

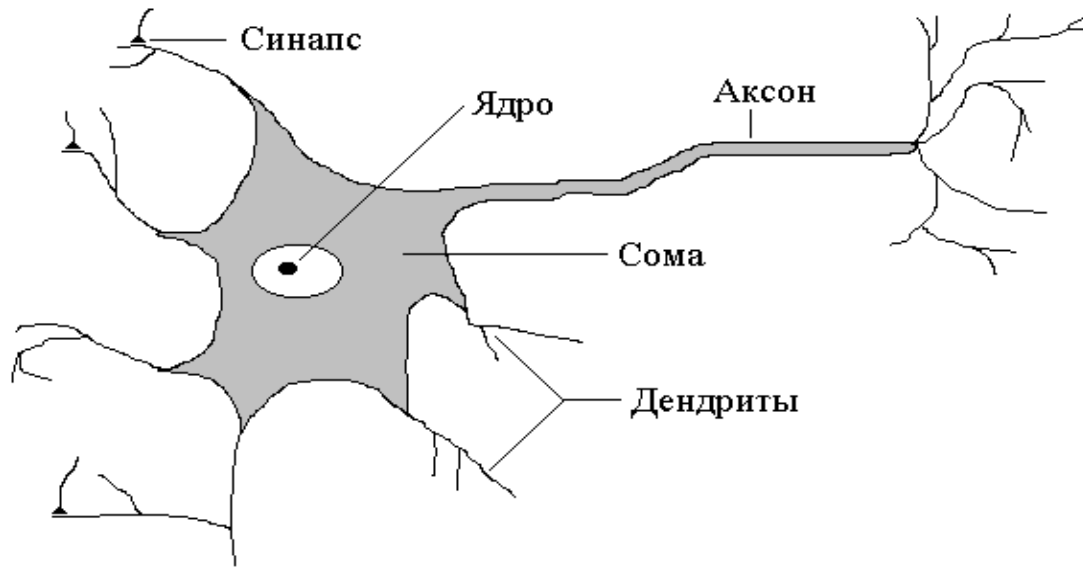
**Севастопольский государственный университет
Институт информационных технологий**

Методы и системы искусственного интеллекта

Введение в нейросети

Бондарев Владимир Николаевич

Биологический нейрон

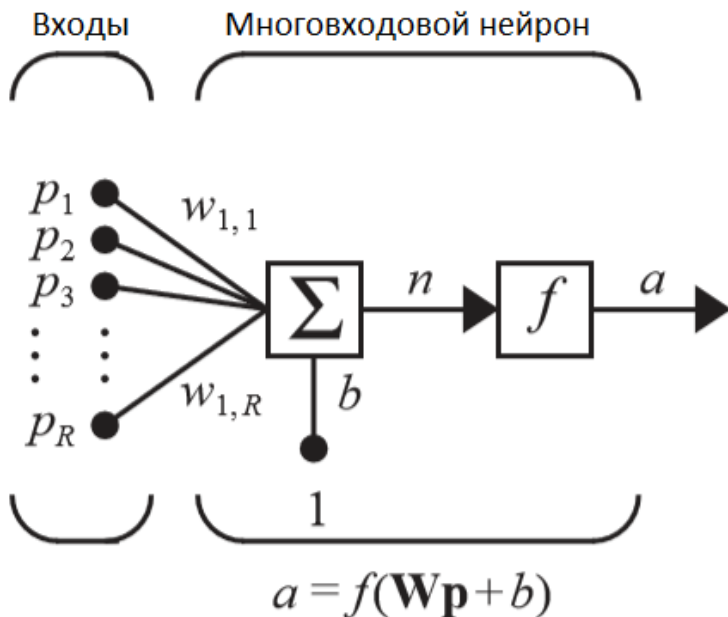


Дендриты – ветвеобразные отростки, которые обеспечивают сбор сигналов от других нейронов или рецепторов.

Сомма нейрона представляет тело клетки. В соме происходят сложные биохимические процессы, благодаря которым осуществляются нелинейные преобразования сигналов, поступающих через дендриты.

Аксон является отростком клетки, по которому ее выходной сигнал поступает на дендриты других нейронов. Аксон разветвляется на большое число волокон. Место соединения волокон с дендритами называется **синапсом**.

Формальный нейронный элемент Маккаллоха-Питтса



p_1, p_2, \dots, p_R — ВХОДНЫЕ СИГНАЛЫ

$w_{11}, w_{12}, \dots, w_{1R}$ — веса синаптических связей

Сетевой выход n равен:

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \dots + w_{1,R}p_R + b.$$

Или матричной форме

$$n = \mathbf{W}\mathbf{p} + b,$$

где \mathbf{W} — матрица весов ($1 \times R$), \mathbf{p} — вектор входных сигналов, b — смещение (порог); R — количество входов НЭ.

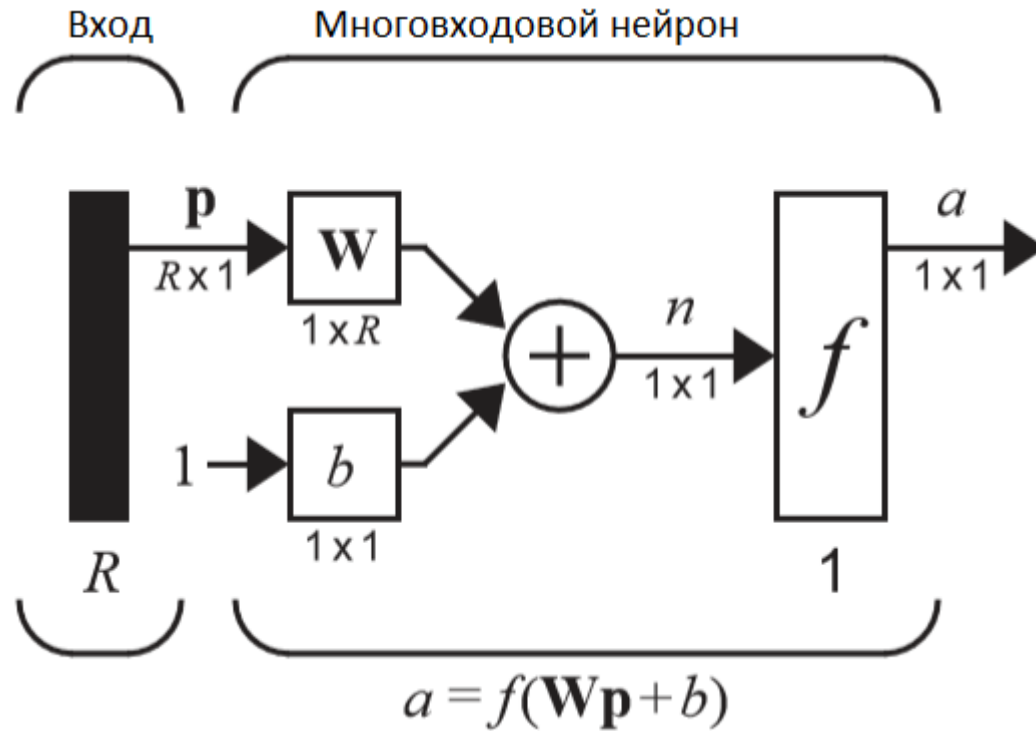
Выход нейрона

$$a = f(\mathbf{W}\mathbf{p} + b).$$

Функция преобразования $f(n)$ для НЭ Маккаллоха-Питтса соответствует функции Хевисайда

$$f(n) = H(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

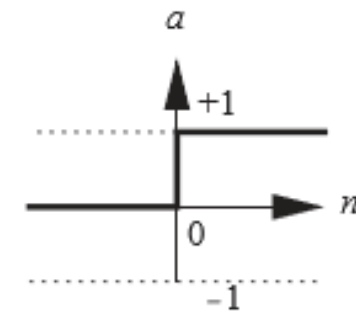
Многовходовой нейрон – упрощенное обозначение



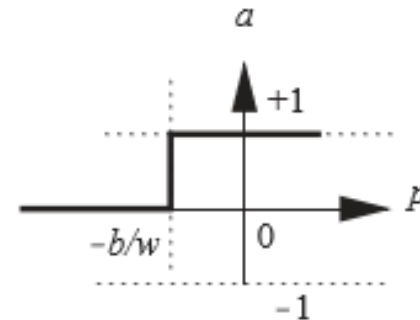
Функции активации

Часто используемые функции:

1) пороговая функция (единичная функция Хевисайда)

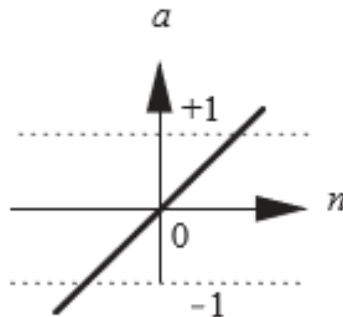


$$a = \text{hardlim}(n)$$

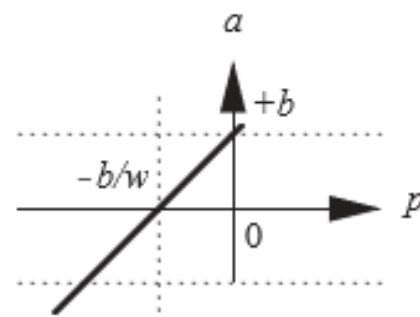


$$a = \text{hardlim}(wp + b)$$

2) линейная функция : $a=n$



$$a = \text{purelin}(n)$$

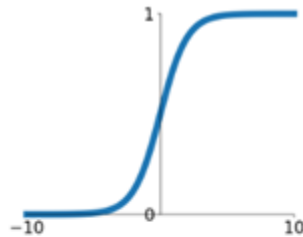


$$a = \text{purelin}(wp + b)$$

Активационные функции

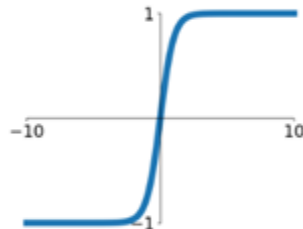
Sigmoid

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



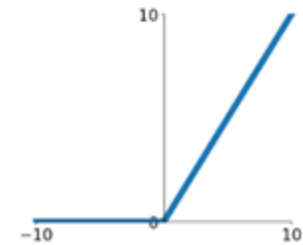
tanh

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



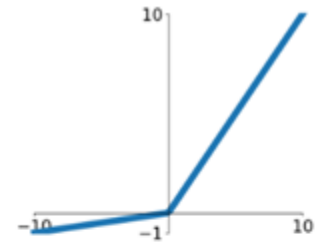
ReLU

$$\max(0, x)$$



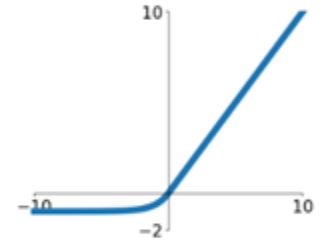
Leaky ReLU

$$\max(0.1x, x)$$



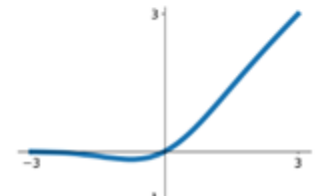
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



GELU

$$\approx x\sigma(1.702x)$$

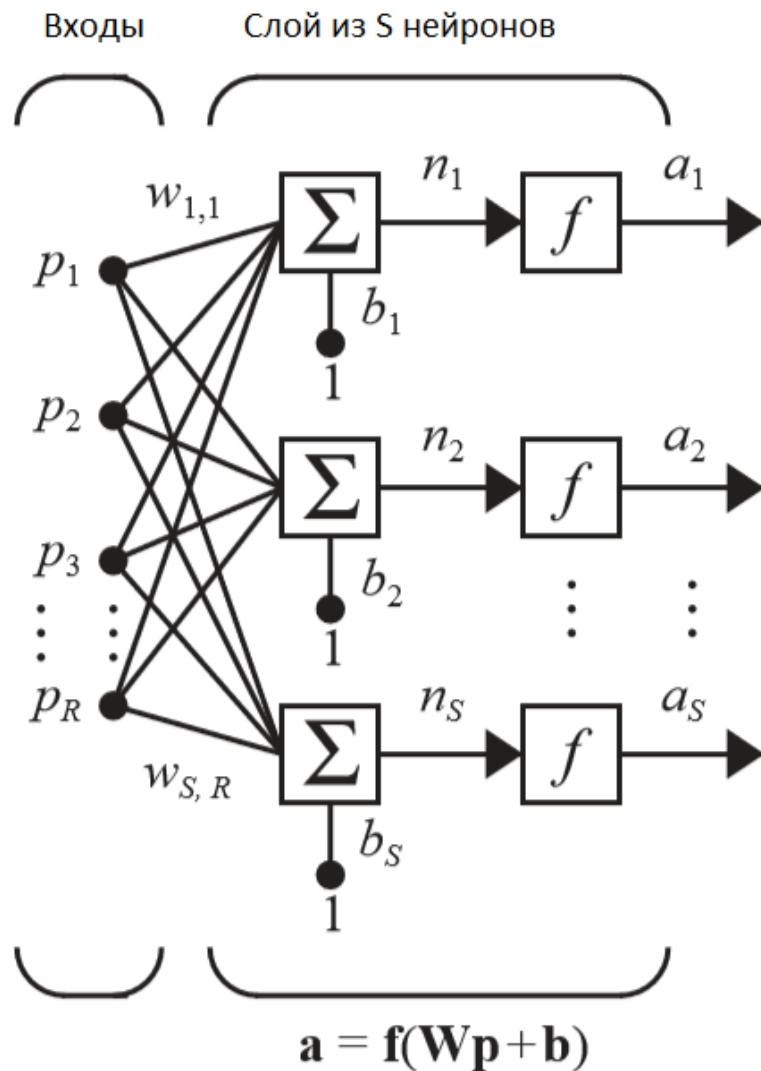


Архитектура нейронных сетей

ИНС состоит из большого числа взаимосвязанных НЭ.

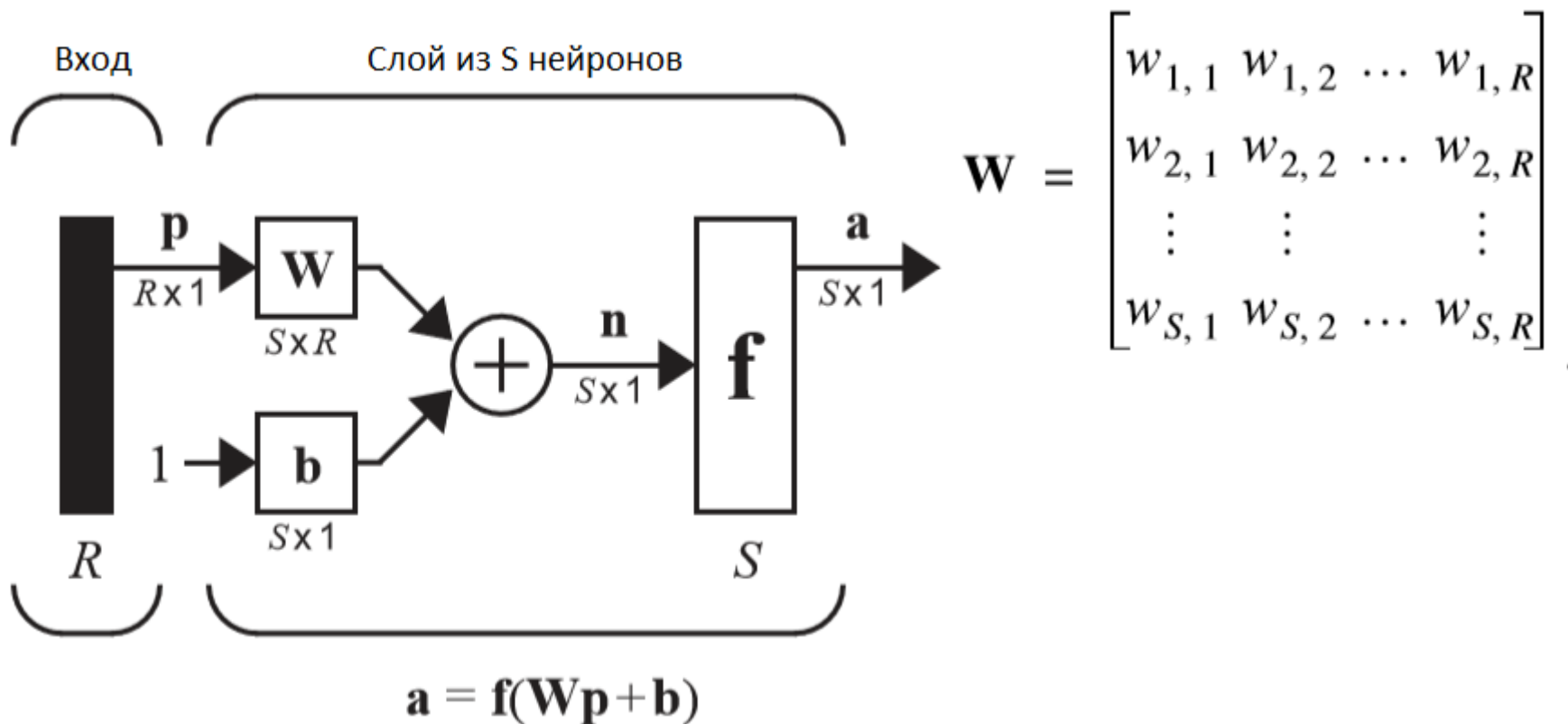
Выделяют следующие основные разновидности архитектур ИНС:

- **однослойные ИНС с прямыми связями;**



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}.$$

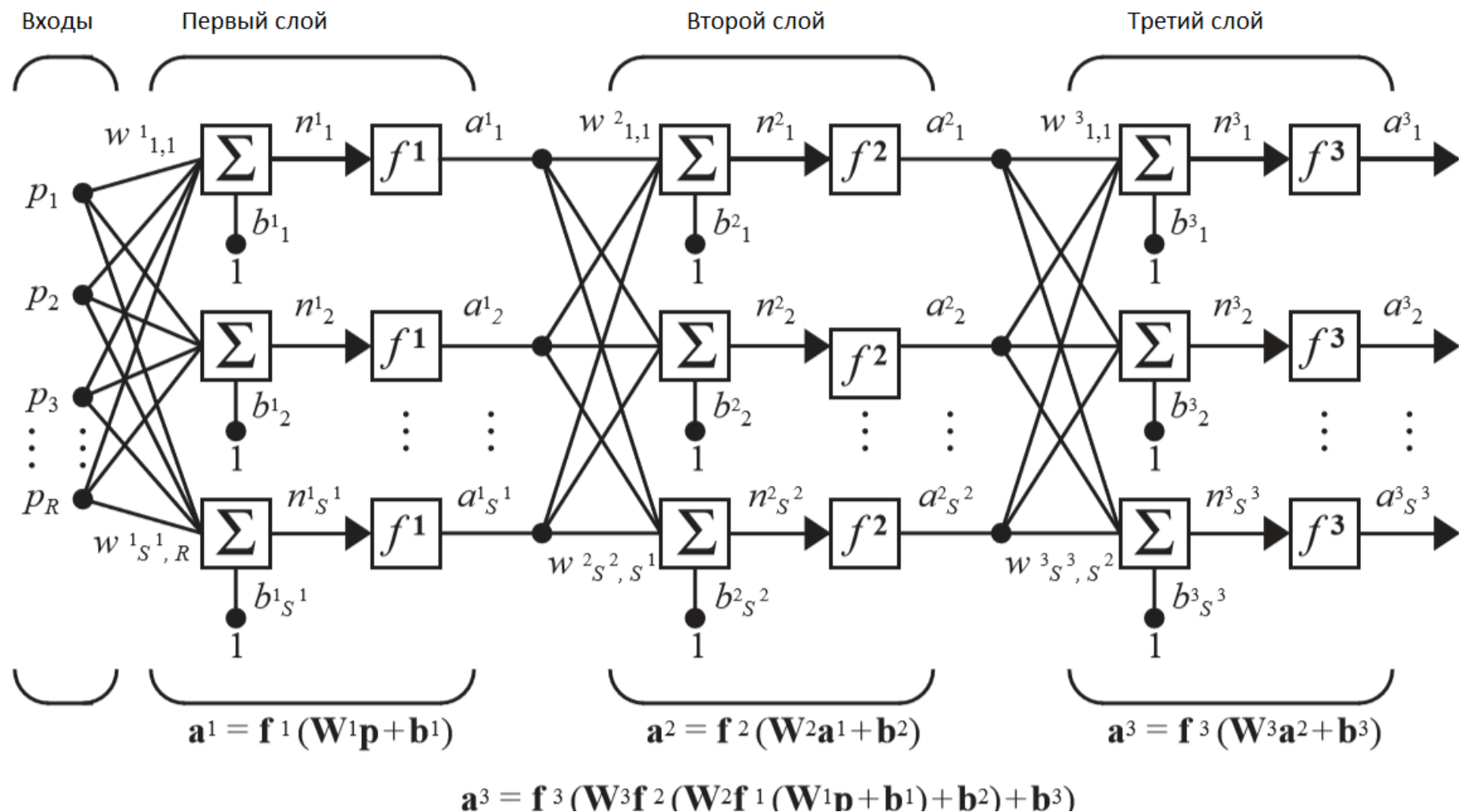
Слой из S нейронов – упрощенное обозначение



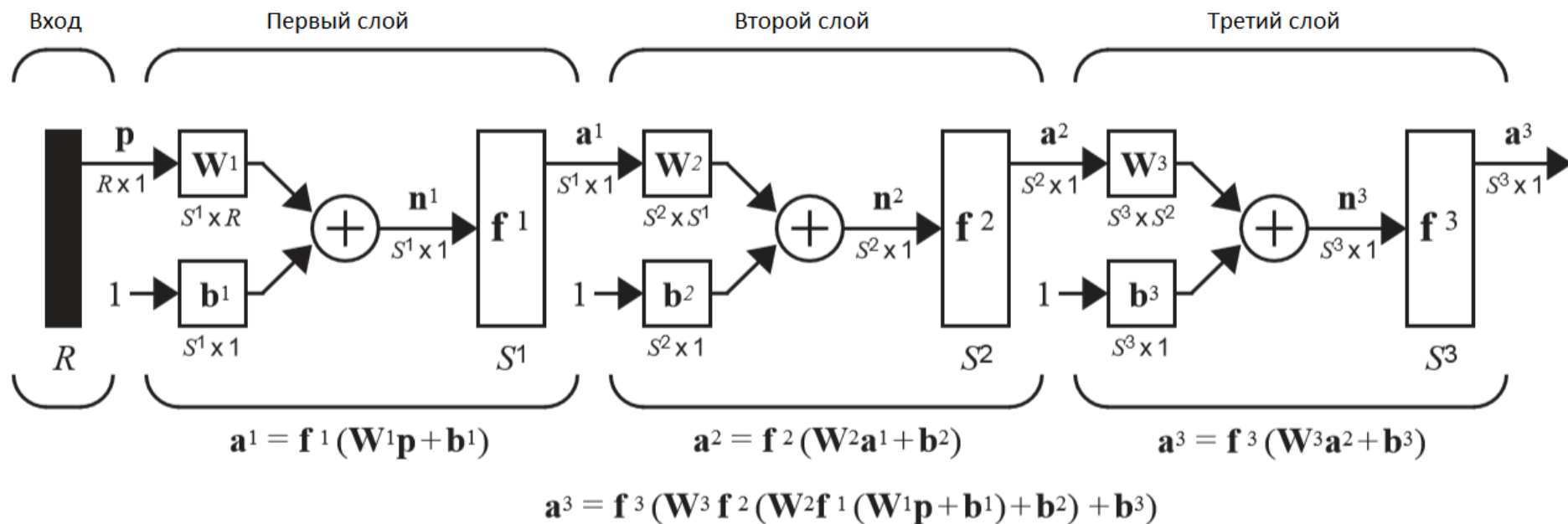
Архитектура нейронных сетей

- многослойные ИНС с прямыми связями

Трехслойная сеть



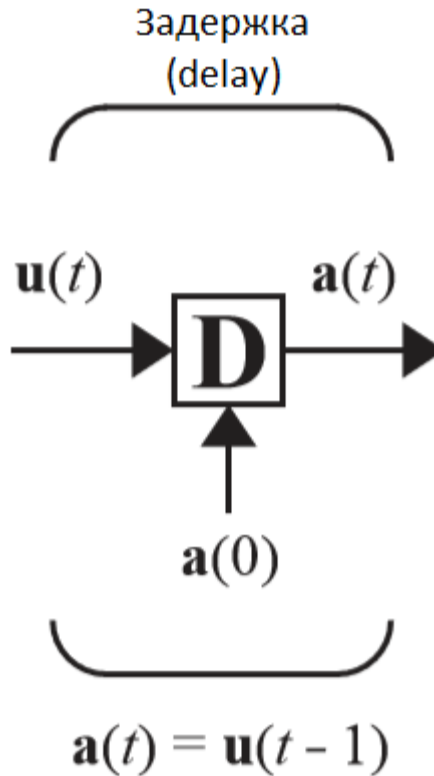
Трехслойная сеть – упрощенное обозначение



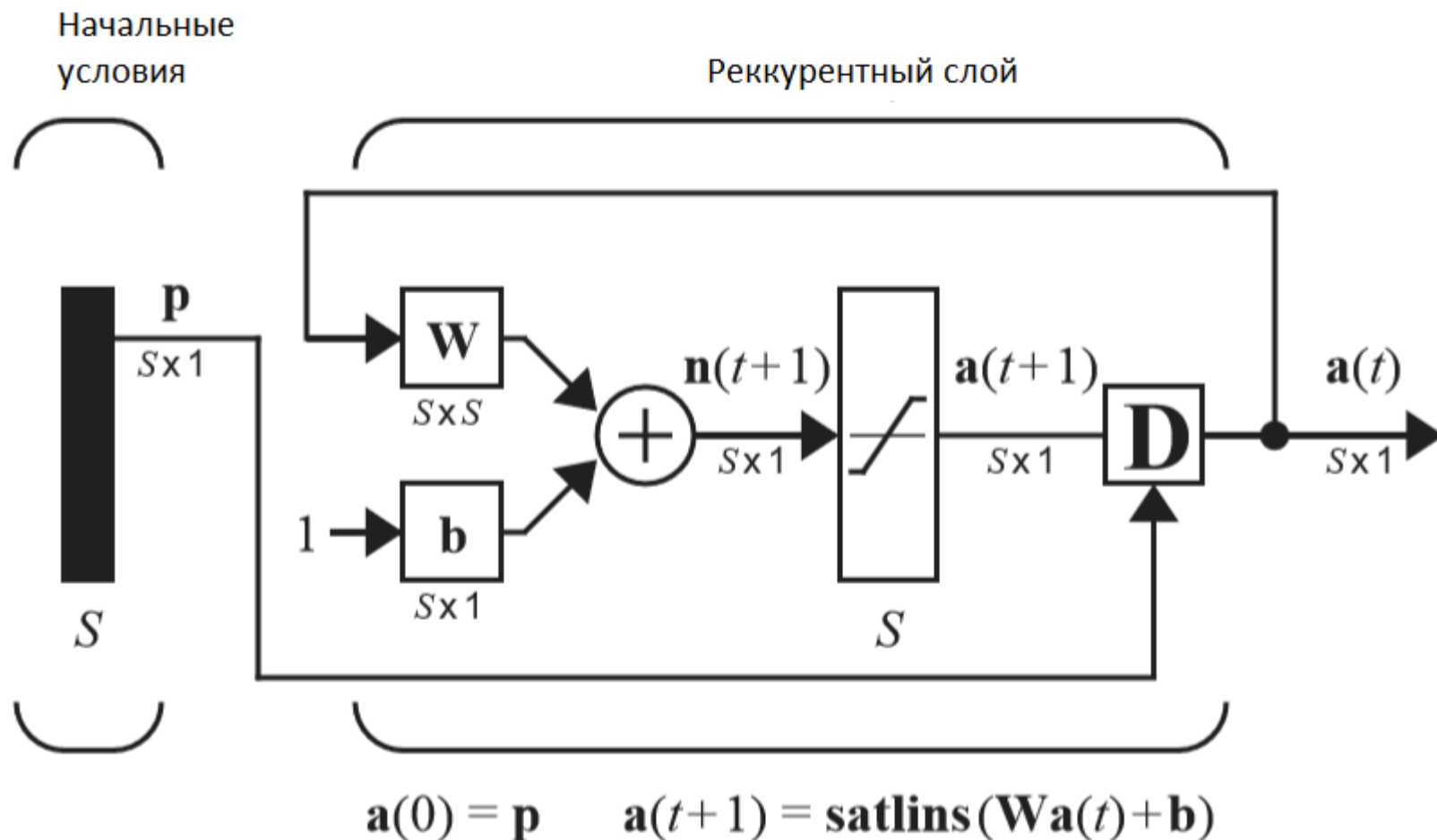
Архитектура нейронных сетей

- рекуррентные (сети с обратными связями)

Для организации обратной связи сети используют блок задержки



Реккурентные сети



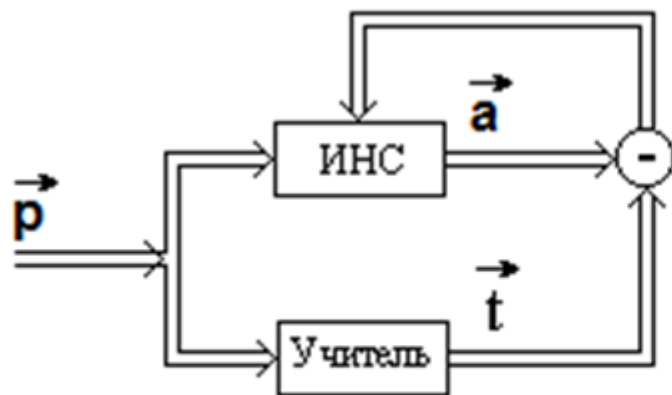
Последующие значения выходов

$$\mathbf{a}(1) = \text{satlins}(\mathbf{W}\mathbf{a}(0) + \mathbf{b}), \mathbf{a}(2) = \text{satlins}(\mathbf{W}\mathbf{a}(1) + \mathbf{b}), \dots$$

Виды обучения ИНС

Различают следующие виды обучения ИНС :

а) - **обучение с учителем**
(supervised learning)



а)

Обучающие данные содержат обучающие примеры $\mathbf{p}(q)$ и метки $\mathbf{t}(q)$:

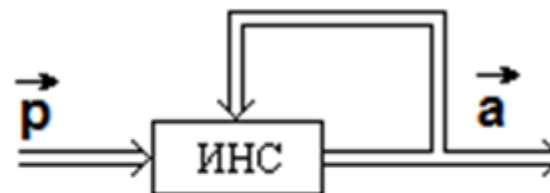
$$D = \{\mathbf{p}(q), \mathbf{t}(q)\}; q = 1..Q$$

Цель обучения:

минимизировать «ошибку»

$$\mathbf{e}(q) = L(\mathbf{t}(q) - \mathbf{a}(q))$$

б) - **обучение без учителя**
(unsupervised learning)



б)

Обучающие данные содержат только обучающие примеры $\mathbf{p}(q)$:

$$D = \{\mathbf{p}(q)\}; q = 1..Q$$

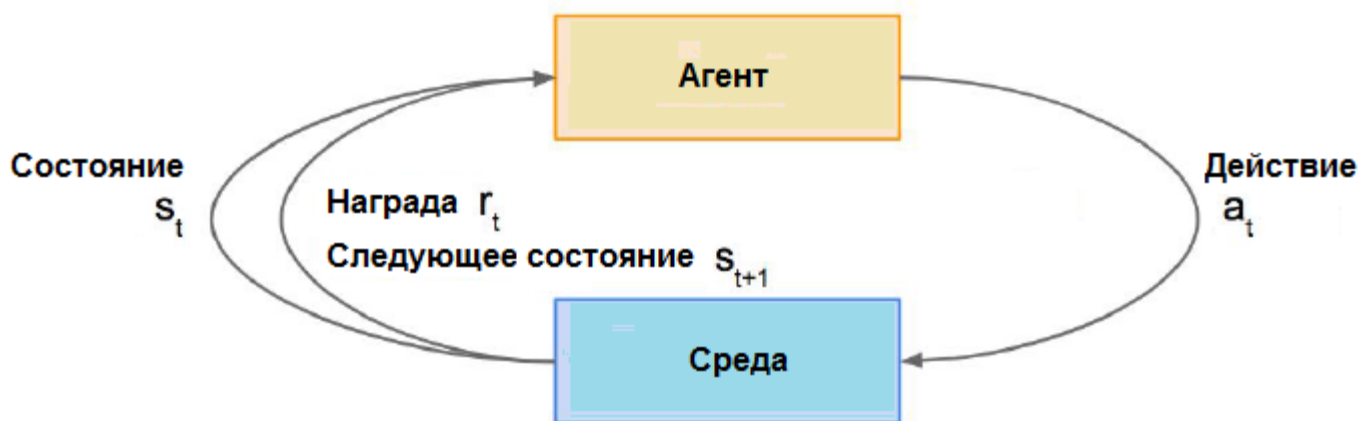
Цель обучения: в общем случае восстановить многомерное распределение вероятностей входных данных

Виды обучения ИНС

в) - обучение с подкреплением (RL, reinforcement learning)

Имеется агент, взаимодействующий с окружающей средой, которая в ответ на действия агента, обеспечивает его вознаграждением.

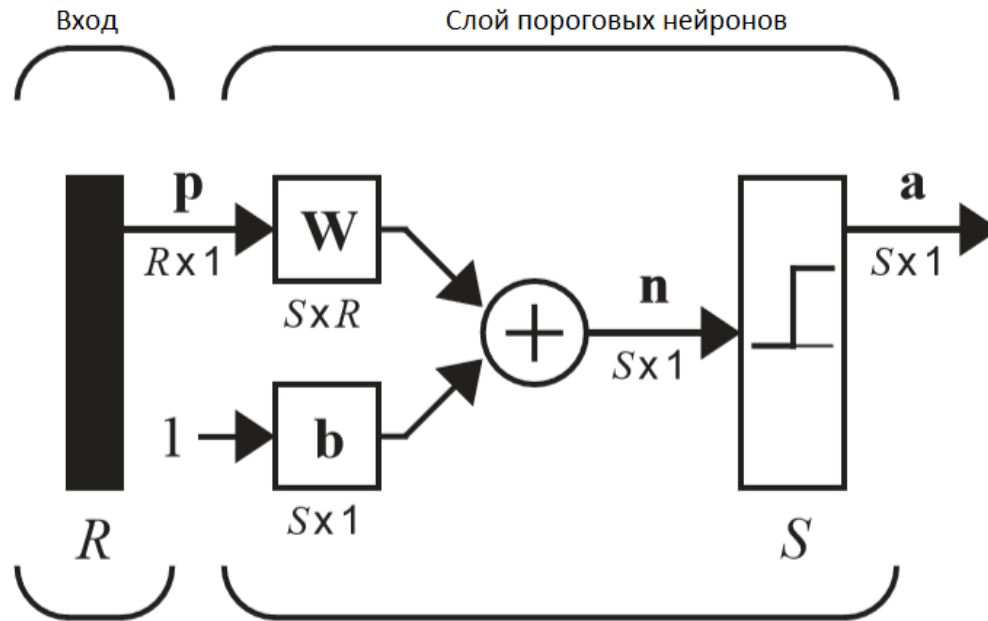
Цель обучения: научиться действовать, так чтобы максимизировать вознаграждение



В среде RL вы не учите агента, что и как он должен делать, вместо этого **вы даете агенту награду за каждое выполненное действие**. Таким образом, обучение превращается в процесс проб и ошибок.

Персептрон: архитектура и математическое описание

Общая архитектура персептрона
(однослойного)



Выход сети :

$$\mathbf{a} = \text{hardlim}(\mathbf{W}\mathbf{p} + \mathbf{b})$$

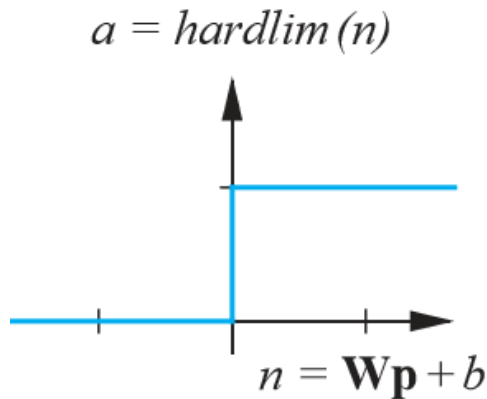
Персептрон: архитектура и математическое описание

Матрица весов персептрона

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}, \quad {}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}.$$

Тогда выход отдельного нейрона сети будет равен:

$$a_i = \text{hardlim}(n_i) = \text{hardlim}({}_i\mathbf{w}^T \mathbf{p} + b_i).$$



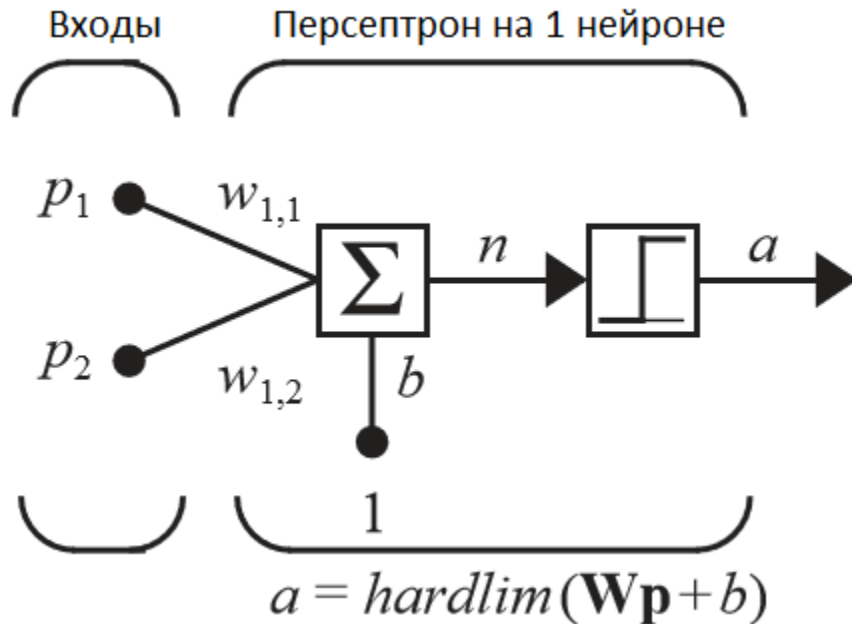
Если скалярное произведение i -ой строки матрицы ${}_i\mathbf{w}^T$ на входной вектор \mathbf{p} , будет больше или равно $-b_i$, то выход будет равен 1, иначе - 0.

Таким образом, каждый нейрон сети делит пространство входных сигналов на две области. Персептрон, состоящий из S нейронов, может классифицировать входные образы на 2^S классов.

Исследуем границы между эти областями.

Простой персептрон: граница решения

Рассмотрим персептрон из одного нейрона (**простой персептрон**) с 2-мя входами.



Выход нейрона:

$$\begin{aligned} a &= \text{hardlim}(n) = \text{hardlim}(\mathbf{W}\mathbf{p} + b) \\ &= \text{hardlim}({}_1\mathbf{w}^T \mathbf{p} + b) \\ &= \text{hardlim}(w_{1,1}p_1 + w_{1,2}p_2 + b) \end{aligned}$$

Нейрон разделяет входное пространство на 2 области.

Граница решения между областями определится из условия:

$$n = {}_1\mathbf{w}^T \mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = 0.$$

Рассмотрим конкретный пример. Пусть

$$w_{1,1} = 1, w_{1,2} = 1, b = -1.$$

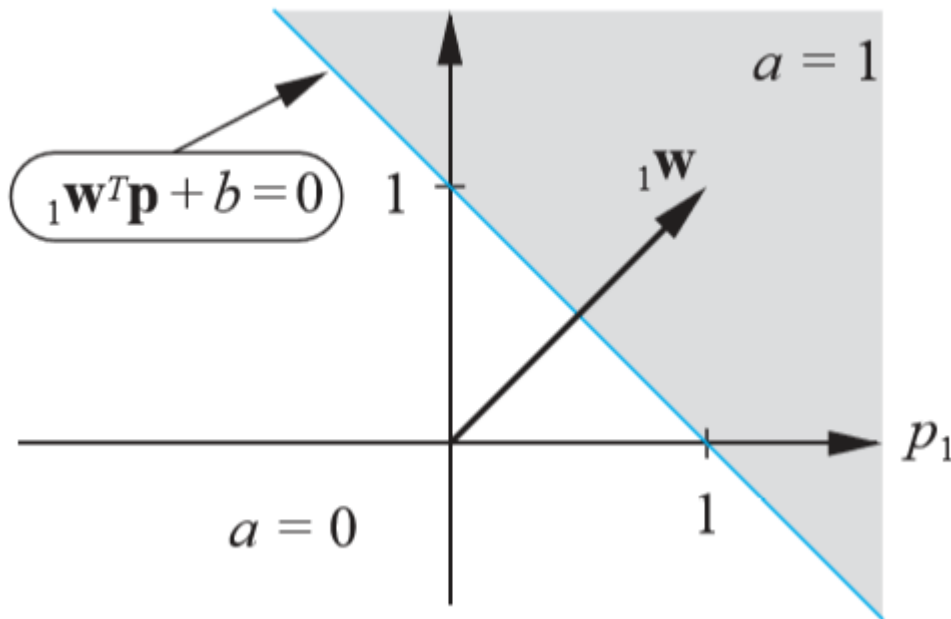
Простой персептрон: граница решения

Тогда граница решения между областями определится из условия:

$$n = {}_1\mathbf{w}^T \mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = p_1 + p_2 - 1 = 0.$$

Это уравнение задаёт границу решения в виде линии. Чтобы изобразить линию, найдем точки её пересечения с осями (p_1 , p_2):

$$p_1 = -\frac{b}{w_{1,1}} = -\frac{-1}{1} = 1 \quad p_2 = -\frac{b}{w_{1,2}} = -\frac{-1}{1} = 1$$



С одной стороны границы выход равен 1, а с другой - равен 0. Например для точки

$$\mathbf{p} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}^T$$

выход равен 1, т.к.

$$a = \text{hardlim} \left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1 \right) = 1$$

Граница решения: простой персептрон из одного нейрона

Мы также можем построить границу графически.

Отметим, что **граница всегда ортогональна вектору ${}_1\mathbf{w}$** , как изображено на рис.

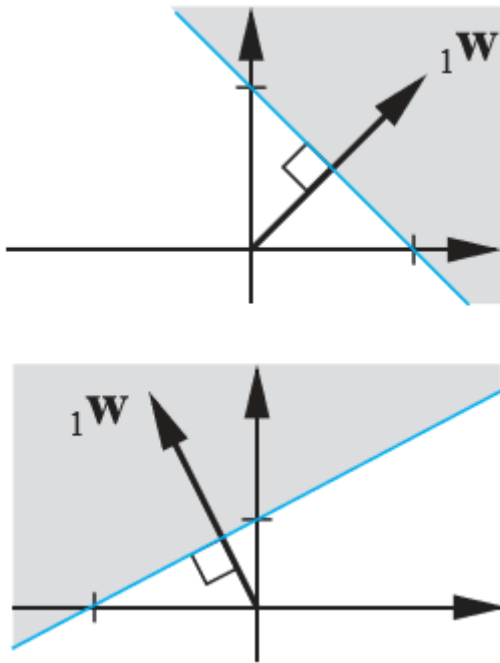
Действительно, граница решения определяется условием:

$${}_1\mathbf{w}^T \mathbf{p} + b = 0. \quad (1)$$

Для всех точек, принадлежащих границе, скалярное произведение входного вектора \mathbf{p} на вектор весов имеет одно и то же значение. Это означает, что эти входные векторы \mathbf{p} имеют одно и то же значение проекции на вектор весов, т.о., *их концы должны лежать на линии ортогональной вектору весов*.

Отметим, что **вектор весов всегда направлен в сторону области, где выход нейрона равен 1**.

После того как определено направление вектора весов, **значение смещения** определяется путем выбора точки на границе и решения уравнения (1)



Правило обучения простого персептрона

Правило обучения персептрона представляет собой процедуру **обучения с учителем** и заключается в поиске параметров (весов и смещений), которые обеспечивают совпадения желаемой t и действительной реакции персептрона a на заданный входной вектор \mathbf{p} . Обучение осуществляется на основе обучающего **множества примеров** «вход-выход»:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

В ходе выполнения процедуры обучения персептрона на его вход подается каждый входной вектор \mathbf{p} обучающего множества, вычисляется реакция a , которая сравнивается с требуемым выходом t и вычисляется ошибка e :

$$e = t - a.$$

В зависимости от значения ошибки корректируются веса персептрона :

$$e = 1, \quad {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}.$$

$$e = -1, \quad {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}.$$

$$e = 0, \quad {}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}.$$

Приведенные три правила можно записать в виде одного выражения:

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + (t - a)\mathbf{p} = {}_1\mathbf{w}^{old} + e\mathbf{p}$$

Для обновления смещений (\sim весу, для которого $p=1$) используется выражение:

$$b^{new} = b^{old} + e.$$

Правило обучения простого персептрона

Алгоритм обучения бинарного персептрона при $y=+1/-1$:

1. Инициализируйте все веса нулевыми значениями: $\mathbf{w} = \mathbf{0}$;
2. Для каждой обучающей точки данных \mathbf{x} с признаками $\mathbf{f}(\mathbf{x})$ и истинной меткой класса $y^* \in \{-1, +1\}$ повторяйте:
 - (a) Классифицируйте точку данных \mathbf{x} , используя текущий вектор весов \mathbf{w} :

$$y = \begin{cases} +1, & \text{если } net_w(\mathbf{x}) = \mathbf{w}^T \mathbf{f}(\mathbf{x}) \geq 0 \\ -1, & \text{если } net_w(\mathbf{x}) = \mathbf{w}^T \mathbf{f}(\mathbf{x}) < 0 \end{cases},$$

где y предсказанная метка;

(b) Сравните предсказанную метку y с истинной меткой y^* :

- если $y = y^*$, ничего не делайте;
- иначе, если $y \neq y^*$, обновите веса: $\mathbf{w} \leftarrow \mathbf{w} + y^* \mathbf{f}(\mathbf{x})$.

3. Если для каждой обучающей точки данных все метки предсказаны правильно, то завершите работу. В противном случае повторите шаг 2.

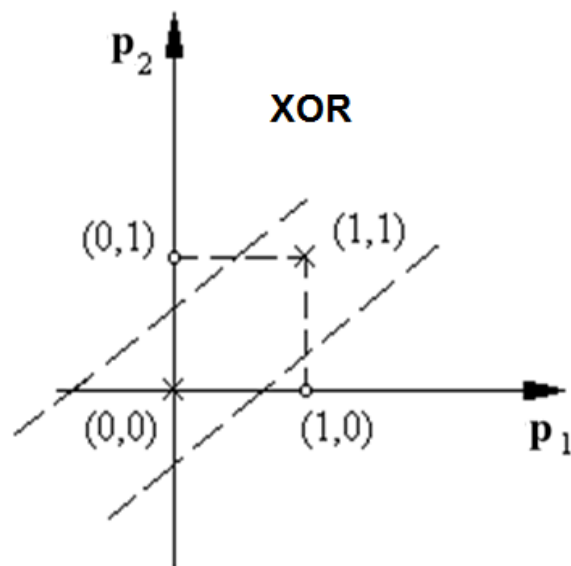
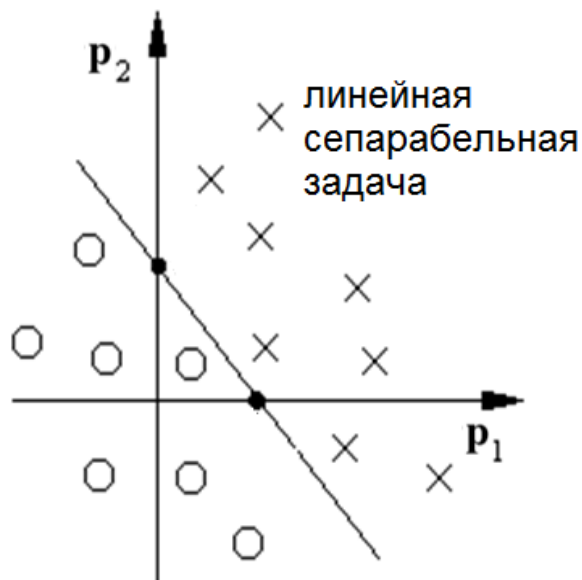
Ограничения персептрона

Сетевая функция простого персептрона играет роль дискриминантной функции. В общем случае эта функция описывает уравнение гиперплоскости:

$${}_1\mathbf{w}^T \mathbf{p} + b = 0.$$

Поэтому простой персептрон может классифицировать только такие образы входного пространства, которые разделимы с помощью гиперплоскости. Иными словами, задачи классификации, решаемые простым персептроном, – это **линейные сепарабельные задачи**.

Классификация в двумерном пространстве



Задачи классификации

Здесь отклик модели Y – это категориальная переменная, которая обозначает один из классов, например, $C=\{\text{спам, не спам}\}$, или класс рукописной цифры $C=\{0,1,\dots,9\}$.

При этом целью является:

- Построение классификатора $\hat{Y} = \hat{f}(\mathbf{p})$, который предсказывает метку класса принадлежности входного вектора \mathbf{p} ;
- Оценка ошибок классификации;
- Понимание роли различных предикторов $\mathbf{p} = (p_1, p_2, \dots, p_R)$.

Наиболее распространенным подходом для количественного описания оценки \hat{f} является вычисление **частоты ошибок** (error rate) **на обучающем наборе**, выраженной в виде доли ошибок:

$$Err_{Tr} = \frac{1}{N} \sum_{i=1}^N I(y_i \neq \hat{y}_i)$$

где $I(y_i \neq \hat{y}_i)$ -- индикаторная переменная, равная 1, если $y_i \neq \hat{y}_i$ и 0, если $y_i = \hat{y}_i$.

По результатам обучения оценивается **частота ошибок на тестовых данных**.

$$Err_{Te} = \frac{1}{M} \sum_{i=1}^M I(y_i \neq \hat{y}_i)$$

Задачи регрессии

Задачи регрессии — это задача машинного обучения, в которой выход модели y является непрерывной переменной. Признаки могут быть как непрерывными, так и категориальными. Обозначим набор входных признаков модели как $\mathbf{x} \in \mathbb{R}^n$ т. е. $\mathbf{x} = (x_1, \dots, x_n)$. Тогда модель прогнозирования, соответствующая **линейной регрессии**, запишется в виде:

$$\hat{y} = \text{net}_w(\mathbf{x}) = w_0 + w_1 x_1 + \dots + w_n x_n$$

Для оценки качества обучения модели используется функции потерь L2, которая соответствует среднему квадрату ошибки предсказания

$$\text{Loss}(\mathbf{w}) = \frac{1}{2N} \sum_{j=1}^N (y^j - \text{net}_w(\mathbf{x}^j))^2 = \frac{1}{2N} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$$

где каждая строка матрицы \mathbf{X} — это вектор \mathbf{x}^j , соответствующий j -ой точке данных; y^j — истинное значение. Дифференцируя функцию потерь по \mathbf{w} и приравнявая производные нулю, можно найти оценку оптимального вектора весов.

Логистическая регрессия

Логистическая регрессия позволяет предсказать категориальную переменную. Для этого линейная комбинация входных признаков преобразуется в вероятность с помощью логистической (сигмовидной) функции:

$$P(y | \mathbf{x}; \mathbf{w}) = 1 / (1 + e^{-\mathbf{w}^T \mathbf{x}})$$

Хотя логистическая регрессия и называется регрессией, она используется для решения задачи классификации, а не задачи регрессии. Значения логистической функции находятся в интервале от 0 до 1 и она определяет вероятность принадлежности точки данных \mathbf{x} к классу с меткой 1.

Например, если после обучения значение логистической функции будет больше 0,5, то \mathbf{x} классифицируется как точка с меткой 1, а в противном случае — с меткой 0.

Оценить веса логистической регрессии можно, используя алгоритм градиентного спуска. При этом в качестве функции потерь используют отрицательный логарифм вероятности: $-\log(P(y | \mathbf{x}; \mathbf{w}))$.

Хорошей (с низкими потерями) является модель, которая назначает высокую вероятность истинному выходу y , при соответствующем входе \mathbf{x}

Многоклассовая логистическая регрессия

Позволяет классифицировать точки данных по K различным категориям, а не только по двум. В этом случае мы строим такую модель, которая даёт оценки вероятностей принадлежности точки данных к одной из K возможных категорий. Для этого вместо логистической функции используется **softmax** функция, которая определяет вероятность отнесения точки данных с признаками \mathbf{x}_i , к классу k как:

$$P(y_i = k \mid \mathbf{W}; \mathbf{x}_i) = \frac{e^{\mathbf{w}_k \mathbf{x}_i}}{\sum_{j=1}^K e^{\mathbf{w}_j^T \mathbf{x}_i}} = \frac{e^{s_k}}{\sum_{j=1}^K e^{s_j}}$$

Аргументы экспоненты s_j , называемые в данном случае **логитами** (logits), могут быть вычислены с помощью однослойной нейронной сети, содержащей K нейронов с весами \mathbf{w}_j и вектором входа \mathbf{x}_i .

В ходе обучения такой сети необходимо найти такую матрицу \mathbf{W} , которая минимизирует функцию потерь в виде отрицательного логарифма вероятности

$$L(y_i \mid \mathbf{W}; \mathbf{x}_i) = -\log(P(y_i = i \mid \mathbf{W}; \mathbf{x}_i))$$

Можно показать, что эта функция потерь соответствует **кросс-энтропии**.

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \begin{array}{l} \text{softmax} \\ \text{функция} \end{array}$$



cat	3.2
car	5.1
frog	-1.7

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{softmax функция}$$



cat	3.2	exp →	24.5
car	5.1		164.0
frog	-1.7		0.18

ненормализован.
вероятности

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{softmax функция}$$



Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{softmax функция}$$



Softmax классификатор

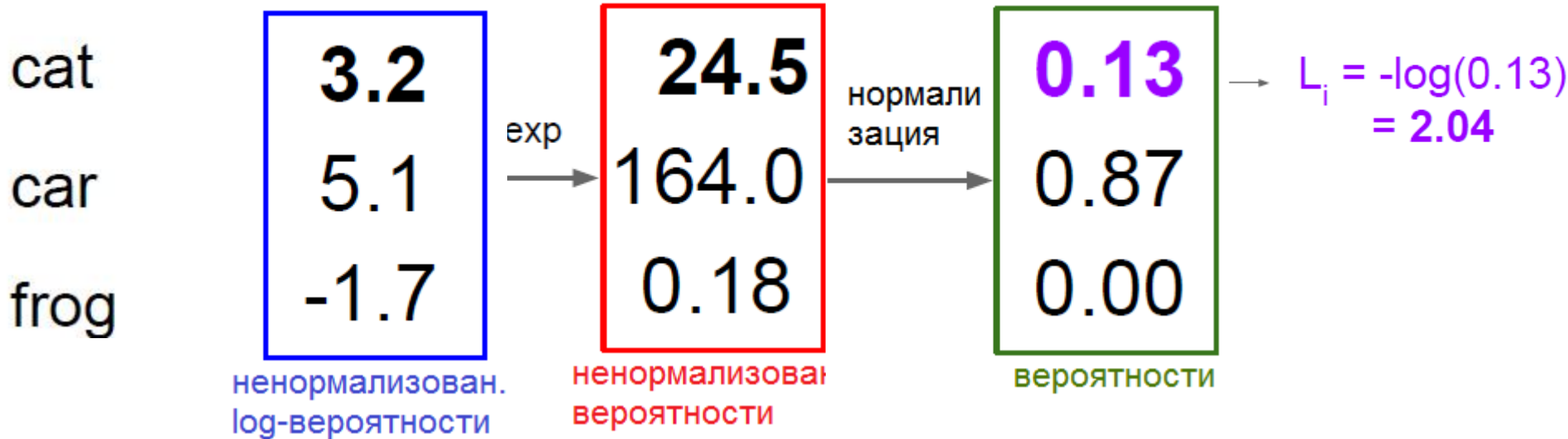
Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{softmax функция}$$



Функция потерь задается в виде логарифма правдоподобия

$$L_i = -\log P(Y = y_i | X = x_i)$$



Оценка максимума правдоподобия - найти такие W , которые максимизируют правдоподобие на обучающих данных (или минимизируют лог. правдоподобия)

В.Бондарев

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$s = f(x_i; W)$

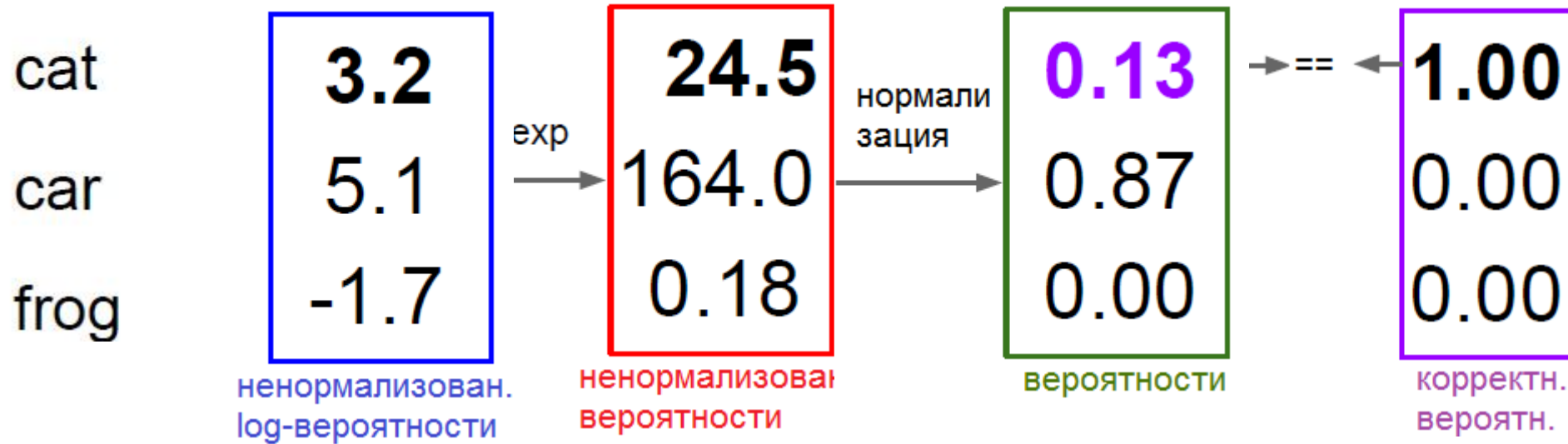
$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$

softmax
функция



Функция потерь задается в виде логарифма правдоподобия

$$L_i = -\log P(Y = y_i | X = x_i)$$



Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$s = f(x_i; W)$

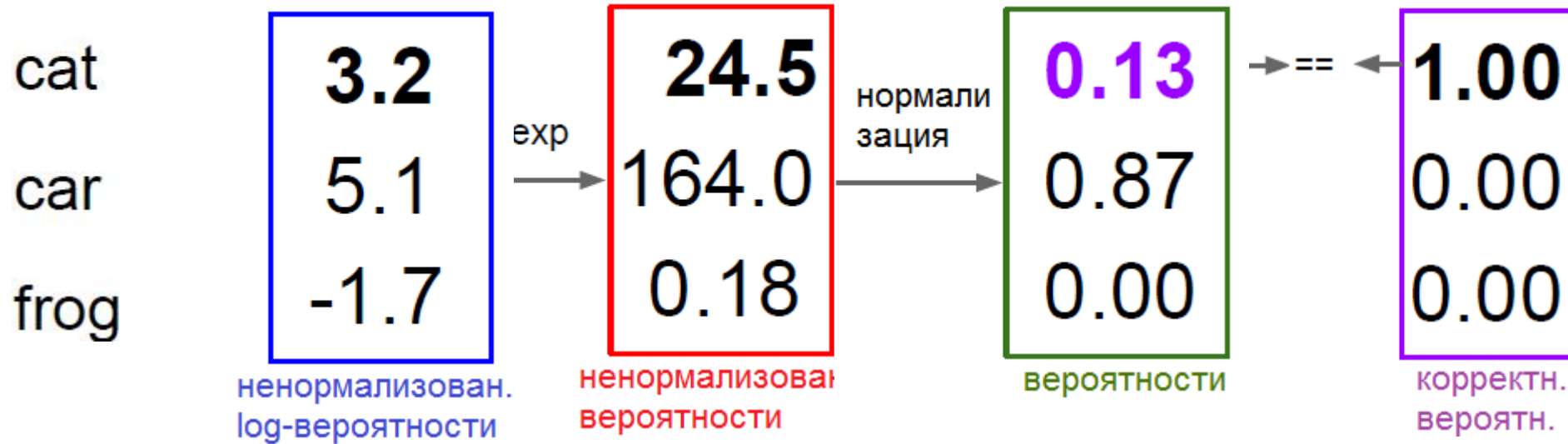
$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$

softmax
функция



Функция потерь задается в виде логарифма правдоподобия

$$L_i = -\log P(Y = y_i | X = x_i)$$



Кросс-энтропия: $H(Q, P) = -\sum_{j=1}^J Q(y_j) \log(P(y_j)) = -\log P(y_i)$

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W)$$

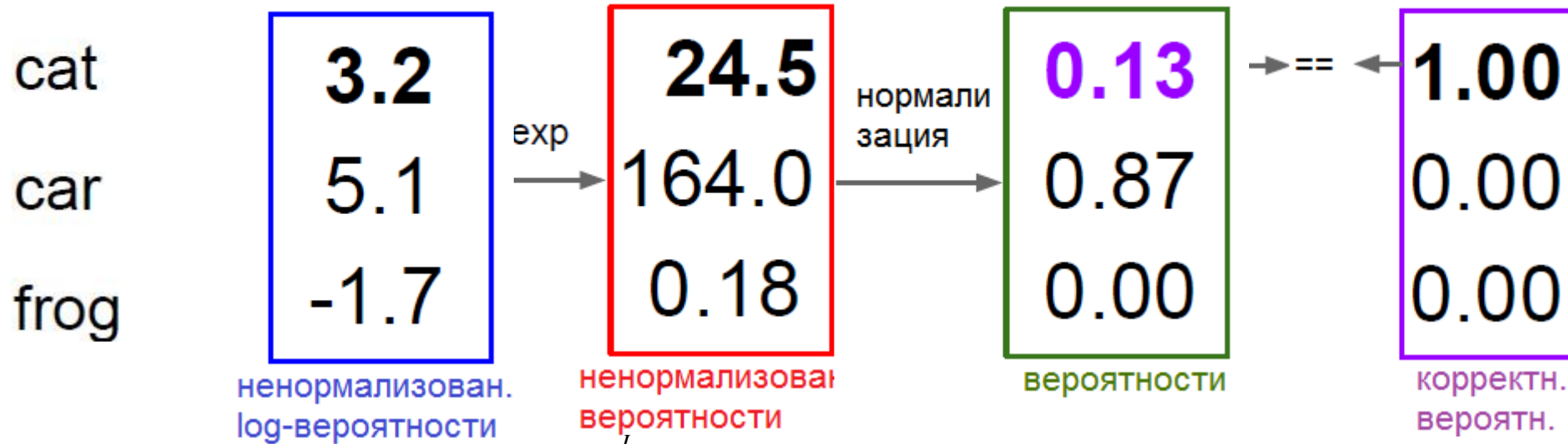
$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

softmax
функция



Функция потерь задается в виде логарифма правдоподобия

$$L_i = -\log P(Y = y_i | X = x_i)$$



Кросс-энтропия: $H(Q, P) = -\sum_{j=1}^J Q(y_j) \log(P(y_j)) = -\log P(y_i)$

Бинарная кросс-энтропия: $H(q, p) = -[q \log(p) + (1 - q) \log(1 - p)]$ В.Бондарев

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \begin{array}{l} \text{softmax} \\ \text{функция} \end{array}$$



Максимизировать вероятность корректного класса == минимизировать К-Э

$$L_i = -\log P(Y = y_i | X = x_i), \quad L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat **3.2**

car 5.1

frog -1.7

Вопрос: Какие возможные мин/макс значения для L_i ?

Ответ: $\min=0$, $\max=\infty$

Softmax классификатор

Интерпретирует значения оценок s (scores) как **вероятности**:

$$s = f(x_i; W) \quad P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \begin{array}{l} \text{softmax} \\ \text{функция} \end{array}$$



Максимизировать вероятность корректного класса == минимизировать к-э

$$L_i = -\log P(Y = y_i | X = x_i), \quad L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

cat **3.2**

car 5.1

frog -1.7

Вопрос: На этапе инициализации все s примерно равны, какие будут значения у L_i ?

Ответ: $\log(C)$, в примере $\log(10) \approx 2.3$

Итеративные алгоритмы оптимизации

Цель оптимизации заключается в поиске вектора \mathbf{x} , который минимизирует целевую функцию $F(\mathbf{x})$.

Будем использовать для этого алгоритмы последовательного приближения – **итеративные алгоритмы**. Поиск начинается с начального значения \mathbf{x}_0 и на каждом шаге

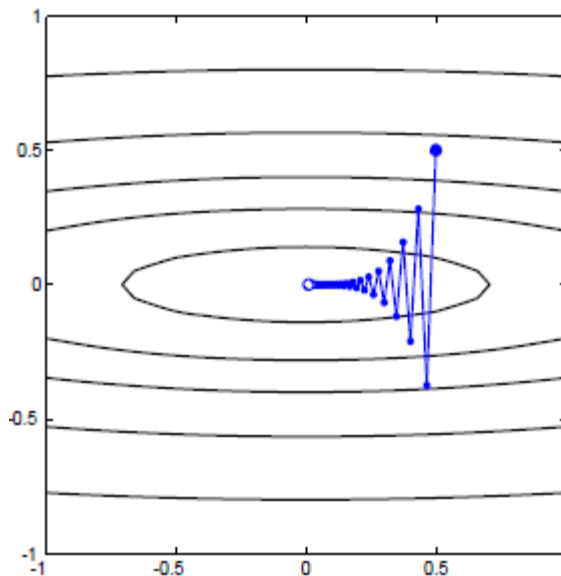
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

или

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k,$$

где \mathbf{p}_k – вектор, определяющий направление поиска; α_k – скорость обучения, определяющая длину шага.

Алгоритмы отличаются выбором вектора направления поиска \mathbf{p}_k и способами вычисления значений скорости обучения



Алгоритм наискорейшего спуска (steepest descent algorithm - SDA)

При поиске минимума должно выполняться условие : $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$. (1)

Рассмотрим ряд Тейлора для ЦФ в районе т. \mathbf{x}_{k+1}

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta \mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta \mathbf{x}_k, \quad \mathbf{g}_k \equiv \nabla F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k}. \quad (2)$$

где \mathbf{g}_k - градиент .

Чтобы выполнялось условие (1), второй член (2) должен быть отрицательным:

$$\mathbf{g}_k^T \Delta \mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0 \quad \text{или} \quad \mathbf{g}_k^T \mathbf{p}_k < 0.$$

Любой вектор \mathbf{p}_k , удовлетворяющий этому условию, соответствует направлению спуска. **Направление наискорейшего спуска :**

$$\mathbf{p}_k = - \mathbf{g}_k.$$

Соответственно процедура **SDA** определяется выражением:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k.$$

Есть два подхода определения α_k :

1) использовать фиксированное значение, например $\alpha_k = 0.01$ или переменное, но предопределенное, например $\alpha_k = 1/k$.

2) минимизировать на каждом шаге $F(\mathbf{x})$ по отношению к α_k вдоль \mathbf{p}_k

Отметим, что направление наискорейшего спуска ортогонально линиям равных контуров целевой функции.

Общий вид функции потерь

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Средние потери по всем обучающим примерам. Предсказания модели должны совпадать с обучающими метками}} + \underbrace{\lambda R(W)}_{\text{Не позволяет модели детально следовать обучающим данным. Улучшает обобщающие свойства модели.}}$$

Средние потери по всем обучающим примерам. Предсказания модели должны совпадать с обучающими метками

Не позволяет модели детально следовать обучающим данным. Улучшает обобщающие свойства модели.

L2 регуляризация:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

L1 регуляризация:

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

Эластичная сеть (L1+L2):

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Стохастический градиентный спуск и миниблоки

Алгоритм SDA:

1. Инициализировать веса \mathbf{w} случайными значениями;
2. **While** \mathbf{w} не сошлись **do**:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \text{Loss}(\mathbf{w})}{\partial \mathbf{w}}$$

Здесь α – скорость обучения, $\frac{\partial \text{Loss}(\mathbf{w})}{\partial \mathbf{w}}$ – градиент функции потерь. Здесь обновление весов на каждой итерации выполняют на вектор пропорциональный градиенту функции потерь, которая должна вычисляться при усреднении по всем обучающим примерам.

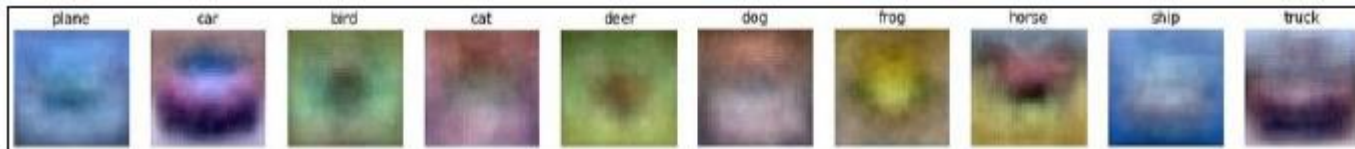
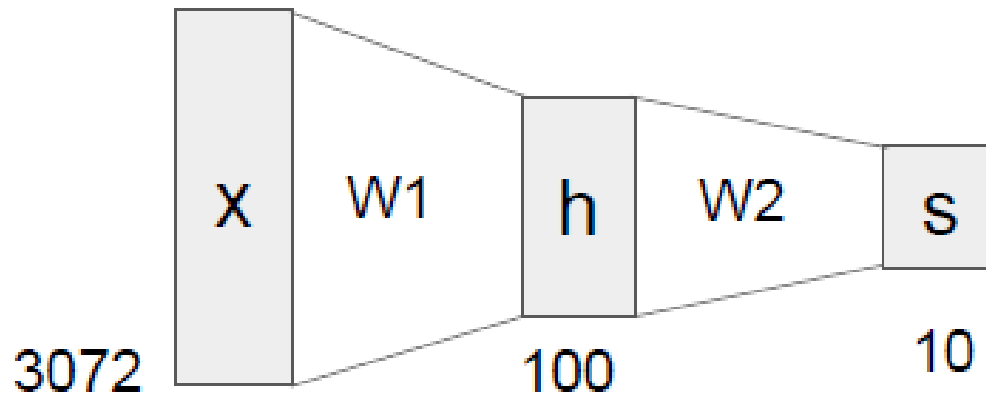
При **стохастическом градиентном спуске** на каждой итерации алгоритма используется только одна случайная точка данных для вычисления градиента. Поэтому стохастический градиентный спуск может привести к зашумленным градиентам и, таким образом, затруднить сходимость.

Мини-блочный градиентный спуск является компромиссом между стохастическим и SDA алгоритмом, поскольку он использует **мини-блок** (batch) размером m точек данных каждый раз для вычисления градиентов. Размер мини-блока m является гиперпараметром, выбираемым пользователем.

Полносвязные нейронные сети

Линейная сеть: $f=W_1x$

Двухслойная нейронная сеть: $f=W_2 \max(0, W_1 x)$



Полносвязные нейронные сети

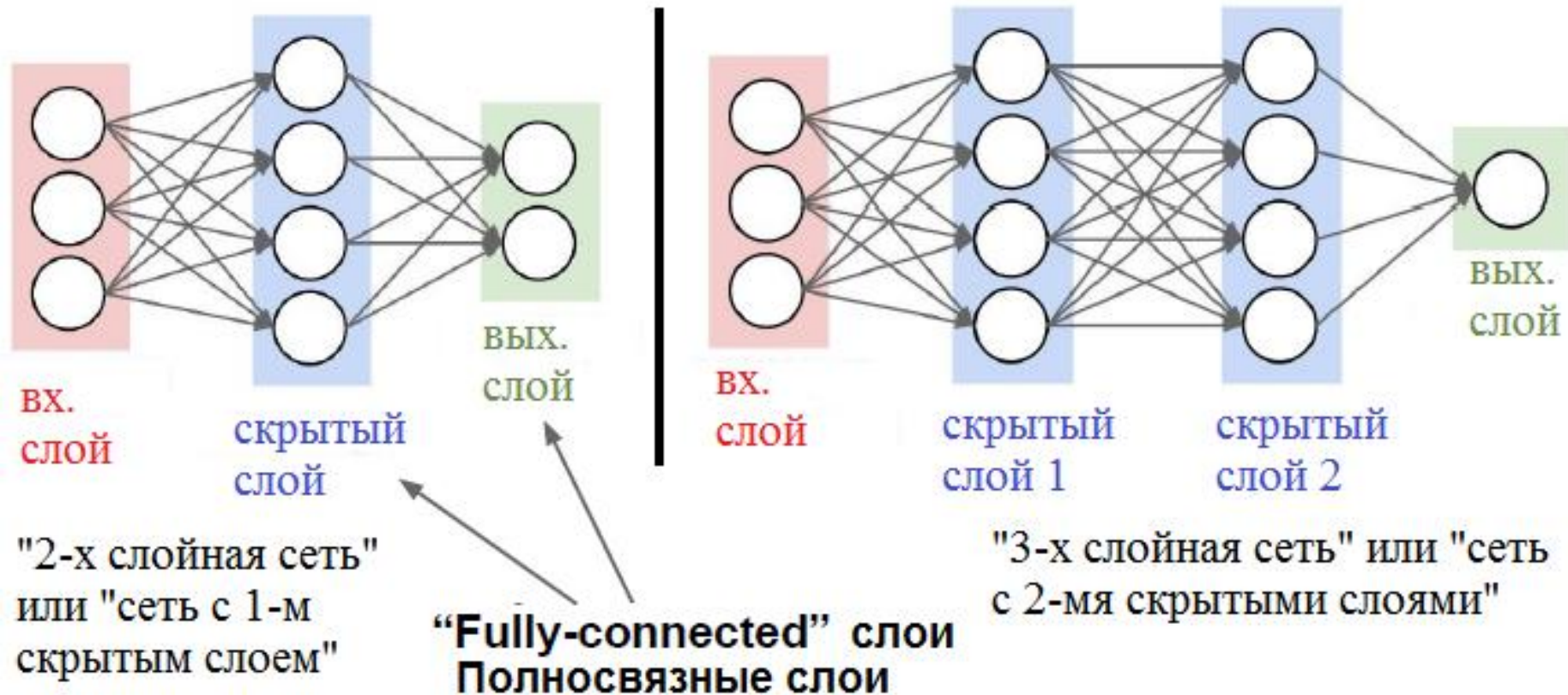
Ранее: Линейная функция рейтинга: $f=Wx$

Двухслойная нейронная сеть: $f=W_2 \max(0, W_1 x)$

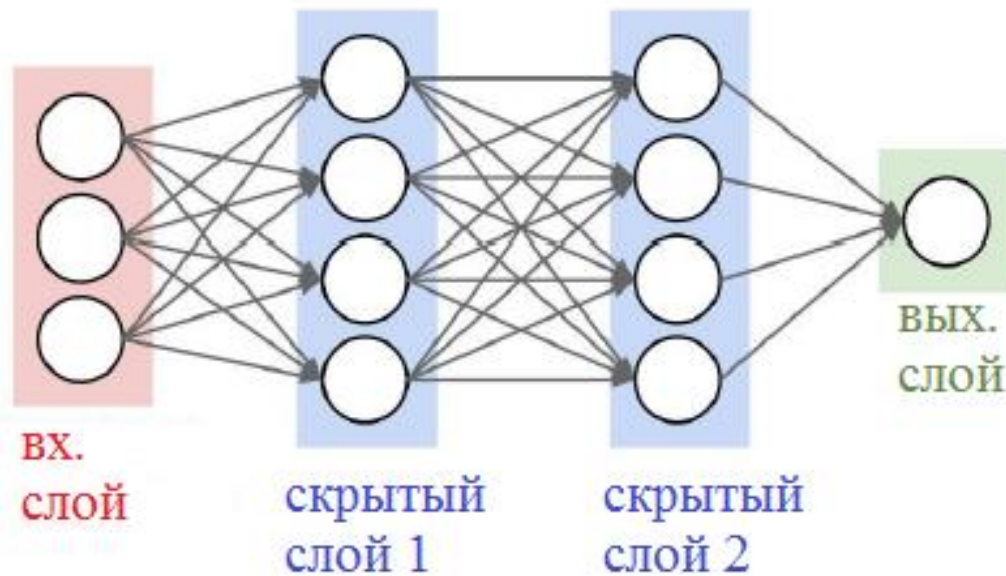
или

трехслойная сеть: : $f=W_3 \max(0, W_2 \max(0, W_1 x))$

Архитектуры нейросетей



Пример прямых вычислений в нейросети



```
# прямой путь вычислений в 3-х слойной сети
f=lambda x:1.0/(1.0+np.exp(-x)) #функция активации - сигмоида
x=np.random.randn(3,1) #случайный входной вектор (3x1)
h1=f(np.dot(W1,x)+b1) #выход активаций 1-го скрытого слоя (4x1)
h2=f(np.dot(W2,h1)+b2) #выход активаций 2-го скрытого слоя (4x1)
out=np.dot(W3,h2)+b3 #выходной нейрон (1x1)
```