Общие результаты:

Был реализован алгоритм LAS для обучения модели распознавания речи(ASR). Удалось достичь метрик 0.2 CER и 0.38 WER на тестовом датасете (LibriSpeech: test-clean),

```
test_clean_CER_(Argmax): 0.20453171065767414
test_clean_WER_(Argmax): 0.3752777800030405
test_other_CER_(Argmax): 0.3650813055022221
test_other_WER_(Argmax): 0.6291003369033128
```

что соответствует 5 баллам.

Так же были выполнены следующие бонусные пункты:

1. Реализована LAS модель (+3)
2. Реализован beam search для LAS (+ 3)
3. Использовался BPE токенайзер (+1)

Воспроизведение результатов:

0. Установить все необходимые пакеты

```
pip install -r requirements.txt
```

1. Запустить скрипт обучения токенайзера:

```
python train_tokenizer.py
```

2. Запустить первый скрипт обучения (конфиг las):

```
python train.py
```

3. Запустить скрипт обучения с другими параметрами (конфиг las_continue):

```
python train.py
```

4. Запустить скрипт инференса с необходимыми параметрами

Архитектура модели:

Вдохновлялся статьей SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition, однако авторы не захотели раскрывать детали имплементации модели и вместо этого сослались на другую статью:
Model Unit Exploration for Sequence-to-Sequence Speech Recognition в которой авторы были более благосклонны и почти предоставили подробное описание архитектуры, однако все же сослались на третью статью где я и почерпнул последние недостающие детали Very deep convolutional networks for end-to-end speech recognition

Модель состоит из следующих частей:
0) Модель ожидает спектрограмму с spec_dim = 80

1. Listener состоит из 2 частей:
    1. 2 слоя Conv2d со страйдом 2(нужен для сокращения размерности), число каналов 32(входные после преобразования спектрограммы - 40)
    2. 3 слоя двусторонней LSTM с hidden_dim=256

2. Attention: здесь все просто - классический attention из оригинальной статьи LAS(product attention)
3. Decoder представляет собой 2 слоя односторонней LSTM, скрытое состояние первого слоя передается в аттеншн и результат добавляется к эмбеддингу предыдущего токена.

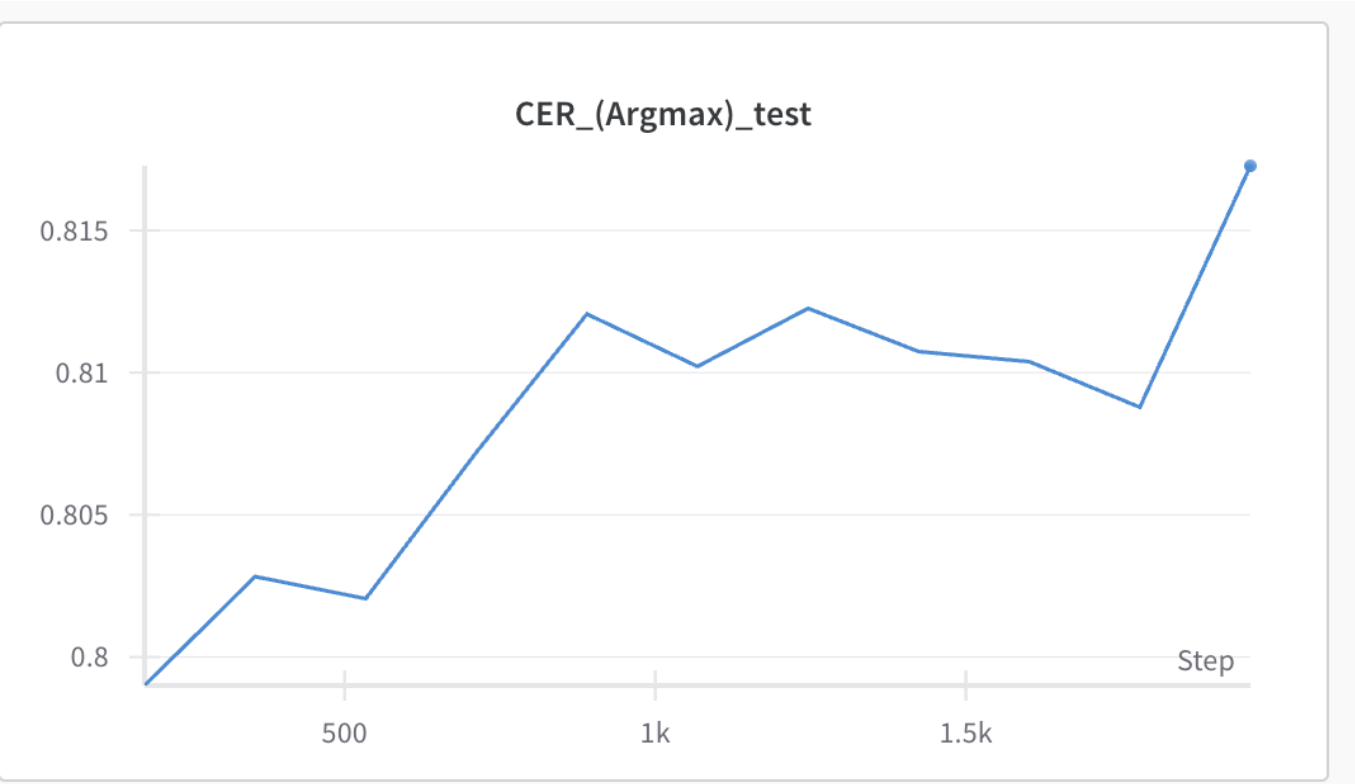
Стоит сказать, что собранная мной модель является некоторой комбинацией идей из статей перечисленных выше - архитектура аналогично моделям из SpecAugment, однако я уменьшил количество параметров в итоге получилось что-то похоже на small модель из второй статьи. При этом часть параметров модели мне так и не удалось найти(например размер эмбеддинга, так что пришлось выкручиваться).

Особенности обучения:

Функция потерь классический логарифм вероятности правильного ответа(в нашем случае токена), однако из-за того что модель это LAS мы предсказываем токен не для конкретного момента времени из аудиозаписи, а скорее работаем как генеративная LLM модель. По этому использовать ctc декодер нельзя и как следствие мы всегда предсказываем результат сразу для всей аудиозаписи.
Однако т.к в начале наша модель будет очень плохо предсказывать токены, то необходимо гайдить её - подсказывать правильный токен. Работает это следующим образом: во время обучения с вероятностью tf_rate(0.9 или 0.7 в зависимости от фазы обучания) мы подставляем следующий токен из правильной транскрипции записи.
Без этого модель не обучается совсем.

В итоге я не использовал никакие аугментации для финального обучения - LAS достаточно тяжело учиться и сам по себе, добавление аугментаций еще утяжелило бы этот процесс(собственно у меня не получилось добиться нормального обучения). Если верить первой статье добавление аугментаций действительно улучшает качество,

однако только в очень долгосрочной перспективе(они обучались неделю на 32 Google Cloud TPU, у меня таких ресурсов не было), по этому сосредоточился на более скромном сетапе.
Пример запуска с аугментациями:
https://wandb.ai/torchrik/pytorch_template_asr_example/runs/ey8sr6dd?nw=nwusertorchrik



Также проблемой были проблемой с градиентом - в середине обучения он по норме достаточно быстро растет и из-за этого обучения почти останавливается, в одной из статей упоминался clipping градиента, собственно им я и воспользовался, что позволило сильно улучшить качество.

Первая часть обучения(еще без клиппинга):
https://wandb.ai/torchrik/pytorch_template_asr_example/runs/7ql8d6n3?nw=nwusertorchrik

Вторая часть обучения(уже с клипингом)
https://wandb.ai/torchrik/pytorch_template_asr_example/runs/9gamsbnc?nw=nwusertorchrik
(ретроспективно первые две части можно склеить в одну)
И финальная часть(здесь применяется более слабый гайдинг, что повышает качество генерации):
https://wandb.ai/torchrik/pytorch_template_asr_example/runs/o2fpk7vd?nw=nwusertorchrik

BPE tokenizer:
Честно говоря я не пробовал обычный токенайзер, т.к кажется для LAS он работал бы хуже. Размер словаря подбирал просто - посмотрел на тот, который использовался в первой статье(1000) взял в 10 раз меньше(100) потому что модель тоже была в 10 раз меньше) Но было бы интересно поэксперементировать с другими датасетами тоже.

Beam search

Beam search позволил достаточно сильно улучшить качество модели, реализация есть на гитхабе, здесь лишь хочу отметить, что я встроил его в модель, т.к из-за особенностей архитектуры в beam search надо перезаписывать hidden_state decoder.

Основная сложность

Самым сложным было понять архитектуру LAS, на это ушло первые пару дней. Действительно довольно много нетривиальных моментов на мой взгляд, которые слабо поясняются в статьях.

Дальнейшие улучшения:

Я уверен, что из этой архитектуры можно выбить гораздо лучший результат, однако у меня не хватило времени на эксперемнты(
Точно хорошей идеей является добавить LM для инференса(хотя мне кажется, что здесь прирост в качестве будет не таким сильным как для классических архитектур с ctc, т.к наша модель уже своего рода "языковая модель").
Так же было бы интересно обучить большую модель с аугментациями, на это просто не хватило времени.