Лабораторная работа 1

Крухмалев Константин,Рашо Елизавета,Фролова Дарья, M34371

21 апреля 2022 г.

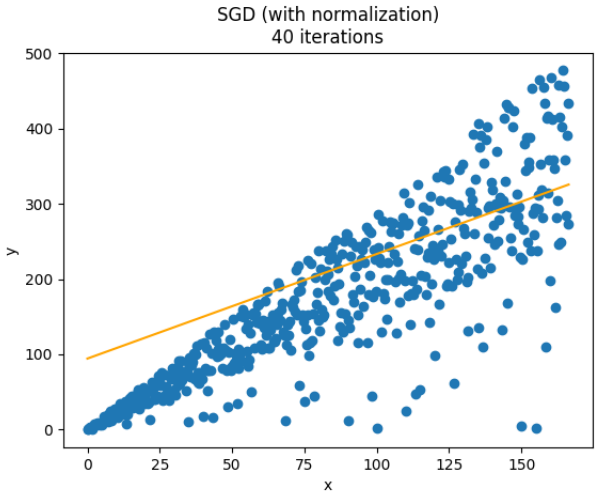
https://github.com/Konstantin343/optimization-methods-itmo/tree/main/lab2

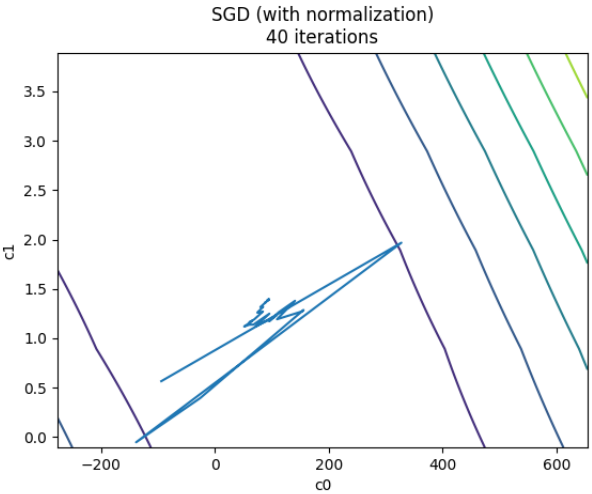
1 Стохастический градиентный спуск

1.1 Описание

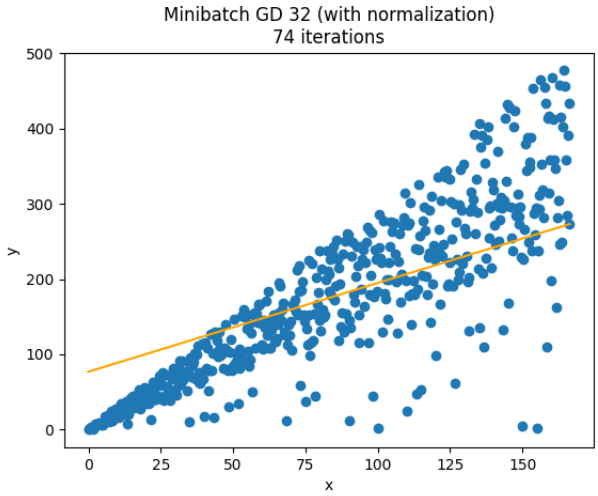
Оптимизационный алгоритм, отличающийся от обычного градиентного спуска тем, что градиент оптимизируемой функции считается на каждом шаге не как сумма градиентов от каждого элемента выборки, а как градиент от одного, случайно выбранного элемента.
Выбрана функция ошибки MSE, количество используемых объектов - batch_size, передается алгоритмы как аргумнт.
Для графиков сгенерирован датасет с однмерной линейной регрессией.

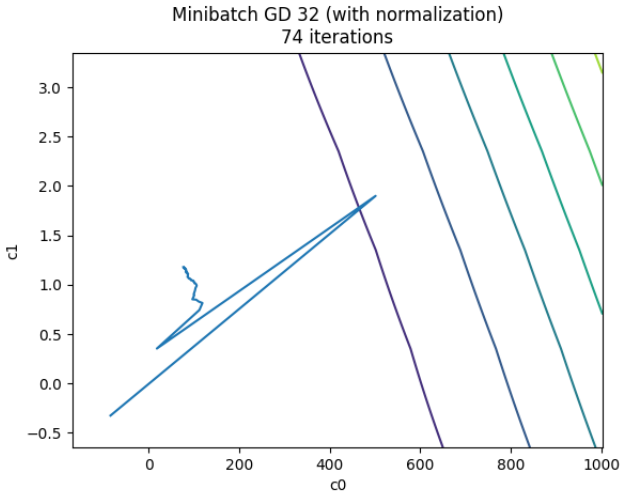
1.2 1 - SGD



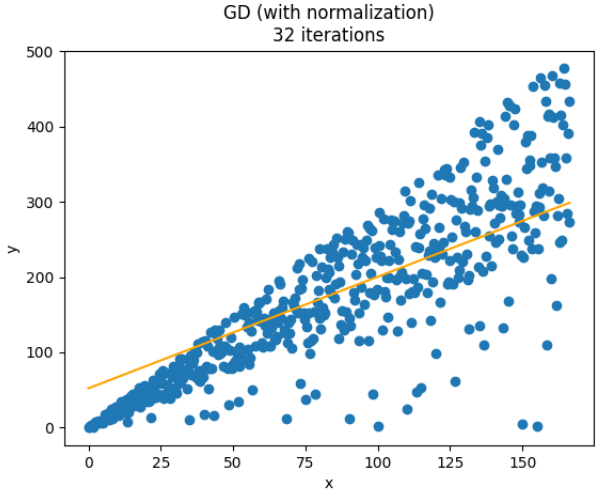


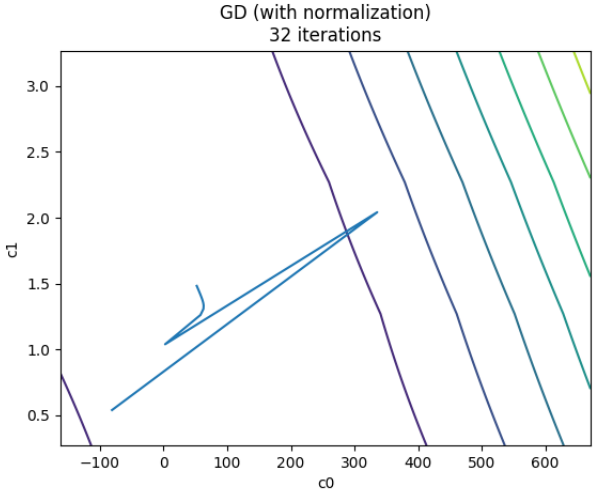
1.3 2..n-1 - Minibatch GD





1.4 n - GD





1.5 Выводы

Наиболее точные результаты с меньшим числом итераций получены в процессе полного градиентного спуска.
Стохастический градиентный спуск дает наимение точные результаты, однако затрачивает меньше итераций, чем Minibatch GD и работает быстрее по времени.

2 Scaling (нормализация)

2.1 Описание

В лабораторной выполняетс minimax нормализация данных.
При нормализации для каждой координаты выполняется следующее:

$$v_i^{norm} = (v_i - min_i)/(max_i - min_i)$$

При денормализации коэффициентов выполняется следующее:

$$v_0 = min_y + \sum_{i=1}^{n} (v_i * min_i)/(max_i - min_i)$$

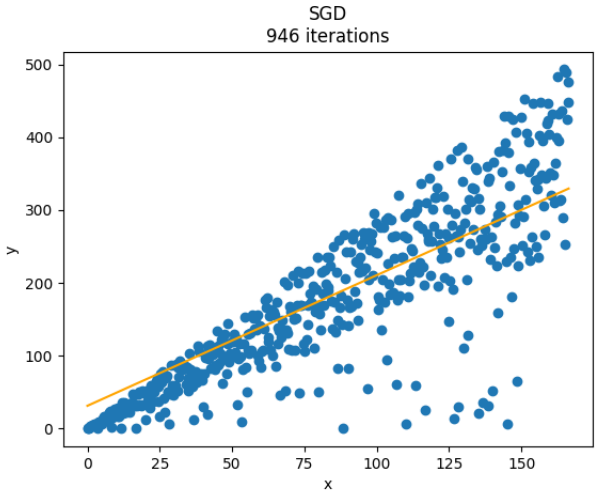
$$v_i = v_i^{norm}/(max_i - min_i) * (max_y - min_y)$$

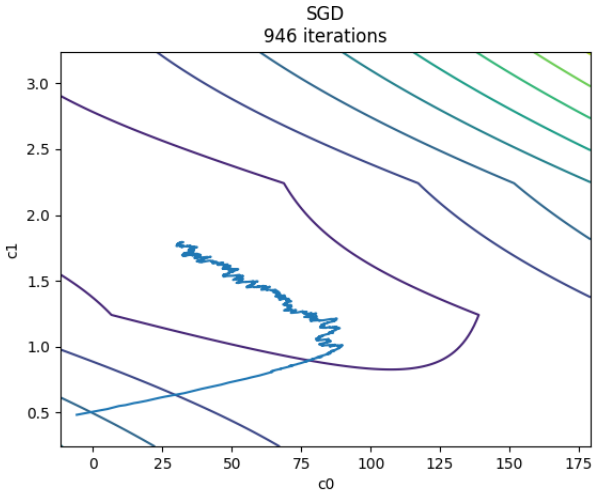
2.2 Выводы

Без предварительной нормализации во всех предыдущих примерах достигаются слишком большие значения ошибки, в следствие чего алгоритм не завершает свою работу.
При денормализации коэффициентов может быть потеряна точность, но алгоритм всегда завершается.

3 Модификации градиентного спуска

3.1 SGD

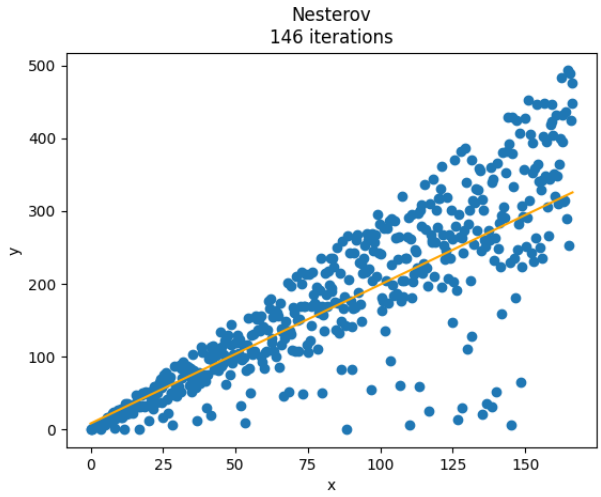


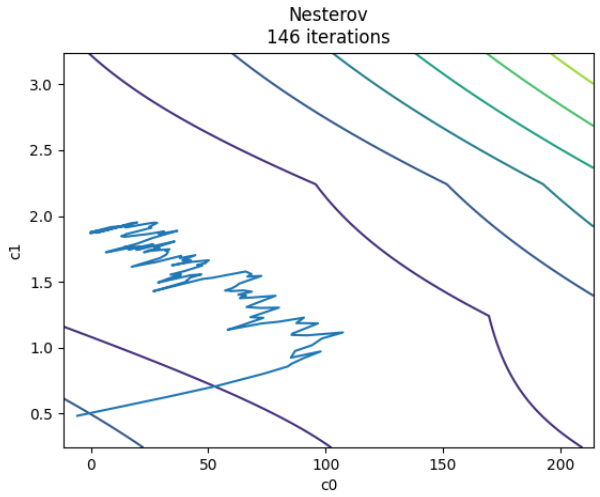


3.2 Nesterov

Улучшение метода SGD, увеличивающие скорость сходимости. Вместо того чтобы высчитывать градиент в текущей точке, будем использовать градиент в точке “предсказанной” на основании сдвига, расчитанного на предыдущем шаге.

$$\Delta_{p+1} = *\Delta_p + \eta * \delta L(w_p \gamma \Delta_p)$$
$$w_{p+1} = w_p \Delta_{p+1}$$



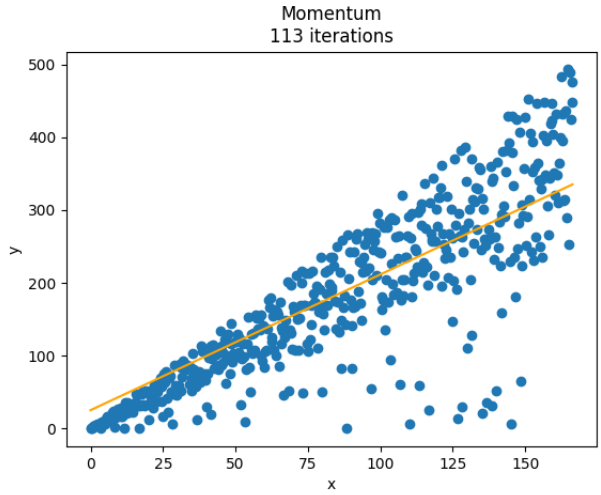


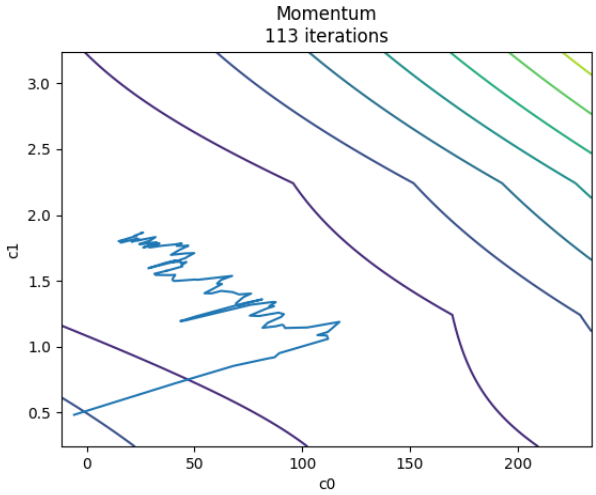
Здесь исходят из предположения, что основной вклад в вектор сдвига даёт первое слагаемое, а слагаемое с градиентом лишь "уточняет". Логично поэтому уточняющий градиент расчитывать в окрестности новой точки, а не в текущей.

3.3 Momentum

Одна из проблем с SGD в том, что когда функция попадает в "овраг", т.е. по одному из направлений имеем быстрый спуск, а по другому медленный, то SGD приводит к осциляции и крайне медленной сходимости к минимуму.
Для решения данной проблемы был предложен подход, который увеличивает шаг по направлению к минимуму, и уменьшает осциляцию. Это достигается за счёт того, что сдвиг параметров расчитывается как взвешенная сумма сдвига на предыдущем шаге и нового на основе градиента:

$$\Delta_{p+1} = \gamma * \Delta_p + \eta * \Delta L(w_p)$$
$$w_{p+1} = w_p - \Delta_{p+1}$$





3.4 AdaGrad

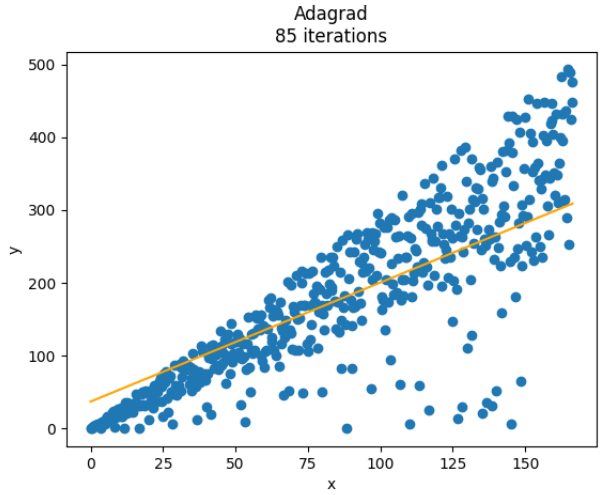
Вместо единого скаляра η в качестве скорости обучения, на каждой итерации будем определять вектор $\eta_p = (\eta_p^{(1)}, \ldots, \eta_p^{(d)})$.
В данном случае η, которую мы использовали ранее будет просто начальной скоростью обучения. Для первой итерации мы положим $\eta_p^{(i)} = \eta, i = 1, \ldots, d$.

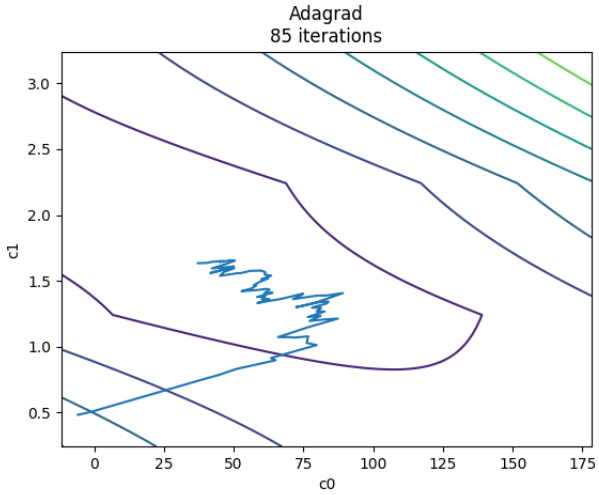
$$G_p^{(i)} = \sum_{j=1}^{p} (g_j^{(i)})^2, i = 1, ..., d$$

$$\eta_p^{(i)} = \frac{\eta}{\sqrt{G_p^{(i)} + \epsilon}}$$

Изменение параметров осуществляем практически по той же формуле, что и раньше:

$$w_{p+1} = w_p \eta_p * \Delta L(w_p)$$





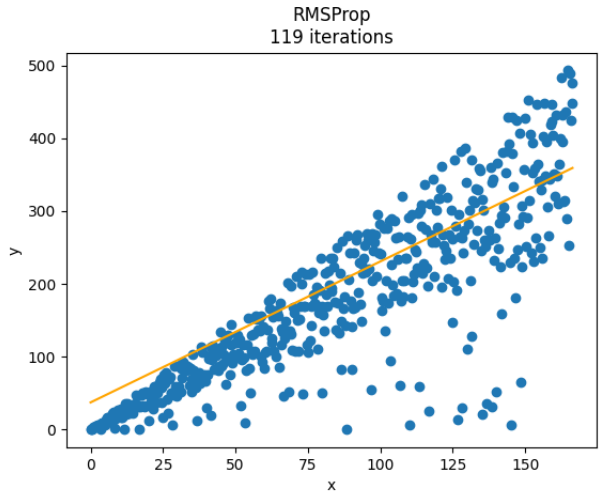
3.5 RMSProp

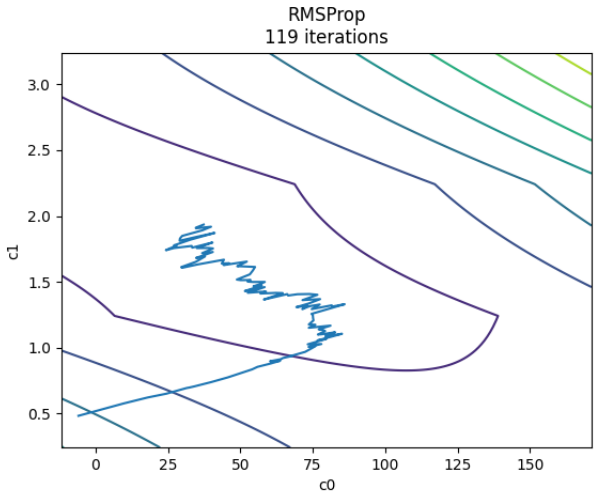
Основная проблема Adagrad заключается в том, что знаменатель в коэффициенте скорости обучения постоянно растёт, соответственно, через некоторое время для части параметров скорость обучения упадёт до нуля.
Так же Adagrad не избавляет нас от необходимости выбирать начальное значение η.

Предлагается следить за скользящим средним градиентов штрафной функции:

$$v_p = \beta * v_{p-1} + (1 - \beta)\Delta L(w_{p-1})^2$$

$$w_{p+1} = w_p - \eta * \frac{\Delta L(w_p)}{\sqrt{v_p} + \epsilon}$$





3.6 Adam

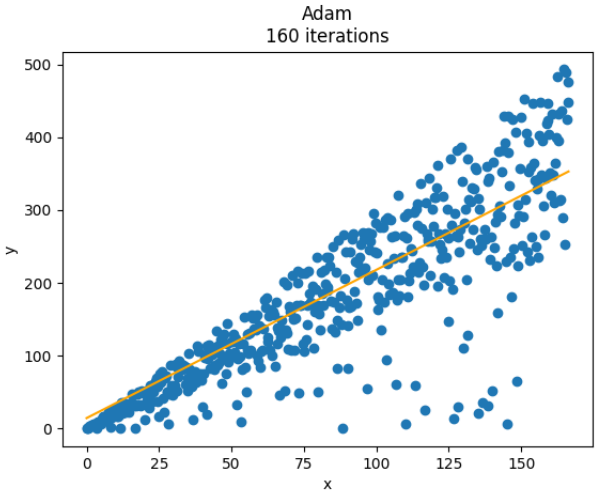
Adam — один из самых эффективных алгоритмов оптимизации в обучении нейронных сетей. Он сочетает в себе идеи RMSProp и оптимизатора импульса. Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), как в RMSProp, Adam также использует среднее значение вторых моментов градиентов.
В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент, а параметры β_1 и β_2 управляют скоростью затухания этих скользящих средних.

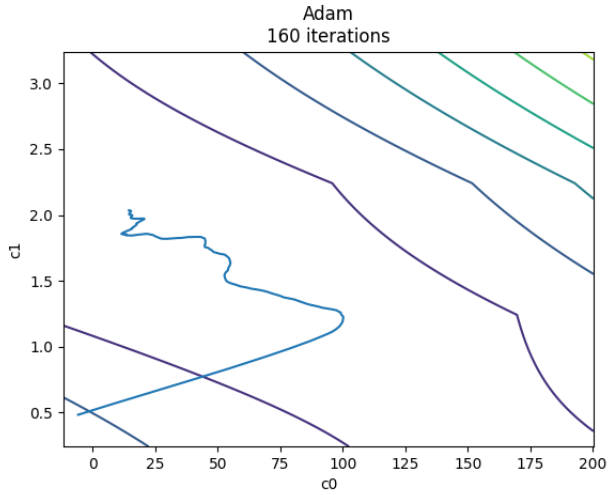
Предлагается следить за скользящим средним градиентов и квадратов градиентов штрафной функции:

$$m_p = \beta_1 * m_{p-1} + (1 - \beta_1)\Delta L(w_{p-1})$$
$$v_p = \beta_2 * v_{p-1} + (1 - \beta_2)\Delta L(w_{p-1})^2$$

$$m'_p = \frac{m_p}{1 - \beta_1^p}$$
$$v'_p = \frac{v_p}{1 - \beta_2^p}$$

$$w_{p+1} = w_p - \eta * \frac{m'_p}{\sqrt{v'_p} + \epsilon}$$





4 Сравнение алгоритмов