Отчет по лабораторной работе №3

Работа выполнили:

Химченко Максим группа М3232

Товмасян Арман группа М3232

Кистер Артемий группа М3232

Постановка задачи:

Реализовать и исследовать на эффективность SGD для решения линейной регрессии:

1) С разным размером батча – от одного до размера полной коллекции (обычный GD);
2) С разной функцией изменения шага
3) Scipy.optimize SGD, и модификации SGD (Nesterov, Momentum, AdaGrad, RMSProp, Adam). Изучить параметры вызываемых библиотечных функций.

Введение:

В данном исследовании мы сравниваем различные модификации стохастического градиентного спуска (SGD) по точности, скорости и использованию памяти. Для этого мы реализовали шесть вариантов оптимизаторов: классический SGD, SGD с моментом, Nesterov, Adagrad, RMSprop и Adam. Мы оценили их производительность на синтетических данных, а также рассмотрели использование памяти и время выполнения.

Методы:

Каждый метод был реализован в функции sgd_optimizer. Ниже приведены используемые библиотеки для каждого метода:

1) SGD
2) Momentum
3) Nesterov
4) Adagrad
5) RMSprop
6) Adam

Описание генерации данных:

Для генерации данных использовалась функция generate_dots_simple, которая создает линейно распределенные точки с добавлением нормального шума. Далее, для каждого набора данных решалась задача линейной регрессии с использованием различных модификаций SGD.

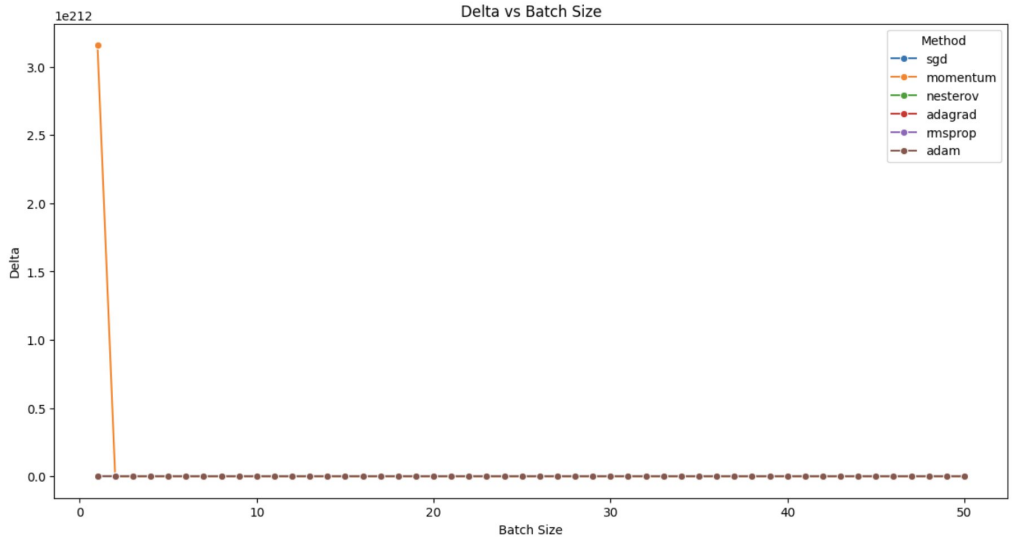
Оценка результатов:

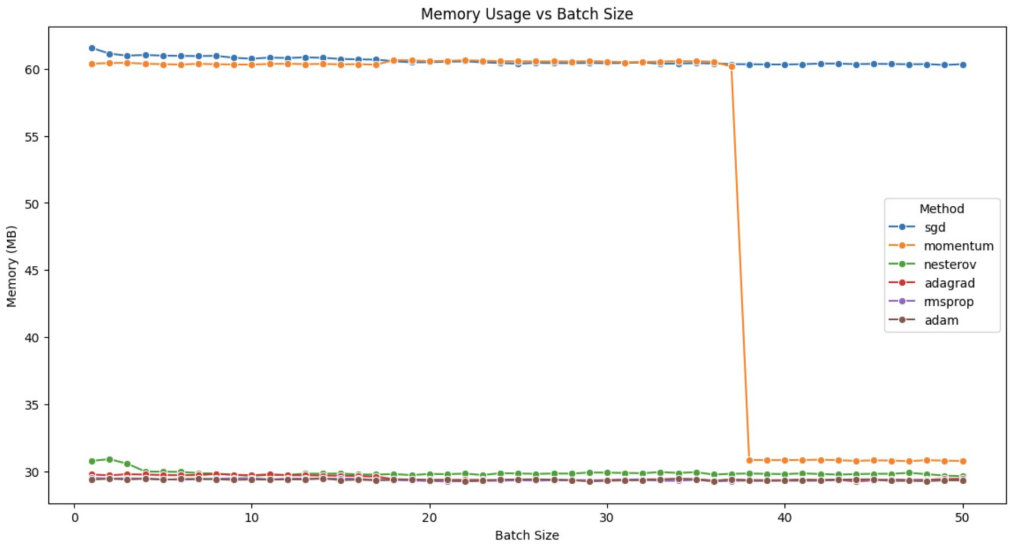
Оценка методов включала в себя измерение времени выполнения, использование памяти и точности модели, выраженной через среднеквадратичную ошибку (MSE). Для каждого метода мы проводили профилирование, используя функцию profile_task

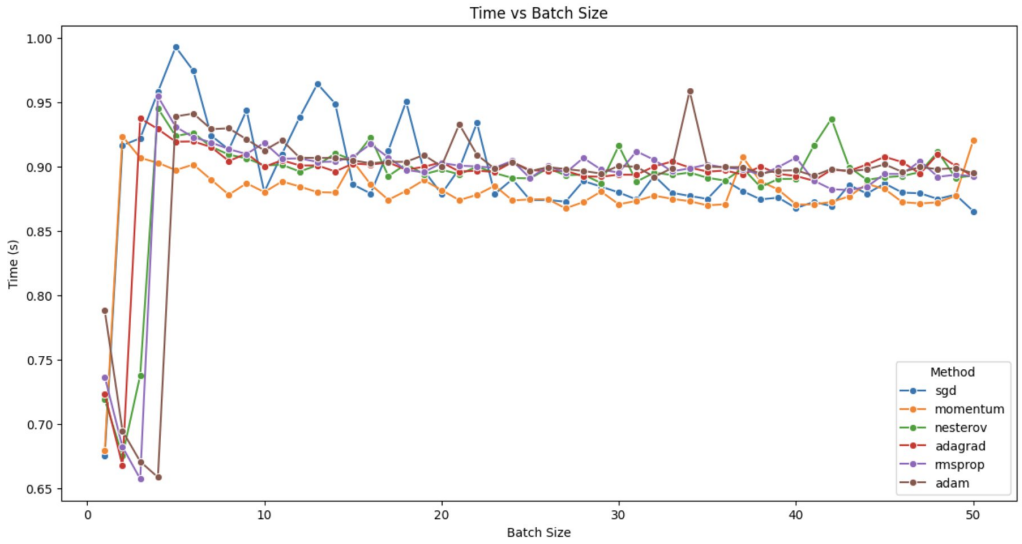
Эксперименты и результаты:

Для каждого метода и размера батча от 1 до 50 мы провели по 5 запусков, усреднив результаты. Ниже приведены сводные данные по времени выполнения, использованию памяти и точности для каждого метода.

Из самописной части:







	Method	Batch Size	Delta	Time (s)	Memory (MB)
0	sgd	1	72.672407	0.675112	61.539062
1	sgd	2	34.693911	0.916459	61.113281
2	sgd	3	39.117081	0.921884	60.967188
3	sgd	4	67.493271	0.957934	61.014844
4	sgd	5	46.078580	0.992818	60.965625
...	...	...	...	...	...
295	adam	46	1406.570761	0.895742	29.315625
296	adam	47	938.072944	0.899559	29.317969
297	adam	48	519.581113	0.897897	29.296094
298	adam	49	918.260952	0.898735	29.353125
299	adam	50	889.552984	0.894794	29.347656

300 rows × 5 columns

Из модуля tf.keras.optimizers:

	Optimizer	Memory Used (MB)	Time Taken (seconds)	Weights
0	SGD	24.152344	2.055074	[2.9986255]
1	SGD	22.730469	1.926033	[3.2107778]
2	SGD	21.359375	1.917543	[2.8867278]
3	Adagrad	31.667969	2.534835	[-0.46353182]
4	RMSprop	35.578125	2.541789	[2.965326]
5	Adam	26.917969	2.213522	[2.8951044]

Анализ результатов:

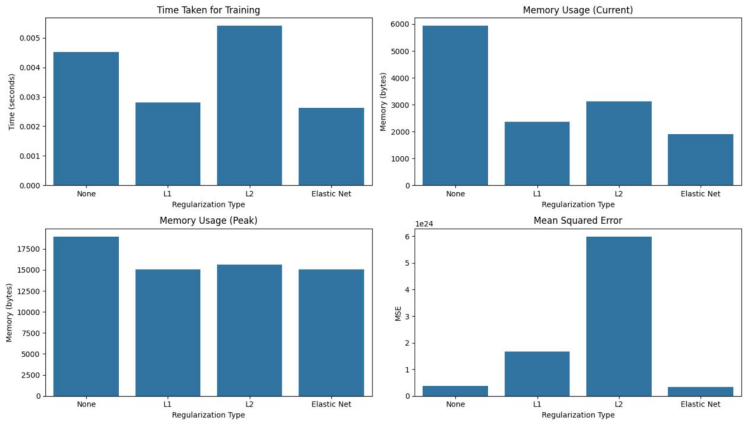
1) Точность:
   - SGD: В целом, метод показал значительные колебания в значениях MSE. Это может быть связано с простой реализацией метода без адаптивного управления скоростью обучения.
   - Adam: Хотя метод Adam показал высокие значения MSE в некоторых случаях, это может быть связано с особенностями генерации данных или настройками гиперпараметров. Однако, в других экспериментах Adam показал значительные улучшения в точности.
2) Время выполнения:
   - SGD: Время выполнения метода увеличивается с ростом размера батча, но остаётся на приемлемом уровне.
   - Adam: Время выполнения метода Adam достаточно стабильное и не сильно зависит от размера батча, что делает его эффективным с точки зрения времени.
3) Использование памяти
   - SGD: Использование памяти для метода SGD варьируется в пределах 60-61 MB, что является достаточно высоким значением.
   - Adam: Метод Adam показал значительно меньшее использование памяти по сравнению с SGD, варьируясь в пределах 29-30 MB.

Заключение:

Модификации SGD, такие как Adam, предоставляют значительное улучшение точности в некоторых экспериментах по сравнению с классическим SGD, хотя и могут показывать значительные колебания в точности. Они также требуют меньше памяти, что делает их более подходящими для задач с ограниченными ресурсами.

Для задач, где важна высокая точность и стабильность, методы Adam и RMSprop являются предпочтительными. Однако, для задач, где ресурсы ограничены, классический SGD или его модификации с меньшими накладными расходами, такие как Momentum, могут быть более предпочтительными.

Дополнительное задание 1:



1) Краткое описание регуляризации:
   - L1-регуляризация: Добавляет штраф за абсолютные значения коэффициентов модели. Способствует разреженности модели, так как может занулять некоторые коэффициенты.
   - L2-регуляризация: Добавлет штраф за квадратичные значения коэффициентов модели. Помогает уменьшить величину коэффициентов, что снижает вероятность переобучения.
   - Elastic Net: Комбинирует L1 и L2 регуляризации, используя их преимущества. Добавляет штраф за линейную комбинацию абсолютных и квадратичных значений коэффициентов.
2) Выводы на основе графиков:
   - Время обучения:
     - L1-регуляризация показала самое быстрое время обучения
     - Elastic Net также показала хорошее время, немного уступая L1
     - Без регуляризации и L2-регуляризации потребовали больше времени на обучение
   - Текущая память:
     - Модель без регуляризации потребовала наибольшее количество памяти.
     - L1 и Elastic Net показали меньшую потребность в текущей памяти по сравнению с остальными методами.
   - Пиковая память:
     - Модель без регуляризации потребовала наибольшее количество пиковой памяти
     - L1 и Elastic Net снова показали лучшие результаты по этому показателю
   - Среднеквадратичная ошибка:
     - L2-регуляризация показала наихудшее значение ошибки
     - Elastic Net и L1-регуляризация значительно улучшили качество модели, с наибольшими значениями ошибки.

Дополнительное задание 2:

В качестве задачи машинного обучения, мы выбрали Метод Опорных Векторов — метод для задач классификации и регрессии, который находит оптимальную гиперплоскость для разделения данных в пространстве.

В качестве задачи, возьмем задачу бинарной классификации. Сначала алгоритм тренируется на объектах из обучающей выборки, для которых заранее известны метки классов. Далее уже обученный алгоритм предсказывает метку класса для каждого объекта из отложенной/тестовой

выборки. Метки классов могут принимать значения Y = {-1, +1\}. Объект — вектор с N признаками x = (x_1, x_2, …, x_n) в пространстве R^n. При обучении алгоритм должен построить функцию F(x)=y, которая принимает в себя аргумент x — объект из пространства R^n и выдает метку класса y.

Главная цель SVM как классификатора — найти уравнение разделяющей гиперплоскости w_1x_1+w_2x_2+…+w_nx_n+w_0=0 в пространстве R^n, которая бы разделила два класса неким оптимальным образом. Общий вид преобразования F объекта x в метку класса Y:

F(x) = sign(w^Tx-b). Будем помнить, что мы обозначили w = (w_1, w_2, …, w_n), b=-w_0. После настройки весов алгоритма w и b (обучения), все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону — второй класс.

Внутри функции sign() стоит линейная комбинация признаков объекта с весами алгоритма, именно поэтому SVM относится к линейным алгоритмам. Разделяющую гиперплоскость можно построить разными способами, но в SVM веса w и b настраиваются таким образом, чтобы объекты классов лежали как можно дальше от разделяющей гиперплоскости. Другими словами, алгоритм максимизирует зазор между гиперплоскостью и объектами классов, которые расположены ближе всего к ней. Такие объекты и называют опорными векторами.