Лабораторная работа №3	*2024*
SGD	**Артемьев Иван Вадимович**
	Никитин Михаил Алексеевич
	Павлов Евгений Андреевич

Цель работы: Реализация метода стохастического градиентного спуска и его модификаций для поиска минимумов функций на языке программирования Python и исследование полученных результатов

Инструментарий: Python 3.12 - библиотеки: NumPy, SciPy, MatPlotLib, Tensorflow

Постановка задачи:

Реализуйте и исследуйте на эффективность следующие методы:

1. Стохастический градиентный спуск
    a. с разным размером батча – от одного до размера полной коллекции
    b. с разной функцией изменения шага (learning rate scheduling)
    c. scipy.optimize: SGD, и модификации SGD (Nesterov, Momentum, AdaGrad, RMSProp, Adam)

Содержание исследования:

Сравните эффективность SGD с разными параметрами и его модификации по точности, скорости и ресурсам: объёму требуемой оперативной памяти и количеству арифметических операций. Иллюстрируйте примеры, чтобы не было скучно.

Описание задачи Линейной регрессии:

Модель зависимости одной (объясняемой, зависимой) переменной *y* от другой или нескольких других переменных (факторов, регрессоров, независимых переменных) *x* с линейной функцией зависимости.

Соответственно данная регрессионная модель выглядит следующим образом:

$$y = \overline{w} * \overline{x} + b$$

Здесь вектор *x* - набор фичей(факторов, регрессоров), *y* - линейно от них зависимая переменная, *w* - вектор весов, *b* - также вес. Соответственно задача заключается в том, чтобы отыскать все эти веса.

Если есть всего одна фича, то можно представить каждую зависимость *y* от *x*, как точку на плоскости и наша задача заключается в нахождении прямой, проходящей через все точки(или как можно ближе ко всем)

Что значит как можно ближе ко всем?

Введем понятие среднеквадратичного отклонения нашей прямой: $\text{MSE(w)} = \frac{1}{k}\sum_{i=0}^{k}(Y - \widehat{Y}(w))^2$ - где k - количество наборов фичей, Y - набор значений зависимой переменной *y*, $\widehat{Y}$ - набор “предсказаний”, то есть для конкретных весов, какие значения зависимой переменной *y* получаются.

Тогда можно переформулировать задачу в задачу оптимизации: минимизировать функцию MSE.

Для решения данной задачи используется метод стохастического градиентного спуска.

Рандомно выбирается поднабор из набора наборов фичей. Для него запускается по сути стандартный градиентный спуск с функцией для оптимизации - MSE. Далее выбирается следующий поднабор из набора наборов фичей и для него повторяются действия и так далее пока не закончится набор. Для выбора поднаборов есть разные стратегии: можно рандомно выбирать наборы фичей и добавлять их поднабор и удалять из общего пула и так делать пока он не закончится, либо же можно рандомно перемешать все наборы наборов фичей и брать по очереди. Данные действия - одна эпоха, для наибольшей точности используется несколько эпох - например 100.

Поговорим о гиперпараметрах в данном алгоритме:

1. *batch size* - размер поднабора на котором запускается GD.
2. *learning rate* - гиперпараметр из GD:
    a. Постоянный
    b. Изменяющийся с течением эпох (*learning rate scheduling*)
        i. Ступенчатый - каждые *epoch drop (=10)* эпох *learning rate* умножается на *drop (=0.5),* то есть уменьшается в некоторое количество раз.
        ii. Экспоненциальный - каждую эпоху *learning rate* пересчитывается следующим образом: $lr = lr * e^{-epoch * k}$, где *lr - learning rate, epoch* - номер эпохи, *k(=0.1)* - гиперпараметр.
3. Гиперпараметры из *learning rate scheduling* (выше описано что они означают)
    a. epoch drop
    b. drop
    c. k
4. *epochs* - количество эпох

Библиотечный SGD и его модификации:

Реализации методов взяты из библиотеки tensorflow.keras

1. ***Стандартный SGD***

   Гиперпараметры:

   - *learning rate*
   - *clipnorm* - захват нормы весов (небольше этого параметра)
   - *clipvalue* - захват весов(небольше чем этот параметр)
   - *global clipnorm* - захват нормы градиента всех весов (небольше этого параметра)
   - *gradient accumulation steps* - модель обновляется каждые заданное количество шагов, а не каждый шаг.

2. ***Momentum***

   Гиперпараметры: все те же и еще *momentum.*

   Веса рассчитываются не как $w = w - lr * g$, а теперь добавляется *velocity:* $velocity = momentum * velocity - lr * g$ и теперь веса рассчитываются: $w = w + velocity$.

3. ***Nesterov***

   Гиперпараметры: те же что и в Momentum.

   Веса рассчитываются*:*

   $velocity = momentum * velocity - lr * g$

   $w = w + momentum * velocity - lr * g$.

4. ***AdaGrad***

   Гиперпараметры: те же, что в SGD и еще *initial_accumulator_value.*

Оптимизатор со скоростью обучения для конкретных параметров, которая адаптируется в зависимости от того, как часто параметр обновляется во время обучения. Чем больше обновлений получает параметр, тем меньше обновлений.

5. ***RMSProp***

   Гиперпараметры: те же, что в Momentum и еще *rho*.

   Модифицируем идею AdaGrad: мы всё так же собираемся обновлять меньше веса, которые слишком часто обновляются, но вместо полной суммы обновлений, будем использовать усредненный по истории квадрат градиента.

6. ***Adam***

   Гиперпараметры: те же, что в SGD и еще *beta_1* и *beta_2*

   Как в RMSProp, Adam также использует среднее значение вторых моментов градиентов. В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент, а параметры beta1 и beta2 управляют скоростью затухания этих скользящих средних.

<u>Сравнение эффективности написанных методов:</u>

<u>Полученные результаты:</u>

https://docs.google.com/document/d/1bPPDY0djBh7lBehrA4CL0rTaumHuzOYw2KkH20tK6HI/edit?usp=sharing

<u>“1d, слабая зашумленность”</u>

Запускаем с комбинациями параметров:

epochs = [10, 50, 100, 500]
learning_rates = [0.0005, 0.001, 0.005]

batch_sizes = [1, 5, 20]

1) Self-written SGD STEPPED

Best mse: 271

Best time: 3.94

Лучше всего работает при batch_size = 20, learning_rate = 0.0005

Плохо работает при большом learning_rate

2) Self-written SGD EXPONENTIAL

Best mse: 271

Best time: 1.86

Лучше всего работает при batch_size = 20, learning_rate = 0.001

Плохо работает при большом learning_rate

3) Lib SGD

Best mse: 271

Best time: 569.98

Лучше всего работает при batch_size = 5, learning_rate = 0.005

4) Lib Momentum

Best mse: 271

Best time: 201.98

Лучше всего работает при batch_size = 5, learning_rate = 0.005

5) Lib Nesterov

Best mse: 271

Best time: 173.51

Лучше всего работает при batch_size = 20, learning_rate = 0.005

6) Lib AdaGrad

Модификация не показала достаточно точного результата

7) Lib RMSProp

Best mse: 272

Best time: 13549.29

Показала верный результат только с параметрами batch_size = 1, learning_rate = 0.005

8) Lib Adam

Best mse: 272

Best time: 13549.29

Показала верный результат только с параметрами batch_size = 1, learning_rate = 0.005

Библиотечные методы на малом количестве эпох выдают неверный ответ, при увеличении batch_size начинают работать

быстрее, но уменьшается точность на маленьком количестве эпох.

Самописные методы работают быстрее и точнее

<u>“1d, сильная зашумленность”</u>

Запускаем с комбинациями параметров:

epochs = [10, 50, 100]
learning_rates = [0.0005, 0.001, 0.005]
batch_sizes = [1, 5, 20]

1) Self-written SGD STEPPED

Best mse: 2878

Best time: 3.03

Лучше всего работает при batch_size = 20, learning_rate = 0.001

Плохо работает при большом learning_rate и batch_size

2) Self-written SGD EXPONENTIAL

Best mse: 2878

Best time: 2.45

Лучше всего работает при batch_size = 20, learning_rate = 0.0005

Плохо работает при большом learning_rate и batch_size

3) Lib SGD

Best mse: 2875

Best time: 330.31

Лучше всего работает при batch_size = 5, learning_rate = 0.005

Плохо работает при большом batch_size

4) Lib Momentum

Best mse: 2875

Best time: 201.69

Лучше всего работает при batch_size = 5, learning_rate = 0.005

5) Lib Nesterov

Best mse: 2875

Best time: 167.75

Лучше всего работает при batch_size = 20, learning_rate = 0.005

6) Lib AdaGrad

Модификация не показала достаточно точного результата

7) Lib RMSProp

Модификация не показала достаточно точного результата

8) Lib Adam

Модификация не показала достаточно точного результата

Библиотечные методы работают дольше чем самописные, но некоторые выдают более точный результат.

“3d, слабая зашумленность”

Запускаем с комбинациями параметров:

epochs = [10, 50, 100]
learning_rates = [0.0005, 0.001, 0.005]
batch_sizes = [50, 100, 200]

1) Self-written SGD STEPPED

Best mse: 381

Best time: 53.6

Лучше всего работает при batch_size = 100, learning_rate = 0.005

При маленьком learning_rate выдаёт неверный результат

2) Self-written SGD EXPONENTIAL

Best mse: 382

Best time: 56.52

Лучше всего работает при batch_size = 100, learning_rate = 0.005

При маленьком learning_rate выдаёт неверный результат

3) Lib SGD

Best mse: 378

Best time: 196.58

Лучше всего работает при batch_size = 200, learning_rate = 0.005

4) Lib Momentum

Best mse: 378

Best time: 241.63

Лучше всего работает при batch_size = 200, learning_rate = 0.005

Работает точно с любыми комбинациями параметров

5) Lib Nesterov

Best mse: 378

Best time: 243.96

Лучше всего работает при batch_size = 200, learning_rate = 0.005

Работает точно с любыми комбинациями параметров

6) Lib AdaGrad

Модификация не показала достаточно точного результата

7) Lib RMSProp

Модификация не показала достаточно точного результата

8) Lib Adam

Модификация не показала достаточно точного результата

Библиотечные методы работают дольше чем самописные, но некоторые выдают более точный результат.

“3d, сильная зашумленность”

Запускаем с комбинациями параметров:

epochs = [10, 50, 100]
learning_rates = [0.0005, 0.001, 0.005]
batch_sizes = [50, 100, 200]

1) Self-written SGD STEPPED

Best mse: 396

Best time: 51.09

Лучше всего работает при batch_size = 100, learning_rate = 0.005

При маленьком learning_rate выдаёт неверный результат

2) Self-written SGD EXPONENTIAL

Best mse: 417

Best time: 62.99

Лучше всего работает при batch_size = 200, learning_rate = 0.005

При маленьком learning_rate выдаёт неверный результат

3) Lib SGD

Best mse: 392

Best time: 198.96

Лучше всего работает при batch_size = 200, learning_rate = 0.005

При маленьком learning_rate и количестве эпох выдаёт неверный результат

4) Lib Momentum

Best mse: 392

Best time: 237.51

Лучше всего работает при batch_size = 200, learning_rate = 0.0005

Работает точно с любыми комбинациями параметров

5) Lib Nesterov

Best mse: 392

Best time: 247.69

Лучше всего работает при batch_size = 200, learning_rate = 0.005

Работает точно с любыми комбинациями параметров

6) Lib AdaGrad

Модификация не показала достаточно точного результата

7) Lib RMSProp

Best mse: 404

Best time: 3835.71

Показала верный результат только с параметрами batch_size = 50, learning_rate = 0.005

8) Lib Adam

Best mse: 518

Best time: 3970.91

Показала верный результат только с параметрами batch_size = 50, learning_rate = 0.005

Библиотечные методы работают дольше чем самописные, но некоторые выдают результат немного точнее.

По результатам наших экспериментов получилось, что самописные методы на наших входных данных работали быстрее, но иногда выдавали менее точный результат.

Ссылка на репозиторий с кодом:

https://github.com/Sedromun/lab3-MetOpt