Отчет по большому домашнему заданию 2.

Тимофей Грицаев @tgritsaev

Реализация моделей лежит в src/models, реализации обучения лежат в src/trainers (можно было объединить пайплайны в единый, но я решил не заниматься этим). В конечном счете я обучал CVAE и DCGAN, по ним я и продемонстрирую результаты.

DCGAN

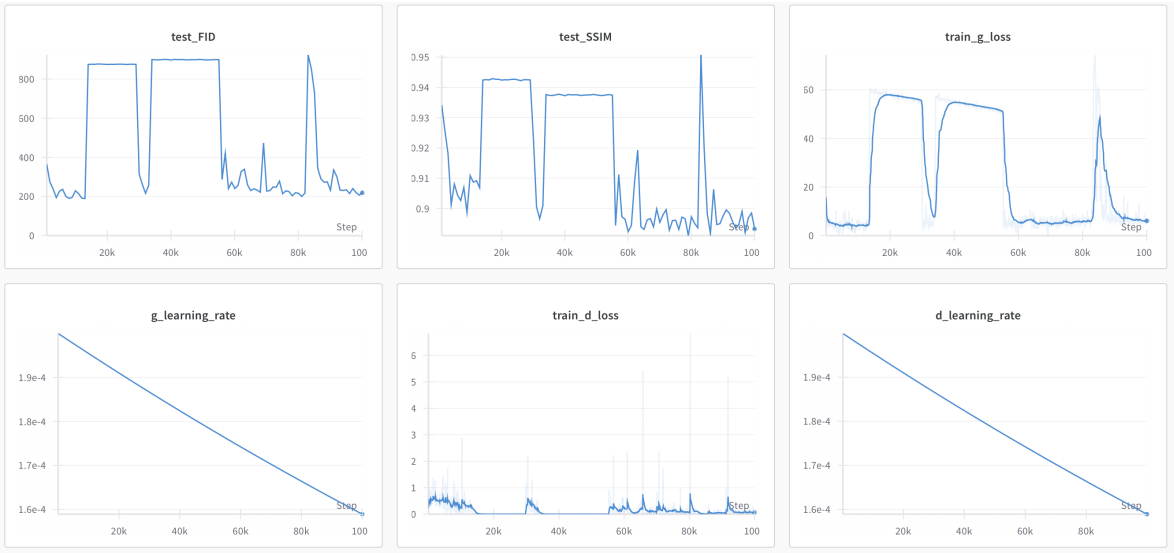
1. Так как я сперва начал с CVAE и горел желанием запустить все три модели, то DCGAN мне пришлось встраивать в свой шаблон, что повлекло за собой ошибки. Ключевая ошибка, которую я долго искал - это нормализация, я ее не делал. Требуется же сперва нормировать изображение к $[0; 1]$, после чего отобразить в промежуток с $[-1; 1]$.

2. При реализации я пользовался pytorch-tutorial:

   - Датасет. Я использовал датасет с котами, реализация в src/datasets/cats_faces.py. Этот датасет сложнее чем предложенный с аниме девочками, поскольку коты имеют более сложную текстуру (например, шерсть) и обьекты из реального мира, а аниме девочки подчиняются каким-то более простым правилам.
   - Обучение. Основа скрипта реализована мной самим, здесь я взял из туториала train_epoch. Реализация в src/trainers/gan_trainer.py.
   - Модель. Полностью взята из туториала. Модель очень простая, генератор состоит из ConvTranspose - BatchNorm - Activation, дискриминатор такой же, только вместо ConvTranspose - Conv. Реализация в src/models/dcgan/dcgan.py.

3. Эксперименты. Для просмотра промежуточных графиков предлагаю обратиться к wandb-проекту, я прикреплю лишь финальный. До запуска ‘all-bug-fixed-[normalization]-wd=0‘ были ошибки, так что с точки зрения экспериментов следует смотреть, начиная отсюда. Опишу осмысленные эксперименты и финальный:

   - ‘all-bug-fixed-[normalization]-wd=0‘ - первый без багов и финальный, после него были эксперименты, но неудачные.



Гифку обучения и картинку с финальными результатами я положил на GitHub. Здесь мы видим, как легко может колапсировать модель, соотвественно каждый период взлета наверх объяснен тем, что генератор попал в моду и генерирует странные шумные одинаковые изображения.

- ‘beta1=0.7‘ вместе beta1=0.5 - стало хуже.
- ‘lower-lr‘ пытался сделать более резкое убывание lr - стало хуже.
- ‘wd=1e-6‘ - стало хуже чем без wd.

4. Параметры взяты из того же туториала. Найти их можно в configs/dcgan_train.json.

```
"generator": {
    "type": "Generator",
    "args": {
        "latent_dim": 100,
        "hidden_dim": 64,
        "n_channels": 3
    }
},
"discriminator": {
    "type": "Discriminator",
    "args": {
        "n_channels": 3,
        "hidden_dim": 64
    }
},
"generator_optimizer": {
    "lr": 2e-4,
    "betas": [0.5, 0.999],
    "weight_decay": 0
},
"discriminator_optimizer": {
    "lr": 2e-4,
    "betas": [0.5, 0.999],
    "weight_decay": 0
},
"generator_lr_scheduler": {
    "gamma": 0.9999977
},
"discriminator_lr_scheduler": {
    "gamma": 0.9999977
},
"trainer": {
    "epochs": 100,
    "iterations_per_epoch": 1000,
    "save_period": 5,
    "log_every_step": 100
},
"wandb": {
    "project": "dl2-gan-generation"
}
```

Таким образом, lr почти константый (так расписание экспоненциальное, а gamma близка к 1), учится модель всего за 4-5 часов на P100 Kaggle-а.

5. Выводы:

- Как всегда, отклоняться от параметров статьи в GAN-ах очень тяжело. Почти любое изменение приводит к заметно худшему результату.
- Mode-colapse - неприятная штука, думаю, с этим можно бороться прибавляя случайный шум к весам.
- До этого я один раз обучал GAN, и там стоял аномально большой wd=1e-2, я подумал, что в GAN-ах классическая практика ставить большой wd, однако, как показало это домашнее задание, не всегда.
- Внимательнее читаем опорный материал, не забываем про нормализацию.
- FID и SSIM хоть и не отображают точь-в-точь результаты, показали себя хорошо, четко реагируя, когда генерация совсем плохая. При этом на хороших генерациях и отличных значениях метрик лучше смотреть глазами. Loss генератора тоже неплохо себя показывает, хотя опять же не является метрикой, так как вовсе зависит от дискриминатора.

- Все же для дешевого решения задачи генерации изображений – GAN-ы лучший вариант на мой взгляд, так как VAE тоже не просто завести, но оно замыливает картинки (хотя не колапсирует и может покрыть более разнообразный датасет, но разнообразный датасет – это уже про сложную задачу). Классический DDPM довольно дорогой в обучении и генерации и может быть вычислительно дороже GAN-ов в 20-30 раз.
- В целом, результаты генерации неплохие, местами видны артефакты, я пытался провести эксперимент на 200 тысяч итераций, чтобы получше обучить, но wd=1e-6 не дал выйти из mode-colapse. Так же можно было бы запускать более сложную модель, DCGAN действительно примитивная модель.

CVAE

Здесь я позволю себе не соблюсти часть требований, поскольку все их выполнил в части про DCGAN, будем считать это бонусом.

1. Ради интереса стал учить conditional задачу, пришлось поискать датасет, потому что не так и много открытых обусловленных датасетов для генерации. Я взял датасет ArtBench-10, он состоит из 10 классов картин, каждый класс наполнен 1000 картинами. У датасета есть две версии в 32x32 и 256x256, в итоге я запустил 256x256 и 256x256, который resize-ил до 64x64. Это довольно сложный датасет, потому что здесь 10000 обьектов (для сравнения котов 15000), и на картинах изображены множество разных обьектов, трудно найти между двумя картинами одно правило, в то время как датасет котов подчиняется более простым правилам, соответственно генерировать котов намного проще. Но найти простой и интересный датасет с метками я не смог.

2. Трудности. Сперва я затупил с коэффициентами в loss-е, и просто складывал kld_loss с reconstruction_loss, прошло немало времени, пока я понял, что нужно брать маленький коэффициент для kld_loss порядка 0.00025. Еще был тупой баг, когда я понял, что нужен коэффициент, и стал умножать reconstruction_loss на 1/0.00025, из-за BatchNorm-ы это, конечно, плохая затея. Ну, и на все датасеты добавил нормализацию кроме MNIST-а, поэтому думал, что модель некорректно реализована, когда добавил нормализацию на MNIST понял, что модель корректна (пусть и обладает невысокой способностью к генерации), а просто ArtBench для обучения очень сложный.

3. В итоге результаты на MNIST-е приличные, можно посмотреть gif-ки реконструкции и генерации на GitHub-е. Графики можно посмотреть в прикрепленном wandb.

4. Пожалуй, было бы интересно посмотреть на работу CVAE на более простом датасете, условно, разных типов котов. Кстати, обуславливание - это то, что плохо получается у GAN-ов, хуже чем у VAE и DDPM, поскольку в последних заложена красивая математика, позволяющая легко добавить обуславливание, а в GAN-ах придется инженерно повозиться.