Глубинное обучение

Организация DL-экспериментов

Максим Рябинин
Старший исследователь, Yandex Research

Программа ML Residency: yandex.ru/yaintern/research_ml_residency

ФКН ВШЭ, 2023

Проблемы при обучении нейросетей

- «Neural net training is a leaky abstraction» — Andrej Karpathy [1]
- Знания теории, архитектур, оптимизаторов порой недостаточно для получения хорошей модели
- No free lunch: универсального наилучшего решения не бывает
- Важно правильно организовать проведение экспериментов

[1] karpathy.github.io/2019/04/25/recipe

Перед началом

- Используйте проверенные временем стандарты
- Вместо своих моделей — архитектуры из популярных публикаций (ResNet в зрении, ELMo/Transformer в текстах) и репозиториев [1,2,3,4]
- Adam со стандартным LR без расписания* обойти нелегко
  *иногда — SGD с инерцией, иногда важно расписание
- Сложные функции потерь/аугментации лучше отложить
- Первые запуски на небольших датасетах, подвыборке или синтетике

[1] github.com/pytorch/vision
[2] github.com/pytorch/fairseq
[3] github.com/huggingface/transformers
[4] github.com/rwightman/pytorch-image-models

От частного к общему

- Чтобы проще находить ошибки, снизьте число факторов влияния
  Баги могут быть в модели, в обучении, в проверке качества и даже в загрузке данных!
- На меньших масштабах можно быстрее итерироваться
- Визуализируйте **всё**, что можете: метрики, примеры работы модели
- DL-код — всё ещё код: полезно писать (как минимум) unit-тесты

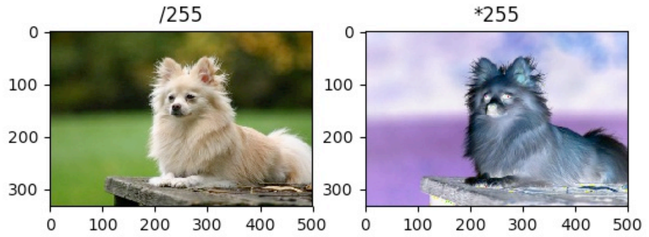
1. Переобучение на одном батче
2. Переобучение (или хотя бы сходимость) на обучающей выборке
3. Адекватное качество на валидации

Типичные ошибки: модели

- Использование ad-hoc архитектур, когда это не требуется
  Здесь же: использование очень больших моделей в первом же эксперименте
- Использование нестабильных/сложных функций потерь
  Здесь же: softmax->NLLLoss, sigmoid вместо softmax, активации перед softmax
- Плохая инициализация: нули/константы вместо Glorot/He/uniform
  Rule of thumb: для кросс-энтропии $\approx \log(K)$ на старте

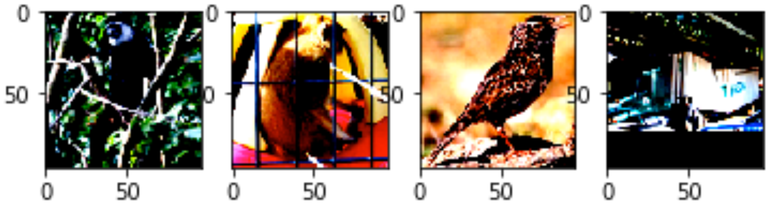
Типичные ошибки: данные

- Отсутствие аугментаций
- Использование некорректных аугментаций/ препроцессинга
  Важно: визуализируйте данные непосредственно на входе в сеть!



- Если используете предобученные модели, препроцессинг должен быть максимально похожим

- Разные* аугментации при обучении и валидации
  *Исключение — random crop и сильные деформации без TTA



- Считывать весь датасет сразу

  Используйте torch.utils.data.Dataset/DataLoader

Типичные ошибки: обучение

- Не забывайте делать zero_grad :)
- model.train()/model.eval() в нужных фазах обучения

  anecdote: иногда стоит обновлять статистики batchnorm на тестсете (и только их)
- loss.item()* перед сохранением значения между итерациями

  *вообще можно и loss.detach(), чтобы избежать CPU<->GPU-синхронизации
- Если сохраняете чекпойнты для дообучения, сохраняйте также параметры оптимизатора (optim.state_dict()) и расписания
- torch.cuda.manual_seed_all не даёт 100% воспроизводимости!

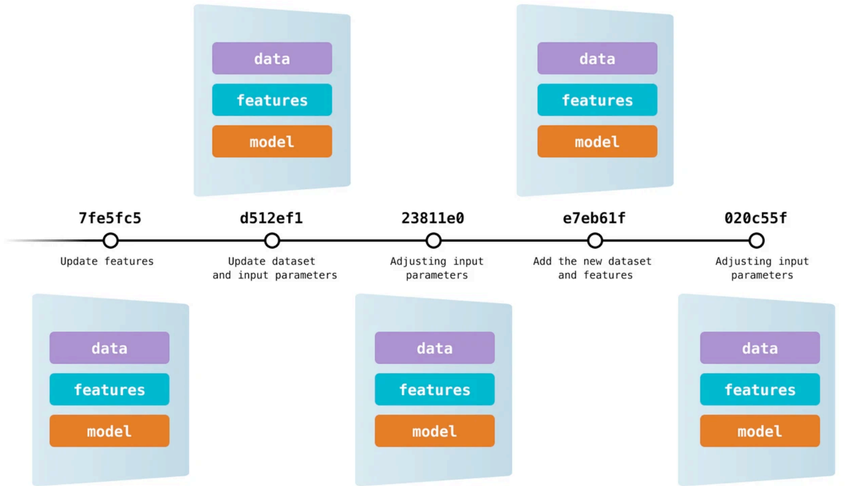
Проведение экспериментов

- Важно вносить только одно изменение за раз
- На ранних стадиях необязательно учить до сходимости
- Если прирост на уровне шума, подумайте, стоит ли изменение того

Организация экспериментов

- Ведите лог всех экспериментов
- В идеале нужно версионировать и модель, и код, и конфигурацию
- Каждый эксперимент — отдельная ветка в Git
- Есть ряд готовых инструментов [1,2,3,4,5,6]

ID	v	trn_loss	val_score	img_size	chans	morph	ssr	backbone	en_dim	de_dim	at_dim	em_dim	epochs
BRISTOL-66	7	0.0674836	6.7402	224	3	True	[0.01, 0.01, 10]	tf_efficientnet_b4_ns	1792	512	256	256	10
BRISTOL-71	6	0.0292068	3.16898	224	3	True	[0.01, 0.01, 10]	tf_efficientnet_b5_ns	2048	512	256	256	10
BRISTOL-65	6	0.0383978	3.78346	224	3	True	[0.01, 0.01, 10]	tf_efficientnet_b5_ns	2048	512	256	256	10
BRISTOL-64	5	0.0444499	5.13568	224	3	True	[0.05, 0.05, 15]	resnet101d	2048	512	256	256	10
BRISTOL-63	4	0.026509	3.81267	224	3	True	[0.01, 0.01, 10]	resnet101d	2048	512	256	256	10



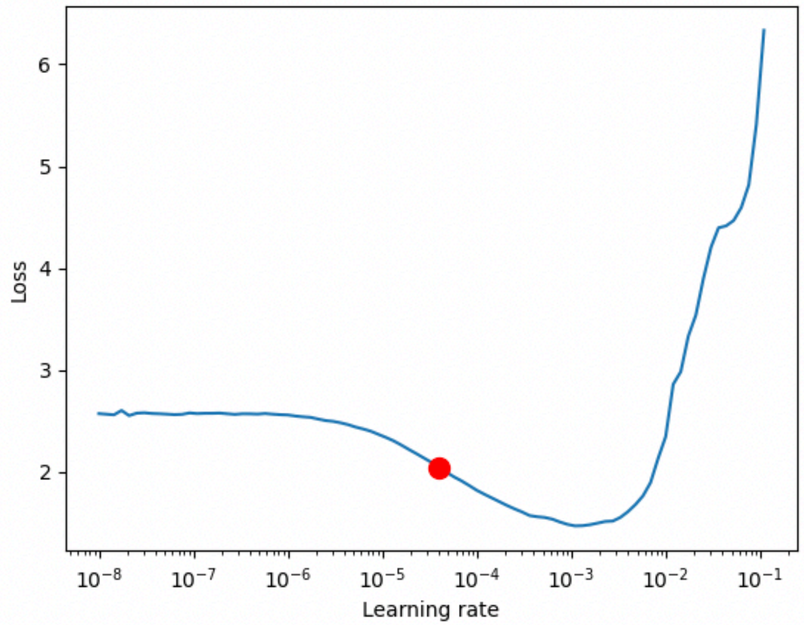
[1] wandb.com
[2] comet.ml
[3] neptune.ai
[4] dvc.org
[5] mlflow.org
[6] clear.ml

Как улучшать качество?

- Работа с данными (количество, качество, обработка) зачастую приносит гораздо больше эффекта
- Функция потерь должна быть максимально близка к метрике
- Архитектуры влияют существенно, но учитывайте свои ресурсы
- Размер батча важен (ряд моделей иначе просто не учится)
  hint: используйте аккумуляцию градиентов!
- Оптимизация гиперпараметров в самую последнюю очередь!
  (random search > hyperparameter tuning > grid search)

LR range test aka LRFinder

- Эвристика для подбора learning rate [1,2,3]
- Увеличиваем learning rate после каждого батча, остановка перед взрывом значений loss-функции
- Реализовано в off-the-shelf DL-фреймворках [4,5,6], но несложно реализовать самому [1]
- Аналогично можно искать оптимальное значение для momentum



[1] sgugger.github.io/how-do-you-find-a-good-learning-rate.html
[2] arxiv.org/abs/1506.01186
[3] arxiv.org/abs/1803.09820

[4] catalyst.dl.callbacks.scheduler.LRFinder
[5] fastai.Learner.lr_find
[6] pytorch_lightning.tuner.lr_finder.lr_find

Выводы

- Пользуйтесь проверенными техниками и опытом других людей
- Начните с небольших экспериментов
- Когда всё протестировано, можно масштабироваться
- Отслеживайте все доступные метрики
- Тестируйте одно изменение за раз
- Сохраняйте код/конфигурацию всех экспериментов и их результаты