Методы оптимизации
Отчёт по лабораторной работе №3

Выполнили:

Евсеева Арина М3234

Шульпин Егор М3232

Гаврилюк Виталий М3232

Постановка задачи

1. Предисловие. Задача линейной регрессии;

2. Стохастический градиентный спуск с разным размером батча;

3. Стохастический градиентный спуск с функцией изменения шага;

4. Модификации стохастического градиентного спуска (с использованием TensorFlow):

- **Momentum;**
- **Nesterov;**
- **Adaptive gradient algorithm (AdaGrad);**
- **Root Mean Square Propagation (RMSProp);**
- **Adaptive Moment Estimation (Adam);**

5. Стохастический градиентный спуск для полиномиальной регрессии с добавлением регуляризации в модель разных методов регуляризации (L1, L2, Elastic);

6. Решение одной из задач машинного обучения.

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:

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

Стохастический градиентный спуск с разным размером батча

Нам может потребоваться много итераций, чтобы градиентный спуск сошелся. Более того, время выполнения одной итерации может быть очень большим хотя бы потому, что нам нужно каждый раз просмотреть весь датасет. Поэтому для каждого шага градиентного спуска будем использовать не точный градиент, а его оценку: выберем несколько батчей — посчитаем на них градиенты и усредним. Такой вид градиентного спуска называют стохастическим.

(SGD — stochastic gradient descent)

Стохастический градиентный спуск с одноэлементным батчем

Из обучаюшего набора $p_1, p_2, ..., p_m$ будем выбирать случайный элемент p_j. Функция потери вычисляется только на одном элементе p_j:

$$H_j(w) = L(w, p_j).$$

Точнее нам нужна не сама функция потерь, а ее градиент по w:

$$g = \nabla H_j(w)$$

Этот вектор g используем на очередном шаге стохастического градиентного спуска для вычисления следующего приближения w_{k+1} параметров модели по очередному приближению w_k:

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

Стохастический градиентный спуск мини-пакетом (Mini-Batch)

На каждом шаге используем пакет из фиксированного небольшого числа элементов обучающей выборки.Использование небольшого набора случайных элементов обучающей выборки для приближенного вычисления градиента функции потери позволяет уменьшить случайные вариации вектора градиента по сравнению с использованием только одного элемента. Мы фиксируем небольшой размер smини-пакета. Выбираем набор из s элементов обучающей выборки, $s \leq m$:

$$B = (p_{i_1}, p_{i_2}, ..., p_{i_s}) \subseteq (p_1, p_2, ..., p_m)$$

Для мини-пакета B из s элементов функция потери вычисляется следующим образом:

$$H_B(w) = \frac{1}{s} \cdot \sum_{p \in B} L(w, p)$$

Градиент этой функции:

$$g = \nabla H_B(w)$$

используется на очередном шаге стохастического градиентного спуска. После выполнения очередного шага формируется новый мини-пакет, состоящий либо из s случайно выбранных элементов обучающего набора.

Использование стохастического градиентного спуска в задаче двухмерной линейной регрессии

Рассмотрим задачу линейной регрессии вида $y = wx + b$, которую наглядно можно показать с помощью двухмерных графиков. Функция ошибки в данном случае будет равна $H(w_k) = \sum_{i=1}^{n}(w_k x_i + b - y_i)^2$, а ее градиент по i-ому направлению - $\nabla H_i(w_k) = 2x_i \cdot \sum_{i=1}^{n}(w_k x_i + b - y_i)$.

Создадим тестовый набор данных X, Y, например:

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

Полученные результаты:

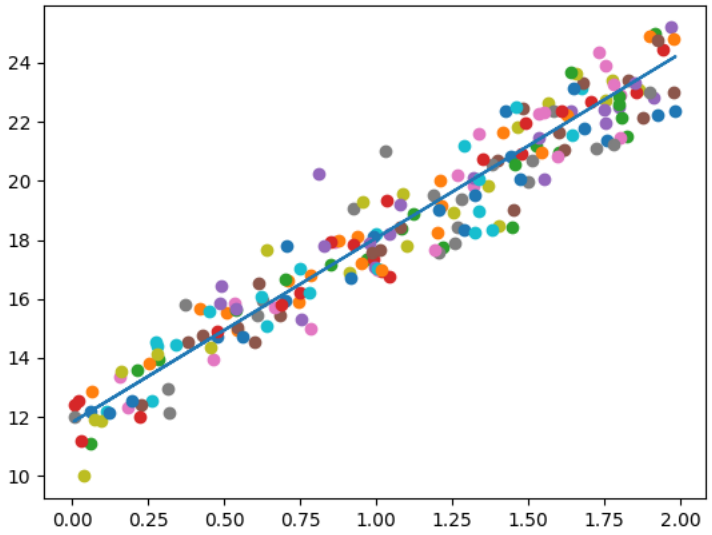

Рис. 1: Результат с размером батча 1

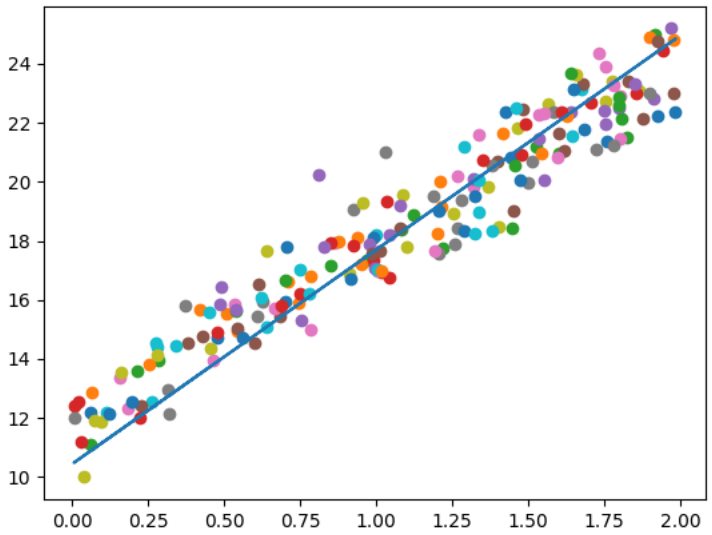

Рис. 2: Результат с размером батча 32

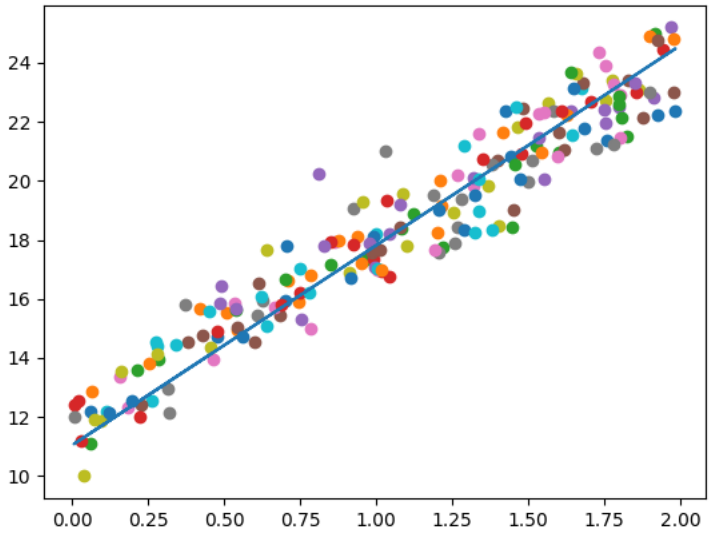

Рис. 3: Результат с размером батча 50

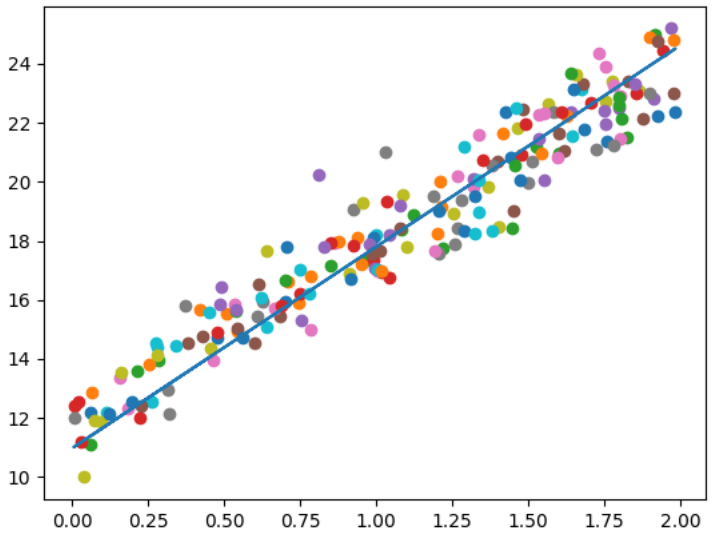

Рис. 4: Результат с размером батча 64

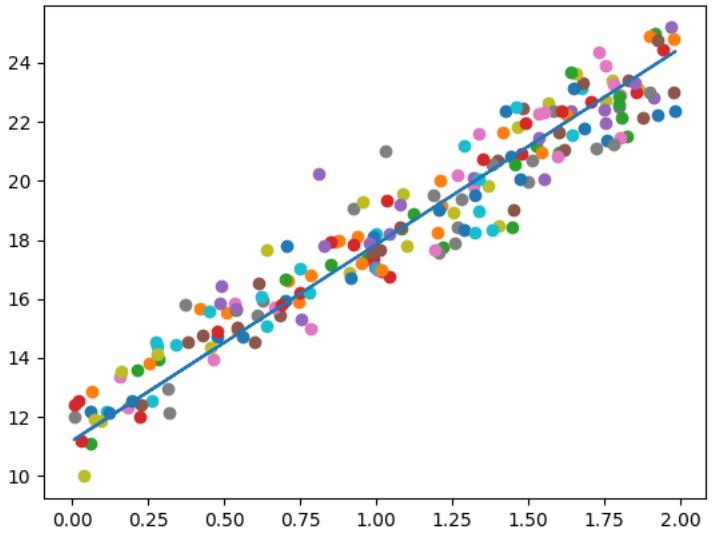

Рис. 5: Результат с размером батча 100

Batch size	w	b	Iterations
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

В исследовании были рассмотрены разные размеры батча: единичный и полный размер коллекции, размер половины коллекции, а также степени двойки, входящие в данный промежуток. 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.

Стохастический градиентный спуск с функцией изменения шага

Функция линейного затухания (linear decay)

Возьмем стохастический градиентный спуск с функцией линейного затухания шага. 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 (it).

Начальные параметры для сравнения обычного стохастического градиентного спуска с постоянным шагом и стохастического градиентного спуска с функцией линейного затухания шага:
$\alpha_{min} = 10^{-3}$;
$\alpha_{start} = 10^{-1}$;
$decay = 10^{-3}$.

Ступенчатая функция (step decay)

Еще одной функцией изменения шага может быть ступенчатая функция. При этом значение шага будет вычисляться по формуле $\alpha = \alpha_{start} \cdot \beta^{\lfloor \frac{it}{decay} \rfloor}$, где α_{start} - начальное значение шага, $decay$ - период затухания.

Начальное параметры для сравнения обычного стохастического градиентного спуска с постоянным шагом и стохастического градиентного спуска со ступенчатой функцией шага:
$\alpha_{start} = 10^{-1}$;
$decay = 100$.

Сравнение функций шага

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

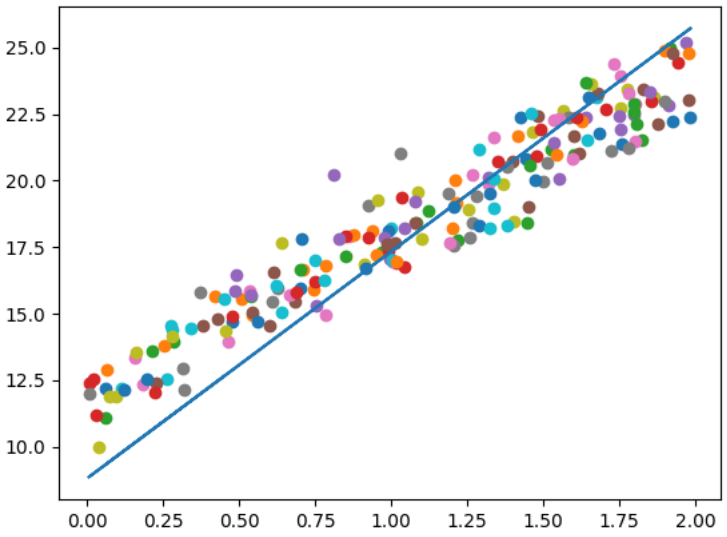

Рис. 6: SGD с фиксированным шагом

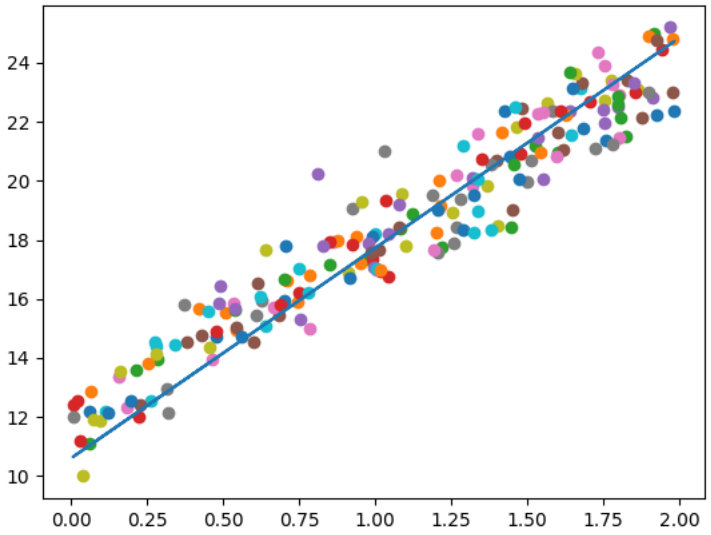

Рис. 7: SGD с функцией линейного затухания

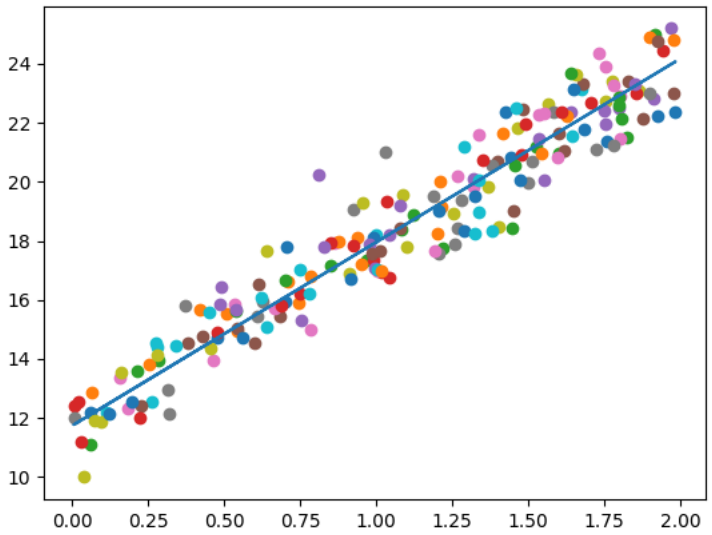

Рис. 8: SGD со ступенчатой функцией

learning rate scheduling	w	b
Fixed	8.53	8.80
Linear decay	7.13	10.59
Step decay	6.22	11.73

Исходя из полученных результатов можно сделать вывод, что лучшую сходимость имеет SGD со ступенчатой функцией. Так же при оптимизированном выборе шага требуется меньше итераций, чем при фиксированном шаге.

Модификации стохастического градиентного спуска (с использованием 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:

```
1    np.random.seed(543)
2    X = -1.43 * 1.45 * np.random.rand(10000, 1)
3    Y = 7 * X + 2 * np.random.rand(10000, 1)
4    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 $epoch = 30$.

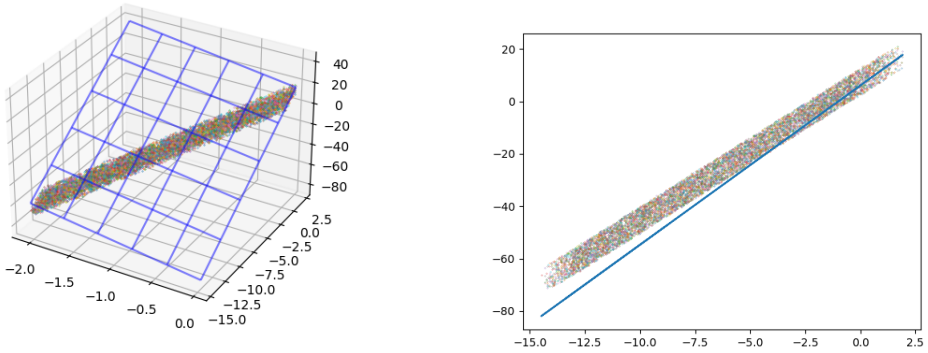

Рис. 9: SGD без модификаций

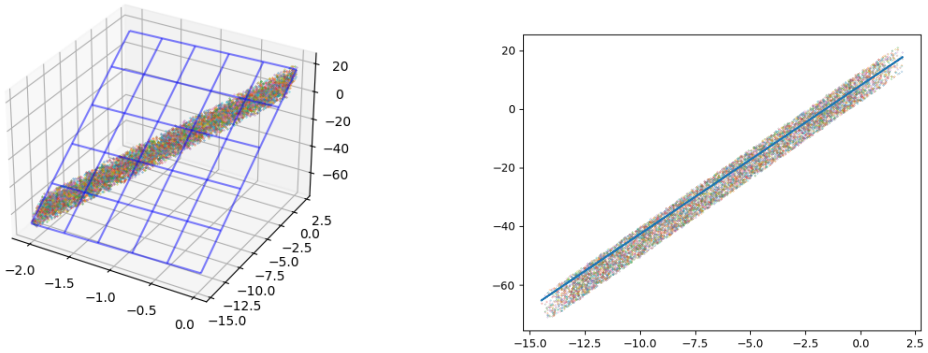

Рис. 10: SGD с модификацией Momentum

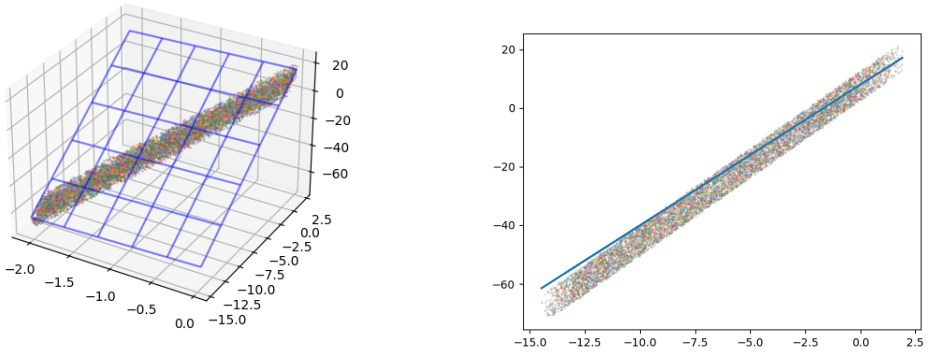

Рис. 11: SGD с модификацией Nesterov

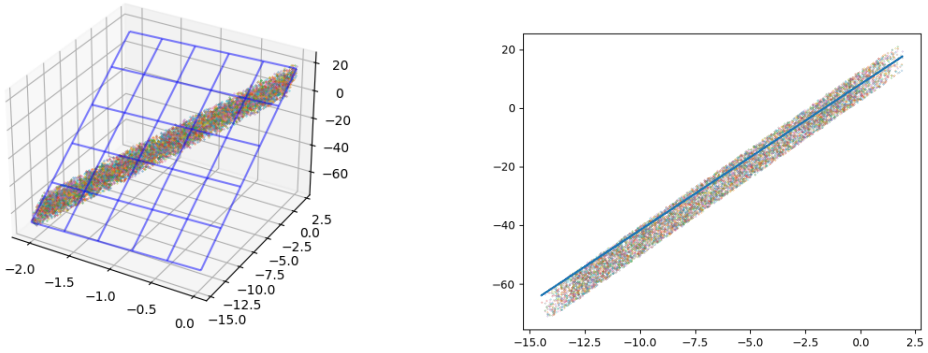

Рис. 12: SGD с модификацией AdaGrad

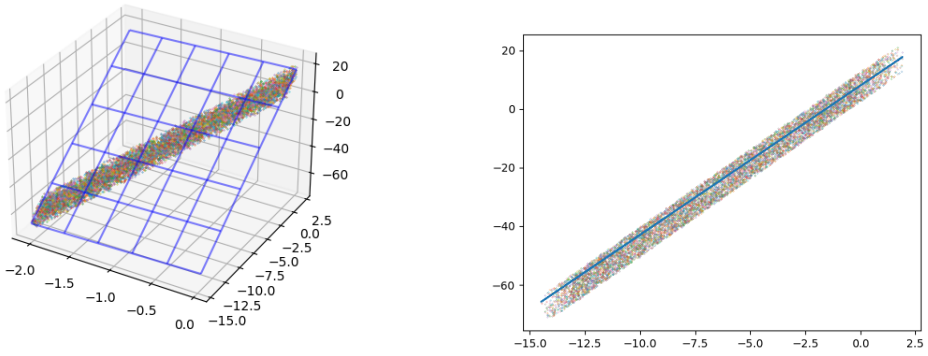

Рис. 13: SGD с модификацией RMSProp

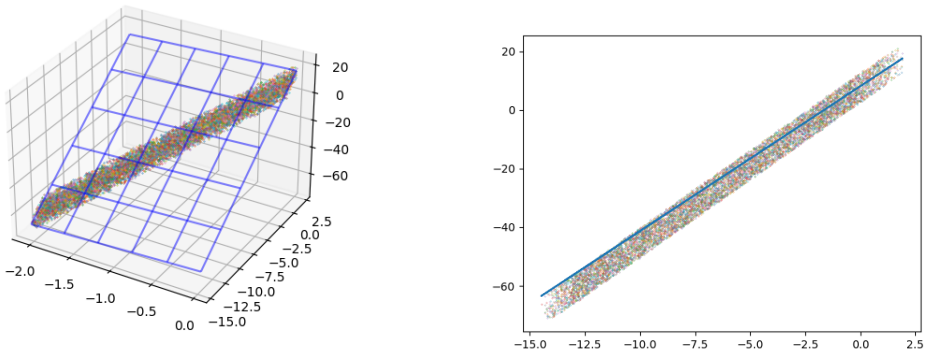

Рис. 14: SGD с модификацией Adam

	memory
SGD	363,67 Mb
Momentum	381,34 Mb
Nesterov	383,42 Mb
AdaGrad	378,17 Mb
RMSProp	384,97 Mb
Adam	390,52 Mb

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.

Полиноминальная регрессия

Полиномиальная регрессия означает приближение данных (x_i, y_i) полиномом k–й степени $A(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3 + \ldots + h \cdot x^k$. Задача полиноминальной регрессии заключается в нахождении полинома f(x) степени не больше k, минизирующего следующую функцию:

$$H(w) = \sum_{i=1}^{m} (f(x_i) - y_i)^2.$$

Методы регуляризации

L1

L1 регуляризация добавляет в функцию потерь дополнительное слагаемое налагающее штраф за сложность модели, то есть высокие веса:

$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i |a_i|$$

L1 обнуляет значения некоторых параметров, что в случае с линейными моделями приводит к отбору признаков.

L2

L2 регуляризация так же добавляет к оптимизационной функции модели штрафную функцию:

$$L_2 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i a_i^2$$

Эта штрафная функция является суммой квадратов весов модели, умноженных на гиперпараметр регуляризации. Это означает, что L2 регуляризация штрафует большие значения весов, заставляя их приближаться к нулю, но в отличие от L1 регуляризации не зануляет их полностью. L2-функция — более эффективна с точки зрения вычислительных функций. L1 позволяет уменьшить значения некоторых коэффициентов до 0, то есть вывести из поля исследования лишние переменные. Это полезно, если на какое-либо явление влияют тысячи факторов и рассматривать все их оказывается бессмысленно.

Elastic

Оба метода регуляризации объединены в технике эластичной сети (Elastic). Она оптимально подходит, когда независимые переменные сильно коррелированы между собой. В этих случаях модель сможет попеременно применять L1 и L2, в зависимости от того, какая лучше подходит с учетом входных данных.

Сравнение методов регуляризации

Для исследования использовались два полинома степеней 8 и 9.

Начальные параметры:

- Первый полином: $1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!}$;
- Второй полином: $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$;

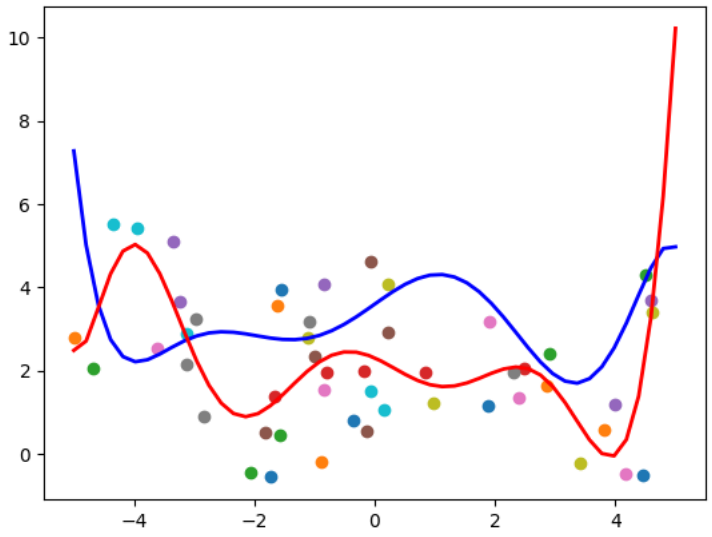

Рис. 15: SGD без метода регуляризации

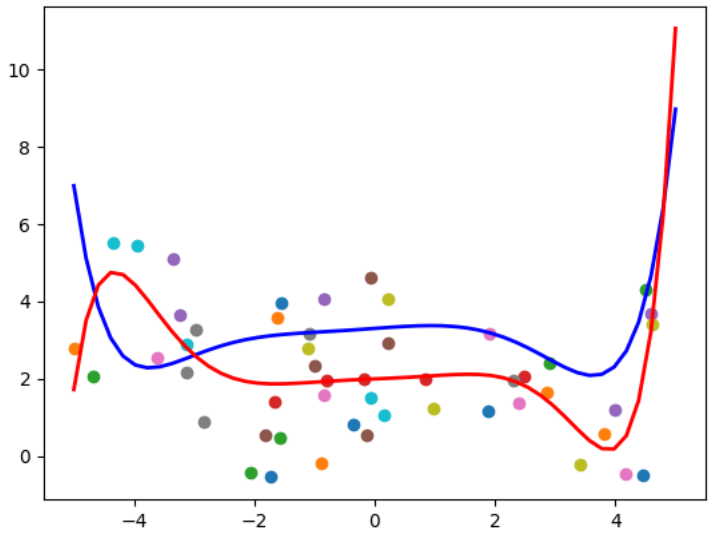

Рис. 16: SGD с методом L1

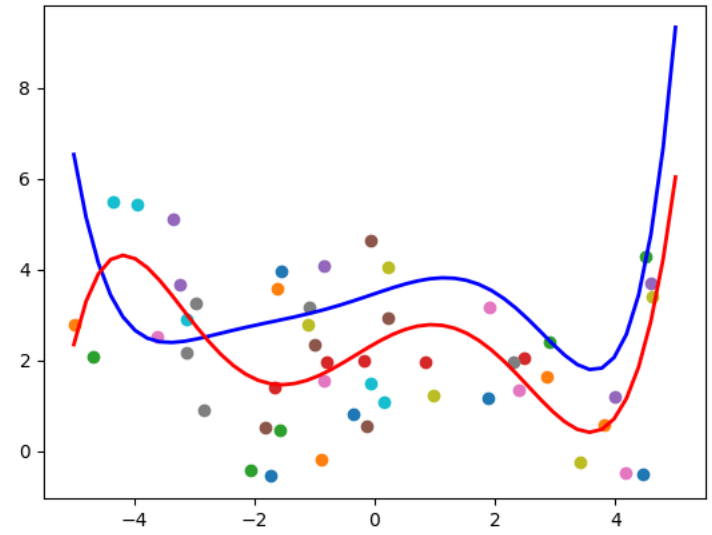

Рис. 17: SGD с методом L2

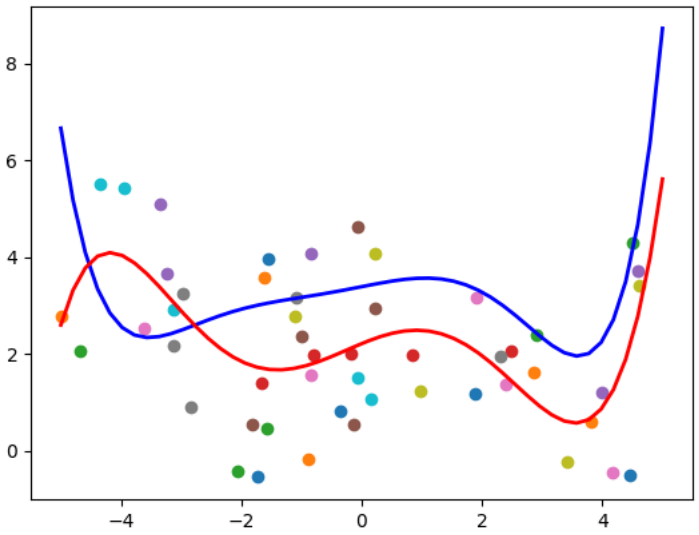

Рис. 18: SGD с методом Elastic

Из полученных графиков можем сделать выводы:

- В работе стохастического градиентного спуска без метода регуляризации была выявлена проблема - переобучение.
- Все методы регуляризации смогли избежать переобучения.
- L1 дает более “плоский“ график, чем L2, однако это может привести к тому, что мы можем упустить какие-то скопления точек.
- Как и было заявлено Elastic оказался чем-то средним между L1 и L2.

Активное обучение

Постановка задачи оптимизации в активном обучении

Активное обучение — это подход в машинном обучении, при котором модель может выбирать наиболее информативные данные для обучения. Это особенно полезно, когда маркировка данных является дорогостоящей или трудоемкой. Основная идея активного обучения заключается в том, чтобы выбрать подмножество данных, которые принесут наибольшую пользу для улучшения модели.

Преимущества активного обучения:

- **Экономия ресурсов**: маркировка данных может быть дорогостоящей и трудоемкой, активное обучение позволяет сократить количество данных, которые необходимо разметить, выбирая только наиболее информативные примеры.
- **Улучшение качества модели**: выбирая наиболее информативные данные, модель может быстрее обучаться и достигать более высокой точности, чем при случайном выборе данных.

- **Адаптивность**: активное обучение позволяет модели адаптироваться к новым данным и изменяющимся условиям, выбирая данные, которые наиболее актуальны для текущей задачи.

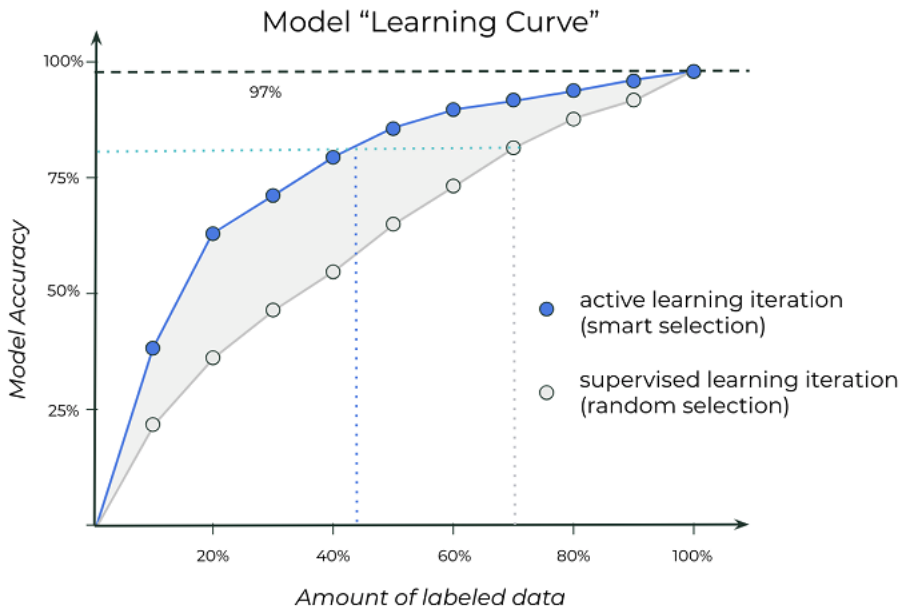

Рис. 19: Преимущество активного обучения

Активное обучение основывается на нескольких ключевых концепциях:

- **Неопределенность**: модель выбирает данные для маркировки, в которых она наиболее неуверенна. Так модель может сосредоточиться на примерах, которые смогут улучшить её способности.
- **Разнообразие**: выбирает данные, которые представляют различные аспекты данных, чтобы избежать переобучения на однотипных примерах и улучшить обобщающую способность модели.
- **Информативность**: выбирает данные, которые, как ожидается, принесут наибольшую пользу для обучения, основываясь на различных критериях, таких как изменение параметров модели или прирост информации.

Процесс активного обучения обычно включает следующие шаги:

1. **Инициализация**: начальная модель обучается на небольшом наборе размеченных данных.
2. **Выбор данных**: модель предсказывает метки для неразмеченных данных и выбирает наиболее информативные примеры для маркировки.
3. **Маркировка данных**: выбранные данные передаются на разметку экспертам.
4. **Обновление модели**: модель переобучается на обновленном наборе данных, включающем новые размеченные примеры.
5. **Повторение**: процесс выбора данных, маркировки и обновления модели повторяется до тех пор, пока не будет достигнута желаемая точность или не исчерпан бюджет на маркировку.

Пример решения задачи активного обучения для спам-фильтра

Задача: создание спам-фильтра для классификации электронных писем на спам и не спам.

Цель: оптимизировать процесс выбора писем для маркировки, чтобы минимизировать количество размеченных данных, необходимых для достижения высокой точности модели.

Формулировка задачи

- Дано: большой набор неразмеченных писем U и небольшой набор размеченных писем L.
- Найти: подмножество писем из U, которые, будучи размеченными, максимально улучшат модель спам-фильтра.

Целевая функция: максимизировать прирост информации или уменьшить неопределенность модели.

Ограничения: ограниченное количество писем, которые можно разметить (например, бюджет на маркировку).

Методы выбора данных:

- Неопределенность.
- Разнообразие.
- Информативность.

Решение задачи

Начальная фаза

Собирается небольшой набор размеченных писем L, который используется для начального обучения модели. Большой набор неразмеченных писем U остается доступным для выбора данных.

Обучение начальной модели

Модель спам-фильтра обучается на начальном размеченном наборе L.

Выбор данных для маркировки

- Модель предсказывает метки для писем из набора U.
- Используя метод неопределенности, выбираются письма, для которых модель наиболее неуверенна (например, предсказания близки к порогу классификации).
- Эти письма передаются на разметку.

Обновление размеченного набора

Размеченные письма добавляются в набор L. Неразмеченные письма удаляются из набора U.

Повторение процесса

Модель переобучается на обновленном наборе L. Процесс выбора данных для маркировки повторяется до тех пор, пока не будет достигнута желаемая точность или не исчерпан бюджет на маркировку.

Оценка модели

После завершения активного обучения модель оценивается на тестовом наборе данных, чтобы проверить её точность и способность классифицировать новые письма.

Пример иллюстрации задачи

Сценарий

У нас есть 10,000 неразмеченных писем и 100 размеченных писем. Мы хотим создать спам-фильтр, который будет классифицировать письма с высокой точностью, но у нас ограниченный бюджет на разметку дополнительных писем.

Шаги

1. **Начальная модель**. Обучаем начальную модель на 100 размеченных письмах. Модель может быть не очень точной из-за малого количества данных.
2. **Активное обучение**.
   - Применяем модель к 10,000 неразмеченным письмам.
   - Выбираем 200 писем, для которых модель наиболее неуверенна (например, вероятность спама около 50%).
   - Отправляем эти письма на разметку.
3. **Обновление модели**. Добавляем 200 размеченных писем к набору размеченных данных. Переобучаем модель на 300 размеченных письмах.
4. **Повторение**. Повторяем процесс выбора и разметки данных несколько раз, пока не достигнем желаемой точности или не исчерпаем бюджет.
5. **Оценка**. Оцениваем финальную модель на тестовом наборе данных, чтобы убедиться в её точности и надежности.

Таким образом, активное обучение позволяет эффективно использовать ограниченные ресурсы для маркировки данных, выбирая наиболее информативные письма для обучения модели спам-фильтра. Это помогает минимизировать количество размеченных данных, необходимых для достижения высокой точности модели, что особенно важно в условиях ограниченного бюджета на разметку.

Пример решения задачи с активным обучением был взят из: https://habr.com/ru/articles/593615/