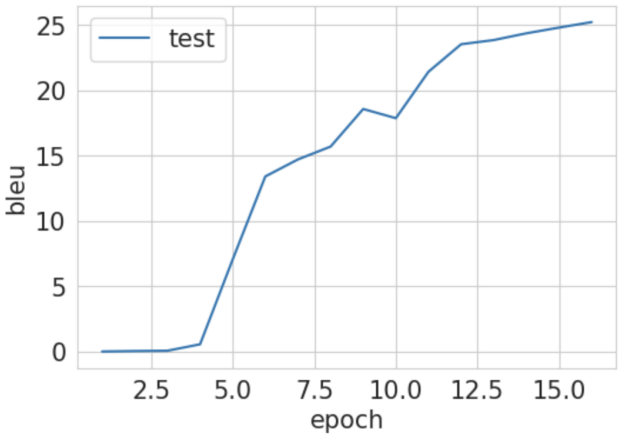



```
0.0 22.267500000000002
```

2.2 Scheduler

Я использовал до этого константу в качестве lr = 1e-4 (дано в статье), но хочется что-то другое. Да и на графике видно, что наша модель выходит на плато под конец, а одним из способов борьбы с ним - уменьшение lr. Вспомните им лекцию, откроем оригинальную статью (может и не прям оригинальная) [2], везде упоминается магический warmup, смысл которого сначала поднимать наш lr линейно, а потом как-то его бросать вниз. Чаще всего lr поднимают $\frac{4}{10}$ от всего числа эпох (так в оригинальной статье он поднимался 4000 эпох при 10000 обучающих). Первым делом я решил сам его себе написать отдельным классом, в котором я первые эпохи поднимаю линейно, а после опускаю с помощью косинусного шедулера. Перебрав параметры (сколько шагов должен подниматься и до какого значения), у меня получились такие результаты:

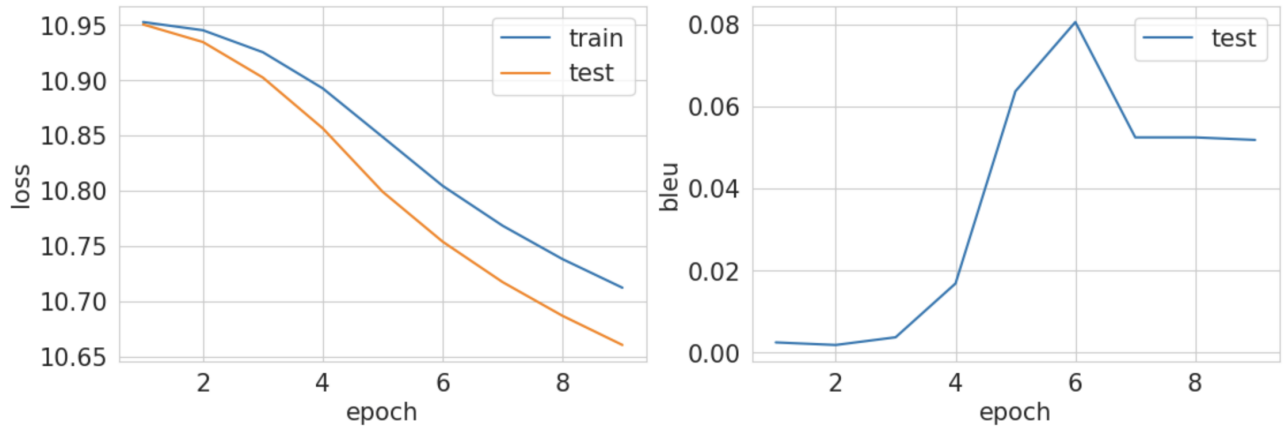





```
0.0 25.225625
```

Разница очевидна! И это только на половине эпох. Но все равно модель выходит на плато. Вторым вариантом я решил реализовать шедулер по формуле из статьи, где $lr =$

$d\_model^{-0.5} * min(epoch^{-0.5}, epoch * warm\_ep^{-1.5}) * 0.01$. В оригинале нет домножения на одну тысячную, это я сам сделал, так как показания lr с моим количеством эпох без него были слишком большие. Но результат оказался печальным:




```
0.0 0.05187500000000001
```

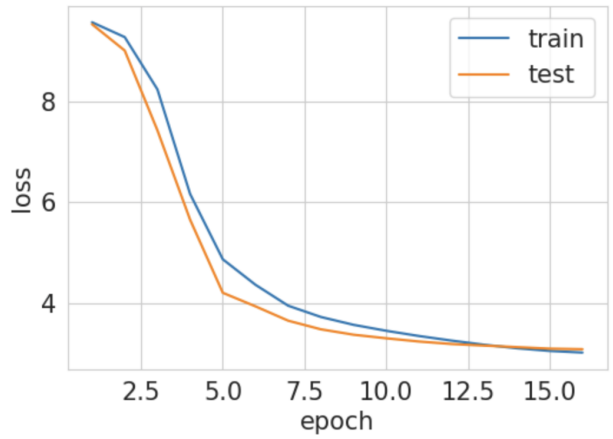
2.3 Вокабуляр

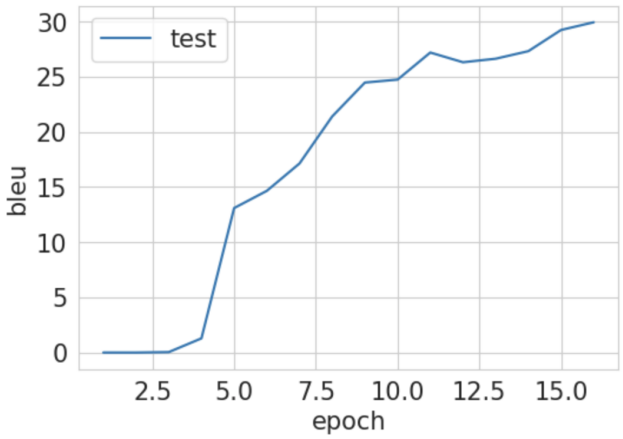
Давайте посмотрим, что у нас тут по словам. Особенно нас скорее интересует количество редких слов (сначала идет статистика по английским словам, потом по немецким):

```
В vocab у нас:
22684 слов, которые встерчаются в тексте 1 раз(а)
7832 слов, которые встерчаются в тексте 2 раз(а)
4321 слов, которые встерчаются в тексте 3 раз(а)
2900 слов, которые встерчаются в тексте 4 раз(а)
2042 слов, которые встерчаются в тексте 5 раз(а)
1552 слов, которые встерчаются в тексте 6 раз(а)
1195 слов, которые встерчаются в тексте 7 раз(а)
953 слов, которые встерчаются в тексте 8 раз(а)
813 слов, которые встерчаются в тексте 9 раз(а)
714 слов, которые встерчаются в тексте 10 раз(а)
А чаще всего слово встречалось 223874 раз
-------------------------------------------------
В vocab у нас:
68914 слов, которые встерчаются в тексте 1 раз(а)
16929 слов, которые встерчаются в тексте 2 раз(а)
8226 слов, которые встерчаются в тексте 3 раз(а)
4876 слов, которые встерчаются в тексте 4 раз(а)
3360 слов, которые встерчаются в тексте 5 раз(а)
2436 слов, которые встерчаются в тексте 6 раз(а)
1885 слов, которые встерчаются в тексте 7 раз(а)
1520 слов, которые встерчаются в тексте 8 раз(а)
1174 слов, которые встерчаются в тексте 9 раз(а)
1033 слов, которые встерчаются в тексте 10 раз(а)
А чаще всего слово встречалось 267010 раз
-------------------------------------------------
```

Мда, что-то их тут многовато. С одной стороны это хорошо иметь большой вокабуляр, но с другой стороны мы ничем его не сможем выучить, а их веса будут только нам мешать в вычислениях и тратить наши ресурсы. Но все же стоит с ними поиграть нам. Но вот есть проблема, я не адаптировал код под то, что у нас некоторые слова будут являться unk (значениями по умолчанию). Поэтому loss и (особенно) метрика bleu могут быть неверными в сравнении с другими версиями. Поэтому в качестве результатов теста я буду приводить показания из бота на тестовой выборке.

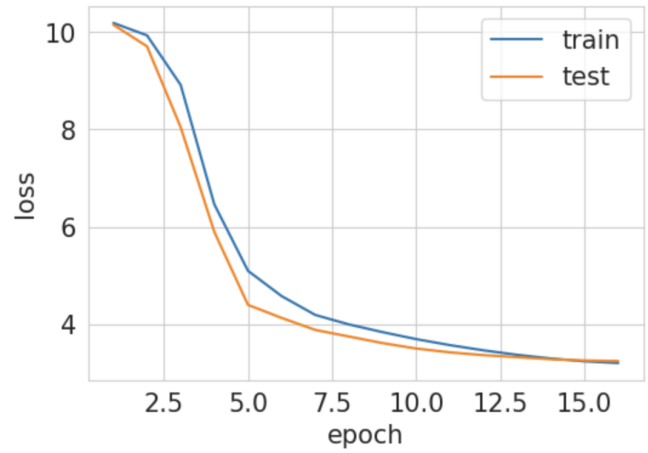
Подкинув кубик, я решил остановиться минимальном количестве слова в тексте = 8 (как в английском языке, так и в немецком). Работала чисто интуиция, да и соблазнило то, что слов на английском после этого порога становится меньше 1000. Результаты безумно обрадовали! Спойлер: это моя финальная модель.

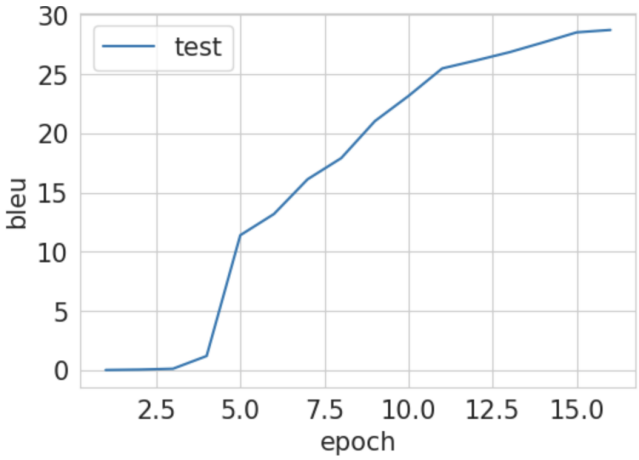





```
0.0 29.95
```

Но я продолжил эксперименты и решил выбрать en=3, de=5 (то есть, чтобы в каждом вокабуляре было около 26к слов. Результаты тоже был хорошим, но все же немного уступал предыдущему:






```
0.0 28.742499999999996
```

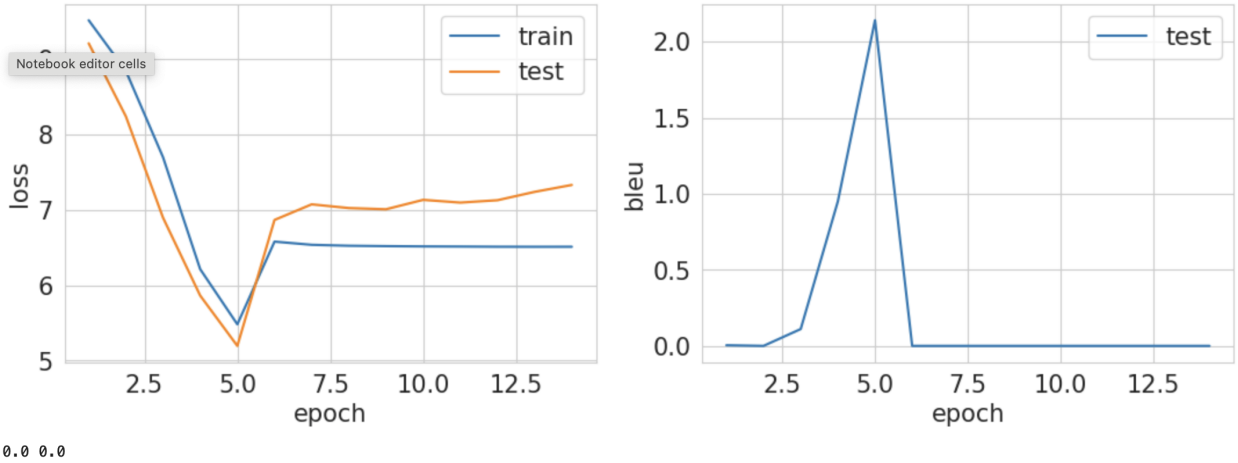
На переборе этого гиперпараметра я решил остановиться, так как сами по себе они на тестовой выборке давали качество bleu = 26. Очевидно, что уменьшая количество параметров (даже для редких слов), я не смогу так сильно улучшить модель. Сейчас я понимаю, что все же лучше было бы продолжить эксперимент и попытаться поднят порог отсева с 8 до 10 условно и так далее.

2.4 Прочие эксперименты

SWA: по твоему совету я решил использовать и этот способ улучшить модель, подгонял номера эпох, где мы должны были брать веса модели, но в любом случае это усреднение на выходе выдавало качество на val выборке на 4 меньше, чем значения оригинальной моде-

ли. Возможно, мне стоило бы дальше пускать модель обучаться, что loss более четко выходил на плато, тогда бы усреднение сработало, но хз хз хз.

Пробовал также я большие размеры у модели, но выдавалась вот такая фигня и становилось очень грустно:



Пробовал больше эпох, но это просто модель начинала больше переобучаться.

Хотел увеличить показания dropout до 0.2, но не могу найти результаты теста (помню, что стало хуже).

2.5 Решающее улучшение

До этого я генерировал новые слова, как и в домашнем задании 3 про анекдоты (также и в статье на pytroch). Но это слишком просто, поэтому я решил использовать beam search. В самом начале для оценки качества предложения я просто складывал вероятности слов. После до меня дошло, что это неправильно и нужно их нормировать относительно длины предложения. Также я играл немного с размером окна beam search. Я нашел еще одну статью про улучшения показаний моделей переводчиков для разных языков [3], там я еще раз убедился в надобности нормировке и в том, что 5-6 это оптимальное значение для задачи немецкий-английский (хотя мои эксперименты это тоже подтверждают). Вот такие результаты я получал на тестовой выборке в боте (цифра - размер окна):

bs, 5	**bs, norm, 5**	**bs, norm, 10**
25.6	26.25	26.09

Но всё равно, это так далеко от заветных 28. Было далеко, пока я не заметил это...

2.6 Фаталити, Невнимательность вин

Моя невнимательность меня скоро погубит. Я ничего не понимал, у всех моих друзей вышло 28 с просто базовой моделью из статьи pytorch вместе с beam search, у меня же по сути +- то же самое, но я нахожусь в низинке. Я решил просто проверить, что печатает мне выходная модель и был ошарашен! Оказывается, моя модель выдавала четко все слова, но после последнего никогда не печатала знак препинания (точку, вопросительный знак и тд) и сразу без пробела печатала
n. Не знаю способ оценки нас в боте, но я исправил все - пишу везде в конце нужный знак препинания вместе с переходом на новую строку, и... и мое решение взлетело на 28.42! Я расслабился и кайфанул.

3 Вывод

Тут явно мне хватило опыта с прошлого бдз, я четко понимал что и где нужно делать. Конечно, сейчас я смотрю на решение и понимаю, что вот здесь мог сделат опыт, вот здесь не делать и тд. Но это все не критично. Самое главное, что мне мешало на протяжении всего бдз - невнимательность, я тупо много ошибок делал из-за нее, но все же смог ее победить и закрыть публичный скор на 6! Для этой работы было убито: три кагл аккаунта, примерно 0 ресурсов на датасфере Я (тк мне тупо часами не давали ГПУ-шки), немного нервов, так как я любил ставить на ночь модели, просыпался и видел, что кагл принимал ислам и тупо крашился (модель переставала обучаться, а ошибок не выкидывало). В любом случае супер интересно было, супер интересный курс, пошел делать бонусы! (далее идет список литературы, что я нашел и использовал в решении или же просто читал)

Список литературы

[1] *PyTorch Transformer*. URL: https://pytorch.org/tutorials/beginner/translation_transformer.html.

[2] *Оригинальная статьяr*. URL: https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fPaper.pdf.

[3] *Статья про улучшение модели*. URL: https://aclanthology.org/W17-3204.pdf.