Отчет по лабораторной работе №3

Метод стохастического градиентного спуска и его модификации

Аксенова Валерия, Коваленко Александр, Шустров Андрей

Описание метода:

Для задачи многомерной регрессии был реализован базовый алгоритм стохастического градиентного спуска (SGD).

Интерфейс позволяет настраивать следующие гиперпараметры:

- Тип регуляризации (L1, L2 или Elastic),

- Стратегию выбора шага (const, step/time/exp decay)

- Seed для генератора случайных чисел для гарантированной воспроизводимости результатов.

- Классические learning rate, размер батча, количество эпох обучения.

Архитектура модуля спроектирована таким образом, чтобы автоматически собирать ключевые метрики качества модели на каждом шаге обучения и по итогам эпохи.

Для того чтобы подчеркнуть чувствительность алгоритма к настройке гиперпараметров, в реализацию не были включены дополнительные оптимизации.

Библиотечные методы:

- SGD

- SGD+Momentum

- SGD+Nesterov

- Adagrad

- RMSprop

- Adam

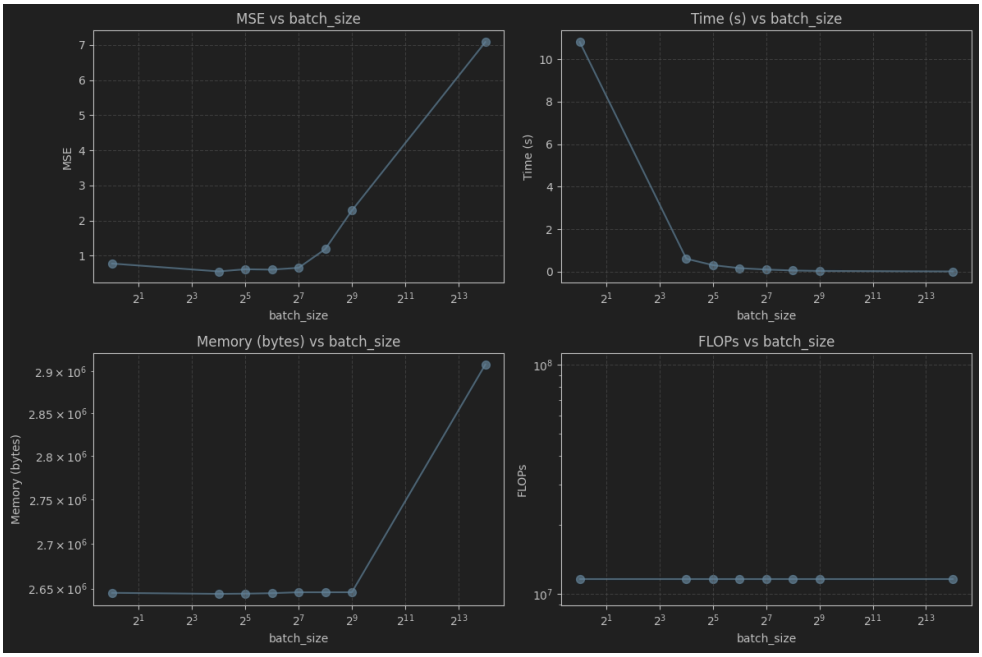
Сравнивались реализации из Keras и PyTorch

Модификация: метод неявного стохастического градиентного спуска (*Implicit SGD*).

Исследование:

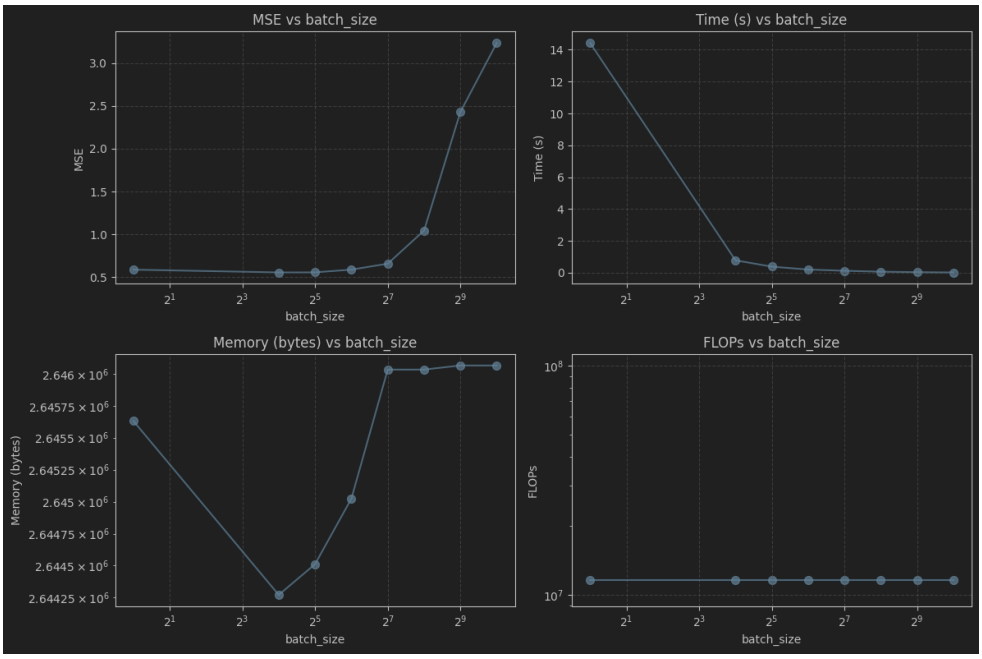
В качестве датасета был выбран *California Housing* из *scikit-learn*. Датасет состоит из 20680 образцов и 8 признаков.

Разный размер батча:

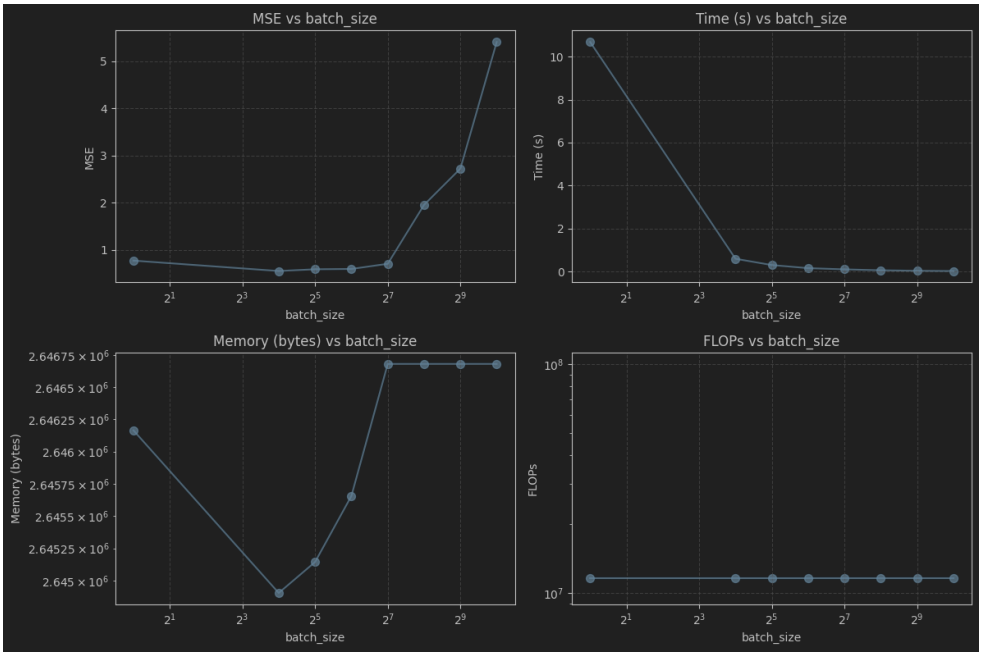


При переборе размера батча от 1 до полного объёма данных мы получен типовой для SGD результат: очень маленькие батчи дают относительно низкую ошибку и хорошее обобщение, но одноэпохальное обучение занимает много времени. С ростом батча время эпохи стремительно падает и выходит на плато уже при величинах порядка 128, причём суммарные FLOPs остаются практически неизменными, а потребление памяти стабильно до очень больших батчей (до ~2048), после чего начинает расти; однако при слишком крупных батчах (≥128) качество модели резко ухудшается (MSE заметно растёт), так что оптимальным с точки зрения баланса MSE и скорости оказывается batch_size≈32.

Применение регуляризации и стратегий выбора шага:

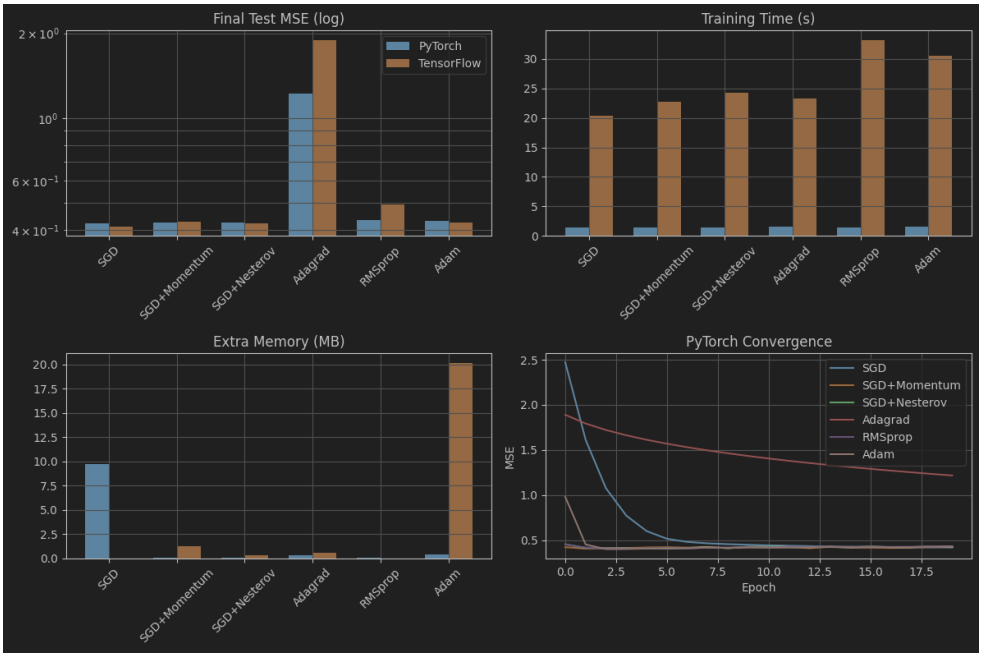


После добавления Elastic регуляризации видна стоимость штрафов: MSE слегка выросла по всем размерам батча (минимум стал чуть более высоким и сместился в область меньших батчей), время одной эпохи увеличилось (особенно при малых батчах из-за вычисления L1/L2-членов), тогда как профиль потребления памяти и общее число FLOPs остались практически теми же.



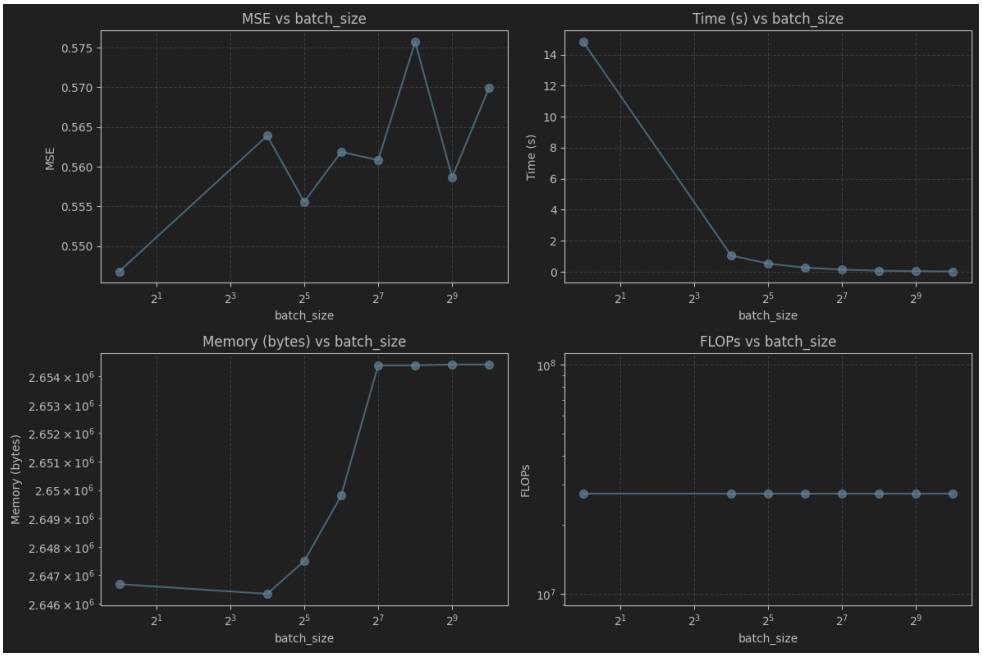
При использовании exp decay стратегии выбора шага (без какой-либо регуляризации) все ключевые закономерности остались. Стратегия лишь сгладила и немного сдвинула минимум MSE в область меньших батчей за счёт более мягкого уменьшения шага по ходу обучения, но не повлияла на профили времени, памяти и вычислительной сложности.

Библиотечные реализации:



При сравнении PyTorch и Keras на задаче California Housing большинство оптимизаторов дают близкий финальный MSE (~0.4–0.5), за исключением Adagrad в TF (≈1.9), но PyTorch оказывается в 10–20 раз быстрее (1.5–2 с против 20–33 с) и расходует значительно меньше дополнительной памяти для адаптивных методов (<1 МБ против ≈20 МБ у TF Adam). При этом в PyTorch Adam, Momentum/Nesterov и RMSprop сходятся за 3–5 эпох, plain SGD - за ~10–15, а Adagrad всё ещё медленно уменьшает ошибку после 20 эпох.

Модификация:



При использовании неявного (implicit) SGD видно, что качество (MSE) стало гораздо более устойчивым к выбору размера батча: разброс между минимальным и максимальным значением составляет всего ≈0.03 (минимум при самых малых батчах ≈0.547, максимум при средних/больших ≈0.576), тогда как в явном варианте он изменялся в разы. Остальные параметры также остались примерно константными. Это говорит о том, что implicit SGD даёт более ровную кривую обобщающей способности без потери эффективности по времени и ресурсам, позволяя использовать крупные батчи для ускорения обучения с минимальным ущербом для точности.