MACHINE LEARNING ACCELERATOR

Tabular Data – Lecture 3

Course Overview

Lecture 1

- Introduction to ML
- Model Evaluation
  - Train-Validation-Test
  - Overfitting
- Exploratory Data Analysis
- K Nearest Neighbors (KNN)

Lecture 2

- Feature Engineering
- Tree-based Models
  - Decision Tree
  - Random Forest
- Hyperparameter Tuning
- AWS AI/ML Services

Lecture 3

- Optimization
- Regression Models
- Regularization
- Boosting
- Neural Networks
- AutoML

Optimization

Optimization in Machine Learning

- Мы создаем и обучаем модели машинного обучения, надеясь на :

ML Model Features → **ML Model (Rules)** → **ML Model Target**

↕ error

- В реальности …

ML Model Features → **ML Model (Rules)** → **ML Model Prediction**

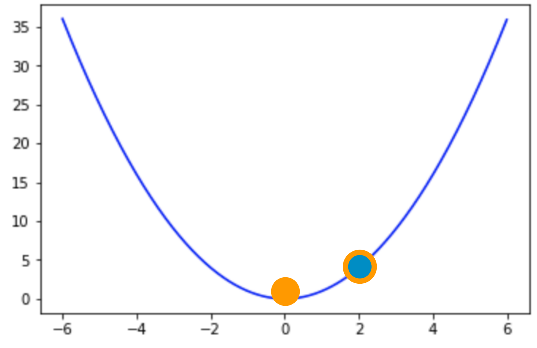
- **Изучите все более совершенные модели**, чтобы общая ошибка модели становилась все меньше и меньше… в идеале, **как можно меньше!**

Optimization

- В ML используйте оптимизацию, чтобы **минимизировать функцию ошибок модели** ML.
  - **Error function:** $error = f(w)$, where w = input, f = function, $error$ = output
  - **Оптимизация функции ошибок**:
    - Минимизация f означает поиск w, при которых достигается наименьшее значение $error$

$$error = f(w) = 0.6w^2$$

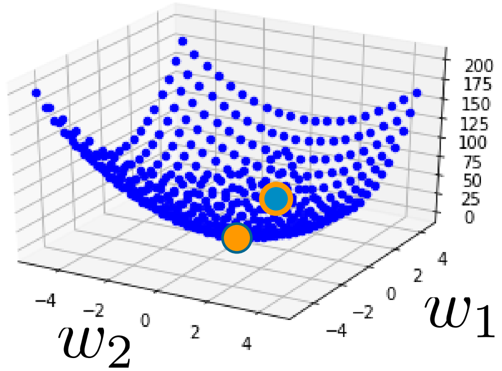
$error$



w

$$error = f(w_1, w_2) = 3w_1^2 + 5w_2^2$$

$error$



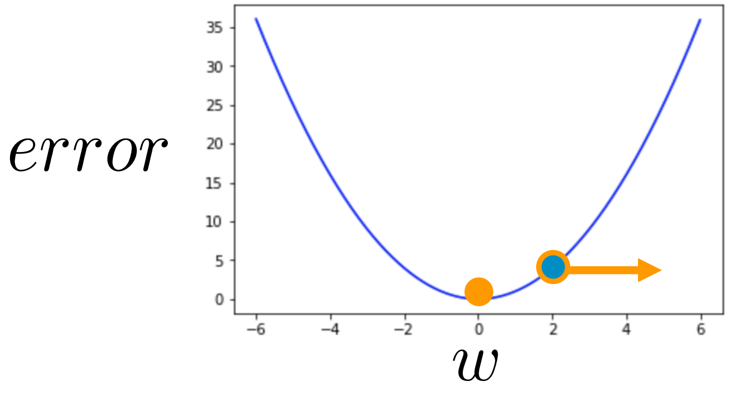
w_2 w_1

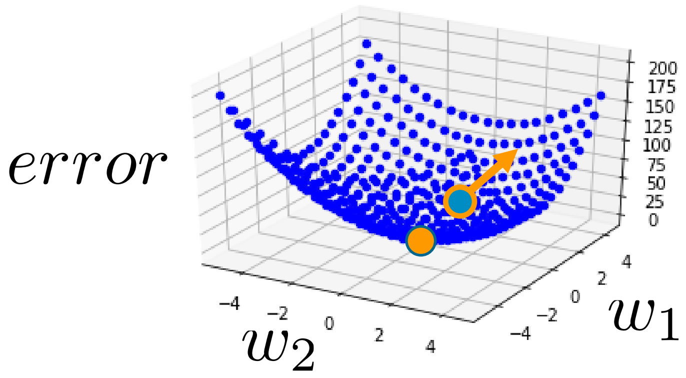
Gradient Optimization

- **Градиент:** вектор с координатами частных производных, указывающий направление и скорость самого быстрого увеличения функции
  - Его можно вычислить с помощью частных производных функции f по w в каждой входной переменной.
  - Поскольку у него есть направление, градиент является «вектором».

$$error = f(w) = 0.6w^2$$

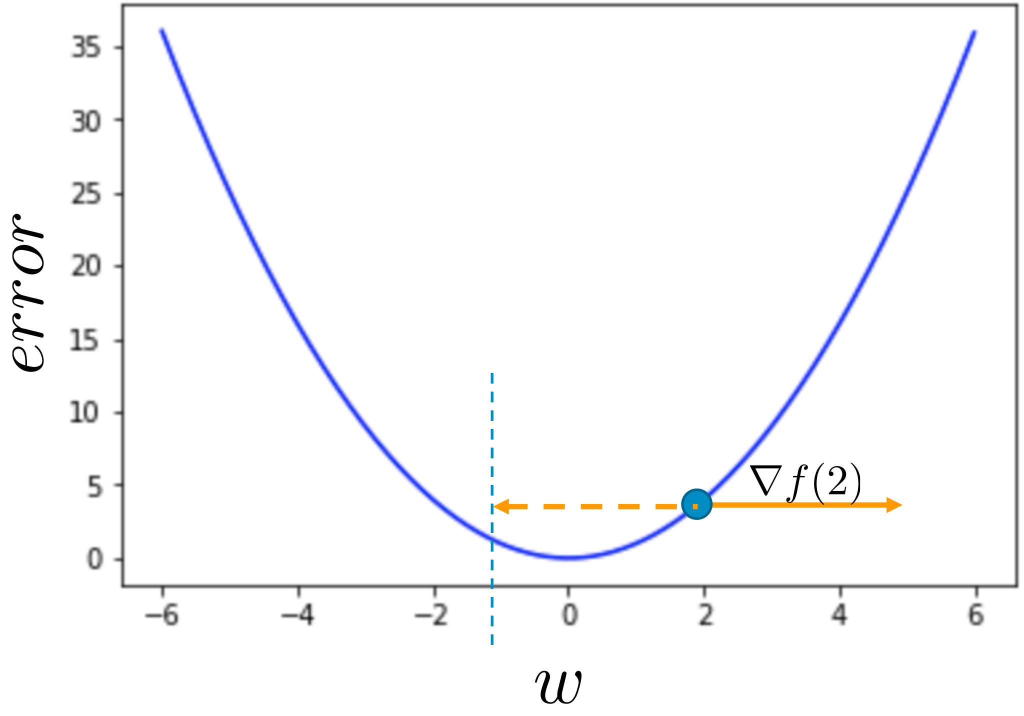
$$error = f(w_1, w_2) = 3w_1^2 + 5w_2^2$$





Gradient Example

$$error = f(w) = 0.6w^2$$



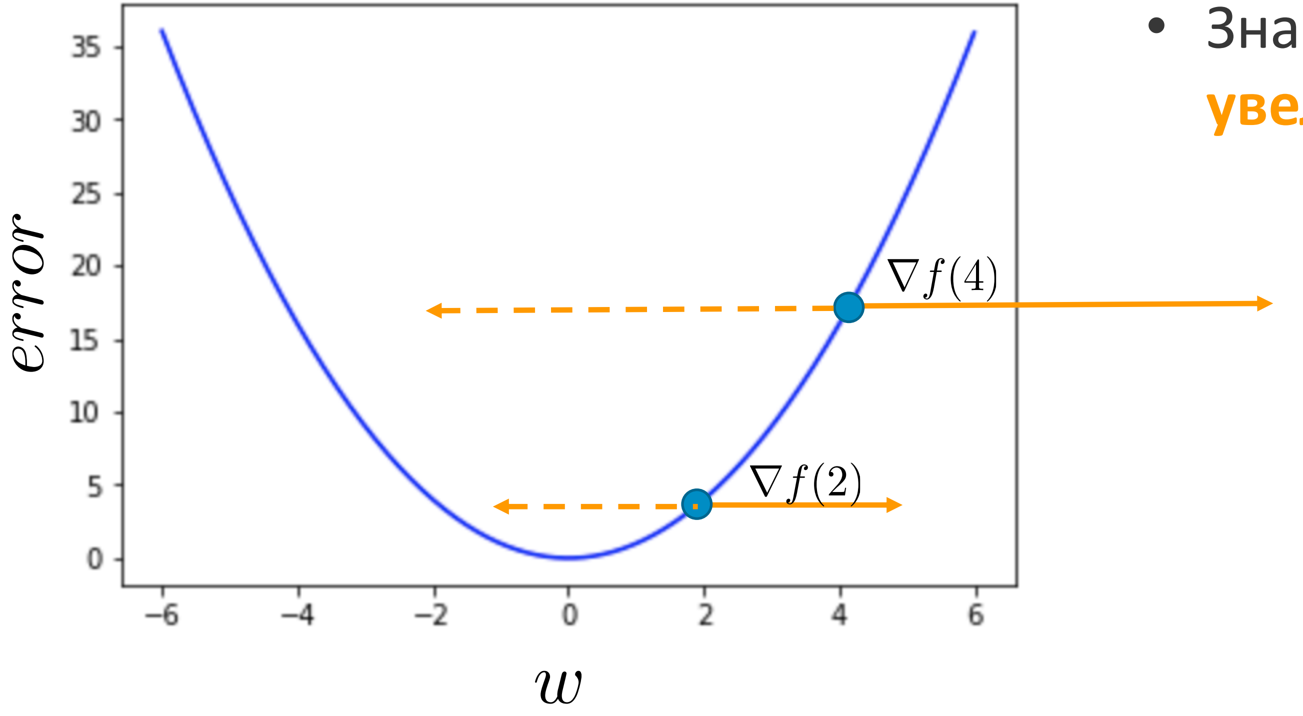
$f(w) = 0.6w^2$, с вектором $\nabla f =< 1.2w >$

- Знак градиента **показывает направление увеличения функции**: + right and – left

$$\nabla f(2) =< 2.4 > \Rightarrow |\nabla f(2)| = 2.4$$

Gradient Example

$$error = f(w) = 0.6w^2$$



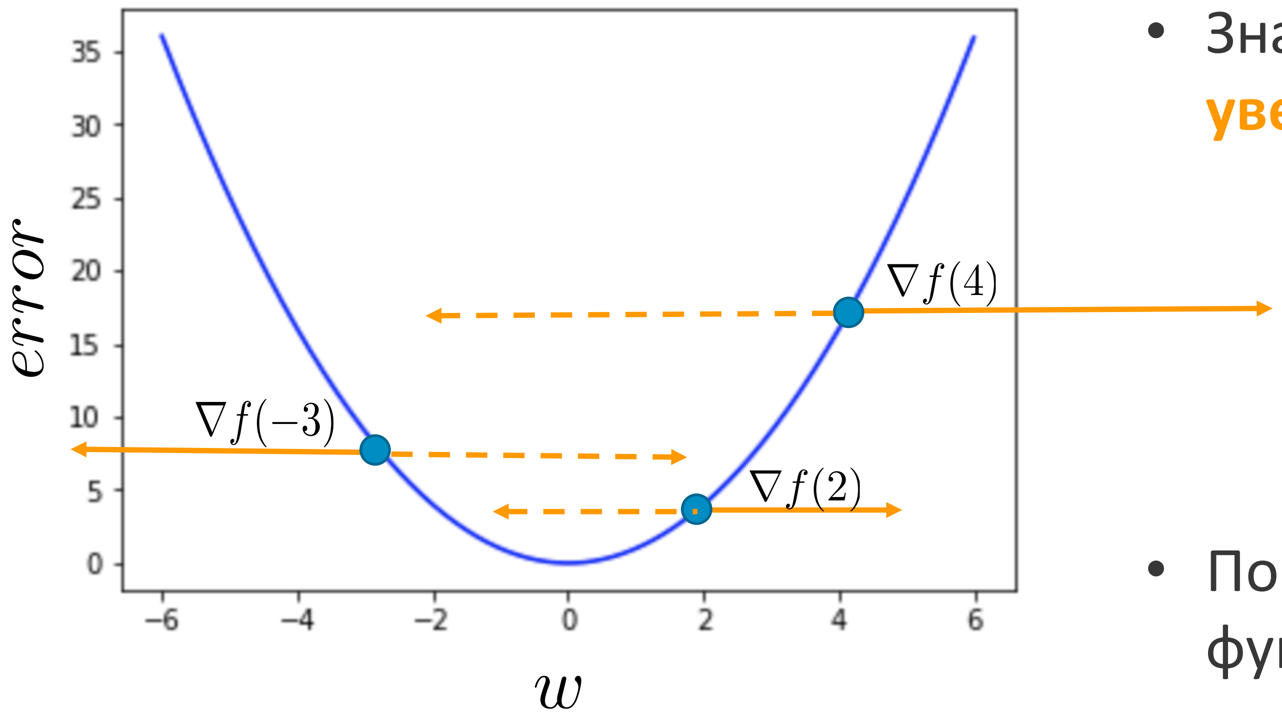
$f(w) = 0.6w^2$, with gradient vector $\nabla f =< 1.2w >$

- Знак градиента **показывает направление увеличения функции**: + right and – left

$$\nabla f(2) =< 2.4 > \Rightarrow |\nabla f(2)| = 2.4$$
$$\nabla f(4) =< 4.8 > \Rightarrow |\nabla f(4)| = 4.8$$

Gradient Example

$$error = f(w) = 0.6w^2$$



$f(w) = 0.6w^2$, with gradient vector $\nabla f =< 1.2w >$

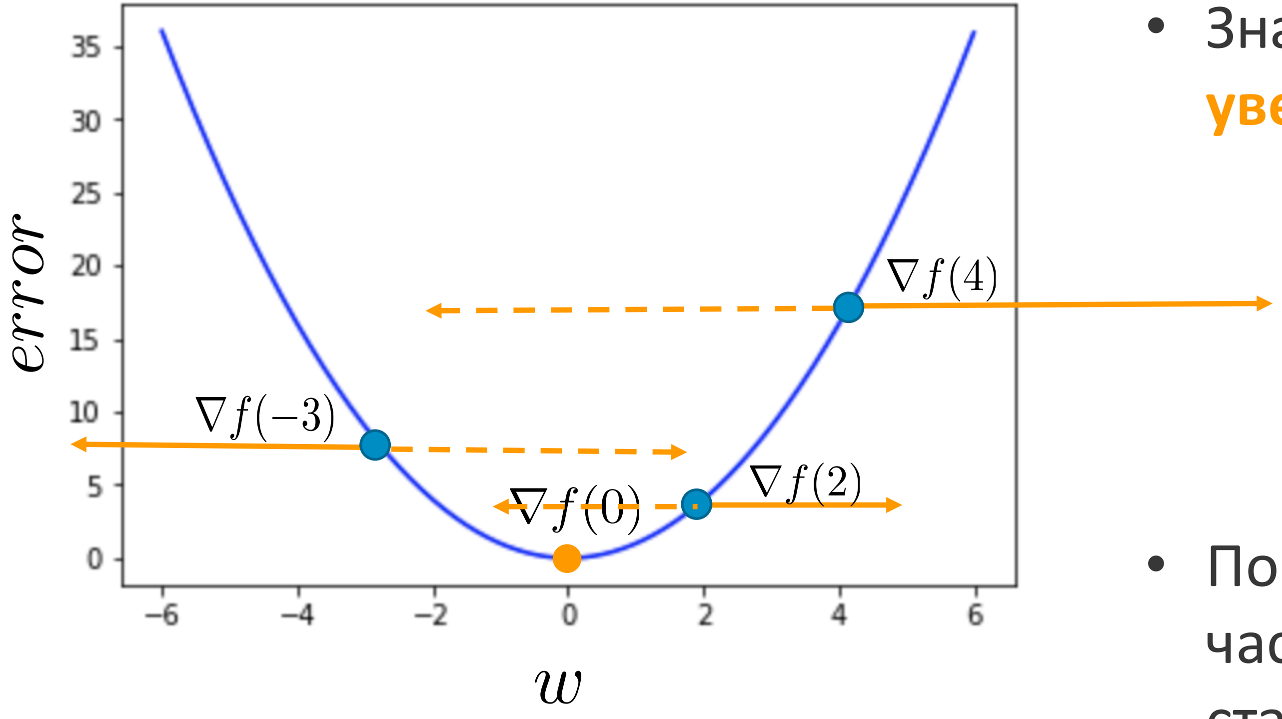
- Знак градиента **показывает направление увеличения функции**: + right and – left

$$\nabla f(2) =< 2.4 > \Rightarrow |\nabla f(2)| = 2.4$$
$$\nabla f(4) =< 4.8 > \Rightarrow |\nabla f(4)| = 4.8$$
$$\nabla f(-3) =< -3.6 > \Rightarrow |\nabla f(-3)| = 3.6$$

- По мере продвижения к нижней части функции градиент становится меньше

Gradient Example

$$error = f(w) = 0.6w^2$$



$f(w) = 0.6w^2$, with gradient vector $\nabla f =< 1.2w >$

- Знак градиента **показывает направление увеличения функции** : + right and – left

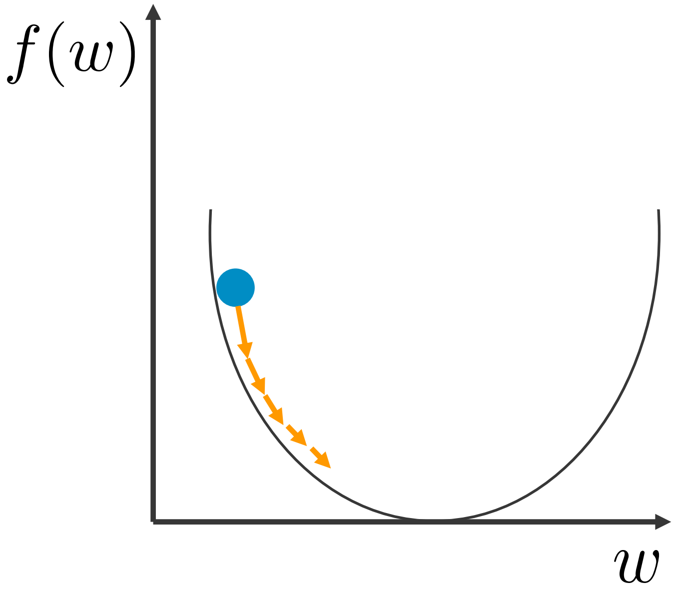
$$\nabla f(2) =< 2.4 > \Rightarrow |\nabla f(2)| = 2.4$$
$$\nabla f(4) =< 4.8 > \Rightarrow |\nabla f(4)| = 4.8$$
$$\nabla f(-3) =< -3.6 > \Rightarrow |\nabla f(-3)| = 3.6$$
$$\nabla f(0) =< 0 > \Rightarrow |\nabla f(0)| = 0$$

- По мере того, как мы приближаемся к нижней части функции, градиент становится меньше и становится равным нулю (т.е. функция больше не может меняться, не может больше уменьшаться - **она достигла минимума**!)

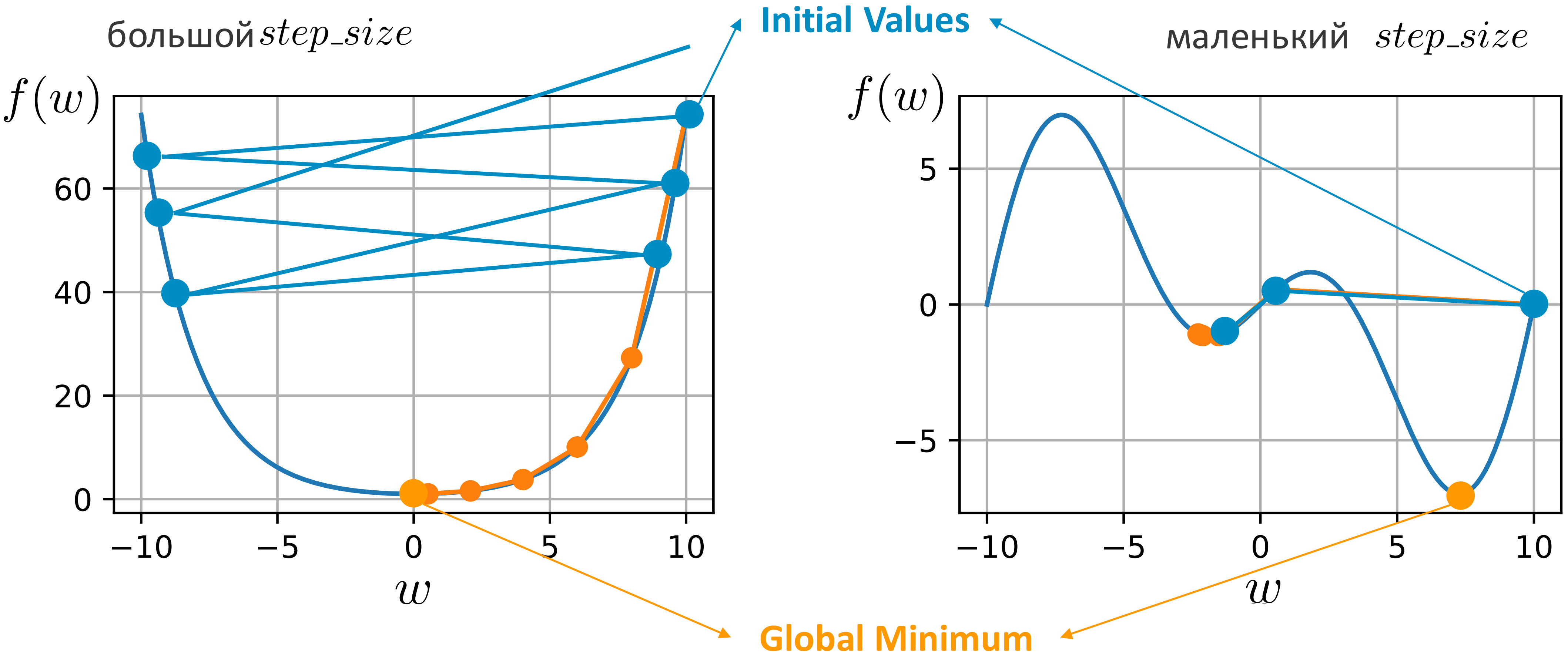
Gradient Descent Method

- **Метод градиентного спуска** использует градиенты для итеративного поиска минимума функции.
- Шаги (пропорциональные размеру градиента) к минимуму в направлении, противоположном градиенту.
- **Gradient Descent Algorithm**:
  - Начать с начальной точки
  - Update: w

$$w_{new} = w_{current} - step\_size * gradient$$

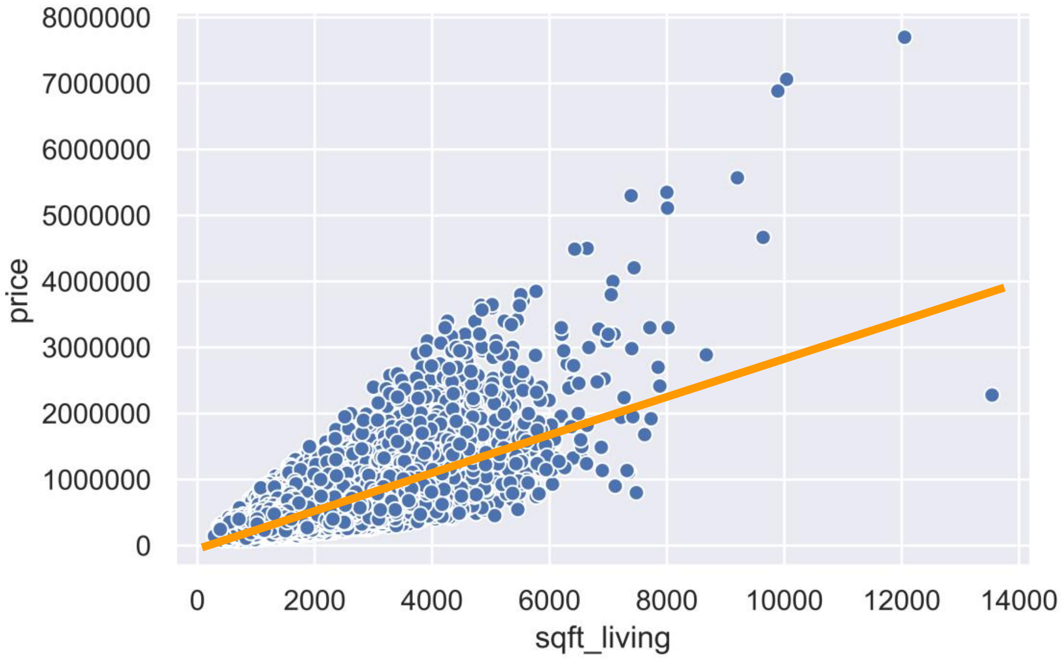


Gradient Descent Method



Regression Models

Linear Regression



Мы используем (линейную) регрессию для прогнозирования **числовых (непрерывных) значений**.

Пример: Как изменение цены дома y (цель, результат) соотносится с его жилой площадью x в квадратных футах (особенность, атрибут)?

$$price = w_0 + w_1 * sqft\_living$$

для $sqft\_living = 6000$

$$price = w_0 + w_1 * 6000$$

Множественная линейная регрессия (Multiple Linear Regression)

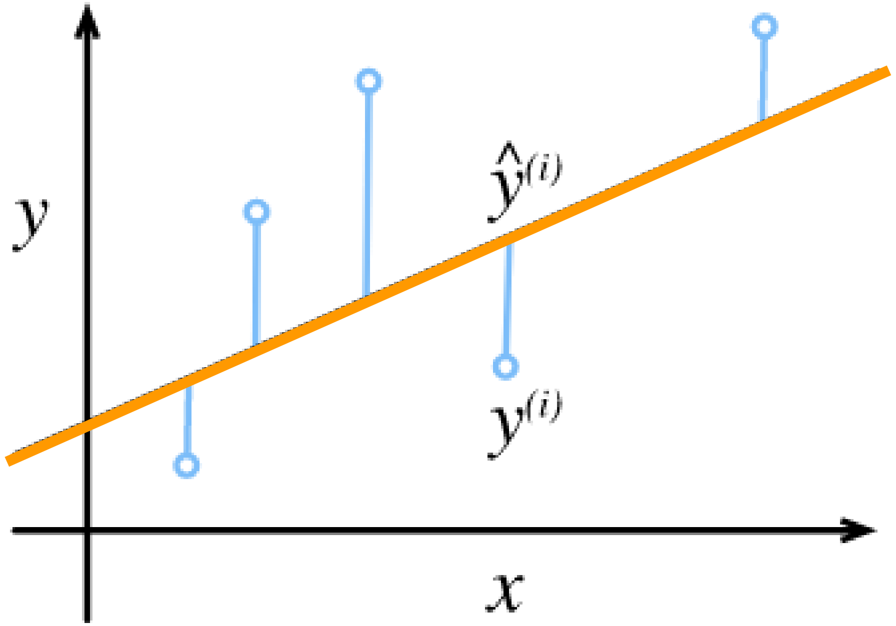
Example: Как цена дома (цель, результат) y изменяется в зависимости от жилой площади (объект) x_1, количества спален (объекта) x_2, почтового индекса x_3 ,...? То есть с использованием **нескольких функций**...

$$price = w_0 + w_1 * sqft\_living + w_2 * bedrooms + w_3 * zip\_code + \ldots$$

Использование уравнения множественной линейной регрессии:

- w_1 предполагая, что все остальные переменные остаются прежними, на сколько увеличение $sqft\_living$ на 1 квадратный фут увеличивает цену недвижимости
- w_2 предполагая, что все остальные переменные останутся прежними, увеличение на 1 спальню $bedrooms$ приведет к увеличению цены и т.д.

Linear Regression



Regression line $y = w_0 + w_1 x$

определяются: w_0 (intercept - отрезок), (slope - наклон) w_1.

Вертикальное смещение для каждой точки x данных от линии - это ошибка между (истинным значением y - меткой) и (предсказанием $\hat{y}$ на основе w_0 w_1).

Лучшая «линия» минимизирует сумму квадратов ошибок (SSE):

$$\sum (y^{(i)} - \hat{y}^{(i)})^2$$

Обучение (Fitting) модели: градиентный спуск

- Для модели **Linear Regression**:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_q x_q \,,$$

с признаками $x_1, \ldots, x_q$, и параметрами/весами $(w_0, w_1, \ldots, w_q) = \mathbf{w}$

- Минимизируем **Mean Squared Error** функцию ошибки (loss, cost):

$$MSE = \frac{1}{n} \sum_{i=0}^{n} (y^{(i)} - \hat{y}^{(i)})^2$$

i : index; n : number of samples

$y^{(i)}$: output; $\hat{y}^{(i)}$: model prediction

- Итеративно обновляем параметры/веса используя **Gradient Descent**:

$$\mathbf{w}_{new} = \mathbf{w}_{current} - step\_size * gradient$$

От регрессии к классификации

Линейная регрессия была полезна при прогнозировании непрерывных значений

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_q x_q$$

Можем ли мы использовать аналогичный подход для решения задач **классификации?**

Самая простая задача классификации - это бинарная классификация, где $y \in \{0, 1\}$.

Examples:

Email: Spam or Not Spam

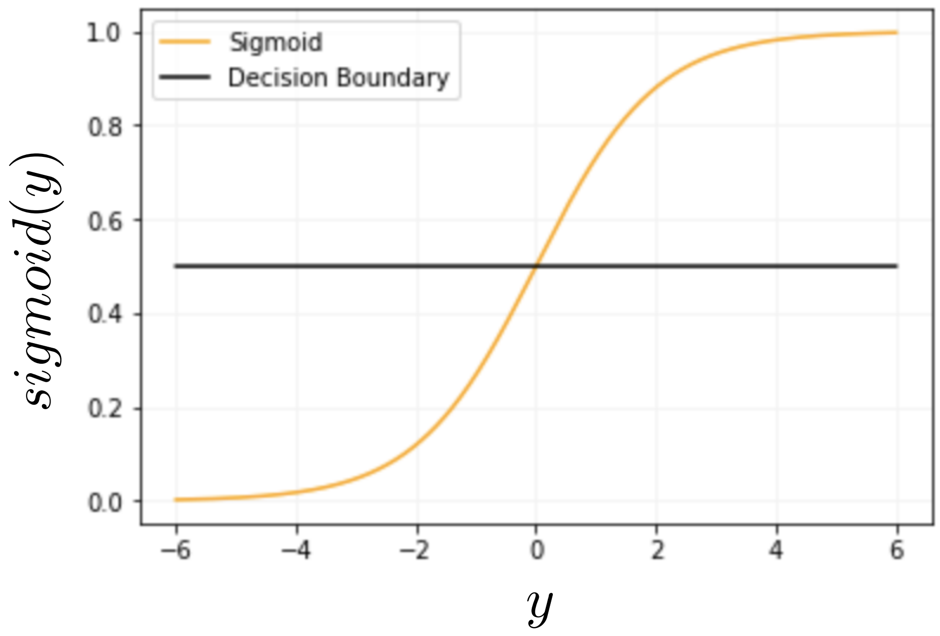
Text: Positive or Negative отношение к продукту

Image: Cat or Not Cat

Логистическая регрессия (Logistic Regression)

Идея: мы можем применить сигмоидальную функцию к

$$y = w_0 + w_1 x_1 + \dots + w_q x_q$$



- **Sigmoid (Logistic) function**

$$sigmoid(y) = \frac{1}{1 + e^{-y}}$$

сглаживает значения y в диапазон 0 –1.

- Можно определить «Границу решения» на 0.5
  - If $sigmoid(y) <$ 0.5, округлить вниз (class 0)
  - if $sigmoid(y) \geq$ 0.5, округлить вверх (class 1)
- Наше **уравнение регрессии** становится:

$$sigmoid(w_0 + w_1 x_1 + \dots + w_q x_q)$$

Log-Loss (Binary Cross-Entropy)

Log-Loss: Числовое значение, которое измеряет качество (производительность) двоичного классификатора, когда выход модели представляет собой вероятность от 0 до 1:

$$LogLoss = -(y * \log(p) + (1 - y) * \log(1 - p))$$

y : true class ∈ {0, 1}, p =$sigmoid(y)$: probability of class, and $\log$: logarithm

- Поскольку выходные данные логистической регрессии находятся в диапазоне от 0 до 1, Log-Loss является подходящей **функцией стоимости** для **логистической регрессии**.
- Чтобы улучшить обучение модели логистической регрессии, минимизируйте Log-Loss.

Log-Loss (Binary Cross-Entropy)

Example: Давайте посчитаем Log-Loss

$$LogLoss = -(y * \log(p) + (1-y) * \log(1-p))$$

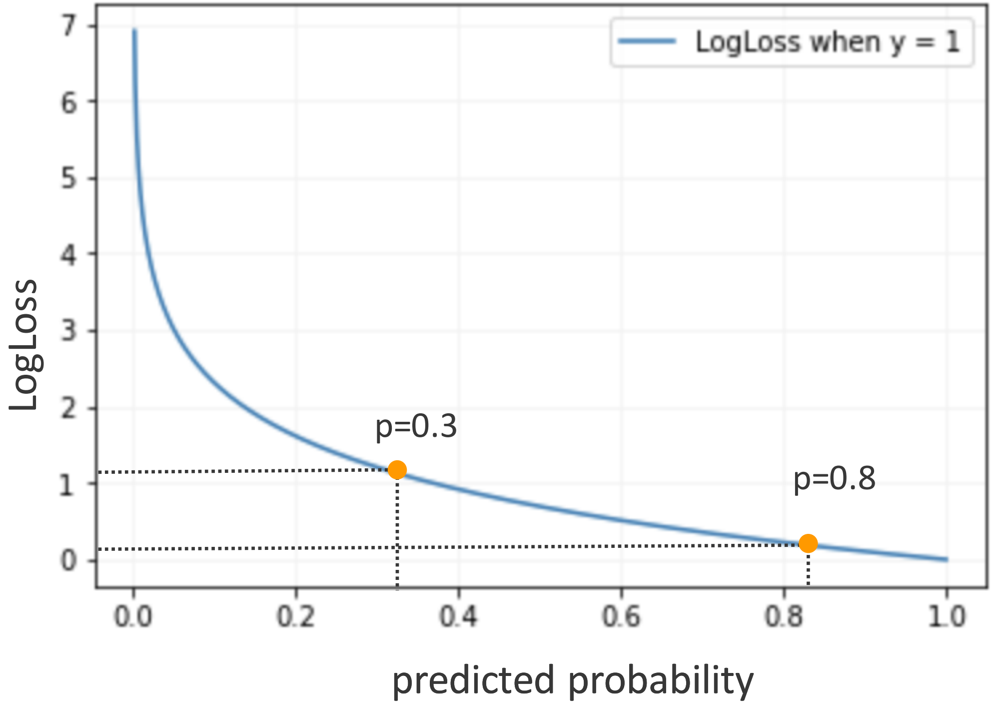
для следующих сценариев:

- y: true class = 1, p = 0.3

$$-(1 * \log(0.3) + (1-1) * \log(0.7)) = \boxed{1.20}$$

- y: true class = 1, p = 0.8

$$-(1 * \log(0.8) + (1-1) * \log(0.2)) = \boxed{0.22}$$



Лучшее предсказание дает меньше значение Log-Loss

Обучение модели: Gradient Descent

- For a **Logistic Regression** model:

$$y = sigmoid(w_0 + w_1 x_1 + ... + w_q x_q), \quad sigmoid(y) = \frac{1}{1 + e^{-y}}$$

с фичами $x_1, ..., x_q$, и параметрами/весами $(w_0, w_1, \ldots, w_q) = \mathbf{w}$

- Минимизируем **LogLoss**:

$$LogLoss = \sum_{i=0}^{n} -(y^{(i)} * \log(p^{(i)}) + (1 - y^{(i)}) * \log(1 - p^{(i)}))$$

i : index; n# samples

$y^{(i)}$: output

$p^{(i)}$: model prediction

- Итеративно обновляем параметры/веса с помощью **Gradient Descent**:

$$\mathbf{w}_{new} = \mathbf{w}_{current} - step\_size * gradient$$

Regularization

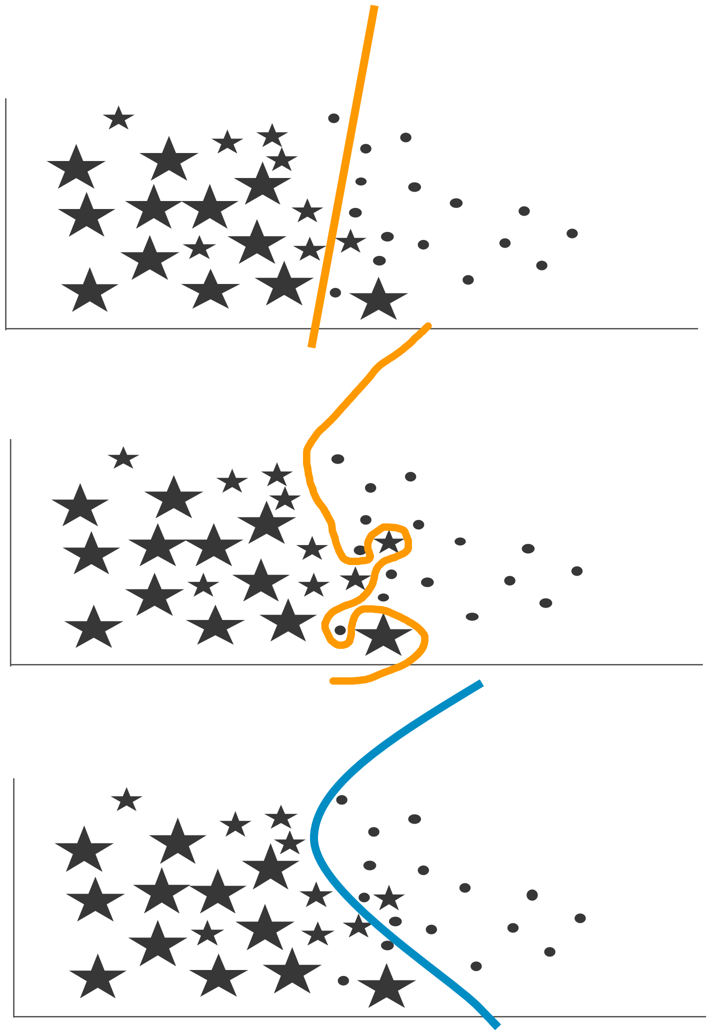
Regularization

Недообучение (Underfitting): Слишком простая модель, мало функций, маленькие веса, слабое обучение.

Переобучение (Overfitting): Слишком сложная модель, слишком много функций, большие веса, слабое **обобщение**.

«Хорошая модель»: компромисс между уровнем обученности и сложностью модели (уменьшение числа признаков, уменьшение веса).

Регуляризация делает и то, и другое: штрафует большие веса, иногда полностью сводя их к нулю!



Regularization

- Настройте **сложность модели**, добавив штрафной балл **penalty** за сложность к функции стоимости (подумайте о функции ошибок, минимизируя ее до наилучшего соответствия к цели!):

$$C_{regularized}(w) = C(w) + \alpha * penalty(w)$$

- Настройка степени регуляризации с помощью параметра, **α**
- Standard regularization types:
  - **L2 regularization (Ridge):** $penalty(w) = \|w\|_2^2 = \sum w_i^2$ (L2: популярный выбор)
  - **L1 regularization (LASSO):** $penalty(w) = \|w\|_1 = \sum |w_i|$ (L1: полезно для сокращения числа функции, так как большинство весов сжимаются до 0 - разреженность)
  - L2 и L1 (**ElasticNet**)

- **Примечание: важно сначала масштабировать элементы!**

Regression in sklearn

LinearRegression: **sklearn** Linear Regression (and regularization)

LinearRegression()

Ridge*(alpha=1.0),* **RidgeCV***(alpha=1.0, cv=5)*

Lasso*(alpha=1.0),* **LassoCV***(alpha=1.0, cv=5)*

ElasticNet*(alpha=1.0, l1_ratio=0.5),* **ElasticNetCV***(cv=5)*

LogisticRegression: **sklearn** Logistic Regression (and regularization)

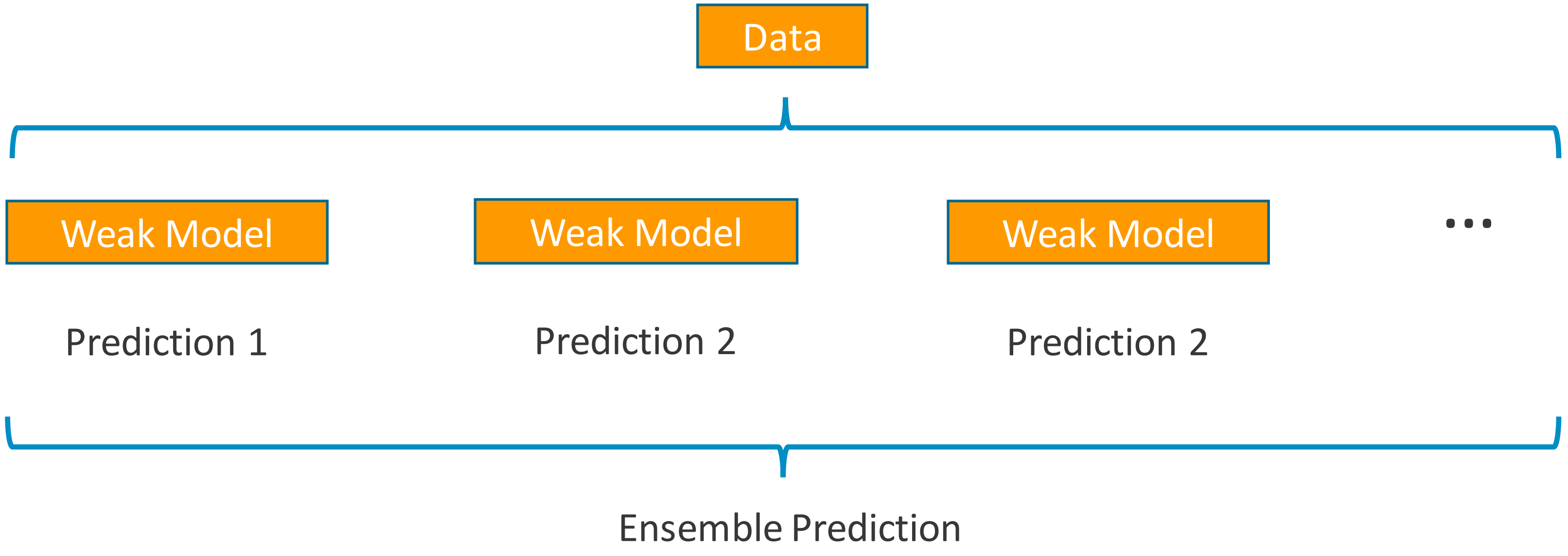
LogisticRegression(*penalty='l2', C=1.0, l1_ratio=None)*

LogisticRegressionCV(*penalty='l2', C=1.0, l1_ratio=None, cv=5)*

Ensemble Methods: Boosting

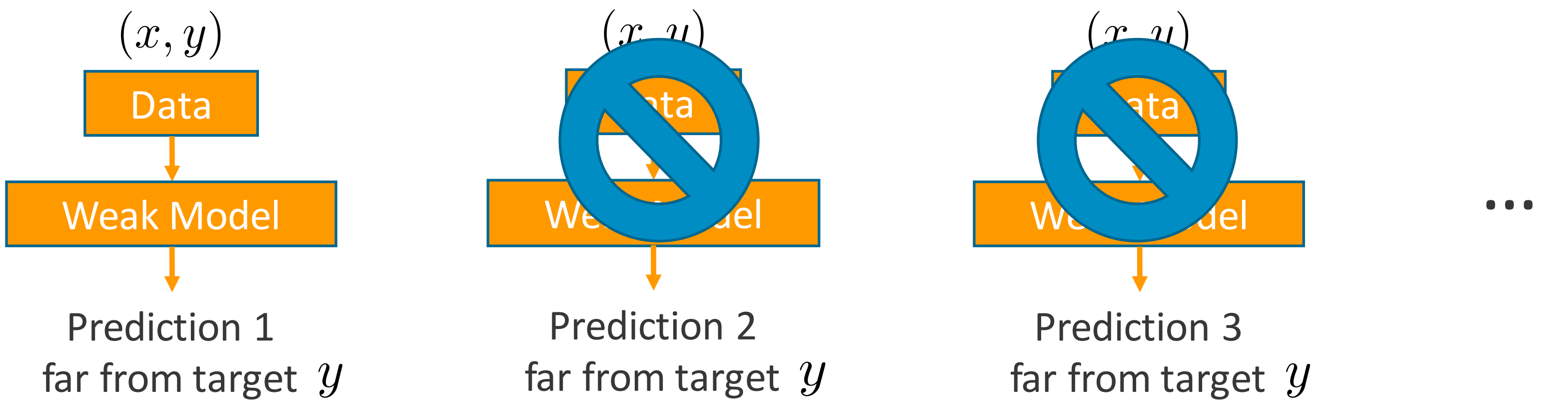
Boosting

Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



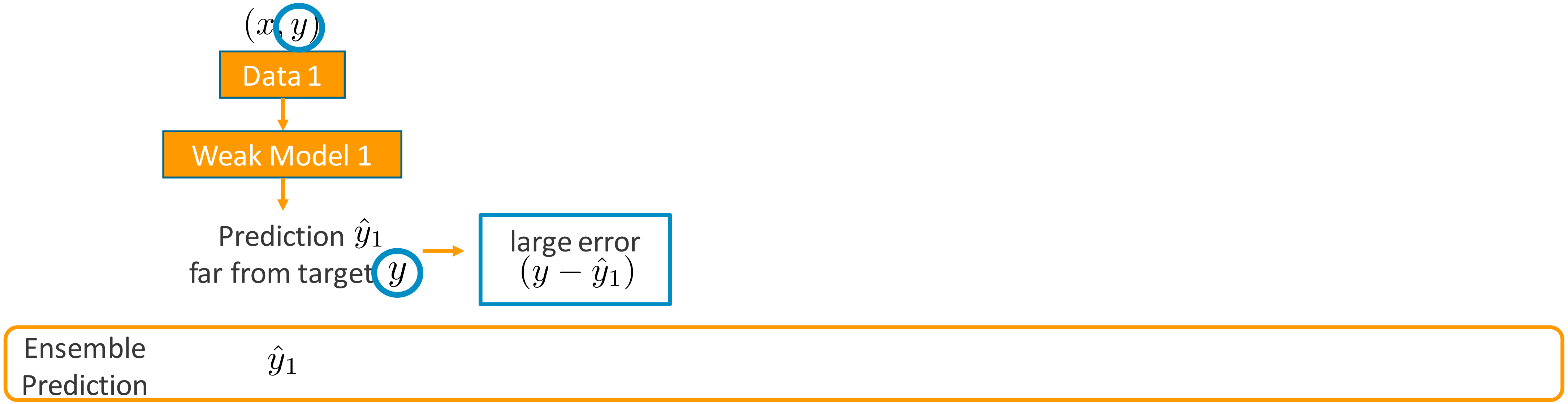
Boosting

Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



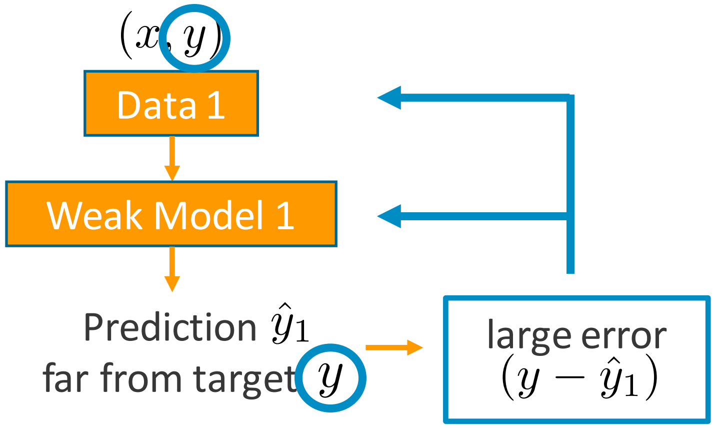
Boosting

Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



Boosting

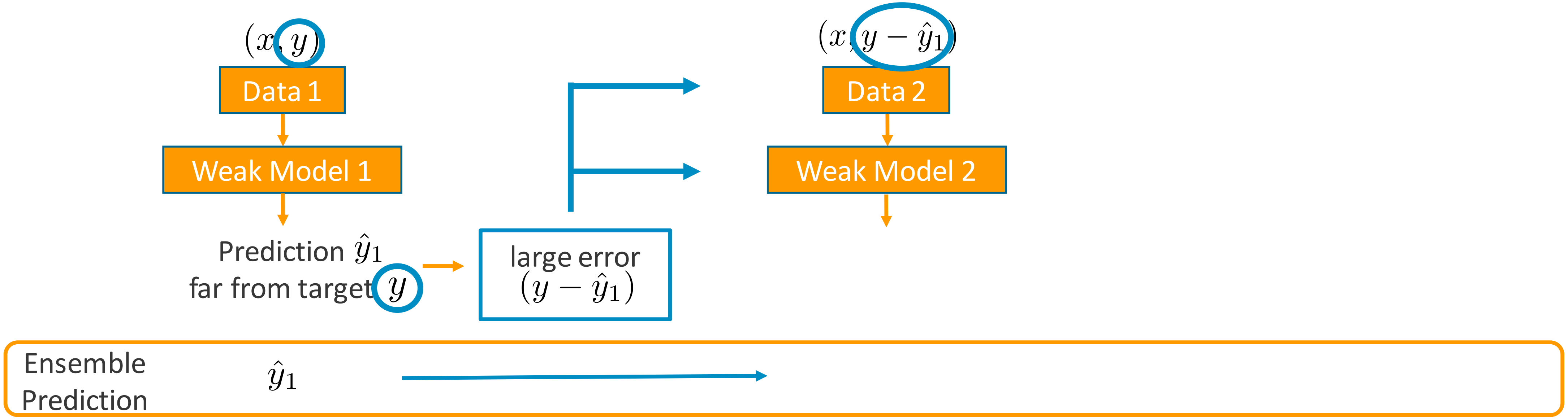
Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



Ensemble Prediction $\hat{y}_1$

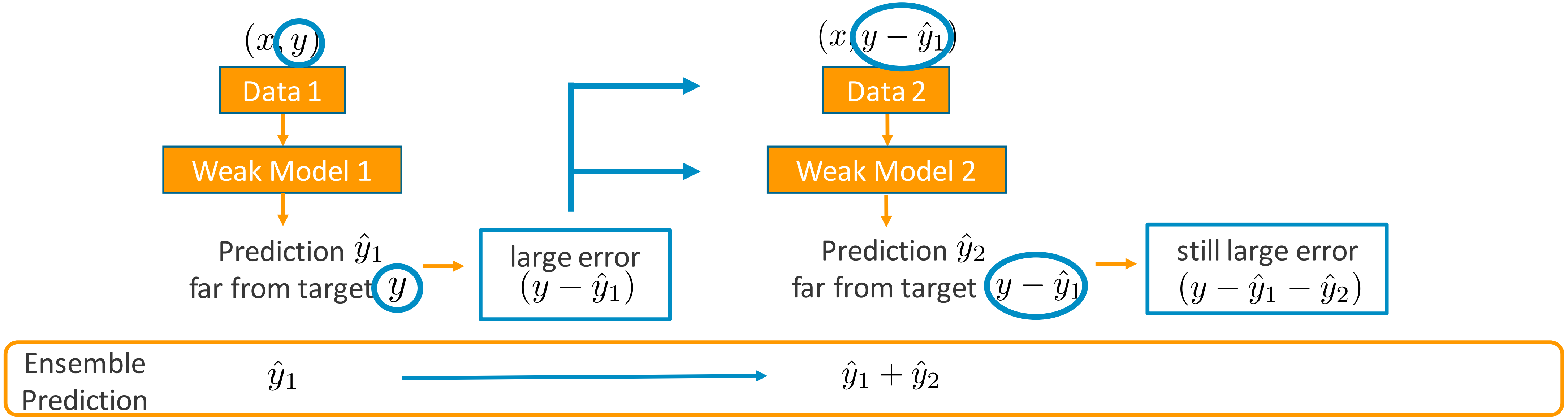
Boosting

Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



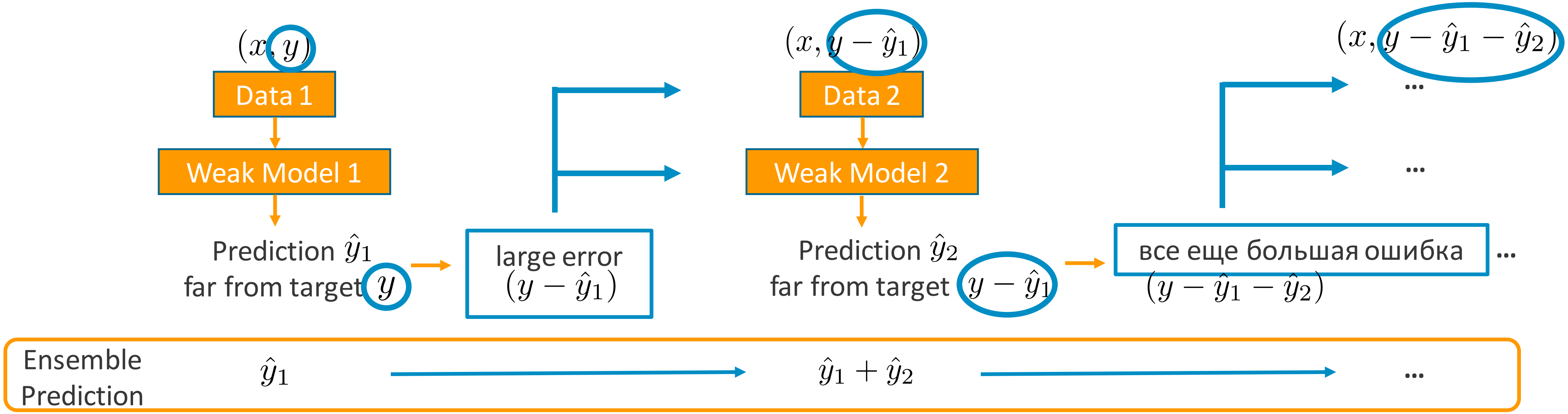
Boosting

Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



Boosting

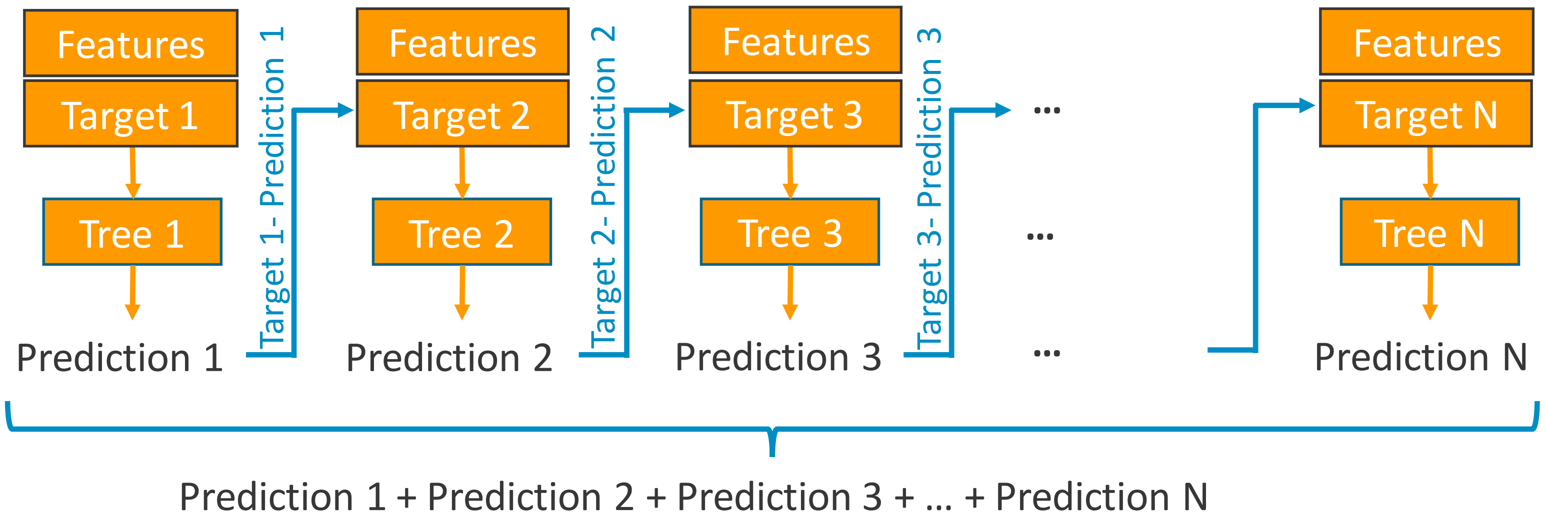
Boosting method: построить несколько слабых моделей последовательно, каждая последующая модель пытается **повысить** (улучшить) производительность всего ансамбля, преодолевая/уменьшая ошибки предыдущей модели.



Gradient Boosting Machines (GBM)

Gradient Boosting Machines (GBM): Бустинг деревьев решений (Boosting trees)

- Обучите слабую модель на заданном датасете и сделайте с ее помощью прогнозы.
- Итеративно создайте новую модель, чтобы научиться предсказывать ошибки прогнозирования предыдущей модели (используйте предыдущую ошибку прогнозирования в качестве новой цели!)



Gradient Boosting in Python

- **sklearn** GBM algorithms:
  - GradientBoostingClassifier (Regressor)
  - HistGradientBoostingClassifier (Regressor) – faster, experimental
- Дополнительные сторонние библиотеки обеспечивают эффективные с точки зрения вычислений альтернативные реализации GBM с лучшими результатами:
  - XGBoost (E**x**treme **G**radient **Boost**ing): эффективные вычисления, память
  - LightGBM: намного быстрее
  - CatBoost (**Cat**egory Gradient **Boost**ing): быстро, поддерживает категории

Gradient Boosting in sklearn

GradientBoostingClassifier: sklearn's Gradient Boosting classifier (есть также версия Regressor) - .fit(), .predict()

GradientBoostingClassifier*(n_estimators=100, learning_rate = 0.1, min_samples_split=2, min_samples_leaf=1, max_depth=3)*

Полный интерфейс больше.

Обратите внимание на сочетание параметров, связанных с повышением уровня (скорости обучения – величина шага), и параметров, связанных с деревом.

Gradient Boosting in sklearn

HistGradientBoostingClassifier: sklearn's Light GBM classifier (есть также версия Regressor), .fit(), .predict()

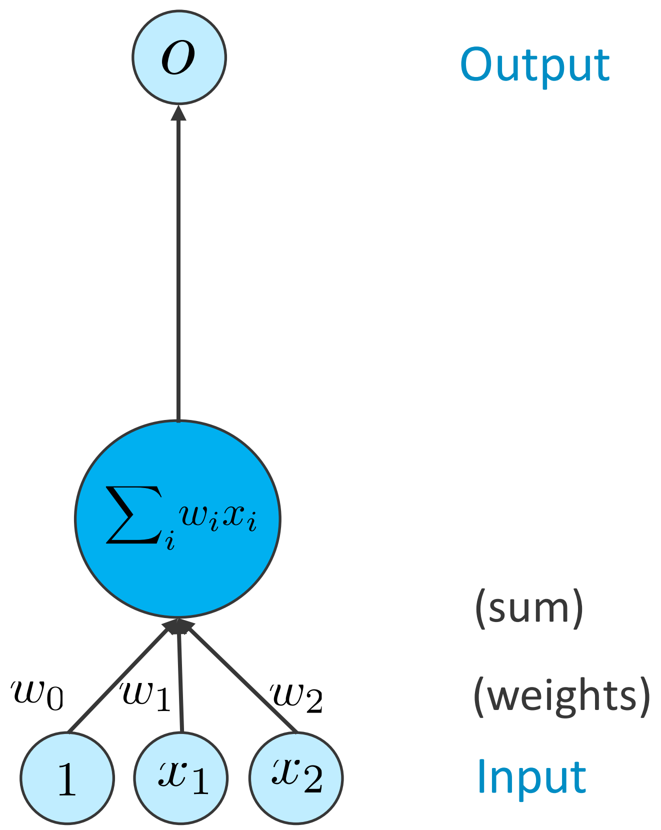
from sklearn.experimental import enable_hist_gradient_boosting

HistGradientBoostingClassifier*(max_iter=100, learning_rate = 0.1, max_leaf_nodes=31, min_samples_leaf=20, max_depth=None)*

Полный интерфейс больше.

Neural Networks

Возвратимся к модели регрессии

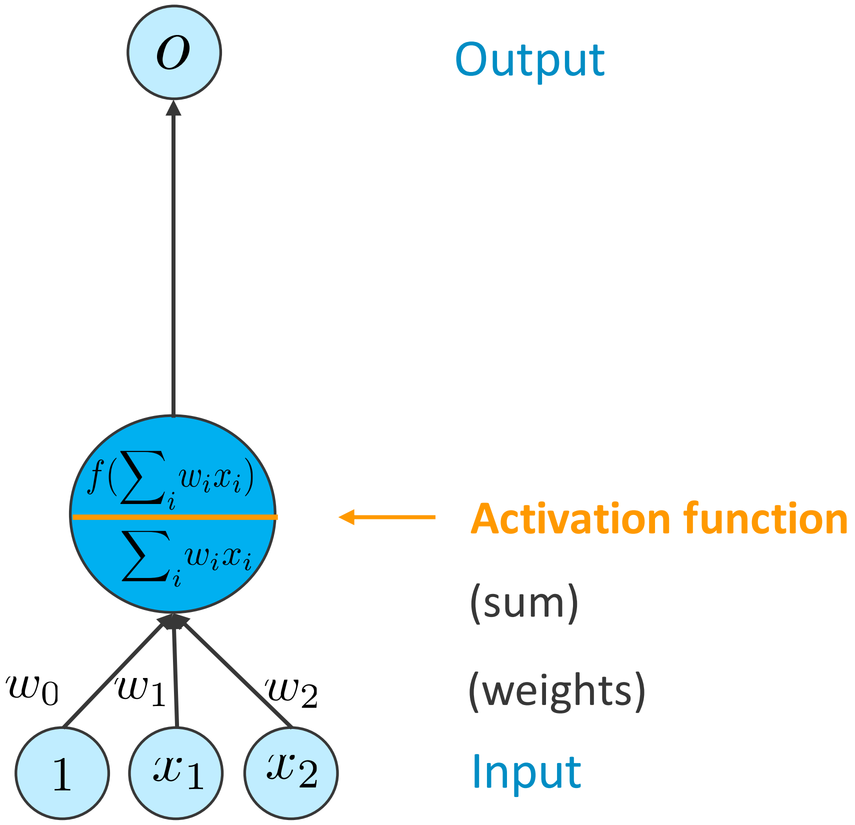


Linear Regression*: Дано $\{x_i\}$, предсказать y :

$$y = w_0 + w_1 x_1 + \ldots + w_q x_q$$

* В основном предполагая, что выход зависит только от взаимодействий входов в первой степени.

Возвратимся к модели регрессии



Linear Regression*: Дано $\{x_i\}$, предсказать y:

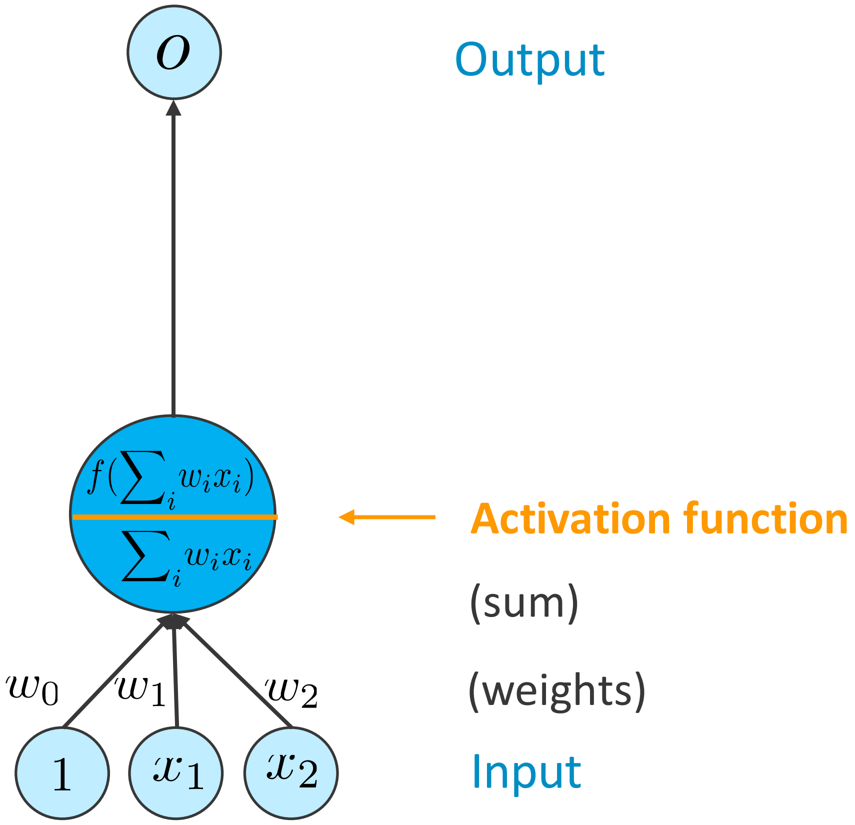
$$y = f(w_0 + w_1 x_1 + \ldots + w_q x_q)$$

где f **линейная функция**:

$$f(x) = x$$

* Линейная функция активации

Возвратимся к модели регрессии



Logistic Regression*: Дано $\{x_i\}$, предсказать y , где $y \in \{0, 1\}$:

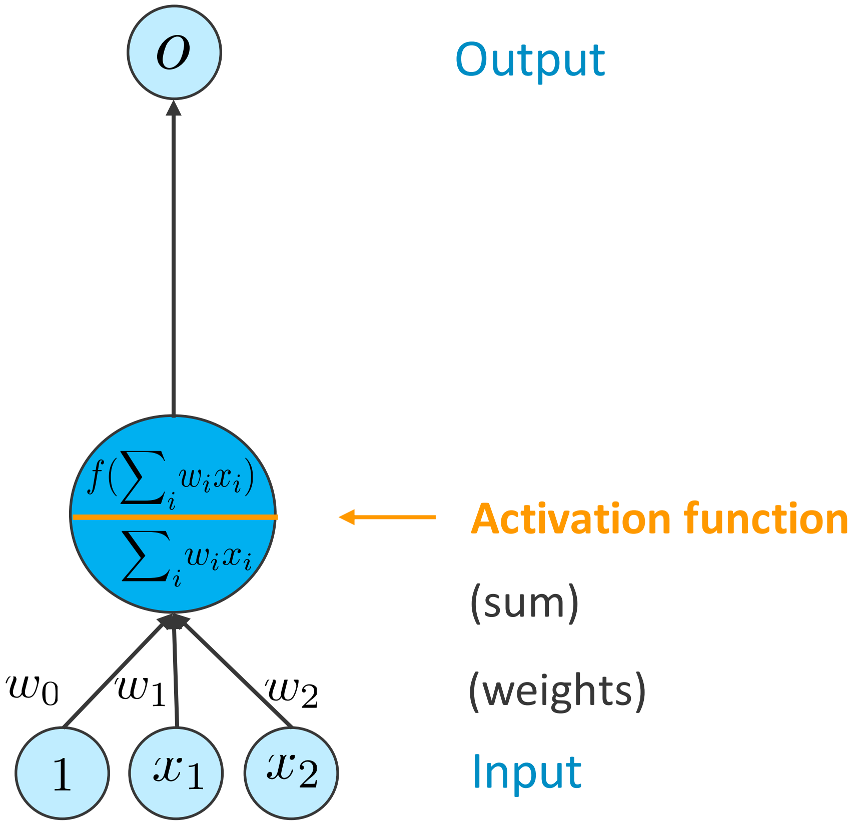
$$y = f(w_0 + w_1 x_1 + \ldots + w_q x_q)$$

где f **logistic function**:

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

* Нелинейная функция активации/бинарный классификатор

Perceptron (Rosenblatt, 1957)

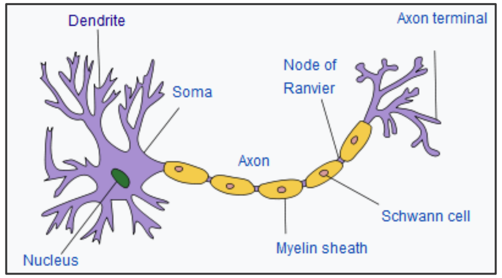


Perceptron*: Given $\{x_i\}$, predict y, where $y \in \{-1, +1\}$

$$y = f(w_0 + w_1 x_1 + \ldots + w_q x_q)$$

where f is the **step function**:

$$f(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$



* Non-linear activation function / binary classifier

Теорема Колмогорова

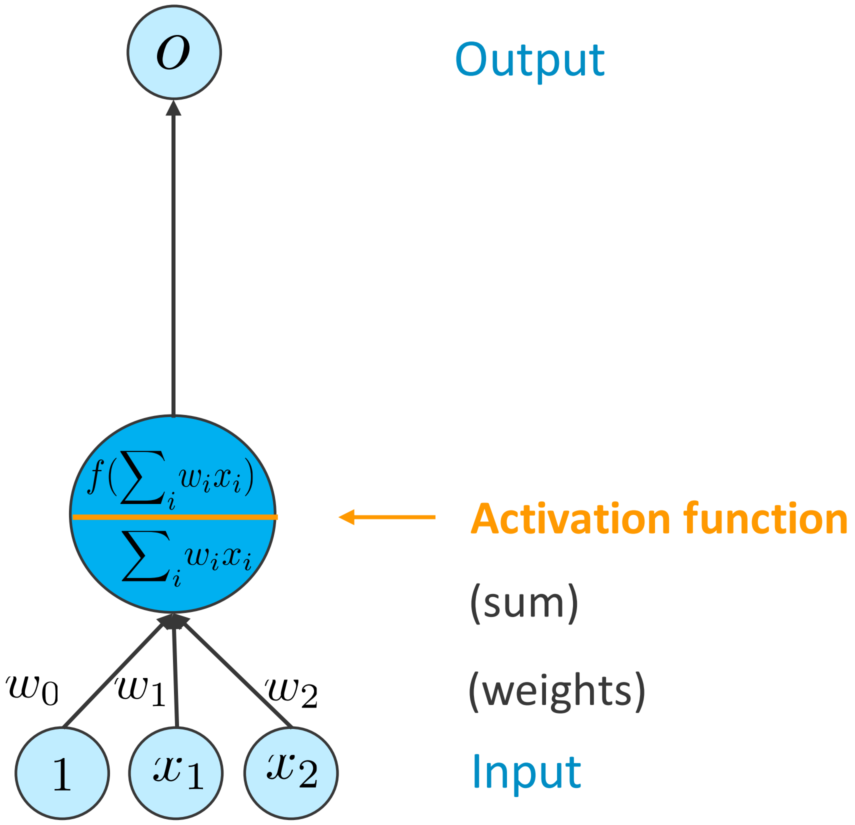
Трина́дцатая пробле́ма Ги́льберта — одна из 23 задач, которые Давид Гильберт предложил 8 августа 1900 года на II Международном конгрессе математиков. Она была мотивирована применением методов номографии к вычислению корней уравнений высоких степеней, и касалась представимости функций нескольких переменных, в частности, решения уравнения седьмой степени как функции от коэффициентов, в виде суперпозиции нескольких непрерывных функций двух переменных.

Проблема была решена В.И. Арнольдом совместно с **А.Н. Колмогоровым**, доказавшими, что любая непрерывная функция любого количества переменных представляется в виде суперпозиции *непрерывных* функций одной и двух переменных:

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{q,p}(x_p) \right)$$

где функции Φ и ψ - непрерывные.
Нейронные сети могут с любой точностью имитировать любой непрерывный автомат. Для нейронных сетей полученные результаты означают, что от функции активации нейрона требуется только нелинейность

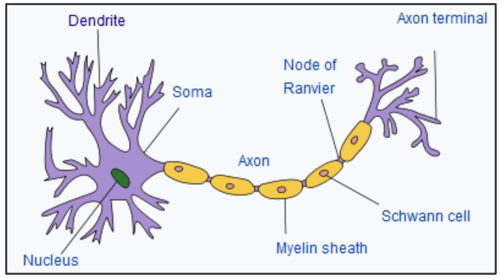
Искусственный нейрон (Artificial Neuron)



Artificial Neuron*: Given $\{x_i\}$, predict y:

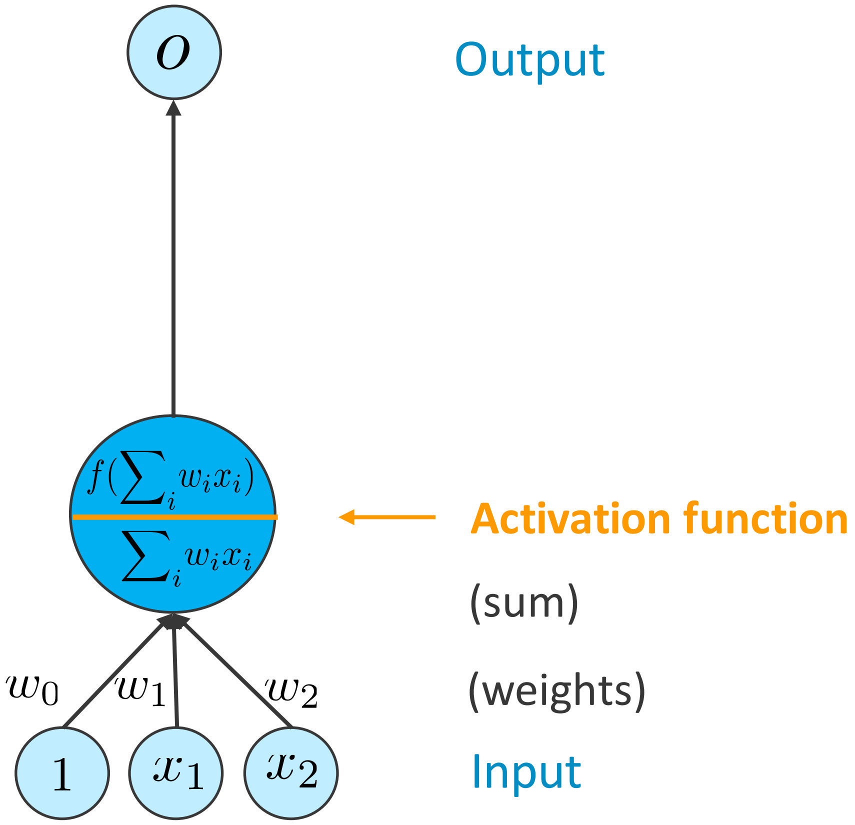
$$y = f(w_0 + w_1 x_1 + \ldots + w_q x_q)$$

where f is a **nonlinear activation function** (sigmoid, tanh, ReLU, …)



* Similar to how neurons in the brain function

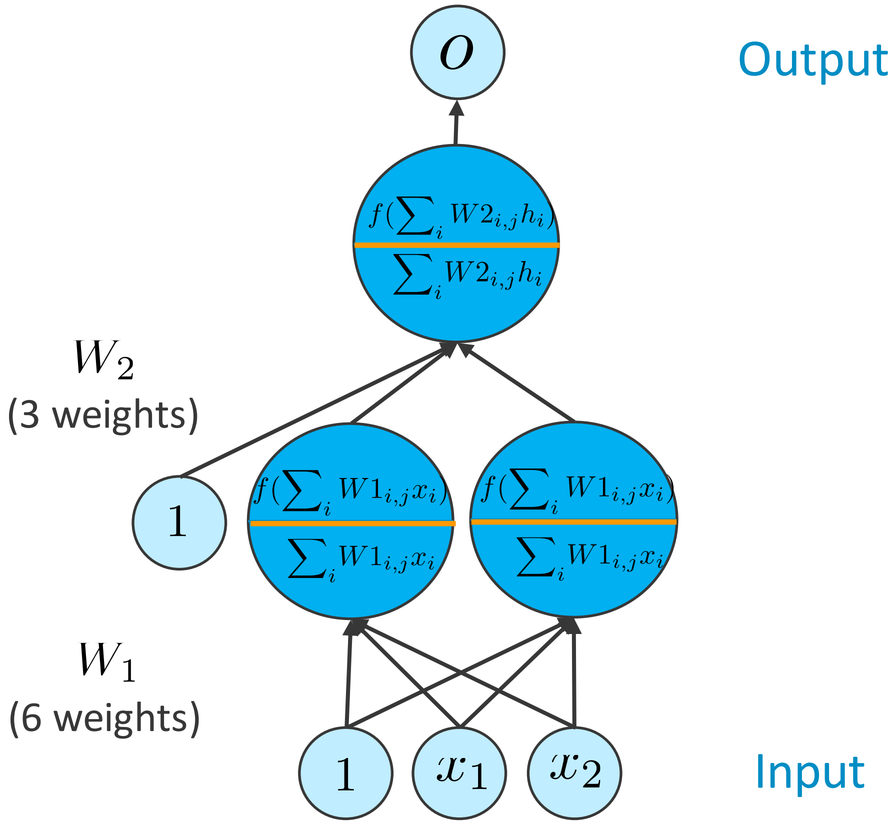
Artificial Neuron



Artificial Neuron: Улавливает в основном линейные взаимодействия в данных.

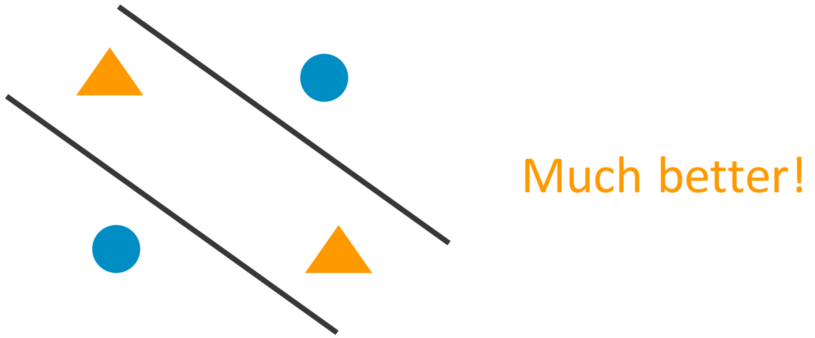
Вопрос: Можем ли мы использовать аналогичный подход для обнаружения нелинейных взаимосвязей в данных?

Neural Network/Multilayer Perceptron

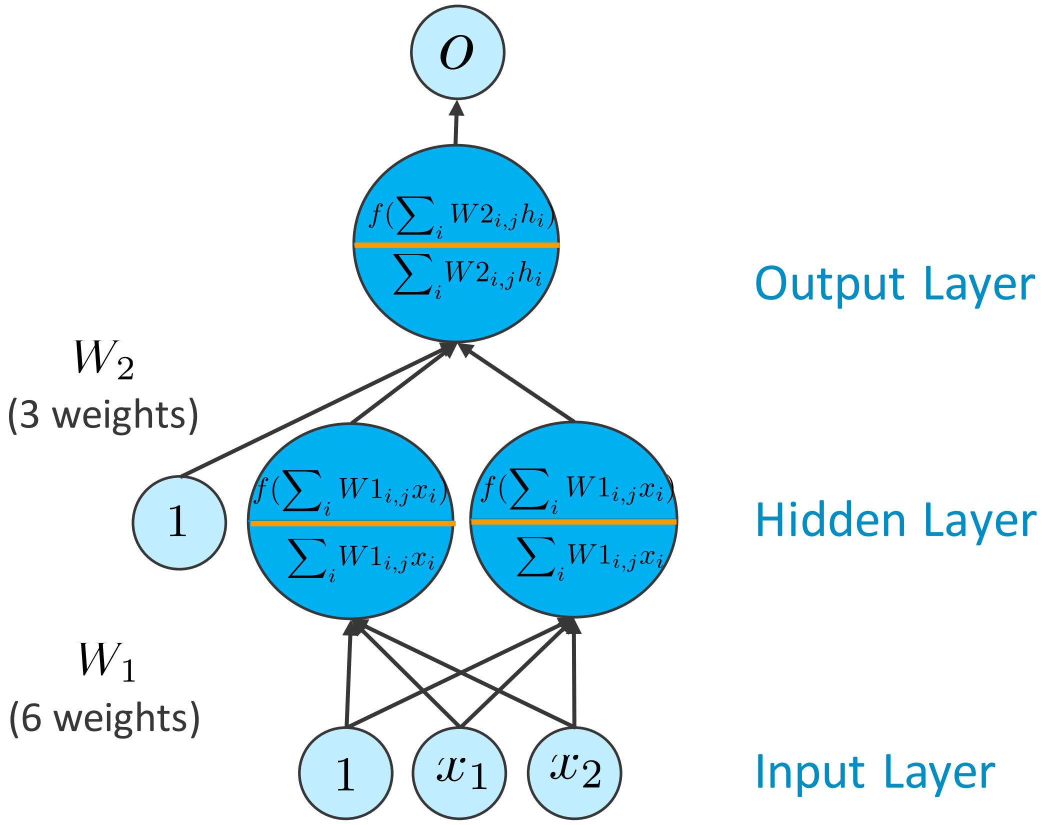


Artificial Neuron: Улавливает в основном линейные взаимодействия в данных.

Вопрос: Можем ли мы использовать аналогичный подход для обнаружения нелинейных взаимосвязей в данных?



Neural Network/Multilayer Perceptron

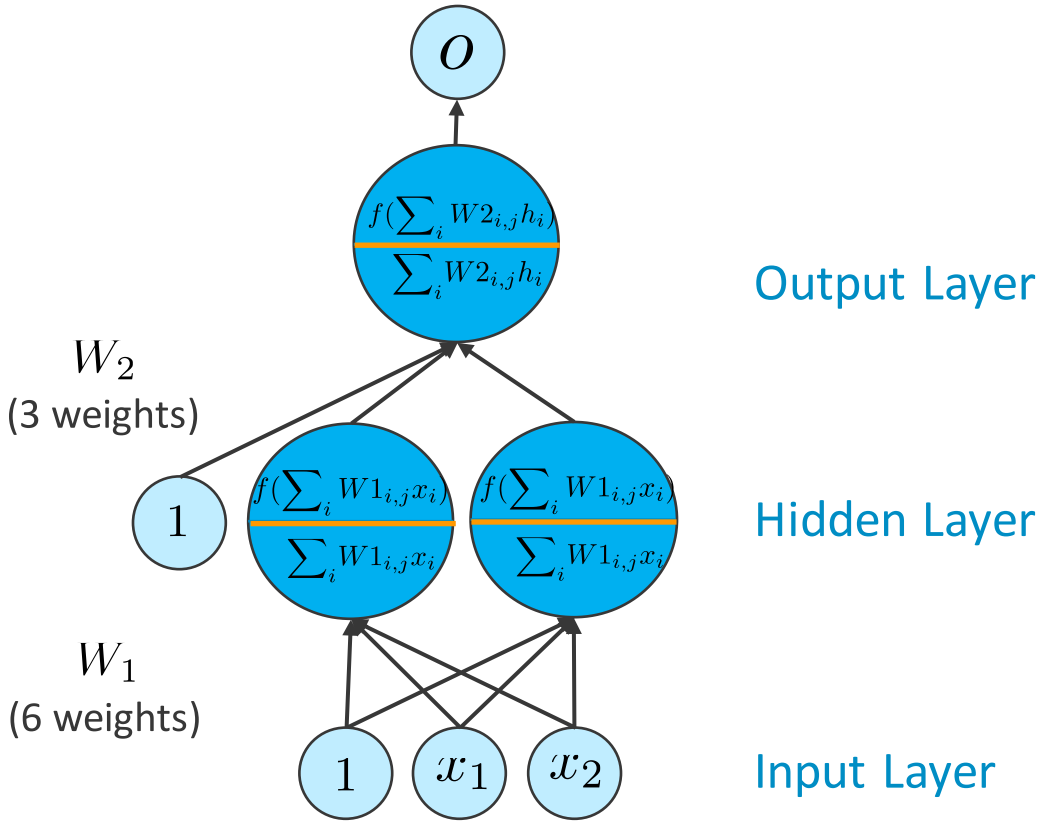


Artificial Neuron: Улавливает в основном линейные взаимодействия в данных.

Вопрос: Можем ли мы использовать аналогичный подход для обнаружения нелинейных взаимосвязей в данных?

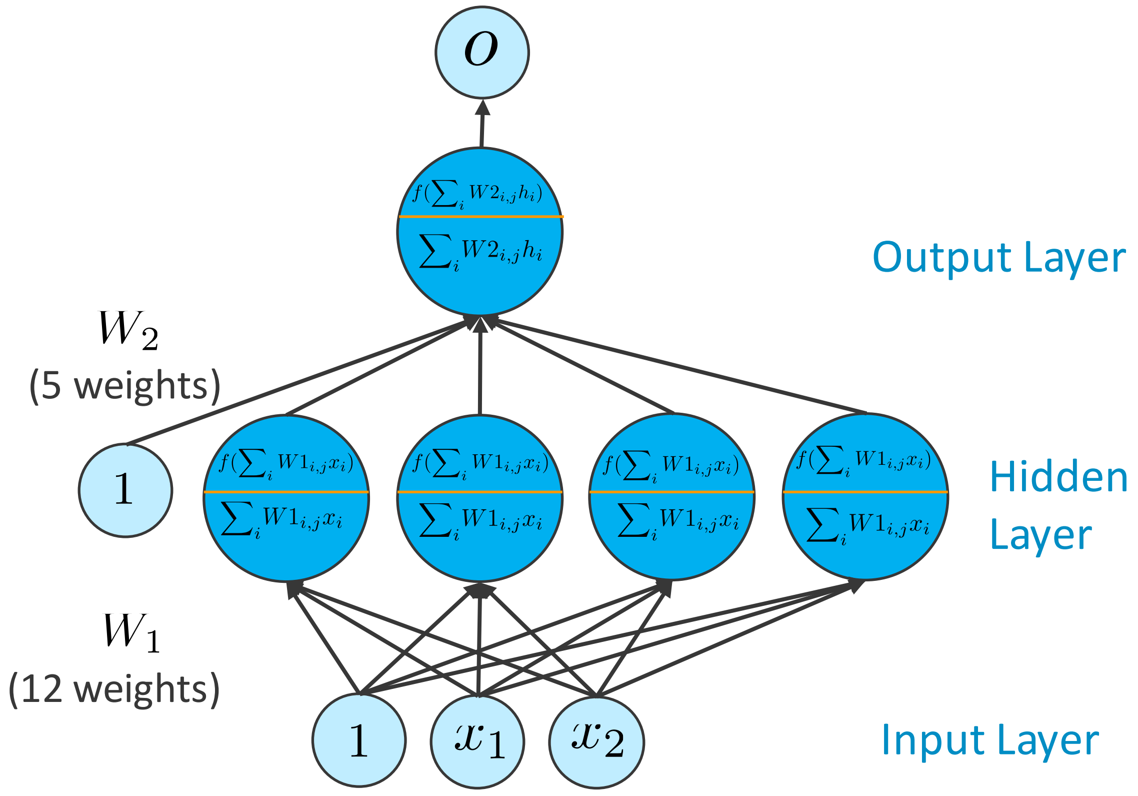
Neural Network/Multilayer Perceptron (MLP): используйте больше искусственных нейронов, объединяете их в **слой**!

Neural Network/Multilayer Perceptron



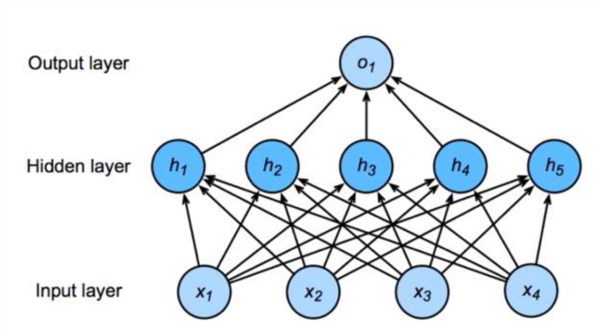
- Нейронная сеть, состоящая из входного, скрытого и выходного слоев
- Каждый слой связан со следующим слоем
- Функция активации применяется к каждому скрытому слою (и выходному слою)

Neural Network/Multilayer Perceptron

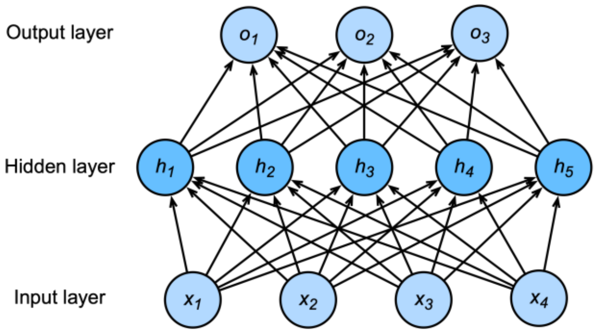


- Нейронная сеть, состоящая из входного, скрытого и выходного слоев
- Каждый слой связан со следующим слоем
- Функция активации применяется к каждому скрытому слою (и выходному слою)

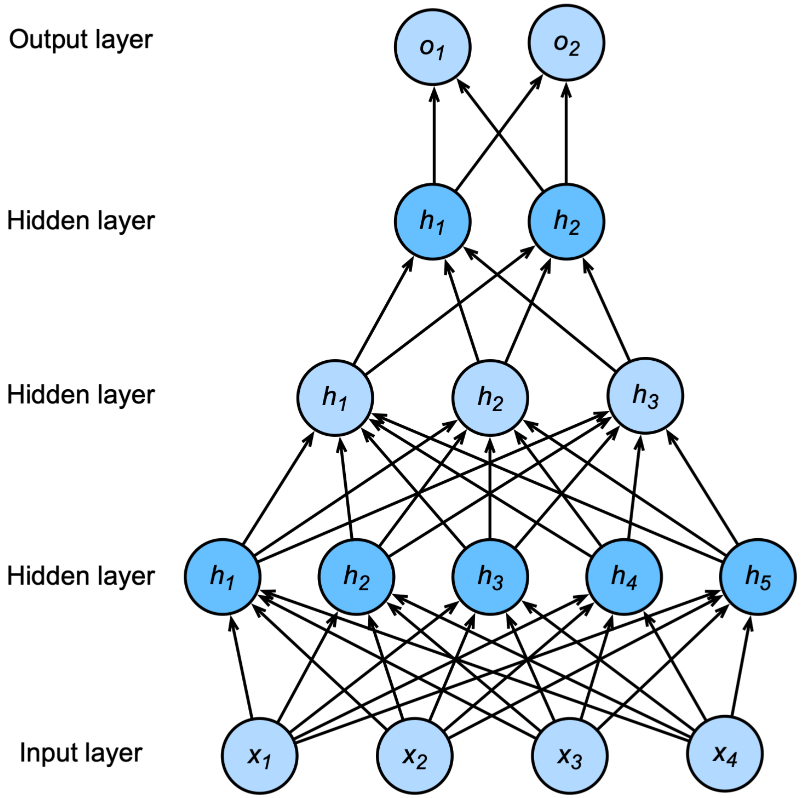
Neural Networks



MultiLayer Network: Two layers (one hidden layer, output layer), with five hidden neurons in the hidden layer, and one output neuron.

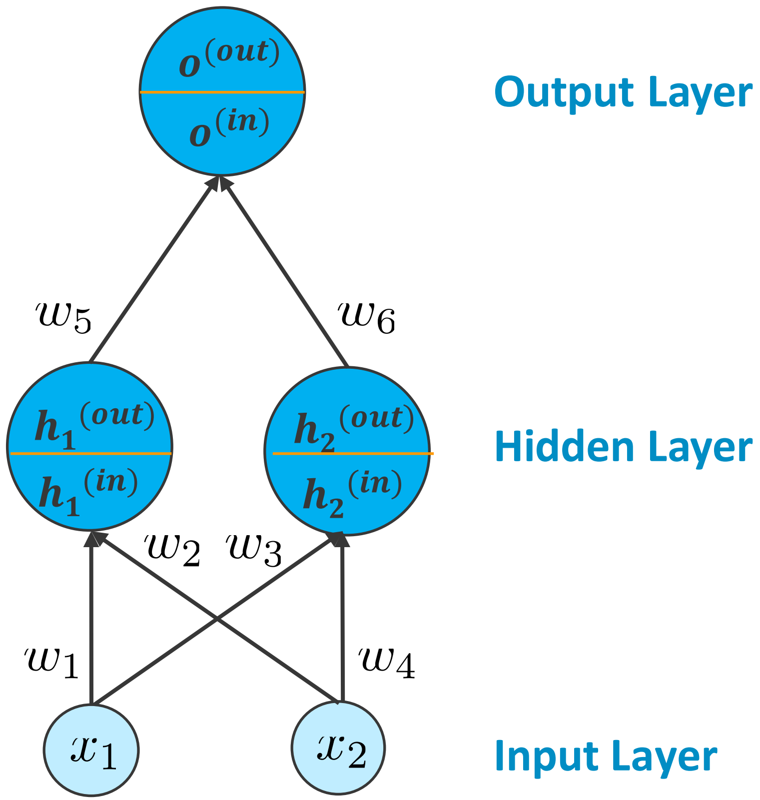


MultiLayer Network: Two layers (one hidden layer, output layer), with five hidden neurons in the hidden layer, and three output neurons.



MultiLayer Network: Four layers (three hidden layer, output layer), with five-three-two hidden neurons in the hidden layers, and two output neurons.

Создание и обучение нейронной сети (Build and Train a Neural Network)

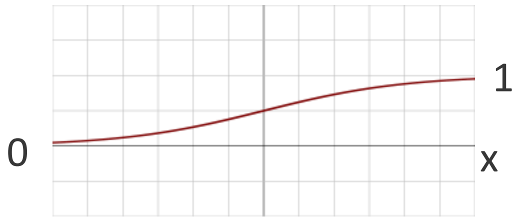
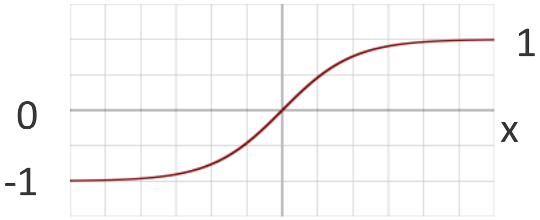
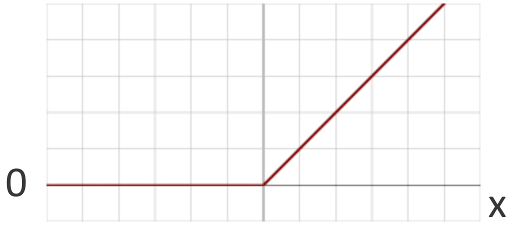


Построим нейронную сеть для задачи бинарной классификации:

- (без смещения (bias) – для простоты)
- 2 inputs: x_1 = 0.5 and x_2 = 0.1
- 1 hidden layer with 2 neurons
- 1 output neuron in the output layer

Activation Functions

- **“Как перейти от ввода линейно взвешенной суммы к нелинейному выводу?”**

Name	Plot	Function	Description
Logistic (sigmoid)		$f(x) = \dfrac{1}{1+e^{-x}}$	Самая распространенная функция активации. Выход находится в диапазоне (0,1).
Hyperbolic tangent (tanh)		$f(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	Выход находится в диапазоне (-1, 1).
Выпрямленный линейный блок (ReLU)		$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$	Популярная функция активации. Все, что меньше 0, приводит к нулевой активации.

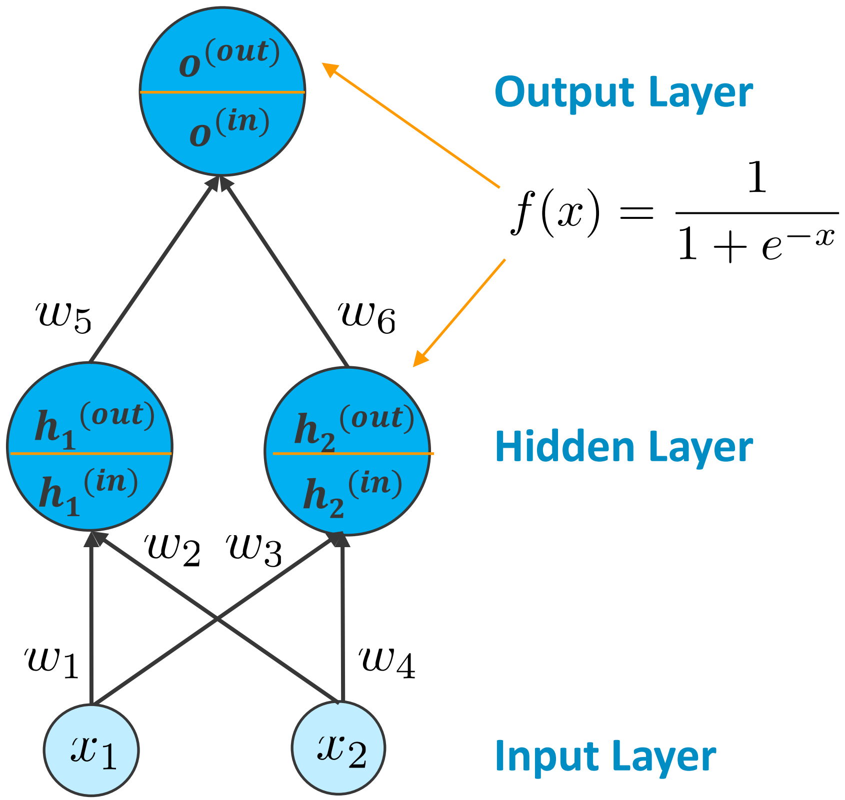
Также важны производные этих функций (т. Колмогорова и для реализации градиентного спуска).

Output Activations/Functions

- “Как вывести / спрогнозировать результат”

Problem	Description	Name	Function
Binary classification	• Вероятность выхода для каждого класса в (0,1)	Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Multi-class classification	• Вероятность выхода для каждого класса в (0,1) • Сумма выходных данных должна быть 1 (распределение вероятностей) • Обучение способствует повышению значений целевого класса, снижению других	Softmax	$f(x_i) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$
Regression		Linear/ ReLU	$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases}$

Создание и обучение нейронной сети (Build and Train a Neural Network)



Мы строим нейронную сеть для задачи бинарной классификации с:

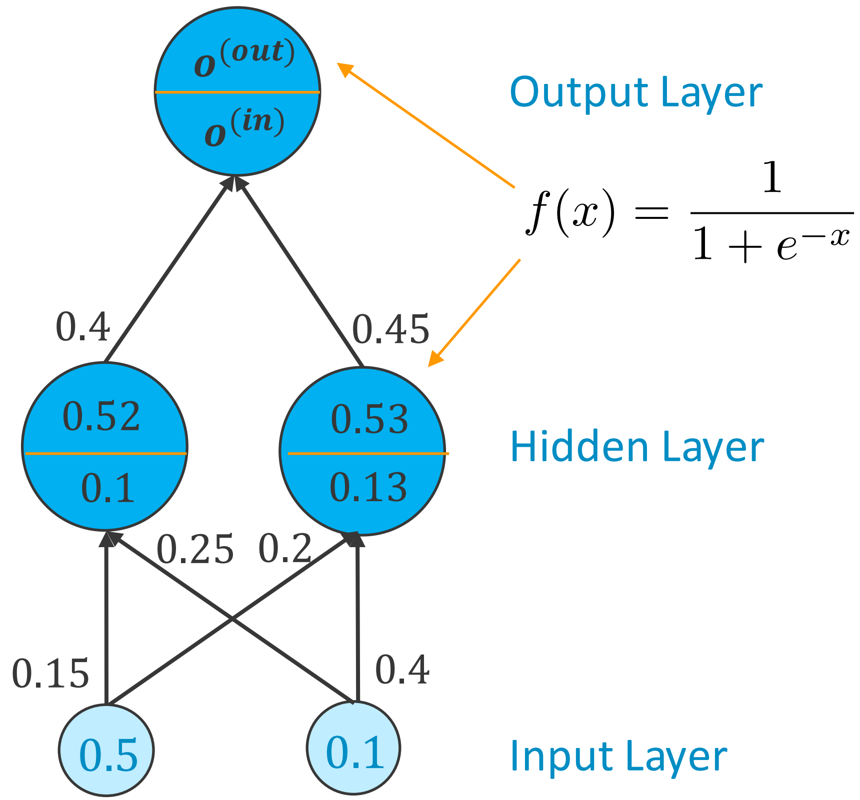
- (без смещения (bias) - для простоты)
- 2 inputs: x_1 = 0.5 and x_2 0.1
- 1 hidden layer with 2 neurons
- 1 output neuron in the output layer
- Все нейроны имеют sigmoid activation function: $$f(x) = \frac{1}{1+e^{-x}}$$

$$\operatorname{th} x = \frac{\operatorname{sh} x}{\operatorname{ch} x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$\sigma'(x) = (1+\sigma(x)) \cdot (1-\sigma(x))$ — для гиперболического тангенса

$\sigma'(x) = \sigma(x) \cdot (1-\sigma(x))$ — для логистической функции

Прямое распространение (Forward Pass)



$$w_1 = 0.15, w_2 = 0.25, w_3 = 0.2, w_4 = 0.3,$$
$$w_5 = 0.4, w_6 = 0.45:$$

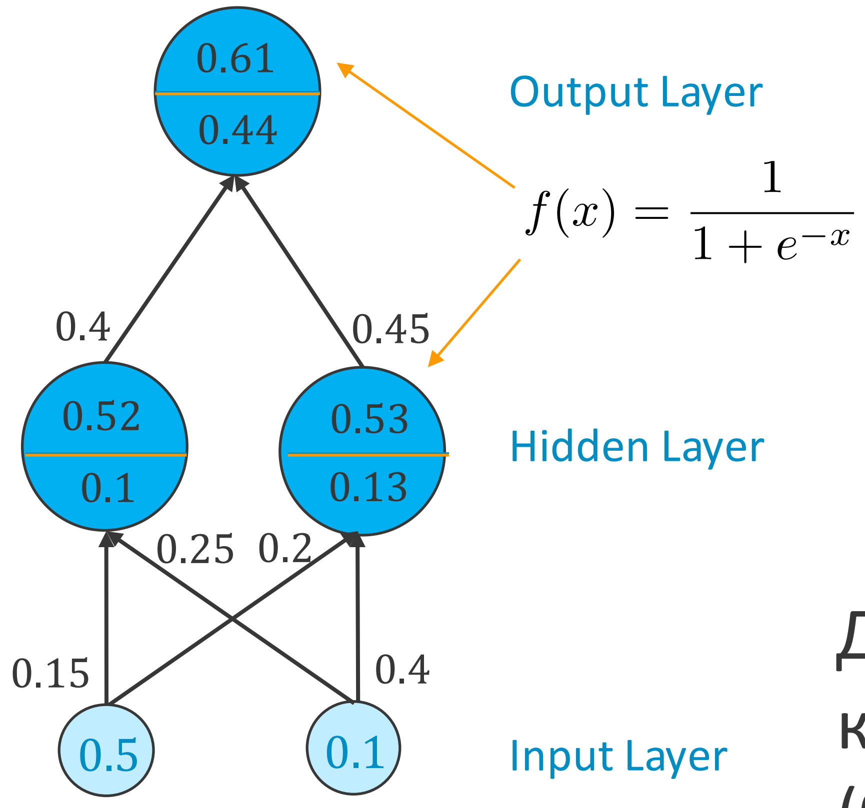
$$\begin{aligned} h_1^{(in)} &= w_1 * x_1 + w_2 * x_2 \\ &= 0.15 * 0.5 + 0.25 * 0.1 = 0.1 \end{aligned}$$

$$h_1^{(out)} = \frac{1}{1+e^{-h_1^{(in)}}} = \frac{1}{1+e^{-0.1}} = 0.52$$

По аналогии,

$$h_2^{(in)} = 0.13, h_2^{(out)} = 0.53$$

Прямое распространение (Forward Pass)



$$w_1 = 0.15, w_2 = 0.25, w_3 = 0.2, w_4 = 0.3,$$
$$w_5 = 0.4, w_6 = 0.45 :$$

$$o^{(in)} = w_4 * h_1^{(out)} + w_5 * h_2^{(out)}$$
$$= 0.4 * 0.52 + 0.45 * 0.53 = 0.44$$

$$o^{(out)} = \frac{1}{1+e^{-o^{(in)}}} = \frac{1}{1+e^{-0.44}} = 0.61$$

Для двоичной классификации мы бы классифицировали эту точку входных данных (0,5, 0,1) как класс 1 (как 0,61> 0,5).

Cost Functions

- “Как сравнить результаты с истинным значением?”

Problem	Name	Function	Notes
Binary classification	Cross entropy for logistic	$C = -\frac{1}{n} \sum_{examples} y \ln(p) + (1 - y) \ln(1 - p)$	Обозначения для классификации • n = training examples • i = classes • p = prediction (probability) • y = true class (1/yes, 0/no)
Multi-class classification	Cross entropy for Softmax	$C = -\frac{1}{n} \sum_{examples} \sum_{classes} y_i \ln(p_i)$	
Regression	Mean Squared Error	$C = \frac{1}{n} \sum_{examples} (y - p)^2$	Обозначения for Regression • n = training examples • p = prediction (numeric, $\hat{y}$ • y = true value

Обучение нейронных сетей – метод обратного распространения ошибки (Backpropagation)

- Функция стоимости (Cost) выбирается в зависимости от задачи: двоичная, мультиклассовая классификация или регрессия.
- Обновите веса сети, применив метод градиентного спуска и **обратное распространение ошибки**.

- Формула обновления веса:

$$w_{new} = w_{old} - learning\_rate * \frac{\partial C}{\partial w}$$

C : Cost

Gradient по w

Обучение (Training)

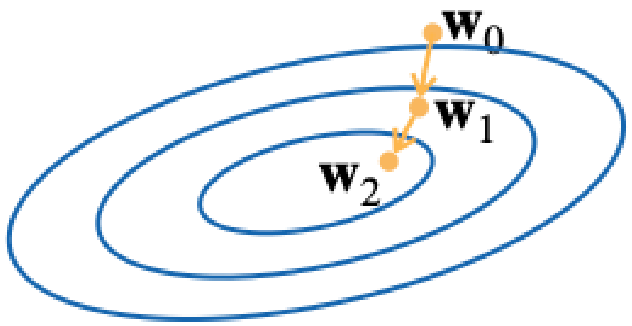
Чтобы «обучить» модель, нам нужно оптимизировать функцию стоимости $C(w)$.

- Также называется целевой функцией или функцией потерь.
- w относится к весам / параметрам / коэффициентам модели
- **Обратное распространение** ищет веса, которые минимизируют функцию стоимости.

Backpropagation

Градиентный спуск (Gradient descent):

- Метод оптимизации, используемый для обучения нейронных сетей
- Итеративное движение в направлении наискорейшего спуска
- При каждом обновлении веса:

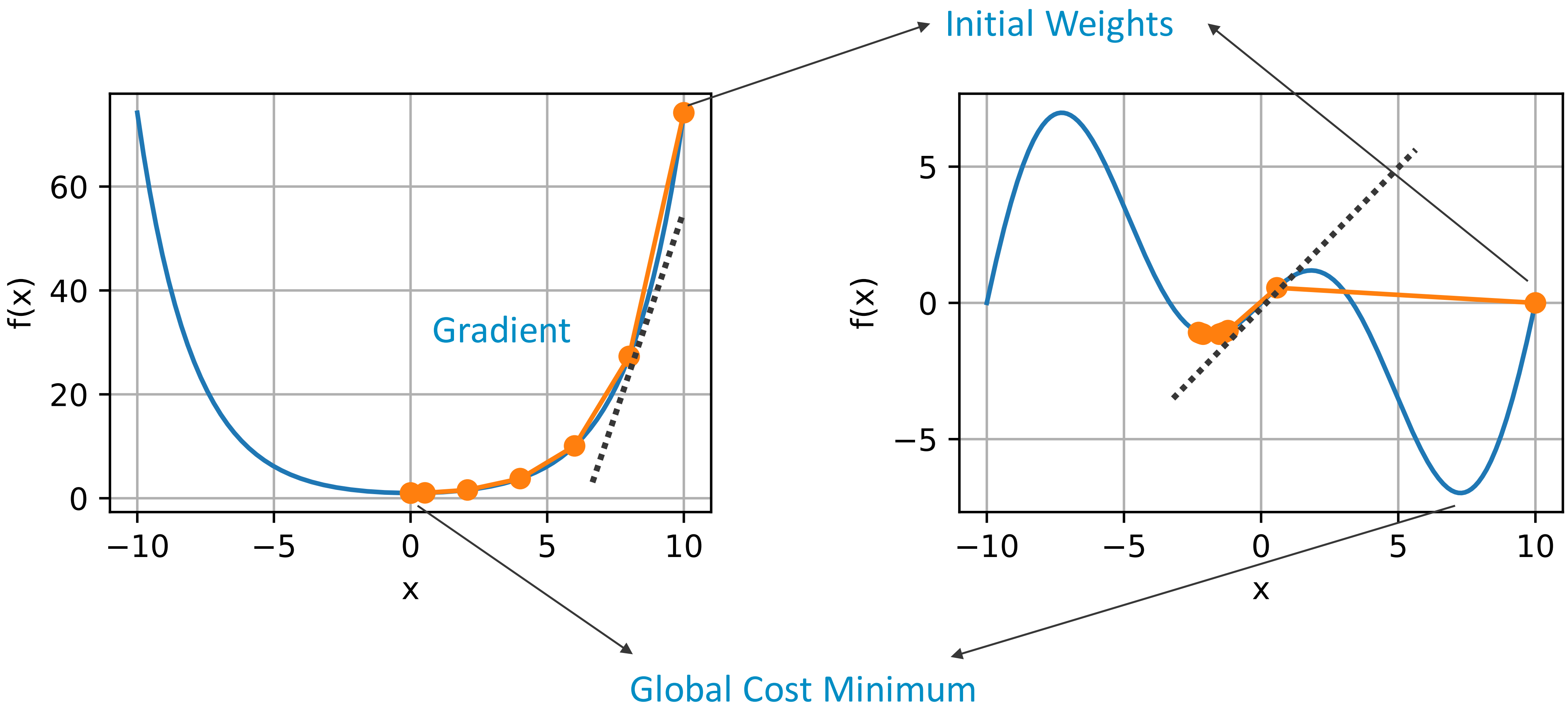


$$w_{new} := w_{old} - \Delta \mathrm{w}\,,\ \ \text{where}$$

$$\Delta \mathrm{w} \ = \text{learning\_rate} * \frac{\partial C}{\partial \mathrm{w}_{\text{old}}}$$

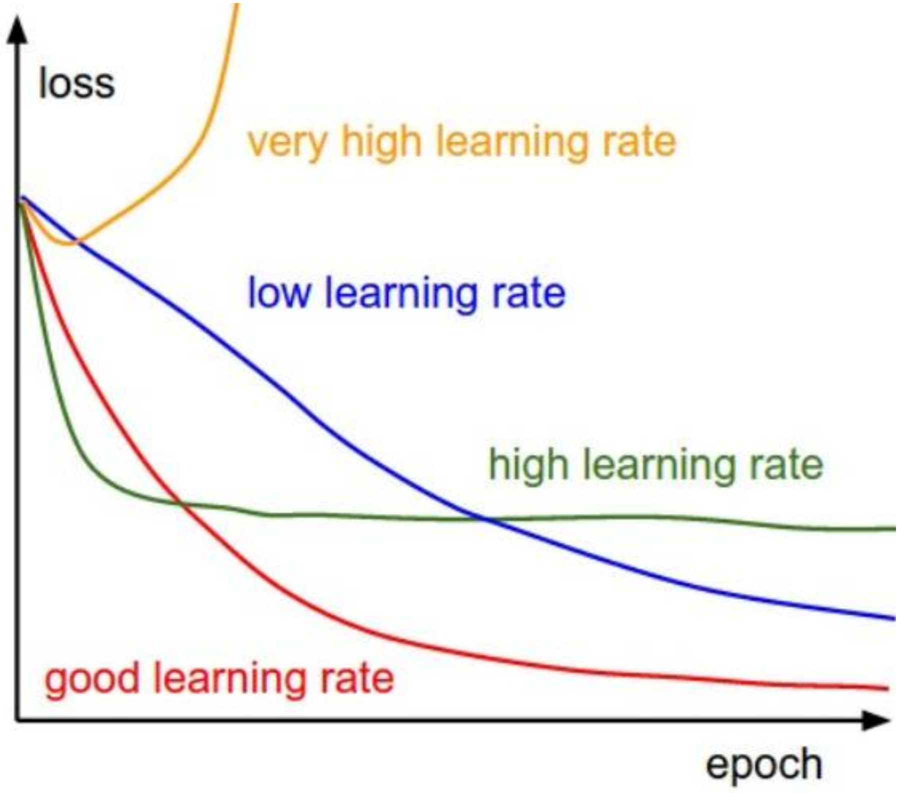
Gradient по w_{old}

Gradient Descent

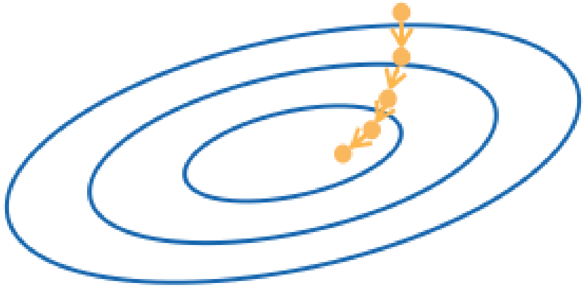


Learning Rate

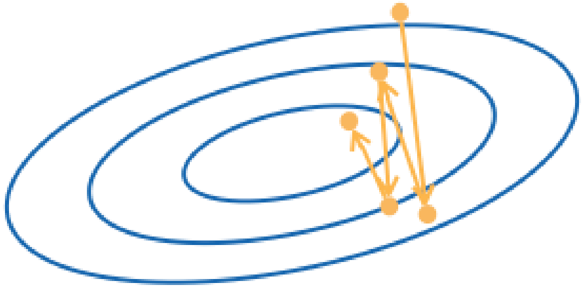
Обновление веса Δw - это произведение скорости обучения η и отрицательного градиента функции стоимости



$$\Delta w = -\eta \Delta C(w)$$



Too small…



Too big…

Backpropagation

Метод обратного распространения ошибки (англ. backpropagation) — метод вычисления градиента, который используется при обновлении весов многослойного перцептрона. Впервые метод был описан в 1974 г**. А.И. Галушкиным**, а также независимо и одновременно Полом Дж. Вербосом. Далее существенно развит в 1986 г. Дэвидом И. Румельхартом, Дж. Е. Хинтоном и Рональдом Дж. Вильямсом и независимо и одновременно С.И. Барцевым и В.А. Охониным. Это итеративный градиентный алгоритм, который используется с целью минимизации ошибки работы многослойного перцептрона и получения желаемого выхода.
Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении обратном прямому распространению сигналов в обычном режиме работы. Для возможности применения метода обратного распространения ошибки функция активации нейронов должна быть дифференцируема. Метод является изменением классического метода градиентного спуска.

Backpropagation

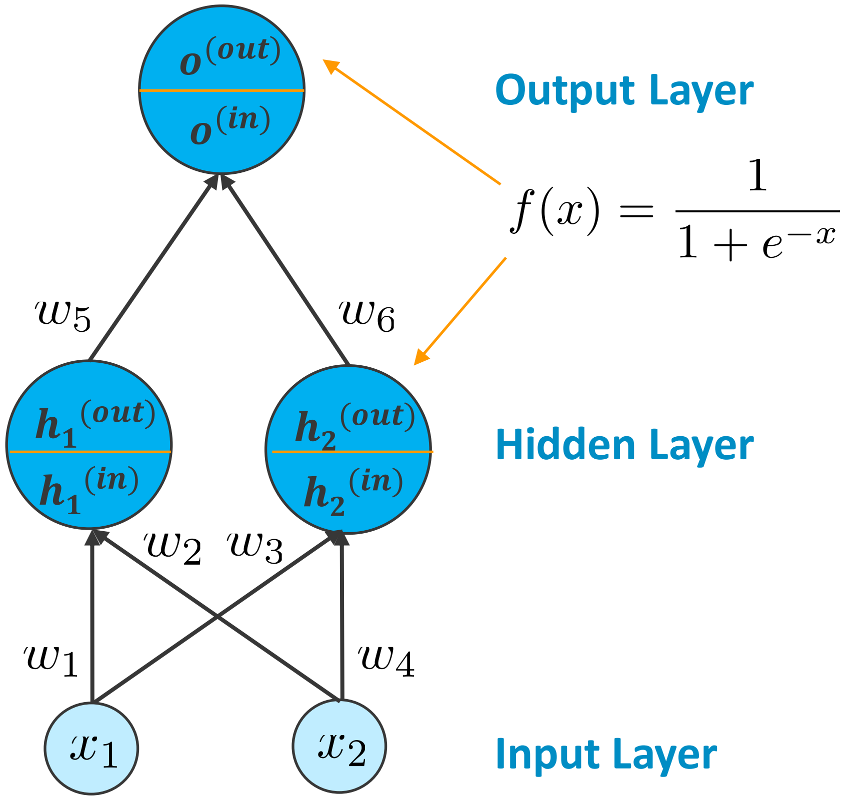
Обозначим через $w_{i,j}$ вес, стоящий на ребре, соединяющем i-й и j-й узлы, а через o_i выход i-го узла. Если нам известен обучающий пример (правильные ответы сети t_k), то функция ошибки, полученная по методу наименьших квадратов, выглядит так:

$$E(\{w_{i,j}\}) = \frac{1}{2} \sum_{k \in \mathrm{Outputs}} (t_k - o_k)^2$$

Как модифицировать веса? Мы будем реализовывать стохастический градиентный спуск, то есть будем подправлять веса после каждого обучающего примера и, таким образом, «двигаться» в многомерном пространстве весов. Чтобы «добраться» до минимума ошибки, нам нужно «двигаться» в сторону, противоположную градиенту, то есть, на основании каждой группы правильных ответов, добавлять к каждому весу

$$\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}},$$

где $0 < \eta < 1$ — множитель, задающий скорость «движения».



Backpropagation

Производная считается следующим образом. Пусть сначала $j \in \mathrm{Outputs}$, то есть интересующий нас вес входит в нейрон последнего уровня. Сначала отметим, что $w_{i,j}$ влияет на выход сети только как часть суммы $S_j = \sum_i w_{i,j} x_i$, где сумма берётся по входам *j*-го узла. Поэтому

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial S_j} \frac{\partial S_j}{\partial w_{i,j}} = x_i \frac{\partial E}{\partial S_j}$$

Аналогично, S_j влияет на общую ошибку только в рамках выхода *j*-го узла o_j (напоминаем, что это выход всей сети). Поэтому

$$\frac{\partial E}{\partial S_j} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial S_j} = \left(\frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \mathrm{Outputs}} (t_k - o_k)^2 \right) \left(\frac{\partial \, \mathrm{f}(S)}{\partial S} \Big|_{S=S_j} \right) =$$

$$= \left(\frac{1}{2} \frac{\partial}{\partial o_j} (t_j - o_j)^2 \right) (o_j(1-o_j)) 2\alpha = -2\alpha o_j (1-o_j)(t_j - o_j).$$

Backpropagation

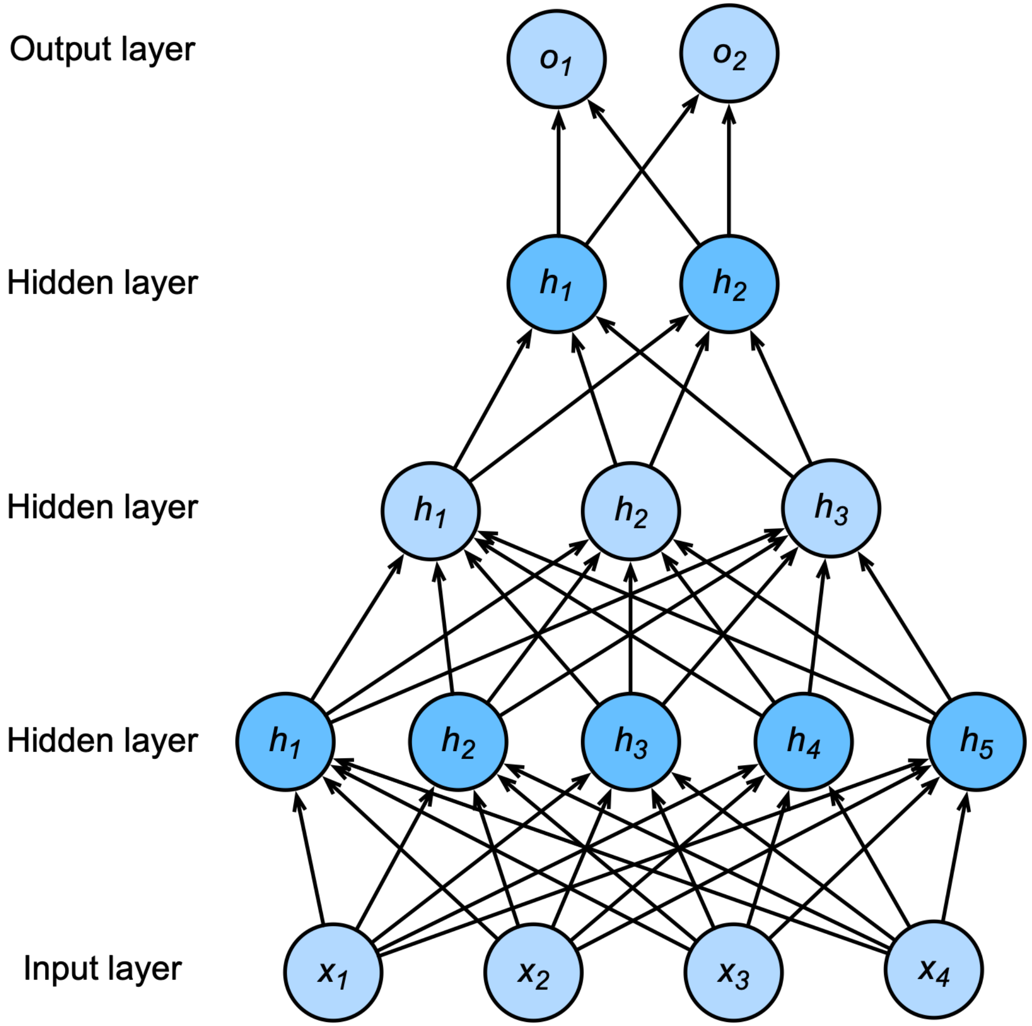
Если же *j*-й узел — не на последнем уровне, то у него есть выходы; обозначим их через Children(*j*). В этом случае

$$\frac{\partial E}{\partial S_j} = \sum_{k \in \mathrm{Children}(j)} \frac{\partial E}{\partial S_k} \frac{\partial S_k}{\partial S_j},$$

и

$$\frac{\partial S_k}{\partial S_j} = \frac{\partial S_k}{\partial o_j} \frac{\partial o_j}{\partial S_j} = w_{j,k} \frac{\partial o_j}{\partial S_j} = 2\alpha w_{j,k} o_j (1 - o_j).$$

α — коэффициент инерциальности для сглаживания резких скачков при перемещении по поверхности целевой функции (в функции активации - константа).



Но $\frac{\partial E}{\partial S_k}$ — это в точности аналогичная поправка, но вычисленная для узла следующего уровня

Backpropagation

Поскольку мы научились вычислять поправку для узлов последнего уровня и выражать поправку для узла более низкого уровня через поправки более высокого, можно уже писать алгоритм. Именно из-за этой особенности вычисления поправок алгоритм называется **алгоритмом обратного распространения ошибки** (backpropagation):

- для узла последнего уровня

$$\delta_j = -2\alpha o_j(1-o_j)(t_j-o_j)$$

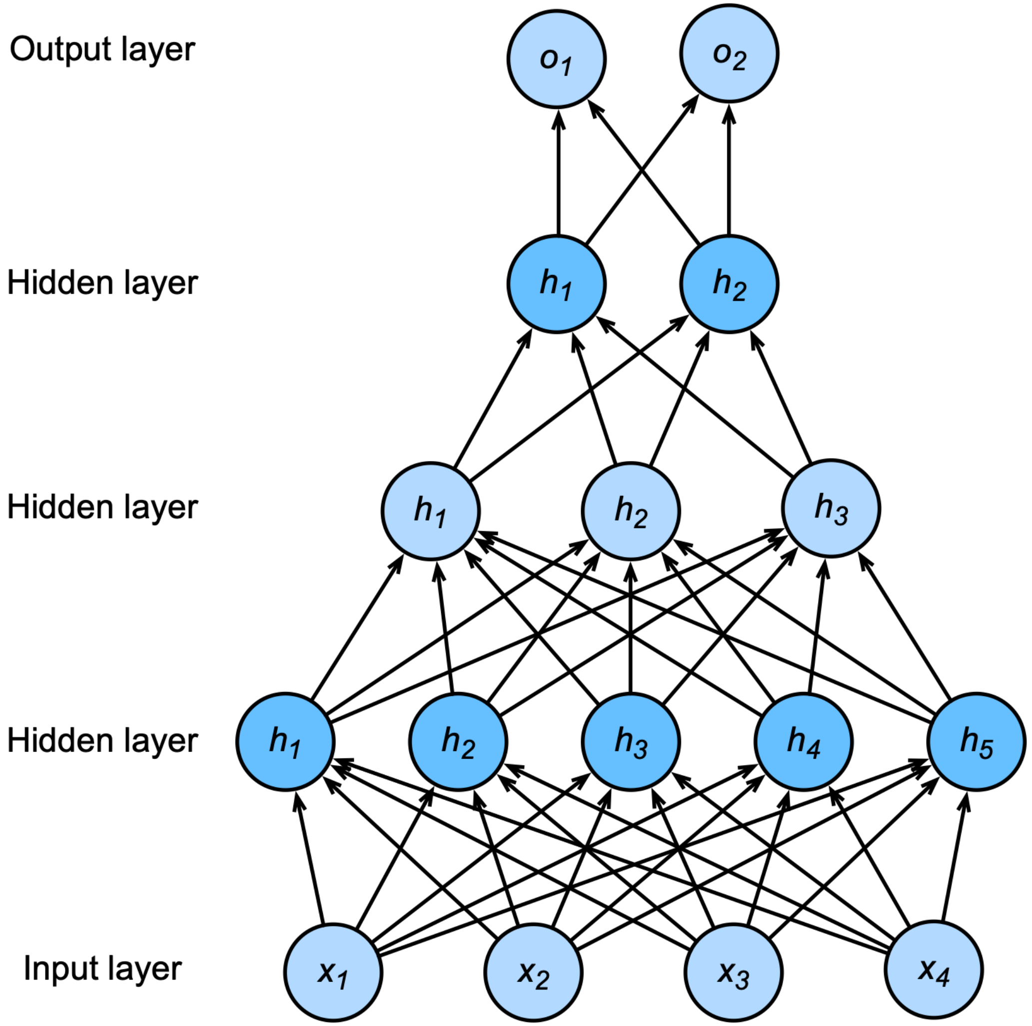
- для внутреннего узла сети

$$\delta_j = 2\alpha o_j(1-o_j) \sum_{k \in \mathrm{Children}(j)} \delta_k w_{j,k}$$

- для всех узлов

$$\Delta w_{i,j} = -\eta \delta_j o_i \,,$$

где o_i это тот же x_i в формуле для $\frac{\partial E}{\partial w_{i,j}}$.



Backpropagation

Алгоритм: ***BackPropagation*** $(\eta, \alpha, \{x_i^d, t^d\}_{i=1,d=1}^{n,m}, \text{steps})$

1. Инициализировать $\{w_{ij}\}_{i,j}$ маленькими случайными значениями, $\{\Delta w_{ij}\}_{i,j} = 0$
2. Повторить NUMBER_OF_STEPS раз:

   .Для всех d от 1 до m:

   1. Подать $\{x_i^d\}$ на вход сети и подсчитать выходы o_i каждого узла.
   2. Для всех $k \in Outputs$

      $$\delta_k = -o_k(1 - o_k)(t_k - o_k).$$

   3. Для каждого уровня l, начиная с предпоследнего:

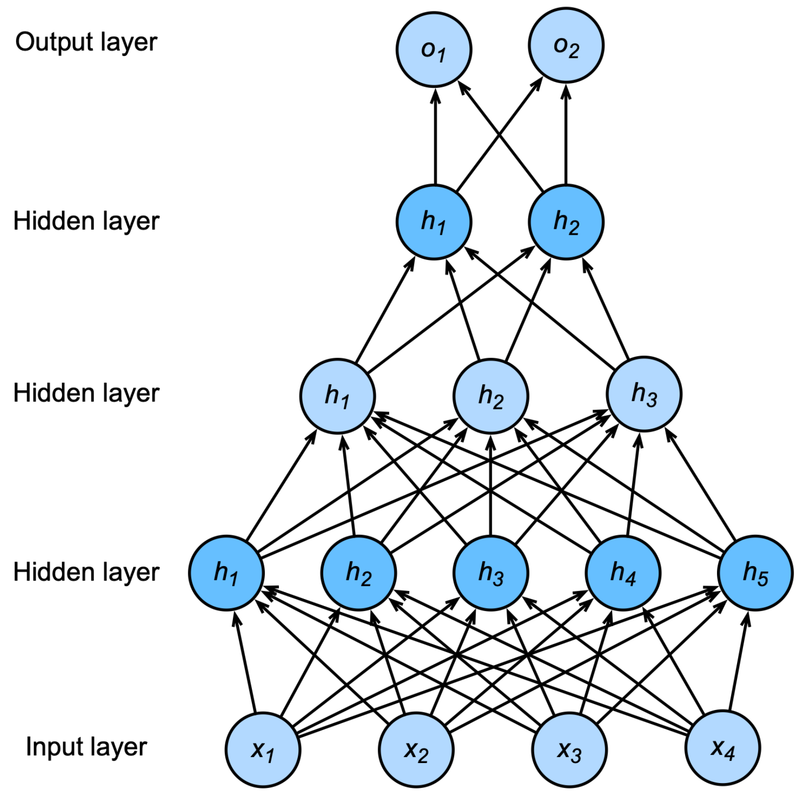
      Для каждого узла j уровня l вычислить

      $$\delta_j = o_j(1 - o_j) \sum_{k \in Children(j)} \delta_k w_{j,k}.$$

   4. Для каждого ребра сети {i, j}

      $$\Delta w_{i,j}(n) = \alpha \Delta w_{i,j}(n-1) + (1-\alpha)\eta \delta_j o_i.$$
      $$w_{i,j}(n) = w_{i,j}(n-1) - \Delta w_{i,j}(n).$$

3. Выдать значения w_{ij}.



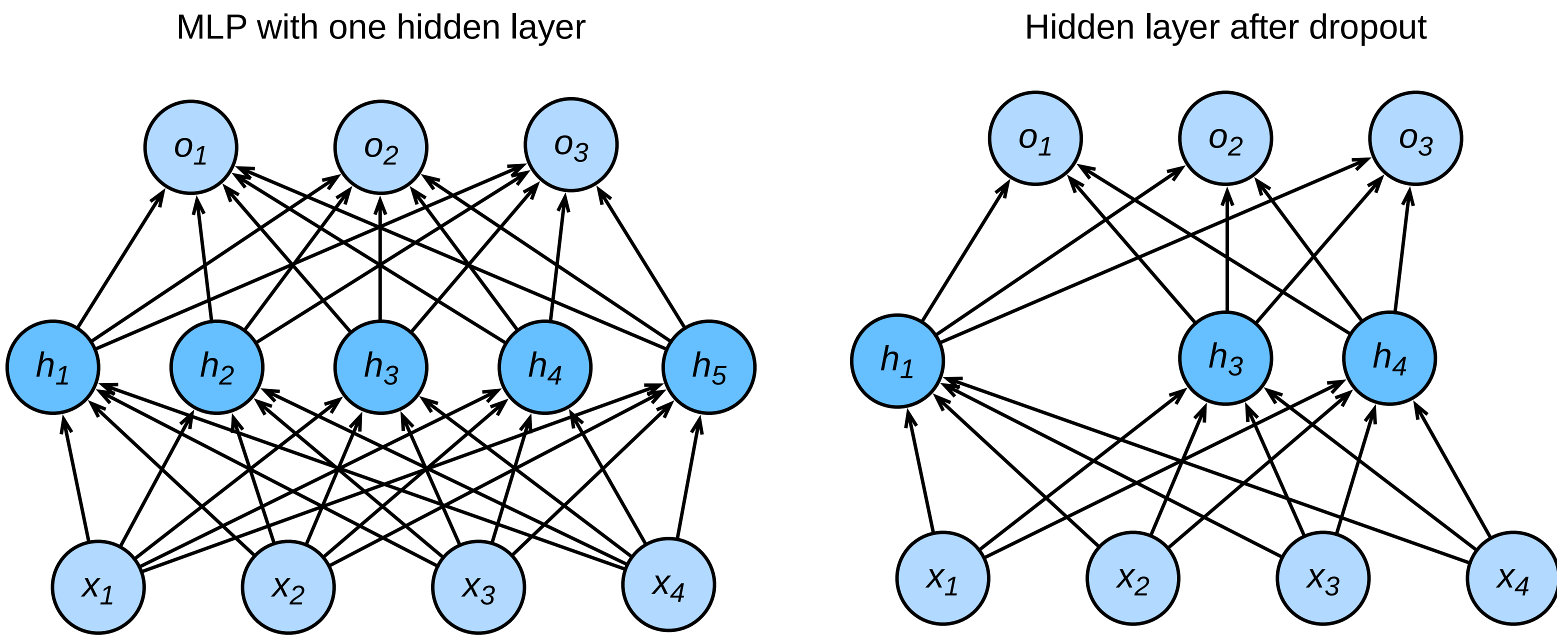
Резюме: жаргоны глубокого обучения

Model Design

- Architectures (# число слоев (layer), нейронов слое, взаимосвязи между слоями и нейронами и др.)
- Activation Function
  - Дифференцируемое «нелинейное отображение» (A differentiable “**nonlinear mapping**”)
- Output Function
  - Предсказываемая функция ”y” (A function to **predict**)
- Cost/Loss Function
  - Дифференцируемая функция для оптимизации модели (A *differentiable* function to **optimize** the model)
- Evaluation Function
  - Часто недифференцируемая функция для оценки модели (An often *non-differentiable* function to **evaluate** the model)

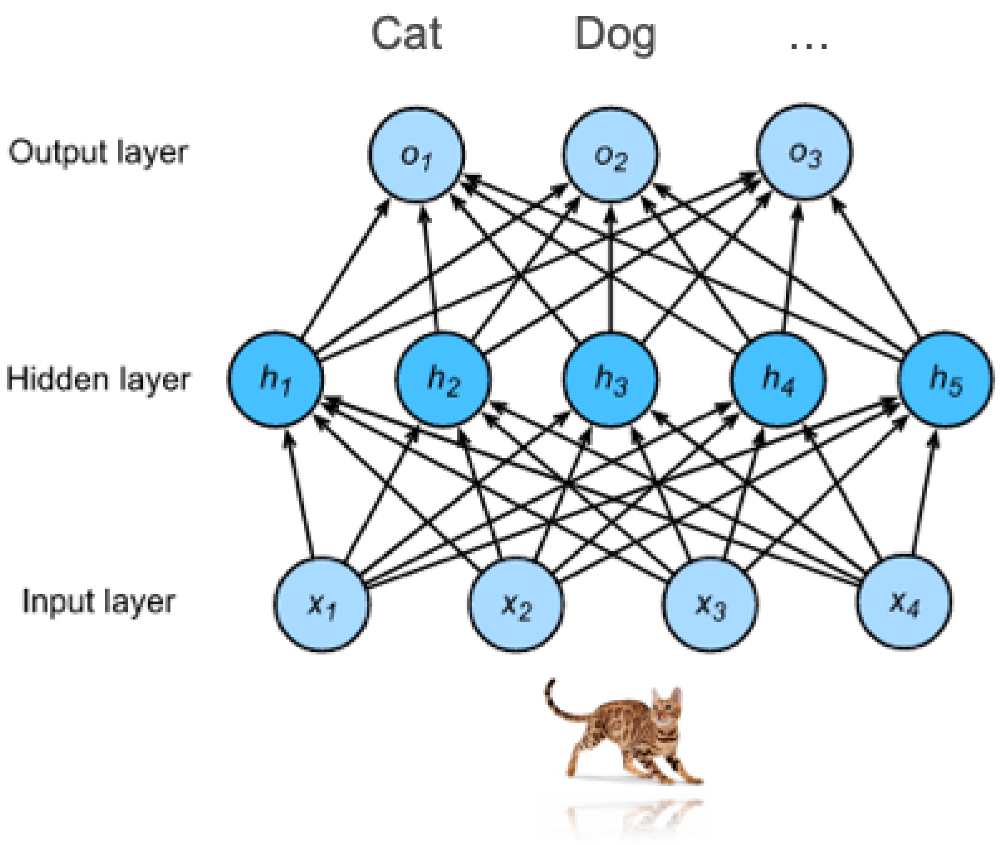
Dropout (исключение, прореживание)

- Техника регуляризации для предотвращения переобучения.
- Случайным образом удаляет некоторые узлы с фиксированной вероятностью во время обучения.



Почему нейронные сети?

- Автоматически извлекайте полезные функции из входных неструктурированных данных: фото, видео, тексты и др.
- В последние годы глубокое обучение достигло самых современных результатов во многих областях машинного обучения.
- Три столпа глубокого обучения :
  - Data
  - Compute
  - Algorithms



Создание и обучение нейронных сетей (Build and Train Neural Networks)

- Как создавать и использовать эти модели? машинного обучения?
- Неужели все может быть так просто?

```
(nn.Dense(64 ,activation='relu'),      # Layer 1
 nn.Dropout(.4),                       # Apply random 40% dropout to
 nn.Dense(128, activation='relu'),     # Layer 2
 nn.Dropout(.3),                       # Apply random 30% dropout to
 nn.Dense(1, activation='sigmoid'))    # Output layer
```

TensorFlow

PyTorch

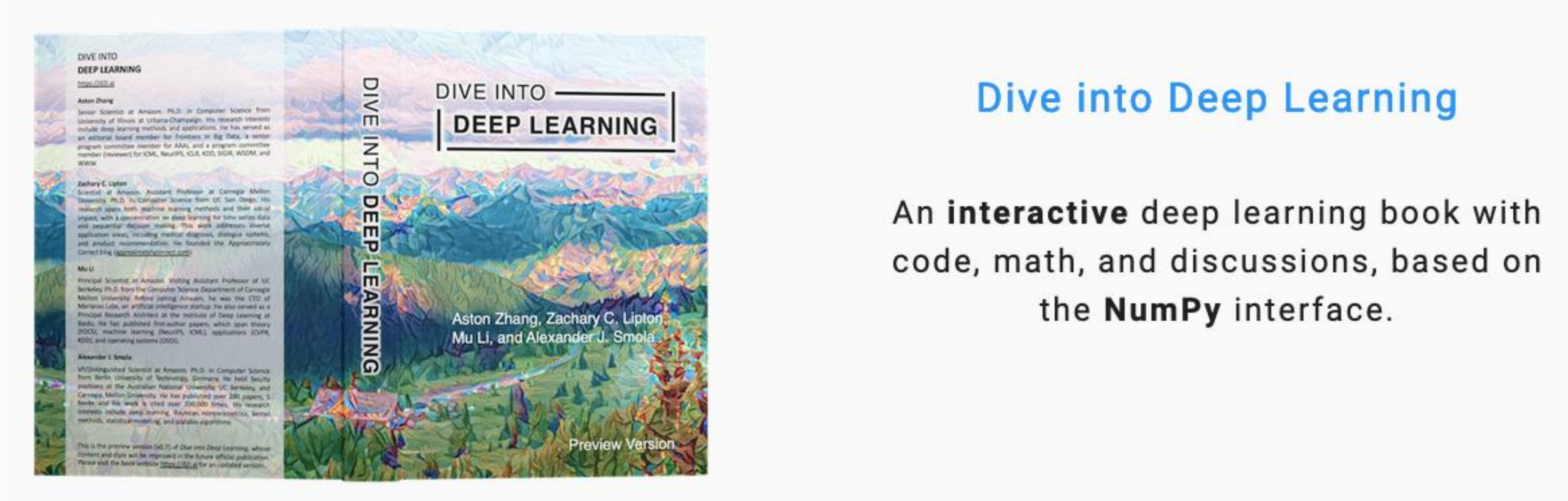
mxnet

Keras

Microsoft Cognitive Toolkit

theano

Dive into Deep Learning



E-book on **Deep Learning** by Amazon Scientists, available here: https://d2l.ai

Related chapters:

Chapters 3: Linear Neural Networks: https://d2l.ai/chapter_linear-networks/index.html

Chapters 4: Multilayer Perceptrons: https://d2l.ai/chapter_multilayer-perceptrons/index.html

MXNet

- Библиотека глубокого обучения с открытым исходным кодом для обучения и развертывания нейронных сетей.
- С интерфейсом **Gluon** мы можем легко определять и обучать нейронные сети.

mxnet

Amazon SageMaker MLA-TAB-Lecture3-MXNet.ipynb

MACHINE LEARNING UNIVERSITY

Собираем все вместе: Lecture 3

- В этой записной книжке мы продолжаем работать с нашим набором данных обзора, чтобы предсказать целевое поле.
- Ноутбук решает следующие задачи:
  - Исследовательский анализ данных
  - Разделение набора данных на обучающий и тестовый наборы
  - Масштабирование данных, кодирование категорий, векторизация текста
  - Обучение нейронной сети
  - Проверка показателей производительности на тестовом наборе

Amazon SageMaker MLA-TAB-Lecture3-Neural-Networks.ipynb

AutoML

AutoML

AutoML помогает автоматизировать некоторые задачи, связанные с разработкой и обучением модели машинного обучения, например:

- Preprocessing and cleaning data
- Feature selection
- ML model selection
- Hyper-parameter optimization

Auto GLUON AutoML

- AutoML Toolkit (AMLT) с открытым исходным кодом, созданный Amazon AI.
- Простота использования - встроенное приложение

Tabular Prediction

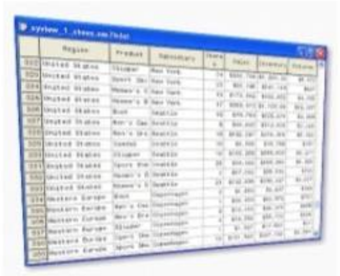
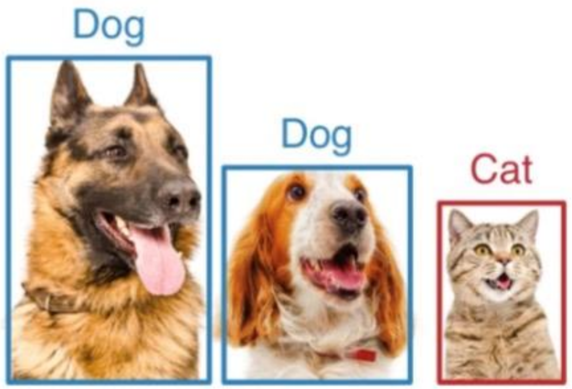


Image Classification

Object Detection



Text Classification

Auto GLUON AutoML

С **AutoGluon** современные результаты машинного обучения могут быть достигнуты с помощью нескольких строк кода Python.

```
>>> from autogluon import TabularPrediction as task
>>> predictor = task.fit(train_data=task.Dataset(file_path=TRAIN_DATA.csv), label=COLUMN_NAME)
>>> predictions = predictor.predict(task.Dataset(file_path=TEST_DATA.csv))
```

Спасибо