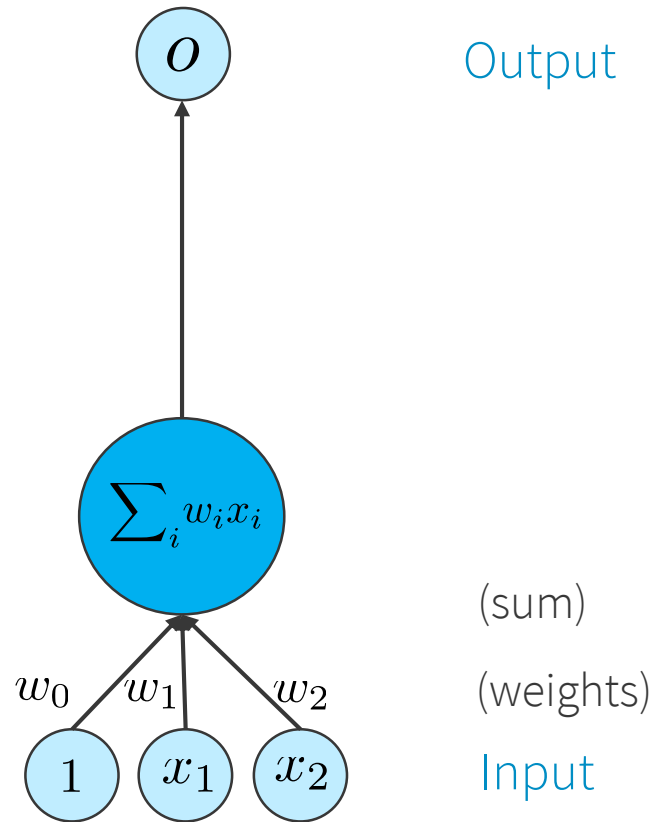


Neural Networks

Возвратимся к модели регрессии

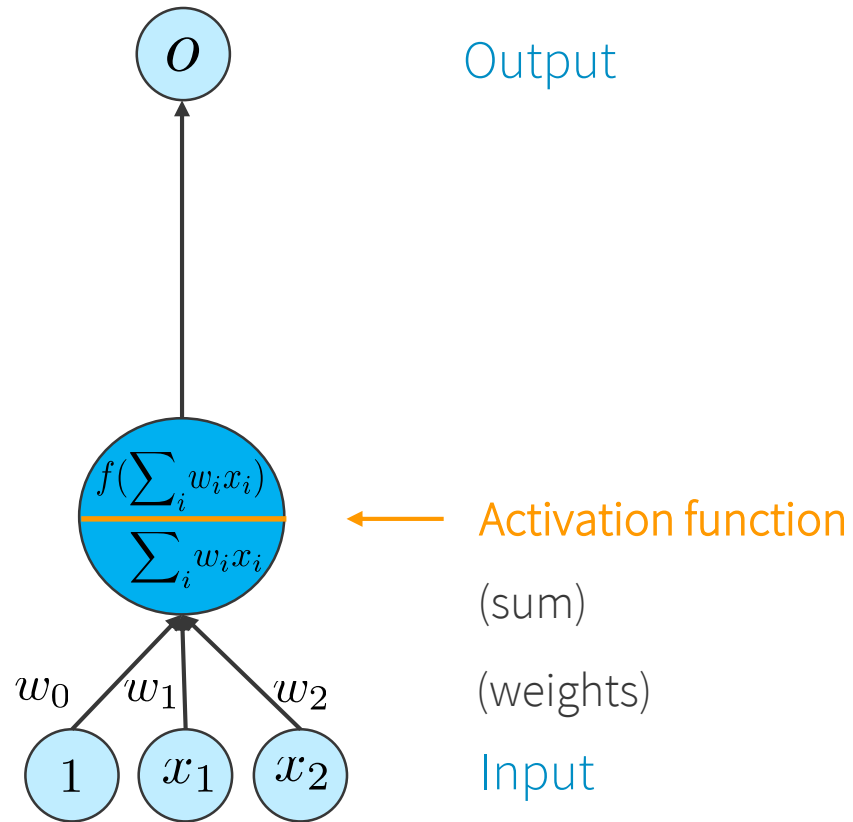


Linear Regression*: Дано $\{x_i\}$,
предсказать y :

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

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

Возвратимся к модели регрессии



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

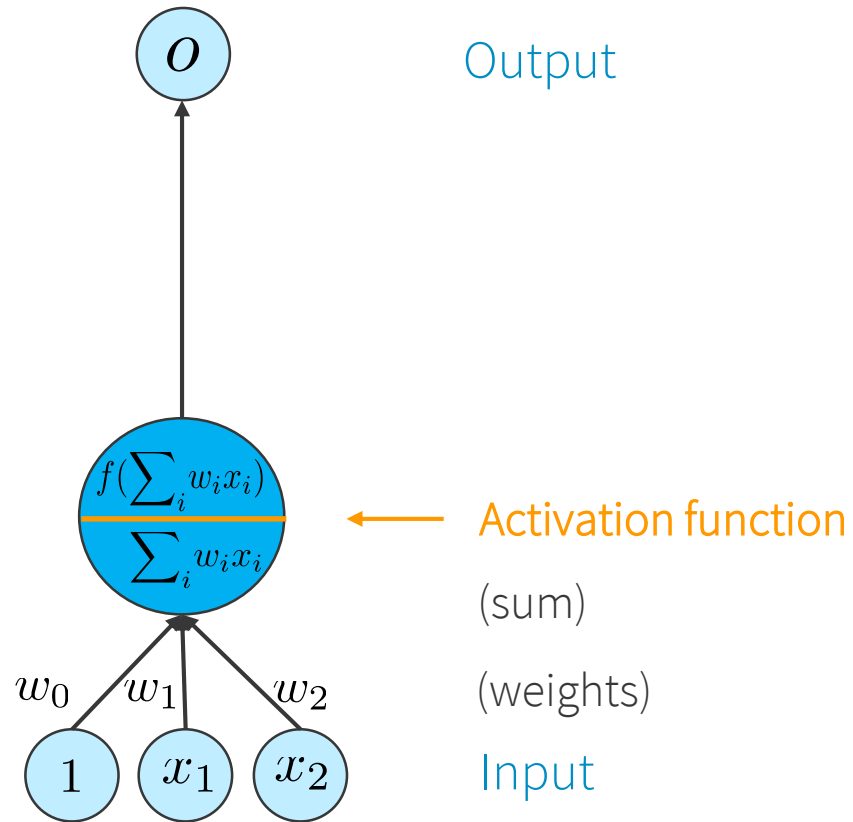
$$y = f(w_0 + w_1 x_1 + \dots + w_q x_q)$$

где f **линейная функция**:

$$f(x) = x$$

* Линейная функция активации

Возвратимся к модели регрессии



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

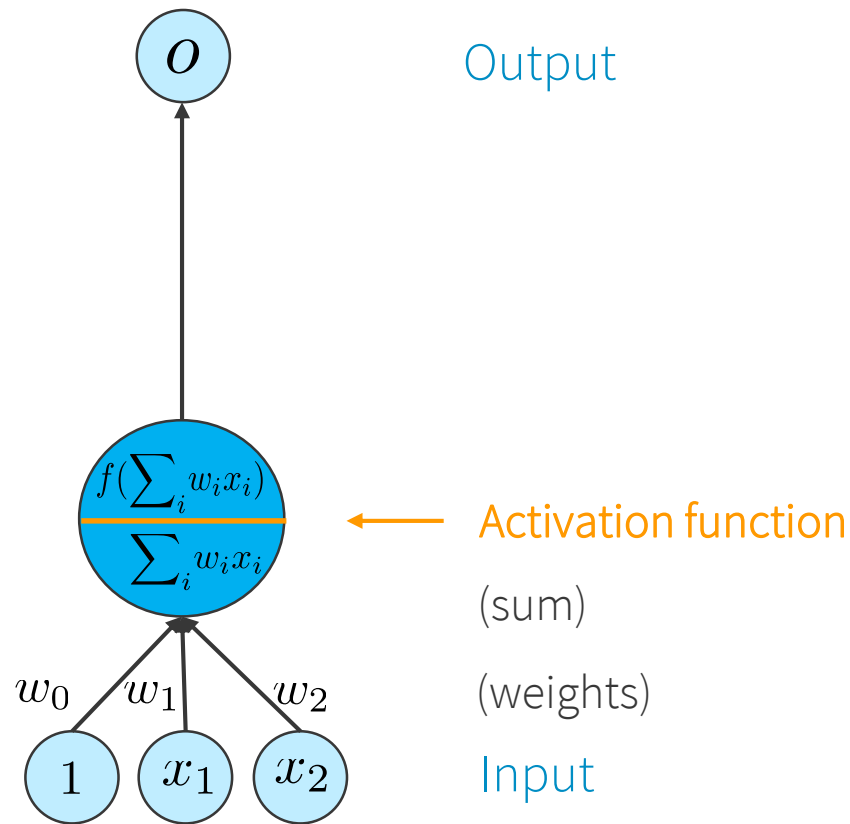
$$y = f(w_0 + w_1 x_1 + \dots + w_q x_q)$$

где f **logistic function**:

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

^{*} Нелинейная функция активации/бинарный классификатор

Perceptron (Rosenblatt, 1957)

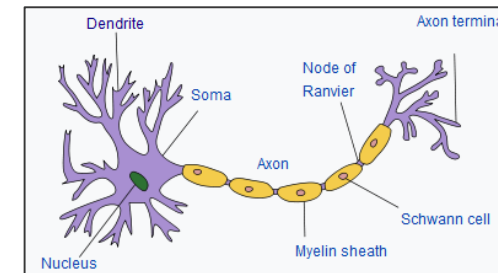


Perceptron^{*}: Given $\{x\}$, predict y
where $y \in \{-1, +1\}$

$$y = f(w_0 + w_1x_1 + \dots + w_qx_q)$$

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.

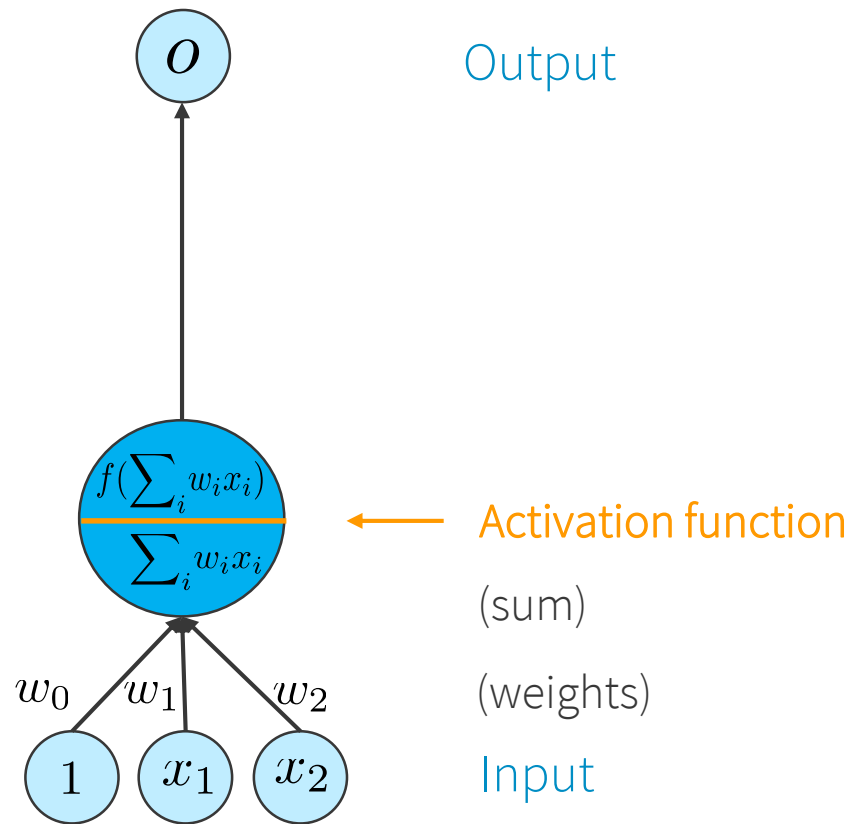
Проблема была решена В.И. Арнольдом совместно с **А.Н. Колмогоровым**, доказавшими, что любая непрерывная функция любого количества переменных представляется в виде суперпозиции *непрерывных* функций одной и двух переменных:

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

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

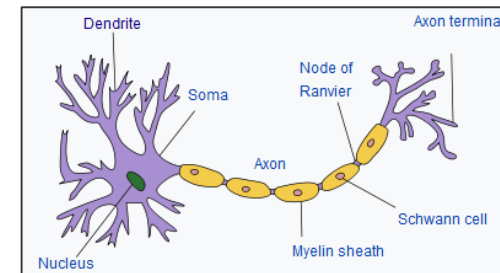
Искусственный нейрон (Artificial Neuron)



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

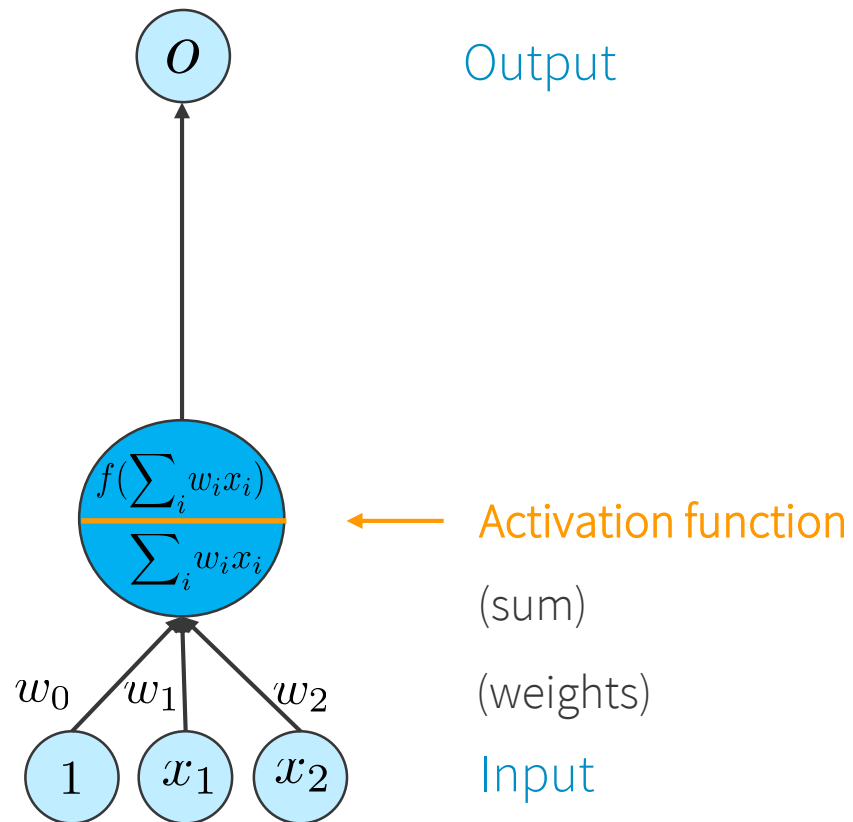
$$y = f(w_0 + w_1x_1 + \dots + w_qx_q)$$

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



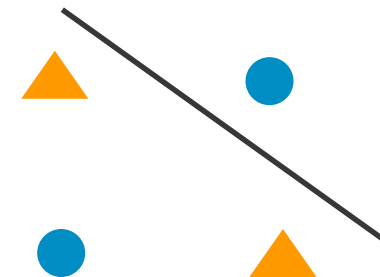
^{*} Similar to how neurons in the brain function

Artificial Neuron



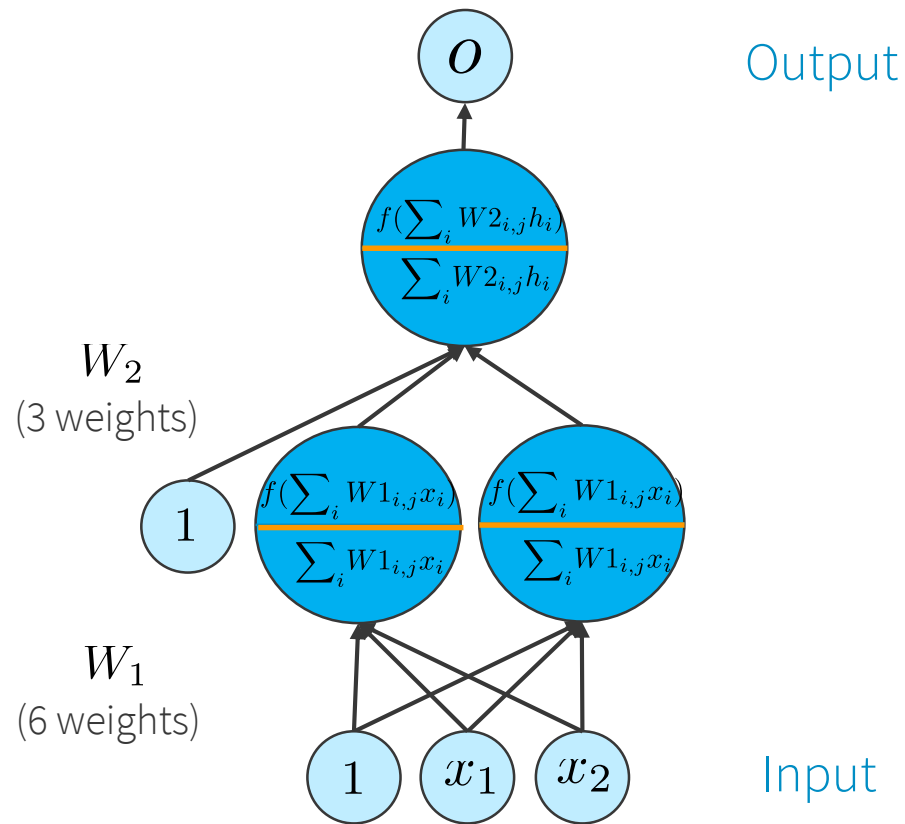
Artificial Neuron: Улавливает в основном линейные взаимодействия в данных.

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



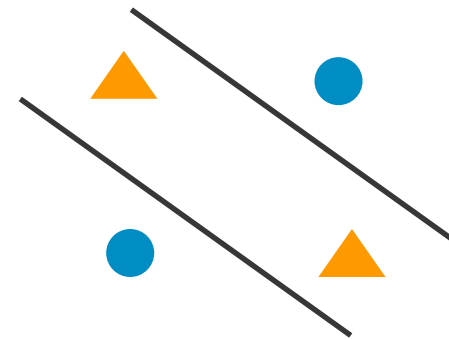
Not a very good classifier
...

Neural Network/Multilayer Perceptron

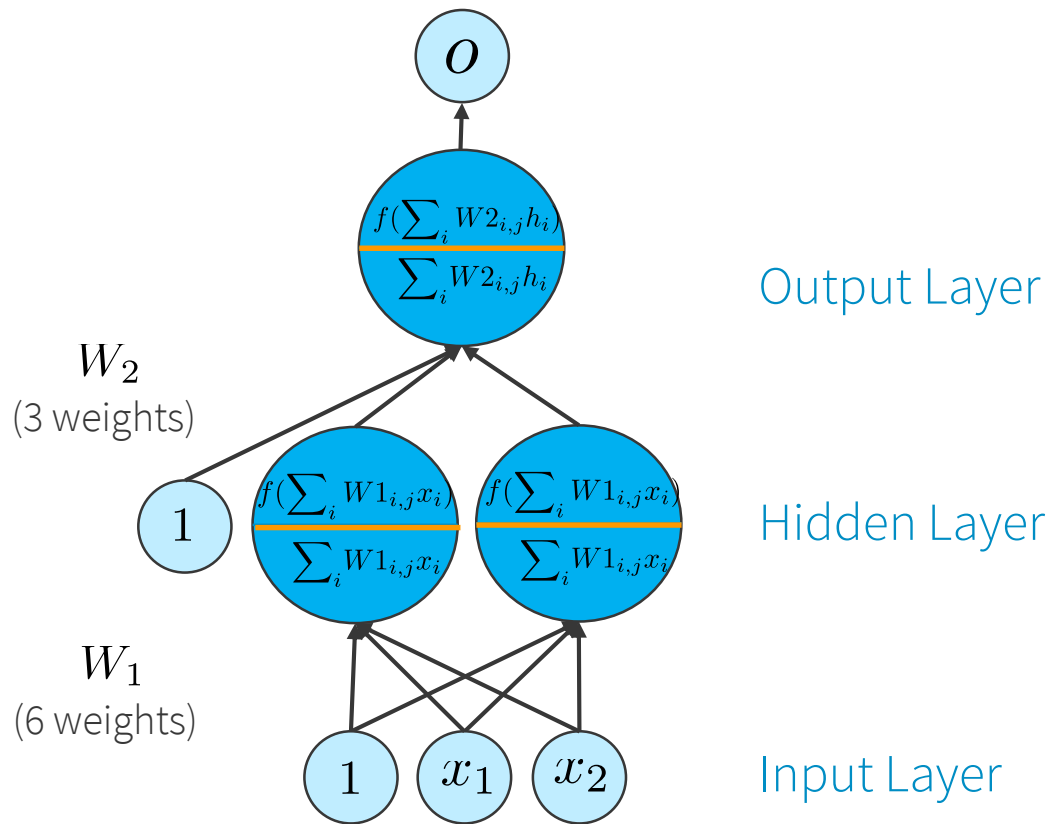


Artificial Neuron: Улавливает в основном линейные взаимодействия в данных.

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



Neural Network/Multilayer Perceptron

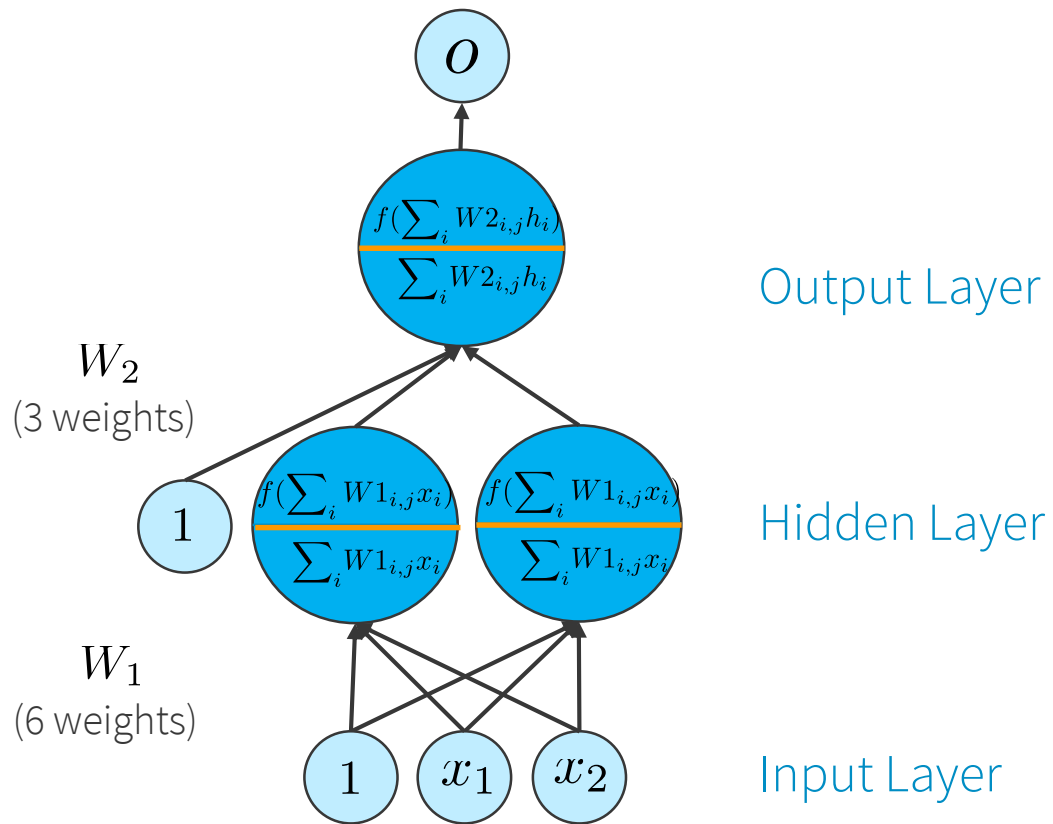


Artificial Neuron: Улавливает в основном линейные взаимодействия в данных.

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

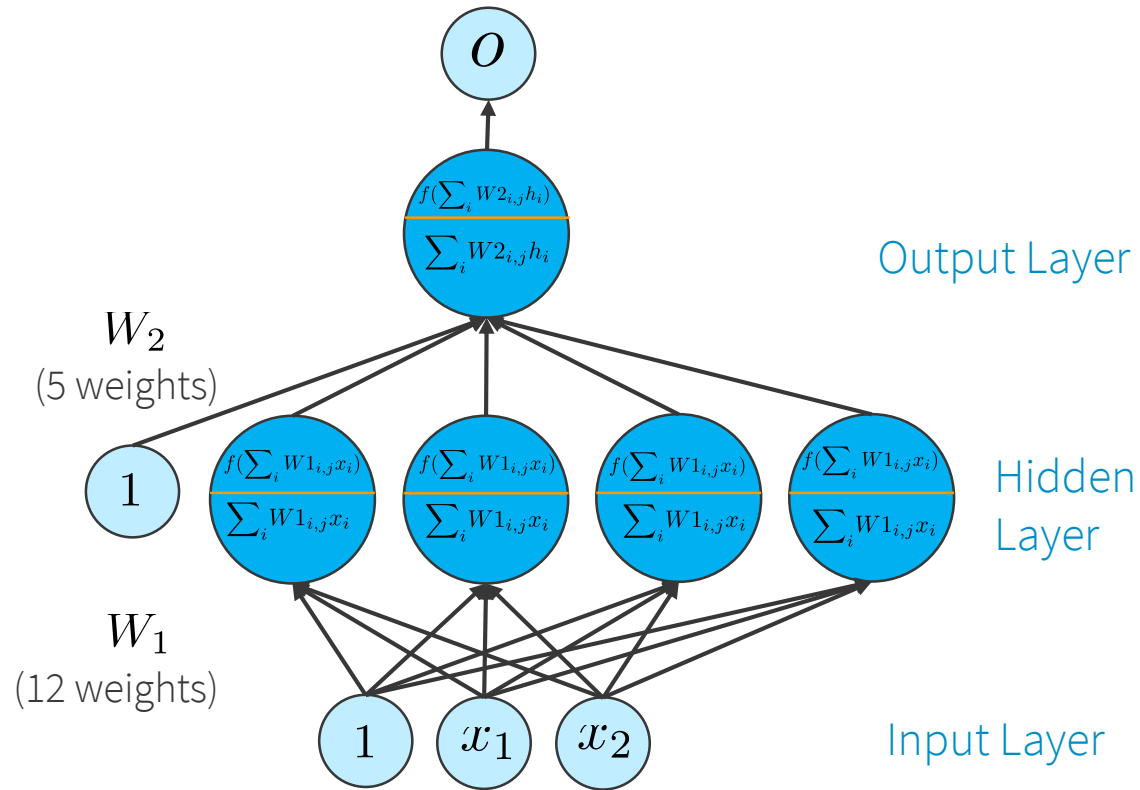
Neural Network/Multilayer Perceptron (MLP): используйте больше искусственных нейронов, объединяете их в **слой**!

Neural Network/Multilayer Perceptron



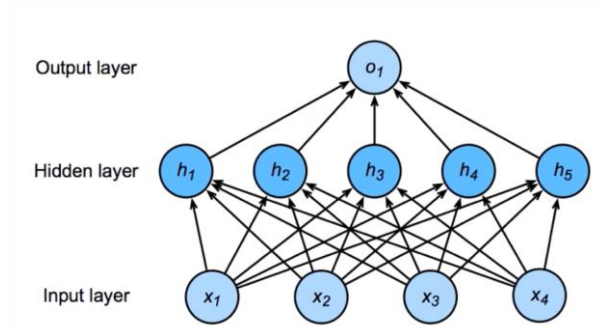
- Нейронная сеть, состоящая из **ВХОДНОГО, СКРЫТОГО И ВЫХОДНОГО** слоев
- Каждый слой связан со следующим слоем
- **Функция активации** применяется к каждому скрытому слою (и выходному слою)

Neural Network/Multilayer Perceptron

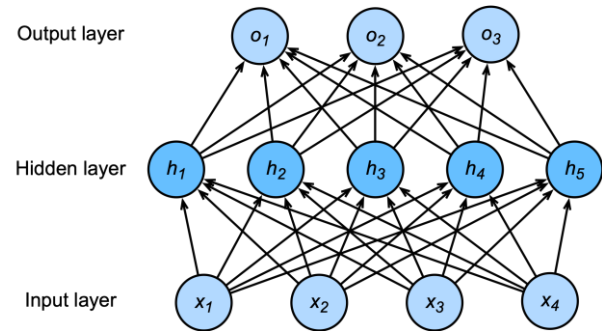


- Нейронная сеть, состоящая из **ВХОДНОГО, СКРЫТОГО И ВЫХОДНОГО** слоев
- Каждый слой связан со следующим слоем
- **Функция активации** применяется к каждому скрытому слою (и выходному слою)

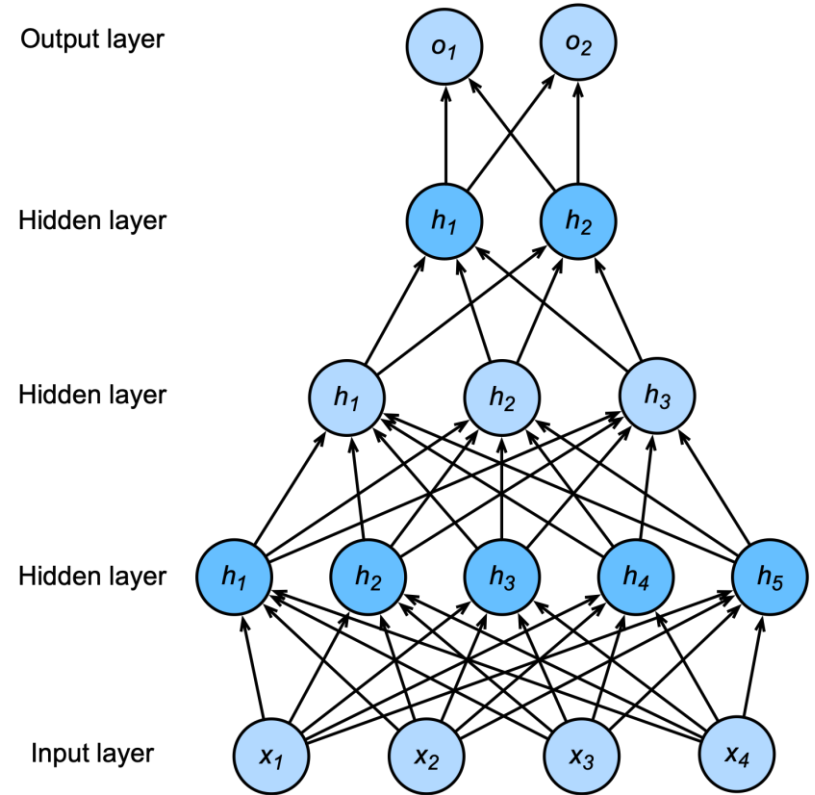
Neural Networks



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

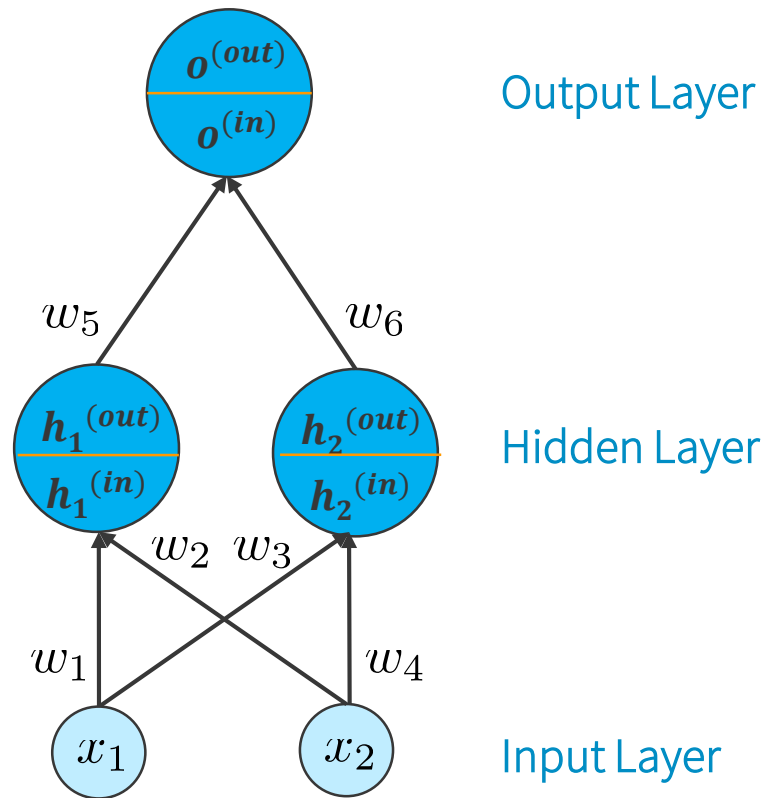


MultiLayer Network: Two layers (one hidden layer, output layer), with five hidden neurons in the hidden layer, and three output neurons.



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)

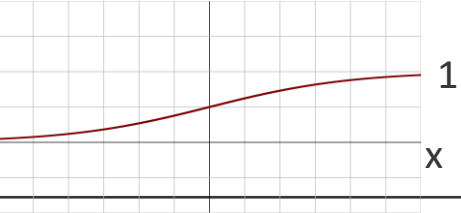
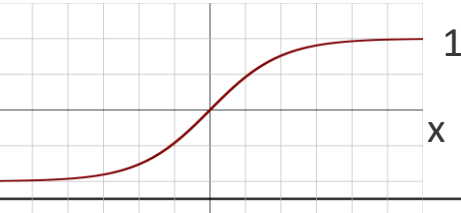
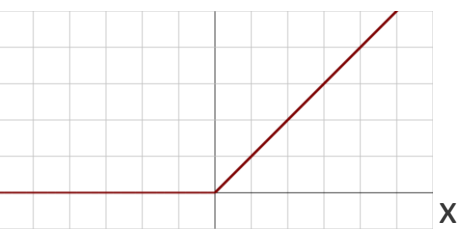


Построим нейронную сеть для задачи бинарной классификации:

- (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)		$f(x) = \frac{1}{1 + e^{-x}}$	Самая распространенная функция активации. Выход находится в диапазоне (0,1).
Hyperbolic tangent (tanh)		$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	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.

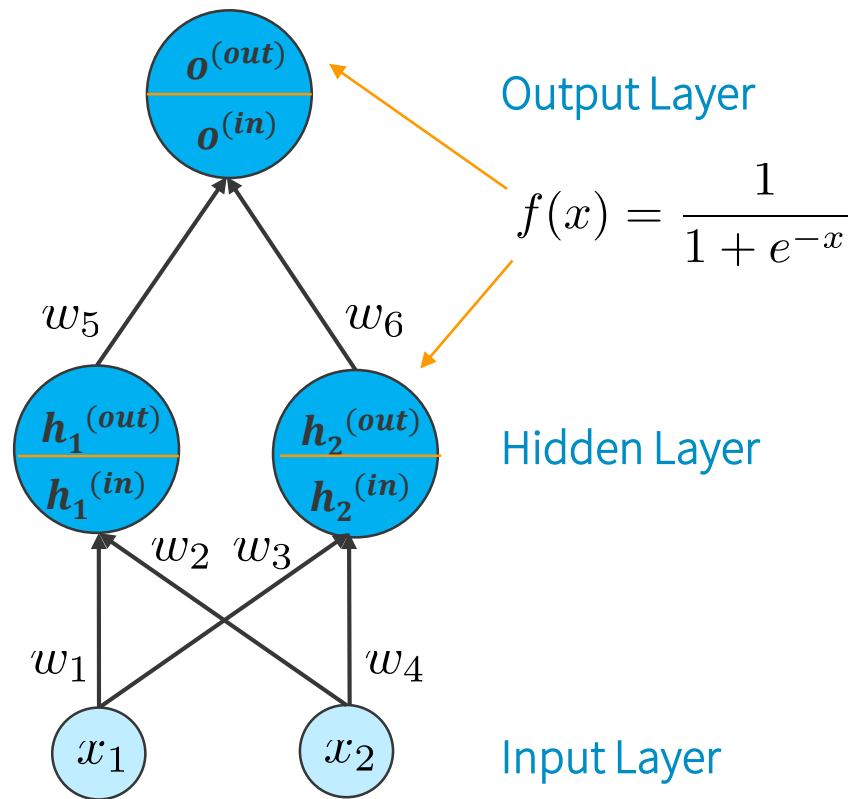
Также важны производные этих функций (т. Колмогорова и для реализации градиентного спуска).

Output Activations/Functions

- “Как вывести / спрогнозировать результат”

Problem	Description	Name	Function
Binary classification	<ul style="list-style-type: none">• Вероятность выхода для каждого класса в (0,1)	Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$
Multi-class classification	<ul style="list-style-type: none">• 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:

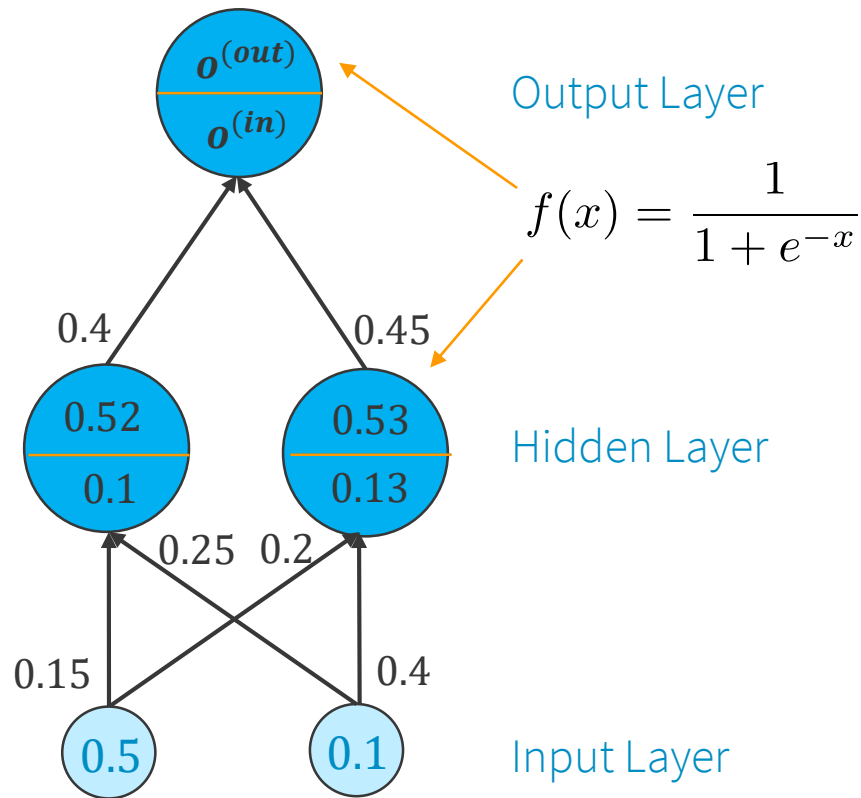
$$f(x) = \frac{1}{1 + e^{-x}}$$

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

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

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

Прямое распространение (Forward Pass)



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

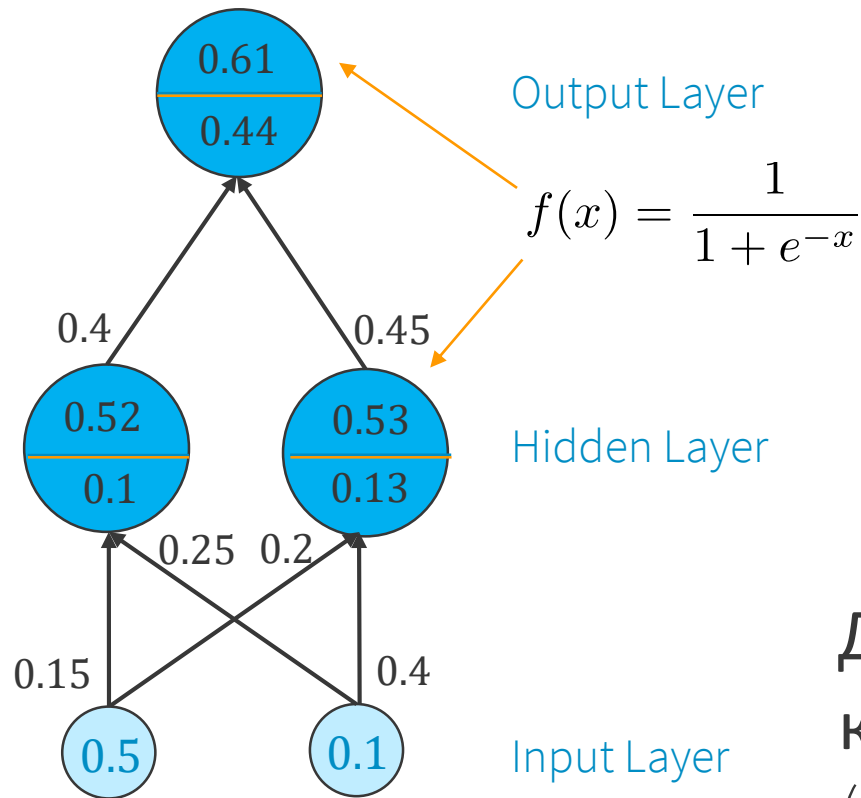
$$h_1^{(in)} = w_1 * x_1 + w_2 * x_2 \\ = 0.15 * 0.5 + 0.25 * 0.1 = 0.1$$

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

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

- “How to compare the outputs with the truth?”

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 <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • 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

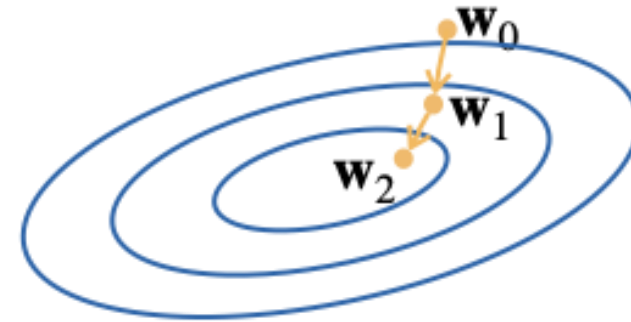
Градиентный спуск (Gradient descent):

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

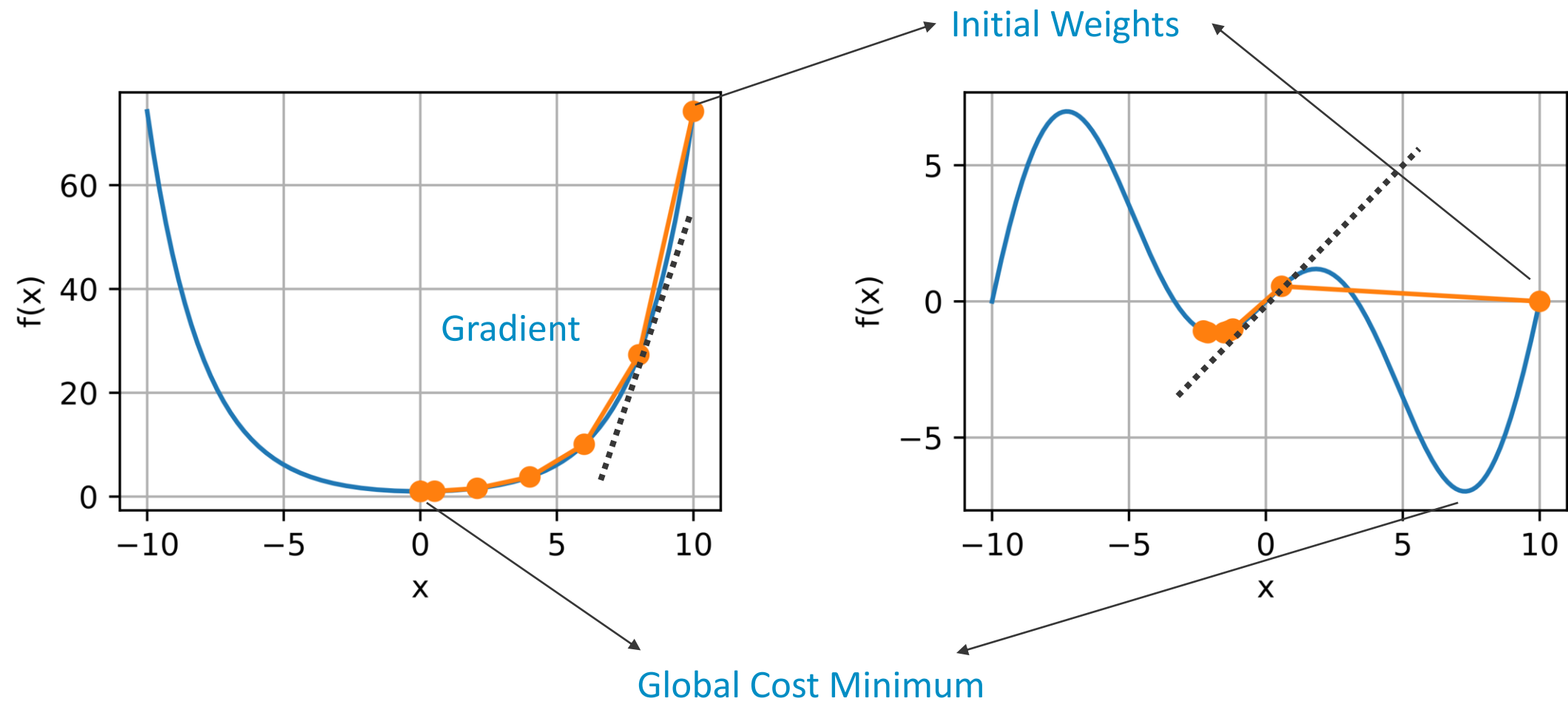
$w_{new} := w_{old} - \Delta w$, where

$$\Delta w = \text{learning_rate} * \frac{\partial C}{\partial w_{old}}$$

Gradient по w_{old}



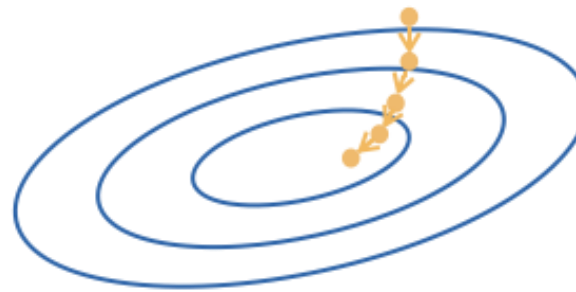
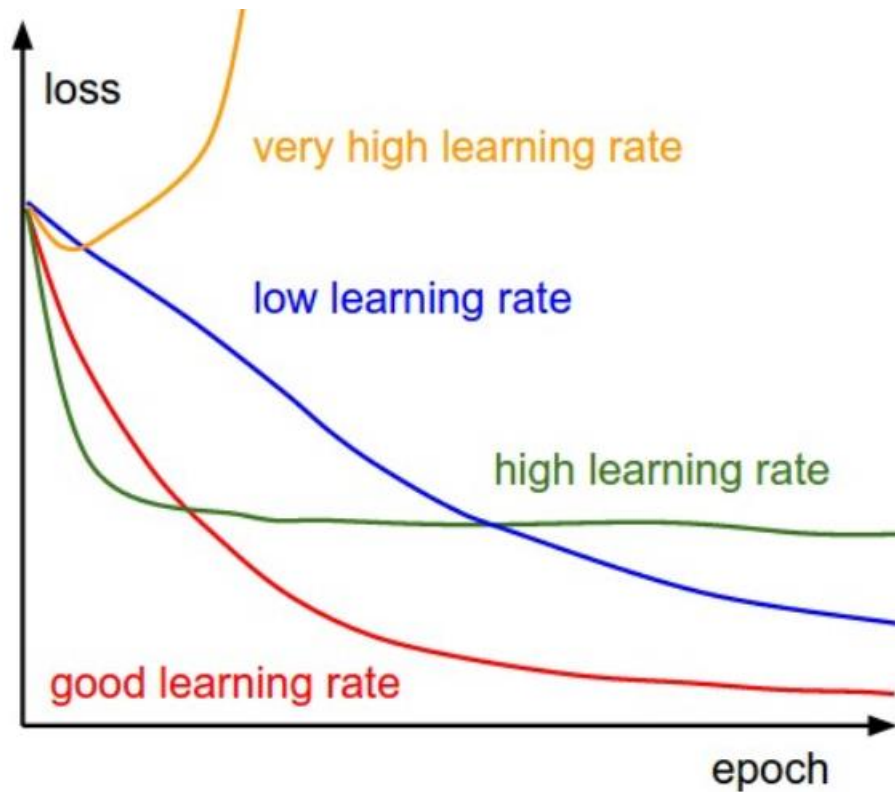
Gradient Descent



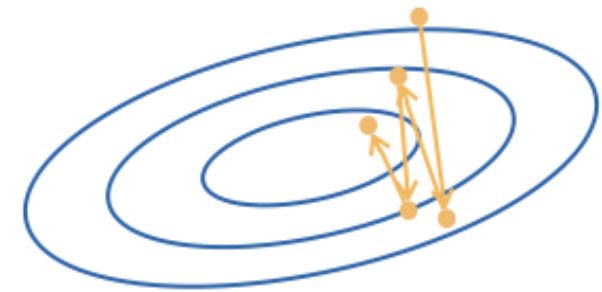
Learning Rate

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

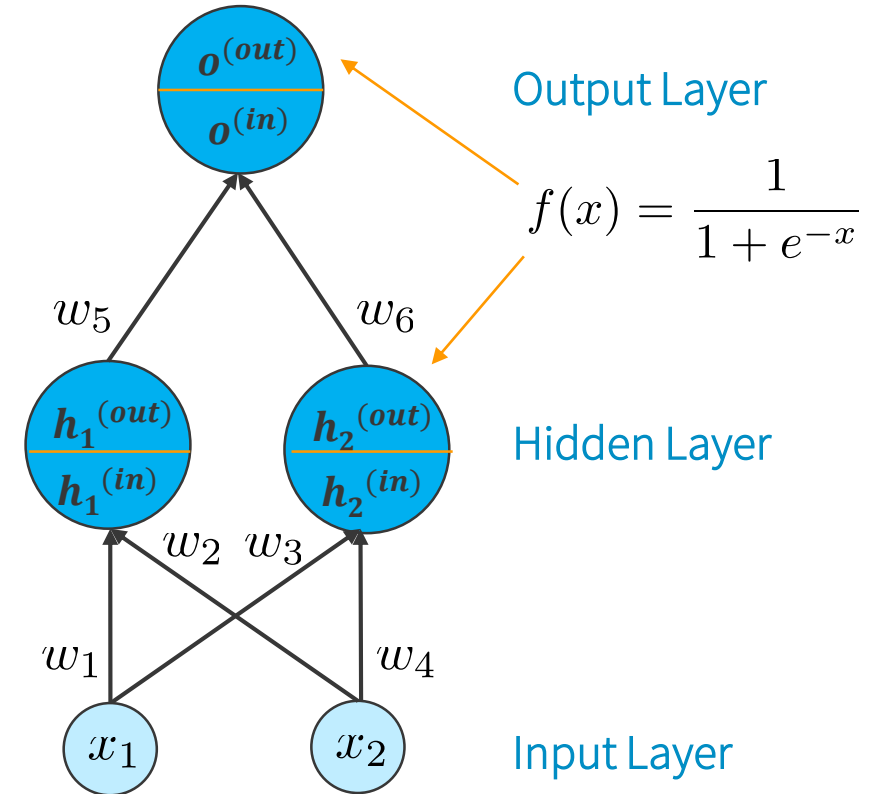
Обозначим через $w_{i,j}$ вес, стоящий на ребре, соединяющем i -й и j -й узлы, а через O_i выход i -го узла. Если нам известен обучающий пример (правильные ответы сети t_k), то функция ошибки, полученная по методу наименьших квадратов, выглядит так:

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

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 $j \in \text{Outputs}$, 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

$$\begin{aligned} \frac{\partial E}{\partial S_j} &= \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial S_j} = \left(\frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{Outputs}} (t_k - o_k)^2 \right) \left(\frac{\partial f(S)}{\partial S} \Big|_{S=S_j} \right) = \\ &= \left(\frac{1}{2} \frac{\partial}{\partial o_j} (t_j - o_j)^2 \right) (o_j(1 - o_j)) 2\alpha = -2\alpha o_j(1 - o_j)(t_j - o_j). \end{aligned}$$

Backpropagation

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

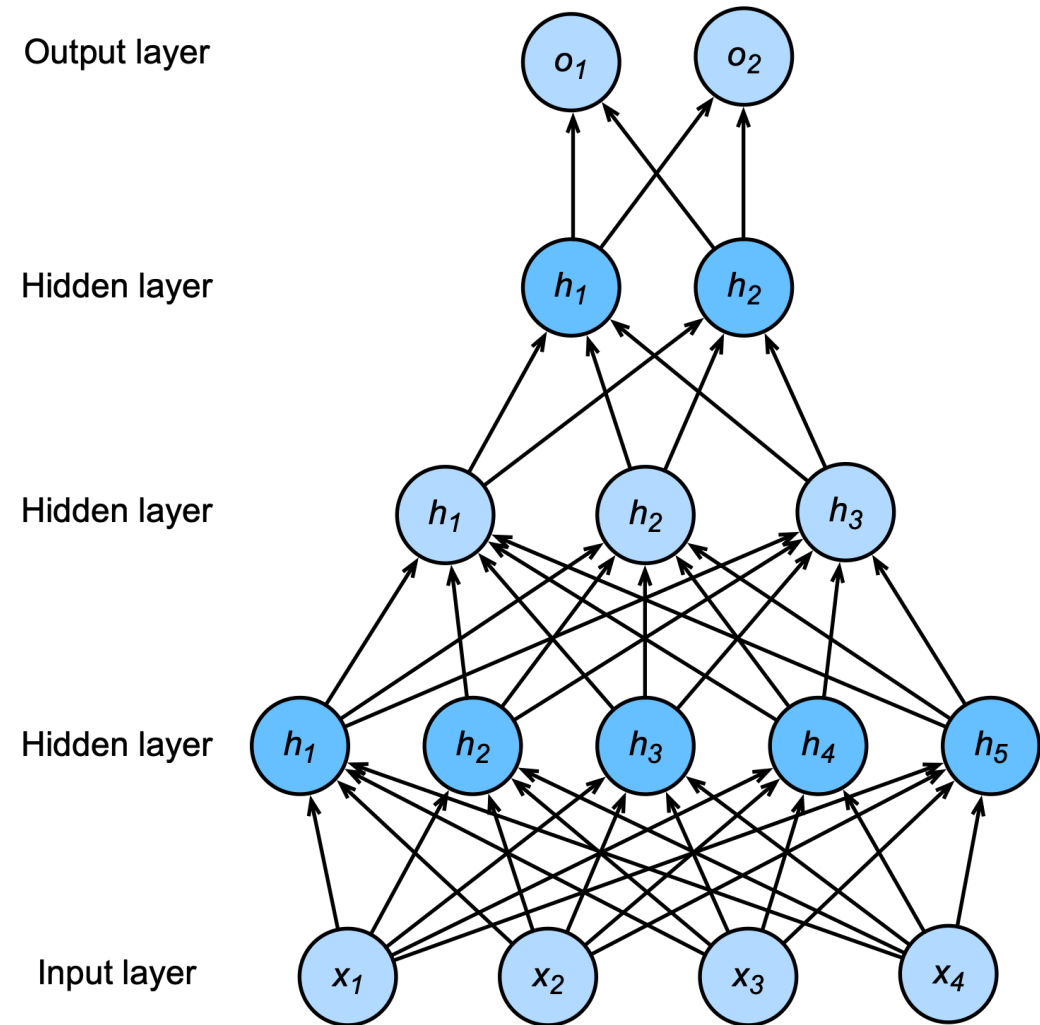
$$\frac{\partial E}{\partial S_j} = \sum_{k \in \text{Children}(j)} \frac{\partial E}{\partial S_k} \frac{\partial S_k}{\partial S_j},$$

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

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

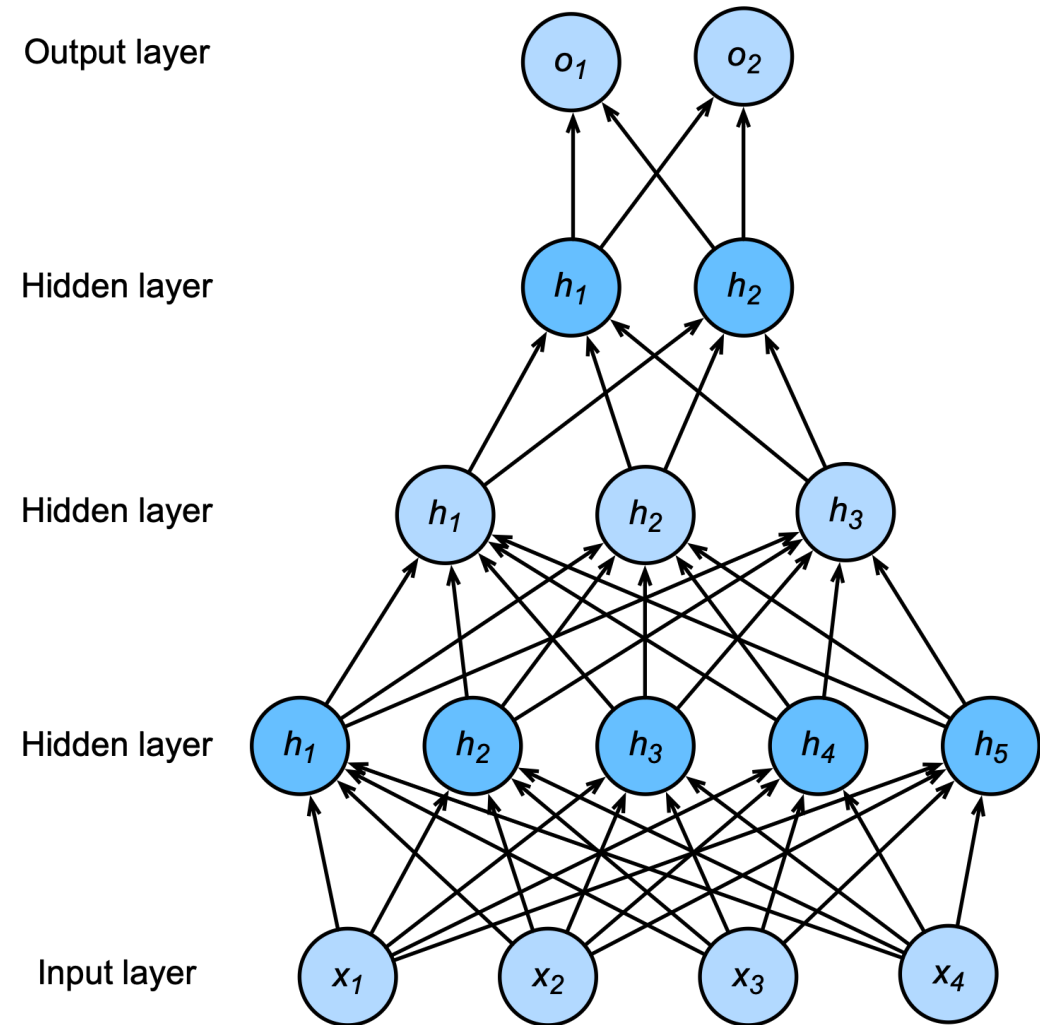
- для внутреннего узла сети

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

- 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 $k \in \text{Outputs}$

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

3. Для каждого уровня l , начиная с предпоследнего:

Для каждого узла j уровня l вычислить

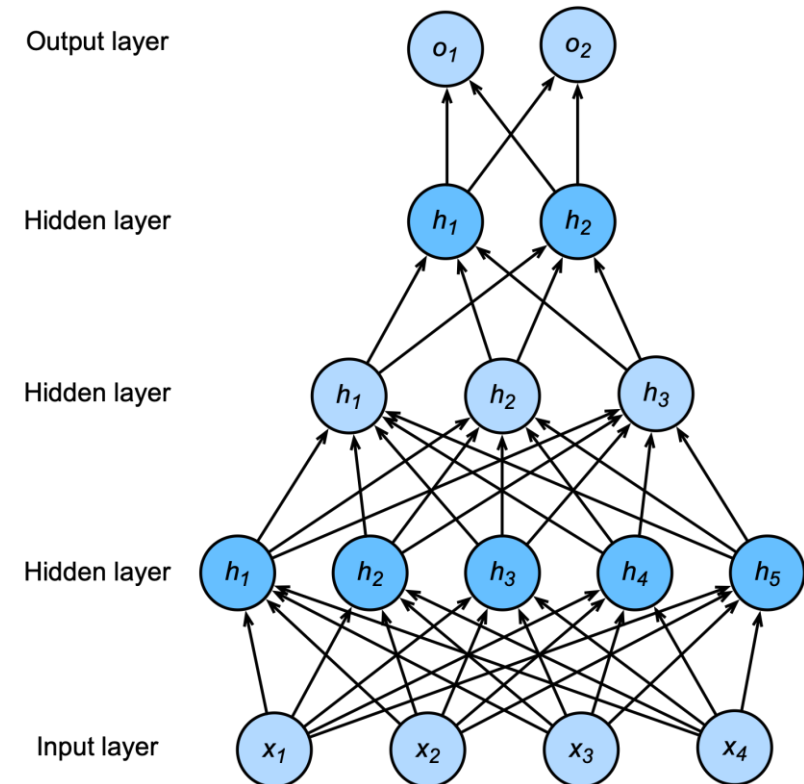
$$\delta_j = o_j(1 - o_j) \sum_{k \in \text{Children}(j)} \delta_k w_{j,k}.$$

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

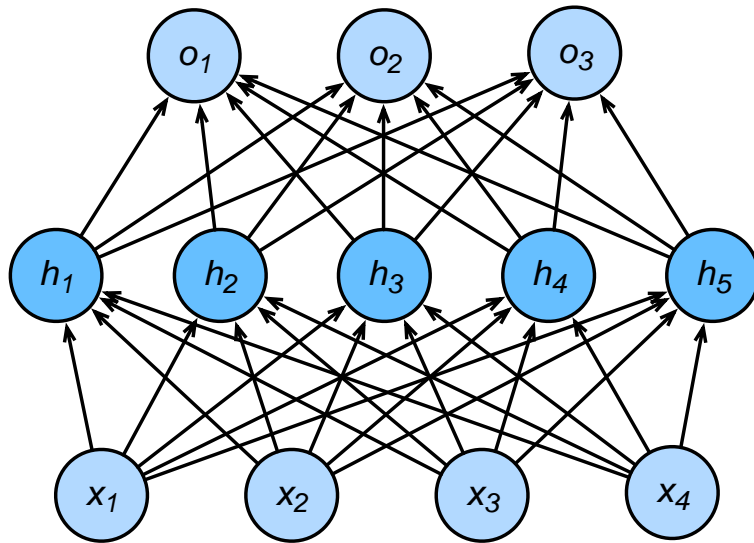
Model Design

- 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)

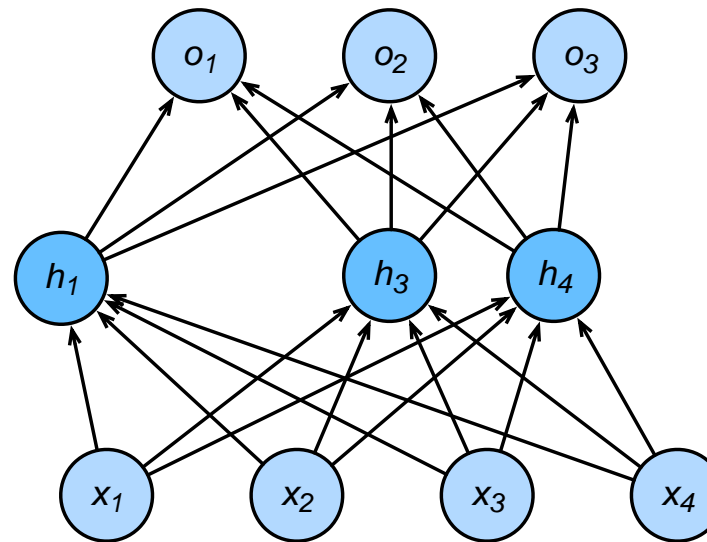
Dropout (исключение, прореживание)

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

MLP with one hidden layer

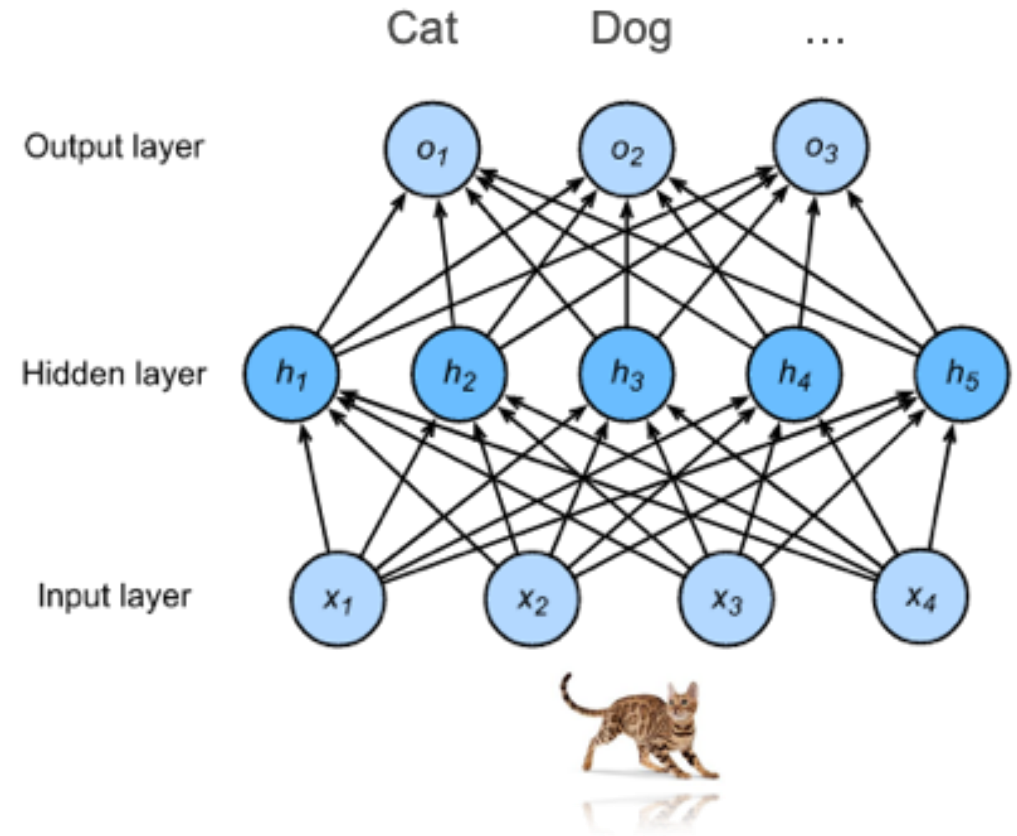


Hidden layer after dropout



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'),  
 nn.Dropout(.4),  
 nn.Dense(128, activation='relu'),  
 nn.Dropout(.3),  
 nn.Dense(1, activation='sigmoid'))
```

*# Layer 1
Apply random 40% dropout to
Layer 2
Apply random 30% dropout to
Output layer*



Keras

theano



Microsoft
Cognitive
Toolkit

AutoML

AutoML

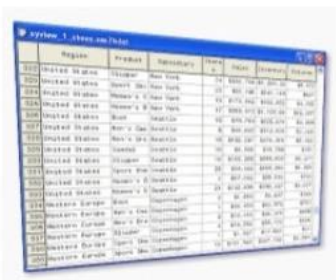
AutoML помогает автоматизировать некоторые задачи, связанные с разработкой и обучением модели машинного обучения, например:

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

Auto AutoML

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

Tabular
Prediction

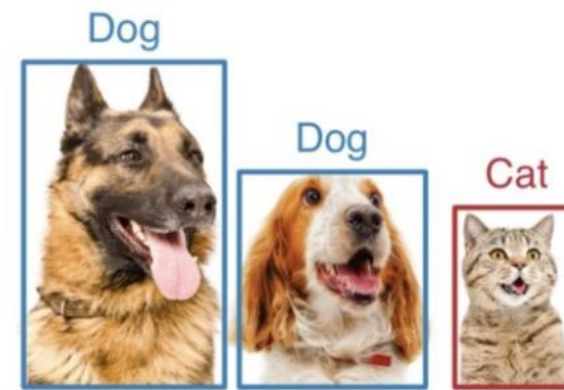


	Region	Property	Bedrooms	Bathrooms	Price	Square Feet
1	United States	House	3	2	\$100,000	1,500
2	United States	House	4	3	\$150,000	2,000
3	United States	House	3	2	\$120,000	1,800
4	United States	House	4	3	\$180,000	2,200
5	United States	House	3	2	\$110,000	1,600
6	United States	House	4	3	\$160,000	2,100
7	United States	House	3	2	\$130,000	1,900
8	United States	House	4	3	\$170,000	2,300
9	United States	House	3	2	\$140,000	2,000
10	United States	House	4	3	\$190,000	2,400

Image
Classification



Object
Detection



Text
Classification



Auto 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