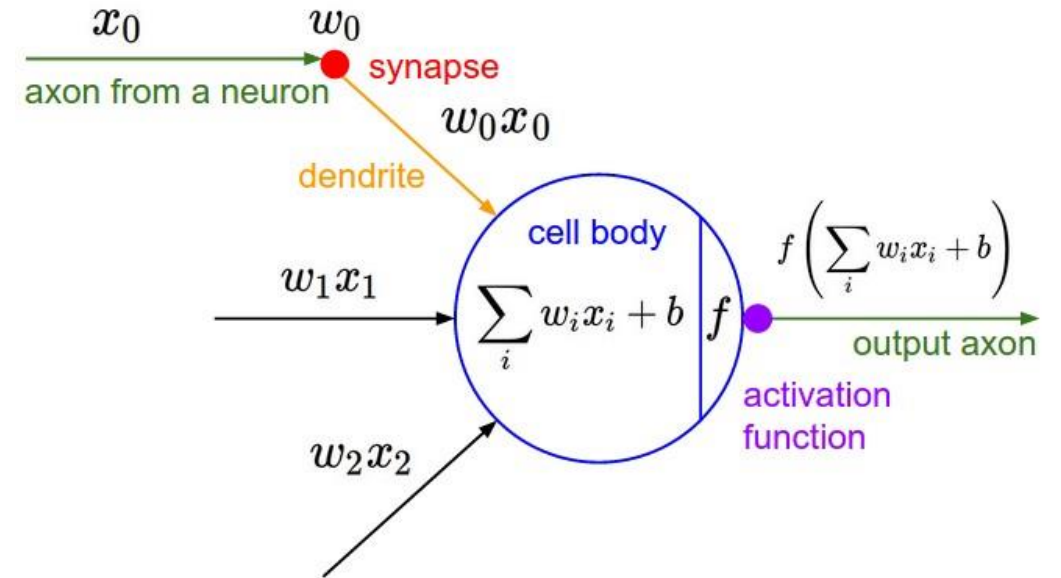
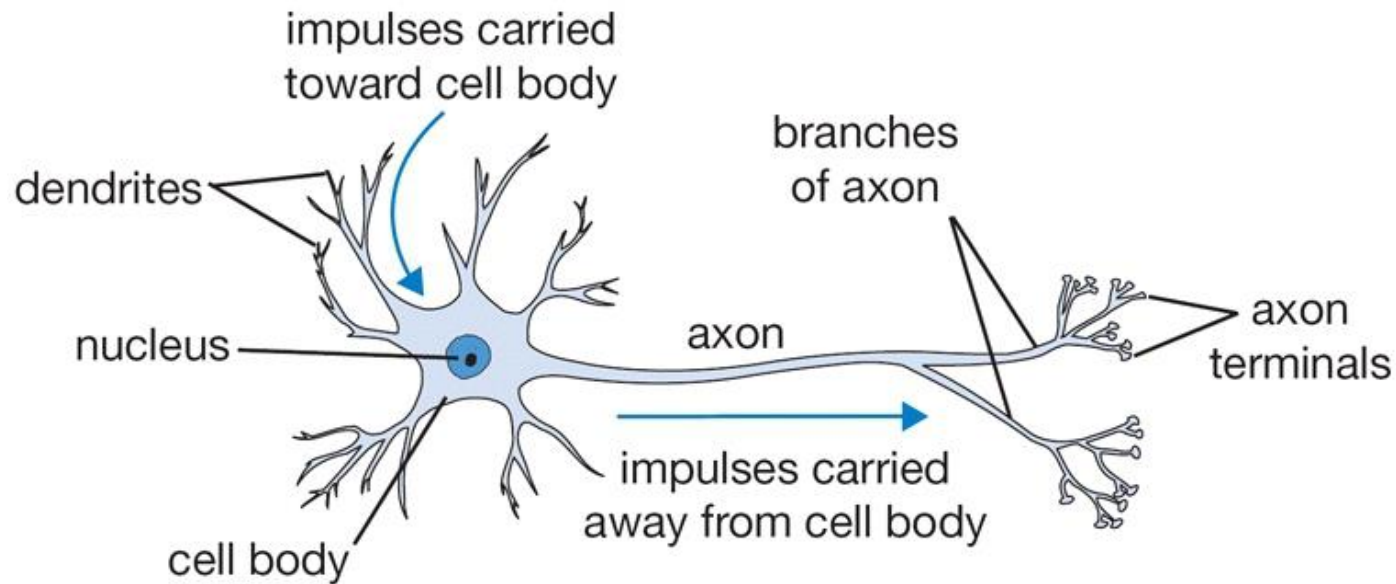


# Глубокое обучение для обработки изображений

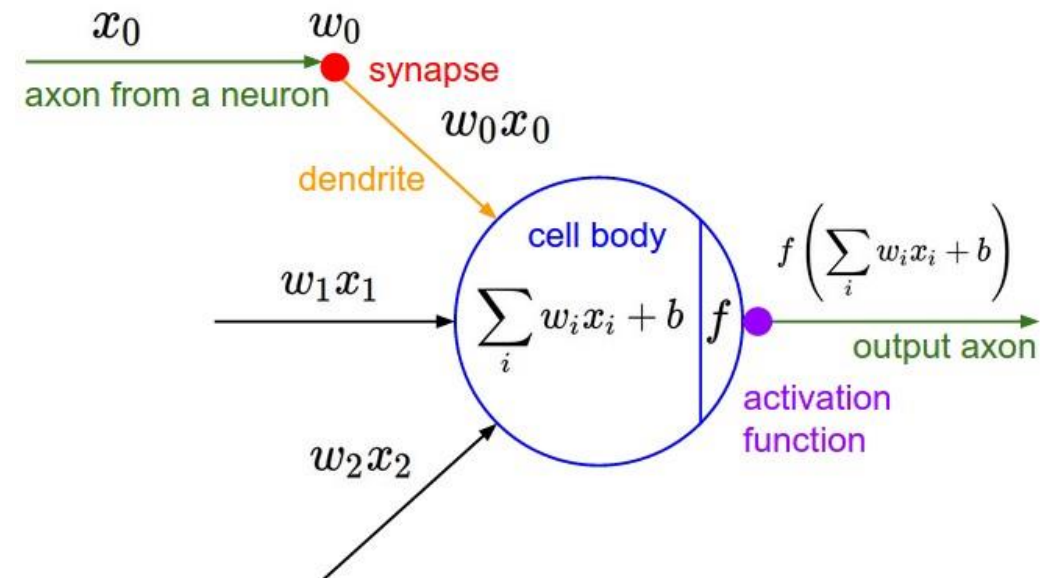
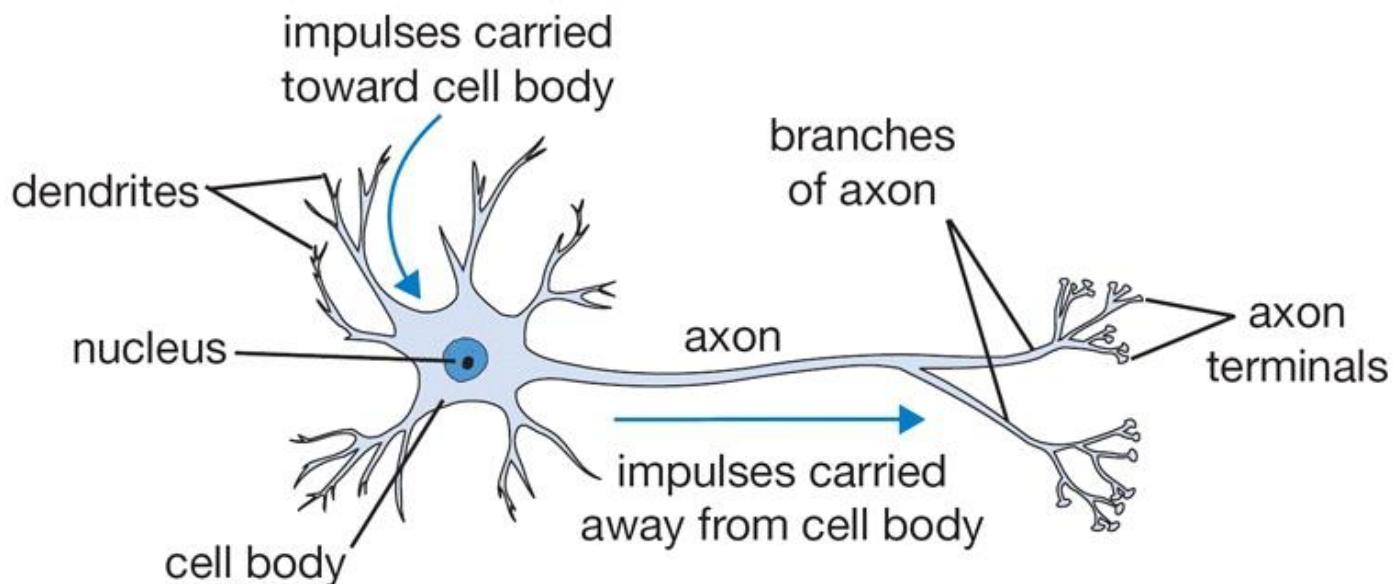
## Лекция 1

# Связь с биологией



Слева: изображений биологического нейрона. Справа: математическая модель

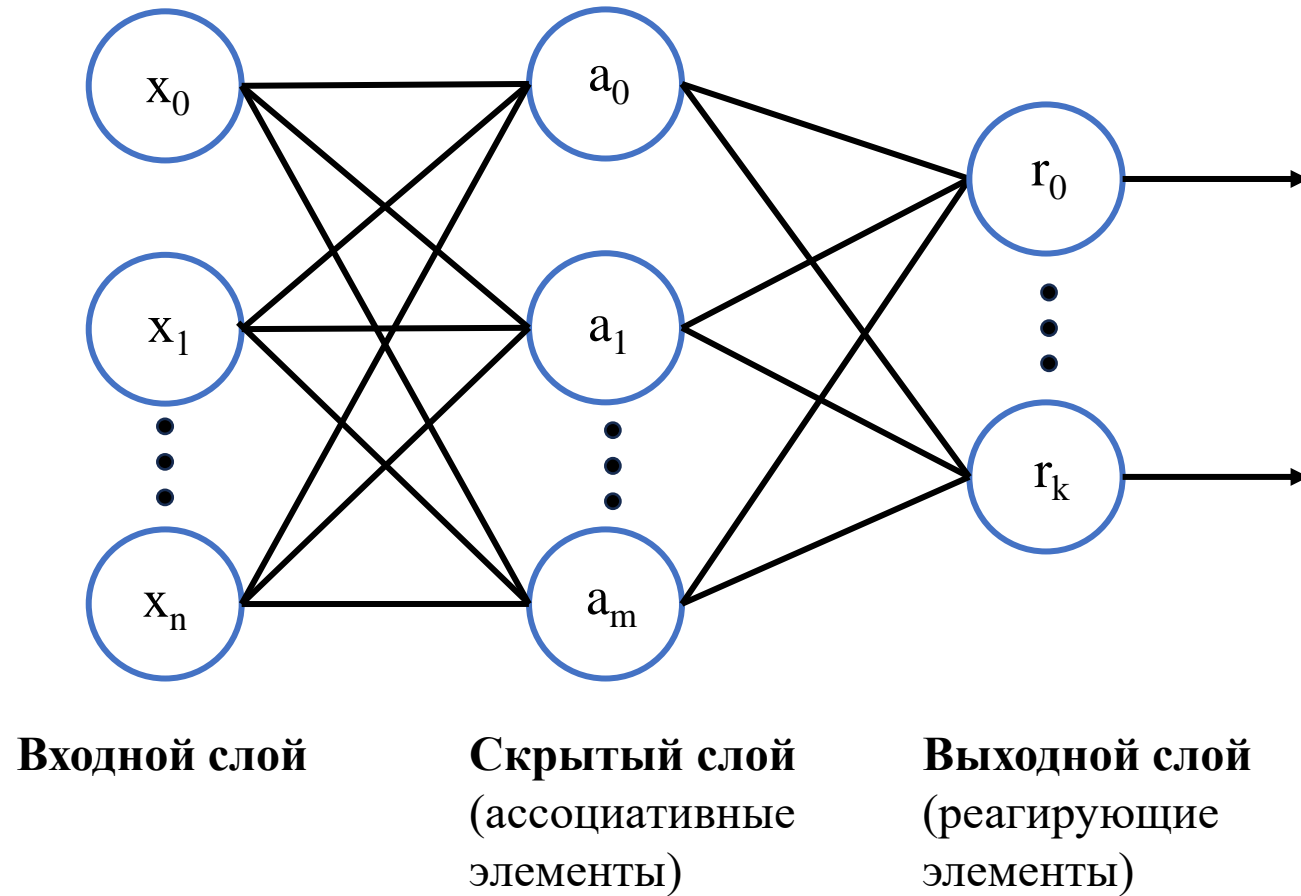
# Связь с биологией



Слева: изображений биологического нейрона. Справа: математическая модель

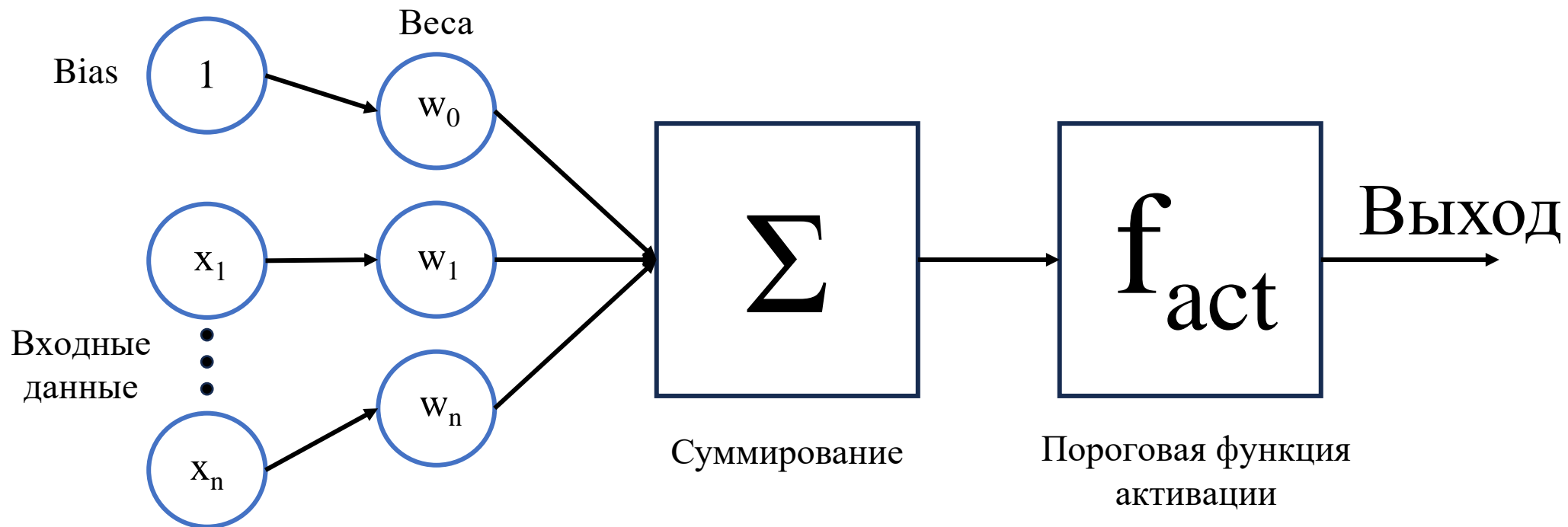
- **Математическая модель** нейрона является довольно грубым приближением **биологического нейрона**, поэтому проводить аналогии между ними нужно с большой осторожностью!

# Перцептрон Розенблатта (1958 г.)



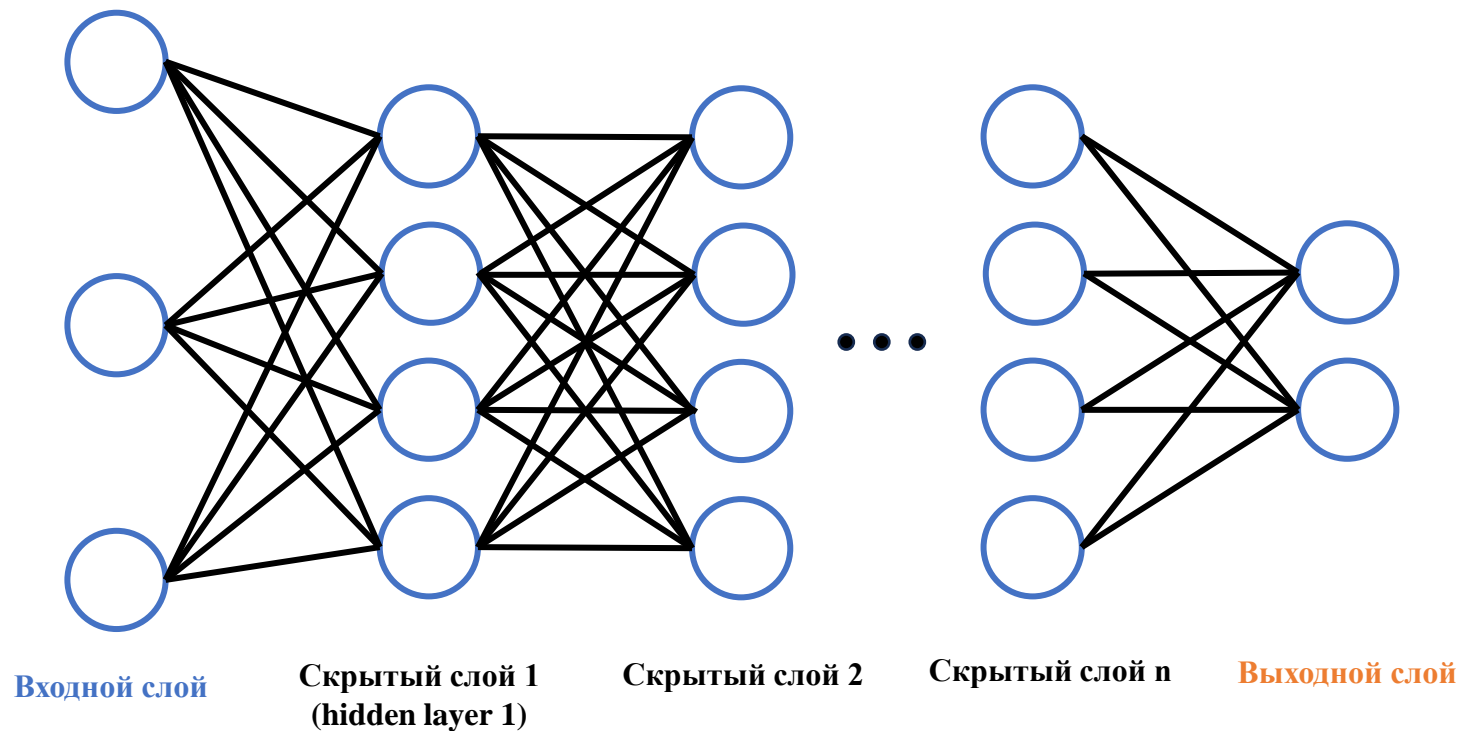
Перцептрон Розенблатта. По сути он стал первой моделью искусственной нейронной сети

# Перцептрон Розенблатта (1958 г.)



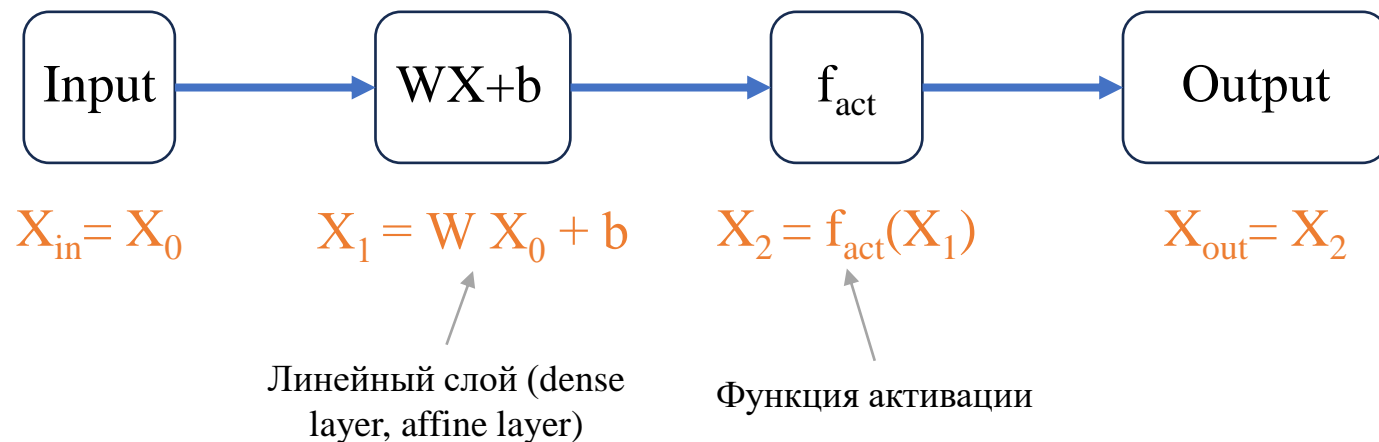
Модель нейрона. Основной “кирпичик” в построение перцептрона

# Искусственная нейронная сеть



В перцептроне Розенблатта предполагался один скрытый слой. В общем случае количество скрытых слоев может быть достаточно большим и зависит от выбранной архитектуры.

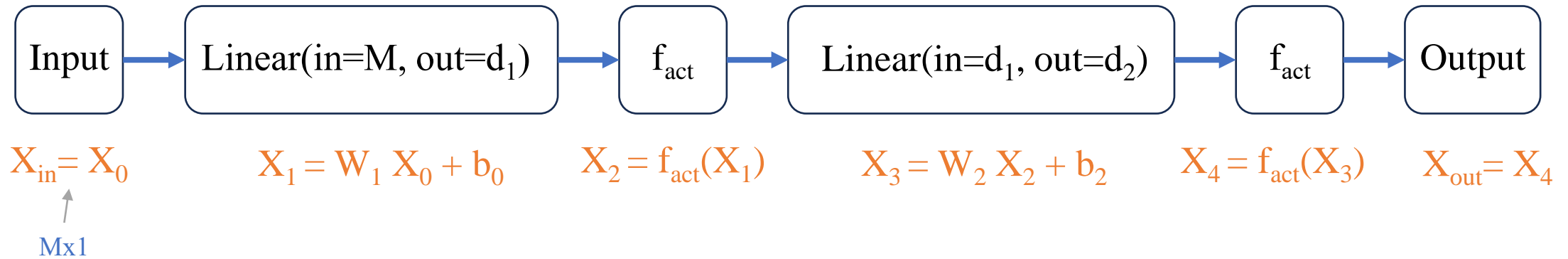
# Нейронная сеть как вычислительный граф



На этом примере можно узнать граф, соответствующий *логистической регрессии*, если функцию активации выбрать в виде *сигмоиды*

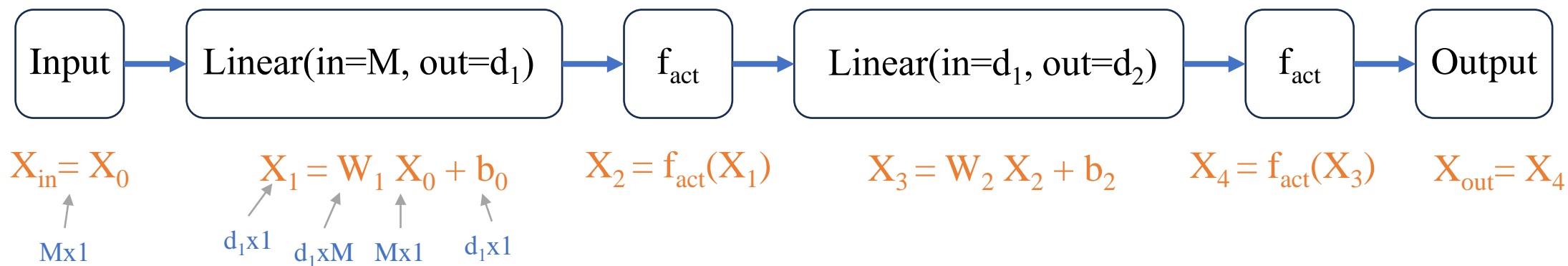
Нейросеть можно представить в виде соединённых вычислительных блоков — в виде **вычислительного графа (computational graph)**. Графы могут быть нелинейными, с обратной связью и т.д.

# Размерности матриц

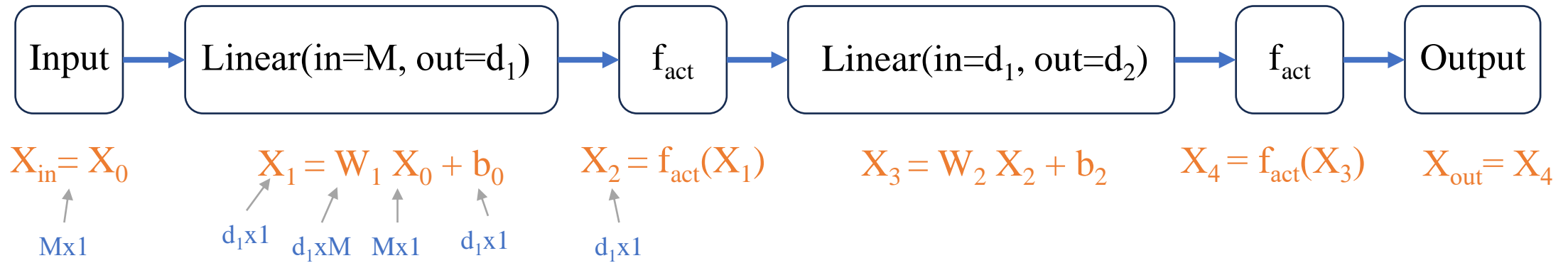




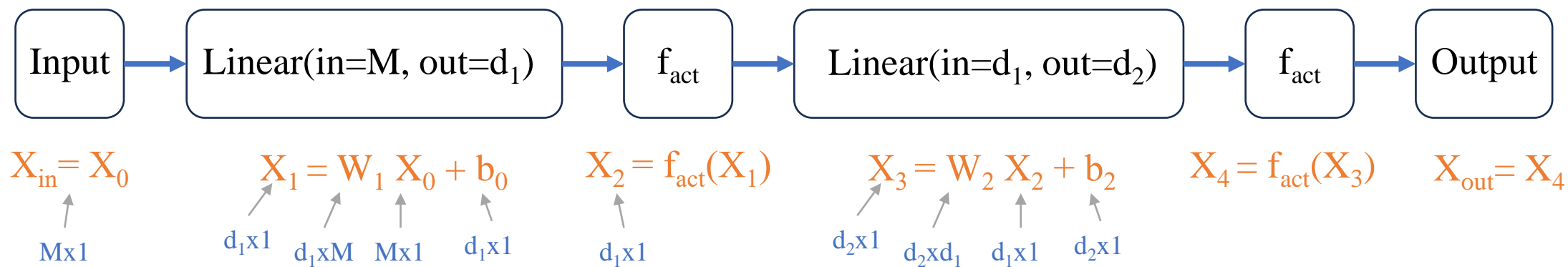
# Размерности матриц



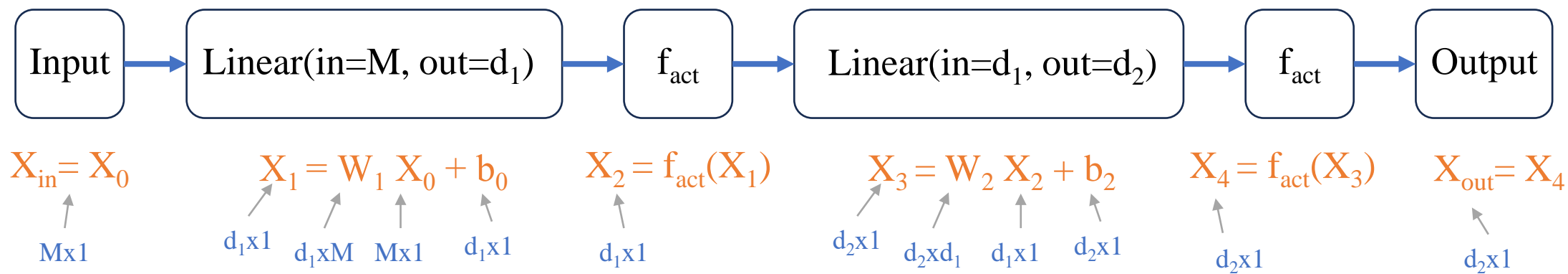
# Размерности матриц



# Размерности матриц

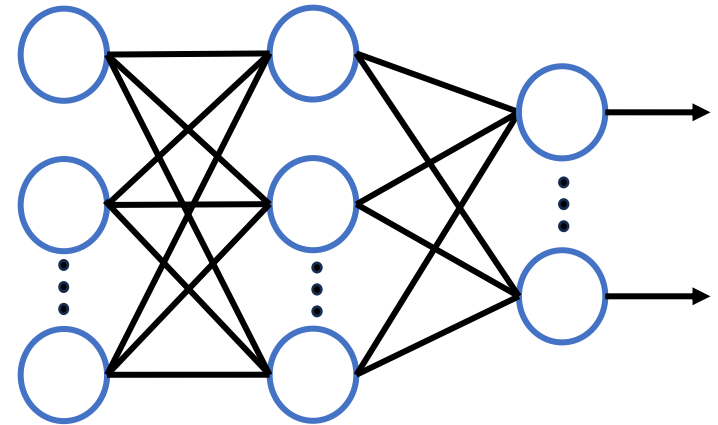


# Размерности матриц



# Универсальная теорема аппроксимации

**Теорема Цыбенко** (Hornik et al., 1989; Cybenko, 1989) утверждает, что нейронная сеть прямого распространения с одним скрытым слоем и с произвольной функцией активации может аппроксимировать любую непрерывную функцию многих переменных с любой точностью при условии правильного подбора количества нейронов скрытого слоя и весов.

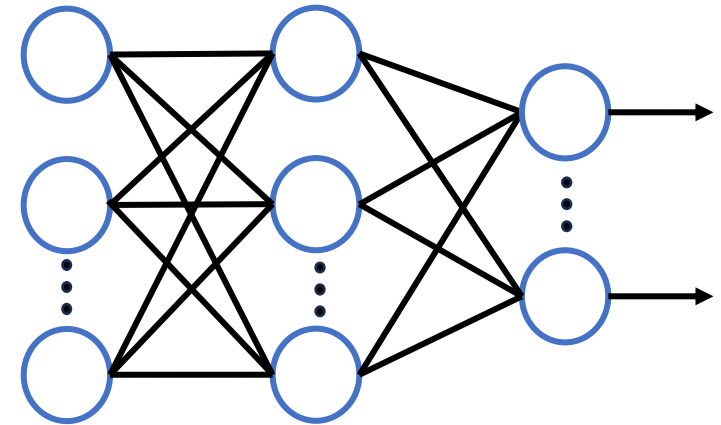


# Универсальная теорема аппроксимации

**Теорема Цыбенко** (Hornik et al., 1989; Cybenko, 1989) утверждает, что нейронная сеть прямого распространения с одним скрытым слоем и с произвольной функцией активации может аппроксимировать любую непрерывную функцию многих переменных с любой точностью при условии правильного подбора количества нейронов скрытого слоя и весов.

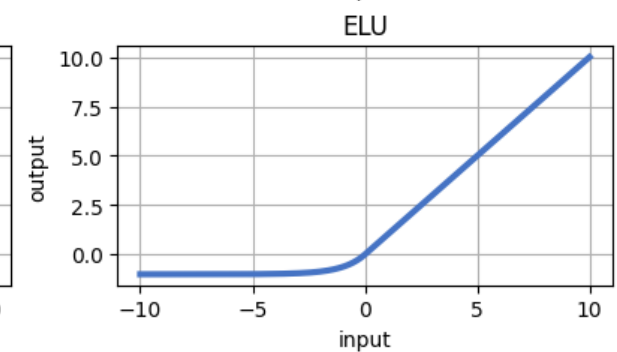
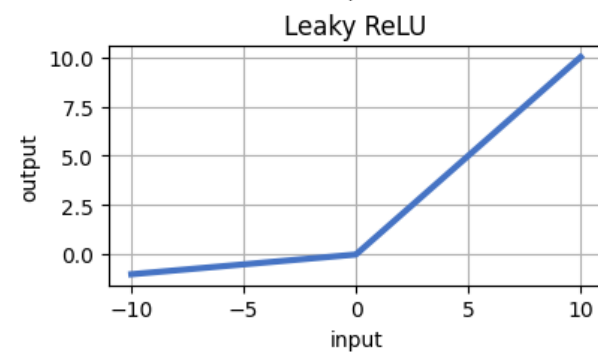
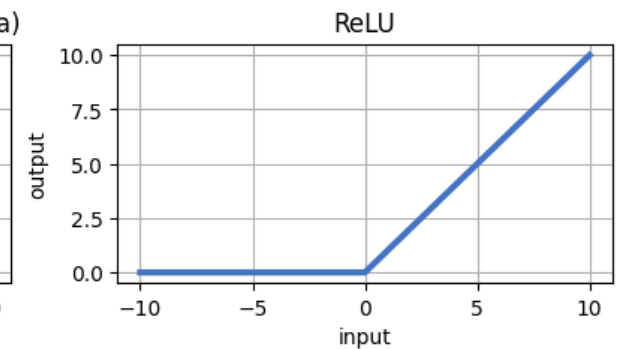
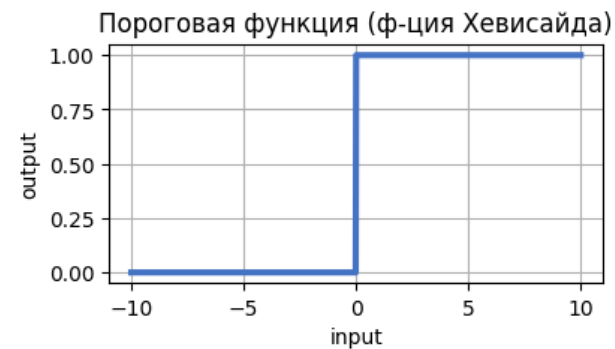
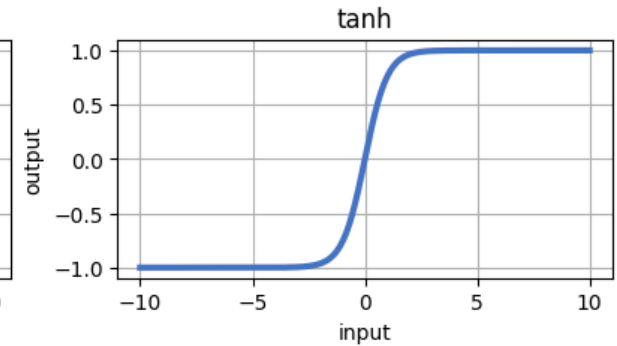
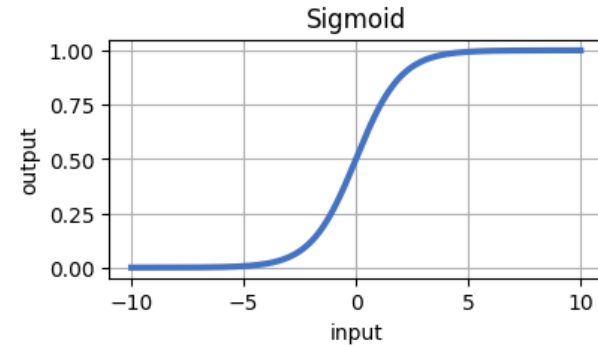
## Проблемы:

- Не гарантируется, что алгоритм обучения сможет найти оптимальные параметры
- Количество необходимых нейронов может быть очень большим
- Из-за переобучения оптимизатор может выбрать не ту функцию, что приведет к росту количества ошибок на тестовом датасете



# Функции активации

- Сигмоида
- Гиперболический тангенс
- Пороговая функция
- ReLU
- Leaky ReLU
- ELU
- И другие



# Сигмоида (sigmoid function)

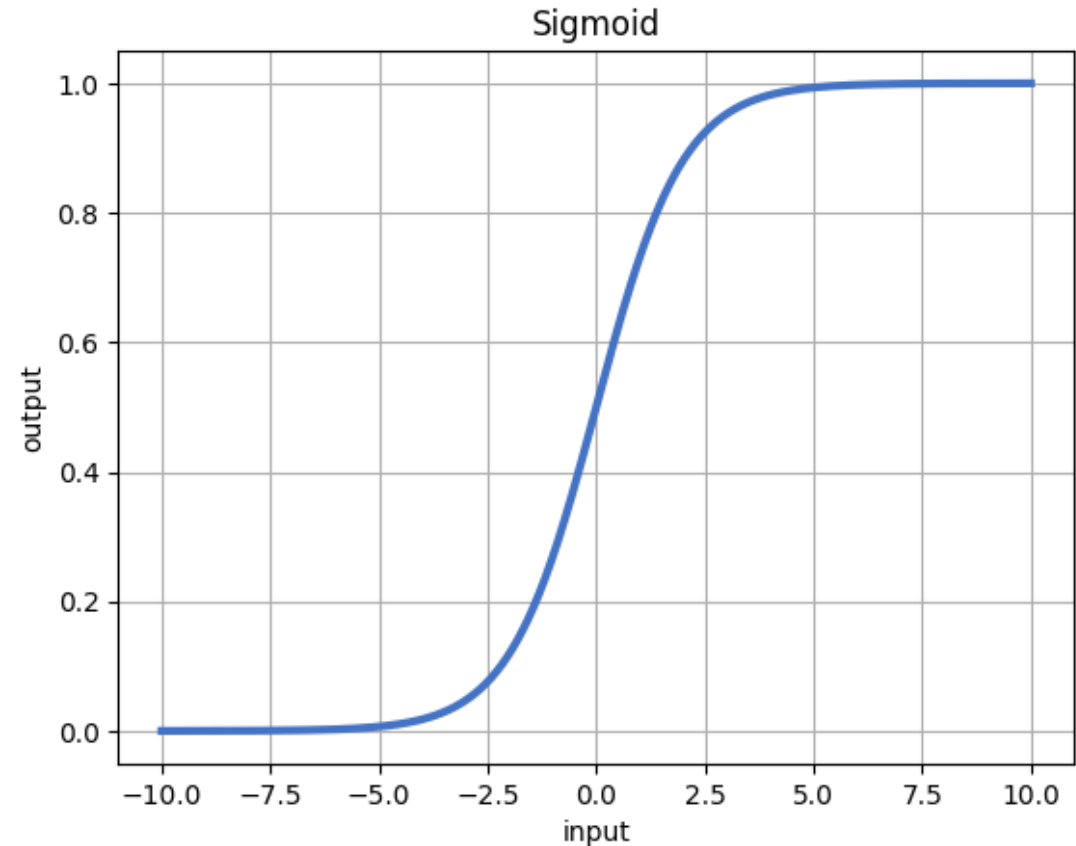
## Плюсы:

- Позволяет интерпретировать выход как вероятность
- Область значений ограничена

## Минусы:

- Смещена относительно нуля
- При больших или малых значениях входа значения производной близко к нулю, что может привести к затуханию градиента (важно для обучения)
- Требуется вычисления экспоненты

На практике внутри сети почти не используется. Чаще всего её можно встретить на последнем слое с целью получить вероятности для двух классов (для бинарной классификации)



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



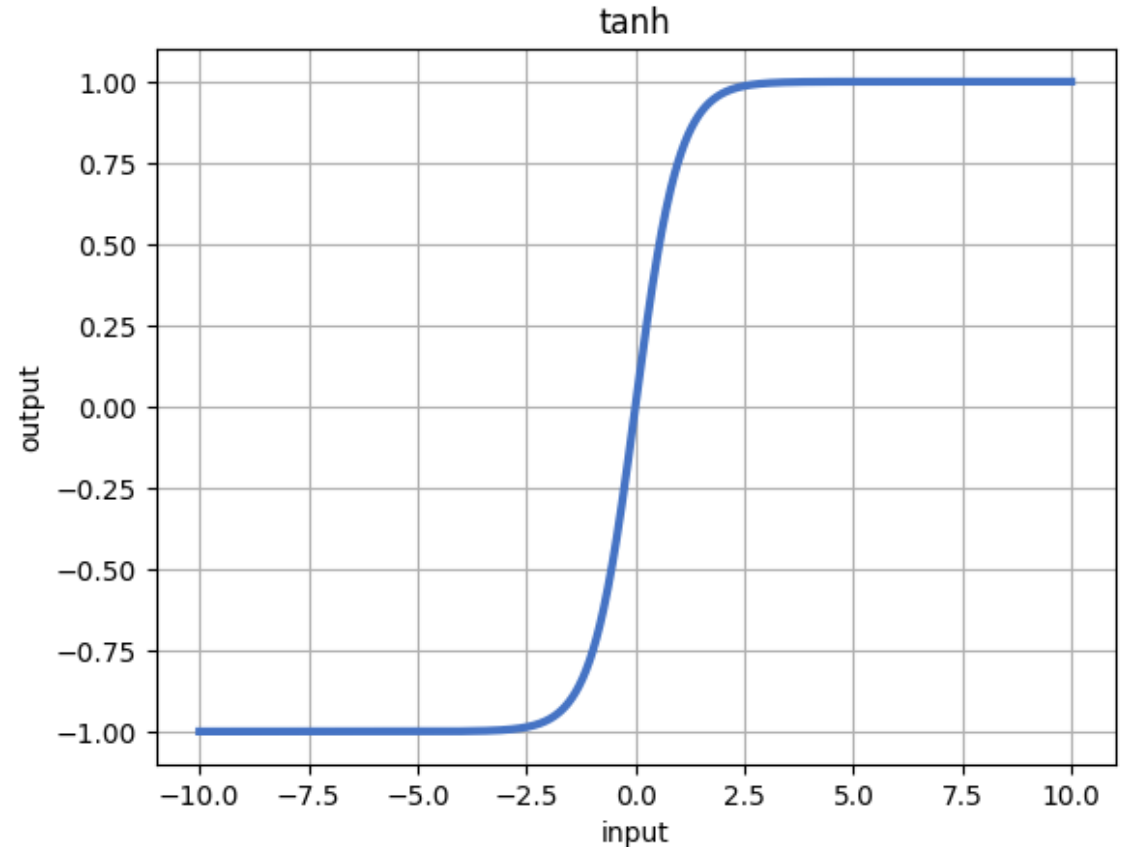
# Гиперболический тангенс (tanh)

## Плюсы:

- Симметричен относительно нуля
- Область значений ограничена

## Минусы:

- При больших или малых значениях входа значения производной близко к нулю, что может привести к затуханию градиента (важно для обучения)
- Требуется вычисления экспоненты

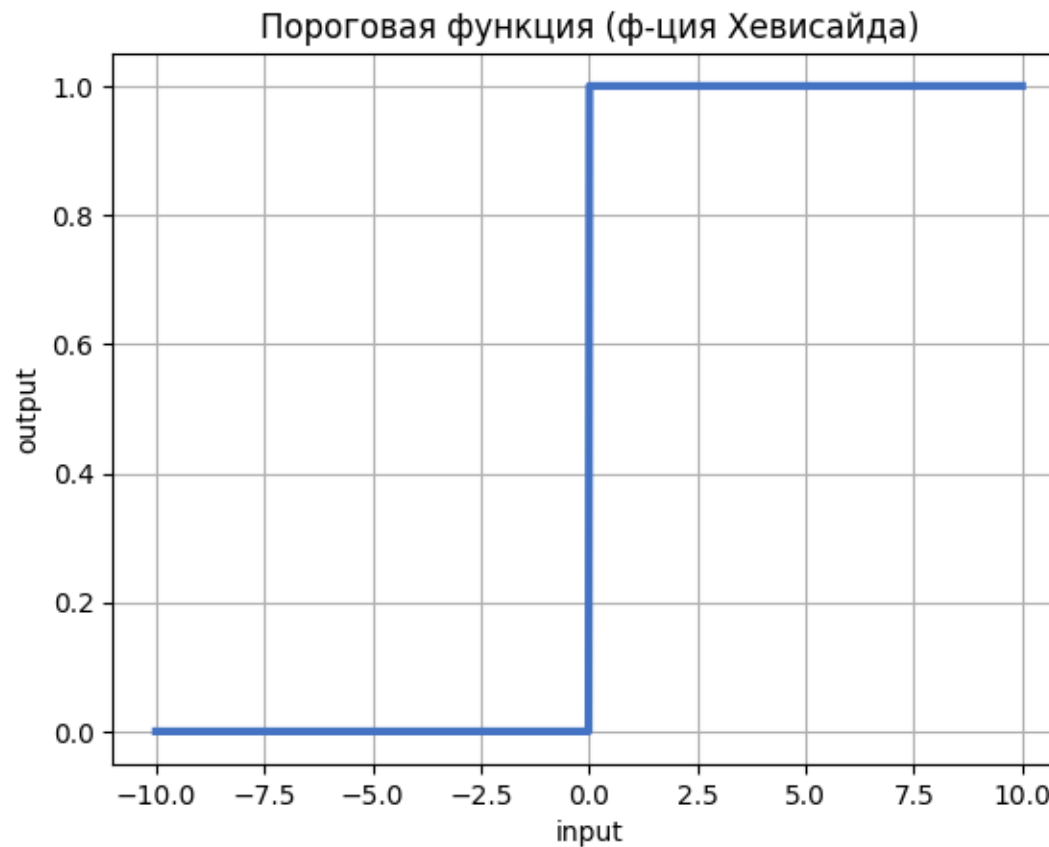


$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

# Пороговая функция (функция Хевисайда)

## Минусы:

- Может возвращать лишь два значения
- Градиент равен нулю во всех областях



$$\theta(x) = \begin{cases} 0, & x < 0; \\ 1, & x \geq 0. \end{cases}$$

# ReLU (усеченное линейное преобразование, Rectified Linear Unit)

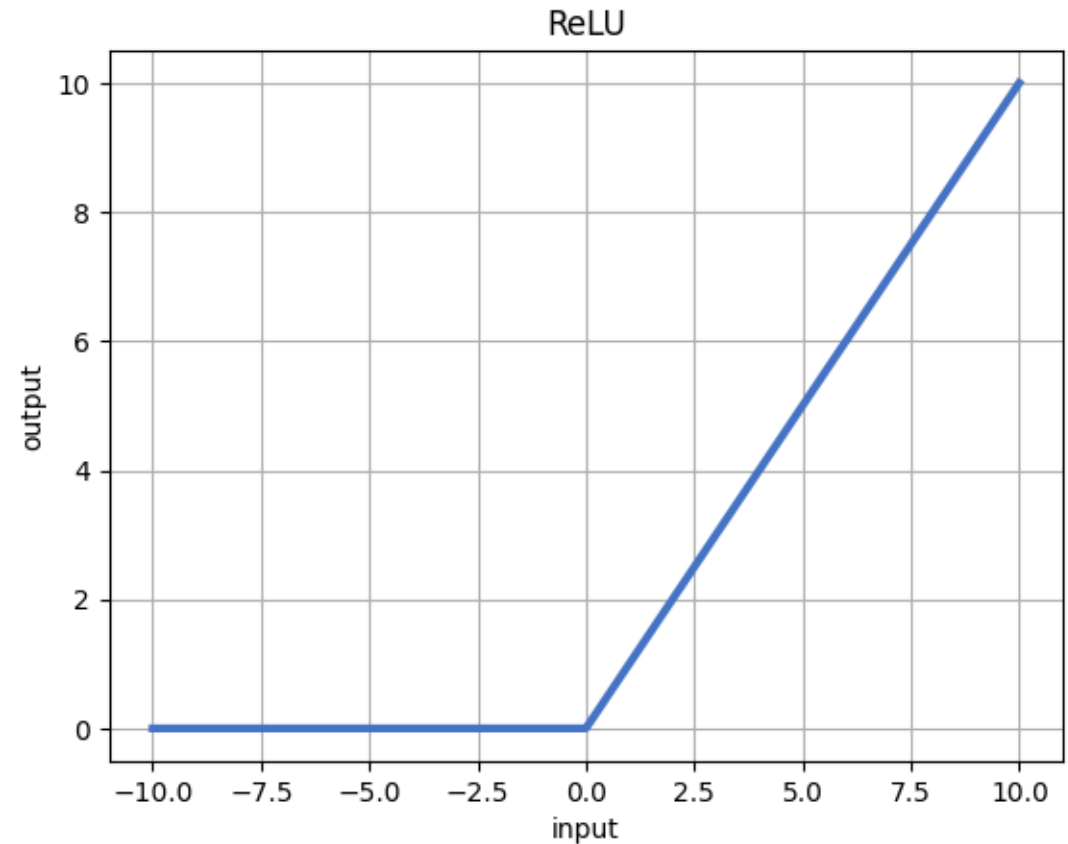
## Минусы:

- Для отрицательных значений аргумента равна нулю, что может привести к затуханию градиента

## Плюсы:

- Простота вычисления

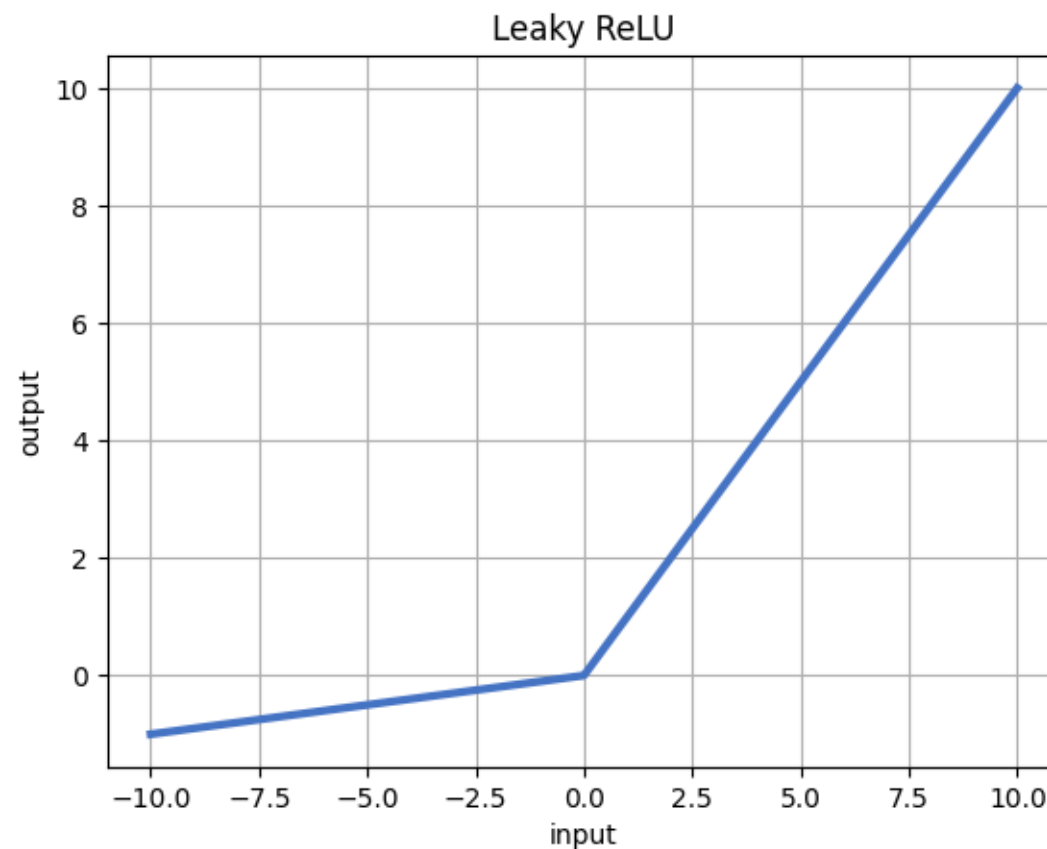
Одна из наиболее часто используемых функций активации.



$$ReLU(x) = \max(0, x)$$

# Leaky ReLU

Очень похожа на ReLU; благодаря уклону в отрицательной области становится более симметричной.  
Также обеспечивает ненулевой градиент в областях слева и справа от нуля

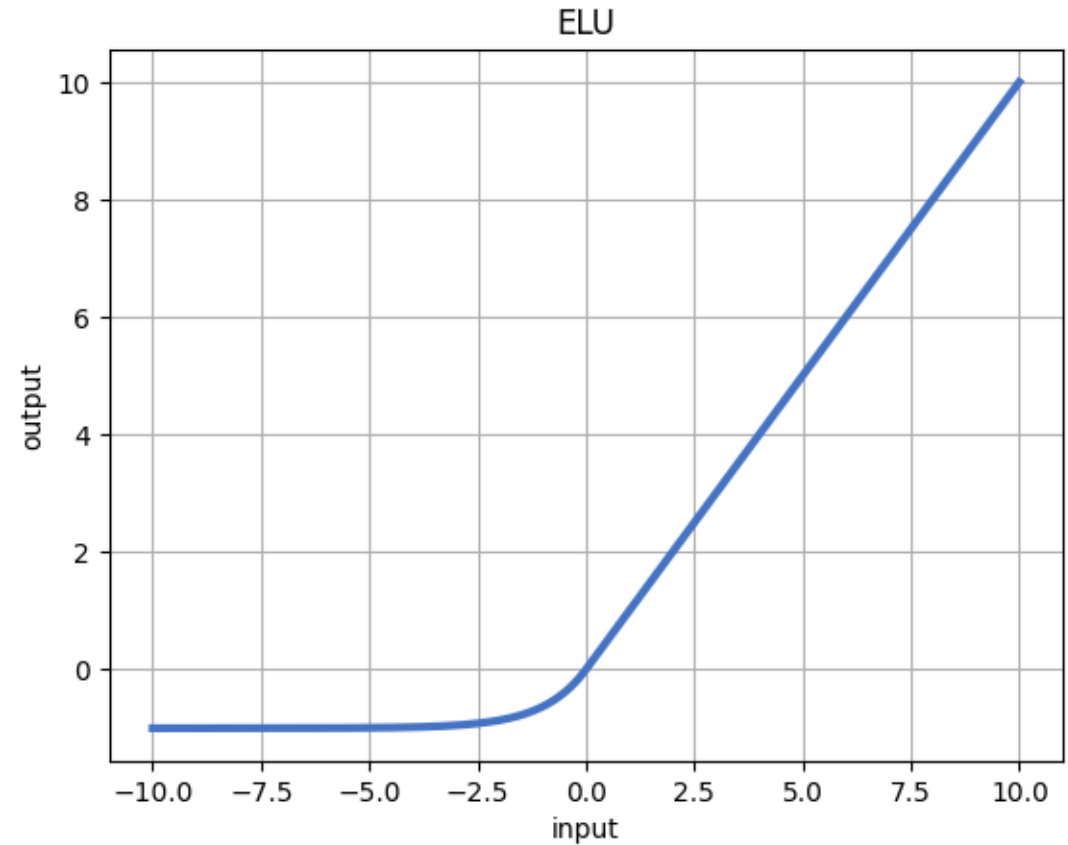


$$\text{Leaky ReLU}(x) = \max(\alpha x, x),$$

где  $0 < \alpha \ll 1$

# ELU (Exponential Linear Unit)

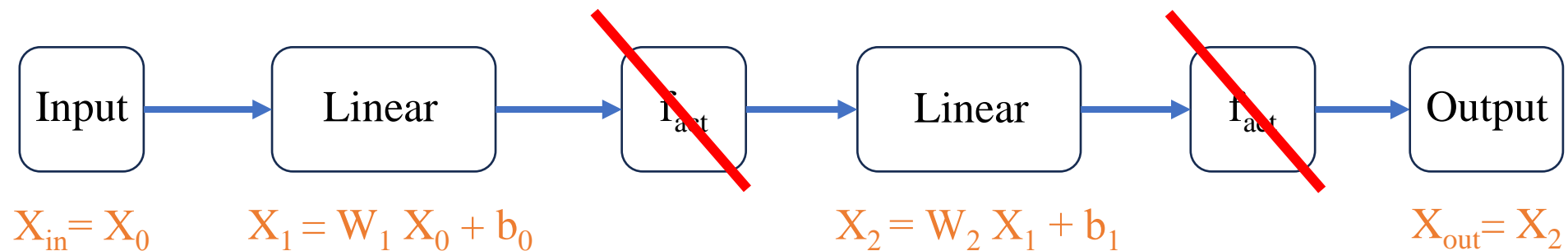
Попытка сделать более гладкую функцию в отрицательной области. Иногда может хорошо работать, но требует вычисления экспоненты



$$ELU(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0; \\ x, & x > 0. \end{cases}$$

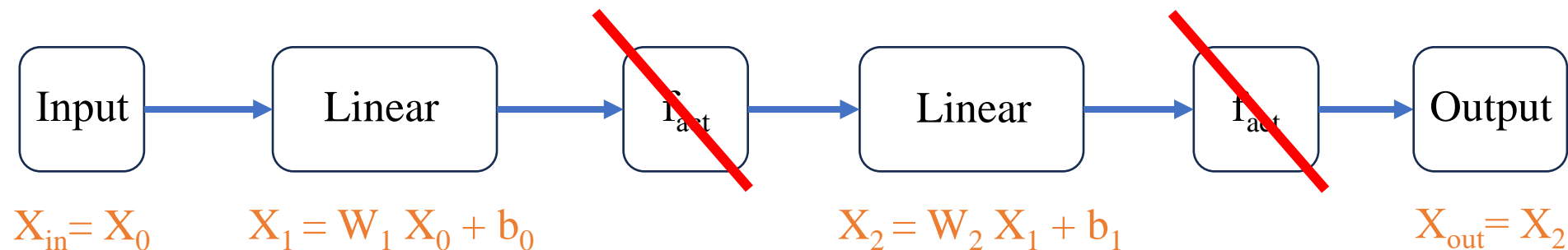
где  $\alpha > 0$

# Зачем нужны активации?



Найдем выход данного вычислительного графа:

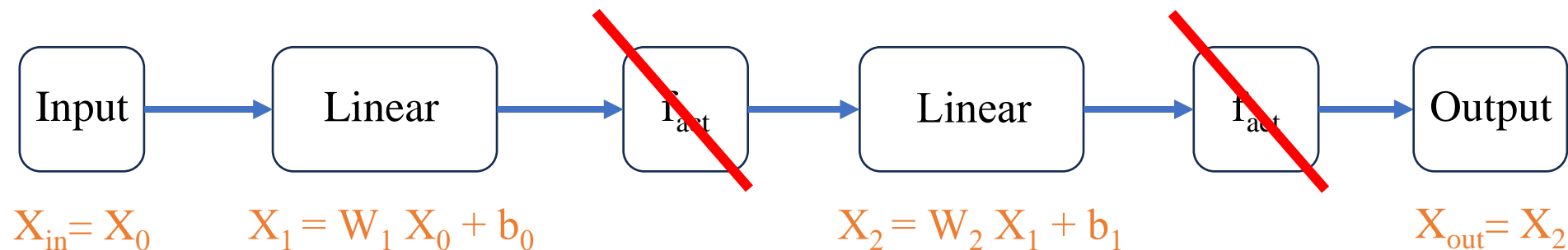
# Зачем нужны активации?



Найдем выход данного вычислительного графа:

$$\begin{aligned} X_{\text{out}} &= W_2 X_1 + b_1 = \\ &= W_2 (W_1 X_0 + b_0) + b_1 = \\ &= \textcolor{brown}{W}_2 \textcolor{brown}{W}_1 X_0 + \textcolor{blue}{W}_2 b_0 + b_1 = \\ &= \tilde{W} X_0 + \tilde{b} \end{aligned}$$

# Зачем нужны активации?



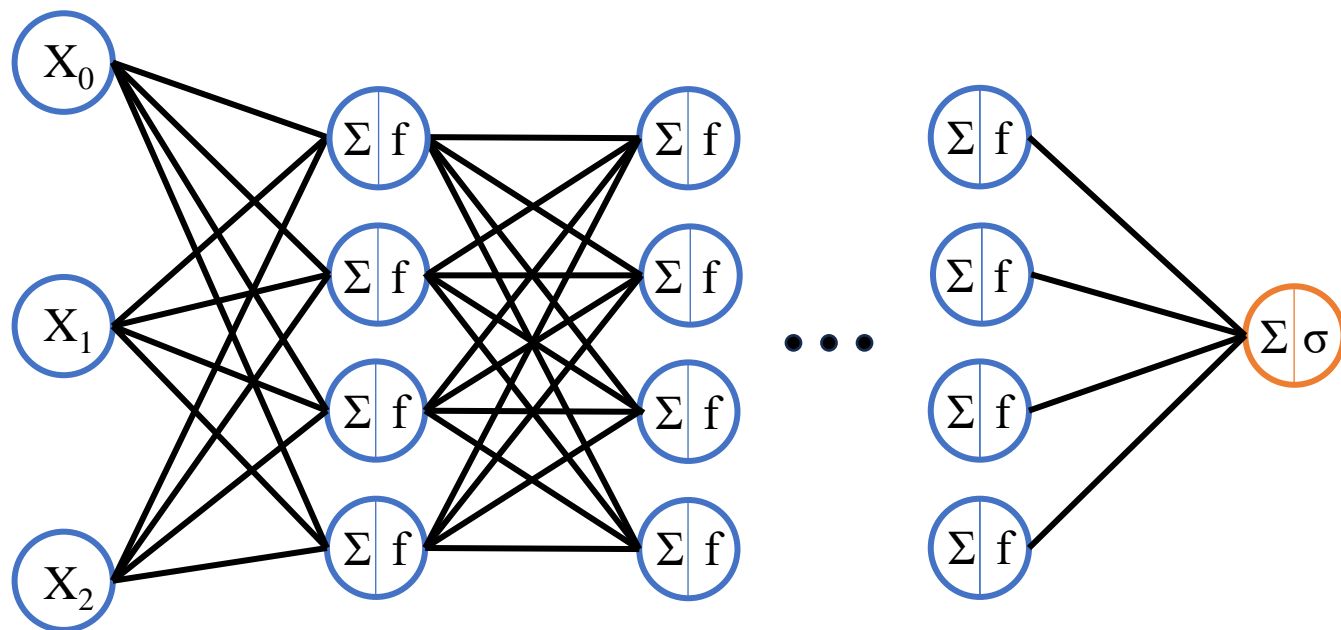
Найдем выход данного вычислительного графа:

$$\begin{aligned} X_{\text{out}} &= W_2 X_1 + b_1 = \\ &= W_2 (W_1 X_0 + b_0) + b_1 = \\ &= \mathbf{W_2 W_1} X_0 + \mathbf{W_2 b_0} + b_1 = \\ &= \tilde{W} X_0 + \tilde{b} \end{aligned}$$

В результате двух последовательных *линейных* преобразований мы, очевидно, также получили *линейное отображение*. Добавление функций активации позволяет нам получать *нелинейные преобразование* и как следствие восстанавливать более сложные зависимости в данных.

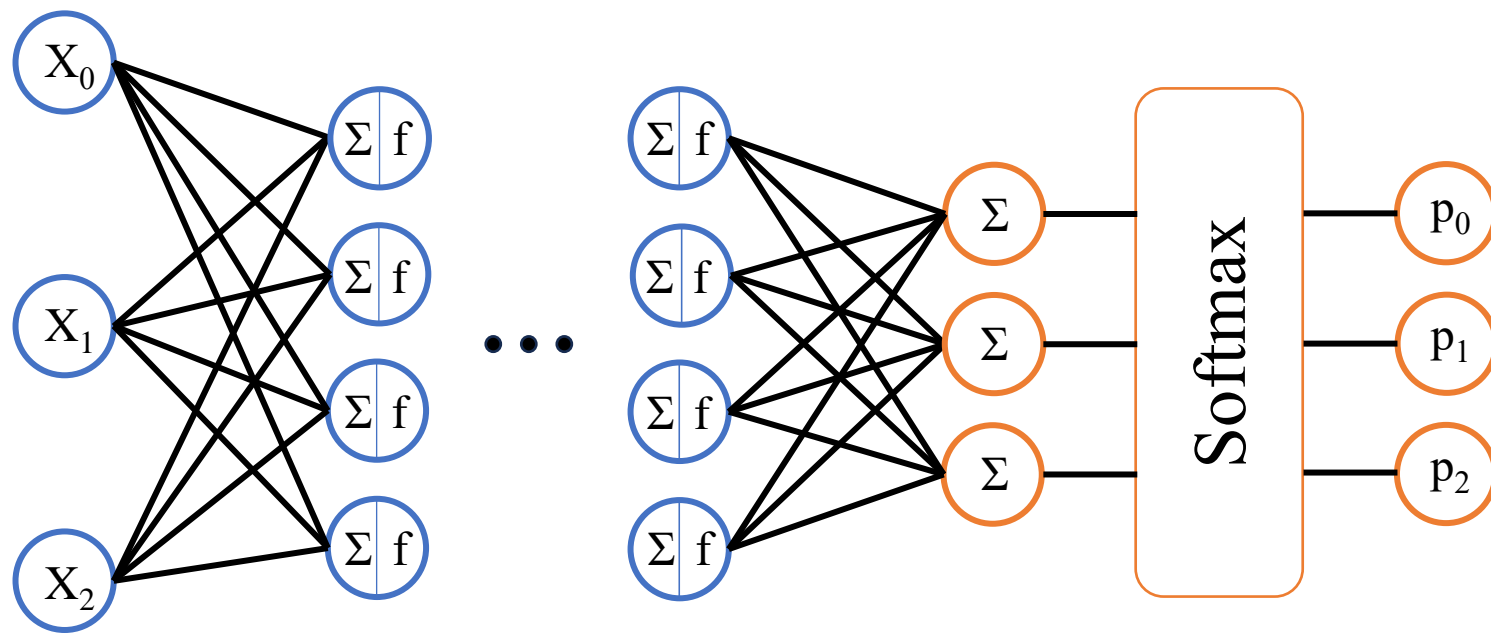


# Бинарная классификация



В случае **бинарной классификации** мы хотим получить вероятности принадлежности объекта к одному из классов. Для этого можно в качестве последней активации выбрать **сигмоиду**, которая переведет вещественный выход нейронной сети в интервал от 0 до 1.

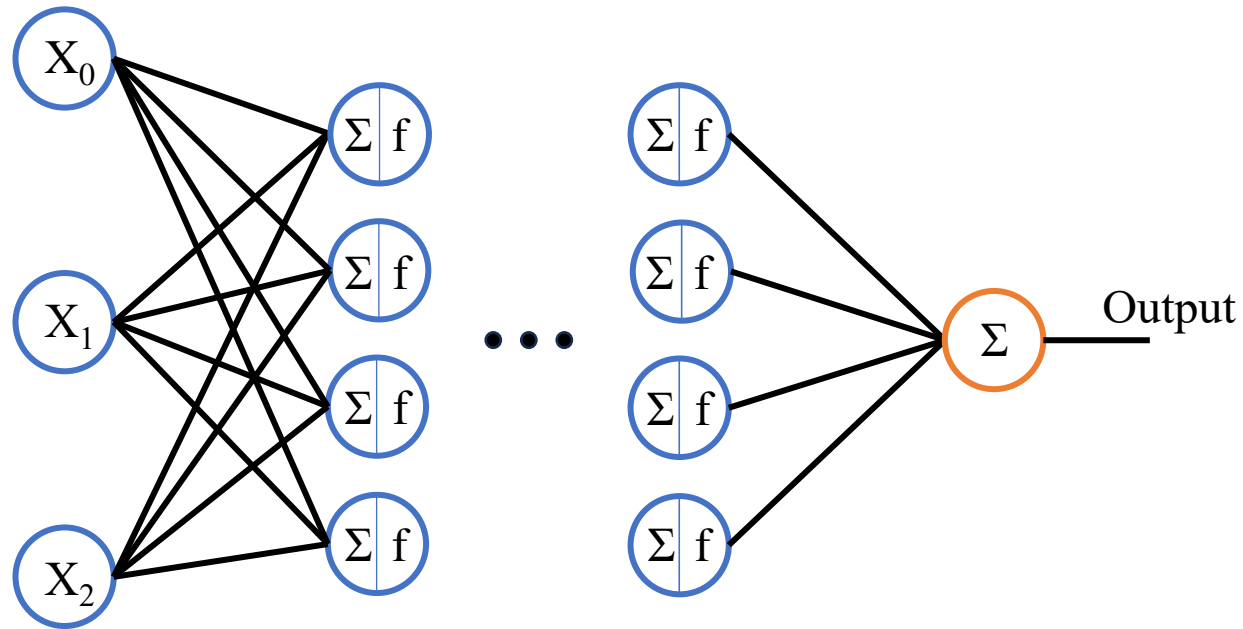
# Многоклассовая классификация



Если мы решаем задачу в которой больше двух классов, то необходимо настроить архитектуру так, чтобы на выходе она выдавала  $K$  вещественных чисел, соответствующих числу классов, а затем превратить их в вероятности (с некоторыми оговорками) принадлежности к тому или иному классу. Сделать это можно с помощью функции **softmax**:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

# Регрессия



В задаче **регрессии** нам необходимо предсказывать вещественное число, поэтому функции активации на последнем слое обычно нет.

Если, например, предсказывается число от 0 до 100, то можно воспользоваться сигмойдой на последнем слое и умножить ответ на 100. Также можно превратить задачу регрессии в задачу классификации, если предсказываются целые числа из некоего диапазона. Естественно использование тех или иных приемов зависит от решаемой задачи.