Методы оптимизации
Отчёт по лабораторной работе №3

Авторы работы:
Галкин Глеб M3234
Дьяков Максим M3234
Кирьяк Александр M3234

Ссылка на репозиторий

Постановка задачи

1. Задача линейной регрессии.
2. Стохастический градиентный спуск с разным размером батча.
   - Одноэлементный батч
   - Mini-batch
   - Сравнение SGD с разными размерами батча
3. Стохастический градиентный спуск с разной функцией изменения шага.
   - Функция линейного затухания
   - Ступенчатая функция
   - Сравнение функций шага
4. Модификации стохастического градиентного спуска (TensorFlow).
   - Momentum
   - Nesterov
   - AdaGrad
   - RMSProp
   - Adam
   - Сравнение модификаций

Основное задание

1 Задача линейной регрессии

Линейная регрессия - используемая в статистике регрессионная модель зависимости одной переменной y от другой или нескольких других независимых переменных (регрессоров) x с линейной функцией зависимости. Если имеется несколько независимых переменных $(x_1, x_2, \ldots, x_n)$, то модель линейной регрессии может быть представлена в виде: $y = w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b$, где:

- x - независимая переменная;
- w - коэффициент наклона линии регрессии;
- b - смещение (относительно начала координат);
- y - предсказываемая (зависимая) величина.

Таким образом, задачей линейной регресии является поиск значений коэффициентов w_i и b, при этом, минимизируя разницу между реальными значениями y и предсказанными $\hat{y}$, где разница, обычно, вычисляется по формуле:

$$L(w) = \frac{1}{m} \cdot \sum_{i=1}^{m} (\hat{y_i} - y_i)^2.$$

2 Стохастический градиентный спуск с разным размером батча

Градиентный спуск может потребовать много итераций, при этом каждая итерация рассматривает весь датасет. Поэтому вместо вычисления полного градиента функции в стохастическом градиентном спуске, мы можем выбрать несколько случайных батчей и посчитать их градиенты. Дальше полученные градиенты усредняются. Такой подход делает вычисления более эффективными, но при этом менее точными.

Рассмотрим несколько обобщений стохастического градиентного спуска в зависимости от размера батча.

2.1 Одноэлементный батч

Выберем случайный элемент e_k из набора $e_1, ..., e_m$. Функция потери и её градиент:

$$H_k(w) = L(w, e_k)$$
$$g = \nabla H_k(w)$$

Тогда приближения параметров модели на каждом шаге пересчитываются следующим образом:

$$w_{k+1} = w_k - \alpha \cdot g,$$

где α — learning rate.

2.2 Mini-batch

Вместо одного элемента будем выбирать несколько. Это потенциально замедлит вычисление, но сделает его более точным.

Пусть мы выбрали набор E, состоящий из s элементов. Функция потери и её градиент:

$$H_E(w) = \frac{1}{s} \cdot \sum_{e \in E} L(w, e)$$
$$g = \nabla H_E(w)$$

2.3 Сравнение SGD с разными размерами батча

Сравнивать будем на задаче двумерной линейной регрессии: $y = wx + b$. Функция ошибки в данном случае будет равна $H(w_k) = \sum_{i=1}^{n}(w_k x_i + b - y_i)^2$, а её градиент по i-ому направлению $\nabla H_i(w_k) = 2x_i \cdot \sum_{i=1}^{n}(w_k x_i + b - y_i)$.

Тестовые данные:

```
np.random.seed(234)
X = 2 * np.random.rand(200, 1)
Y = 6 * X + np.random.randn(200, 1) + 12
```

Результаты:

Размер батча	w	b	Кол-во итераций
1	6.82	11.13	622
32	7.51	10.12	372
50	6.78	11.03	602
64	6.83	10.97	738
100	6.66	11.18	676

Таблица 1: Сравнение SGD с разными размерами батча

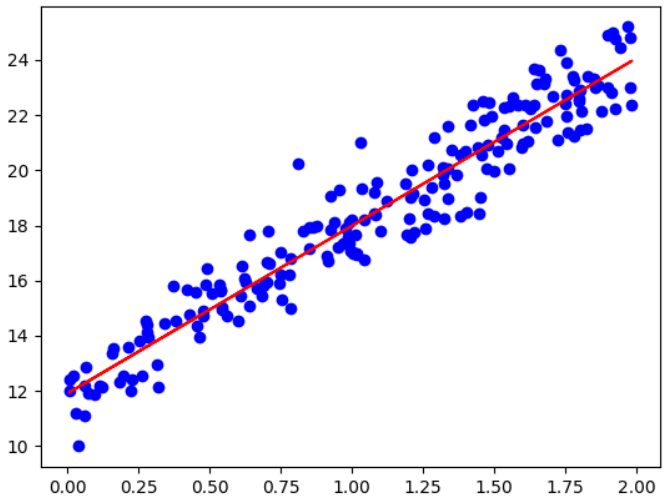

Рис. 1: batch size = 1

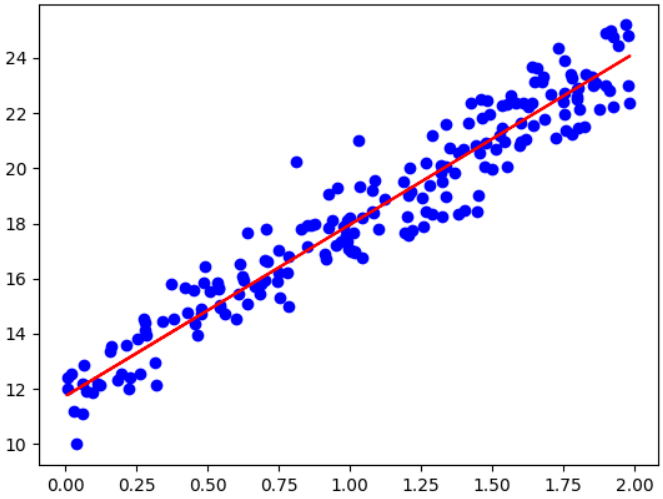

Рис. 2: batch size = 32

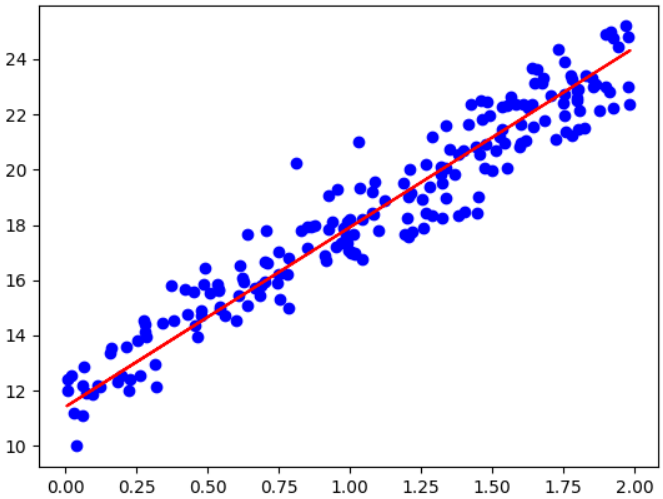

Рис. 3: batch size = 50

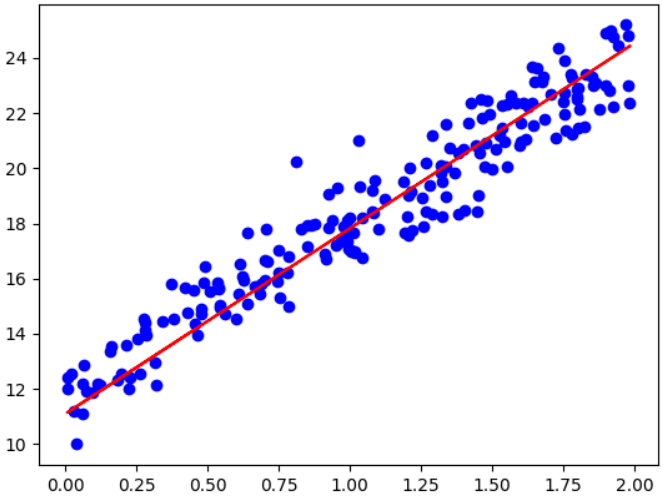

Рис. 4: batch size = 64

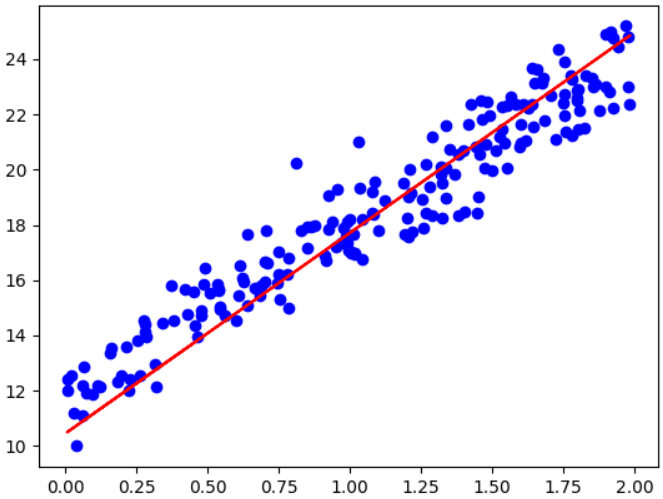

Рис. 5: batch size = 100

Хорошую сходимость показал размер батча равный 32. При этом значение функции ошибки на граничных размерах было максимальным. Из чего можно сделать вывод, что батчевый градиентный спуск (с размером батча равным степени двойки) позволяет ускорить сходимость, не теряя точность.

3 Стохастический градиентный спуск с разной функцией изменения шага

Как мы убедились на примере первых 2-х лабораторных, постоянный шаг - зло. Попробуем оптимизировать наш метод изменяя шаг по мере "спуска".
Рассмотрим функцию линейного затухания и ступенчатую функцию для выбора размера шага на каждой итерации.

3.1 Функция линейного затухания

Значение шага будет изменяться по формуле $\alpha = max(\alpha_{min}, \alpha_{start} - it \cdot decay)$, где α_{min}– минимальное значение шага, α_{start}– начальное значение шага, $decay$ - величина уменьшения шага на каждой итерации.

Начальные параметры:
$\alpha_{min} = 10^{-3}$;
$\alpha_{start} = 10^{-1}$;
$decay = 10^{-3}$.

3.2 Ступенчатая функция

Значение шага будет вычисляться по формуле $\alpha = \alpha_{start} \cdot \beta^{* \frac{it}{decay}}$, где α_{start} - начальное значение шага, $decay$ - период затухания.

Начальное параметры:
$\alpha_{start} = 10^{-1}$;
$decay = 100$.

3.3 Сравнение функций шага

Для сравнения разных подходов изменения шага зафиксируем количество эпох $epochs = 20$ и размер батча $batch = 32$.

Результаты:

Используемый алгоритм	w	b
Обычный SGD	8.53	8.80
SGD с линейным затуханием	7.13	10.59
SGD со ступенчатой функцией	6.22	11.73

Таблица 2: Сравнение SGD с разными размерами функциями изменения шага

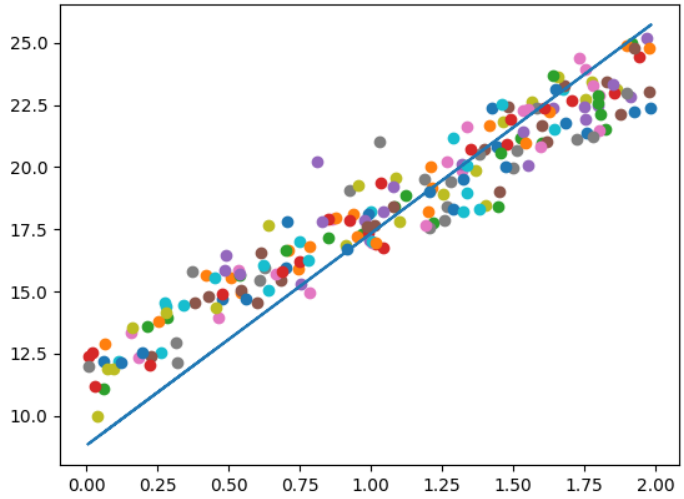

Рис. 6: Обычный SGD

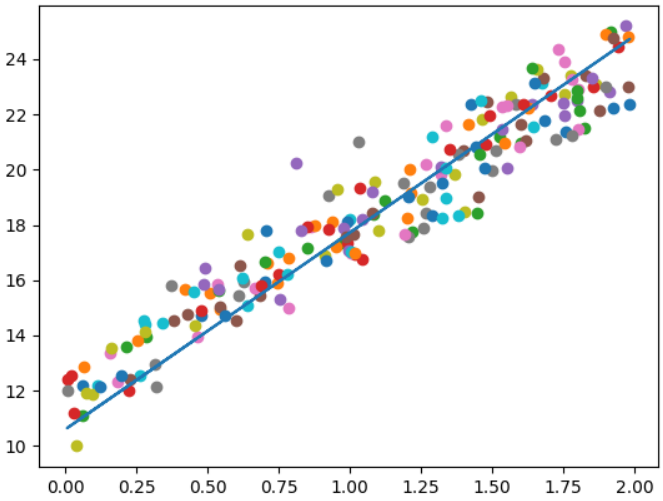

Рис. 7: SGD с линейным затуханием

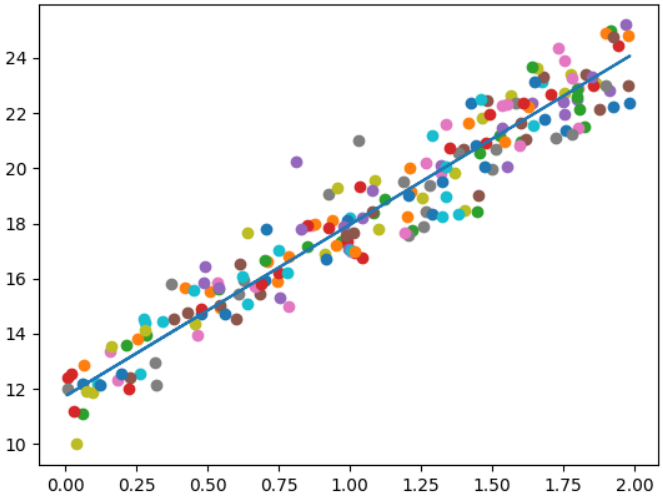

Рис. 8: SGD со ступенчатой функцией

Видим, что наилучшую сходимость обеспечило использование ступенчатой функции для нахождения оптимального размера шага. При этом обе функции существенно уменьшили количество итераций алгоритма. Фиксированный шаг снова проиграл...

4 Модификации стохастического градиентного спуска (TensorFlow)

Momentum

Метод Momentum является ускорением и стабилизацией процесса оптимизации. Основная идея данного метода заключается в накапливание импульса, тем самым, учитывая предыдущие значения градиента. Такой подход помогает эффективно двигаться к глобальному минимуму, преодолевая локальные. С математической точки зрения, данный метод использует экспоненциальное сглаживание для значения градиента. На каждой итерации значения вычисляются по формулам:

$$v_{k+1} = \beta \cdot v_k - \alpha \cdot \nabla f(w_k)$$

$$w_{k+1} = w_k + v_{k+1}$$

Nesterov

В 1983 году Юрием Нестеровым была предложена модификация метода Momentum, которая считает градиент не в текущей точке, а в точке, в которую мы бы пошли на предыдущем шаге. Такой подход позволяет исправлять ошибки на текущем шаге. В таком случае, значения считаются по данным формулам:

$$v_{k+1} = \beta \cdot v_k - \alpha \cdot \nabla f(w_k - \beta \cdot v_k)$$

$$w_{k+1} = w_k + v_{k+1}$$

AdaGrad

Метод моментов и метод Нестерова учитывают только историю изменения градиента, но никак не связаны с самими оптимизируемыми параметрами. Идея заключается в том, что некоторые параметры могут быстрее достигать своего оптимума, чем другие. Поэтому хотелось бы параметры близкие к оптимуму менять с меньшим шагом, а более далекие – с большим. AdaGrad реализует данную идею. В нем шаг для параметров зависит от величины их колебаний: чем они больше, тем меньше шаг.

$$G_{k+1} = G_k + (\nabla f(w_k))^2$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{G_{k+1} + \epsilon}} \cdot \nabla f(w_k)$$

RMSProp

Модификация метода AdaGrad — метод RMSprop — вместо суммы использует экспоненциальное скользящее среднее в знаменателе. Основное отличие RMSProp в том, что он вместо хранения истории квадратов по каждому параметру, просто берет корень из среднего квадратов градиентов по всем параметрам.

$$G_{k+1} = \gamma \cdot G_k + (1 - \gamma) \cdot (\nabla f(w_k))^2$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{G_{k+1} + \epsilon}} \cdot \nabla f(w_k)$$

Adam

Adam достаточно часто применяется при обучении нейронных сетей. Фактически, этот алгоритм является очередной модификацией алгоритма AdaGrad, использующий сглаженные версии среднего и среднеквадратического градиентов:

$$v_{k+1} = \beta_1 \cdot v_k + (1 - \beta_1) \cdot \nabla f(w_k)$$

$$G_{k+1} = \beta_2 \cdot G_k + (1 - \beta_2) \cdot (\nabla f(w_k))^2$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{G_{k+1} + \epsilon}} \cdot v_{k+1}$$

Сравнение модификаций

Тестовые данные

Для сравнения сходимости и объему оперативной памяти необходимо создать довольно большой датасет, например, размера 10^4:

```
np.random.seed(543)
X = -1.43 * 1.45 * np.random.rand(10000, 1)
Y = 7 * X + 2 * np.random.rand(10000, 1)
Z = 2 * X + 5 * Y + 10 * np.random.rand(10000, 1) + 3
```

Для каждого метода установим одинаковые начальные параметры:

- Размер батча $batch = 32$;
- Количество эпох $epoch = 30$.

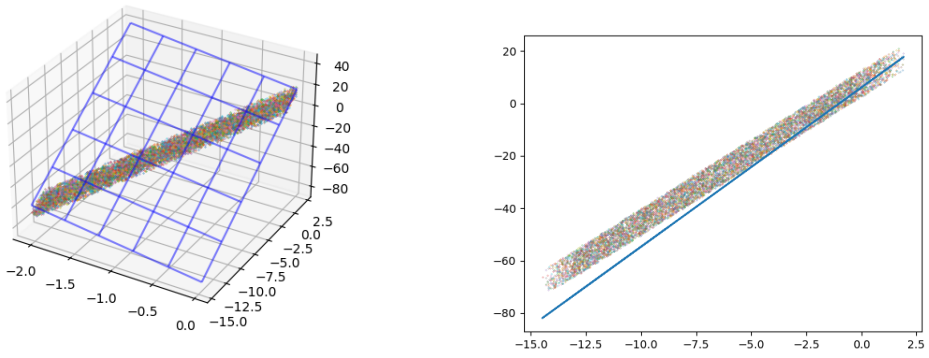

Рис. 9: SGD без модификаций

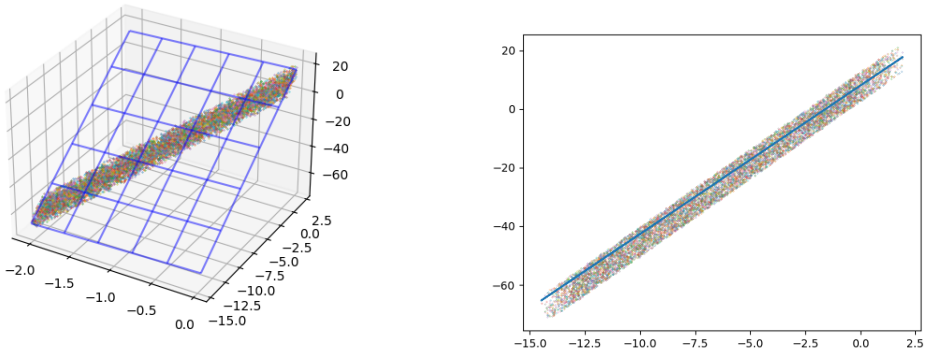

Рис. 10: SGD с модификацией Momentum

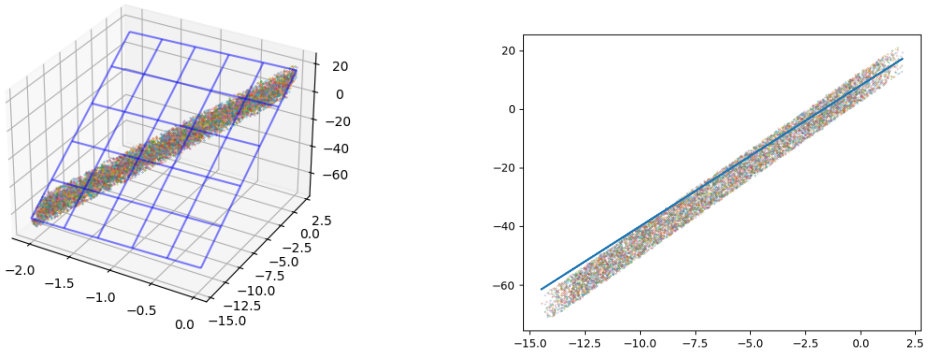

Рис. 11: SGD с модификацией Nesterov

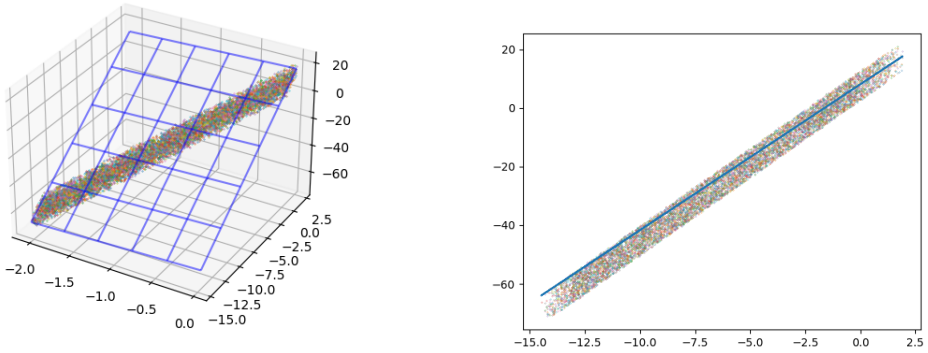

Рис. 12: SGD с модификацией AdaGrad

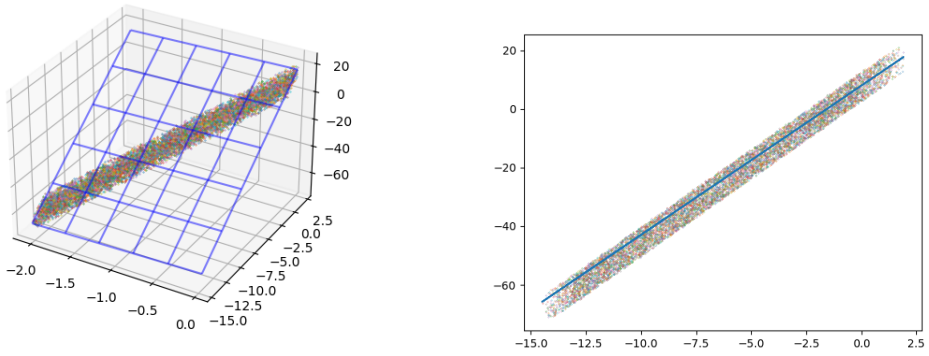

Рис. 13: SGD с модификацией RMSProp

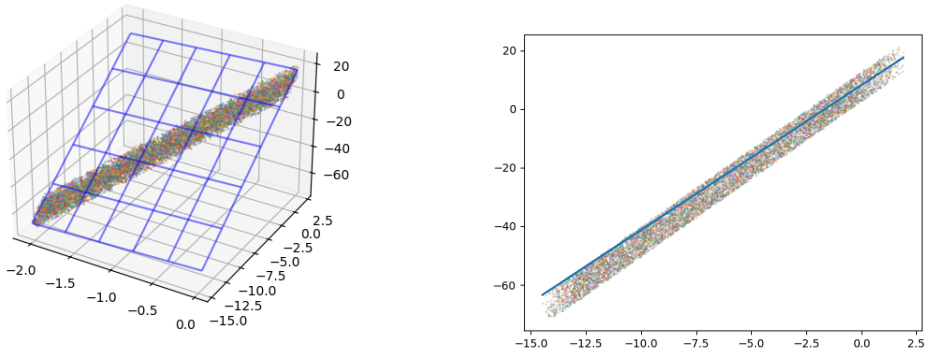

Рис. 14: SGD с модификацией Adam

Метод	Используемая память
SGD	312.9Mb
Momentum	313.2Mb
Nesterov	313.3Mb
AdaGrad	316.3Mb
RMSProp	317.2Mb
Adam	319.5Mb

Таблица 3: Сравнение SGD с разными размерами батча

Вывод

- Метод SGD без модификаций показал худшую сходимость среди остальных методов, при этом потребляя наименьшее количество памяти.
- Метод Momentum является наиболее оптимальным по машинным ресурсам, при этом менее надеждным и быстросходящимся.
- Метод Nesterov имеет надежность и скорость сходимости на уровне метода Momentum, при этом требует большего количества машинных ресурсов.
- Метод AdaGrad является методом имеющим среднии характеристики по всем оцениваемым параметрам. Главный недостаток данного алгоритма – постоянное уменьшение шага обучения, так как мы складываем квадраты градиентов и делим на корень квадратный из этой величины.
- Методы RMSProp и Adam являются самыми надежными и быстросходящиеся методы, при это требующие большее количество машинных ресурсов.

Полученные результаты вполне оправданы, ведь модификации используют дополнительную память, значит SGD потребляет меньше памяти, а Adam потребляет наибольшее количество памяти, так как хранит моменты первого и второго порядка для каждого параметра.
Поэтому, если задача требует минимальных ресурсов, лучший выбор — SGD. Для задач, где требуется более сложная адаптация шага обучения и есть доступ к большим вычислительным ресурсам, Adam может быть более подходящим вариантом, несмотря на его высокие требования к памяти и вычислительной мощности.