# Градиентный спуск с линейной регрессией

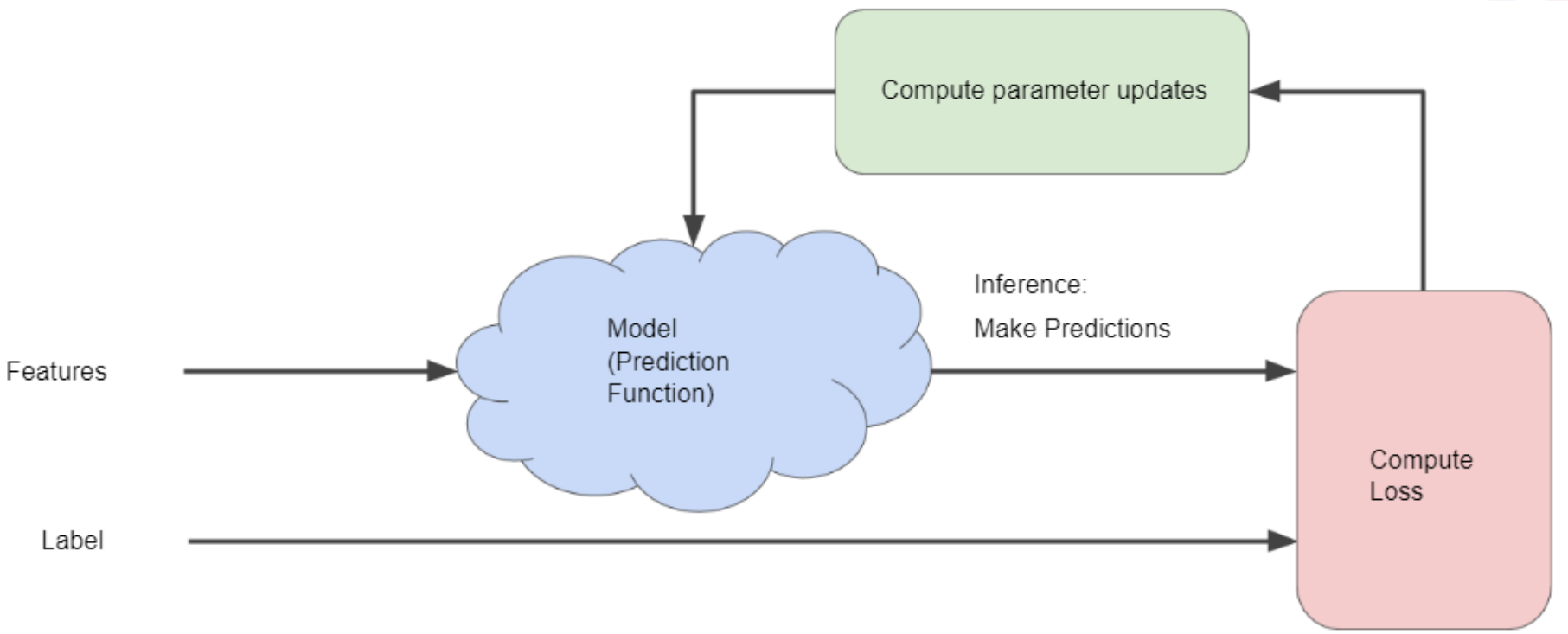

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

# 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:

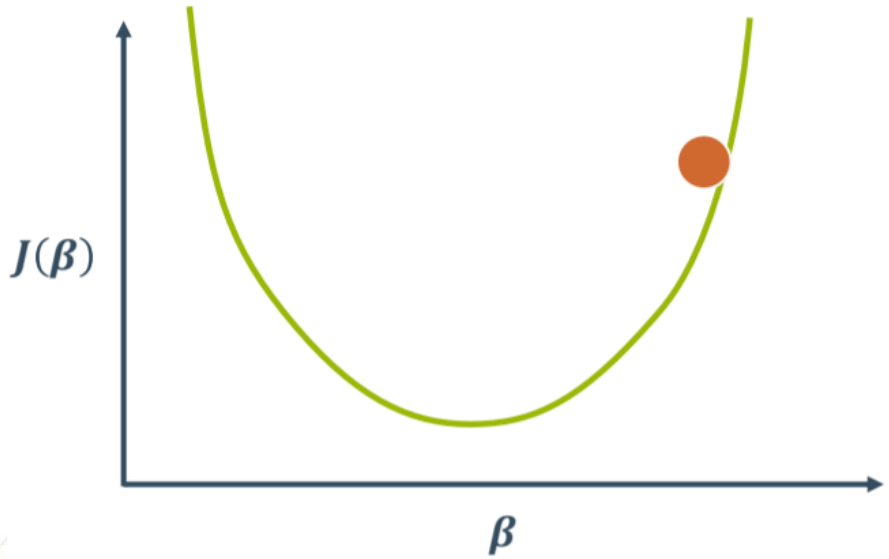
Итеративные стратегии распространены в машинном обучении, в первую очередь потому, что они хорошо масштабируются для больших наборов данных



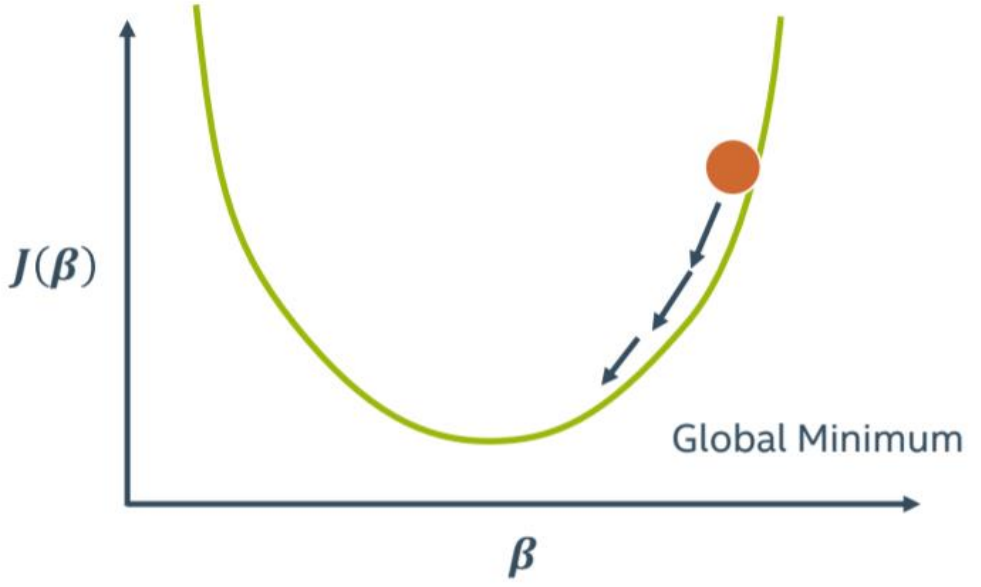
# Градиентный спуск

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 w1 . Для задачи регрессии, которую мы рассматриваем, результирующий график потерь всегда будет выпуклым




Начальное значение функции $J(\beta)$




Затем постепенно двигайтесь к минимуму

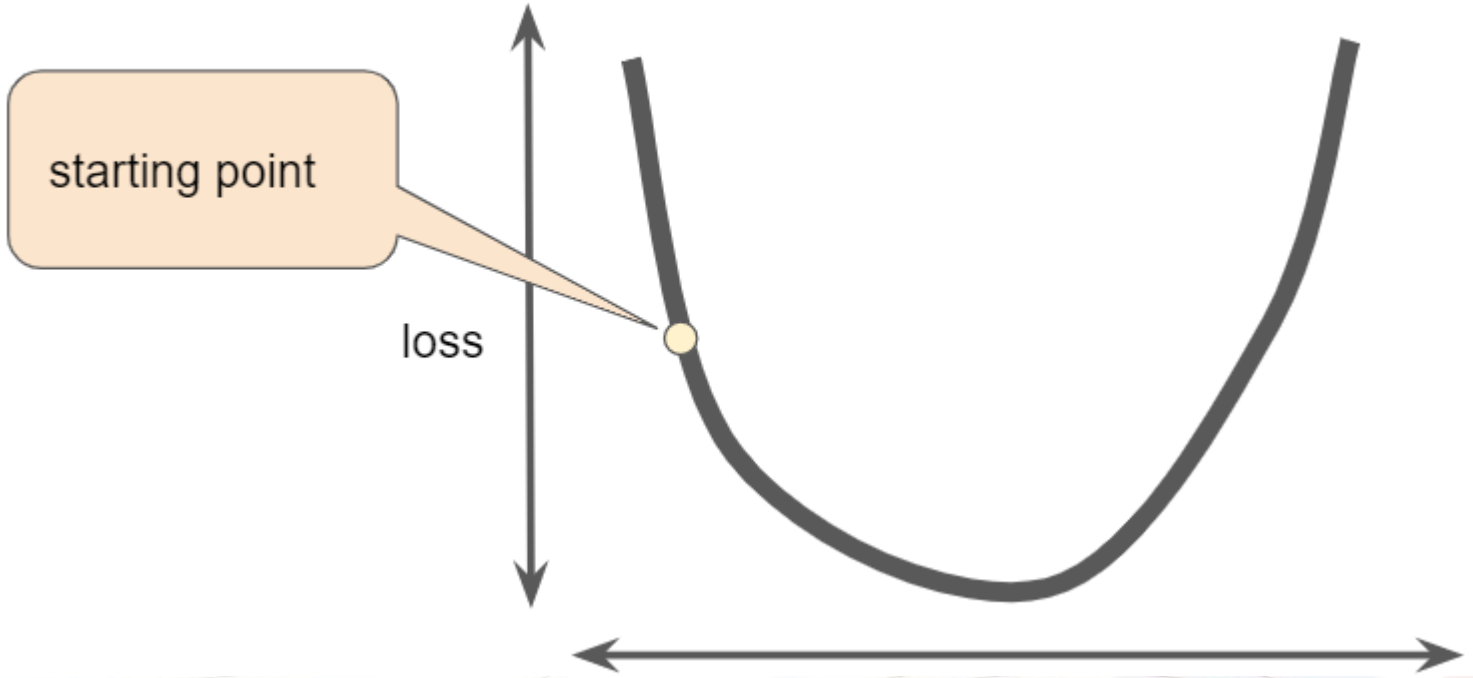
# Градиентный спуск

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 $\beta$ 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 $\beta$. Отправная точка не имеет большого значения; поэтому многие алгоритмы просто устанавливают $\beta$=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 $\beta_i$)

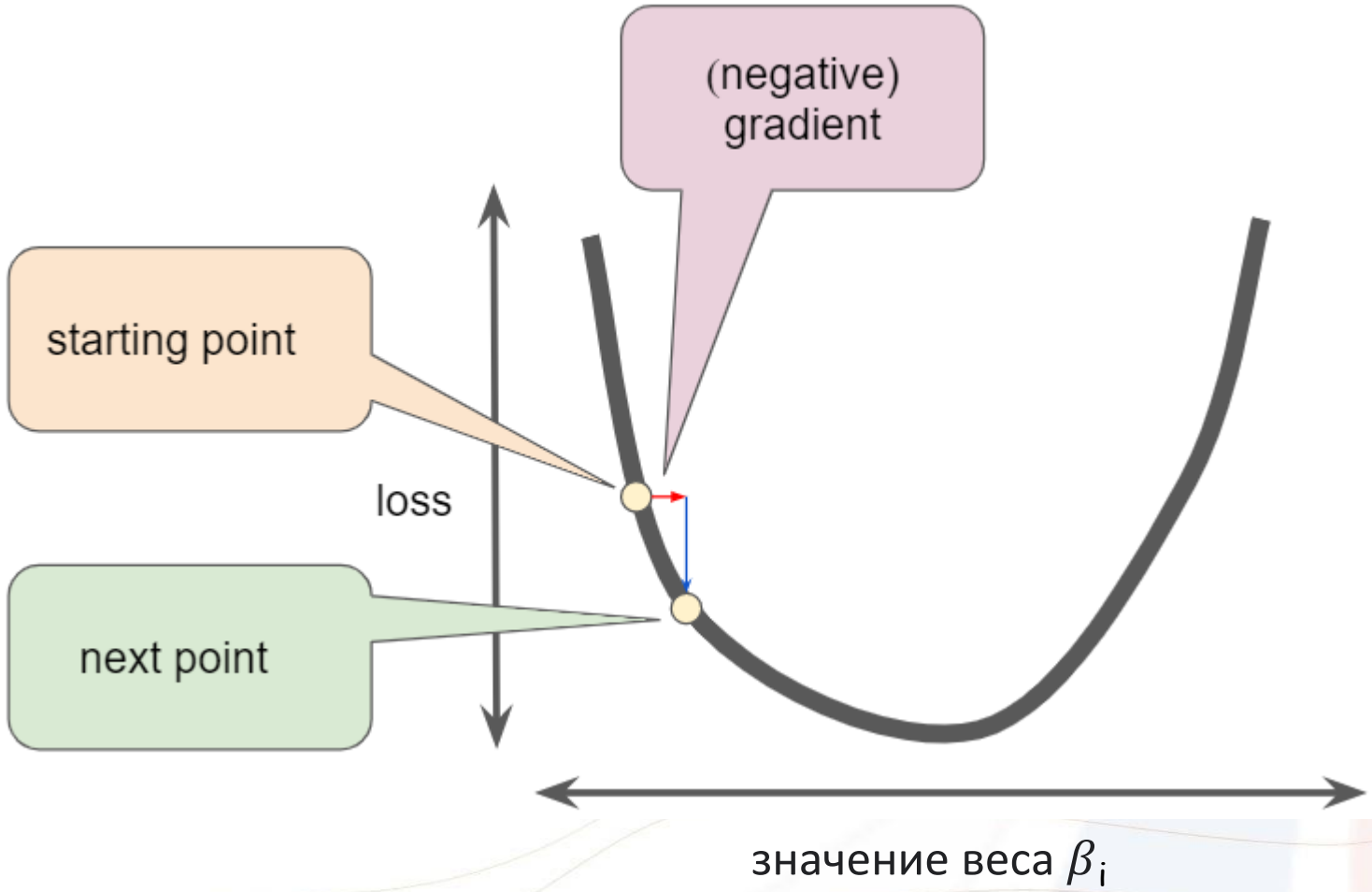


# Градиентный спуск

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**

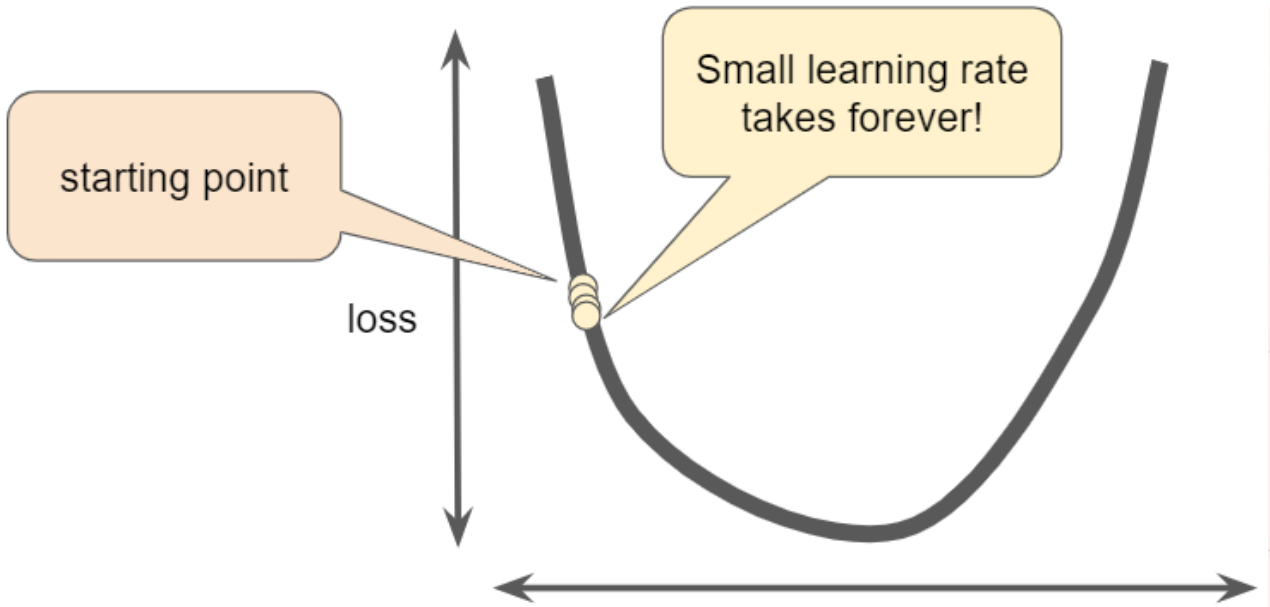


# Минимизация потерь: скорость обучения

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.

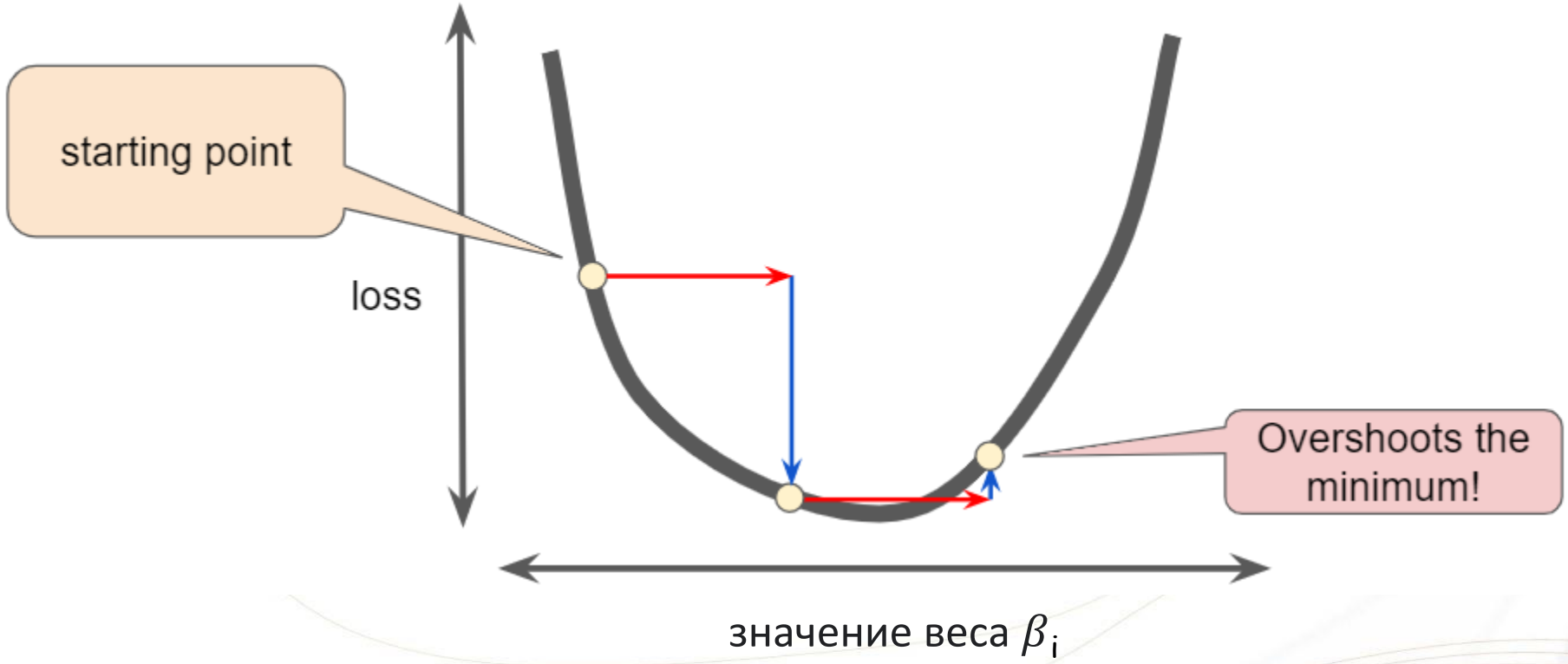
Гиперпараметры - это ручки, которые программисты настраивают в алгоритмах машинного обучения. Большинство программистов машинного обучения тратят немало времени на настройку скорости обучения. Если вы выберете слишком низкую скорость обучения, обучение займет слишком много времени.

## Скорость обучения слишком мала



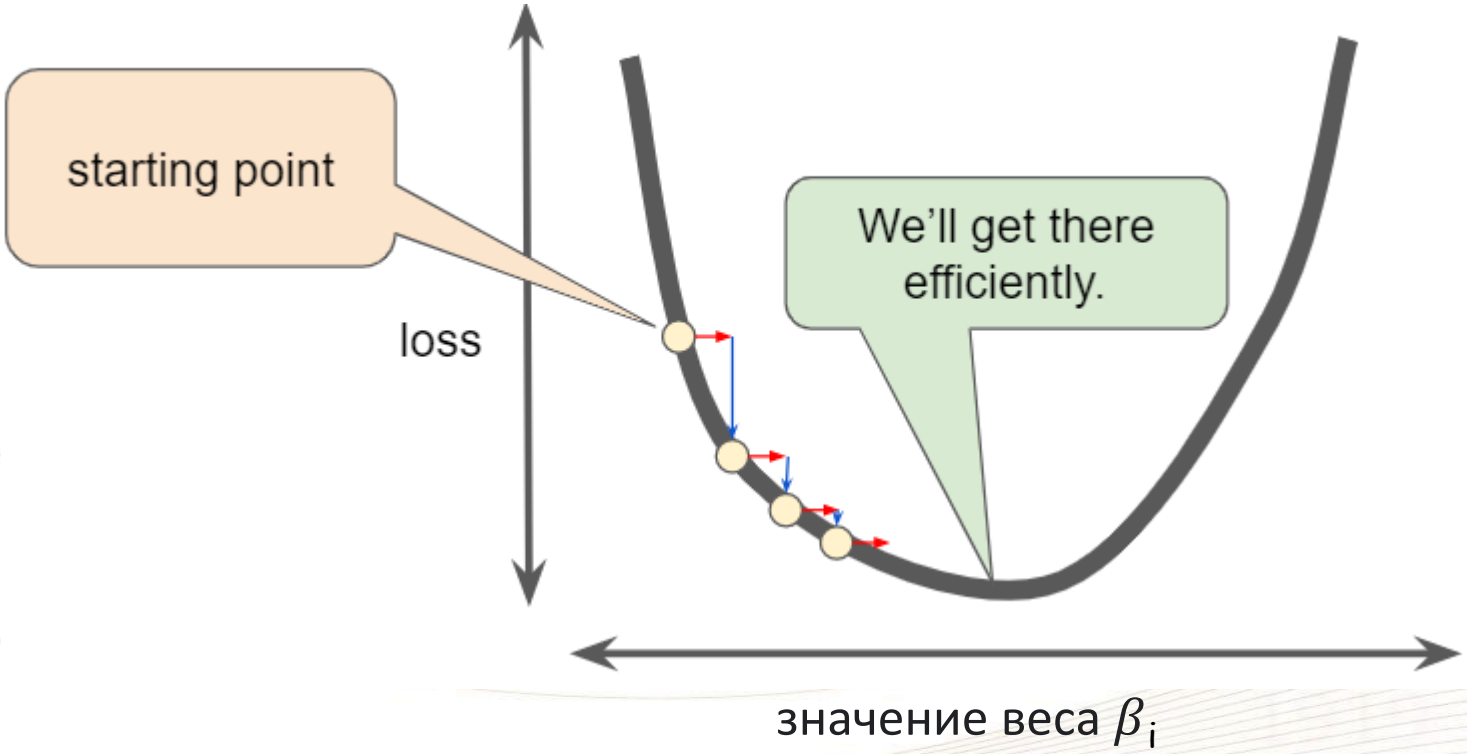
# Минимизация потерь: скорость обучения

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:



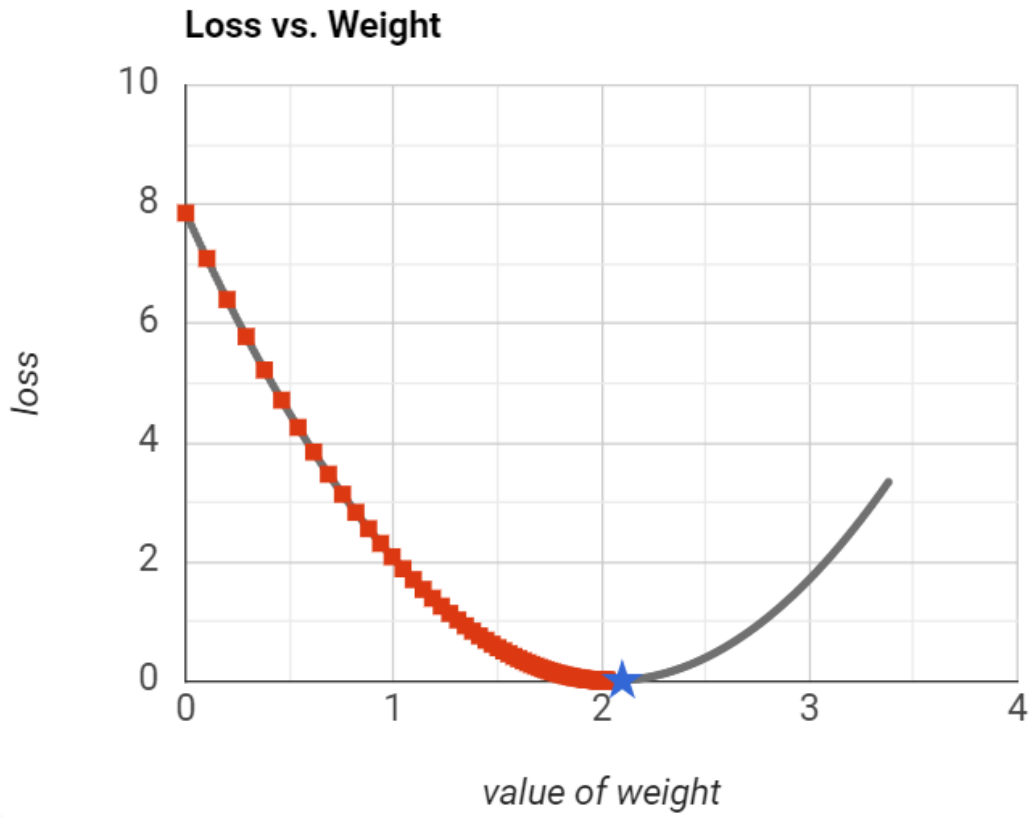
# Минимизация потерь: скорость обучения

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.

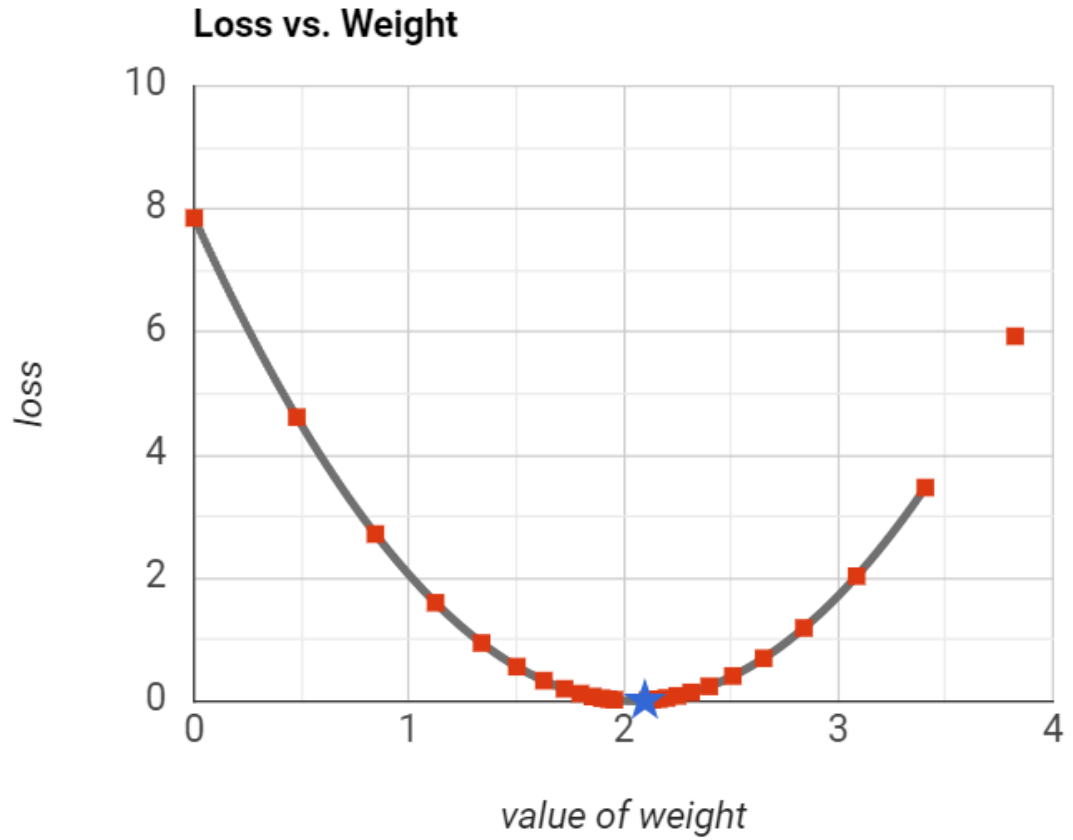


# Сокращение потерь: оптимизация скорости обучения

Скорость (уровень) обучения = 0.08
Число шагов = 96

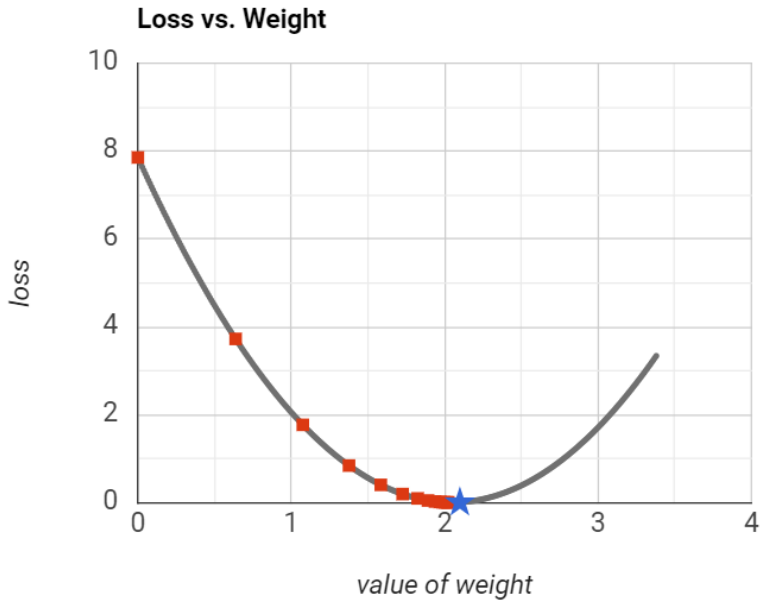



Скорость (уровень) обучения = 3
Число шагов = 25

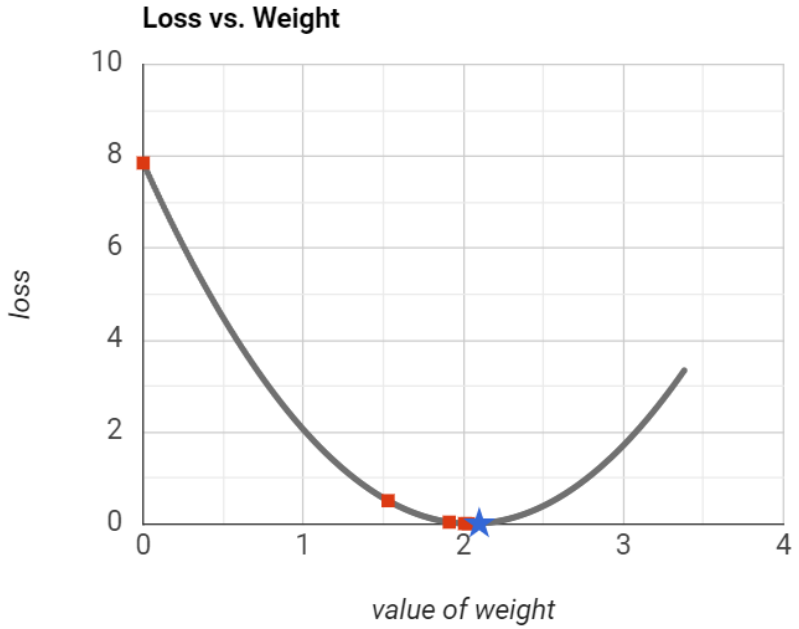


# Сокращение потерь: оптимизация скорости обучения

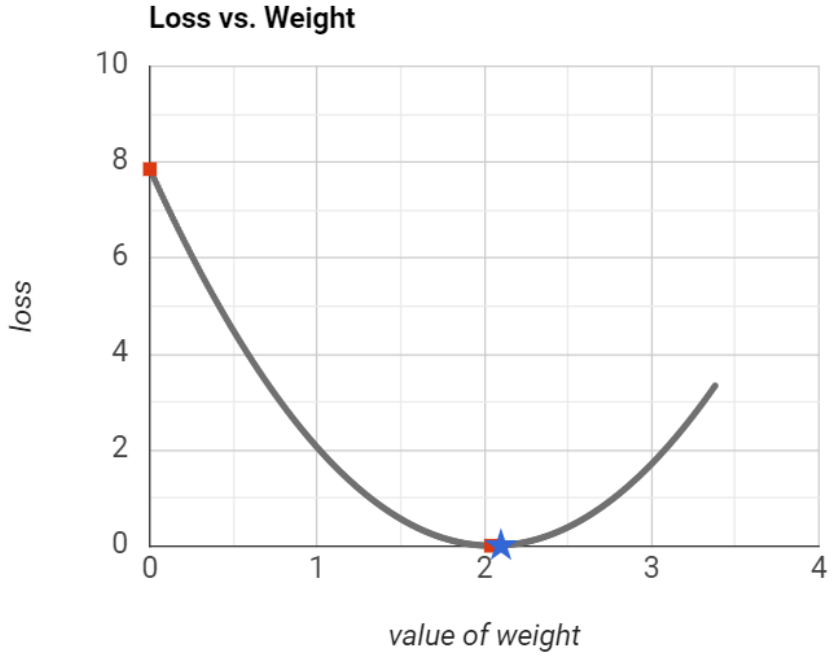
Скорость (уровень) обучения = 1.5
Число шагов = 13




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.

# Сокращение потерь: стохастический градиентный спуск

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.

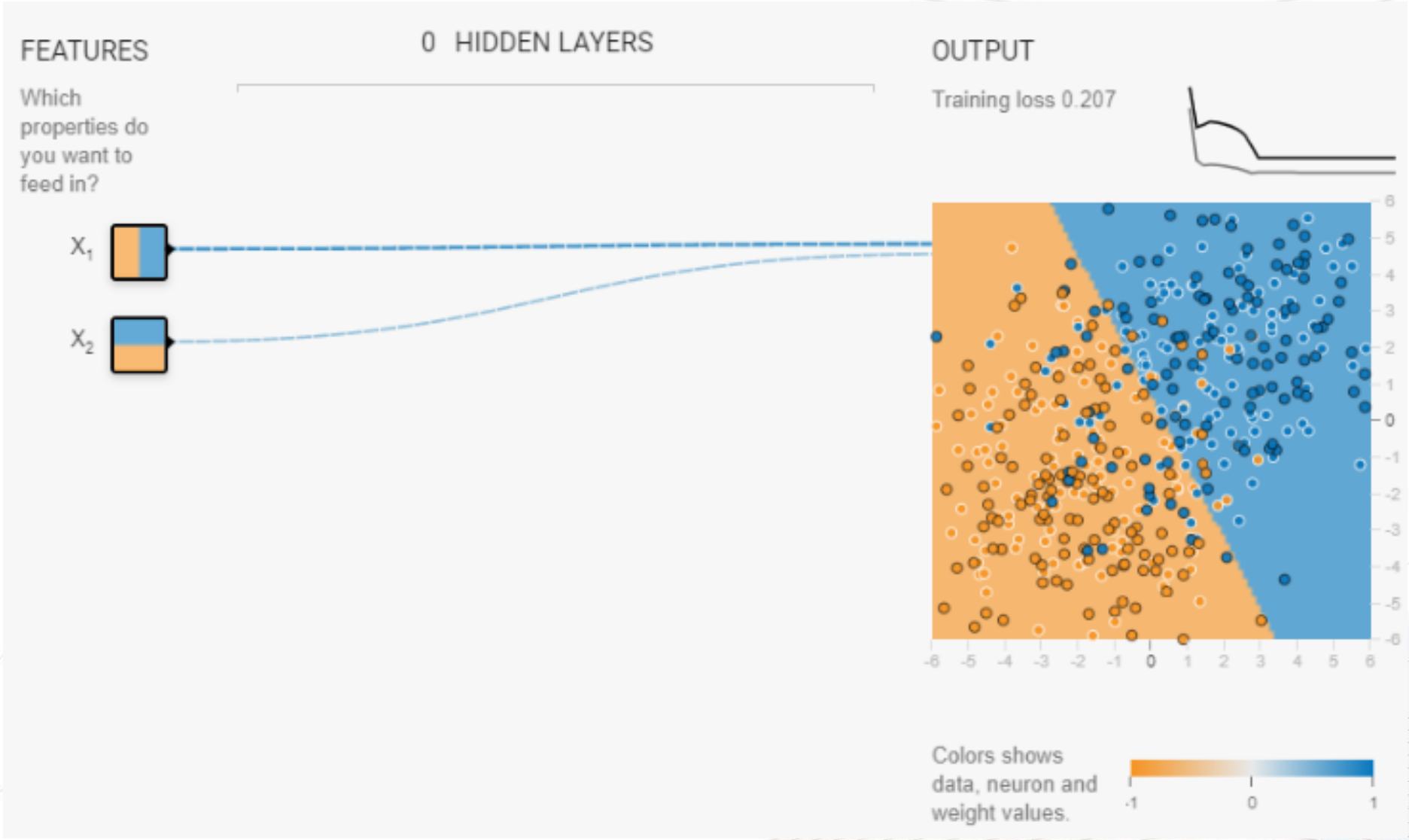
**Стохастический градиентный спуск** (**SGD**) доводит эту идею до крайности - он использует только один пример (размер пакета 1) на одну итерацию. При достаточном количестве итераций SGD работает, но очень шумно. Термин «стохастический» указывает, что один пример, содержащий каждую партию, выбирается 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).

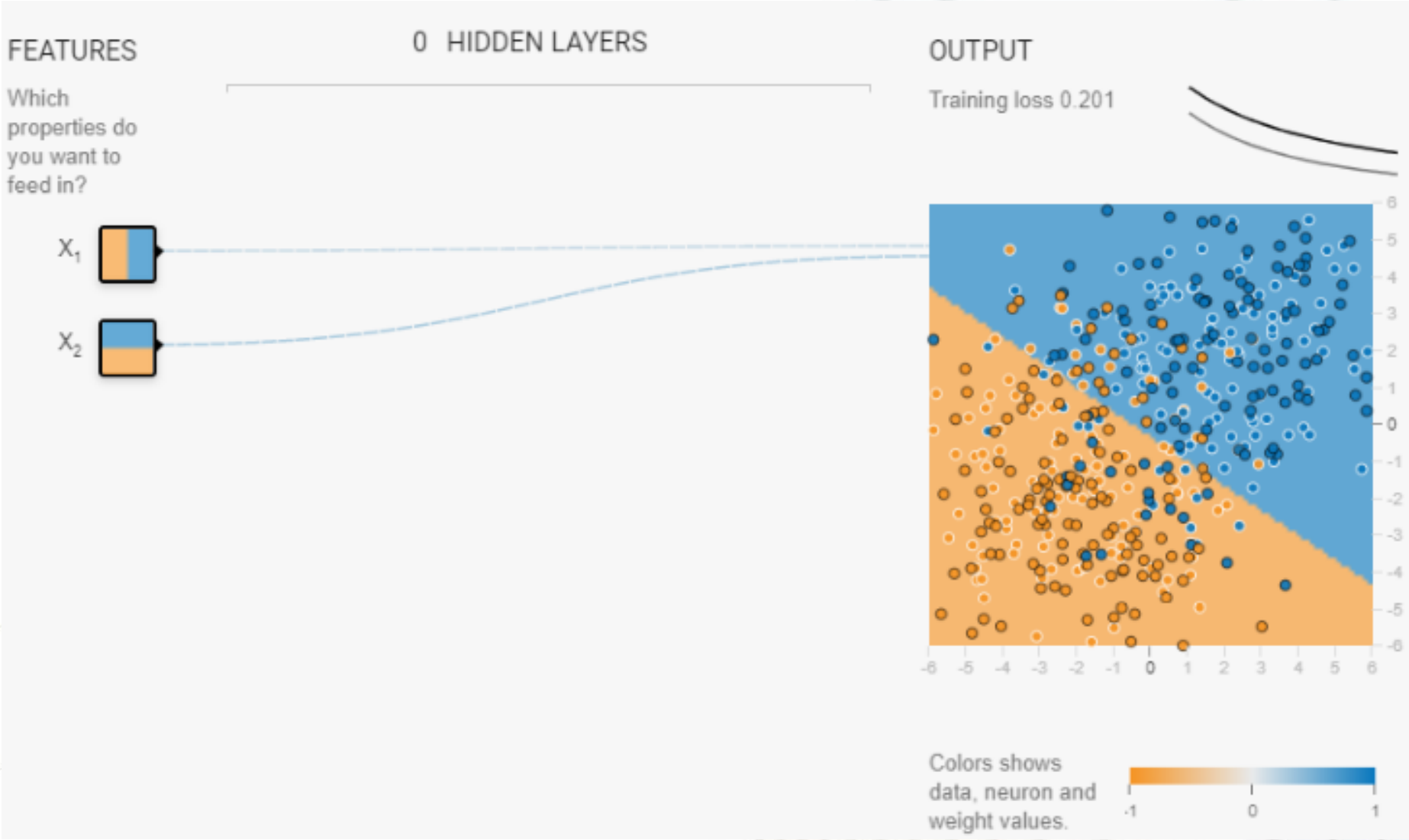
# Сокращение потерь: стохастический градиентный спуск

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