Искусственные нейронные сети для обработки текстов

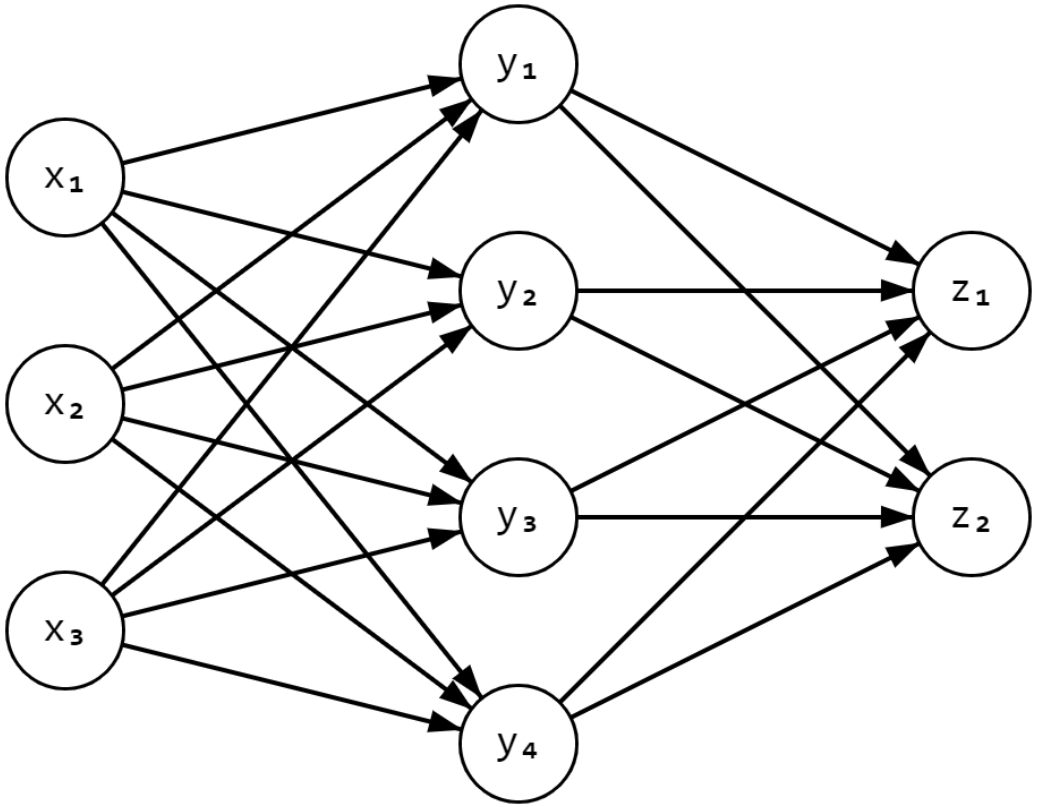
Лекция 3

Перминов Андрей Игоревич

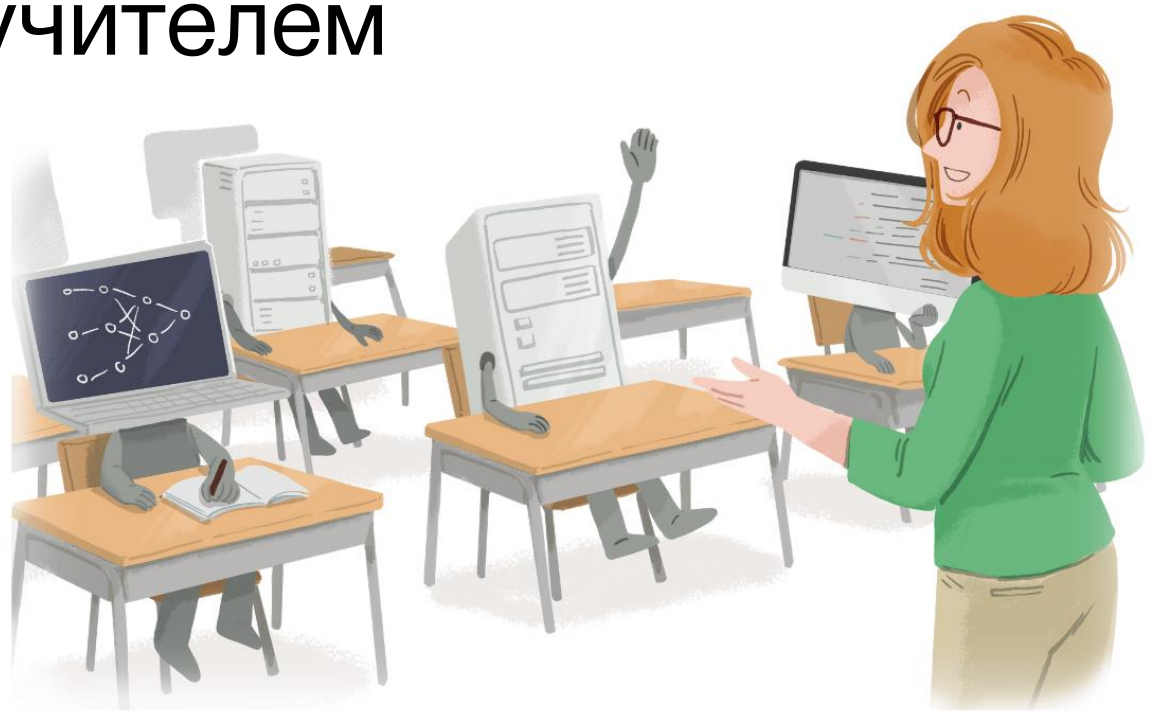
27 сентября 2023

Введение

Нейронные сети – алгоритмы машинного обучения, активно используемые для решения задач автоматической обработки различных данных (текстов, изображений , аудио, видео, …)



Обучение с учителем



Пусть:

X — множество объектов

Y — множество ответов

$\check{y}: X \to Y$ — неизвестная зависимость

Дано:

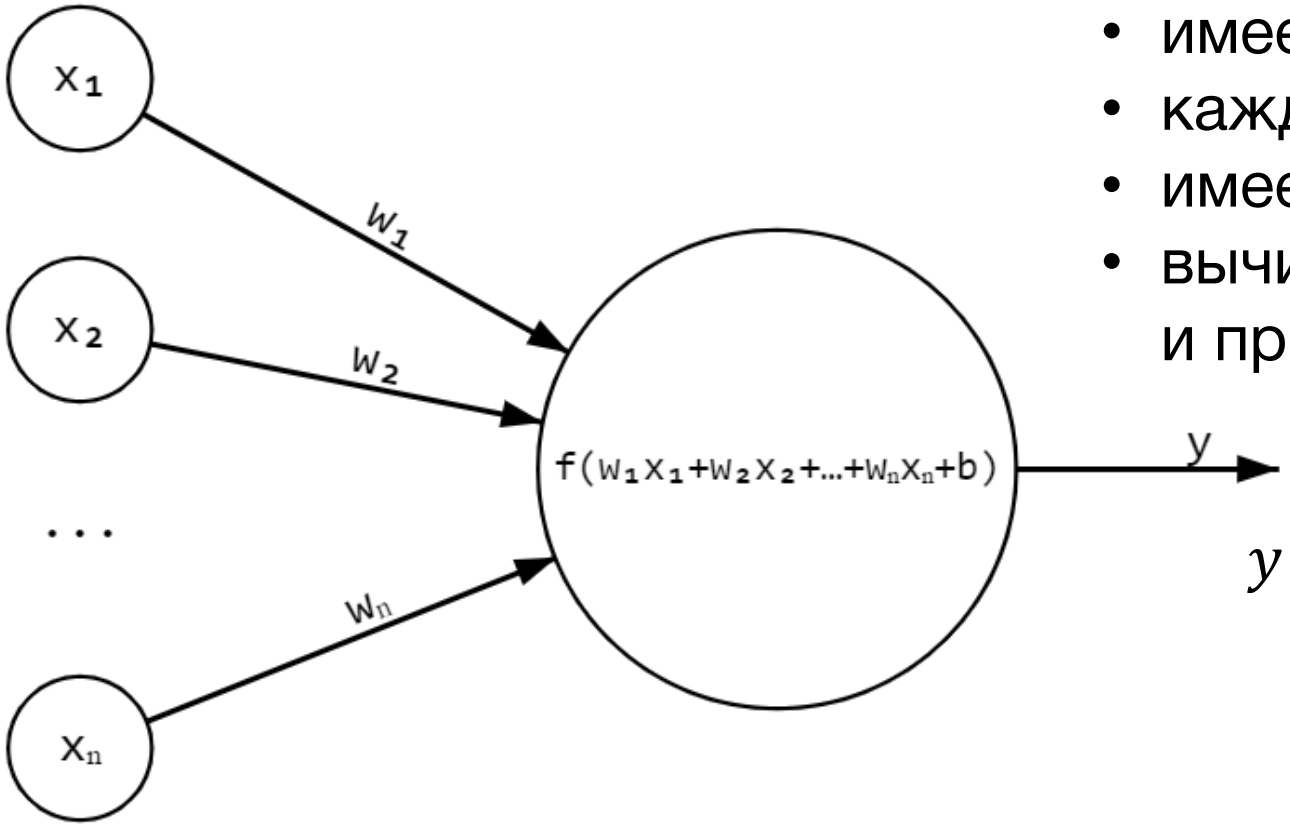
$\{x_1, \dots, x_n\} \subset X$ — обучающая выборка

$\check{y}_i = \check{y}(x_i) \in Y$ — ответы на обучающей выборке

Найти:

$y: X \to Y$ — решающую функцию, приближающую $\check{y}$ на всем множестве X

Простейшая нейронная сеть – искусственный нейрон



- имеет n входов $(x_1, \dots, x_n)$ и один выход (y)
- каждый вход имеет вес $(w_1, \dots, w_n)$
- имеется дополнительный вес смещения (b)
- вычисляет линейную комбинацию входов и применяет к ней функцию активации:

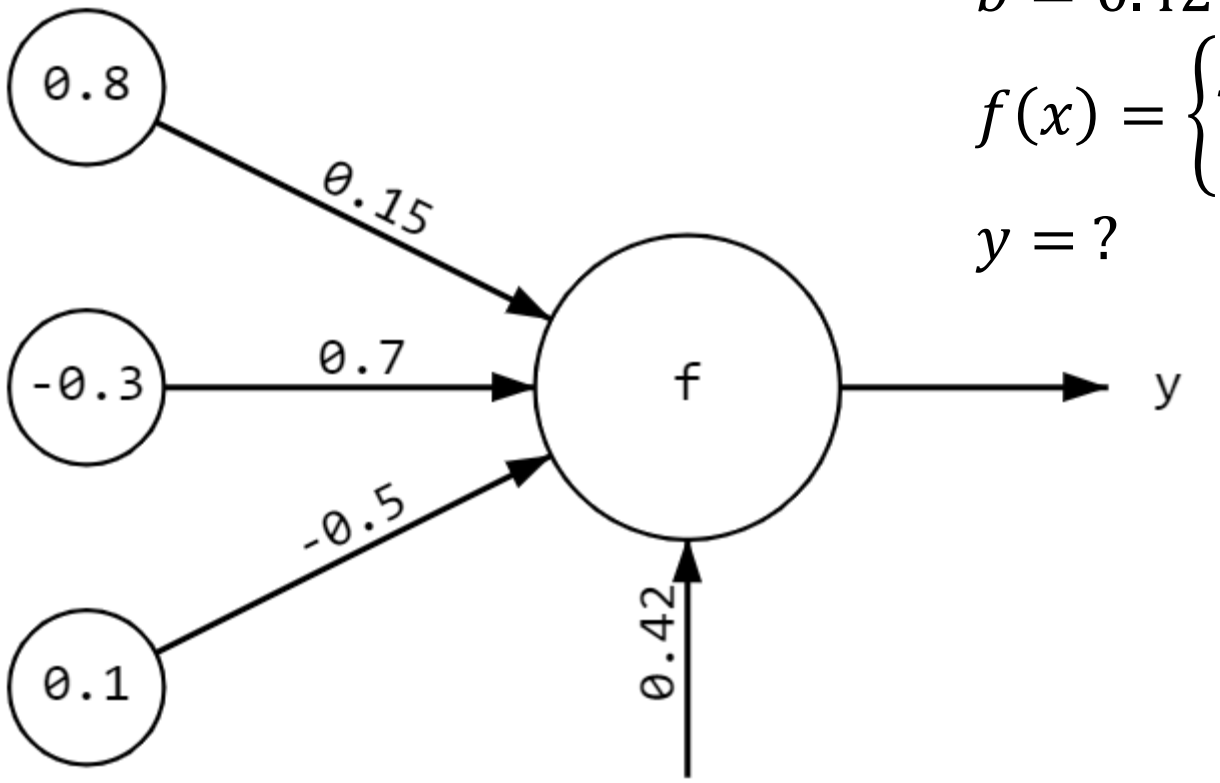
$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right) = f(w^T \cdot x + b)$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = ?$

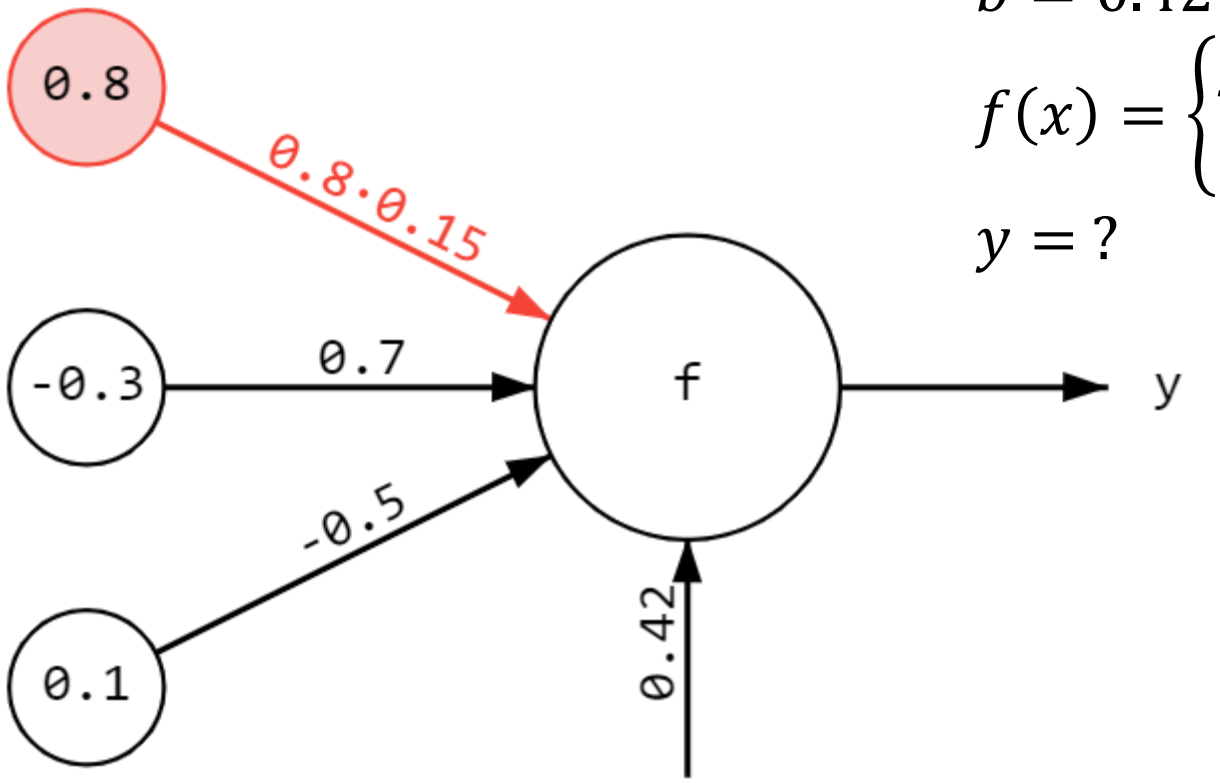


Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = ?$



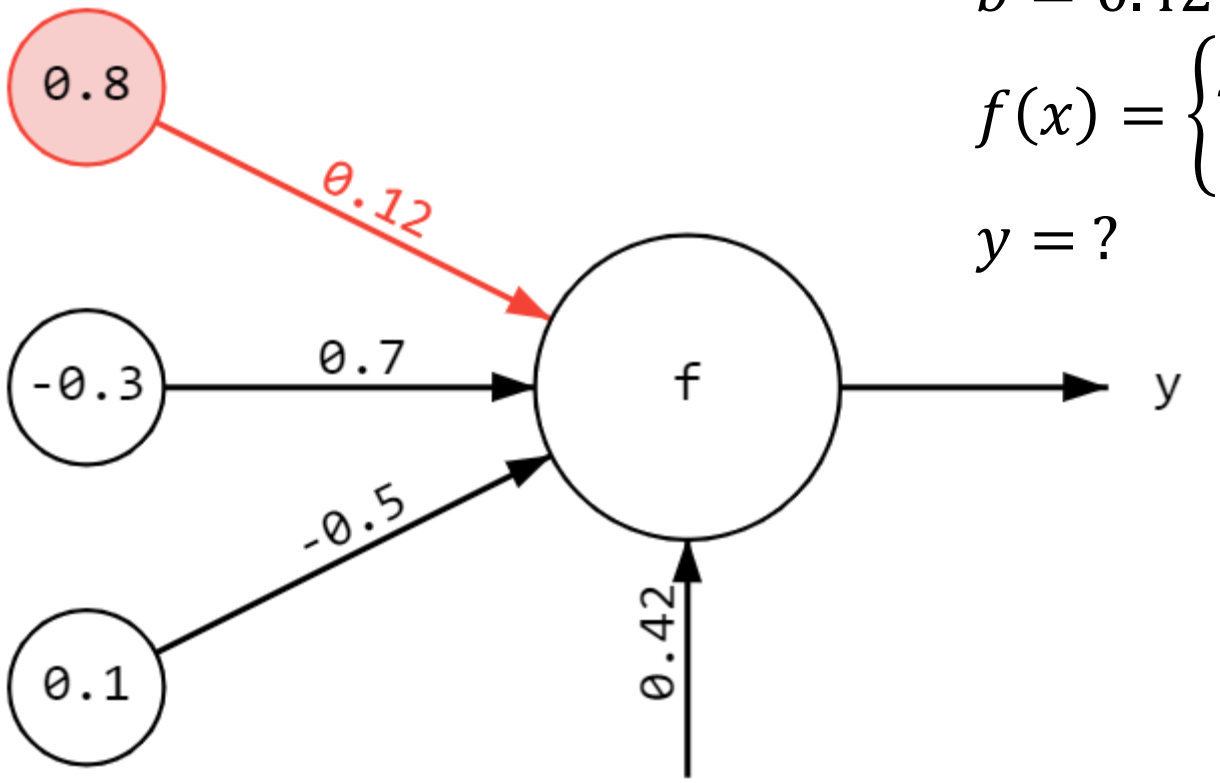
$$y = f(0.8 \cdot 0.15 +$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

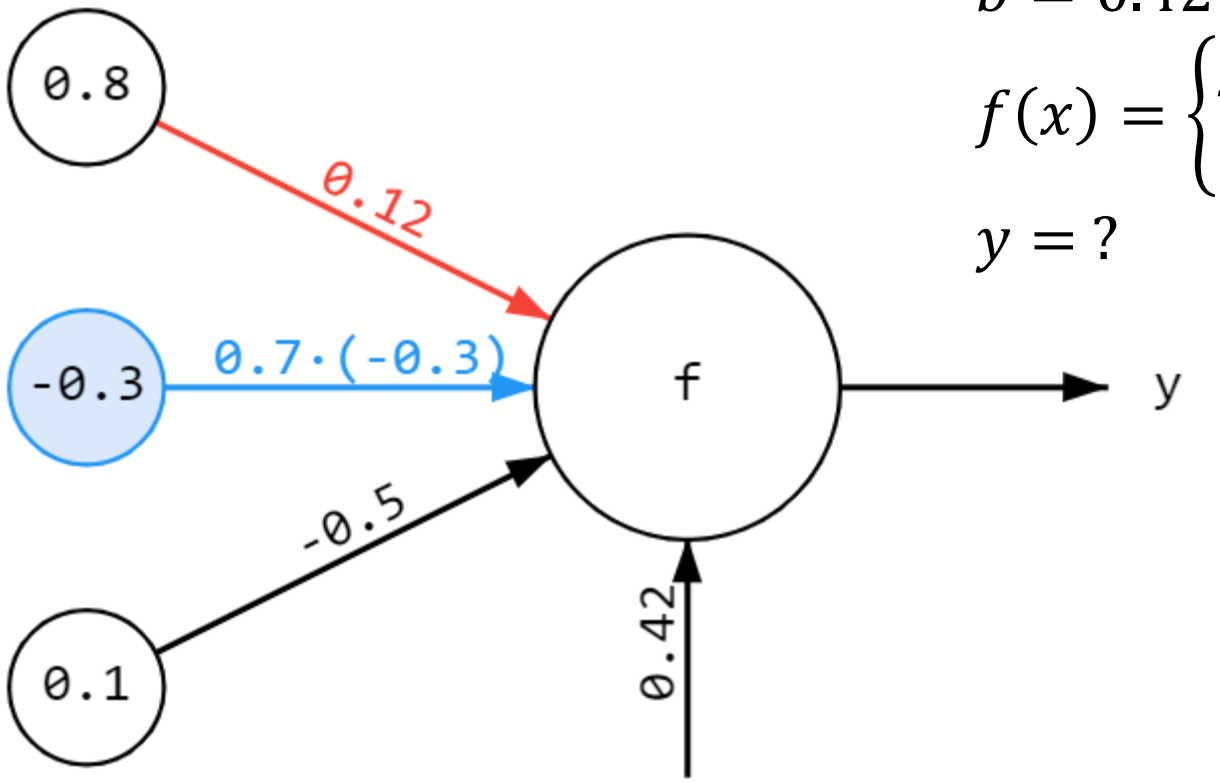
$y = ?$



$$y = f(0.12 +$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$
$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$
$y = ?$



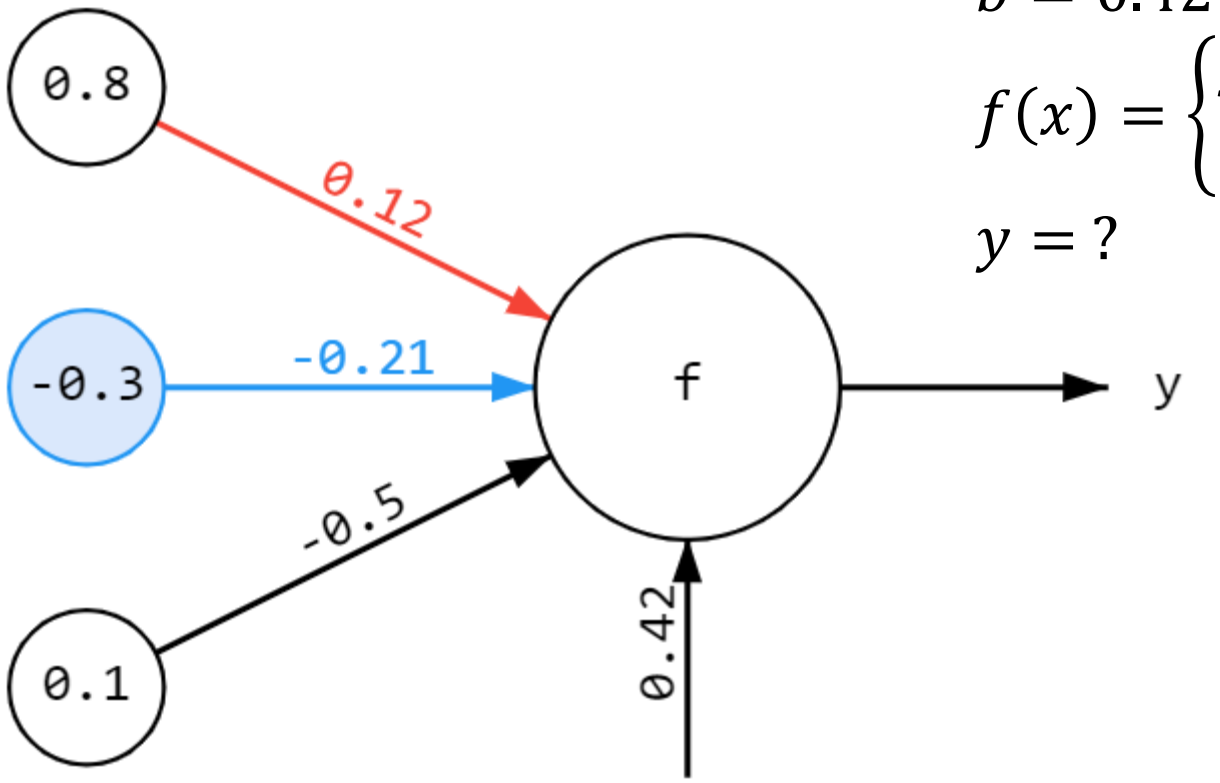
$y = f(0.12 + 0.7 \cdot (-0.3) +$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = ?$



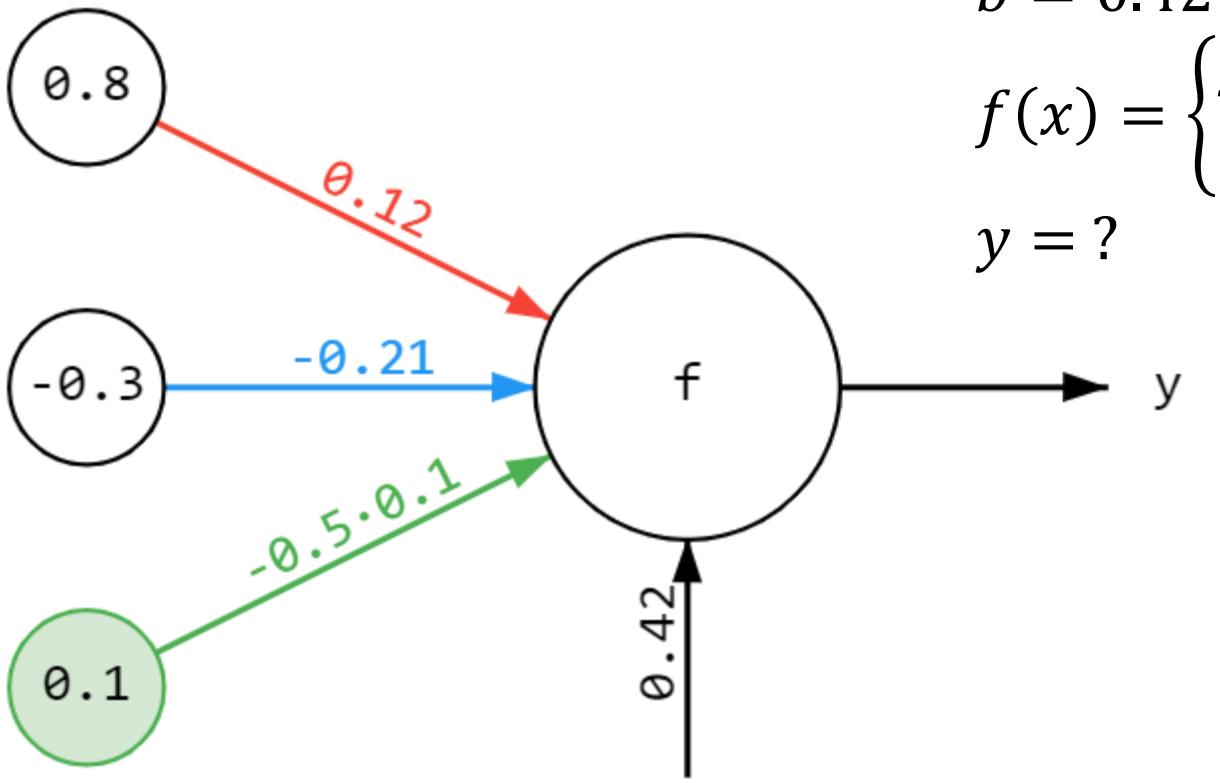
$$y = f(0.12 - 0.21 +$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = ?$



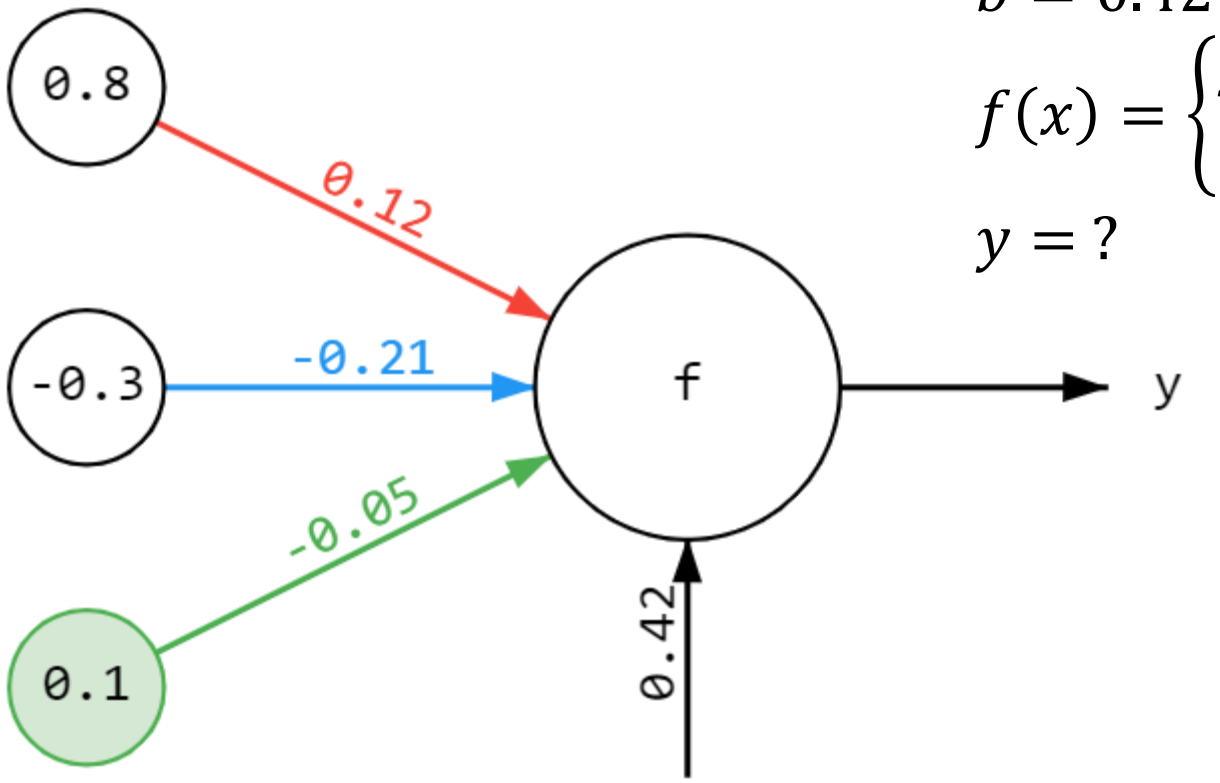
$$y = f(0.12 - 0.21 + (-0.5) \cdot 0.1 +$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = ?$



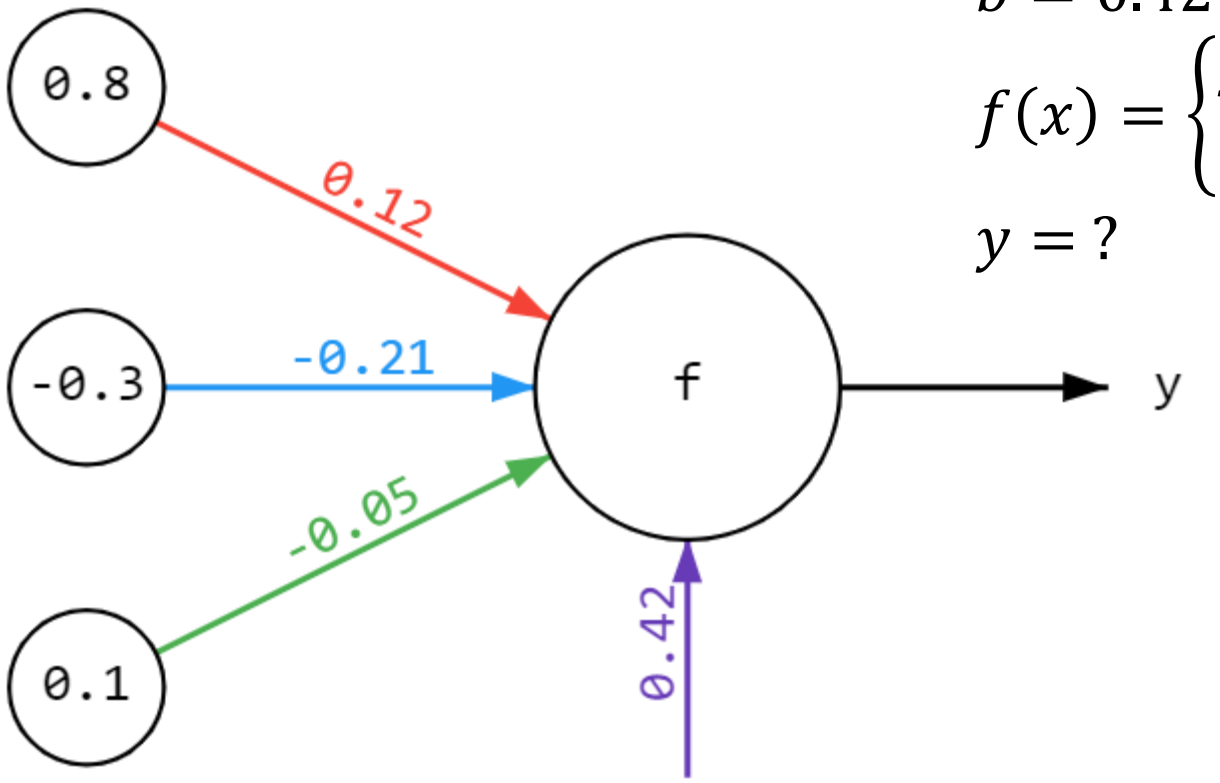
$$y = f(0.12 - 0.21 - 0.05 +$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = ?$



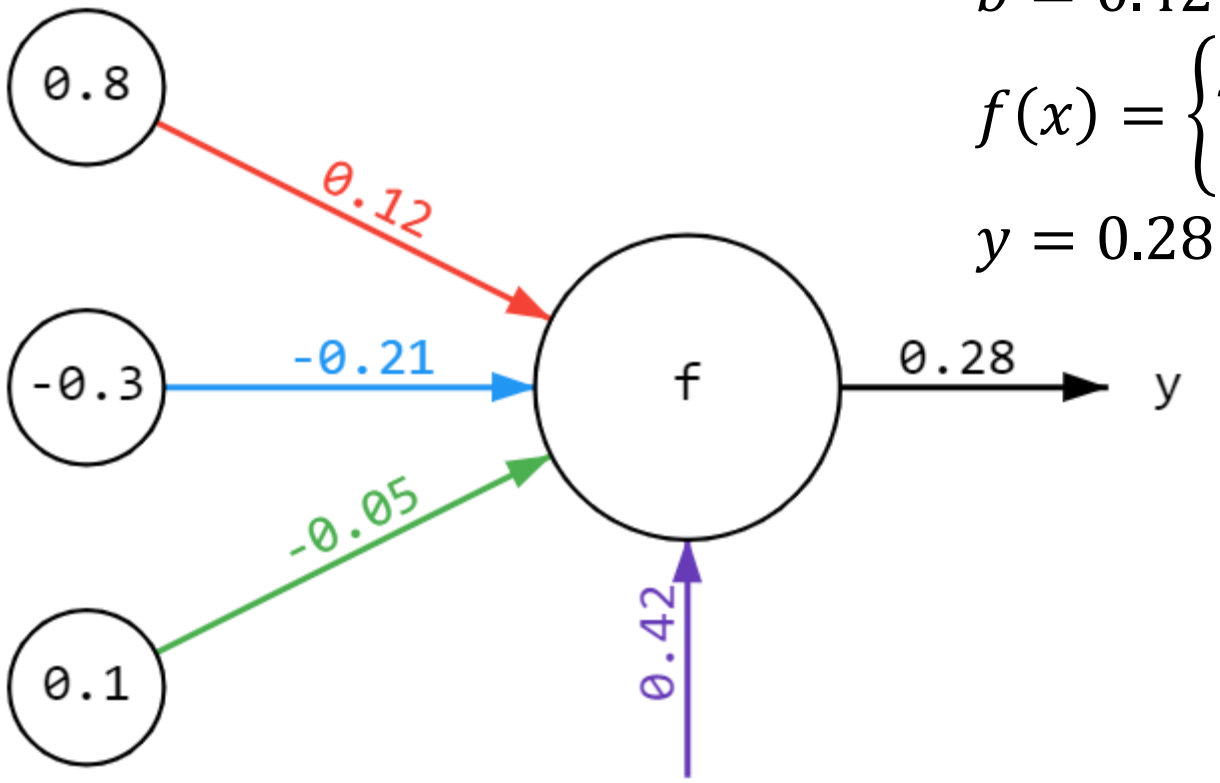
$$y = f(0.12 - 0.21 - 0.05 + 0.42)$$

Искусственный нейрон. Пример

$x_1 = 0.8, x_2 = -0.3, x_3 = 0.1$
$w_1 = 0.15, w_2 = 0.7, w_3 = -0.5$
$b = 0.42$

$$f(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$y = 0.28$



$$y = f(0.28)$$

Функции активации

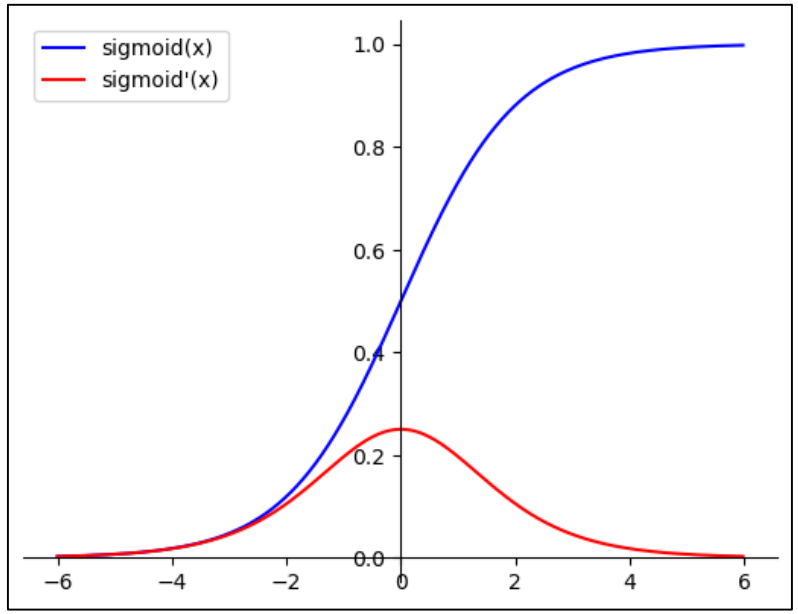
Функция активации – это некоторая дифференцируемая функция (обычно нелинейная), применяемая к линейной комбинации входных значений.

Наиболее распространённые функции активации:

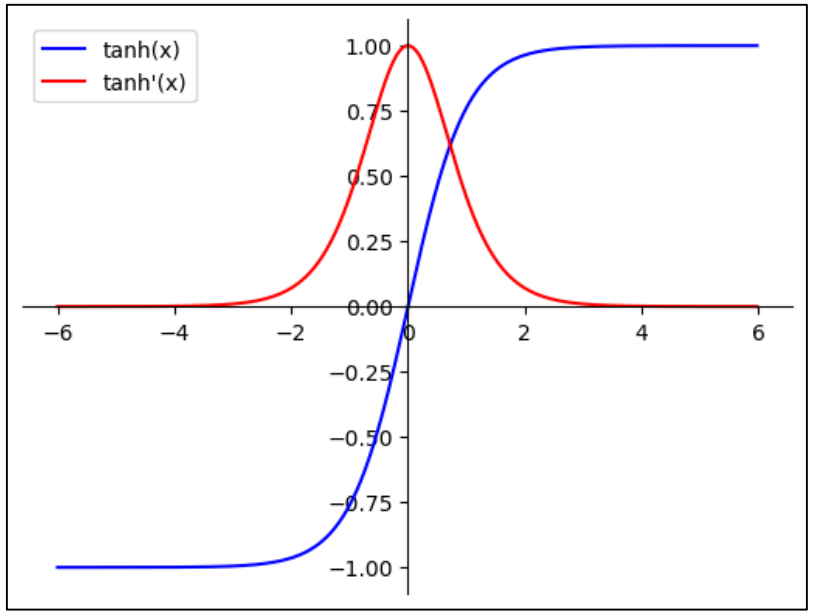
Название	Функция	Производная	Область значений
Сигмоидальная (sigmoid)	$\frac{1}{1+e^{-x}}$	$f(x)\big(1-f(x)\big)$	$(0,1)$
Гиперболический тангенс (tanh)	$\tanh(x)$	$1-f^2(x)$	$(-1,1)$
Выпрямитель (ReLU)	$f(x)=\begin{cases}x, x>0\\0, x\le 0\end{cases}$	$f(x)=\begin{cases}1, x>0\\0, x\le 0\end{cases}$	$[0,+\infty)$

Вопрос: почему функции активации обычно нелинейные? Чем плоха линейная функция?

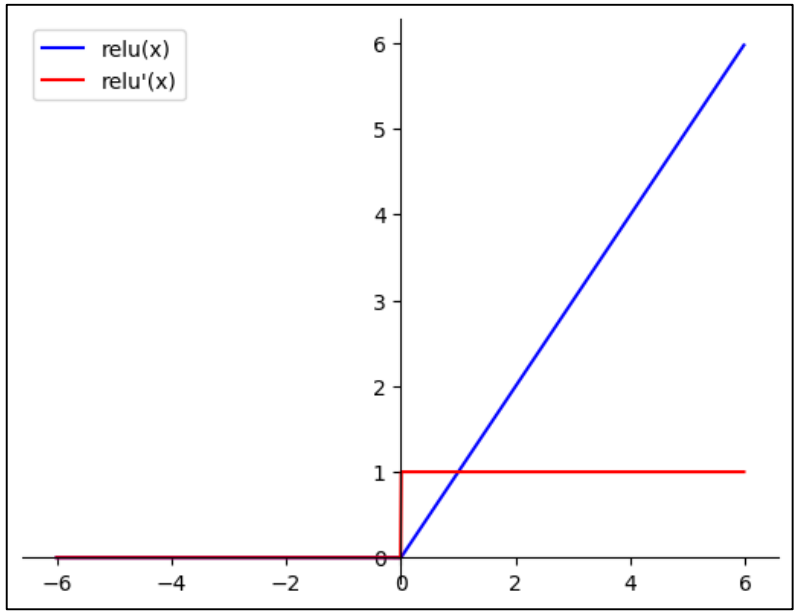
Функции активации



$sigmoid$



$tanh$



$ReLU$

Логистическая регрессия

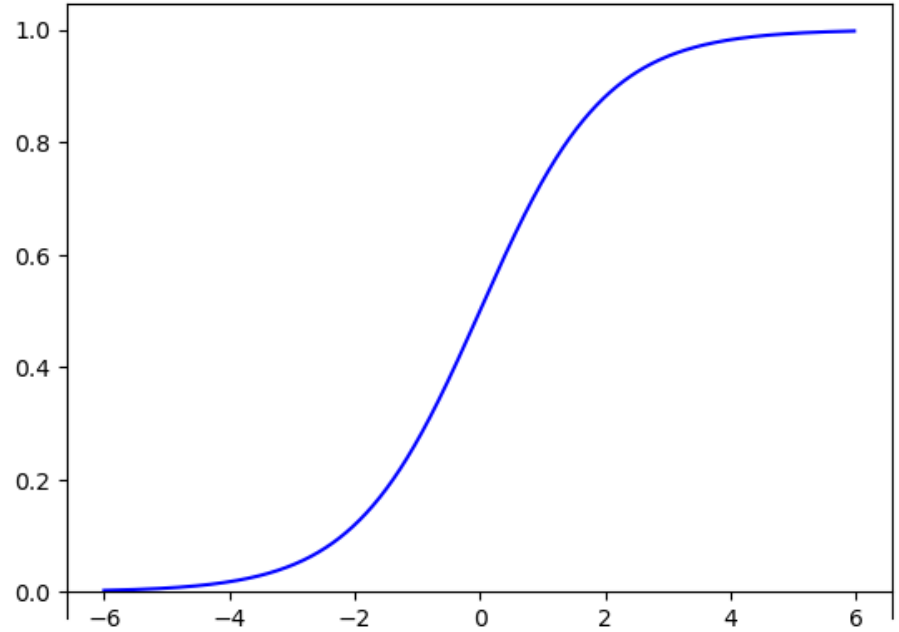
Дано:
$\{x_1, \dots, x_n\} \subset \mathbb{R}^d$ – обучающая выборка
$\tilde{y}_i = \tilde{y}(x_i) \in \{0,1\}$ – ответы на обучающей выборке

Решение:

$$y = \sigma(w^T \cdot x) = \frac{1}{1 + e^{-w^T \cdot x}} \in (0,1)$$

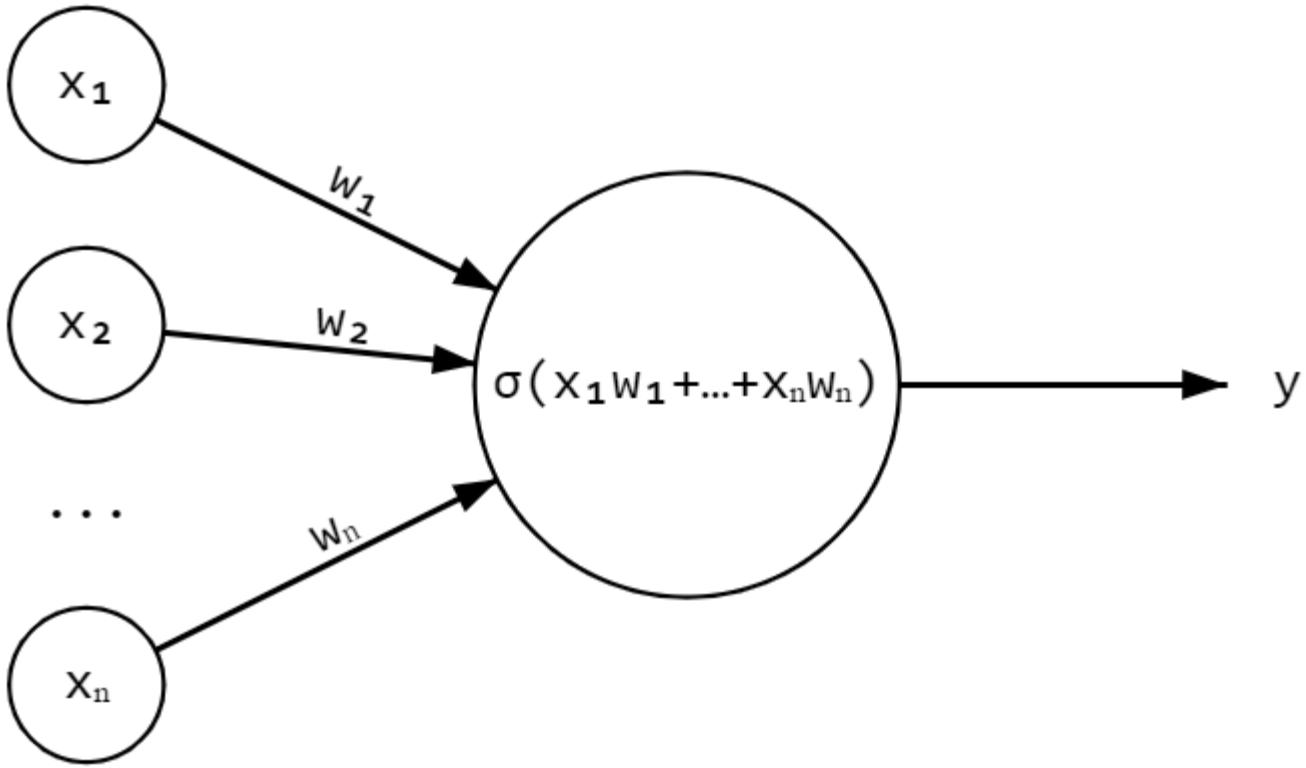
$$w = argmin_w \, L(y, \tilde{y})$$

$$L(y, \tilde{y}) = -\frac{1}{n} \sum_{i=1}^{n} \tilde{y}_i \log y(x_i) + (1 - \tilde{y}_i) \log\big(1 - y(x_i)\big)$$



Логистическая функция (сигмоида)

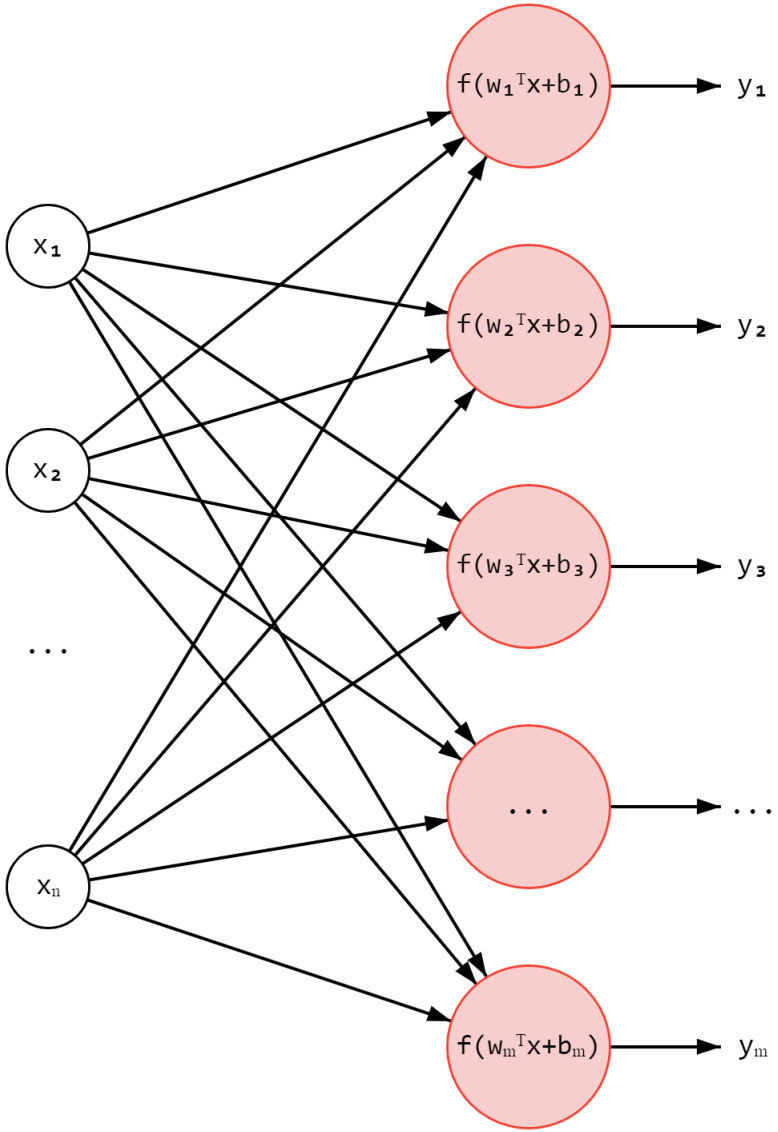
Логистическая регрессия – это нейрон



$$y = \sigma(w^T \cdot x) = \frac{1}{1 + e^{-w^T \cdot x}}$$

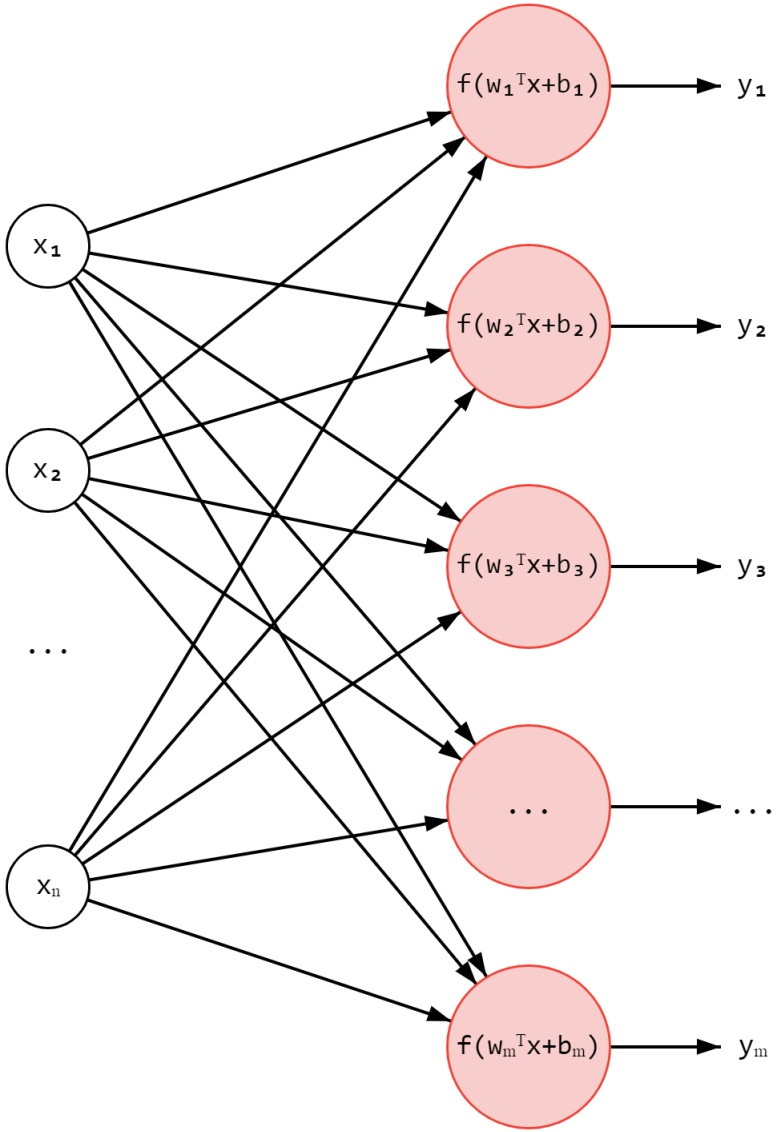
Оптимизируя функцию потерь методом градиентного спуска, находим параметры $w_1, \dots w_n$.

Полносвязный слой. Персептрон



- состоит из m независимых нейронов
- функция активации у каждого нейрона обычно общая для всего слоя
- соответствующие входы нейронов соединены между собой – слой содержит n входов
- имеет $(n+1) \cdot m$ весовых коэффициентов
- выходом слоя является вектор, а не число, в отличие от одного нейрона

Полносвязный слой. Персептрон



Удобно представлять слой в виде матриц весов:

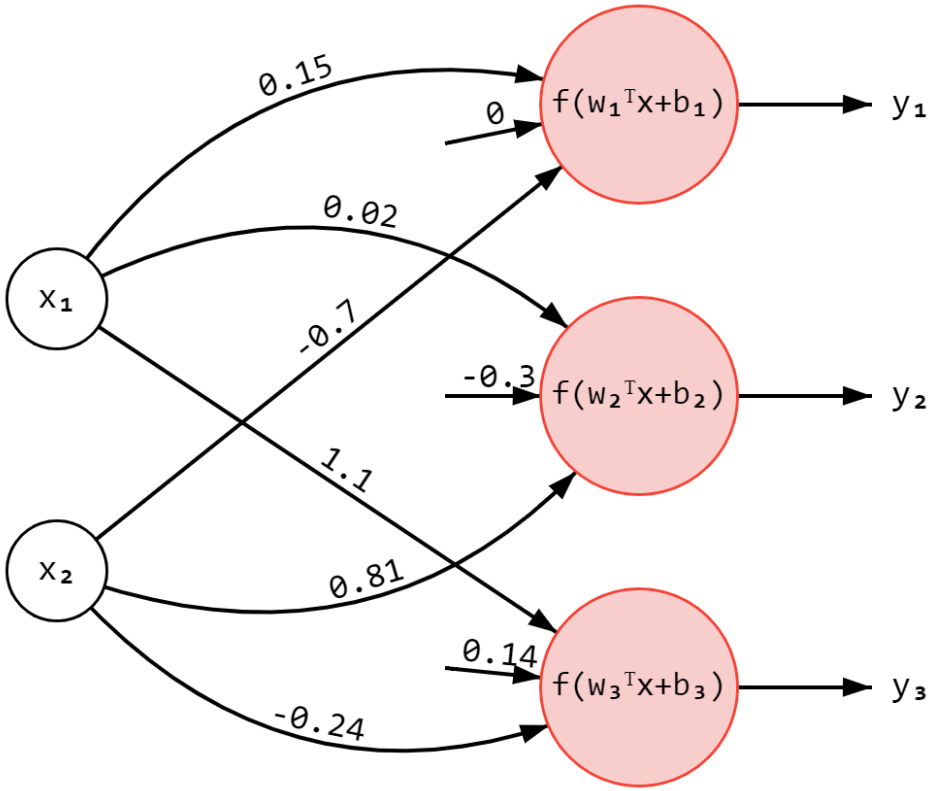
$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \dots & \dots & \dots & \dots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{bmatrix}$$

Вход слоя – такой же вектор, как для нейрона

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

Выходное значение слоя вычисляется аналогично нейрону: $y = f(Wx + b)$

Полносвязный слой. Пример



Пусть слой имеет 2 входа и 3 выхода, функция активации – tanh, а весовые коэффициенты и входной сигнал такие:

$$W = \begin{bmatrix} 0.15 & -0.7 \\ 0.02 & 0.81 \\ 1.1 & -0.24 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ -0.3 \\ 0.14 \end{bmatrix} \quad x = \begin{bmatrix} 0.5 \\ -0.3 \end{bmatrix}$$

$$Wx + b = \begin{bmatrix} 0.15 & -0.7 \\ 0.02 & 0.81 \\ 1.1 & -0.24 \end{bmatrix} \cdot \begin{bmatrix} 0.5 \\ -0.3 \end{bmatrix} + \begin{bmatrix} 0 \\ -0.3 \\ 0.14 \end{bmatrix} = \begin{bmatrix} 0.285 \\ -0.533 \\ 0.762 \end{bmatrix}$$

$$y = f(Wx + b) = \begin{bmatrix} \tanh(0.285) \\ \tanh(-0.533) \\ \tanh(0.762) \end{bmatrix} \approx \begin{bmatrix} 0.277 \\ -0.487 \\ 0.642 \end{bmatrix}$$

Softmax функция активации

Функция $softmax(z)$ преобразует вектор z размерности n в вектор той же размерности, где каждая координата полученного вектора представлена вещественным числом в интервале [0,1] и сумма координат равна 1.

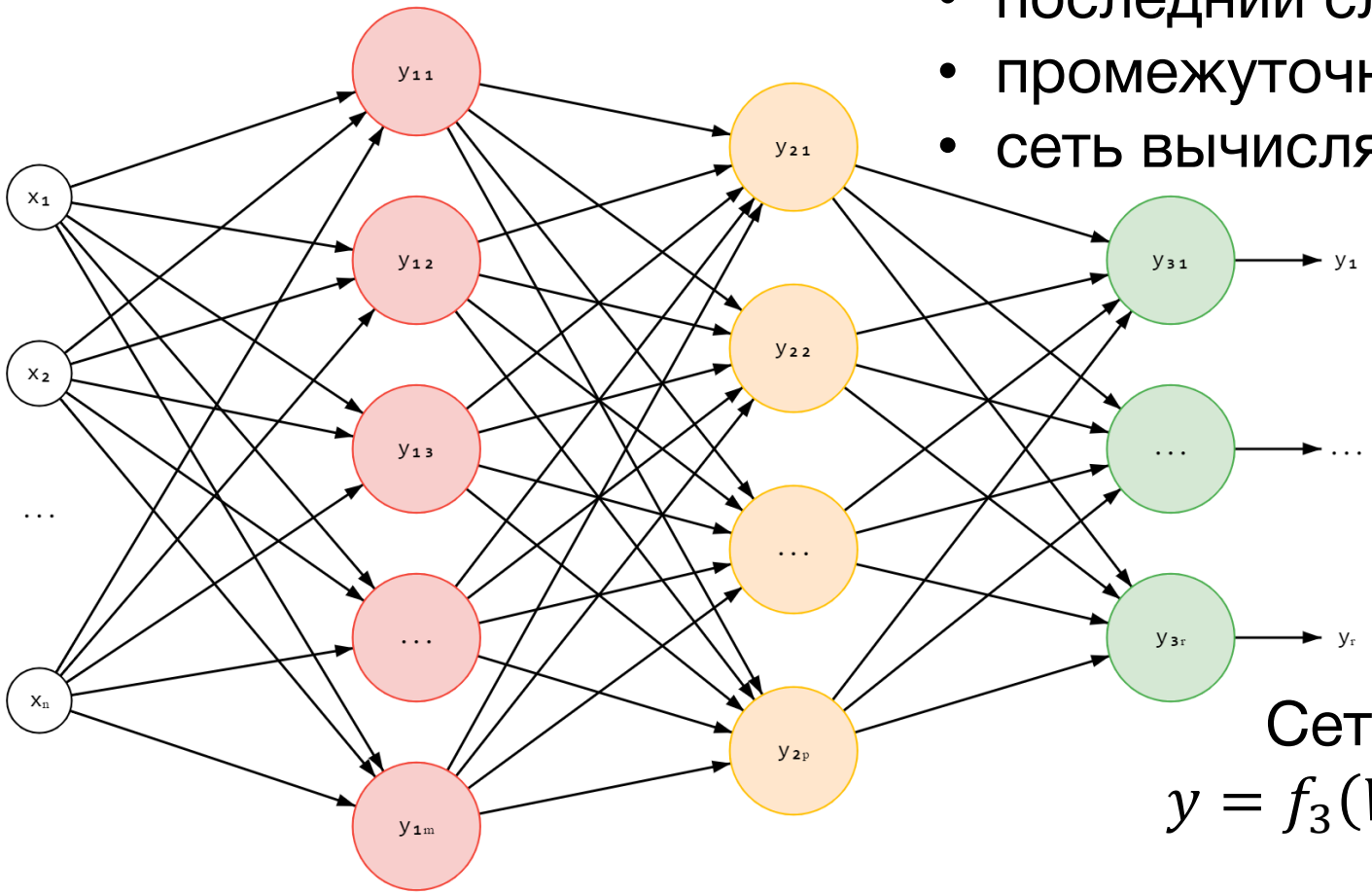
$$softmax(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

Используется для классификации на 2 и более классов.

Пример:
Пусть $z = [1, 0, 0.5, -1, 1]$
Тогда $softmax(z) \approx [0.322, 0.118, 0.195, 0.043, 0.322]$

Многослойная нейронная сеть

- каждый слой – полносвязный
- выход слоя является входом следующего слоя
- последний слой называют **выходным**
- промежуточные слои называют **скрытыми**
- сеть вычисляет выход следующим образом:



Слой 1: $y_1 = f_1(W_1 x + b_1)$

Слой 2: $y_2 = f_2(W_2 y_1 + b_2)$

...

Слой i: $y_i = f_i(W_i y_{i-1} + b_i)$

Выходной слой: $y = f_n(W_n y_{n-1} + b_n)$

Сеть моделирует следующую функцию:

$$y = f_3(W_3 \cdot f_2(W_2 \cdot f_1(W_1 x + b_1) + b_2) + b_3) \ldots$$

Обучение нейронной сети с учителем

$(x_i, y_i), i \in 1..n$ – обучающая выборка, где x_i – объекты, y_i – правильные ответы
$f(x, \theta)$ – дифференцируемая нейронная сеть с параметрами θ

Процесс обучения заключается в поиске таких параметров θ, при которых нейронная сеть будет выдавать «близкие» к правильным ответам на элементах обучающей выборки.

Для оценки «близости» ответов используются различные функции ошибки (потерь) $L(y,\ t)$, где y – выход сети, а t – ожидаемое значение, например:

$$MSE: (y - t)^2$$
$$MAE: |y - t|$$
$$BCE: -t \ln y - (1 - t) \cdot \ln(1 - y)$$

Обучение нейронной сети с учителем

Чтобы обучить сеть, применяем её ко всем элементам входной выборки и сравниваем получаемые ответы с правильными и строим функционал качества:

$$L = L(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i, \theta), y_i) = \frac{1}{n}\sum_{i=1}^{n} l_i$$

Так как $L(\theta)$ - дифференцируемая функция, то, оптимизируя её параметры с помощью градиентного спуска, можно попытаться попасть в локальный минимум:

$$\theta^{\mathrm{i}+1} = \theta^{\mathrm{i}} - \eta \cdot \nabla L\left(\theta^{\mathrm{i}}\right)$$

Самая сложная часть – вычисление градиента $\nabla L(\theta^{\mathrm{i}})$

Стохастический градиентный спуск (SGD)

Вычисление L при больших n – трудоёмкая задача.

Решение – стохастический градиентный спуск:

- каждую итерацию выбирается случайный элемент из обучающей выборки (x_j)
- рассчитывается функция потерь $l_j = l\left(f\left(x_j, \theta^i\right), y_j\right)$
- обновляются параметры: $\theta^{\mathrm{i}+1} = \theta^{\mathrm{i}} - \eta \cdot \nabla l_j\left(\theta^{\mathrm{i}}\right)$

Недостатки:

- подвержен выбросам, из-за чего необходимо использовать малое значение η, что приводит к большому числу шагов для сходимости
- невозможно распараллелить вычисление ошибки по элементам обучающей выборки

Пакетный градиентный спуск (minibatch SGD)

- каждую итерацию произвольно выбирается m $(1 < m \ll n)$ примеров из обучающей выборки
- вычисляется суммарная функция потерь на выбранных примерах:

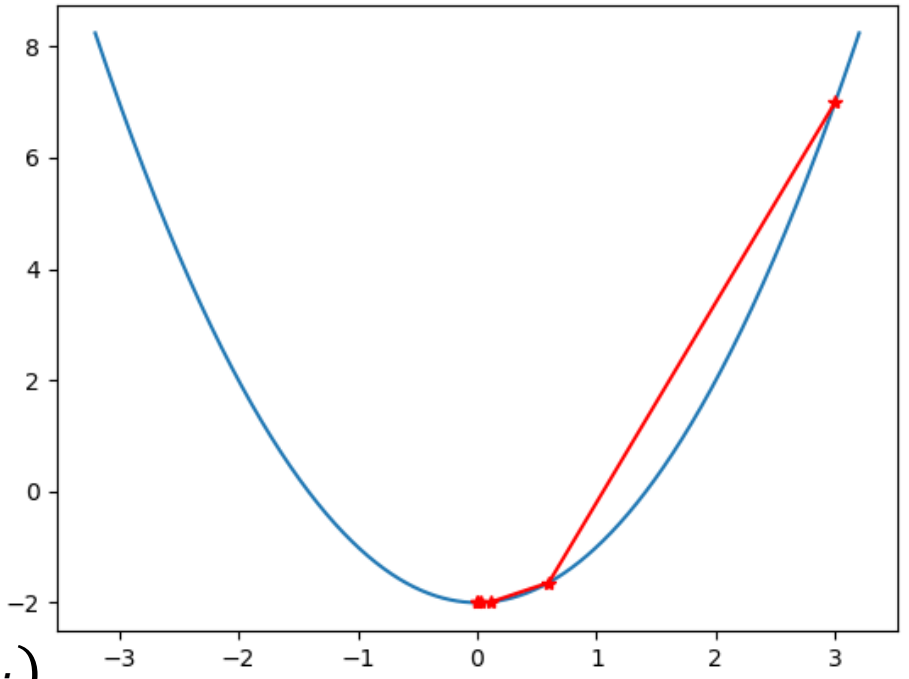
$$l_{batch} = \frac{1}{m}\sum_{j=1}^{m} l\left(f\left(x_j, \theta^i\right), y_j\right)$$

- обновляются параметры:

$$\theta^{\mathrm{i}+1} = \theta^{\mathrm{i}} - \eta \cdot \nabla\left[\frac{1}{m}\sum_{j=1}^{m} l\left(f\left(x_j, \theta^i\right), y_j\right)\right]$$

Более устойчив к выбросам, так что можно выбирать больший шаг η
Ошибку l_{batch} можно считать параллельно

Градиентный спуск. Пример



Найти минимум функции $f(x) = x^2 - 2$ с помощью градиентного спуска.
В качестве начальной точки возьмём $x_0 = 3$
Шаг спуска выберем $\eta = 0.4$
Градиент функции равен $\nabla f(x) = 2x$

Запустим итерационный процесс: $x_{i+1} = x_i - \eta \cdot \nabla f(x_i)$

$x_1 = x_0 - \eta \cdot \nabla f(x_0) = 3 - 0.4 \cdot 2 \cdot 3 = 0.6, f(x_1) = -1.64$

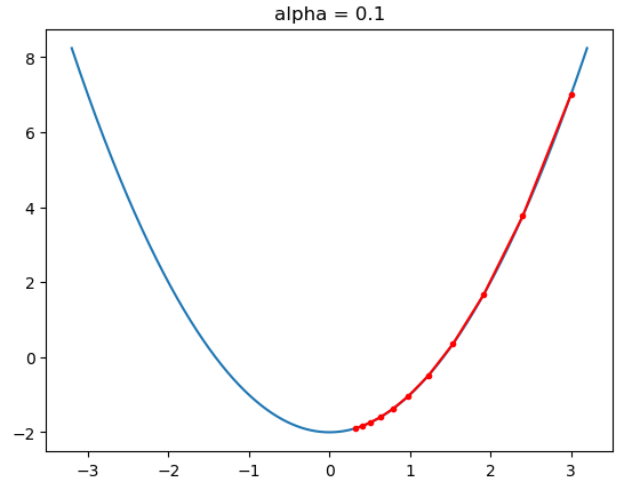
$x_2 = x_1 - \eta \cdot \nabla f(x_1) = 0.6 - 0.4 \cdot 2 \cdot 0.6 = 0.12, f(x_2) = -1.9856$

$x_3 = x_2 - \eta \cdot \nabla f(x_2) = 0.12 - 0.4 \cdot 2 \cdot 0.12 = 0.024, f(x_3) = -1.9994$

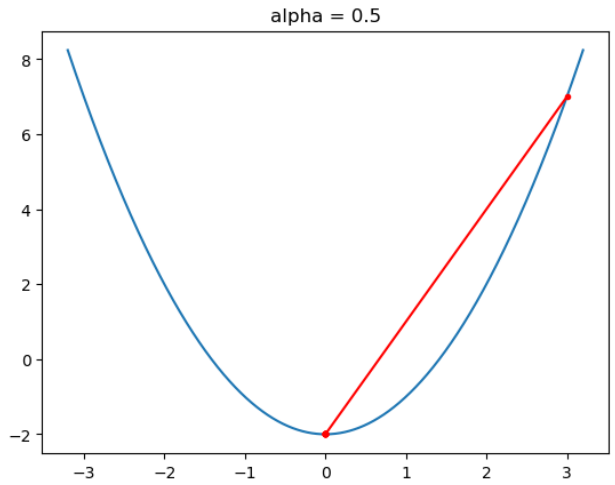
$x_4 = x_3 - \eta \cdot \nabla f(x_3) = 0.024 - 0.4 \cdot 2 \cdot 0.024 = 0.0048, f(x_4) = -1.9997$

$x_5 = x_4 - \eta \cdot \nabla f(x_4) = 0.0048 - 0.4 \cdot 2 \cdot 0.0048 = 0.00096, f(x_4) = -1.9999$

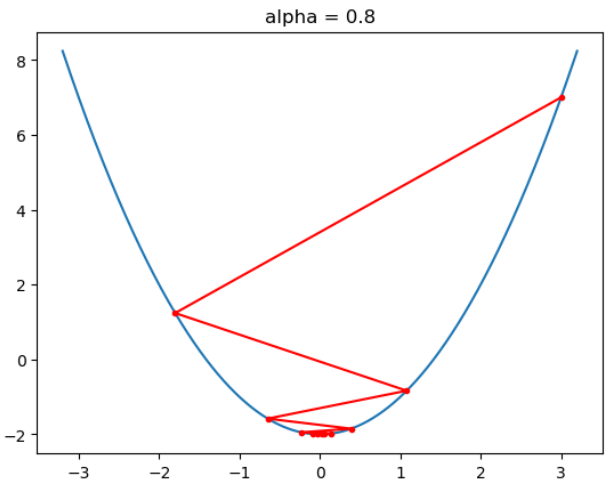
Градиентный спуск. Шаг имеет значение



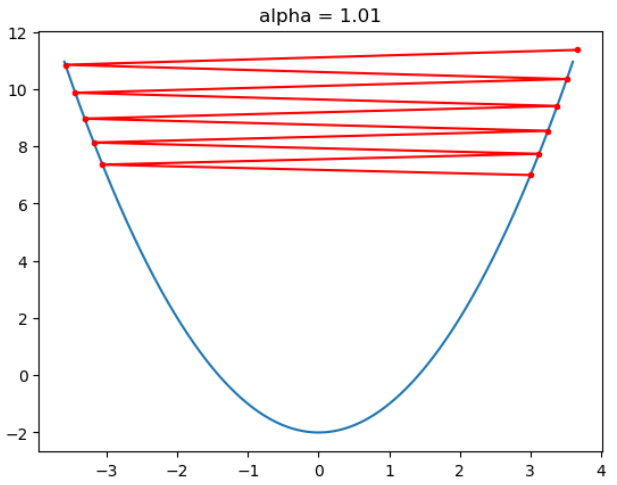
Шаг слишком мал
Долгая сходимость



Идеальный шаг
Моментальная сходимость



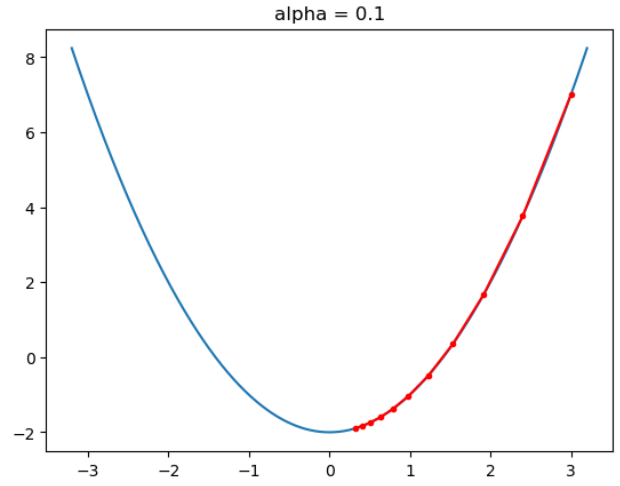
Большой шаг
Плохая сходимость,
градиент постоянно
меняет знак



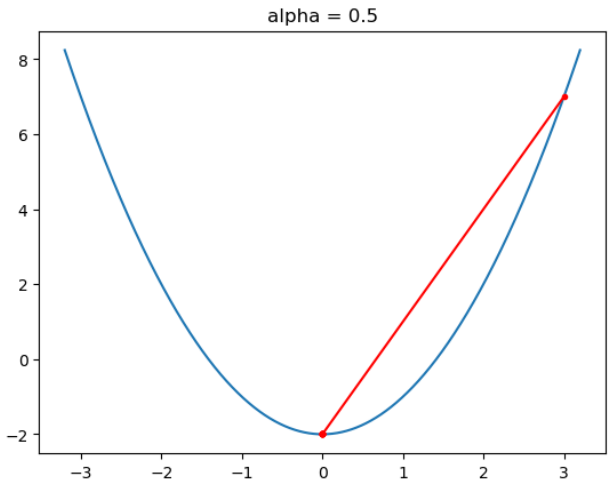
Очень большой шаг
Расходимость

Как выбирать оптимальный шаг?

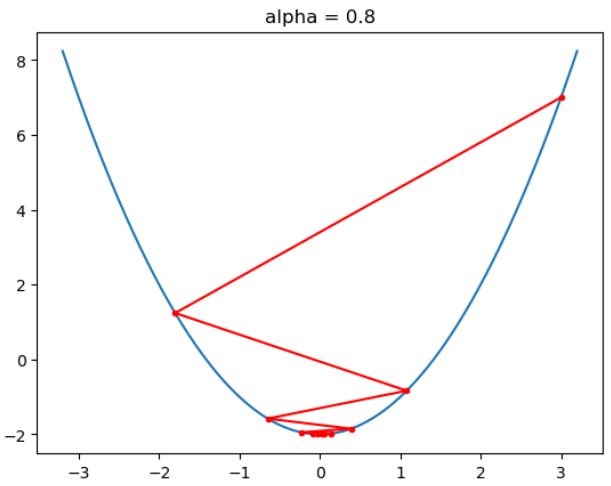
Градиентный спуск. Шаг имеет значение



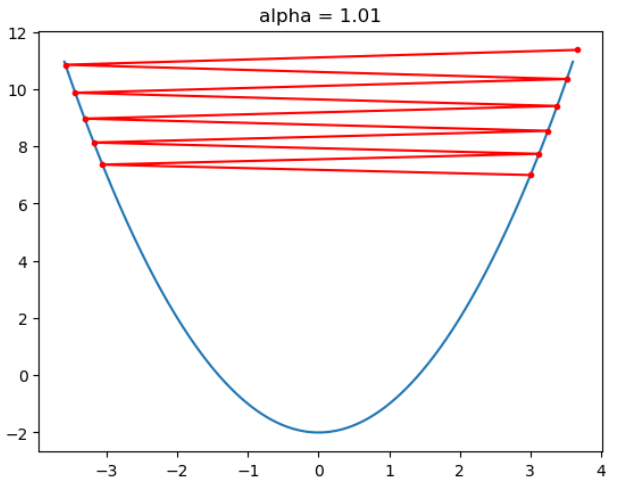
Шаг слишком мал
Долгая сходимость



Идеальный шаг
Моментальная сходимость



Большой шаг
Плохая сходимость,
градиент постоянно
меняет знак



Очень большой шаг
Расходимость

Как выбирать оптимальный шаг – эксперименты!

Обучение сети методом градиентного спуска

Обучение выполняется некоторое количество «эпох» - полных итераций по всей обучающей выборке. Иногда обучение останавливают раньше, если на валидационной выборке качество перестаёт улучшаться (early stopping).

Поскольку выбор шага довольно сложная задача, обычно используются адаптивные методы градиентного спуска, такие как SGD with momentum, Adam, Adagrad, RMSprop и т.д. Зачастую эти методы используют информацию о градиенте на прошлых итерациях и подстраивают шаг при каждом обновлении.

Оптимизаторы градиентного спуска

SGD с моментом:

$$m_t = \beta m_{t-1} + \eta \cdot \nabla \mathrm{L}(\theta)$$
$$\theta = \theta - m_t$$

Nesterov accelerated gradient:

$$v_t = \beta v_{t-1} + \eta \cdot \nabla \mathrm{L}(\theta - \beta v_{t-1})$$
$$\theta = \theta - v_t$$

Adagrad:

$$G_t = G_{t-1} + \nabla \mathrm{L}(\theta)^2$$
$$\theta = \theta - \eta \frac{\nabla \mathrm{L}(\theta)}{\sqrt{G_t + \epsilon}}$$

SGD:

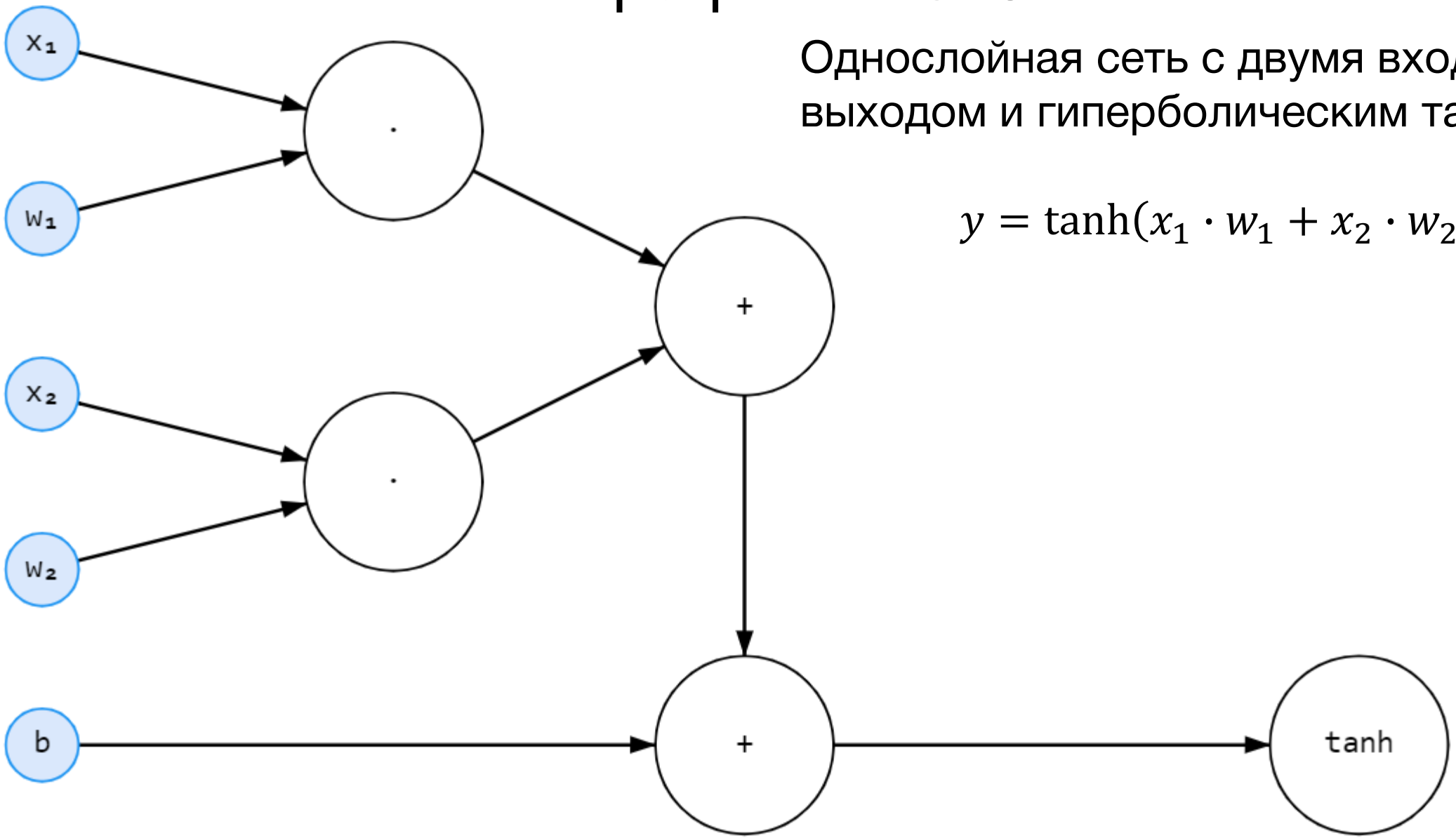
$$\theta = \theta - \eta \cdot \nabla \mathrm{L}(\theta)$$

Adam:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \cdot \nabla \mathrm{L}(\theta)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \cdot \nabla \mathrm{L}(\theta)^2$$
$$\widehat{m_t} = \frac{m_t}{1 - \beta_1^t}, \qquad \widehat{v_t} = \frac{v_t}{1 - \beta_2^t}$$
$$\theta = \theta - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t} + \epsilon}}$$

Вопрос: как вычислять градиент $\nabla L(\theta)$ по каждому параметру θ_i глубокой нейронной сети (сложной дифференцируемой функции)?

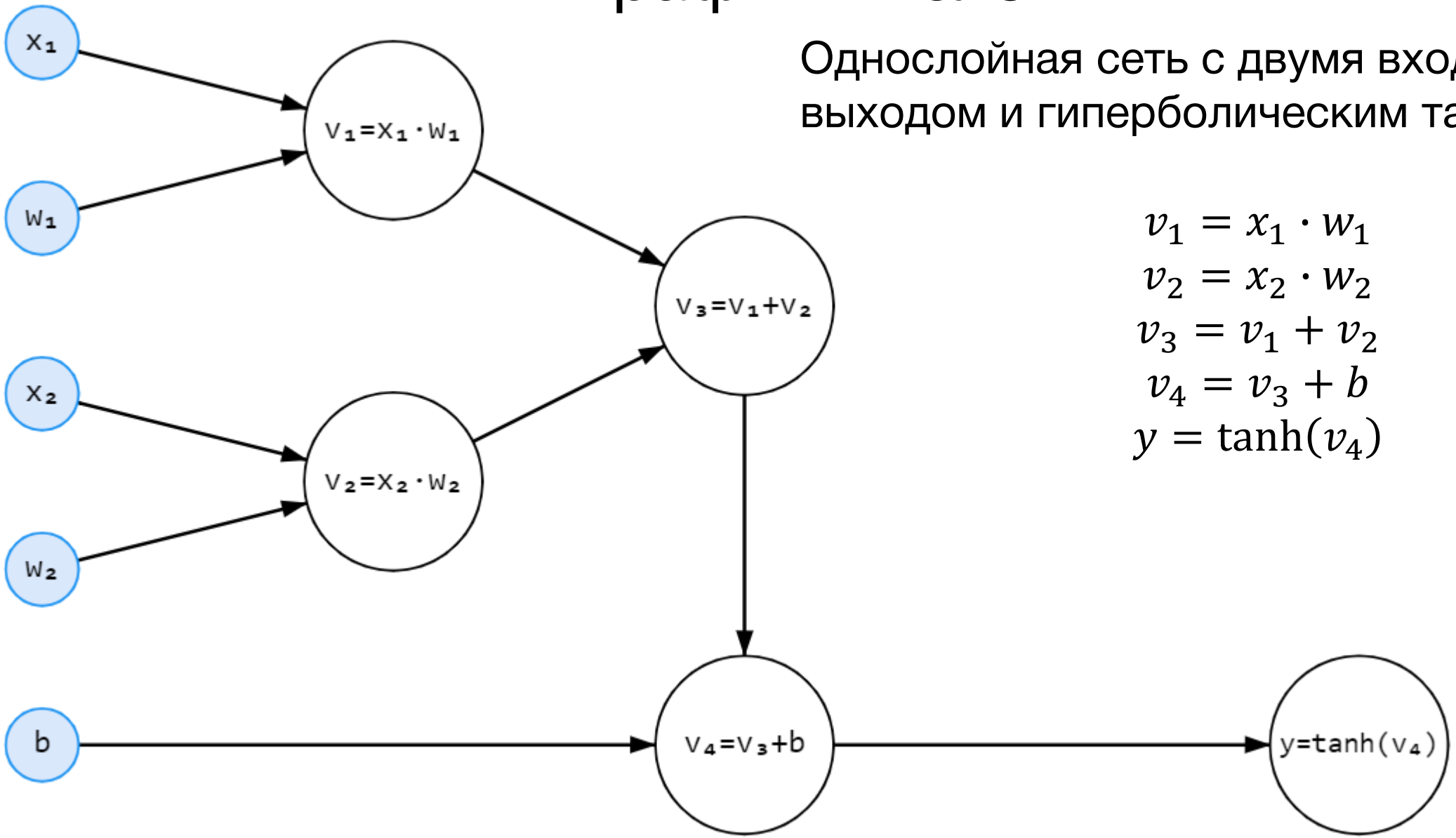
Граф вычислений



Однослойная сеть с двумя входами, одним выходом и гиперболическим тангенсом:

$$y = \tanh(x_1 \cdot w_1 + x_2 \cdot w_2 + b)$$

Граф вычислений



Однослойная сеть с двумя входами, одним выходом и гиперболическим тангенсом:

$$v_1 = x_1 \cdot w_1$$
$$v_2 = x_2 \cdot w_2$$
$$v_3 = v_1 + v_2$$
$$v_4 = v_3 + b$$
$$y = \tanh(v_4)$$

Обратное распространение ошибки

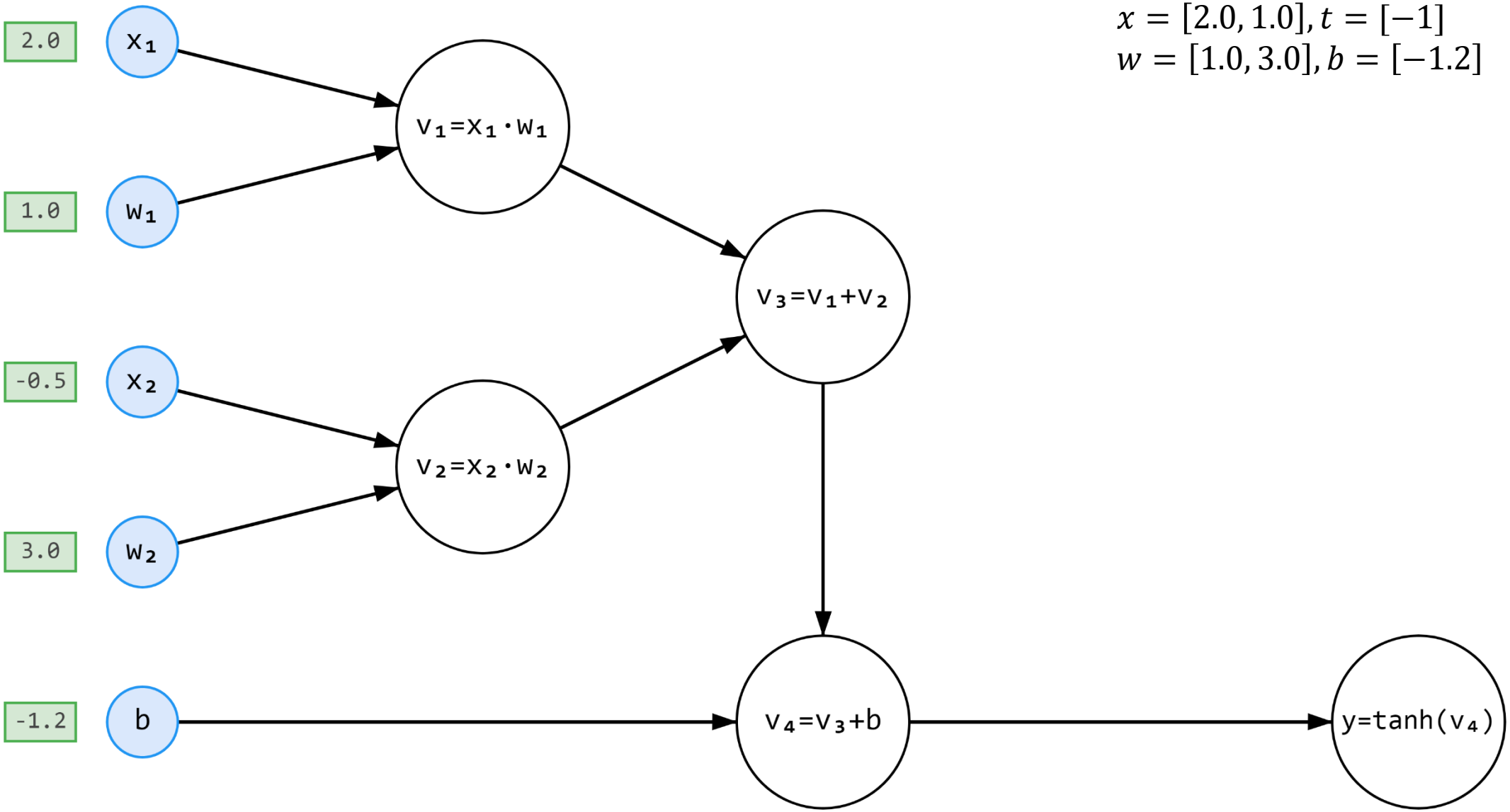
Обратное распространение ошибки (backpropagation) – алгоритм, позволяющий вычислить частные производные на произвольном графе вычислений. Состоит из двух этапов:

Прямое распространение (forward pass): входные сигналы проходят по графу через каждый узел и сохраняют выходные значения и частные производные узлов.

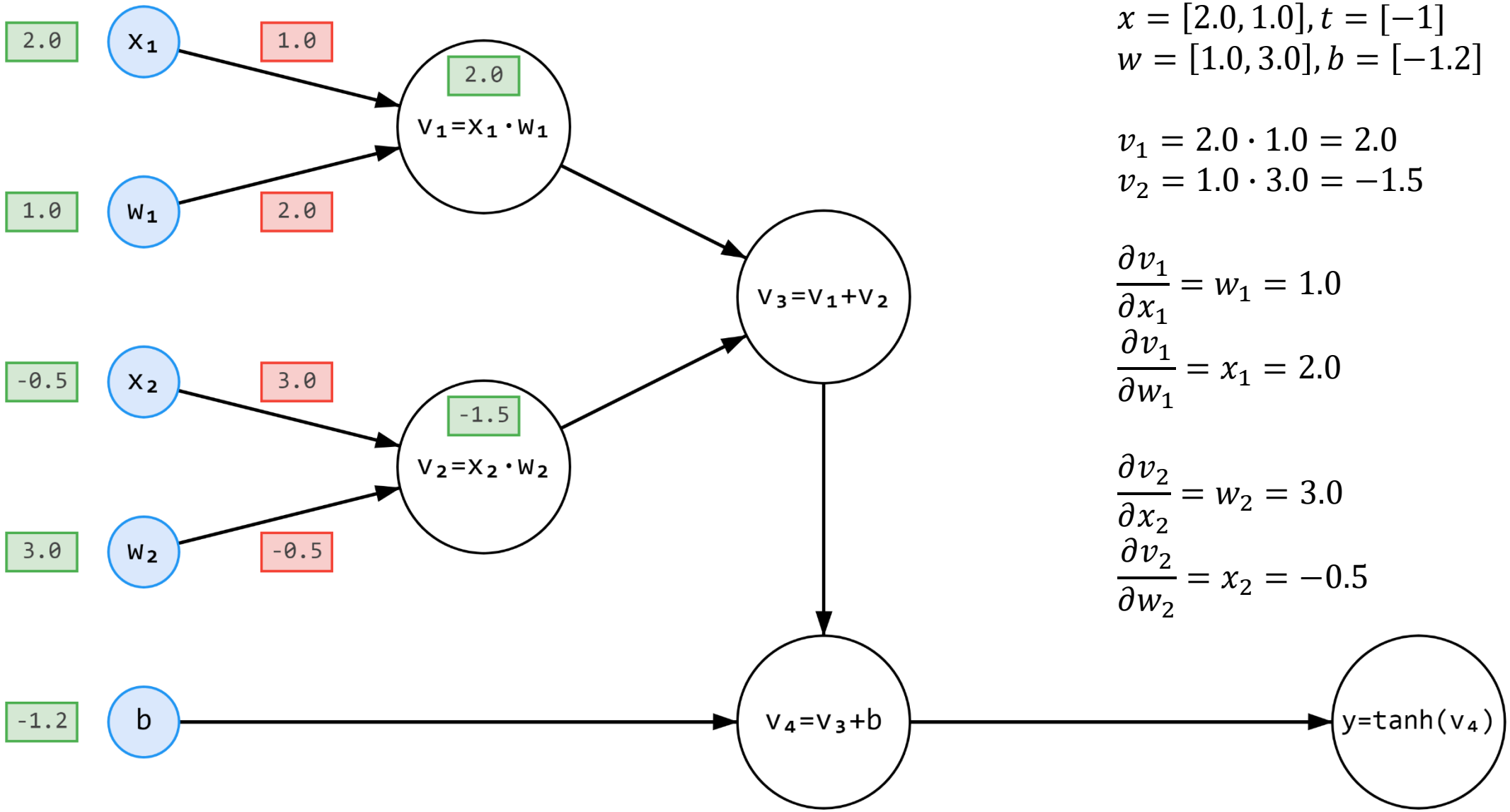
Обратное распространение (backward pass): на основе вычисленных значений при прямом проходе вычисляются частные производные реализуемой функции по каждому из её аргументов – входных узлов графа.

Прямое распространение

$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$



Прямое распространение



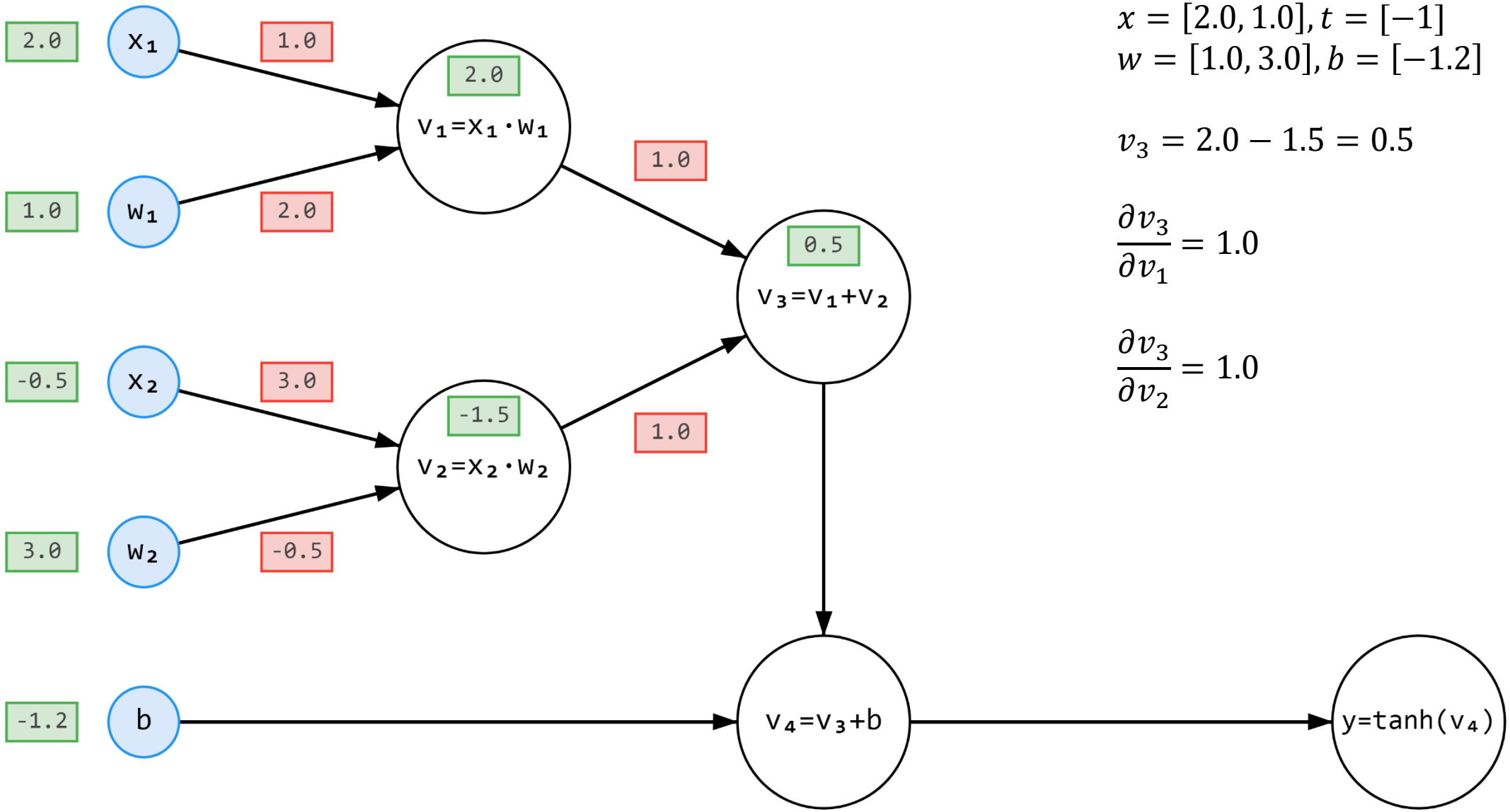
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$

$v_1 = 2.0 \cdot 1.0 = 2.0$
$v_2 = 1.0 \cdot 3.0 = -1.5$

$\frac{\partial v_1}{\partial x_1} = w_1 = 1.0$
$\frac{\partial v_1}{\partial w_1} = x_1 = 2.0$

$\frac{\partial v_2}{\partial x_2} = w_2 = 3.0$
$\frac{\partial v_2}{\partial w_2} = x_2 = -0.5$

Прямое распространение



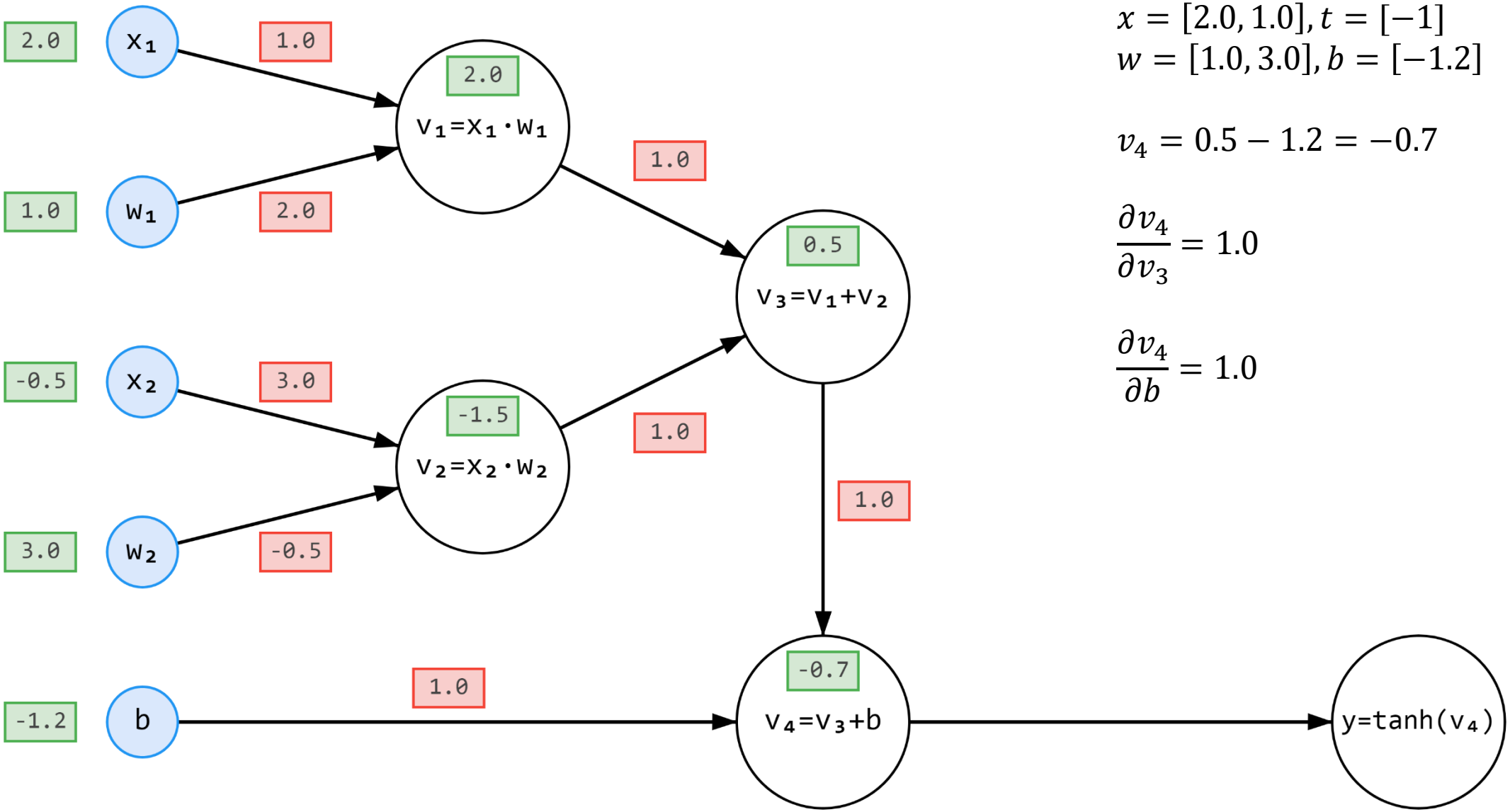
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$

$v_3 = 2.0 - 1.5 = 0.5$

$\frac{\partial v_3}{\partial v_1} = 1.0$

$\frac{\partial v_3}{\partial v_2} = 1.0$

Прямое распространение



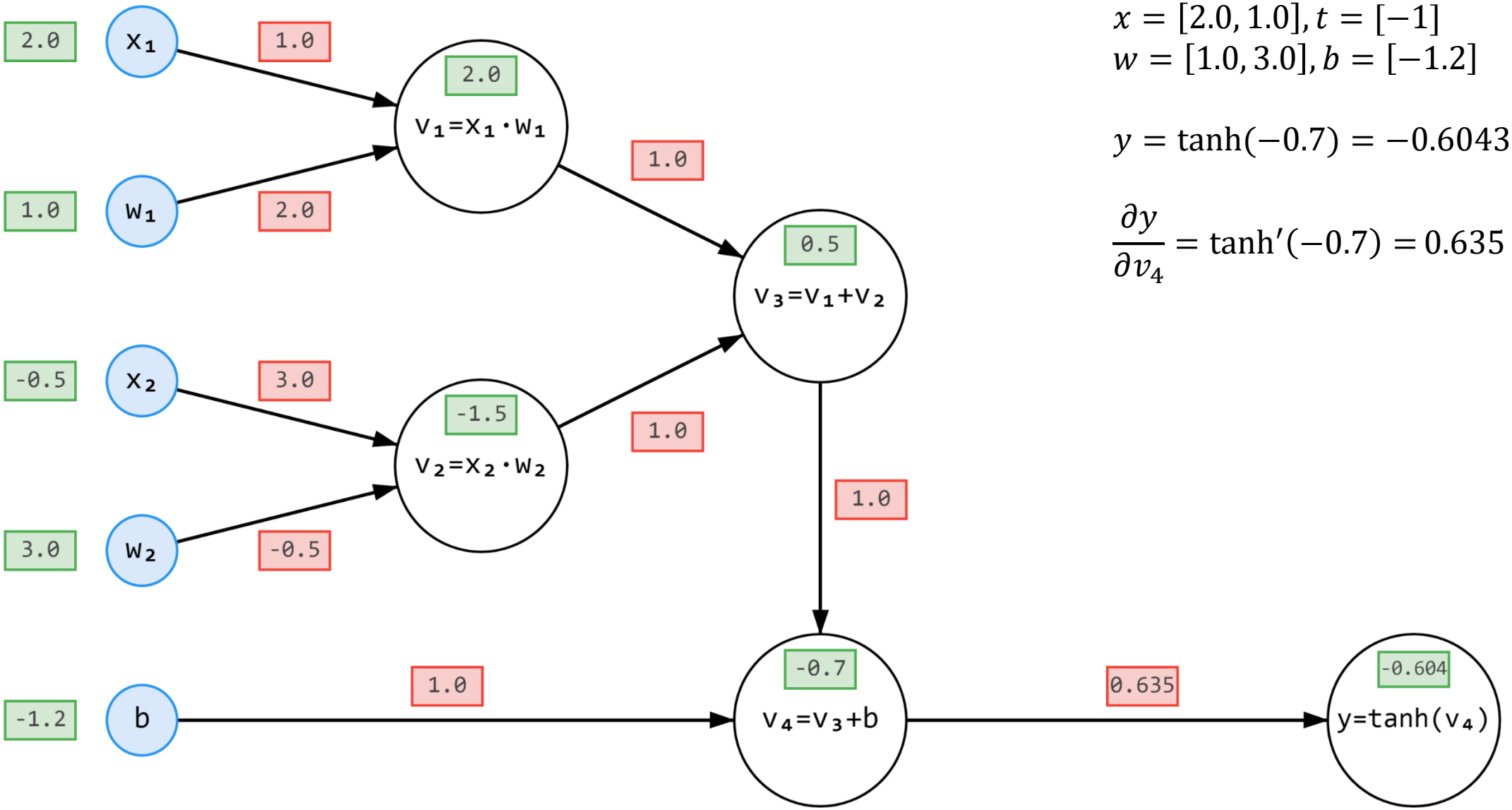
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$

$v_4 = 0.5 - 1.2 = -0.7$

$\frac{\partial v_4}{\partial v_3} = 1.0$

$\frac{\partial v_4}{\partial b} = 1.0$

Прямое распространение

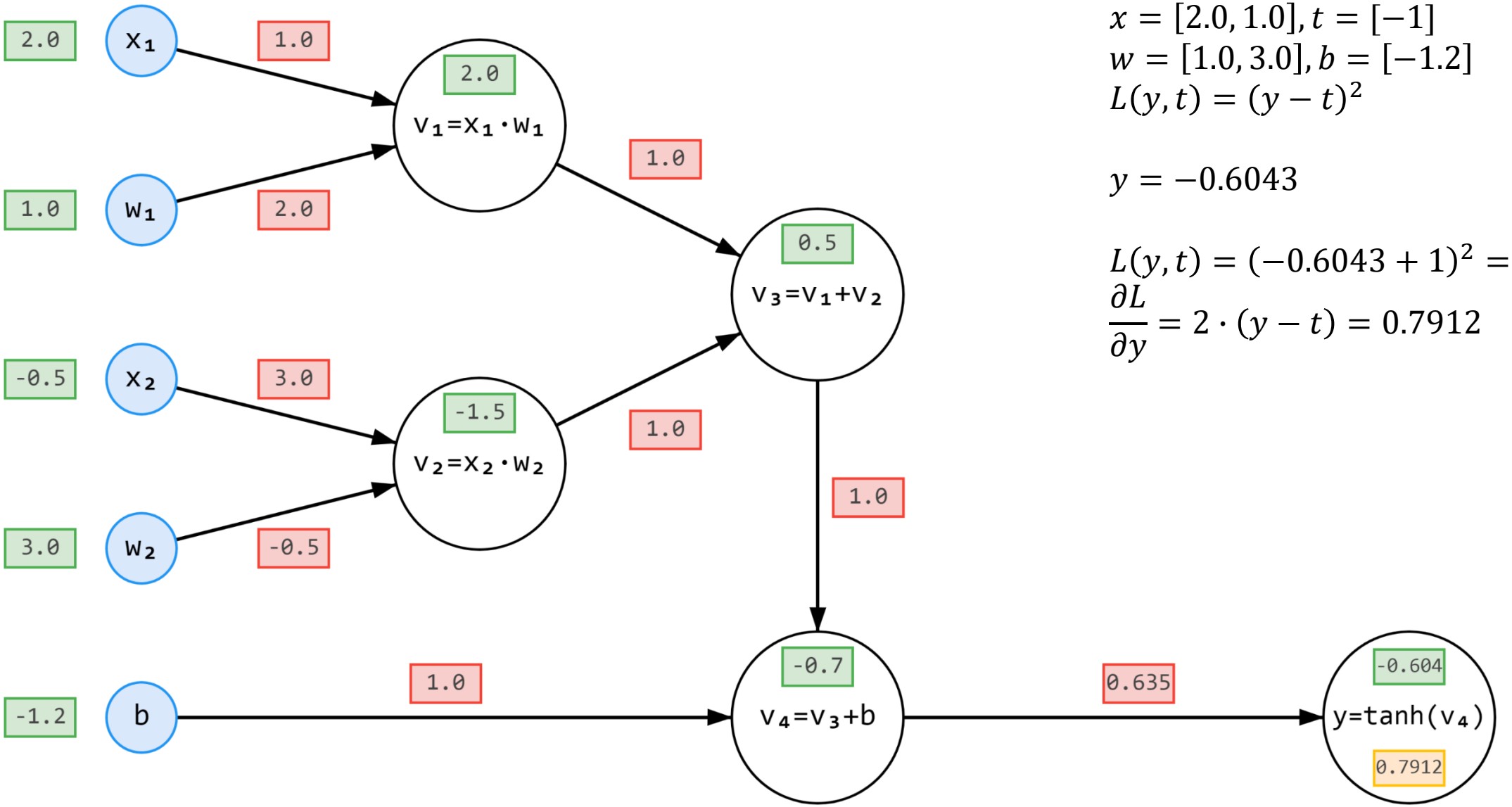


$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$

$y = \tanh(-0.7) = -0.6043$

$\frac{\partial y}{\partial v_4} = \tanh'(-0.7) = 0.635$

Обратное распространение

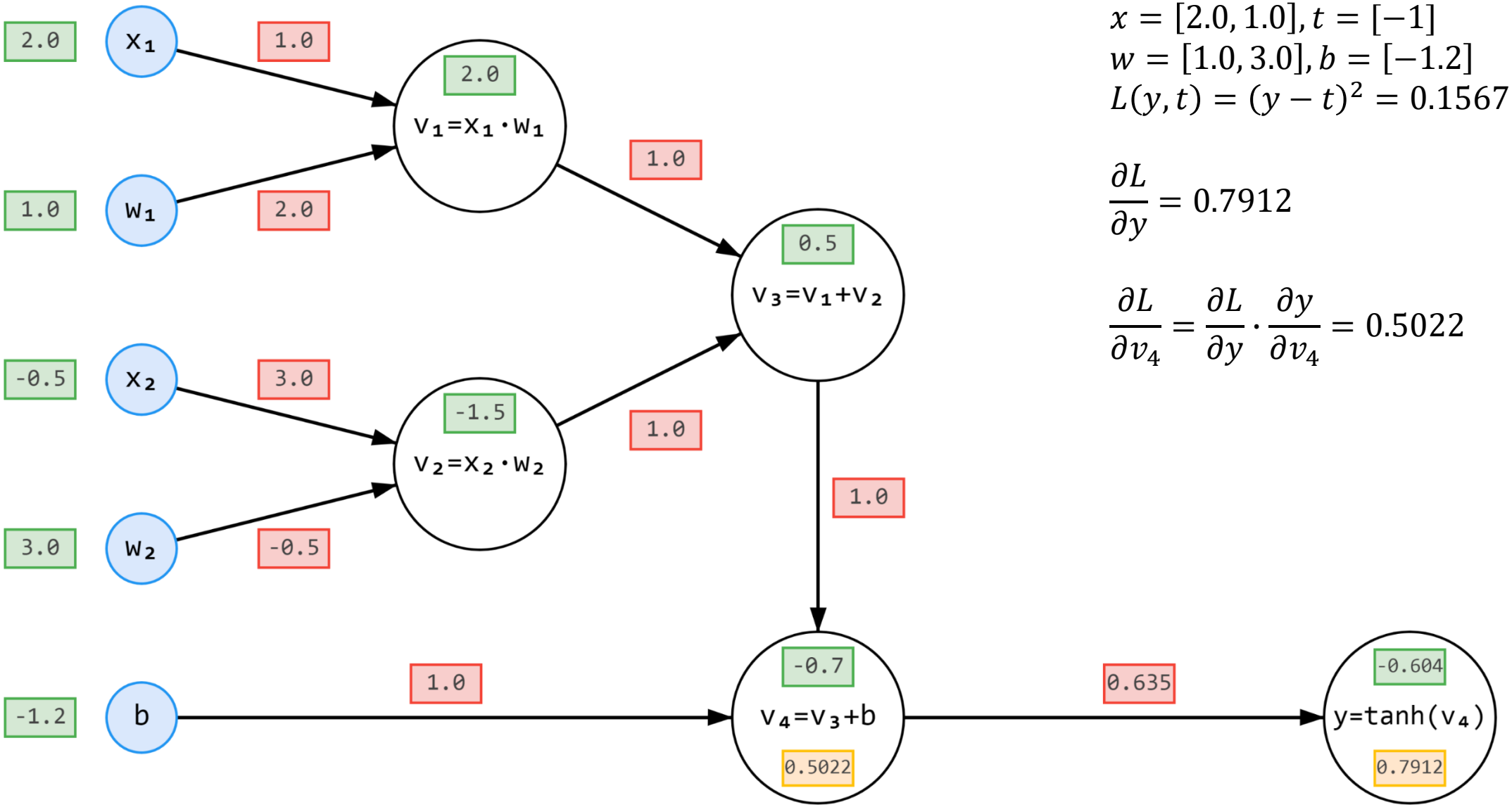


$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$
$L(y,t) = (y-t)^2$

$y = -0.6043$

$L(y,t) = (-0.6043 + 1)^2 = 0.1567$
$\frac{\partial L}{\partial y} = 2 \cdot (y-t) = 0.7912$

Обратное распространение

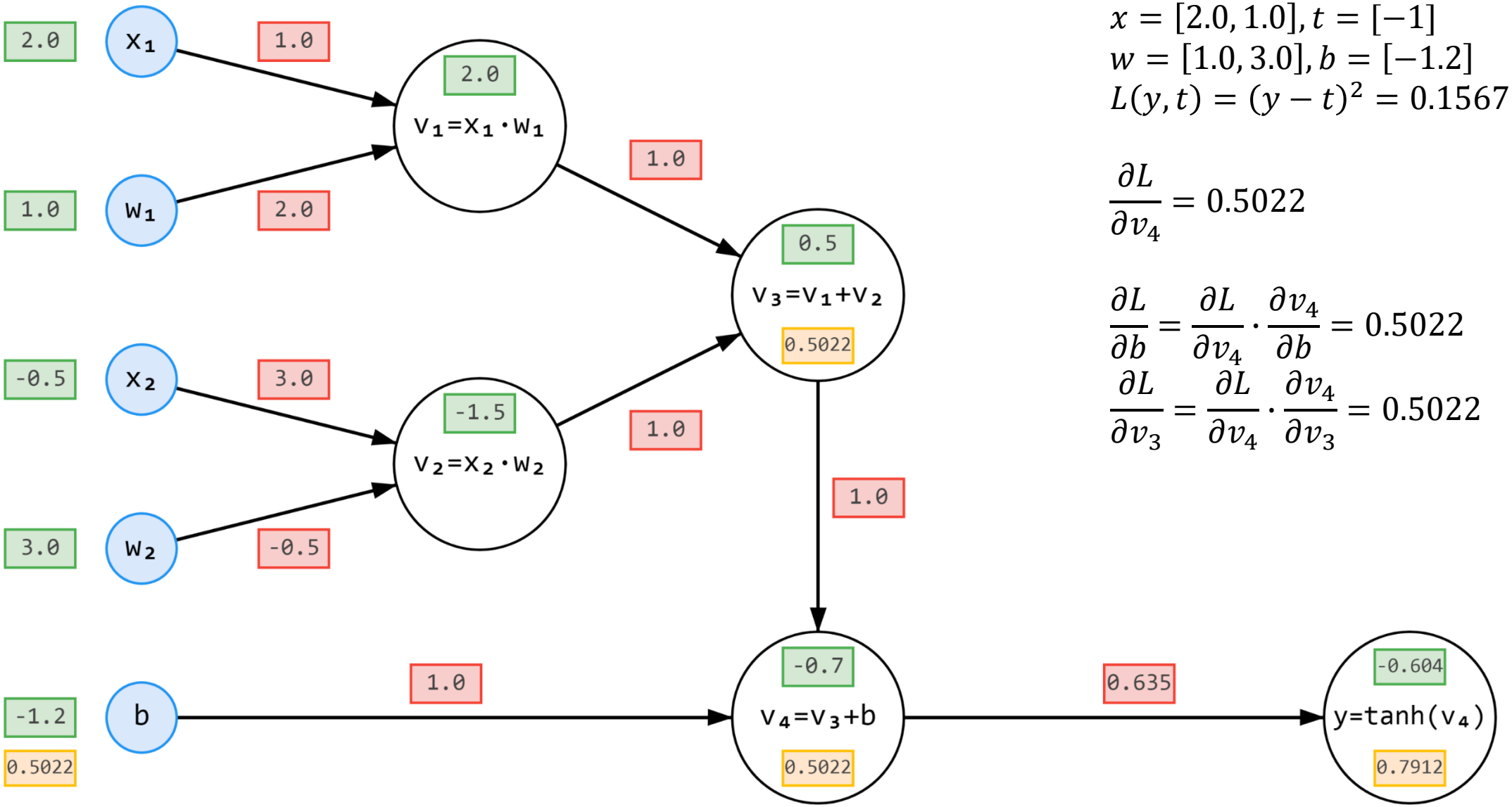


$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$
$L(y,t) = (y-t)^2 = 0.1567$

$$\frac{\partial L}{\partial y} = 0.7912$$

$$\frac{\partial L}{\partial v_4} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial v_4} = 0.5022$$

Обратное распространение



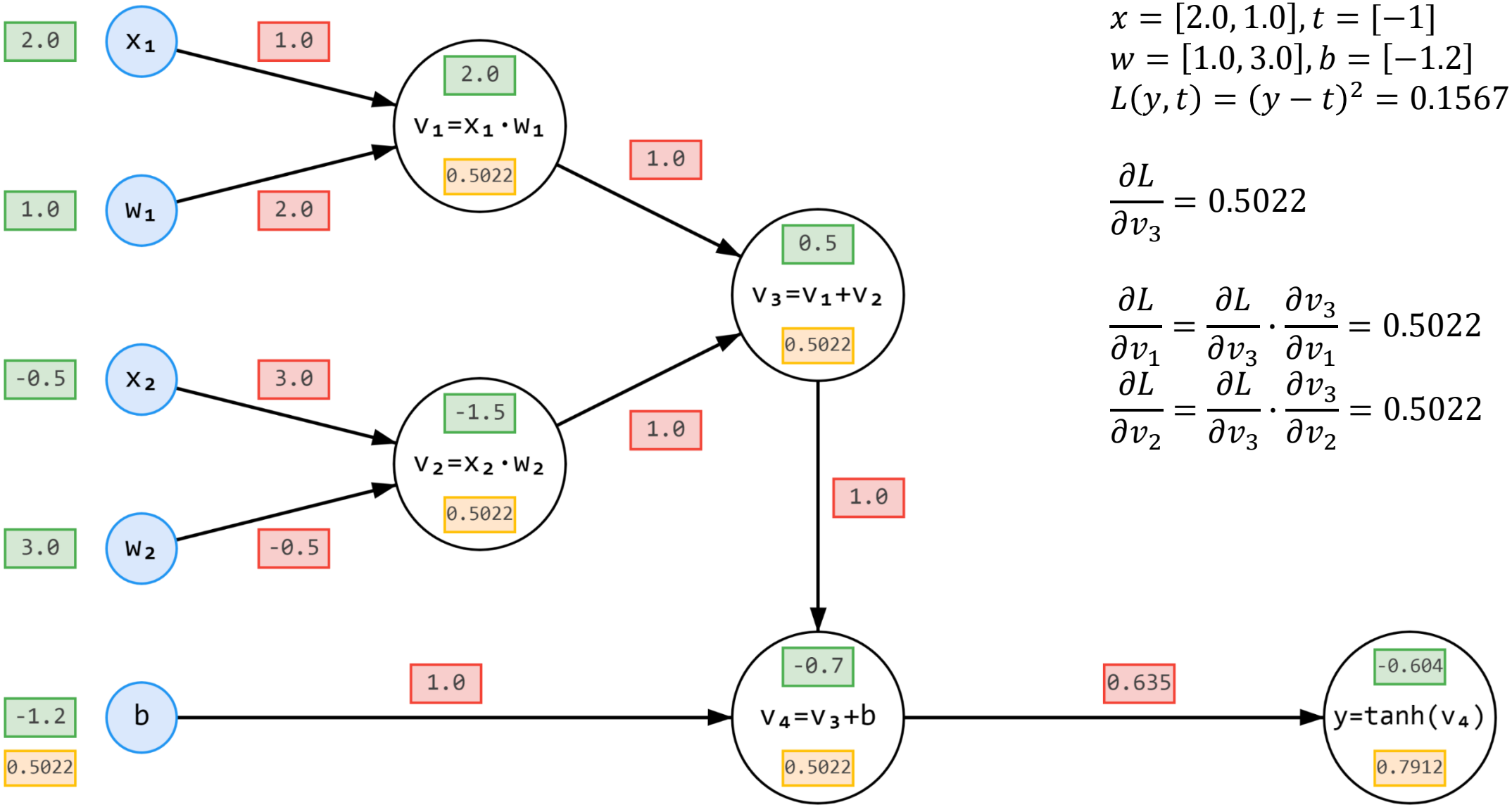
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$
$L(y,t) = (y-t)^2 = 0.1567$

$$\frac{\partial L}{\partial v_4} = 0.5022$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial v_4} \cdot \frac{\partial v_4}{\partial b} = 0.5022$$

$$\frac{\partial L}{\partial v_3} = \frac{\partial L}{\partial v_4} \cdot \frac{\partial v_4}{\partial v_3} = 0.5022$$

Обратное распространение



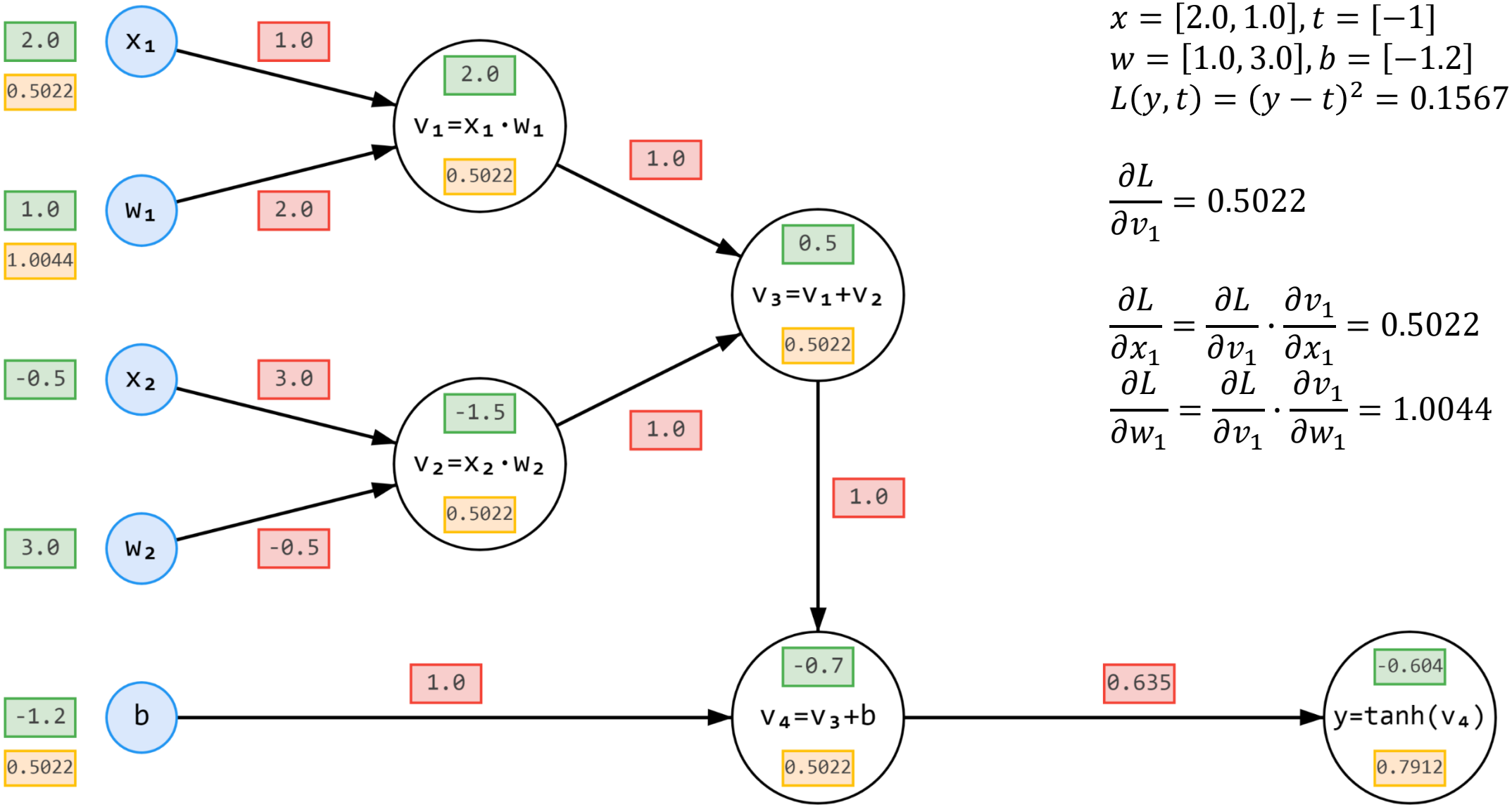
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$
$L(y,t) = (y-t)^2 = 0.1567$

$$\frac{\partial L}{\partial v_3} = 0.5022$$

$$\frac{\partial L}{\partial v_1} = \frac{\partial L}{\partial v_3} \cdot \frac{\partial v_3}{\partial v_1} = 0.5022$$

$$\frac{\partial L}{\partial v_2} = \frac{\partial L}{\partial v_3} \cdot \frac{\partial v_3}{\partial v_2} = 0.5022$$

Обратное распространение



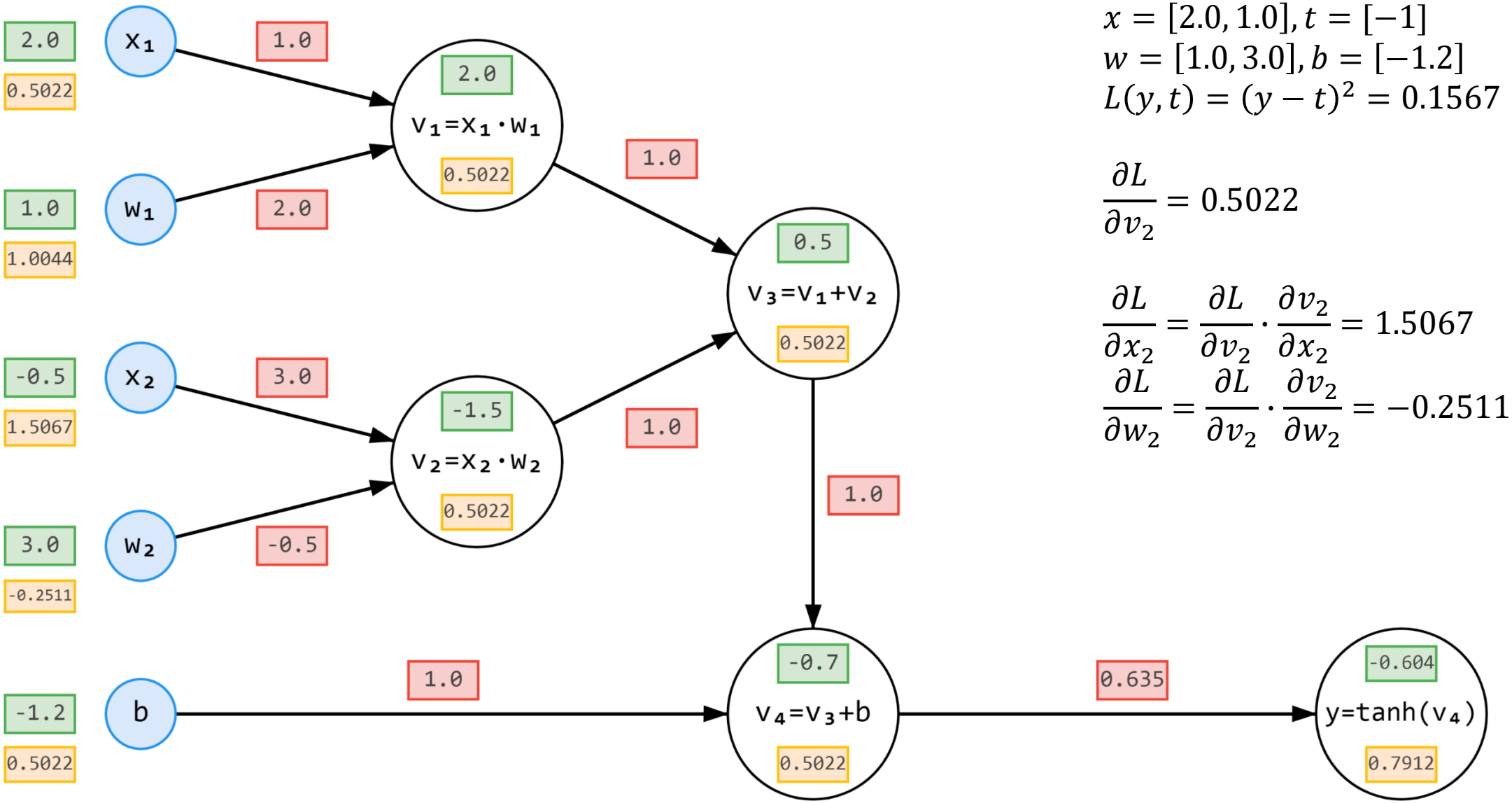
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$
$L(y,t) = (y-t)^2 = 0.1567$

$$\frac{\partial L}{\partial v_1} = 0.5022$$

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial v_1} \cdot \frac{\partial v_1}{\partial x_1} = 0.5022$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial v_1} \cdot \frac{\partial v_1}{\partial w_1} = 1.0044$$

Обратное распространение



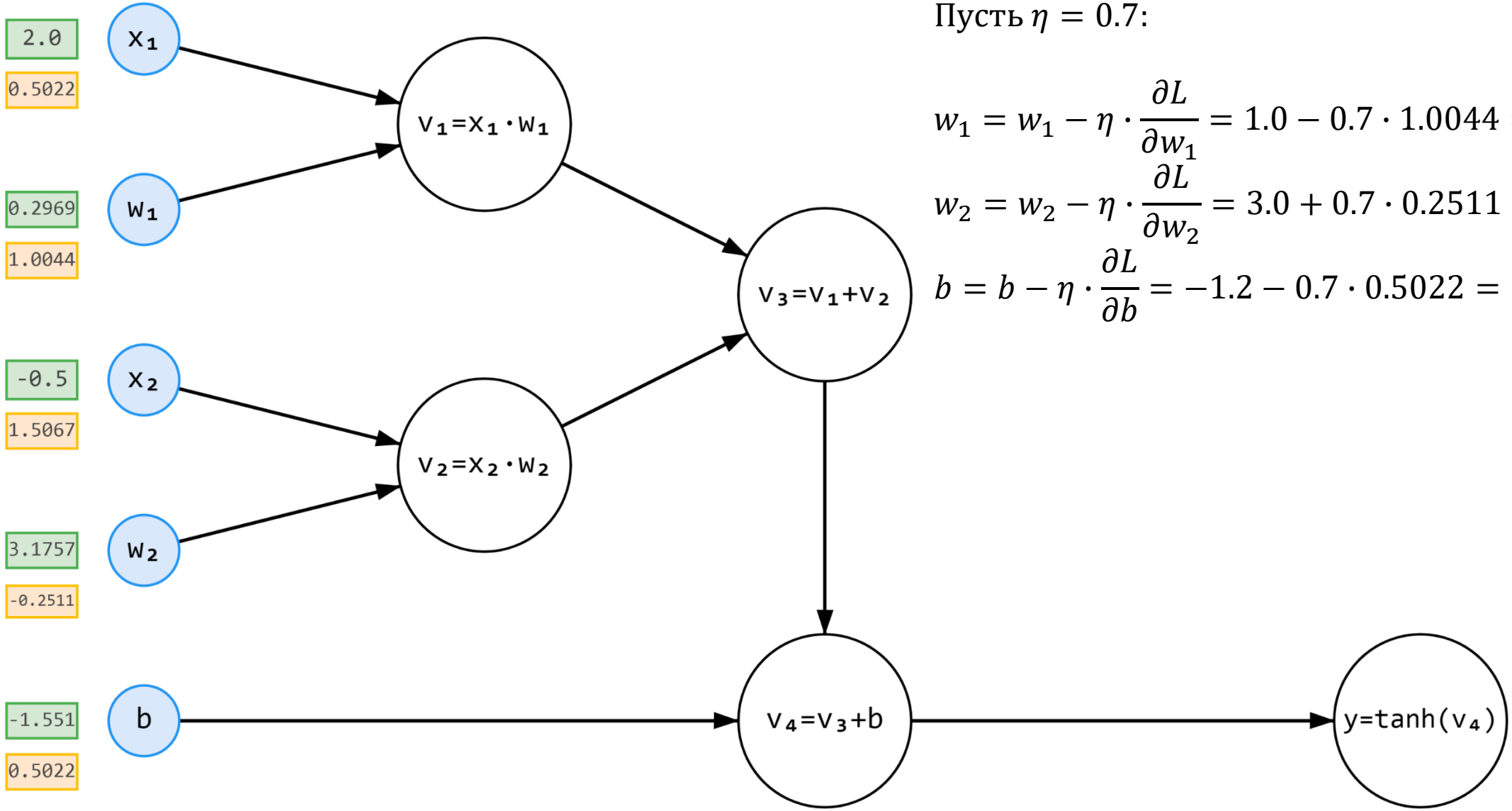
$x = [2.0, 1.0], t = [-1]$
$w = [1.0, 3.0], b = [-1.2]$
$L(y,t) = (y-t)^2 = 0.1567$

$$\frac{\partial L}{\partial v_2} = 0.5022$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial v_2} \cdot \frac{\partial v_2}{\partial x_2} = 1.5067$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial v_2} \cdot \frac{\partial v_2}{\partial w_2} = -0.2511$$

Обновление весовых коэффициентов



Пусть $\eta = 0.7$:

$$w_1 = w_1 - \eta \cdot \frac{\partial L}{\partial w_1} = 1.0 - 0.7 \cdot 1.0044 = 0.2969$$

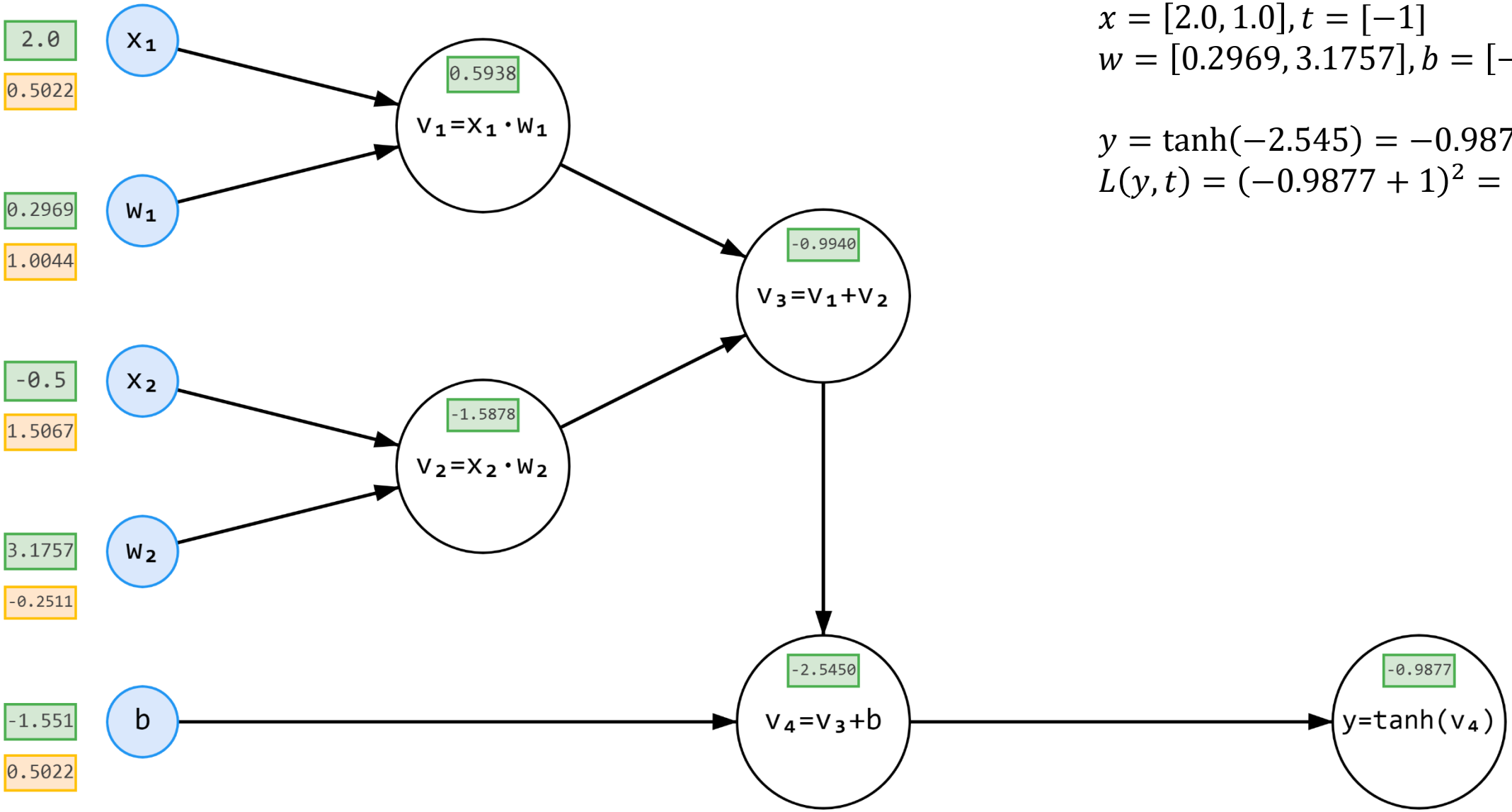
$$w_2 = w_2 - \eta \cdot \frac{\partial L}{\partial w_2} = 3.0 + 0.7 \cdot 0.2511 = 3.1757$$

$$b = b - \eta \cdot \frac{\partial L}{\partial b} = -1.2 - 0.7 \cdot 0.5022 = -1.5515$$

Прямое распространение (here we go again)

$x = [2.0, 1.0], t = [-1]$
$w = [0.2969, 3.1757], b = [-1.551]$

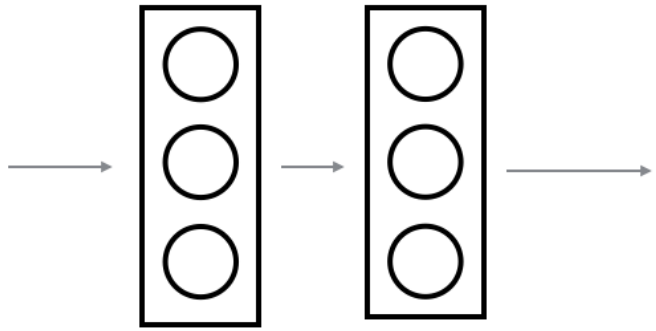
$y = \tanh(-2.545) = -0.9877$
$L(y, t) = (-0.9877 + 1)^2 = 0.00015$



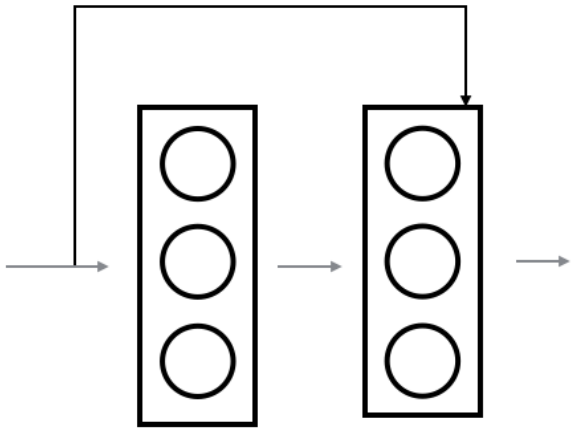
Проблема затухающего градиента

При backpropagation в глубокой нейронной сети градиент на первых слоях может быть очень маленьким из-за последовательного умножения на $\left|\frac{\partial z}{\partial y}\right| \ll 1$, поэтому эти слои будут обновляться «слабо».

Чтобы с этим бороться, выбирают специальные функции активации и разрабатывают специальные архитектуры, распространяющие градиент на первые слои (skip / residual connections).



without skip connection



with skip connection

Начальная инициализация

Перед обучением нейронной сети нужно проинициализировать ее параметры θ. Обычно используется нормальное распределение с нулевым средним и среднеквадратичным отклонением обратнопропроциональным количеству коэффициентов: $W \sim N\left(0, \frac{1}{\dim(W)}\right)$.

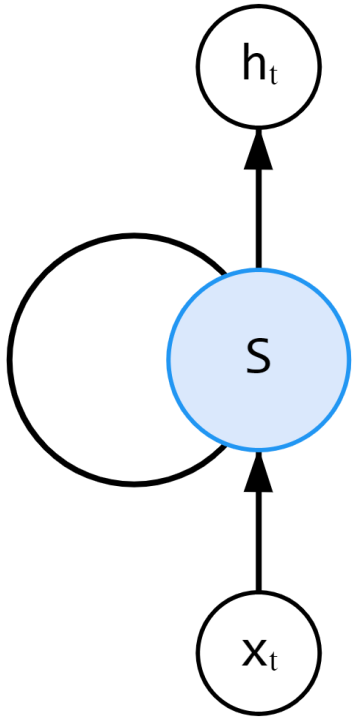
Существуют техники, учитывающие функции активации и размеры слоя (He, Xavier).

Нужно аккуратно подбирать начальную инициализацию весов, функции активации, размеры и количество слоев на валидационной выборке – проблемы недообучения / переобучения

Вопрос: почему бы не инициализировать все веса нулём? А константой?

Рекуррентные нейронные сети (RNN)

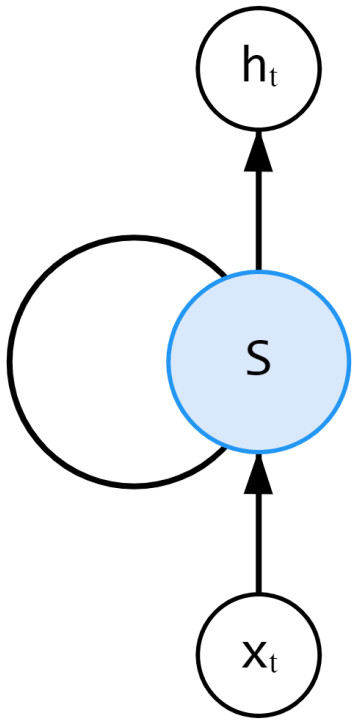
- могут обрабатывать последовательности произвольной длины
- выходные значения сети, получаемые на очередном временном шаге, используются на следующем шаге – сеть имеет состояние
- итоговое выходное значение будут получено лишь после завершения обработки всей последовательности (на последнем временном шаге)



$$h_0 = 0$$
$$h_t = f(x_t, h_{t-1}), t \in [1, k]$$

Рекуррентные нейронные сети (RNN)

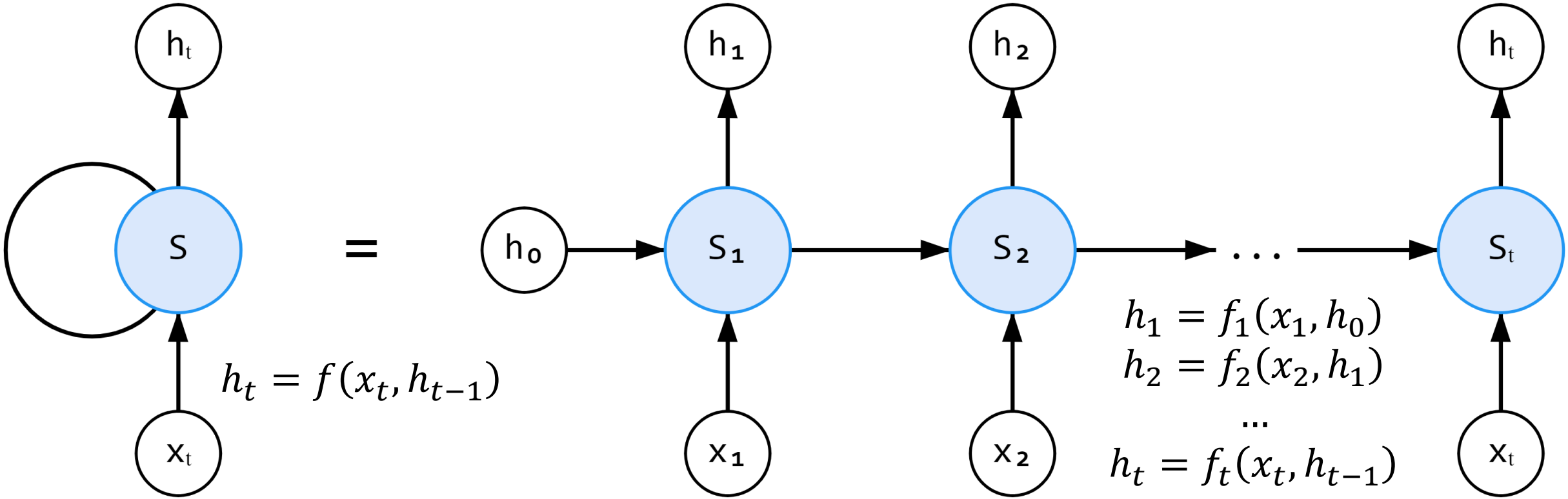
- при вычислении учитывается историческая информация благодаря рекуррентной связи
- размер модели не увеличивается с размером входных данных
- вычисления идут медленнее в сравнении с полносвязными сетями



$$h_0 = 0$$
$$h_t = f(x_t, h_{t-1}), t \in [1, k]$$

Разворачивание рекуррентной нейронной сети (unfold)

Работу RNN можно представить в виде последовательного применения многослойной нерекуррентной сети, в которой каждое состояние обрабатывается с помощью общих весовых коэффициентов.



Простейшая рекуррентная сеть

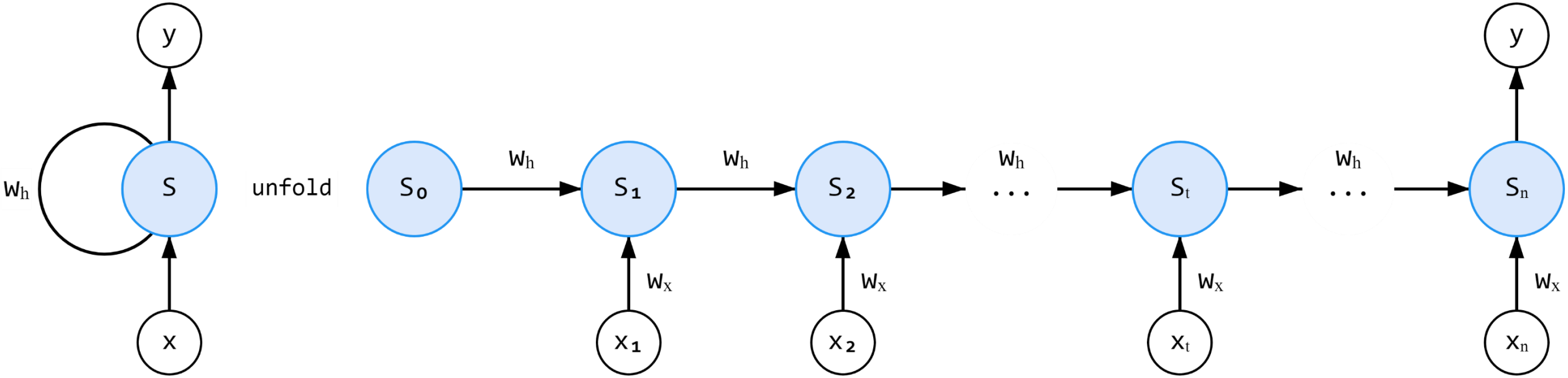
$$h_0 = 0$$
$$h_t = f(W_x \cdot x_t + W_h \cdot h_{t-1} + b)$$

W_x – входные весовые коэффициенты
W_h – весовые коэффициенты рекуррентной связи
b – вес смещения
x_t – очередной элемент входной последовательности
h_t – выходное значение (состояние)
h_{t-1} – состояние на прошлом временном шаге

Рекуррентная сеть. Пример

Сеть, подсчитывающая количество единиц, находящихся в двоичном входном потоке: $h_0 = 0, h_t = w_x \cdot x_t + w_h \cdot h_{t-1}, y = h_n$

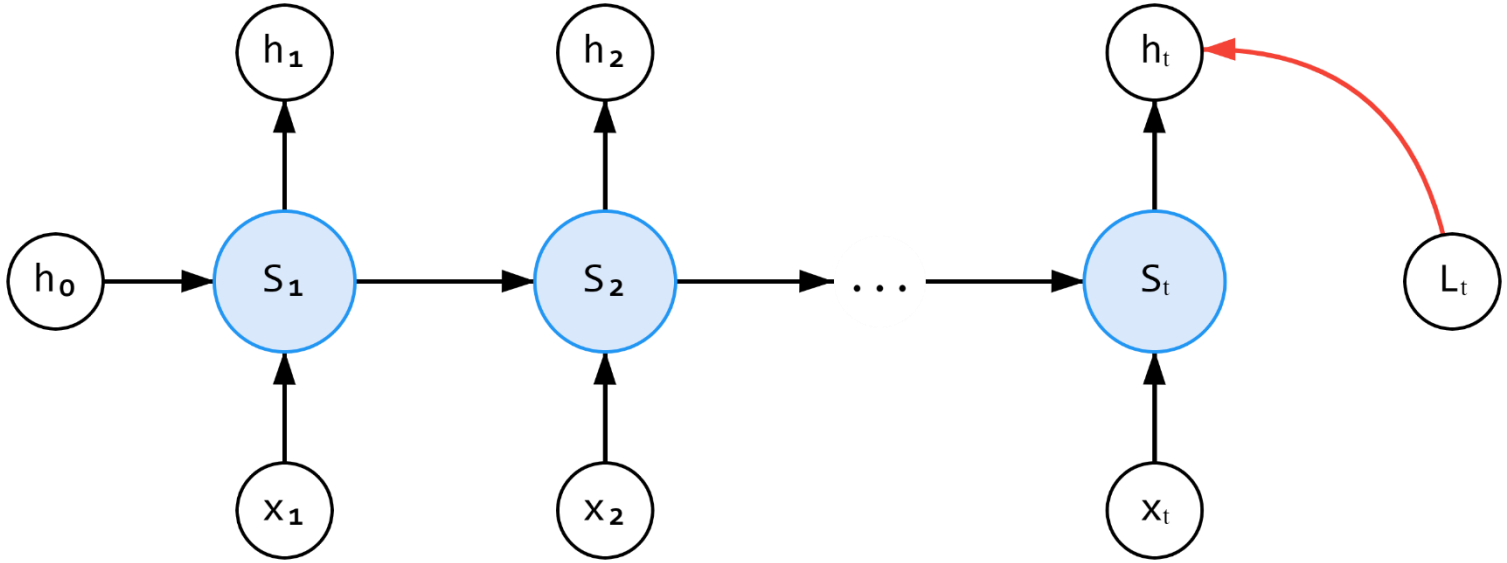
$w_x = 1, w_h = 1 \rightarrow h_t = x_t + h_{t-1}$



Обратное распространение во времени

Ошибка распространяется от последнего к первому временному шагу, при этом разворачиваются все временные шаги. Это позволяет рассчитывать ошибку для каждого временного шага и обновлять веса.

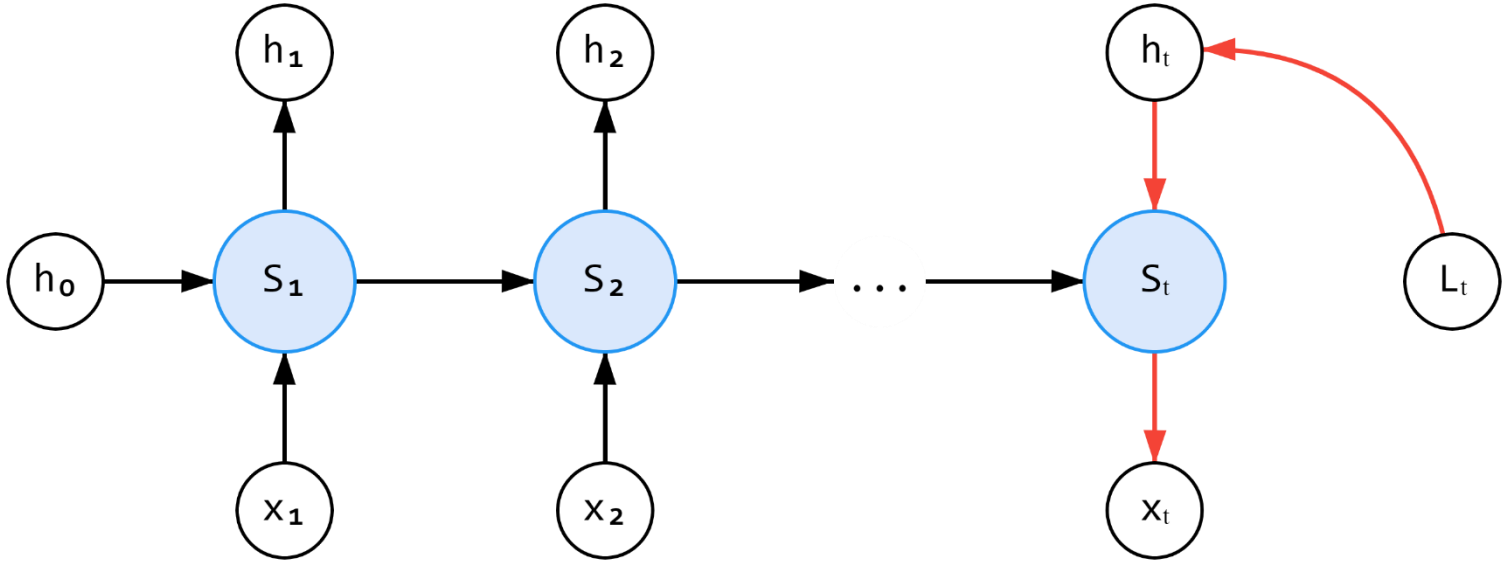
Из-за разворачивания по всем временным шагам обратное распространение ошибки является весьма трудозатратным при большом размере входной последовательности.



Обратное распространение во времени

Ошибка распространяется от последнего к первому временному шагу, при этом разворачиваются все временные шаги. Это позволяет рассчитывать ошибку для каждого временного шага и обновлять веса.

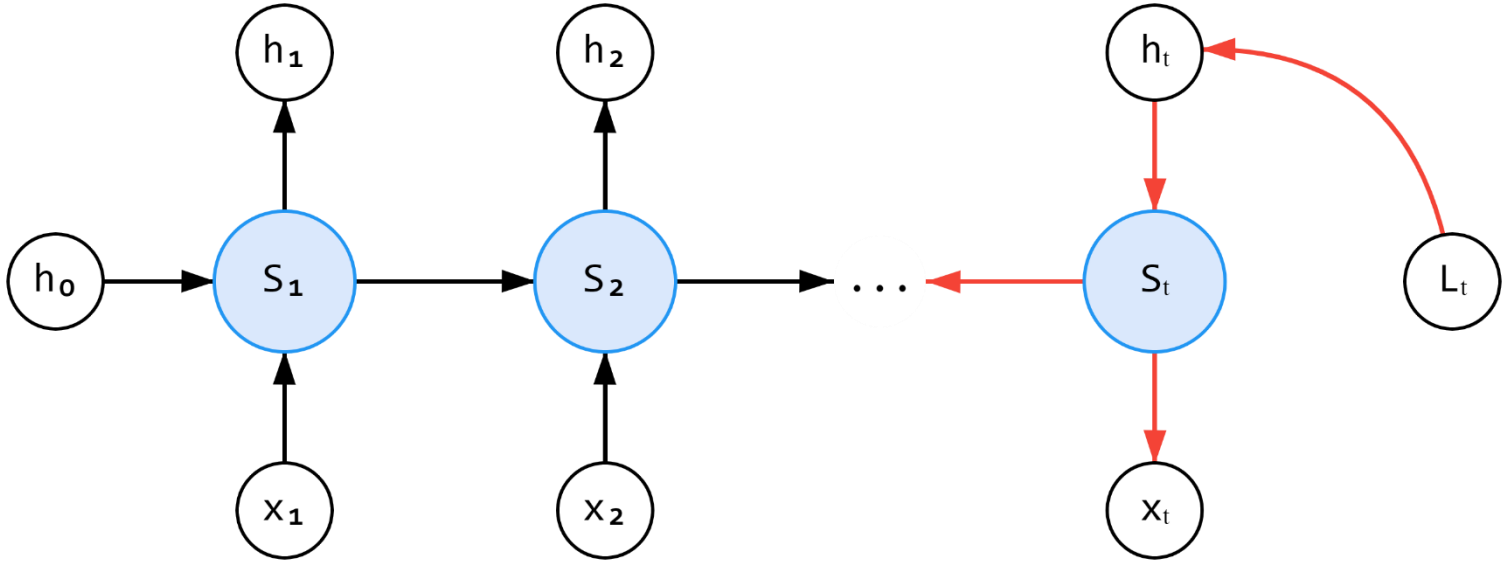
Из-за разворачивания по всем временным шагам обратное распространение ошибки является весьма трудозатратным при большом размере входной последовательности.



Обратное распространение во времени

Ошибка распространяется от последнего к первому временному шагу, при этом разворачиваются все временные шаги. Это позволяет рассчитывать ошибку для каждого временного шага и обновлять веса.

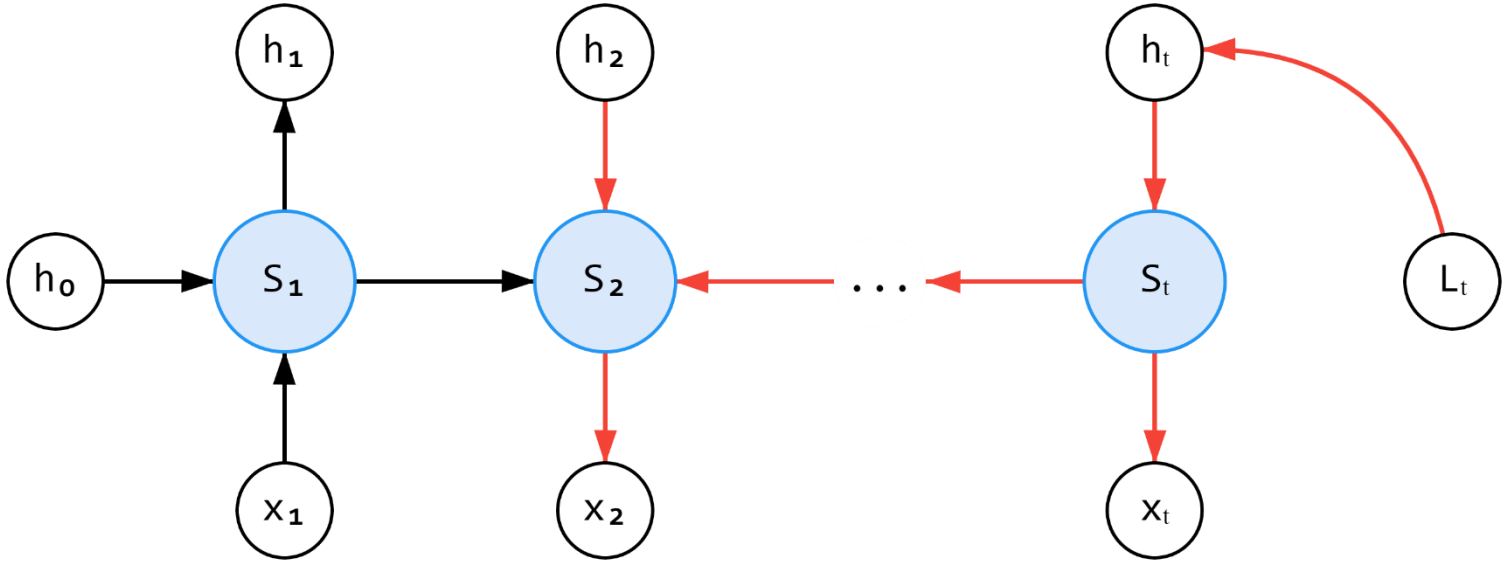
Из-за разворачивания по всем временным шагам обратное распространение ошибки является весьма трудозатратным при большом размере входной последовательности.



Обратное распространение во времени

Ошибка распространяется от последнего к первому временному шагу, при этом разворачиваются все временные шаги. Это позволяет рассчитывать ошибку для каждого временного шага и обновлять веса.

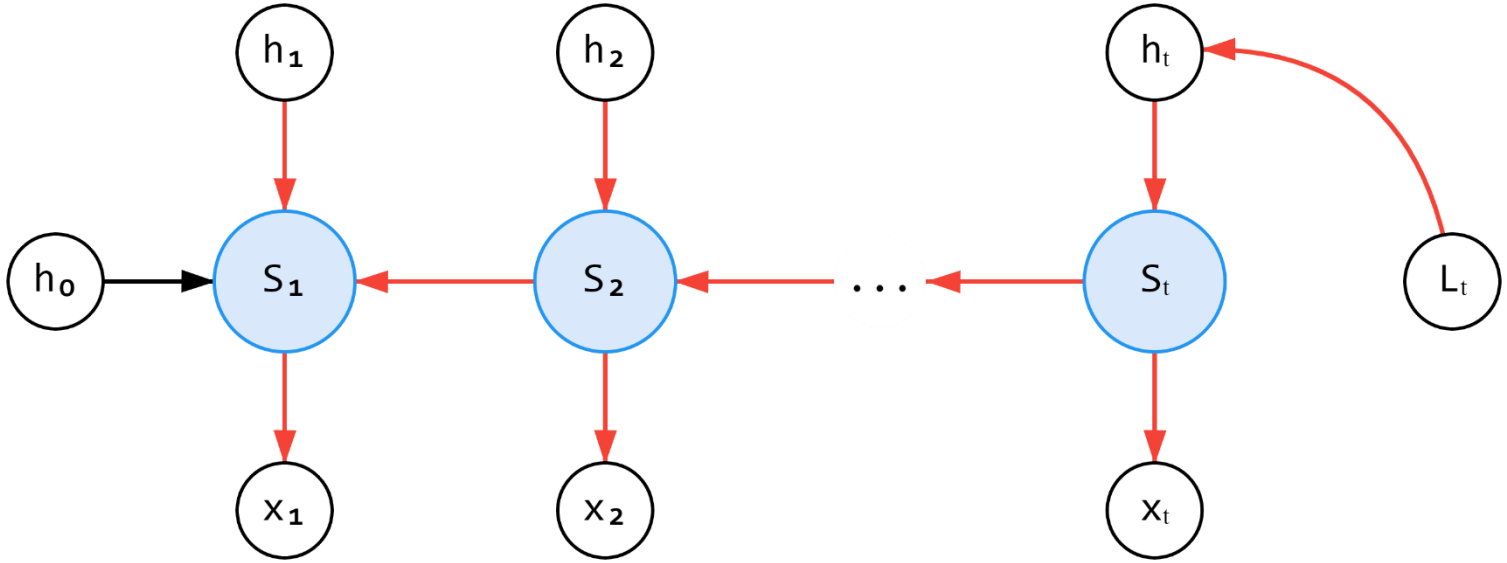
Из-за разворачивания по всем временным шагам обратное распространение ошибки является весьма трудозатратным при большом размере входной последовательности.



Обратное распространение во времени

Ошибка распространяется от последнего к первому временному шагу, при этом разворачиваются все временные шаги. Это позволяет рассчитывать ошибку для каждого временного шага и обновлять веса.

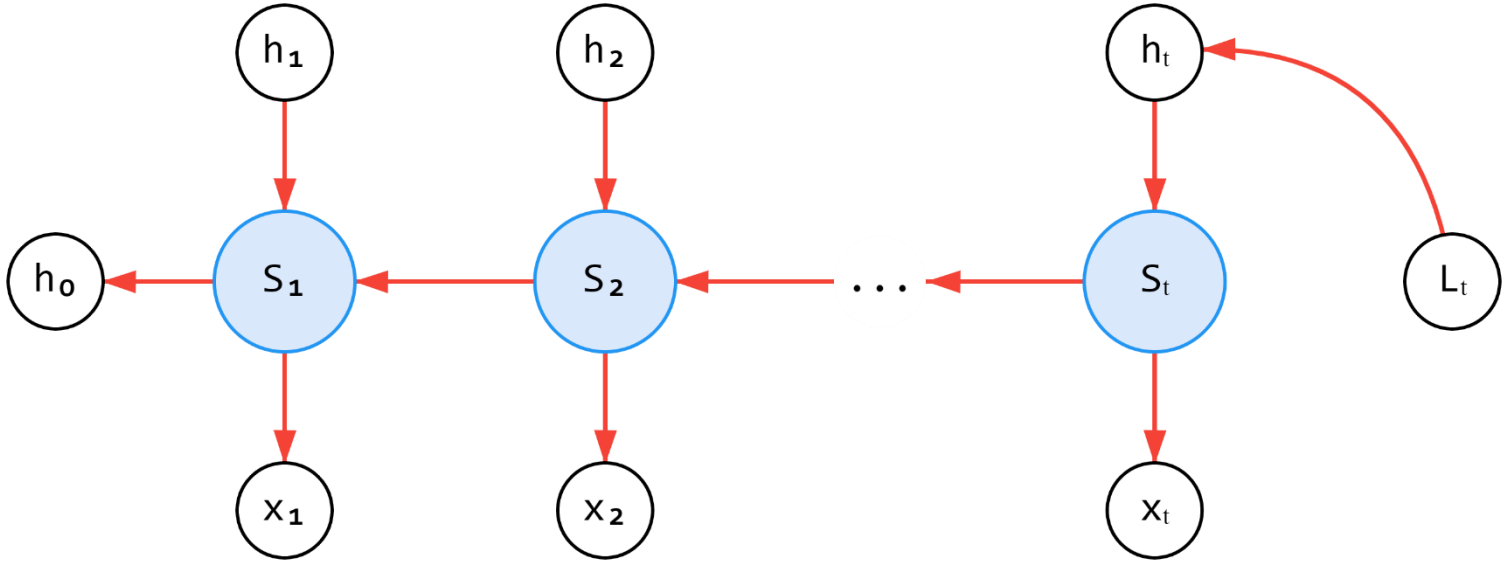
Из-за разворачивания по всем временным шагам обратное распространение ошибки является весьма трудозатратным при большом размере входной последовательности.



Обратное распространение во времени

Ошибка распространяется от последнего к первому временному шагу, при этом разворачиваются все временные шаги. Это позволяет рассчитывать ошибку для каждого временного шага и обновлять веса.

Из-за разворачивания по всем временным шагам обратное распространение ошибки является весьма трудозатратным при большом размере входной последовательности.



Проблема catastrophic forgetting

Пусть на вход дано 3 вектора $x1, x2, x3$
Развернем выход сети при $b = 0$:

$$h_3 = f\left(W_x \cdot x_3 + W_h \cdot f\left(W_x \cdot x_2 + W_h \cdot h_1\right)\right)$$

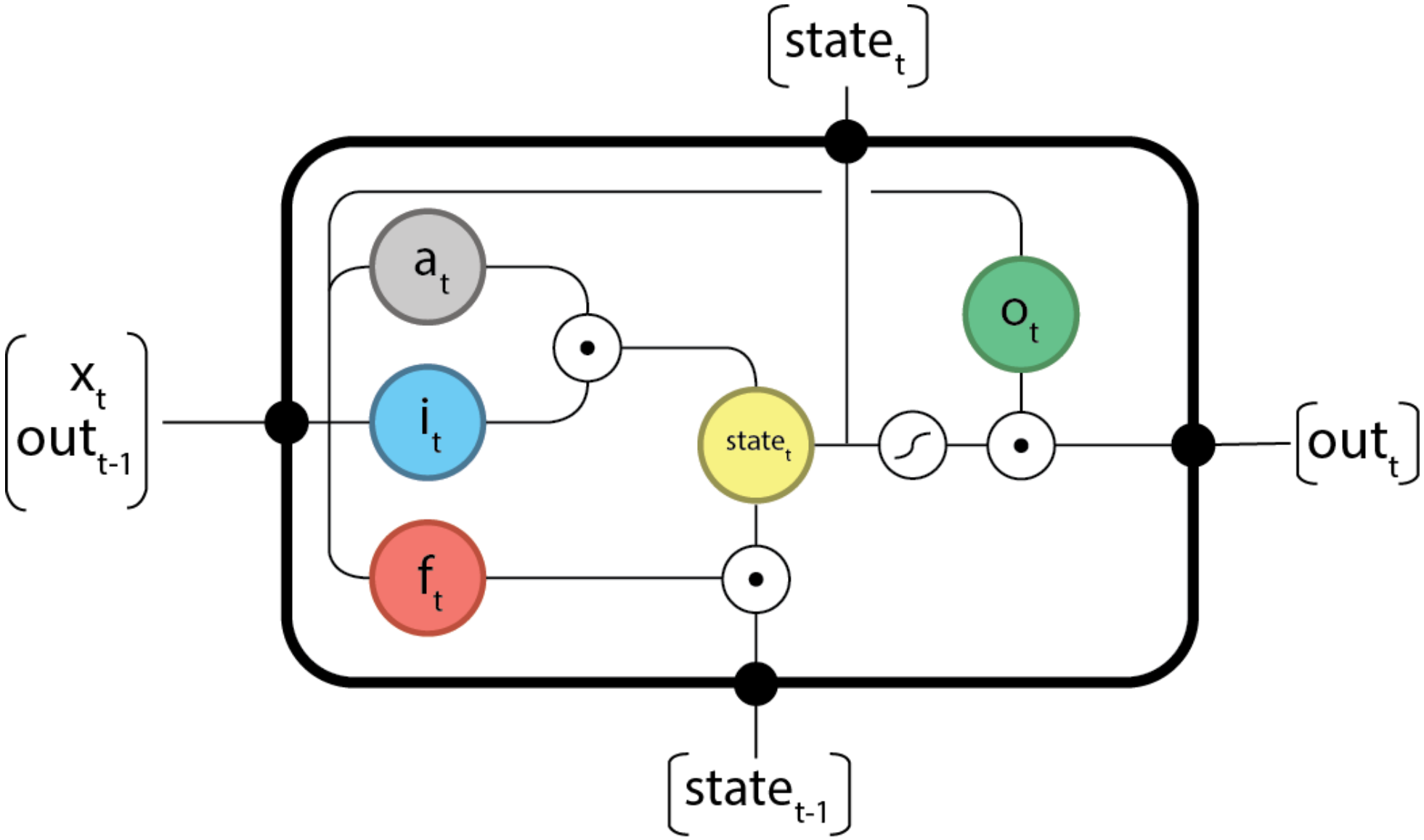
Считая, что мы работаем со скалярами:

$$\frac{\partial h_3}{\partial h_1} = f'\left(W_x \cdot x_3 + W_h \cdot f\left(W_x \cdot x_2 + W_h \cdot h_1\right)\right) \cdot W_h \cdot f'\left(W_x \cdot x_2 + W_h \cdot h_1\right) \cdot W_h$$

Если $|f'| < 1$, то $\left|\frac{\partial h_3}{\partial h_1}\right| \ll 1$.

Это означает, что предыдущий выход практически не влияет на текущее решение сети. Нейронная сеть забывает, что было ранее.

Долгая краткосрочная память (LSTM)



Долгая краткосрочная память (LSTM)

$a_t = tanh(W_a \cdot x_t + U_a \cdot h_{t-1} + b_a)$ – input activation
$\mathrm{i}_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$ – input gate (что запоминаем)
$f_t = \sigma\left(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f\right)$ – forget gate (что забываем)
$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$ – output gate

$state_t = a_t \odot i_t + f_t \odot state_{t-1}$ – внутреннее состояние
$h_t = \tanh(state_t) \odot o_t$ – выходное значение

$\odot$ – поэлементное произведение
$\frac{\partial state_t}{\partial state_{t-1}} = diag(f_t)$ – лишена проблемы катастрофического забывания, градиент не затухает

Пример работы LSTM

Весовые коэффициенты:

$$W_a = [0.45\ 0.25], U_a = [0.15], b_a = [0.2]$$
$$W_i = [0.95\ 0.8], U_i = [0.8], b_i = [0.65]$$
$$W_f = [0.7\ 0.45], U_f = [0.1], b_f = [0.15]$$
$$W_o = [0.6\ 0.4], U_o = [0.25], b_o = [0.1]$$

Входная последовательность:

$$x_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, x_1 = \begin{bmatrix} 0.5 \\ 3 \end{bmatrix}$$

Пример работы LSTM

$$a_0 = tanh(W_a \cdot x_0 + U_a \cdot h_{-1} + b_a) = tanh\left([0.45\ 0.25] \cdot \begin{bmatrix}1\\2\end{bmatrix} + [0.15] \cdot [0] + [0.2]\right) = 0.81775$$

$$\mathrm{i}_0 = \sigma(W_i \cdot x_0 + U_i \cdot h_{-1} + b_i) = \sigma\left([0.95\ 0.8] \cdot \begin{bmatrix}1\\2\end{bmatrix} + [0.8] \cdot [0] + [0.65]\right) = 0.96083$$

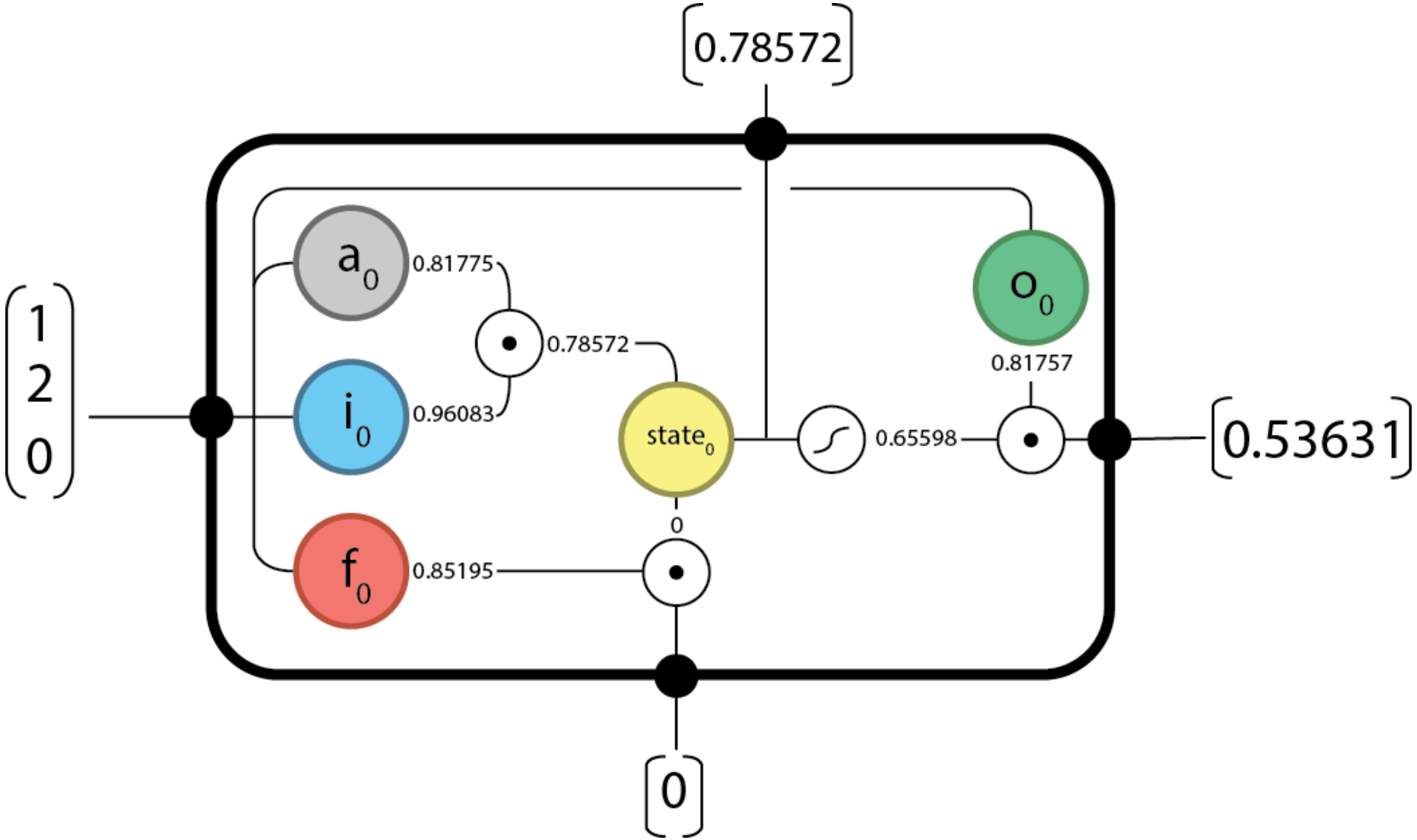
$$f_0 = \sigma\left(W_f \cdot x_0 + U_f \cdot h_{-1} + b_f\right) = \sigma\left([0.7\ 0.45] \cdot \begin{bmatrix}1\\2\end{bmatrix} + [0.1] \cdot [0] + [0.15]\right) = 0.85195$$

$$o_0 = \sigma(W_o \cdot x_0 + U_o \cdot h_{-1} + b_o) = \sigma\left([0.6\ 0.4] \cdot \begin{bmatrix}1\\2\end{bmatrix} + [0.25] \cdot [0] + [0.1]\right) = 0.81757$$

$$state_0 = a_0 \odot i_0 + f_0 \odot state_{-1} = \ 0.81775 \odot 0.96083 + 0.85195 \odot 0 = 0.78572$$

$$h_0 = \tanh(state_0) \odot o_0 = \tanh(0.78572) \odot 0.81757 = 0.53631$$

Пример работы LSTM



Пример работы LSTM

$$a_1 = tanh(W_a \cdot x_1 + U_a \cdot h_0 + b_a) = tanh\left([0.45\ 0.25] \cdot \begin{bmatrix}0.5\\3\end{bmatrix} + [0.15] \cdot [0.53631] + [0.2]\right) = 0.8498$$

$$\mathrm{i}_1 = \sigma(W_i \cdot x_1 + U_i \cdot h_0 + b_i) = \sigma\left([0.95\ 0.8] \cdot \begin{bmatrix}0.5\\3\end{bmatrix} + [0.8] \cdot [0.53631] + [0.65]\right) = 0.98118$$

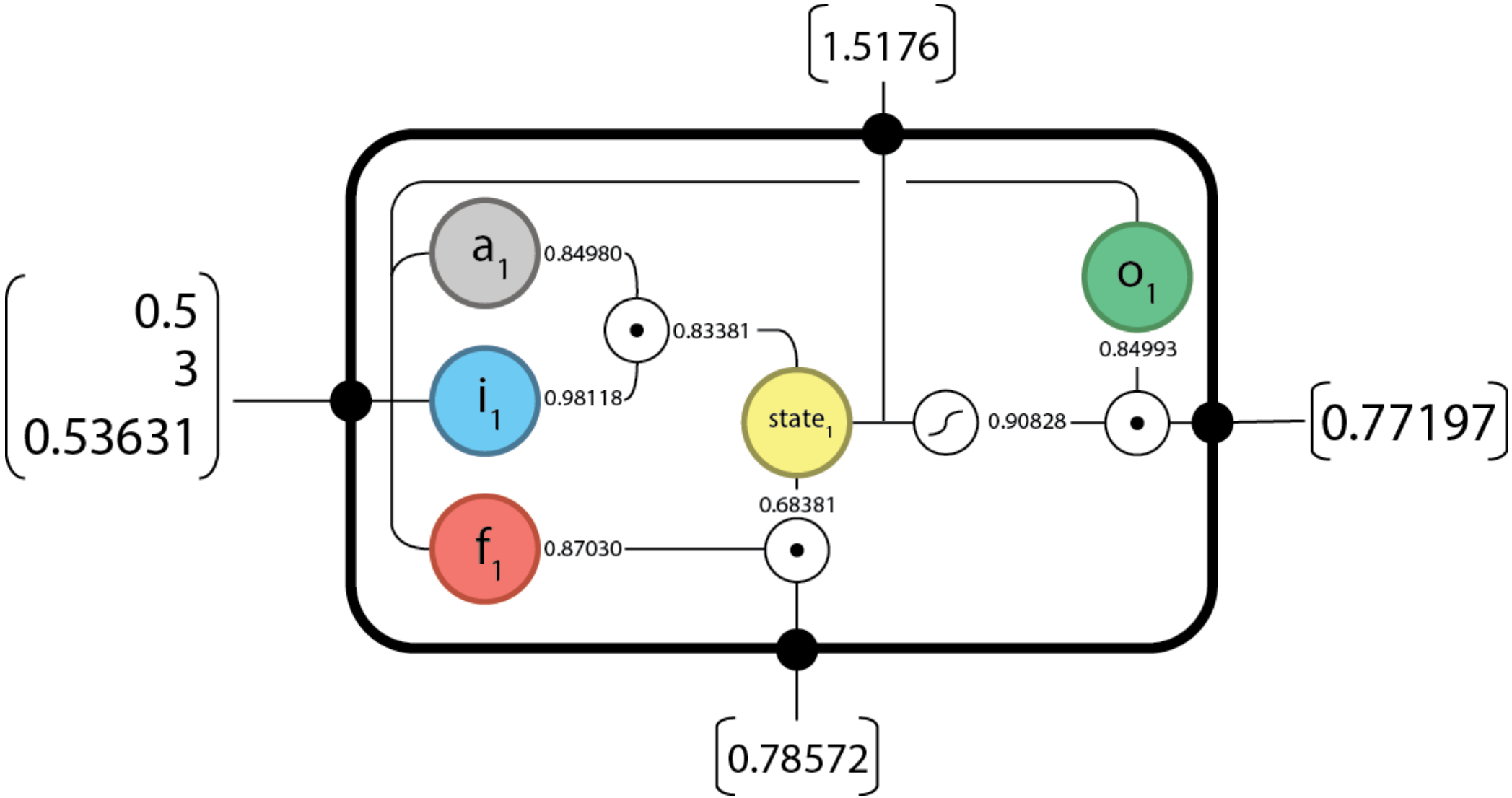
$$f_1 = \sigma\left(W_f \cdot x_1 + U_f \cdot h_0 + b_f\right) = \sigma\left([0.7\ 0.45] \cdot \begin{bmatrix}0.5\\3\end{bmatrix} + [0.1] \cdot [0.53631] + [0.15]\right) = 0.8703$$

$$o_1 = \sigma(W_o \cdot x_1 + U_o \cdot h_0 + b_o) = \sigma\left([0.6\ 0.4] \cdot \begin{bmatrix}0.5\\3\end{bmatrix} + [0.25] \cdot [0.53631] + [0.1]\right) = 0.84993$$

$$state_1 = a_1 \odot i_1 + f_1 \odot state_0 = 0.8498 \odot 0.98118 + 0.8703 \odot 0.78572 = 1.5176$$

$$h_1 = \tanh(state_1) \odot o_1 = \tanh(1.5176) \odot 0.84993 = 0.77197$$

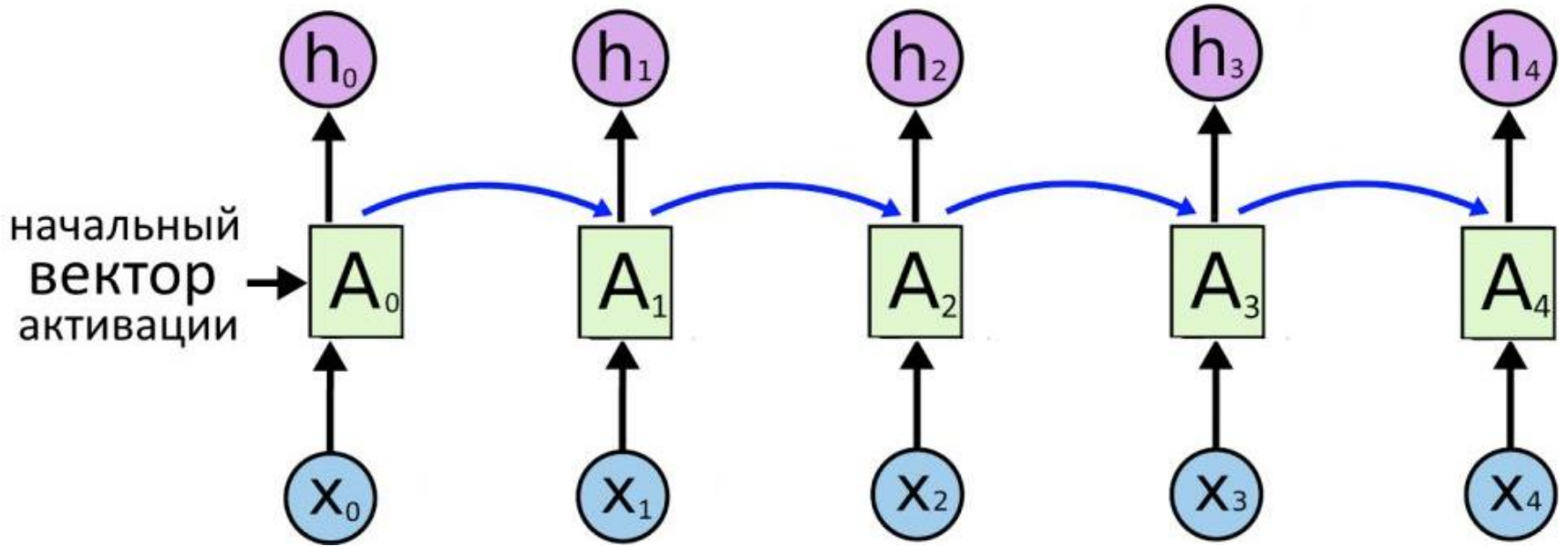
Пример работы LSTM



Двунаправленная LSTM (Bidirectional LSTM)

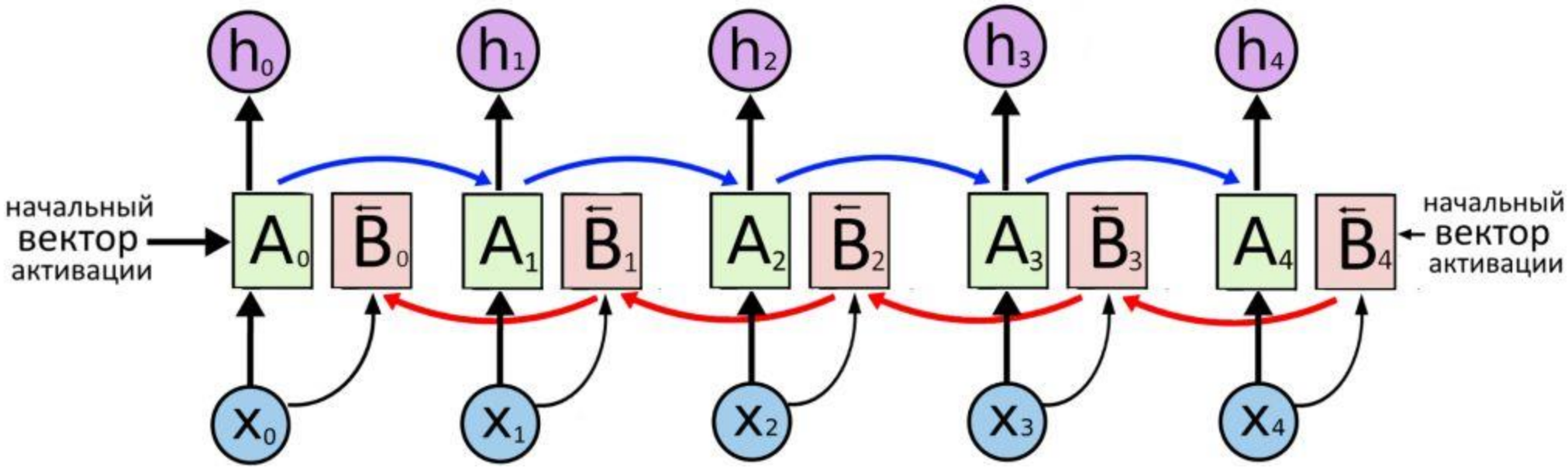
В задачах обработки текстов важен не только левый контекст словоформы, но и правый:

Александр Пушкин **родился** в Москве



Двунаправленная LSTM (Bidirectional LSTM)

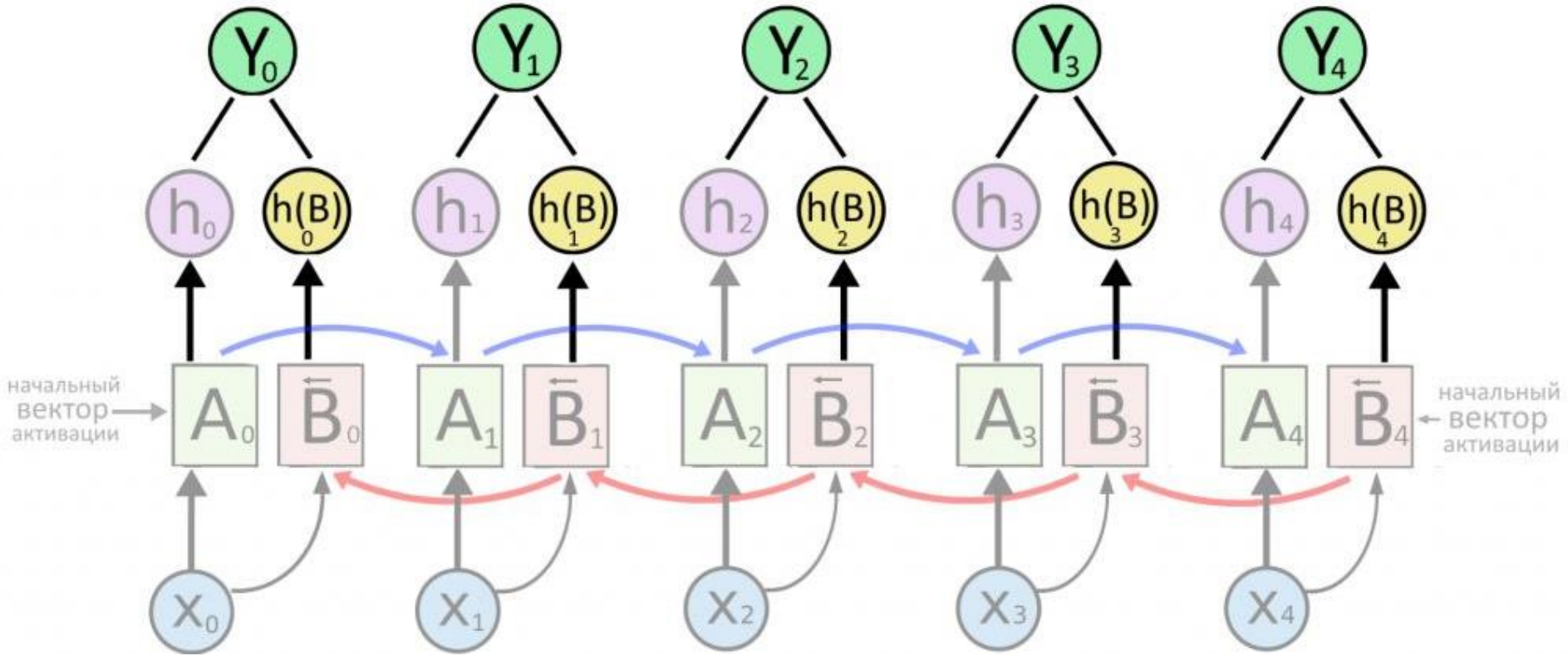
Поэтому используют Bidirectional LSTM – две параллельные LSTM, первой на вход подается исходная последовательность, второй – исходная последовательность в обратном порядке.



Двунаправленная LSTM (Bidirectional LSTM)

Поэтому используют Bidirectional LSTM – две параллельные LSTM, первой на вход подается исходная последовательность, второй – исходная последовательность в обратном порядке.

Выходы $\overrightarrow{h_t}$ и $\overleftarrow{h_t}$ конкатенируются.



Агрегация закодированного контекста

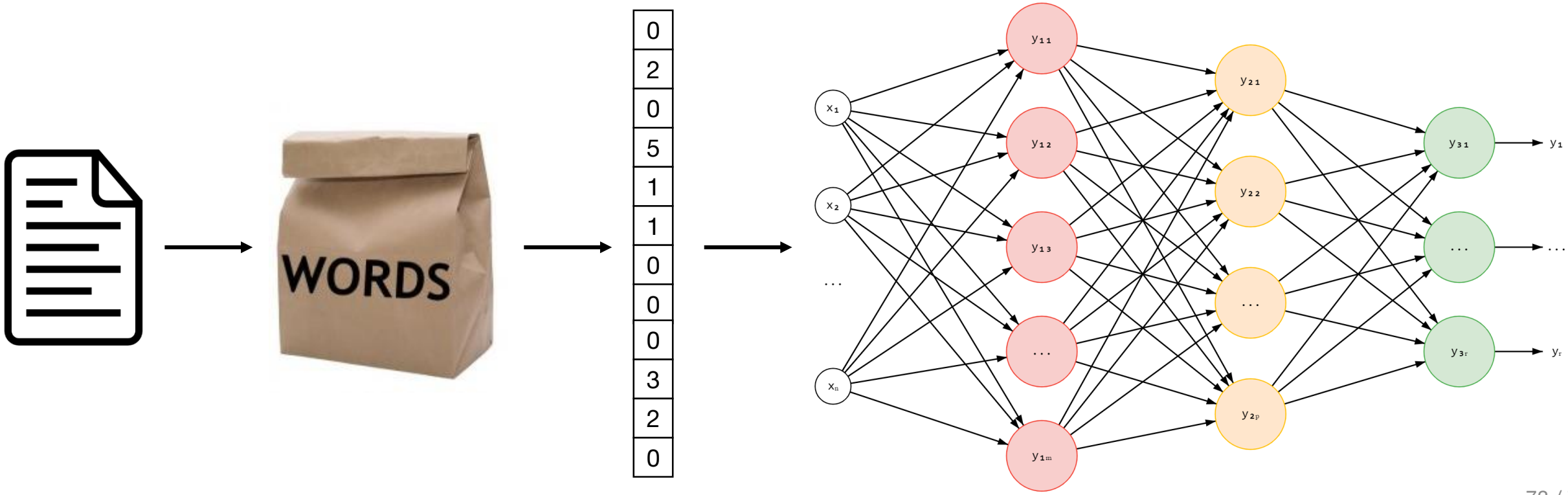
Мы закодировали исходную последовательность векторов словоформ $x_1, \dots, x_k$ в последовательность векторов $h_1, \dots, h_k$, которые учитывают контекст.

Если мы хотим классифицировать эту последовательность, то нужно построить вектор фиксированной размерности, который далее отправится в классификатор:

- последний выход h_k
- среднее (mean pooling): $\frac{\sum_{j=1}^{k} h_j}{k}$
- максимум (max pooling): $\max_j h_j$
- механизм внимания

Построение признакового пространства

Поскольку нейронные сети оперируют числами, а текст таковым не является, необходимо построить признаковое пространство. Возможным признаковым пространством может оказаться мешок слов (bag of words, BOW):



Мешок слов

Тексты из обучающей выборки разбиваются на токены (например, слова) и строится словарь W всех слов в номер этого слова.

Каждое слово словаря представляется вектором v размерности $|W|$, в котором $v_i = 1$, если i – номер слова в словаре, а иначе $v_i = 0$. Такая кодировка называется **one-hot encoding**.

0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Для построения вектора текста, можно сложить one-hot вектора каждого из слов. На каждой позиции такого вектора будет находиться количество раз, которое это слово встречается в тексте.

Мешок слов

Обучающие тексты:

1. the red dog
2. cat eats dog
3. dog eats food
4. red cat eats

Текст	the	red	dog	cat	eats	food
the red dog	1	1	1	0	0	0
cat eats dog	0	0	1	1	1	0
dog eats food	0	0	1	0	1	1
red cat eats	0	1	0	1	1	0
red dog eats and red cat eats food	0	2	1	1	2	1

Словарь слов:

- the
- red
- dog
- cat
- eats
- food

слово	the	red	dog	cat	eats	food
the	1	0	0	0	0	0
red	0	1	0	0	0	0
dog	0	0	1	0	0	0
cat	0	0	0	1	0	0
eats	0	0	0	0	1	0
food	0	0	0	0	0	1

Вопросы?