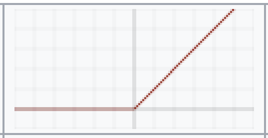
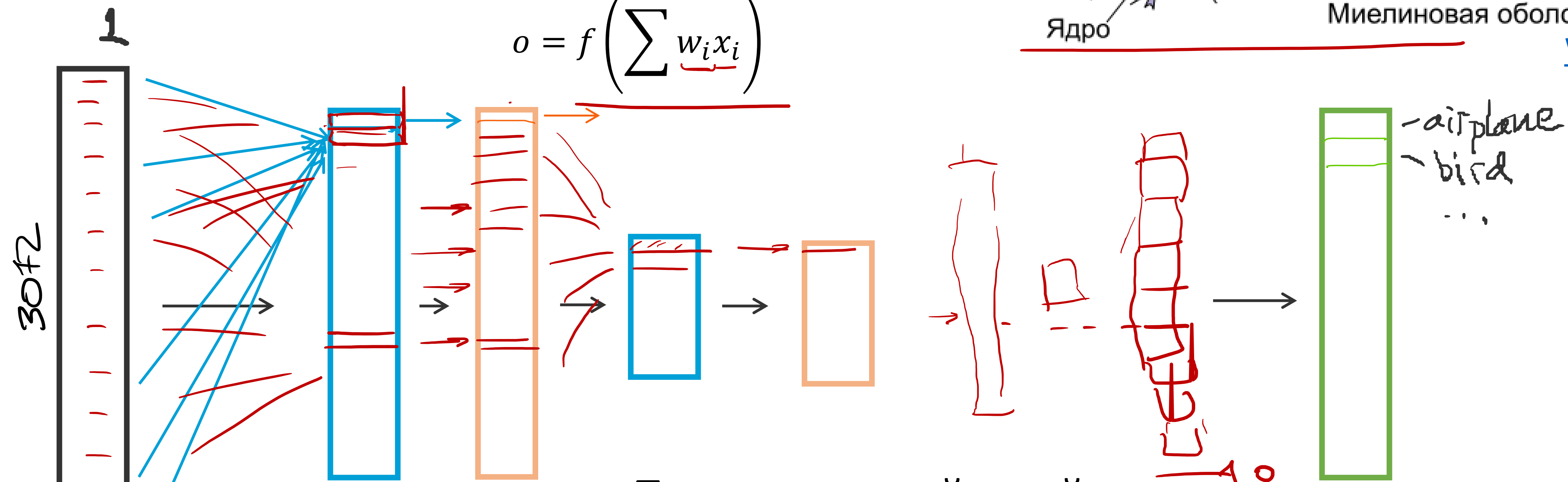
Лекция 4

Нейронные сети — подробности

Нейронная сеть
Neural network

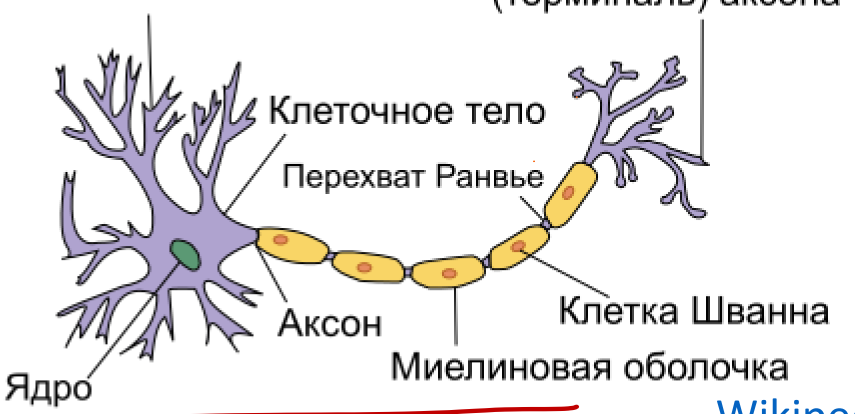
Rectified linear unit (ReLU)[10]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

$$o = f\left(\sum w_i x_i\right)$$



Полносвязный слой
Fully-connected layer

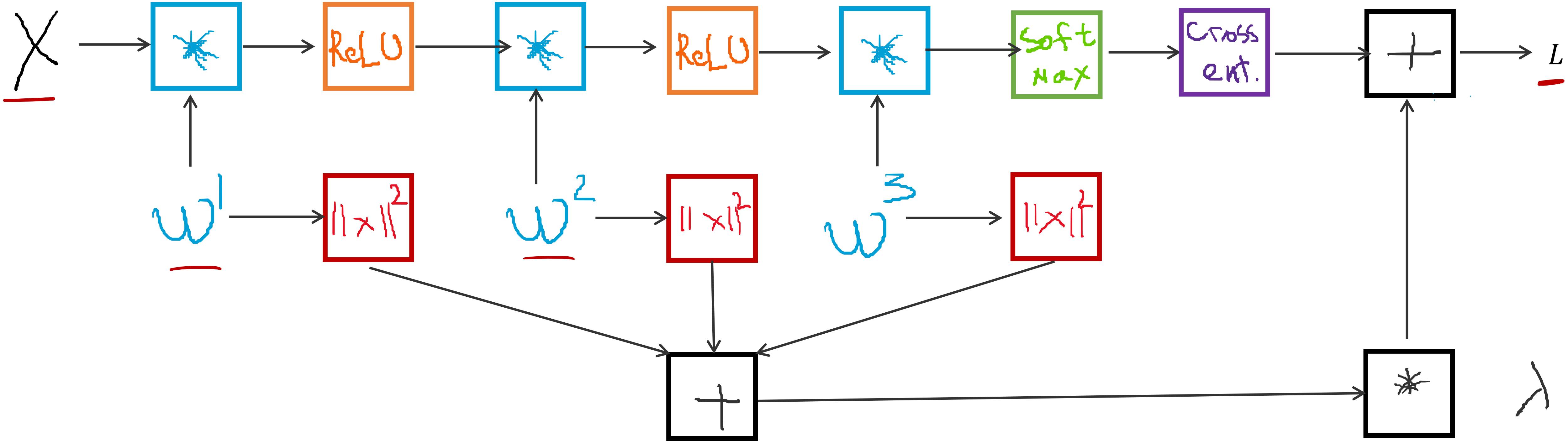
Типичная структура нейрона



Wikipedia

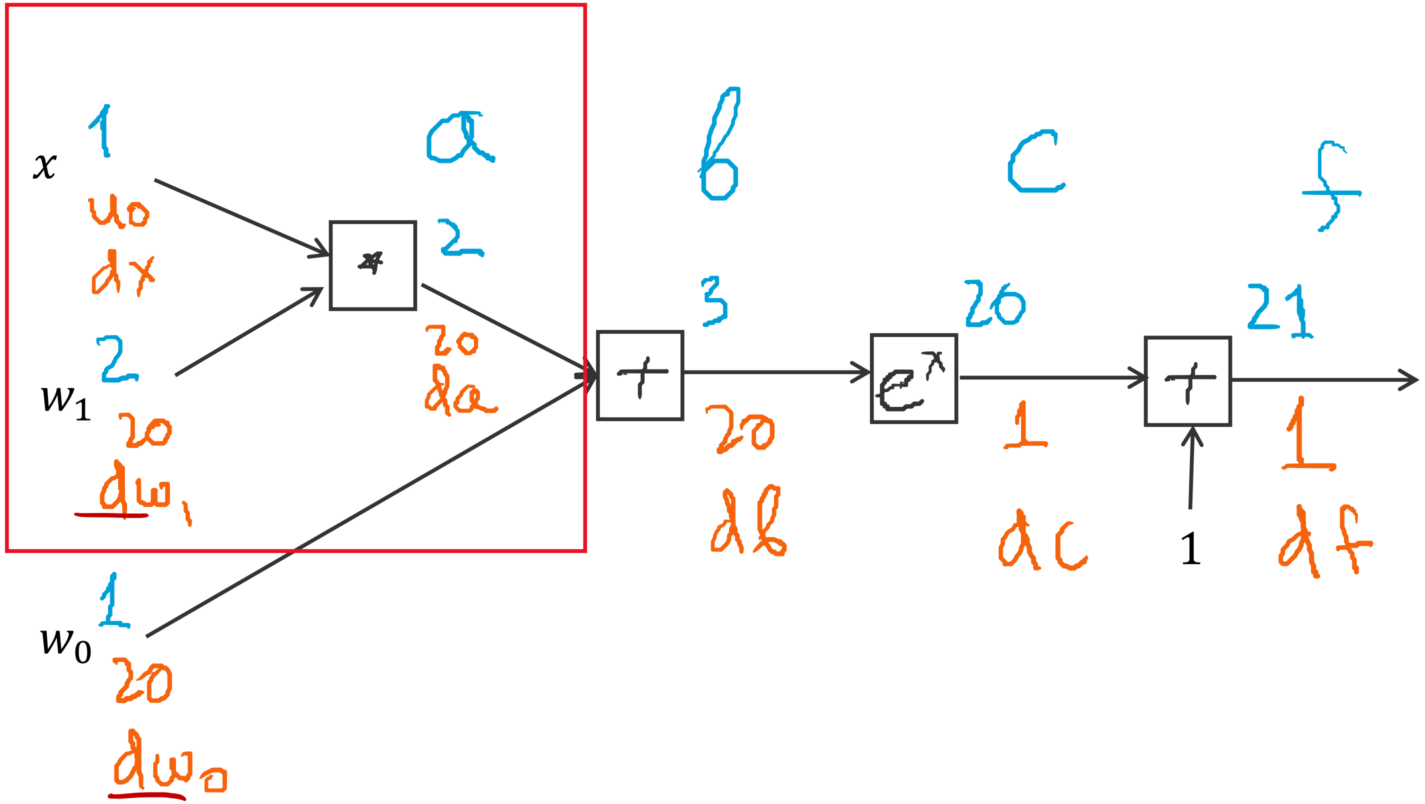
Граф вычислений
Computational graph

$$L = -\sum_j \ln p(c = y_j | x_j) + \lambda R(\omega)$$

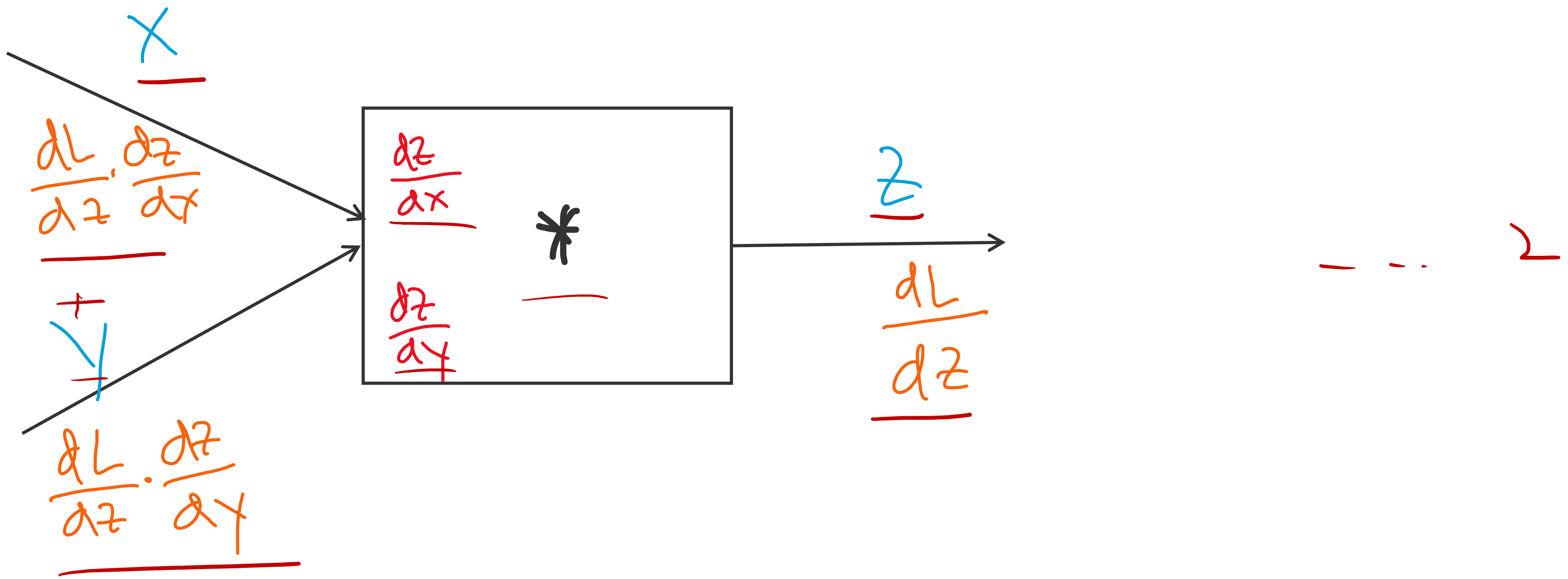


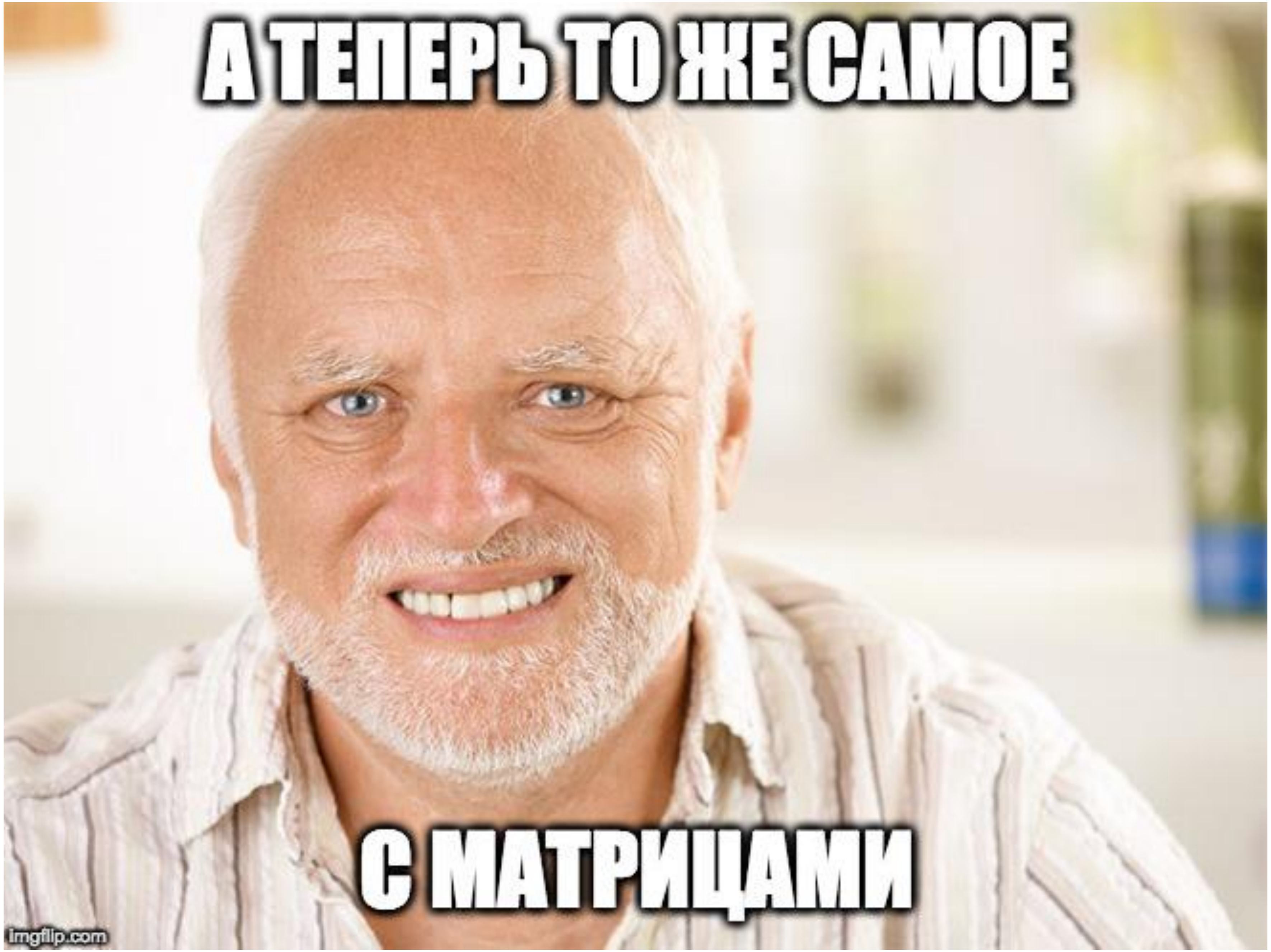
Обратное распространение ошибки Backpropagation

$f(g(x))$ $\qquad \frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$ $\qquad f(x,w) = 1 + e^{w_1 x + w_0}$



Общая схема вычисления градиента

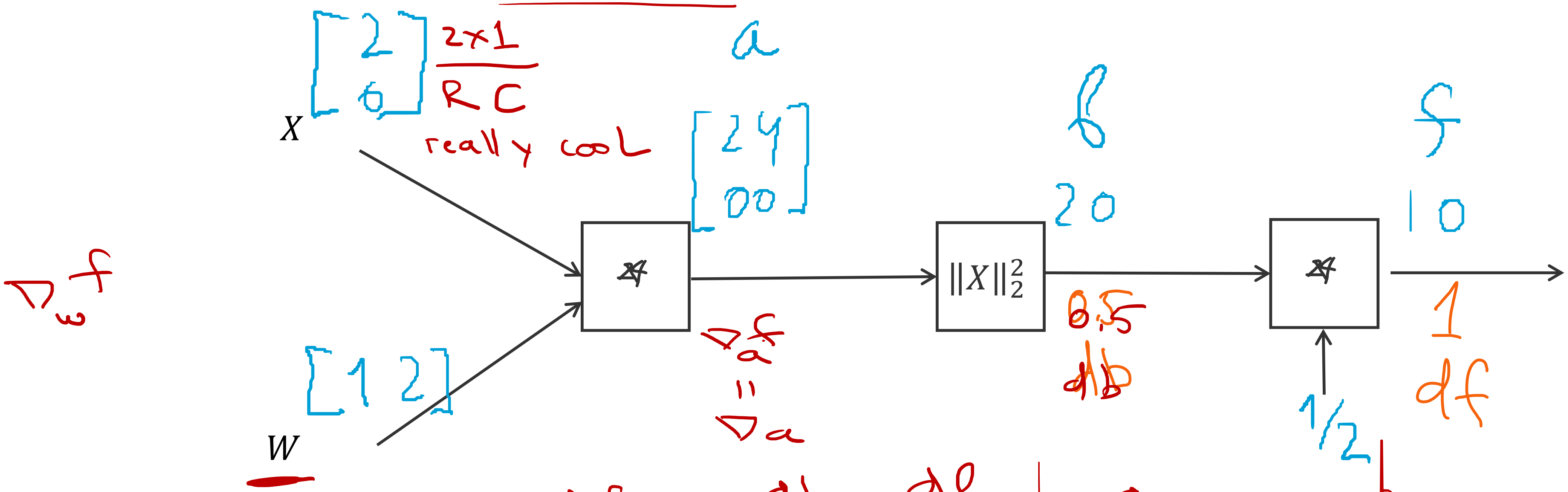



А ТЕПЕРЬ ТО ЖЕ САМОЕ
С МАТРИЦАМИ
imgflip.com

А теперь с матрицами

L2

$$f = \frac{1}{2}\|X \cdot W\|_2^2$$



$$b = \sum a_{ij}^2$$

$$\frac{df}{da_{ij}} = \underbrace{\frac{df}{db}}_{=0.5} \cdot \underbrace{\frac{db}{da_{ij}}}_{=2a_{ij}} \Big|\; a_{ij}$$

$$\nabla_a f = a$$

А теперь с матрицами

$$f = \frac{1}{2}\|X \cdot W\|_2^2$$

$\underbrace{\nabla_w f}_{1\times 2} = X^T \cdot \nabla_a f$

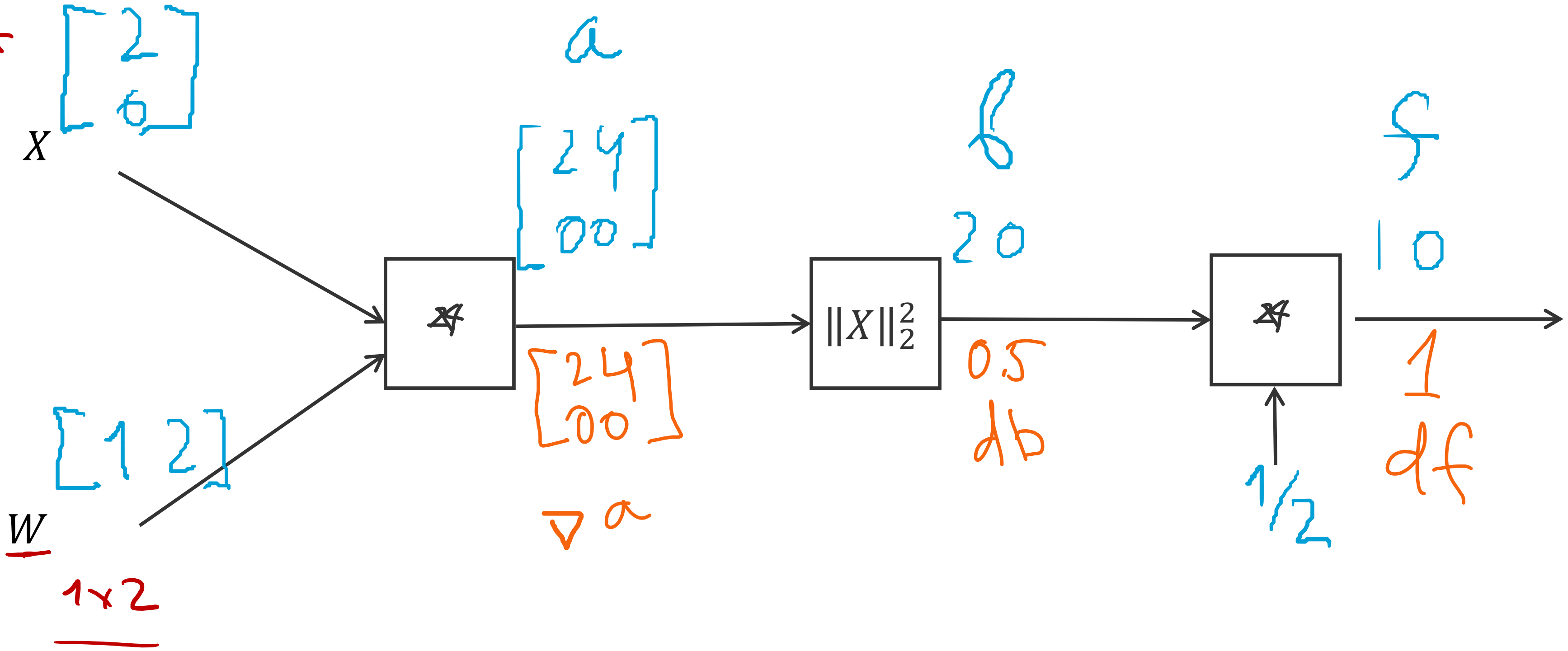
$1\times 2 \cdot 2\times 2$

$\Downarrow$

1×2

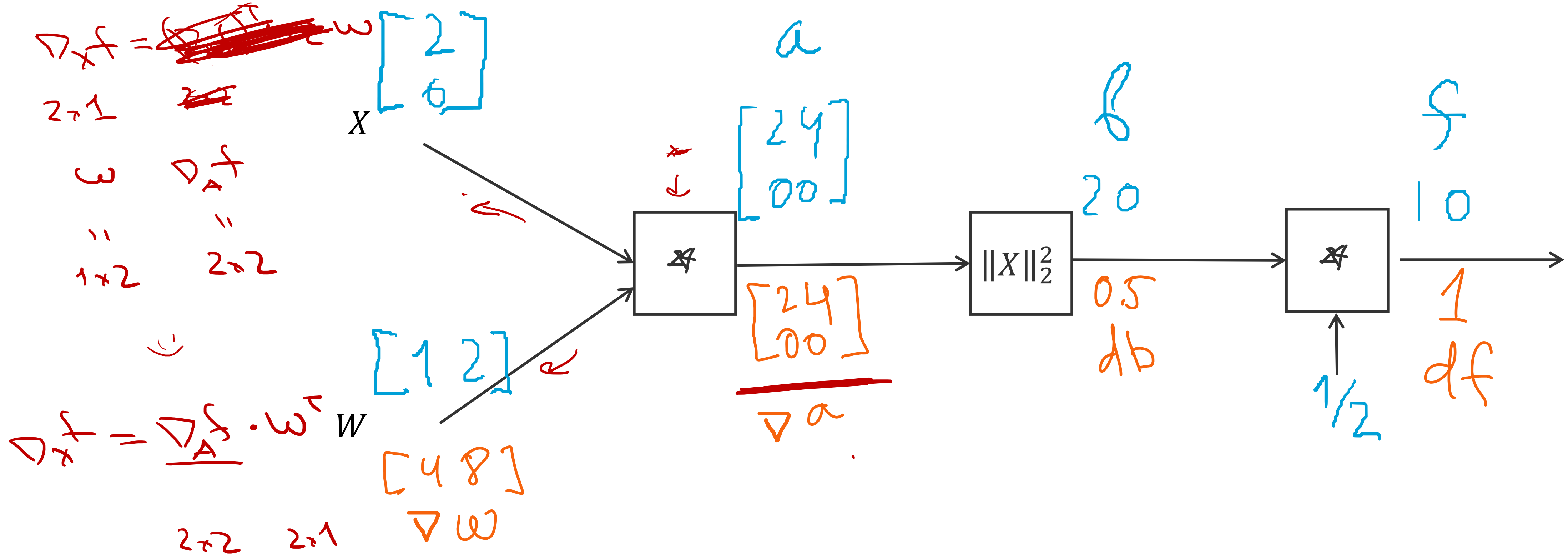
$\underbrace{\nabla_a f}_{2\times 2}$

$a = \underbrace{X^T}_{2\times 1} \cdot w$

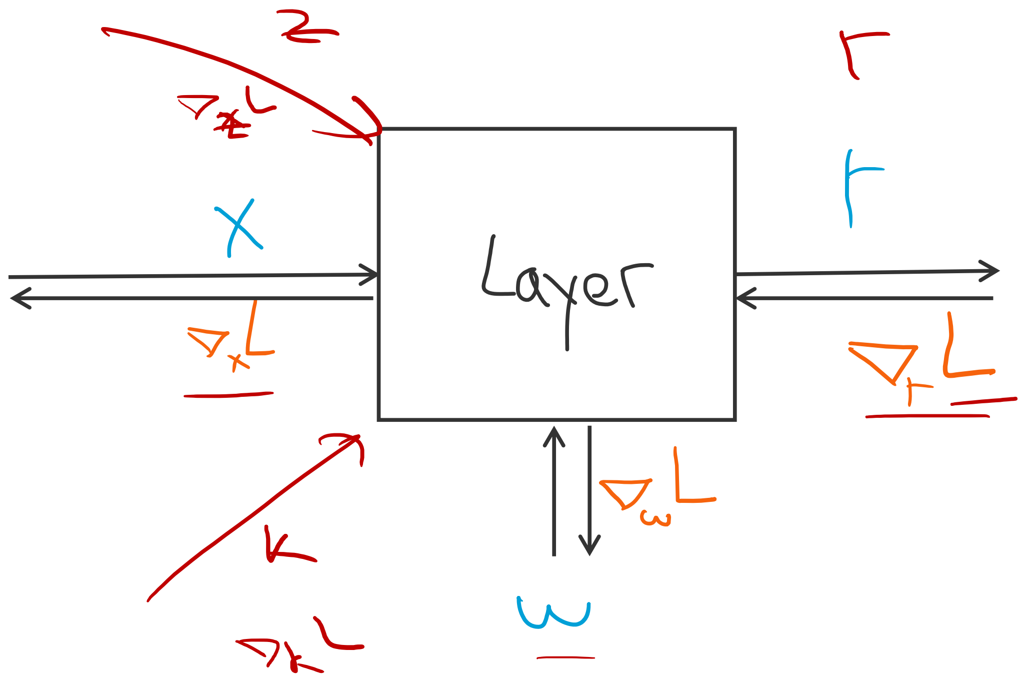


А теперь с матрицами

$$f = \frac{1}{2}\|\underline{X} \cdot \underline{W}\|_2^2$$



Интерфейс слоя
Layer interface

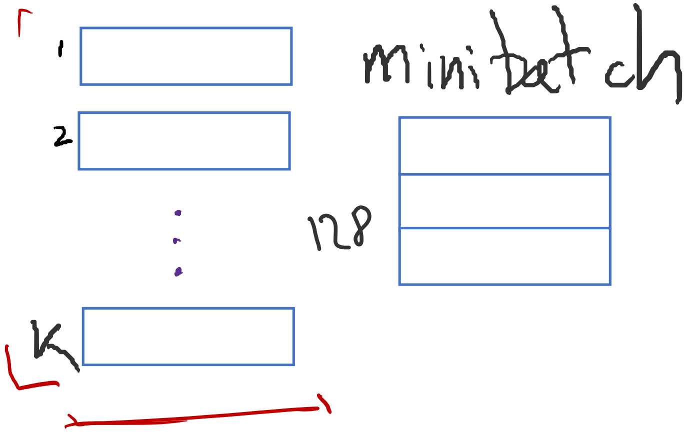


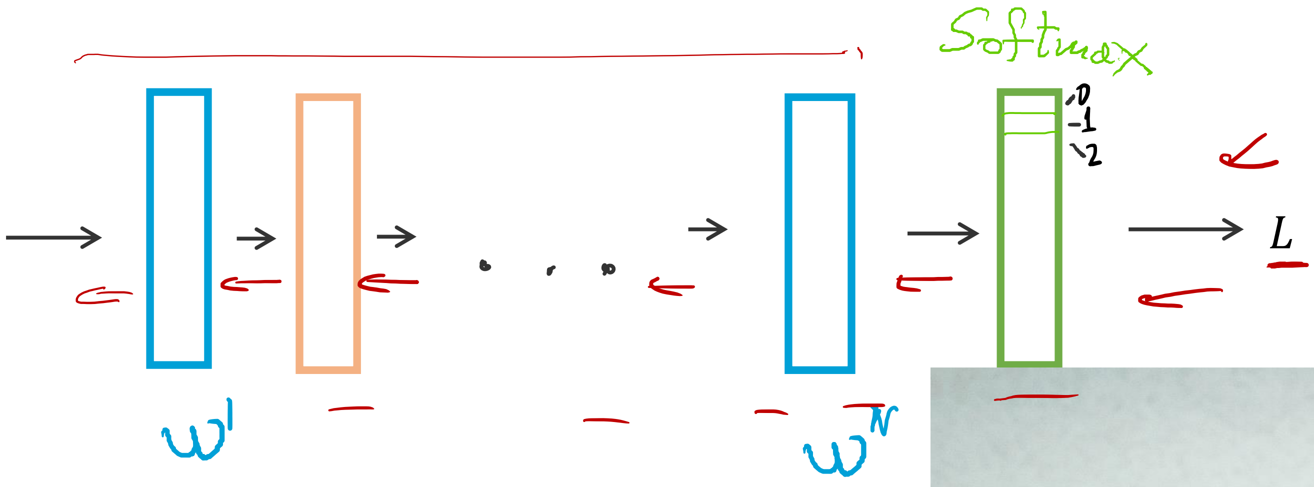

```python
class Layer:
    def forward(self, x):
        result = ... # промежуточные вычисления
        self.x = x # сохраняем значения, которые нам
                   # понадобятся при обратном прохоже
        return result

    def backward(self, grad):
        dx = ... # используем сохраненные значения чтобы
        dw = ... # вычислить градиент по x и по w
        self.w.grad += dw # аккумулируем градиент dw
        return dx
```

Общая схема тренировки

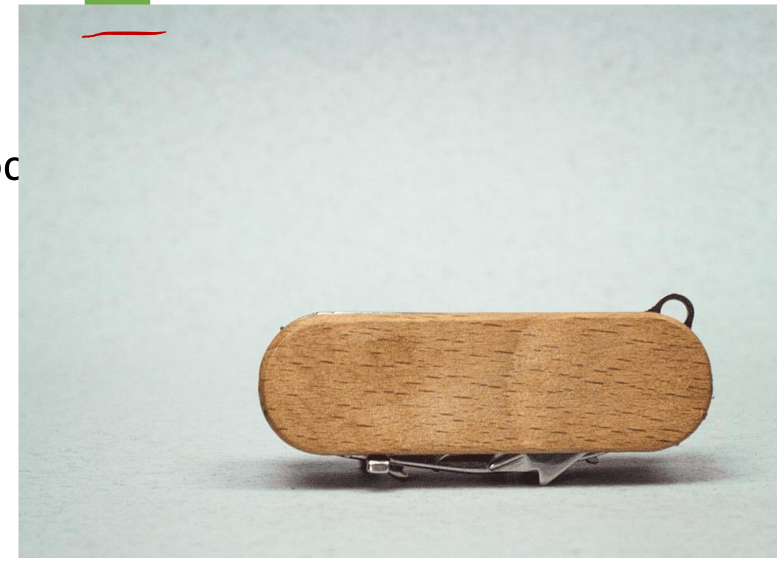
Прямой проход (forward pass) – посчитать loss

Взять mini-batch





Обратный проход (backward pass) – пос



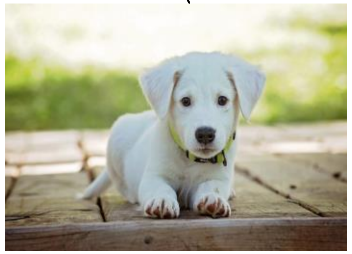
Обновить параметры

$$\begin{cases} \overrightarrow{w^1} = \overrightarrow{w^1} - \eta \overrightarrow{\nabla_{w^1}} L \\ \dots \\ \overrightarrow{w^n} = \overrightarrow{w^n} - \eta \overrightarrow{\nabla_{w^n}} L \end{cases}$$

Multi-class classification

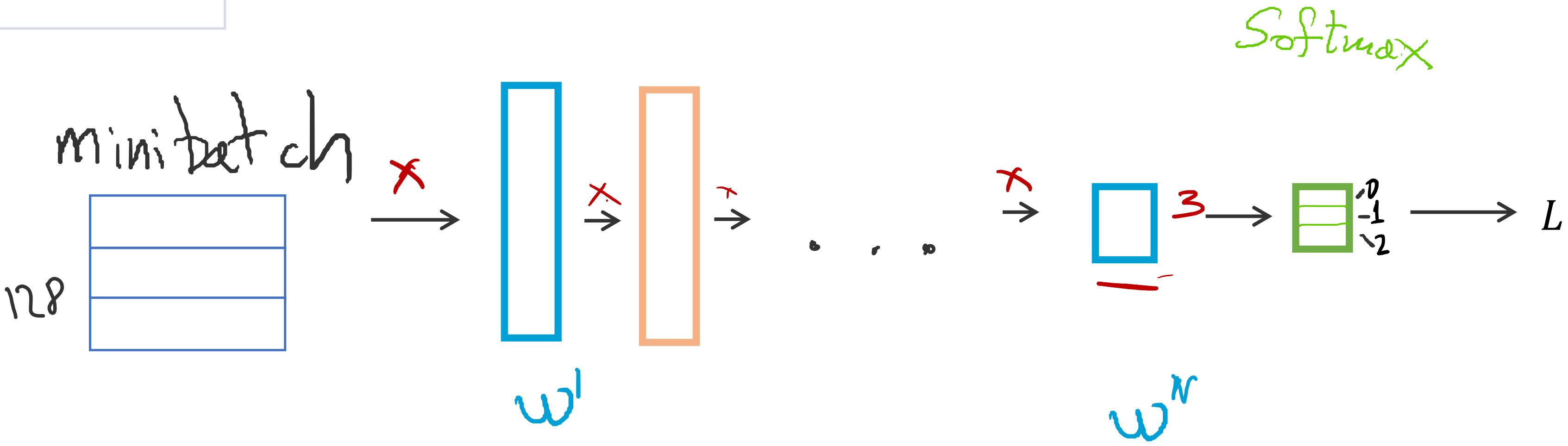
0

1



2

ground truth (gt)



$$p(C=0|x) = \frac{e^{y_0}}{e^{y_0}+e^{y_1}+\cdots+e^{y_n}} = \frac{e^{y_0}}{\sum_i e^{y_i}}$$

. . .

cross-entropy

$$L = -\frac{1}{n}\sum_{s} \ln p(c = gt_s|x_s)$$

Multi-class labeling

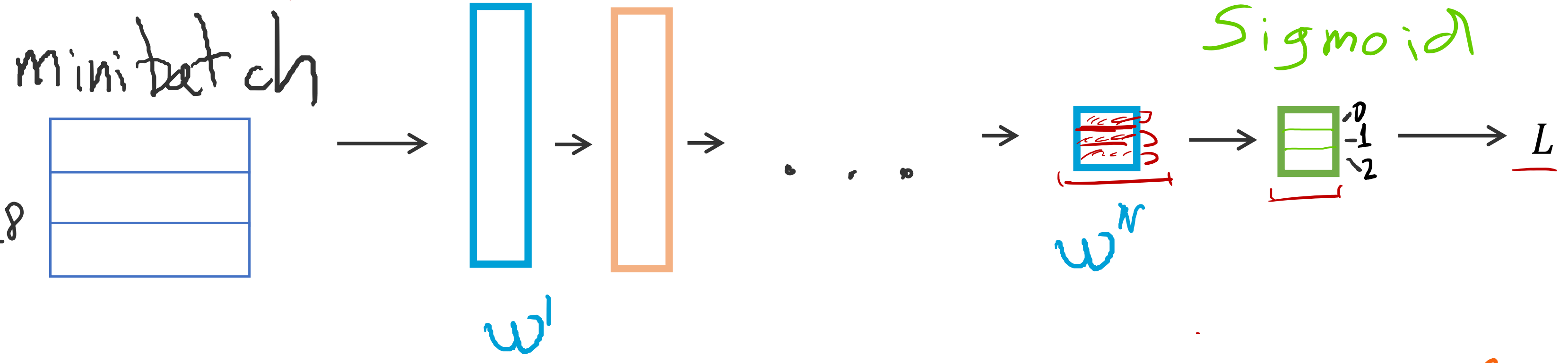
0/1 – binary

1
0
1

1
1
0

0
0
1

ground truth (gt)



Binary cross entropy loss

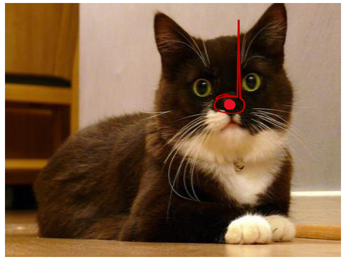
$$p(C = 0|x) = \frac{1}{1 + e^{-y_0}}$$

$$p(C = 1|x) = \frac{1}{1 + e^{-y_1}}$$

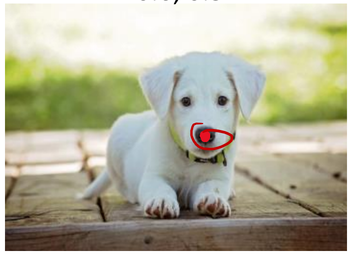
$$L = -\frac{1}{n}\sum_s \sum_c gt_c \ln p(c|x_s) + (1 - gt_c) \cdot \ln\left(1 - p(c|x_s)\right)$$

Regression

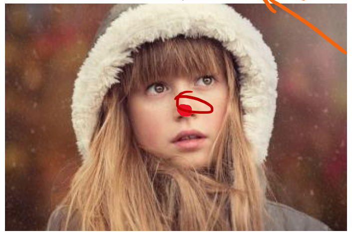
0.7, 0.3



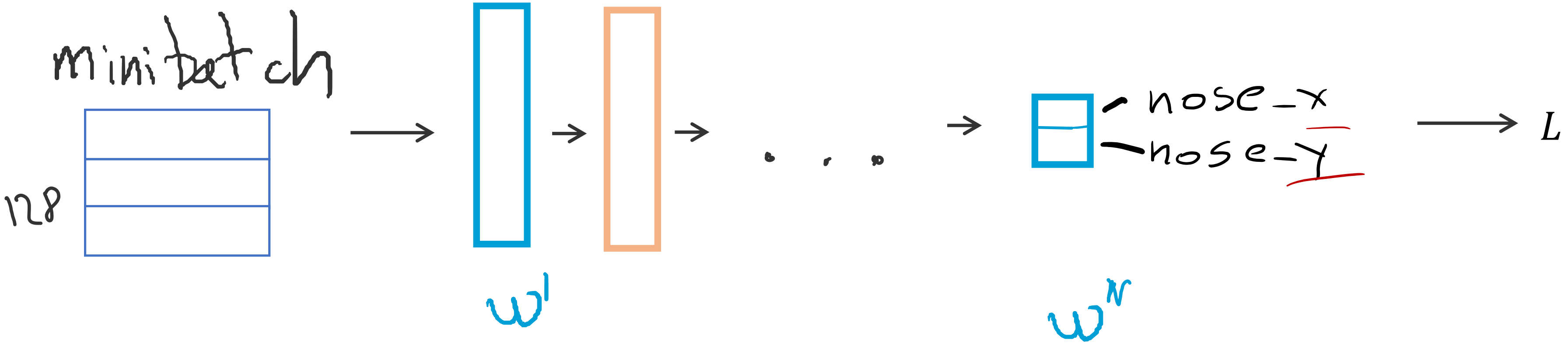
0.6, 0.5



0.5, 0.5



ground truth (gt)

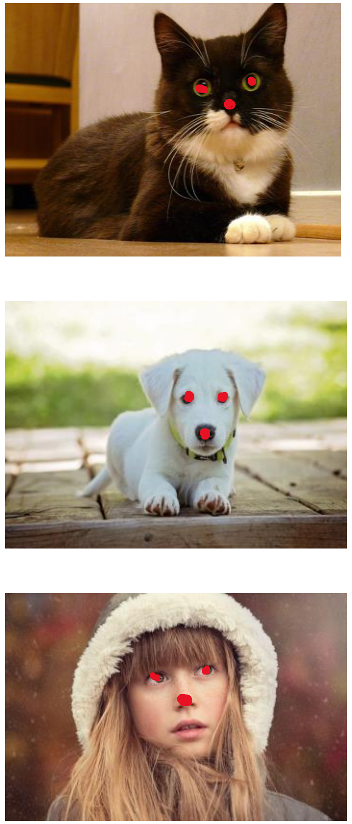


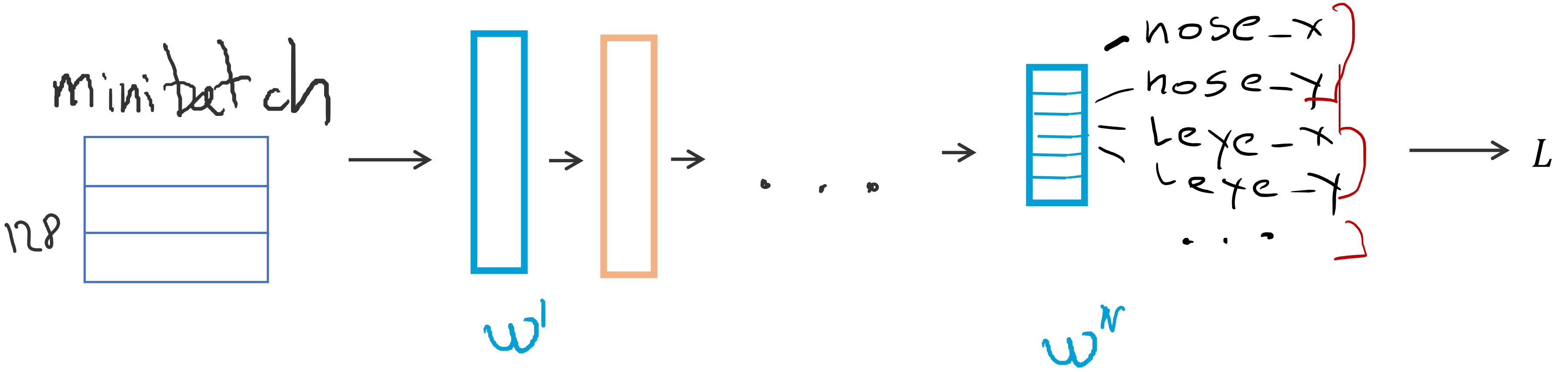
L1 loss

L2 loss

$$L = \frac{1}{n} \sum_{s} \sum_{i} |y_i - gt_i|^2$$

Regression

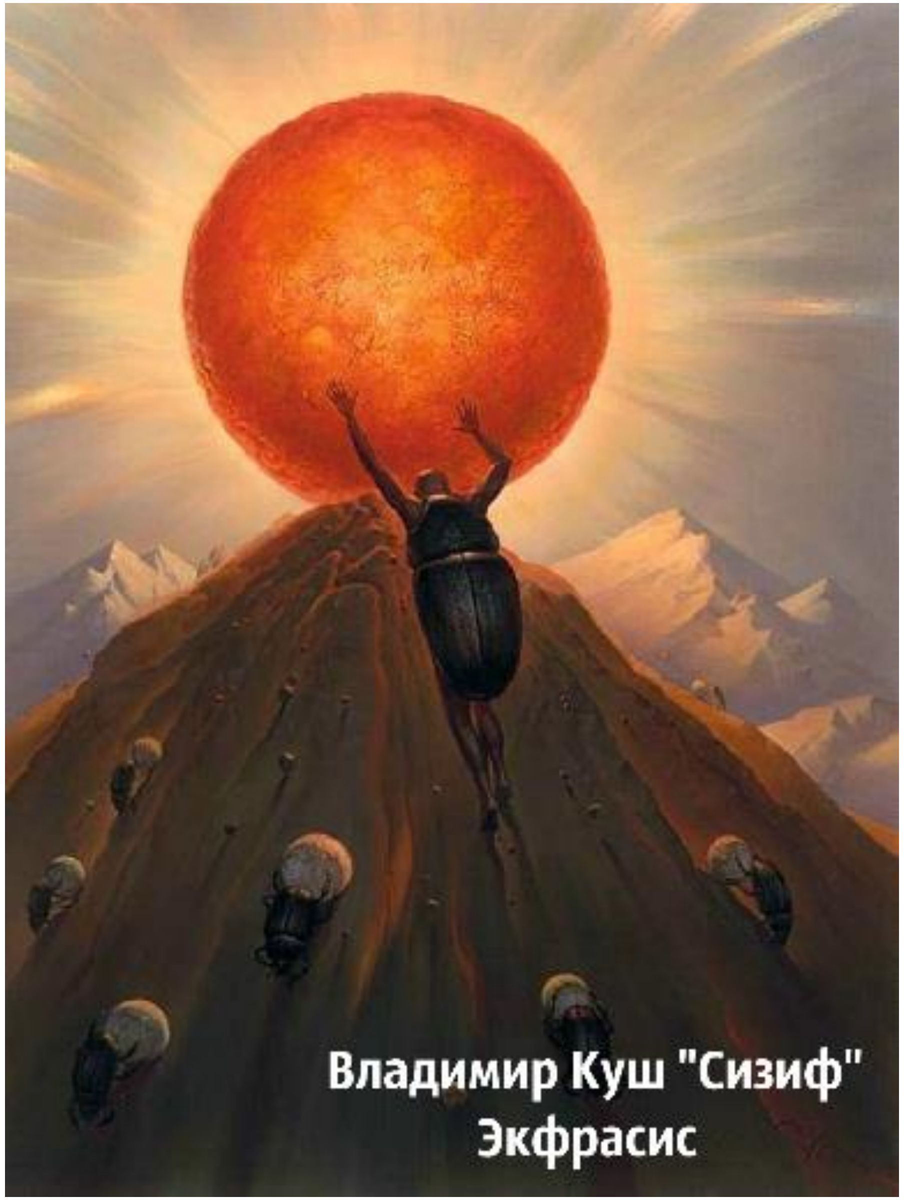




L1 loss

$$L = \frac{1}{n} \sum_{s} \sum_{i} |y_i - gt_i|$$

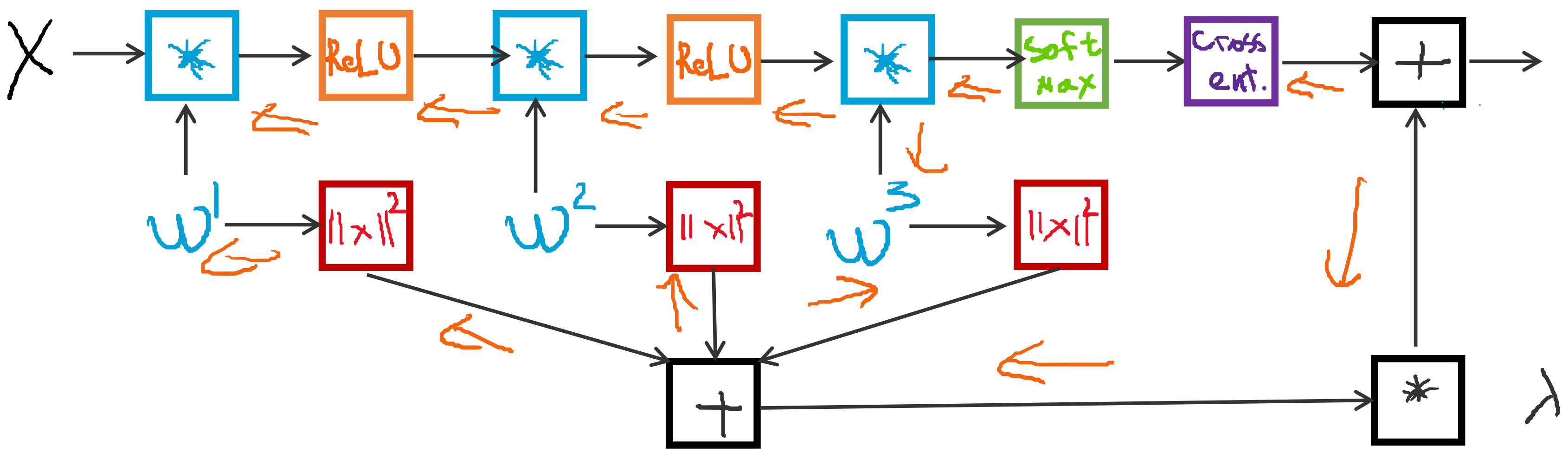
Восход солнца вручную



Библиотеки для глубокого обучения!
Deep Learning Frameworks!

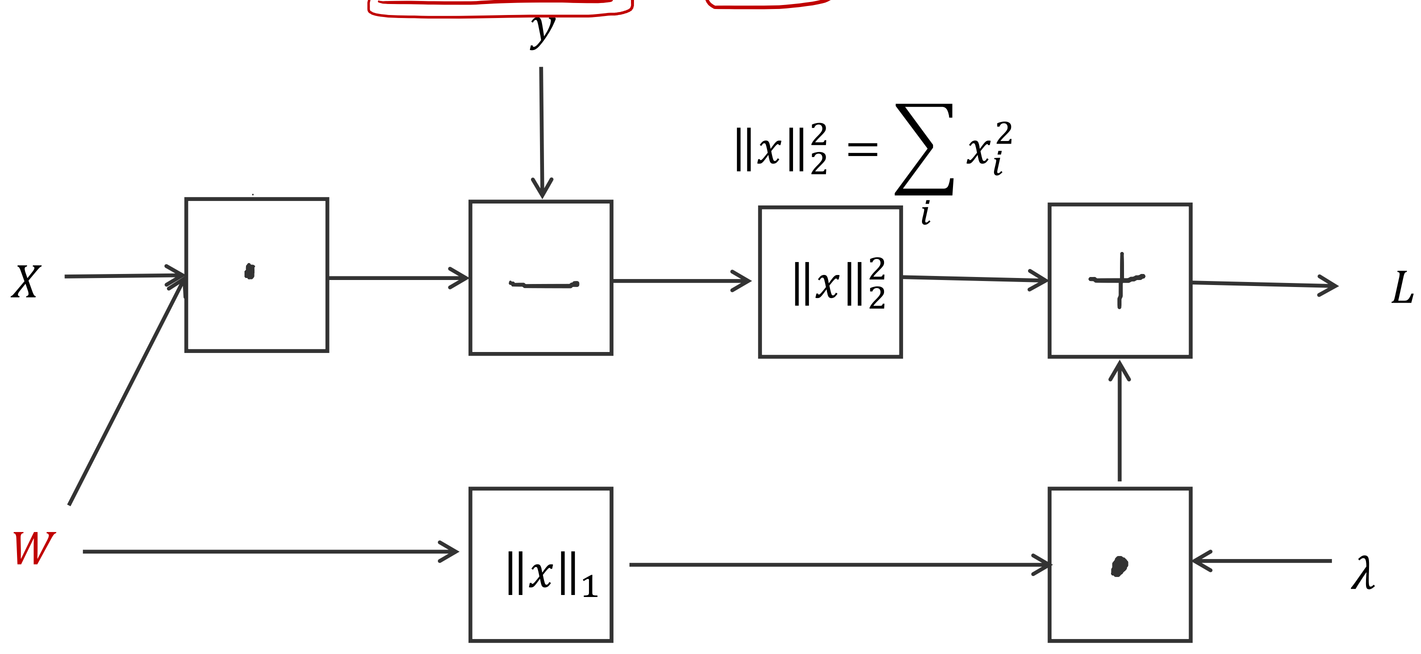
Что они позволяют

1. Задать граф вычислений
2. Посчитать градиенты на графе через backpropagation
3. Сделать это все на GPU



PyTorch: Tensors

$$L = \|X \cdot W - y\|_2^2 + \lambda\|W\|_1$$



$$\|x\|_2^2 = \sum_i x_i^2$$

$$\|x\|_1 = \sum_i |x_i|$$

```
np.random.seed(2)
Y = np.random.randn(2, 5)
X = np.random.randn(2, 3)
W = np.random.randn(3, 5)
lam = 0.5

pred = X.dot(W)
pred_loss = np.sum((pred - Y)**2)
reg_loss = lam * np.sum(np.abs(W))
loss = pred_loss + reg_loss
```

```
import torch
Y = torch.Tensor(np.random.randn(2, 5))
X = torch.Tensor(np.random.randn(2, 3))
W = torch.Tensor(np.random.randn(3, 5))
lam = 0.5
pred = X.matmul(W)
pred_loss = torch.sum((pred - Y)**2)
reg_loss = lam * torch.sum(torch.abs(W))
loss = pred_loss + reg_loss
```

PyTorch: autograd

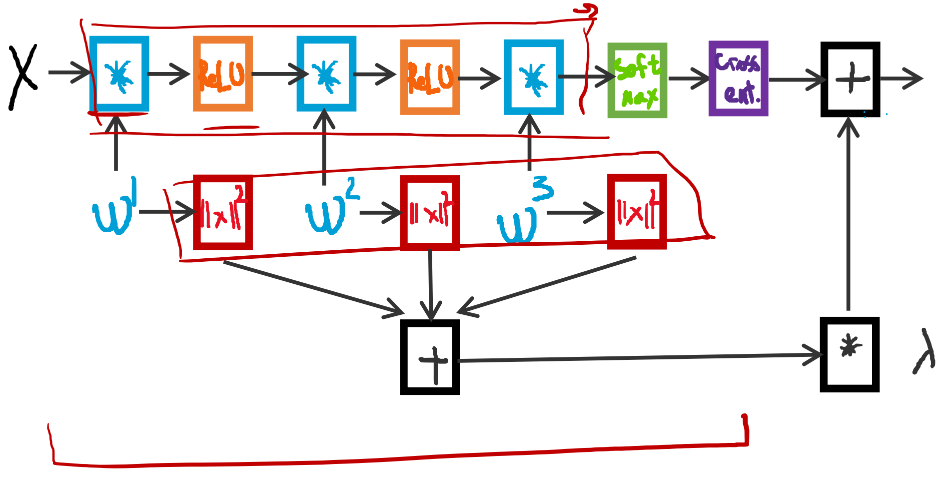
```python
import torch
Y = torch.Tensor(np.random.randn(2, 5))
X = torch.Tensor(np.random.randn(2, 3))
W = torch.Tensor(np.random.randn(3, 5))
lam = 0.5
pred = X.matmul(W)
pred_loss = torch.sum((pred - Y)**2)
reg_loss = lam * torch.sum(torch.abs(W))
loss = pred_loss + reg_loss
```

forward

```python
Y = torch.Tensor(np.random.randn(2, 5)).cuda()
X = torch.Tensor(np.random.randn(2, 3)).cuda()
W = torch.Tensor(np.random.randn(3, 5)).cuda().requires_grad_(True)
l = 0.5

for i in range(100):
    pred = X.matmul(W)
    pred_loss = torch.sum((pred - Y)**2)
    reg_loss = l * torch.sum(torch.abs(W))
    loss = pred_loss + reg_loss

    loss.backward()
    W.data.add_(-0.1*W.grad.data)
    W.grad.zero_()
```


BUT WAIT
THERE'S MORE
memegenerator.net

PyTorch: Modules




```
import torch.optim as optim
y = torch.LongTensor(np.random.randint(0,3,size=2))

nn = torch.nn.Sequential(
    torch.nn.Linear(3, 10),
    torch.nn.ReLU(),
    torch.nn.Linear(10, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 3)
)

optimizer = optim.SGD(nn.parameters(), lr=0.01, weight_decay=0.05)

for i in range(100):
    optimizer.zero_grad()
    pred = nn(X)
    criterion = torch.nn.CrossEntropyLoss()
    loss = criterion(pred, y)
    loss.backward()
    optimizer.step()
```

L2

PyTorch: Modules

```
nn = torch.nn.Sequential(
    torch.nn.Linear(3, 10),
    torch.nn.ReLU(),
    torch.nn.Linear(10, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 3))

nn(X)
```

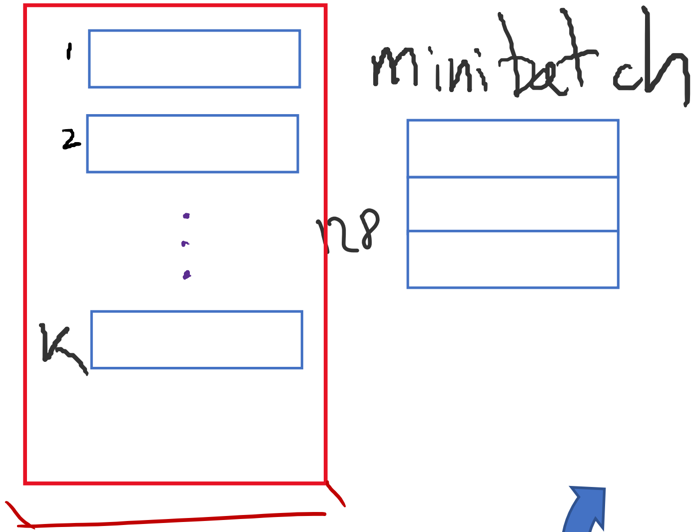
```
import torch.nn.functional as F

class Net(torch.nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = torch.nn.Linear(3, 10)
        self.fc2 = torch.nn.Linear(10, 20)
        self.fc3 = torch.nn.Linear(20, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

nn = Net()
nn(X)
```

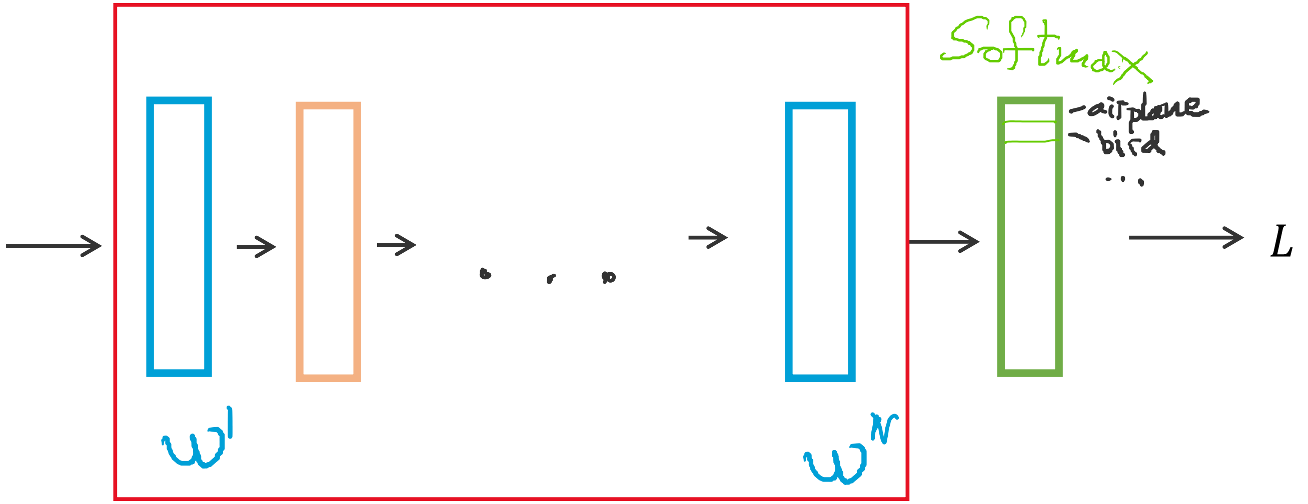

WE NEED TO GO
DEEPER

Погружаемся в детали

Прямой проход (forward pass) – посчитать loss

Взять mini-batch





Обратный проход (backward pass) – посчитать градиент

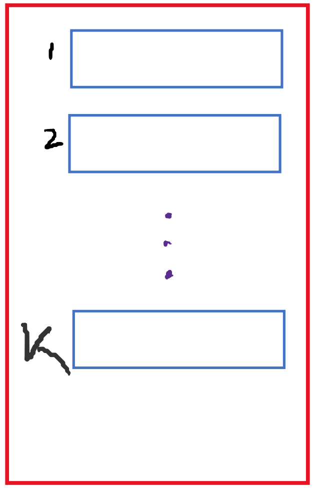
Обновить параметры

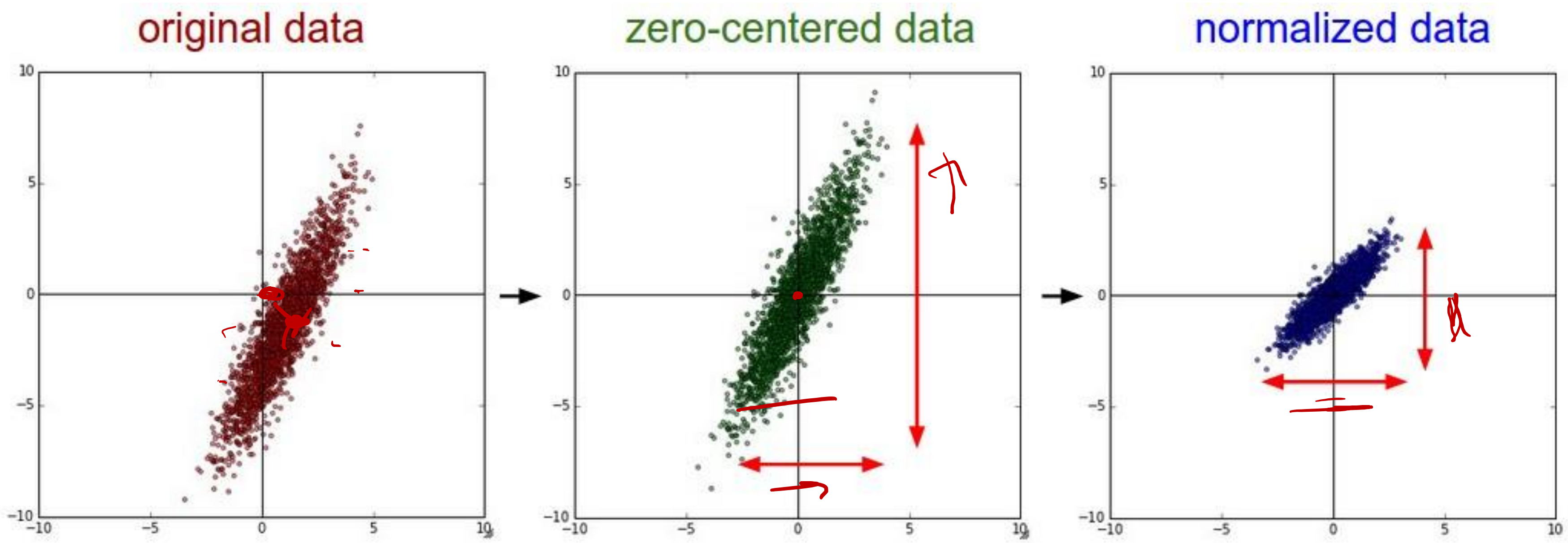
$$\overrightarrow{w^1} = \overrightarrow{w^1} - \eta \overrightarrow{\nabla_{w^1}} L$$

…

$$\overrightarrow{w^n} = \overrightarrow{w^n} - \eta \overrightarrow{\nabla_{w^n}} L$$

Подготовка данных
Data preprocessing





$$x' = x - \bar{x}$$

среднее

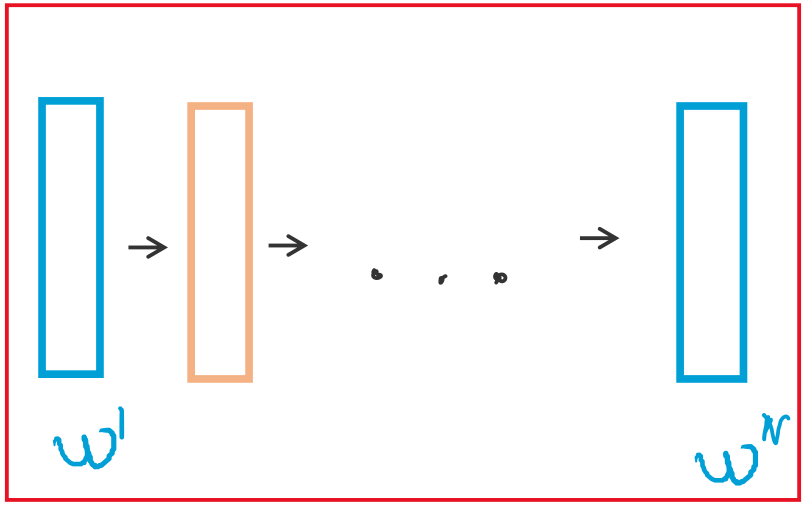
$$x' = \frac{x - \bar{x}}{\sigma}$$

std

$$\sigma = \frac{\sqrt{(x - \bar{x})^2}}{N - 1}$$

http://cs231n.github.io/

Детали работы сети

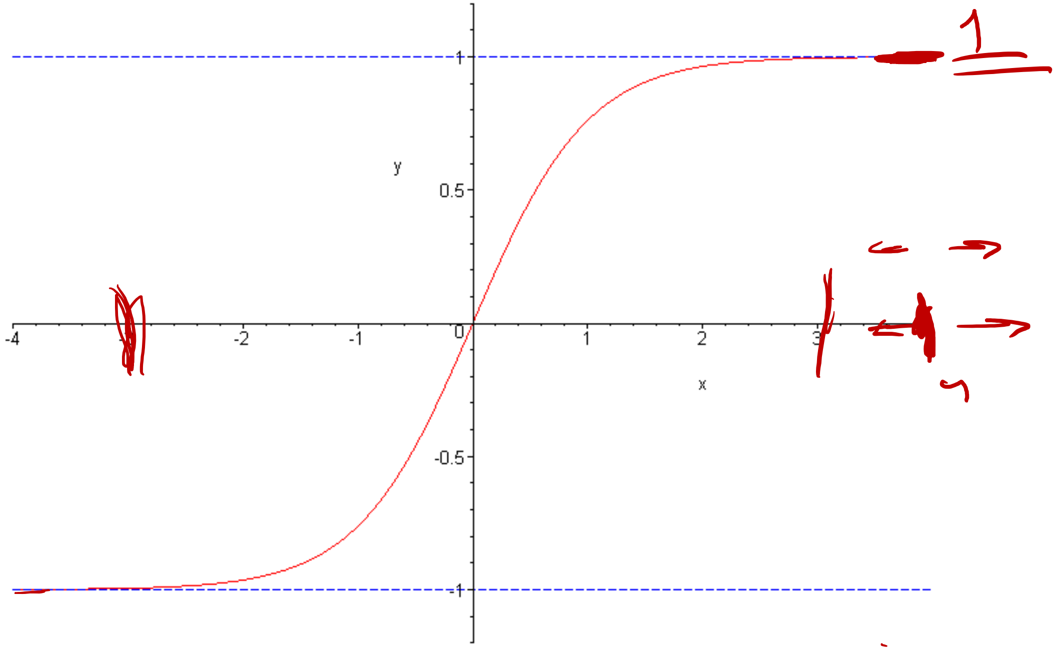


- Выбор активационной функции

- Инициализация весов

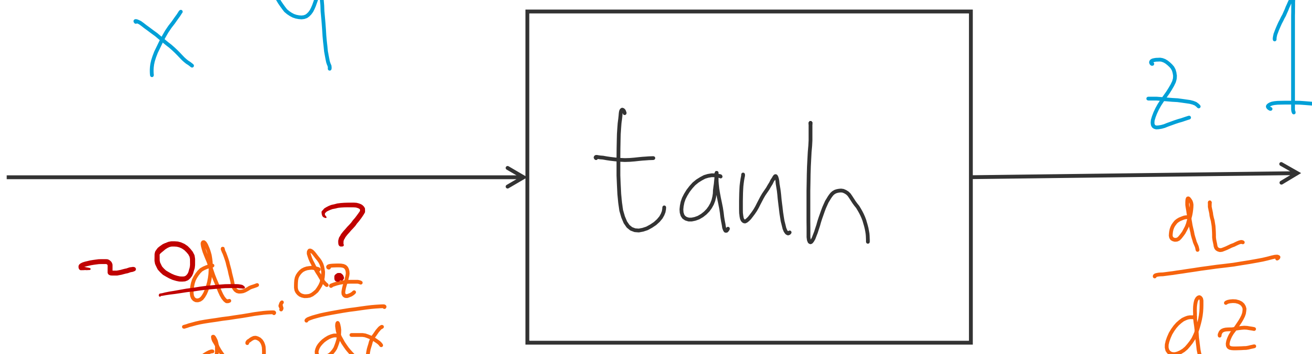
Проблема: Vanishing gradients

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



x y

z 1



$\sim 0 \quad \frac{dL}{dz} \cdot \frac{dz}{dx}$?

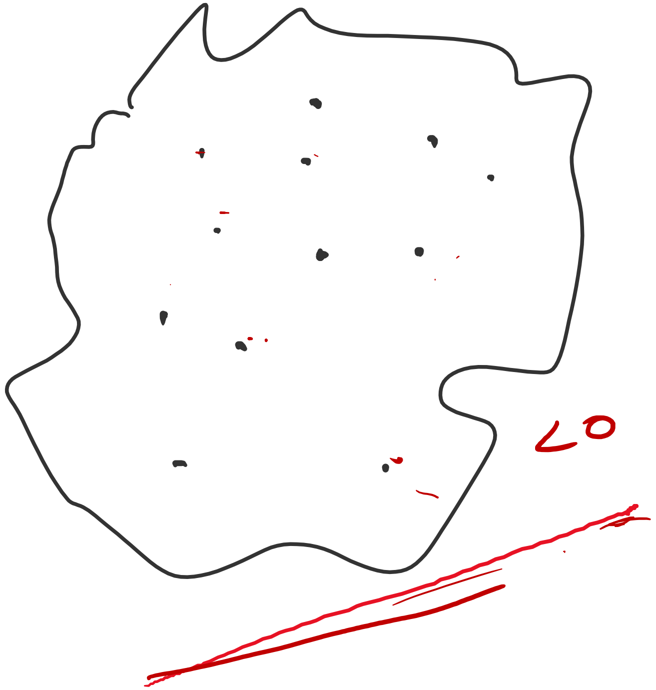
$\frac{dL}{dz}$ 1

Rectifier Linear Unit
ReLU

U

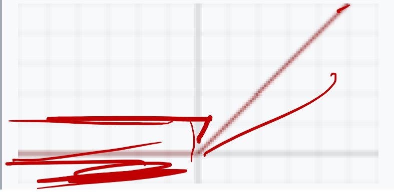
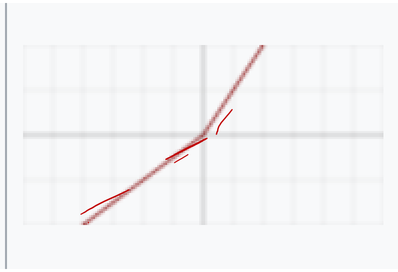
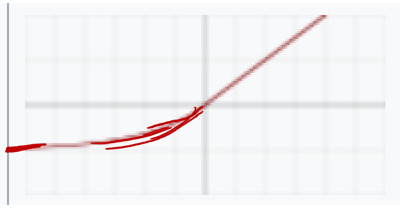
[Wikipedia](#)

Data 10^6



20

ReLU

Rectified linear unit (ReLU)[10]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[11]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Exponential linear unit (ELU)[14]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

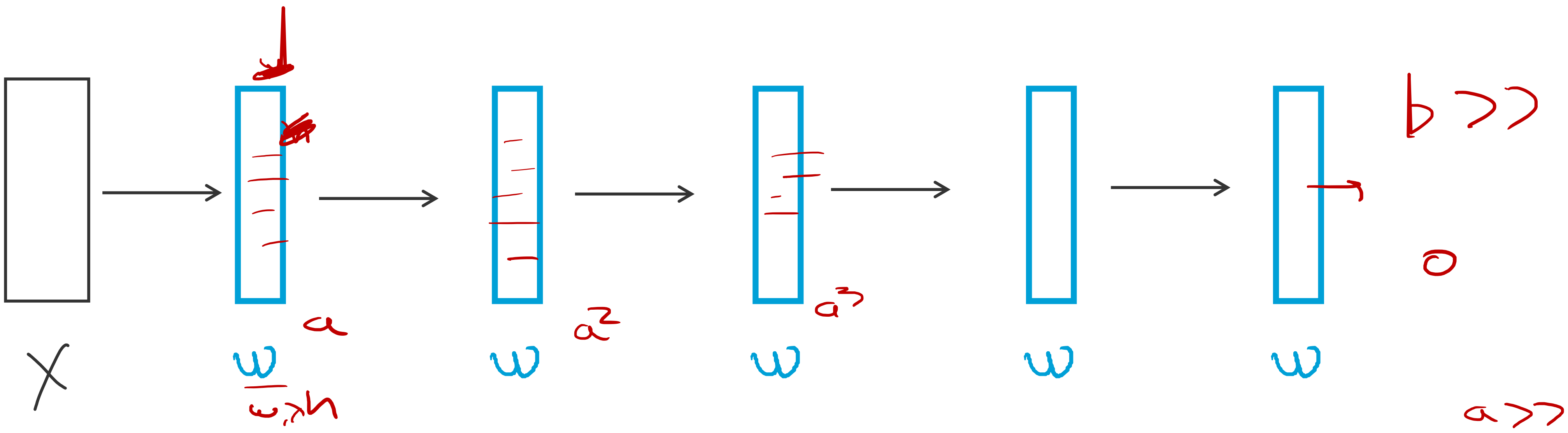
[Rectifier Nonlinearities Improve Neural Network Acoustic Models'13](#)

[Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)'15](#)

Инициализация весов
Weight initialization

0-1

```
W = a * random(width, height)
```



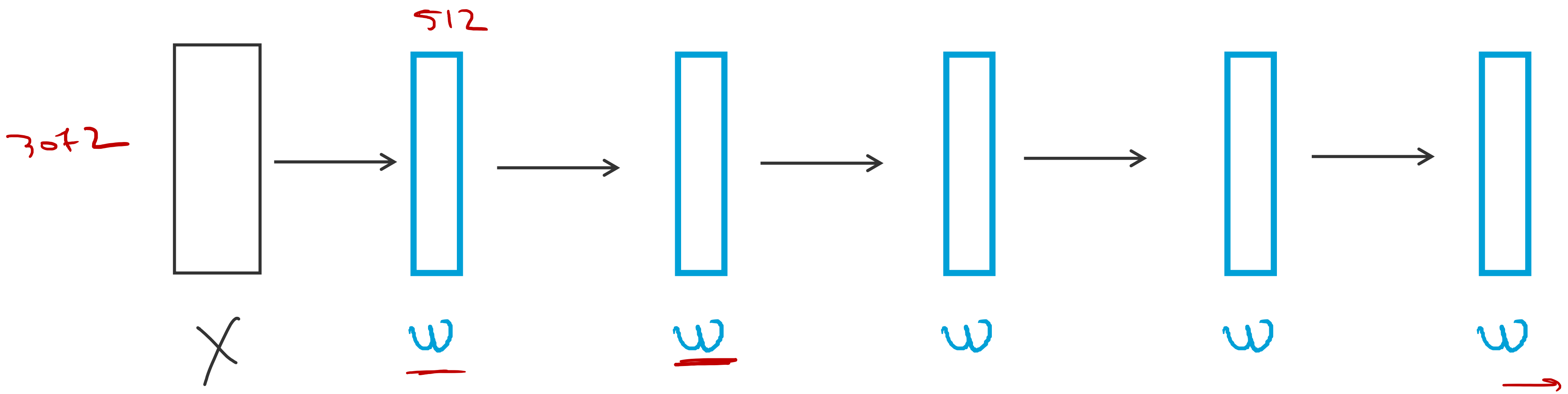
a>>1

a<<1

Как выбрать a?

Xavier initialization

```
a = 1 / sqrt(in_num)
W = a * random(in_num, out_num)
```

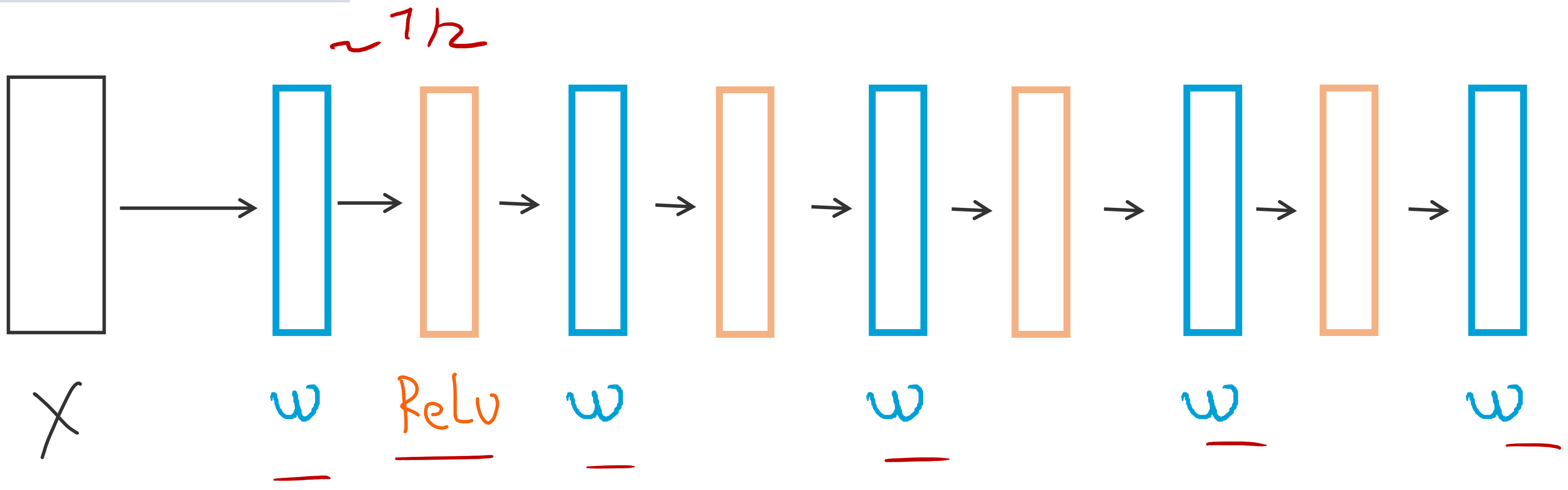



```
torch.nn.init.xavier_normal_(
```

[Understanding the difficulty of training deep feedforward neural networks'10]

He initialization

```
a = 1 / sqrt(in_num / 2)
W = a * random(in_num, out_num)
```

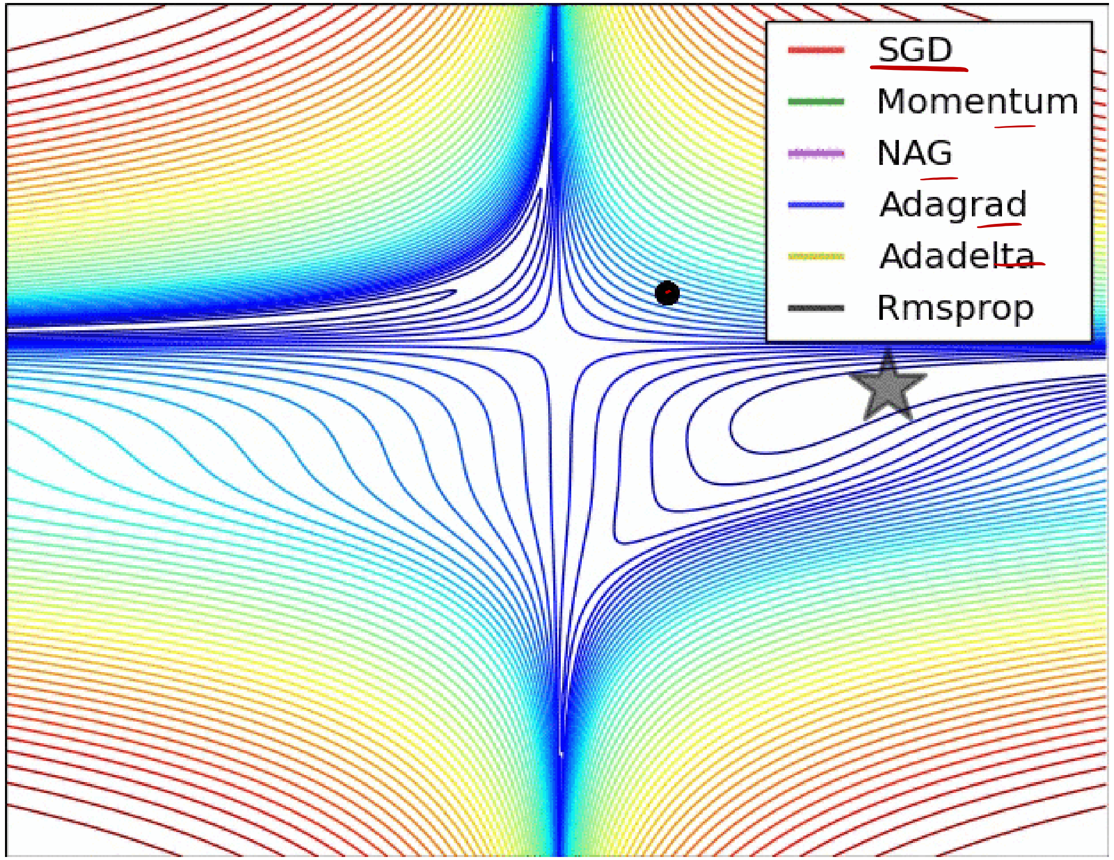



```
torch.nn.init.kaiming_normal_(
```

[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification’15]()

Обновление параметров
Update rules

```
w = w - learning_rate * gradient
```



Обновить параметры

$$\overrightarrow{w^1} = \overrightarrow{w^1} - \eta\overrightarrow{\nabla_{w^1}}L$$

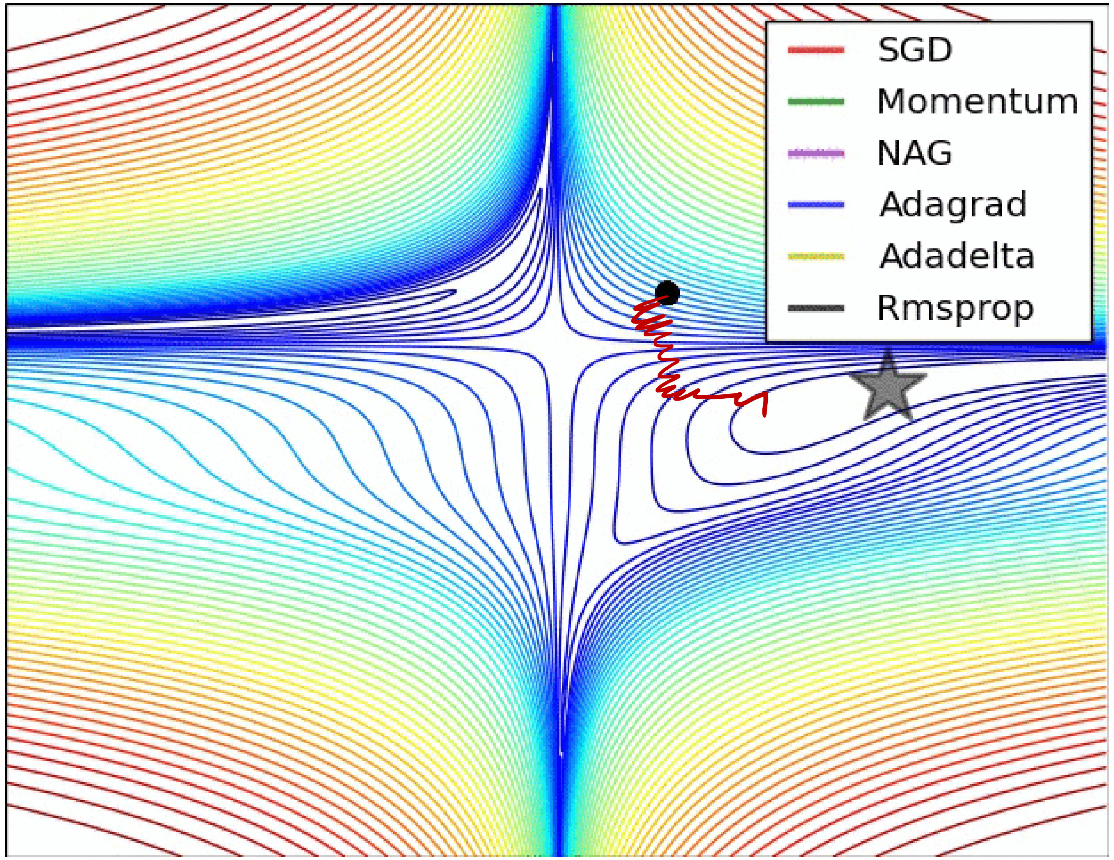
…

$$\overrightarrow{w^n} = \overrightarrow{w^n} - \eta\overrightarrow{\nabla_{w^n}}L$$

Momentum (импульс)

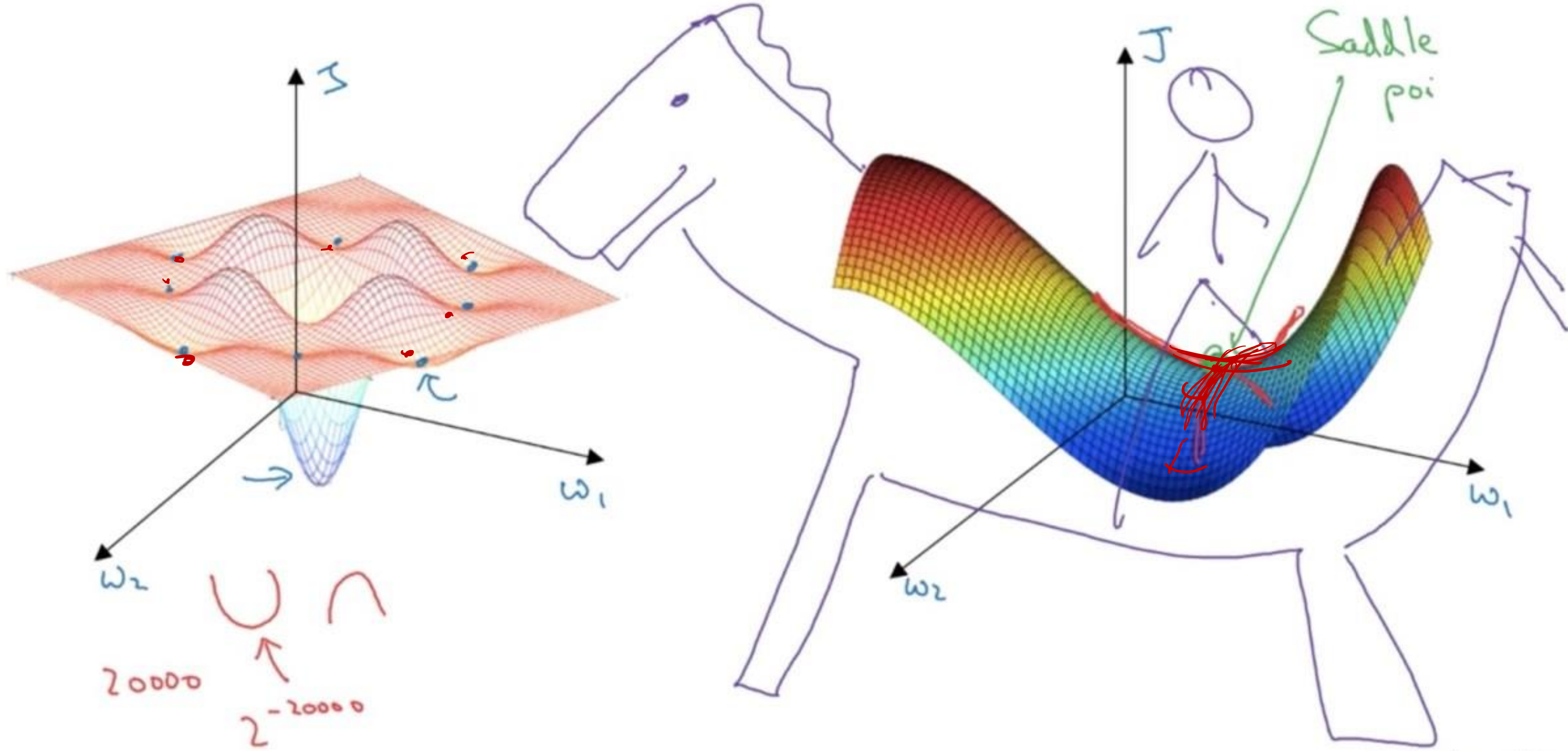
0.9-0.99

```
velocity = momentum * velocity - learning_rate * gradient
w = w + velocity
```




```
torch.optim.SGD(
```

Local optima in neural networks

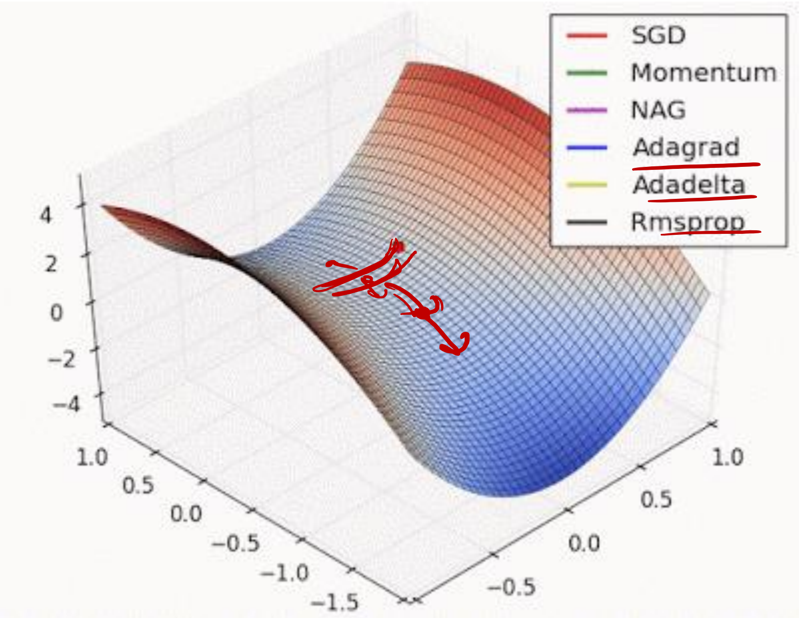


Andrew Ng

[Deep Learning on Coursera]

Adagrad

```
accumulated += gradient ** 2
adaptive_learning_rate = learning_rate / sqrt(accumulated)

w := w - adaptive_learning_rate * gradient
```




```
torch.optim.Adagrad(
```

[Adaptive Subgradient Methods for Online Learning and Stochastic Optimization’11]

RMSProp

`torch.optim.RMSprop(`

0.9 - 0.99

```
accumulated = rho * accumulated + (1-rho) * gradient ** 2
adaptive_learning_rate = learning_rate / sqrt(accumulated)

w = w - adaptive_learning_rate * gradient
```

[Neural Networks for Machine Learning, Lecture 6.5 – rmsprop'12]

RMS Prop
+ Momentum

Рекомендуем

Adam

`torch.optim.Adam(`

```
velocity = beta1 * velocity+ (1-beta1) * gradient
accumulated = beta2 * accumulated + (1-beta2) * gradient ** 2

adaptive_learning_rate = learning_rate / sqrt(accumulated)

w = w - adaptive_learning_rate * velocity
```

[Adam: A Method for Stochastic Optimization'14]

В следующий раз