



# Введение в нейронные сети

Лектор – Дахова Елизавета



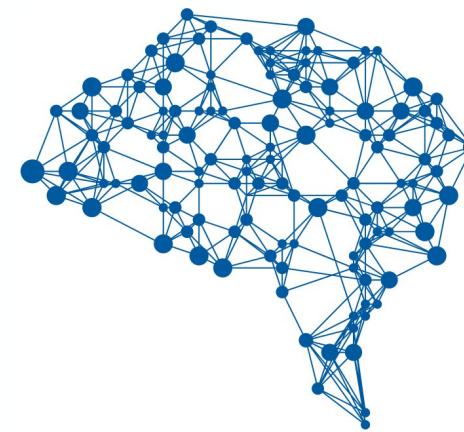
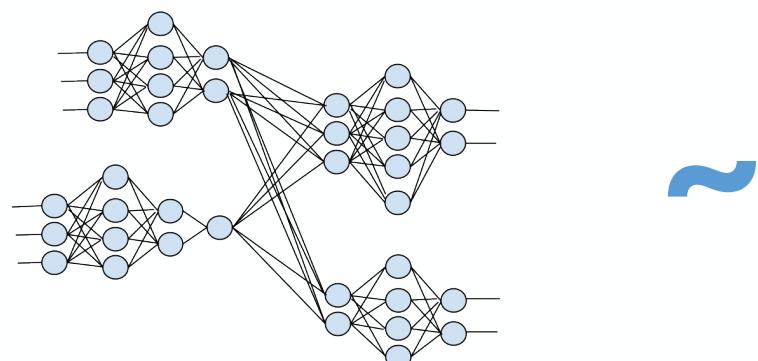
# Повторение



# Что такое нейронные сети?

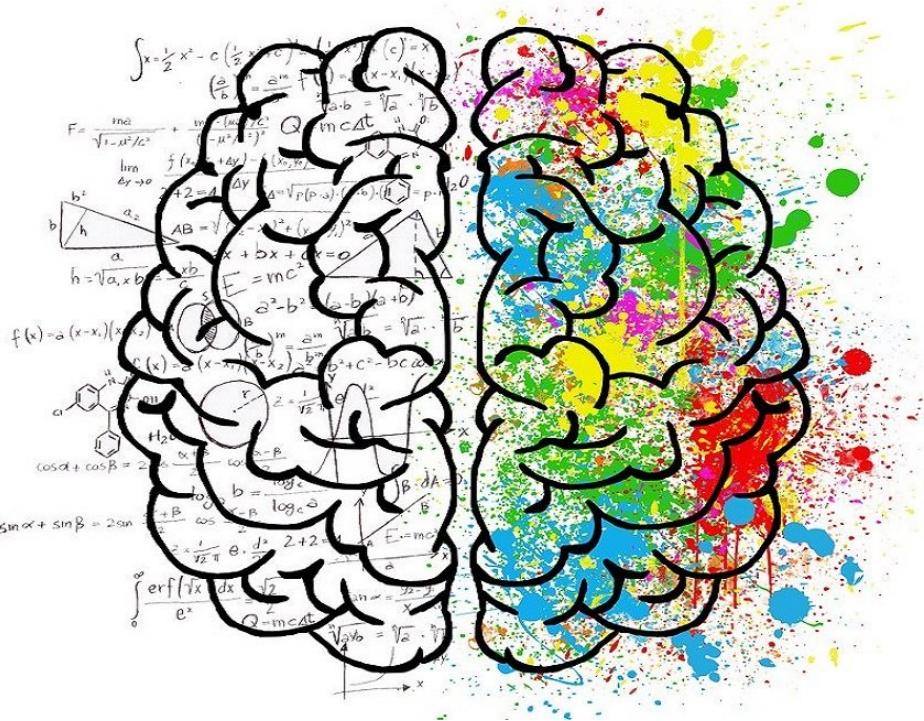
**Нейронная сеть** — математическая модель, а также ее программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

Это понятие возникло при изучении процессов, протекающих в мозге.





# Мозг



Мозг – это

- Система обработки информации
- Сложный, нелинейный, параллельный компьютер

Решает множество сложных задач

из области распознавания образов, обработки сигналов и прочее

# Примеры



Задача распознавания  
знакомого лица в толпе  
займет **0.1-0.2 секунды**



# Примеры

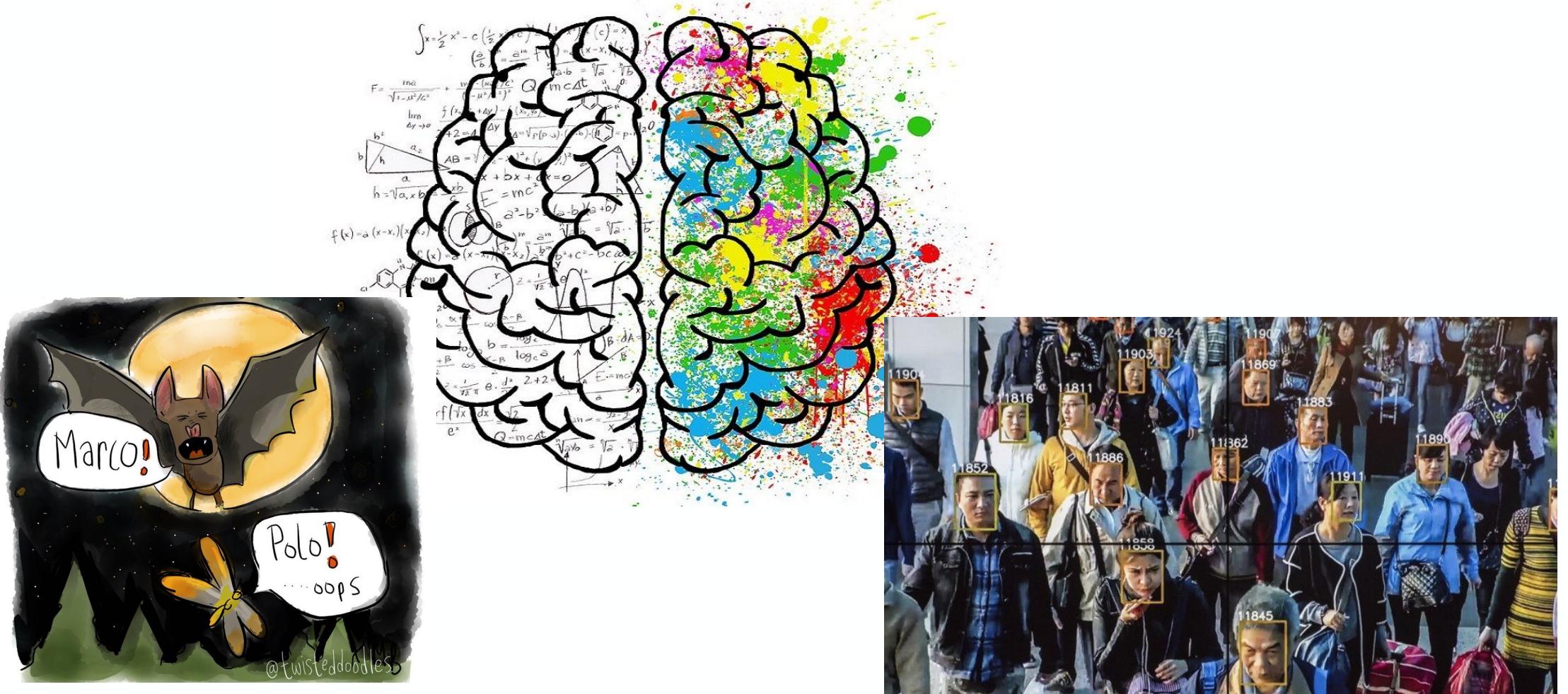
Эхо-локация:

летучая мышь получает информацию про положение, размер, скорость объекта и др..



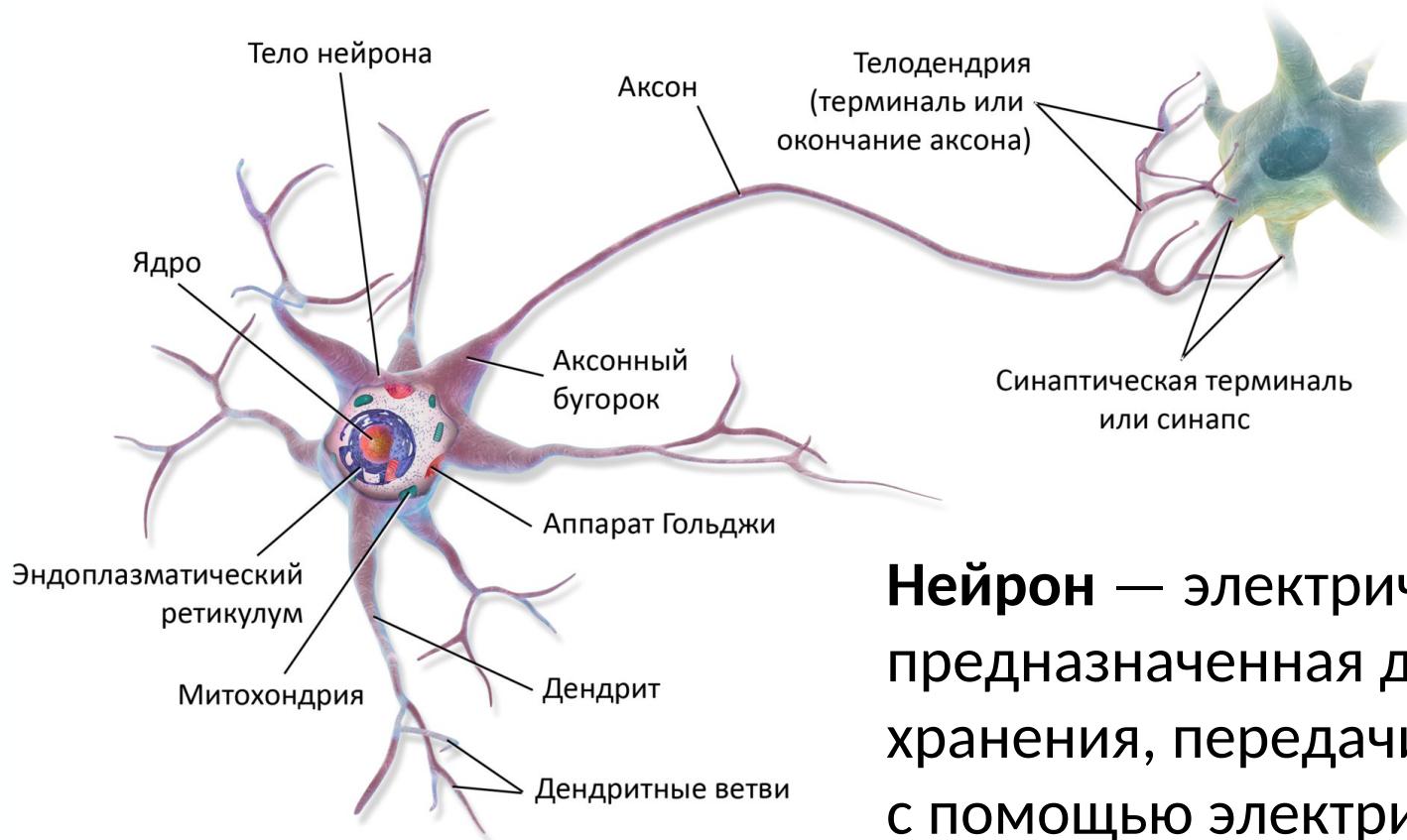


# Почему мы справляемся со столь сложными задачами?





# Биологический нейрон



**Структурно-функциональная единица нервной системы — это нейрон.**

**Нейрон** — электрически возбудимая клетка, предназначенная для приёма, обработки, хранения, передачи и вывода информации с помощью электрических и химических сигналов.



# Почему мы справляемся со столь сложными задачами?



Опыт



Нейронная сеть



Способность решать  
сложные задачи

Обучение  
/ training

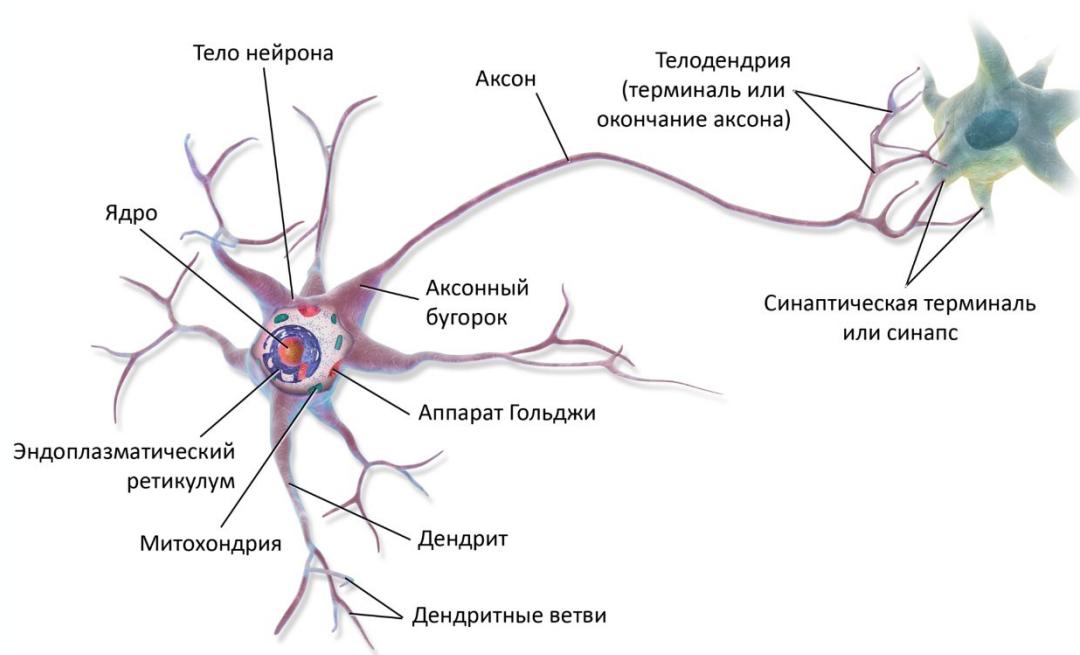
Применение  
/ inference



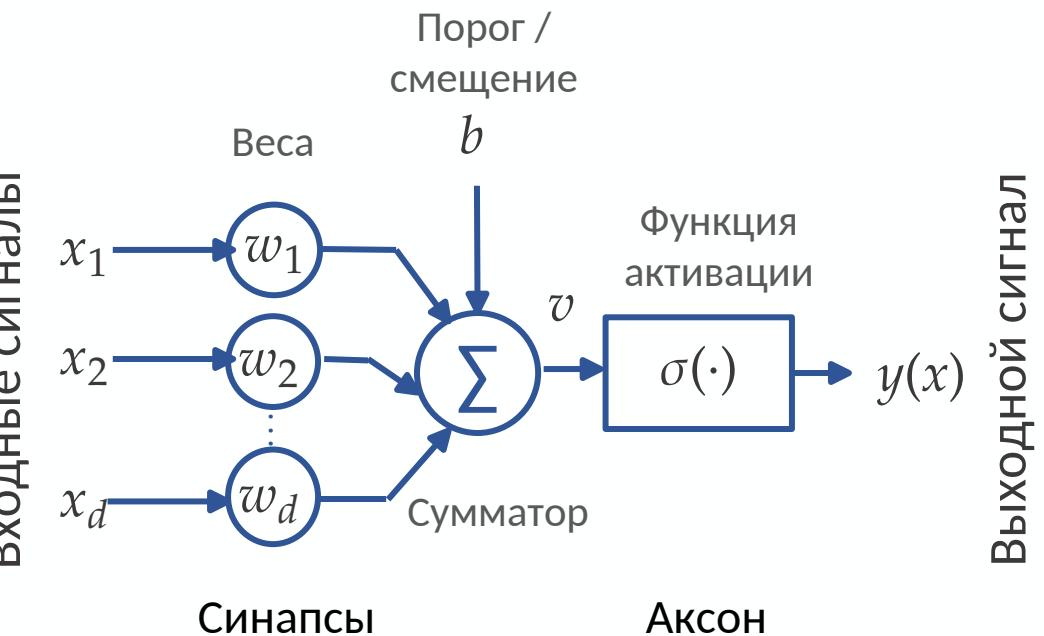


# Бионика: нейронные сети

## Биологический нейрон



## Искусственный нейрон



Нейрон накапливает заряд,  
и если заряд больше порога,  
то нейрон подает сигнал дальше.

Модель МакКалюка-Питца,  
где  $\sigma$  — пороговая функция активации



# Историческая справка

1943 г. - Мак-Каллок и Питц

- Модель нейрона как простейшего процессорного элемента.
- Предложена конструкция нейронов для логич. и арифм. операций
- Принципиальная возможность выполнить любые вычисления сетью нейронов.

1949 г. – Хебб

- Открытие принципа формирования нейронных связей, взаимодействия нейронов.
- Принцип обучения нейронной сети.

1958 г. – Розенблatt

- Модель «персептрана» - устройства, моделирующего процесс человеческого восприятия.



# Области применения

**Распознавание лиц / Face detection**

Цель — определить личность человека по снимку.



# Области применения

## Перенос стиля / Style transfer

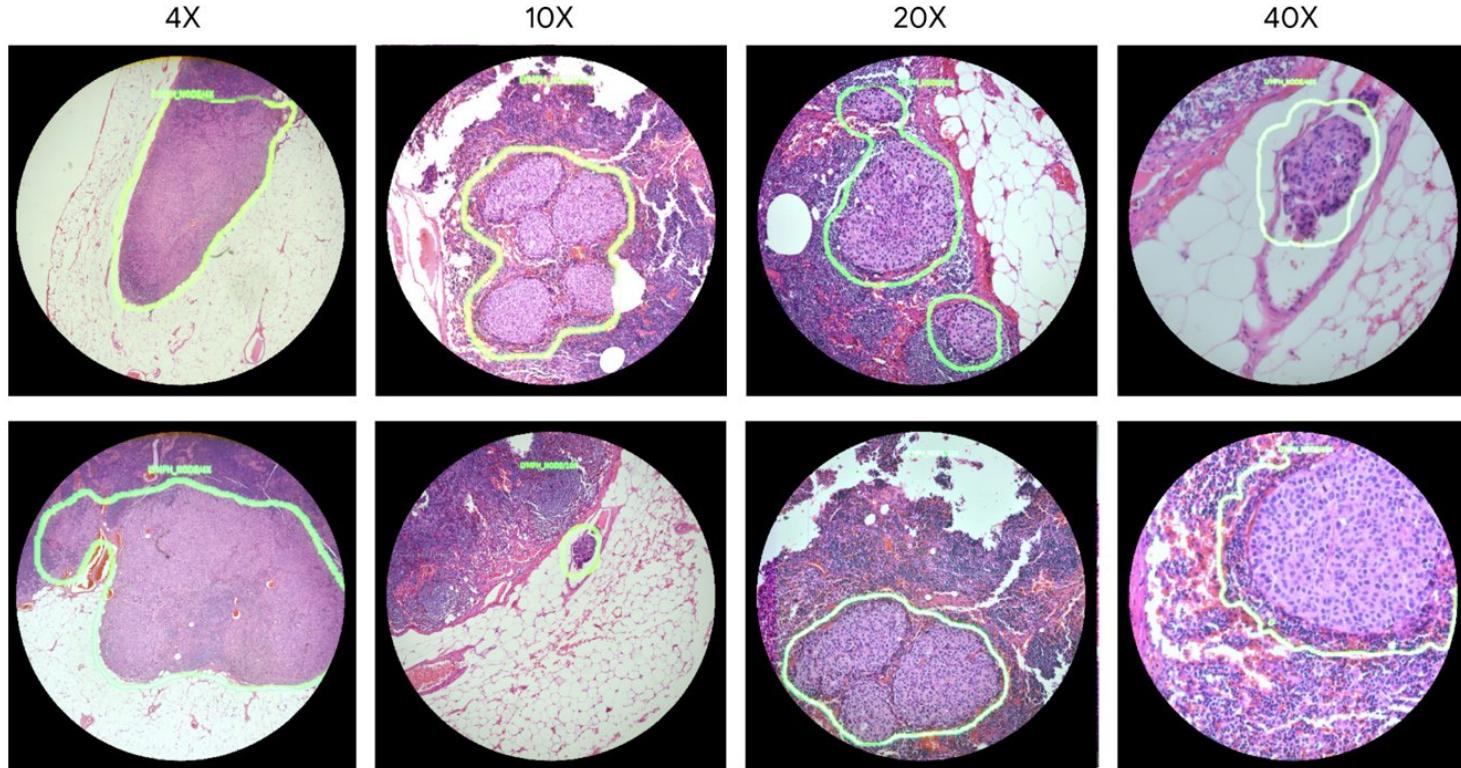
Цель — изменить изображение так, чтобы новое изображение сохранило структуру исходного, но переняло стиль целевого изображения.



# Области применения

**Сегментация опухоли / Tumor segementation**

Цель — для каждого пикселя определить принадлежит ли он опухоли или нет.





# Области применения

**Машинный перевод / Machine Language translation**

Цель — перевести текст на одном языке на другой язык.

The screenshot shows the Yandex.Translate website. At the top, there are tabs for 'Яндекс Переводчик' (Yandex Translator), 'Текст' (Text), 'Сайты' (Websites), 'Документы' (Documents), and 'Картинки' (Images). On the right, there are buttons for 'Для бизнеса' (For business) and a user profile icon. Below the tabs, the source language is set to 'АНГЛИЙСКИЙ' (English) and the target language is 'РУССКИЙ' (Russian). A double-headed arrow icon is between them. In the main translation area, the English sentence 'neural networks are so cool!' is on the left, and its Russian translation 'нейронные сети - это так круто!' is on the right. There are 'X' and 'undo' icons between the two text boxes. At the bottom, there are audio recording icons, character count indicators ('28 / 10000'), a keyboard icon, another audio icon, and a 'Перевести в Google' (Translate to Google) button. To the right of the translate button are icons for bookmarking, sharing, and rating (like and dislike).



# Области применения

**Распознавание речи / speech recognition**

Цель — перевести человеческую речь в текст.





# Области применения

**Прогноз погоды / whether forecasting**

Цель — предсказать температуру, осадки, скорость и направление ветра и тд.





# Области применения

**Предсказание продаж / sales prediction**

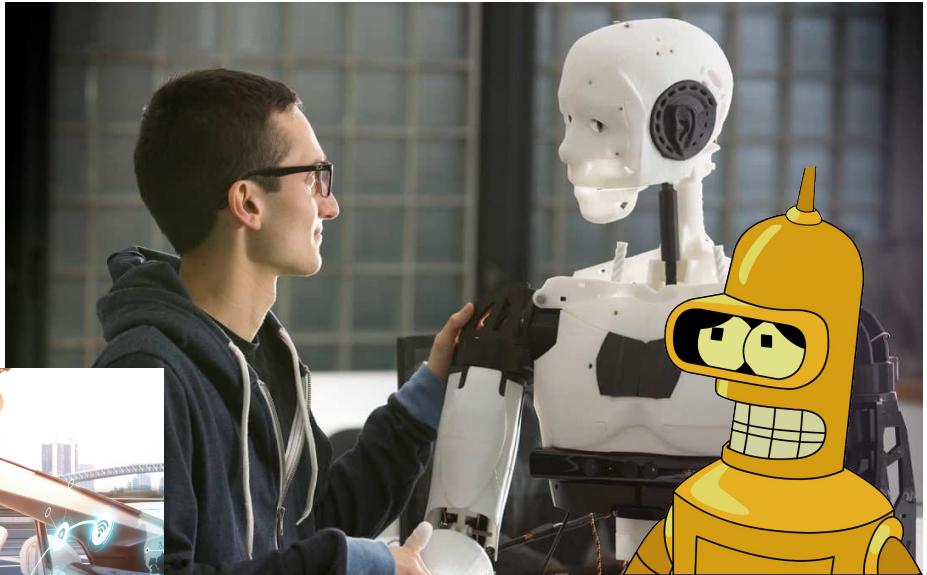
Цель — предсказать продажи товаров на некоторый период времени вперед.





# Области применения

И многое другое...



# Модель нейрона

Обозначим

$x = (x_1, \dots, x_d)^T \in \mathbb{R}^d$  — один объект, где  $x_1, \dots, x_d$  — признаки;

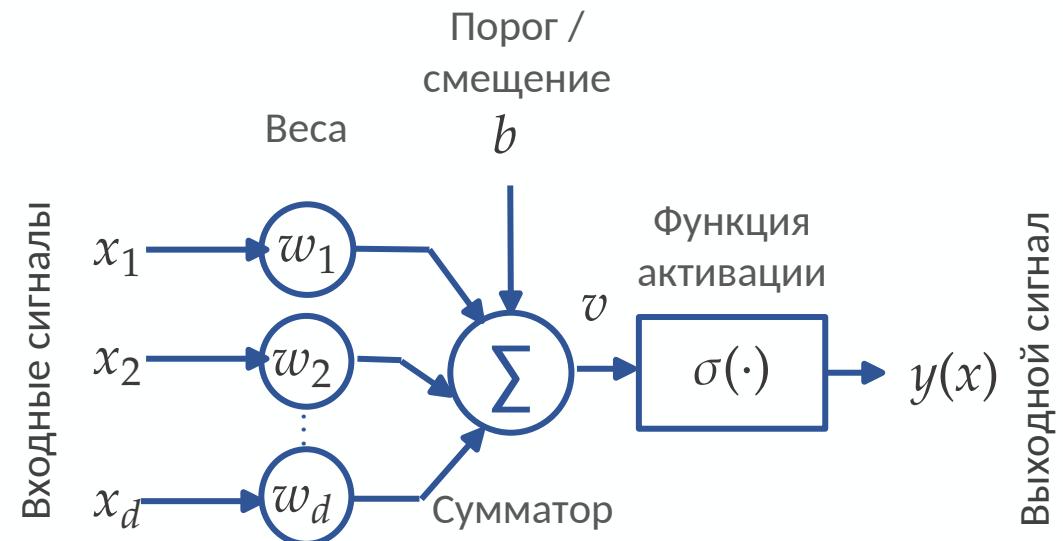
$w = (w_1, \dots, w_d)^T \in \mathbb{R}^d$  — вектор весов;

$b \in \mathbb{R}$  — смещение.

Выход нейрона —

$$y(x) = \sigma(\langle x, w \rangle + b) = \sigma\left(\sum_{j=1}^d w_j x_j + b\right),$$

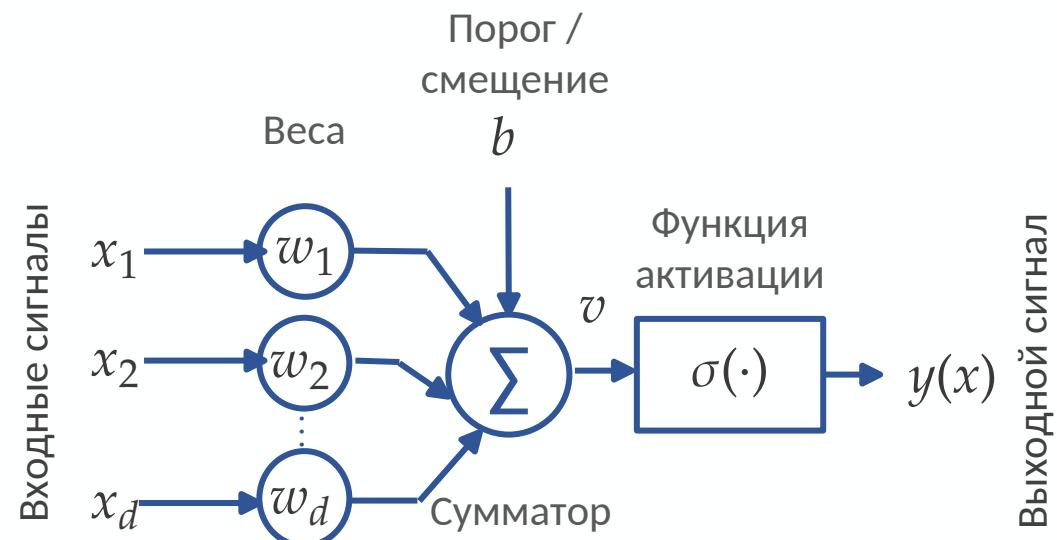
где  $\sigma$  — некоторая кус.-дифф. функция, назовем ее **функцией активации**.



# Что-то знакомое...

На что похожа эта формула?

$$y(x) = \sigma(\langle x, w \rangle + b) = \sigma\left(\sum_{j=1}^d w_j x_j + b\right)$$



# Что-то знакомое...

На что похожа эта формула?

$$y(x) = \sigma(\langle x, w \rangle + b) = \sigma\left(\sum_{j=1}^d w_j x_j + b\right)$$

Линейная регрессия для 1 эл.

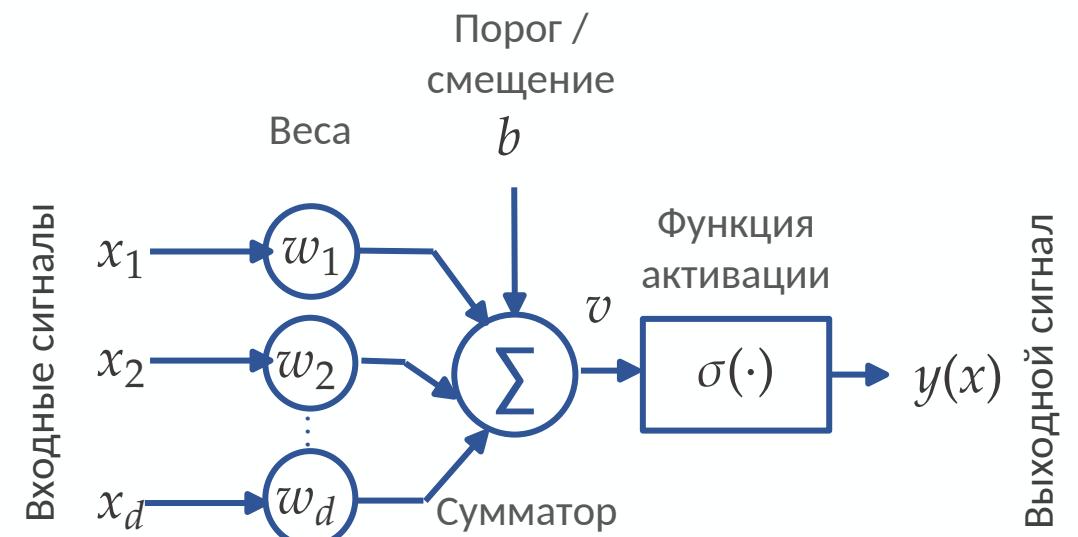
$$y(x) = \langle x, w \rangle + b = \sigma(\langle x, w \rangle + b) = \sigma\left(\sum_{j=1}^d x_j w_j + b\right), \quad \text{где } \sigma(z) = z \text{ — линейная ф-я.}$$

Логистическая регрессия для 1 эл.

$$y(x) = \sigma(\langle x, w \rangle + b) = \sigma\left(\sum_{j=1}^d x_j w_j + b\right), \quad \text{где } \sigma(z) = \frac{1}{1+e^{-z}} \text{ — логистическая сигмоида.}$$

Пуассоновская регрессия для 1 эл.

$$y(x) = \sigma(\langle x, w \rangle + b) = \sigma\left(\sum_{j=1}^d x_j w_j + b\right), \quad \text{где } \sigma(z) = e^z.$$



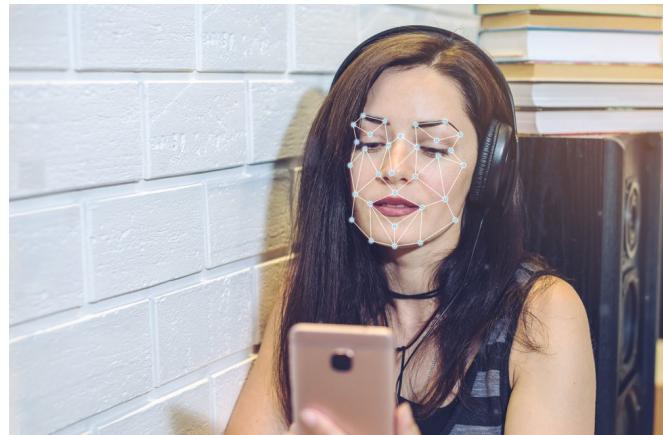


# Более сложные задачи

- У линейной и логистической регрессий ограниченная область применения.
- Для того, чтобы решить нелинейную задачу,  
нужно делать сложные преобразования с признаками.
- Один нейрон не справится со сложными задачами... 😞

Вспомним, что в нервной системе очень много нейронов.

Значит, будем использовать **больше нейронов!**



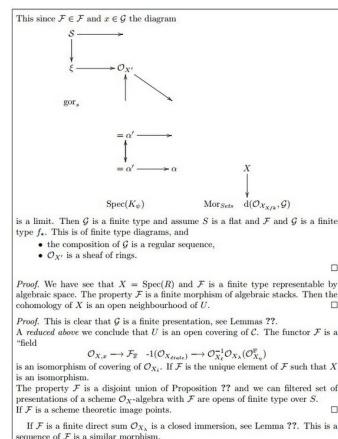
Proof. Omitted. □

**Lemma 0.1.** Let  $\mathcal{C}$  be a set of the construction.  
Let  $\mathcal{C}$  be a gerber covering. Let  $\mathcal{F}$  be a quasi-coherent sheaves of  $\mathcal{O}$ -modules. We have to show that  
$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

*Proof.* This is an algebraic space with the composition of sheaves  $\mathcal{F}$  on  $X_{\text{étale}}$  we have  
$$\mathcal{O}_{\mathcal{O}_X}(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$
 where  $\mathcal{G}$  defines an isomorphism  $\mathcal{F} \rightarrow \mathcal{G}$  of  $\mathcal{O}$ -modules. □

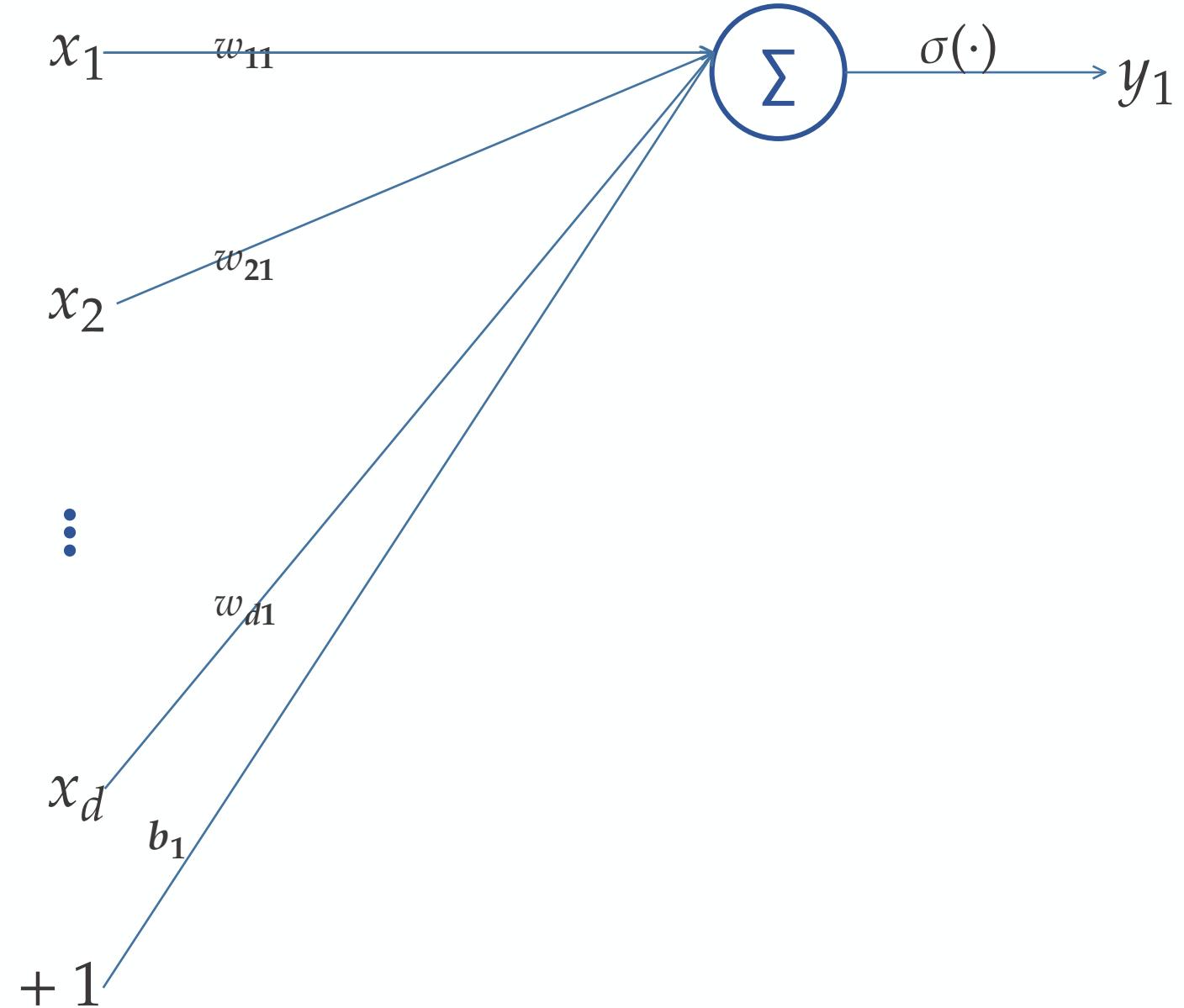
**Lemma 0.2.** This is an integer  $Z$  is injective.  
*Proof.* See Spaces, Lemma ???. □

**Lemma 0.3.** Let  $S$  be a scheme. Let  $X$  be a scheme and  $X$  is an affine open covering. Let  $\mathcal{U} \subset X$  be a canonical and locally of finite type. Let  $X$  be a scheme. Let  $X$  be a scheme which is equal to the formal complex.  
The following to the construction of the lemma follows.  
Let  $X$  be a scheme. Let  $X$  be a scheme covering. Let  
$$b: X \rightarrow Y' \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$
 be a morphism of algebraic spaces over  $S$  and  $Y$ .  
Proof. Let  $X$  be a nonzero scheme of  $X$ . Let  $X$  be an algebraic space. Let  $\mathcal{F}$  be a quasi-coherent sheaf of  $\mathcal{O}_X$ -modules. The following are equivalent  
(1)  $\mathcal{F}$  is an algebraic space over  $S$ .  
(2) If  $X$  is an affine open covering.  
Consider a common structure on  $X$  and  $X$  the functor  $\mathcal{O}_X(U)$  which is locally of finite type. □



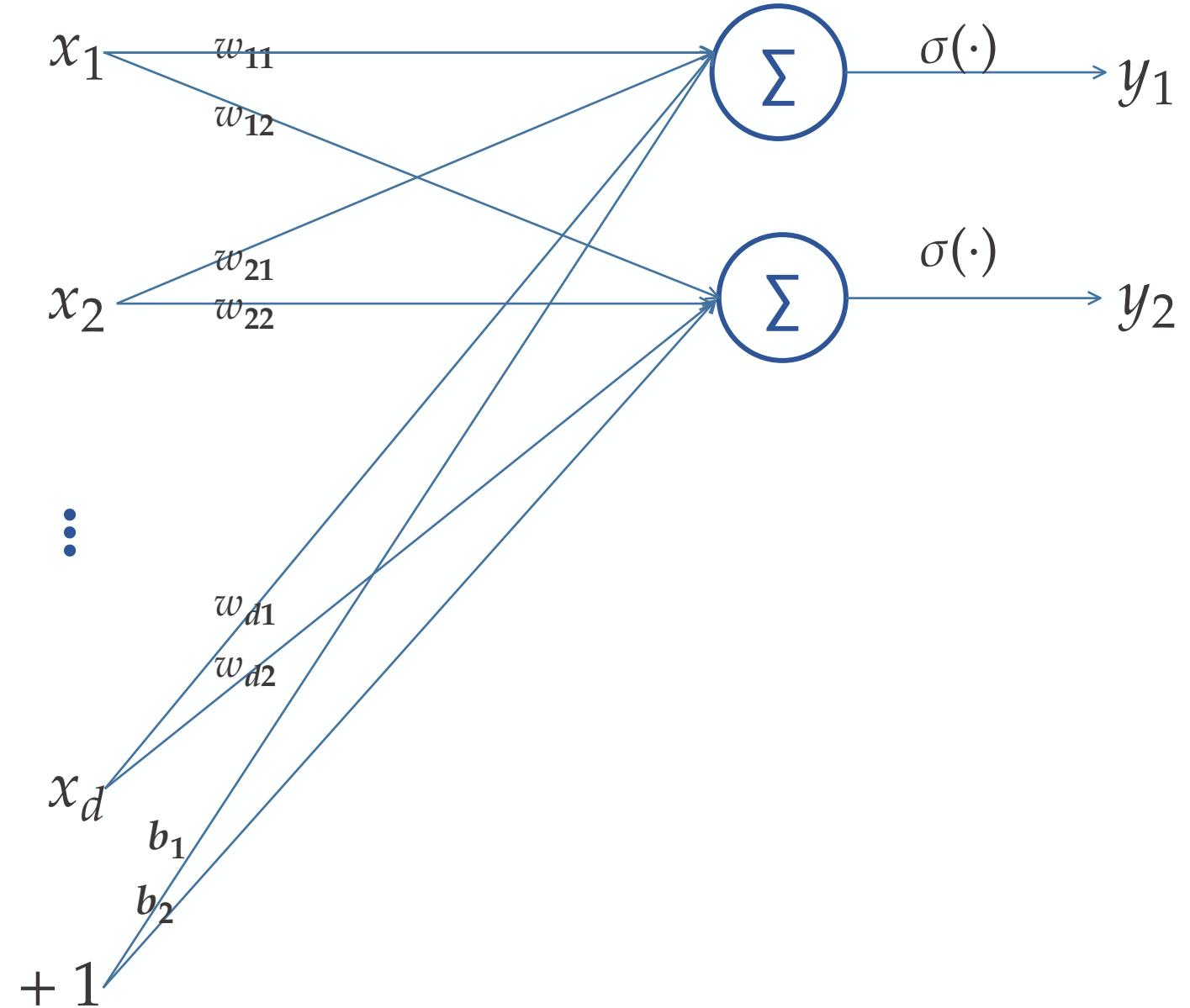
# Однослойная нейронная сеть

Один нейрон  
с весами  $w_{11}, \dots, w_{d1}$   
и смещением  $b_1$ .



# Однослойная нейронная сеть

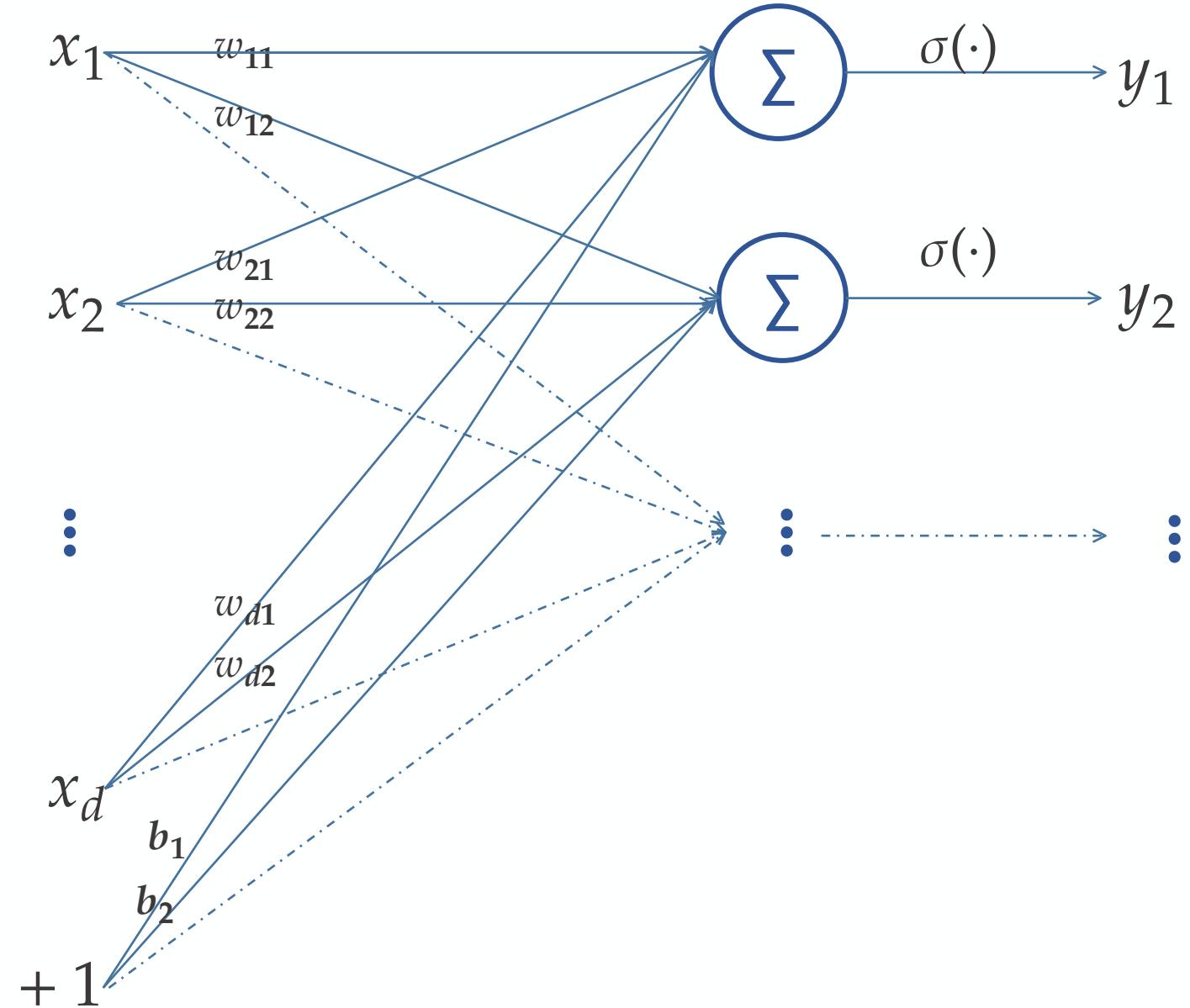
Два нейрона  
с весами  $w_{11}, \dots, w_{d1}$   
и  $w_{12}, \dots, w_{d2}$   
и смещениями  $b_1$  и  $b_2$





# Однослойная нейронная сеть

Больше нейронов



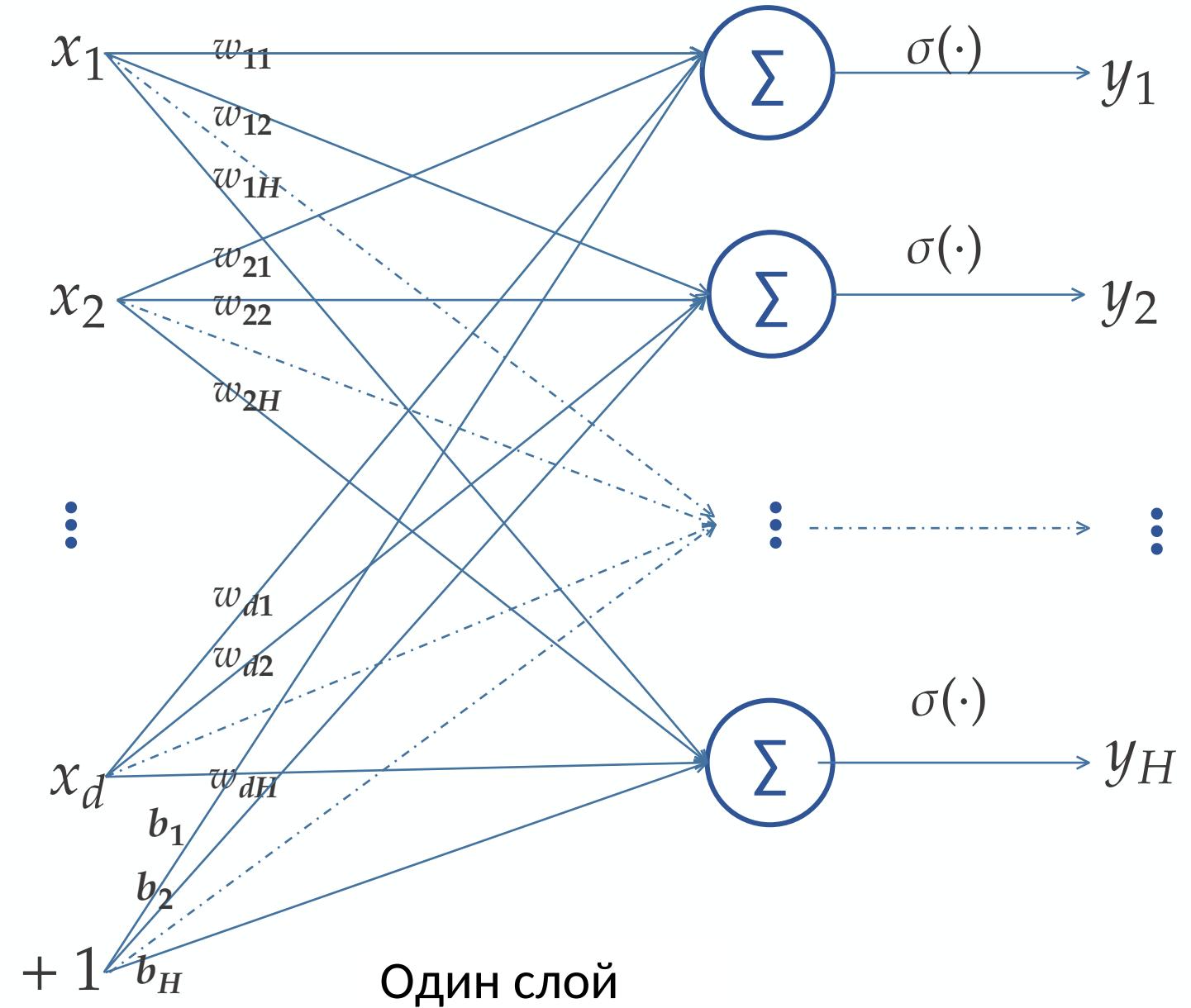


# Однослойная нейронная сеть

Слой размера  $H$   
— набор из  $H$  нейронов.

$(w_{jh})_{jh} \in \mathbb{R}^{d \times H}$  и  $(b_h)_h$   
— параметры модели

Такую нейронную сеть  
(слой нейронной сети)  
называют  
**полносвязной(ым)**





# Однослойная нейронная сеть

## Матричное представление

Пусть  $x = (x_1, x_2, \dots, x_d)$  — элемент выборки.

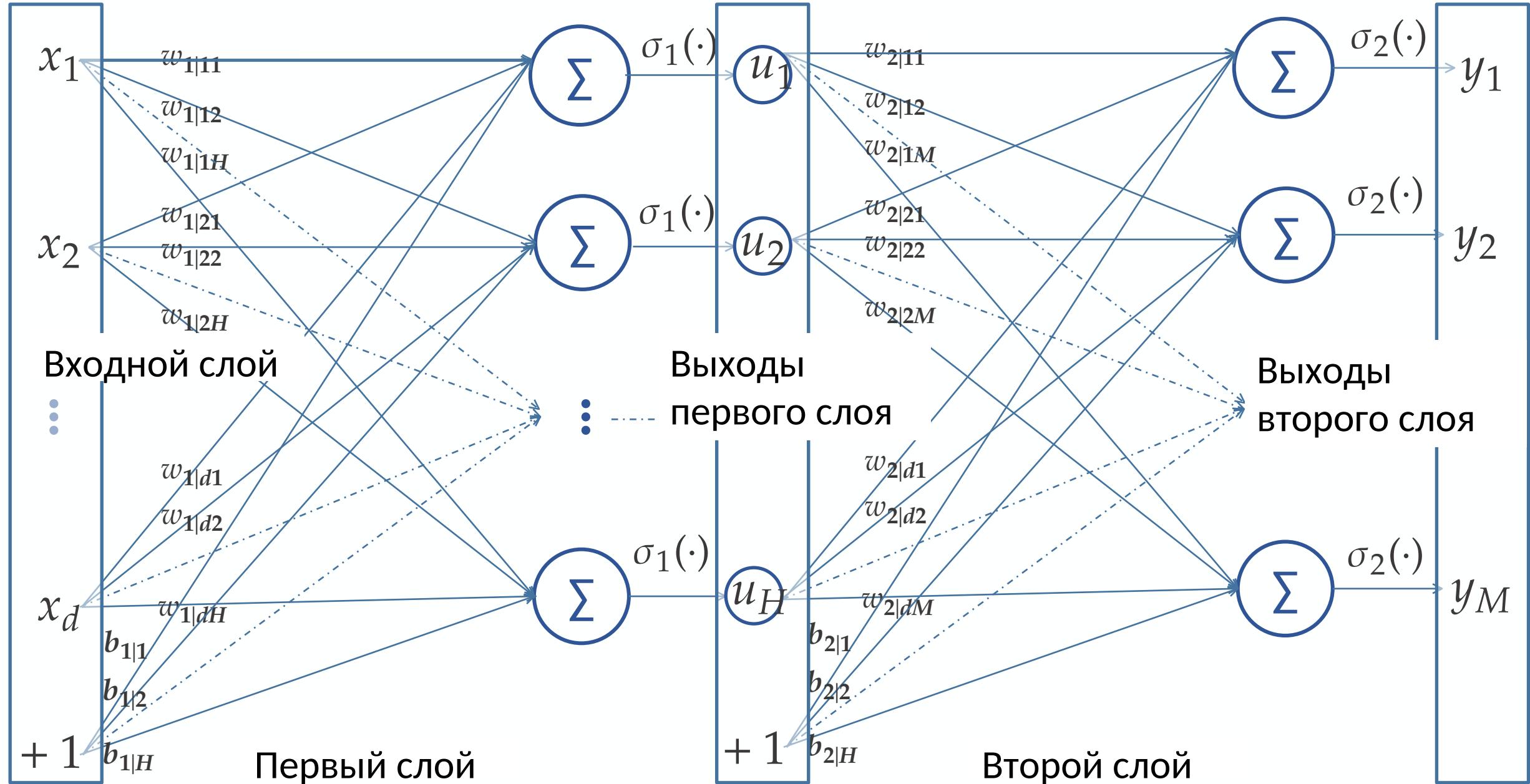
$s = (s_1, s_2, \dots, s_H)$  — выходы нейронов до применения функции активации.

$y = (y_1, y_2, \dots, y_H)$  — выход слоя.

Тогда работу слоя можно описать операциями:

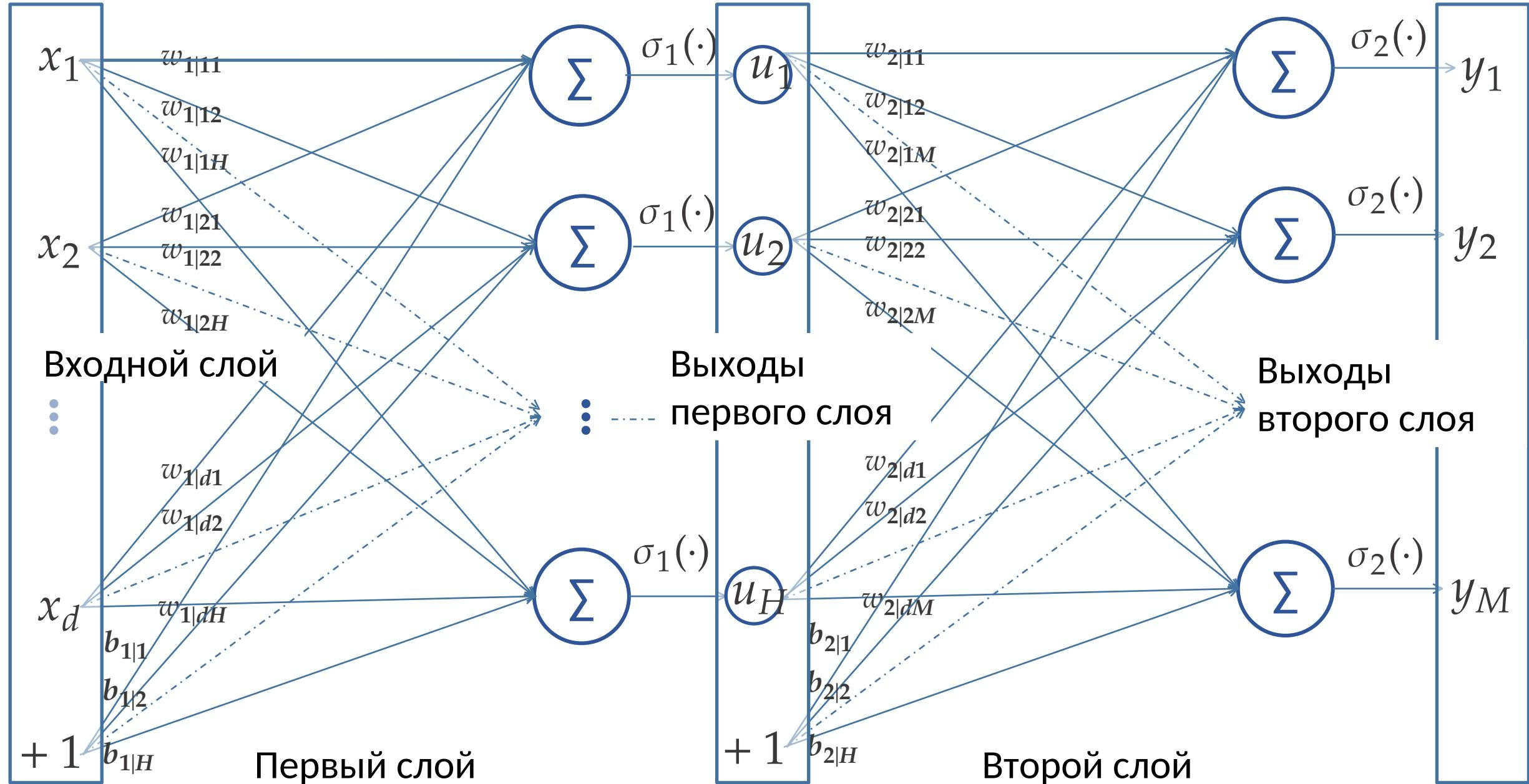
$$1) s = (x_1, x_2, \dots, x_d) \times \begin{pmatrix} w_{11} & \cdots & w_{1h} & \cdots & w_{1H} \\ w_{21} & \cdots & w_{2h} & \cdots & w_{2H} \\ \dots & \dots & \dots & \dots & \dots \\ w_{d1} & \cdots & w_{dh} & \cdots & w_{dH} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_H \end{pmatrix}^T = x^T \cdot W + b^T$$

$$2) y = (y_1, y_2, \dots, y_H) = (\sigma(s_1), \sigma(s_2), \dots, \sigma(s_H)) = \sigma(s) = \boxed{\sigma(x^T W + b^T)}$$





# Двухслойная нейронная сеть





# Двухслойная нейронная сеть

## Матричное представление

Пусть  $x = (x_1, x_2, \dots, x_d)$  — элемент выборки,

$u = (s_1, s_2, \dots, s_H)$  — выход I слоя,  $y = (y_1, y_2, \dots, y_M)$  — выход II слоя,

$W_1 = (w_{1|jh})_{jh}$  — м-ца весов I слоя,  $W_2 = (w_{2|hm})_{hm}$  — м-ца весов II слоя,

$b_1 = (b_{1|1}, \dots, b_{1|H})^T$  — в-р сдвигов I слоя,  $b_2 = (b_{2|1}, \dots, b_{2|H})^T$  — в-р сдвигов II слоя.

Тогда работу двухслойной нейронной сети можно представить как:

$$1) u = \sigma_1(x^T W_1 + b_1^T)$$

$$2) y = \sigma_2(u^T W_2 + b_2^T) = \boxed{\sigma_2\left(\sigma_1\left(x^T W_1 + b_1^T\right)^T W_2 + b_2^T\right)}$$



# Двухслойная нейронная сеть

Назовем функцию  $\sigma(z)$  сигмоидой, если  $\lim_{z \rightarrow -\infty} \sigma(z) = 0$  и  $\lim_{z \rightarrow +\infty} \sigma(z) = 1$ .

$\sigma(z) = \frac{e^z}{1+e^z}$  — логистическая сигмоида, один из примеров такой функции.

## Теорема (Цыбенко, 1989)

Если  $\sigma(z)$  — непрерывная сигмоида, то для любой непрерывной на  $[0, 1]^d$  функции  $f(x)$  существуют такое  $H$  и значения параметров  $w_{1,h} \in \mathbb{R}^d$ ,  $w_{2,h} \in \mathbb{R}^d$ ,  $b \in \mathbb{R}$ ,

что двухслойная нейросеть  $g(x) = \sum_{h=1}^H w_{2,h} \cdot \sigma(x^T w_{1,h} + b)$

равномерно приближает  $f(x)$  с любой точностью  $\varepsilon$ :

$|g(x) - f(x)| < \varepsilon$ , для всех  $x \in [0, 1]^d$



# Двухслойная нейронная сеть

## Выводы

- С помощью линейных операций и функций активаций  $\sigma$  от одного аргумента можно вычислять **любую непрерывную функцию** на заданном интервале с любой желаемой точностью.
- **Двух слоев** в нейронной сети теоретически достаточно.

## Замечания

- Теорема ничего не говорит о количестве нейронов в каждом слое, о значении весов и сдвигов, и виде функции активации.
- Двумя слоями такая цель теоретически достигается, но сложно.

Только в одной коре головного мозга число слоев равно 6.

- Дополнительные слои - удобный способ преобразования признаков, переход из одного признакового пространства в более удобное для решения задачи.

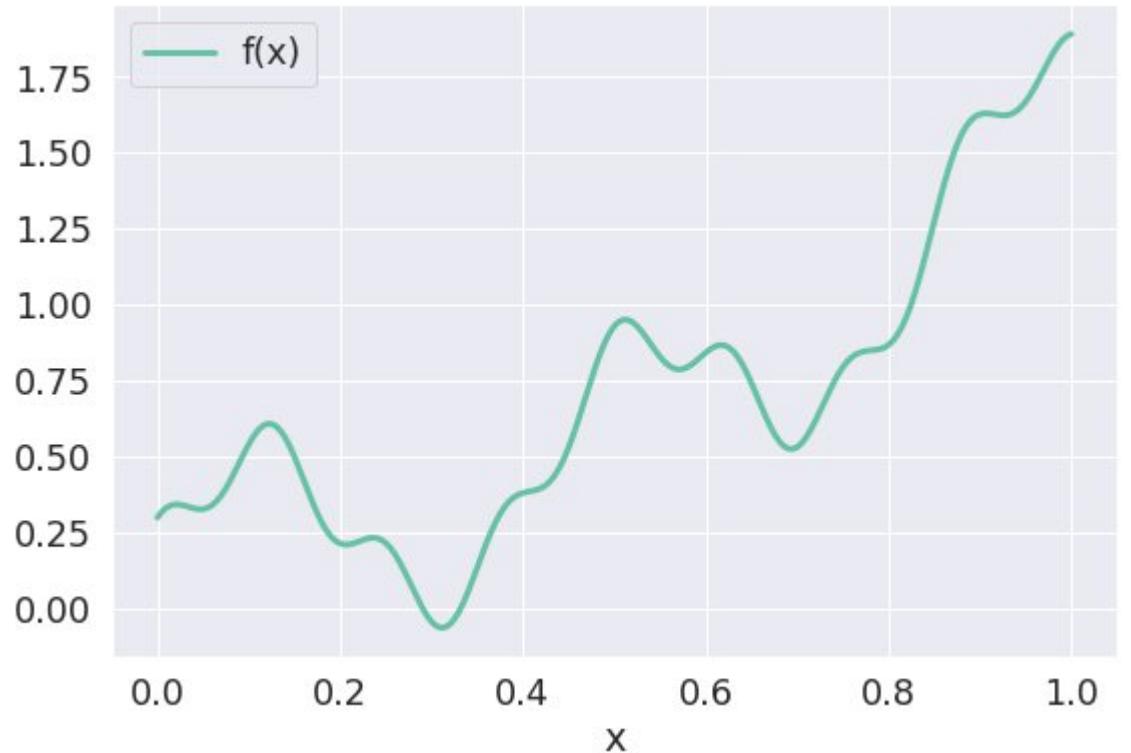


Что-то новенькое

# Пример

Возьмем  $f(x) = 0.2 + 0.4 x^2 + 0.3 x \sin(15 x) + 0.1 \cos(50 x)$ .

*Как аппроксимировать её нейронной сетью?*



# Пример

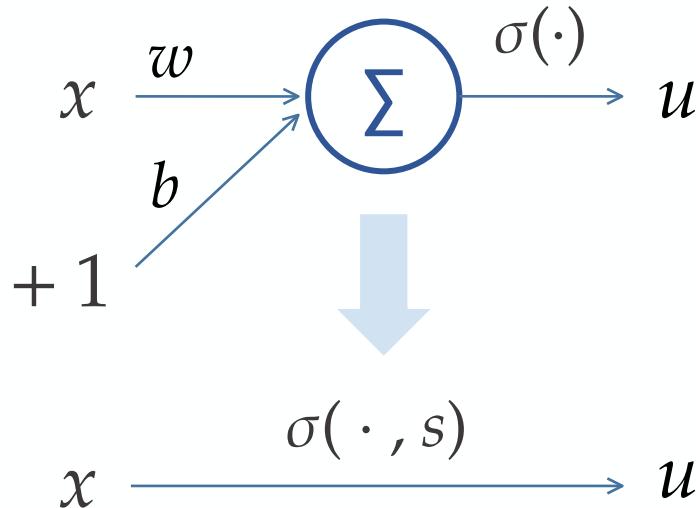
Как аппроксимировать  $f(x)$  нейронной сетью?

Рассмотрим **один нейрон**  $u = \sigma(wx + b)$  с функцией активации  $I\{z > 0\}$ .

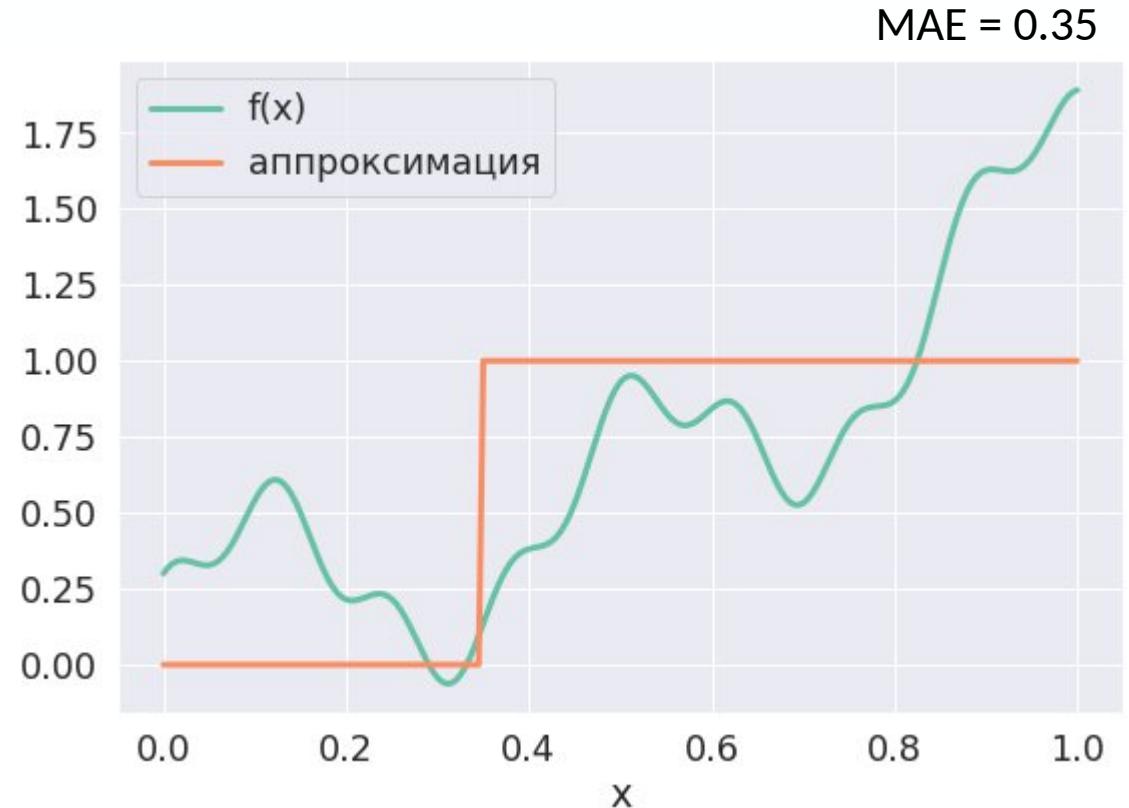
Заметим, что  $u > 0$  в точке  $x = -b/w$ . Обозначим  $s = -b/w$ .

Итоговая функция зависит только от  $s$

$\Rightarrow$  будем далее работать с  $s$ , а не  $w, b$ .



Положим  $s = 0.35$ .

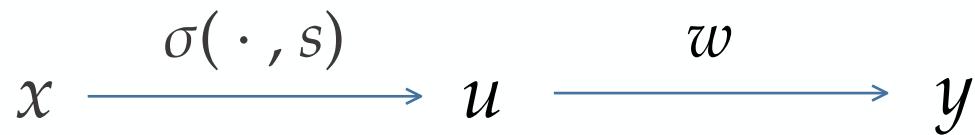


# Пример

Как аппроксимировать  $f(x)$  нейронной сетью?

**Добавим еще один слой.** Теперь у нас двухслойная нейронная сеть.

На втором слое один нейрон  $y = wi$ .



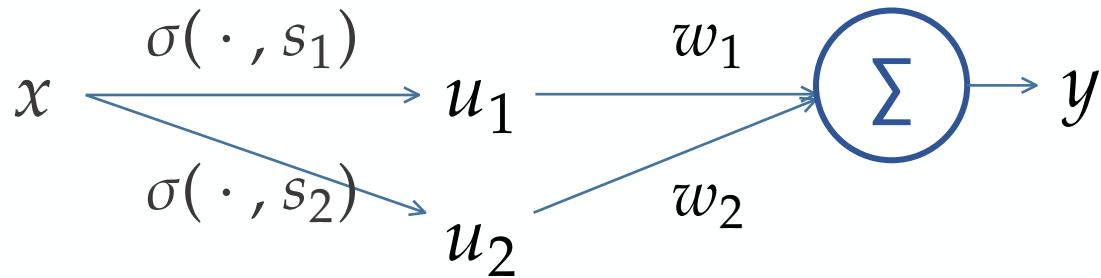
Положим  $w = 0.8$ .



# Пример

Как аппроксимировать  $f(x)$  нейронной сетью?

Добавим нейрон на первом слое.



Положим  $s_2 = 0.7$ ,  $w_2 = -w_1 = -0.8$ .

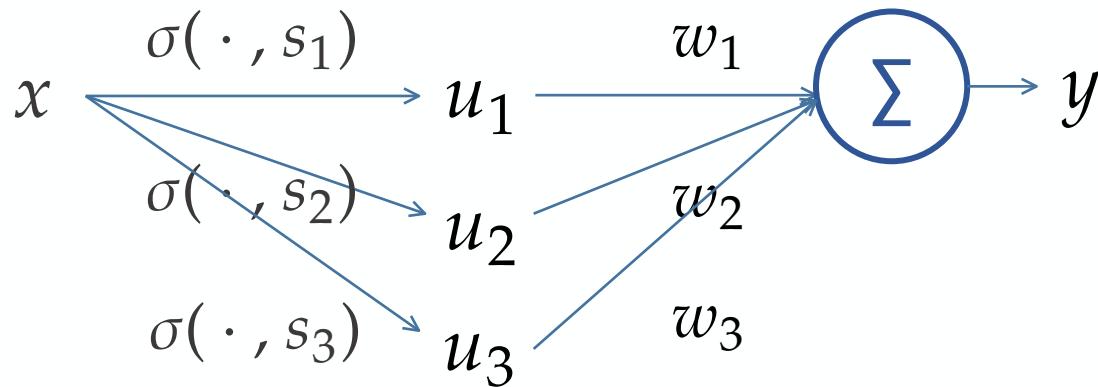
Получилась **ступенька**,  
которая приближает часть функции.



# Пример

Как аппроксимировать  $f(x)$  нейронной сетью?

Добавим еще нейрон на первом слое.



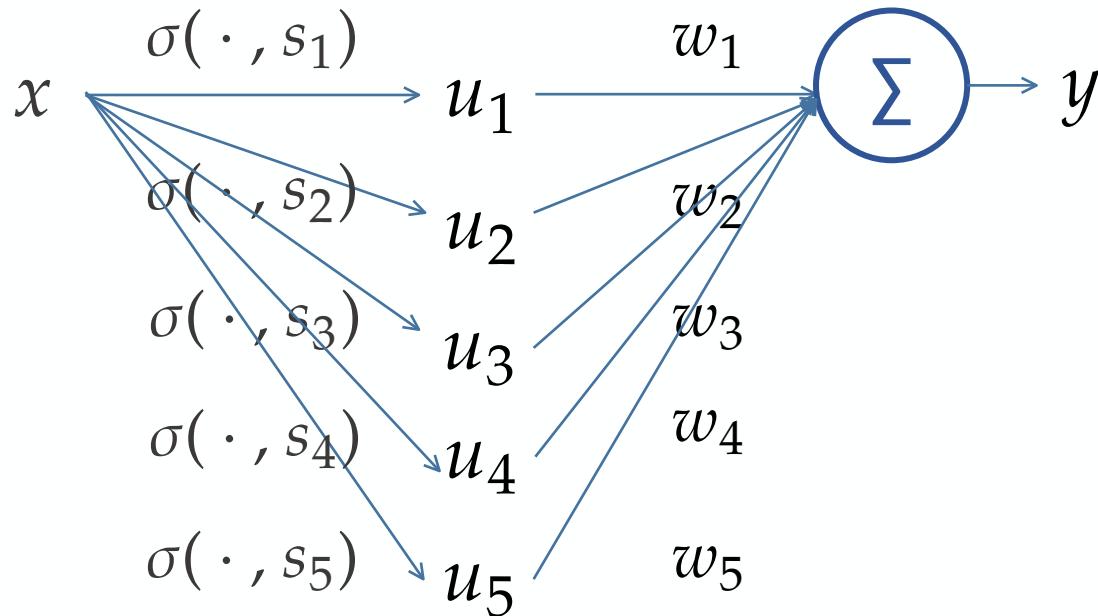
Положим  $s_3 = s_2 = 0.7$ ,  $w_3 = 1.5$ .



# Пример

Как аппроксимировать  $f(x)$  нейронной сетью?

Добавим еще два нейрона на первом слое.



Положим  $s_4 = 0, s_5 = s_1 = 0.35,$   
 $w_4 = -w_5 = 0.4.$



# Пример

## Вывод

- С помощью двухслойной нейронной сети получилось аппроксимировать сложную функцию.
- Мы не прибегали к генерации сложных признаков, которые понадобились бы, например, для линейной регрессии.
- Если увеличить число нейронов, то приближение получается более точным.

Но! Сейчас мы подбирали параметры самостоятельно.





# Нейронная сеть

Мы параметризовали модель нейронной сети.

Из теоремы Цыбенко вытекает, что существуют параметры, при которых мы сможем аппроксимировать любую непр. функцию на заданном интервале.

Хотим автоматически находить параметры сети!

А как?

Вспомним линейную регрессию...



# Решение задачи регрессии

Линейная регрессия:  $\hat{y} = Xw + b$ , т.е.  $\hat{y}_i = \sum_{j=1}^d x_{ij}w_j + b$

где  $X = (x_{ij})_{ij}$  — матрица входных данных,  $i \in \{1, \dots, n\}, j \in \{1, \dots, d\}$

$\hat{y} = (\hat{y}_1, \dots, \hat{y}_d)^T$  — вектор предсказания,

$w = (w_1, \dots, w_d)^T$  — вектор весов,  $b$  — сдвиг.

Задачу можно решить аналитически. А можно с помощью **градиентного спуска**.

**Зададим функцию**, которую мы хотим минимизировать

$$L = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \text{ — MSE (Mean Squared Error)}$$

Пусть  $\theta = (w, b)$ , тогда оптимизация будет следующей:

$$\theta_t = \theta_{t-1} - \eta \nabla L(\theta_t), \quad \text{где } \eta \text{ — скорость обучения}$$



# Обучение нейронной сети

Обозначим все параметры сети как  $\theta$ .

Пусть  $\mathcal{L}(\hat{y}_\theta, y)$  — **функция потерь** на объекте  $x$ .

Она сравнивает предсказания сети  $\hat{y}_\theta$  с откликом  $y$  на объекте  $x$ .

Минимизируем **эмпирический риск** по обучающей выборке  $x_1, \dots, x_n$ :

$$Q(w) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}_{\theta,i}, y_i) \rightarrow \min_{\theta}$$

Решаем задачу минимизации с помощью **градиентного спуска**

$$\theta_t = \theta_{t-1} - \eta \nabla Q(\theta_t), \quad \text{где } \eta \text{ — скорость обучения}$$



# Обучение нейронной сети

## Задача

Посчитать градиенты функции потерь по всем параметрам.

## Наивный подход

- Посчитать для каждого параметра градиент отдельно.
- Подсчет одного градиента линеен по количеству параметров.  
⇒ сложность такой процедуры **квадратична** по количеству параметров.

**Можно ускорить!**



# Метод обратного распространения ошибки

Метод вытекает из **формулы производной сложной функции**.

Если  $f(x) = g_m(g_{m-1}(\dots(g_1(x))\dots))$

то  $\frac{\partial f}{\partial x} = \frac{\partial g_m}{\partial g_{m-1}} \frac{\partial g_{m-1}}{\partial g_{m-2}} \dots \frac{\partial g_2}{\partial g_1} \frac{\partial g_1}{\partial x}$

Градиенты будем вычислять последовательно  
от выхода нейронной сети к входу, начиная с  $\frac{\partial g_m}{\partial g_{m-1}}$   
и умножая каждый раз на частные производные предыдущего слоя.

Тогда сложность будет **линейной**.



# Метод обратного распространения ошибки. Пример

Рассмотрим частный случай применения метода обратного распространения ошибки на двухслойной полносвязной нейронной сети.

$$x_{ij}, j = 1, \dots, D$$

—  $j$  признак объекта  $x_i$

$$u_h(x_i), h = 1 \dots H$$

— выход I слоя на объекте  $x_i$

$$\hat{y}_m(x_i), m = 1, \dots, M$$

— выход сети (выход II слоя) на объекте  $x_i$

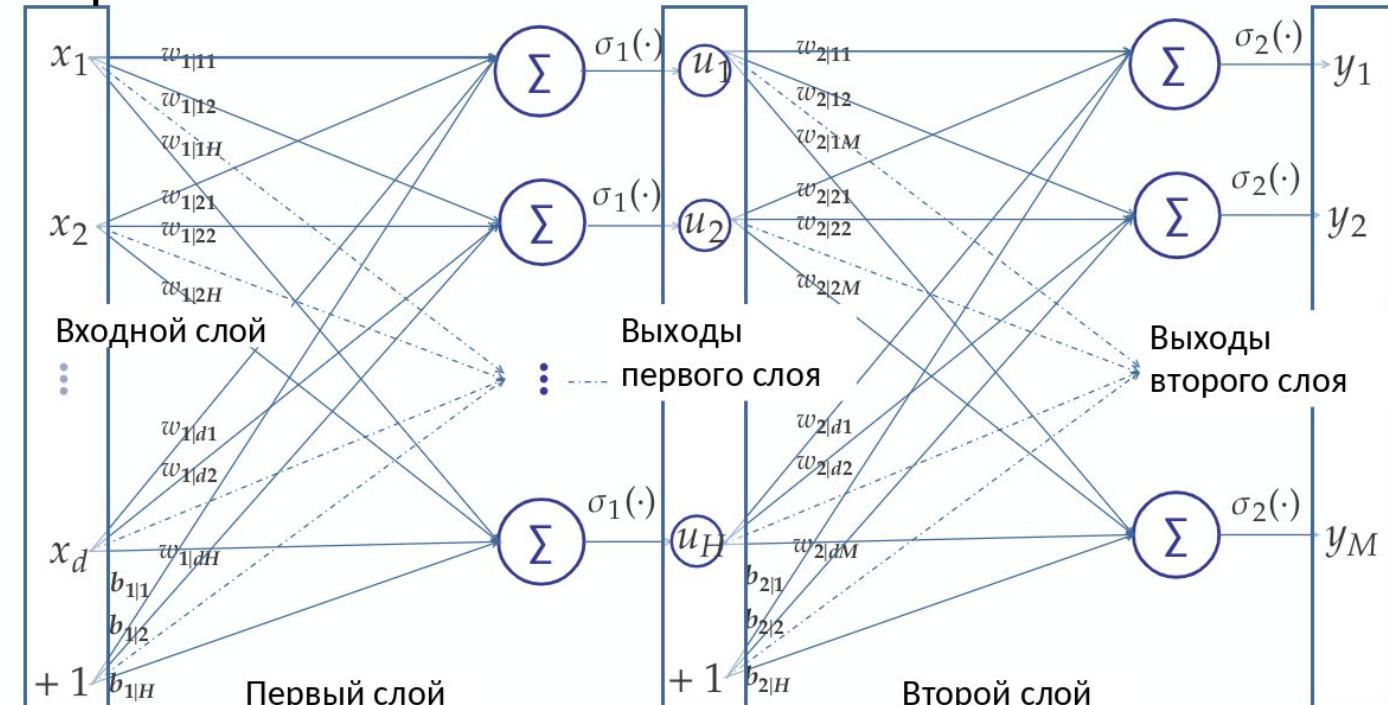
$$y_m(x_i), m = 1, \dots, M$$

— отклик на объекте  $x_i$

Для примера рассмотрим

квадратичную функцию потерь:  $\mathcal{L}(\hat{y}_i, y_i) = \frac{1}{2} \sum_{m=1}^M (\hat{y}_m(x_i) - y_{im})^2$  на объекте  $x_i$

Будем искать производные функции потерь по весам  $\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial w_{1|jh}}$  и  $\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial w_{2|hm}}$ .



Подсчет производных по сдвигам оставим в качестве упражнения.



# Метод обратного распространения ошибки. Пример

Решим промежуточную задачу — найдем

$$\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial \hat{y}_m} \quad \text{и} \quad \frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial u_h}$$

Выражения для  $\hat{y}_m(x_i)$  и  $u_h(x_i)$

$$\hat{y}_m(x_i) = \sigma_2 \left( \sum_{h=0}^H w_{2|h m} u_h(x_i) + b_{2|m} \right), \quad u_h(x_i) = \sigma_1 \left( \sum_{j=0}^d w_{1|j h} x_{ij} + b_{1|h} \right)$$

Напоминание:  $\mathcal{L}(\hat{y}_i, y_i) = \frac{1}{2} \sum_{m=1}^M (\hat{y}_m(x_i) - y_{im})^2$

$$\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial \hat{y}_m} = \hat{y}_m(x_i) - y_{im} = \varepsilon_{im}$$

$$\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial u_h} = \sum_{m=1}^M \frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial \hat{y}_m} \cdot \frac{\partial \hat{y}_m(x_i)}{\partial u_h} = \sum_{m=1}^M \varepsilon_{im} \cdot \sigma'_2 \left( \sum_{h=0}^H w_{2|h m} u_h(x_i) + b_{2|m} \right) \cdot w_{2|h m}$$



# Метод обратного распространения ошибки. Пример

Теперь, имея частные производные  $\mathcal{L}(\hat{y}_i, y_i)$  по  $\hat{y}_m$  и  $u_h$ ,  
легко выписать градиент  $\mathcal{L}(\hat{y}_i, y_i)$  по весам

Выражения для  $\hat{y}_m(x_i)$  и  $u_h(x_i)$

$$\hat{y}_m(x_i) = \sigma_2\left(\sum_{h=0}^H w_{2|hm} u_h(x_i) + b_{2|m}\right), \quad u_h(x_i) = \sigma_1\left(\sum_{j=0}^d w_{1|jh} x_{ij} + b_{1|h}\right)$$

$$\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial w_{2|hm}} = \frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial \hat{y}_m} \cdot \frac{\partial \hat{y}_m}{\partial w_{2|hm}} = \frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial \hat{y}_m} \cdot \sigma'_2\left(\sum_{h=0}^H w_{2|hm} u_h(x_i) + b_{2|m}\right) \cdot u_h(x_i)$$

$$\frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial w_{1|jh}} = \frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial u_h} \cdot \frac{\partial u_h}{\partial w_{1|jh}} = \frac{\partial \mathcal{L}(\hat{y}_i, y_i)}{\partial u_h} \cdot \sigma'_1\left(\sum_{j=0}^d w_{1|jh} x_{ij} + b_{1|h}\right) \cdot x_{ij}$$



# Обучение нейронной сети

## Итоговый алгоритм

1. Инициализировать все веса.
2. Повторить NUMBER\_OF\_STEPS раз:
  - 2.1 Прямое распространение (Forward pass) — вычисляем выходы всех нейронов
  - 2.2 Вычисляем ошибку предсказания (Loss)
  - 2.3 Обратное распространение (Backward pass) — считаем последовательно производные с выхода сети до входа.
  - 2.4 Обновление весов (Update / Optimizer step) — обновляем веса, производя шаг оптимизации



Везде свои тонкости



# Обучение нейронной сети

Подставим эмпирический риск в формулу градиентного спуска

$$\theta_t = \theta_{t-1} - \eta \nabla \left( \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}_{\theta_{t-1}, i}, y_i) \right)$$

Нейронные сети способны обучаться на огромном количестве данных.  
Если мы будем считать эмпирический риск **по всем данным сразу**,  
то у нас все **может не поместиться в память**.

Разные виды градиентного спуска решают эту проблему по-своему.



# Обучение нейронной сети

## Виды градиентного спуска

- **Batch Gradient Descent**

Разбиваем данные на блоки (батчи).

$$\theta_t = \theta_{t-1} - \eta \nabla \left( \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}_{\theta_{t-1}, i}, y_i) \right)$$

Для каждого блока считаем градиенты и накапливаем их.

Производим обновление параметров после подсчета градиента по всем данным.

- **Mini-batch Gradient Descent**

Разбиваем данные на блоки (батчи).

$$\theta_t = \theta_{t-1} - \eta \nabla \left( \frac{1}{b} \sum_{k=1}^b \mathcal{L}(\hat{y}_{\theta_{t-1}, i_k}, y_{i_k}) \right)$$

Для каждого блока считаем градиент и обновляем параметры.

$(x_{i_1}, \dots, x_{i_b})$  — текущий батч. **Самый популярный метод.**

- **Стохастический градиентный спуск**

Выбираем один элемент  $x_i$  случайно.

Считаем градиент и производим обновление.

Эквивалентно Mini-Batch GD для размера батча, равного 1.

$$\theta_t = \theta_{t-1} - \eta \nabla \left( \mathcal{L}(\hat{y}_{\theta_{t-1}, i}, y_i) \right)$$



# Функции активации

## Зачем нужны функции активации?

- Для того, чтобы делать нелинейные преобразования.
- Кроме того, по теореме Цыбенко, используя функции активации типа сигмоиды мы можем аппроксимировать любую функцию используя двухслойную нейронную сеть.

# Функции активации

## Сигмоида

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

«+»:

- гладкий градиент, уменьшает скачки в значениях
- ограниченный диапазон значений — [0, 1]
- позволяет получать ~ вероятности

«-»:

- затухающий градиент!
- вычислительно сложно



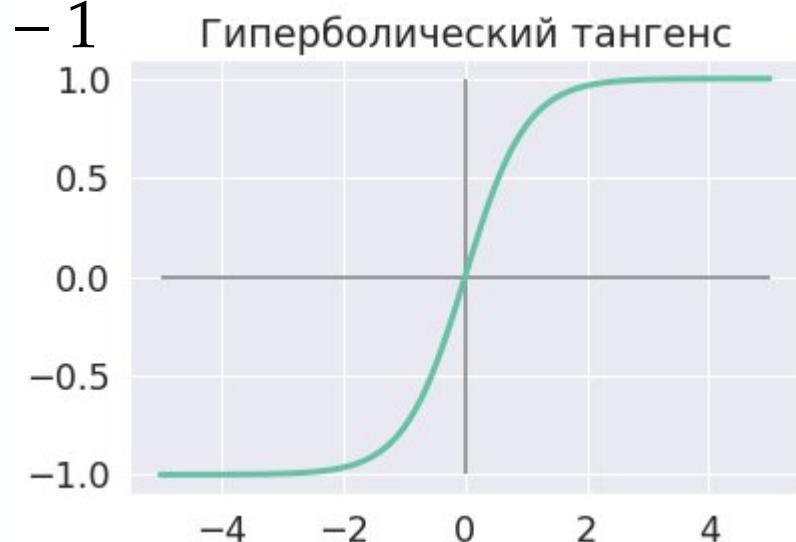
## Гиперболический тангенс

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = 2\sigma(2z) - 1$$

Те же плюсы и недостатки, но еще

«+»:

- ограниченный диапазон значений — [-1, 1]
- большая амплитуда градиента вблизи нуля
- лучше сигмоиды, когда не нужна нормализация [0, 1]





# Функции активации

## ReLU

$$relu(z) = \max(0, z)$$

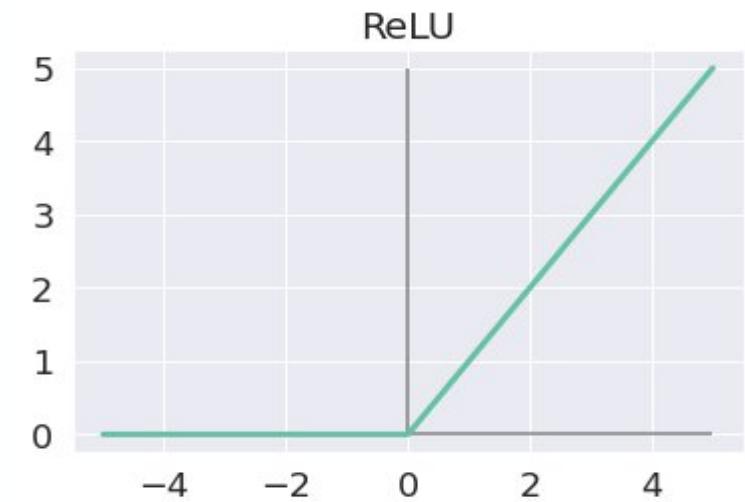
«+»:

- вычислительно просто
- ускоряет сходимость за счет выключения нейронов

«-»:

- The Dying ReLU problem

нулевые градиенты ведут к тому, что часть сети не обучается, может перестать обучаться вся сеть



## Leaky ReLU

$$lrelu(z) = z \cdot I\{z > 0\} + \alpha z \cdot I\{z \leq 0\}$$

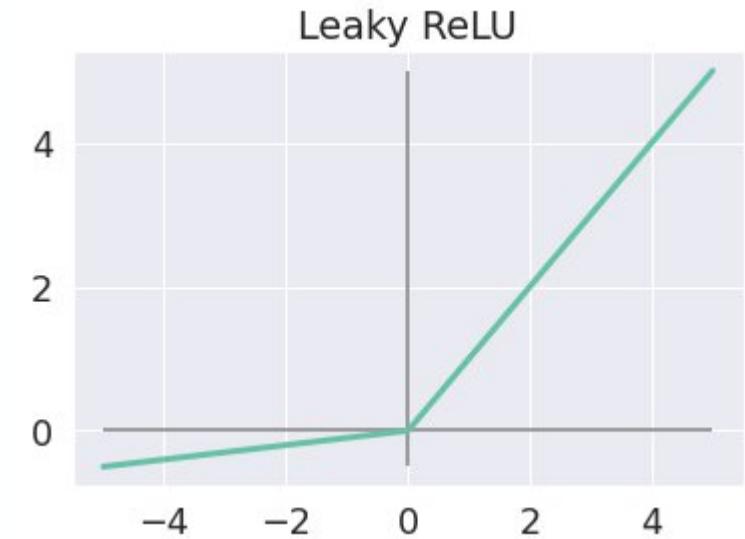
«+»:

- Устраняет The Dying ReLU problem.
- Сходимость все равно быстрая.

«-»:

- Иногда хуже ReLU

Обычно  $\alpha = 0.01$



# Функции активации

## PReLU

$$prelu(z) = z \cdot I\{z > 0\} + \alpha z \cdot I\{z \leq 0\}$$

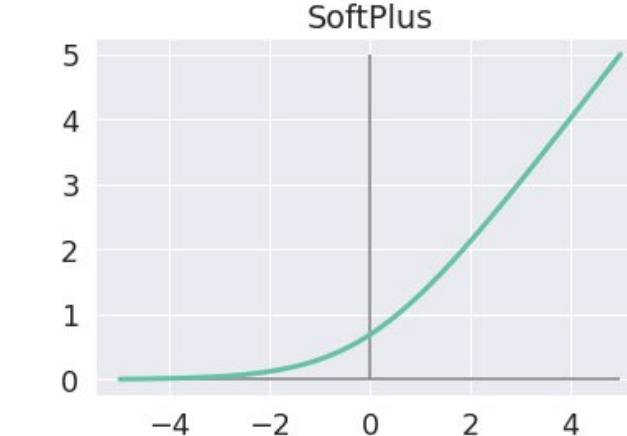
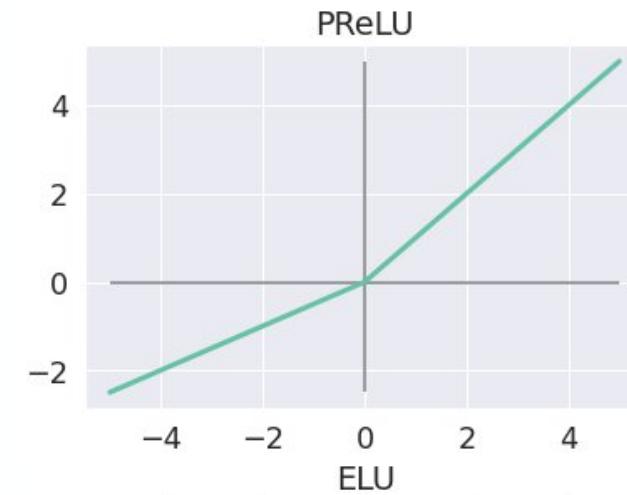
Здесь  $\alpha$  - обучаемый параметр сети.

## ELU

$$elu(z) = z \cdot I\{z > 0\} + \alpha(e^z - 1) \cdot I\{z \leq 0\}$$

## SoftPlus

$$softplus(z) = \log_e(1 + e^z)$$





# Предсказание вероятности

Для задачи классификации часто требуется предсказать распределение вероятностей по классам.

⇒ Нужен многомерный выход.

На выходе последнего слоя без функции активации получим значения, не суммирующиеся в 1.

Чтобы отнормировать значения применяется функция активации softmax.

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{i=1}^K e^{z_i}}, k \in [1, K], K - \text{кол-во классов.}$$

Отнормировали  $K$  значений на отрезок  $[0,1]$  так, что они суммируются в 1.



# Предсказание вероятности

Для задачи классификации часто требуется предсказать распределение вероятностей по классам.

⇒ Нужен многомерный выход.

На выходе последнего слоя без функции активации получим значения, не суммирующиеся в 1.

Чтобы отнормировать значения применяется функция активации softmax.

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{i=1}^K e^{z_i}}, k \in [1, K], K - \text{кол-во классов.}$$

Отнормировали  $K$  значений на отрезок  $[0,1]$  так, что они суммируются в 1.

**Но почему бы просто не взять argmax из значений как метку класса?**



# Предсказание вероятности

Для задачи классификации часто требуется предсказать распределение вероятностей по классам.

⇒ Нужен многомерный выход.

На выходе последнего слоя без функции активации получим значения, не суммирующиеся в 1.

Чтобы отнормировать значения применяется функция активации softmax.

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{i=1}^K e^{z_i}}, k \in [1, K], K - \text{кол-во классов.}$$

Отнормировали  $K$  значений на отрезок  $[0, 1]$  так, что они суммируются в 1.

**Но почему бы просто не взять argmax из значений как метку класса?**

Градиент будет нулевым практически везде, что не позволит сети нормально обучаться.



# Предсказание вероятности

Для задачи классификации часто требуется предсказать распределение вероятностей по классам.

⇒ Нужен многомерный выход.

На выходе последнего слоя без функции активации получим значения, не суммирующиеся в 1.

Чтобы отнормировать значения применяется функция активации softmax.

$$\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_{i=1}^K e^{z_i}}, k \in [1, K], K - \text{кол-во классов.}$$

Отнормировали  $K$  значений на отрезок  $[0,1]$  так, что они суммируются в 1.

**Но почему бы просто не взять argmax из значений как метку класса?**

Градиент будет нулевым практически везде, что не позволит сети нормально обучаться.

## Замечания

- Softmax на самом деле не является ф-й активации, т. к. принимает на вход вектор, а не скаляр.
- Простое деление на сумму не отмасштабирует значения на  $[0,1]$ , если какие-то из значений отриц.



BCE!