Семинар: Градиентный спуск.

В [6]:
```
from typing import Iterable, List

import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

Часть 1. Градиентный спуск

Функционал ошибки, который мы применяем в задаче регр
Error:

$$Q(w, X, y) = \frac{1}{\ell}\sum_{i=1}^{\ell}(\langle x_i, w\rangle - y_i)^2$$

где x_i — это i-ый объект датасета, y_i — правильный ответ д
нашей линейной модели.

Можно показать, что для линейной модели, функционал ош
матричном виде следующим образом:

$$Q(w, X, y) = \frac{1}{l}(y - Xw)^T(y - Xw)$$

или

$$Q(w, X, y) = \frac{1}{l}||Xw - y||^2$$

где X — это матрица объекты-признаки, а y — вектор прави

Для того чтобы воспользоваться методом градиентного спу
градиент нашего функционала. Для MSE он будет выгляде

$$\nabla_w Q(w, X, y) = \frac{2}{l}X^T(Xw - y)$$

Ниже приведён базовый класс `BaseLoss`, который мы буд
реализации всех наших функционалов ошибки (= функций
его не нужно. У него есть два абстрактных метода:

1. Метод `calc_loss`, который будет принимать на вход с
   ответы `y` и веса `w` и вычислять значения функционал
2. Метод `calc_grad`, который будет принимать на вход с
   ответы `y` и веса `w` и вычислять значения градиента (

В [7]:
```
import abc


class BaseLoss(abc.ABC):


    @abc.abstractmethod
    def calc_loss(self, X: np.ndarray, y: np.n

        raise NotImplementedError

    @abc.abstractmethod
    def calc_grad(self, X: np.ndarray, y: np.n
        raise NotImplementedError
```

Реализация этого абстрактного класса: Mean Squared Error

Данный код определяет BaseLoss класс с использованием модель линейной регрессии, обучает ее на масштабирова датасета, предсказывает значения y_train и y_test с помощ выводит значение средней квадратичной ошибки (MSE) и к детерминации (R2) для train и test датасетов, чтобы провер Метод calc_loss() вычисляет значение потерь, а calc_grad() функции потерь

Задание 1.1: Реализуйте класс `MSELoss`

Он должен вычислять значение функционала ошибки (лос $\nabla_w Q(w, X, y)$ по формулам (выше)

В [8]:
```
class MSELoss(BaseLoss):
    def calc_loss(self, X: np.ndarray, y: np.n
        return (np.linalg.norm(np.matmul(X,w)

    def calc_grad(self, X: np.ndarray, y: np.n
        return 2/len(y)*np.matmul(np.transpose
```

Данный определяет класс с именем MSELoss, который реа MSE. Он состоит из двух методов: calc_loss и calc_grad.

Метод calc_loss вычисляет потери MSE с учетом входных д y и весов модели w. Он использует формулу для вычислен между прогнозируемыми и фактическими значениями: резу количество точек данных, чтобы получить среднюю квадра

Метод calc_grad вычисляет градиент потерь MSE по отнош принимает те же входные параметры, что и calc_loss, и исп вычисления градиента. Она представляет направление и в подъема или спуска в функции потерь

Теперь мы можем создать объект `MSELoss` и при помощи нашего функционала ошибки и градиент:

В [9]:

```
loss = MSELoss()

X = np.arange(200).reshape(20, 10)
y = np.arange(20)

w = np.arange(10)

print('X:\n', X)
print('y:\n', y)
print('w:\n', w, '\n')


print(loss.calc_loss(X, y, w))
print(loss.calc_grad(X, y, w))


assert loss.calc_loss(X, y, w) == 27410283.5,
assert np.allclose(
    loss.calc_grad(X, y, w),
    np.array(
        [
            1163180.0,
            1172281.0,
            1181382.0,
            1190483.0,
            1199584.0,
            1208685.0,
            1217786.0,
            1226887.0,
            1235988.0,
            1245089.0,
        ]
    ),
), "Метод calc_grad реализован неверно"
print("Всё верно!")
```

```
X:
 [[  0   1   2   3   4   5   6   7   8   9]
 [ 10  11  12  13  14  15  16  17  18  19]
 [ 20  21  22  23  24  25  26  27  28  29]
 [ 30  31  32  33  34  35  36  37  38  39]
 [ 40  41  42  43  44  45  46  47  48  49]
 [ 50  51  52  53  54  55  56  57  58  59]
 [ 60  61  62  63  64  65  66  67  68  69]
 [ 70  71  72  73  74  75  76  77  78  79]
 [ 80  81  82  83  84  85  86  87  88  89]
 [ 90  91  92  93  94  95  96  97  98  99]
 [100 101 102 103 104 105 106 107 108 109]
 [110 111 112 113 114 115 116 117 118 119]
 [120 121 122 123 124 125 126 127 128 129]
 [130 131 132 133 134 135 136 137 138 139]
 [140 141 142 143 144 145 146 147 148 149]
 [150 151 152 153 154 155 156 157 158 159]
 [160 161 162 163 164 165 166 167 168 169]
 [170 171 172 173 174 175 176 177 178 179]
 [180 181 182 183 184 185 186 187 188 189]
 [190 191 192 193 194 195 196 197 198 199]]
y:
 [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
w:
 [0 1 2 3 4 5 6 7 8 9]

27410283.5
[1163180. 1172281. 1181382. 1190483. 1199584. 1208
 1235988. 1245089.]
Всё верно!
```

Сначала я импортирую необходимые библиотеки для числ
определяю класс с именем MSELoss, который содержит д
calc_grad, описанные выше.

Метод calc_loss принимает на вход три параметра: X, y и w

- X - входные данные;
- y - фактические выходные данные;
- w - веса.

Внутри метода я вычисляю прогнозируемый результат (y_h
входных данных (X) на веса (w). Затем вычисляю потери M
между фактическим результатом (y) и прогнозируемым рез
его по количеству выборок (n).

Метод calc_grad также принимает те же параметры, что и с
градиент потерь MSE по отношению к весам (w). Градиент
использованием производной формулы потерь MSE. Отри
гарантирует, что градиент указывает в направлении умень

Наконец, мы создаем экземпляр MSELoss класса и демонс
передавая образцы данных (X, y и w) методам calc_loss an
выводятся на консоль

Для вычисления градиента все готово, реализуем градиент
формула для одной итерации градиентного спуска выгляд

$$w^t = w^{t-1} - \eta \nabla_w Q(w^{t-1}, X, y)$$

Где w^t — значение вектора весов на t-ой итерации, а η — п
отвечающий за размер шага.

Задание 1.2: Реализуйте функцию `gradient_descent`

Функция должна принимать на вход начальное значение ве
`w_init` , матрицу объектов-признаков `X` , вектор правилы
функции потерь `loss` , размер шага `lr` и количество итер

Функция должна реализовывать цикл, в котором осуществ
спуска (градиенты берутся из `loss` посредством вызова м
формуле выше и возвращать траекторию спуска (список из
каждом шаге).

В [10]:

```
def gradient_descent(
    w_init: np.ndarray,
    X: np.ndarray,
    y: np.ndarray,
    loss: BaseLoss,
    lr: float,
    n_iterations: int = 100000,
) -> List[np.ndarray]:
    grad_list = []
    for i in range(n_iterations):
        grad_list.append(w_init)
        w_init = w_init - lr * loss.calc_grad(
    return grad_list

```

Функция gradient_descent принимает следующие параметр

- w_init - начальные значения параметров.
- X - входные данные.
- y - целевые значения.
- loss - экземпляр класса функции потерь, который вычи
- lr - скорость обучения.
- n_iterations - количество итераций (по умолчанию уста

Функция инициализирует пустой список grad_list для сохра
на каждой итерации. Затем она выполняет итерации n_iter
обновляя значения параметров с помощью правила обнов

w_init = w_init - lr * loss.calc_grad(X, y, w_init) На каждой ите
параметров w_init добавляются к grad_list. Наконец, функц
значений параметров.

Мы определяем простую задачу линейной регрессии с вхо
целевыми значениями y. Мы инициализируем значения па
создаем экземпляр [0, 0] класса, который вычисляет потер
устанавливаем скорость обучения на MeanSquaredError и
lr.0.01n_iterations1000

Затем мы вызываем gradient_descent функцию, передавая
параметров, входные данные, целевые значения, функцию
и количество итераций. Функция возвращает список, содер

Теперь создадим синтетический датасет и функцию, котора
траекторию градиентного спуска по истории.
(Если не оговорено иное, то в задачах используются указа
`n_features`,
`n_objects`, `batch_size`, `num_steps`).

В [11]:
```
np.random.seed(1337)

n_features = 2
n_objects = 300
batch_size = 10
num_steps = 43

w_true = np.random.normal(size=(n_features,))

X = np.random.uniform(-5, 5, (n_objects, n_fea
X *= (np.arange(n_features) * 2 + 1)[np.newaxi
y = X.dot(w_true) + np.random.normal(0, 1, (n_
w_init = np.random.uniform(-2, 2, (n_features)

print(X.shape)
print(y.shape)
```

```
(300, 2)
(300,)
```

Предоставленный код реализует алгоритм градиентного сп
регрессии. Давайте разберем структуру кода:

Функция gradient_descent принимает следующие параметр

- w_init: Начальные веса для модели линейной регресси
- X: Функции ввода (независимые переменные).
- y: Целевая переменная (зависимая переменная).
- loss: Объект функции потерь, который вычисляет град
- lr: Скорость обучения, которая определяет размер шаг
- n_iterations: Количество итераций для алгоритма (по ум
  значение 100 000). Программа инициализирует необхо
  массивы для алгоритма. Он генерирует случайные дан
  устанавливает начальные веса w_init и печатает форм

Цикл градиентного спуска: основная часть кода - это for ци
n_iterations несколько раз. На каждой итерации он вычисля
потерь относительно весов и обновляет веса, используя ск

Возвращаемое значение - список массивов весов на каждо
позволяет нам анализировать сходимость алгоритма.

В [12]:
```
loss = MSELoss()
w_list = gradient_descent(w_init, X, y, loss,
print(loss.calc_loss(X, y, w_list[0]))
print(loss.calc_loss(X, y, w_list[-1]))
```

```
425.5891768045026
0.867074968324295
```

Инициализация потерь: мы создаем экземпляр функции пс
среднеквадратичной ошибке (MSE), используя MSELoss().
использоваться для оценки производительности нашей мо,

Градиентный спуск: мы вызываем gradient_descent функци
веса (w_init), входные данные (X), целевые значения (y), ф
скорость обучения (0.01) и количество шагов (num_steps).
алгоритм градиентного спуска и возвращает список значен

Расчет потерь: Мы вычисляем потери, используя calc_loss
передаем входные данные (X), целевые значения (y) и пер
веса из w_list списка. Это позволяет нам сравнить начальн
потерь.

В [13]:
```
w_init
```

Out[13]:
```
array([0.62074297, 1.79288146])
```

В [14]:
```
w_list[-1]
```

Out[14]:
```
array([-0.66688623, -0.48984218])
```

В [15]:

```
def plot_gd(w_list: Iterable, X: np.ndarray, y

    w_list = np.array(w_list)
    meshgrid_space = np.linspace(-2, 2, 100)
    A, B = np.meshgrid(meshgrid_space, meshgri

    levels = np.empty_like(A)
    for i in range(A.shape[0]):
        for j in range(A.shape[1]):
            w_tmp = np.array([A[i, j], B[i, j]
            levels[i, j] = loss.calc_loss(X, y

    plt.figure(figsize=(15, 6))
    plt.title("GD trajectory")
    plt.xlabel(r"$w_1$")
    plt.ylabel(r"$w_2$")
    plt.xlim(w_list[:, 0].min() - 0.1, w_list[
    plt.ylim(w_list[:, 1].min() - 0.1, w_list[
    plt.gca().set_aspect("equal")

    CS = plt.contour(
        A, B, levels, levels=np.logspace(0, 1,
    )
    CB = plt.colorbar(CS, shrink=0.8, extend="


    plt.scatter(w_list[:, 0], w_list[:, 1])
    plt.plot(w_list[:, 0], w_list[:, 1])

    plt.show()
```

Код состоит из функции с именем plot_gd, которая принима

- w_list - параметр, представляющий список весовых век вектор соответствует определенной итерации алгоритм
- x и y - параметры, представляющие входные функции соответственно.
- loss - параметр, представляющий функцию потерь, исп потерь для каждого вектора веса.

Затем код переходит к созданию пространства сетки с пом Это пространство сетки используется для создания сетки т

Далее используется вложенный цикл для вычисления поте сеточном пространстве. Потери вычисляются с использова объекта.

После вычисления потерь для каждого балла код создает и устанавливает заголовок, метку по оси x и метку по оси y и plt.ylabel соответственно.

Ограничения по осям x и y устанавливаются на основе мин максимального значений весовых векторов в w_list. Соотно устанавливается равным с помощью plt.gca().set_aspect("e

Контурный график создается с помощью plt.contour визуал
пространстве meshgrid. levels Параметр определяет уровн
контуры. cmapПараметр определяет цветовую карту, испол

К графику добавляется цветная полоса, использующая plt.
диапазона значений потерь.

Весовые векторы в w_list отображаются в виде точек расс

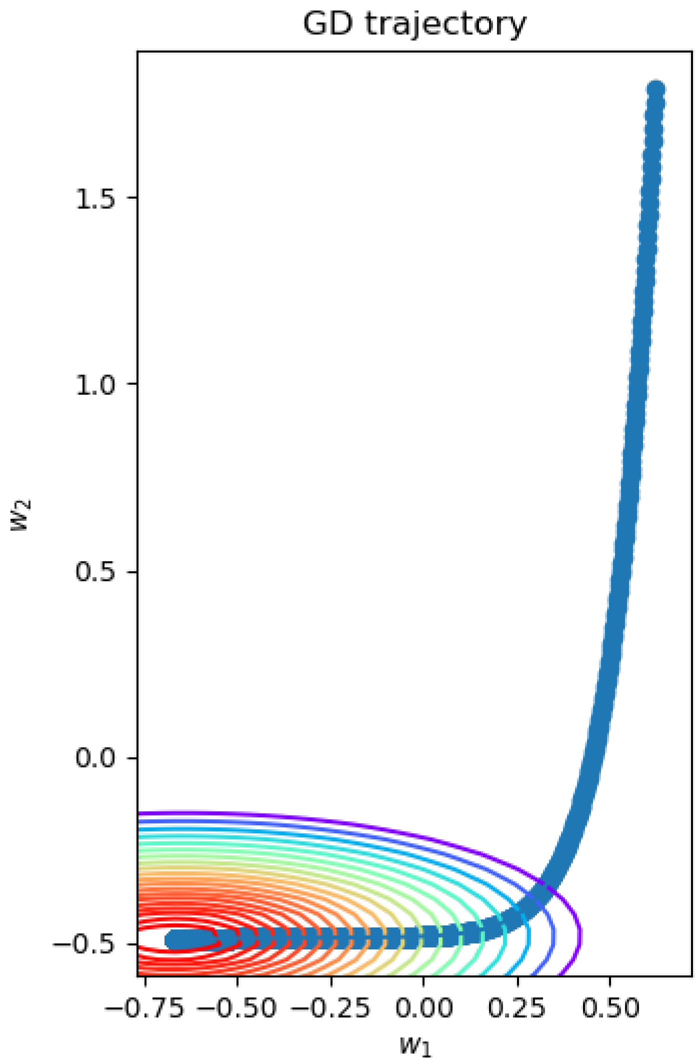
**Задание 1.3: При помощи функций `gradient_descent`
траекторию градиентного спуска для разных значений**
`lr`). Используйте не менее четырёх разных значений для
шага вычисляйтете значение функционала ошибки при пом
первой и последней итерациях градиентного спуска.

Сделайте и опишите свои выводы о том, как параметр `lr`
градиентного спуска

Подсказки:

- Функция `gradient_descent` возвращает историю вес
функцию `plot_gd`
- Хорошие значения для `lr` могут лежать в промежутке

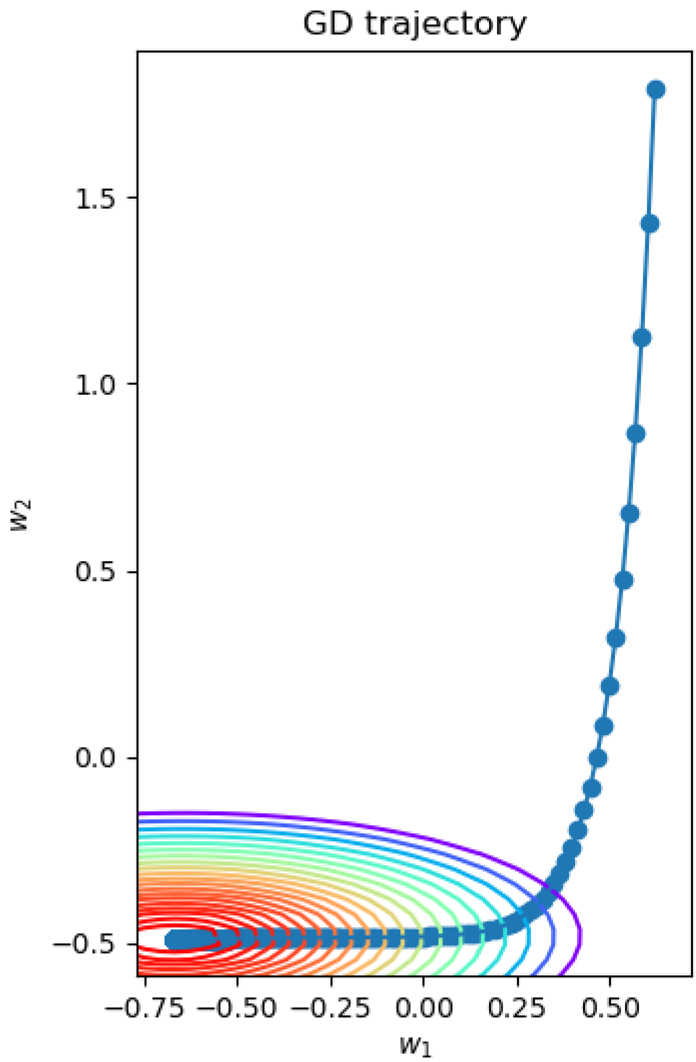
В [16]:
```
w_list = gradient_descent(w_init, X, y, loss,
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  412.53549322933384
Last iteration loss:  0.8670644395649092
```

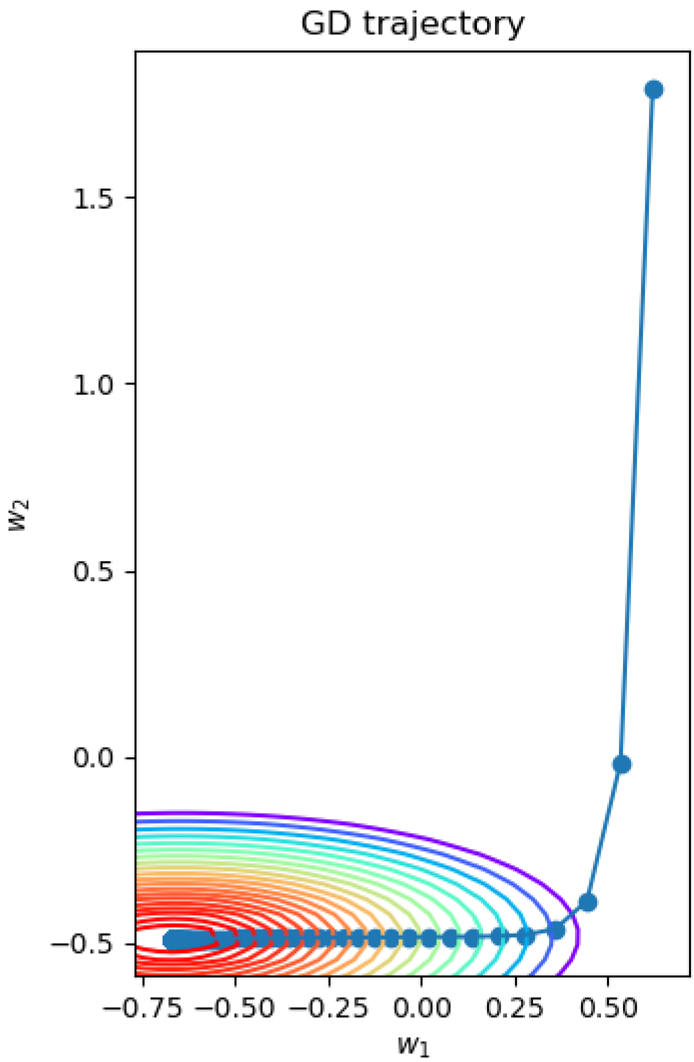
В [17]:

```
w_list = gradient_descent(w_init, X, y, loss,
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



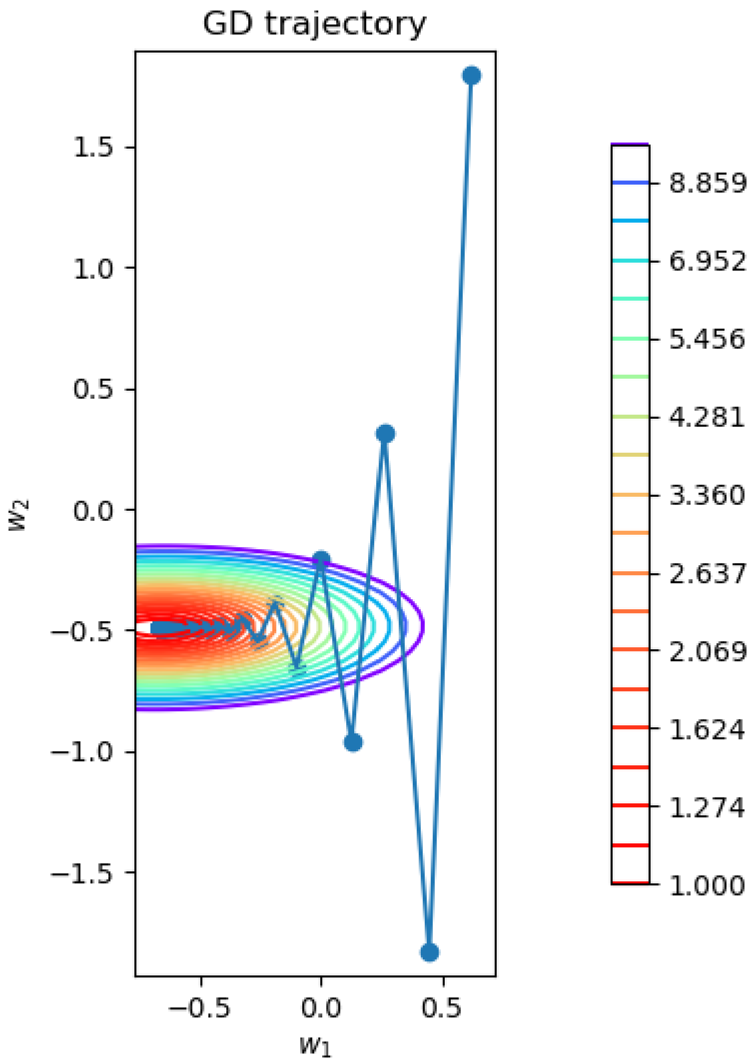

```
First iteration loss:  304.4618116242326
Last iteration loss:  0.8670644395649092
```

В [18]:
```
w_list = gradient_descent(w_init, X, y, loss,
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  29.051696934645758
Last iteration loss:  0.8670644395649092
```

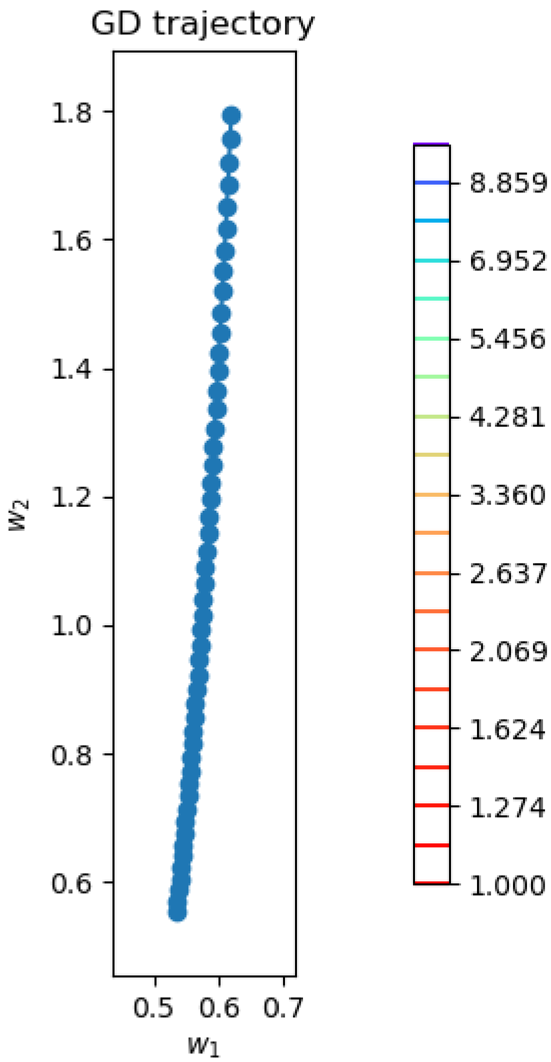
В [19]:
```
w_list = gradient_descent(w_init, X, y, loss,
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  155.26258214352075
Last iteration loss:  0.8670644395649092
```

Различия минимальны. Причиной тому, по всей видимости
настройках функции по умолчанию задан слишком больш
(n_iterations = 100000). Для улучшения наглядности, значи
показатель

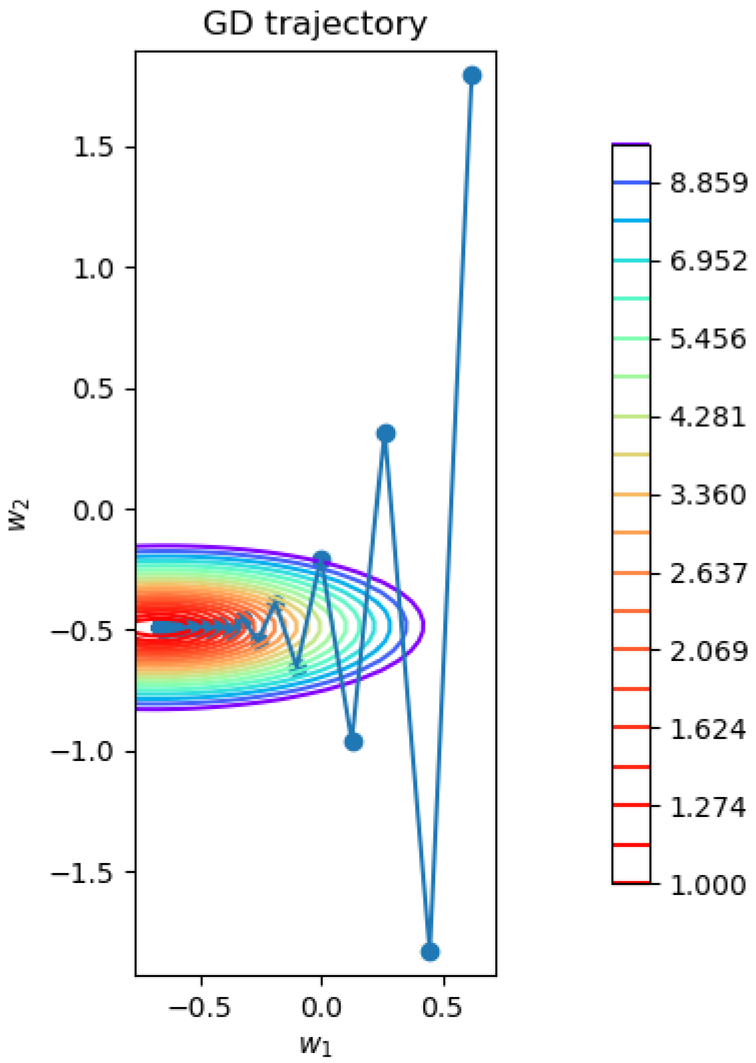
В [22]:
```
w_list = gradient_descent(w_init, X, y, loss,
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  412.53549322933384
Last iteration loss:  97.42290021496474
```

Очевидно, что мы не достигли минимального значения, и з
возросло до 100

В [107]:
```
w_list = gradient_descent(w_init, X, y, loss,
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  155.26258214351958
Last iteration loss:  0.8670654544468456
```

На основании проведенного анализа можно сделать вывод
малом количестве шагов градиентный спуск может не дост

С увеличением числа шагов ситуация улучшается, и спуск
шаге 0,1 достигается значение потерь, аналогичное значен
шагов. На шаге размером 0,3 произошло нечто, приведшее
1,642 * 10 ^ 59.

Задание 1.4: Реализуйте функцию `stochastic_gradien`

Функция должна принимать все те же параметры, что и фу
но ещё параметр `batch_size` , отвечающий за размер бат

Функция должна как и раньше реализовывать цикл, в котор
градиентного спуска, но на каждом шаге считать градиент
только по случайно выбранной части.

Подсказка: для выбора случайной части можно использова
(https://numpy.org/doc/stable/reference/random/generated/nun
правильным параметром `size`, чтобы выбрать случайные
проиндексировать получившимся массивом массив `X`:

```
batch_indices = np.random.choice(X.shape[0], si
ace=False)
batch = X[batch_indices]
```

(*здесь np.random.choice генерирует из np.arange(X.shape[*
replace=False - без повторения)

В [39]:
```
X.shape[0]
```

Out[39]:
```
300
```

В [40]:
```
def stochastic_gradient_descent(
    w_init: np.ndarray,
    X: np.ndarray,
    y: np.ndarray,
    loss: BaseLoss,
    lr: float,
    batch_size: int,
    n_iterations: int = 1000,
) -> List[np.ndarray]:
    grad_list = []
    batch_indices = np.random.choice(X.shape[0
    batch = X[batch_indices]
    batch_y = y[batch_indices]

    for i in range(n_iterations):
        grad_list.append(w_init)
        w_init = w_init - lr * loss.calc_grad(
    grad_list.append(w_init)
    return grad_list
```

Функция stochastic_gradient_descent принимает следующие

- w_init - начальные значения параметров модели.
- X - функции ввода.
- y - целевые значения.
- loss - экземпляр функции потерь, которая вычисляет гр
- lr - скорость обучения.
- batch_size - размер пакета, используемого для вычисл
- n_iterations - количество итераций (по умолчанию уста

Функция инициализирует пустой список grad_list для хране
на каждой итерации. Затем она случайным образом выбир
входных данных, X используя np.random.choice. Эти индек
создания пакета входных функций batch и соответствующи
batch_y.

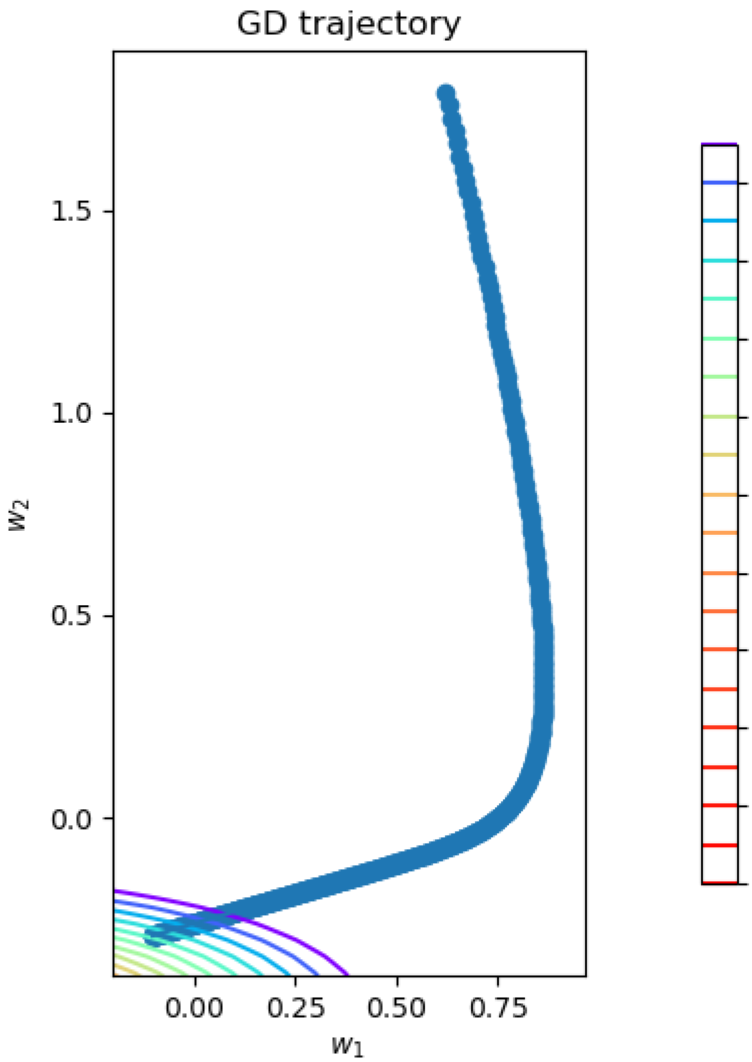
Затем функция входит в цикл, который повторяется n_itera
итерации она добавляет текущие значения параметров w_
обновляются с помощью формулы, w_init = w_init - lr * loss.
w_init)где loss.calc_grad вычисляется градиент функции по
параметров.

Задание 1.5: При помощи функций `stochastic_gradier` нарисуйте траекторию градиентного спуска для разны (параметра `lr`) и размера подвыборки (параметра `ba` менее трех разных значений для `lr и batch_size` . (Для размера подвыборки вычисляйтете значение функционала метода `calc_loss` на первой и последней итерациях стох спуска).

Сделайте и опишите свои выводы о том, как параметры `l` поведение стохастического градиентного спуска. Как отлич стохастического градиентного спуска от обычного?

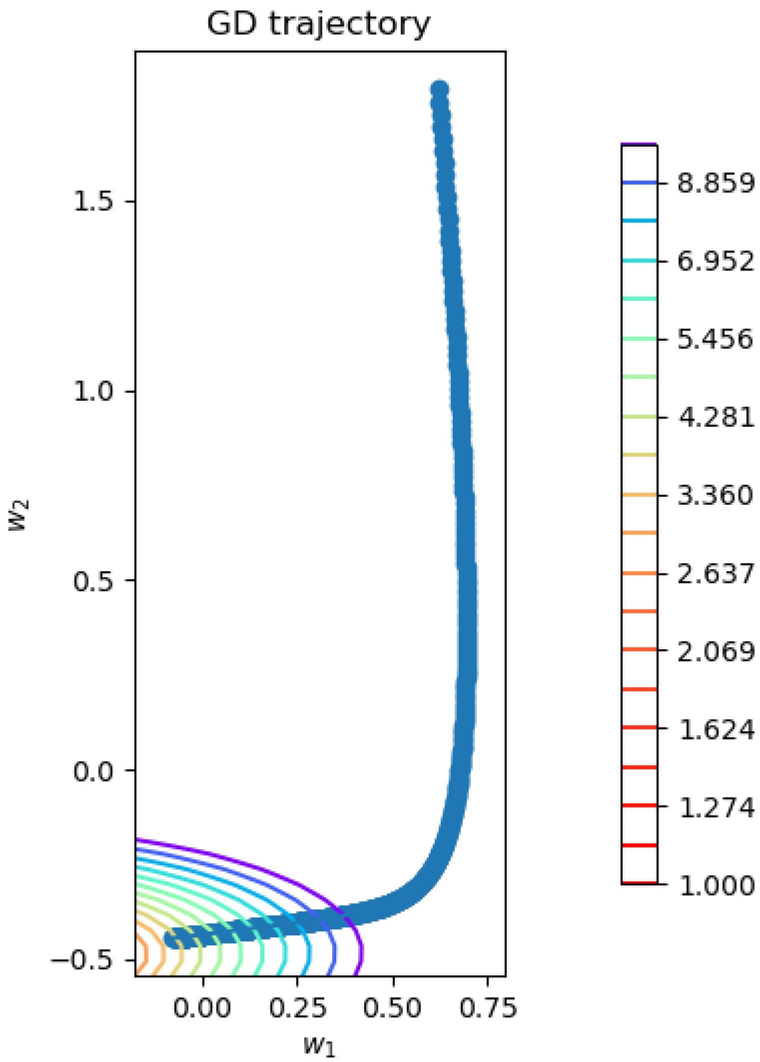
Обратите внимание, что в нашем датасете всего 300 объе больше этого числа не будет иметь смысла.

В [64]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



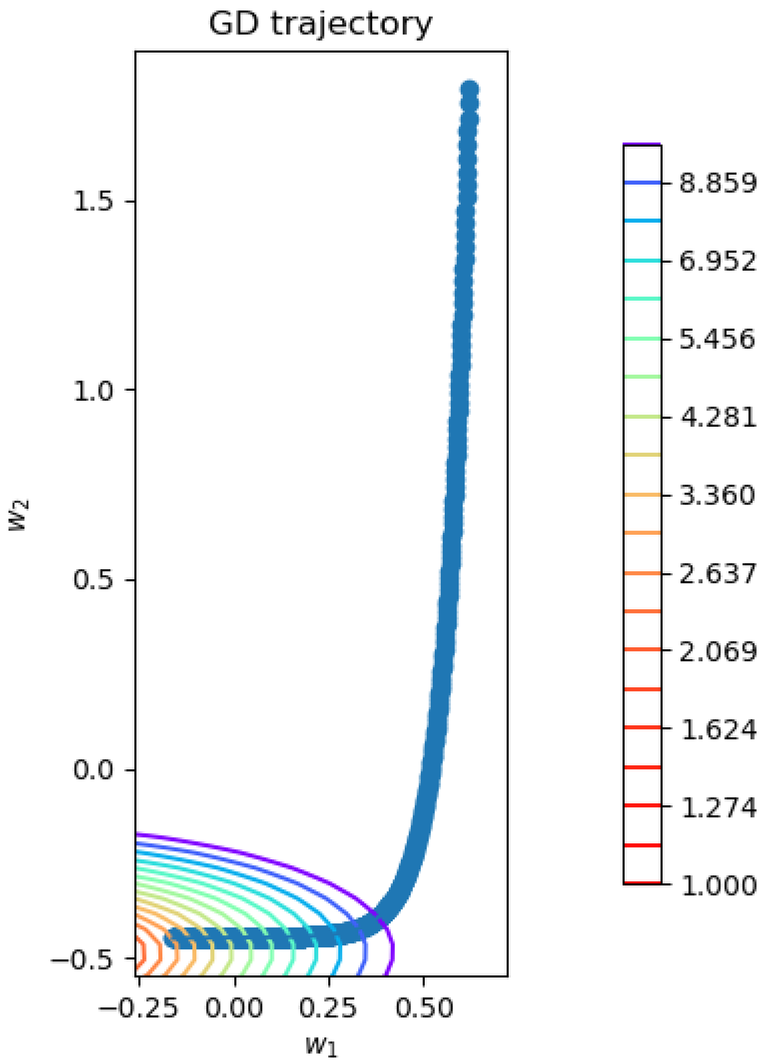

```
First iteration loss:  413.8806671894657
Last iteration loss:  6.333267456575259
```

В [65]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



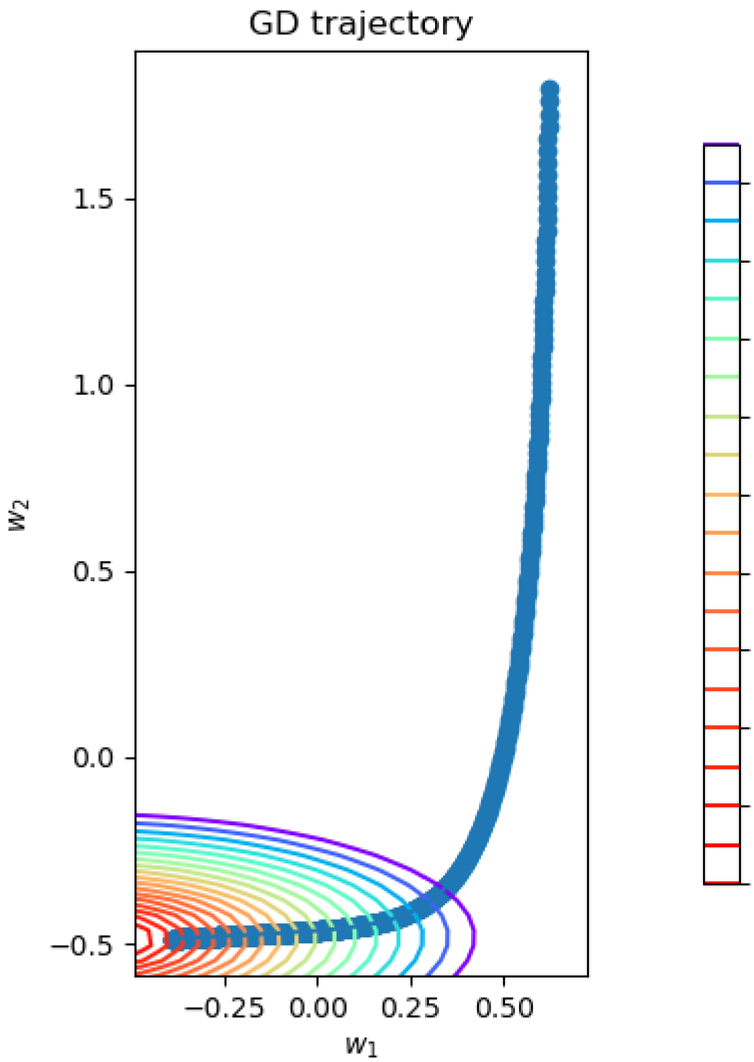

```
First iteration loss:  413.51183006890705
Last iteration loss:  3.6912358545991735
```

В [66]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  411.923969305594
Last iteration loss:  2.9742634047508036
```

В [68]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```

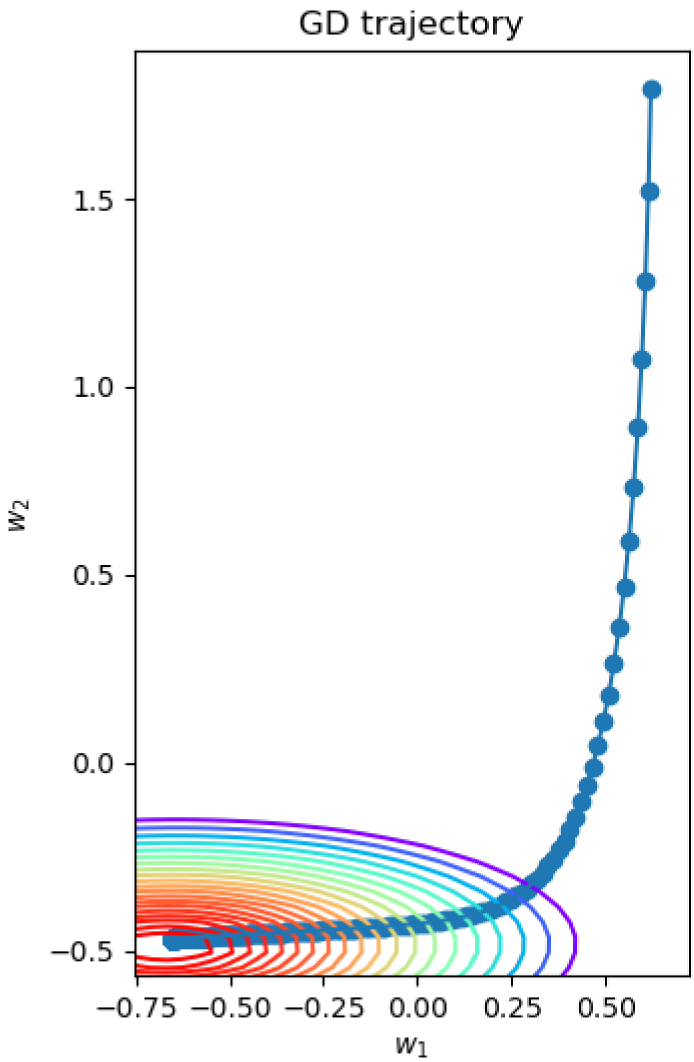



```
First iteration loss:  413.200675003692
Last iteration loss:  1.4725004316102306
```

С увеличением количества объектов точность модели повь

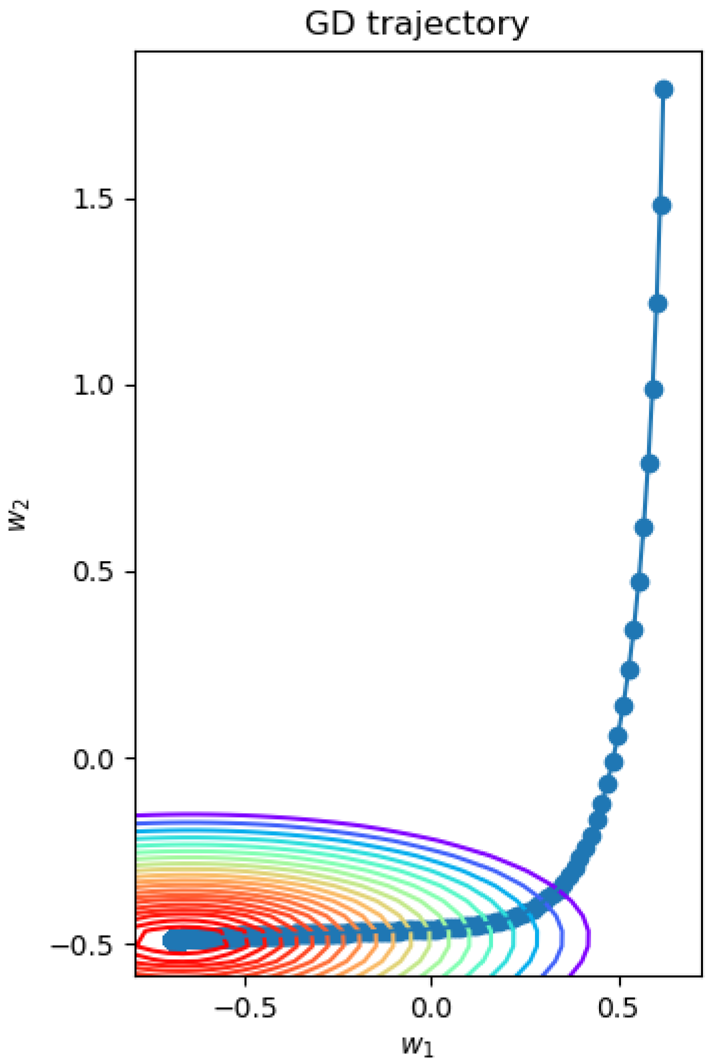
При 100 объектах она уже почти достигает минимума. Дале
те же значения объектов и увеличить размер шага.

В [71]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  333.11166740747166
Last iteration loss:  0.9072502252882477
```

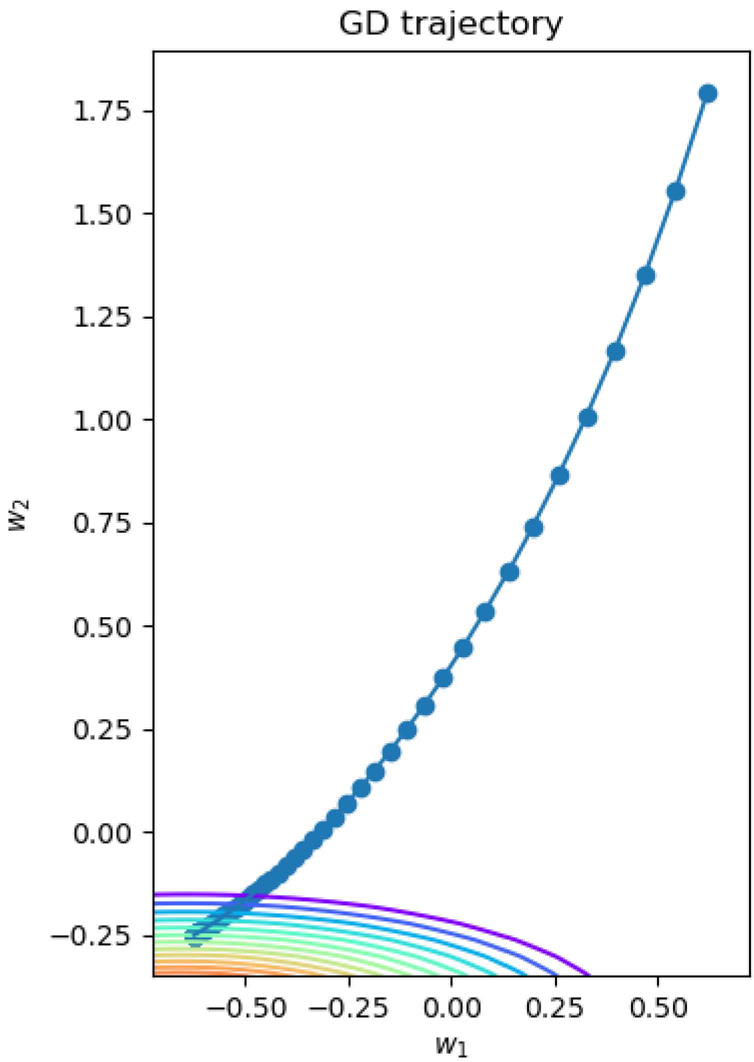
В [72]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  320.97618013755493
Last iteration loss:  0.8740066409316497
```

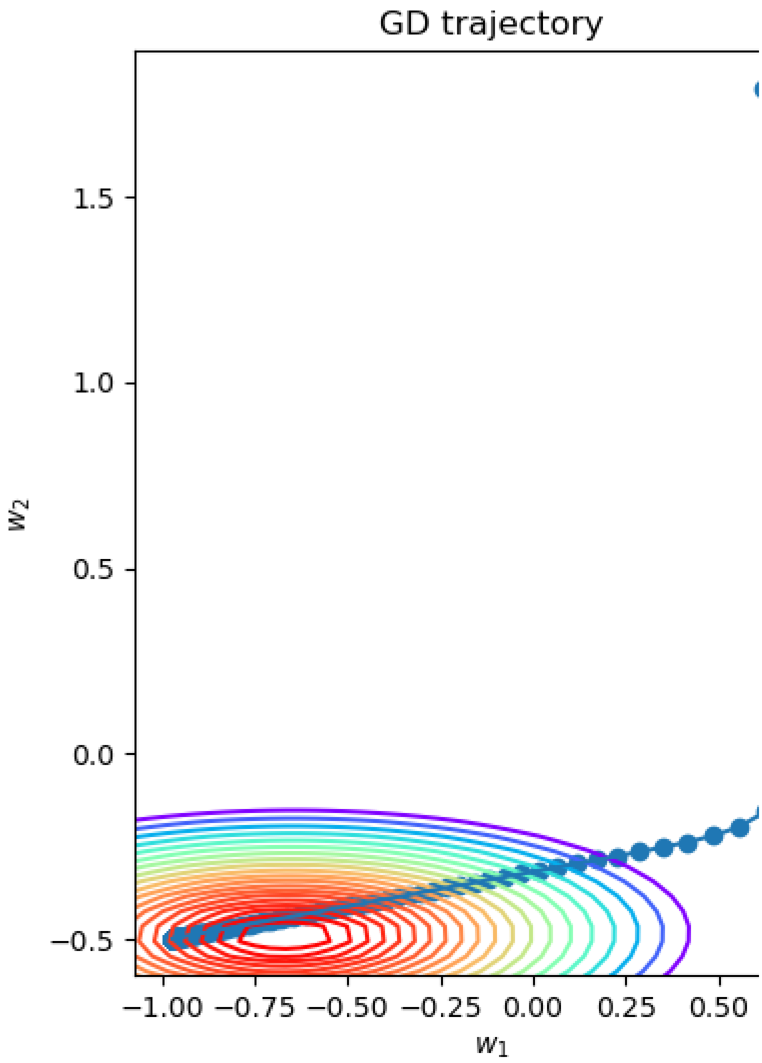
Очень близко к минимуму подошли. Видно, что если увели
успевать доходить до минимума. Можно еще попробовать
взять, например, 100000, как в простом градиентном спуск
мы бы получили эффективый loss.

В [73]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



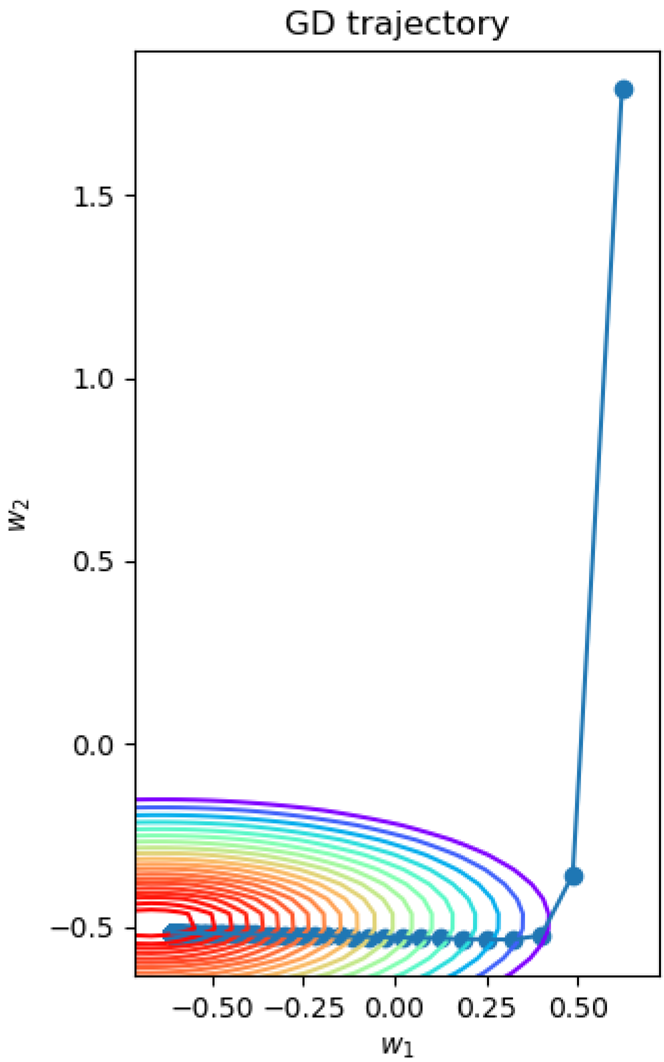

```
First iteration loss:  342.9916309778918
Last iteration loss:  5.447094712720428
```

В [74]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



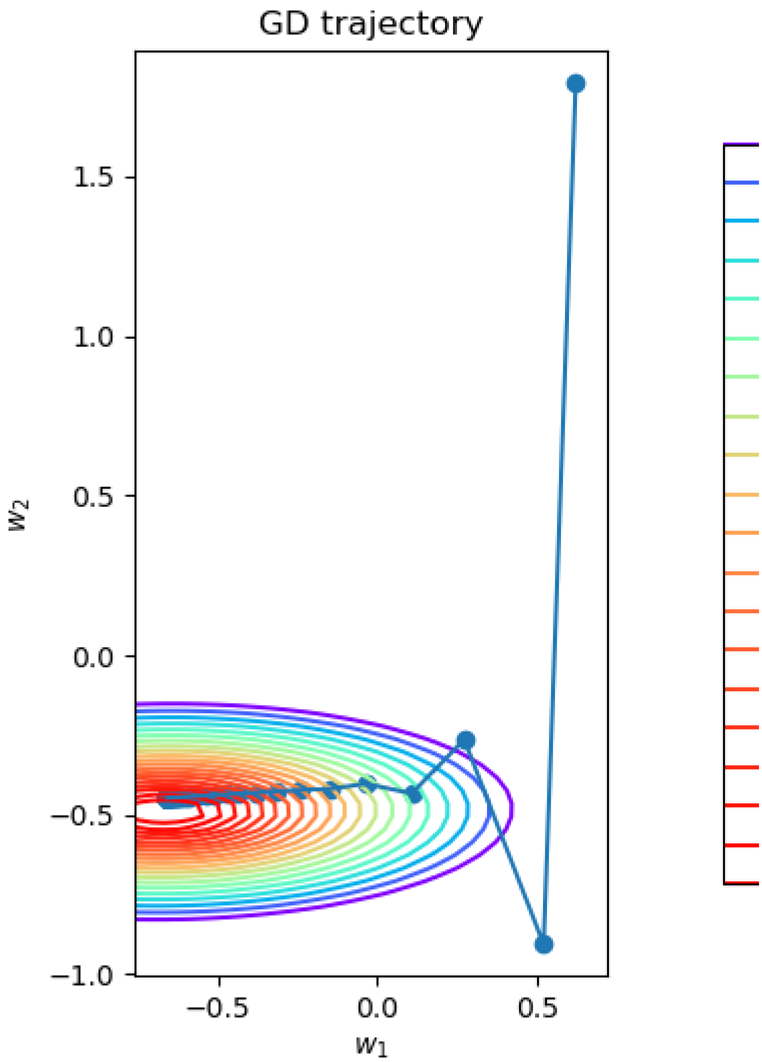

```
First iteration loss:  108.46361207138666
Last iteration loss:  1.5739612050157534
```

В [76]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



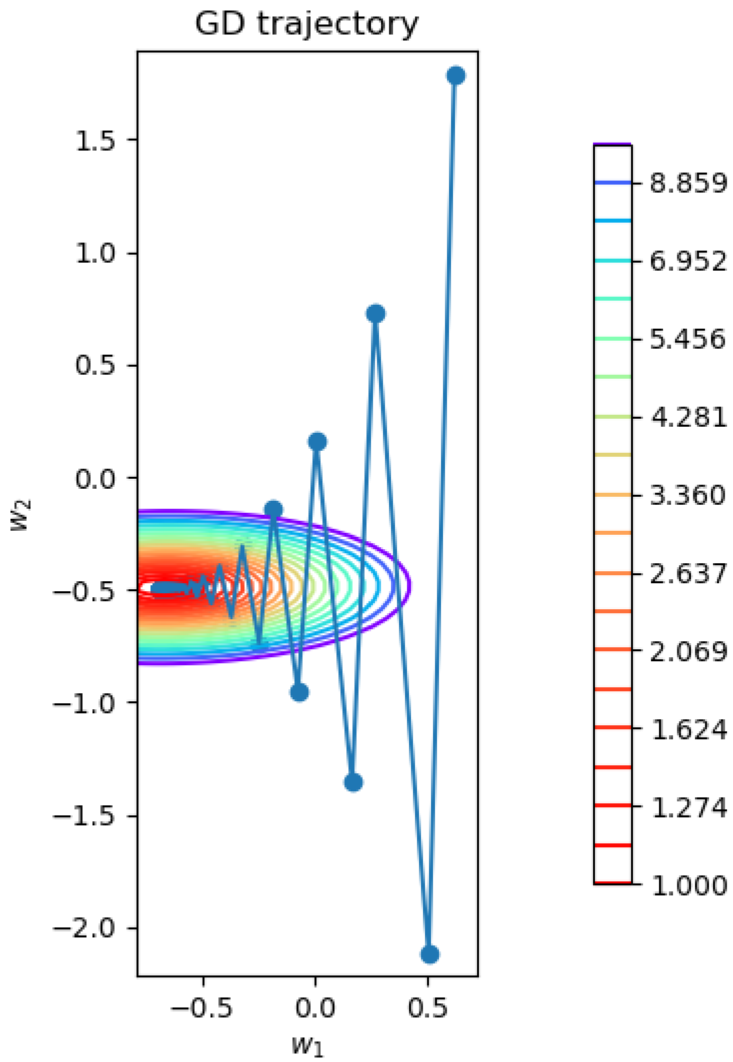

```
First iteration loss:  12.324328164519052
Last iteration loss:  0.9494737268620215
```

В [114]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  26.047064784562675
Last iteration loss:  1.0157915932020185
```

В [116]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  224.6651069191318
Last iteration loss:  0.8728661708031257
```

Исследование показывает, что длина шага (lr) и batch size ı влияние на процесс градиентного спуска. Малые значения что точка минимума не будет достигнута, а большой batch вычислительных ресурсов. Необходимо найти оптимальны вычислений и качеством результата. Рекомендуется устанс который обеспечивает хороший результат, требуя вдвое ме Стохастический метод менее требователен и выполняется часть объектов, но также демонстрирует хорошие результа

Можно заметить, что поведение градиентного спуска, особ версии, очень сильно зависит от размера шага.

Как правило, в начале спуска мы хотим делать большие ш подойти поближе к минимуму, а позже мы уже хотим делат более точнее этого минимума достичь и не "перепрыгнуть"

Также следует учитывать, что стохастический метод менее
выполняется быстрее, но может демонстрировать хорошие
учитывая только часть объектов.

Рекомендуется выбирать оптимальный баланс между скор
качеством результата. Например, установка batch_size рав
хороший результат и требовать вдвое меньше вычислений

Чтобы достичь такого поведения мы можем постепенно ум
увеличением номера итерации. Сделать это можно, напри
итерации длину шага по следующей формуле:

$$\eta_t = \lambda \left(\frac{s_0}{s_0 + t} \right)^p$$

где η_t — длина шага на итерации t, λ — начальная длина ш

Задание 1.6: Реализуйте функцию `stochastic_gradien`
шагом по формуле выше. Параметр s_0 возьмите равным
нового аргумента функции `p` .

В [118]:

```
def stochastic_gradient_descent(
    w_init: np.ndarray,
    X: np.ndarray,
    y: np.ndarray,
    loss: BaseLoss,
    lr: float,
    batch_size: int,
    p: float,
    n_iterations: int = 1000,
) -> List[np.ndarray]:

    grad_list = []
    batch_indices = np.random.choice(X.shape[0
    batch = X[batch_indices]
    batch_y = y[batch_indices]
    for i in range(n_iterations):
        grad_list.append(w_init)
        w_init = w_init - lr*(1/(1+i))**p * lo
    return grad_list


```

Функция stochastic_gradient_descent принимает несколько

- w_init - начальные значения параметров модели.
- X - входные характеристики набора данных.
- y - целевые значения набора данных.
- loss - экземпляр класса функции потерь, который вычи
- lr - скорость обучения алгоритму SGD.
- batch_size - размер пакета, используемого для каждой
- p - параметр мощности, используемый при снижении с
- n_iterations - количество итераций для выполнения (по

Функция возвращает список массивов numpy, grad_list кото
значения параметров на каждой итерации.

Мы сначала определяем класс функции потерь, MeanSqua
среднеквадратичную ошибку потерь и ее градиент. Затем м
случайные данные X и y для демонстрационных целей.

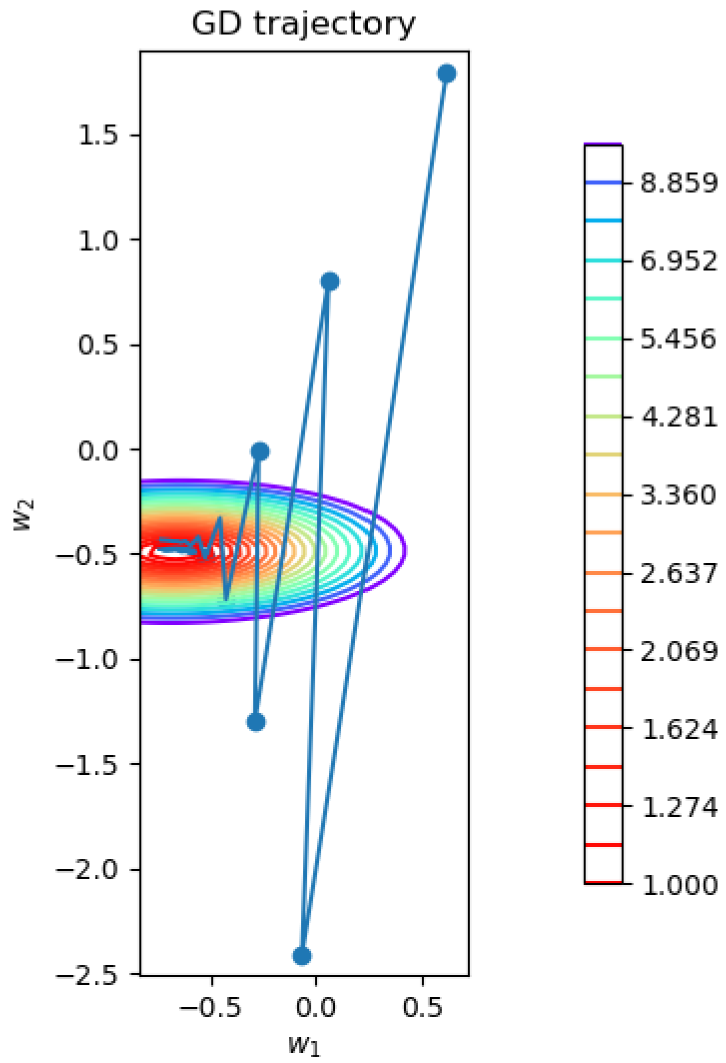
Далее мы инициализируем параметры модели w_init в вид
создаем экземпляр MeanSquaredError функции потерь и за
такие как скорость обучения (lr), размер пакета (batch_size
количество итераций (n_iterations).

Наконец, мы применяем stochastic_gradient_descent функц
входными данными и сохраняем обновленные значения па
Затем мы печатаем обновленные значения параметров на

Задание 1.7: При помощи новой функции `stochastic_g` функции `plot_gd` нарисуйте траекторию градиентного значений параметра `p`. Используйте не менее четырёх р
Хорошими могут быть значения, лежащие в промежутке от
возьмите равным 0.01, а параметр `batch_size` равным 1
параметра `p` вычисляйтете значение функционала ошибк
`calc_loss` на первой и последней итерациях стохастичес

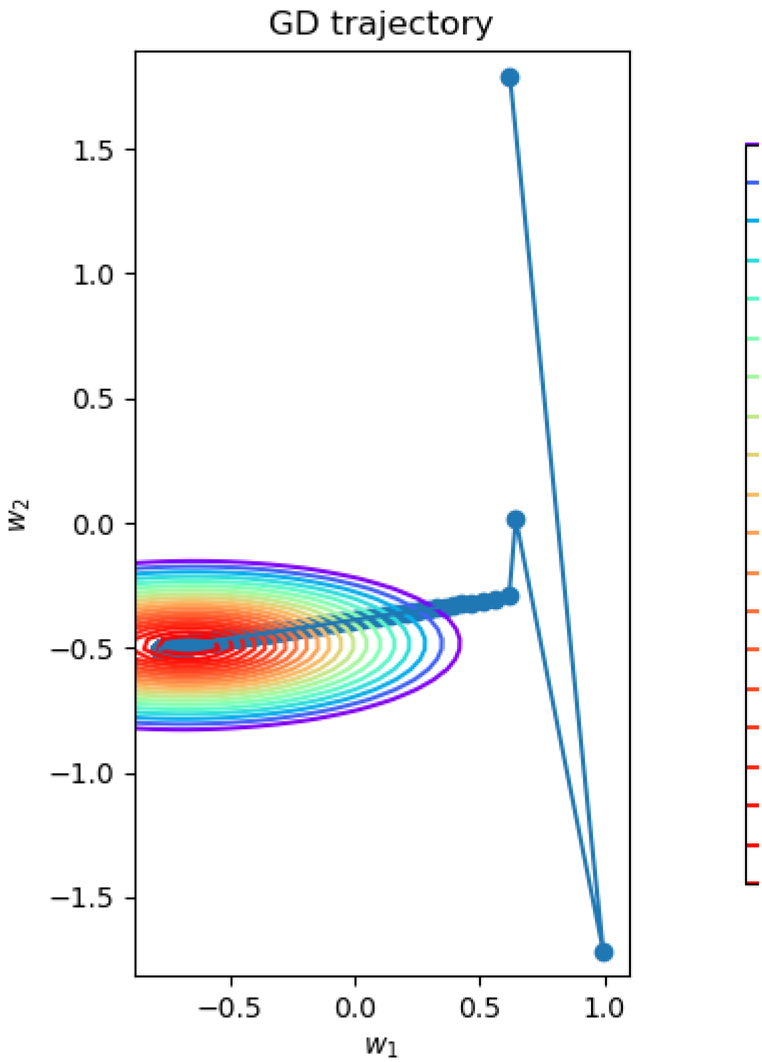
Сделайте и опишите свои выводы о том, как параметр `p` в
стохастического градиентного спуска

В [119]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



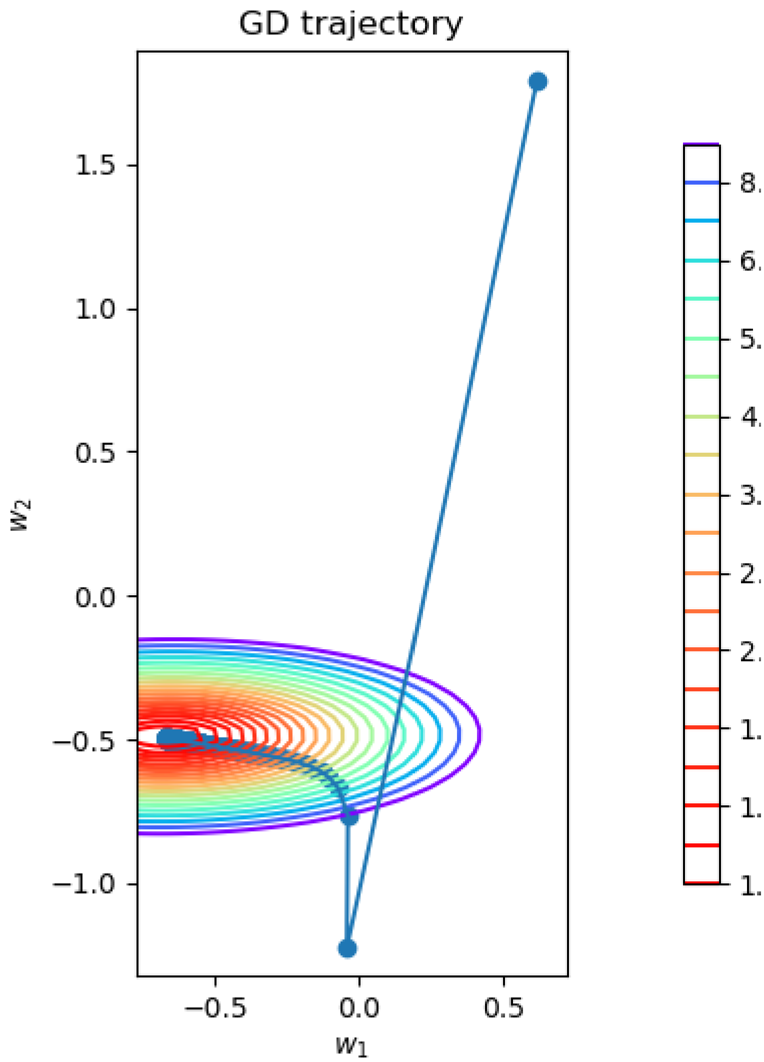

```
First iteration loss:  298.2310061325282
Last iteration loss:  1.171475933393098
```

В [121]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```



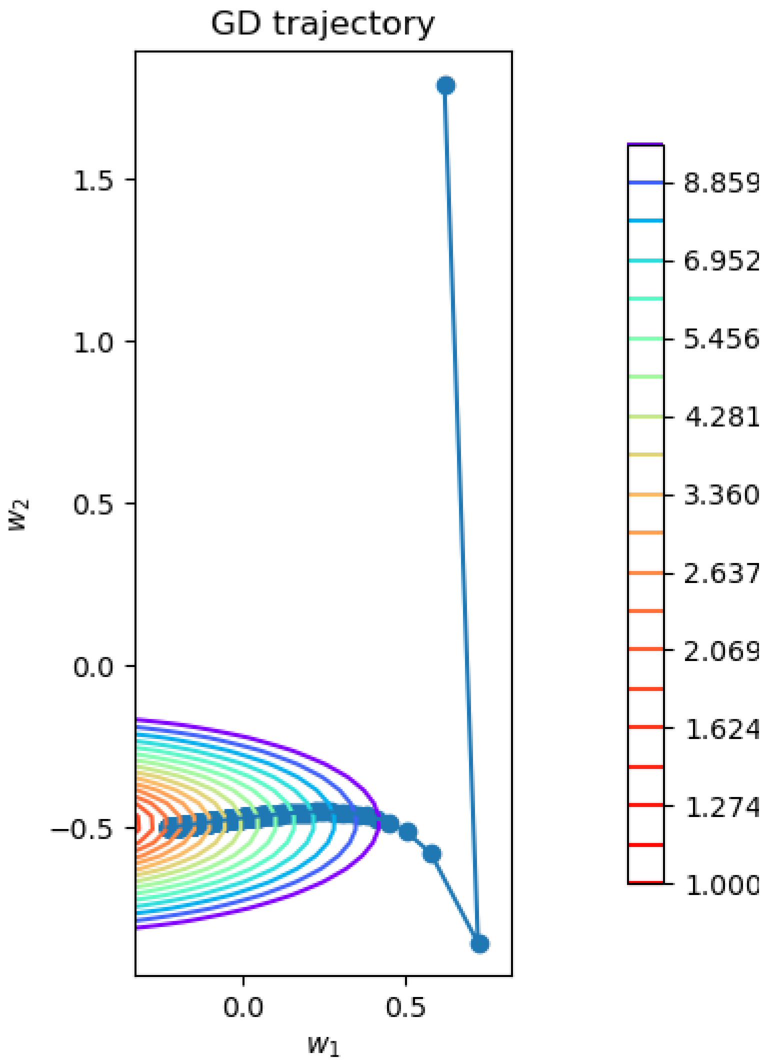

```
First iteration loss:  144.5542390836199
Last iteration loss:  0.9786857142942068
```

В [122]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  47.28956578327367
Last iteration loss:  0.8697160468376856
```

В [123]:
```
w_list = stochastic_gradient_descent(w_init, X
plot_gd(w_list, X, y, loss)
print('First iteration loss: ',loss.calc_loss(
print('Last iteration loss: ',loss.calc_loss(X
```




```
First iteration loss:  27.01798314255404
Last iteration loss:  2.353688398024119
```

По результатам исследования видно, что при увеличении з
величина шагов уменьшается. При значениях p от 0,1 до 0,
происходит недостаточно быстро, вследствие чего возмож
минимума. При значении 0,75 достигается оптимальное со
дальнейшее увеличение значения p приводит к слишком м
результате чего алгоритм не доходит до точки минимума (з
является слишком большим).

Задание 1.8: Сравните сходимость обычного градиент стохастической версии: Нарисуйте график зависимости з ошибки (лосса) (его можно посчитать при помощи метода из датасета и w с соответствующей итерации) от номера и полученных при помощи обычного и стохастического град **одинаковыми параметрами**. В SGD параметр `batch_siz` `p=0` .

В [126]:
```
print(n_features)
print(n_objects)
print(batch_size)
print(num_steps)
```

```
2
300
10
43
```

В [147]:
```
steps = np.arange(50)

w_list_gd = gradient_descent(w_init, X, y, los
grad_list=[(loss.calc_loss(X, y, w_list_gd[i])

w_list_sgd = stochastic_gradient_descent(w_ini
st_grad_list=[(loss.calc_loss(X, y, w_list_sgd
```

В [148]:
```
plt.plot(steps, grad_list, label='GD', color='
plt.plot(steps, st_grad_list, label='SGD')
plt.xlabel('steps')
plt.ylabel('Функционал ошибки')
plt.title('Изменение функционала ошибки от шаг
plt.legend()
plt.show()
```