Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Отчет к практическому заданию №4 по МОМО: Минимизация суммы функций

Студент 517 группы:
Оспанов А.М.

Москва, 2016

1 Введение

В данной работе будут реализованы и протестированы два популярных метода для минимизации суммы функций: SGD (Stochastic Gradient Descent) и SVRG (Stochastic Variance Reduced Gradient).

Тестирование будет проводится на следующих двух моделях из области машинного обучения: 1) логистическая регрессия; 2) автокодировщик (глубокая нейронная сеть)

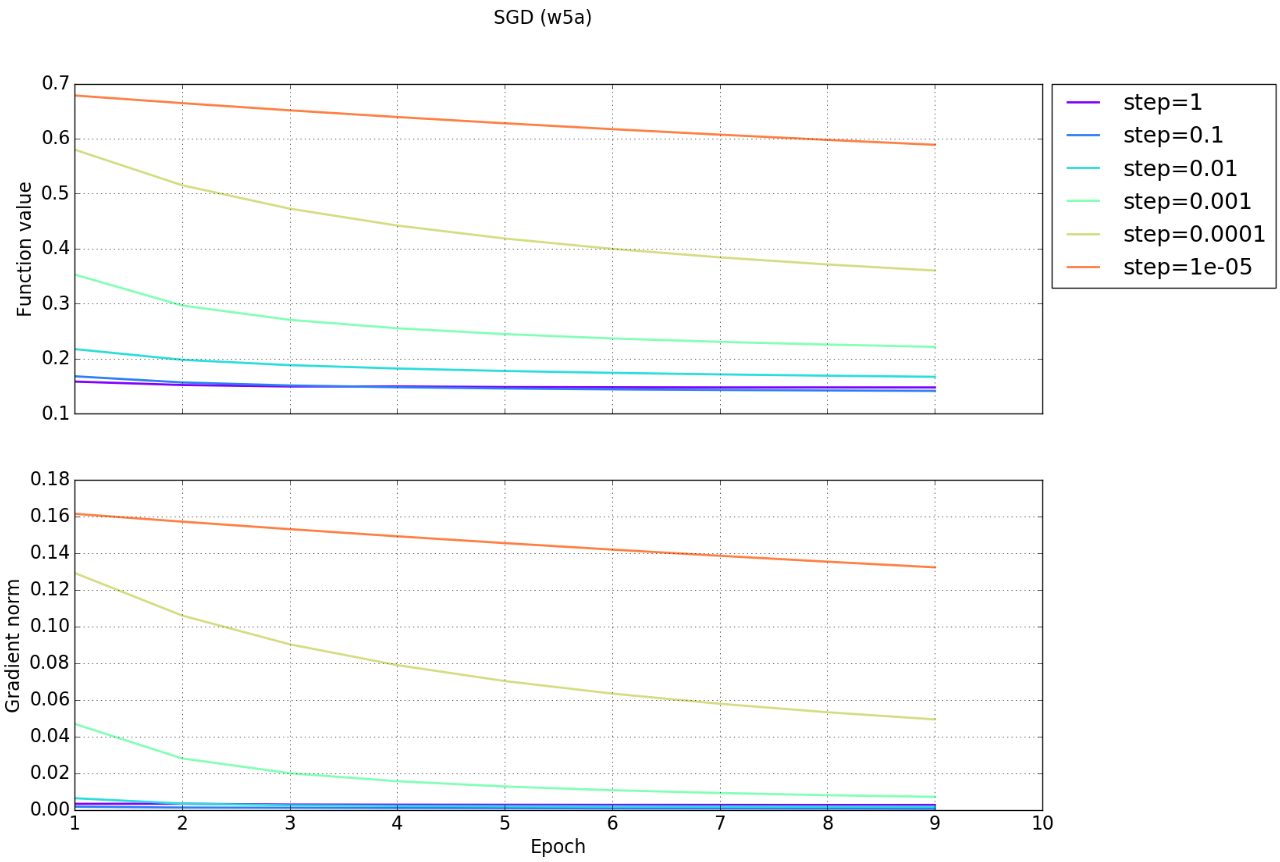
Код написан на языке Python 3 с использованием библиотеки numpy.

2 SGD (Stochastic Gradient Descent)

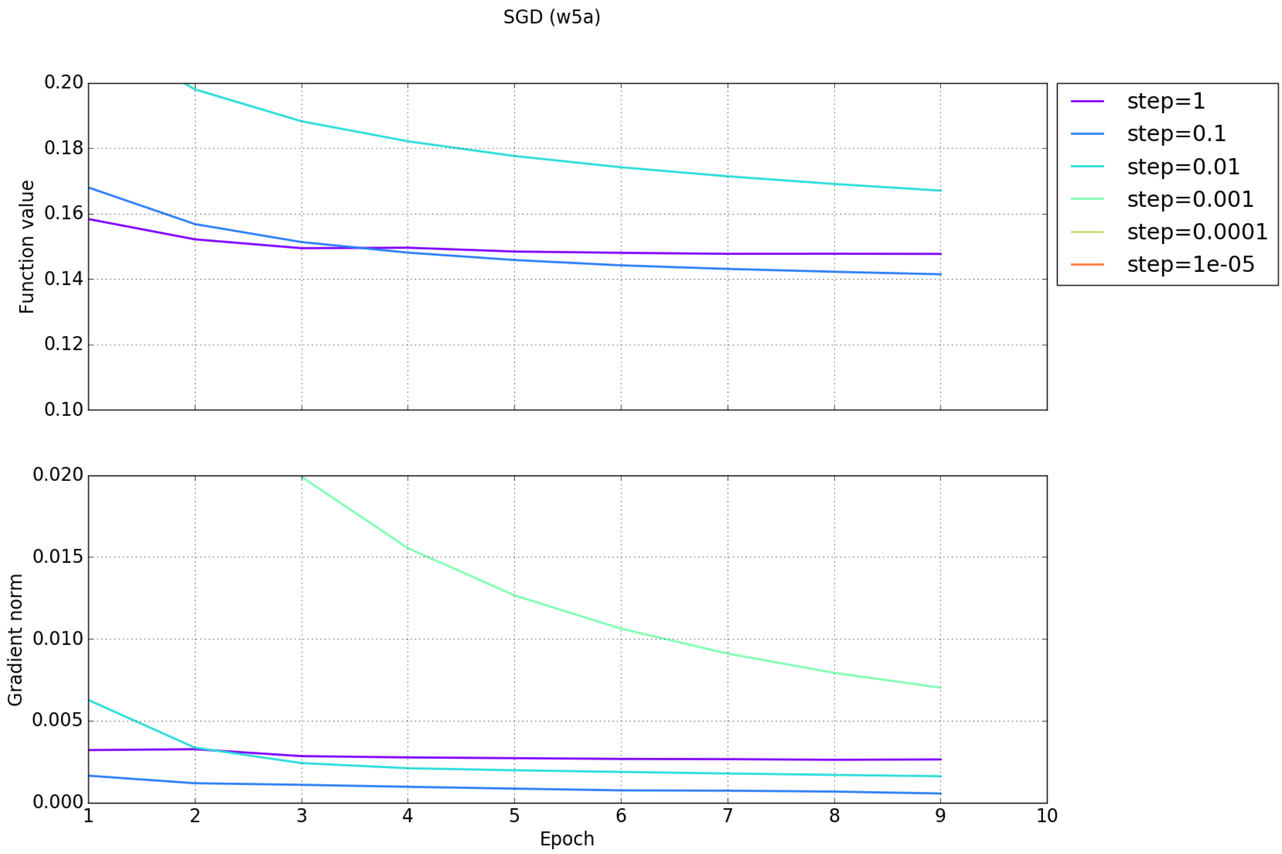
2.1 Зависимость скорости сходимости от длины шага

Запустим метод для модели логистической регрессии.

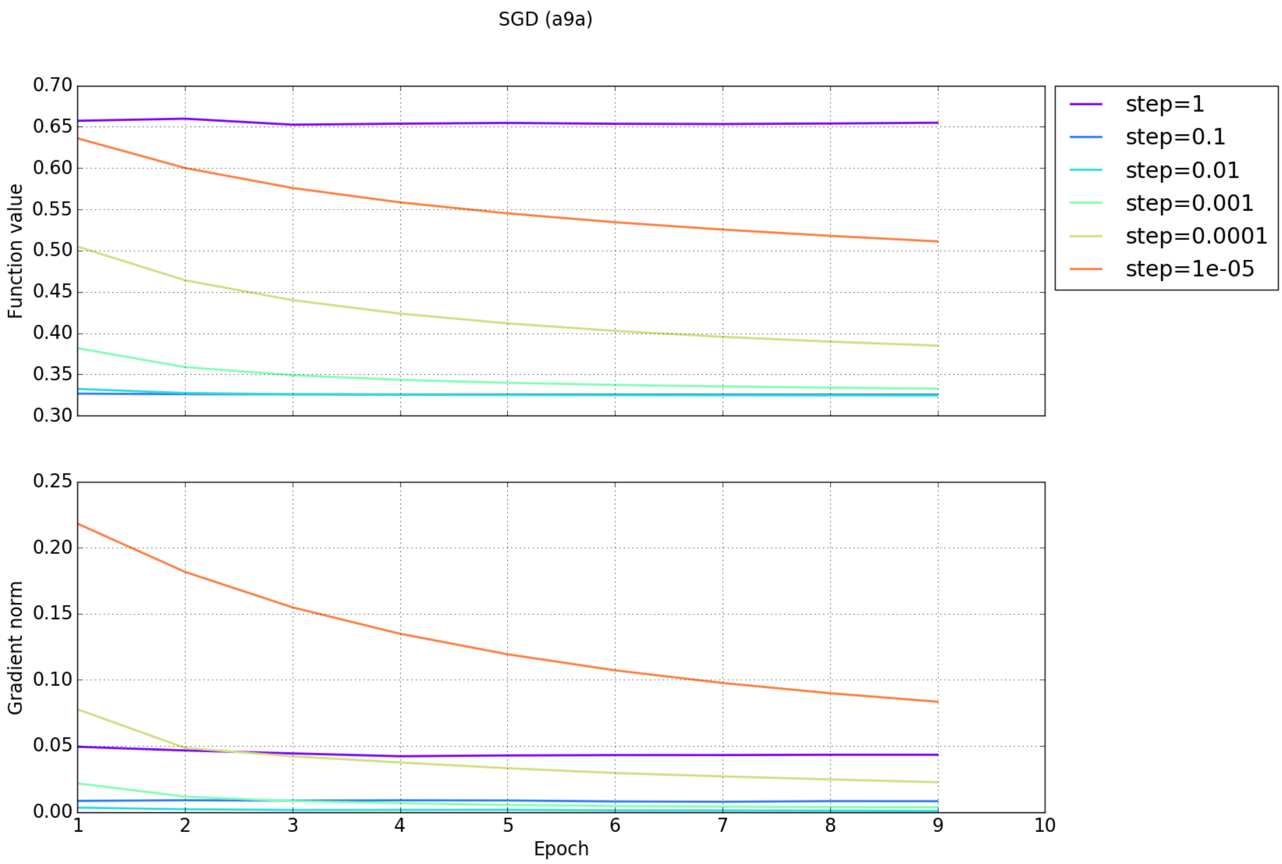
На данных w5a видно, что, чем больше шаг, тем быстрее сходится метод



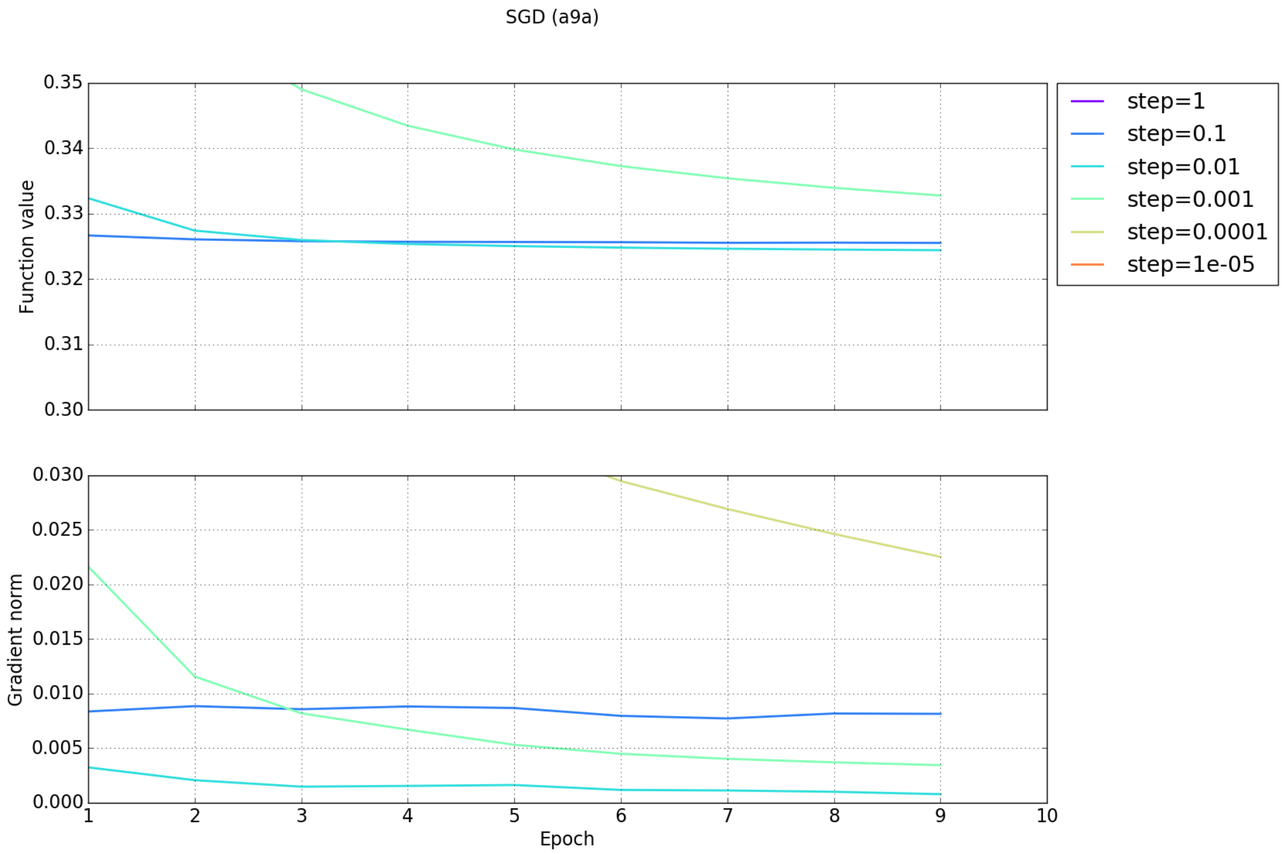
Посмотрим в увеличенном варианте графика. Тут хорошо видно, что при шаге 0.1, метод сходится быстрее. Т.е. пока будем считать, что выбрали шаг = 0.1.



Далее посмотрим график для данных a9a. Тут примерно тот же случай, что и с w5a, и чем больше шаг, тем быстрее метод сходится.

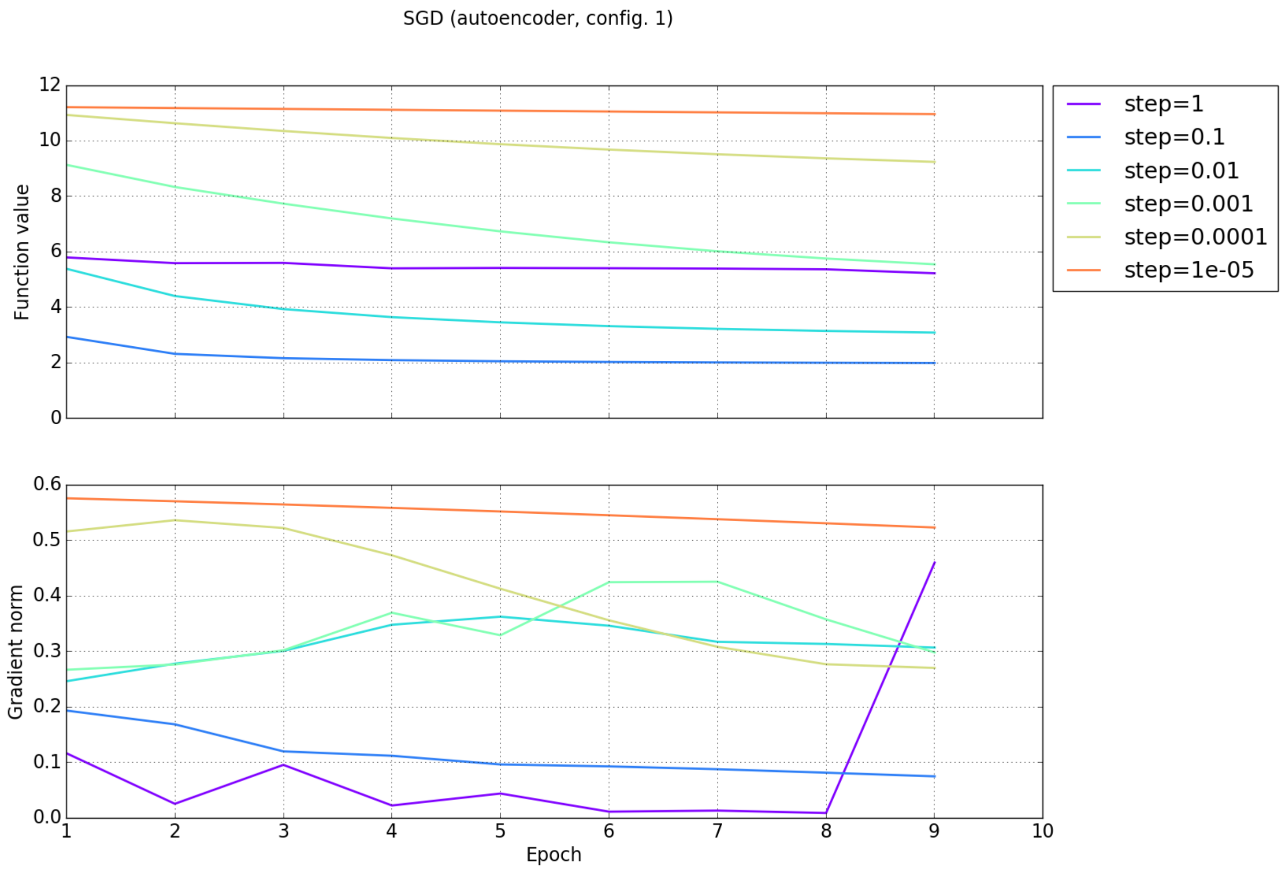


Посмотрим увеличенный вариант. В этом случае, шаг 0.1 показывает хорошие результаты, но метод быстрее при step $= 0.01$. У нас появился еще один кандидат.

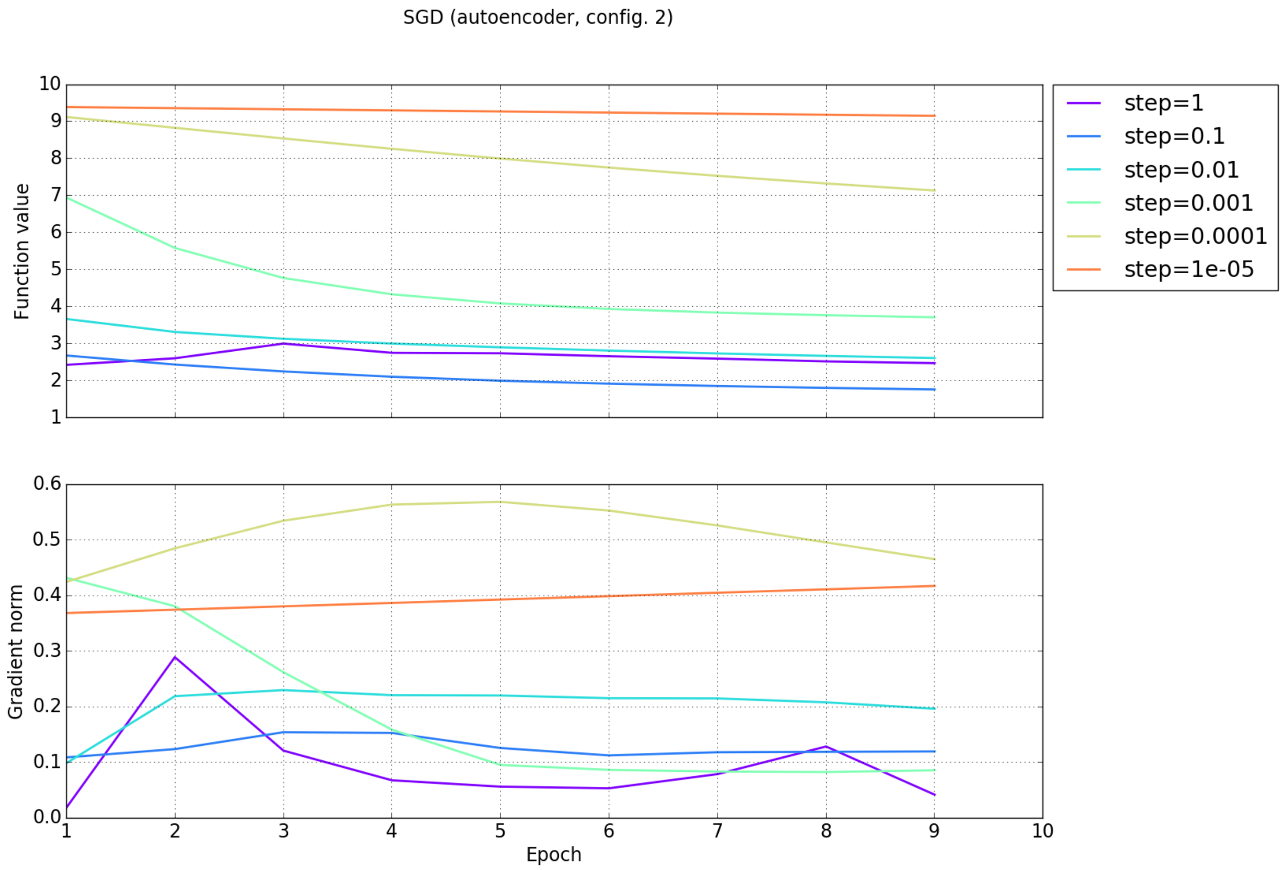


Теперь запустим метод для автокодировщика.

Посмотрим график для атокодировщика с конфигурацией 1. Тут безусловно step $= 0.1$ показывает лучший результат.



Аналогично и с автокодировщиком с конфигурацией 2.



Таким образом, лучшим размером шага, при котором метод сходится быстро является 0.1

3 SVRG (Stochastic Variance Reduced Gradient)

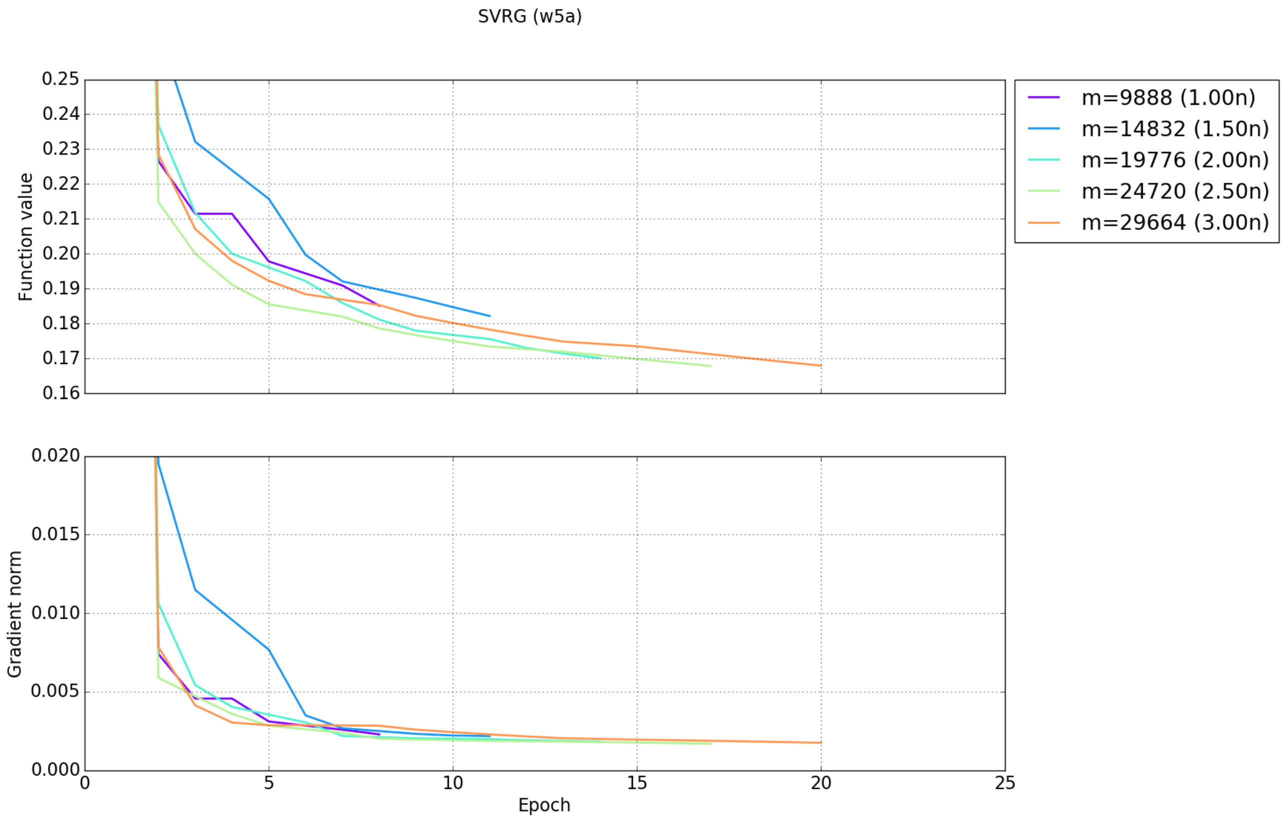
3.1 Оптимизация скорости работы SVRG

В базовом методе SVRG во внутреннем цикле высчитываются градиенты для случайного i из равномерного распределения. Но при этом, для всех $i \in 1, \dots, n$ градиенты высчитаны во внешнем цикле. Т.е. если мы будем хранить эти градиенты, то вызов оракула во внутреннем цикле не понадобится. Таким образом, мы получаем выигрыш в скорости на $O(n)$ (n вызовов оракула). Но при этом мы проигрываем по памяти на $O(n * d)$ (d - размерность пространства). Но считаем, что в памяти мы не ограничены. Результаты тестов (тесты были сделаны на данных w5a и a9a) показали, что прирост в скорости 30%, что является очень хорошим результатом.

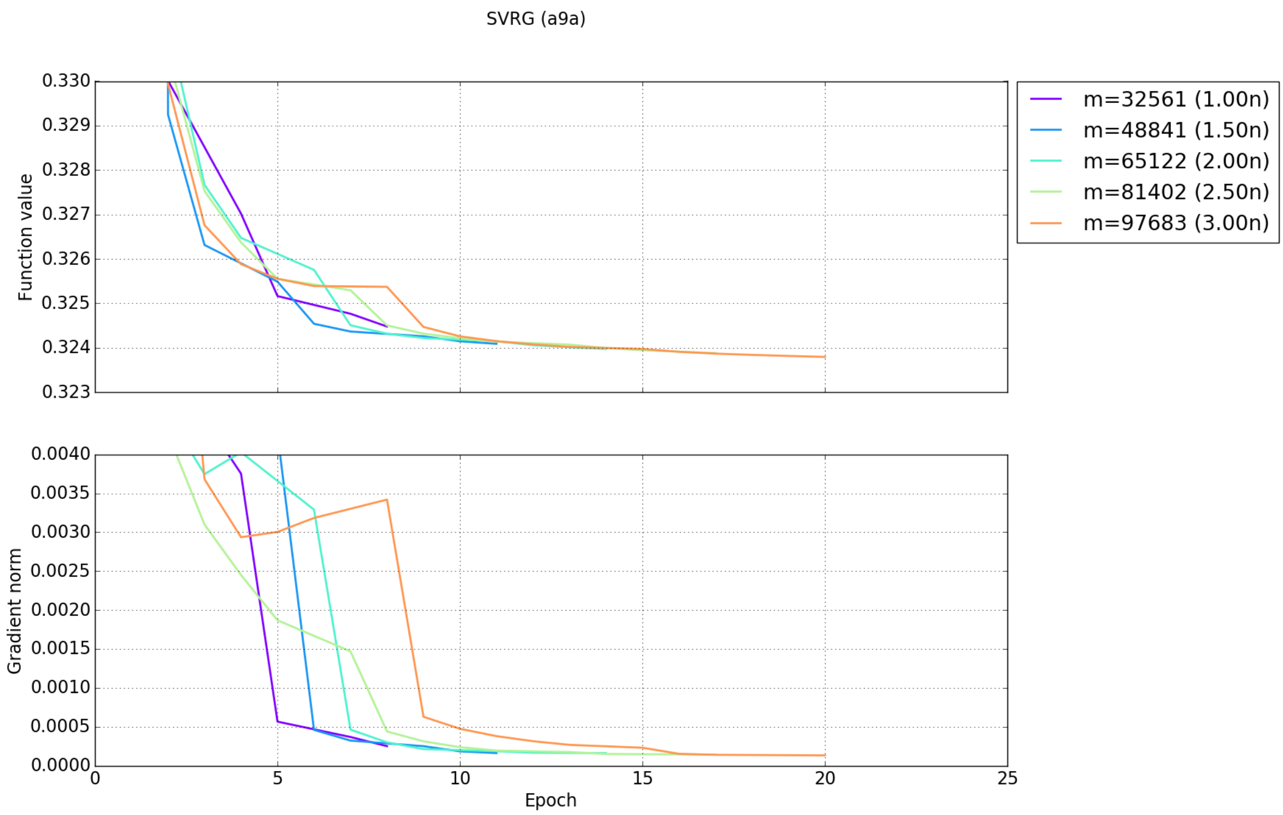
3.2 Скорость сходимости в зависимости от числа итераций внутреннего цикла

Запустим метод для модели логистической регрессии.

На данных w5a видно, что метод ведет себя примерно одинаково для всех m из теста. Но также видно, что начиная с $m = 2n$, точность не сильно увеличивается, но время работы увеличивается достаточно.

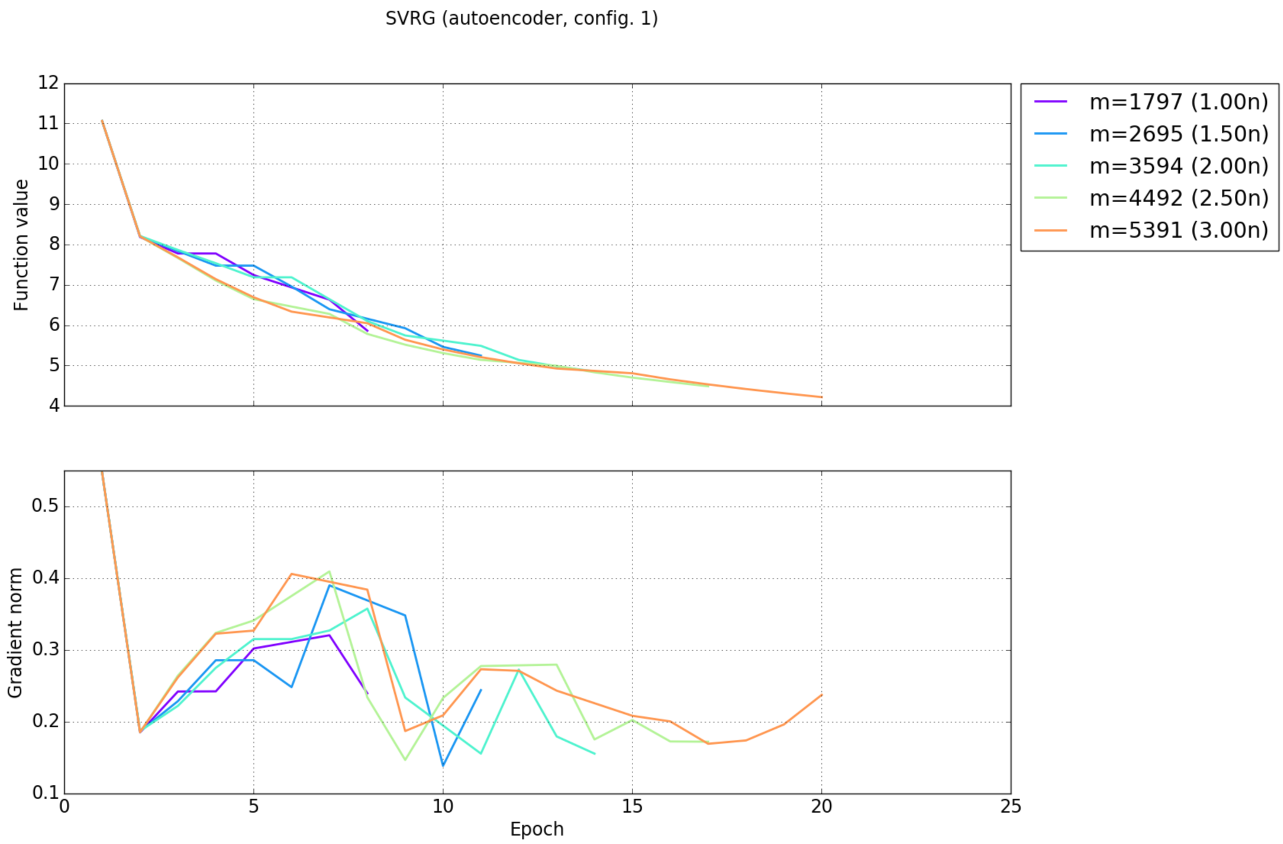


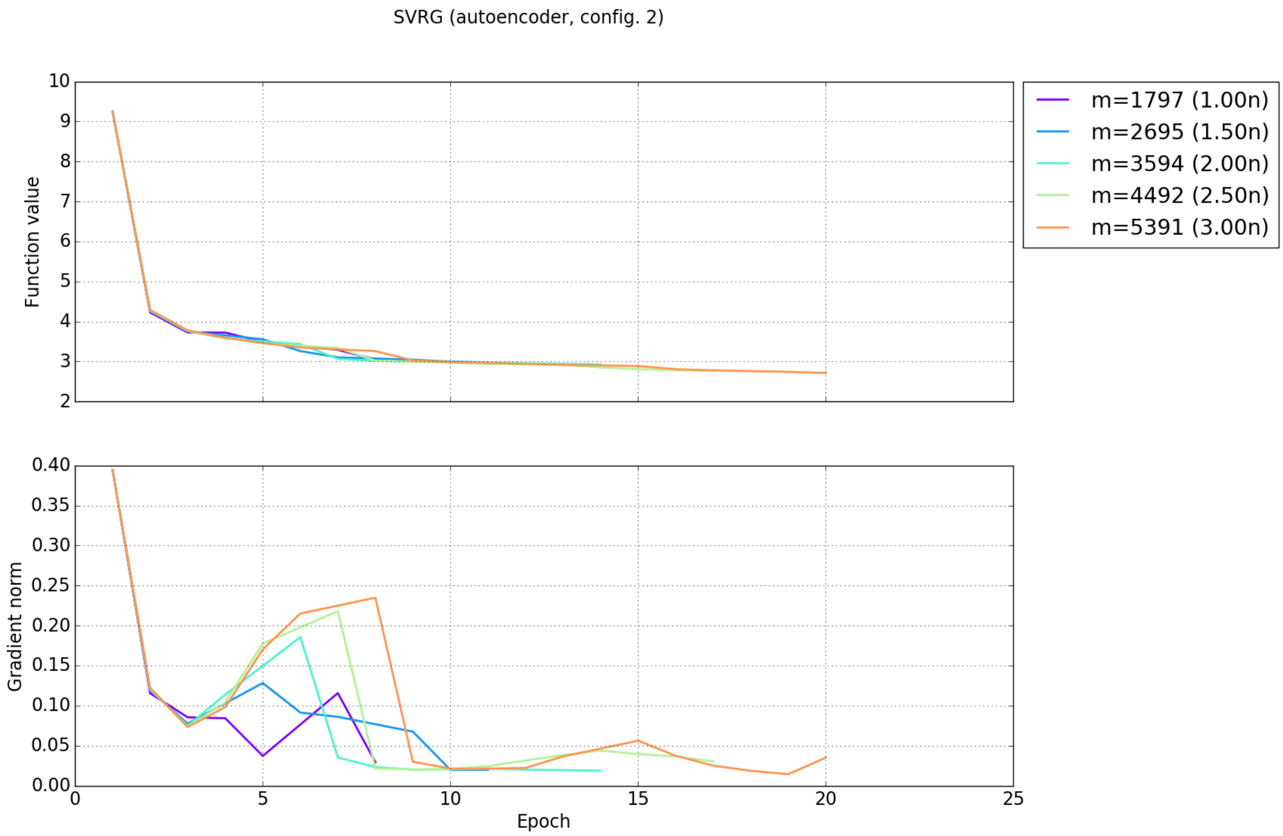
Далее посмторим график для данных a9a. Тут примерно тот же случай, что и с w5a.



Теперь запустим метод для автокодировщика.

Посмотрим график для атокодировщика с конфигурацией 1 и с конфигурацией 2. Тут ф-ции сходятся также примерно одинаково, но интересен график градиентов. Хотя градиент "скачет", ф-ции сходятся. Это объясняется тем, что оптимизационная задача автокодировщика не является выпуклой. Такой же эффект можно видеть и для SGD, но в случае SGD график более гладкий, потому, что SGD не использует сумму градиентов.





Таким образом, число нутренних итераций выбирется равным $2n$ (где n - число функций).

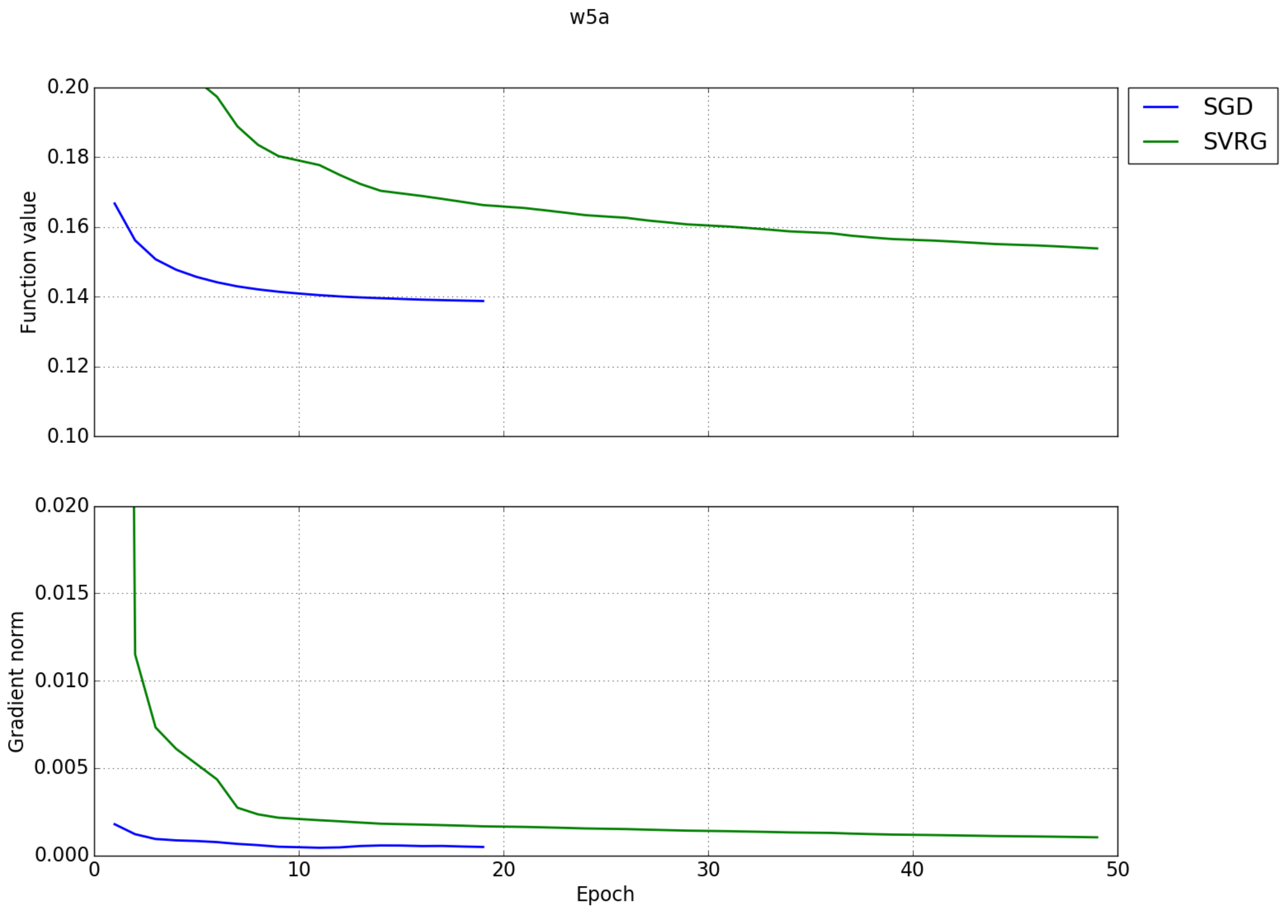
4 Сравнение SGD и SVRG

Параметры SGD: длина шага $= 0.1$; количество итераций $20n$

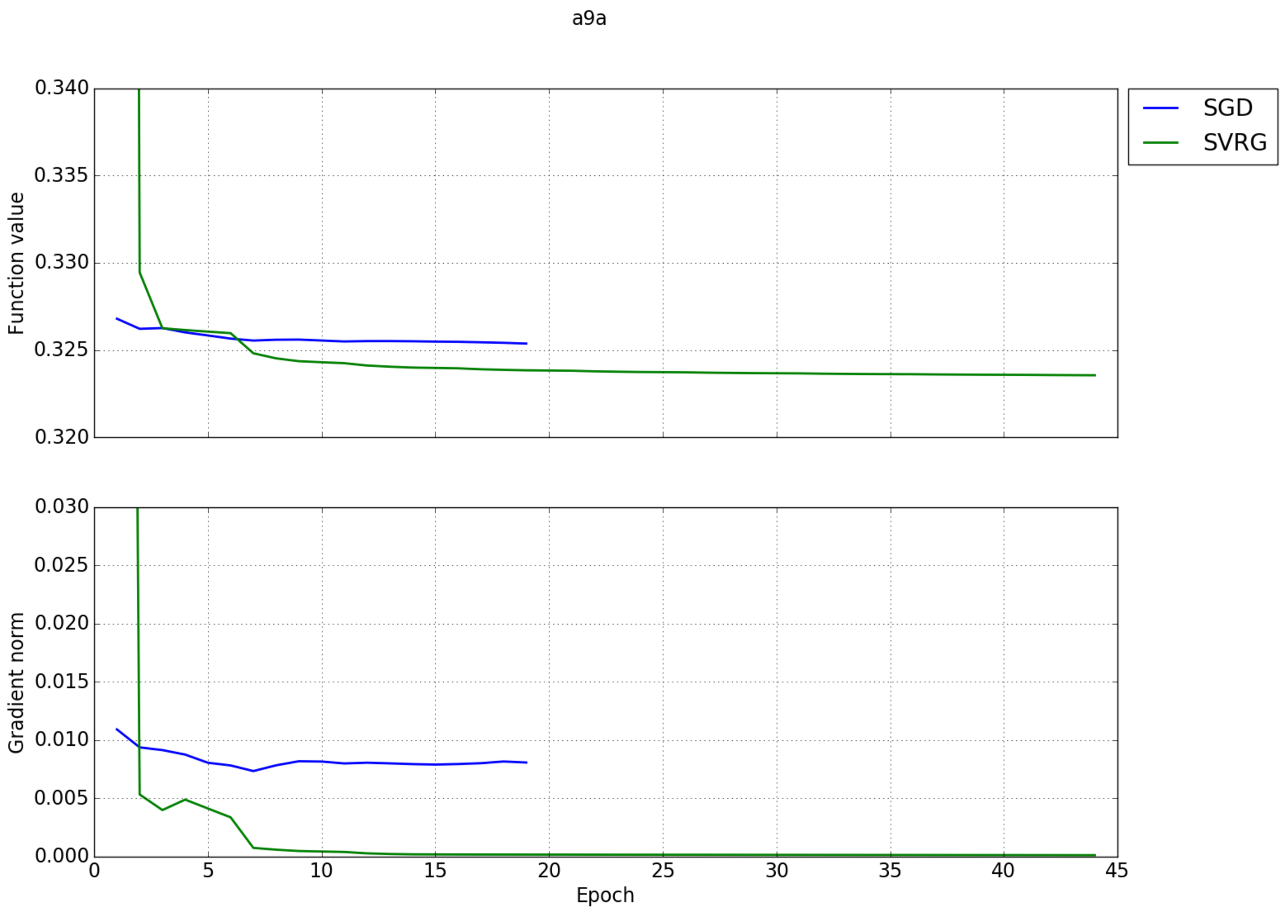
Параметры SVRG: число стадий 10

Сперва сравним методы на задаче логистической регрессии.

Хорошо видно, что на данных w5a метод SVRG не может достичь минимума SGD даже проделав более чем в два раза больше эпох. Хотя теоретически, SVRG должен работать лучше. Допустим, что это из-за данных и сравним методы на данных a9a.

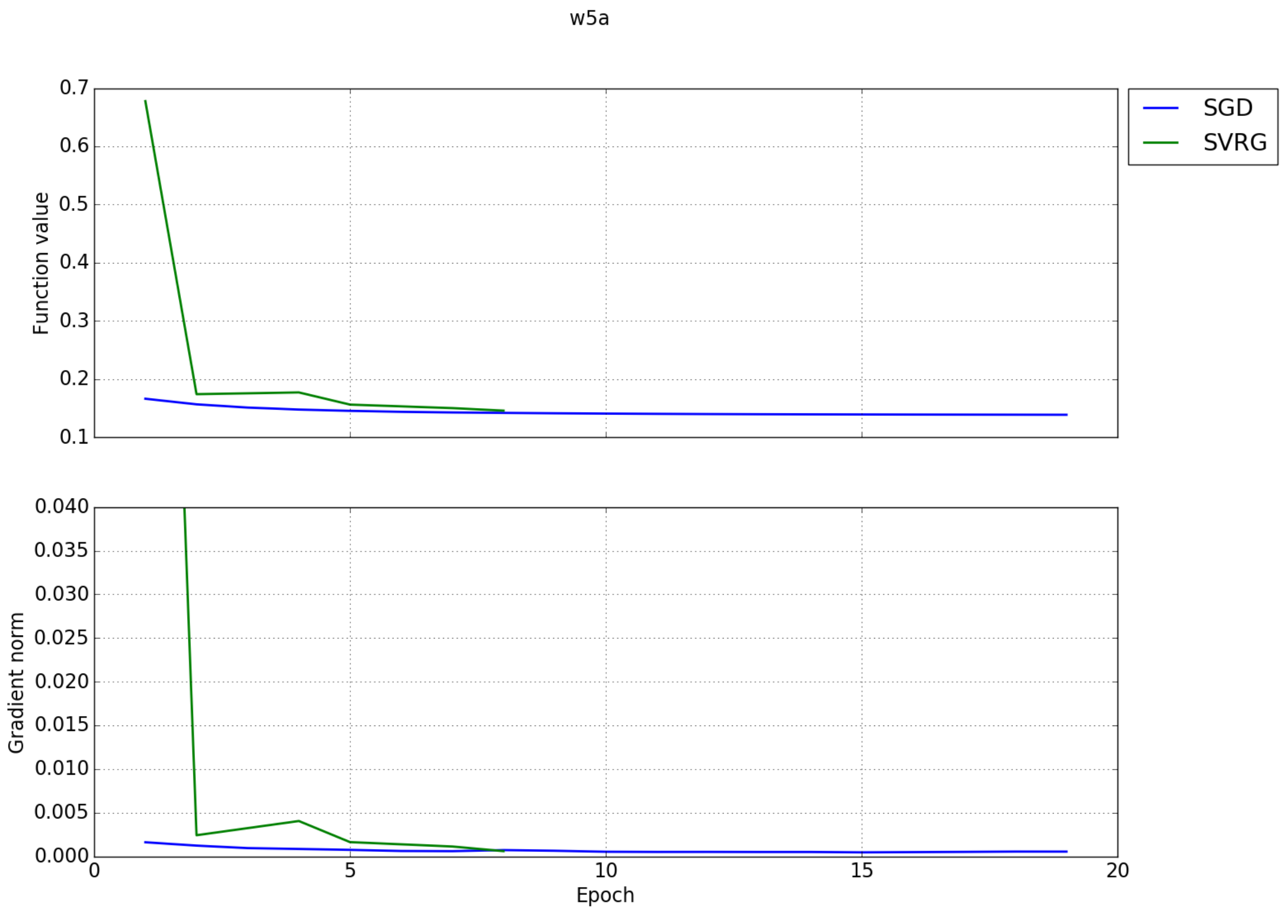


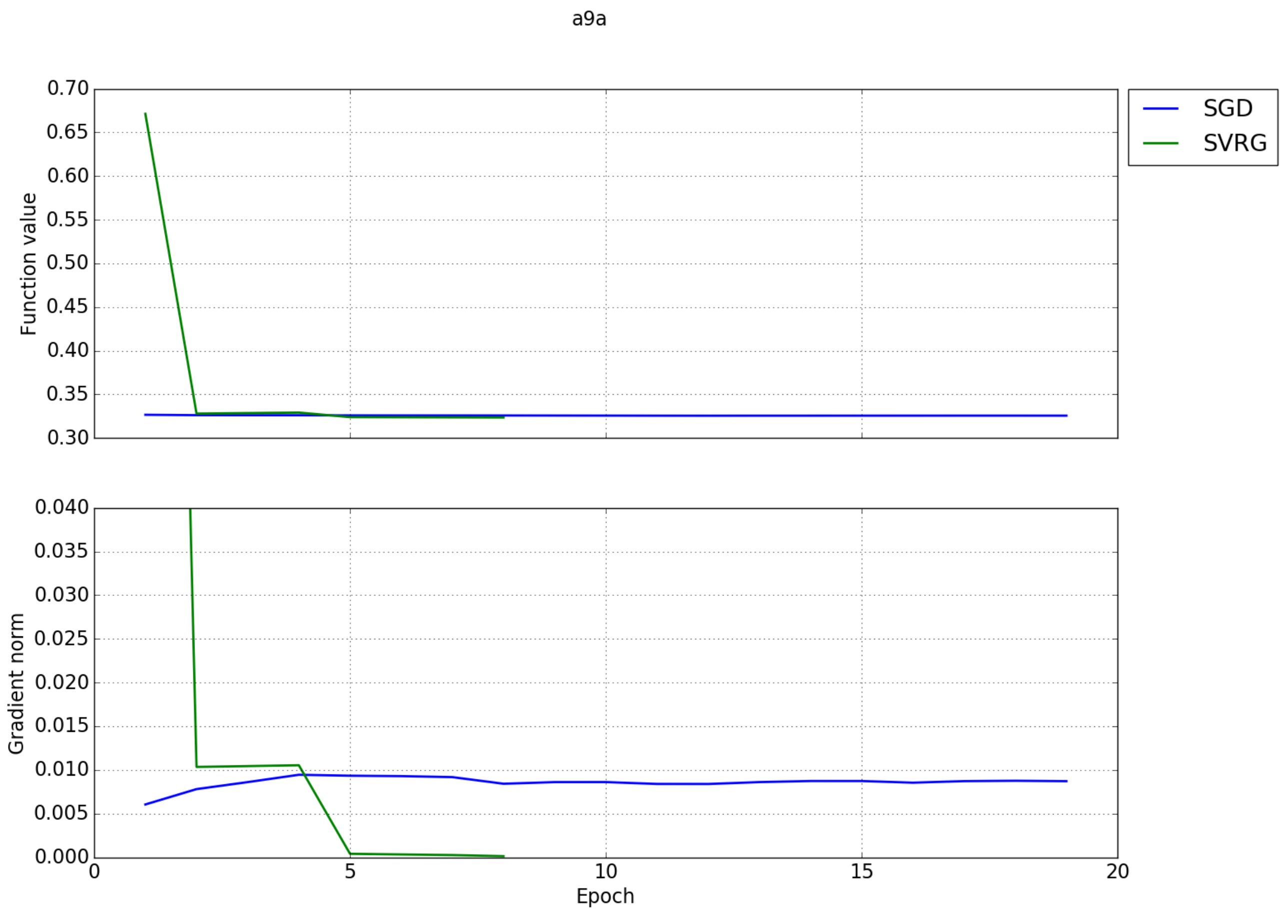
На данных a9a вроде бы все хорошо и SVRG работает лучше.



Но почему на w5a SVRG работает хуже SGD? Может из-за выбора шага? Давайте проверим, поставив шаг равным константе 0.1. Т.е. $L = 1$ (число стадий 3)

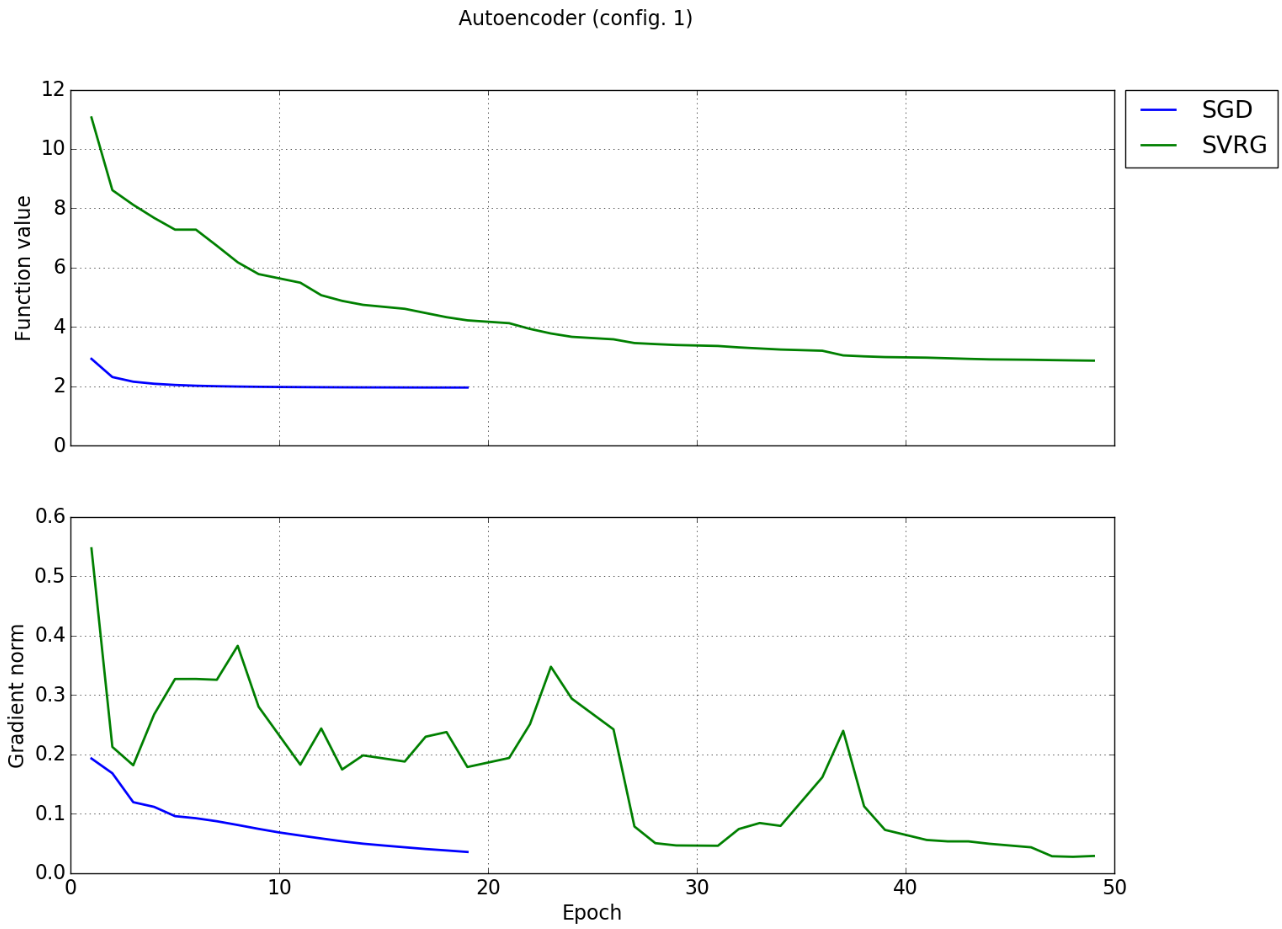
На графиках видно, что SVRG все таки быстрее сходится и даже "проваливается" ниже SGD. Но почему метод Нестерова работает не так хорошо, как хотелось бы? Ответ на этот вопрос пока остается загадкой =)

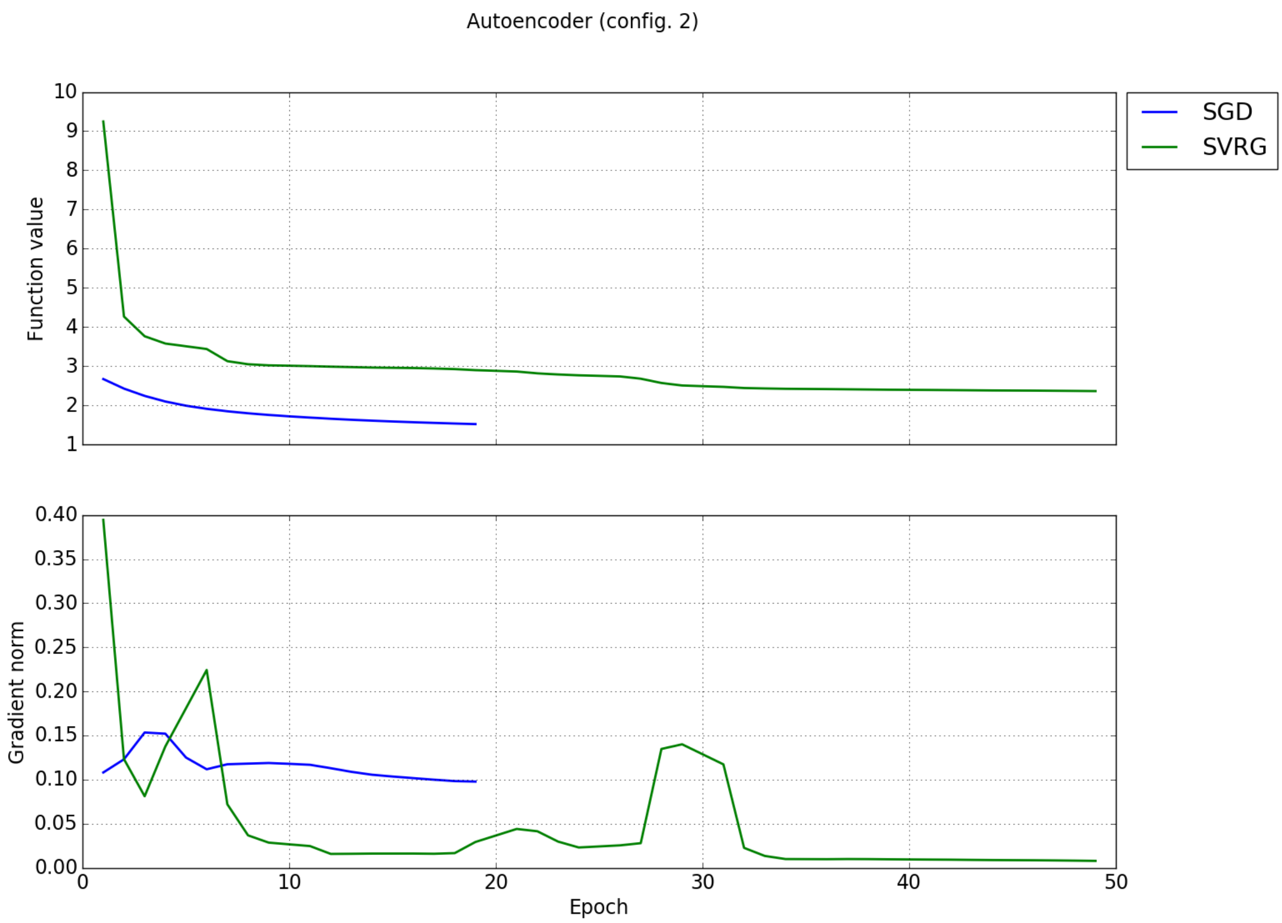




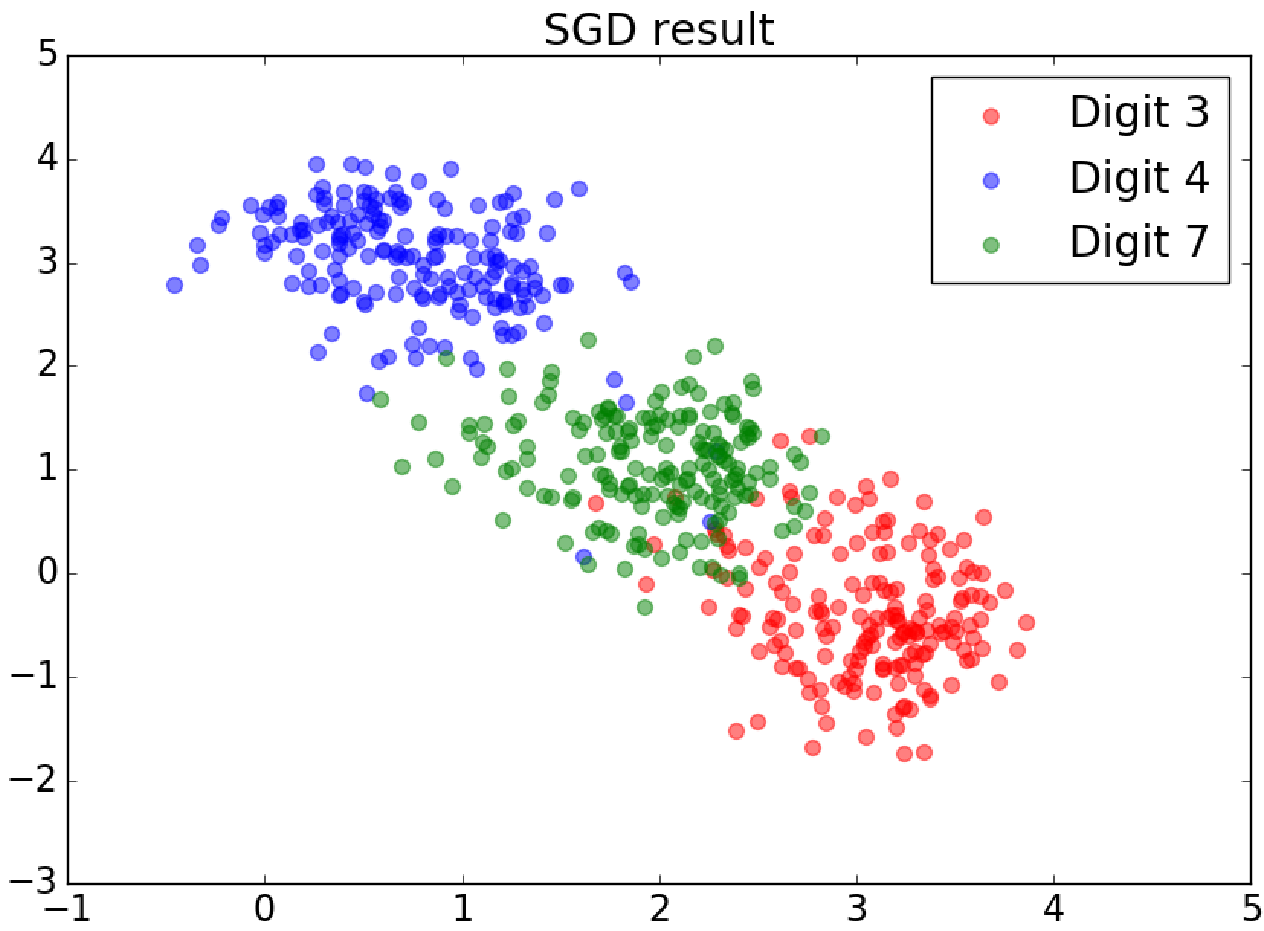
Теперь посмотрим сравнение скоростей для автокодировщика.

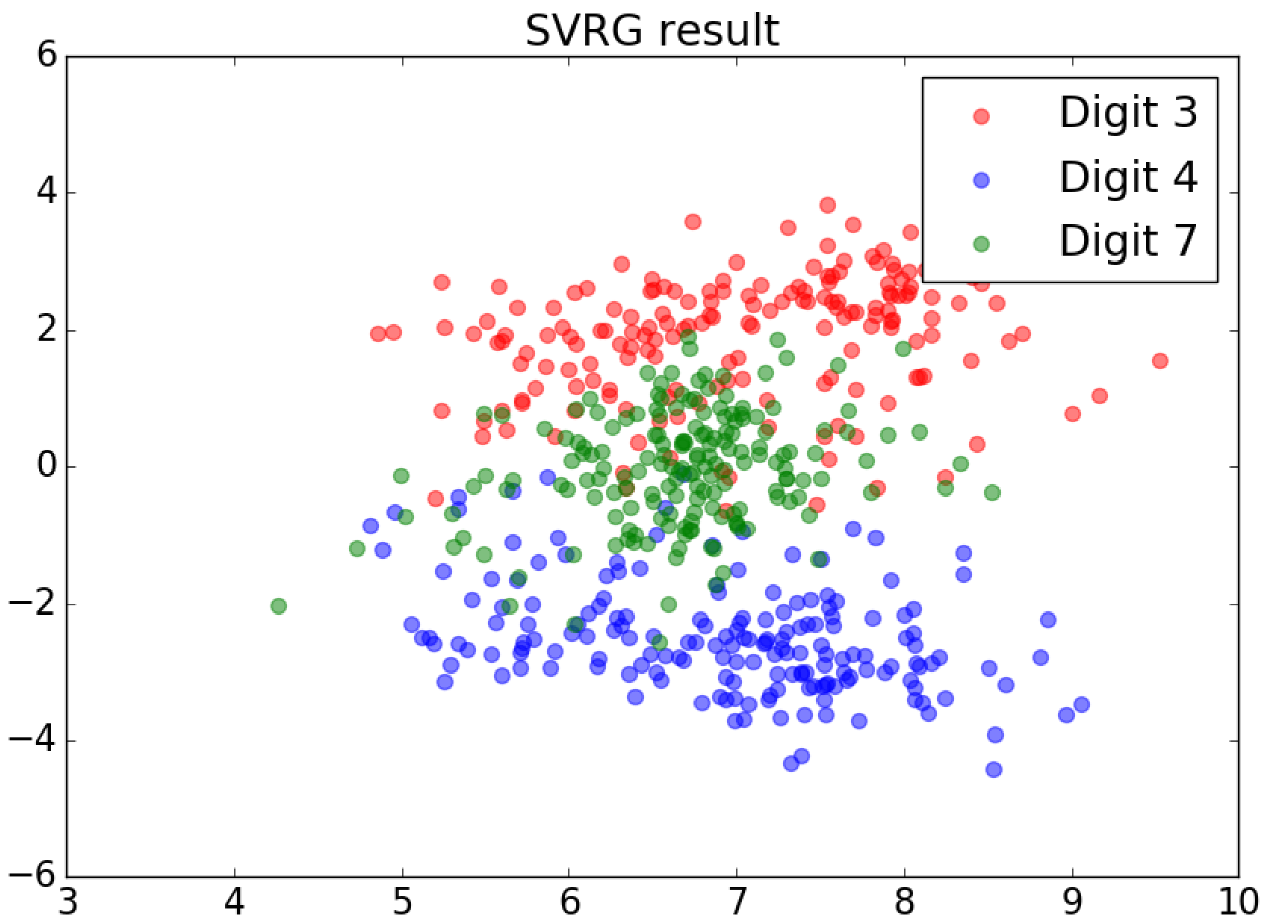
Тут SVRG никак не лучше SGD, и скорее всего это из-за эмпирического подбора шага.





Также, разница между работой SGD и SVRG видна на результатах работы двух методов:


SGD result
Digit 3
Digit 4
Digit 7


SVRG result
Digit 3
Digit 4
Digit 7

5 Заключение

В работе были исследованы методы SVRG и SGD и получены следующие результаты:

1. Метод SGD бысрее сходится, чем SVRG с подбором шага по методу Нестерова.

2. Метод SGD бысрее и точнее сходится при размере шага $= 0.1$.

3. Метод SVRG с постоянным размером шага бысрее сходится, чем SGD.

4. Метод SVRG бысрее и точнее сходится при количестве внутренних итераций равному $2n$.

5. Метод SVRG работает быстрее, если сохранять градиенты во внешнем цикле.

6. Метод SVRG с постоянным размером шага бысрее сходится, чем SVRG с эмпирическим подбором шага.