ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Градиентный спуск на языке Python

Аннотация к занятию: на данном занятии обучающиеся познакомятся с алгоритмом градиентного спуска для функции одной переменной на языке Python. Визуализируют работу алгоритма и поговорят про выбор критерия остановки алгоритма.

Цель занятия: сформировать у обучающихся представление об алгоритме градиентного спуска для функции одной переменной на языке Python.

Задачи занятия:

- познакомить обучающихся с алгоритмом градиентного спуска для функции одной переменной на языке Python;
- визуализировать работу алгоритма градиентного спуска;
- рассмотреть критерий остановки алгоритма;
- применить полученные знания на практике.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	2 мин.	Добрый день! Добро пожаловать на урок!	Приветствие. Создание в классе атмосферы психологического комфорта
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	10 мин.	**Вопрос для обсуждения** Давайте повторим, что такое градиентная оптимизация? **Ответы обучающихся** На этом занятии мы реализуем алгоритм градиентного спуска для функции одной переменной на языке Python. Визуализируем работу алгоритма и немного поговорим про выбор критерия остановки алгоритма.	Способствовать обсуждению мотивационных вопросов
Изучение нового материала	50 мин.	Как обычно, начинаем мы с загрузки библиотек. Это знакомые нам NumPy и Matplotlib. Теперь определим нашу функцию f. Возьмём f, равную x в квадрате умножить на синус x: график этой функции мы	Для справки: https://habr.com/ru/post/547424/ Перед уроком рекомендуется ознакомиться с

		видели на предыдущем занятии, когда визуализировали работу градиентного спуска. Вот функция f, которая принимает на вход значение x и выдаёт значение функции f в точке x. Также заведём функцию f_dash, которая будет принимать на вход координату x и выдавать значение производной функции f в точке x. Назовём её f_dash, потому что dash с английского переводится как штрих. По сути, это функция f штрих. Формула производной функции f находится в этой ячейке. Это записано кодом. Прежде чем идти дальше, давайте нарисуем график функции f, чтобы понимать, с чем мы имеем дело. Запишем функцию draw_plot, которая может отрисовать график функции. На вход она принимает функцию, график которой нужно отрисовать. И ещё она принимает на вход два числа: x_begin и x_end. Функция будет рисовать график функции f для значений координаты x, лежащих между x_begin и x_end. Устроена функция очень просто: мы генерируем тысячу значений координаты x между x_begin и x_end, получаем значения функции для этих значений и рисуем график с помощью plt.plot(). Давайте запустим отрисовку для нашей функции f и значений x от одного до восьми.	материалами, представленными на сайте.

		Получим вот такую картинку. Так выглядит график функции f для значений x от 1 до 8. Хорошо. Мы определили функции, которые по значению x могут выдавать значение функции в точке x и значение производной функции в точке x. Пора реализовать алгоритм градиентного спуска и протестировать его на этой функции f. Здесь в ячейке написан алгоритм градиентного спуска. Вы уже видели его на слайде в лекции. Мы говорили о том, что бывают разные критерии остановки алгоритма. В лекции мы рассматривали такой вариант: когда изменение координаты delta x при движении алгоритма становится очень маленьким, мы останавливаем алгоритм. А можно смотреть не на delta x, а на delta y: изменение значения функции на двух последних шагах. Если оно становится маленьким, то останавливать алгоритм. Логика здесь такая же: когда мы подходим близко к точке минимума, функция начинает убывать медленно, значение функции меняется несильно. Давайте в нашем коде используем критерий остановки с delta y. Дома вы сможете самостоятельно поменять критерий остановки на delta x и посмотреть, как от этого изменится поведение алгоритма. Итак, начнём реализацию алгоритма. Сперва напишем функцию, которая будет реализовывать второй и третий шаг алгоритма. То есть по текущей точке x вычислять производную функции в точке x, затем вычислять delta x	

		как альфа умножить на значение производной, и, наконец, получать новую координату точки x как x новое равно x минус дельта x. Назовём функцию gradient_descent_step, то есть шаг градиентного спуска. На вход она будет принимать функцию f_dash, которая вычисляет производные функции f, точку x, из которой нужно сдвинуться, и значение альфа. **Что мы тут делаем:** ● Сначала вычисляем производную функции f в точке x. Затем получаем delta x как альфа умножить на значение производной. И, наконец, вычисляем новую координату x, куда мы сдвинемся как x минус delta x. Возвращаем эту новую координату как результат. ● Проверим, что функция работает. Вызовем её, передадим ей на вход функцию f_dash, точку x, равную 2, и значение альфа равно ноль целых одна десятая. На выходе получим значение около одной целой восьми десятых. Это значение точки x, куда мы сдвинемся, если будем идти градиентным спуском из точки x равно 2 нашей функции f. ● Идём дальше. Что в этой ячейке, мы обсудим чуть позже. Теперь: это, собственно, функция, которая реализует градиентный спуск. Она принимает на вход функцию f, f_dash, значение начальной координаты x, из которой мы будем искать минимум градиентным спуском, а также значения alpha и epsilon.	

		Что функция делает? Сначала получим значение функции в текущей точке x. y равно f(x). Заведём два массива xs и ys. В xs будем складывать значения координат x точек, которые мы будем посещать во время работы алгоритма градиентного спуска. В ys будут значения функции в этих точках. Эти массивы нужны будут для визуализации градиентного спуска на графике: мы будем отрисовывать точки, которые посетили. Положим в xs и ys начальное значение x и f(x). Теперь переходим к самому алгоритму. В бесконечном цикле мы делаем следующее: Сначала вызываем функцию gradient_descent_step. Она берёт на вход f_dash, текущее значение x и альфа. И выдаёт нам новое значение координаты x, которое мы записываем в ту же переменную x. Далее вычисляем значение функции для новой точки x. Теперь поймём, насколько изменилось значение функции за этот шаг. Для этого вытащим значение функции, которое было на предыдущем шаге, для прошлого значения x. Так как мы все значения y записываем в массив ys, то последний элемент этого массива — это как раз значение функции на предыдущем шаге. Тогда изменение значения функции за шаг — это модуль y минус previous_y. Отлично. Теперь запишем новые значения x и y этого шага в массивы xs и ys.	

		Проверим, больше ли y_diff, чем epsilon, или нет. Если y_diff меньше, чем epsilon, останавливаем работу алгоритма: считаем, что мы уже где-то рядом с точкой минимума. Если y_diff больше, чем epsilon, продолжаем: тогда начнётся новая итерация цикла, x снова обновится и так далее. Вот и всё. Вот такой простой код для алгоритма градиентного спуска. Давайте добавим к нему ещё код для вывода информации в процессе работы. После каждой итерации на экран будет выводиться текущее значение x, текущее значение f(x) и разница между значениями f(x) на двух последних шагах. Теперь давайте запустим алгоритм. Зададим значения альфа и epsilon. Пусть они будут пять тысячных и одна тысячная. Зададим начальную координату точки, из которой будем двигаться. Пусть это x равно два с половиной. Запустим алгоритм. Подадим в функцию f, f_dash, x, alpha и epsilon. Видим, что алгоритм отработал быстро и напечатал нам кучу информации о том, по каким точкам он шёл. Это отлично, но в таком виде сложно понять, как двигался алгоритм: как выглядел его путь, пришёл ли он в точку, близкую к точке минимума или нет. Поэтому давайте добавим в наш алгоритм визуализацию процесса. Для этого вернёмся в ячейку выше, которая была закомментирована, и посмотрим на неё. Это функция визуализации процесса градиентного спуска. Мы будем вызывать её на каждом шаге работы алгоритма, и в итоге	

		будем видеть, как двигается точка и как она движется к минимуму шаг за шагом. Эта функция принимает на вход функцию f, x_begin и x_end, а также массивы xs и ys. Что делает функция: во-первых, отрисовывает график функции f, как мы делали это выше. Эти две строки — тот же самый код, который был в функции выше, где мы просто рисовали график функции. Затем отрисовываем точки из массивов xs и ys. xs — это тот массив xs, который формируется в функции градиентного спуска: это массив значений координат x точек, которые градиентный спуск посещает шаг за шагом. ys — массив значений функции f в точках xs. Тут мы просто отрисуем все эти точки с помощью plt.scatter. Дальше в этой строчке мы подписываем на графике последнюю точку из массива xs. Рядом с ней пишем её координату и значение функции в ней. Так как мы будем вызывать эту функцию отрисовки на каждом шаге градиентного спуска, массмвы xs и ys будут пополняться новыми точками, и мы будем видеть, как эти точки движутся по графику. Ну и всё, подписываем график x и вызываем plt.show. Теперь вернёмся в нашу функцию gradient_descent. Здесь вместо вывода текста будем вызывать функцию draw_plot. Подаём в неё переменные. Запускаем ячейку. И снова вызовем функцию gradient_descent с теми же параметрами альфа и epsilon из той же точки x два с половиной.	

		Смотрите: теперь мы видим визуализацию. С каждым шагом алгоритма на графике появляется новая точка, и мы видим, как меняются координаты x и y. У нас получилась точно такая же визуализация, как на слайде. Теперь вы можете самостоятельно поменять значения alpha, epsilon и начальную точку x и посмотреть, как будет в этом случае работать алгоритм градиентного спуска. Вы также можете изменить сами функции f и f_dash и увидеть визуализацию алгоритма оптимизации для любой другой функции одной переменной. В лекции мы упомянули, что критерии остановки алгоритма бывают разные. Мы уже рассмотрели два: сравнивать delta x с epsilon или delta y с epsilon. Сейчас мы поймем, зачем могут быть нужны другие критерии остановки. Давайте поменяем значение альфа на одну десятитысячную и запустим алгоритм ещё раз из точки x равно два с половиной. epsilon оставим тем же. Запускаем. Видим, что функция уже отработала, но точка при этом никуда не сдвинулась. На самом деле, это неудивительно. Смотрите, мы сильно уменьшили значение альфа. Поэтому шаг delta x стал очень маленьким. Из-за этого за первый же шаг алгоритма значение функции изменилось на величину, меньшую epsilon. Алгоритм завершился, посчитав, что мы, вероятно, уже находимся у точки минимума. Значение функции f(x) же поменялось очень мало.	

		Однако мы видим, что это не так: мы не дошли до точки минимума. Просто точка два с половиной находится на таком месте функции, где скорость убывания функции очень маленькая, из-за этого значение f(x) при первом же шаге алгоритма с малым alpha поменялось очень мало, и алгоритм остановился. Получается, в этом случае наш алгоритм работает неверно: неверно определяет, когда мы можем остановиться. Давайте это исправим: сделаем так, чтобы алгоритм работал лучше в этом случае. Зададим какое-то количество шагов, которые алгоритм обязательно должен пройти до завершения. Например, 100. Теперь, даже если в какой-то момент delta y будет меньше epsilon, но алгоритм ещё не сделал 100 шагов, остановиться он не сможет. Закодим это. Добавим переменную step_count, которая изначально равна нулю. Далее каждый шаг будет увеличивать её на 1. В код условия остановки добавим, что останавливаться алгоритм может только если delta_y меньше epsilon и step >= 100. Запустим ячейку. Теперь, если мы снова запустим алгоритм с alpha равно одной десятитысячной, он не остановится в самом начале и всё же дойдёт до точки минимума. Разумеется, такое решение проблемы не идеально. Всё равно может возникнуть ситуация, когда сто шагов	

		прошли, а алгоритм в этот момент застрял в месте функции, где производная мала по модулю, но это не локальный минимум. Но идеально решить проблему нельзя. Мы можем только попытаться запустить алгоритм несколько раз из разных начальных точек с разными alpha и epsilon и разными критериями остановки и надеяться, что в какой-то из этих раз алгоритм придёт достаточно близко к минимуму. Мы просто возьмём минимальное значение из всех тех, что выдадут нам разные запуски алгоритма с разными значениями. Но мы всё равно не сможем гарантировать, что нашли точку минимума. Может возникнуть вопрос: как выбирать значения alpha и epsilon. Мы только что увидели, что при уменьшении alpha до одной десятитысячной у алгоритма начались проблемы, которые мы попытались решить с помощью добавления ещё одного критерия остановки. А ещё мы понимаем, что если выбрать значение альфа слишком большим, алгоритм будет перескакивать через минимум и может остановиться достаточно далеко от точки минимума. Можете сами проверить, как будет работать алгоритм, задав большое значение альфа. То же самое и с epsilon. Если взять значение epsilon слишком большим, алгоритм остановится далеко от минимума. Если взять слишком маленьким, то алгоритм может никогда не дойти настолько близко к минимуму и никогда не завершиться. Конечно, можно в этом случае ограничить число шагов алгоритма сверху. Обычно так и	

		делают, но это тоже может привести к проблемам. Можете подумать, к каким. Выбор значений alpha и epsilon — это сложный вопрос. Ответа на него опять же нет. Всё, что можно пытаться делать, как мы сказали выше, — перебрать несколько разных значений alpha и epsilon, запустить алгоритм несколько раз и взять как точку минимума минимальную из всех точек, к которым пришли все эти варианты алгоритмов. Опять же, вы можете самостоятельно менять в коде значения alpha, epsilon, x, поменять критерии остановки и посмотреть, как будет вести себя алгоритм градиентного спуска при каждом значении параметров. Разумеется, можете изменить саму функцию x и запускать градиентный спуск для других функций. Не забудьте только при этом также поменять f'.	
Закрепление изученного материала	15 мин.	**Вопросы для обсуждения** ● В чём основная цель алгоритма? ● Как отобразить результат графике?	Педагог организует беседу по вопросам
Этап подведения итогов занятия (рефлексия)	8 мин.	**Вопросы для обсуждения** ● Чему я научился? ● С какими трудностями я столкнулся? ● Какие вопросы остались? Что осталось непонятным?	Педагог способствует размышлению обучающихся над вопросами

Информация о домашнем задании, инструктаж по его применению	5 мин.	В домашнем задании вам предстоит реализовать алгоритм градиентного спуска для функции двух переменных. Не пугайтесь: это практически ни капли не сложнее того, что мы делали сейчас на практике. А картинки графиков функций двух переменных получатся очень красивыми и интересными.	

Рекомендуемые ресурсы для дополнительного изучения:

1. Градиент. [Электронный ресурс] – Режим доступа: https://ru.abcdef.wiki/wiki/Gradient
2. Обзор градиентных методов в задачах математической оптимизации. [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/413853/
3. Градиентный спуск в Python. [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/547424/