Отчет по домашнему заданию по KWS

Никита Степанов

19 ноября 2021 г.

Аннотация

В этом отчете я описываю свои эксперименты и размышления в хронологическом порядке. Здесь не очень много времени уделяется точным цифрам, графикам и сравнению моделей, все это можно найти в .ipynb ноутбуках в репозитории(см. README.md что где искать).

Baseline и streaming

Работу над домашним заданием я начал с того, что порефакторил и частично переписал код с семинарского ноутбука. После этого я реализовал streaming CRNN, демонстрация которого находится в файле streaming_demo.ipynb. Здесь особо не о чем говорить, в том смысле, что это вещи достаточно технические, и в них нужно было просто разобраться, а не проводить какие-то эксперименты. Но на это ушло довольно много времени, поэтому я решил это упомянуть :)

Дистилляция

Когда я начал работать над оптимизацией модели, первое, с чем я решил разобраться — это дистилляция, потому что я уже знал, что это такое, и что это не должно занять много усилий. Я прочитал оригинальную статью, и взял оттуда некоторые параметры, например, температуру выбрал равной 4. Выбор этот я сделал более менее случайно, но все-таки он немного обоснован тем, что в экспериментах на MNIST-е для очень маленьких student-моделей лучше всего работали температуры в диапазоне от 2.5 до 4. Также я взял из статьи параметр α, т.е. коэффициент, на который умножается hard loss. В экспериментах в speech recognition авторы использовали $\alpha = \frac{1}{2}$, и я взял такое же значение. Таким образом итоговая функция ошибки у меня считается по формуле $\mathcal{L} = \text{softloss} + \frac{1}{2}\,\text{hardloss}$. Больше я эти параметры не подбирал, потому что у меня и так получились достаточно хорошие результаты, но об этом позже.

После этого я начал подбирать параметры для student-модели. Я посмотрел на кол-во параметров в разных слоях изначальной модели и увидел, что большая часть находится в GRU, поэтому я решил в первом эксперименте просто уменьшить hidden size с 64 до 32. Уже это в принципе дало мне достаточно хорошие результаты по памяти: кол-во потребляемой памяти уменьшилось более чем в 3 раза. По скорости работы тоже наблюдалось заметное улучшение, хоть и не такое сильное, примерно в 2 раза. Качество на валидации при этом было достаточно хорошим: всего лишь 2.5×10^{-5}. Тут стоит отметить, что исходную модель я обучил на 1.6×10^{-5}.

Квантизация

После первого эксперимента я решил пока отложить эксперименты с дистилляцией, потому что она уже дала достаточно много. Я подумал, что если получится квантизовать модель в qint8, не сильно потеряв в качестве, то по памяти улучшение будет уже больше, чем в 10 раз, и по скорости тоже должен быть хороший прирост. И у меня действительно получилось хорошо квантизовать, используя dynamic quantization в GRU, attention и в классификаторе, но выигрыша по времени это дало довольно мало. Дело в том, что thop не умеет считать MACs для квантизованных слоев. Локально я замерял время выполнения с помощью %timeit, и ускорение было не особо сильным, что у бейзлайна, что у дистиллированной модели.

Дистилляция Part II

Я понял, что так просто ускорить модель в 10 раз не выйдет, и нужно либо писать pruning, либо подбирать параметры для дистилляции. Я выбрал второе, потому что код уже был написан, и я к тому времени провел всего лишь один эксперимент. Во втором эксперименте я просто уменьшил hidden size до 16, качество получилось примерно 4.9×10^{-5}. Это

уже почти впритык к порогу, поэтому может показаться странным, что я после этого продолжил уменьшать параметры student-модели. Но дело в том, что когда я в первый раз провел эксперимент, у меня, видимо из-за рандома, качество было получше, и выглядело так, как будто есть еще пространство для улучшения.

Параметры в 3-ем эксперименте я подбирал более аккуратно. Так как квантизация не особо ускорила мои модели, я задался целью получить ускорение в 10 только с помощью дистилляции. И подбирал я параметры следующим образом: вбивал какие-то комбинации параметров и запускал thop.profile на необученной модели. MACs не зависит от того, обучена ли модель, поэтому я мог заранее посмотреть какие комбинации параметров могут дать ускорение в 10 раз, не обучая при этом модель. Попробовав разные комбинации, я остановился на следующих изменениях: hidden size = 16, cnn out channels = 4, kernel size = (10, 20), stride = (5, 10). Модель с такими параметрами имеет как раз примерно в 10 раз меньше MACs, чем исходная, при этом такая конфигурация, на мой взгляд, выглядит довольно здраво и имеет шанс обучиться на приемлемое качество. Дальше я сделал дистилляцию с такими параметрами и обнаружил, что модель действительно обучается на приемлемое качество. Что интересно, в финальной версии ноутбука у этой модели качество равно $\approx 4.7 \times 10^{-5}$, т.е. лучше, чем у предыдущей модели. Думаю, это связано, с тем, что изменение kernel size и stride, возможно, вообще повлияло в лучшую сторону, хоть и параметров стало меньше, а уменьшение cnn out channels тоже, видимо, не особо мешает модели обучаться. Но так или иначе, я с помощью одной лишь дистилляции получил модель, которая более чем в 10 раз ускоряет исходную, и по памяти оптимизирует тоже более чем в 10 раз. Но так как я до этого уже немного поработал с квантизацией и разобрался, как она делается, я решил еще применить и ее ко всем этим моделям. Если кратко резюмировать, то потребление памяти упало очень сильно, время работы чуть-чуть улучшилось и метрики чуть ухудшились, графики и более подробный анализ можно найти в comparison.ipynb. Но в итоге моя лучшая модель ускорила baseline примерно в 10 раз и уменьшила потребление памяти в 38.

Трудности, с которыми столкнулся

Если так подумать, то часть с ускорением модели мне далась довольно легко. Я тем не менее потратил немало времени на то, чтобы разобраться получше с дистилляцией и квантизацией. Еще я в очередной раз напоролся на то, что в .ipynb ноутбуках очень легко накосячить с воспроизводимостью, потому что можно позапускать ячейки в каком-то порядке, получить хороший результат, забыть, в каком порядке запускались ячейки, а потом выясняется, что если прогнать ноутбук с начала, то результаты не воспроизводятся. После этого домашнего задания я серьезно призадумался о целесообразности их использования, в случае если в них выполняются какие-то тяжелые операции.

В итоге получается, что основная часть работы в моем случае была заключена в реализации стриминга и в переписывании кода. В стриминге сложность заключалась в том, что нужно было разобраться, как там все пересчитывать, и аккуратно все написать, нигде не набагав. Написание template-а и рефакторинг заняли так много времени, пожалуй, потому что я никогда с нуля не писал проекты на pytorch-е, поэтому приходилось многие моменты обдумывать, как все это хорошо и красиво сделать. Но это вообще было необязательно делать, поэтому вряд ли это можно отнести прямо к трудностям, скорее мне самому хотелось потренироваться в подобных вещах.

Если подытожить, то можно сказать, что каких-то больших сложностей я не испытывал во время выполнения домашнего задания, но вся работа вместе взятая все равно заняла довольно много времени. Возможно, у меня все так легко пошло с ускорением модели, потому что сгенерировалось удачное разделение на train и val. Базовая модель обучилась на 1.6×10^{-5} вместо положенных 5×10^{-5} явно неспроста. Возможно, тут помогла магия torch.manual_seed(3407), но на войне все средства хороши :) Еще раз замечу, что код, генерирующий мое разделение, находится в train_val_split.py, и его результат воспроизводится.