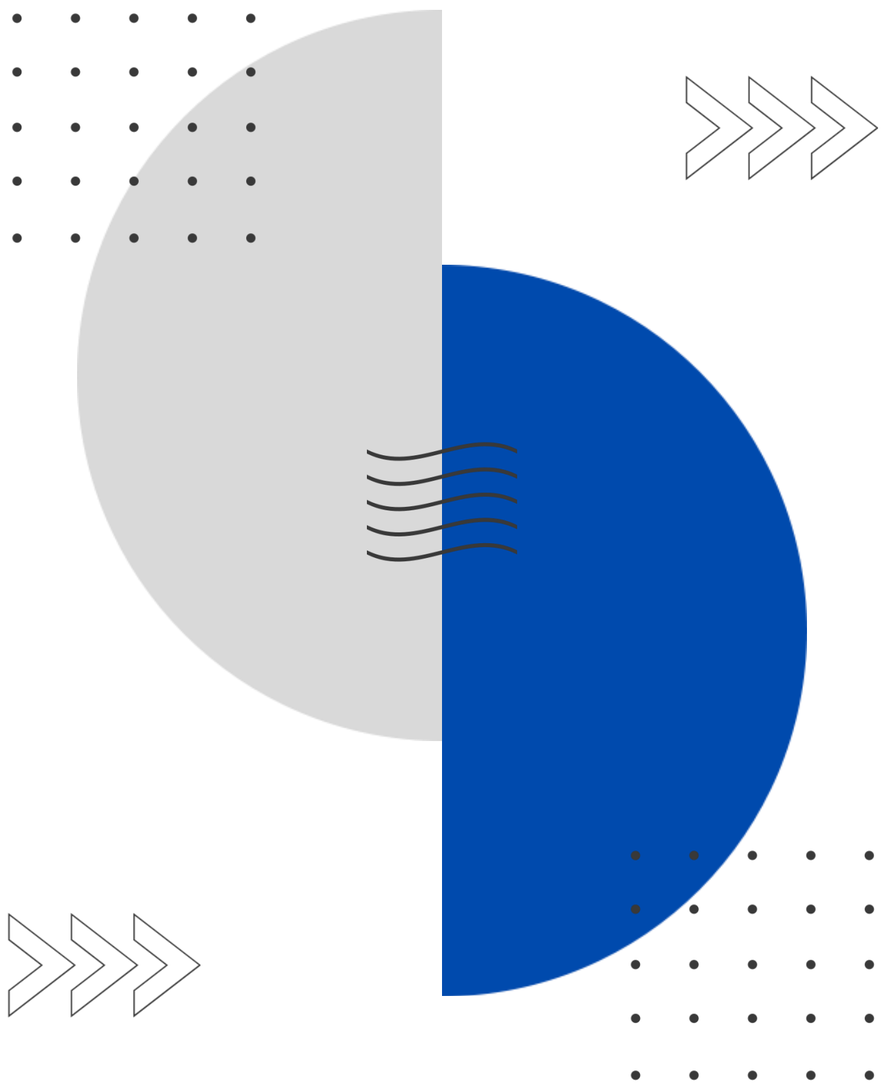


# ВВЕДЕНИЕ В МАШИННОЕ ОБУЧЕНИЕ

Лекция №8

# Линейная модель

Where we are...

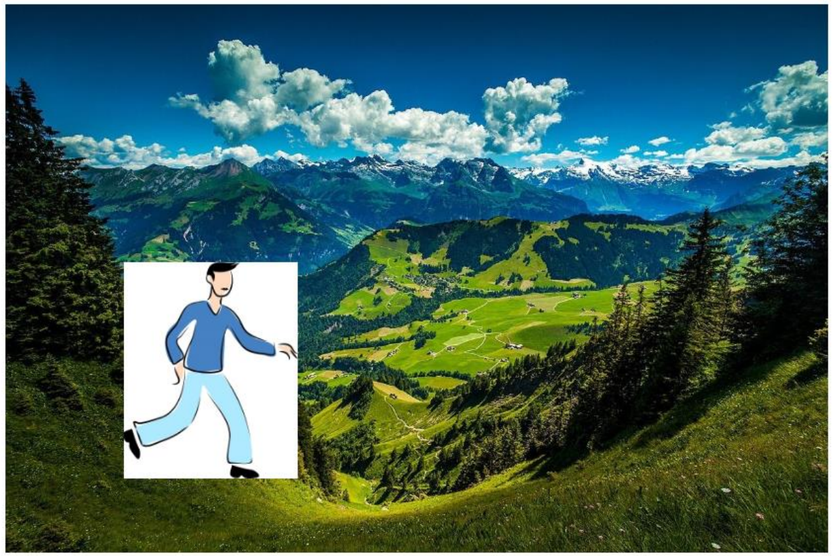
$$s = f(x; W) = Wx$$ Linear score function

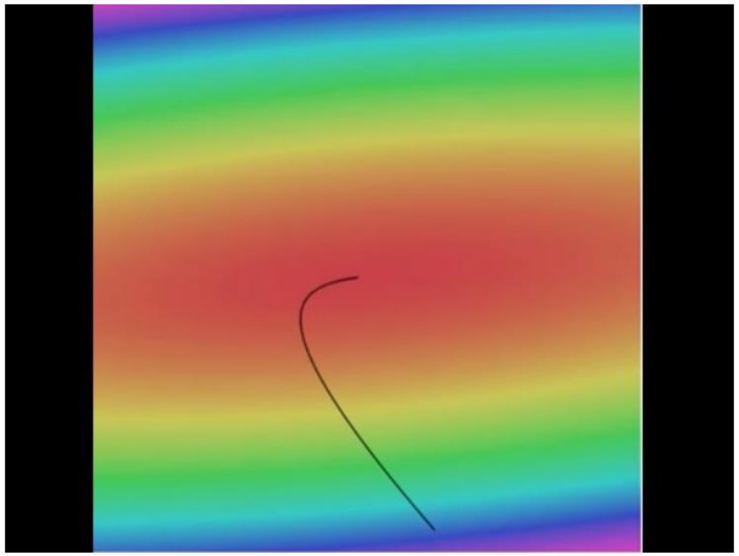
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM loss (or softmax)

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$ data loss + regularization

How to find the best W?

# Оптимизация линейной модели





```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

# Численная производная

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

**Numerical gradient**: slow :(, approximate :(, easy to write :)
**Analytic gradient**: fast :), exact :), error-prone :(

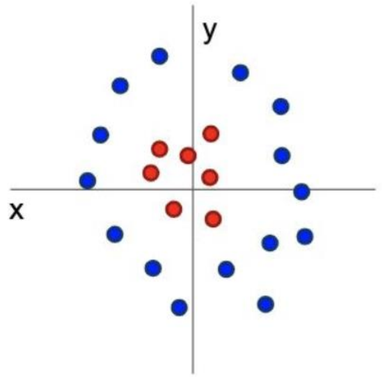
In practice: Derive analytic gradient, check your implementation with numerical gradient

# Проблемы линейных моделей

## Visual Viewpoint

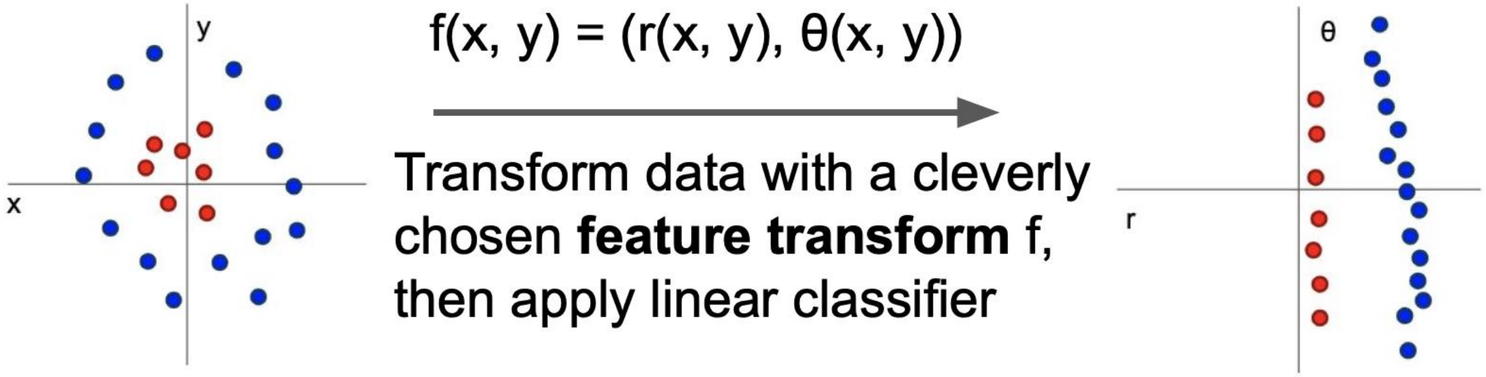

Linear classifiers learn one template per class

## Geometric Viewpoint

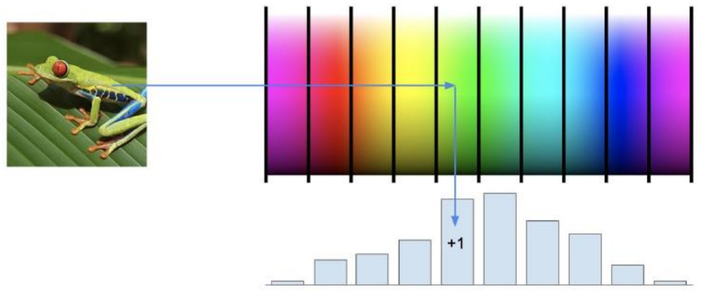



Linear classifiers can only draw linear decision boundaries

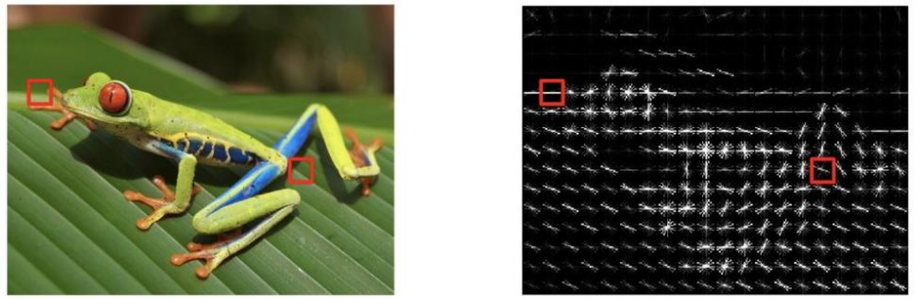
# Одно из решений – преобразование признаков



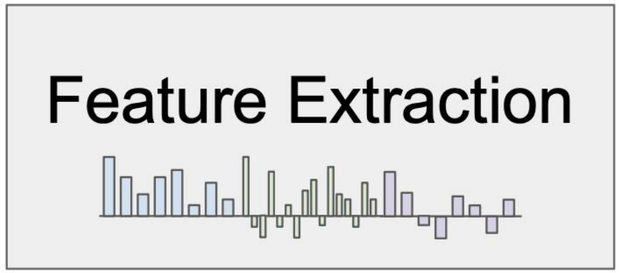

Color Histogram




Histogram of Oriented Gradients (HoG)



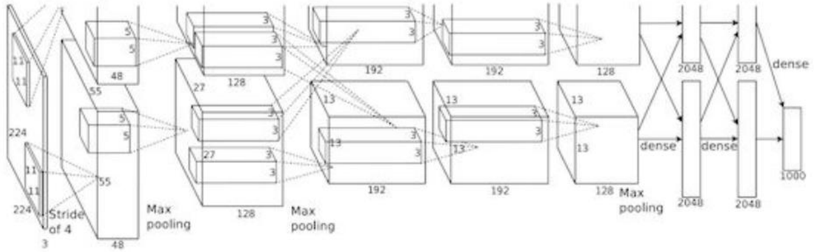
# Признаки vs Нейронная сеть




f

**10** numbers giving scores for classes

training

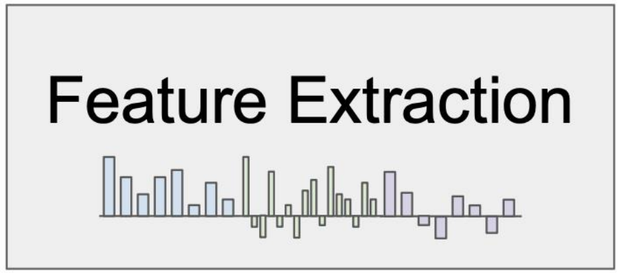



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

**10** numbers giving scores for classes

training

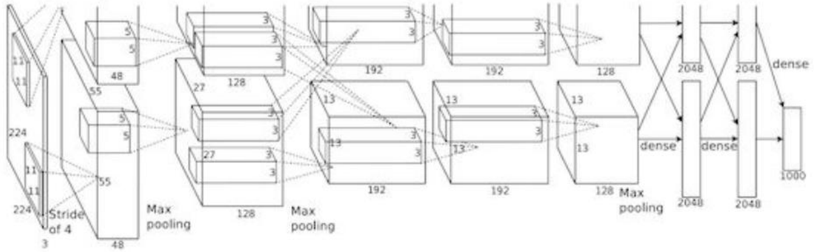
# Признаки vs Нейронная сеть




f

**10** numbers giving scores for classes

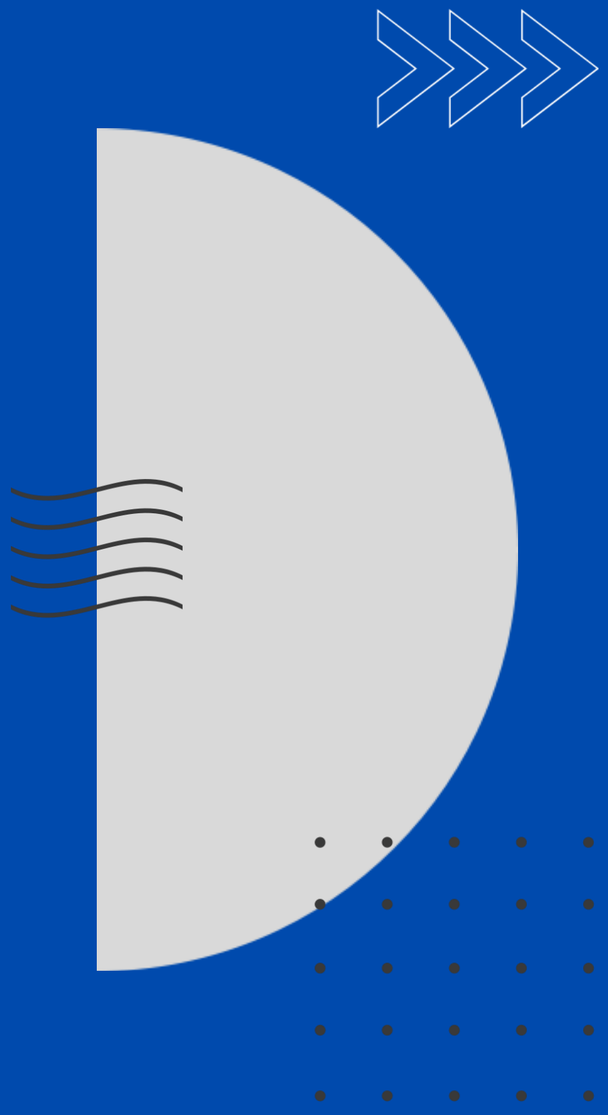
training




Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

**10** numbers giving scores for classes

training



01

# Нейронная сеть

# Нейронная сеть

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Нейронная сеть

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network

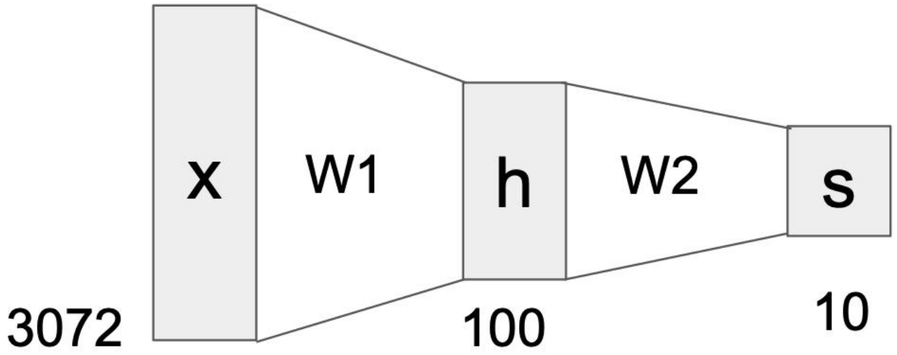
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2}$$

# Нейронная сеть

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$




$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

# Нейронная сеть

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

The function $\max(0, z)$ is called the **activation function.**

**Q:** What if we try to build a neural network without one?

$$f = W_2 W_1 x$$

# Нейронная сеть

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

The function $\max(0, z)$ is called the **activation function.**

**Q:** What if we try to build a neural network without one?

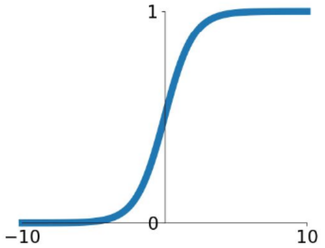
$$f = W_2 W_1 x \qquad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, f = W_3 x$$

**A**: We end up with a linear classifier again!

# Функции активации

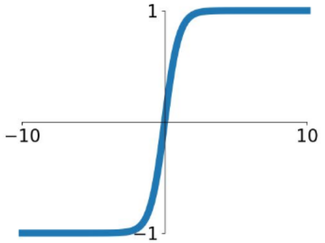
**Sigmoid**

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



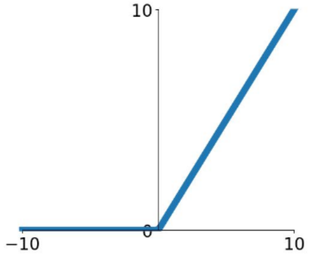

**tanh**

$$\tanh(x)$$



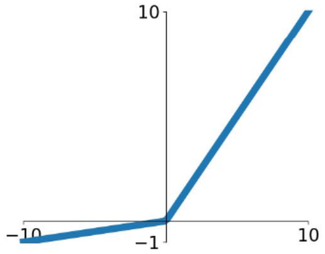

**ReLU**

$$\max(0, x)$$




**Leaky ReLU**

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

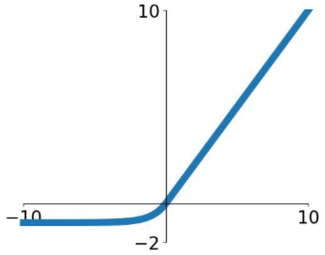



**Maxout**

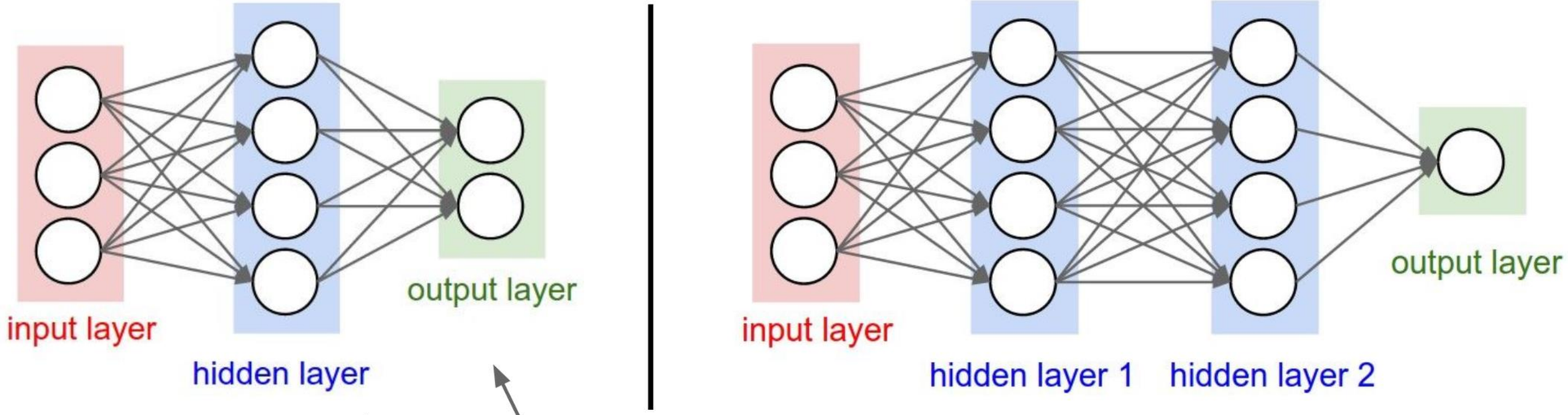
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

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



# Архитектура нейронных сетей

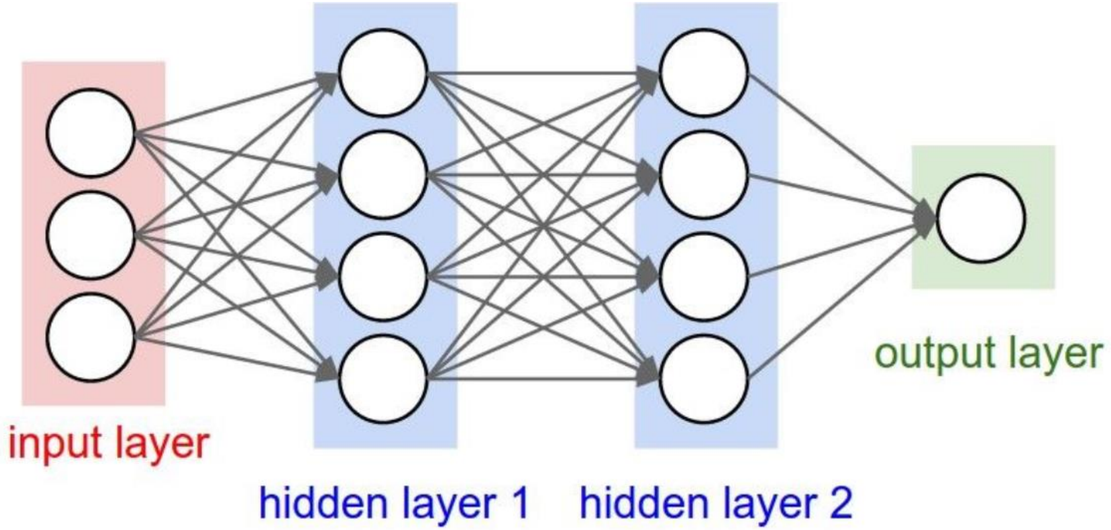



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

**“Fully-connected” layers**

# Прямое распространение (feed-forward)



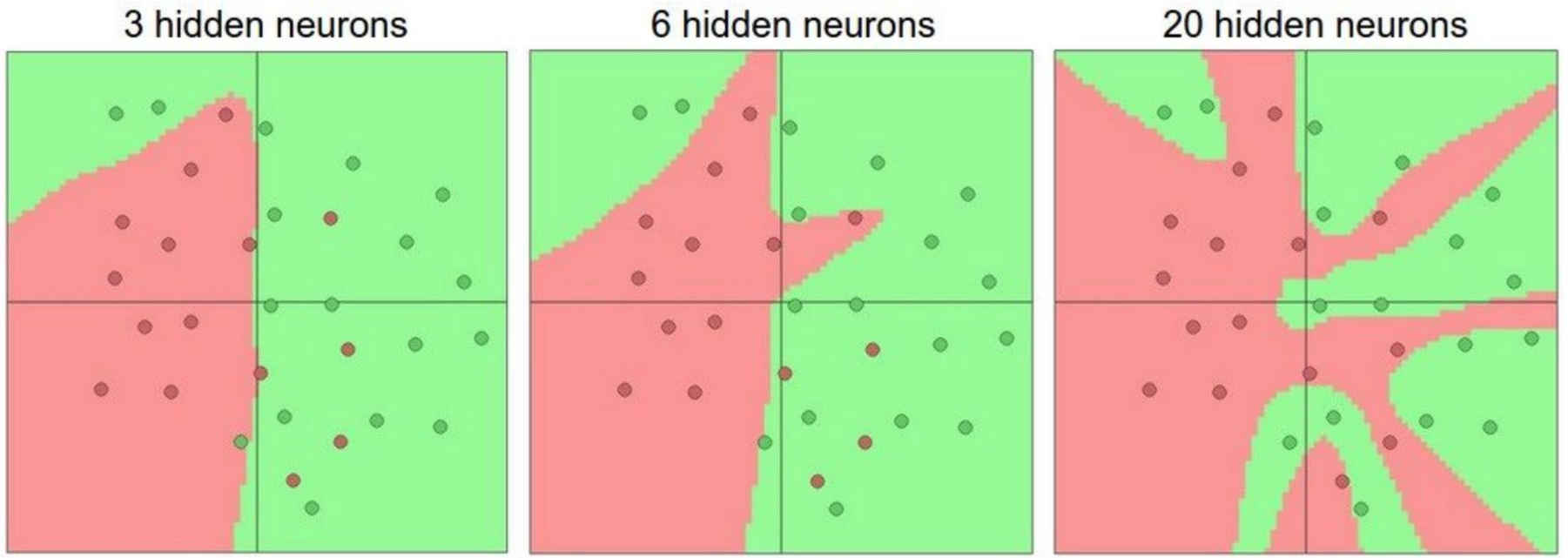

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

# Нейронная сеть в 20 строк

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

*Fei-Fei Li & Justin Johnson & Serena Yeung | cs231n*

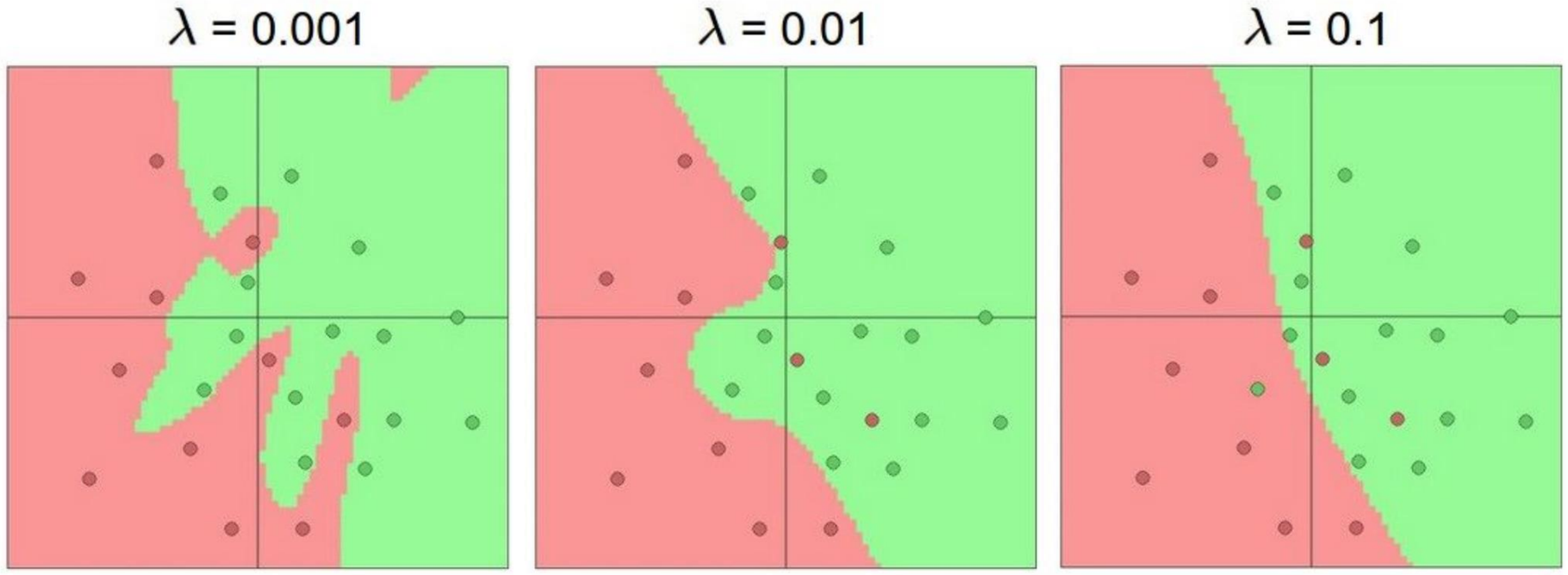
# Сложность нейронной сети



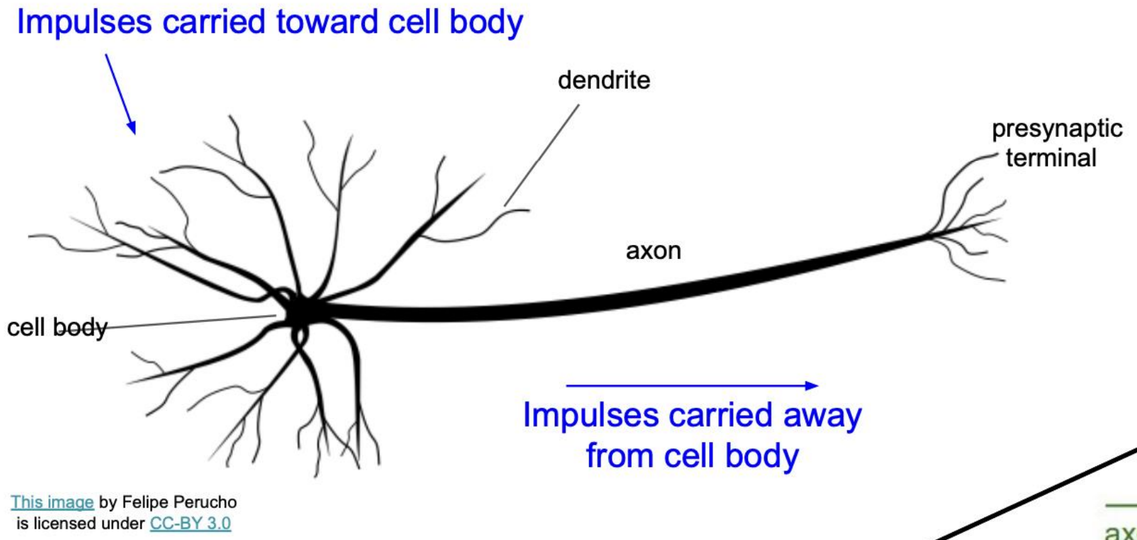

more neurons = more capacity

# Сложность нейронной сети

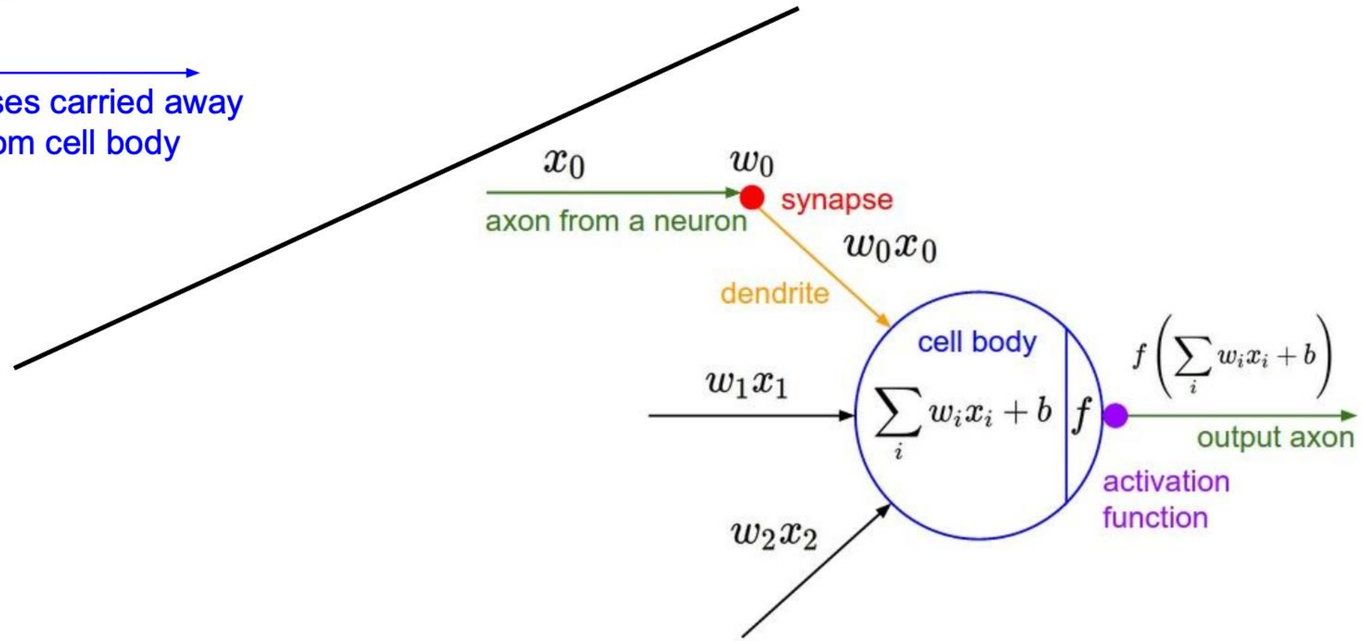
Do not use size of neural network as a regularizer. Use stronger regularization instead:



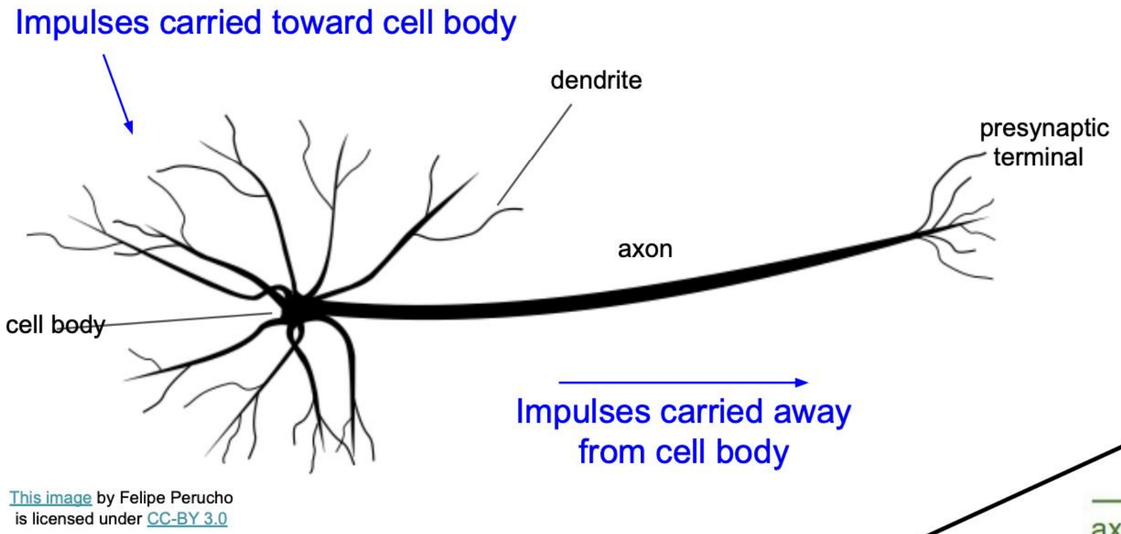
# Перцептрон



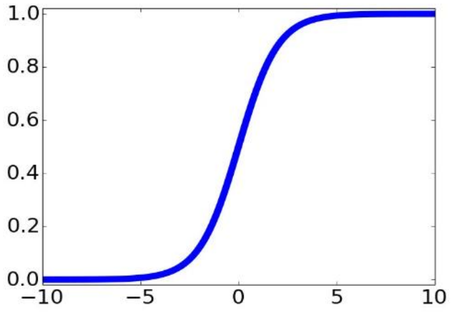


# Перцептрон

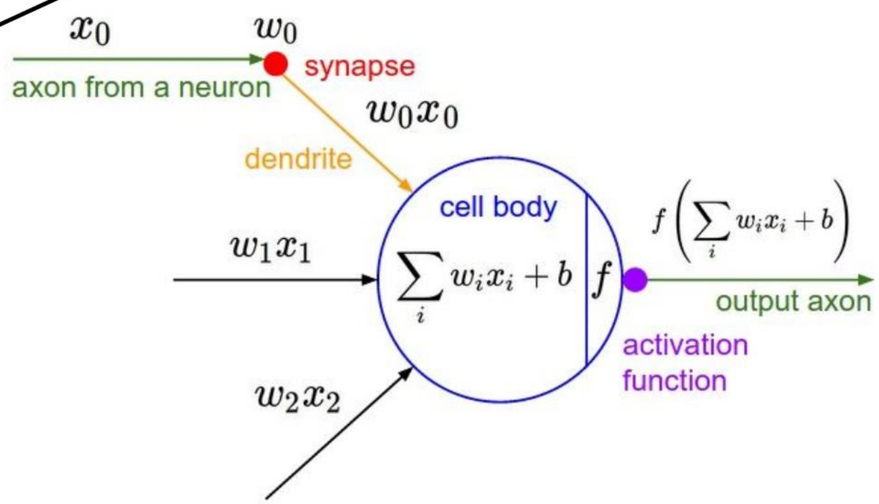




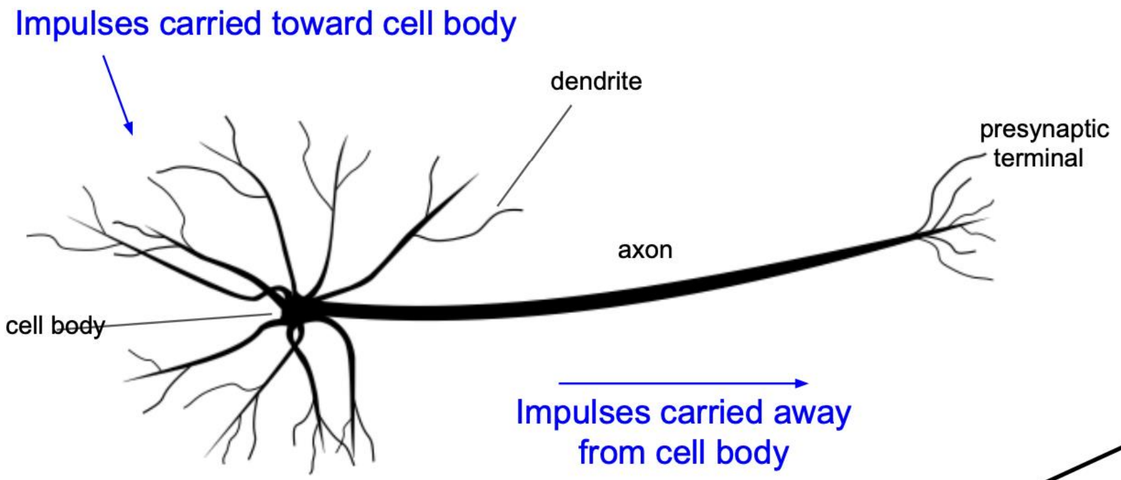




sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

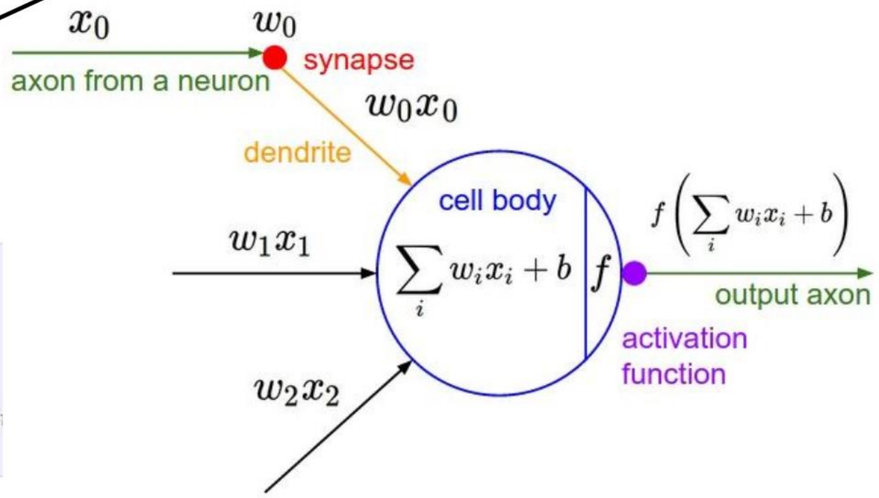


# Перцептрон



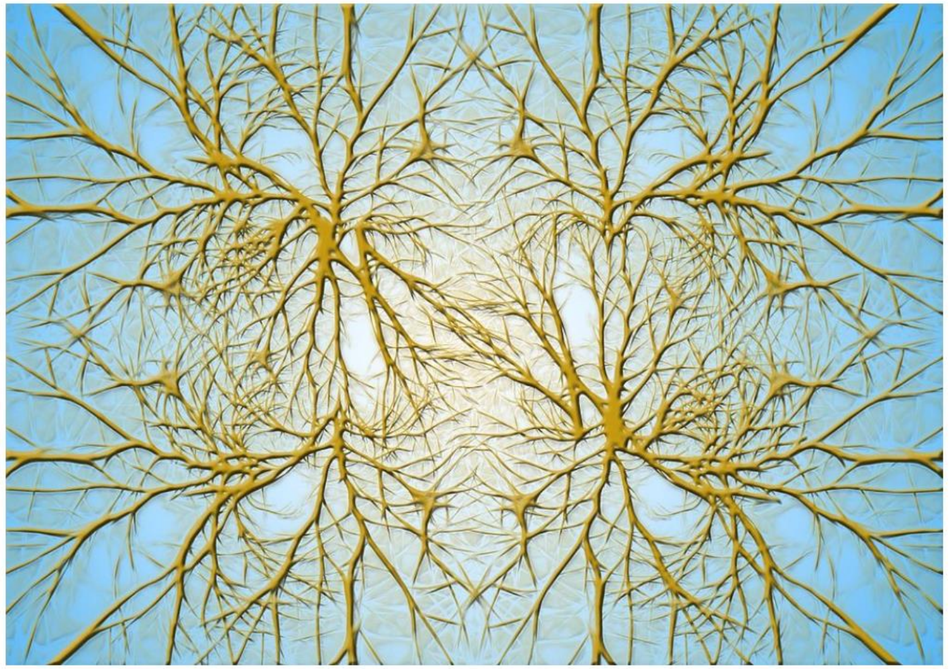



```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation funct
    return firing_rate
```



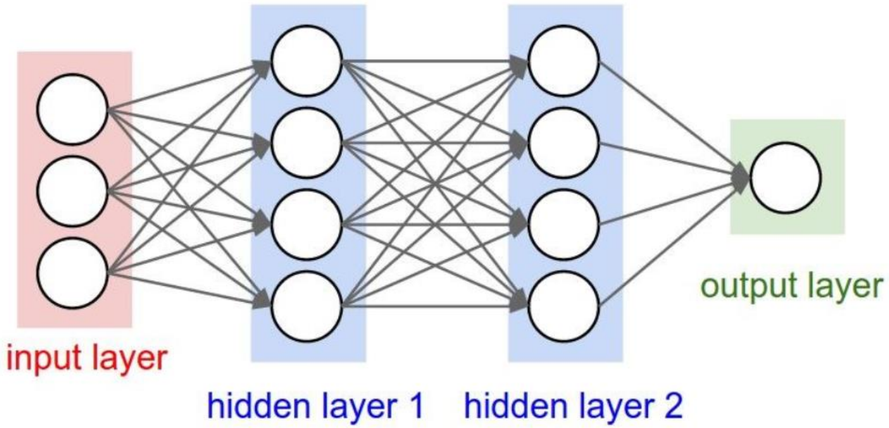
# Искусственная нейронная сеть

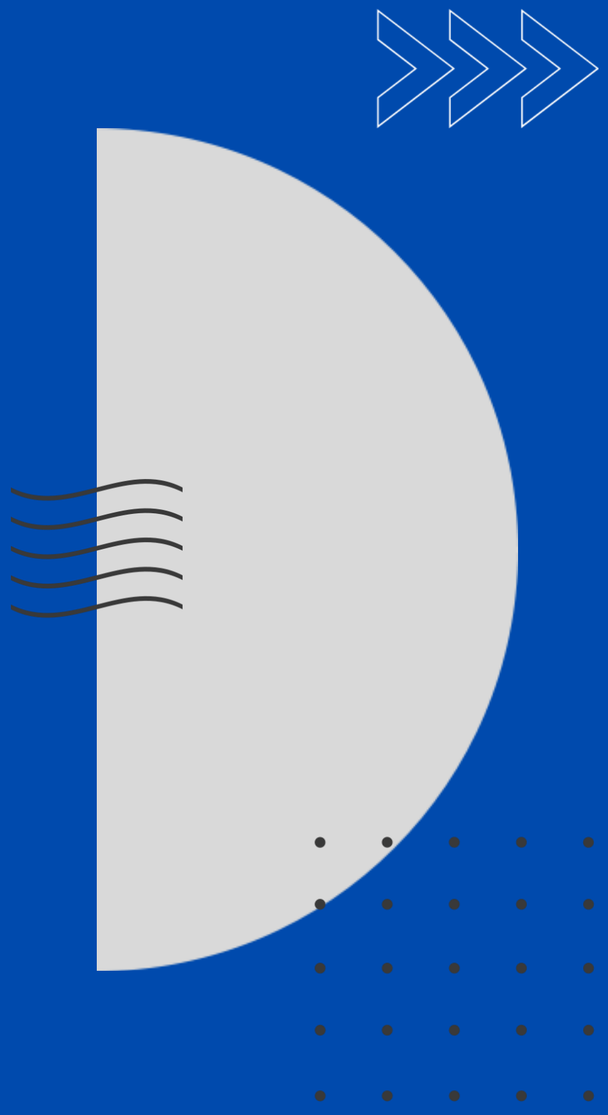
Biological Neurons:
Complex connectivity patterns



This image is CC0 Public Domain

Neurons in a neural network:
Organized into regular layers for computational efficiency





02

# Backpropagation

# Проблемы обучения нейронной сети

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM Loss on predictions

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$ then we can learn $W_1$ and $W_2$

# Выведем $\nabla_W L$

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

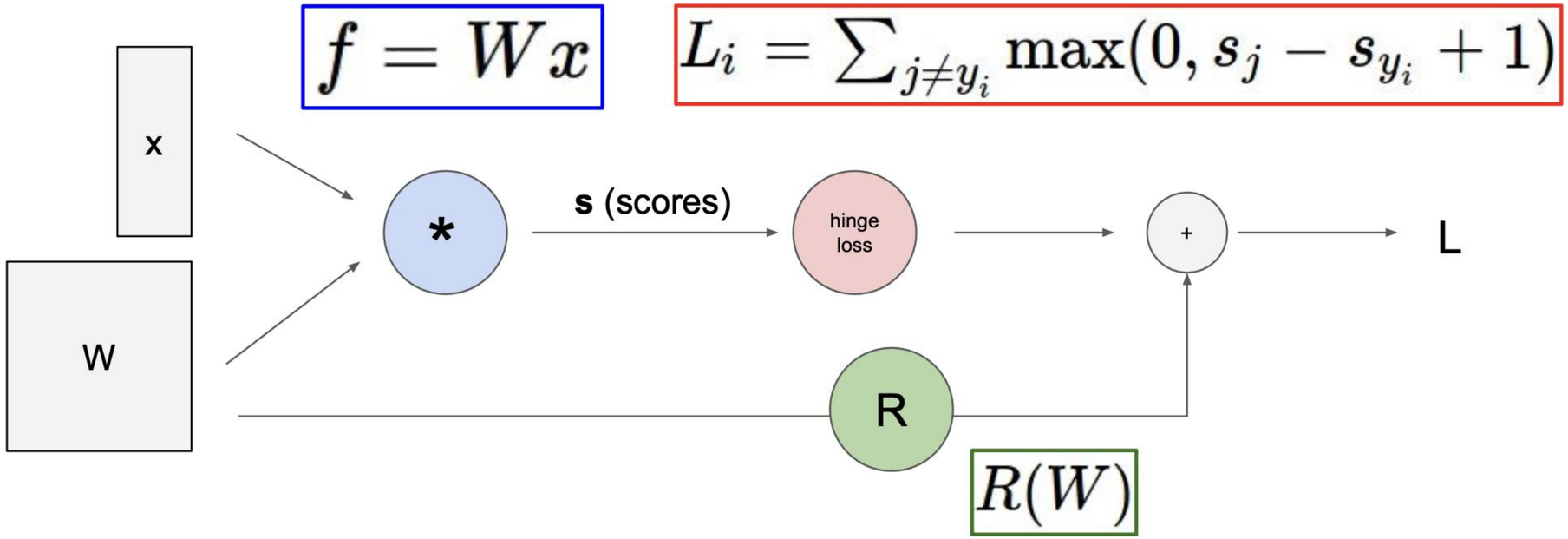
$$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

**Problem**: Very tedious: Lots of matrix calculus, need lots of paper

**Problem**: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

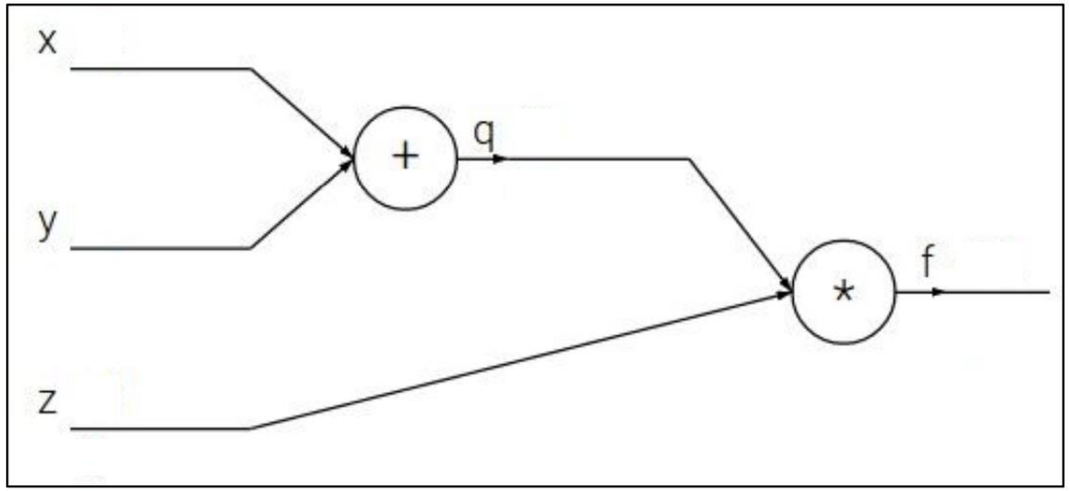
**Problem**: Not feasible for very complex models!

# Концепция получше – граф и обратное распространение ошибки (backpropagation)



# Обратное распространение ошибки (backpropagation)

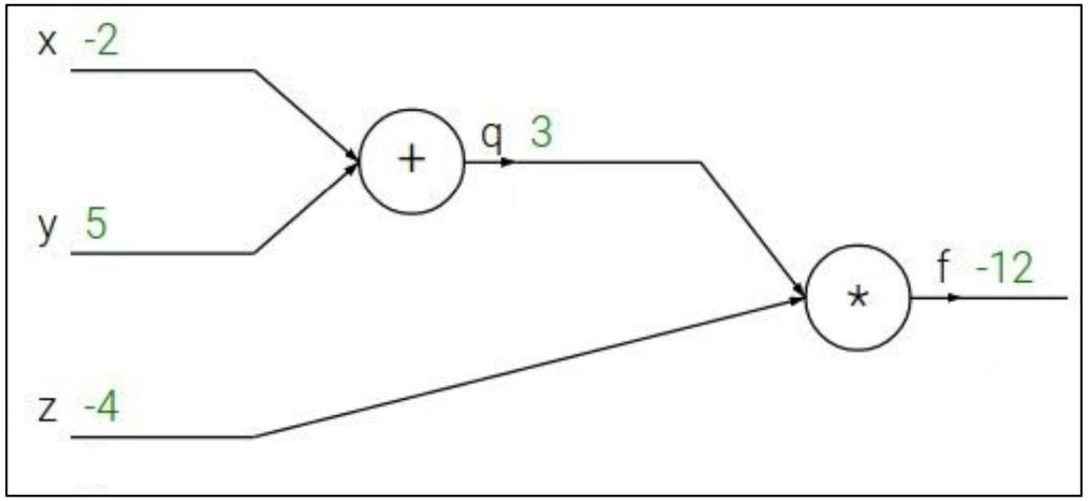
$$f(x, y, z) = (x + y)z$$



# Обратное распространение ошибки (backpropagation)

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



# Обратное распространение ошибки (backpropagation)

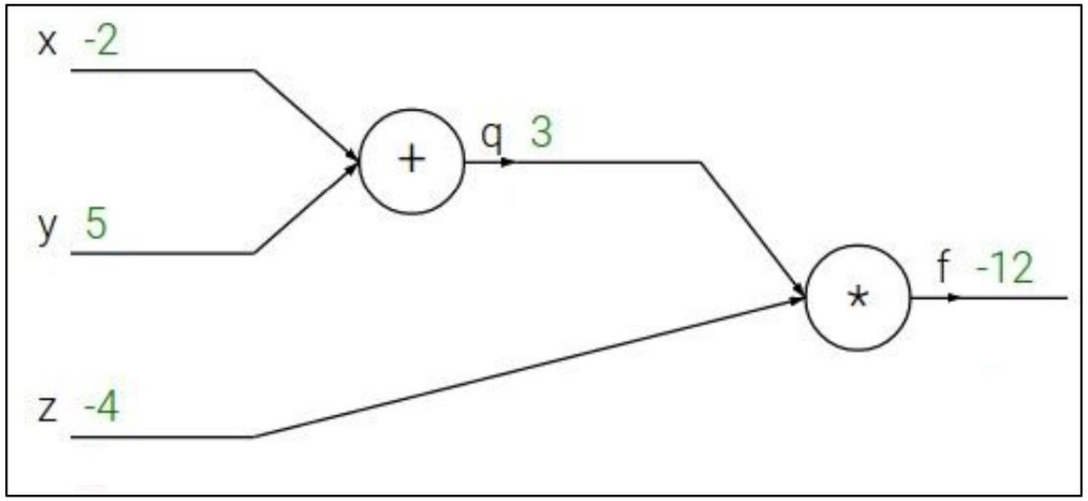
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Обратное распространение ошибки (backpropagation)

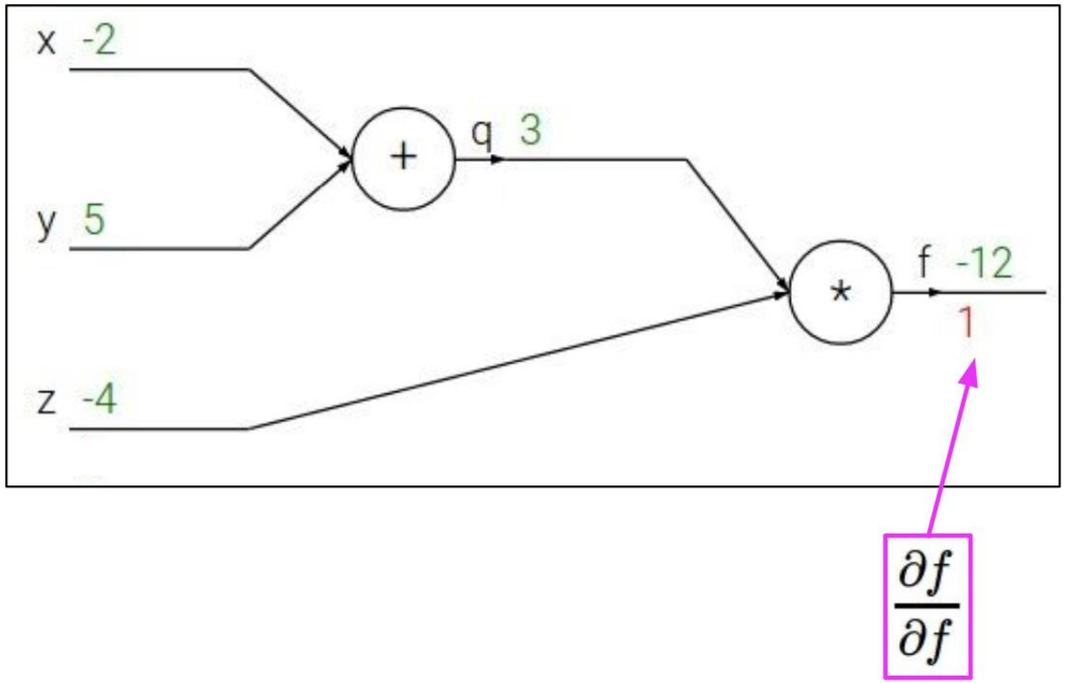
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Обратное распространение ошибки (backpropagation)

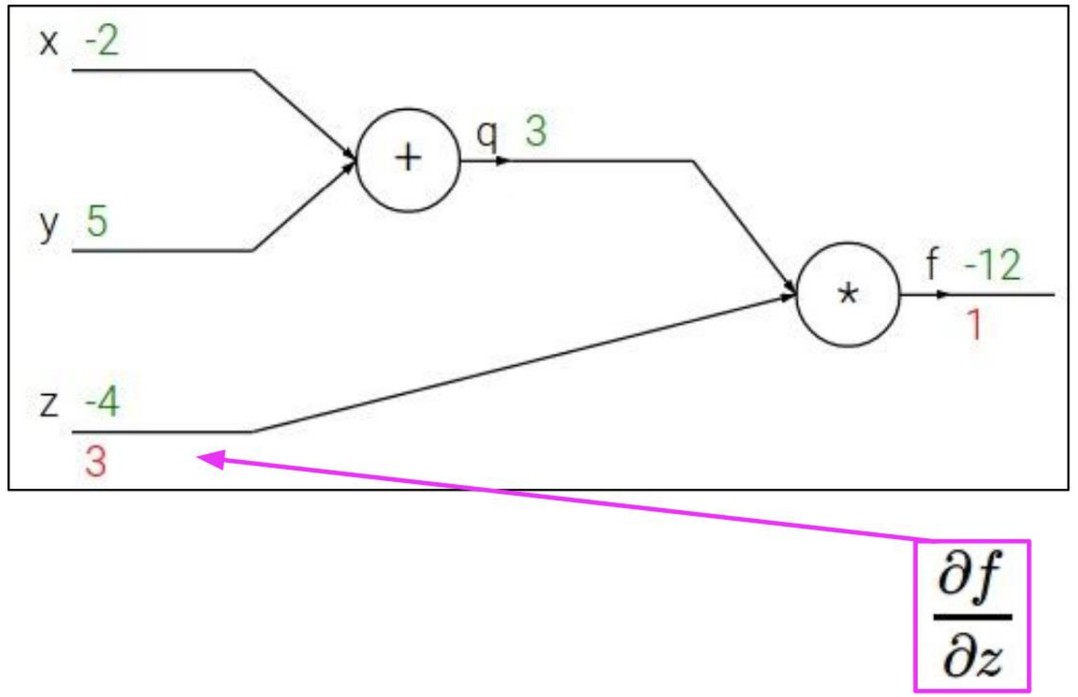
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Обратное распространение ошибки (backpropagation)

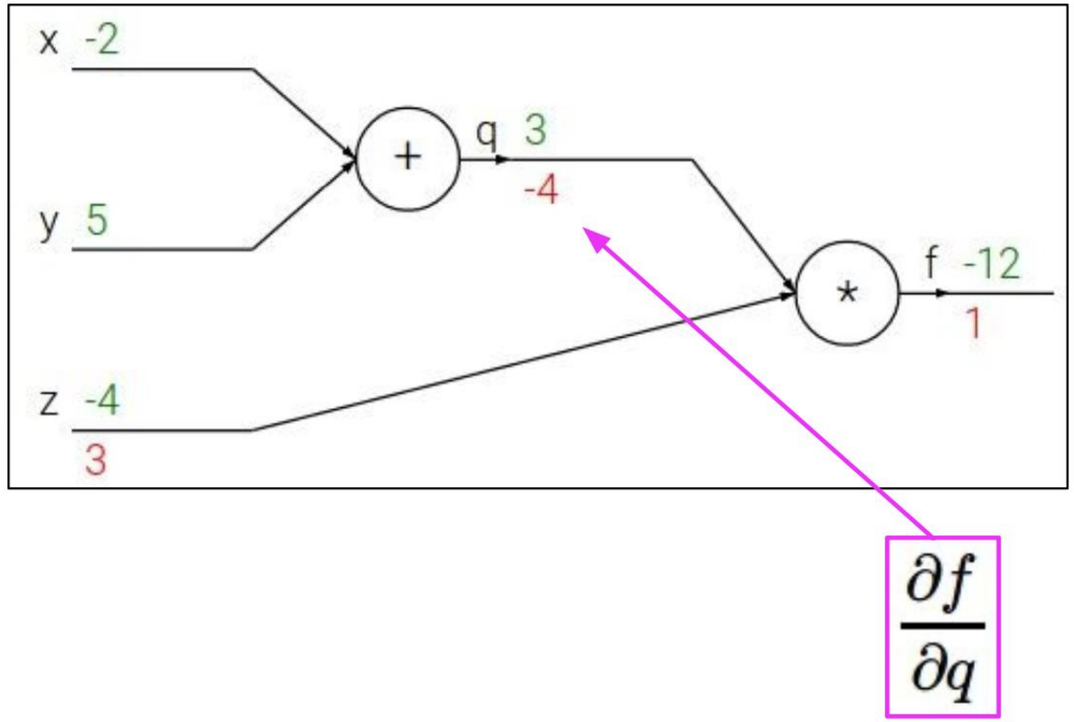
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



# Обратное распространение ошибки (backpropagation)

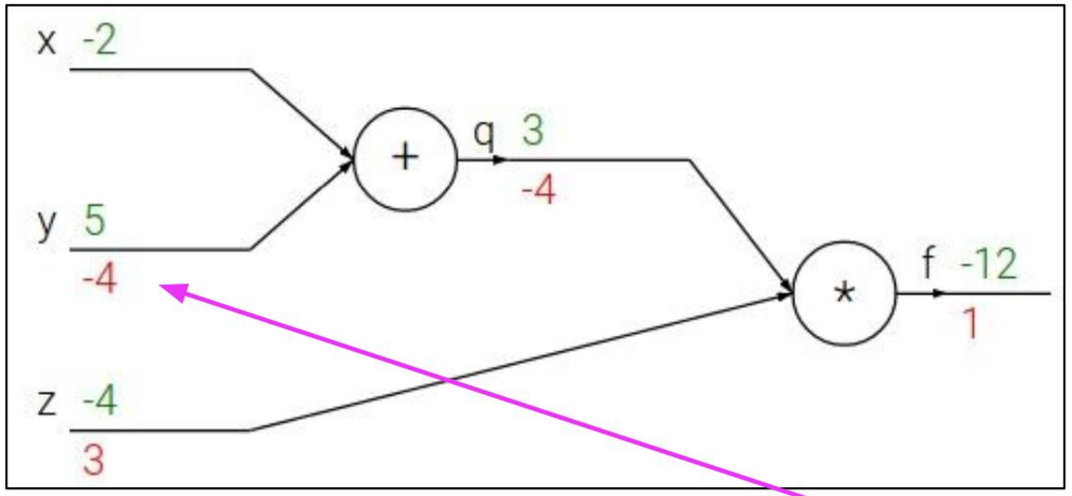
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$




Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream gradient

Local gradient

# Обратное распространение ошибки (backpropagation)

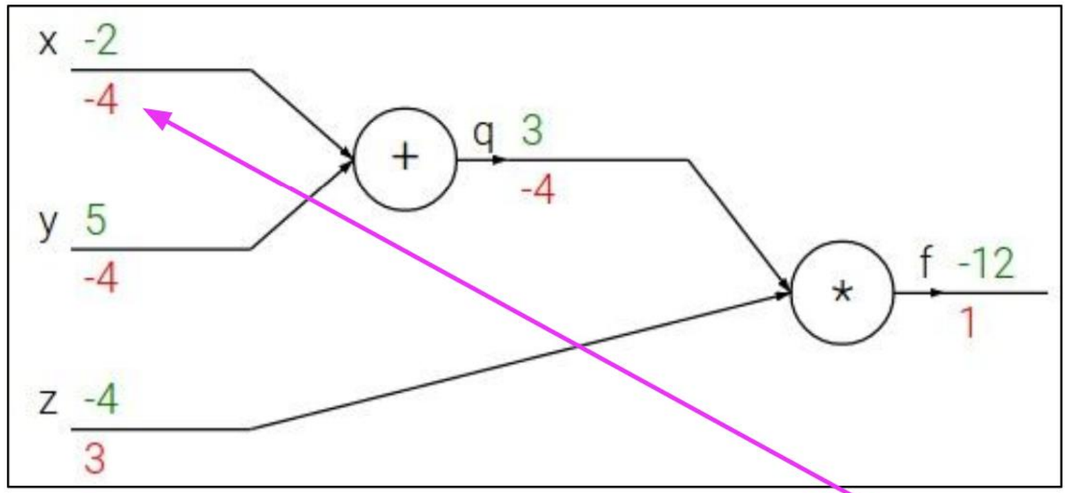
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$




$\frac{\partial f}{\partial x}$

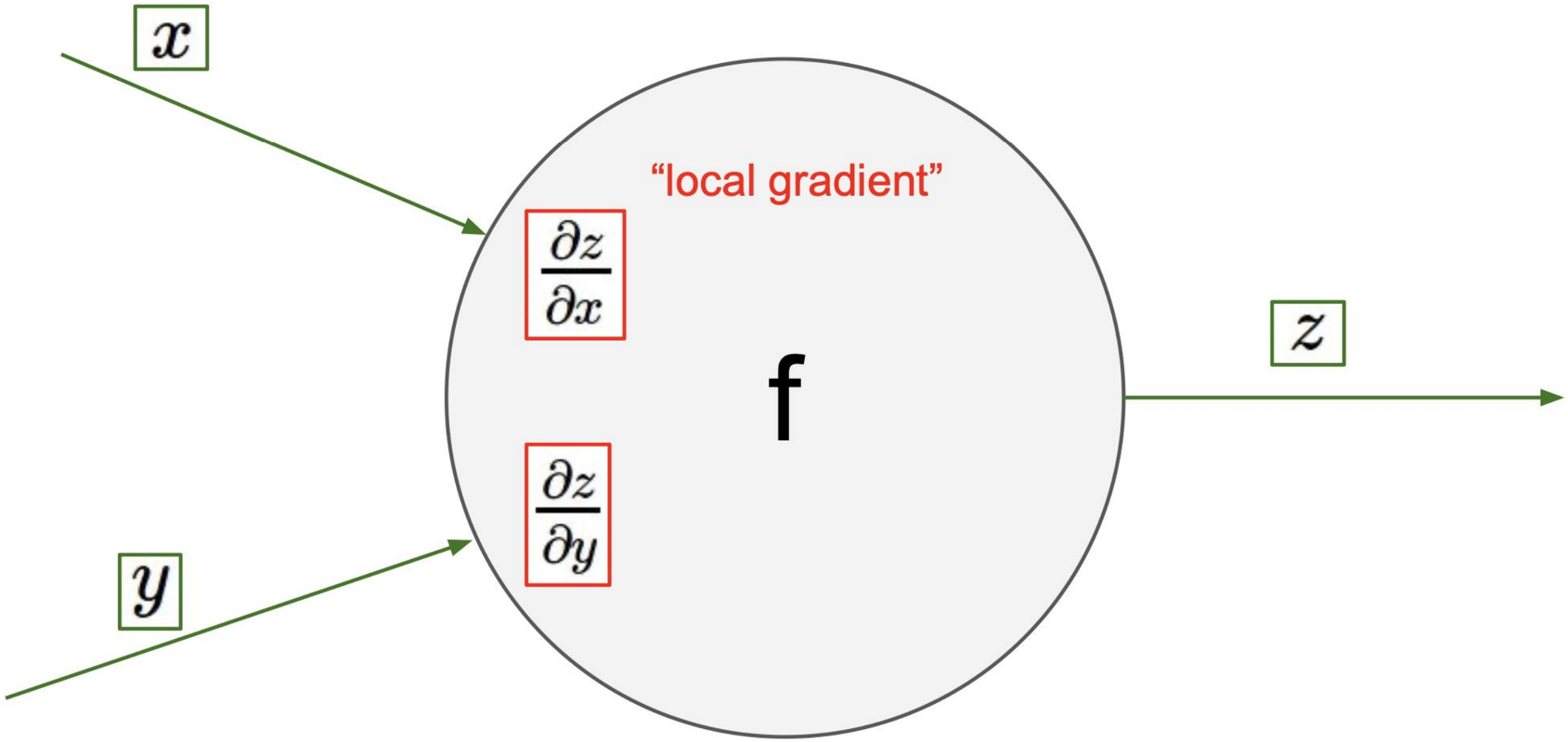
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

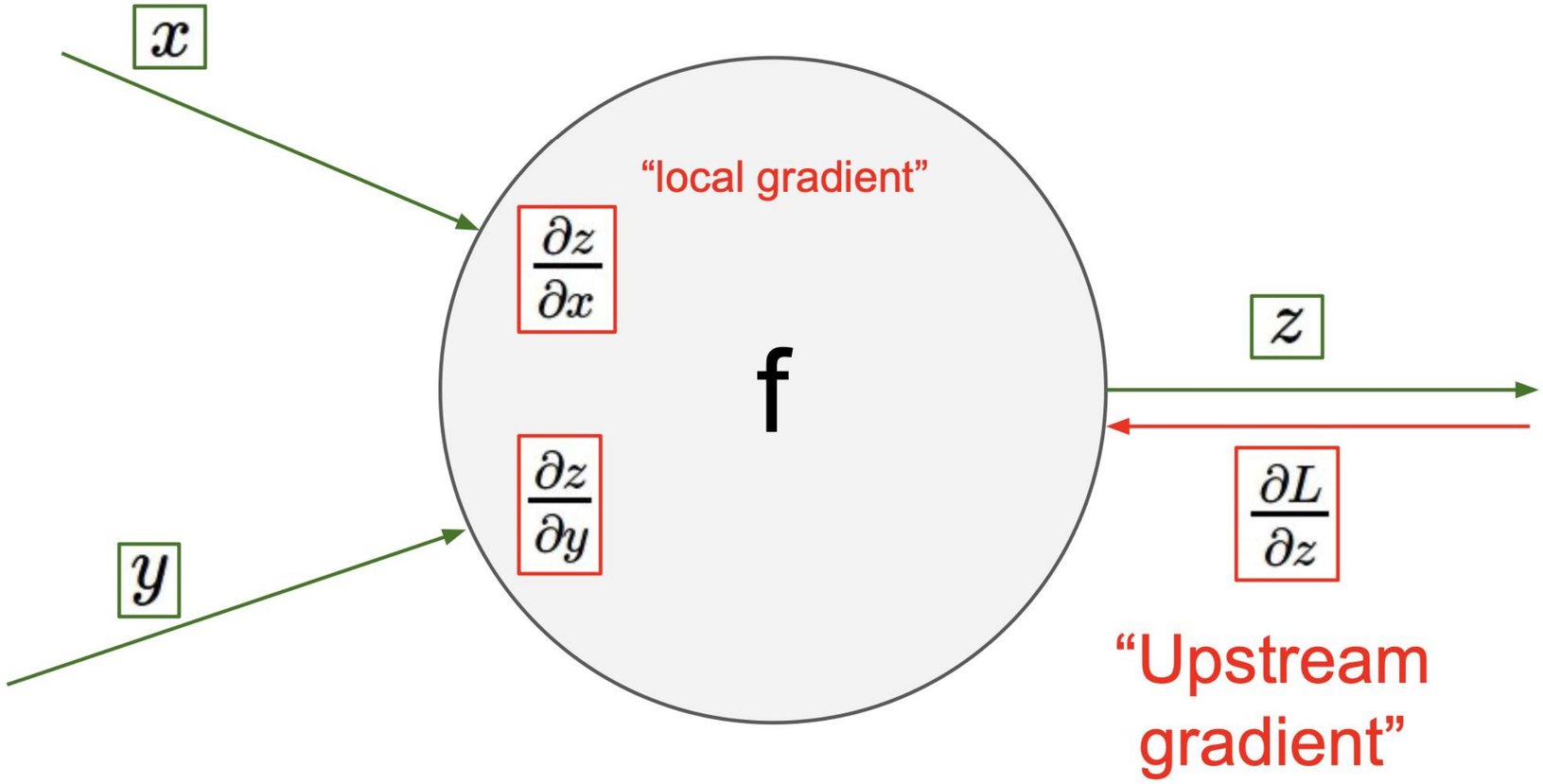
Upstream gradient

Local gradient

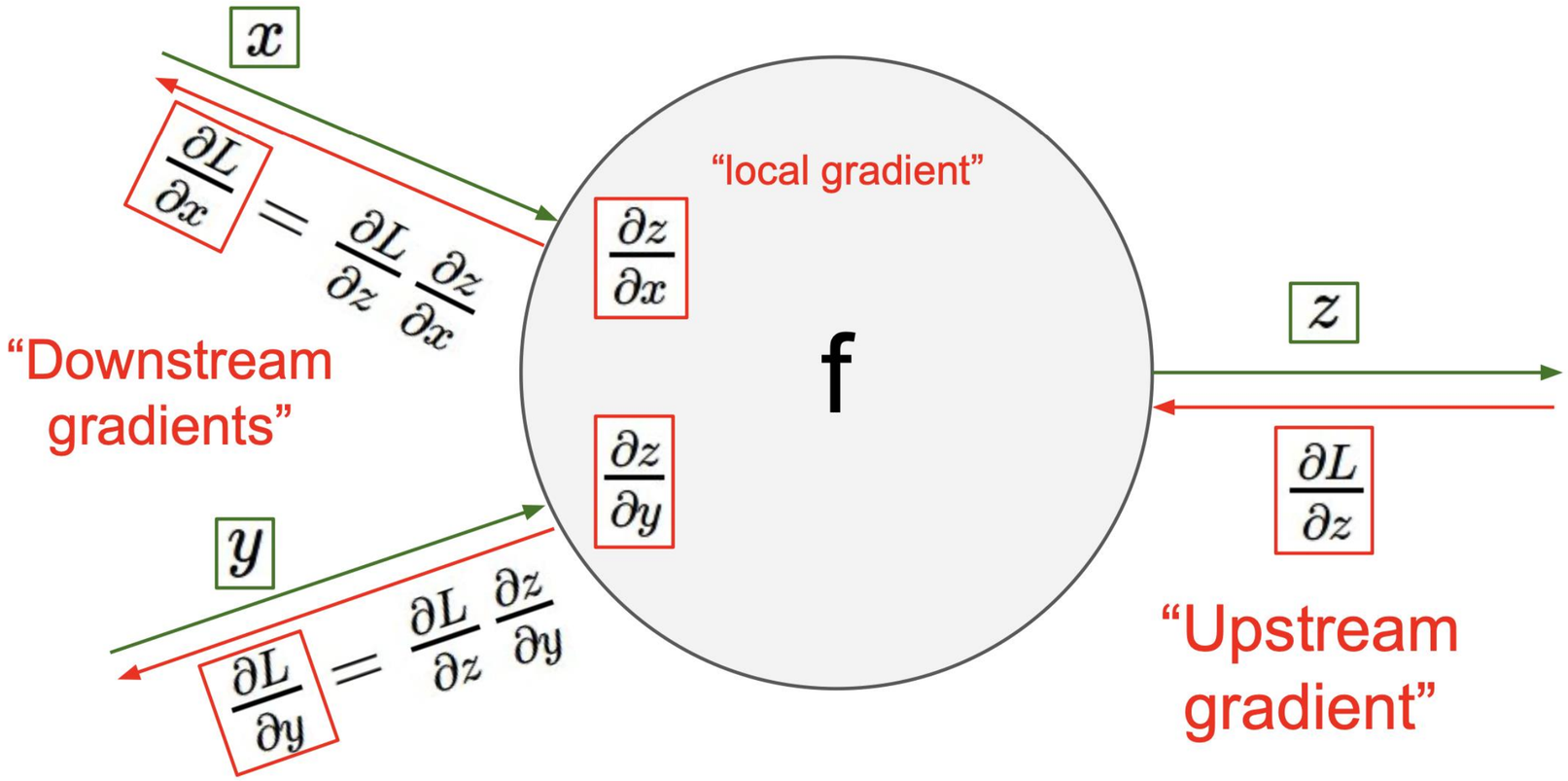
# Обратное распространение ошибки (backpropagation)



# Обратное распространение ошибки (backpropagation)

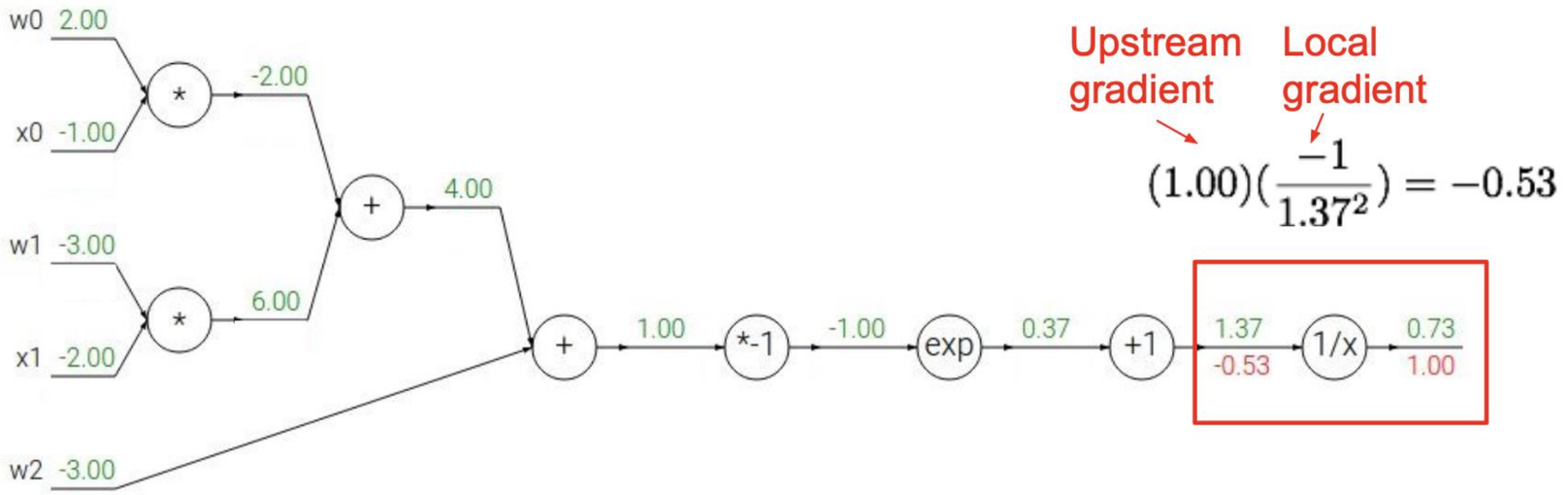


# Обратное распространение ошибки (backpropagation)



# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

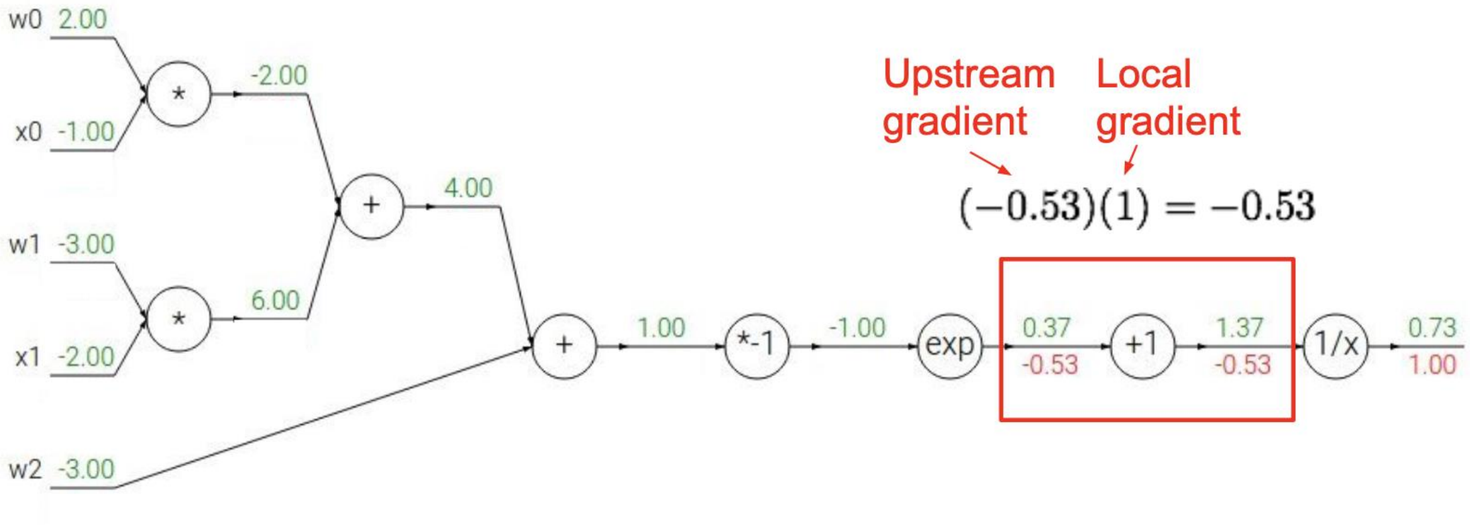
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

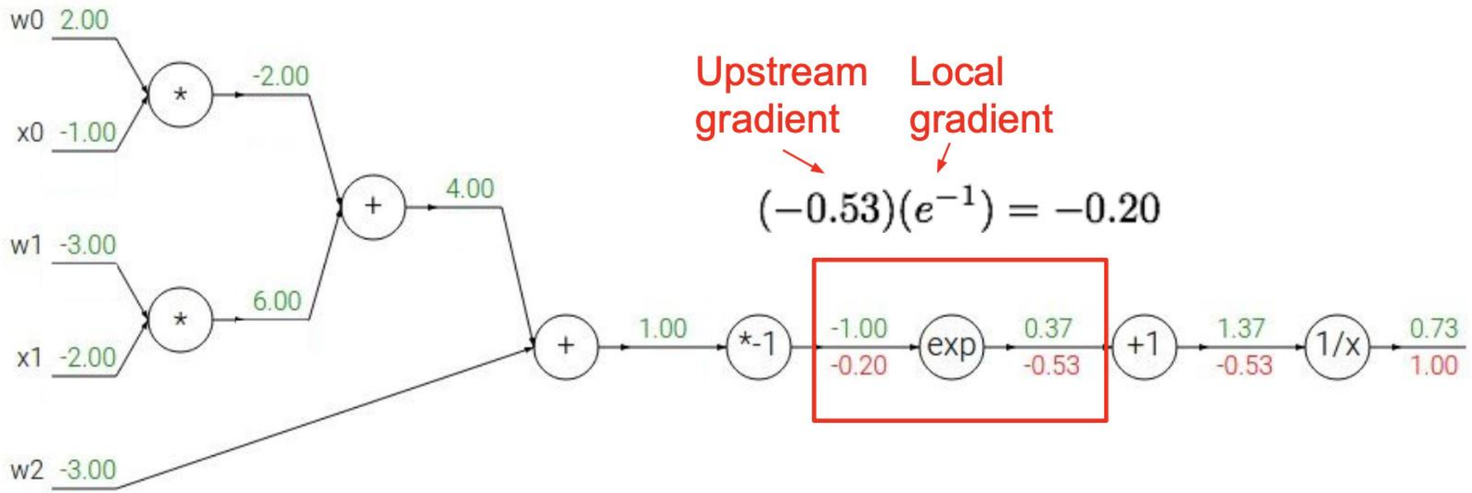
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

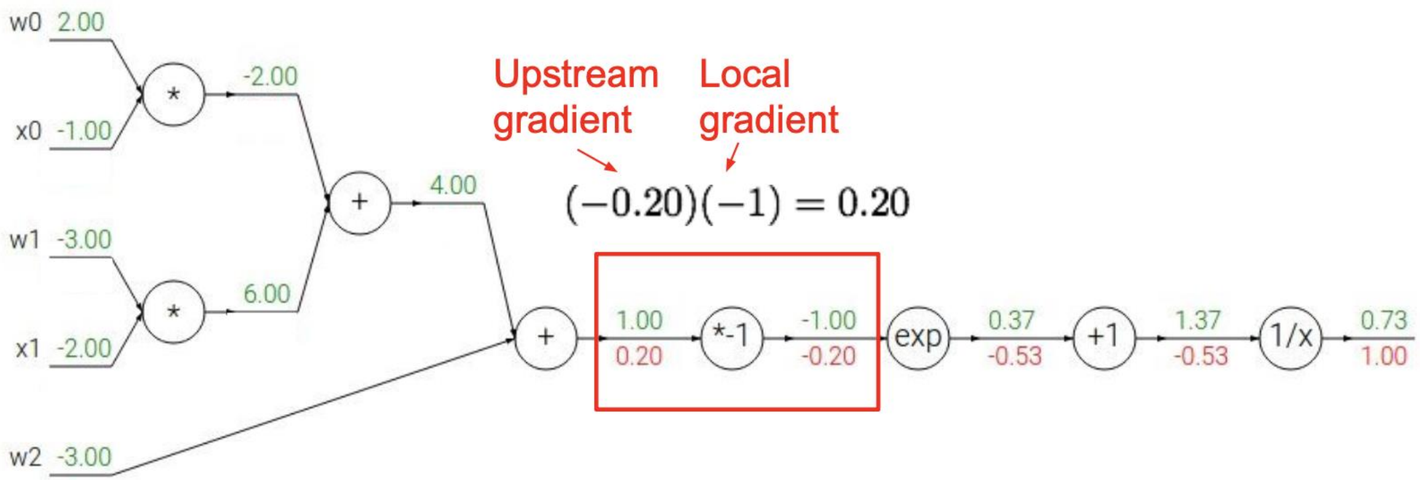
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

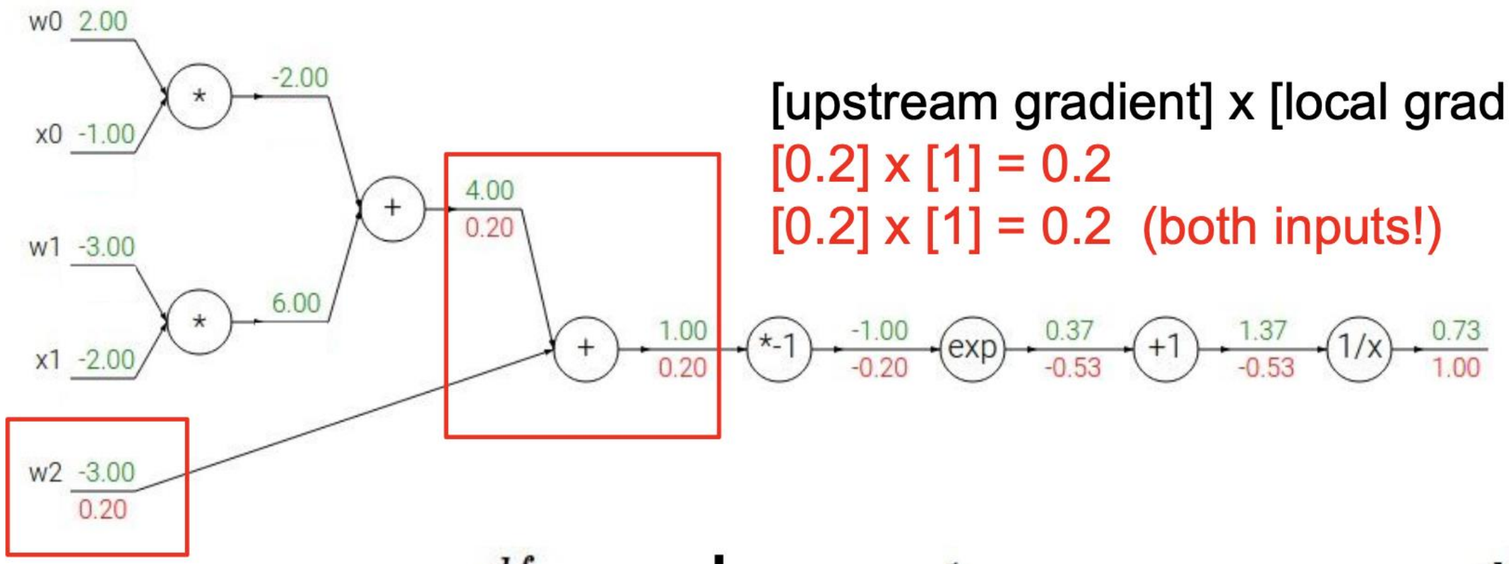
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

[0.2] x [1] = 0.2 (both inputs!)

$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

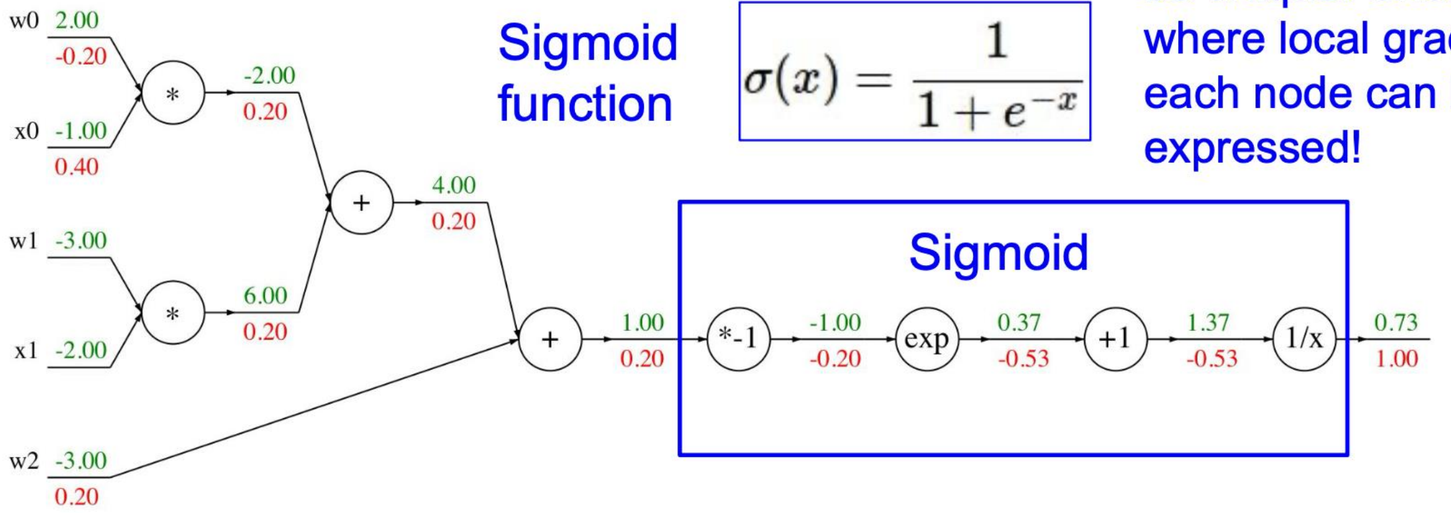
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




Sigmoid function

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

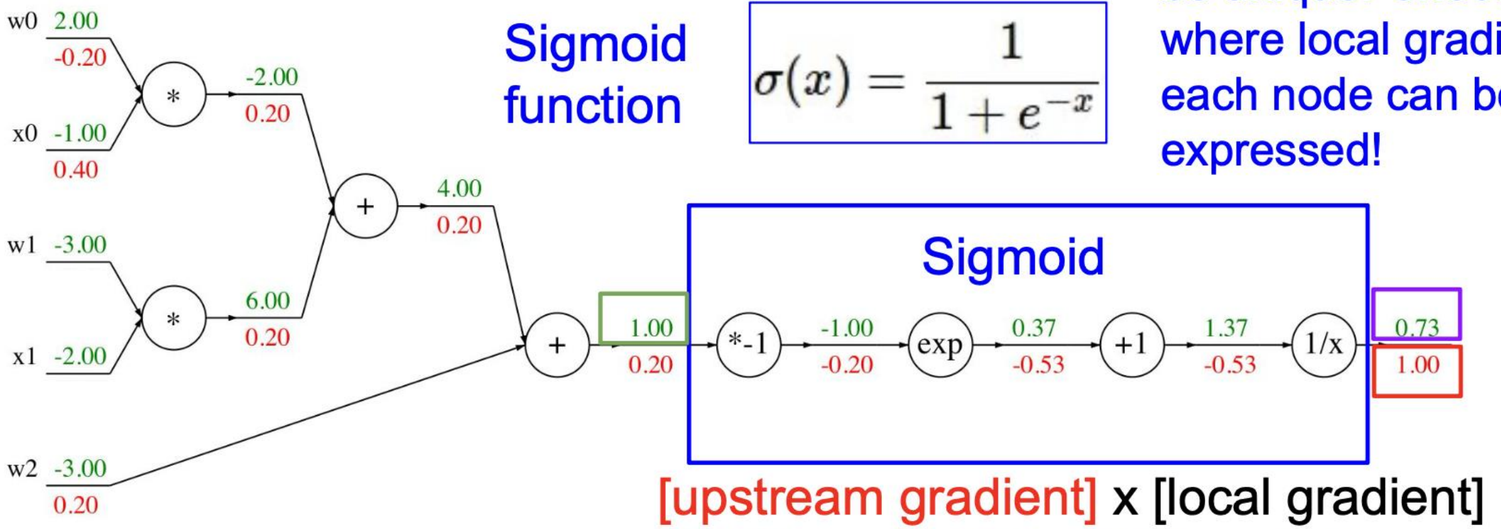
Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

# Другой пример

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$




Sigmoid function

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

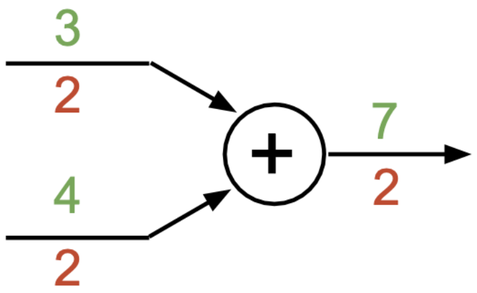
[upstream gradient] x [local gradient]
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient:

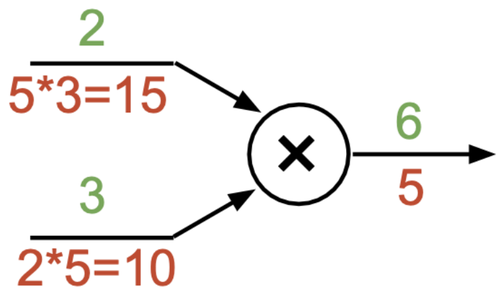
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

# Паттерны в графе градиентов

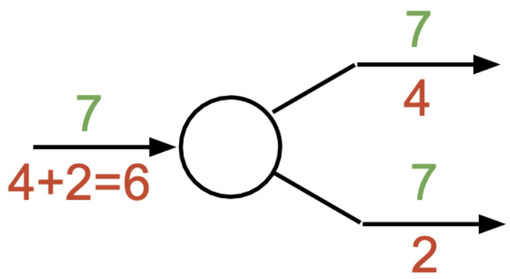
**add** gate: gradient distributor



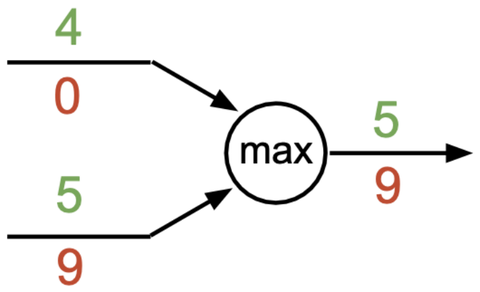

**mul** gate: “swap multiplier”



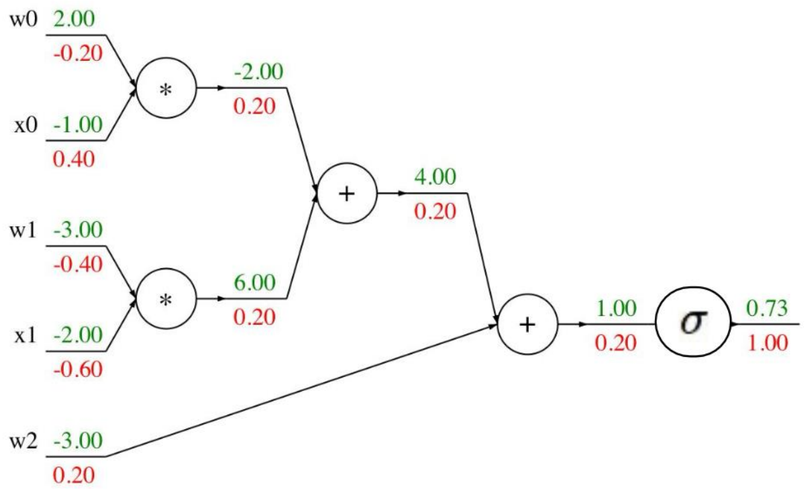

**copy** gate: gradient adder




**max** gate: gradient router



# Код обратного распространения




Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
  s0 = w0 * x0
  s1 = w1 * x1
  s2 = s0 + s1
  s3 = s2 + w2
  L = sigmoid(s3)
```

Backward pass:
Compute grads

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```

# Многомерная задача

## Scalar to Scalar

$x \in \mathbb{R}, y \in \mathbb{R}$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

## Vector to Scalar

$x \in \mathbb{R}^N, y \in \mathbb{R}$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

## Vector to Vector

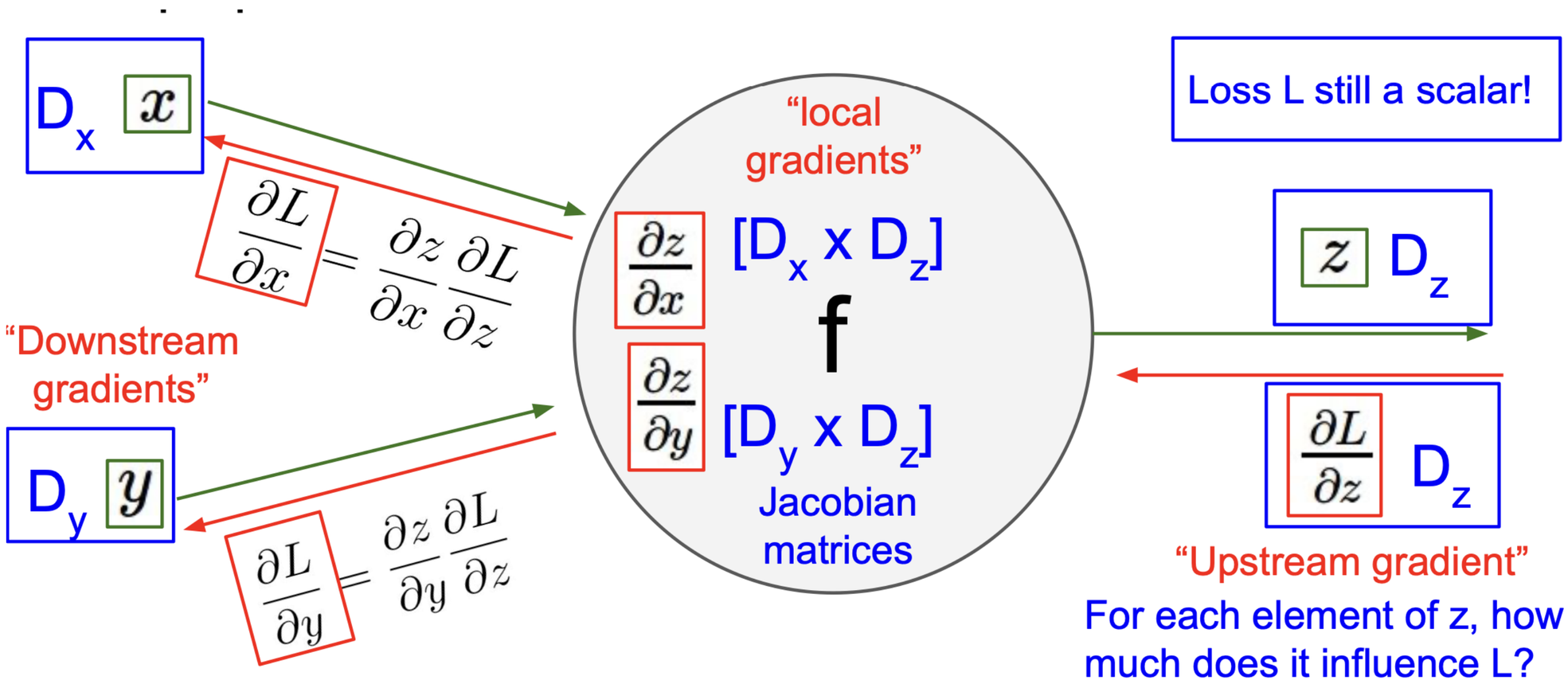
$x \in \mathbb{R}^N, y \in \mathbb{R}^M$

Derivative is **Jacobian**:

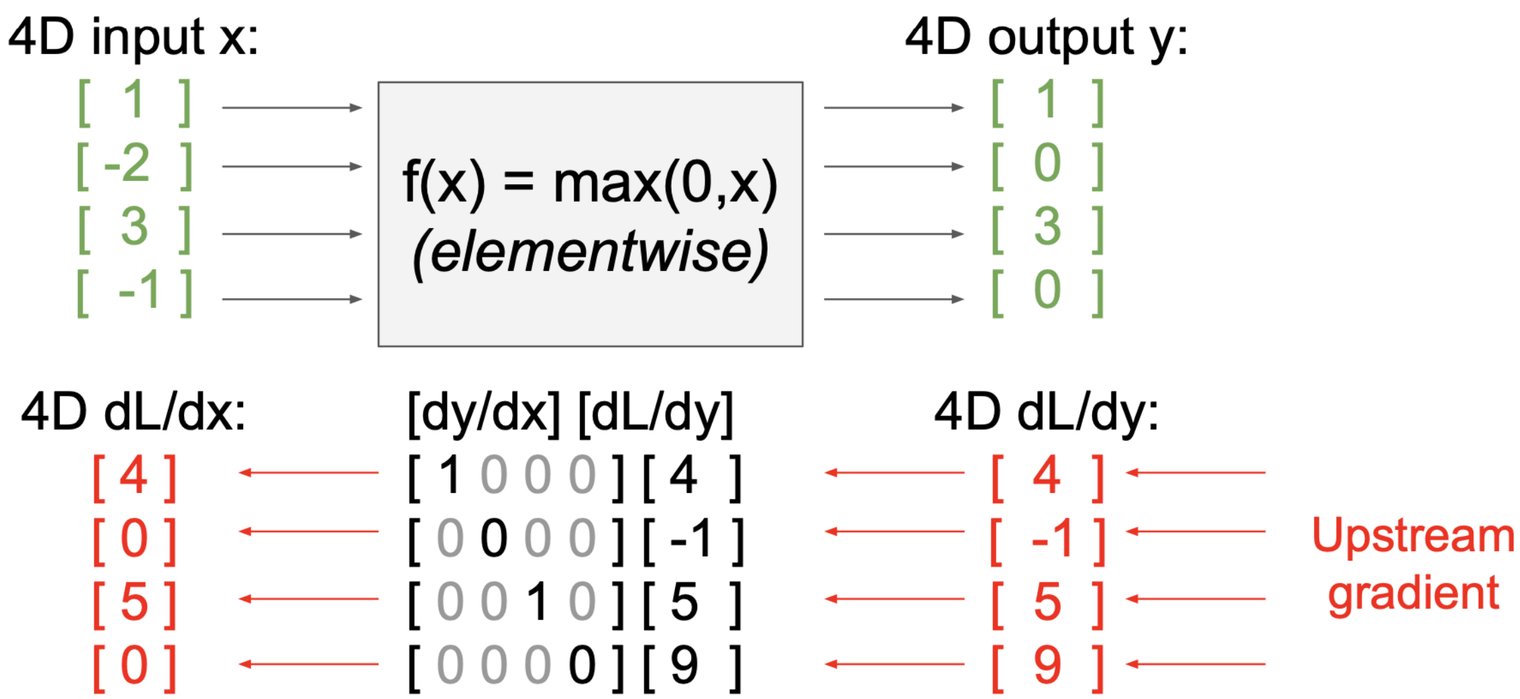
$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

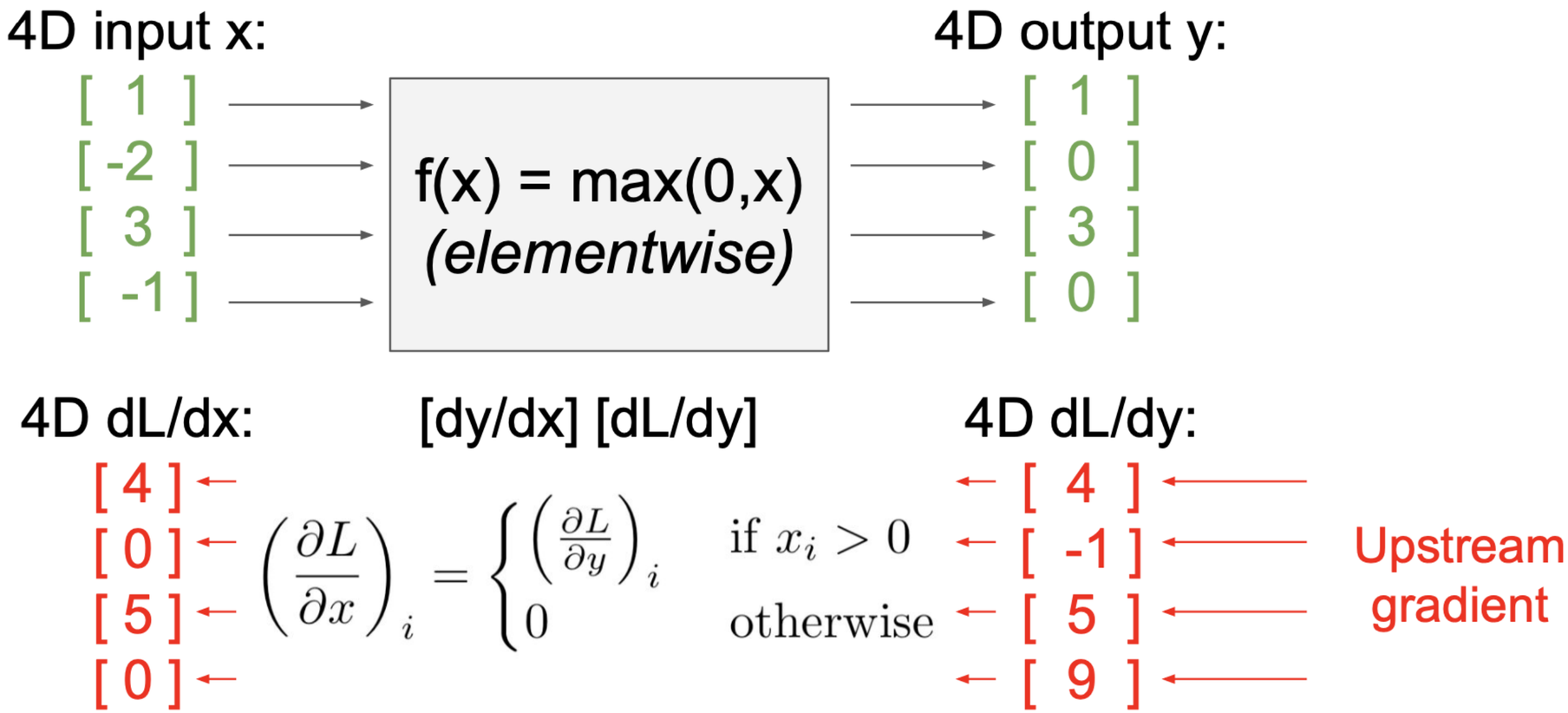
# Backprop с векторами



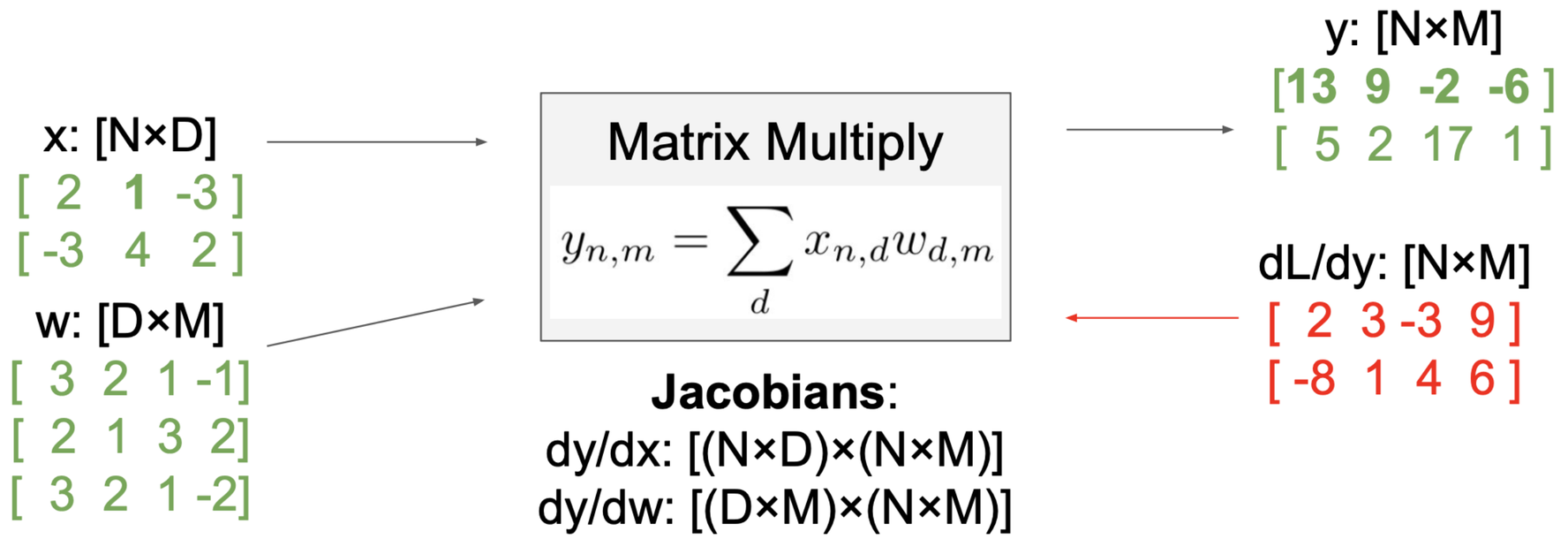
# Backprop с векторами



# Backprop с векторами

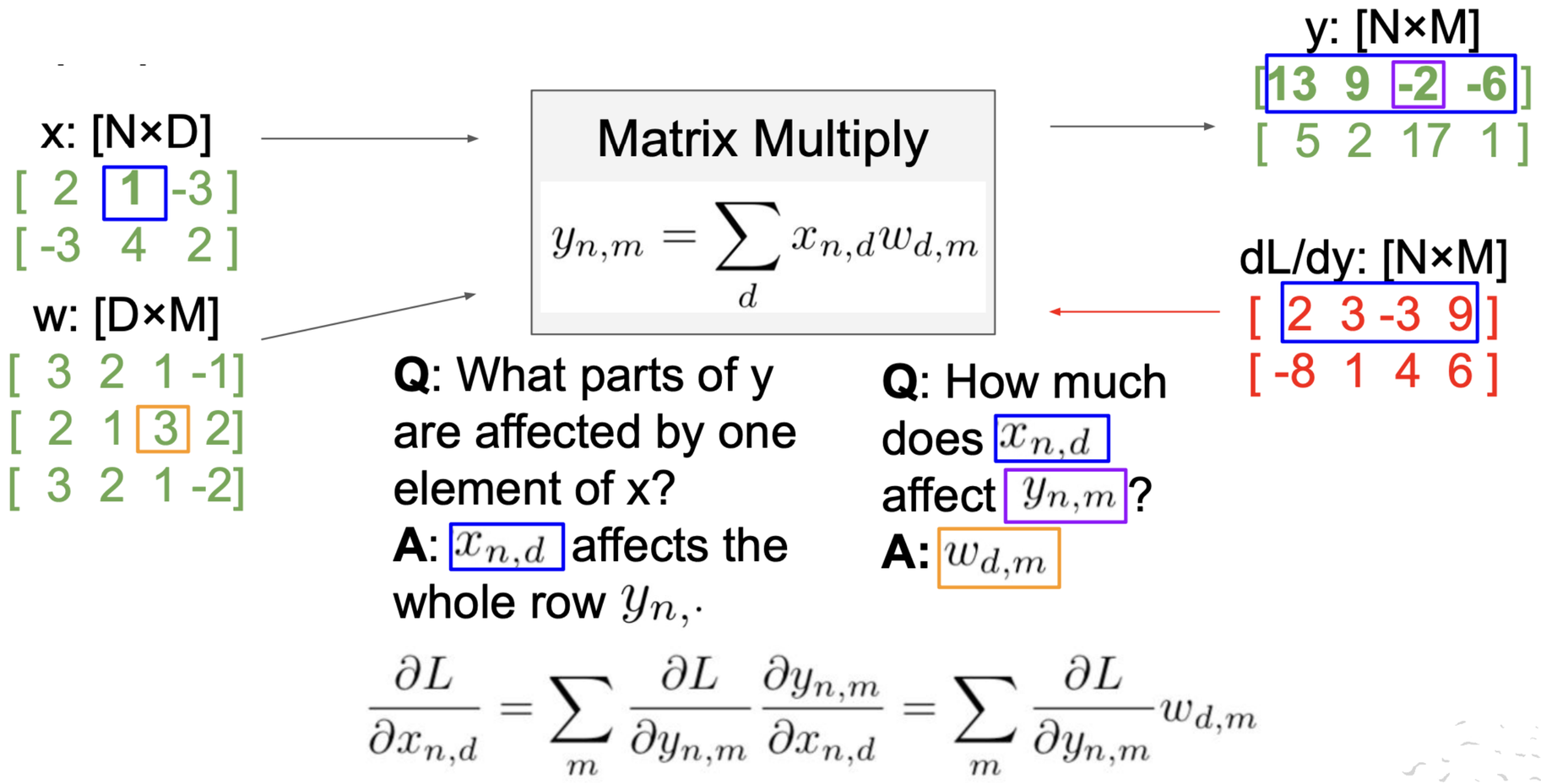


# Backprop с векторами



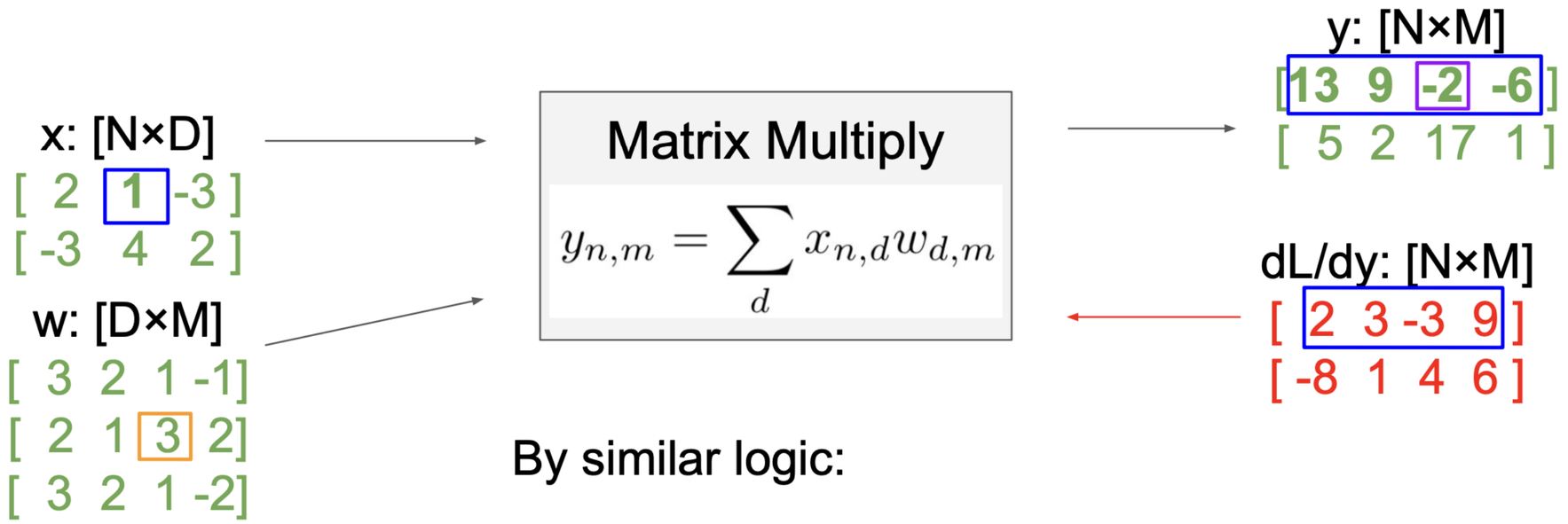

For a neural net we may have
N=64, D=M=4096
Each Jacobian takes 256 GB of memory!
Must work with them implicitly!

# Backprop с векторами




$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

# Backprop с векторами




By similar logic:

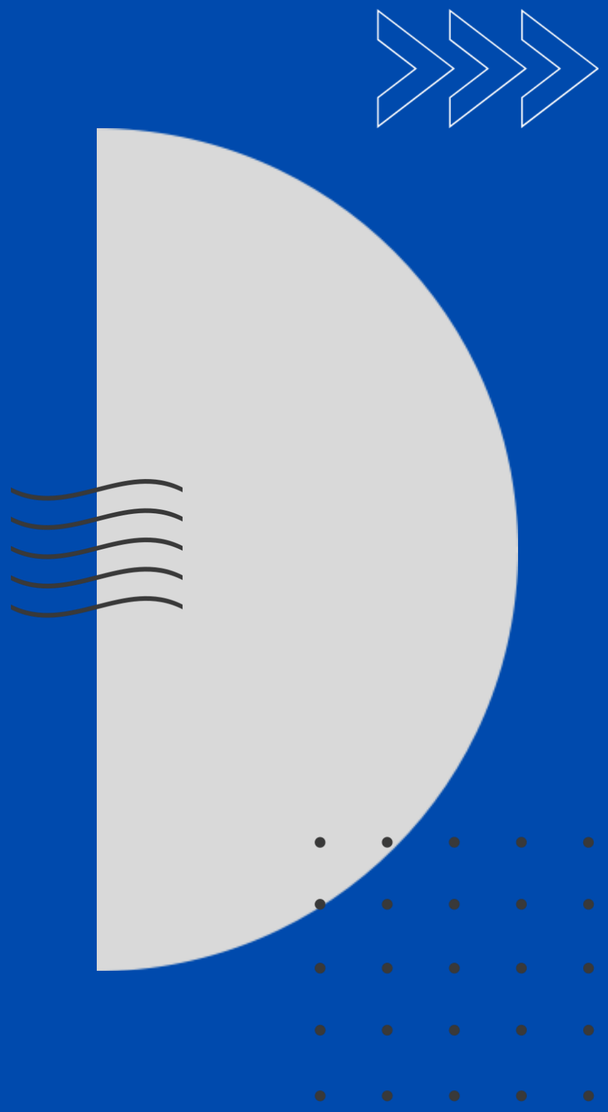
[N×D] [N×M] [M×D]

$$\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y}\right) w^T$$

[D×M] [D×N] [N×M]

$$\frac{\partial L}{\partial w} = x^T \left(\frac{\partial L}{\partial y}\right)$$

These formulas are easy to remember: they are the only way to make shapes match up!



03

# Функция ошибки

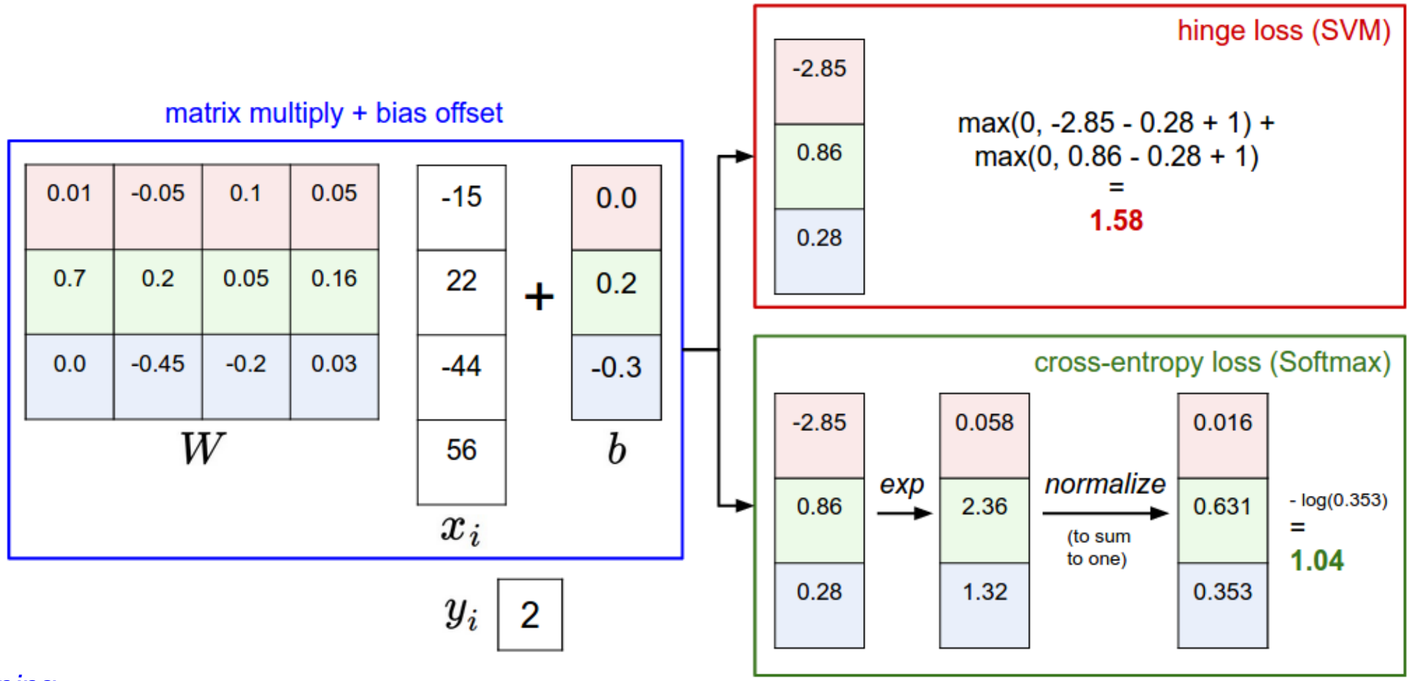
# Softmax classifier

Hinge loss (margin)

$$L_i = C \max(0, 1 - y_i w^T x_i) + R(W)$$

Softmax loss

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log \sum e^{f_j}$$




*Fei-Fei Li & Justin Johnson & Serena Yeung | cs231n*

*A Gentle Introduction to Cross-Entropy for Machine Learning*

# Softmax classifier

## Softmax loss

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

If $i = j$,

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{e^{a_i}\sum_{k=1}^{N} e^{a_k} - e^{a_j}e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_i}\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}} \\
&= p_i(1 - p_j)
\end{aligned}$$

For $i \neq j$,

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{0 - e^{a_j}e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{-e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} \\
&= -p_j . p_i
\end{aligned}$$

[Softmax and Cross Entropy Loss]

# Softmax classifier

## Softmax loss

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

If $i = j$,

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{e^{a_i}\sum_{k=1}^{N} e^{a_k} - e^{a_j}e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_i}\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}} \\
&= p_i(1 - p_j)
\end{aligned}$$

For $i \neq j$,

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{0 - e^{a_j}e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{-e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} \\
&= -p_j . p_i
\end{aligned}$$

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1 - p_j) & if \;\; i = j \\ -p_j . p_i & if \;\; i \neq j \end{cases}$$

$$\frac{\partial p_i}{\partial a_j} = p_i(\delta_{ij} - p_j) \quad , \delta ij = \begin{cases} 1 & if \;\; i = j \\ 0 & if \;\; i \neq j \end{cases}$$

[Softmax and Cross Entropy Loss]

# Softmax classifier

Softmax loss

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

Cross-entropy loss

$$H(p,q) = -\sum_x p(x)\log q(x)$$

Cross-entropy указывает на разницу между тем, каким, по мнению модели, должно быть выходное распределение, и тем, каким на самом деле является исходное распределение.

$$q = e^{f_{y_i}} / \sum_j e^{f_j}$$

$$p = [0, \ldots 1, \ldots, 0]$$

$$H(p,q) = H(p) + D_{KL}(p||q)$$

$$KL(P|Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$$

[Softmax and Cross Entropy Loss]

# Softmax classifier

$$L = -\sum_i y_i log(p_i)$$

$$\frac{\partial L}{\partial o_i} = -\sum_k y_k \frac{\partial log(p_k)}{\partial o_i}$$

$$= -\sum_k y_k \frac{\partial log(p_k)}{\partial p_k} \times \frac{\partial p_k}{\partial o_i}$$

$$= -\sum y_k \frac{1}{p_k} \times \frac{\partial p_k}{\partial o_i}$$

$$\frac{\partial L}{\partial o_i} = -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k.p_i)$$

$$= -y_i(1-p_i) + \sum_{k \neq 1} y_k.p_i$$

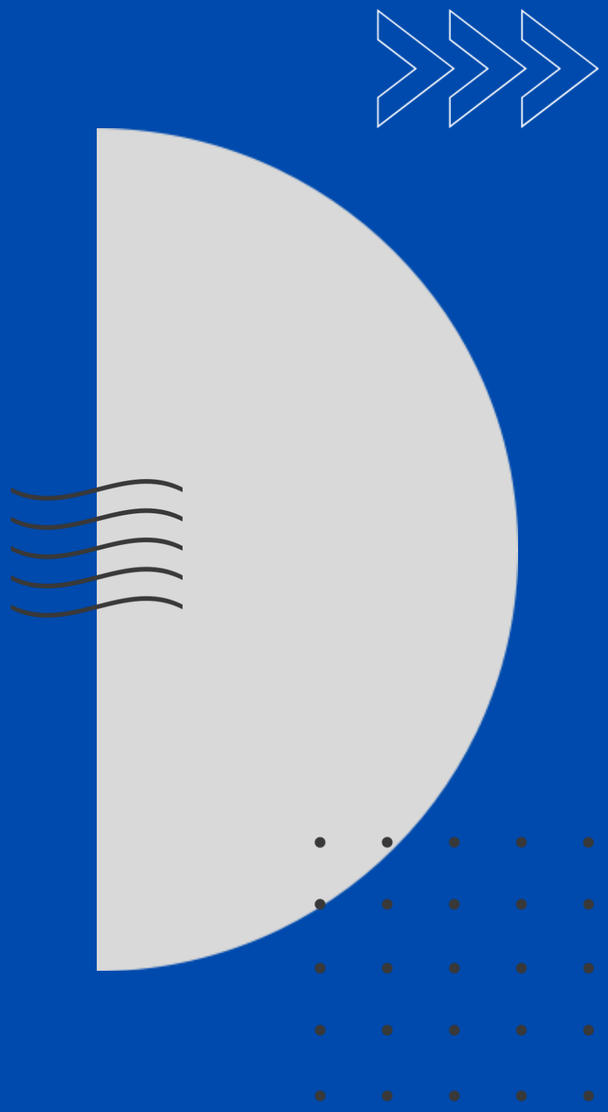
$$= -y_i + y_i p_i + \sum_{k \neq 1} y_k.p_i$$

$$= p_i \left( y_i + \sum_{k \neq 1} y_k \right) - y_i$$

$$= p_i \left( y_i + \sum_{k \neq 1} y_k \right) - y_i$$

$$\frac{\partial L}{\partial o_i} = p_i - y_i$$

[Softmax and Cross Entropy Loss]



# Место для ваших вопросов