Содержание

1 Описание метода

1.1 Постановка задачи

В ходе работы необходимо реализовать *Ускоренный по Нестерову метод адаптивной оценки моментов* с целью решение задачи аппроксимации. В качестве аппроксимируемой функции предлагается использовать

$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{1}$$

В качестве функционала качества используется MSE (среднеквадратичная ошибка)

$$Q(\theta) = \frac{1}{l}\sum_{i=1}^{l} \underbrace{[f_\theta(x_i) - y_i]^2}_{L(\theta, x_i, y_i)} = \frac{1}{l}\sum_{i=1}^{l} L(\theta, x_i, y_i) \to \min_{\theta \in \mathbb{R}^n} \tag{2}$$

1.2 SGD

Стохастический градиентный спуск является одним из самых популярных методов оптимизации для решения задач машинного обучения, поскольку прост в реализации и решают проблему, характерную для обычного градиентного спуска - сложность вычислений на каждом шаге.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \nabla Q(\theta) \tag{3}$$

Обычный градиентый спуск (3) считает градиент для всех объектов в наборе данных (*dataset*). Это может стать проблемой, если набор данных не помещается в память или возникает необходимость добавлять новые объекты в *dataset* (онлайн-обучение).

Стохастический градиентный спуск выполняет обновление параметров для каждого объекта x_i, y_i из набора данных.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \nabla Q(\theta; x_i, y_i) \tag{4}$$

Обычный градиентный спуск выполняет избыточные вычисления для больших наборов данных, поскольку он повторно вычисляет градиенты для аналогичных примеров перед каждым обновлением параметра. SGD устраняет эту избыточность, выполняя одно обновление за раз. Поэтому он обычно намного быстрее, и его также можно использовать для онлайн-обучения.

В тоже время как пакетный градиентный спуск сходится к минимуму области, в которой были заданы начальные параметры, колебания SGD, с одной стороны, позволяют ему перейти к новым и потенциально лучшим локальным минимумам, а с другой стороны, это в конечном итоге затрудняет сходимость к точному минимуму. Однако было показано, что при медленном уменьшении темпа обучения α, SGD демонстрирует то же поведение сходимости, что и обычный градиентный спуск, почти наверняка сходясь к локальному или глобальному минимуму для невыпуклых и выпуклых случаев соответственно [1].

1.3 Nadam

Ускоренный по Нестерову метод адаптивной оценки моментов (Nadam) является гибридом метода адаптивной оценки моментов (Adam) и ускоренного градиентного метода Нестерова (NAG) [2].

Классический метод моментов накапливает убывающую сумму (с коэффициентом убывания μ) предыдущих обновлений в вектор момента и заменяет исходный шаг градиента в (5) этим вектором. То есть мы модифицируем алгоритм, чтобы добавлять на каждом шаге следующее [3]:

$$m_t \leftarrow \mu m_{t-1} + \alpha_t g_t \tag{5}$$

$$\theta_t \leftarrow \theta_{t-1} - m_t \tag{6}$$

Интуитивно, это позволяет алгоритму двигаться вдоль участков с малой кривизной, где обновление постоянно мало, но в том же направлении, и медленнее по участкам с резкими изменениями, где направление обновления значительно колеблется [4].

Исследования показывали, что ускоренный градиент Нестерова (NAG) [5] - который имеет доказуемо лучшую сходимость, чем градиентный спуск для выпуклых нестохастических функций - может быть переименован как своего рода улучшенный метод моментов. Если раскрыть m_t в исходной формуле (6), то видно, что обновление эквивалентно шагу в противоположную сторону предыдущего вектора момента и шагу в направлении текущего антиградиента:

$$\theta_t = \theta_{t-1} - (\mu m_{t-1} + \alpha_t g_t) \tag{7}$$

Однако μm_{t-1} не зависит от текущего градиента g_t, поэтому можно точнее получить направление градиента, обновив параметры с помощью момента перед вычислением градиента. В [4] предлагают изменить вычисление градиента, чтобы добиться этого:

$$g_t \leftarrow \nabla_{\theta_{t-1}} f_t(\theta_{t-1} - \mu m_{t-1}) \tag{8}$$

$$m_t \leftarrow \mu m_{t-1} + \alpha_t g_t \tag{9}$$

$$\theta_t \leftarrow \theta_{t-1} - m_t \tag{10}$$

И классический метод моментов, и NAG определяют m как затухающую сумму по сравнению с предыдущими обновлениями; однако оценка метода адаптивного момента (ADAM) [6] вместо этого определяет его как убывающее среднее по предыдущими градиентами и также включает компонент адаптивной скорости обучения α:

$$m_t \leftarrow \mu_{t-1} + (1 - \mu) g_t \tag{11}$$

$$\theta_t \leftarrow \theta_{t-1} - \alpha_t \frac{m_t}{1 - \mu^t} \tag{12}$$

Использование значений предыдущих градиентов вместо предыдущих обновлений позволяет алгоритму продолжать изменять направление, даже если скорость обучения значительно снизилась к концу обучения, что приводит к более

точной сходимости. Это также позволяет алгоритму напрямую скорректировать *initialization bias*, который возникает из-за инициализации вектора импульса нулем (для этого предназначен $(1-\mu^t)$ в знаменателе). В [6] показывают, что этот алгоритм превосходит ряд других, включая NAG, на небольшом количестве тестов.

Часто помогает постепенно увеличивать или уменьшать μ с течением времени, поэтому обозначим данный параметр как вектор $\mu = [\mu_1, ..., \mu_T]$. Перед изменением правила обновления ADAM покажем, как переписать NAG, чтобы его было проще реализовать. Вместо того, чтобы обновлять параметры только с шагом момента, чтобы вычислить градиент, затем отменять этот шаг, чтобы вернуться в исходное состояние параметров, а затем повторять шаг с моментом во время фактического обновления, мы можем использовать mu_{t+1} только один раз, при обновлении предыдущего временного шага вместо t + 1 раз:

$$g_t \leftarrow \nabla_{\theta_{t-1}} f_t(\theta_{t-1}) \tag{13}$$

$$m_t \leftarrow \mu m_{t-1} + \alpha_t g_t \tag{14}$$

$$\theta_t \leftarrow \theta_{t-1} - (\mu_{t+1} m_t + \alpha_t g_t) \tag{15}$$

Заметим, что строка 15 почти идентична строке 7 - с той лишь разницей, что при обновлении используется $\mu_{t+1} m_t$, а не $\mu_t m_{t-1}$. Также легко увидеть, что и шаг по моменту, и шаг градиента здесь зависят от текущего градиента, в отличие от классического метода моментов. Теперь можно использовать тот же трюк с моментом ADAM: сначала переписать шаг обновления Адама через m_{t-1} и g_t, как в строке 16, затем заменить следующий шаг момента на текущий, как в строке 17 (чтобы контролировать *initialization bias*).

$$\theta_t \leftarrow \theta_{t-1} - \alpha_t \Big(\frac{\mu_t m_{t-1}}{1 - \prod_{i=1}^{t} \mu_i} + \frac{(1-\mu_t) g_t}{1 - \prod_{i=1}^{t} \mu_i} \Big) \tag{16}$$

$$\theta_t \leftarrow \theta_{t-1} - \alpha_t \Big(\frac{\mu_{t+1} m_t}{1 - \prod_{i=1}^{t+1} \mu_i} + \frac{(1-\mu_t) g_t}{1 - \prod_{i=1}^{t} \mu_i} \Big) \tag{17}$$

Меняя ADAM таким образом получаем алгоритм Nadam. Однако такой подход к моменту может быть использован в комбинации с другими алгоритмами, которые также использует адаптивный подбор темпа обучения, например *Adamax* или *Equilibrated gradient descent* (EGD)[6, 7].

Алгоритм:

- Шаг 1. Задать параметры: α - величины шага, β_1, β_2 - параметры оценки моментов, $x^0 \in \mathbb{R}^n$ - начальная точка, ϵ - сглаживающий параметр, $\epsilon_1 > 0$; $m^0 = \mathbf{o}$ - начальное значение первого вектора моментов $M[\nabla f(x)]$, $v^0 = \mathbf{o}$ - начальное значение первого вектора моментов $M[\nabla f(x) \odot \nabla f(x)]$, $k = 0$.

- Шаг 2. $k++$ и найти

$$g^k = \nabla f^k(x^{k-1});$$
$$m^k = \beta_1 m^{k-1} + (1-\beta_1) g^k;$$
$$G^k = g^k \odot g^k;$$
$$v^k = \beta_2 v^{k-1} + (1-\beta_2) G^k;$$
$$\hat{m}^k = \frac{\beta_1 m^k}{1-\beta_1^{k+1}} - \frac{(1-\beta_1) g_k}{1-\beta_1^k}$$
$$\hat{v}^k = \frac{\beta_2 v^k}{1-\beta_2^k}$$

- Шаг 3. Вычислить

$$x^k = x^{k-1} - \alpha \hat{m}^k \cdot \sqrt{\hat{v}^k + \epsilon} \tag{18}$$

- Шаг 4. Проверить выполнение условия

$$||x^{k+1} - x^k|| < \epsilon_1. \tag{19}$$

Если условие выполнено, то $x^* = x^k$. Иначе, перейти к шагу 2.

2 Решение задачи

Функция для аппроксимации имеет вид:

$$f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{20}$$

Для решения задачи аппроксимации необходимо найти такие параметры θ, которые бы минимизировали целевой функционал $Q(\theta)$. При использовании метода **NADAM** используюся первые производные. Таким образом для начала необходимо найти значение вектора градиента.

2.1 Аналитическое нахождение градиентов

$$Q(\theta) = \frac{1}{l}\sum_{i=1}^{l}\underbrace{[f_\theta(x_i) - y_i]^2}_{L(\theta,x_i,y_i)} = \frac{1}{l}\sum_{i=1}^{l} L(\theta, x_i, y_i) = \frac{1}{l}\sum_{i=1}^{l}[\theta_0 + \theta_1 x + \theta_2 x^2 - y_i]^2;$$

Предлагаю отбросить пока знак суммы и работать только с L

$$\nabla L(\theta, x_i, y_i) = \big(\frac{\partial L(\theta, x_i, y_i)}{\partial \theta_0}; \frac{\partial L(\theta, x_i, y_i)}{\partial \theta_1}; \frac{\partial L(\theta, x_i, y_i)}{\partial \theta_2}\big)^T$$

$$\frac{\partial L(\theta, x_i, y_i)}{\partial \theta_0} = 2(\theta_0 + \theta_1 x + \theta_2 x^2 - y_i) = 2(f_\theta(x_i) - y_i)$$

$$\frac{\partial L(\theta, x_i, y_i)}{\partial \theta_1} = 2(\theta_0 + \theta_1 x + \theta_2 x^2 - y_i)x_i = 2(f_\theta(x_i) - y_i)x_i$$

$$\frac{\partial L(\theta, x_i, y_i)}{\partial \theta_2} = 2(\theta_0 + \theta_1 x + \theta_2 x^2 - y_i)x_i^2 = 2(f_\theta(x_i) - y_i)x_i^2$$

Подставляя обратно в $Q(\theta)$, получаем:

$$\nabla Q(\theta) = \big(\frac{\partial Q(\theta)}{\partial \theta_0}; \frac{\partial Q(\theta)}{\partial \theta_1}; \frac{\partial Q(\theta)}{\partial \theta_2}\big)^T =$$

$$\big(\frac{2}{l}\sum_{i=1}^{l}(\theta_0 + \theta_1 x + \theta_2 x^2 - y_i);$$

$$\frac{2}{l}\sum_{i=1}^{l}(\theta_0 + \theta_1 x + \theta_2 x^2 - y_i)x_i;$$

$$\frac{2}{l}\sum_{i=1}^{l}(\theta_0 + \theta_1 x + \theta_2 x^2 - y_i)x_i^2\big)$$

2.2 Код

Для решения использовался язык программирования Python (**Ссылка на Github с проектом**). В файле *optimizators.py* находится код для NADAM и SGD.

Ниже представлен код для подбора параметров методом NADAM и SGD

```
        for i in range(1, niters):
            y_pred = func(w, x)
            cur_loss = metric(y, y_pred)
            loss.append(cur_loss)
            grads = get_grads(y, y_pred, x)

            #Nadam params calculation
            mom1 = beta1 * mom1 + (1-beta1) * grads
            mom2 = beta2 * mom2 + (1-beta2) * grads**2
            m1 = (beta1 * mom1) / (1 - np.power(beta1, i+1)) - \
                ((1-beta1)*grads)/(1-np.power(beta1, i))
            m2 = (mom2*beta2) / (1 - np.power(beta2, i))
            prev_w = w.copy()
            w -= (eta/(np.sqrt(m2)+eps))*m1

            if abs(w - prev_w).all() < early_stopping:
                print("Early Stopping")
                print(f"Current difference < {early_stopping}:", \
                    abs(w - prev_w))
                return w, loss
        return w, loss

```

Листинг 1: NADAM optimizer

```
        for i in range(niters):
            #Shuffle Data each epoch.
            if i % data_size == 0:
                np.random.shuffle(indices)
                e += 1
            #Getting loss and prediction
            y_pred = func(w, x)
            cur_loss = metric(y, y_pred)
            loss.append(cur_loss)
            #take "random" index. We can assume random choice because of
    shuffling at each epoch.
            index = indices[i %  data_size]
            #getting prediction for specific element to get gradient and
    weigths update
            y_pred = func(w, x[index])
            grad = get_grads(y[index], y_pred, x[index])
            prev_w = w.copy()
            w -= eta * grad
            lr_history.append(eta)
            if abs(w - prev_w).all() < early_stopping:
                print("Early Stopping")
                print(f"Current difference < {early_stopping}:", abs(w - prev_w
    ))
                return w, loss, lr_history, e
            if i % decay_counter == 0:
                eta *= lr_decay
        y_pred = func(w, x)
        loss.append(metric(y, y_pred))
        return w, loss, lr_history, e

```

Листинг 2: SGD optimizer

В файле expirements.ipynb размещена визуализация поведения функционала качества в зависимости от номера итерации для различного объема выборки и различных методов оптимизации.

3 Результаты

Для проверки работы алгоритма коэффициенты θ создавались случайным образом. На рисунке ниже показана кривая $f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ с параметрами $\theta = [-2, 4, 4]$.

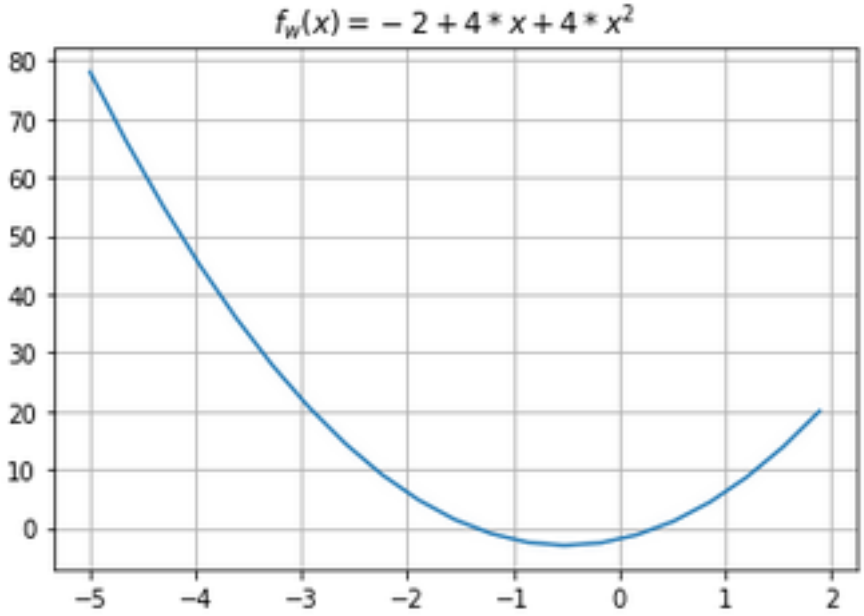

Рис. 1: Случайная кривая

Теперь можно взять 30 случайных точек с этой кривой и использовать их для обучения и проверки работы модели (21 точка для обучения, 9 для проверки). Для этого координаты θ зануляются и решается задача оптимизации - подбор таких параметров, которые минимизировали бы функционал качества $Q(\theta)$.

3.1 Nadam

Параметры для алгоритма: $\alpha = 0.002, \beta_1 = 0.975, \beta_2 = 0.999, \epsilon = 1e - 5$ Критерий остановки: $|\theta^{(i)} - \theta^{(i-1)}| < \epsilon$

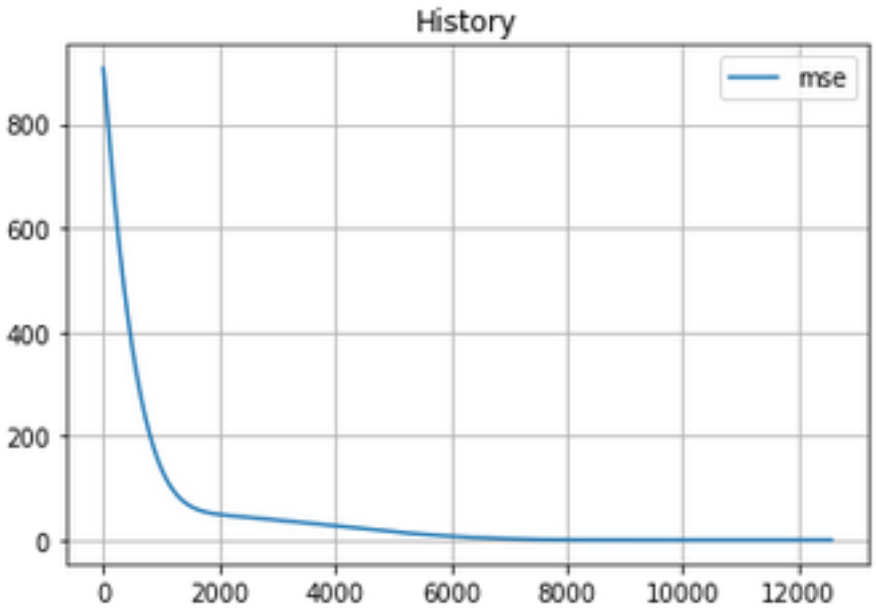

Рис. 2: Изменение MSE в ходе обучения

В ходе обучения алгоритм NADAM находит оптимальные параметры, которые дают глобальный минимум функционалу MSE.

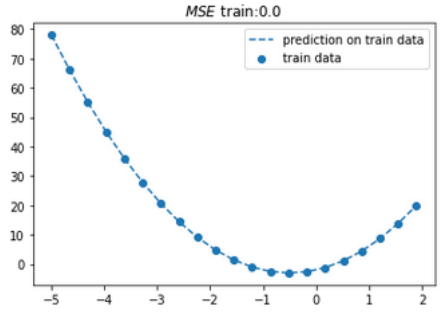

Рис. 3: Обучающая выборка

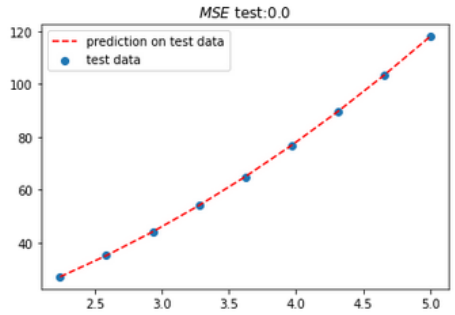

Рис. 4: Тестовая выборка

3.2 SGD

Параметры алгоритма: $\alpha = 0.002, \epsilon = 1e - 5$.

Критерий остановки: $|\theta^{(i)} - \theta^{(i-1)}| < \epsilon$

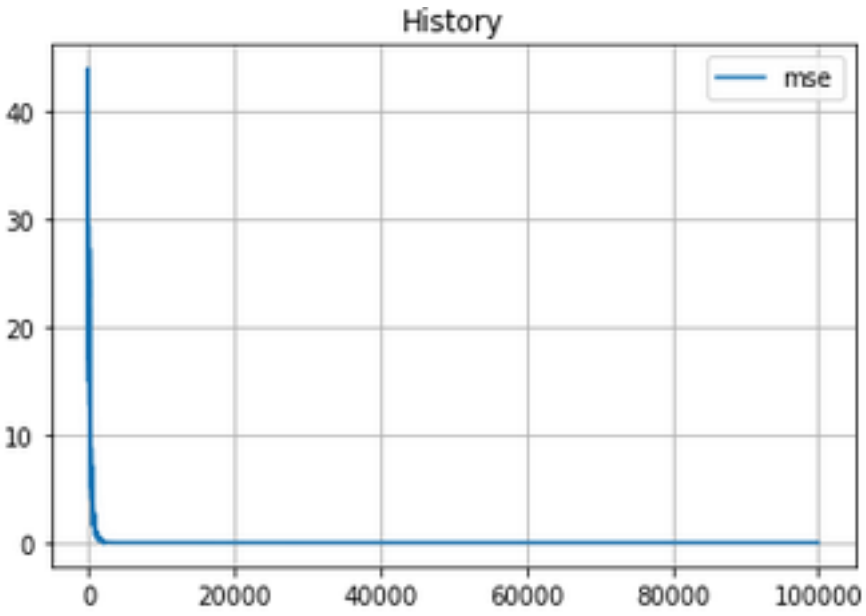

Рис. 5: Изменение MSE в ходе обучения

В ходе обучения алгоритм SGD находит оптимальные параметры, которые дают глобальный минимум функционалу MSE. При этом выход по критерию остановки не происходит, и алгоритм отрабатывает все 100 000 итераций.

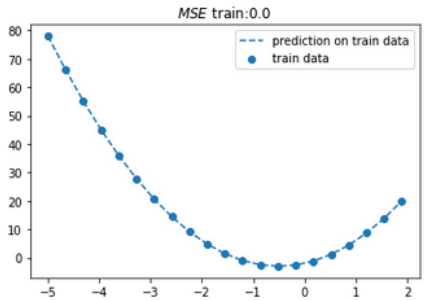

Рис. 6: Обучающая выборка

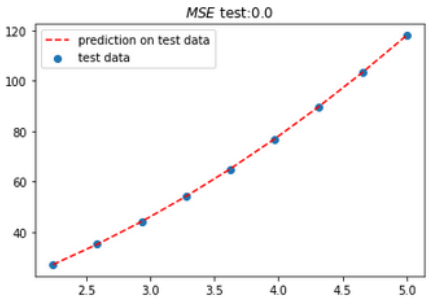

Рис. 7: Тестовая выборка

Больше примеров можно посмотреть по ссылке: **project/experiments.ipynb**

3.3 Влияние параметров на сходимость алгоритма

В методе Nadam в качесте гиперпараметров используются α, β_1, β_2. Данные значения так или иначе влияют на сходимость алгоритма. Для каждой тройки $(\alpha^{(i)}, \beta_1^{(j)}, \beta_2^{(k)})$ создаётся новая модель и сохраняется её история обучения. Чтобы оценить влияние параметров, фиксируется какая-то пара из тройки, а оставшийся элемент меняется. Таким образом можно смотреть на поведение алгоритма, в зависимости от конкретного параметра.

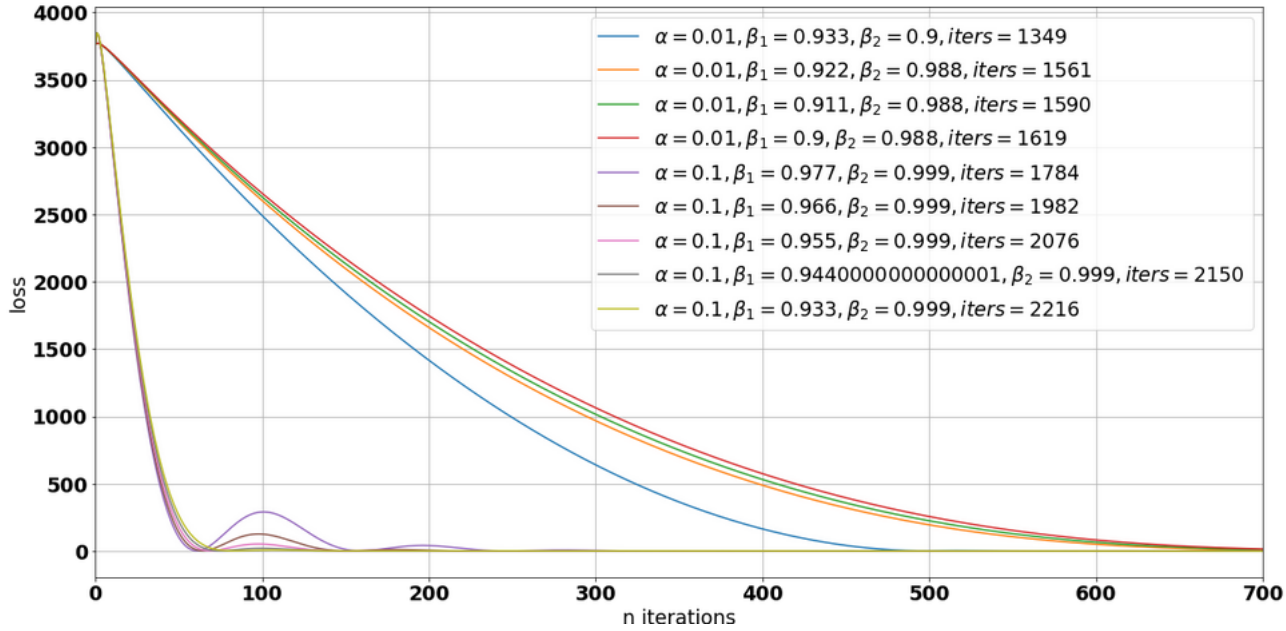

Рис. 8: Изменение MSE в ходе обучения 10 лучших моделей

Зафиксируем параметры β_1, β_2 и будем менять α для изучения поведения графика зависимости MSE от номера итерации.

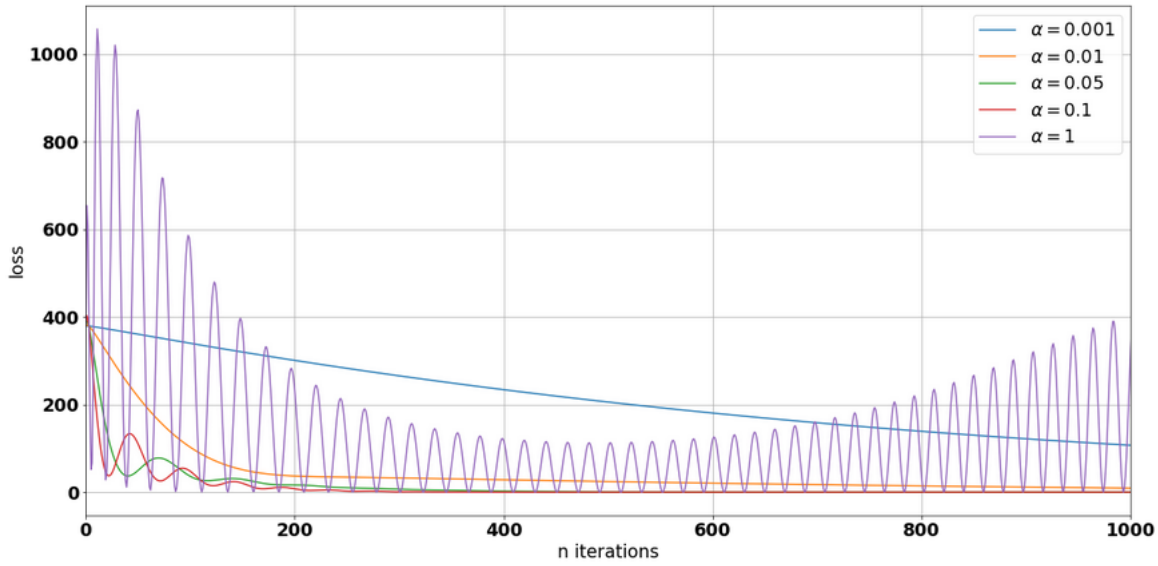

Рис. 9: Изменение MSE для разных параметров α

Как видно из графика, большое значение α не даёт алгоритму сойтись к минимуму. На каждом шаге происходит значительная корректировка весов, сильно меняющая значение целевой функции.

Зафиксируем параметры α, β_2 и будем менять β_1 для изучения поведения графика зависимости MSE от номера итерации.

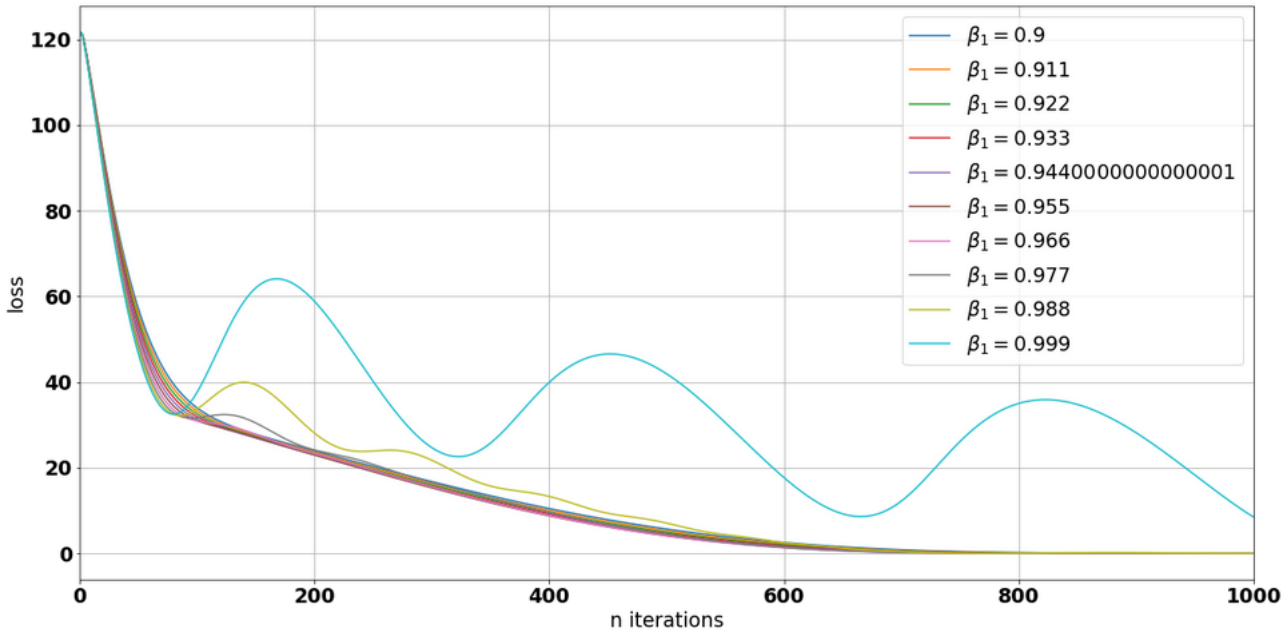

Рис. 10: Изменение MSE для разных параметров β_1

Зафиксируем параметры α, β_1 и будем менять β_2 для изучения поведения графика зависимости MSE от номера итерации.

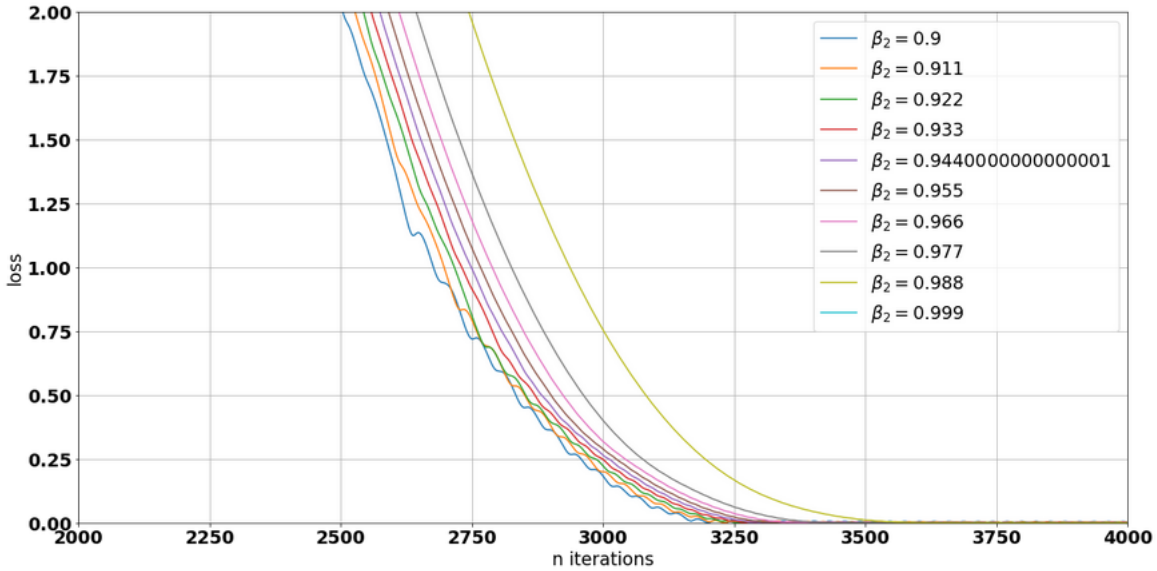

Рис. 11: Изменение MSE для разных параметров β_2

Для значений β_1 близким к 1 график ведёт себя аналогично с параметром $\alpha = 1$. Так, при β_1 алгоритм не сходится к минимуму, при этом каждый раз делает большие по модулю изменения координат искомого вектора θ.

В тоже время, β_2 ведёт себя иначе. Для значений близких к 1, значения целевой функции показывают некоторую нестабильность, повторяя поведения функции при малых β_1 и больших α.

3.4 Вывод

В ходе курсовой работы были исследованы методы для оптимизации Nadam и SGD, а также написана реализация на языке программирования Python. По результатам исследования и экспериментов можно сказать, что Nadam превосходит SGD в скорости сходимости на некоторых наборах данных, при этом требует незначительно большое вычислительных ресурсов для оценки дополнительных параметров.

4 Дополнение

В стохастическом градиентом спуске на каждой итерации рекомендуется выбирать случайный элемент и для него выполнять обновления параметров. В связи с этим предлагается перемешивать элементы один раз в N итераций, где N - размер набора данных, и ввести понятие **эпоха** - один полный проход по набору данных (т.е. алгоритм выполнил изменения параметров для каждого объекта из набора данных 1 раз). Таким образом, для набора данных размера N количество итераций в эпохе будет также равняться N. При этом можно оценить влияние параметра "количество итераций в эпохе"на сходимость.

Для реализации случайного выбора элемента на каждой итерации предлагается сперва вне цикла создать список индексов от 0 до $N - 1$. Перемешать индексы и внутри цикла перебрать все возможные значения индексов. На каждом шаге делать обновления весов. При этом, чтобы понимать, когда закончился перебор всех значений для текущего перемешивания, нужно контролировать номер текущей итерации. Когда остаток от деления номера итерации на N становится 0, то необходимо заново перемешать индексы, чтобы начать новую эпоху.

Реализация процесса перемешивания в SGD:

```
#List of indices from 0 to N-1, where N stands for size of dataset
indices = np.arange(N)
for i in range(max_iters):
    if i % N == 0: #shuffle indices.
        #using np.random.shuffle to shuffle indices in random order
        np.random.shuffle(indices)
    #When indices shuffled simply go through them using loop.
    element_index = i % N
    grad = get_grad(x[element_index], y[element_index], theta)
    update_weights(grad)
```

Листинг 3: Random shuffle

Для анализа влияния количества итераций внутри эпохи создаётся несколько наборов данных размера 10, 30, 50, 100 и 1000. При этом число итерация за одну эпоху будет равно размеру *dataset*

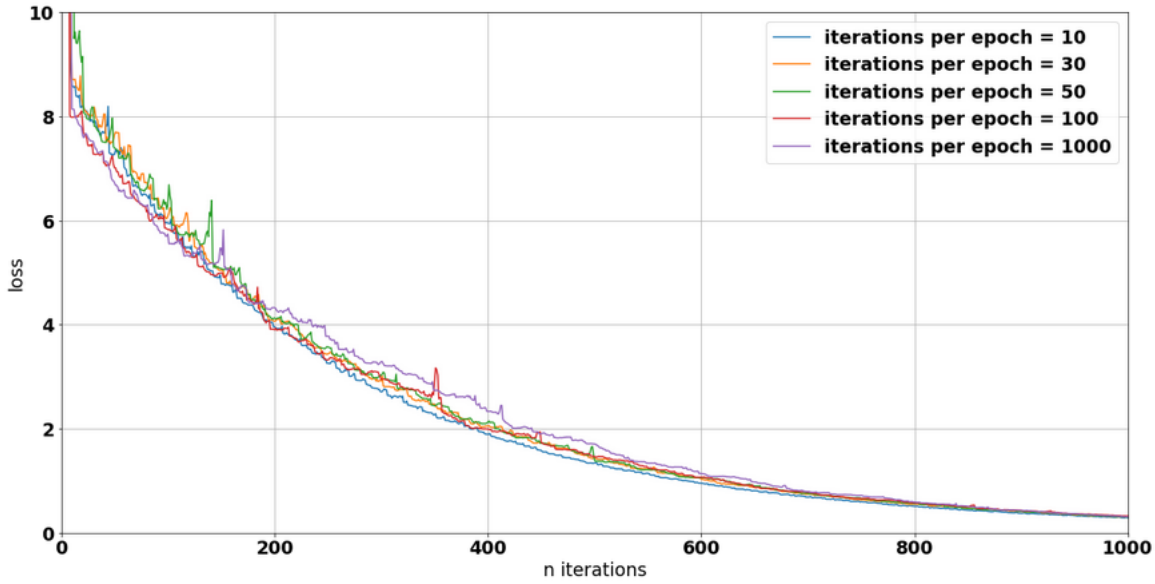

Рис. 12: Изменение MSE для разных параметров *iterations per epoch*

Как видно из графика, количество итераций в эпохе не влияет на сходимость. Заметны незначительные колебания кривых ошибок, которые как раз вызваны случайными перестановками в начале каждой эпохи. При этом, конечно же, количество эпох, которое необходимо для сходимости алгоритма, при прочих равных, обратно пропорционально размеру обучающей выборки. Ведь чем больше размер выборки, тем больше раз за одну эпохи произойдёт обновление параметров.

Важно отметить, что даже при неправильно выбранном параметре α стохастический градиентный спуск может сойтись к минимуму, это достигается за счёт постепенного уменьшения этого параметра с течением времени.

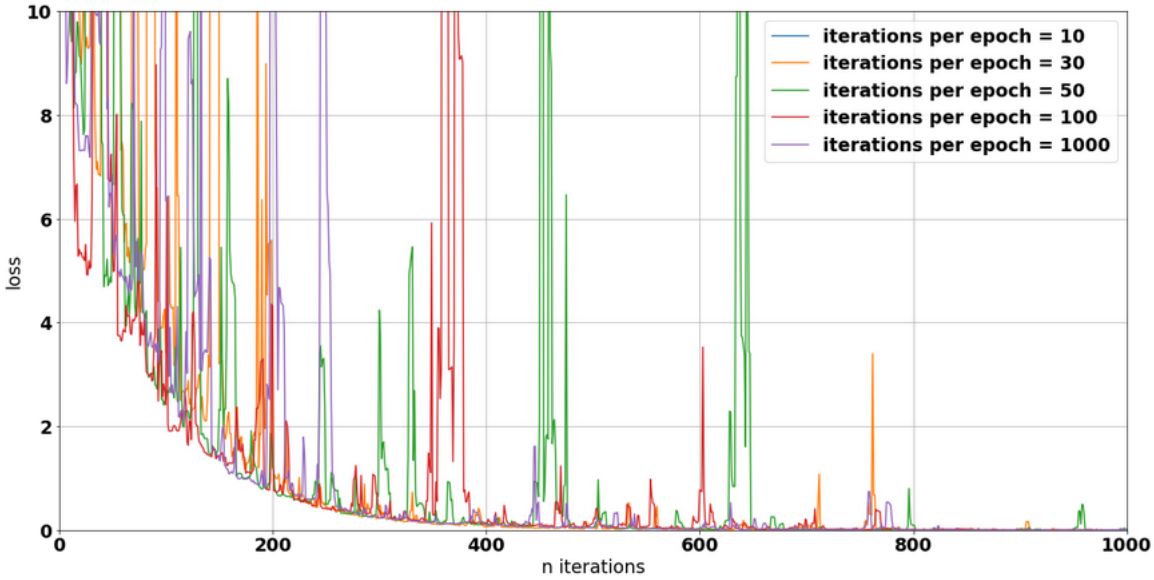

Рис. 13: Изменение MSE для разных параметров *iterations per epoch* при неправильно выбранном параметре $\alpha = 0.007$

При этом метод остаётся очень чувствительным к начальному значению α. На рис. 12 начальное значение $\alpha = 0.02$, а на рис. 13 $\alpha = 0.07$. При выборе $\alpha = 0.1$ метод сразу расходится.

Список литературы

[1] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[2] Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

[3] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[4] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[5] Yu Nesterov. A method of solving a convex programming problem with convergence rate of $(1/k^2)$. In *Sov. Math. Dokl*, volume 27.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Yann Dauphin, Harm De Vries, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in neural information processing systems*, pages 1504–1512, 2015.