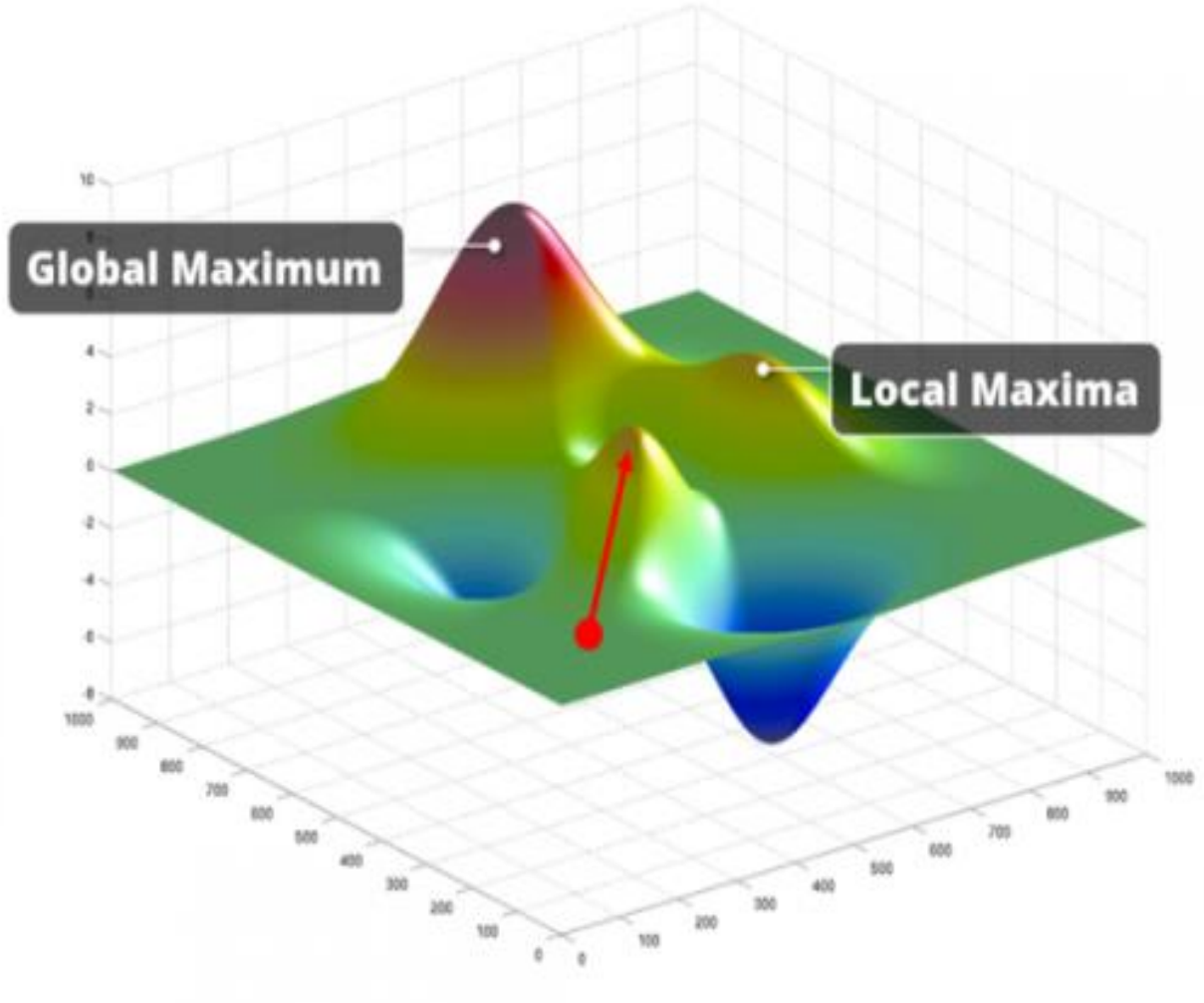
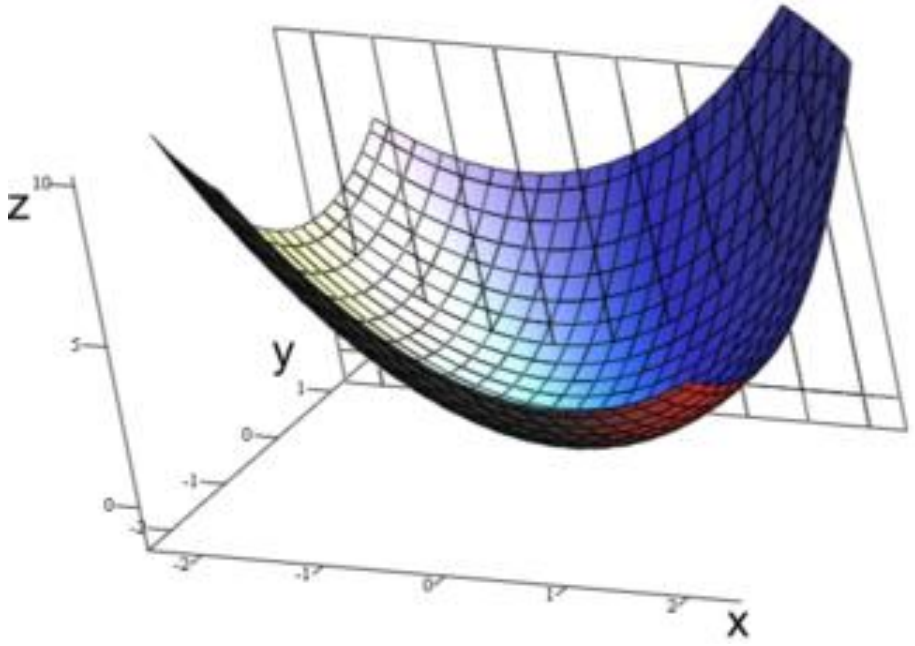
Градиентный спуск в машинном обvчении

Градиентный спуск — самый используемый алгоритм обучения, он применяется почти в каждой модели машинного обучения.
Градиентный спуск — это, по сути, и есть то, как обучаются модели. Без градиентного спуска машинное обучение не было бы там, где сейчас. Метод градиентного спуска с некоторой модификацией широко используется для обучения персептрона и глубоких нейронных сетей, используется и в методе градиентного бустинга – самом популярном метод решения задач классификации и регрессии для структурированных (табличных) данных



Частные производные



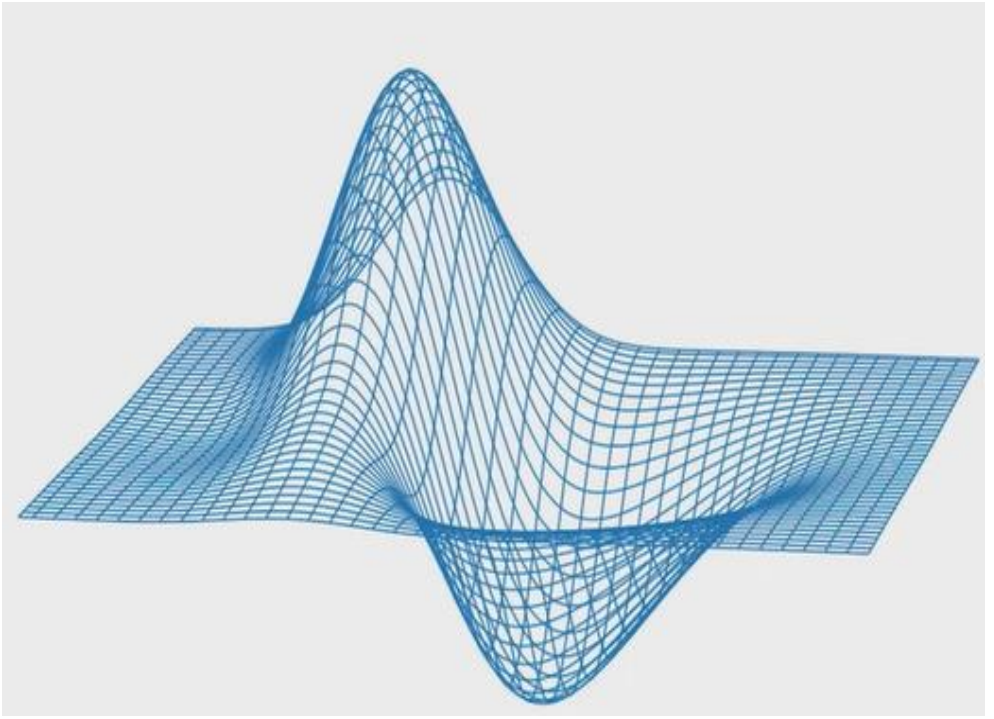


График функции $z = x^2 + xy + y^2$.
Частная производная в точке (1, 1, 3) при постоянном y соответствует углу наклона касательной прямой, параллельной плоскости xz.

$$f(x,y) = e^{x^2+\sqrt{y}}$$

$$\frac{\partial f}{\partial x} = e^{x^2+\sqrt{y}}.(2x)$$

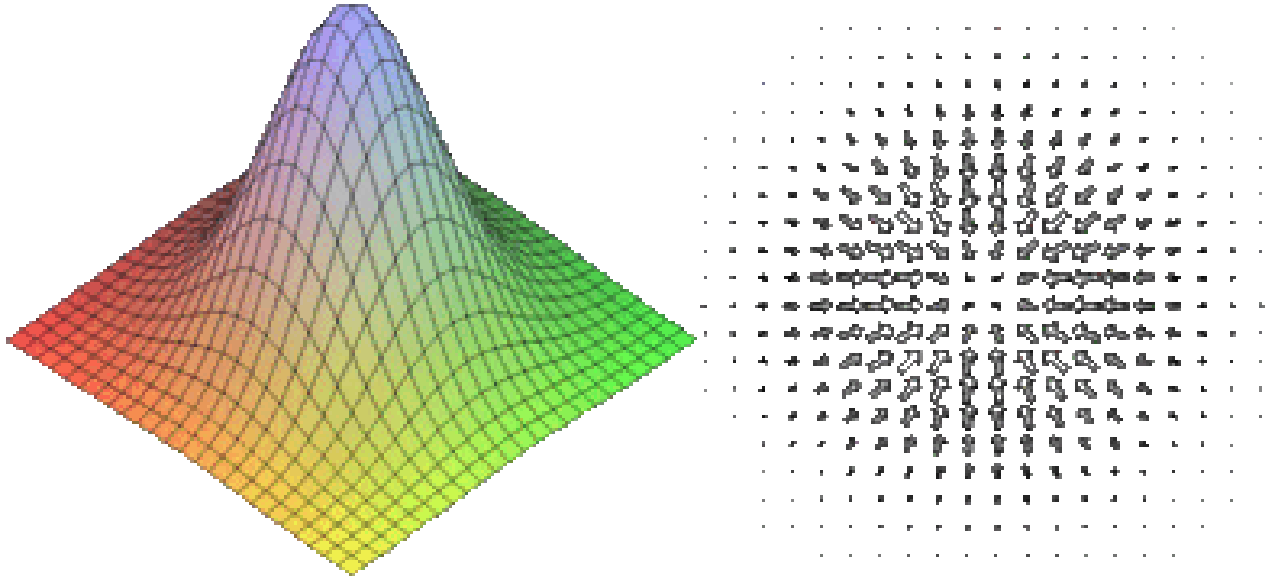
$$\frac{\partial f}{\partial y} = e^{x^2+\sqrt{y}}\left(\frac{1}{2\sqrt{y}}\right)$$

Градиент

Градиент — **вектор**, своим направлением указывающий ***направление*** наибольшего возрастания некоторой величины φ, значение которой меняется от одной точки пространства к другой (скалярного поля), а по величине (модулю) равный ***скорости роста*** этой величины в этом направлении.

Другими словами, градиент — это производная по пространству, но в отличие от производной по одномерному времени, градиент является не скаляром, а векторной величиной.

Например, для функции



Операция градиента преобразует холм (слева), если смотреть на него сверху, в поле векторов (справа). Видно, что векторы направлены «в горку» и чем длиннее, тем круче наклон.

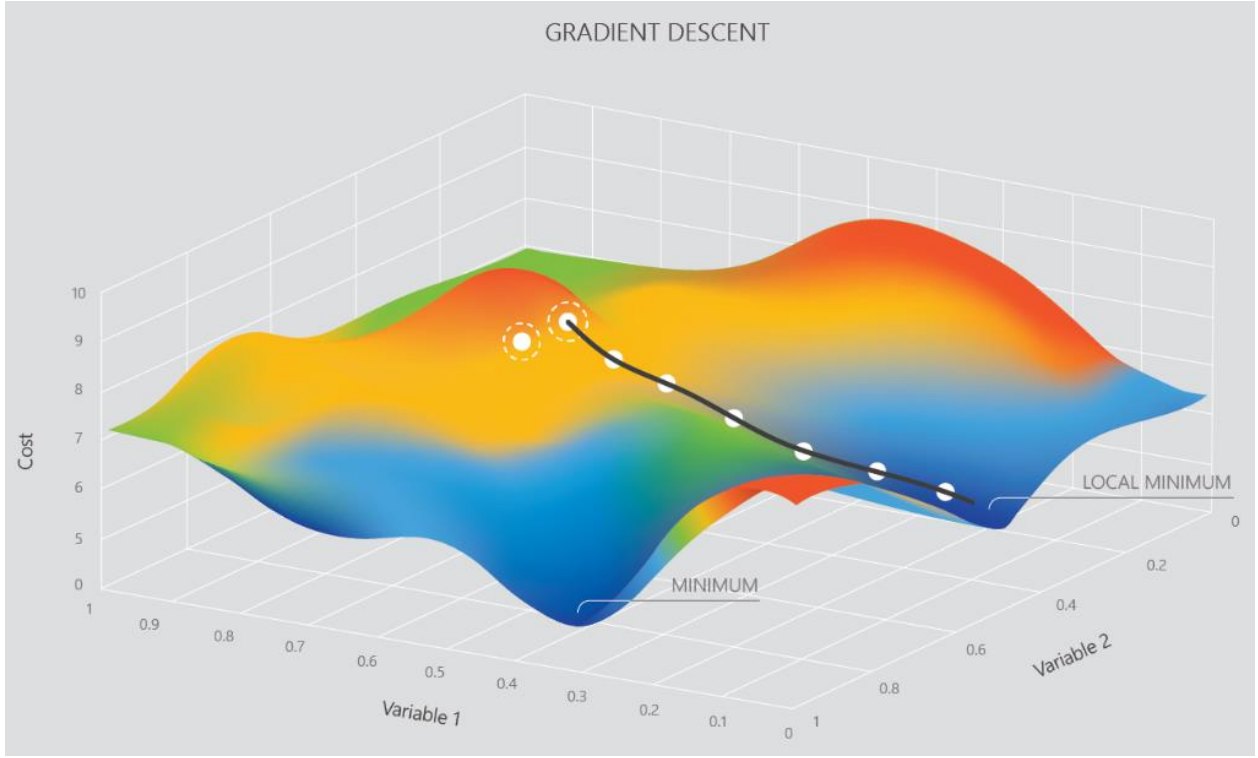
$$\varphi(x,\ \mathrm{y}, \mathrm{z}\ ) = 2\mathrm{x} + 3y^2 - sinz.$$

$$\nabla\varphi = \left(\frac{\partial\varphi}{\partial x},\ \frac{\partial\varphi}{\partial y},\ \frac{\partial\varphi}{\partial z}\right) = (2,\ 6y,\ -\cos z)$$

Что такое градиентный спуск

Градиентный спуск — метод нахождения минимального значения функции потерь (существует множество видов этой функции). Минимизация любой функции означает поиск самой глубокой впадины в этой функции.

Функция стоимости или **Loss** используется, чтобы контролировать ошибку в прогнозах модели машинного обучения. Поиск минимума означает получение наименьшей возможной ошибки или повышение точности модели. Мы увеличиваем точность, перебирая набор учебных данных при настройке параметров нашей модели (весов и смещений для нейронной сети).



Обучение и функция стоимости (Loss function)

Обучение модели означает определение хороших значений для всех весов и смещений из отмеченных примеров. Функция потерь является мерой расхождения между истинным значением оцениваемого параметра и оценкой параметра.

В контролируемом обучении (обучении с учителем) алгоритм машинного обучения строит модель, анализируя множество примеров и пытаясь найти модель, минимизирующую loss; этот процесс называется минимизацией эмпирического риска.

Loss - это штраф за плохой прогноз. То есть loss - это число, указывающее, насколько плохое предсказание модели было на одном примере. Если прогноз модели идеален, потери (ошибки) равны нулю; в противном случае loss больше нуля. Цель обучения модели - найти набор весов и смещений, которые имеют низкие потери в среднем по всем примерам.

Наиболее часто используемой является **квадратичная функция потерь** (L_2-норма)

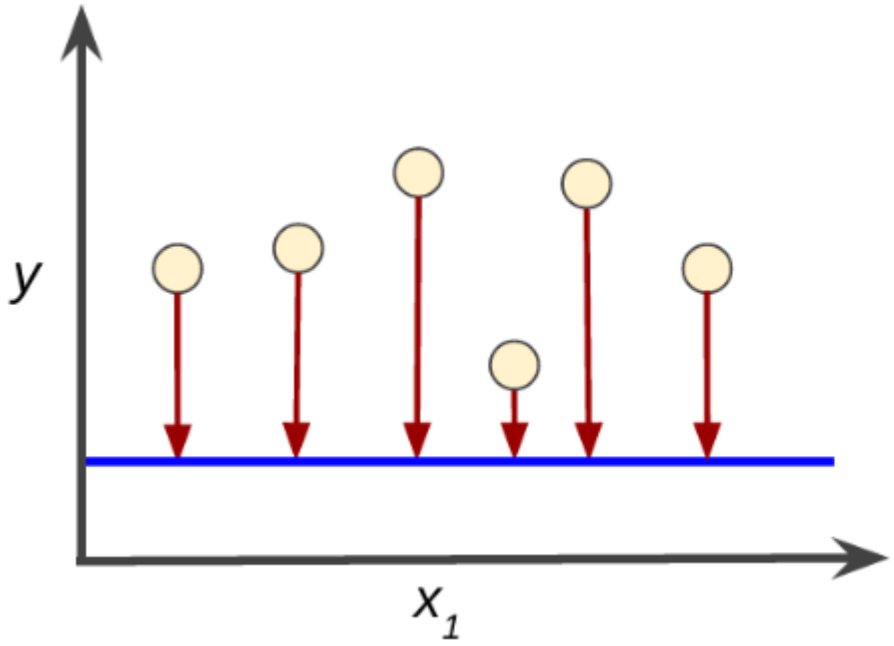
= квадрат разницы между меткой и предсказанием

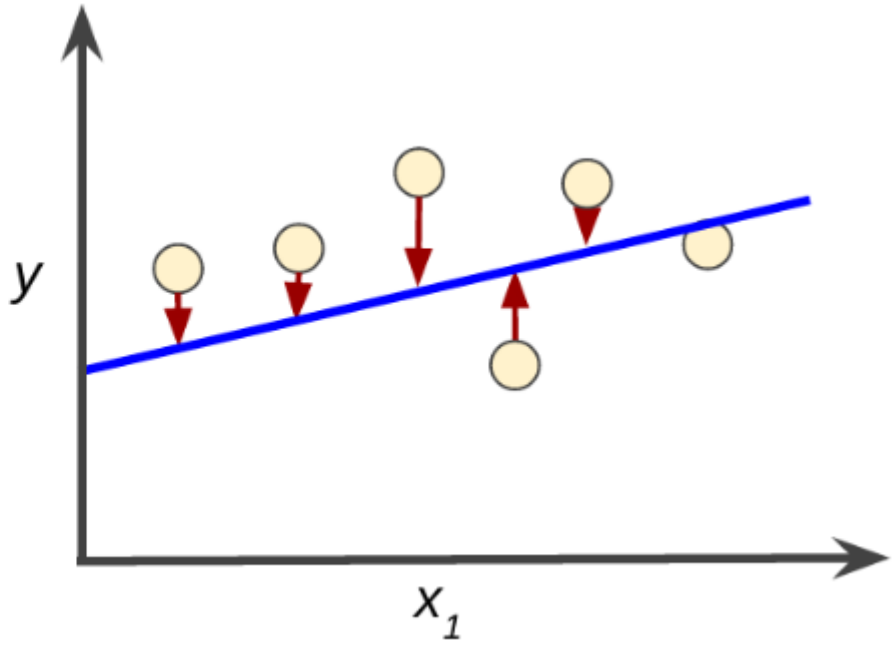
$= (\text{observation} - \text{prediction}(x))^2$

$= (y - y')^2$

Преимуществом квадратичной функции потерь являются инвариантность к знаку - значение функции всегда положительно. Т.е. независимо от знака ошибки результат будет один и тот же. Квадратичная функция потерь используется в моделях, параметры которых оцениваются на основе метода наименьших квадратов, например линейной регрессии.

Например, на рисунке показана модель с высокими loss слева и модель с низкими loss справа. Обратите внимание на следующее на рисунке:
Стрелки представляют loss. Синие линии представляют прогнозы.





Обратите внимание, что стрелки на левом графике намного длиннее, чем их аналоги на правом графике. Ясно, что линия на правом графике - гораздо лучшая прогностическая модель, чем линия на левом графике.
Вы можете задаться вопросом, можете ли вы создать математическую функцию - функцию потерь, которая бы содержательно объединяла отдельные потери.

Среднеквадратичная ошибка (MSE) - это среднеквадратичная потеря на пример по всему набору данных. Чтобы вычислить MSE, суммируйте все возведенные в квадрат потери для отдельных примеров и затем разделите на число примеров:

$$MSE = \frac{1}{N} \sum_{(x,y) \in D} (y - prediction(x))^2$$

- где *(x,y)* это пример в котором:
  - *x* - набор функций (признаков, входов), который модель использует для прогнозирования
  - *y* - метка примера
- *prediction(x)* – предсказанное моделью значение
- *D* - набор данных, содержащий множество отмеченных примеров *(x,y)*
- *N* количество примеров в *D*

Квадратичную потерю (функцию стоимости) можно записать в следующем виде:

$$J\left(y, y^{/}\right) = C(y - y^{/})^2,$$

где де *C* - константа, y, - истинное значение выхода модели (которое должно быть получено в идеальном случае), $y^{/}$ - фактический выход модели.
Хотя MSE обычно используется в машинном обучении, это не единственная практическая функция потерь и не лучшая функция потерь при любых условиях.

В бинарной классификации используется двоичная функция потерь (0-1 *loss function*), которая определяется следующим образом:

$$J\left(y, y^{/}\right) = f(y \neq y^{/}).$$

Как видно, потери определяются появлением двух взаимоисключающих состояний выхода модели.

Используется также и простая функция потерь, равная разности истинного и фактического выходов модели:

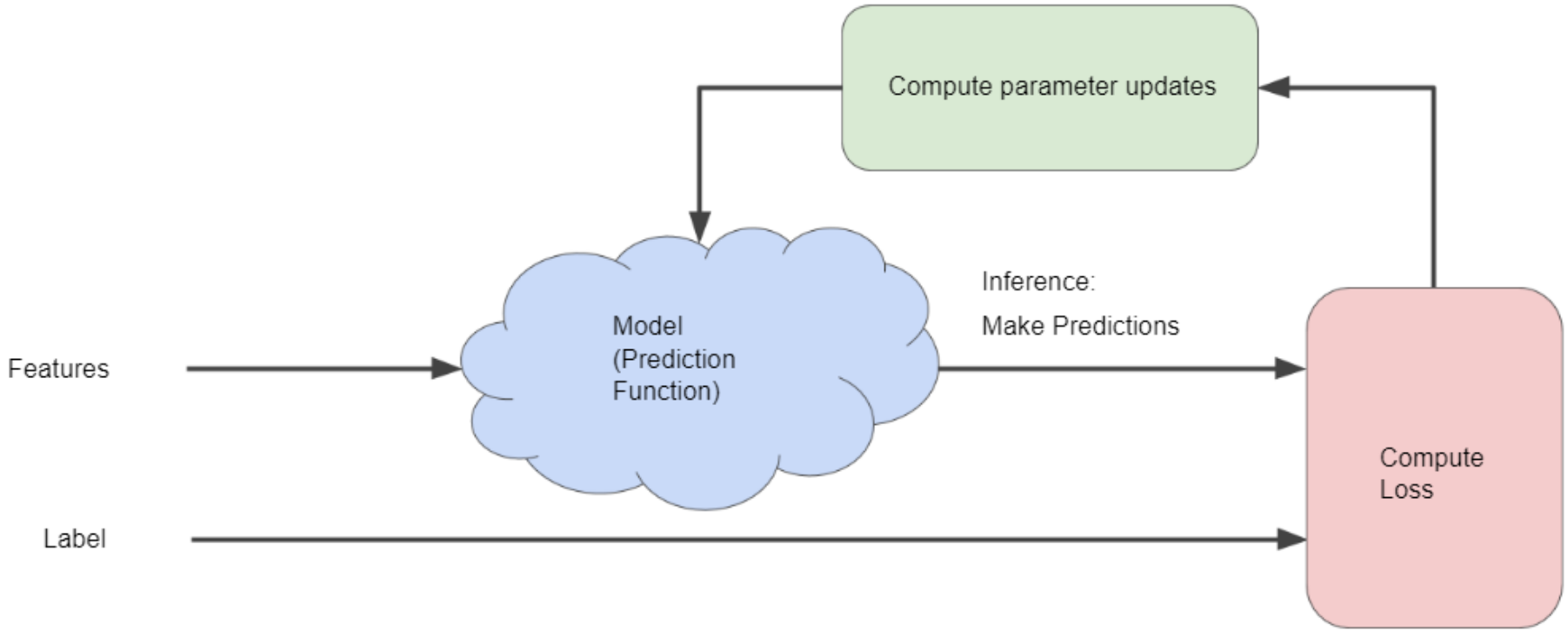
$$J\left(y, y^{/}\right) = f(y - y^{/}).$$

Она используется в тех случаях, где важен знак ошибки, например, при обучении нейронных сетей.

Минимизация потерь: итерационный подход

Чтобы обучить модель, нам нужен хороший способ уменьшить потери модели.
Итеративный подход - это один из широко используемых методов уменьшения потерь, и он так же прост и эффективен, как и спуск с горы.
На следующем рисунке показан итеративный процесс проб и ошибок, который используют алгоритмы машинного обучения для обучения модели:

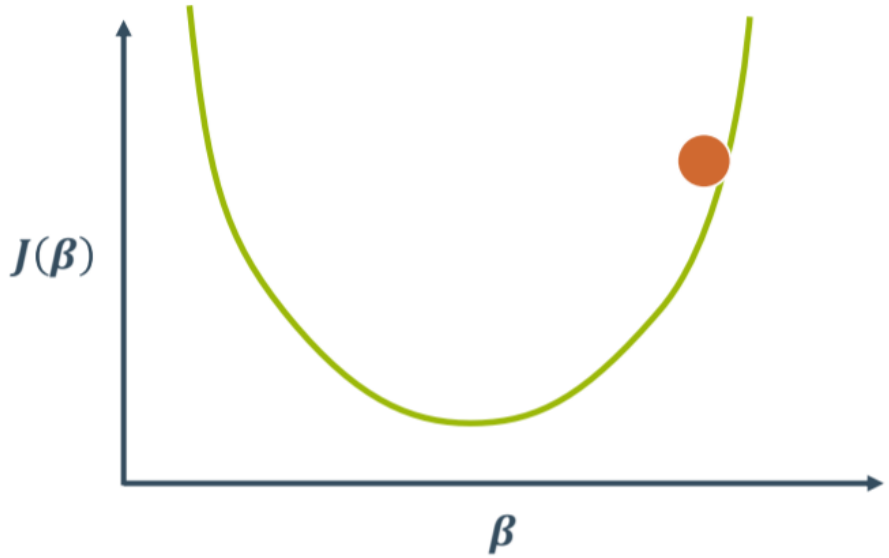
Итеративные стратегии распространены в машинном обучении, в первую очередь потому, что они хорошо масштабируются для больших наборов данных.



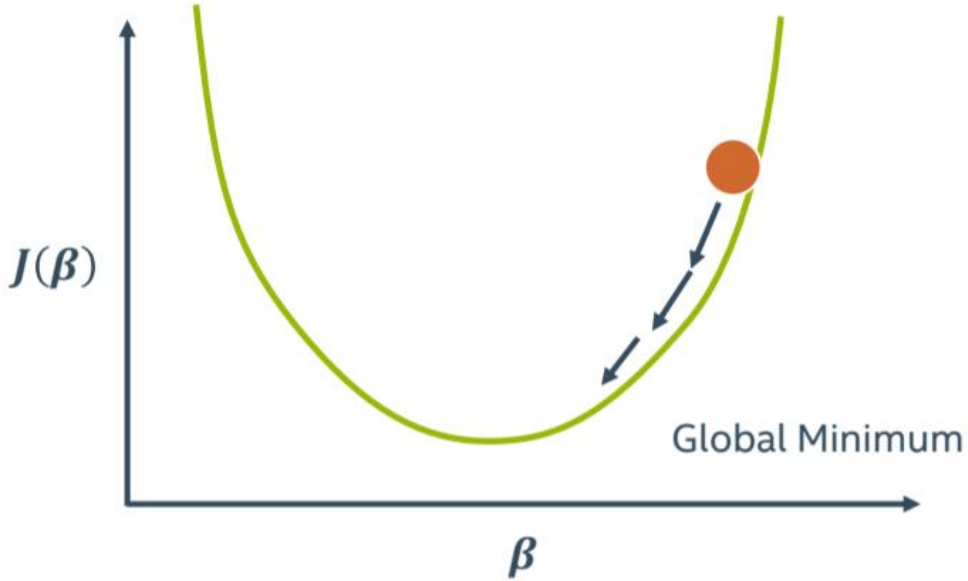
Градиентный спуск

На предыдущем рисунке схема итеративного подхода содержала зеленую волнистую рамку, озаглавленную «Вычислить обновления параметров». Рассмотрим как это происходит.

Предположим, у нас было время и вычислительные ресурсы для расчета потерь для всех возможных значений w 1 . Для задачи регрессии, которую мы рассматриваем, результирующий график потерь всегда будет выпуклым.



Начальное значение функции $J(\beta)$



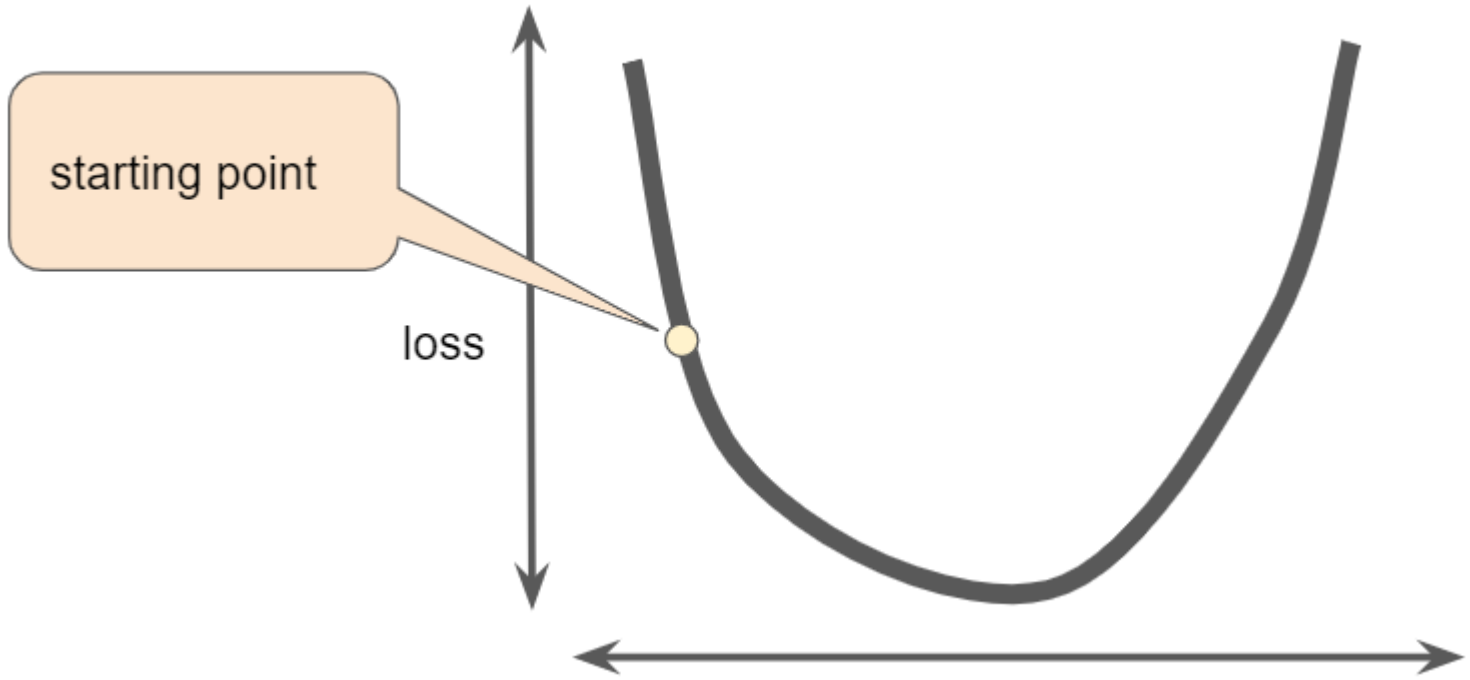
Затем постепенно двигайтесь к минимуму

Градиентный спуск

Выпуклые функции имеют только один минимум; то есть только в одном месте, где наклон равен 0. Этот минимум - то, где функция потерь сходится.

Расчет функции потерь для каждого значения β по всему набору данных был бы неэффективным способом определения точки сходимости. Давайте рассмотрим лучший механизм - очень популярный в машинном обучении - *градиентный спуск*

Исходная точка для градиентного спуска



Первым этапом *градиентного спуска* является выбор начального значения (начальной точки) для β. Отправная точка не имеет большого значения; поэтому многие алгоритмы просто устанавливают β=0 или выбирают случайное значение.

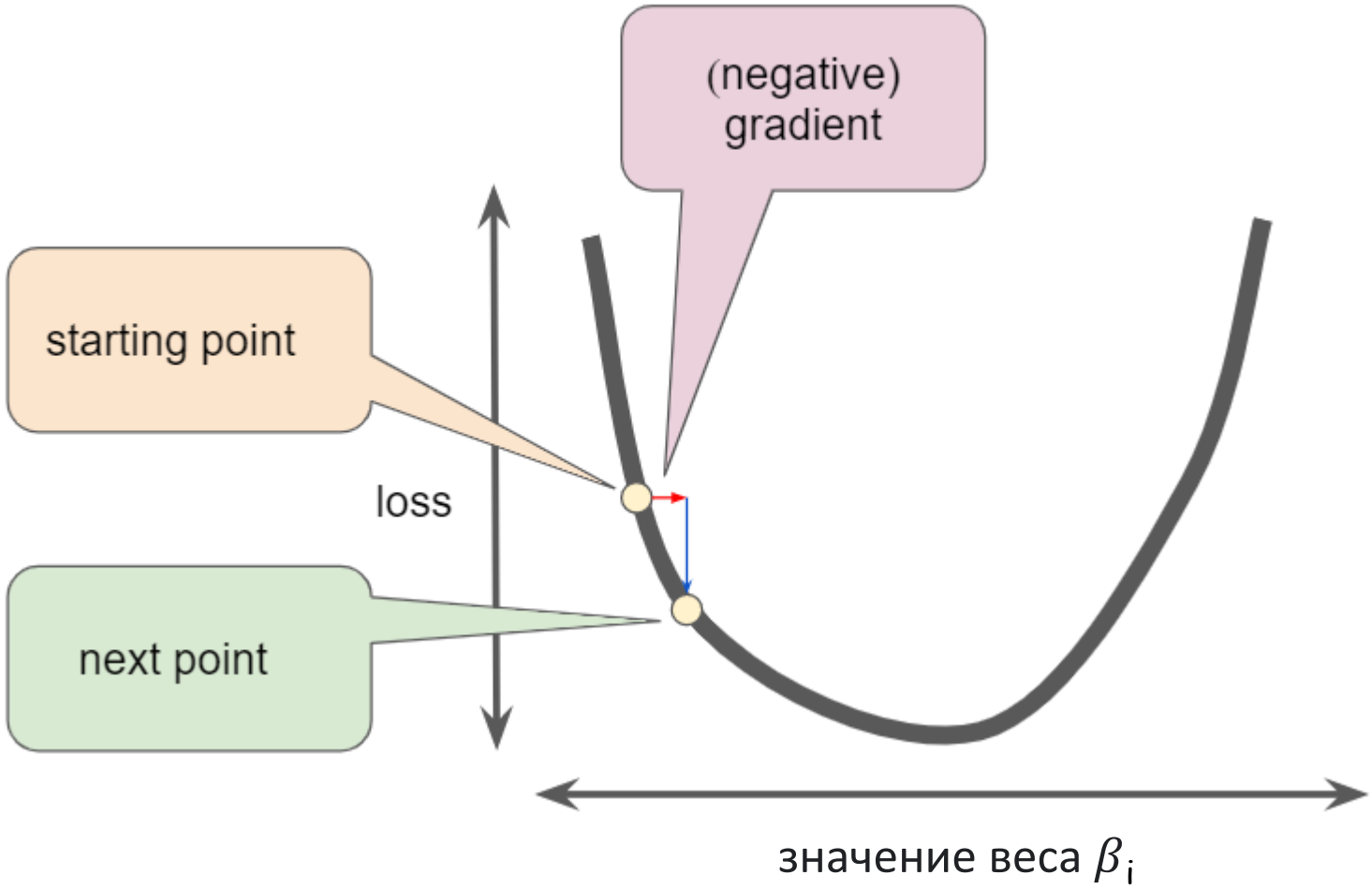
Затем алгоритм градиентного спуска вычисляет градиент функции loss в начальной точке. На рисунке 3, градиент потерь равен **производной** (наклону) кривой и говорит о том, какой путь «теплее» или «холоднее». Когда весов несколько, градиент представляет собой вектор частных производных по весам (параметрам β_i)

Градиентный спуск

Градиент всегда указывает в направлении наискорейшего изменения функции потерь. Алгоритм градиентного спуска делает шаг в направлении отрицательного градиента, чтобы максимально быстро уменьшить потери.

Затем градиентный спуск повторяет этот процесс, приближаясь к минимуму.

Шаг градиента перемещает нас к следующей точке на кривой потерь

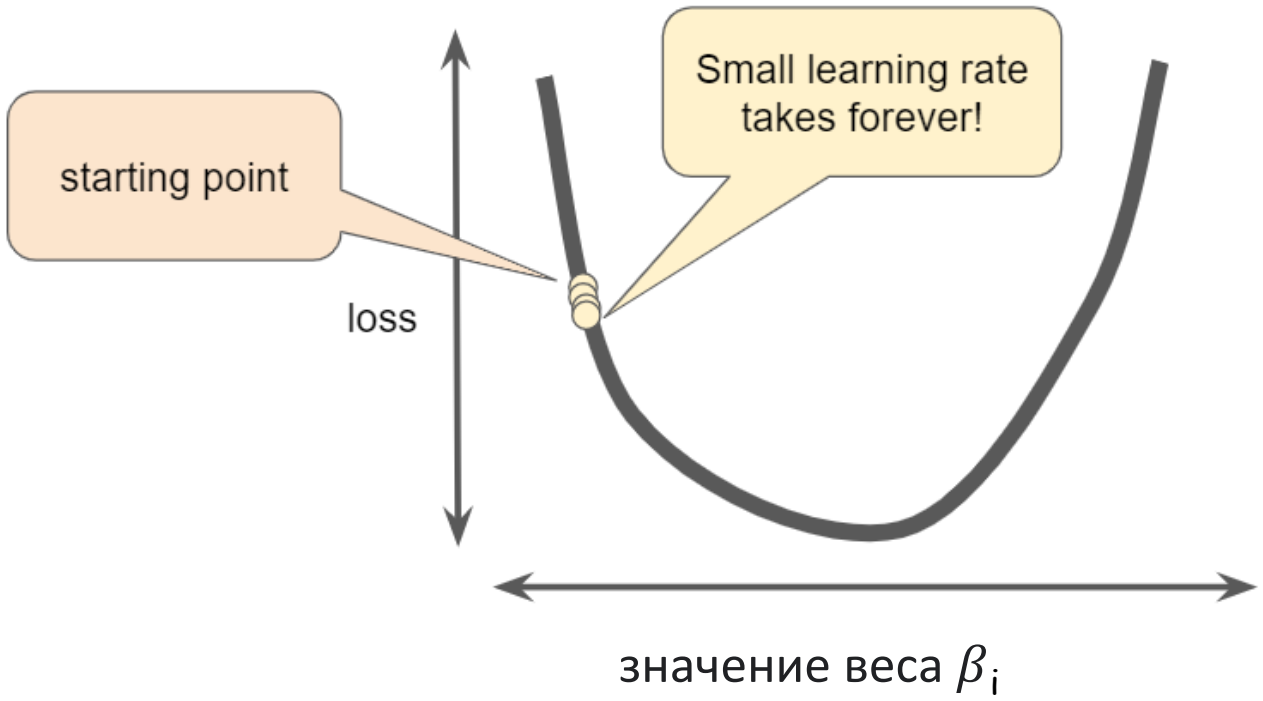


Минимизация потерь: скорость обучения

Как уже отмечалось, вектор градиента имеет как направление, так и величину. Алгоритмы градиентного спуска умножают градиент на скаляр, известный как скорость обучения (также иногда называемый размером шага), чтобы определить следующую точку. Например, если величина градиента равна 2.5, а скорость обучения равна 0.01, то алгоритм снижения градиента выберет следующую точку на 0.025 от предыдущей точки.

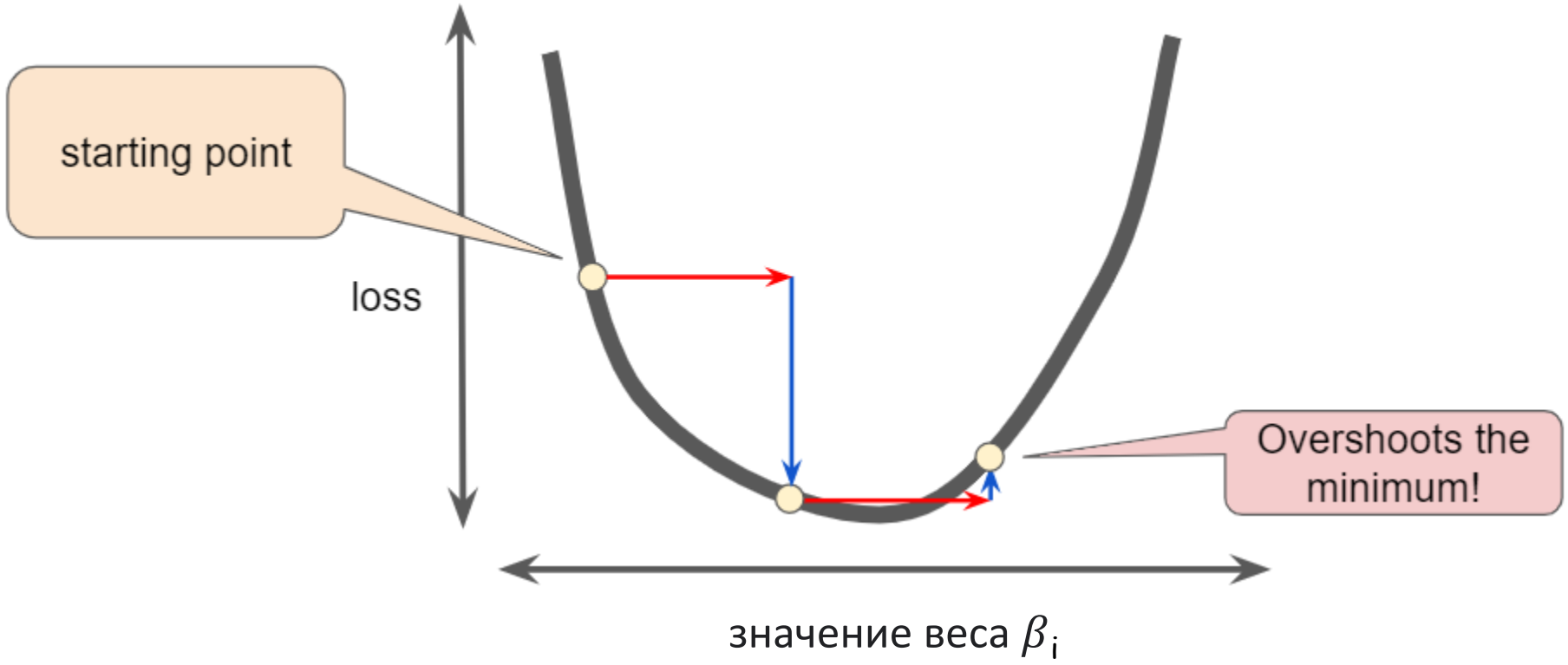
Гиперпараметры - это ручки, которые программисты настраивают в алгоритмах машинного обучения. Большинство программистов машинного обучения тратят немало времени на настройку скорости обучения. Если вы выберете слишком низкую скорость обучения, обучение займет слишком много времени:

Скорость обучения слишком мала



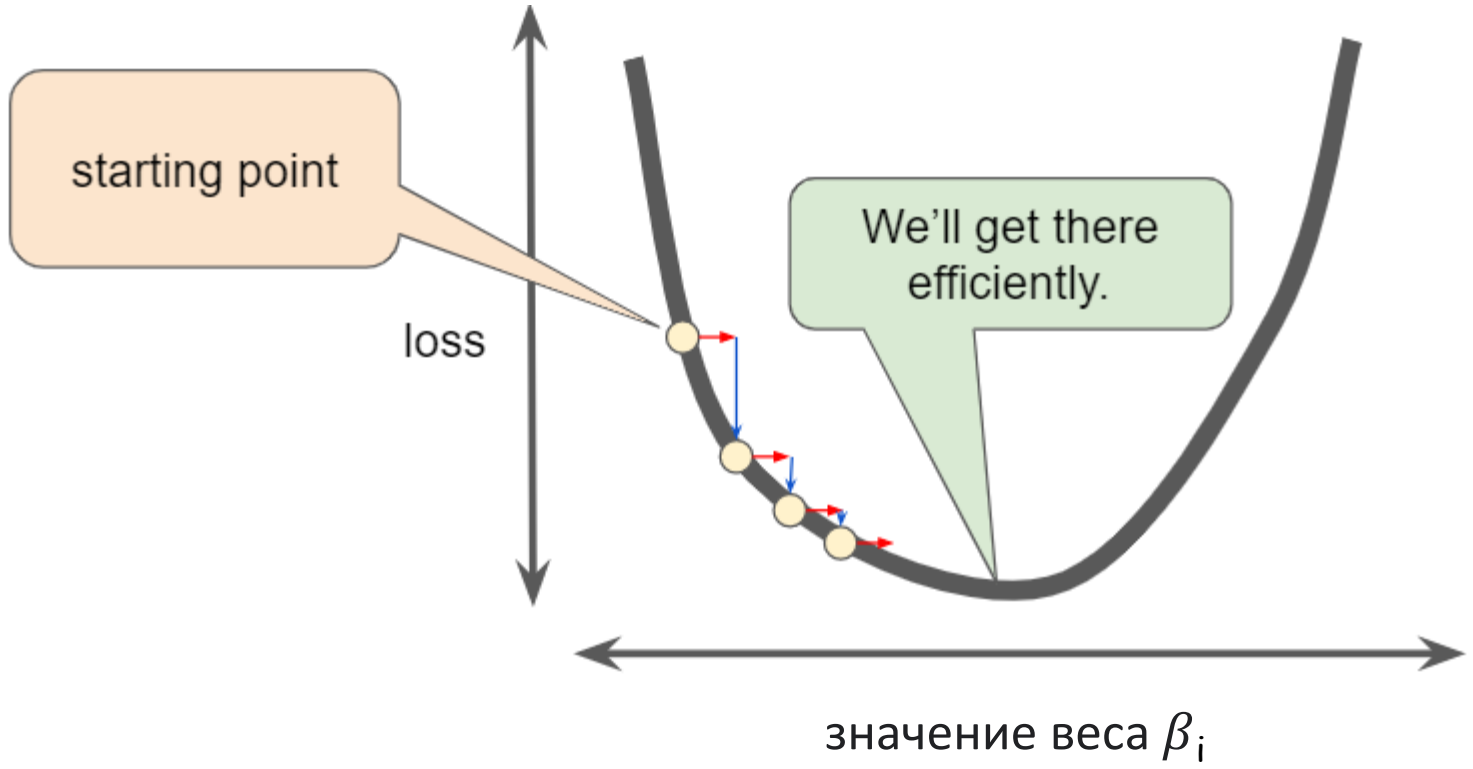
Минимизация потерь: скорость обучения

И наоборот, если вы укажете слишком большую скорость обучения, следующая точка будет беспорядочно подпрыгивать через дно скважины, как эксперимент с квантовой механикой, который оказался ужасно неправильным:



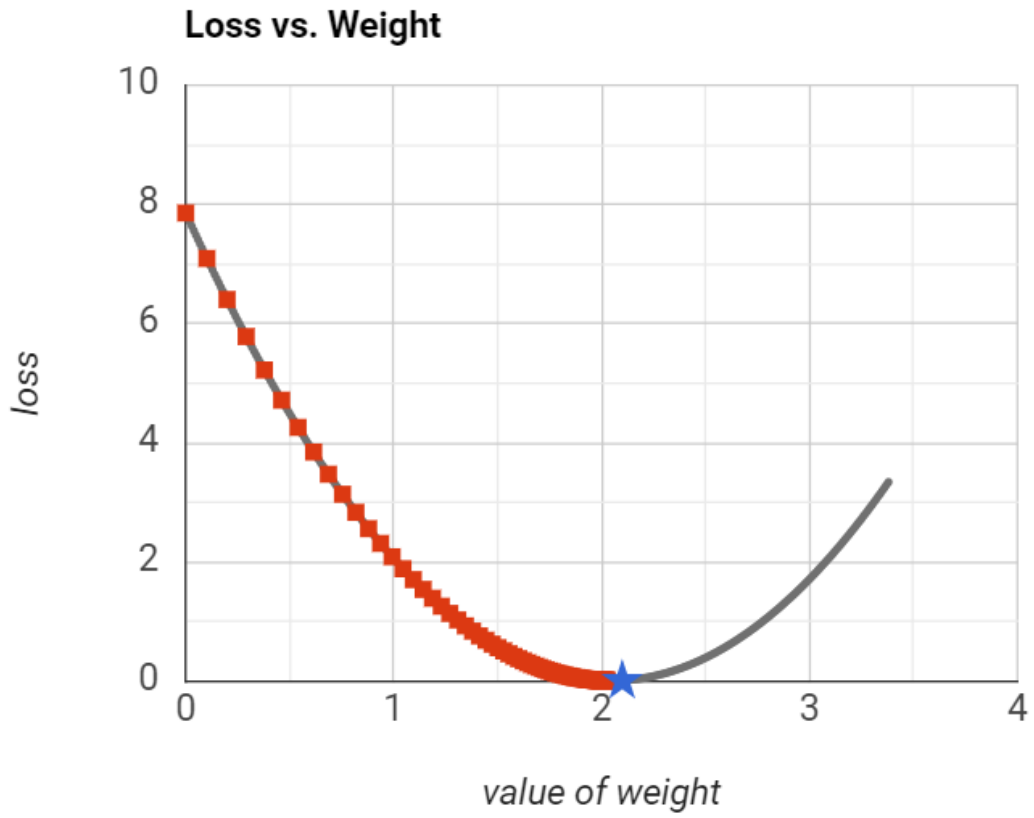
Минимизация потерь: скорость обучения

Для каждой регрессионной задачи существует свой уровень обучения. Его значение связано с тем, насколько плоской является функция потерь. Если вы знаете, что градиент функции потерь мал, тогда вы можете смело попробовать большую скорость обучения, которая компенсирует маленький градиент и приводит к большему размеру шага.

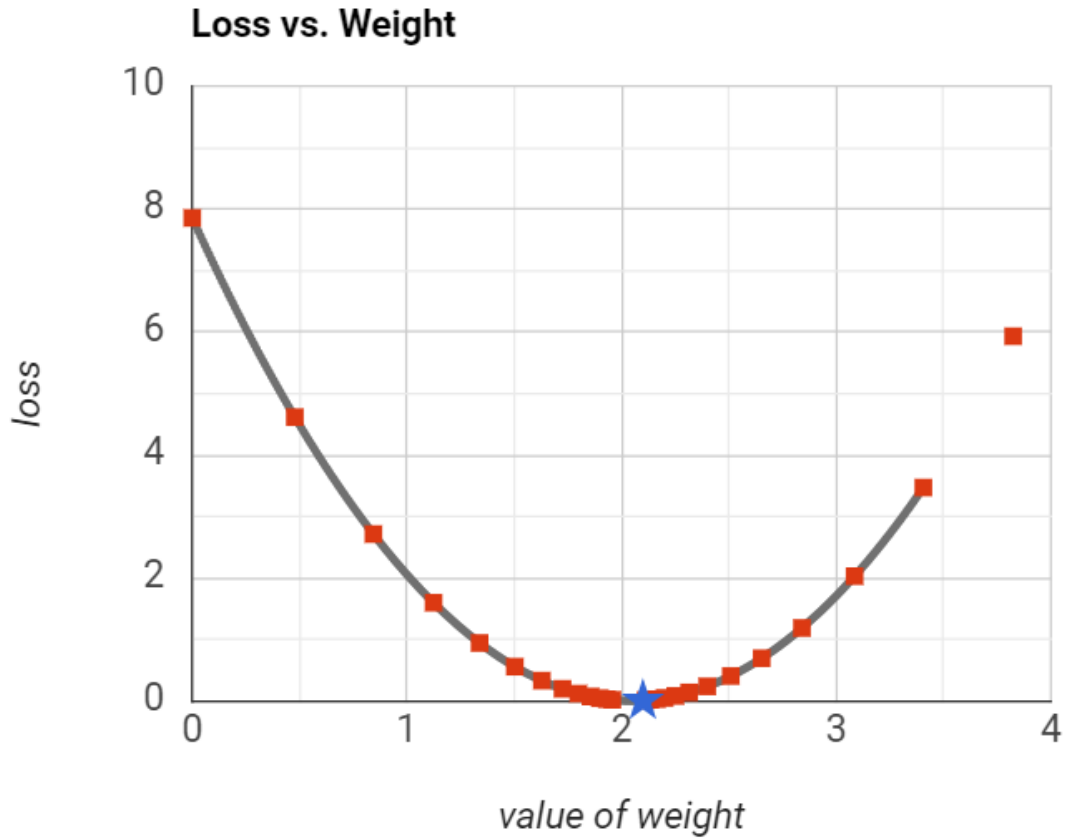


Сокращение потерь: оптимизация скорости обучения

Скорость (уровень) обучения = 0.08
Число шагов = 96

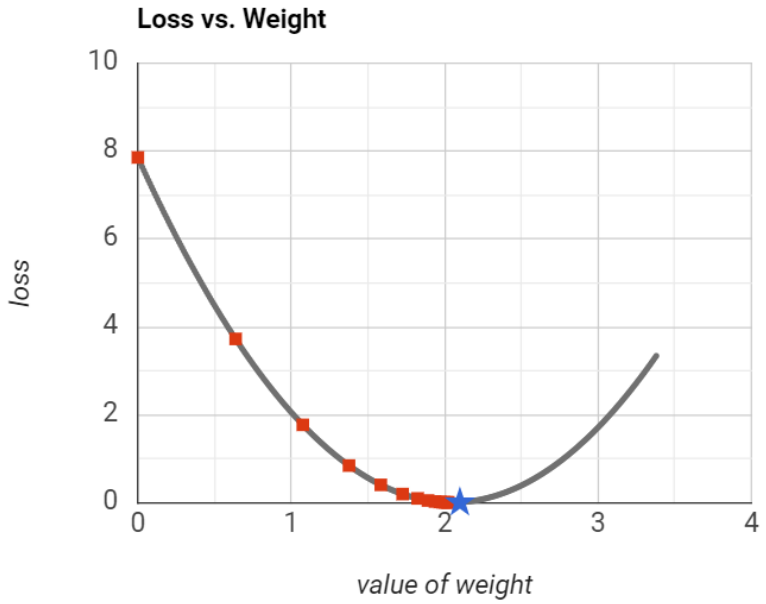


Скорость (уровень) обучения = 3
Число шагов = 25

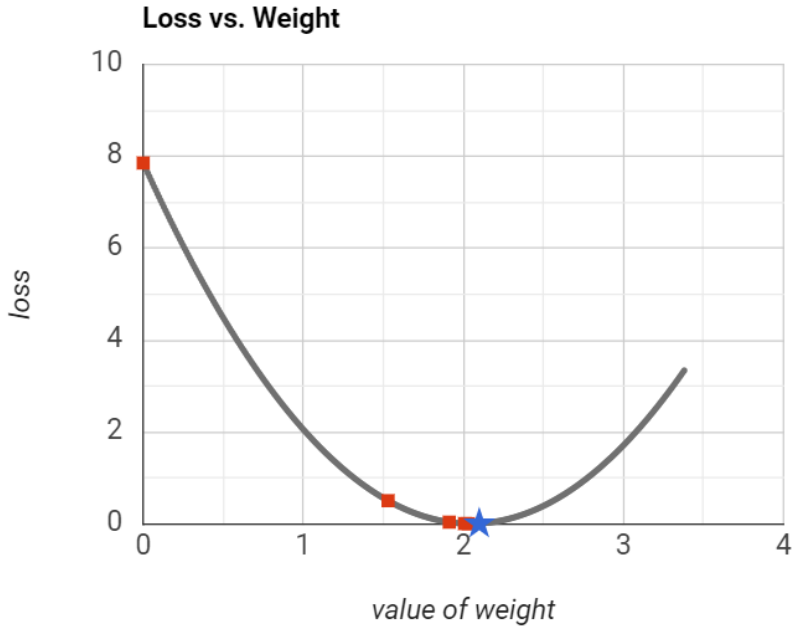


Сокращение потерь: оптимизация скорости обучения

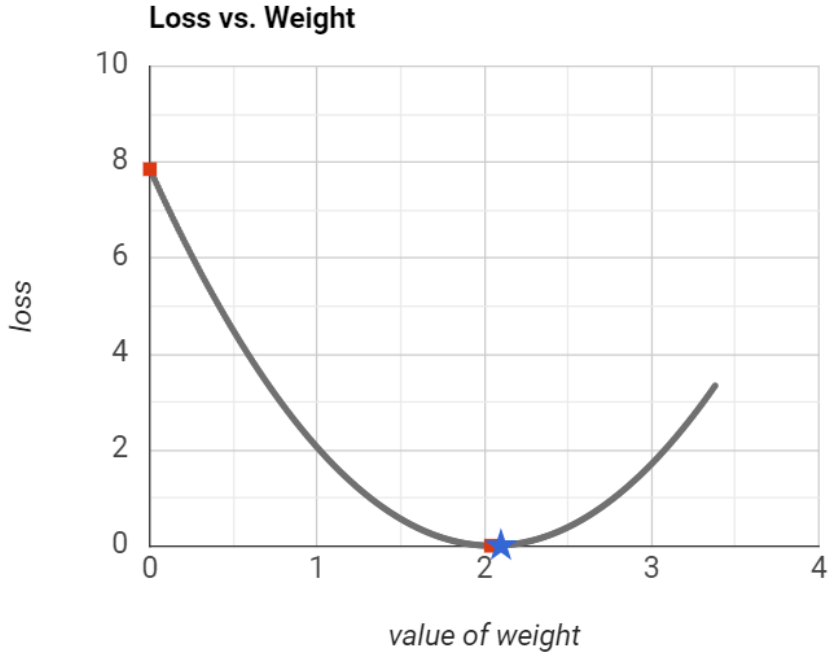
Скорость (уровень) обучения = 1.5
Число шагов = 13



Скорость (уровень) обучения = 3
Число шагов = 4



Скорость (уровень) обучения =1.6
Число шагов = 1



На практике поиск «идеальной» (или почти идеальной) скорости обучения не является существенным для успешного обучения модели. Цель состоит в том, чтобы найти скорость обучения, достаточно большую, чтобы градиентный спуск эффективно сходился, но не настолько большой, чтобы он никогда не сходился.

Сокращение потерь: стохастический градиентный спуск

В градиентном спуске **пакет** - это общее количество примеров, которые вы используете для расчета градиента за одну итерацию. До сих пор мы предполагали, что пакет был весь набор данных. При работе в масштабах Google наборы данных часто содержат миллиарды или даже сотни миллиардов примеров. Кроме того, наборы данных Google часто содержат огромное количество функций. Следовательно, партия может быть огромной. Очень большой пакет может привести к тому, что даже одна итерация займет очень много времени для вычисления.

Большой набор данных со случайно выбранными примерами, вероятно, содержит избыточные данные. Фактически избыточность становится более вероятной с увеличением размера пакета. Некоторая избыточность может быть полезна для сглаживания градиентов шума, но огромные партии, как правило, не имеют гораздо большей прогностической ценности, чем большие партии.

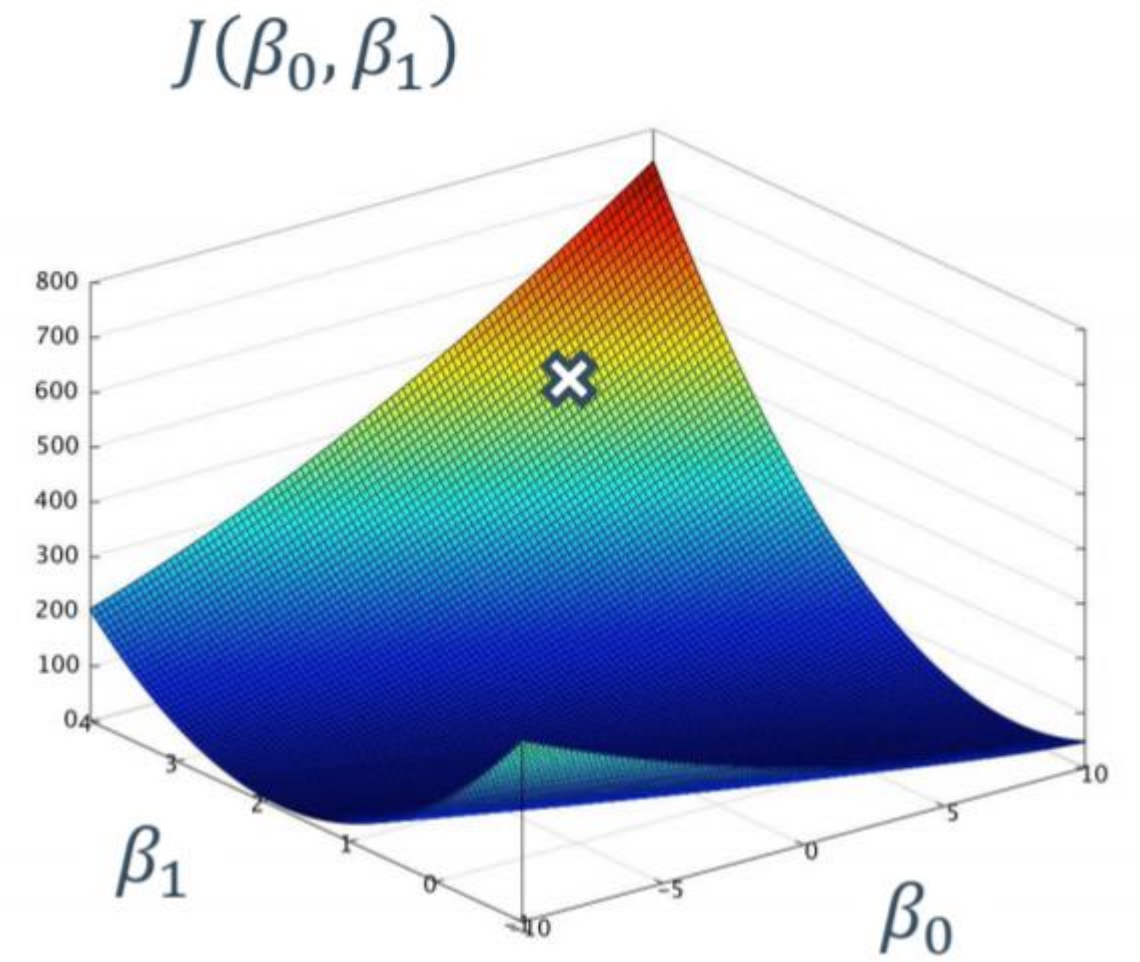
Что если бы мы могли получить правильный градиент в среднем для гораздо меньших вычислений? Выбирая примеры случайным образом из нашего набора данных, мы могли бы оценить (хотя и с шумом) большое среднее значение из гораздо меньшего. **Стохастический градиентный спуск** (**SGD**) доводит эту идею до крайности - он использует только один пример (размер пакета 1) на одну итерацию. При достаточном количестве итераций SGD работает, но очень шумно. Термин «стохастический» указывает, что один пример, содержащий каждую партию, выбирается случайным образом.

Мини-пакетный стохастический градиентный спуск (**mini-batch SGD**) - это компромисс между полной серией итераций и SGD. Мини-партия обычно составляет от 10 до 1000 примеров, выбранных случайным образом. Mini-batch SGD уменьшает количество шума в SGD, но все же более эффективен, чем полный пакетный.

Чтобы упростить объяснение, мы сосредоточились на градиентном спуске для одного объекта. Будьте уверены, что градиентный спуск также работает с наборами признаков, которые содержат несколько признаков (фичей).

Градиентный спуск с линейной регрессией

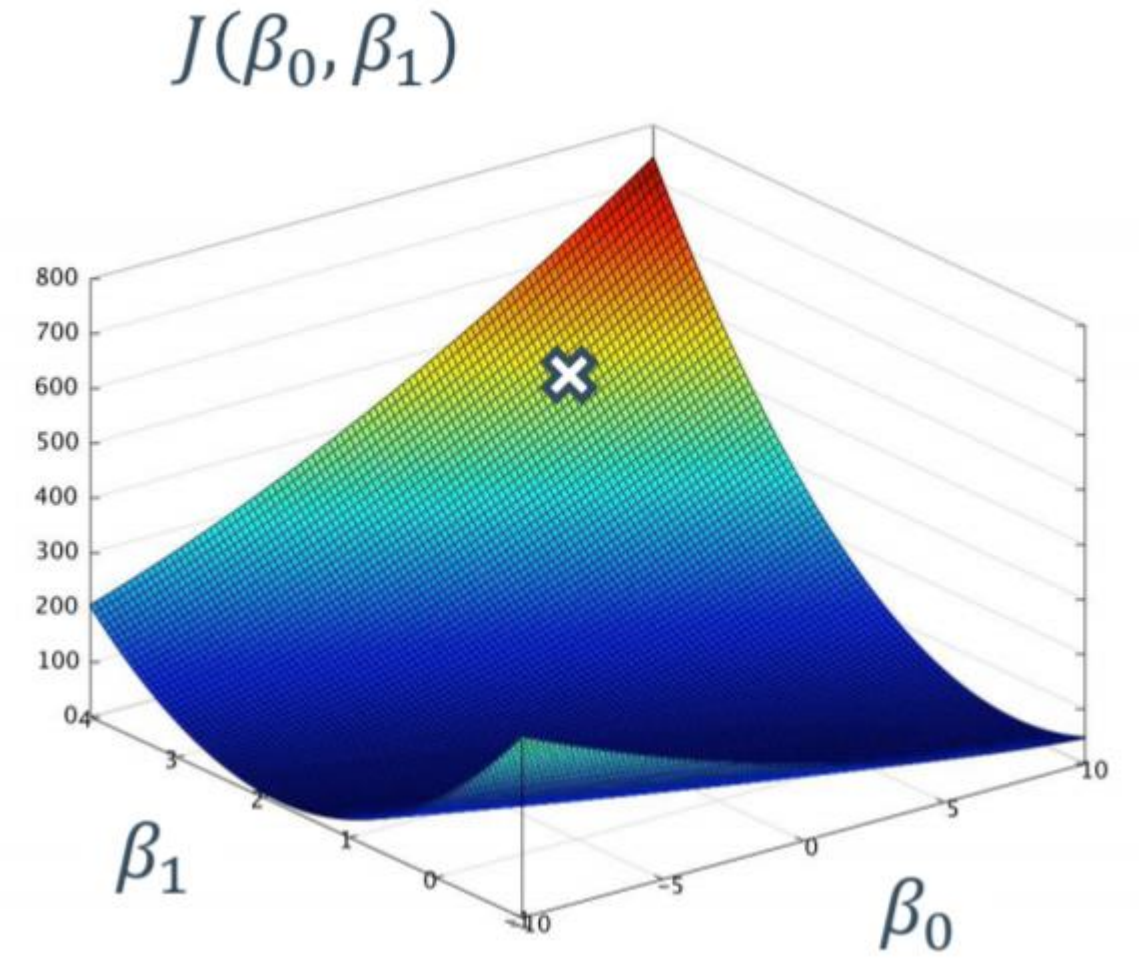
- Теперь представьте, что есть два параметра (β_0, β_1)
- Это более сложная поверхность, на которой должен быть найден минимум
- Как мы можем сделать это, не зная как выглядит $J(\beta_0, \beta_1)$?



Градиентный спуск с линейной регрессией

- Вычислить градиент, $\nabla J(\beta_0, \beta_1)$, который указывает в направлении самой большой увеличение!

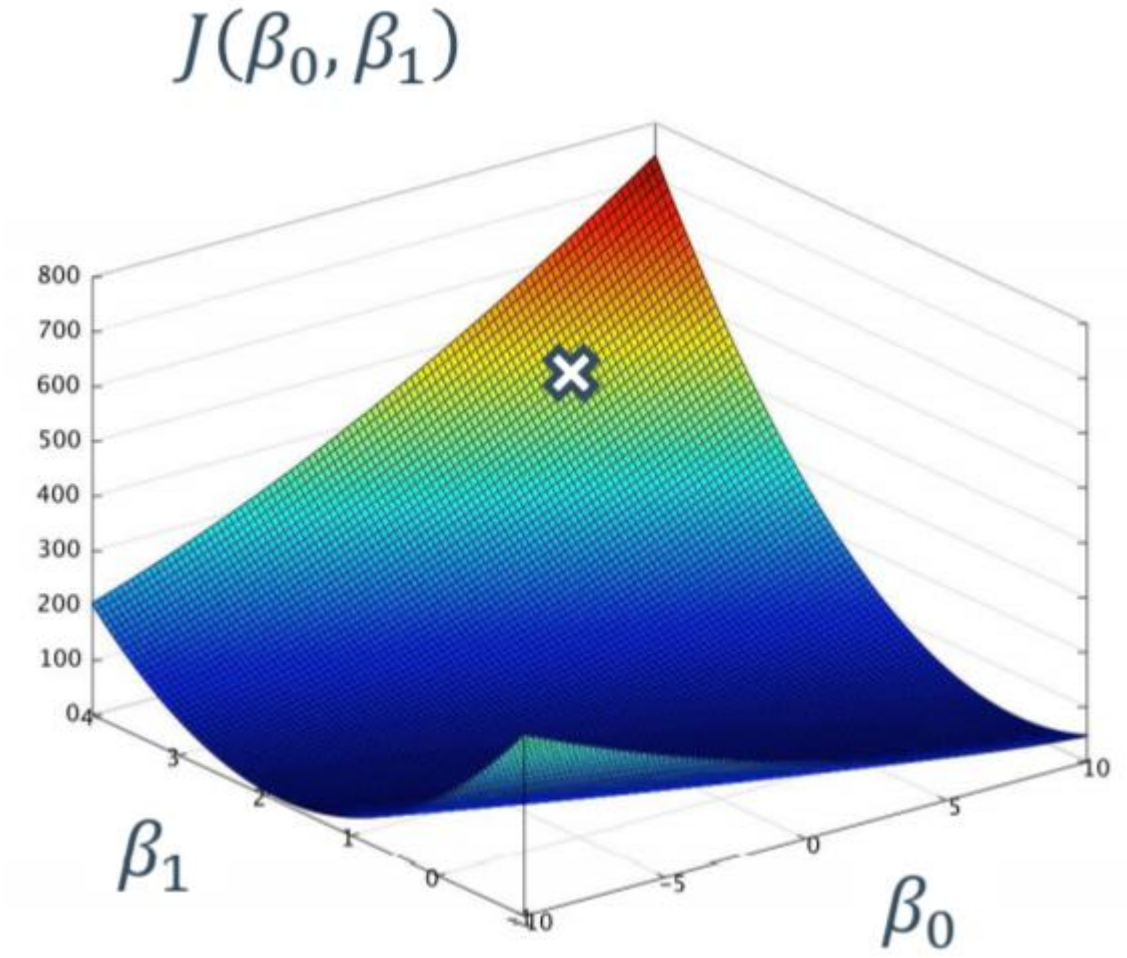
- -$\nabla J(\beta_0, \beta_1)$ (отрицательный градиент) указывает на самое большое снижение в этой точке!



Градиентный спуск с линейной регрессией

- Градиент - это вектор, координаты которого состоят из частных производных параметров

$$\nabla J(\beta_0, \dots, \beta_n) = < \frac{\partial J}{\partial \beta_0}, \dots, \frac{\partial J}{\partial \beta_n} >$$

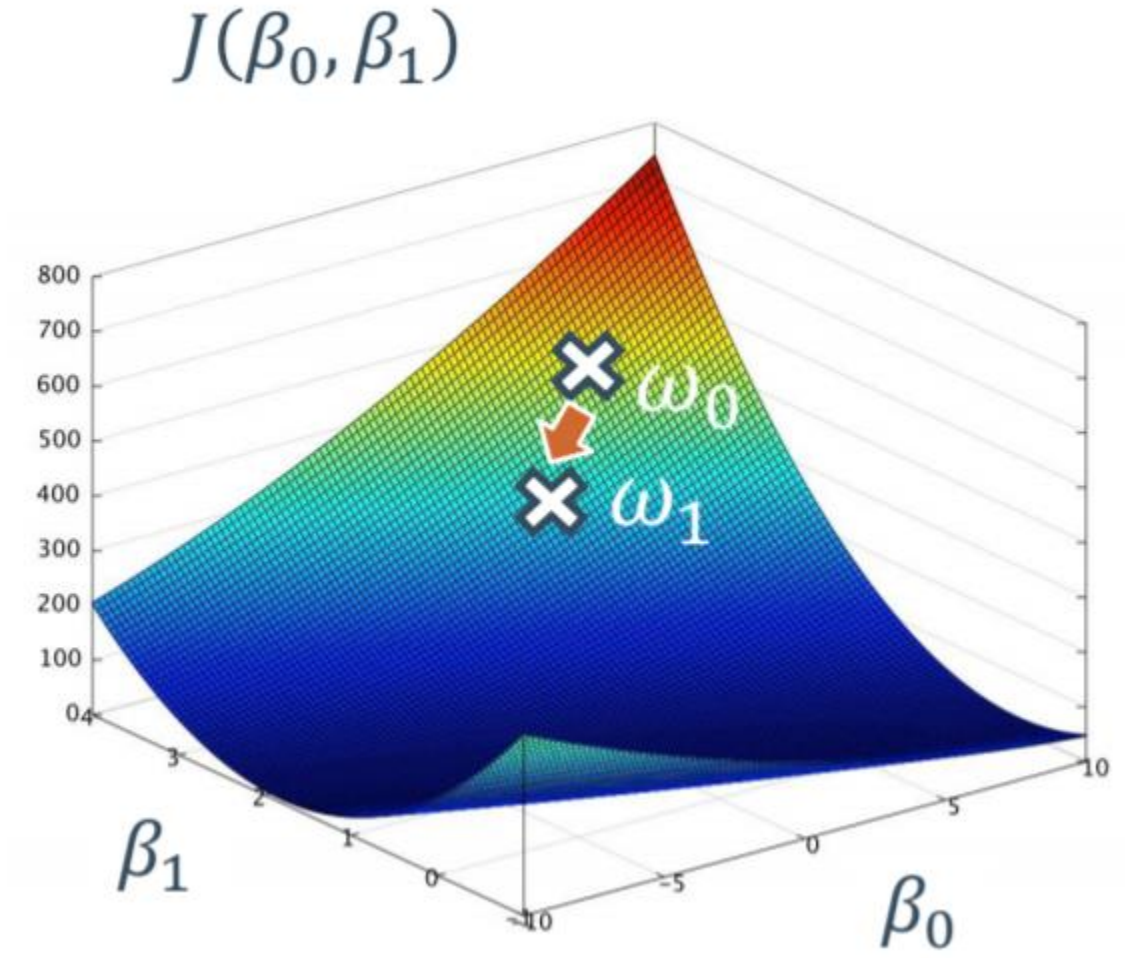


Градиентный спуск с линейной регрессией

- Затем используйте градиент ∇ функции потерь для расчета следующей точки w_1 от текущего w_0:

$$w_1 = w_0 - \lambda\nabla\frac{1}{2n}\sum_{\mathrm{i}=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2$$

- Скорость обучения α - это настраиваемый параметр, который определяет размер шага

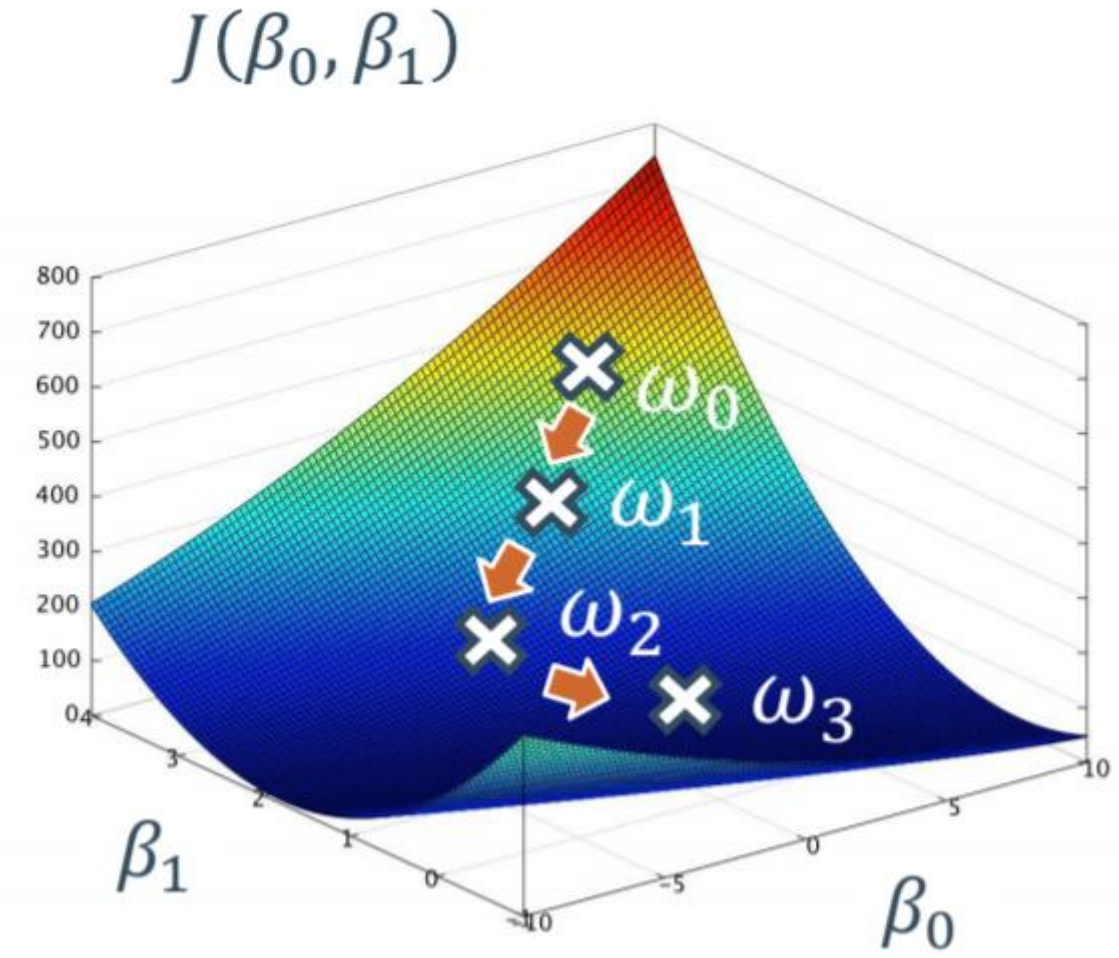


Градиентный спуск с линейной регрессией

- Каждая точка может быть итеративно рассчитана из предыдущей

$$w_2 = w_1 - \lambda\nabla\frac{1}{2n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2\,,$$

$$w_3 = w_2 - \lambda\nabla\frac{1}{2n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2$$

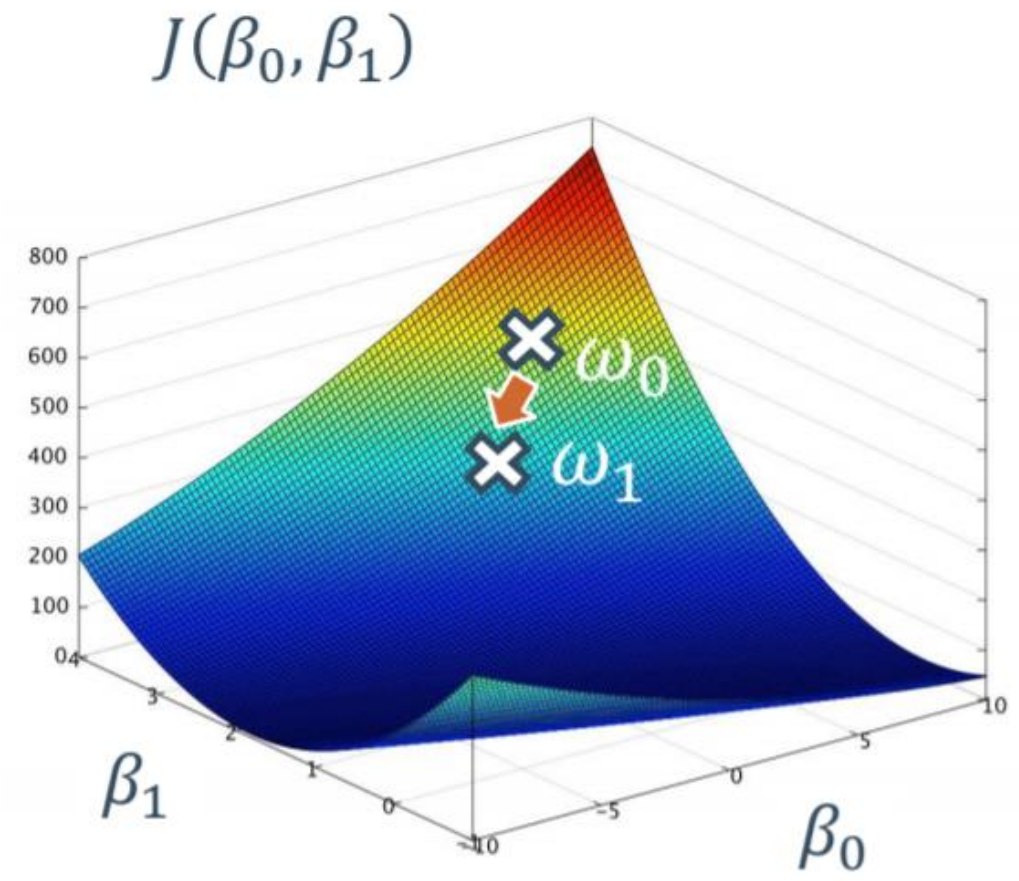


Стохастический градиентный спуск

- Используйте одну точку данных для определения функции градиента и стоимости вместо всех данных

$$w_2 = w_1 - \lambda\nabla\frac{1}{2n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2,$$

$$w_3 = w_2 - \lambda\nabla\frac{1}{2}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2$$

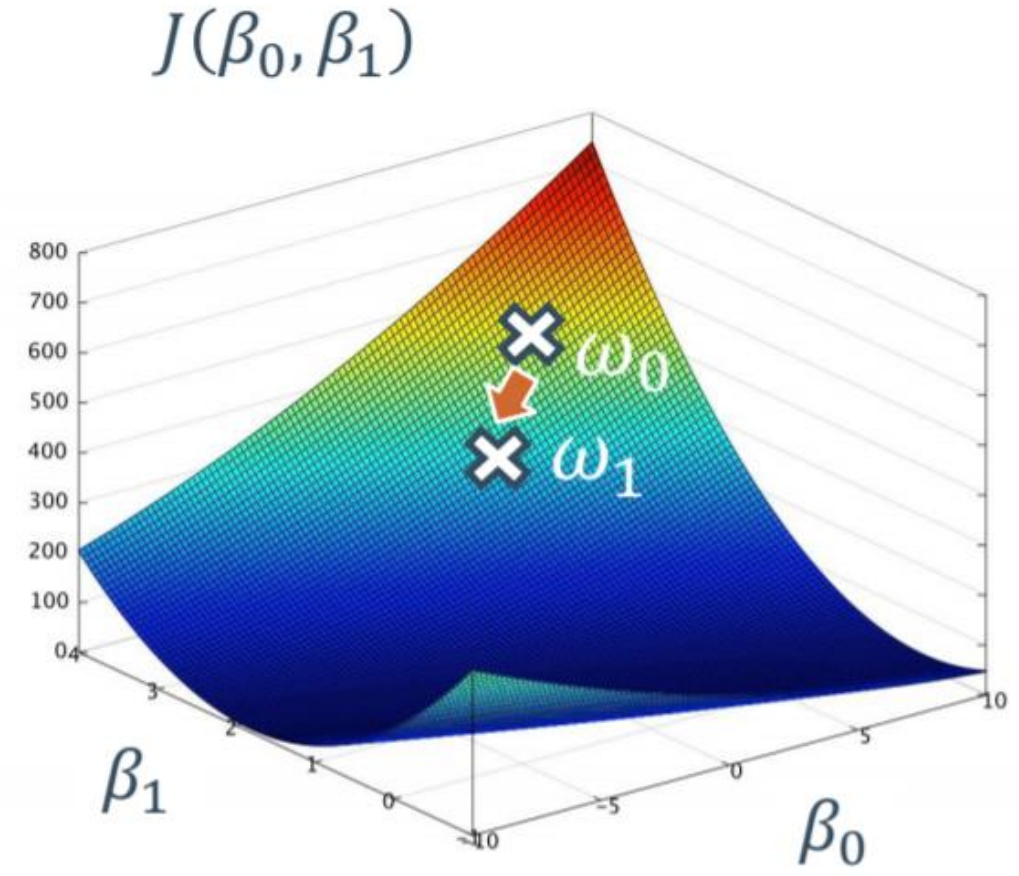


Стохастический градиентный спуск

- Используйте одну точку данных для определения функции градиента и стоимости вместо всех данных

$$w_2 = w_1 - \lambda\nabla\frac{1}{2n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2,$$

$$w_3 = w_2 - \lambda\nabla\frac{1}{2}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2$$



Стохастический градиентный спуск

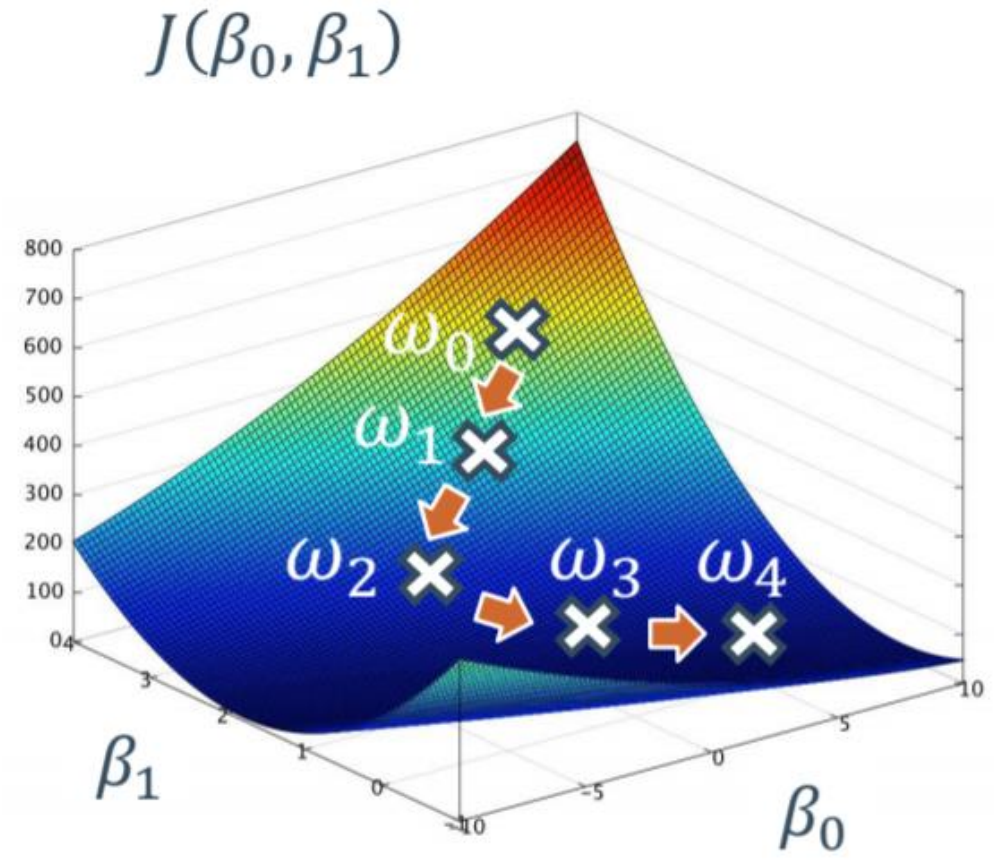
- Используйте одну точку данных для определения функции градиента и стоимости вместо всех данных

$$w_1 = w_0 - \lambda\nabla\frac{1}{2}\left((\beta_0 + \beta_1 x^{(0)}) - y^{(0)}\right)^2,$$

…

$$w_4 = w_3 - \lambda\nabla\frac{1}{2}\left((\beta_0 + \beta_1 x^{(3)}) - y^{(3)}\right)^2$$

- Путь к минимуму менее прямой из-за шума в одной точке данных - «стохастичный»



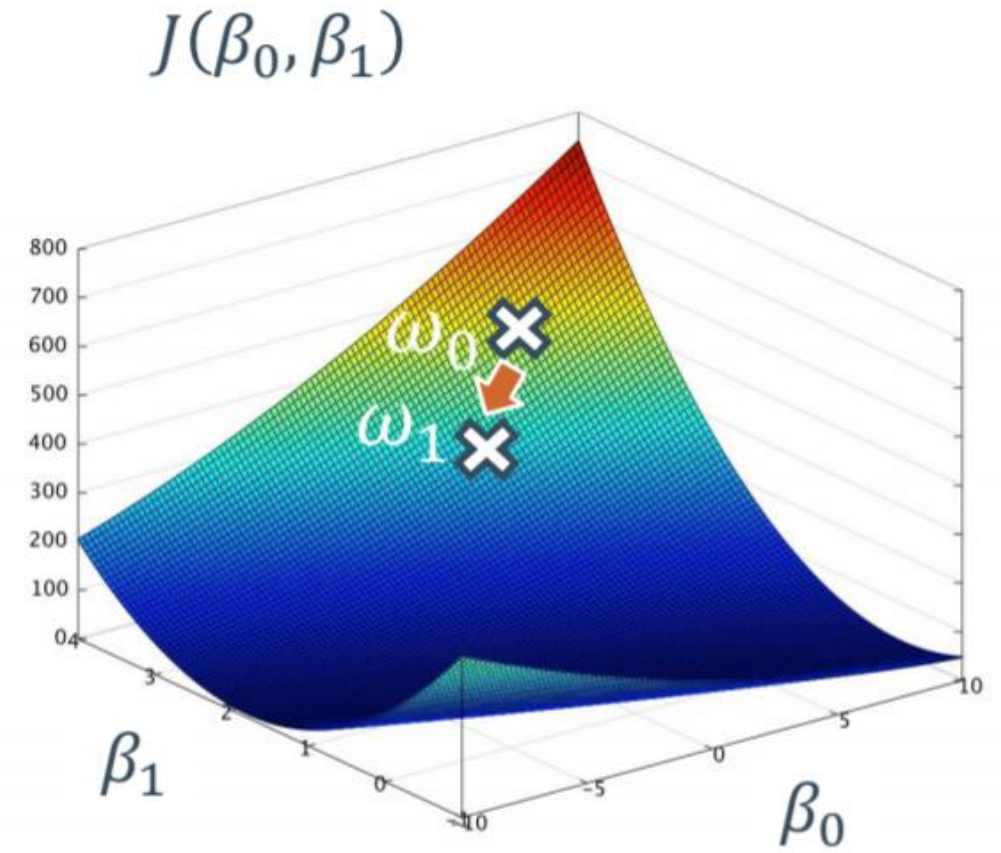
Мини пакетный градиентный спуск

- Выполните обновление для каждого m обучающих примеров

$$w_1 = w_0 - \lambda\nabla\frac{1}{2m}\sum_{i=1}^{m}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2$$

Лучшее обоих миров:

- Снижение объема вычислений относительно классического градиентного спуска
- Менее шумный, чем стохастический градиентный спуск
- Мини-пакетная реализация обычно используемая для *нейронных сетей*
- Размеры партии варьируются от 50–256 точек
- Компромисс между размером партии и скоростью обучения α
- Индивидуальный график обучения: постепенно снижайте скорость обучения в течение определенной эпохи



Градиентный спуск

- Функция потерь

$$J(\beta_0, \beta_1) = \frac{1}{2n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)})^2,$$

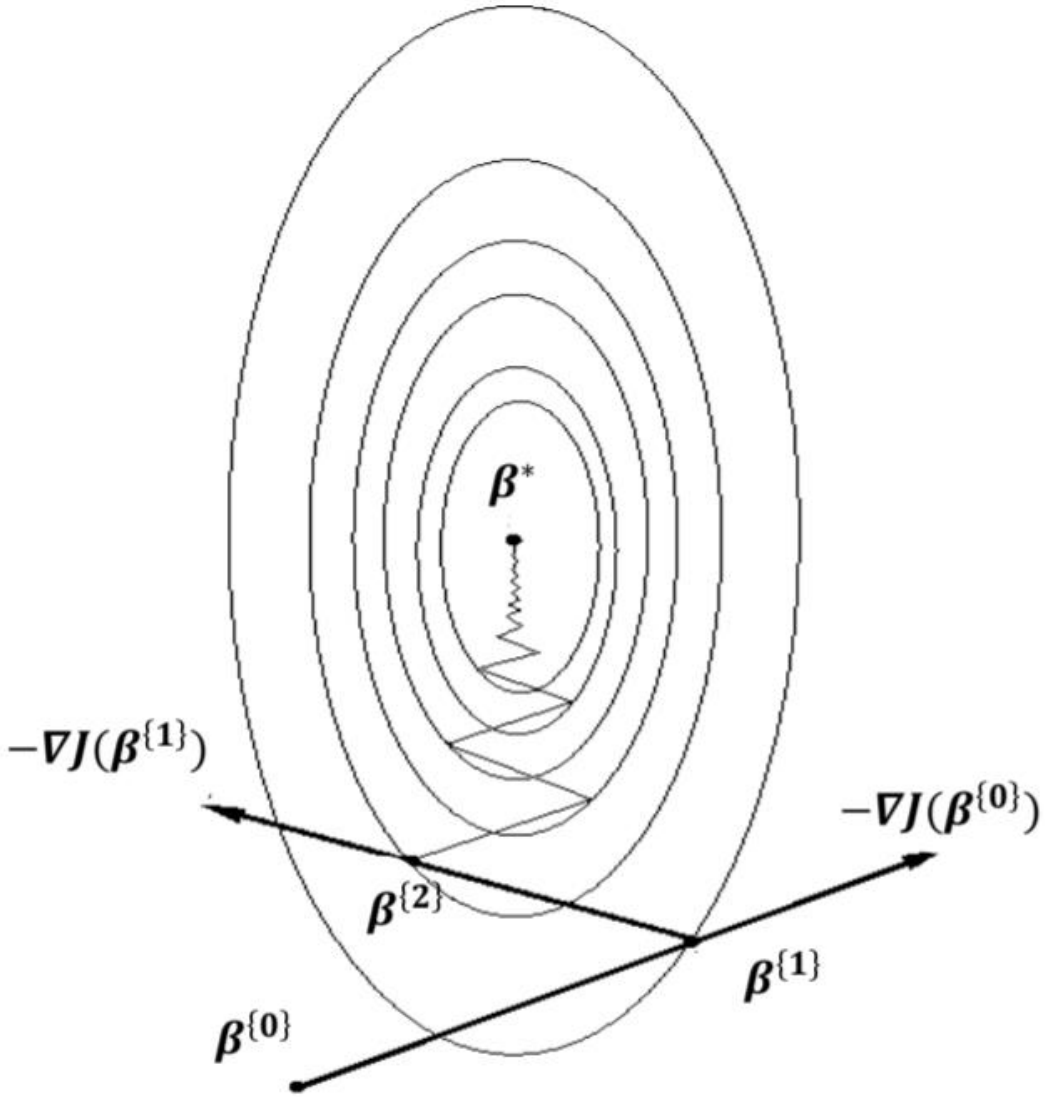
- Метод градиентного спуска

$$\beta_j^{\{k+1\}} = \beta_j^{\{k\}} - \lambda\frac{\partial}{\partial\beta_j}J(\beta_0, \beta_1)$$

- Результат

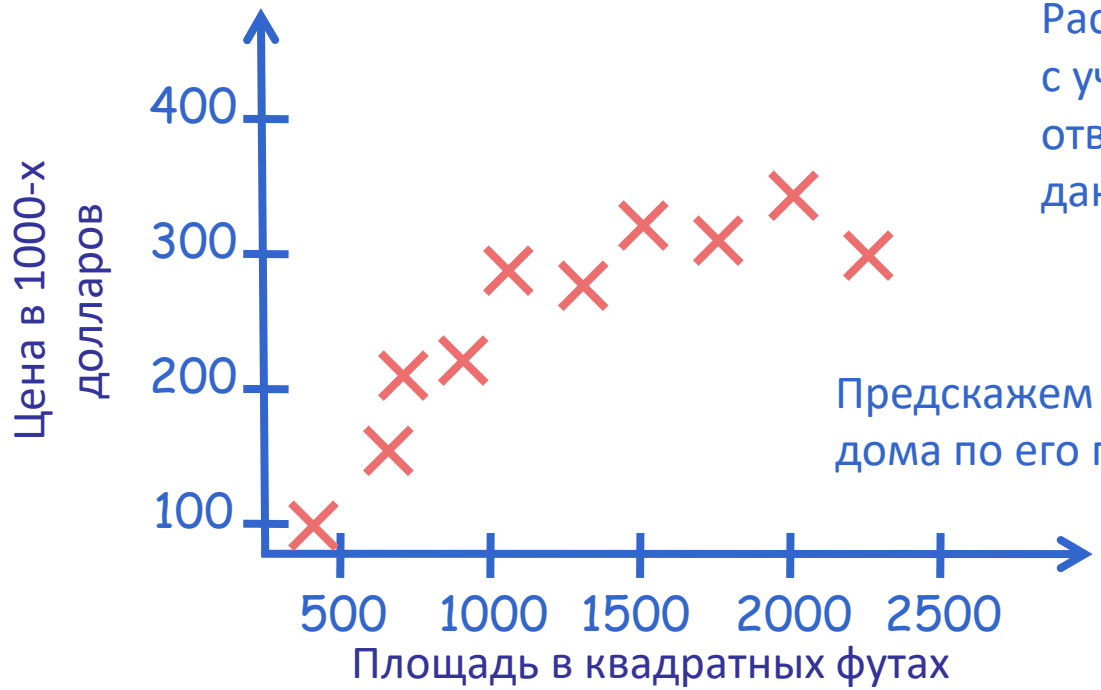
$$\beta_0^{\{k+1\}} = \beta_0^{\{k\}} - \lambda\frac{1}{n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)}))$$

$$\beta_1^{\{k+1\}} = \beta_1^{\{k\}} - \lambda\frac{1}{n}\sum_{i=1}^{n}((\beta_0 + \beta_1 x^{(i)}) - y^{(i)}))x^{(i)}$$



Линейная регрессия с одной переменной

Регрессия (предсказание непрерывной выходной величины, например, цены на недвижимость)



Рассматриваем обучение с учителем (даны правильные ответы для каждого примера в данных)

Предскажем стоимость дома по его площади

Линейная регрессия с одной переменной

Тренировочное множество данных (скажем, всего m)

Площадь (фут^2) – x	**Цена в 1000-х ($) – y**
2104	460
1416	232
1534	315
852	178
…	…

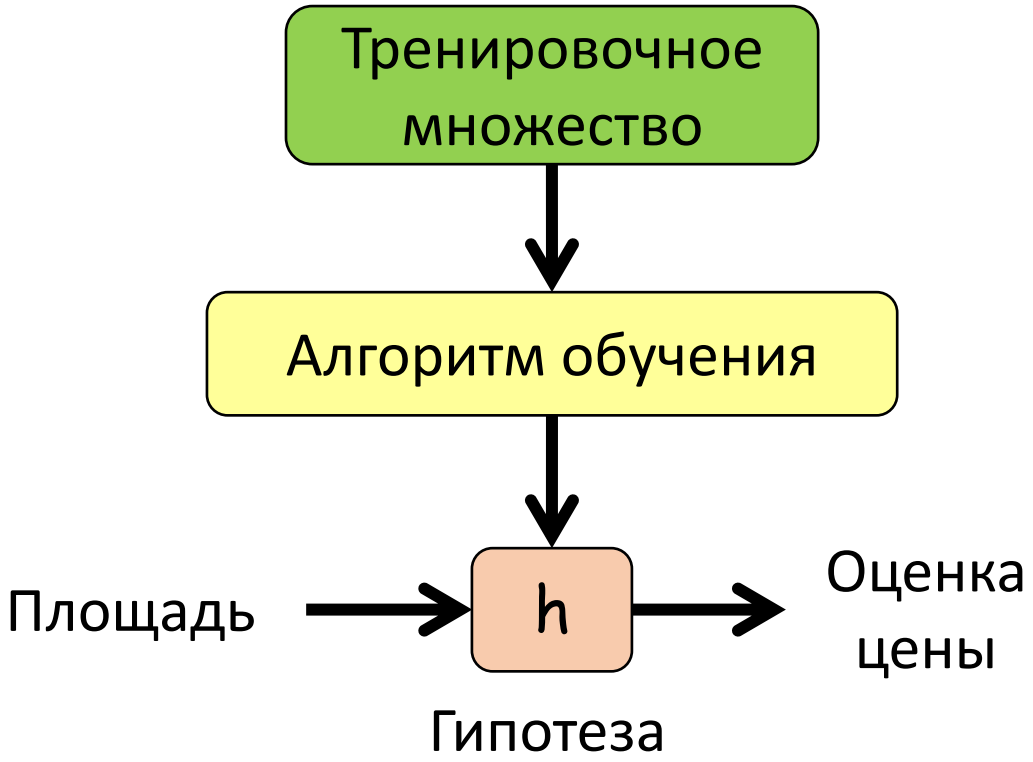
Обозначения: **n** = число тренировочных примеров

x = «входная» переменная / свойство

y = «выходная» переменная / «метка»

($\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$) = i-й тренировочный пример

Линейная регрессия с одной переменной

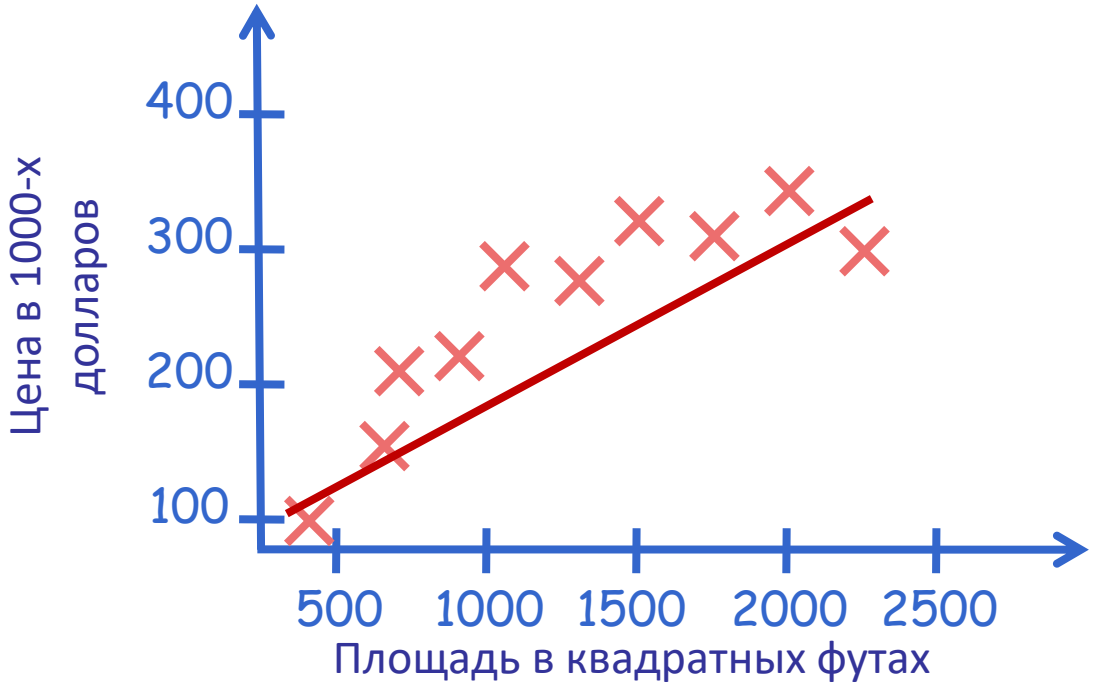


Гипотеза h выглядит так:
$h(x) = \beta_0 + \beta_1 x$

Как оценить β_0 и β_1 ?

Линейная регрессия с одной переменной

- ✓ Последовательность действий

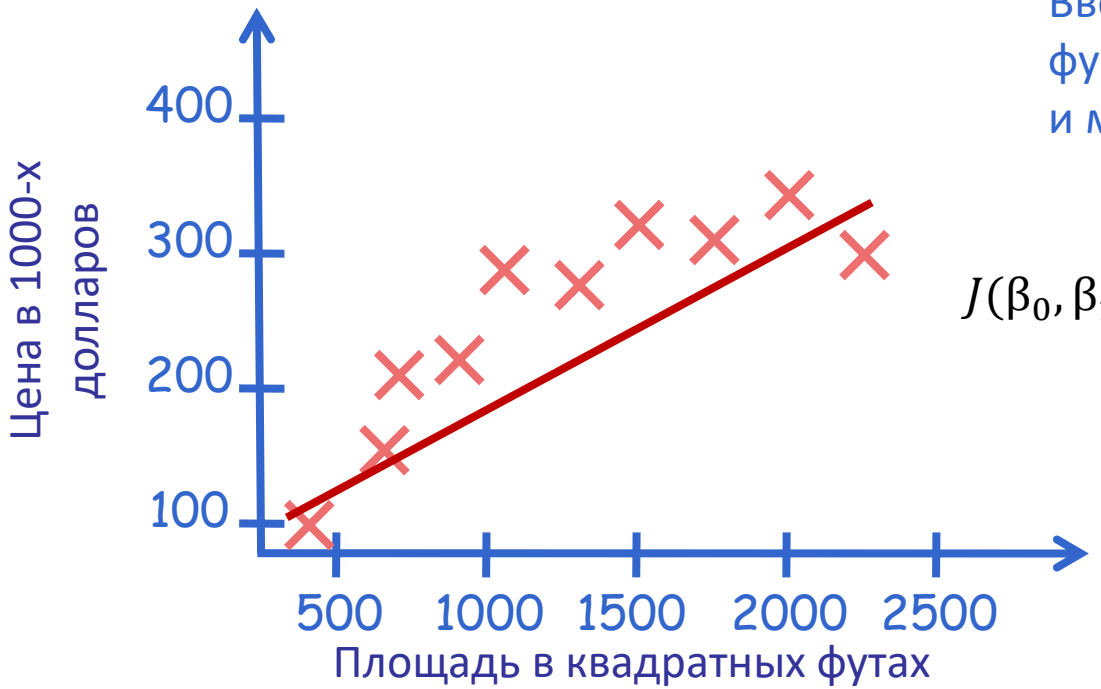


Гипотеза h выглядит так:
$h(x) = \beta_0 + \beta_1 x$

Как оценить β_0 и β_1 ?

Стоимостная функция (Cost Function)

Идея: выбрать β_0 и β_1 так, чтобы h(x) являлась близкой к значениям y для всех тренировочных примеров



Введем стоимостную функцию $J(\beta_0, \beta_1)$ и минимизируем ее:

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)})^2 ,$$

$$\min_{\beta_0, \beta_1} J(\beta_0, \beta_1)$$

Как минимизировать стоимостную функцию для линейной регрессии с одной переменной

- ✓ Использование методов численной оптимизации, например, метода градиентного спуска
  - ✓ Более конкретно будем рассматривать групповой градиентный спуск («Batch» Gradient Descent). Групповой означает, что на каждом этапе градиентного спуска используются все тренировочные примеры
  - ✓ Подход может быть использован и для других методов машинного обучения, например, нейронных сетей

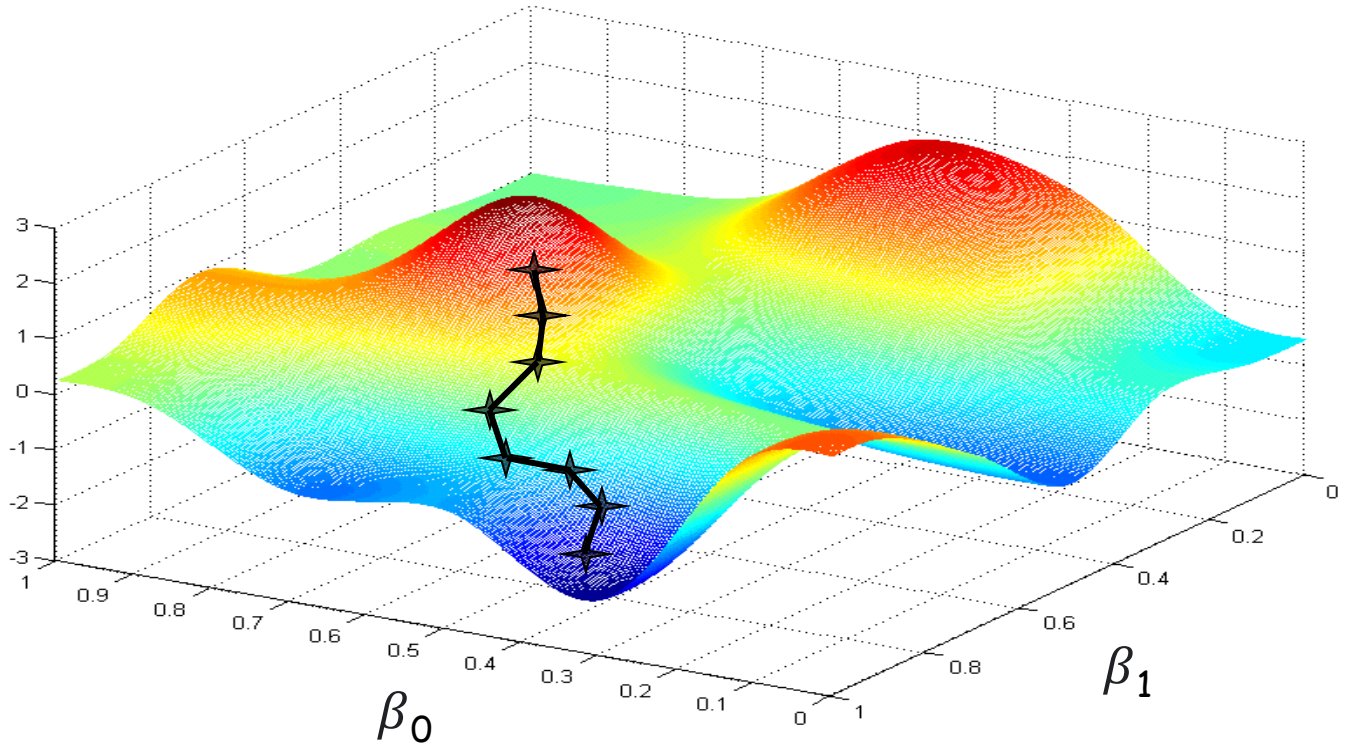
Метод градиентного спуска

- ✓ Постановка задачи
  - ✓ Имеется некоторая стоимостная функция $J(\beta_0, \beta_1)$
  - ✓ Необходимо найти такие значения β_0, β_1, чтобы функция $J(\beta_0, \beta_1)$ стала минимальной

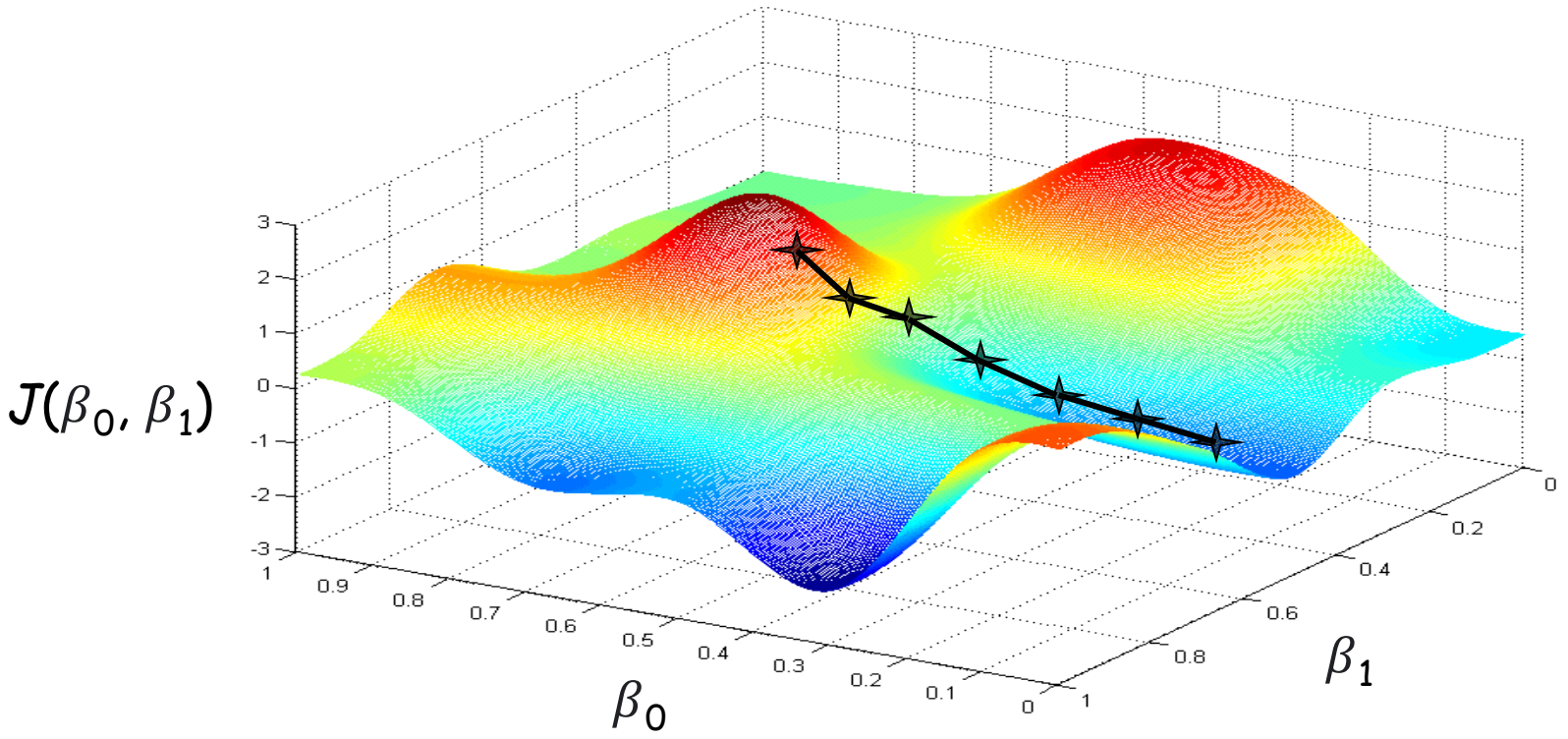
- ✓ Решение задачи
  - ✓ Стартуем из некоторых значений β_0, β_1, например, равных величине ноль
  - ✓ Продолжаем изменение значений β_0, β_1 до тех пор, пока не достигнем минимума. Минимум достижим не всегда!

Пример работы градиентного спуска

$J(\beta_0, \beta_1)$



Пример работы градиентного спуска

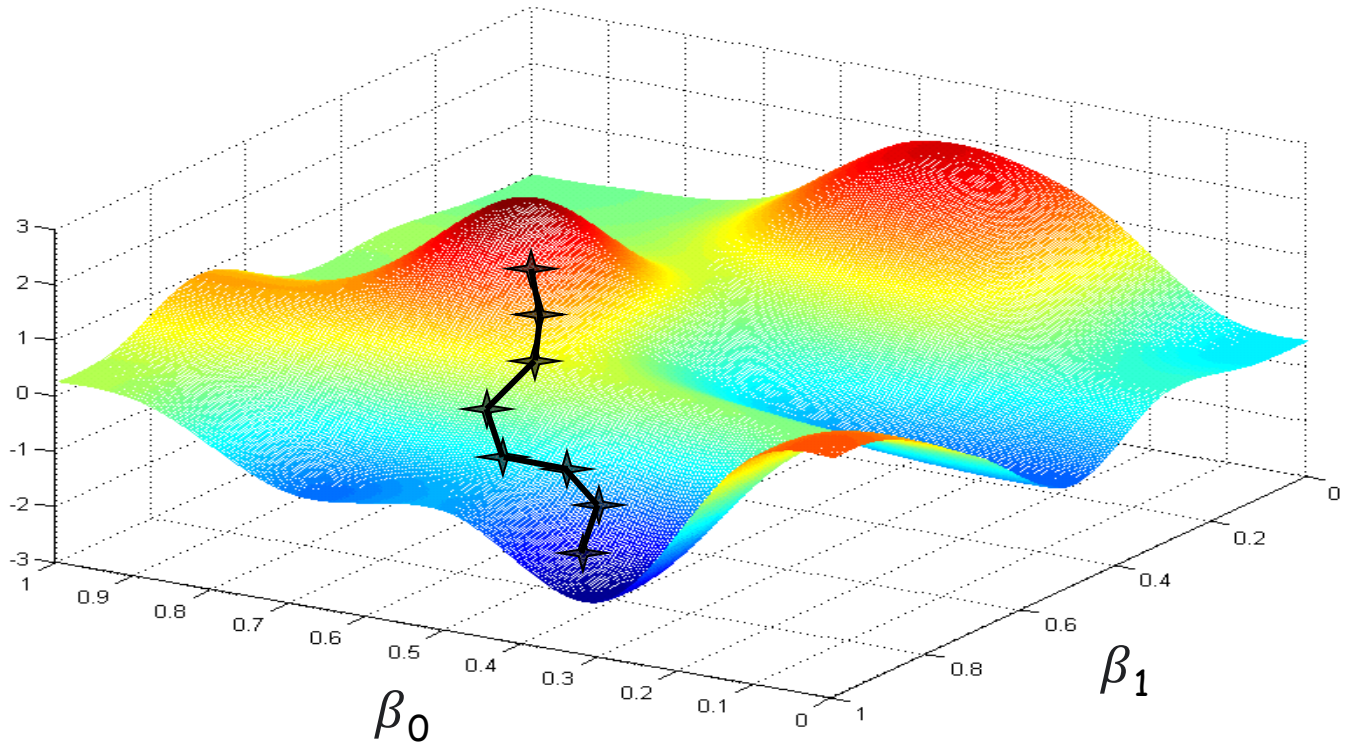


Метод градиентного спуска. Реализация

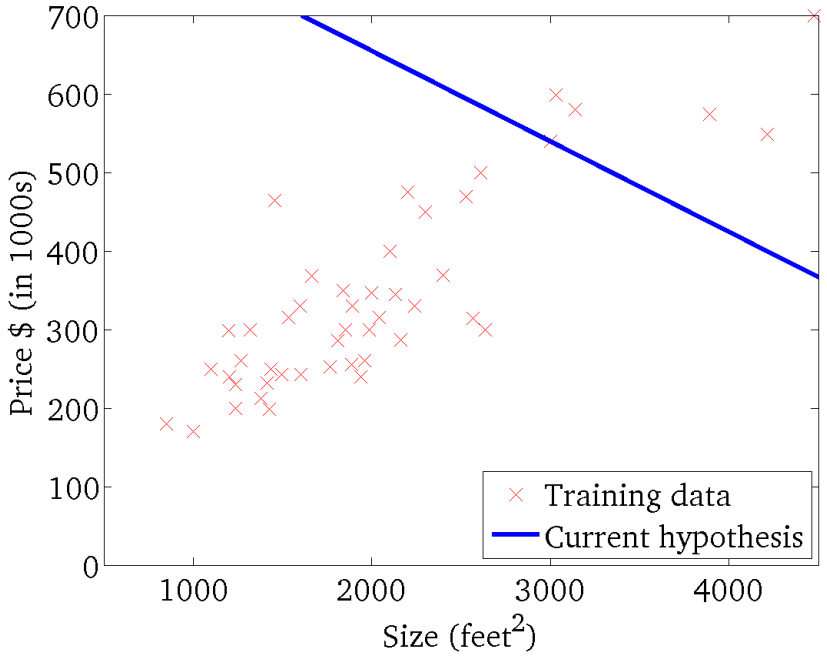
- ✓ Скорость сходимости алгоритма регулируется параметром α
  - ✓ Если α маленькое, то градиентный спуск может быть медленным
  - ✓ Если α большое, то градиентный спуск может проскочить минимум. Алгоритм может не сходиться или даже расходиться
- ✓ Градиентный спуск может сходиться к локальному минимуму, даже если α является фиксированным
  - ✓ При приближении к локальному минимуму градиентный спуск будет автоматически выполнять более малые шаги. Поэтому нет необходимости уменьшать α через некоторое время

Пример формы стоимостной функции

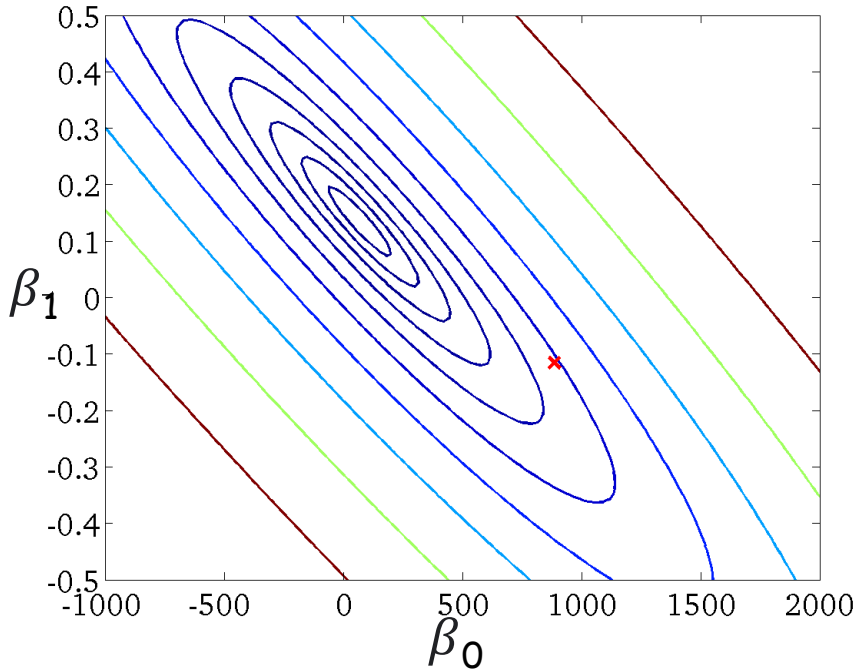
$J(\beta_0, \beta_1)$



Пример работы градиентного спуска для линейной регрессии

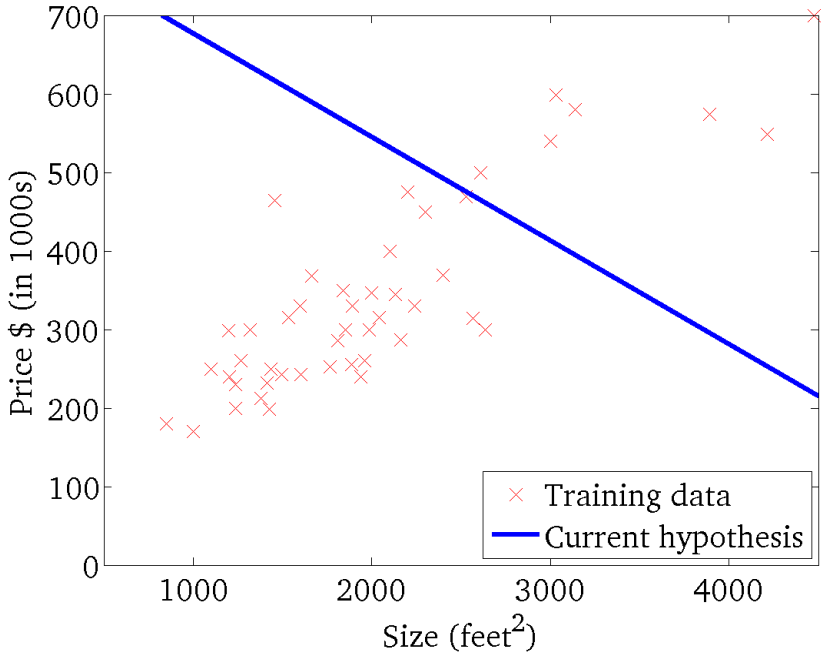


h(x) – это функция x для фиксированных β_0 и β_1

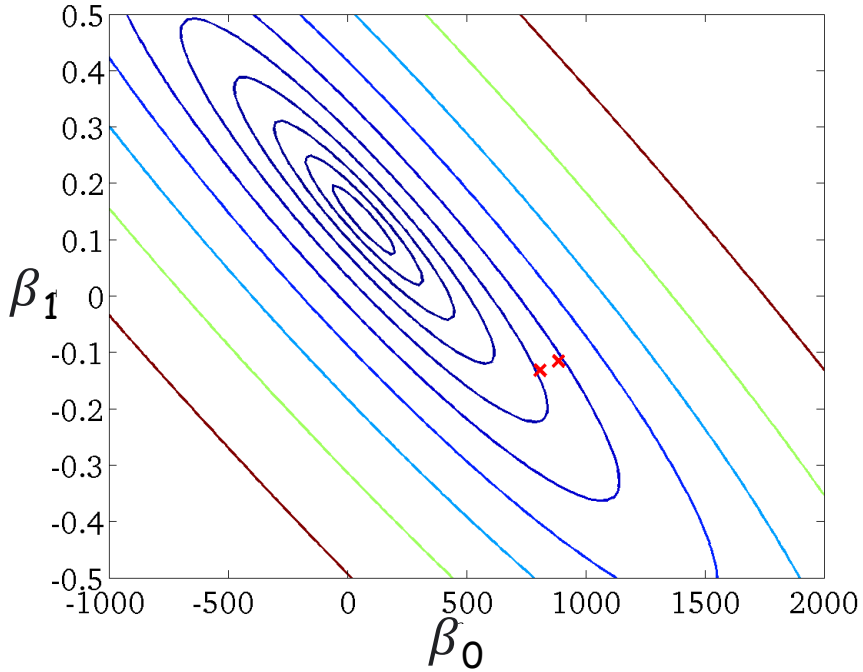


J(β_0, β_1) – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

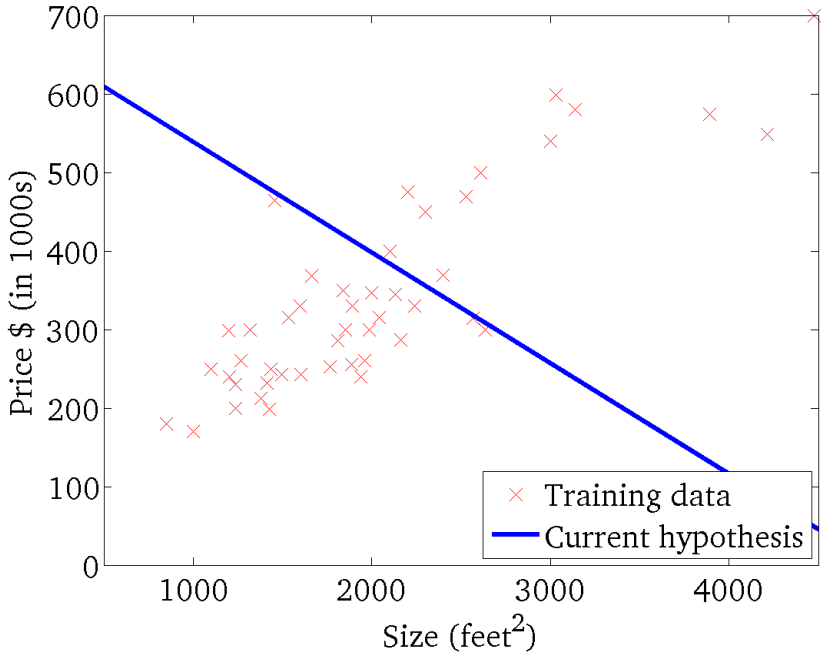


h(x) – это функция x для фиксированных β_0 и β_1

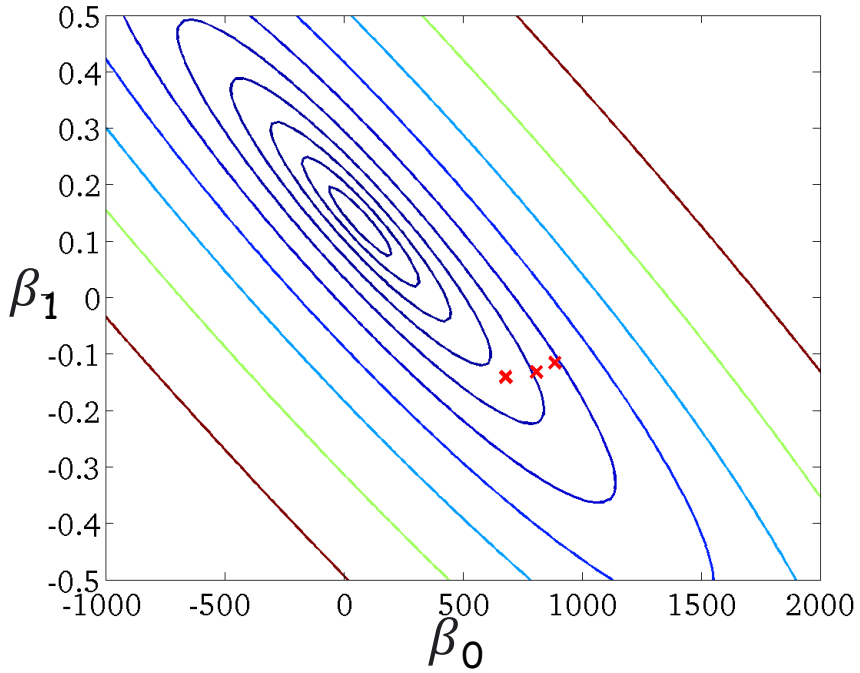


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

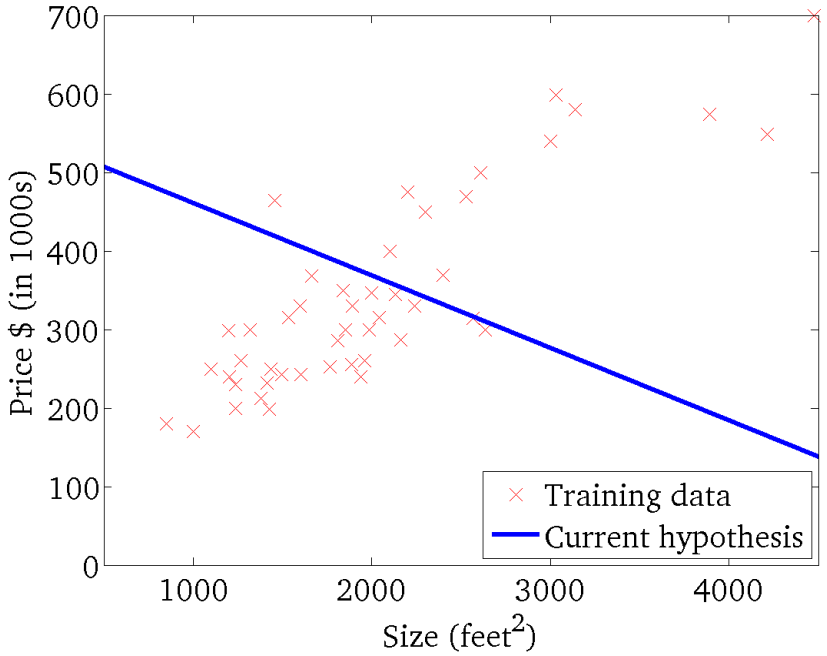


h(x) – это функция x для фиксированных β_0 и β_1

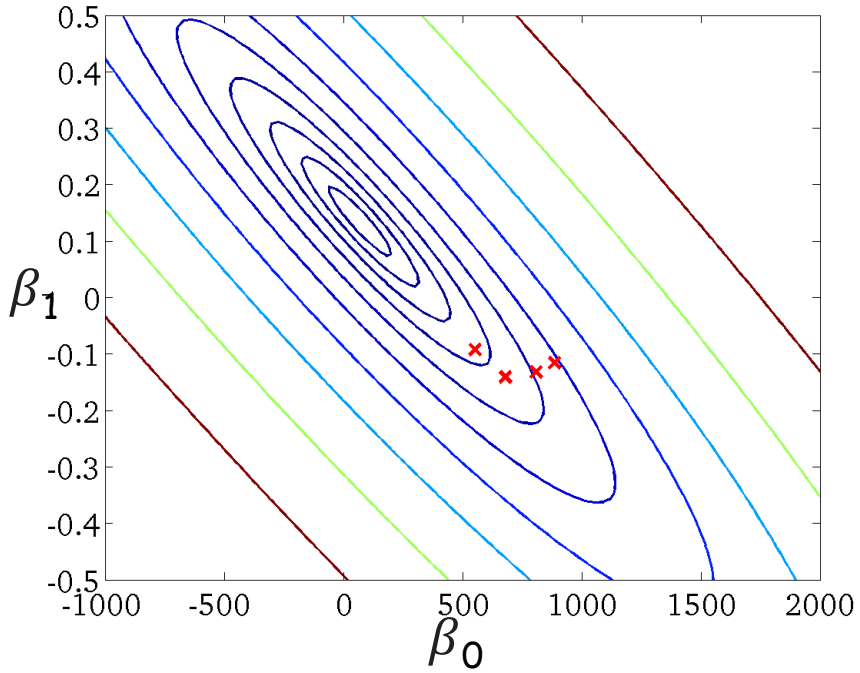


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

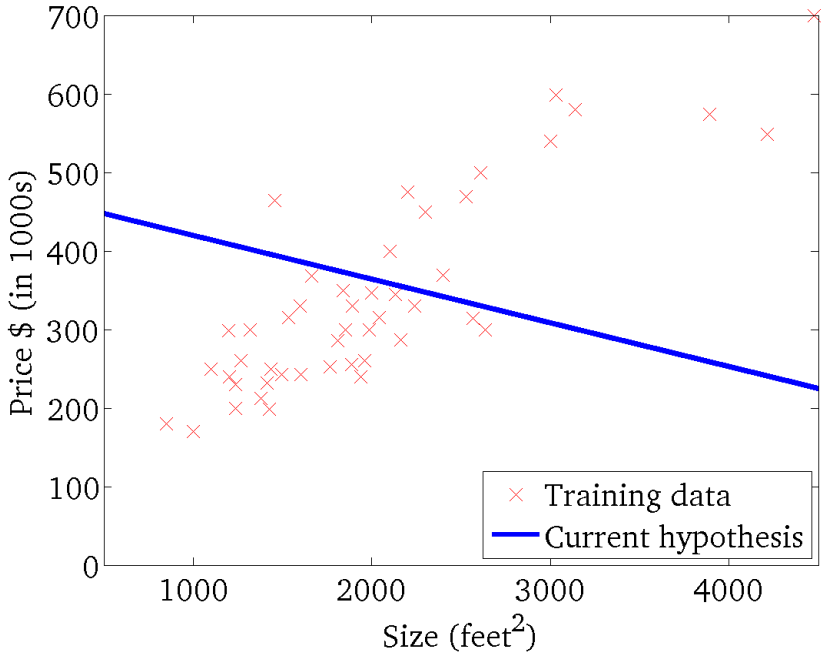


h(x) – это функция x для фиксированных β_0 и β_1

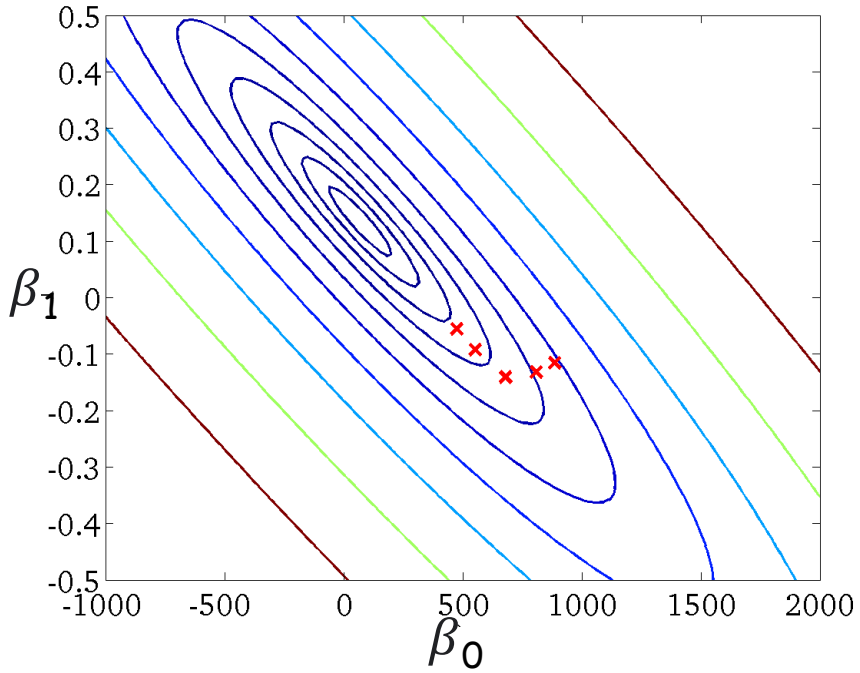


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

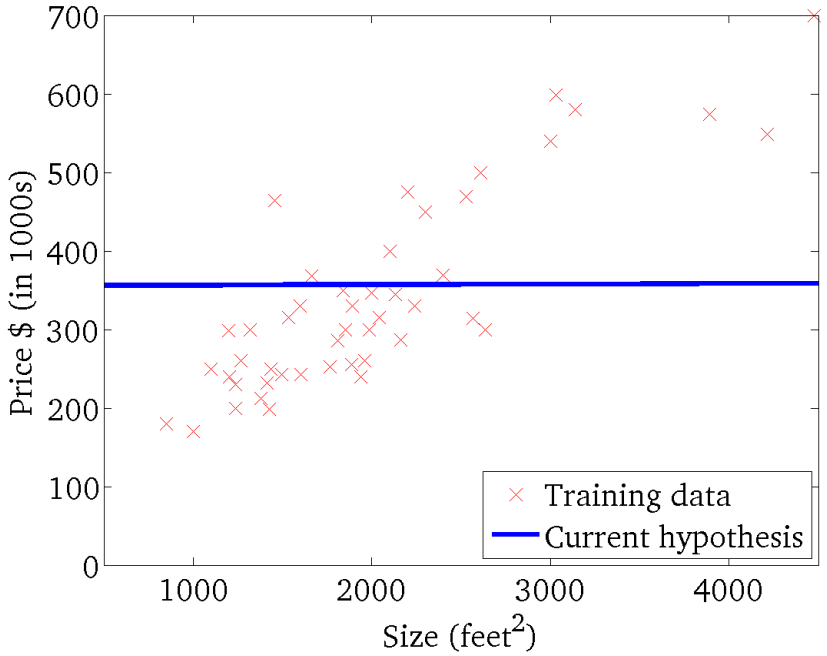


h(x) – это функция x для фиксированных β_0 и β_1

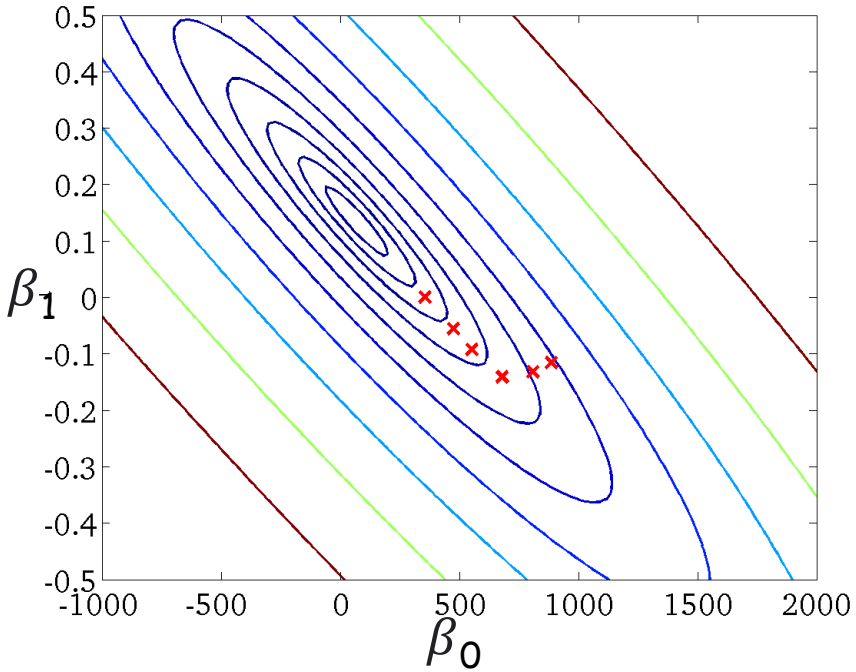


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

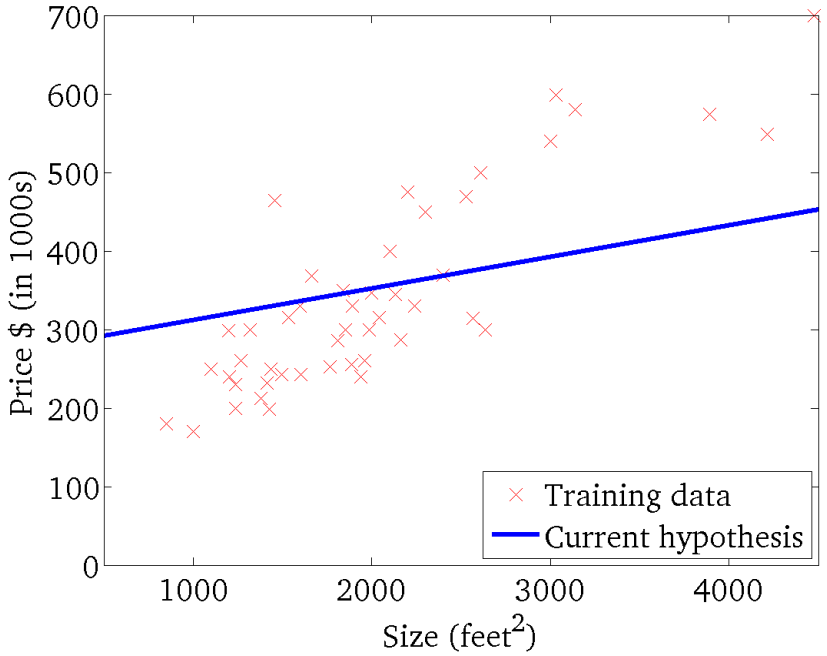


$h_Q(x)$ – это функция x для фиксированных β_0 и β_1

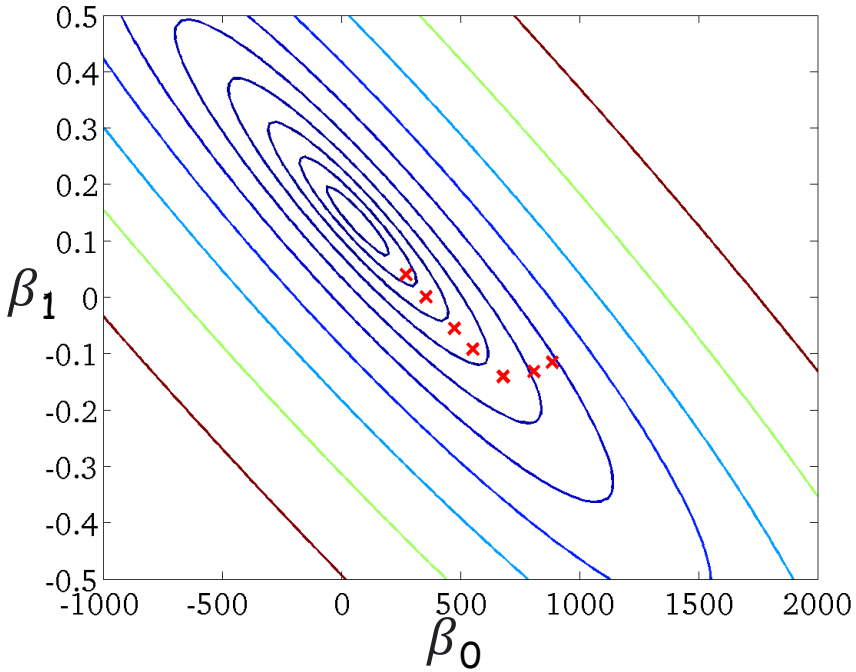


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

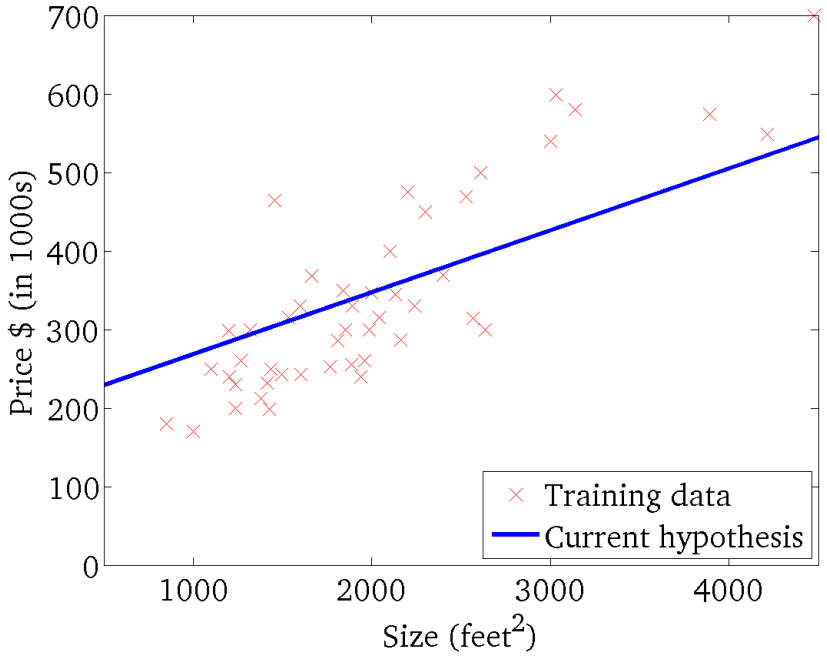


h(x) – это функция x для фиксированных β_0 и β_1

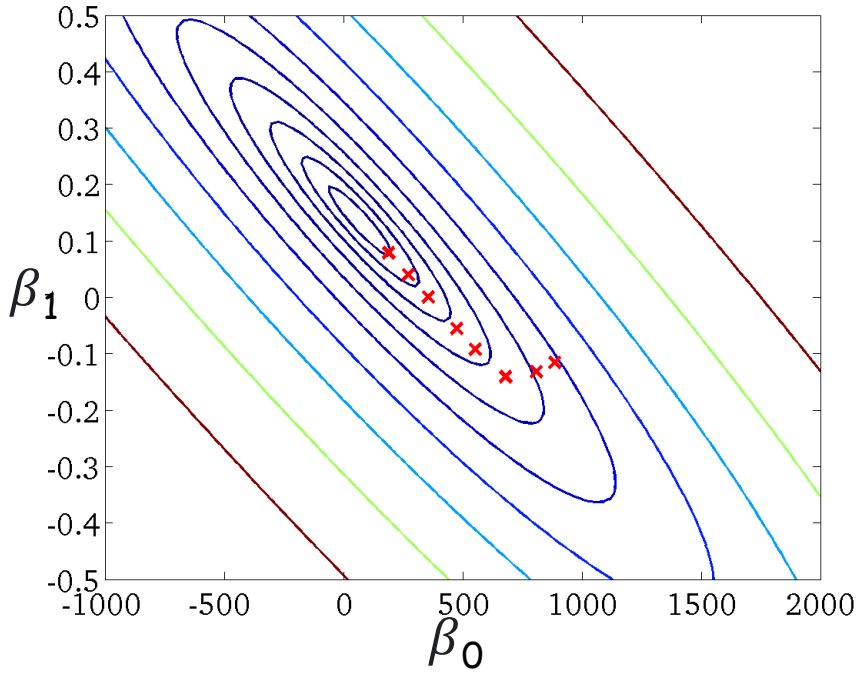


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии

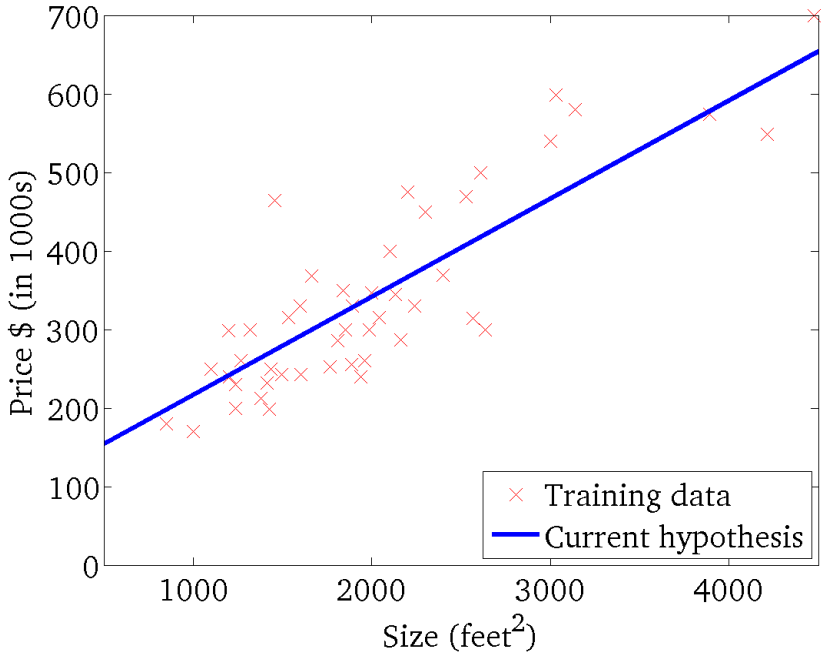


h(x) – это функция x для фиксированных β_0 и β_1

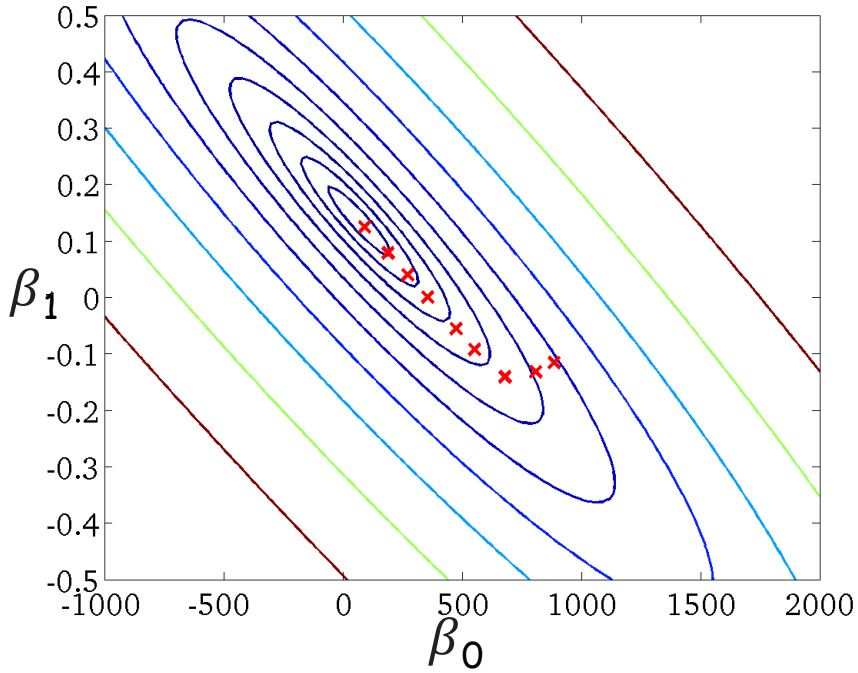


$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

Пример работы градиентного спуска для линейной регрессии



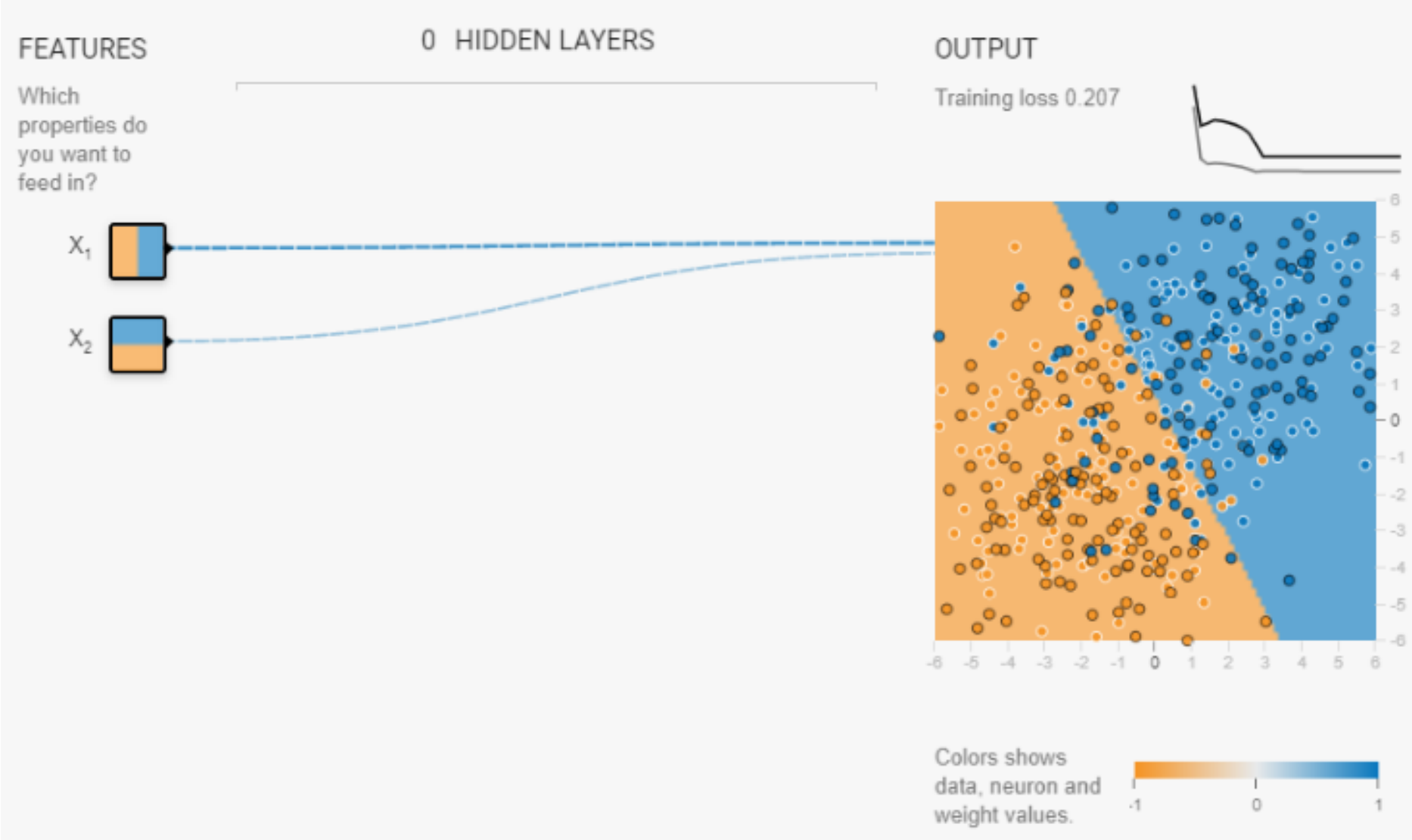
h(x) – это функция x для фиксированных β_0 и β_1



$J(\beta_0, \beta_1)$ – это функция параметров β_0 и β_1

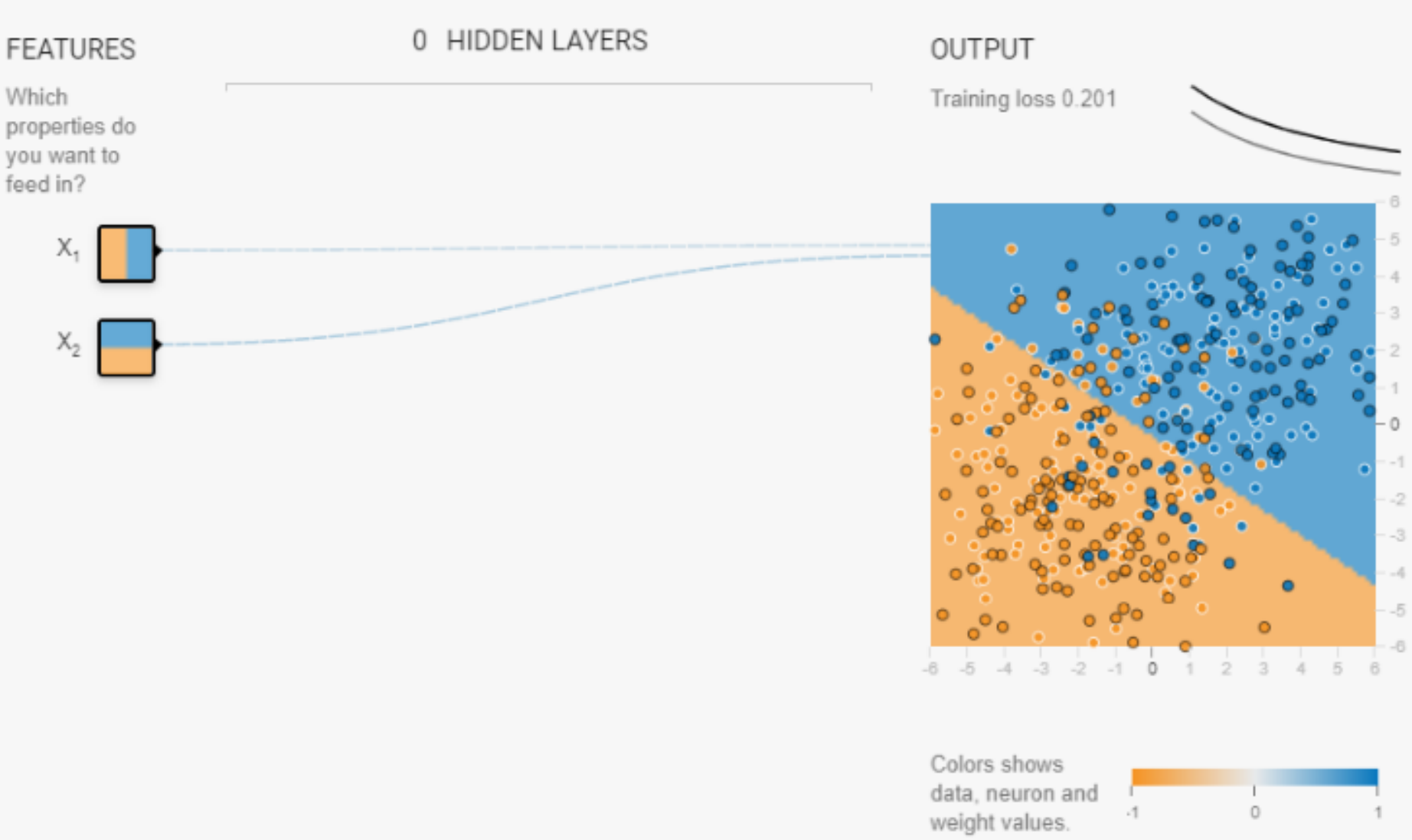
Сокращение потерь: стохастический градиентный спуск

Скорость (уровень) обучения = 0.3, Число эпох обучения = 25



Сокращение потерь: стохастический градиентный спуск

Скорость (уровень) обучения = 0.0001, Число эпох обучения = 40



Сокращение потерь: стохастический градиентный спуск

Скорость (уровень) обучения = 0.1, Число эпох обучения = 5