

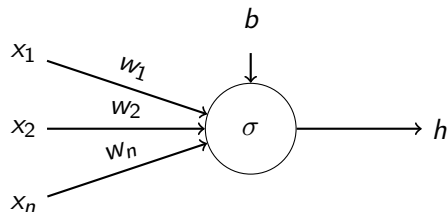
Лекция 1. Основы нейронных сетей

Бабич Павел Васильевич

ИММиКН

2026

Простейшая нейросеть – 1 нейрон



$$a(x) = b + w_1x_1 + w_2x_2 + \dots + w_nx_n = b + \sum_{i=1}^n w_ix_i$$

$$h(x) = \sigma(a(x))$$

- b — смещение
- σ — функция активации
- w_i — веса

- **Линейная регрессия:**

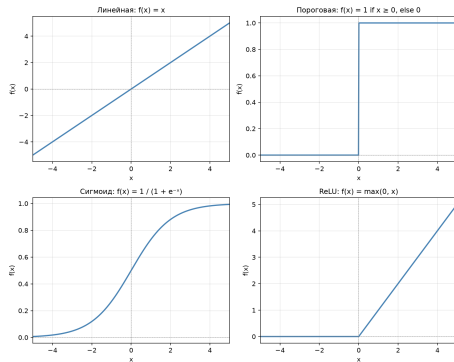
$$a(x) = b + w_1x_1 + \dots + w_nx_n$$

- **Линейный классификатор:**

$$a(x) = \text{th}(b + w_1x_1 + \dots + w_nx_n)$$

- **Логистическая регрессия:**

$$a(x) = \sigma(b + w_1x_1 + \dots + w_nx_n)$$

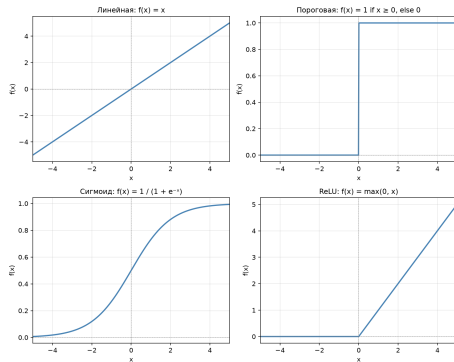


Линейная функция (linear activation function):

$$f(z) = z, \quad \frac{\partial f}{\partial z}(z) = 1$$

Пороговая функция (threshold function):

$$\text{th}(z) = I[z > 0], \quad \frac{\partial \text{th}}{\partial z}(z) = 0$$



Сигмоида (sigmoid activation function):

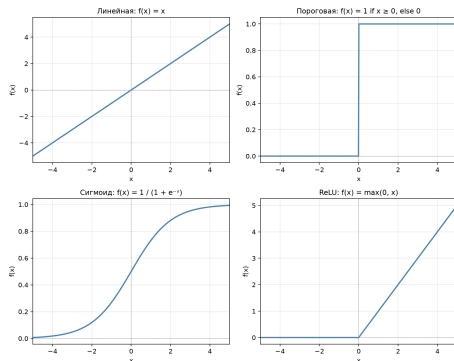
$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1)$$

$$\frac{\partial \sigma}{\partial z}(z) = \sigma(z)(1 - \sigma(z)) > 0$$

Гиперболический тангенс (hyperbolic tangent):

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

$$\frac{\partial \tanh}{\partial z}(z) = 1 - \tanh^2(z)$$



Softmax — обобщение сигмоиды в задачах классификации:

$$\text{softmax}(z_1, \dots, z_k)_i = \frac{1}{\sum_{j=1}^k e^{z_j}} (e^{z_1}, \dots, e^{z_k})$$

- сумма выходов = 1
- выходы интерпретируются как вероятности

$$[0.5, 0.5, 0.1, 0.7] \rightarrow [0.257, 0.257, 0.0172, 0.31]$$

$$[1.0, 0, 1.0, 0, -1.0] \rightarrow [0.07, 0.18, 0.07, 0.18, 0.5]$$

$$[1.0, 1.0, 1.0, 2.0, 1.0] \rightarrow [0.15, 0.15, 0.15, 0.4, 0.15]$$

Минутка кода

```
from torch.nn.functional import softmax  
x = torch.tensor([1, 2, 3], dtype=torch.float32)  
y = softmax(x)
```

Не всегда softmax нужен в явном виде (например, при NLLLoss).

Минутка кода

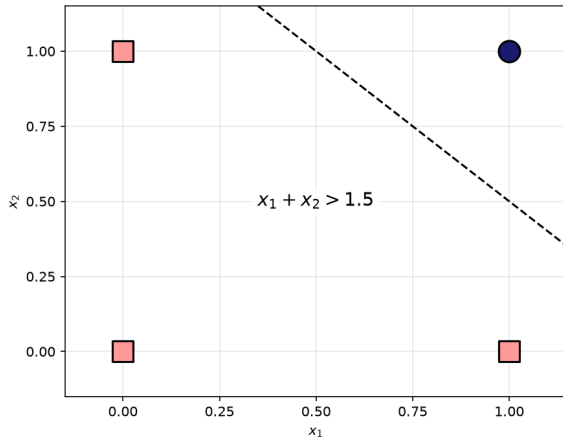
```
from torch.nn import Sigmoid
from torch.nn.functional import sigmoid

x = torch.tensor([1, 2, 3], dtype=float)
s = Sigmoid() # nn.Module
s(x) == sigmoid(x) # tensor([True, True, True])
```

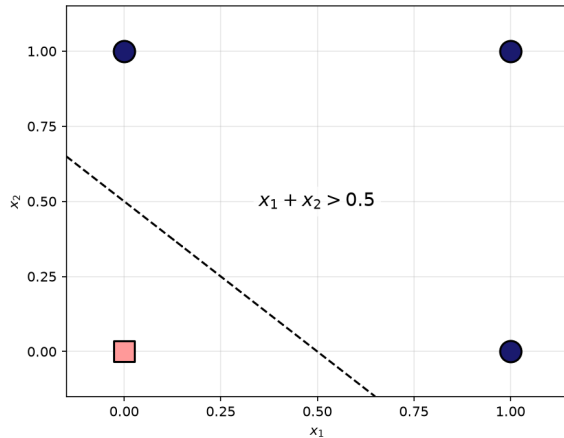
Все нужные функции активации есть: <https://docs.pytorch.org/docs>

Что может один перцептрон

Логическое И

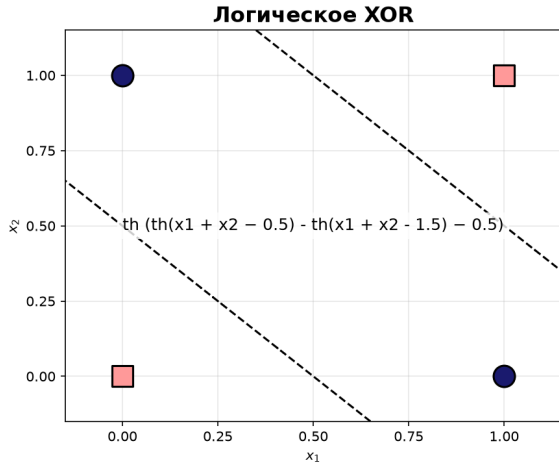


Логическое ИЛИ



Что НЕ может один перцептрон

Исключающее ИЛИ (XOR):



Что НЕ может один перцептрон

Проверим

$$\text{th}(\text{th}(0 + 0 - 0.5) - \text{th}(0 + 0 - 1.5) - 0.5) = \text{th}(0 - 0 - 0.5) = 0 \checkmark$$

$$\text{th}(\text{th}(0 + 1 - 0.5) - \text{th}(0 + 1 - 1.5) - 0.5) = \text{th}(1 - 0 - 0.5) = 1 \checkmark$$

$$\text{th}(\text{th}(1 + 0 - 0.5) - \text{th}(1 + 0 - 1.5) - 0.5) = \text{th}(1 - 0 - 0.5) = 1 \checkmark$$

$$\text{th}(\text{th}(1 + 1 - 0.5) - \text{th}(1 + 1 - 1.5) - 0.5) = \text{th}(1 - 1 - 0.5) = 0 \checkmark$$

Задание на 2 балла (получают первые пятеро студентов)

Можно ли привести такую булеву функцию трех переменных $\psi(x, y, z) : \mathbb{R}^3 \rightarrow \{0, 1\}$, которая не описывается однослойным перцептроном с двумя нейронами в скрытом слое? Если да, приведите ее. Если нет, приведите доказательство.

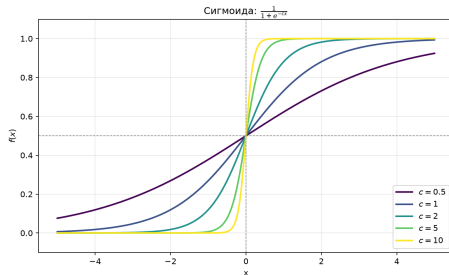
Сигмоида стремится к пороговой функции

$$\sigma_c(\sigma_c(x_1 + x_2 - 1.5) + \sigma_c(-x_1 - x_2 + 0.5) - 0.5)$$

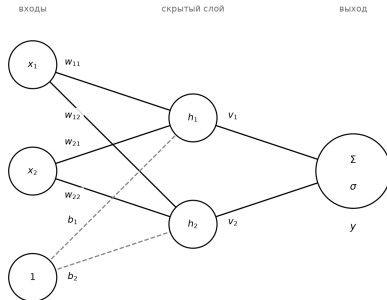
где

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

- Сигмоиду проще обучать — дифференцируемая
- Есть возможность получать «вероятности»

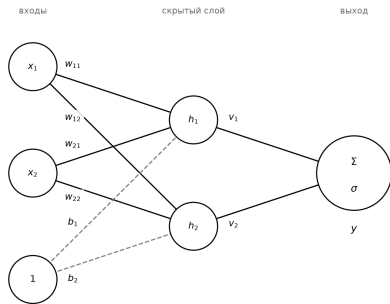


Однослойная нейронная сеть



$$a = \sigma_3(b_3 + v_1 \sigma_1(b_1 + w_{11}x_1 + w_{12}x_2) + v_2 \sigma_2(b_2 + w_{21}x_1 + w_{22}x_2))$$

Однослойная нейронная сеть (матричная форма)



$$X = [x_1, x_2, 1]^T$$

$$h = \sigma \left(\begin{bmatrix} w_{11} & w_{12} & b_1 \\ w_{21} & w_{22} & b_2 \end{bmatrix} X \right) = [h_1, h_2]^T$$

$$a = \sigma \left([v_1 \quad v_2 \quad b_3] [h_1, h_2, 1]^T \right)$$

Теорема об универсальной аппроксимации [Hornik, 1991]

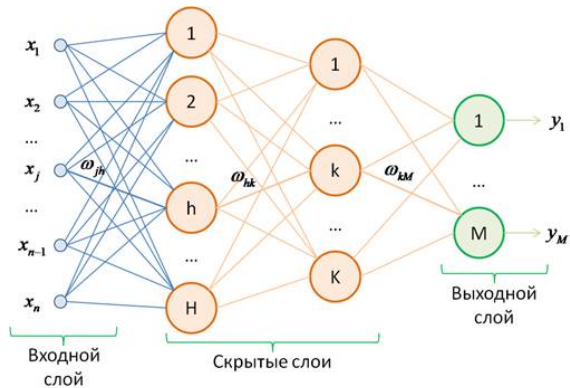
Любую непрерывную функцию можно с любой точностью приблизить нейросетью с одним скрытым слоем с сигмоидной функцией активации на скрытом слое и линейной функцией на выходном слое.

Нейросеть с одним скрытым слоем с фиксированной функцией активации в скрытом слое и линейной функцией активации на выходном слое может равномерно аппроксимировать (при увеличении числа нейронов в первом слое) любую непрерывную функцию на компактном множестве тогда и только тогда, когда функция активации **неполиномиальная**.

Но...

- неизвестно сколько (**МНОГО**) нейронов,
- неизвестно какие по масштабу (**БОЛЬШИЕ**) веса,
- неизвестно как (**НИКАК**) учить.

Полносвязная нейронная сеть



$$\sigma(W_{k_1} \phi_{k-1}(\dots W_2 \phi_1(W_1 x) \dots))$$

Минутка кода

```
class Feedforward(torch.nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Feedforward, self).__init__()
        self.input_size, self.hidden_size = input_size, hidden_size
        self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size,
                                    bias=True)
        self.relu = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(self.hidden_size, 1, bias=False)
        self.sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        hidden = self.fc1(x)
        relu = self.relu(hidden)
        output = self.fc2(relu)
        output = self.sigmoid(output)
        return output
net = Feedforward(3, 5)
x = torch.tensor([1., 2., 3.])
net(x)    # tensor([0.5376])
```

Минутка кода

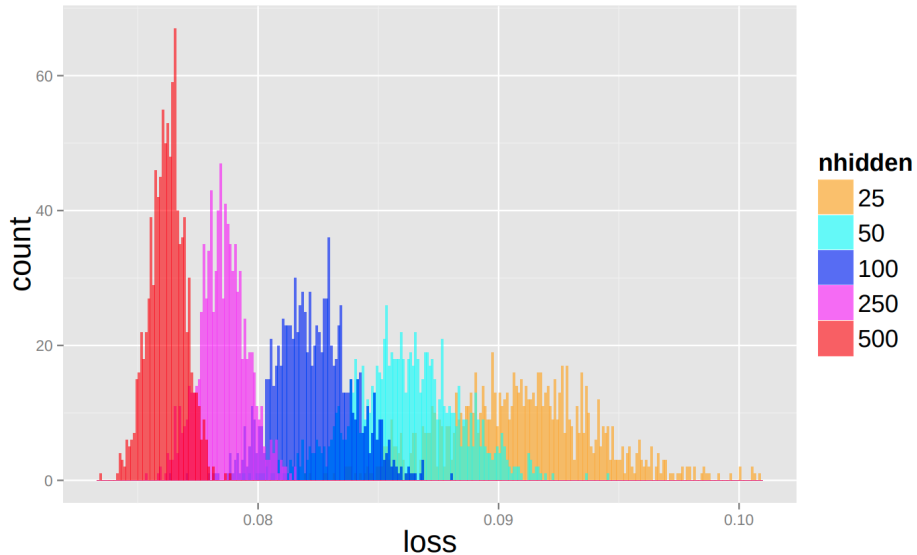
```
net
# Feedforward(
#   (fc1): Linear(in_features=3, out_features=5, bias=True)
#   (relu): ReLU()
#   (fc2): Linear(in_features=5, out_features=1, bias=False)
#   (sigmoid): Sigmoid()
# )

net.fc1.weight.data, net.fc1.bias.data, net.fc2.weight.data

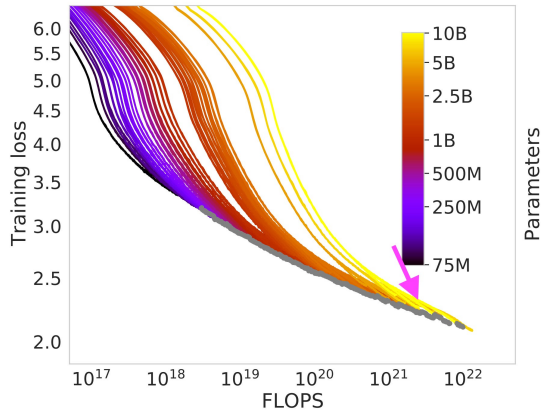
from torch import nn
net = nn.Sequential(nn.Linear(3, 5), nn.ReLU(), nn.Linear(5, 1), nn.
    Sigmoid())
net(x)
```

Как реализовать Feedforward([3, 4, 5, 3, 1])?

Зачем нужны глубокие нейронные сети



Зачем нужны глубокие нейронные сети



Larger models require **fewer samples** to reach the same performance

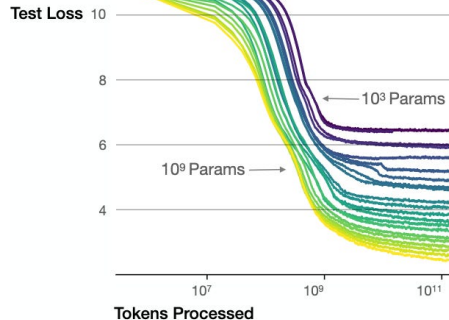


Figure 2 We show a series of language model 1 parameters (excluding embeddings).

Минимизация регуляризованного эмпирического риска:

$$\frac{1}{m} \sum_{i=1}^m (L(a(x_i|w), y_i) + \lambda R(w)) \rightarrow \min$$

Задача оптимизации невыпуклая!

Метод стохастического градиентного спуска (SGD)

$$w^{(t+1)} = w^{(t)} - \eta (\nabla_w [L(a(x_i|w), y_i) + \lambda R(w)])$$

Метод стохастического градиентного спуска (SGD)

- ❶ Случайная инициализация весов: $w^{(0)} \sim \text{norm}(0, \sigma^2)$
- ❷ Цикл по t до сходимости
 - ❶ Выбираем случайный объект x_i
 - ❷ Вычисляем градиент $\nabla_w [L(a(x_i|w), y_i) + \lambda R(w)]$
 - ❸ Адаптация весов:

$$w^{(t+1)} = w^{(t)} - \eta (\nabla_w [L(a(x_i|w), y_i) + \lambda R(w)])$$

Классификация — CrossEntropyLoss

$$L(a = (a_1, \dots, a_I), y) = -\log \frac{e^{a_y}}{\sum_{j=1}^I e^{a_j}} = -a_y + \log \sum_{j=1}^I e^{a_j}$$

Часто для оптимизации делают так:

$$-a_y + \log \sum_{j=1}^I e^{a_j} = -a_y + \max_j a_j + \log \sum_{j=1}^I e^{a_j - \max_j a_j}$$

Минутка кода

```
import torch.nn.functional as F

input = torch.randn(3, 5, requires_grad=True) # m x l
target = torch.randint(5, (3,), dtype=torch.int64)
loss = F.cross_entropy(input, target)
loss.backward()

# input:
# tensor([[ -0.0691, -1.5644, -0.3935,  0.2211,  1.3458],
#         [-2.3893,  1.2955,  0.0244, -2.0838,  1.0790],
#         [-0.6185, -0.0333,  0.2944, -0.6185,  0.5702]], requires_grad=
#         True)
# target: tensor([4, 3, 1])
# loss: tensor(2.1341, grad_fn=<NllLossBackward>)
```

Обратное распространение ошибки (Backpropagation)

Вычисление производной сложной функции

$$L_t(y, f(x, w)) \sim \sum_{k=1}^n L(f(w), f_k(w))$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial f(w)} \cdot \frac{\partial f(w)}{\partial w}$$

$$L \sim L(f_k(f_{k-1}(\dots(f_1(w))\dots)))$$

$$\nabla L = \frac{\partial L}{\partial f_k}(f_{k-1}(\dots)) \cdot \frac{\partial f_k}{\partial f_{k-1}}(f_{k-2}(\dots)) \cdots \frac{\partial f_1}{\partial w}(w)$$

Обратное распространение ошибки (Backpropagation)

Прямое распространение: вычисление ответов, значения ошибки

$$x, w \rightarrow L(w, x, f)$$

Обратное распространение: вычисление градиентов

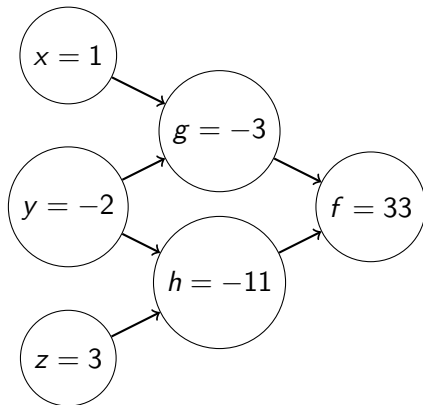
$$x, w \rightarrow \nabla f_1 \rightarrow \nabla f_2 \rightarrow \dots \rightarrow \nabla f_k \rightarrow \nabla L$$

Вычисление градиента на графе

$$f(x, y, z) = g(x, y, z) * h(x, y, z)$$

где $g(x, y, z) = x + 2y$ и $h(x, y, z) = y - 3z$.

Как проводится вычисление функции при $x = 1, y = -2, z = 3$?



Вычисление градиента на графе — «Обратный ход»

$$\frac{\partial f}{\partial g} = h(1, -2, 3) = -11, \quad \frac{\partial f}{\partial h} = g(1, -2, 3) = -3$$

$$\frac{\partial g}{\partial x} = 1, \quad \frac{\partial g}{\partial y} = 2, \quad \frac{\partial g}{\partial z} = 0$$

$$\frac{\partial h}{\partial x} = 0, \quad \frac{\partial h}{\partial y} = 1, \quad \frac{\partial h}{\partial z} = -3$$

Цепное правило:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = -11 \cdot 1 = -11$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial y} + \frac{\partial f}{\partial h} \cdot \frac{\partial h}{\partial y} = -11 \cdot 2 - 3 \cdot 1 = -25$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial h} \cdot \frac{\partial h}{\partial z} = -3 \cdot (-3) = 9$$

Минутка кода: PyTorch

```
import torch
from torch.autograd import Variable

#
x = Variable(torch.Tensor([1]), requires_grad=True)
y = Variable(torch.Tensor([-2]), requires_grad=True)
z = Variable(torch.Tensor([3]), requires_grad=True)

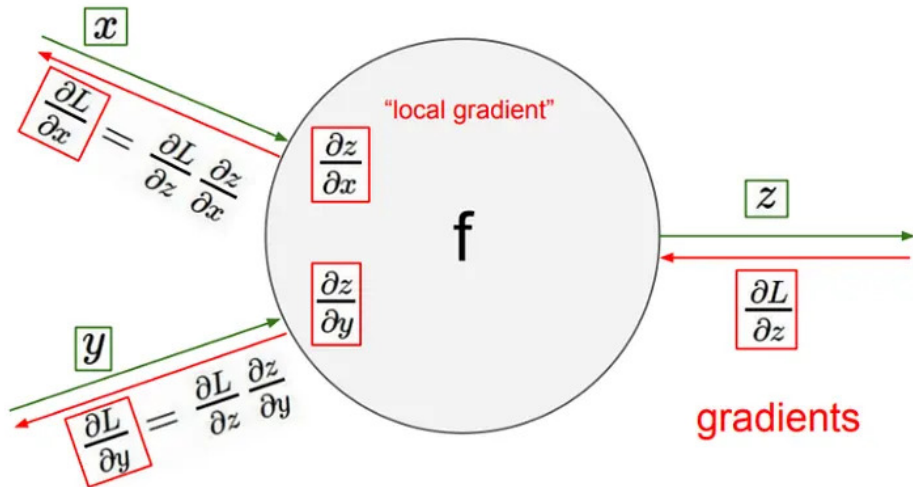
f = (x + 2 * y) * (y - 3 * z)      #      -

f.backward()                       #      -      .

x.grad, y.grad, z.grad            #

# (tensor([-11.]), tensor([-25.]), tensor([9.]))
```

Обратное распространение градиента (Backpropagation)



- Производная: $x \in \mathbb{R}, z \in \mathbb{R} \Rightarrow \frac{\partial z}{\partial x} \in \mathbb{R}$
- Градиент: $x \in \mathbb{R}^n, z \in \mathbb{R} \Rightarrow \nabla_x z = \left(\frac{\partial z}{\partial x_1}, \dots, \frac{\partial z}{\partial x_n} \right) \in \mathbb{R}^n$
- Якобиан: $x \in \mathbb{R}^n, z \in \mathbb{R}^m \Rightarrow \frac{\partial z}{\partial x} = \left(\frac{\partial z_i}{\partial x_j} \right) \in \mathbb{R}^{m \times n}$

$$z = \text{ReLU}(x) = \max(x, 0), \quad x \in \mathbb{R}^n$$

$$\underbrace{\frac{\partial z}{\partial x}}_{n \times n} \underbrace{\frac{\partial L}{\partial z}}_{n \times 1} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} = \begin{bmatrix} I[x_1 > 0] & 0 & \cdots & 0 \\ 0 & I[x_2 > 0] & \ddots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & I[x_n > 0] \end{bmatrix} \frac{\partial L}{\partial z} = \text{diag}(I[x > 0]) \frac{\partial L}{\partial z}$$

$$z = Wx, \quad z \in \mathbb{R}^m, \quad x \in \mathbb{R}^n, \quad W \in \mathbb{R}^{m \times n} \quad (1)$$

$$\frac{\partial z}{\partial x} = \frac{\partial Wx}{\partial x} = \frac{\partial \begin{bmatrix} w_{11}x_1 + \dots + w_{1n}x_n \\ \vdots \\ w_{m1}x_1 + \dots + w_{mn}x_n \end{bmatrix}}{\partial x} = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \dots & w_{mn} \end{bmatrix} = W^T \quad (2)$$

$$z = \text{ReLU}(Wx), \quad z \in \mathbb{R}^m, \quad x \in \mathbb{R}^n, \quad W \in \mathbb{R}^{m \times n}$$

$$\frac{\partial z}{\partial x} = \frac{\partial h}{\partial x} \frac{\partial z}{\partial h} \bigg|_{h=Wx} = W_{n \times m}^T \underbrace{\text{diag}(\nabla \text{ReLU}(Wx))}_{I[Wx > 0]}_{m \times m}$$

Несколько слоёв:

$$z = \text{ReLU}(W_k (\dots W_2 \text{ReLU}(W_1 x)))$$

$$\frac{\partial z}{\partial x} = W_1^T \text{diag}(\nabla \text{ReLU}(Wx)) \cdot W_2^T \text{diag}(\nabla \text{ReLU}(\cdot)) \cdots W_k^T \text{diag}(\nabla \text{ReLU}(\cdot))$$

Производная функции ошибки ПО ВЕСАМ

$$L = L(z), \quad z = Wx, \quad z \in \mathbb{R}^m, \quad x \in \mathbb{R}^n, \quad W \in \mathbb{R}^{m \times n}$$

$$\nabla_W L = \frac{\partial L}{\partial W} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial W} = \frac{\partial L}{\partial z} \cdot x^T \in \mathbb{R}^{m \times n}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial z_j} \cdot x_i$$

Прохождение градиента через нейрон

$$z = \sigma(b + w_1x_1 + \dots + w_nx_n)$$

$$\frac{\partial L}{\partial x_i} = \sigma'(z) \cdot w_i \cdot \frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial w_i} = \sigma'(z) \cdot x_i \cdot \frac{\partial L}{\partial z}$$

При прохождении градиента через нейрон происходит умножение на производную функции активации.

Проблема затухания градиента

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

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

$$\text{ReLU}(z) = \max(0, z)$$

$$\frac{\partial \text{ReLU}}{\partial z}(z) = I[z > 0]$$

ReLU = Rectified Linear Unit

