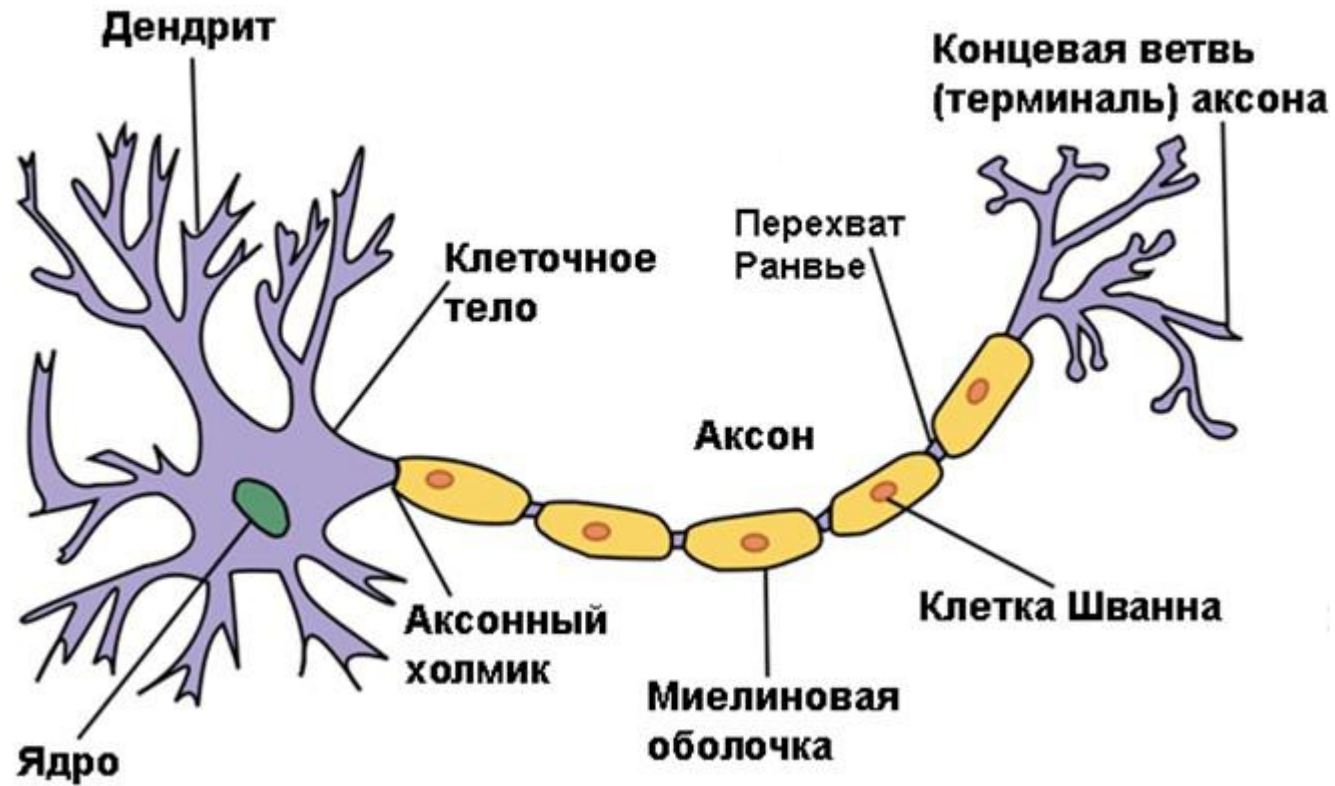


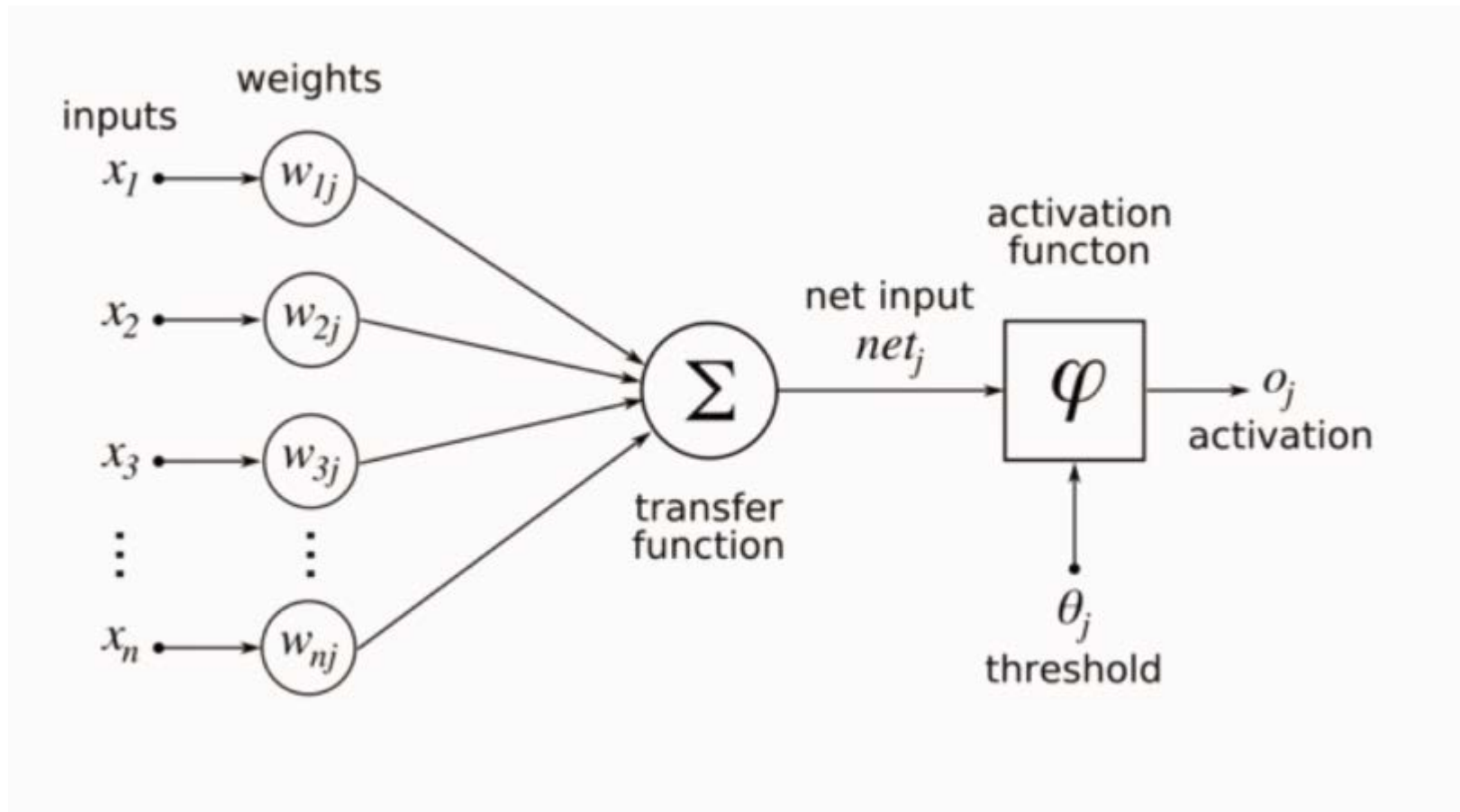
# *Искусственные нейронные сети в задачах интеллектуального анализа данных*

Курс «Интеллектуальные информационные системы»  
Кафедра управления и информатики НИУ «МЭИ»  
Осень 2017 г.

# Что такое нейрон

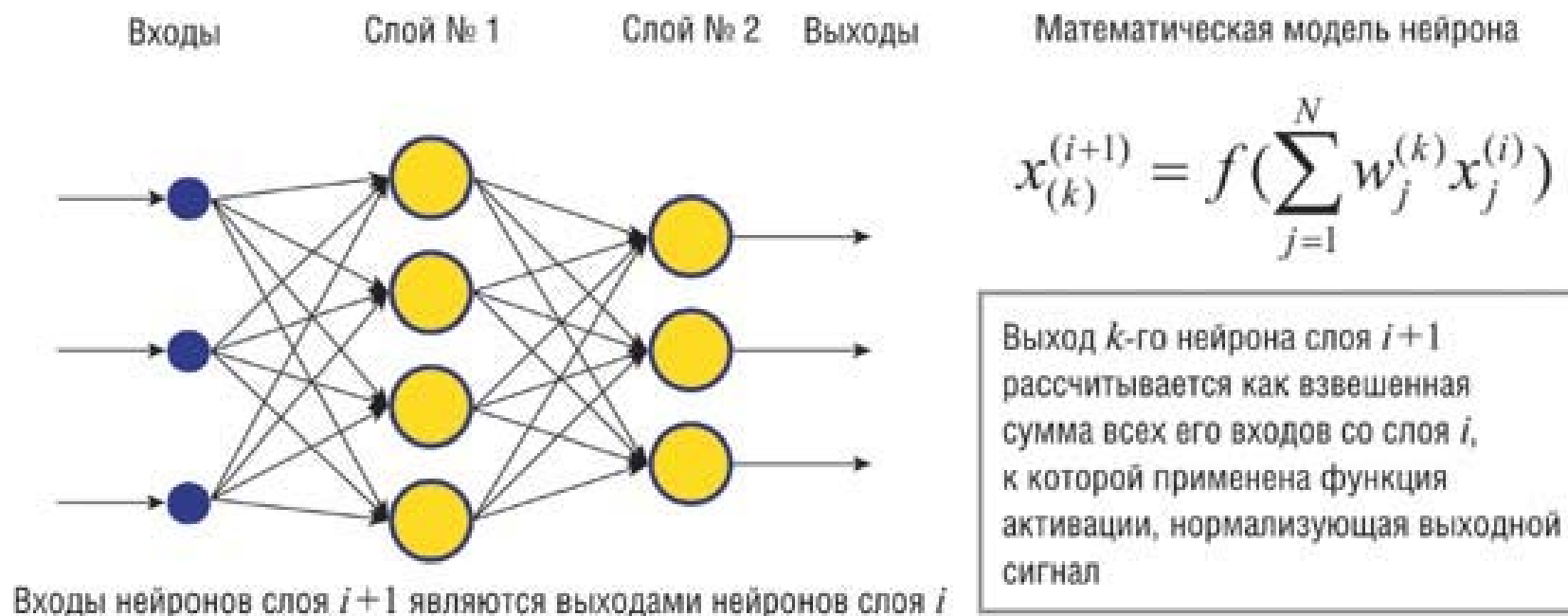


# Что такое нейрон с точки зрения инженера



# Искусственная нейронная сеть

Нейронная сеть (НС) — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования напрямую из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию



# Задачи, решаемые с помощью НС

Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Самыми распространенными применениями нейронных сетей является:

**Классификация** — распределение данных по параметрам. Например, на вход дается набор людей и нужно решить, кому из них давать кредит, а кому нет. Эту работу может сделать нейронная сеть, анализируя такую информацию как: возраст, платежеспособность, кредитная история и тд.

**Предсказание** — возможность предсказывать следующий шаг. Например, цену акций, основываясь на ситуации на фондовом рынке.

**Распознавание** — в настоящее время, самое широкое применение нейронных сетей. Используется в Google, когда вы ищете фото или в камерах телефонов, когда оно определяет положение вашего лица и выделяет его и многое другое.

# Сравнение нейронных сетей

- Человек:  $10^{10}$  нейронов
- Дельфин:  $6 \cdot 10^9$  нейронов
- Кошка:  $3 \cdot 10^8$  нейронов
- Мышь:  $4 \cdot 10^6$  нейронов
- Multi GPU Convolutional Network:  $10^6$  нейронов

[https://ru.wikipedia.org/wiki/Коэффициент\\_энцефализации](https://ru.wikipedia.org/wiki/Коэффициент_энцефализации)

# Как работает НС?



$w1 = 0.5$



$w2 = -0.5$



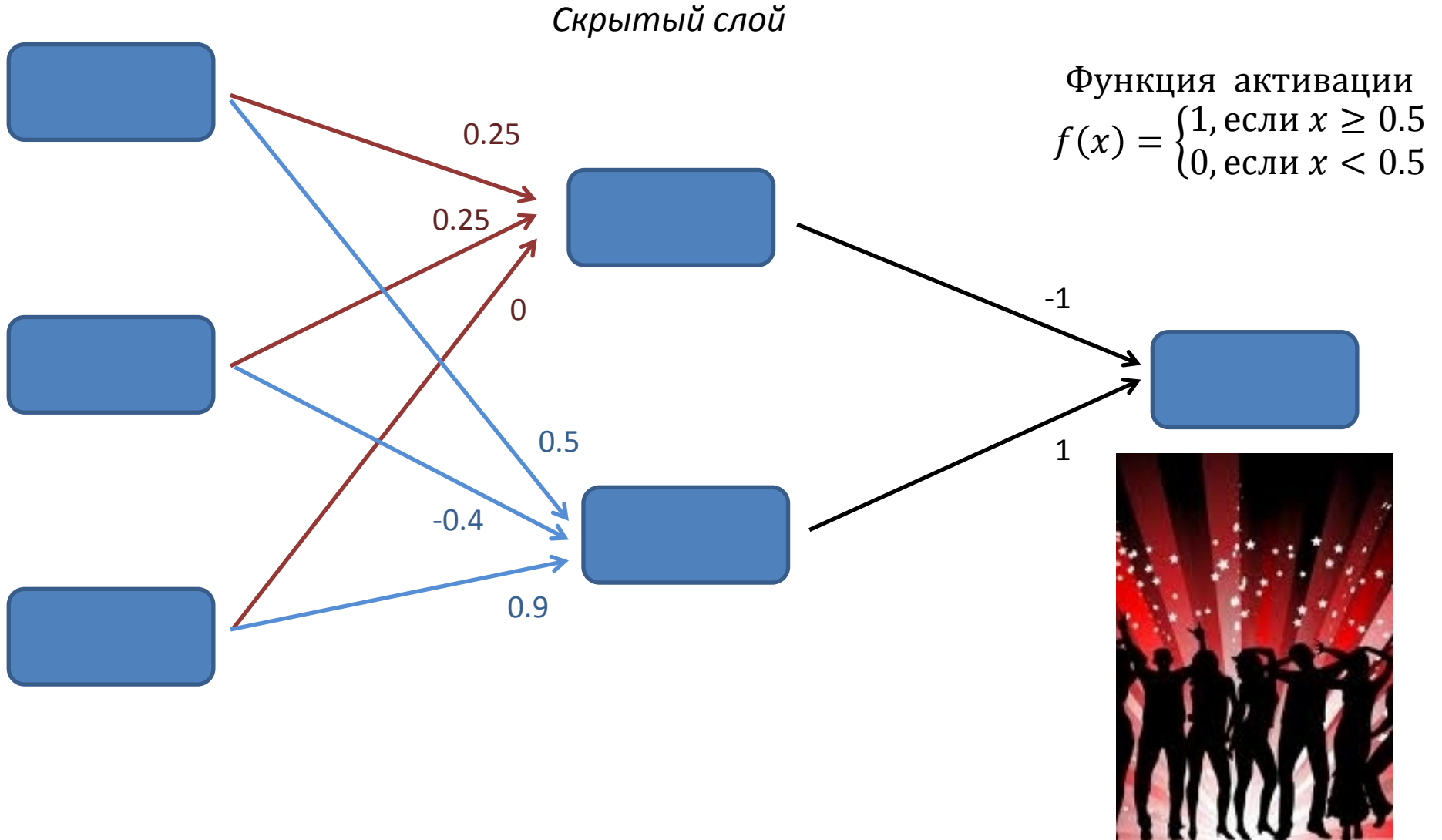
$w3 = 0.5$

Функция активации  
$$f(x) = \begin{cases} 1, & \text{если } x \geq 0.5 \\ 0, & \text{если } x < 0.5 \end{cases}$$

$$x = x1 * w1 + x2 * w2 + x3 * w3$$



# Как работает НС?



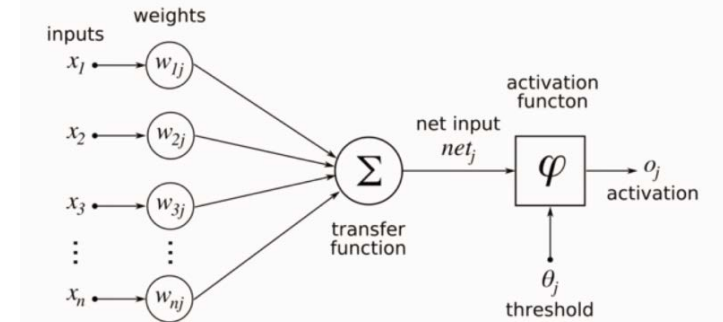
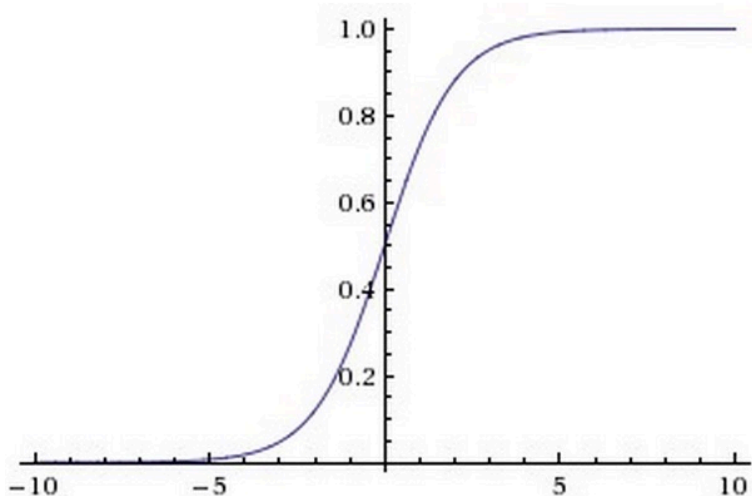


# Виды функции активации. Сигмоида

**Функция активации** (функция возбуждения) – функция, вычисляющая выходной сигнал искусственного нейрона. В качестве аргумента принимает сигнал, получаемый на выходе входного сумматора

Сигмоида (sigmoid) выражается следующей формулой:

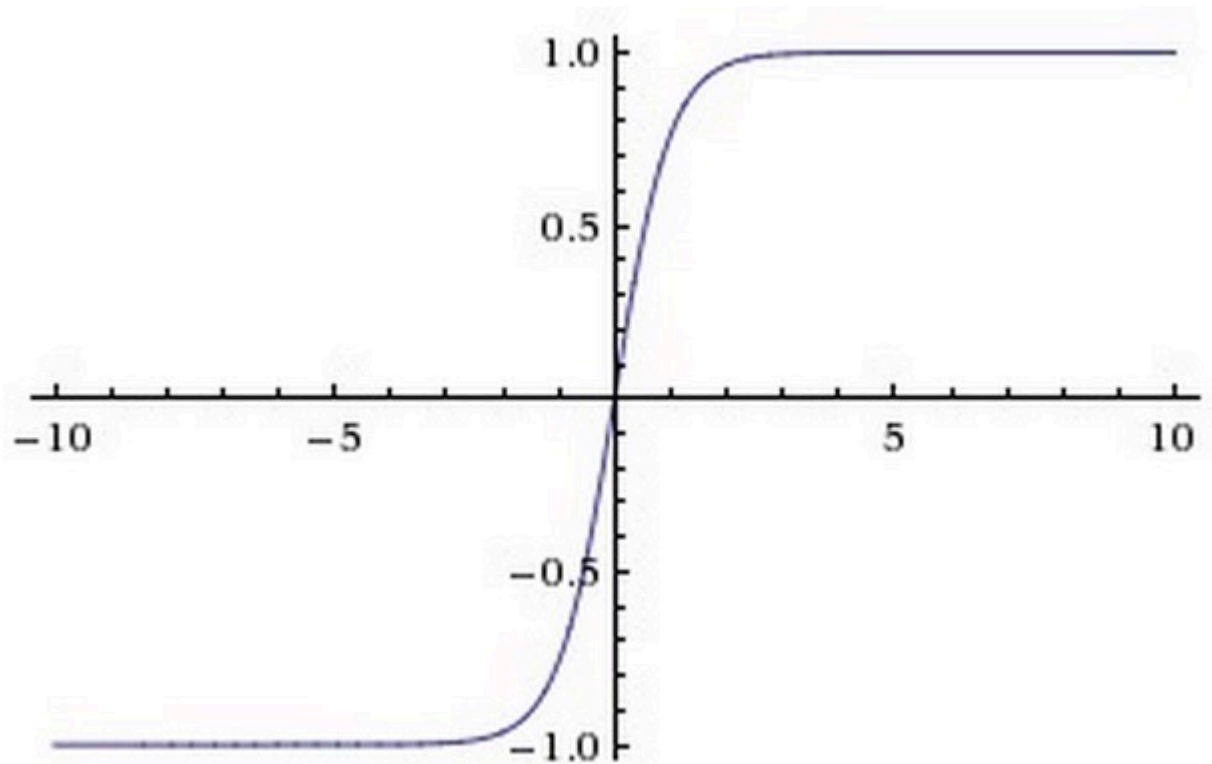
$$\sigma(x) = 1 / (1 + e^{-x})$$



На текущий момент сигмоида используется очень редко. Данная функция имеет два серьезных недостатка:

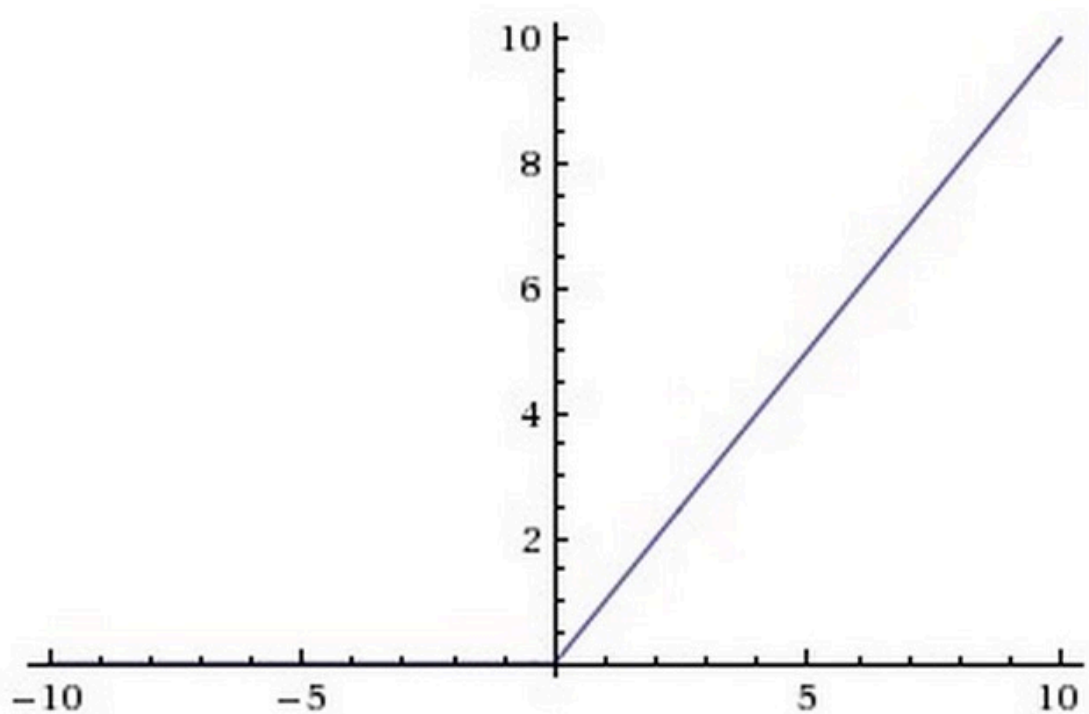
- *Насыщение сигмоиды приводит к затуханию градиентов.* Крайне нежелательное свойство сигмоиды заключается в том, что при насыщении функции с той или иной стороны (0 или 1), градиент на этих участках становится близок к нулю.
- *Выход сигмоиды не центрирован относительно нуля.*

# Виды функции активации. Гиперболический тангенс



Гиперболический тангенс (hyperbolic tangent,  $\tanh$ ) принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от  $-1$  до  $1$ . Подобно сигмоиде, гиперболический тангенс может насыщаться. Однако, в отличие от сигмоиды, выход данной функции центрирован относительно нуля. Следовательно, на практике всегда предпочтительнее использовать гиперболический тангенс, а не сигмоиду.

# Виды функции активации. ReLU



В последние годы большую популярность приобрела функция активации под названием «выпрямитель». Нейроны с данной функцией активации называются ReLU (rectified linear unit). ReLU имеет следующую формулу  $f(x) = \max(0, x)$  и реализует простой пороговый переход в нуле.

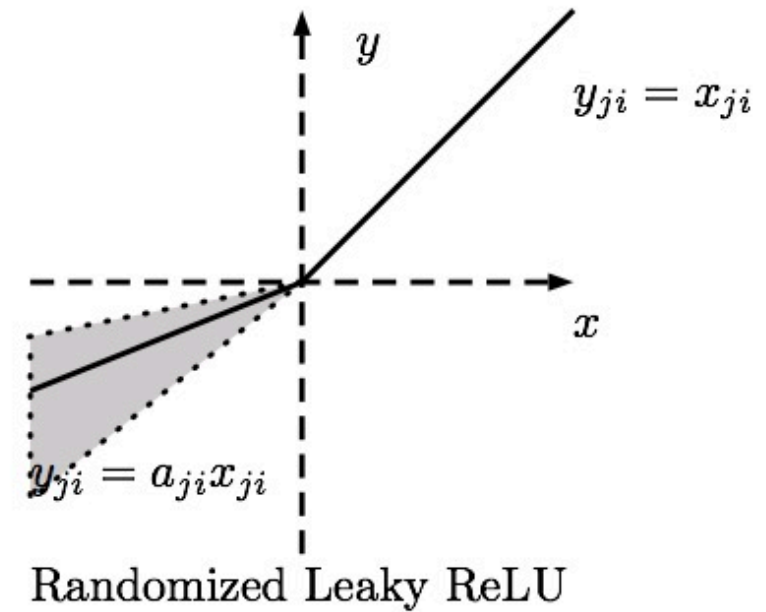
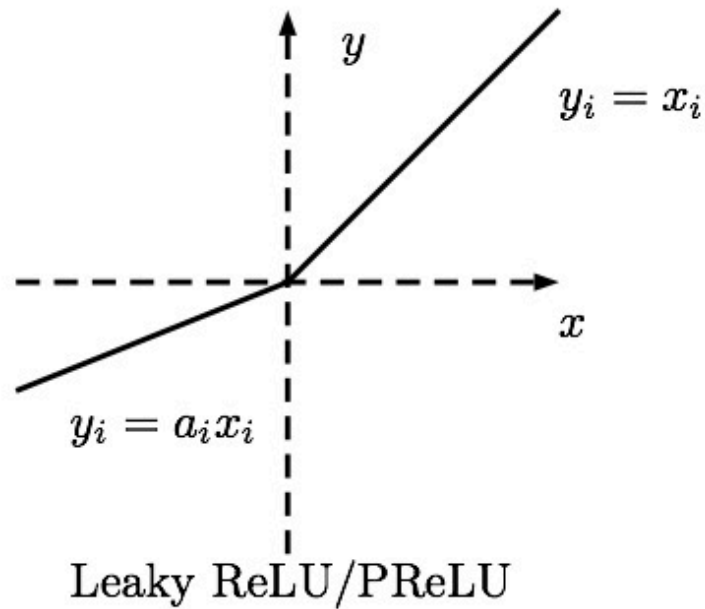
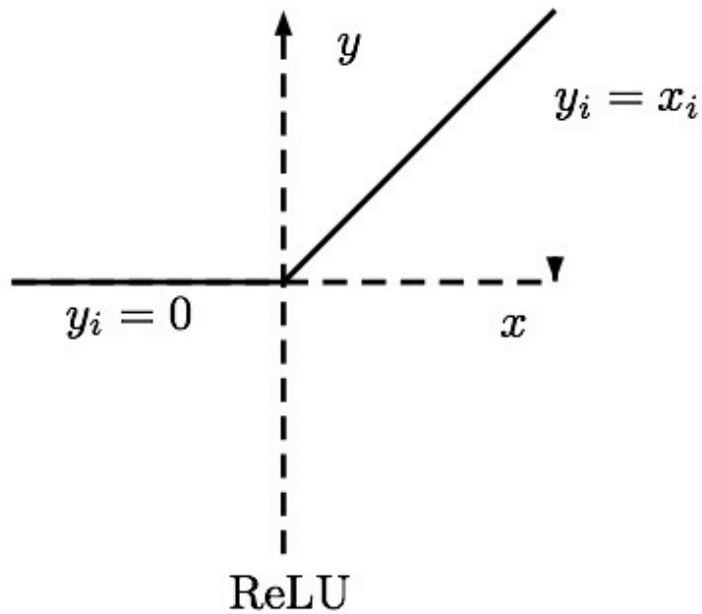
Плюсы:

- Проще вычислять, чем сигмоид и  $\tanh$
- Быстрее обучается за счет отсутствия насыщения

Минусы:

Могут «умирать» в процессе обучения, т.е. переходить в такое состояние, когда на выходе всегда будет 0.

# Виды функции активации. Модификации ReLU



Leaky ReLU - ReLU с «утечкой» представляет собой одну из попыток решить описанную выше проблему выхода из строя обычных ReLU. LReLU имеет на интервале  $x < 0$  небольшое отрицательное значение (угловой коэффициент около 0,01).

Randomized LReLU - угловой коэффициент на отрицательном интервале во время обучения генерируется случайным образом из заданного интервала, а во время тестирования остается постоянным

# Обучение нейронных сетей

Обучение НС – подбор весов  $w_1, \dots, w_n$  таким образом, чтобы сеть решала поставленную нами задачу.

Правила Хэбба (1949г.)

- Если сигнал персептрона неверен и равен нулю, то необходимо увеличить веса тех входов, на которые была подана единица.
- Если сигнал персептрона неверен и равен единице, то необходимо уменьшить веса тех входов, на которые была подана единица.

Будем полагать, что классы помечены числами 0 и 1:

Предъявляем на вход один объект. Если выходной сигнал персептрона совпадает с правильным ответом, то никаких действий предпринимать не надо. В случае ошибки необходимо обучить персептрон правильно решать данный пример.

Ошибки могут быть двух типов. Рассмотрим каждый из них.

Первый тип ошибки: на выходе персептрона  $a(x_i) = 0$ , правильный ответ  $y_i = 1$ .

Для того, чтобы персептрон выдавал правильный ответ необходимо, чтобы скалярное произведение стало больше. Поскольку переменные принимают значения 0 или 1, увеличение суммы может быть достигнуто за счет увеличения весов. Однако нет смысла увеличивать веса при переменных, которые равны нулю. Увеличиваем веса только при тех, которые равны 1. Для закрепления единичных сигналов с  $\omega$ , следует провести ту же процедуру и на всех остальных слоях.

Второй тип ошибки:  $a(x_i) = 1$ ,  $y_i = 0$ .

Для уменьшения скалярного произведения в правой части, необходимо уменьшить веса связей при тех переменных, которые равны 1. Необходимо также провести эту процедуру для всех активных нейронов предыдущих слоев.

# Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки (back propagation) - 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. Метод является модификацией классического метода градиентного спуска.

1. Инициализировать все веса  $w_{i,j}$  маленькими случайными значениями
2. Для каждого обучающего объекта  $x_i$ , для которого известен правильный ответ  $t_i$ :
  1. Подать обучающий объект  $x_i$  на вход сети и посчитать выходы  $o_i$  каждого узла.
  2. Для каждого нейрона  $k$  выходного слоя:
$$\delta_k = f'(x) * (t_k - o_k), \text{ где } f'(x) \text{ — производная от функции активации. Для сигмоида:}$$
$$\delta_k = o_k (1 - o_k) * (t_k - o_k)$$
$$(t_k - o_k) \text{ — ошибка нейрона}$$
3. Для каждого слоя  $l$ , начиная с предпоследнего:  
Для каждого узла  $j$  уровня  $l$  вычислить:

$$\delta_j = f'(x) \sum_{k \in \text{Children}(j)} \delta_k w_{j,k}$$

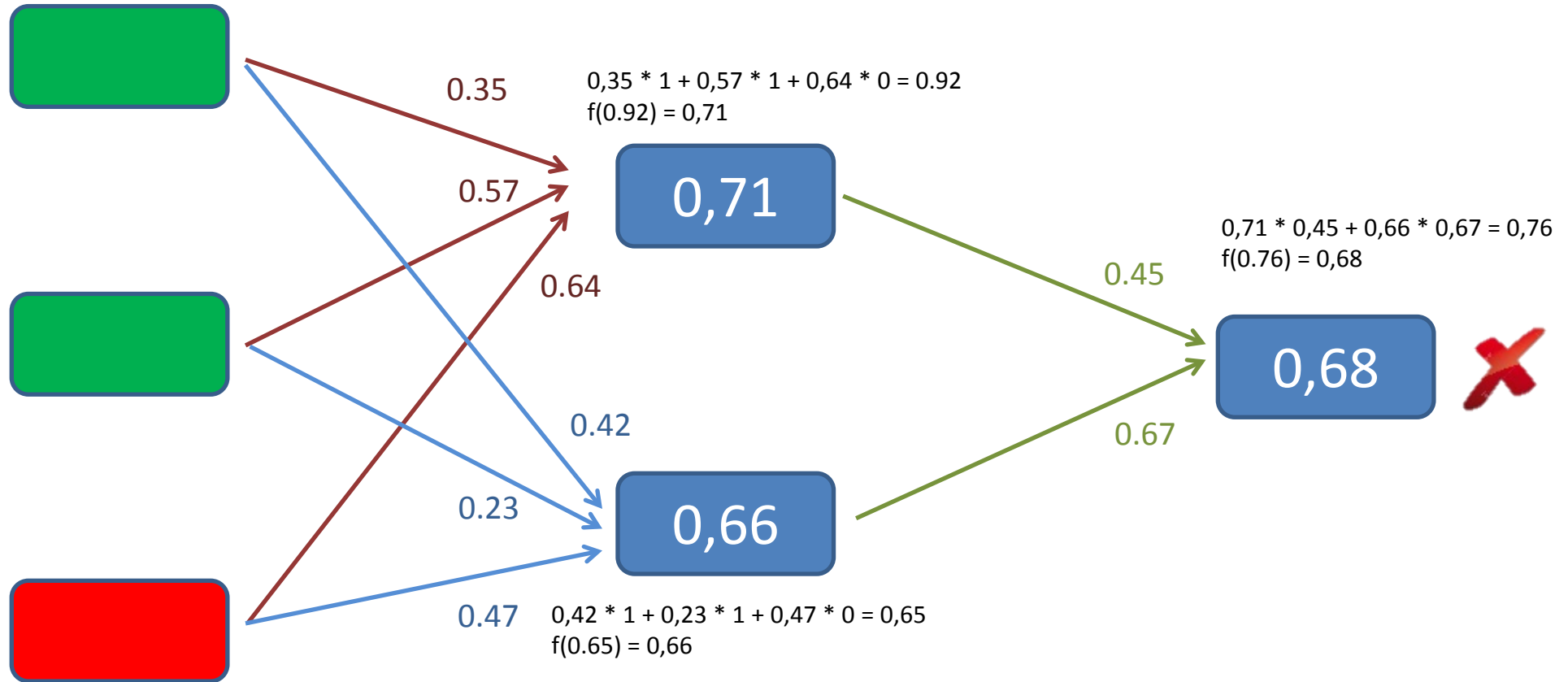
4. Для каждого веса нейрона:

$$w_{i,j} = w_{i,j} + \eta \delta_j o_{i,j}$$

# Алгоритм обратного распространения ошибки

Функция активации

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

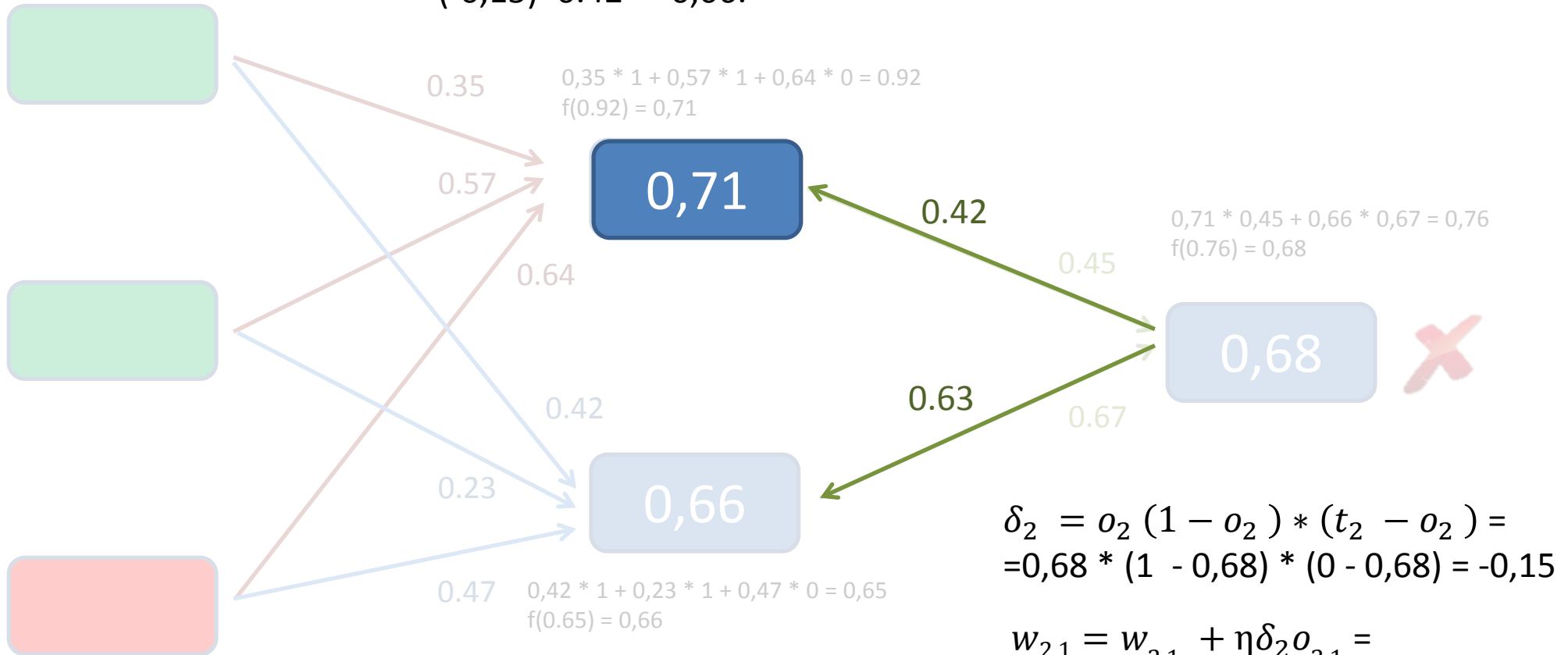
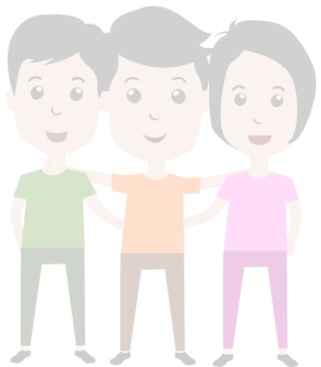


# Алгоритм обратного распространения ошибки

Функция активации

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

Ошибка нейрона (2,1) =  $\sum_{k \in \text{Children}(j)} \delta_k w_{j,k} =$   
 $= (-0,15) * 0,42 = -0,06$ .



$$\delta_2 = o_2 (1 - o_2) * (t_2 - o_2) =$$

$$= 0,68 * (1 - 0,68) * (0 - 0,68) = -0,15$$

$$w_{2,1} = w_{2,1} + \eta \delta_2 o_{2,1} =$$

$$= 0,45 + 0,3 * (-0,15) * 0,71 = 0,42$$

$$w_{2,2} = w_{2,2} + \eta \delta_2 o_{2,2} =$$


$$= 0,67 + 0,3 * (-0,15) * 0,66 = 0,63$$




# Обучающие выборки и эпохи

**Обучающая выборка** включает входные значения и соответствующие им выходные значения набора данных. В ходе обучения *нейронная сеть* находит некие зависимости выходных полей от входных.

**Эпоха обучения** - это один просмотр **всех примеров обучающей выборки** с одновременной коррекцией весов сети (на этих примерах, в зависимости от правильности их решения сетью). Чтобы сеть обучилась, может потребоваться несколько (обычно несколько десятков и сотен) эпох. В течении эпохи не на каждом обучающем примере может происходить коррекция весов сети: т.к. если сеть уже обучилась правильно решать данный конкретный пример, то можно перестать требовать постоянных "улучшений" качества решения этого примера (поскольку эти улучшения уже будут несущественными) и позволить сети направить её внимание на другие примеры, которые пока ещё решаются с недостаточной точностью.

 `for (int i=0;i<maxEpoch;i++)  
 for (int j=0;j<trainSet;j++)`

 `for (int j=0;j<trainSet;j++)  
 for (int i=0;i<maxEpoch;i++)`

**Важно** не путать итерацию с эпохой и понимать последовательность их инкремента.

Сначала  $n$  раз увеличивается итерация, а потом уже эпоха и никак не наоборот. Другими словами, нельзя сначала тренировать нейросеть только на одном примере, потом на другом и тд. Нужно тренировать каждый пример один раз за эпоху.