# Neural Networks p1

# Recap Loss Functions

# Loss Functions

Дано:

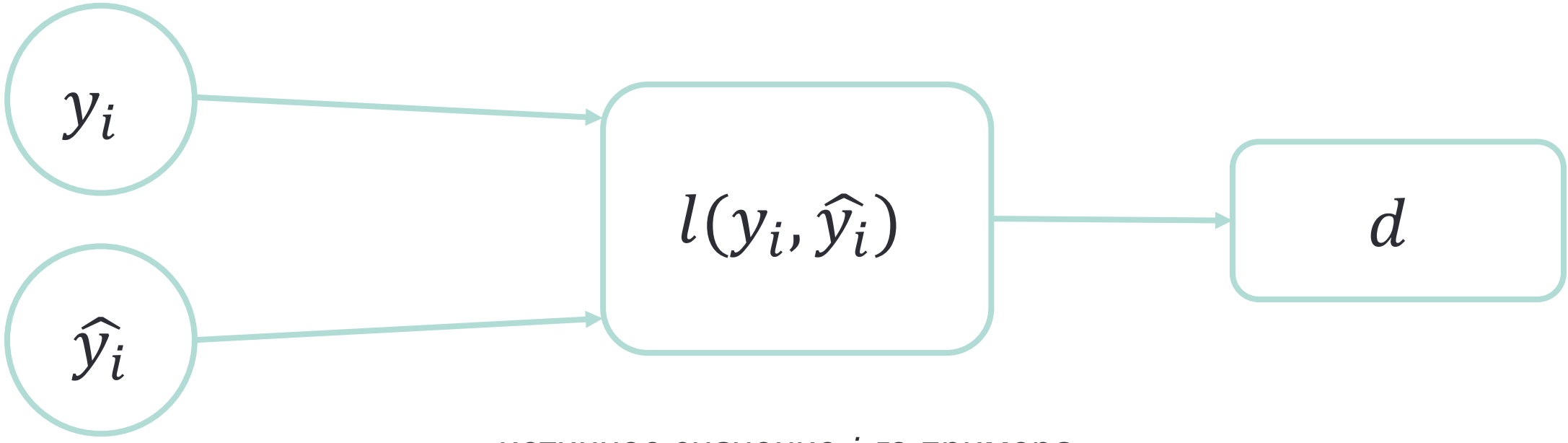
$y$ - истинное значение

$\hat{y}$ - предсказание

Задача:

$\hat{y} \to y$

# Loss Functions: One Data Point



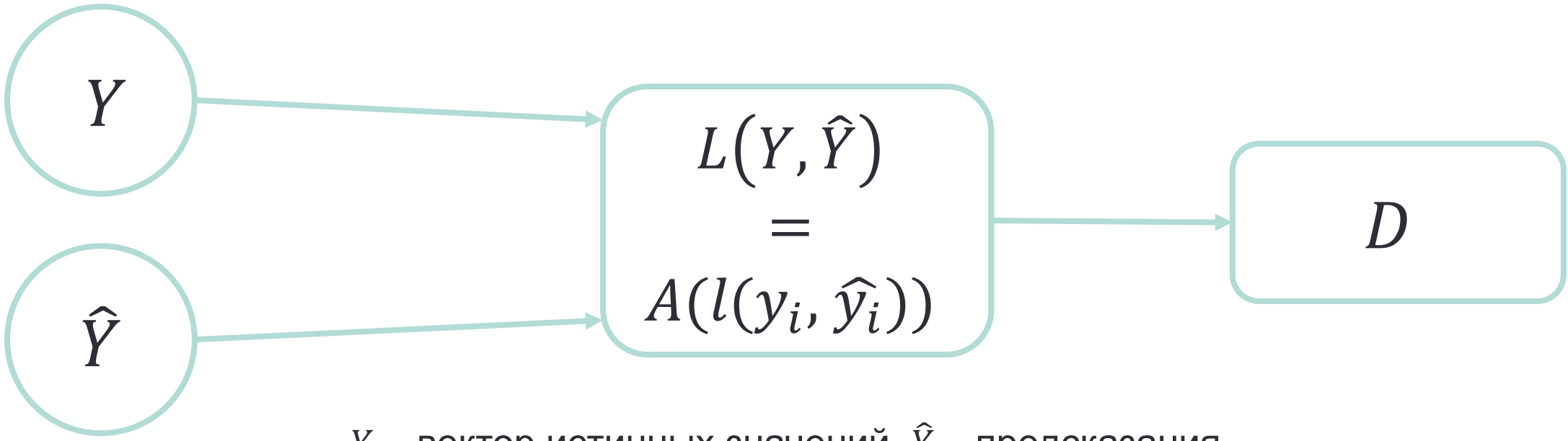

$y_i$ – истинное значение $i$-го примера

$\hat{y}_i$ – предсказание $i$-го примера

$l(y_i, \hat{y}_i)$ – функция потерь на $i$-м примере

$d$ – число, мера близости между $y_i$ и $\hat{y}_i$

# Loss Functions: Many Data Points




$Y$ – вектор истинных значений, $\hat{Y}$ – предсказания

$L(Y, \hat{Y})$ – функция потерь на мн-ве примеров

$A$ – функция агрегации

$D$ – мера близости между $Y$ и $\hat{Y}$

$l(y_i, \hat{y}_i)$ – функция потерь на $i$-м примере

# Loss Functions Examples

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

$$A(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{k} y_{i,c} \log \widehat{y_{i,c}}$$

$$l(y_i, \hat{y}_i) = -\sum_{c=1}^{k} y_{i,c} \log \widehat{y_{i,c}}$$

$$A(x) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

# Loss Functions Examples: Honorable Mentions

## Regression

Mean Absolute Error

Mean Bias Error

Mean Absolute Percentage Error

…

## Classification

Hinge Loss

Exponential Loss

Generalized Smooth Hinge Loss

…

# Recap Linear/Logistic Regression

# Linear Regression

$$\hat{y}_i = x_i{}^T W$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i{}^T W)^2 \to 0$$

$x_i$ – $i$-ый пример из выборки ($m$ признаков на 1)

$W$ – вектор весов ($m$ признаков на 1)

$y_i$ – истинное значение $i$-го примера

$\hat{y}_i$ – предсказание $i$-го примера

# Linear Regression Vectorized Form

$$\hat{Y} = X^T W$$

$$MSE = \frac{1}{n}(Y - X^T W)^T (Y - X^T W) \to 0$$

$X$ – выборка (*m* признаков на *n* примеров)

$W$ – вектор весов (*m* признаков на 1)

$Y$ – вектор истинных значений (*n* примеров на 1)

$\hat{Y}$ – предсказания (*n* примеров на 1)

# Linear Regression Solution: Linear Equations

$$MSE = \frac{1}{n}(Y - X^T W)^T (Y - X^T W) \to 0$$

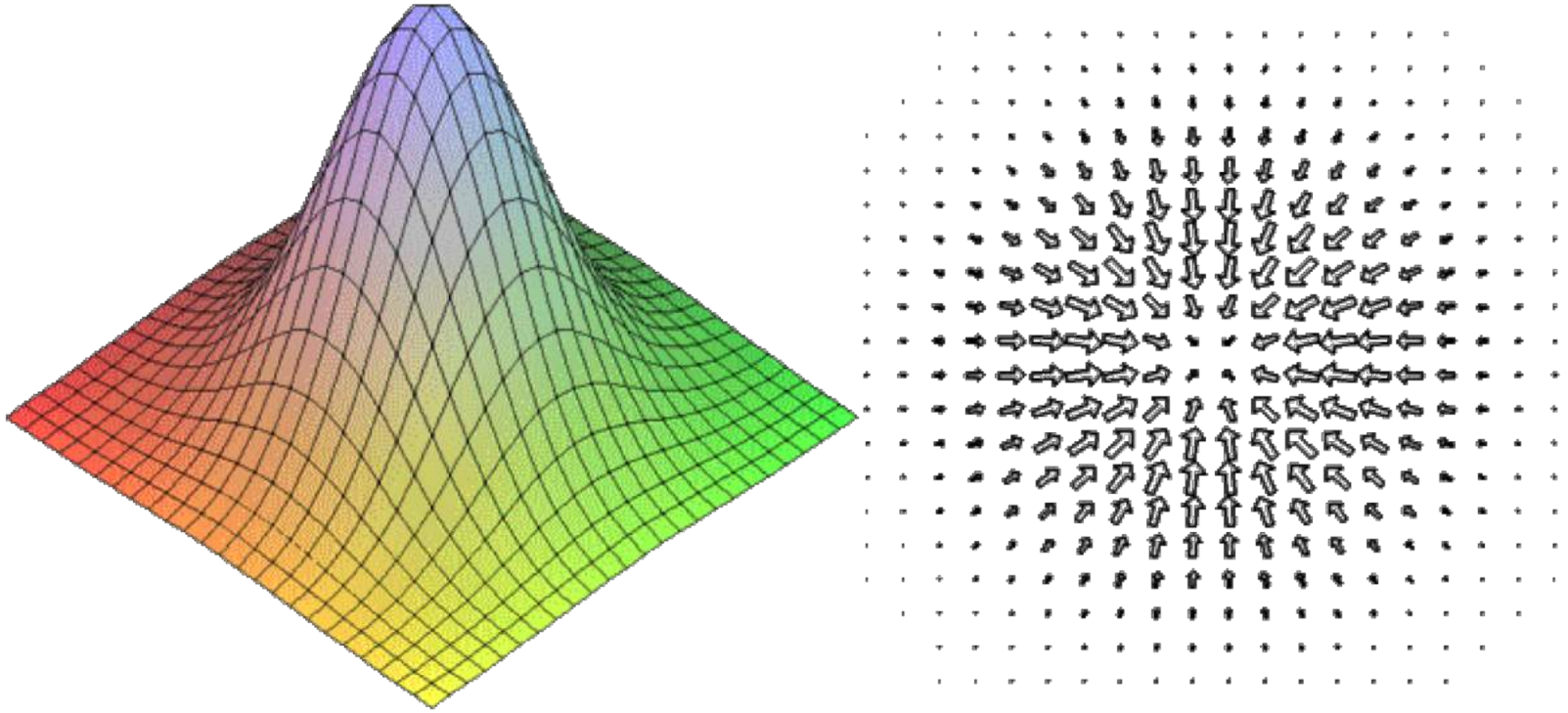
Как подобрать матрицу $W$?

$$W = (X^T X)^{-1} X^T Y$$

В чем проблема?

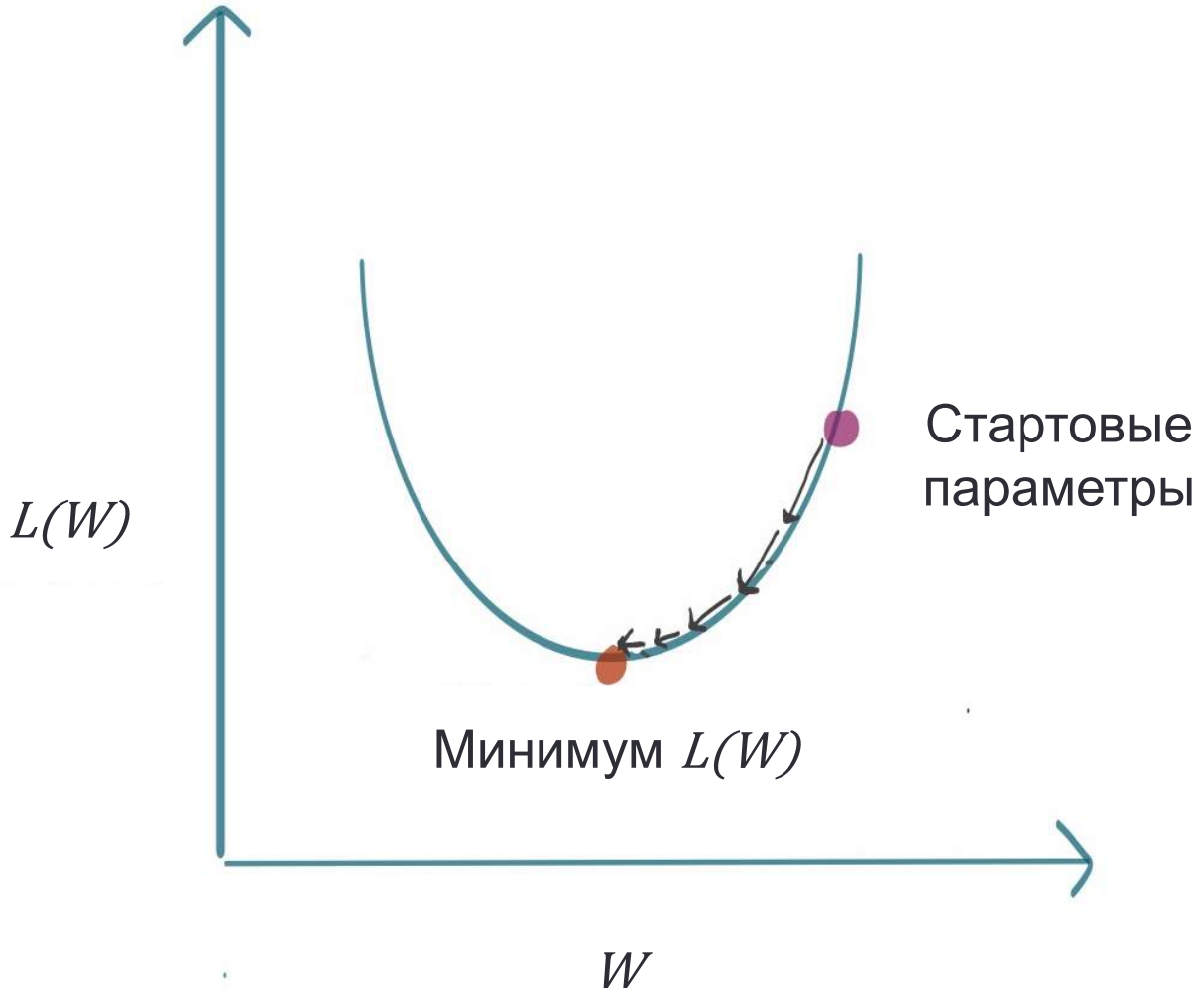
Иногда нельзя взять обратную матрицу от $X^T X$ (e.g. linear dependence)

# Recap: Gradient Descent



Градиент – направление наискорейшего роста функции

# Recap: Gradient Descent




Обновление параметров $W$

$$W_{t+1} = W_t - \alpha \nabla L(W_t)$$

$L(W_t)$ – ф-я с параметрами $W_t$

$\nabla L(W_t)$ – градиент $L(W_t)$

$-\nabla L(W_t)$ – антиградиент $L(W_t)$

$\alpha$ – learning rate

$t$ – момент времени

# Linear Regression Solution: Gradient Descent

$$L(W) = \frac{1}{n}(Y - X^T W)^T (Y - X^T W) \rightarrow 0$$

$$W_{t+1} = W_t - \alpha(\underbrace{\frac{2}{n} X (X^T W - Y)}_{\nabla L(W)})$$

# Linear Regression Solution: Gradient Descent

$$W_{t+1} = W_t - \alpha\left(\frac{2}{n} X \left(X^T W - Y\right)\right)$$

Как подобрать матрицу $W$?

Повторяя обновление параметров до сходимости

---

В чем проблема?

Иногда вся выборка не помещается в память

Иногда сходится слишком долго

# Linear Regression Solution: SGD

Разобьем выборку $X$ на $h$ подвыборок $B$

$$X = \{B_1, \dots, B_h\}$$

$$W_{t+1} = W_t - \alpha\left(\frac{2}{n} B_i \left(B_i{}^T W - Y\right)\right)$$

# Linear Regression Solution: SGD

$$W_{t+1} = W_t - \alpha\left(\frac{2}{n} B_i \left(B_i{}^T W - Y\right)\right)$$

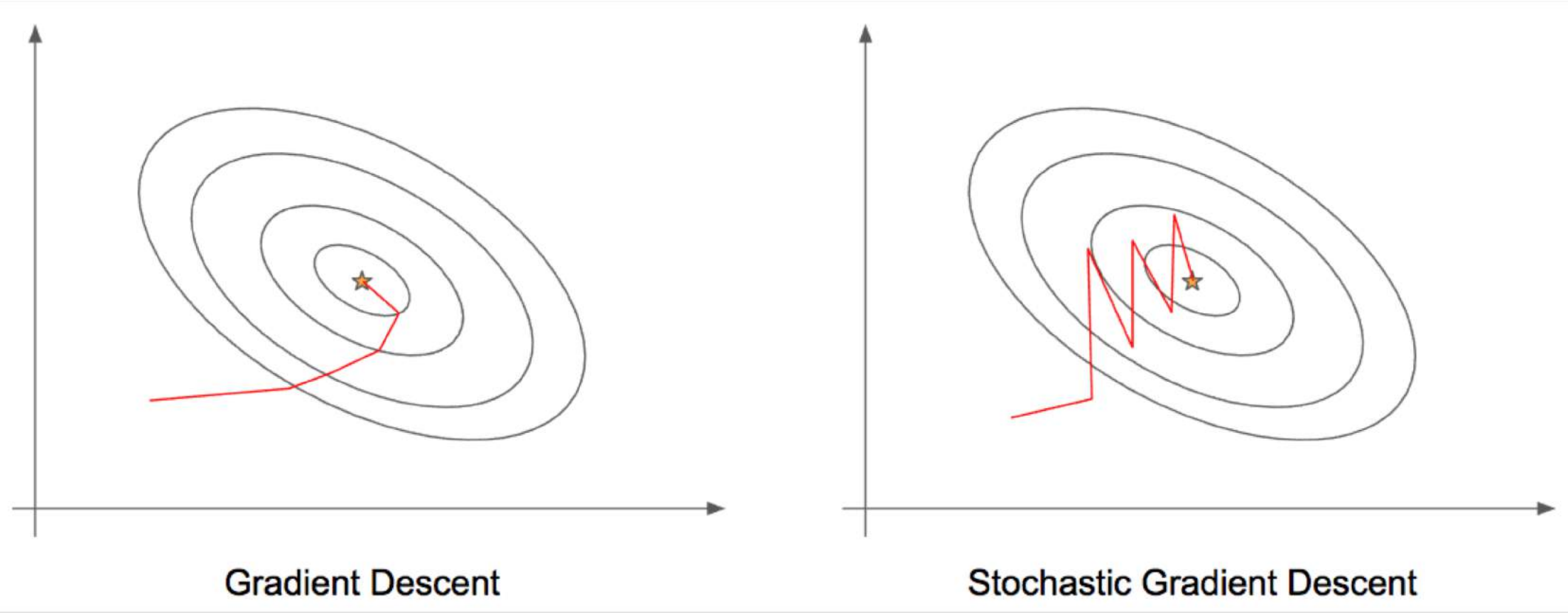
Как подобрать матрицу $W$?

Повторяя обновление параметров до сходимости

В чем проблема?

Обновления параметров могут быть нестабильными

# Linear Regression Solution: SGD



# Binary Logistic Regression

$$\hat{y}_i = \frac{1}{1 + e^{-x_i{}^T W}}$$

$$CE = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)) \to 0$$

$x_i$ – один пример из выборки (*m* признаков на 1)

$W$ – вектор весов (*m* признаков на 1)

# Binary Logistic Regression Vectorized Form

$$\hat{Y} = \frac{1}{1 + e^{-X^T W}}$$

$$CE = -\frac{1}{n}(Y^T \log \hat{Y} + (1 - Y)^T \log(1 - \hat{Y})) \rightarrow 0$$

$X$ – выборка (*m* признаков на *n* примеров)

$W$ – вектор весов (*m* признаков на 1)

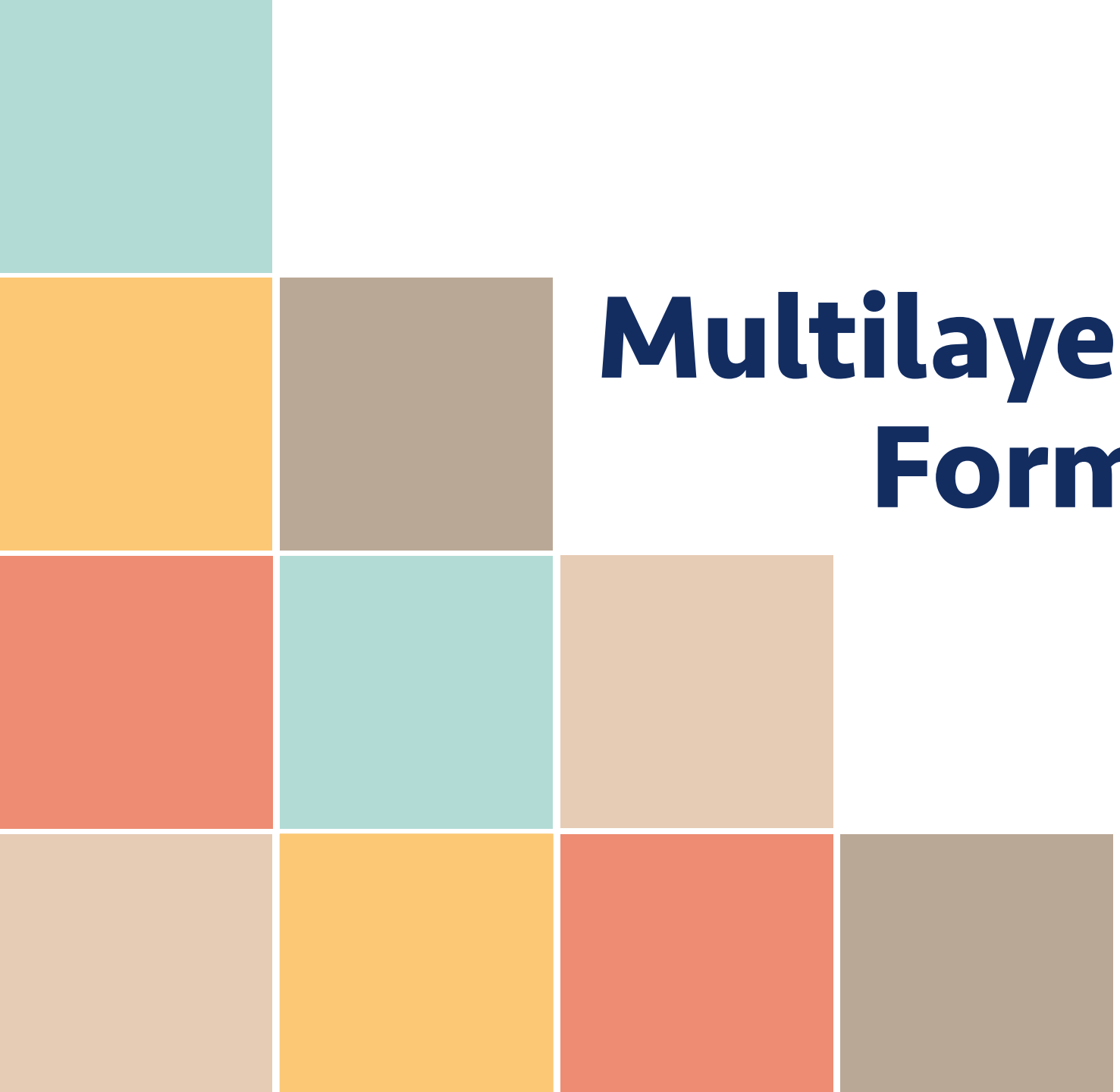
# Logistic Regression Solution: Linear Equations

# Logistic Regression Solution: Gradient Descent

$$\hat{Y} = \frac{1}{1 + e^{-X^T W}}$$

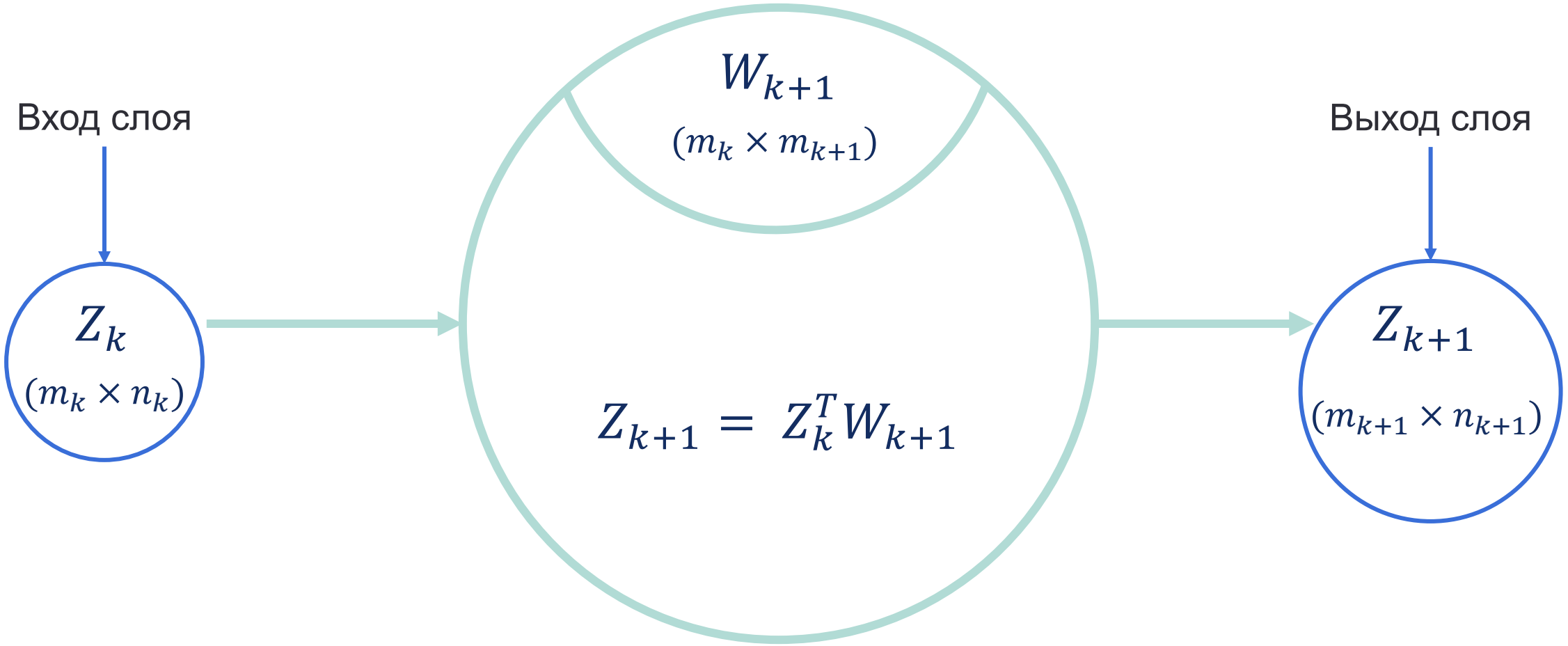
$$L(W) = -\frac{1}{n}(Y^T \log \hat{Y} + (1 - Y)^T \log(1 - \hat{Y})) \rightarrow 0$$

$$W_{t+1} = W_t - \alpha(\underbrace{\frac{1}{n} X \left(\hat{Y} - Y\right)}_{\nabla L(W)})$$

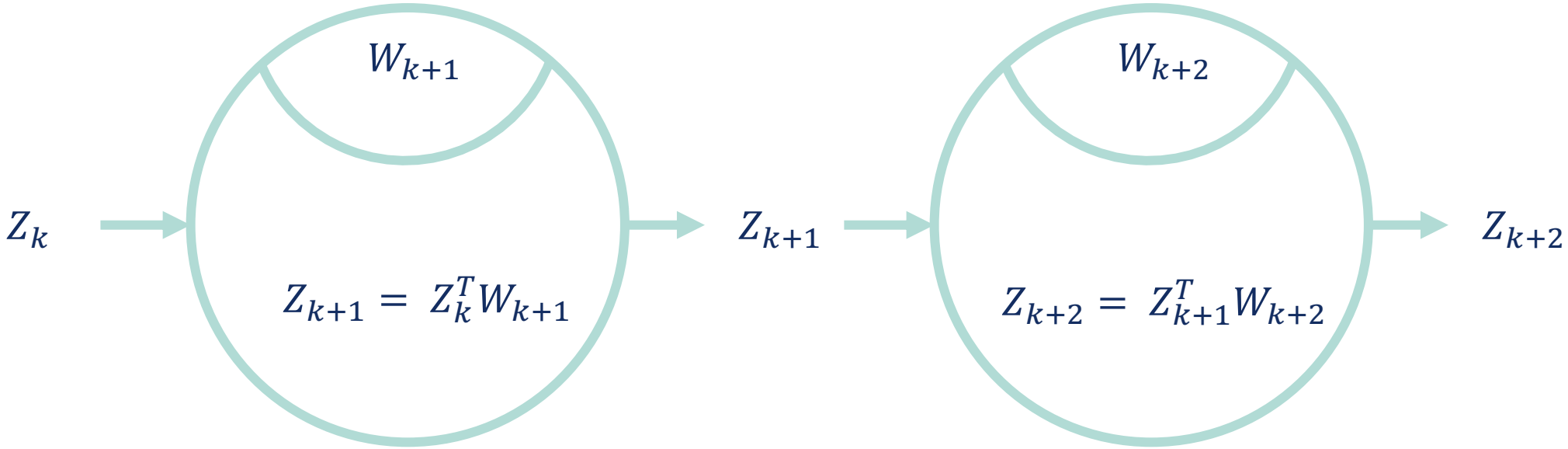


# Multilayer Perceptron: Formulation

# Dense Layer

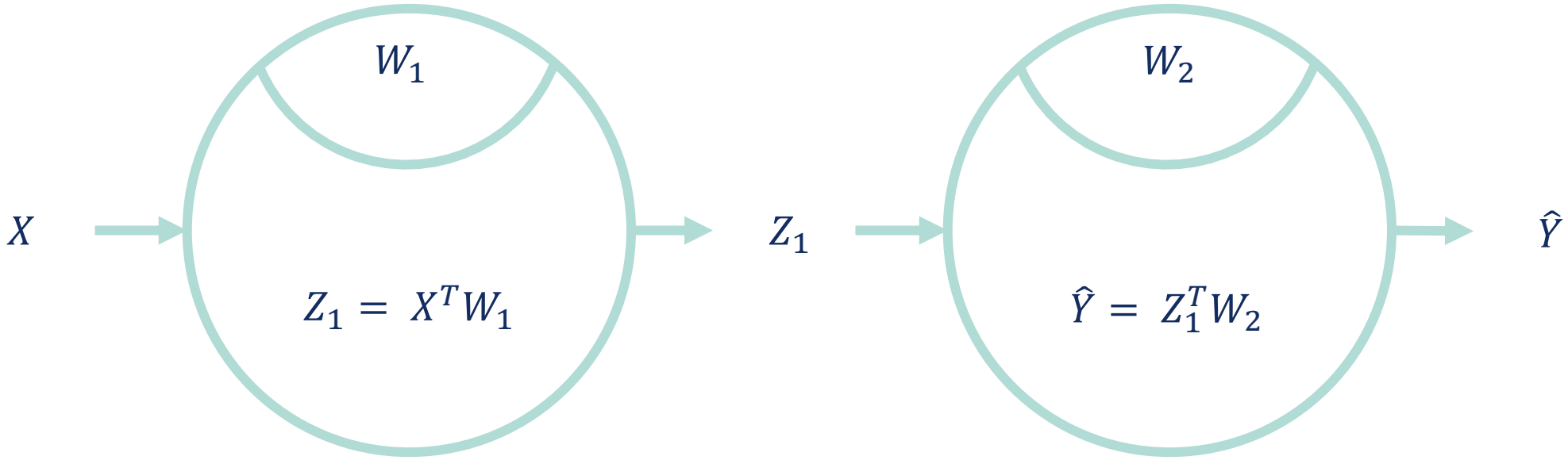

Вход слоя
$Z_k$
$(m_k \times n_k)$
$W_{k+1}$
$(m_k \times m_{k+1})$
$Z_{k+1} = Z_k^T W_{k+1}$
Выход слоя
$Z_{k+1}$
$(m_{k+1} \times n_{k+1})$

# Stacked Dense Layers




$$Z_{k+2} = (Z_k^T W_{k+1})^T W_{k+2}$$

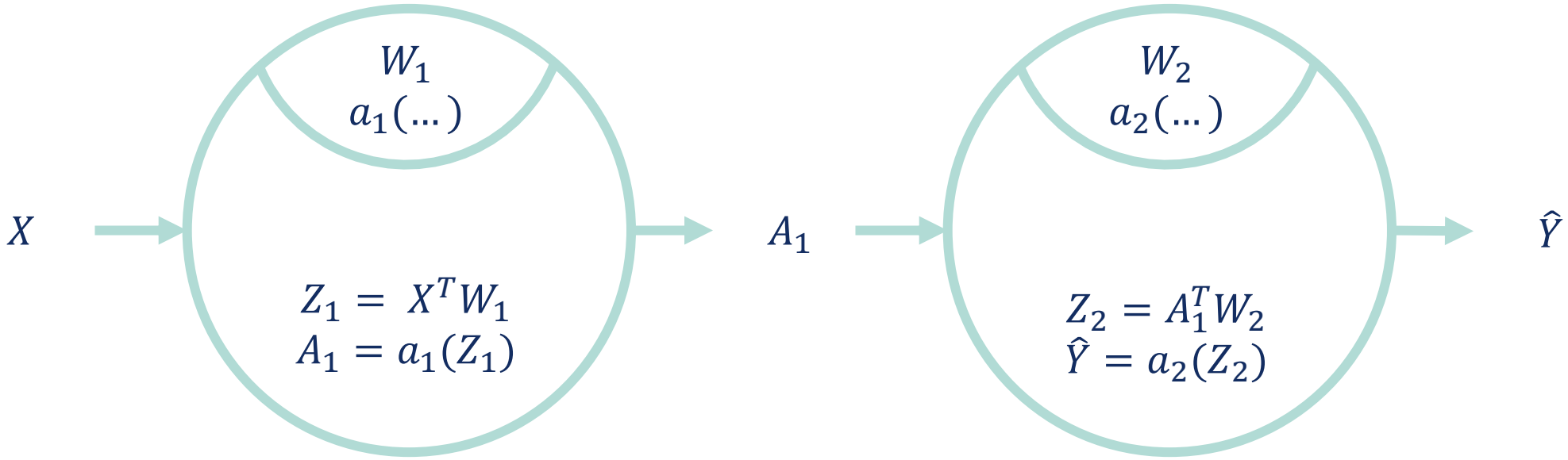
# Stacked Dense Layers




$$\hat{Y} = (X^T W_1)^T W_2 = W_1^T X W_2$$

Линейная функция

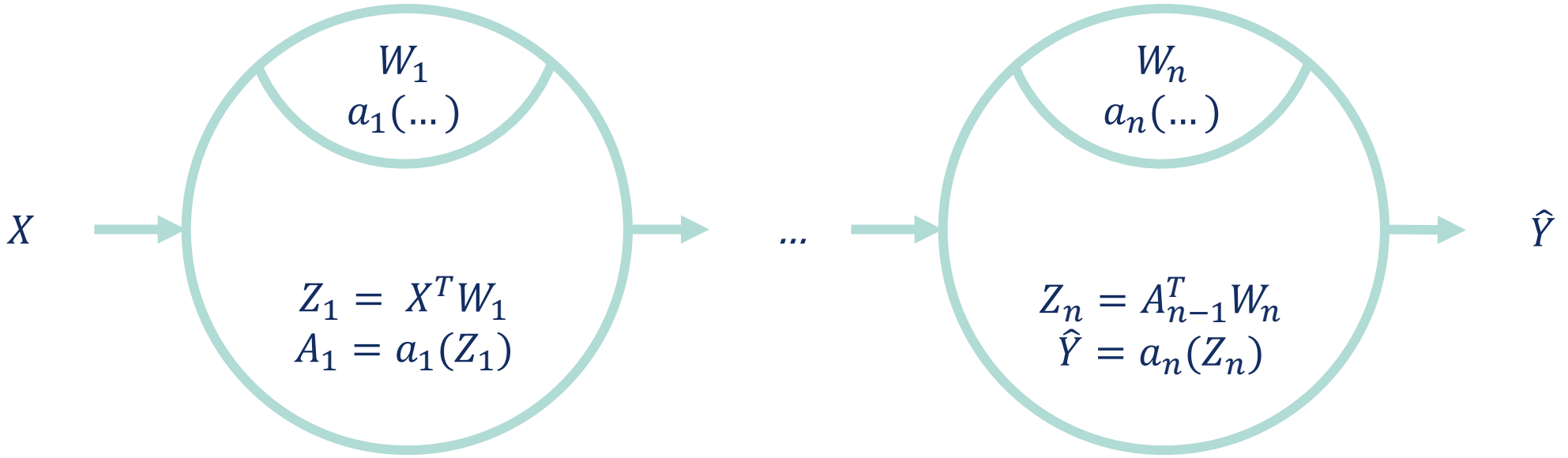
# Stacked Dense Layers with Activations




$$\hat{Y} = a_2(((a_1(X^T W_1))^T W_2)$$

Нелинейная функция

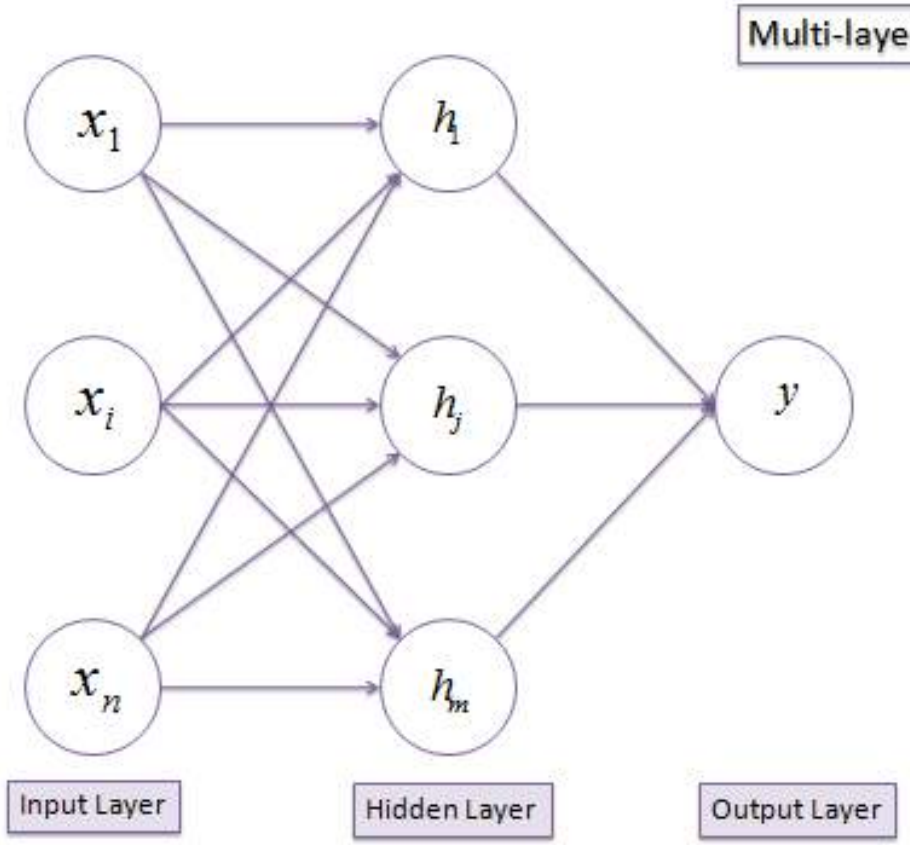
# Stacked Dense Layers with Activations

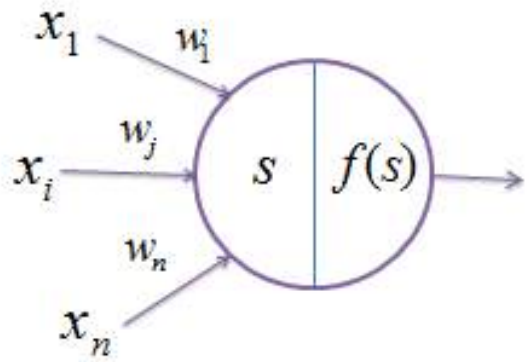



$$\hat{Y} = a_n(((a_{n-1}((\dots)^T W_{n-1}))^T W_n)$$

Нейронная сеть

# Neural Networks: Different Perspective




**Summation**

$$s = \sum w \cdot x$$

**Transformation**

$$f(s) = \frac{1}{1+e^{-s}}$$

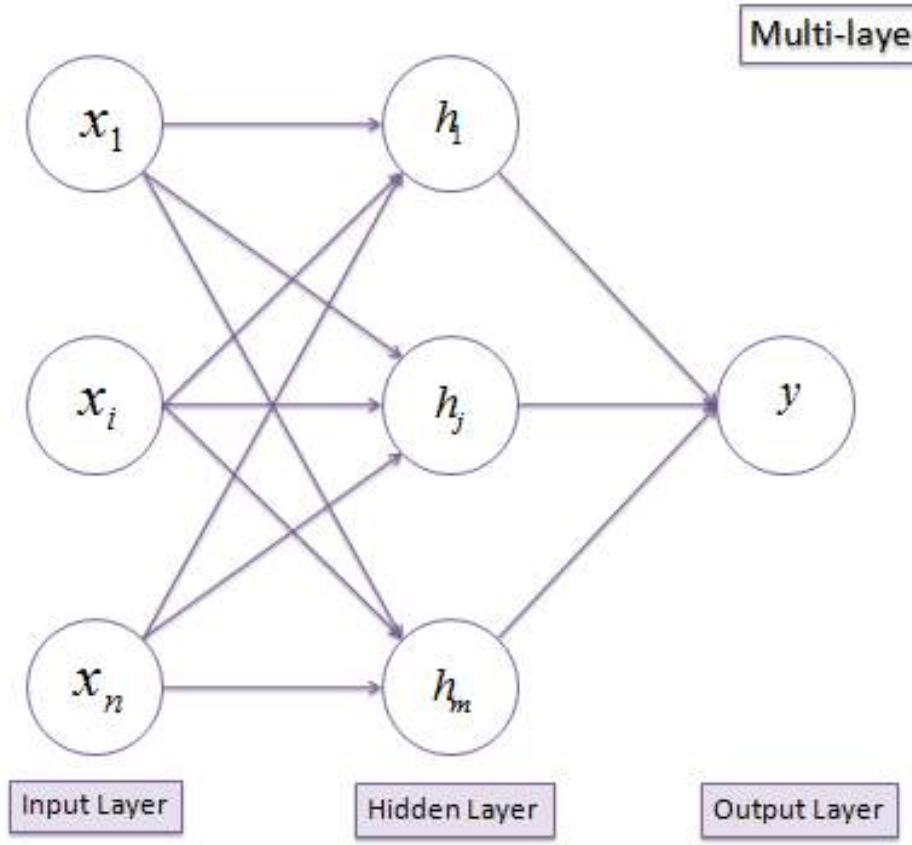
$x_1, \dots, x_n$ – признаки одного примера

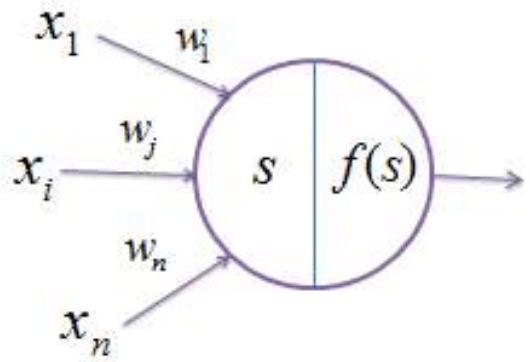
$h_1, \dots, h_n$ – нейроны

$w_i, \dots, w_n$ – веса нейрона для каждого из признаков

$f(s)$ – функция активации

# Neural Networks: Different Perspective




**Summation**

$$s = \sum w \cdot x$$

**Transformation**

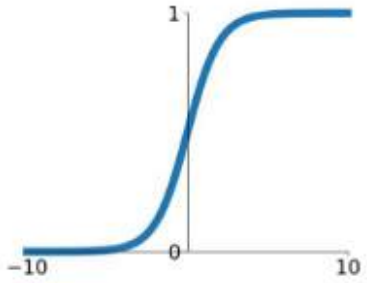
$$f(s) = \frac{1}{1+e^{-s}}$$

В каждом скрытом слое нейронная сеть комбинирует признаки, пришедшие из предыдущего слоя (или напрямую из данных).

# Activation Functions

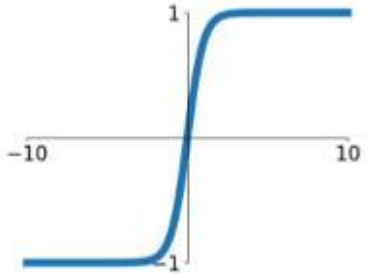
**Sigmoid**

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



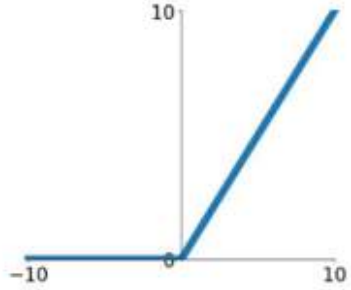

**tanh**

$$\tanh(x)$$



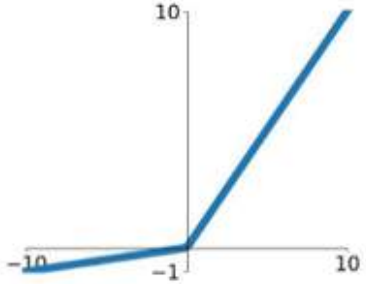

**ReLU**

$$\max(0, x)$$




**Leaky ReLU**

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

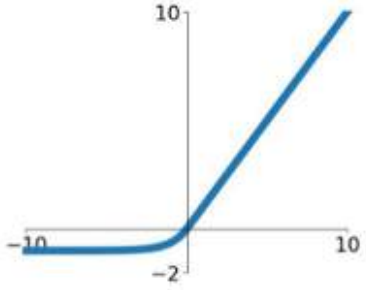



**Maxout**

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

**ELU**

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



# Multilayer Perceptron: Training

# Linear Regression Solution: Gradient Descent

$$L(W) = \frac{1}{n}(Y - X^T W)^T (Y - X^T W) \rightarrow 0$$

$$W_{t+1} = W_t - \alpha(\underbrace{\frac{2}{n} X (X^T W - Y)}_{\nabla L(W)})$$

Чтобы обновить параметры линейной регрессии градиентным спуском нужно взять производную функции потерь по этим параметрам.

С нейронной сетью можно сделать то же самое.

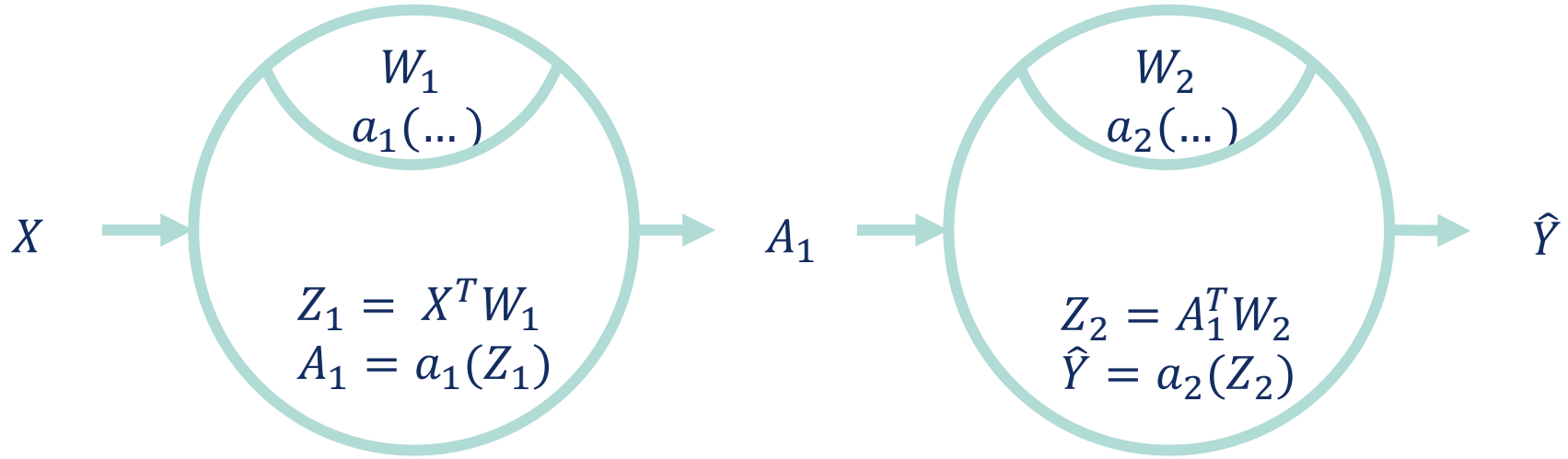
Но как?

# Recap: Chain Rule

$$f = f(g); \quad g = g(x)$$

$$\frac{df}{dx} = \frac{df}{dg} * \frac{dg}{dx}$$

# Gradient of Neural Net



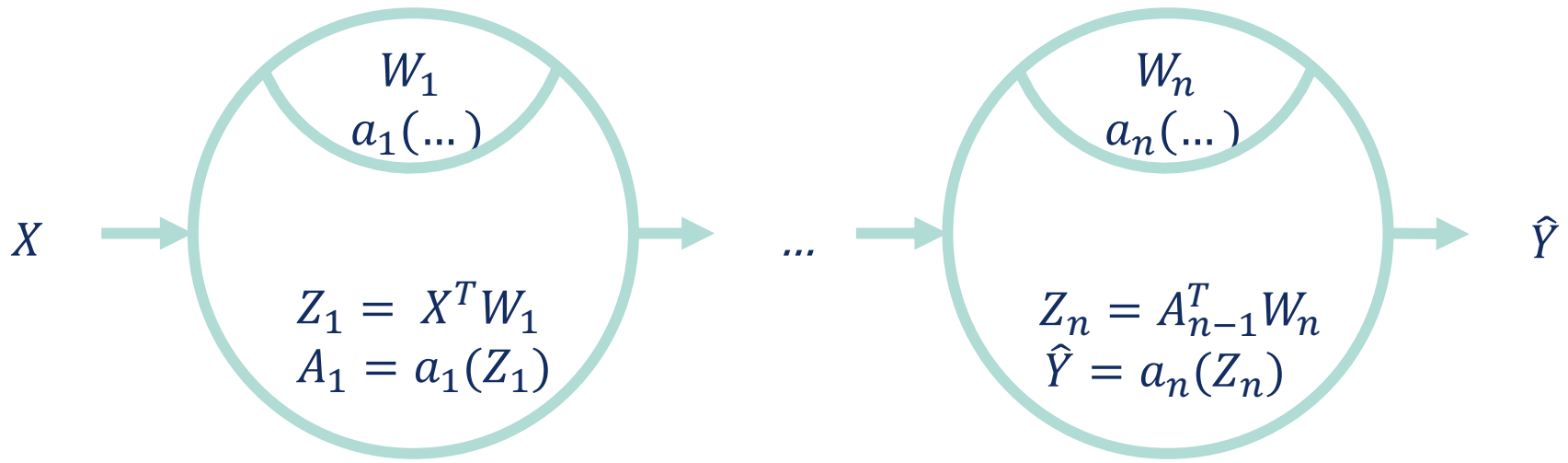

$$\hat{Y} = a_2(A_1^T W_2)$$

$$L(W) = \frac{1}{n}\left(Y - \hat{Y}\right)^T \left(Y - \hat{Y}\right)$$

$$\frac{dL}{dW_2} = \frac{dL}{d\hat{Y}} * \frac{d\hat{Y}}{dW_2}$$

$$\frac{dL}{dW_1} = \frac{dL}{d\hat{Y}} * \frac{d\hat{Y}}{dA_1} * \frac{dA_1}{dW_1}$$

# Stacked Dense Layers with Activations




$$\frac{dL}{dW_{n-1}} = \frac{dL}{d\hat{Y}} * \frac{d\hat{Y}}{dA_{n-1}} * \frac{dA_{n-1}}{dW_{n-1}} = \frac{dL}{dA_{n-1}} * \frac{dA_{n-1}}{dW_{n-1}}$$

$$\frac{dL}{dA_{n-1}}$$

# Finite Difference Method for Gradient Checking

Если нужно проверить аналитические градиенты нейронной сети, то можно посчитать численные градиенты по формуле finite difference.

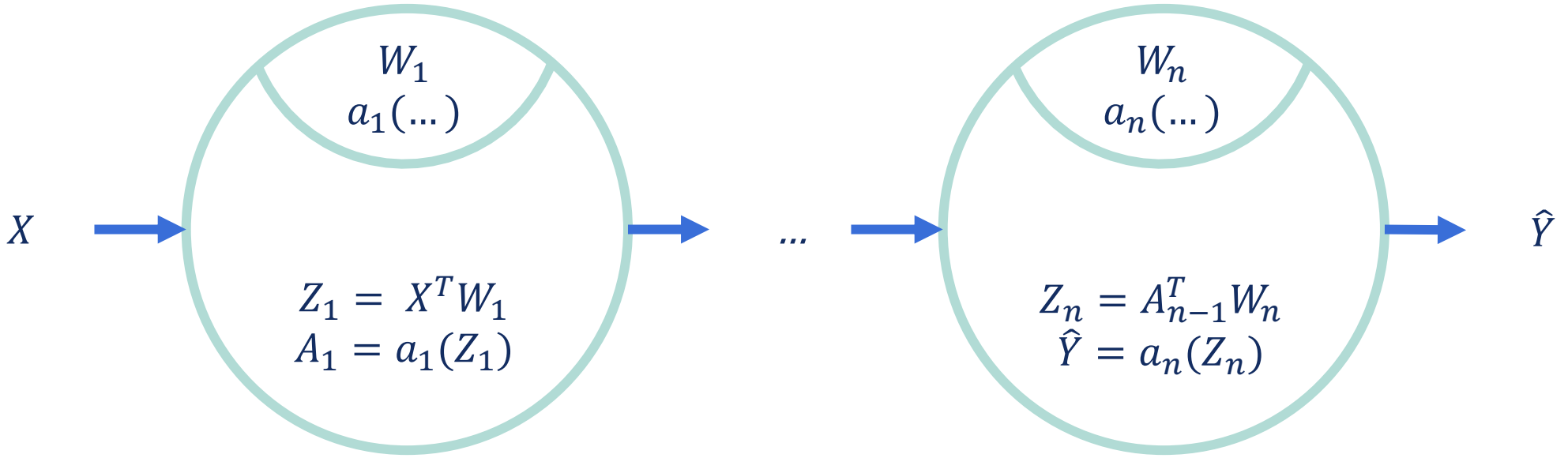
$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

$$NN'_{w_k}(w_1, \dots, w_k, \dots, w_n) = \lim_{\Delta w_k \to 0} \frac{NN(w_k + \Delta w_k) - NN(w_k)}{\Delta w_k}$$

$NN'(w_1, \dots, w_k, \dots, w_n)$ – нейронная сеть, с параметрами $w_1, \dots, w_k, \dots, w_n$

$w_k$ – вес, градиент по которому хочется посчитать

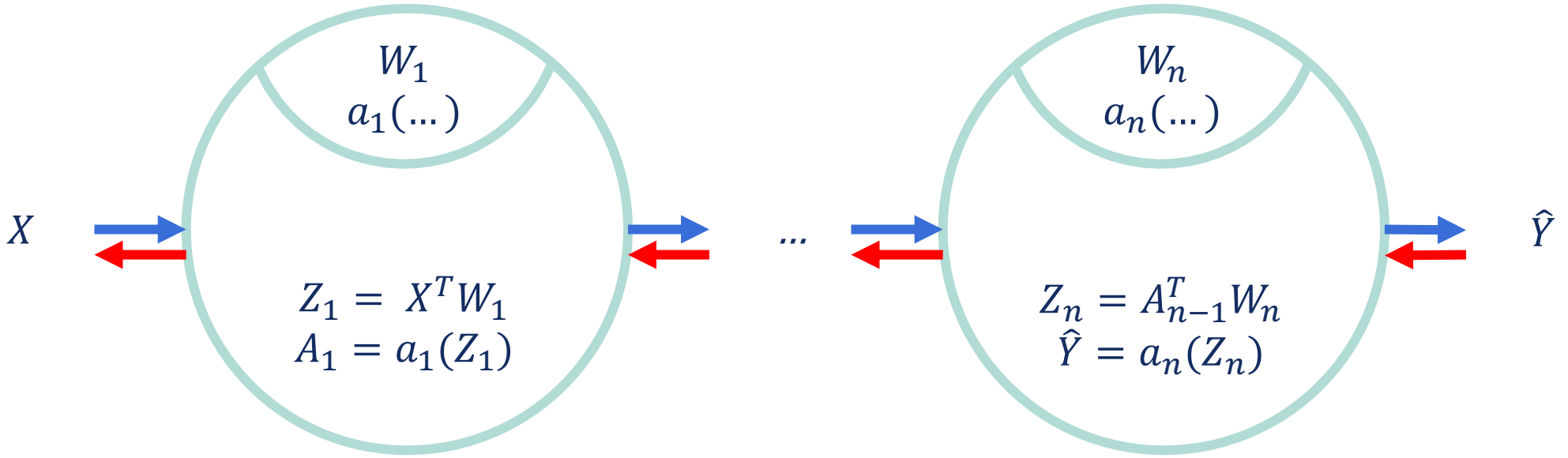
# Forward and Backward Propagation




Forward propagation – процесс последовательного применения операций (слоев) к входным данным нейронной сети ($X$) и получения предсказаний ($\hat{Y}$).

(синие стрелки на картинке)

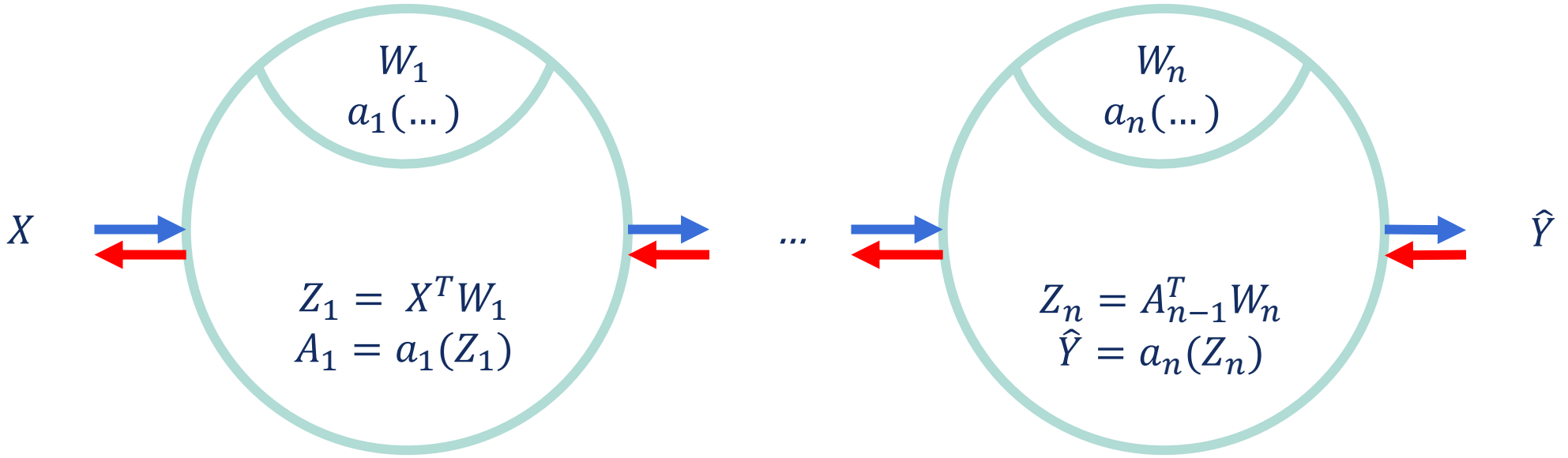
# Forward and Backward Propagation




Backward propagation – процесс последовательного получения производных функции потерь по всем параметрам нейронной сети.

(красные стрелки на картинке)

# Forward and Backward Propagation




В результате backward pass становятся известны производные функции потерь по всем параметрам сети. Таким образом, можно обновить все параметры сети с помощью градиентного спуска.

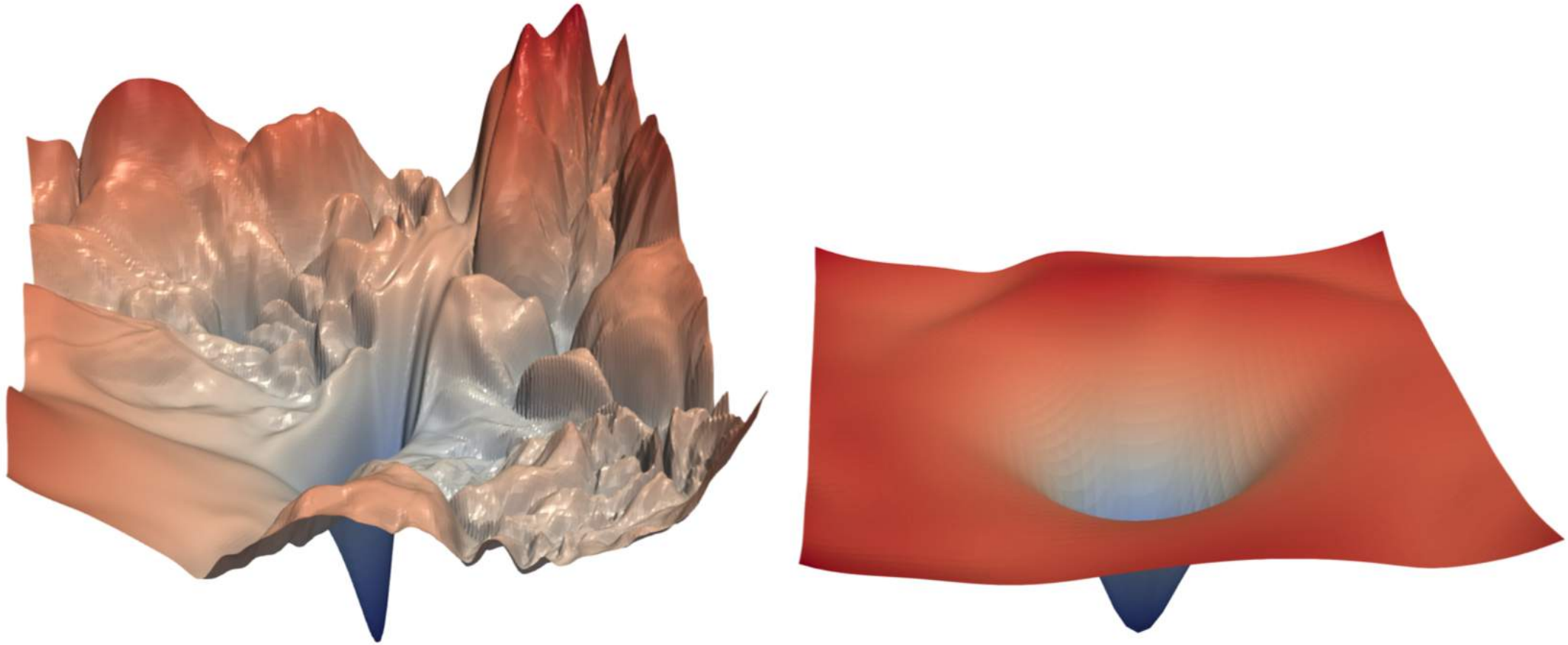
# Neural Networks: Gradient Descent

$$W_{k,\,t+1} = W_{k,\,t} - \alpha\left(\frac{dL}{dW_{k,\,t}}\right)$$

$W_{k,\,t}$ – веса $k$-го слоя нейронной сети в момент времени $t$ (до шага градиентного спуска)

$\frac{dL}{dW_{k,\,t}}$ – производная функции потерь по весам $k$-го слоя сети в момент времени $t$

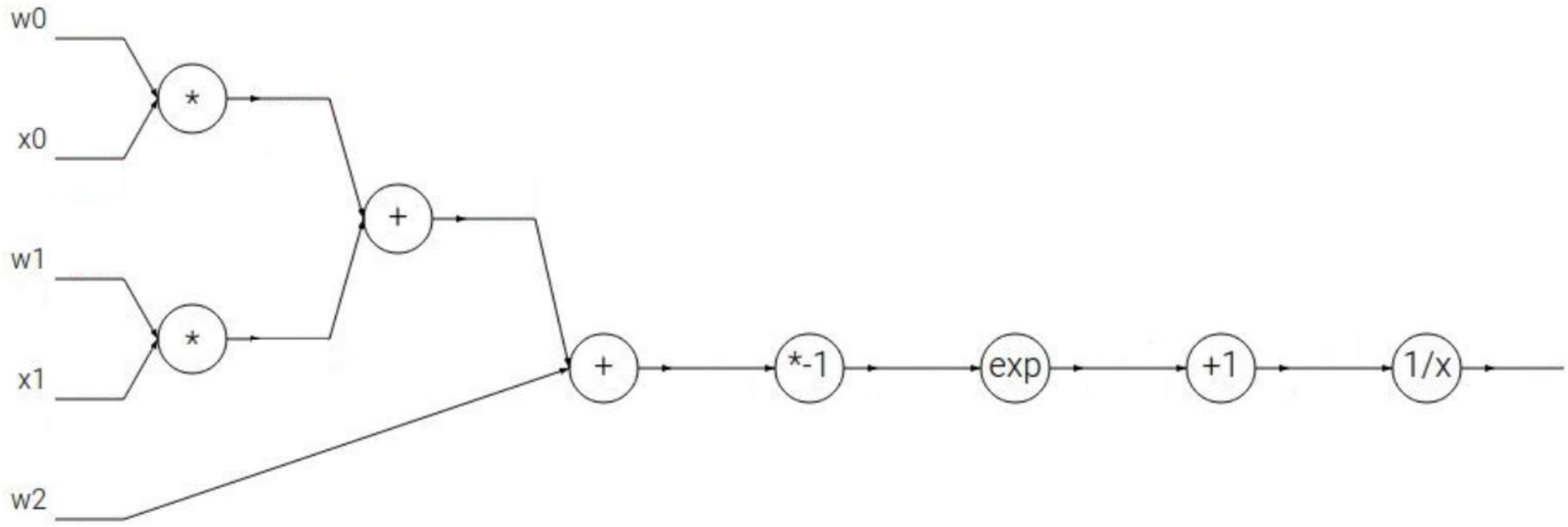
# Loss Surfaces of Neural Networks



# Backpropagation in General

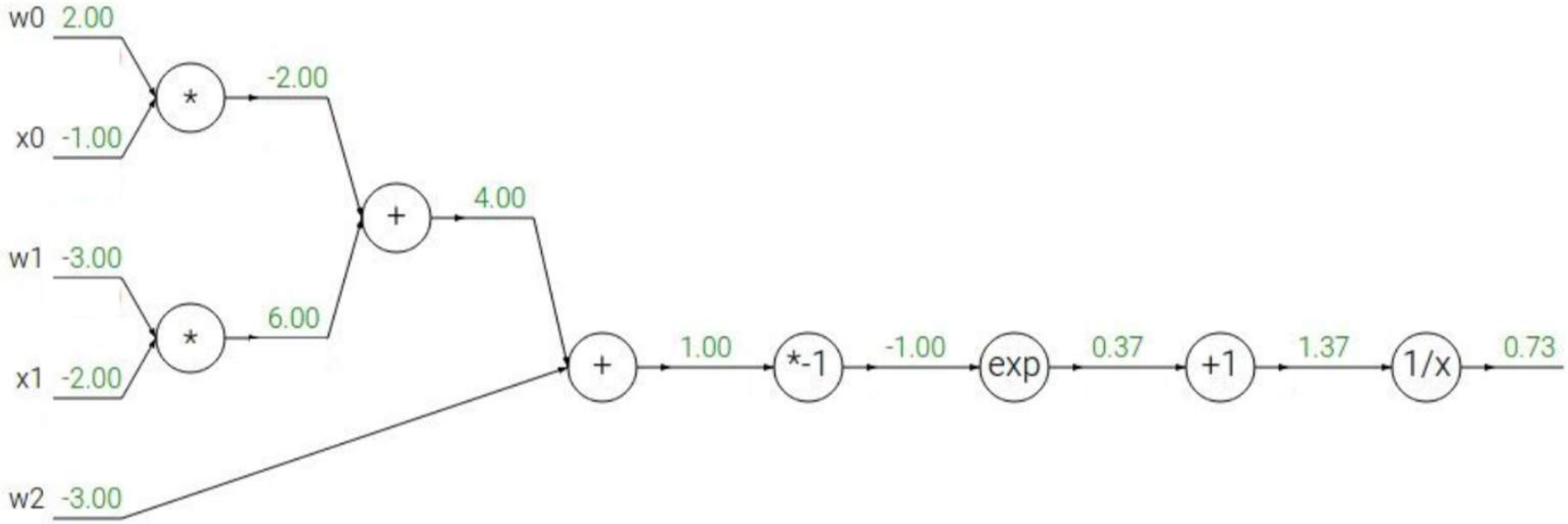
# Forward and Backward Propagation

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




Source: cs231n (http://cs231n.stanford.edu)

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$




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

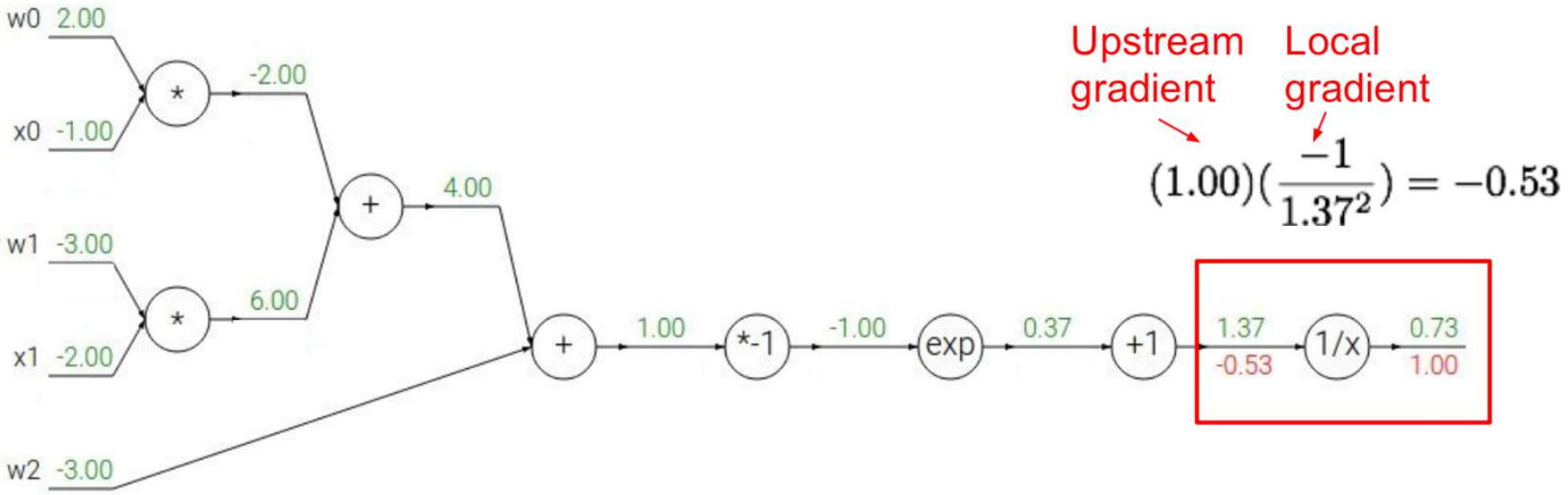
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Source: cs231n (http://cs231n.stanford.edu)

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




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

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

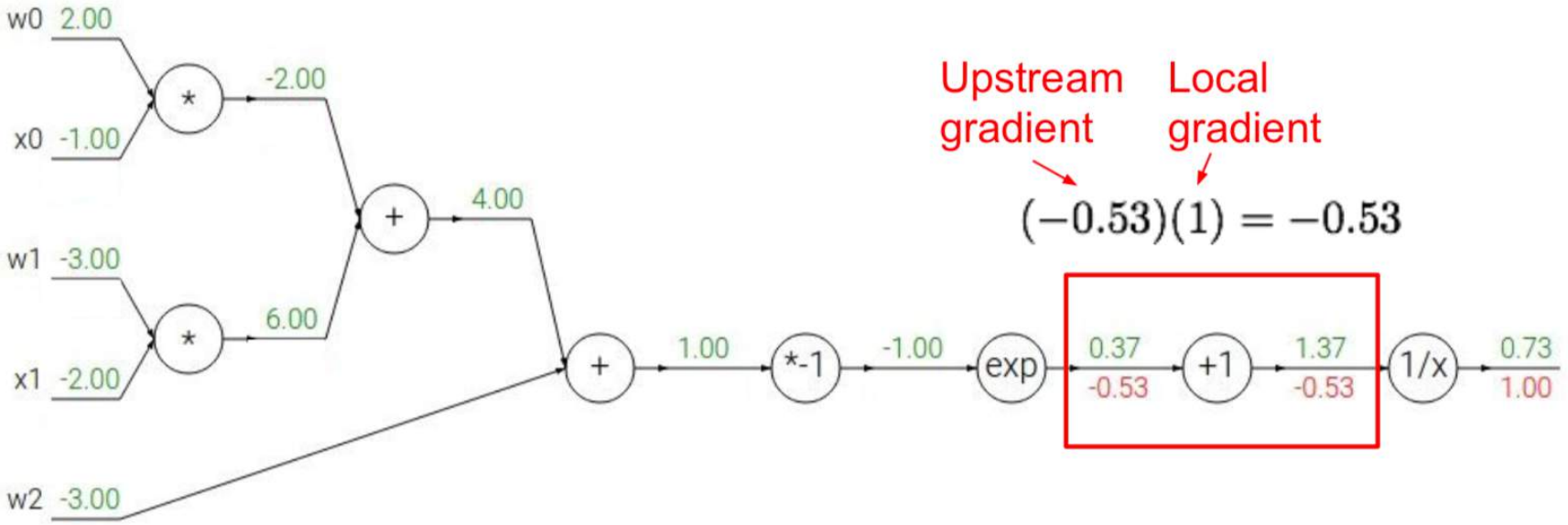
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

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

Upstream gradient

Source: cs231n (http://cs231n.stanford.edu)

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




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

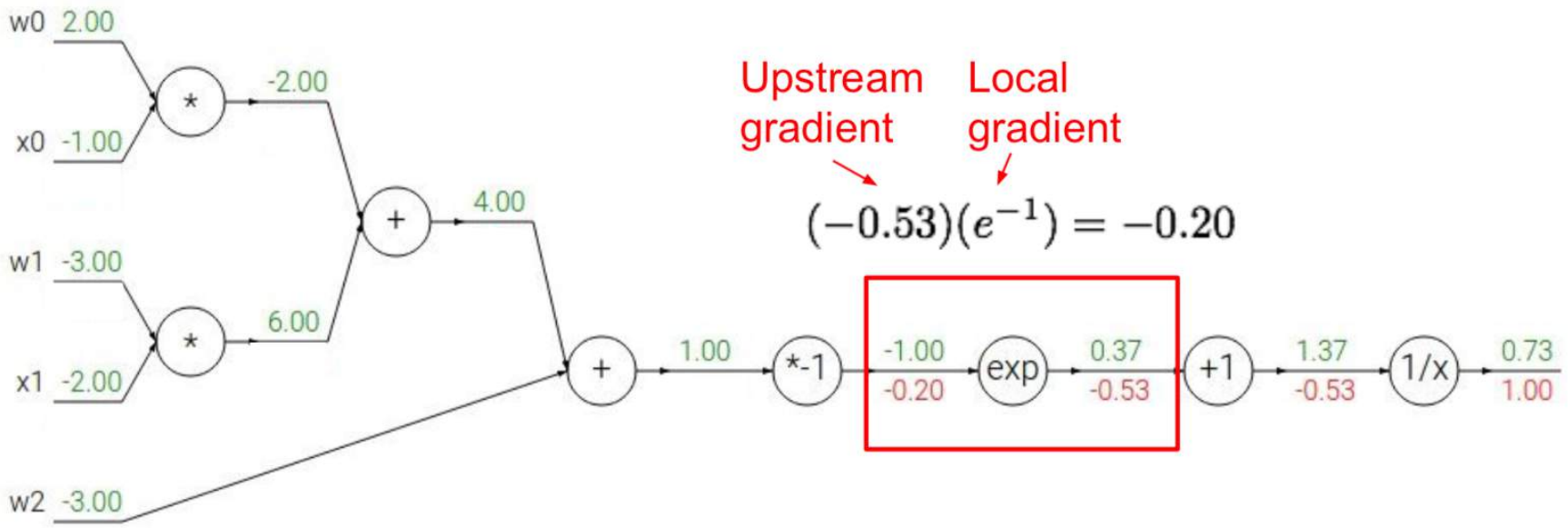
$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

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

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

Source: cs231n (http://cs231n.stanford.edu)

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




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

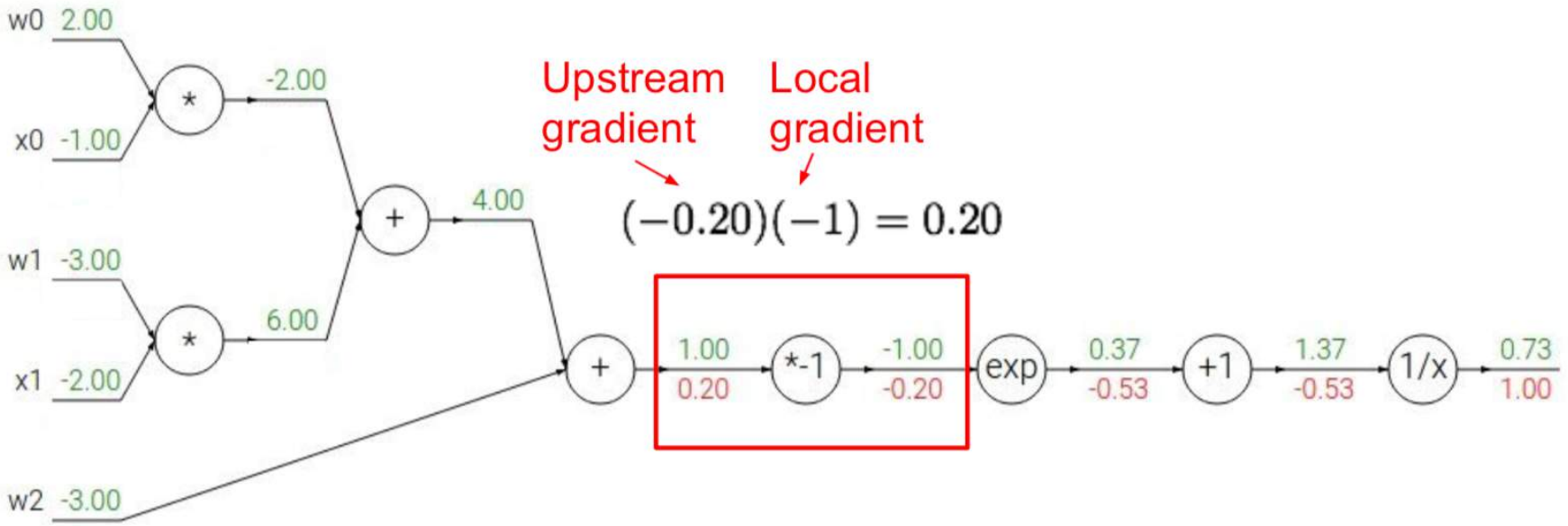
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Source: cs231n (http://cs231n.stanford.edu)

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




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

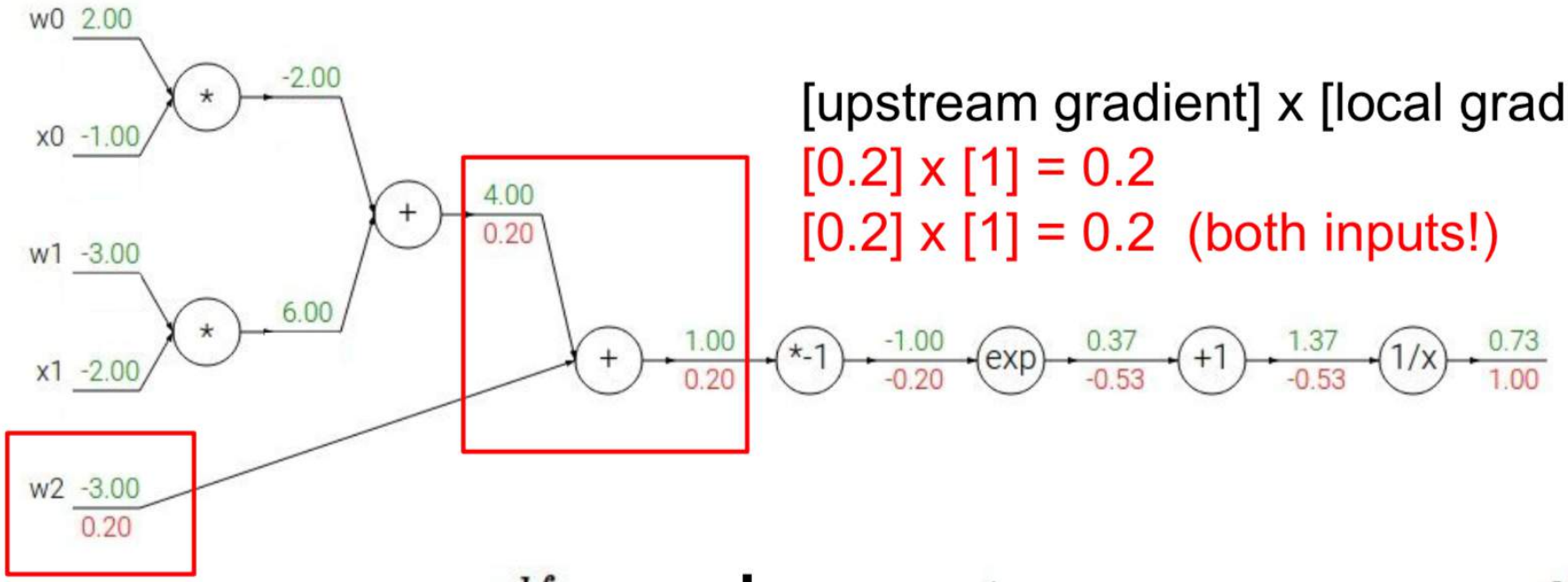
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

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

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

Source: cs231n (http://cs231n.stanford.edu)

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




[upstream gradient] x [local gradient]

[0.2] x [1] = 0.2

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

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

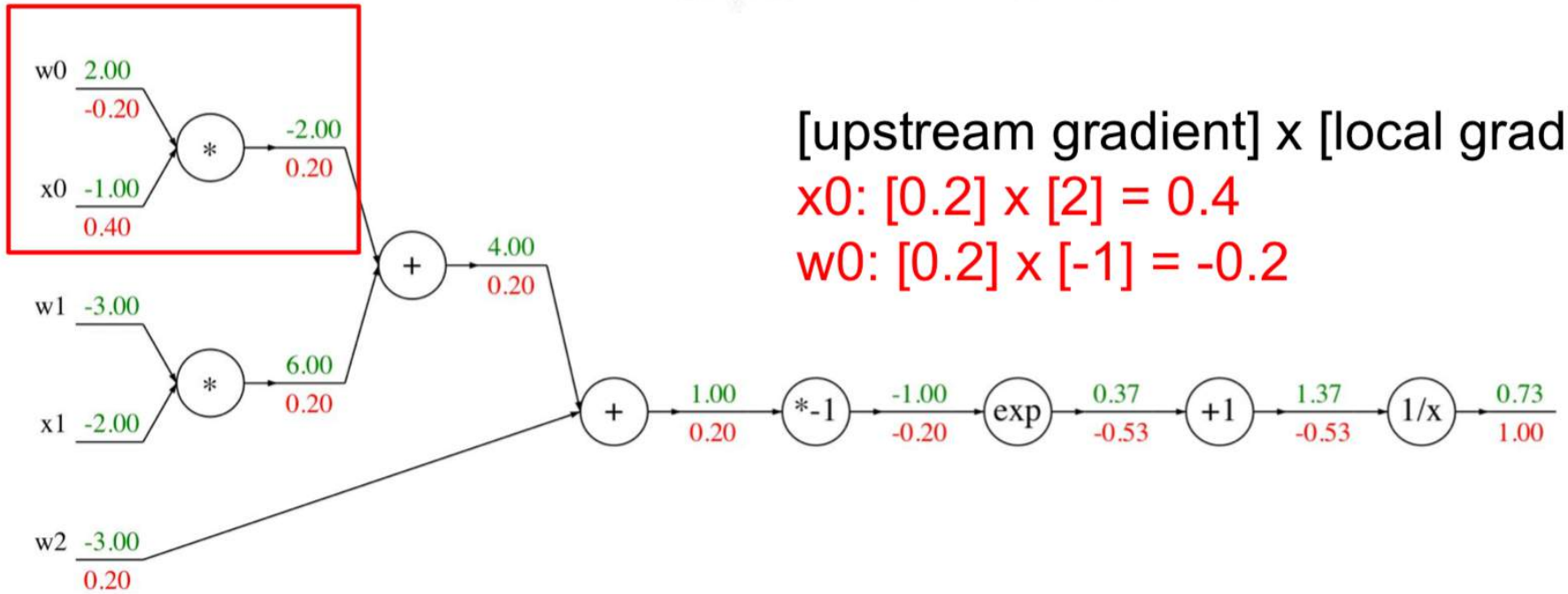
$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

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

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

Source: cs231n (http://cs231n.stanford.edu)

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$




[upstream gradient] x [local gradient]

x0: [0.2] x [2] = 0.4

w0: [0.2] x [-1] = -0.2

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

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

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

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

Source: cs231n (http://cs231n.stanford.edu)

# Differentiable and Non-Differentiable Operations

## Differentiable

- Сумма
- Произведение
- Возведение в степень
- …

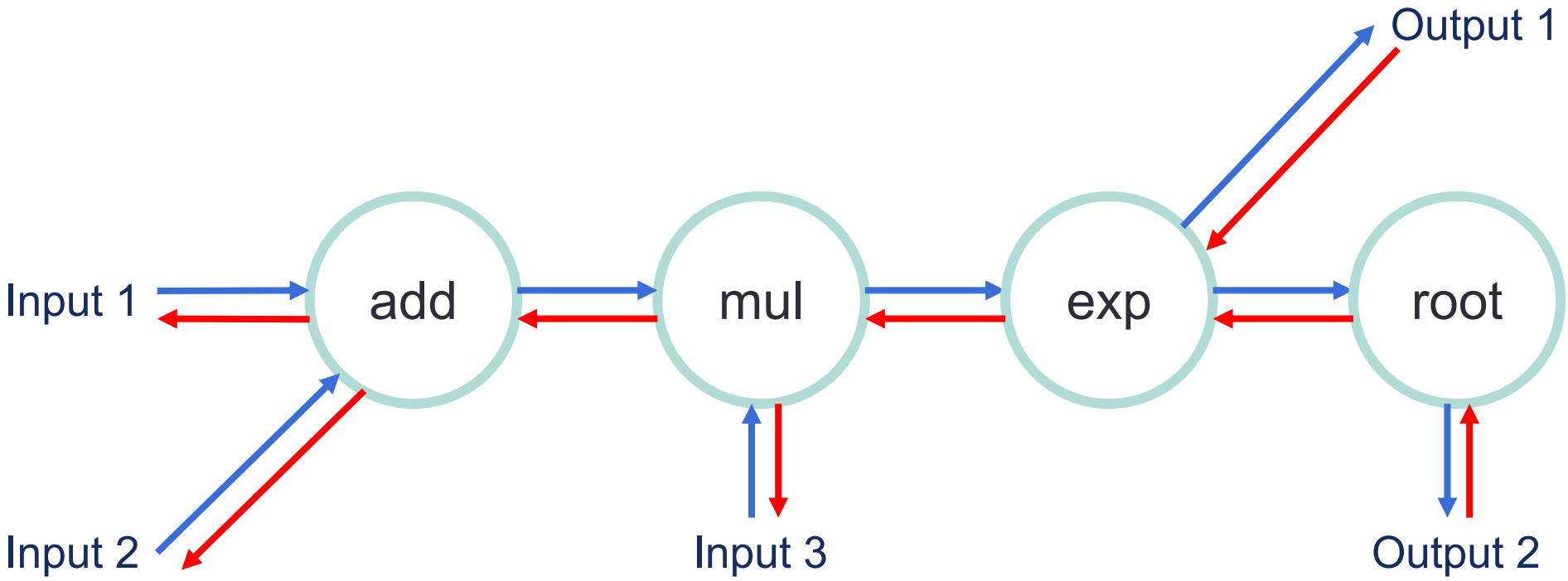
## Non-Differentiable

- Sampling
- Argmax
- Выбор ближайшего соседа
- …

# DL Frameworks

# DL Frameworks

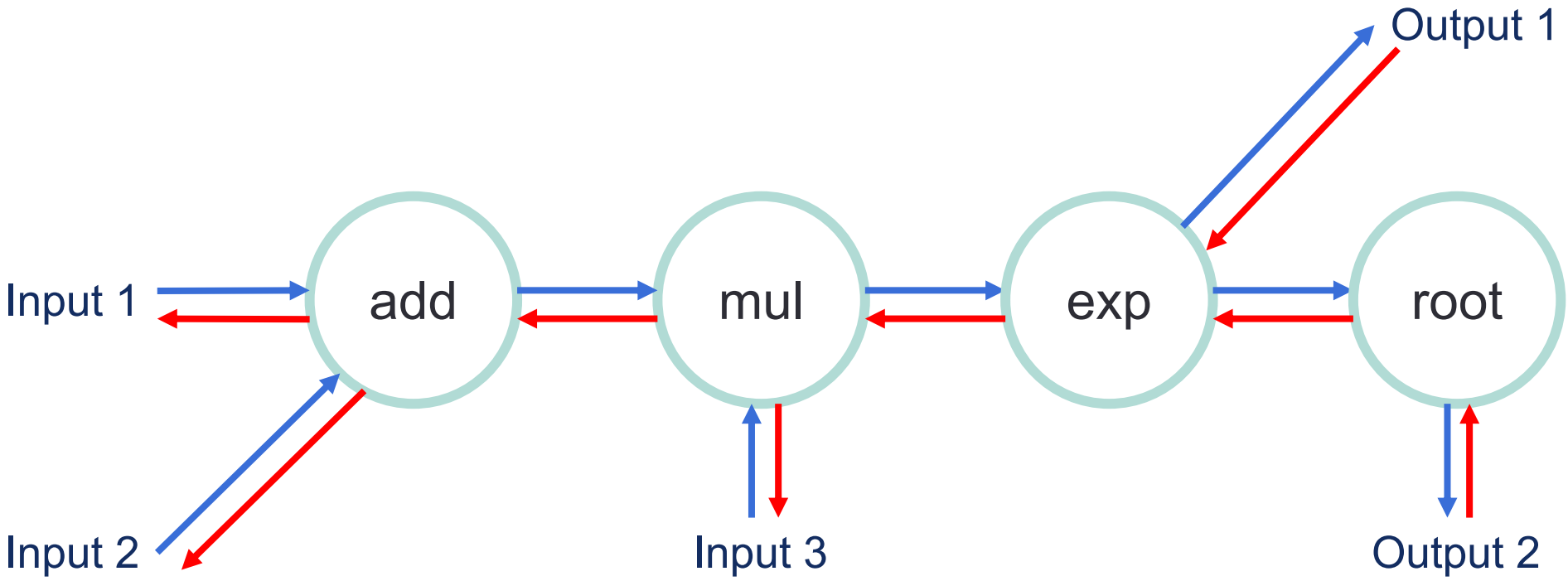
# DL Frameworks: Main Principle




$$output\_2 = root(\exp(mul(input\_3, add(input\_1, input\_2))))$$

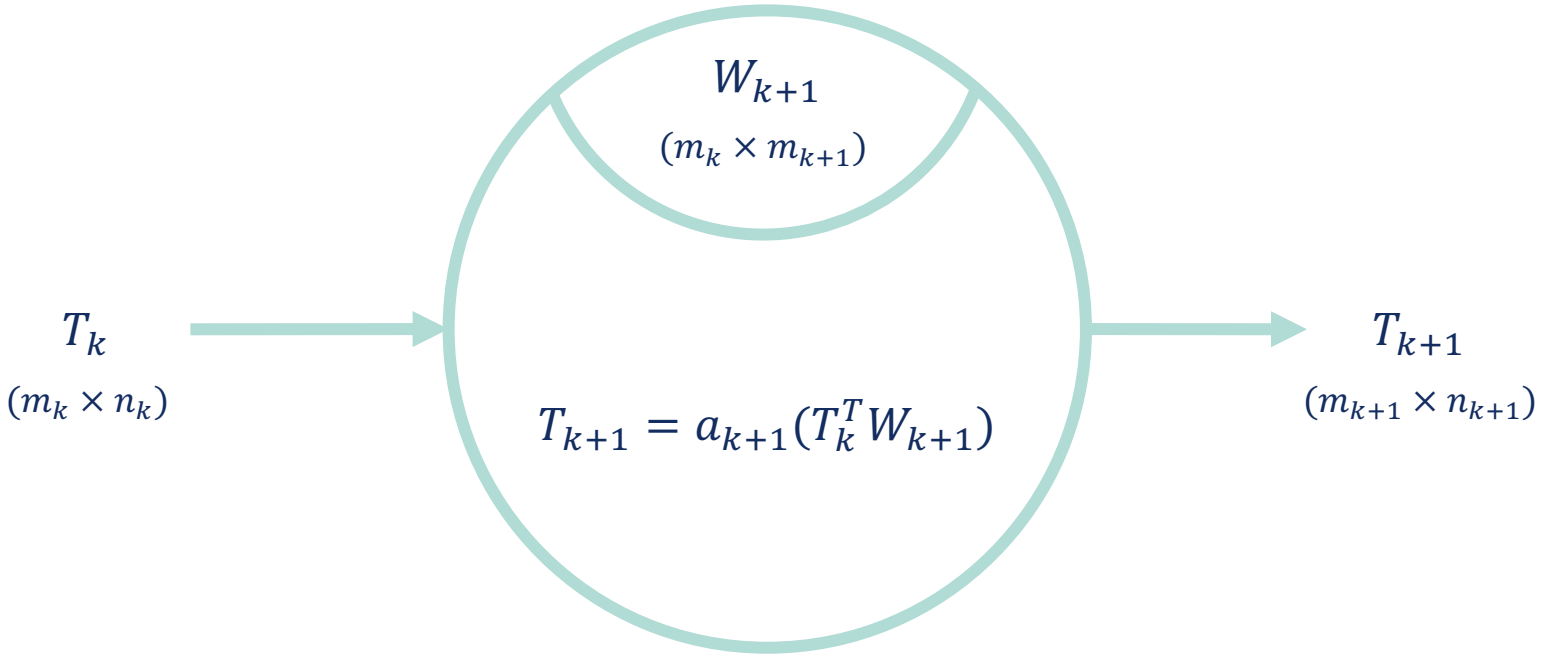
$$output\_1 = \exp(mul(input\_3, add(input\_1, input\_2)))$$

# DL Frameworks: Main Principle



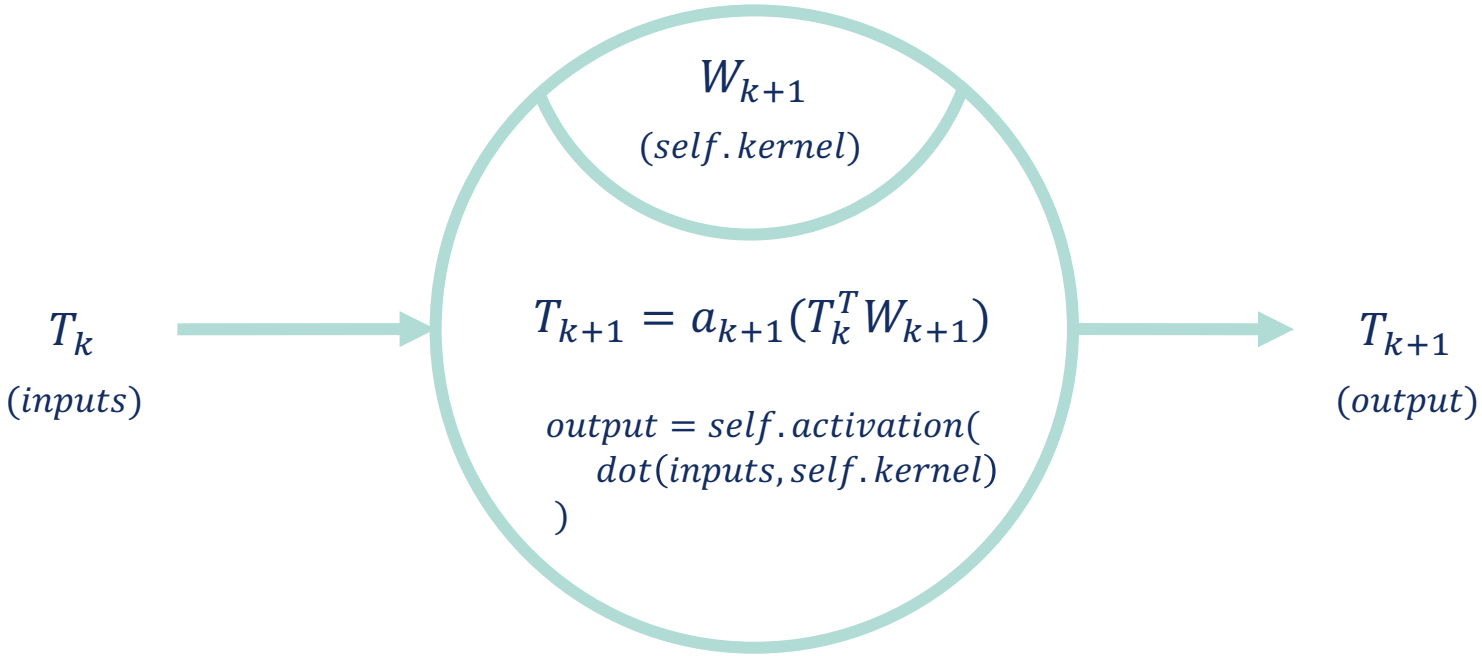

Главное – определить вычислительный граф.
Производные вычисляются автоматически.

# Layers in DL Frameworks



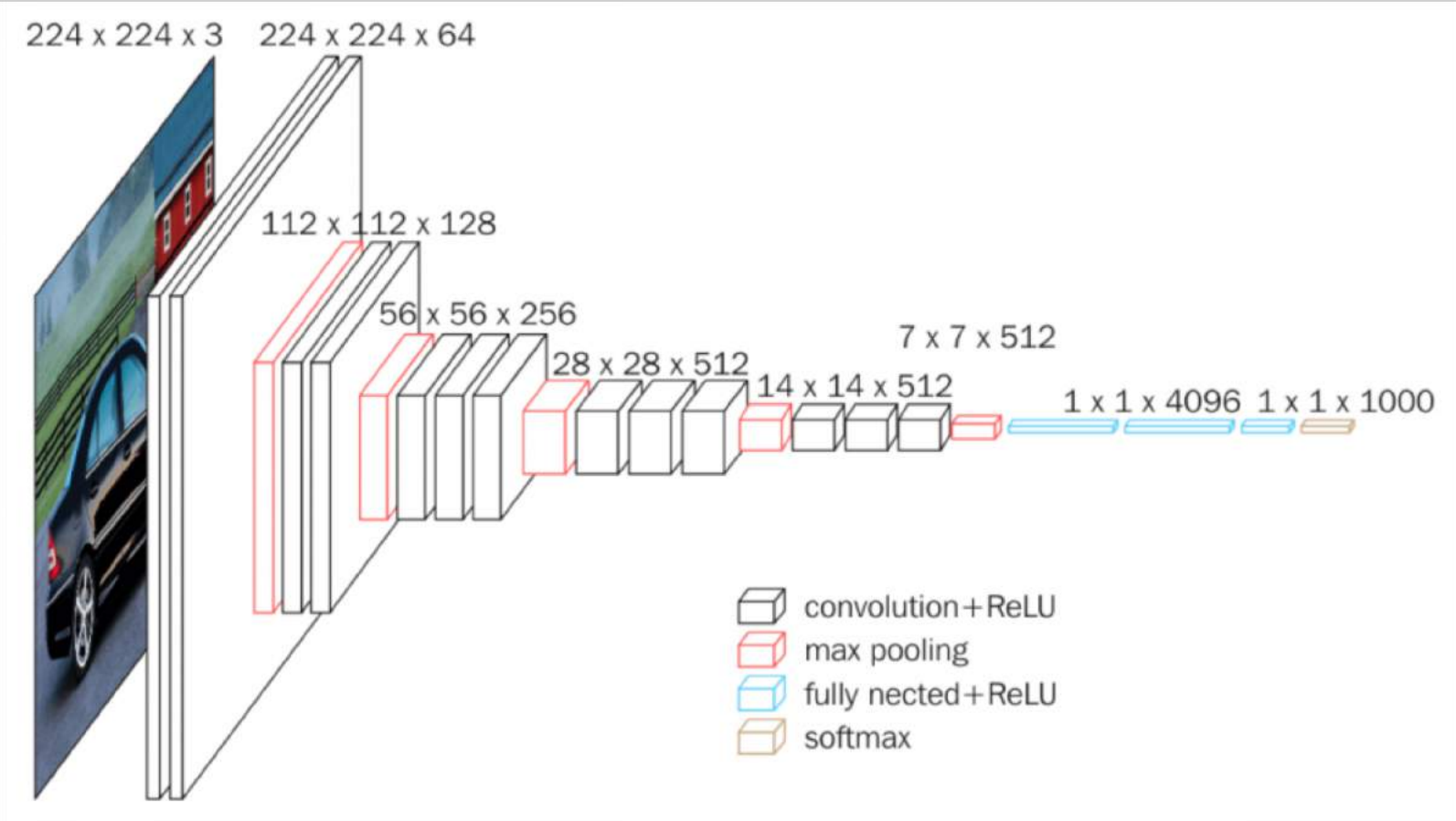

```
def call(self, inputs):
    output = K.dot(inputs, self.kernel)
    if self.use_bias:
        output = K.bias_add(output, self.bias, data_format='channels_last')
    if self.activation is not None:
        output = self.activation(output)
    return output
```

# Layers in DL Frameworks



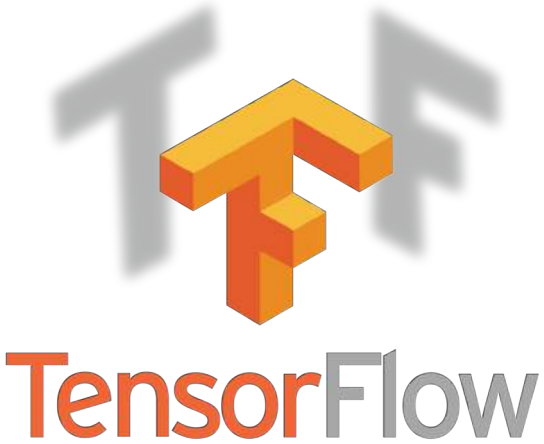

```
def call(self, inputs):
    output = K.dot(inputs, self.kernel)
    if self.use_bias:
        output = K.bias_add(output, self.bias, data_format='channels_last')
    if self.activation is not None:
        output = self.activation(output)
    return output
```

# DL Frameworks




Основная идея фреймворков – сборка архитектур нейронных сетей из автодифференцируемых блоков.

# Static vs Dynamic Graphs




Статичный граф операций

- определяется один раз перед началом выполнения операций
- раньше был быстрее динамического (больше нет)
- некоторые операции либо нельзя либо очень сложно выполнить


Динамический граф операций

- определяется каждый раз при выполнении операций
- более гибкий (можно делать все, что угодно)

# Materials

cs231n Lecture 3: https://www.youtube.com/watch?v=h7iBpEHGVNc&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&t=0s&index=4

Cs231n Lecture 4: https://www.youtube.com/watch?v=d14TUNcbn1k&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&t=0s&index=5

Vectorized implementation of cost functions and Gradient Descent: https://medium.com/ml-ai-study-group/vectorized-implementation-of-cost-functions-and-gradient-vectors-linear-regression-and-logistic-31c17bca9181

What are the top deep learning frameworks to utilize in 2019?: https://www.quora.com/What-are-the-top-deep-learning-frameworks-to-utilize-in-2019