Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Прикладная математика и информатика

Методы оптимизации

Отчет по лабораторной работе №2

Работу выполнили:

Захаров Кирилл М32391

Мавлютов Эрвин М32391

Шилкин Артем М32391

Преподаватель:

Шохов Максим Евгеньевич

г. Санкт-Петербург

2023 г.

Оглавление

Постановка задач:

1. Изучение стохастического градиентного спуска и использование различных батчей.
2. Изучение функций изменения шага (learning rate scheduling) для улучшения сходимости, на примере экспоненциальной и ступенчатой функций.
3. Исследование модификаций (стохастического) градиентного спуска: Nesterov, Momentum, AdaGrad, RMSProp, Adam.
4. Исследование сходимости и сравнение методов по параметрам: объем оперативной памяти, количество арифметических операций, времени выполнения.

Весь проект можно найти по ссылке.

Задание 1. Стохастический градиентный спуск и Minibatch

В случае, если минимизируемая функция имеет вид $f(x) = \sum_{k=0}^{n} f_k(x)$ можно применить метод стохастического градиентного спуска, который заключается в вычислении градиента лишь по одной функции из n суммируемых, то есть следующая точка при спуске будет вычисляться по формуле: $w_{i+1} = w_i + \alpha_k \nabla f_k(w_i)$, где k – случайный номер одной из суммируемых функций. В таком случае мы будем делать больше шагов, но не будем при каждом шаге совершать тяжелые вычисления всего градиента, что зачастую помогает сойтись быстрее. Чтобы точно не игнорировать некоторые слагаемые, можно выбирать их не случайно, а по очереди. Также как одну из модификаций можно предложить считать производную не по единственной функции а по подмножеству мощности от 1 до n (количество функций, по которым считается градиент, в таком случае называется батчем), чтобы достичь баланса между вычислениями функции и количеством шагов.

В качестве f можно выбрать функцию, минимизируемую при решении задачи о линейной регрессии:

$$f(a, b) = \sum_{k=1}^{n} (a \cdot x_k + b - y_k)^2 .$$

Для сравнения эффективности работы градиентного спуска при различных значениях батча и последовательностях α_k будем искать линейную функцию, наиболее точно отражающую поведение точек на плоскости.

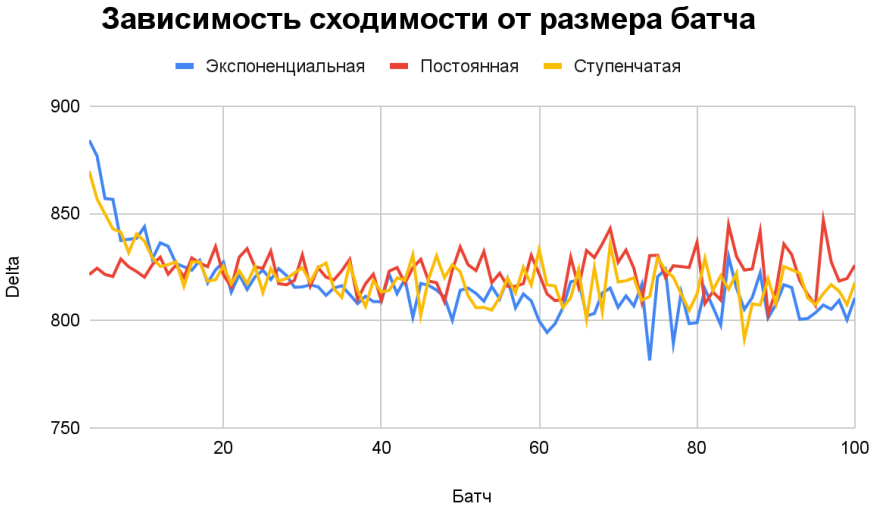

Рис. 1 – Зависимость погрешности от размера батча. delta – разница полученного ответа с ответом, посчитанным аналитически.

Для тестирования будем генерировать 100 случайных точек и запускать градиентный спуск с размером батча от 1 до 100 из фиксированной точки. Видно, что в среднем с ростом батча погрешность уменьшается. Получается что в нашем случае важность точности градиента превышает влияние количества сделанных шагов.

Задание 2. Использование функции изменения шага для улучшения сходимости

Теперь воспользуемся ступенчатой функцией изменения шага, чтобы улучшить сходимость. Будем считать по следующей формуле:

$$step = \frac{lr}{n \cdot epoch} \cdot steps,$$

где lr – это изначальный learning rate;

n - число точек;

$steps$ – число шагов, после которого надо изменить текущий lr;

$step$ – длина шага, на которую мы меняем lr.

Таким образом, lr будет шагами понижаться до 0.

Также воспользуемся экспоненциальной функцией изменения шага, формула проще:

$\alpha' = \alpha * \beta$, где α - текущий lr;

α' - новый lr;

β - экспоненциальный коэффициент.

Задание 3. Модификации градиентного спуска

Momentum

Физическая аналогия для данного метода заключается в понятии инерции в реальном мире: когда лыжник катится с горы, он не застревает на каждом бугорке и не останавливается в небольшой ямке, а спокойно их преодолевает, так как обладает некоторой инерцией. Выяснилось, что применив данный метод в градиентном спуске, мы также игнорируем незначительные локальные минимумы и седла и зачастую находим более показательный минимум.

В наших обозначениях метод выражается двумя последовательностями: v_i – текущая “скорость” и w_i – текущая позиция. $v_{i+1} = \beta v_i - \alpha \nabla f(w_i)$, $w_{i+1} = w_i + v_{i+1}$.

Nesterov

Это другая вариация сглаживания градиента. В этом методе мы немного модифицируем Momentum и при пересчете скорости учитываем градиент не в текущей точке, а в той, в которой оказались бы продолжив идти по текущему вектору, т.е. $v_{i+1} = \beta v_i - \alpha \nabla f(w_i + \beta v_i)$, $w_{i+1} = w_i + v_{i+1}$.

AdaGrad

Здесь мы подбираем размер шага для каждой координаты в зависимости от ее скорости роста: если функция по этой координате растет быстро, мы должны замедлиться, чтобы не пропустить точку минимума, а если наоборот, то можно ускориться и быстрее найти ответ. Идея выражается в следующих формулах: $G_{i+1} = G_i + (\nabla f(w_i))^2$, $w_{i+1} = w_i - \dfrac{\alpha}{\sqrt{G_{i+1} + \varepsilon}} \cdot \nabla f(w_i)$. Здесь все операции выполняются

покомпонентно. Однако здесь G постоянно растет и в будущем шаги могут оказаться слишком маленькими. Эту проблему решает следующий метод.

RMSProp

Для того чтобы исправить проблему AdaGrad'а надо каким-то образом постепенно забывать значения градиента в прошлых точках. Например, можно домножать текущий G на некоторый коэффициент γ:

$$G_{i+1} = \gamma G_i + (1-\gamma)(\nabla f(w_i))^2, \qquad w_{i+1} = w_i - \frac{\alpha}{\sqrt{G_{i+1}+\varepsilon}}\nabla f(w_i),$$

здесь все операции также проводятся покомпонентно.

Adam

Adam объединяет в себе идеи всех предыдущих методов. Теперь мы и сглаживаем градиент, и нормируем переход по каждой координате. Заметим, что в начале при усреднении вектора v его модуль сильно уменьшается в сравнении с градиентом, т.к. $v_1 = 0$. Поэтому будем делить v_i на $1-\beta_1^i$, чтобы компенсировать его уменьшенный модуль в начале. Метод описывается следующими тремя последовательностями:

$$v_{i+1} = \beta_1 v_i + (1-\beta_1)\nabla f(w_i), \qquad G_{i+1} = \beta_2 G_i + (1-\beta_2)(\nabla f(w_i))^2,$$

$$w_{i+1} = w_i + \frac{\alpha}{\sqrt{G_{i+1}+\varepsilon}\cdot(1-\beta_1^{i+1})}v_{i+1}.$$

Понятно, что все методы распространяются на случай mini-batch и стохастического спуска. В таком случае в формулах будет присутствовать градиент не по всей функции, а лишь по слагаемым из батча.

Задание 4. Сравнение работоспособности алгоритмов

Ниже приведены графики зависимости затрачиваемых ресурсов каждым из алгоритмов при разных размерах батча.

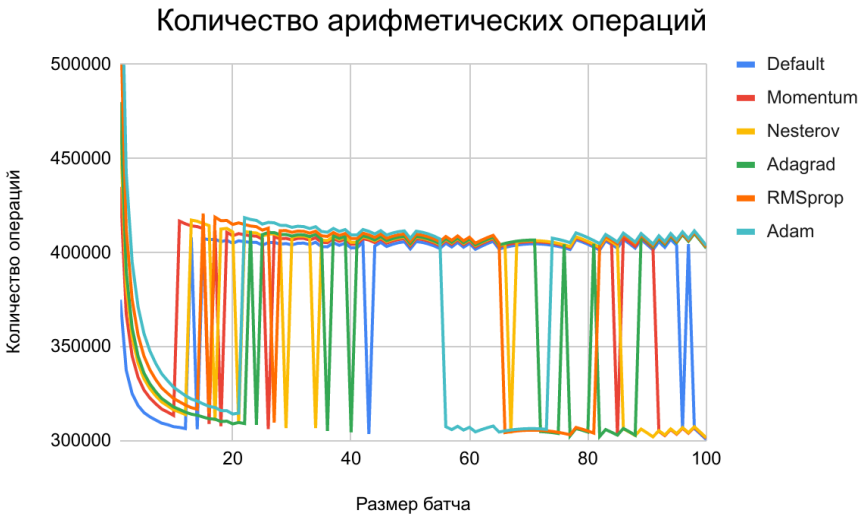

Рис. 2 – График зависимости количества арифм. операций от батча

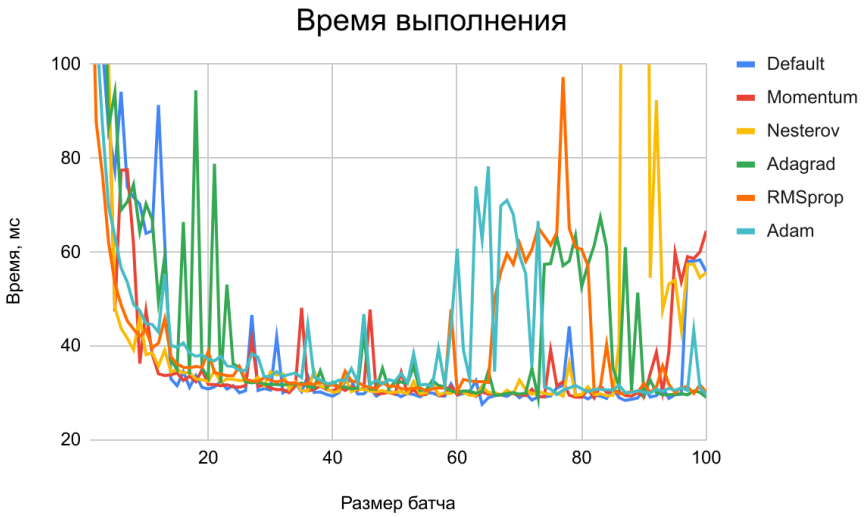

Рис. 3 – График зависимости времени исполнения от размера батча

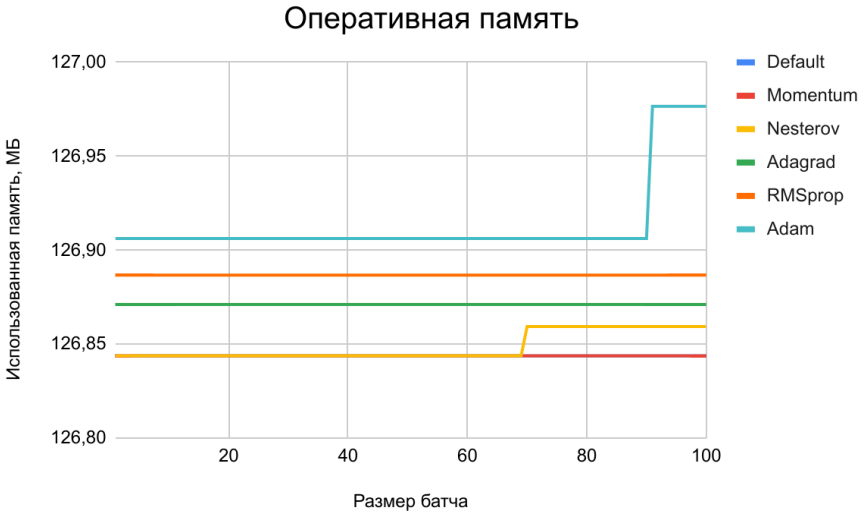

Рис. 4 – График зависимости использованной оперативной памяти в зависимости от размера батча

На графиках количества операций и времени выполнения явно выражается понятие “стохастический”. Видимо, определяющую роль в замерах играет то, какие именно точки для батчей мы выбираем. Кроме того, на графике потребления оперативной памяти видно, что Adam является самым требовательным, что понятно, ведь нам требуется пересчитывать больше последовательностей. Относительная разница в использовании памяти не велика, т.к. бóльшая ее часть отводится на код, который является общим у всех алгоритмов.

Запустим алгоритм с фиксированным размером батча N раз. Будем считать один запуск *достоверным*, если модуль разность найденного минимума и минимума, найденного аналитически, не превосходит некоторого γ. Тогда надежностью алгоритма будем называть отношение количества достоверных запусков к N.

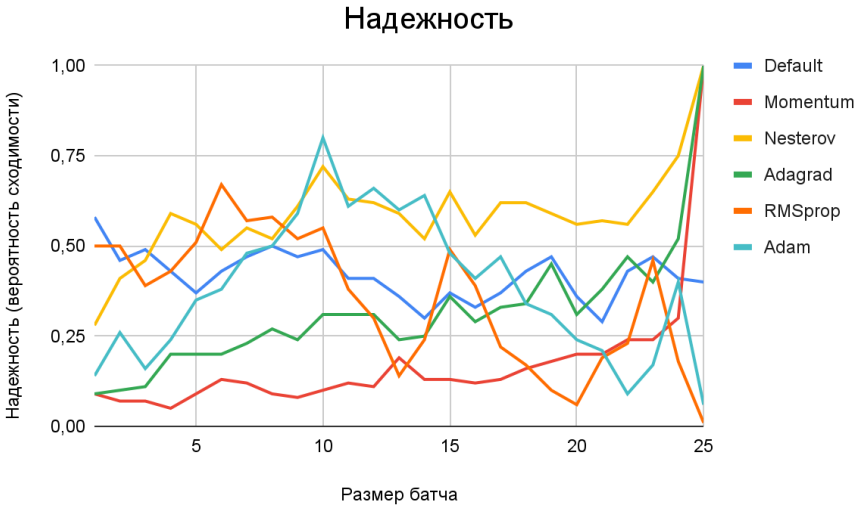

Рис. 5 – График зависимости вероятности сходимости в зависимости от размера батча.

Из графика можно заметить, что методы, усредняющие градиент, более надежны на больших батчах, в остальном же значения довольно случайные.

Задание 5. Построение траекторий спуска различных алгоритмов

Приведены траектории спуска различных алгоритмов, а также линейная функция, на которой достигается минимум (красная линия на графике справа – ответ, вычисленный аналитически), и линейная функция, к которой сошелся алгоритм (зеленая линия на графике справа). При этом алгоритм запускался на таком значении batch, на котором достигался минимум по точности в пределах каждого алгоритма.

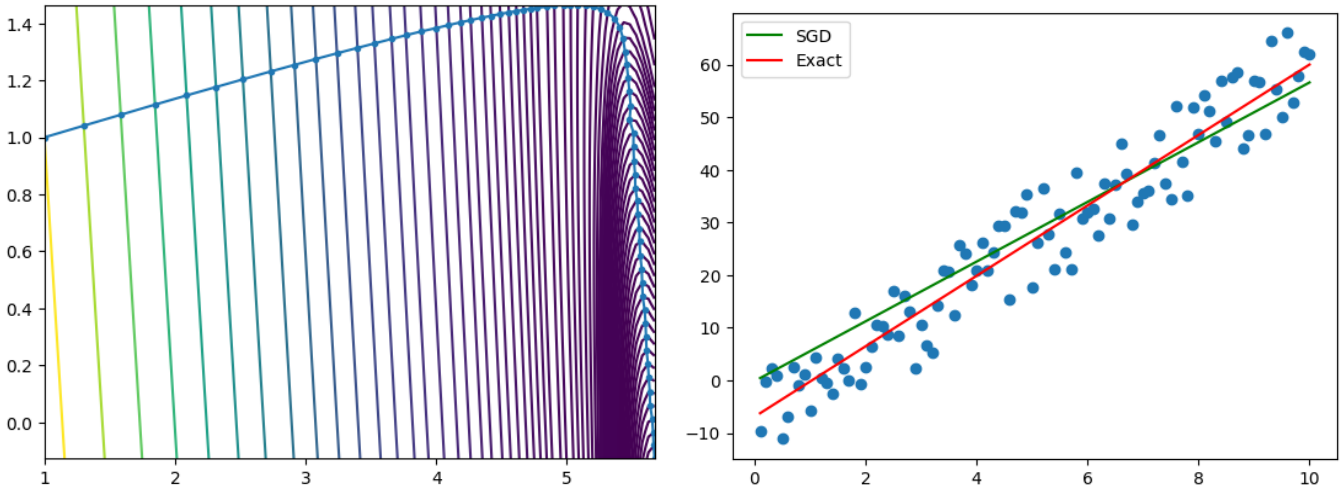

Рис. 6 – график Default при batch = 99, epoch = 50.

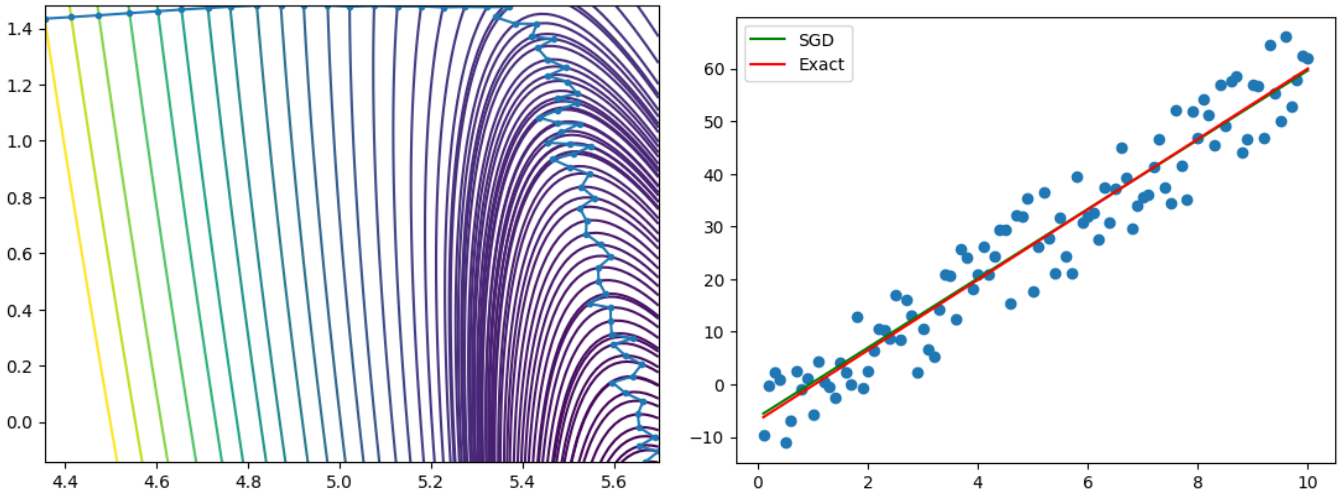

Рис. 7 – график Momentum при batch = 99, epoch = 50.

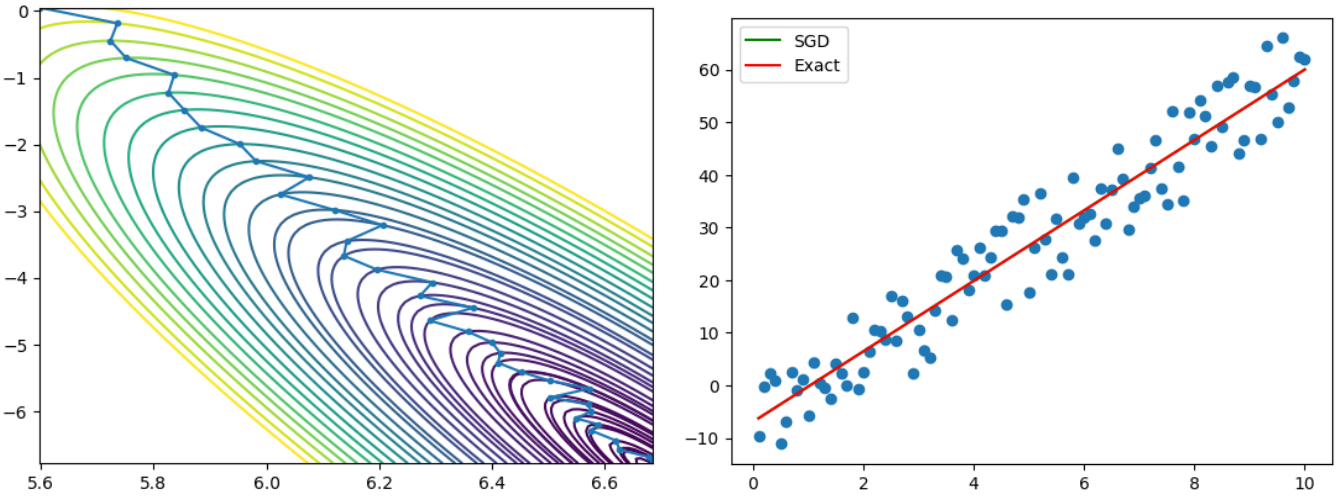

Рис. 8 – график Nesterov при batch = 100, epoch = 50.

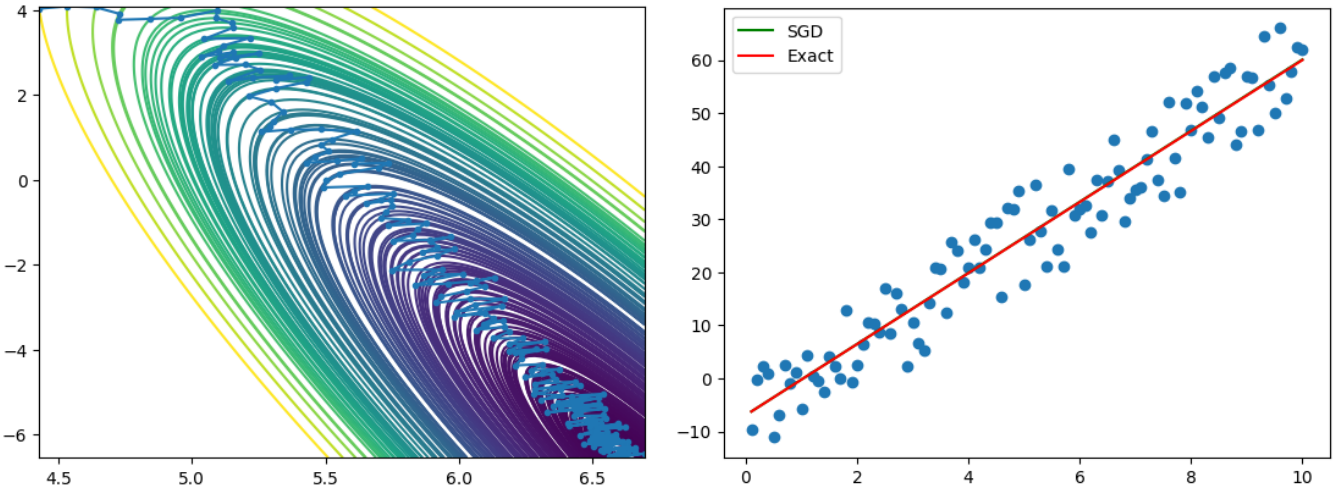

Рис. 9 – график Adagrad при batch = 28, epoch = 50.

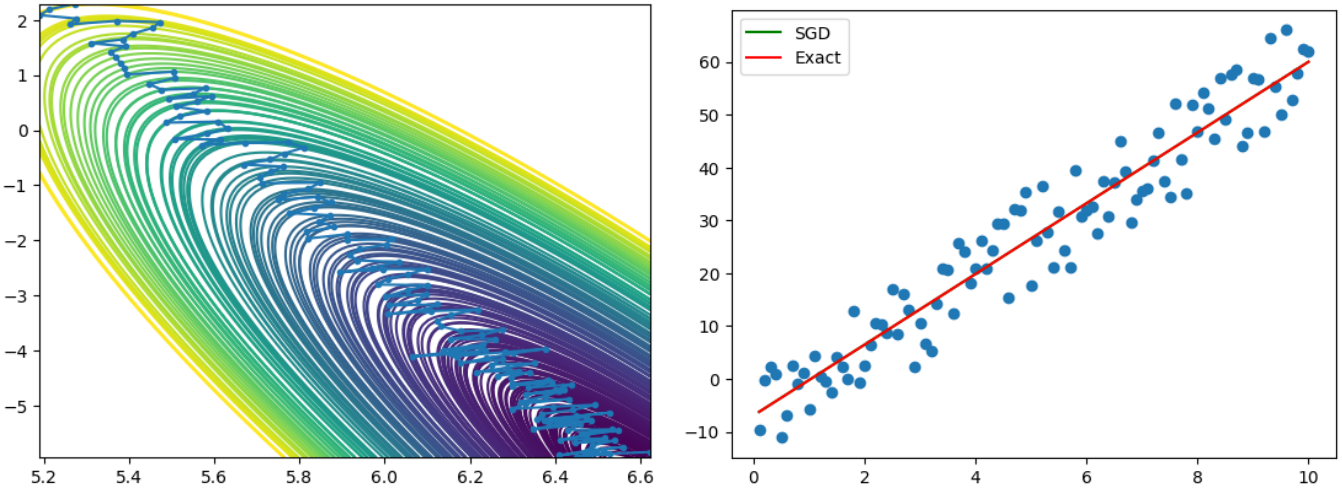

Рис. 10 – график RMSprop при batch = 5, epoch = 50.

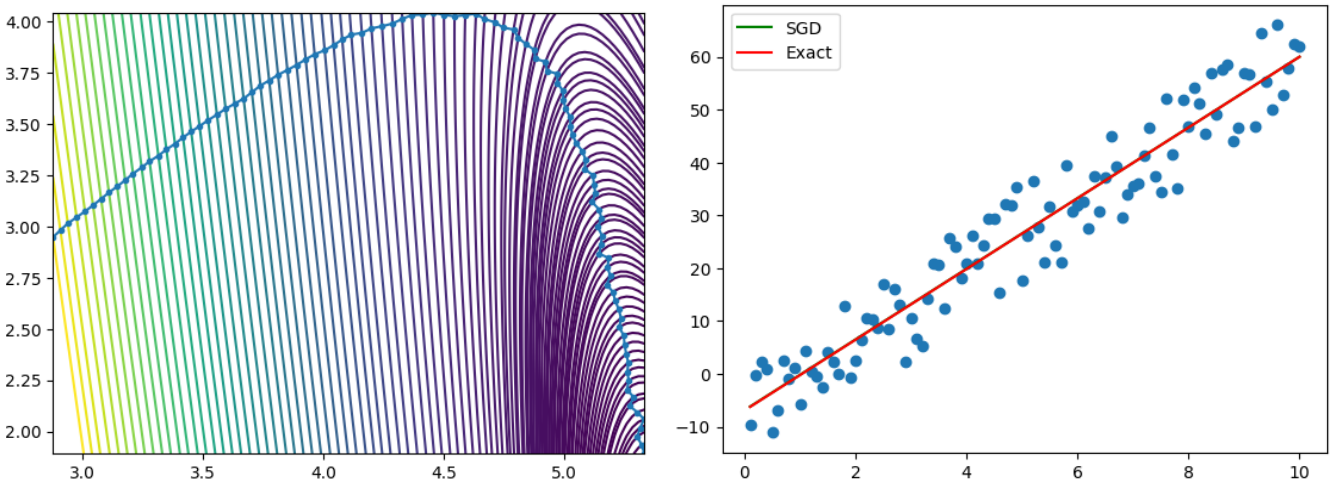

Рис. 11 -– график Adam при batch = 1, epoch = 50.

Замечание. Поскольку количество точек на траектории могло быть очень большим (и тогда бы траектория переставала быть наглядной), то некоторые точки из начала спуска откидывались. Также, чтобы линии уровней не становились очень плотными, точки “прореживались” – если расстояние от точки до предыдущей или следующей меньше, чем ε, то она исключалась из рассмотрения.

Заметим, что алгоритмы, нормирующие координаты накапливая градиент, лучше сходятся при маленьких батчах, т.к. в этом случае модуль градиента оказывается меньше, G растет не так быстро, и получается делать более длинные шаги, а эти алгоритмы упираются именно в этот параметр. В то же время методы, основывающиеся на сглаживании градиента, лучше работают на больших батчах, т.к. в этом случае выигрыш за счет ходьбы по отдельным координатам не так велик, мы все равно усредняем градиенты за последние шаги, однако при маленьких батчах сумма получается меньше по модулю и скорость уменьшается.

Глобальные выводы

Стохастический градиентный спуск обычно сходится за большее количество шагов в сравнении с обычным, однако шаги при этом выбираются значительно быстрее. При выборе промежуточного размера батча можно балансировать между количеством шагов и количеством взятий градиента.

Подбор функции пересчета коэффициента длины шага может обеспечить лучшую сходимость за счет уменьшения градиента по мере приближения к точке минимума.

В большинстве случаев модифицированные алгоритмы градиентного спуска сходятся значительно лучше стандартного. Однако, обычно это требует бóльшие вычислительные ресурсы, кроме того, для корректной работы необходим точный подбор констант. Самый часто используемый алгоритм: Adam, показывает лучшую сходимость при малом размере батча.