курс «Глубокое обучение»

Нейронные сети

Александр Дьяконов

07 февраля 2022 года

План

Простейшая нейросеть – 1 нейрон

Функции активации

(линейная, пороговая, сигмоида, гиперболический тангенс, softmax, LeakyReLU, ELU, Maxout, Exponential Linear Unit, Maxout, Gaussian Error Linear Unit)

Функциональная выразимость нейрона

Теорема об универсальной аппроксимации

Сеть прямого распространения

Необходимость глубоких сетей

Обучение

Обратное распространение градиента (Backpropagation)

Функции ошибки

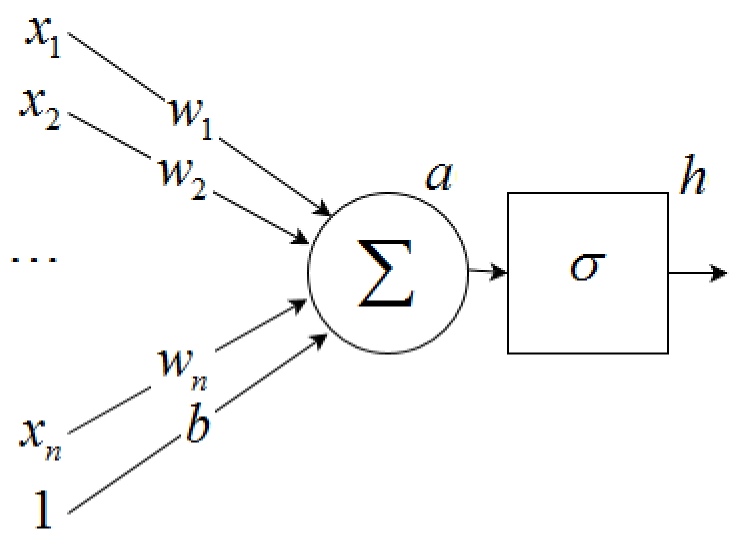
Нейросеть – вычислительный граф

Вычисление градиента на графе

Производные на компьютере

Проблема затухания градиента

Простейшая нейросеть – 1 нейрон



Разделяющая поверхность – линейная

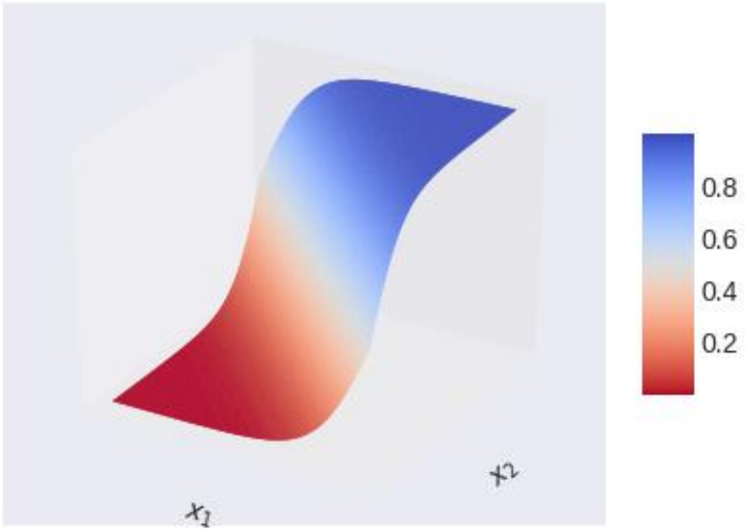
$$a(x) = b + w_1 x_1 + \ldots + w_n x_n = \sum_{t=0}^{n} w_t x_t$$

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

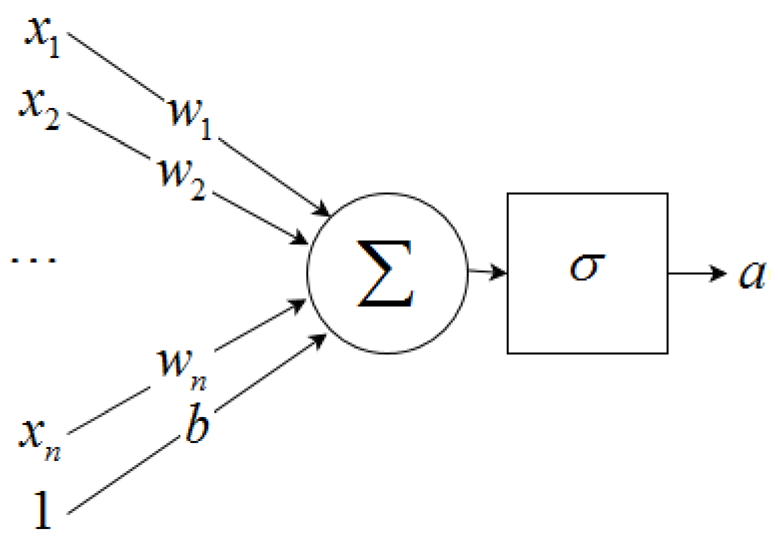
b **– смещение**

σ **– функция активации**

w_t **– веса связей**



Линейные модели – нейросети!



Линейная регрессия

$$a(x) = b + w_1 x_1 + \ldots + w_n x_n$$

Логистическая регрессия

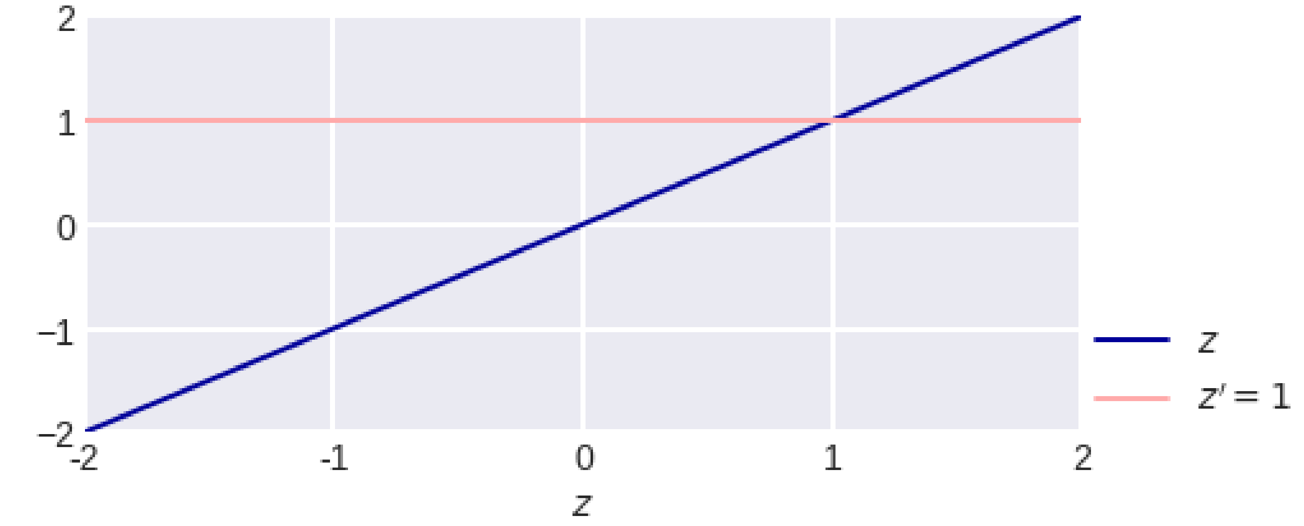
$$a(x) = \sigma(b + w_1 x_1 + \ldots + w_n x_n)$$

Линейный классификатор

$$a(x) = \mathrm{th}(b + w_1 x_1 + \ldots + w_n x_n)$$

Функции активации

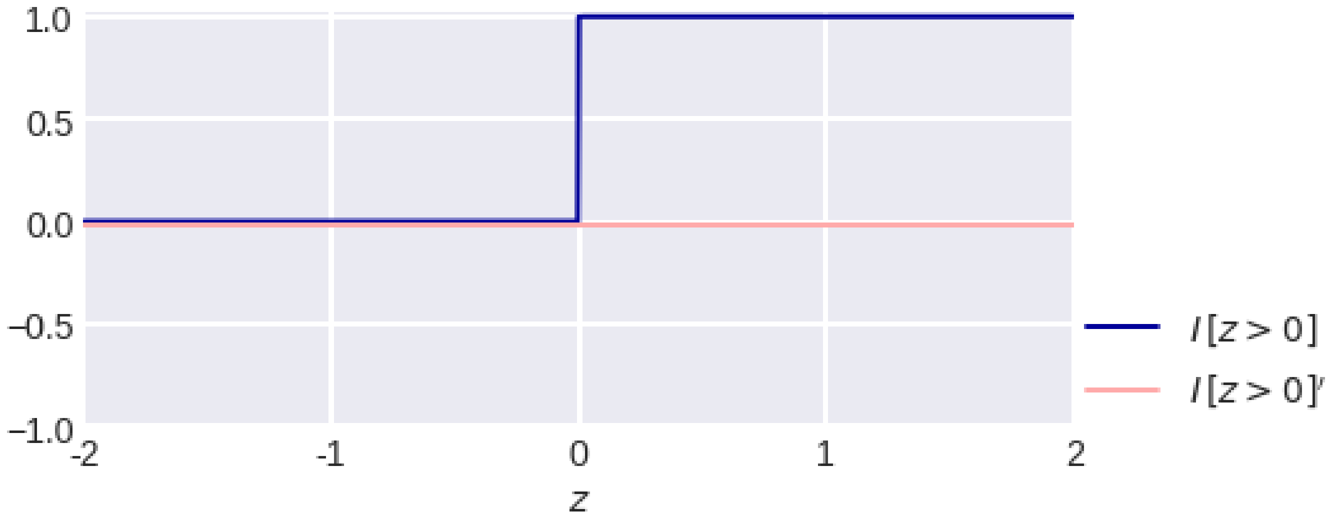
Тождественная функция (линейная / linear activation function)



$$f(z) = z$$

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

Пороговая функция (threshold function)

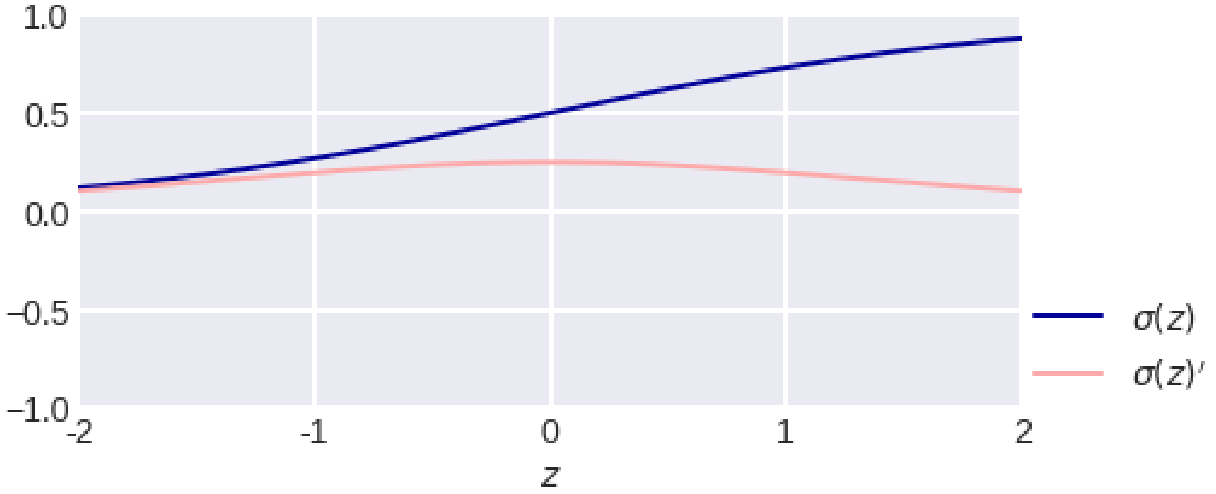


$$\text{th}(z) = I[z > 0]$$

$$\frac{\partial\, \text{th}(z)}{\partial z} = 0$$

Функции активации

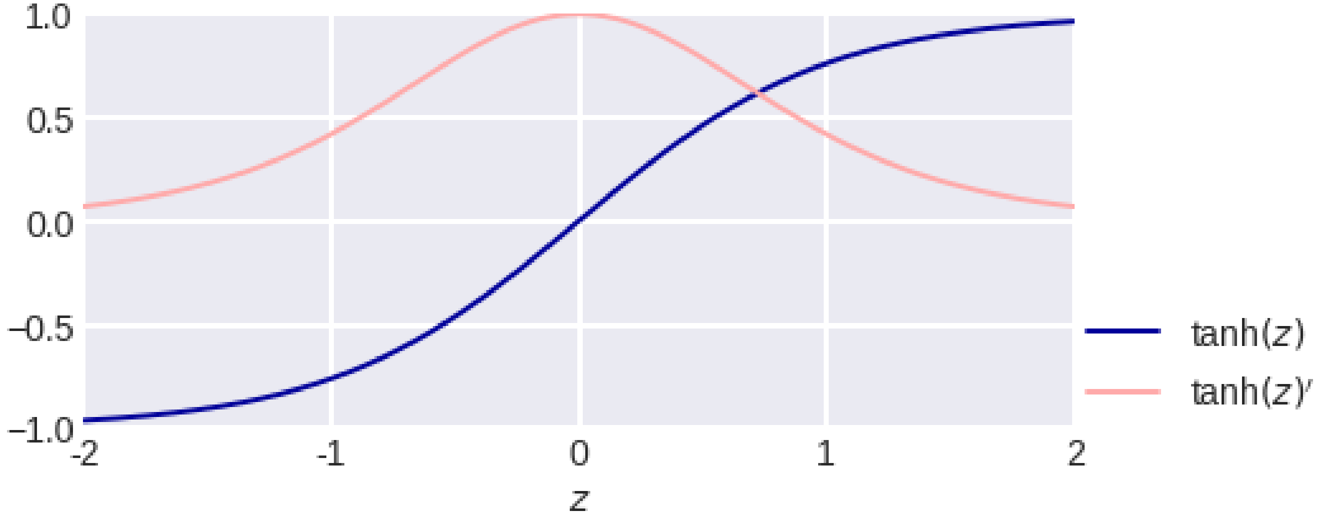
Сигмоида (sigmoid activation function)



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

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

Гиперболический тангенс (hyperbolic tangent)



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

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

Функции активации

обобщение сигмоиды в задачах классификации

$$\mathrm{softmax}(z_1,\ldots,z_k) = \frac{1}{\sum_{t=1}^{k}\exp(z_t)}(\exp(z_1),\ldots,\exp(z_k))^{\mathrm{T}}$$

сумма выходов = 1

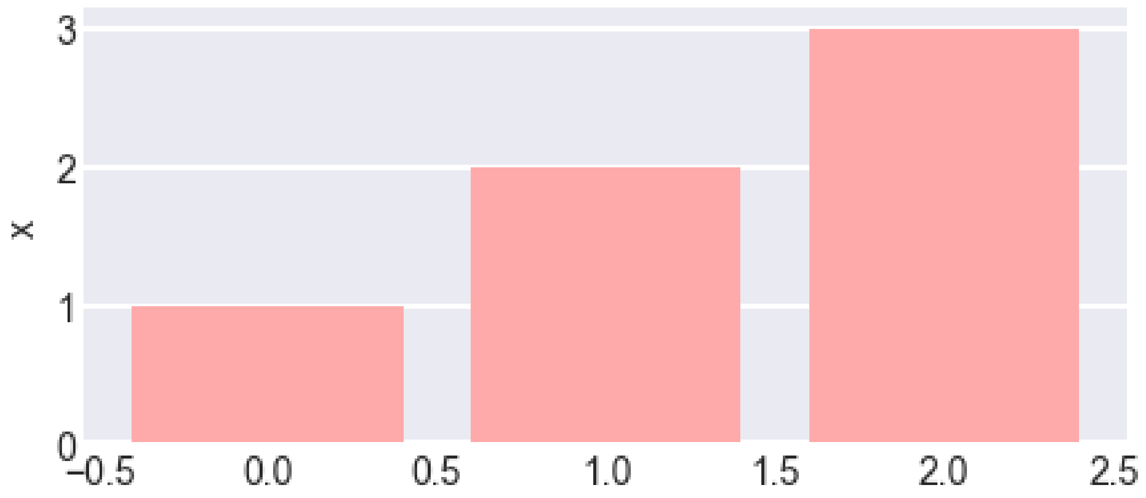
выходы интерпретируются как вероятности

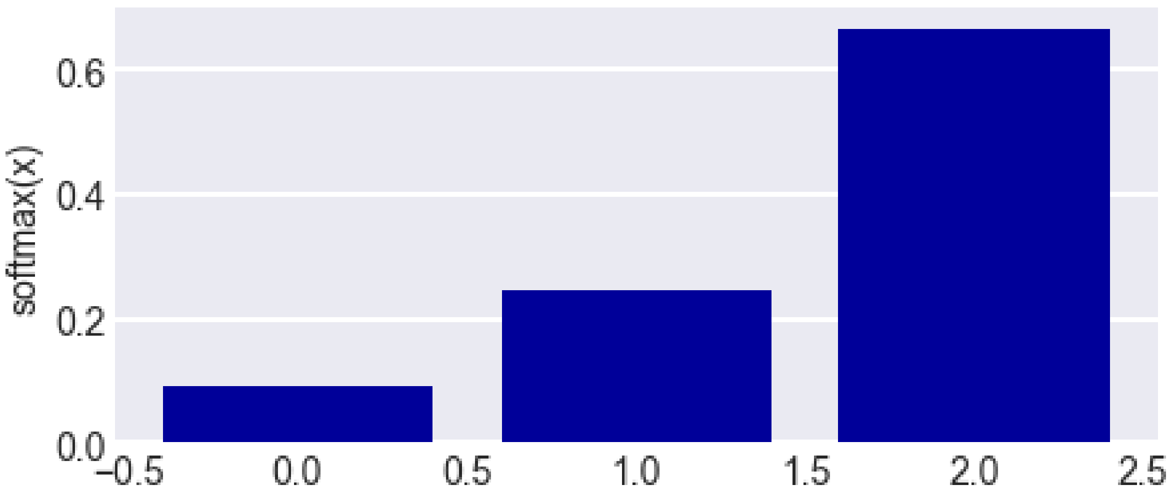
$$[0.5,\ 0.5,\ 0.1,\ 0.7] \to [0.257,\ 0.257,\ 0.172,\ 0.314]$$

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

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

Минутка кода






```
from torch.nn.functional import softmax
x = torch.tensor([1, 2, 3], dtype=float)
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() # создаёт nn.Module
s(x), sigmoid(x) # один результат
```

все нужные функции активации есть…

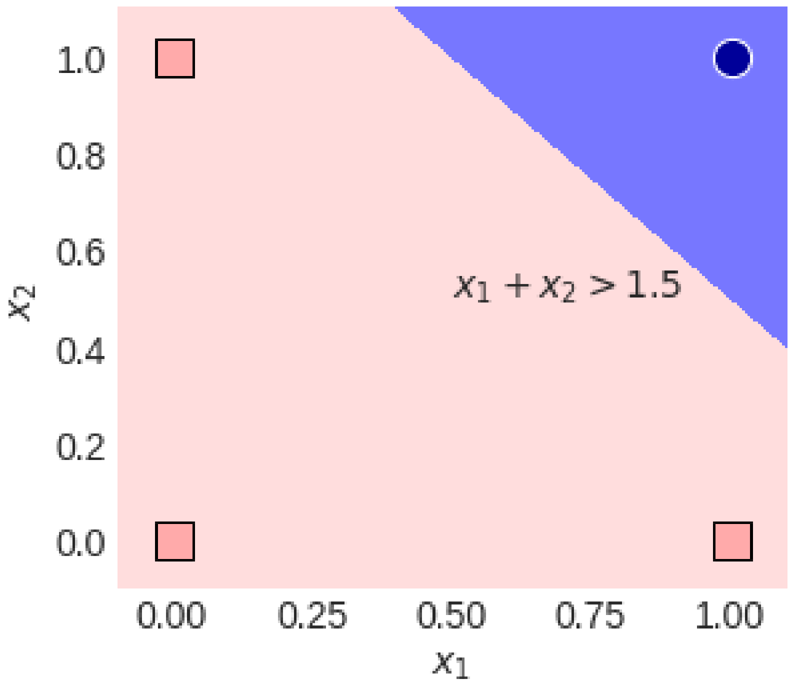
https://pytorch.org/docs/1.9.0/nn.functional.html

Non-linear activation functions

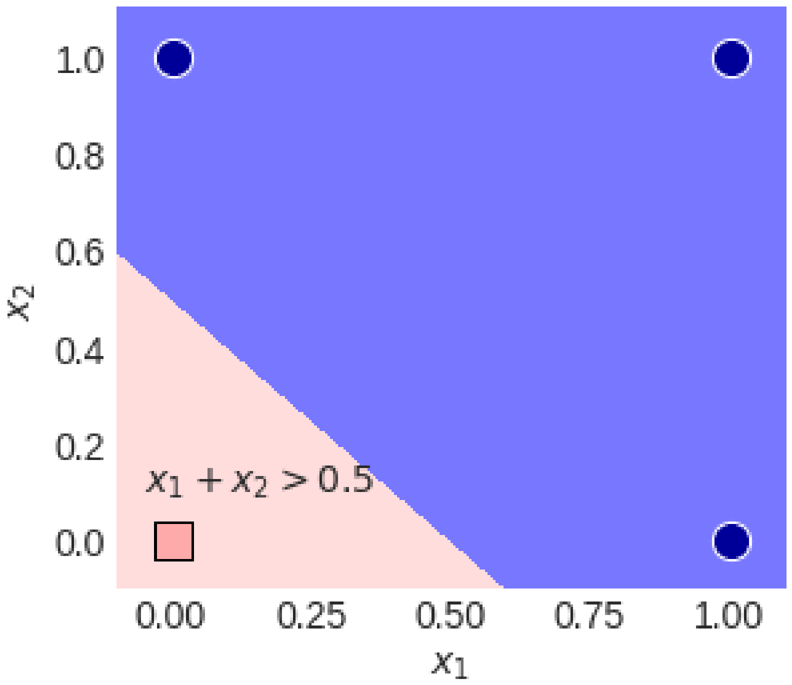
`threshold`	Thresholds each element of the input Tensor.
`threshold_`	In-place version of `threshold()`.
`relu`	Applies the rectified linear unit function element-wise.
`relu_`	In-place version of `relu()`.
`hardtanh`	Applies the HardTanh function element-wise.

Что может один нейрон

Логическое И



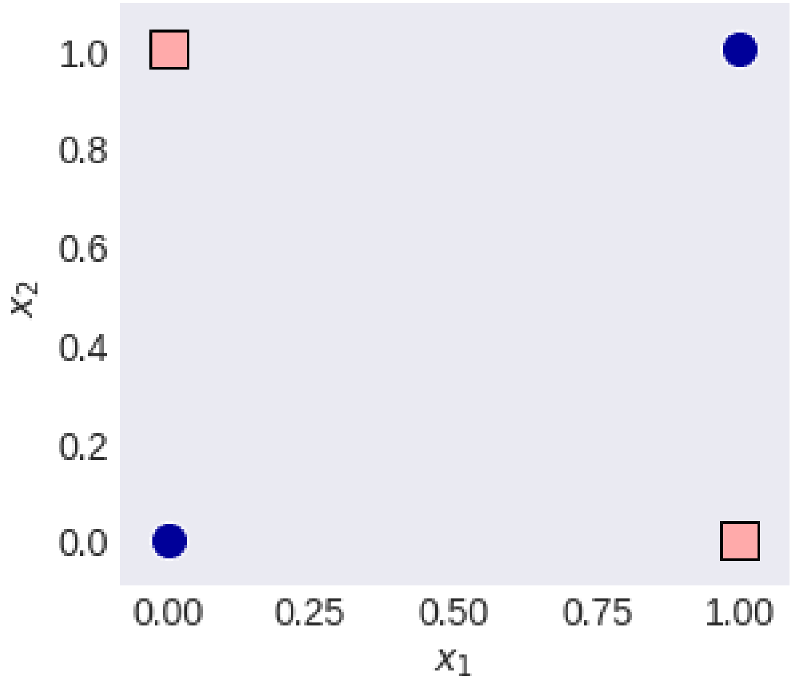
Логическое ИЛИ



Для простоты – пороговая функция активации

Что НЕ может один нейрон

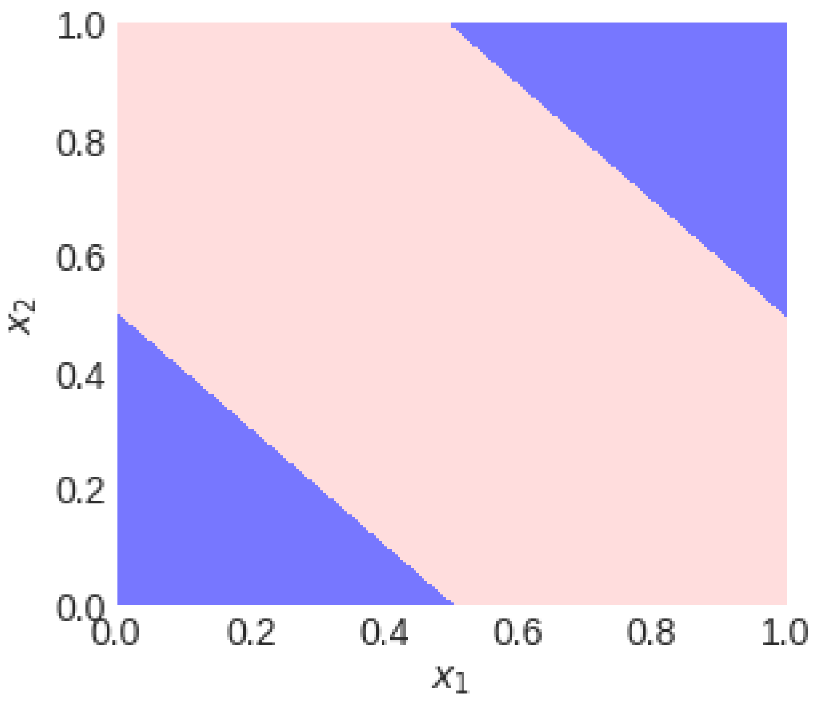
Исключающее ИЛИ / эквивалентность

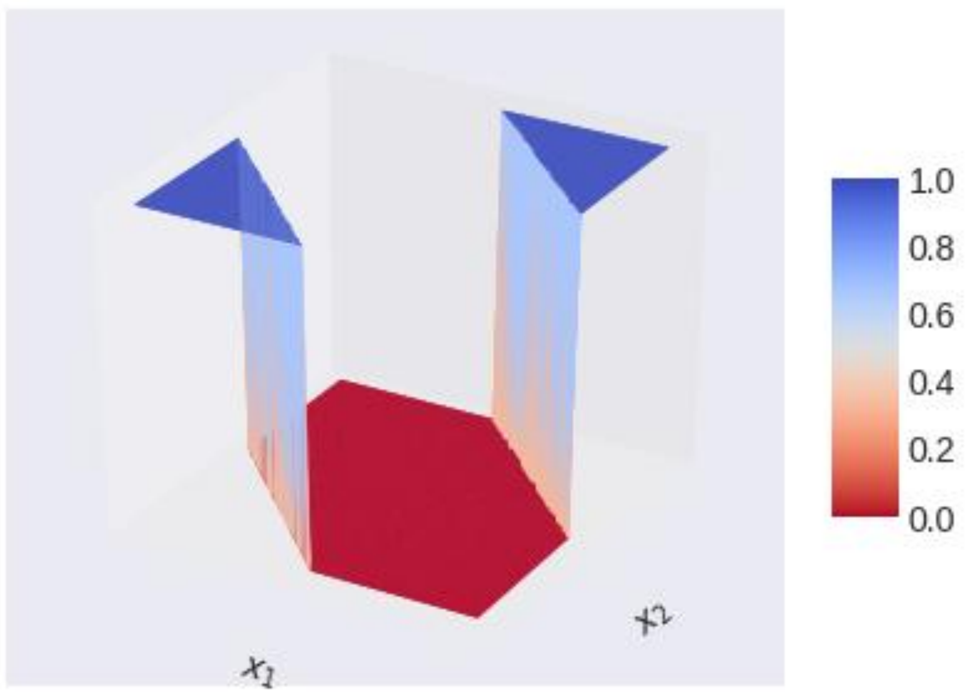


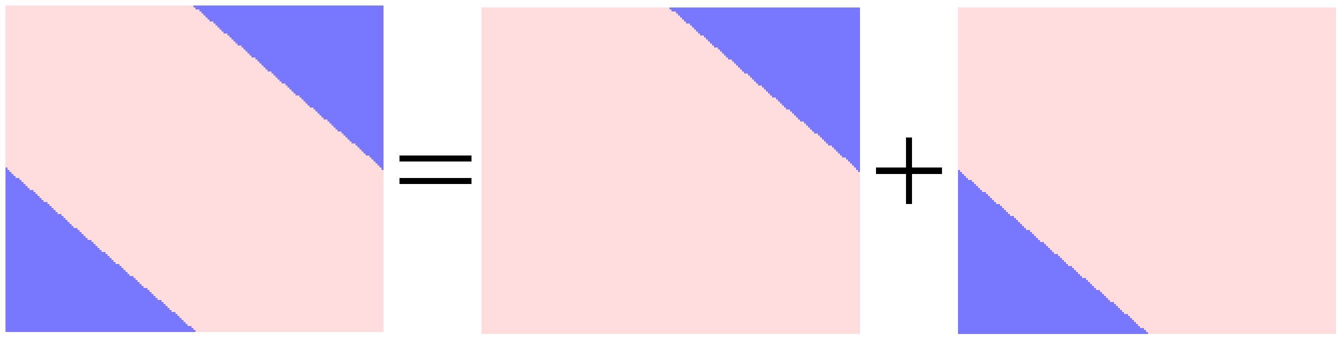
$$\mathrm{th}(\mathrm{th}(x_1 + x_2 - 1.5) + \mathrm{th}(-x_1 - x_2 + 0.5) - 0.5)$$

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

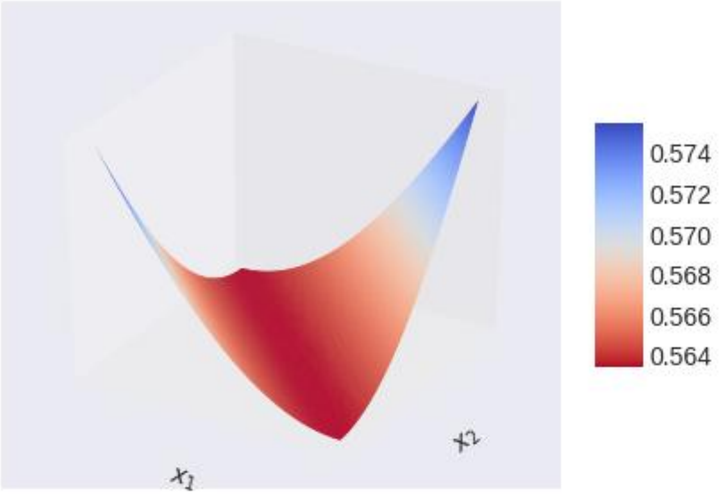
Что НЕ может один нейрон

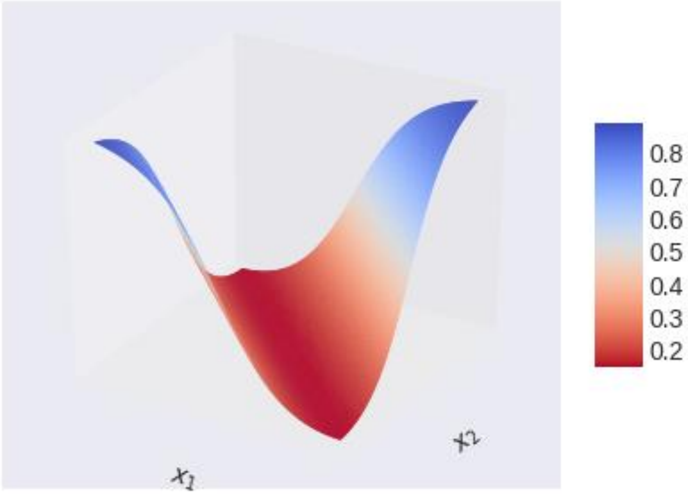


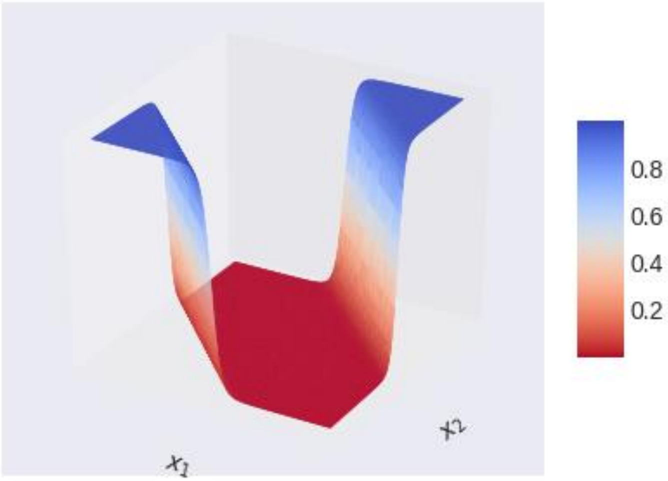




Сигмоида стремится к пороговой функции







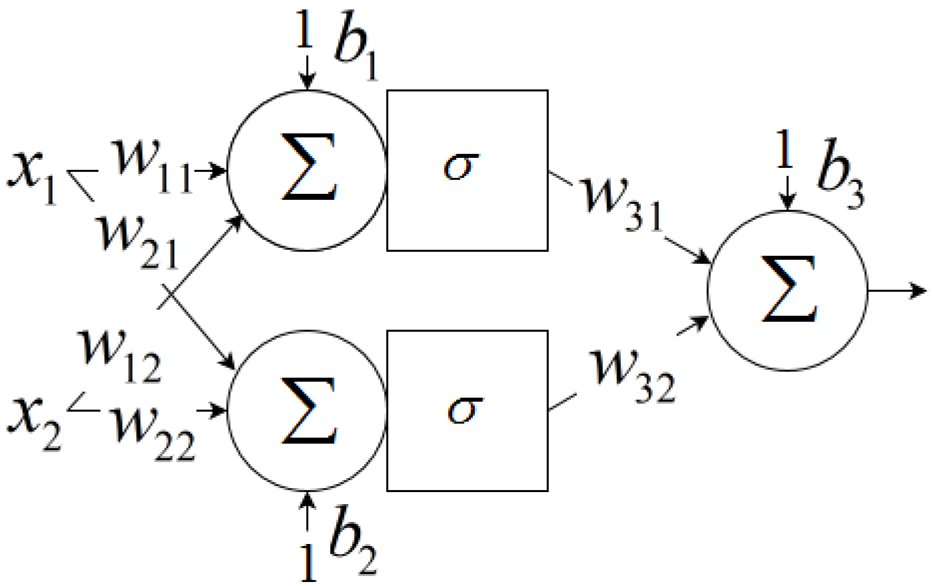
$$\sigma_c(\sigma_c(x_1 + x_2 - 1.5) + \sigma_c(-x_1 - x_2 + 0.5) - 0.5)$$

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

- **Сигмоиду проще обучать – дифференцируемая**
  - **Есть возможность получать «вероятности»**

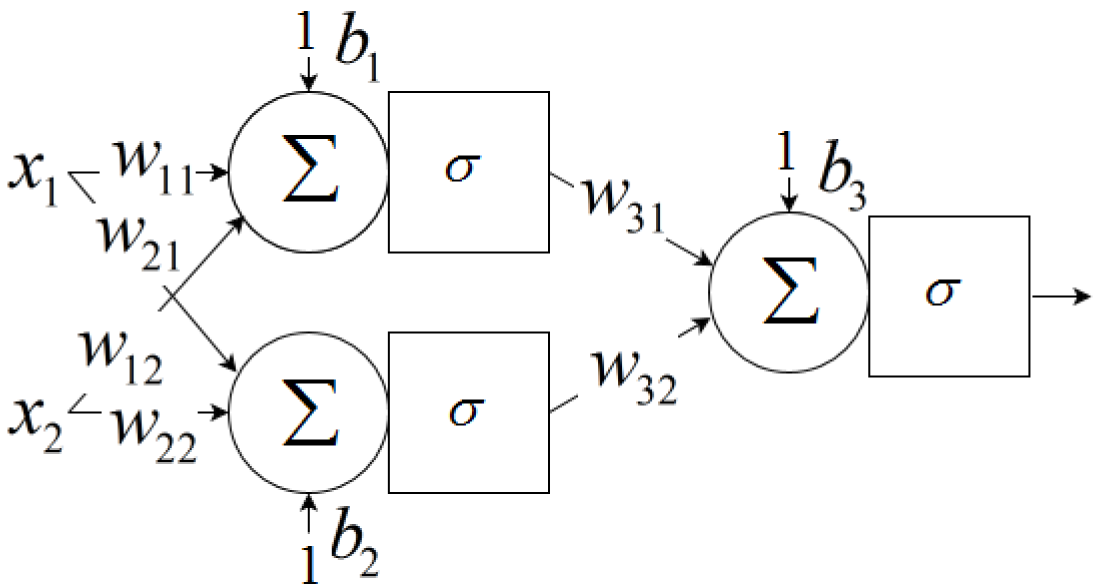
Двуслойная нейронная сеть

Регрессия



$$a = b_3 + w_{31}\sigma(b_1 + w_{11}x_1 + w_{12}x_2) + \\ + w_{32}\sigma(b_2 + w_{21}x_1 + w_{22}x_2)$$

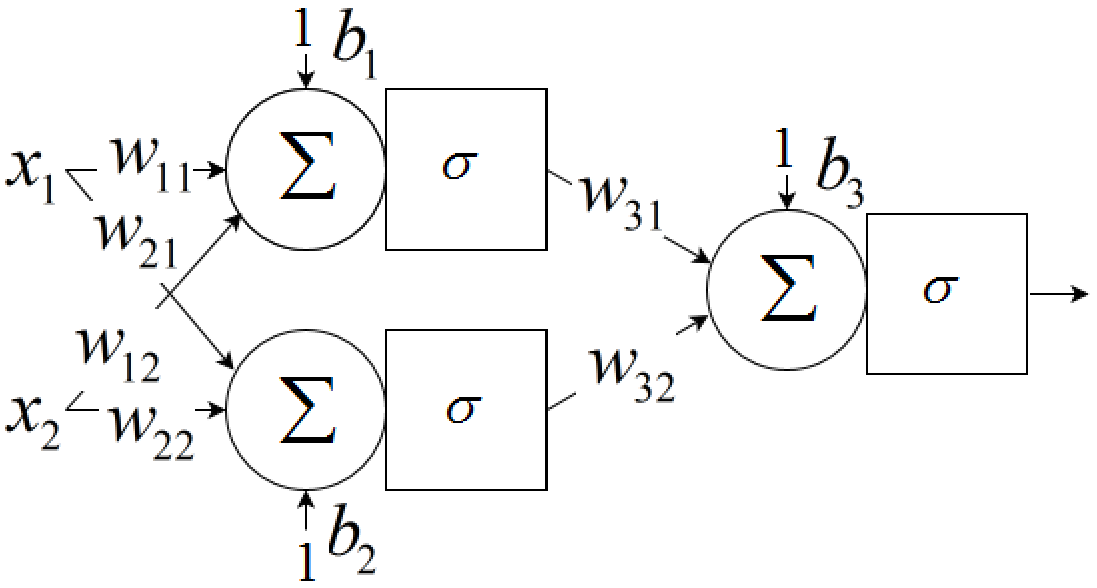
Классификация



$$a = \sigma(b_3 + w_{31}\sigma(b_1 + w_{11}x_1 + w_{12}x_2) + \\ + w_{32}\sigma(b_2 + w_{21}x_1 + w_{22}x_2))$$

Такой нейронной сети хватит… (в первом слое м.б. больше нейронов)

Двуслойная нейронная сеть



$$\sigma\left(\begin{bmatrix} w_{31} & w_{32} & b_3 \end{bmatrix} \begin{bmatrix} \sigma\left(\begin{bmatrix} w_{11} & w_{12} & b_1 \\ w_{21} & w_{22} & b_2 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}\right) \\ 1 \end{bmatrix}\right)$$

Теорема об универсальной аппроксимации [Hornik, 1991]

Любую непрерывную функцию можно с любой точностью приблизить нейросетью глубины 2 с сигмоидной функцией активации на скрытом слое и линейной функции на выходном слое

Нейросеть глубины два с фиксированной функцией активации в первом слое и линейной функцией активации во втором может равномерно аппроксимировать (м.б. при увеличении числа нейронов в первом слое) любую непрерывную функцию на компактном множестве тогда и только тогда, когда функция активации неполиномиальная.

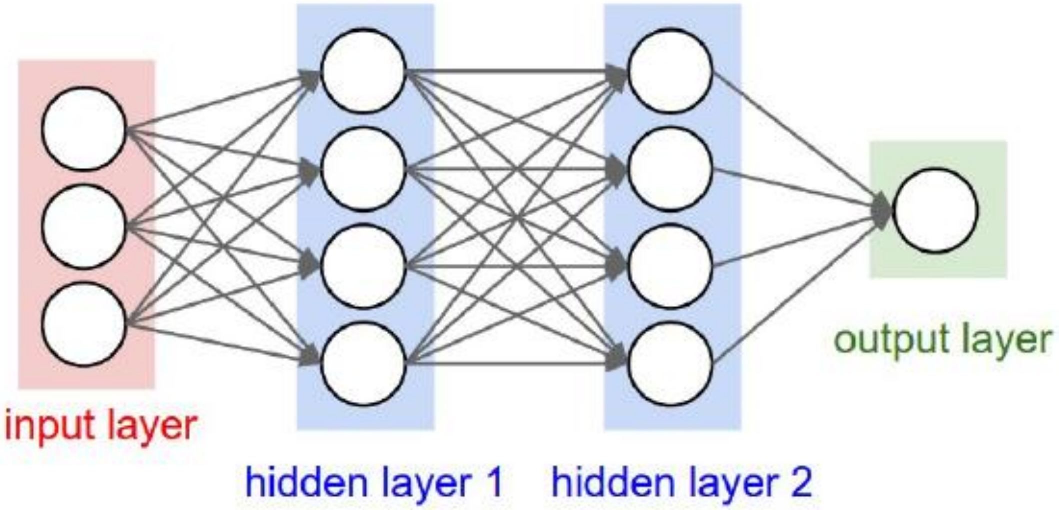
http://www2.math.technion.ac.il/~pinkus/papers/neural.pdf

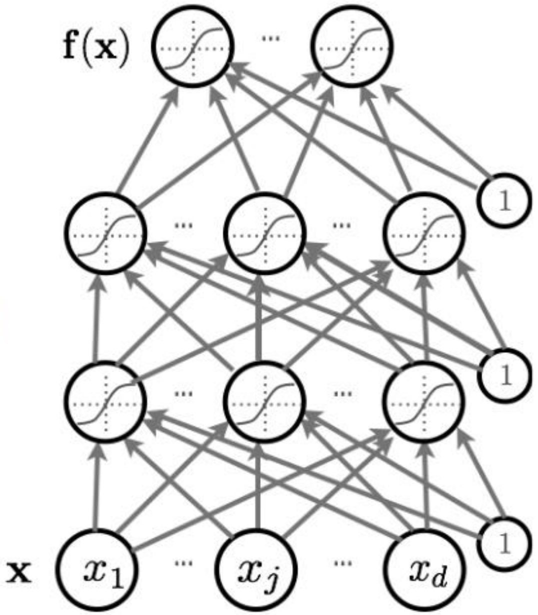
Более того, функция активации м.б. любая (неполиномиальная)!

Но…

- много нейронов (неизвестно сколько)
- экспоненциальные веса
- сложность обучения

Многослойная нейронная сеть – пример нелинейной модели





Ориентированный граф вычислений

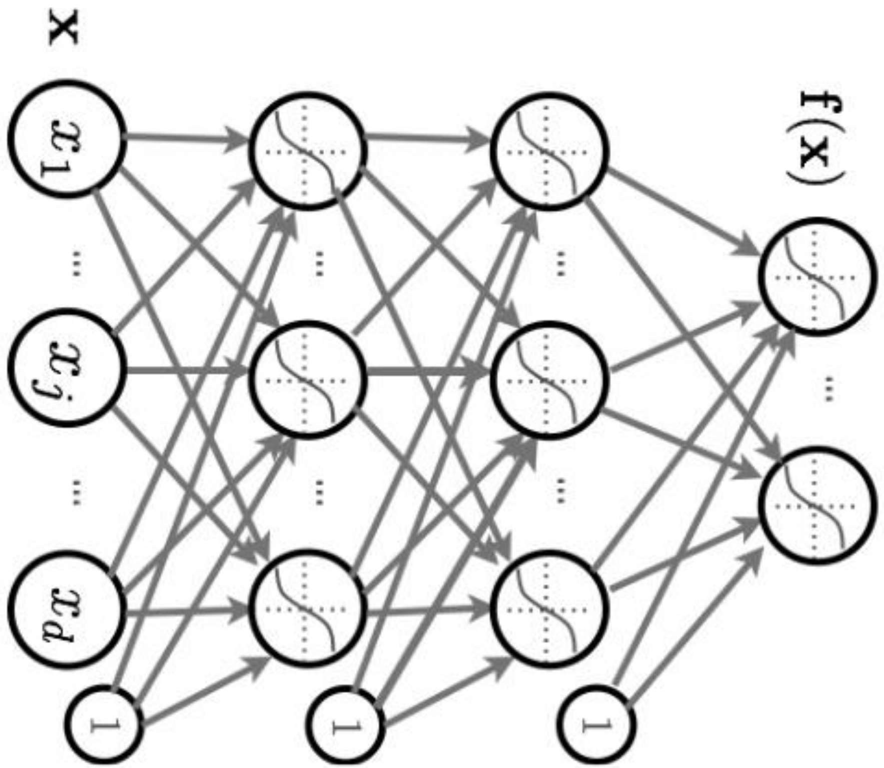
Вершины – переменные или нейроны

Рёбра – зависимости

Сеть прямого распространения – Feedforward Neural Network (FFN)

(т.е. нет циклов)

все нейроны предыдущего слоя связаны с нейронами следующего



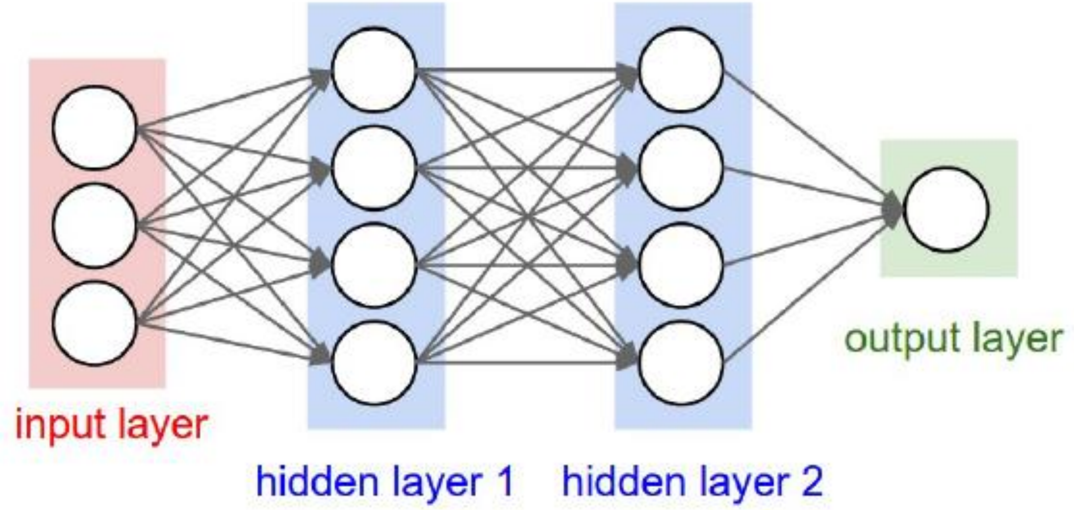
входной слой

один или несколько скрытых слоёв

выходной слой

Важная аналогия

Глубокая НС – последовательное преобразование признакового пространства



$$\varphi_k(W_k \cdot \ldots \cdot \varphi_2(W_2 \cdot \varphi_1(W_1 \cdot x)))$$

иногда чуть другая запись!

Сейчас наука DL, в основном, как правильно представлять (преобразовывать) признаковые пространства

увидим потом и в таких моделях, как кодировщик

нельзя просто преобразовывать… надо что-то ещё требовать

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], grad_fn=<SigmoidBackward>)
```

Минутка кода

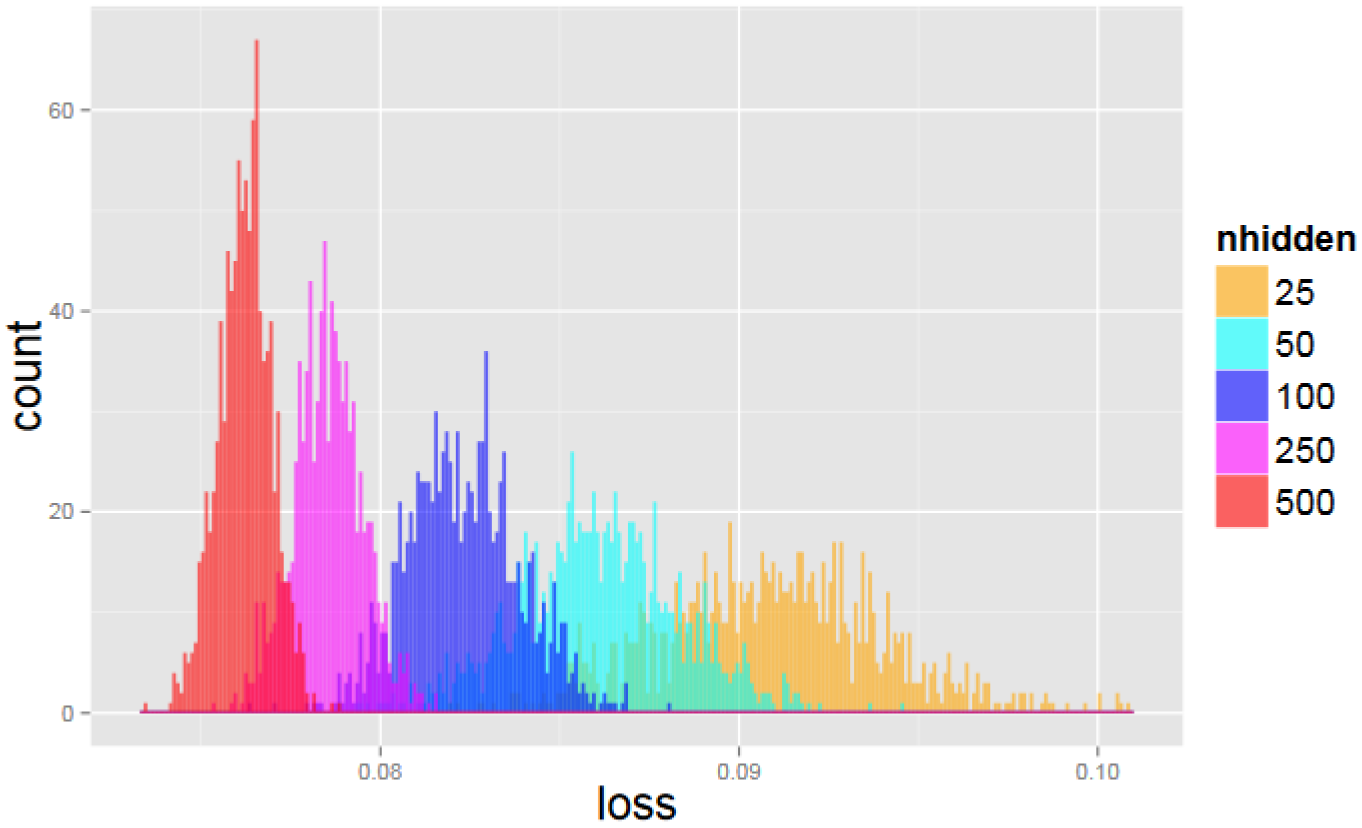
```
net
Feedforward(
  (fc1): Linear(in_features=3, out_features=5, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=5, out_features=1, bias=False)
  (sigmoid): Sigmoid()
)

net.fc1.weight.data, net.fc1.bias.data, net.fc2.weight.data # net.fc2.bias.data
(tensor([[-0.5727,  0.1885, -0.4232],
         [-0.1383, -0.2233, -0.5384],
         [-0.3583,  0.1175, -0.5696],
         [-0.4284,  0.1711,  0.0918],
         [-0.5539, -0.2273,  0.4973]]),
 tensor([ 0.5525, -0.4201, -0.5736,  0.0464, -0.0297]),
 tensor([[-0.1064, -0.3029,  0.4057,  0.0953,  0.2823]]))
```

или

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

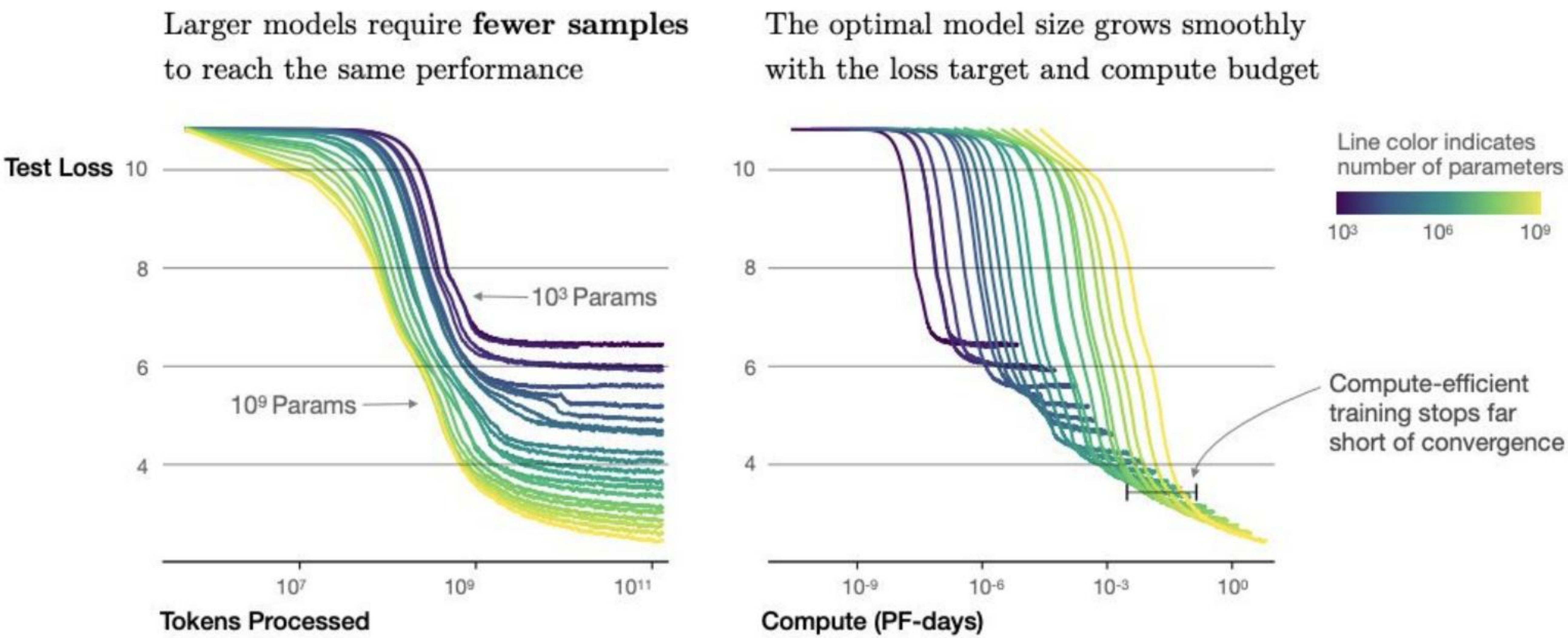
как реализовать `Feedforward([3, 4, 5, 3, 1])`?

Зачем нужны глубокие нейронные сети



Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, Yann LeCun «The Loss Surfaces of Multilayer Networks» 2015, https://arxiv.org/abs/1412.0233

Зачем нужны глубокие нейронные сети



Большие модели требуют меньше данных!

stateofai 2020

Глубокие нейронные сети

- **Много слоёв**
- **Много данных**
- **Достаточно вычислительных мощностей**

Обучение

Как принято…
минимизация регуляризованного эмпирического риска

$$\frac{1}{m}\sum_{t=1}^{m} L(a(x_i \mid w), y_i) + \lambda R(w) \to \min_w$$

Задача оптимизации невыпуклая!

«Настройка» нейронной сети – получение весов w

Метод стохастического градиента (SGD)

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

т.к. очень много слагаемых… и так быстрее;) где здесь неточность?

Метод стохастического градиента

1. **Случайная инициализация весов** $w^{(0)} \sim \mathrm{norm}(0, \sigma^2)$

2. **Цикл по t до сходимости**

   2.1. **Выбираем случайный объект** x_i

   2.2. **Вычисляем градиент**

$$\nabla[L(a(x_i \mid w^{(t)}), y_i) + \lambda R(w^{(t)})]$$

   2.3. **Адаптация весов**

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

почему такая инициализация?

Функции ошибки

Классификация – logloss (CrossEntropyLoss)

$$L((a_1,\ldots,a_l),y) = -\log\frac{\exp(a_y)}{\sum_{j=1}^{l}\exp(a_j)} = -a_y + \log\sum_{j=1}^{l}\exp(a_j)$$

такая реализация не очень «устойчива»

(сумма больших экспонент, разница близких чисел)

– Часто при реализации делают так:

$$-a_y + \max\{a_j\} + \log\left(\sum_{j=1}^{l}\exp(a_j - \max\{a_j\})\right)$$

Минутка кода

```
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, target, loss

(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),
 tensor([4, 3, 1]),
 tensor(2.1341, grad_fn=<NllLossBackward>))
```

перечисляются номера классов

Обратное распространение градиента (Backpropagation)

Идея: вычисление производной сложной функции

$$L = \sum_t \mathrm{loss}(y_t - f(x_t, w)) \sim L(w, f_1(w), \ldots, f_k(w))$$

$$\nabla L(w, f_1(w), \ldots, f_k(w)) = \frac{\partial L}{\partial f_1} \nabla f_1(w) + \ldots + \frac{\partial L}{\partial f_k} \nabla f_k(w)$$

Автоматическое дифференцирование

Прямое распространение
вычисление ответов, значения ошибки

$$x, w \to L(w, x, f_1(w, x), \ldots, f_k(w, x))$$

Обратное распространение
вычисление градиентов

$$x, w, \nabla f_1, \ldots, \nabla f_k \to \nabla L$$

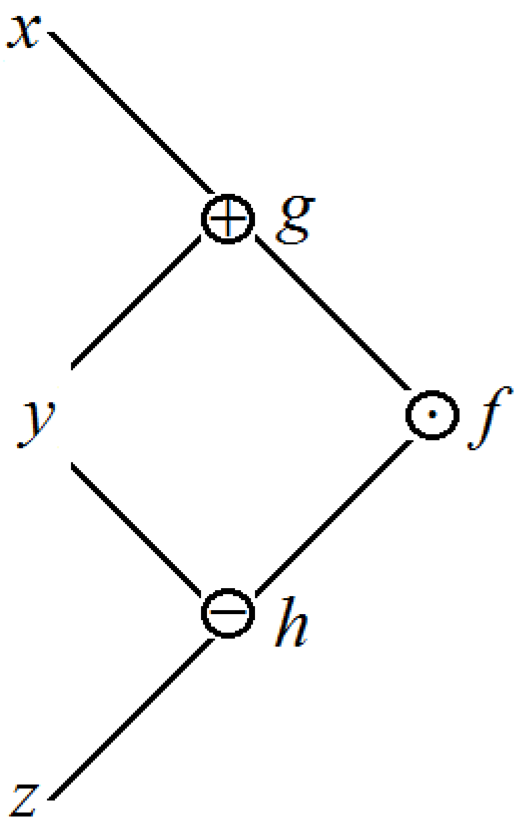
Нейросеть – вычислительный граф

Другой взгляд на НС
вершины – переменные (входные, внутренние, выходные)
рёбра – зависимости (+ веса)
слой – операция

тут м.б. более широкое понятие слоя:
линейная комбинация,
нелинейность,
...

Вычисление градиента на графе

$$f=(x+y)\cdot(y-z)$$



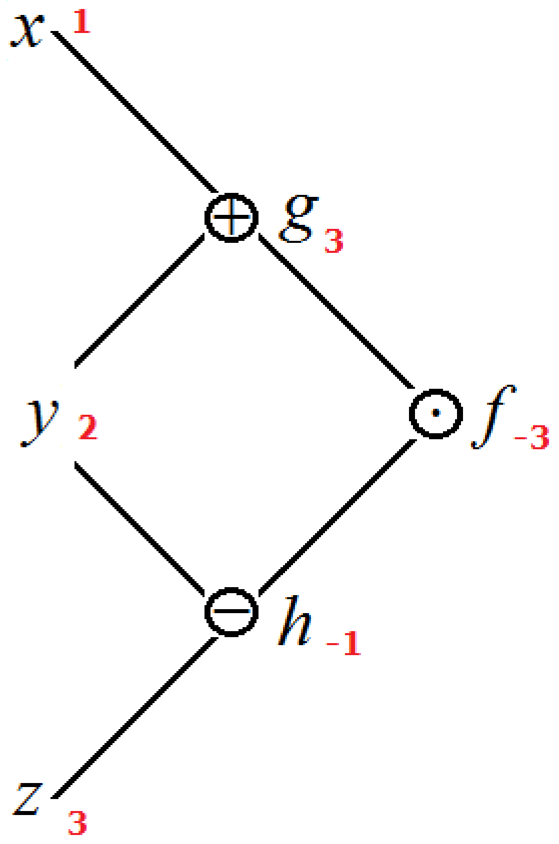
$$f(x,y,z)=\underbrace{(x+y)}_{g(x,y)}\cdot\underbrace{(y-z)}_{h(y,z)}$$

Как проводится вычисление функции?

$$x,y,z=1,2,3$$

Вычисление градиента на графе

$$f = (x + y)\cdot(y - z)$$



$$f(x,y,z) = \underbrace{(x+y)}_{g(x,y)}\cdot\underbrace{(y-z)}_{h(y,z)}$$

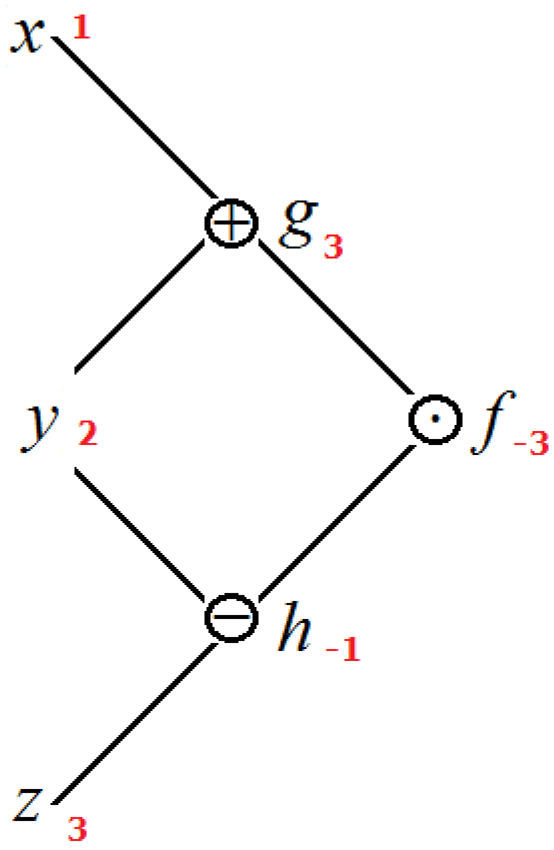
Как проводится вычисление функции?

$$x, y, z = 1, 2, 3$$

«Прямой ход»

Вычисление градиента на графе

$$f=(x+y)\cdot(y-z)$$



$$f(x,y,z)=\underbrace{(x+y)}_{g(x,y)}\cdot\underbrace{(y-z)}_{h(y,z)}$$

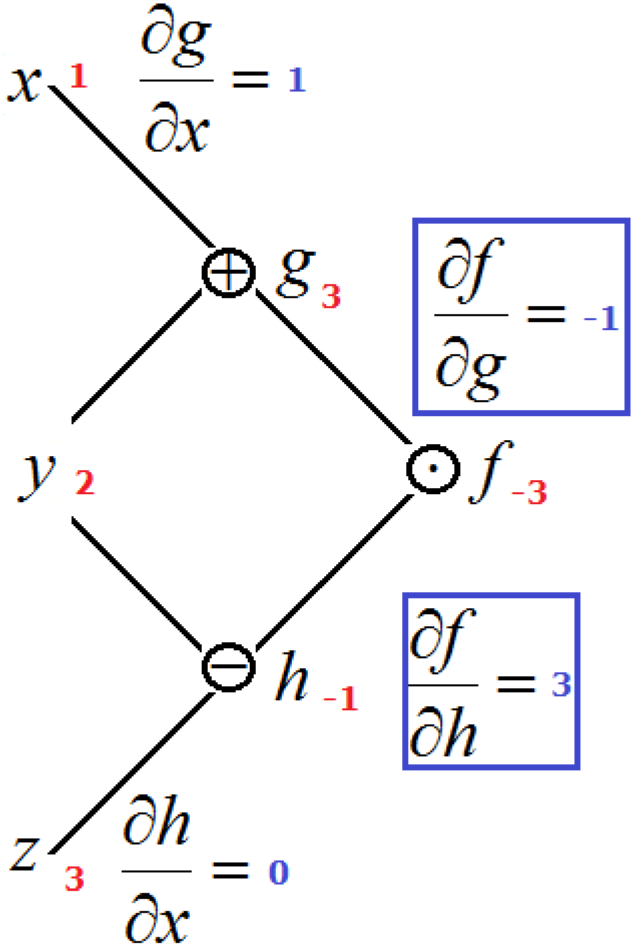
Как проводится вычисление производных?

$$\frac{\partial f}{\partial g}=h,\ \frac{\partial f}{\partial h}=g$$

$$\frac{\partial g}{\partial x}=1,\ \frac{\partial h}{\partial x}=0$$

$$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial g}\frac{\partial g}{\partial x}+\frac{\partial f}{\partial h}\frac{\partial h}{\partial x}$$

Вычисление градиента на графе



$$\frac{\partial f}{\partial x} = \boxed{\frac{\partial f}{\partial g}} \frac{\partial g}{\partial x} + \boxed{\frac{\partial f}{\partial h}} \frac{\partial h}{\partial x}$$

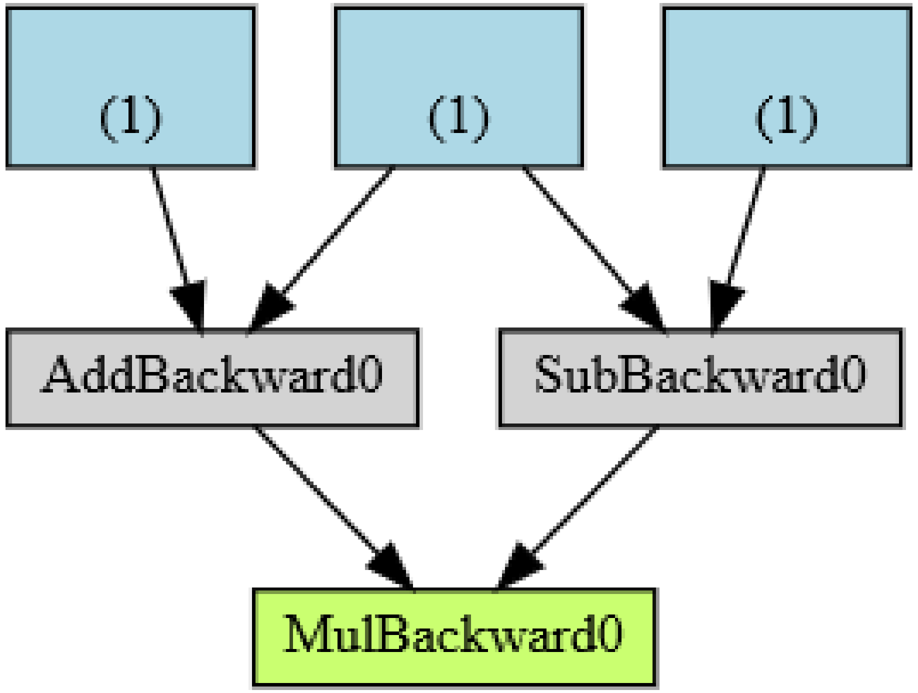
$$f(x,y,z) = \underbrace{(x+y)}_{g(x,y)} \cdot \underbrace{(y-z)}_{h(y,z)}$$

Как проводится вычисление производных?

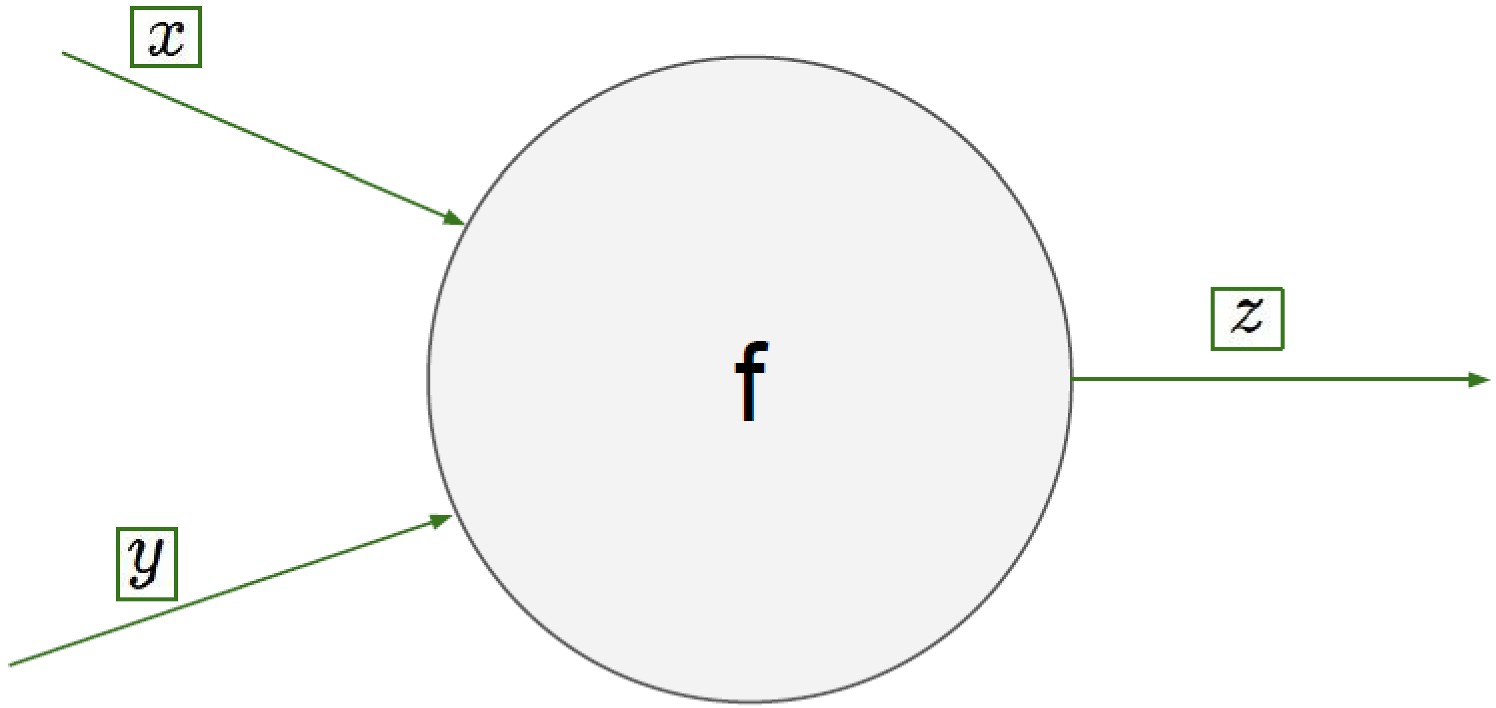
«Обратный ход»

Минутка кода: PyTorch

```
import torch
from torch.autograd import Variable

# переменные
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 + y) * (y - z) # прямой проход - вычисление

f.backward() # обратный проход - выч. производных
x.grad, y.grad, z.grad # производные

(tensor([-1.]), tensor([2.]), tensor([-3.]))

from torchviz import make_dot

make_dot(f)
```



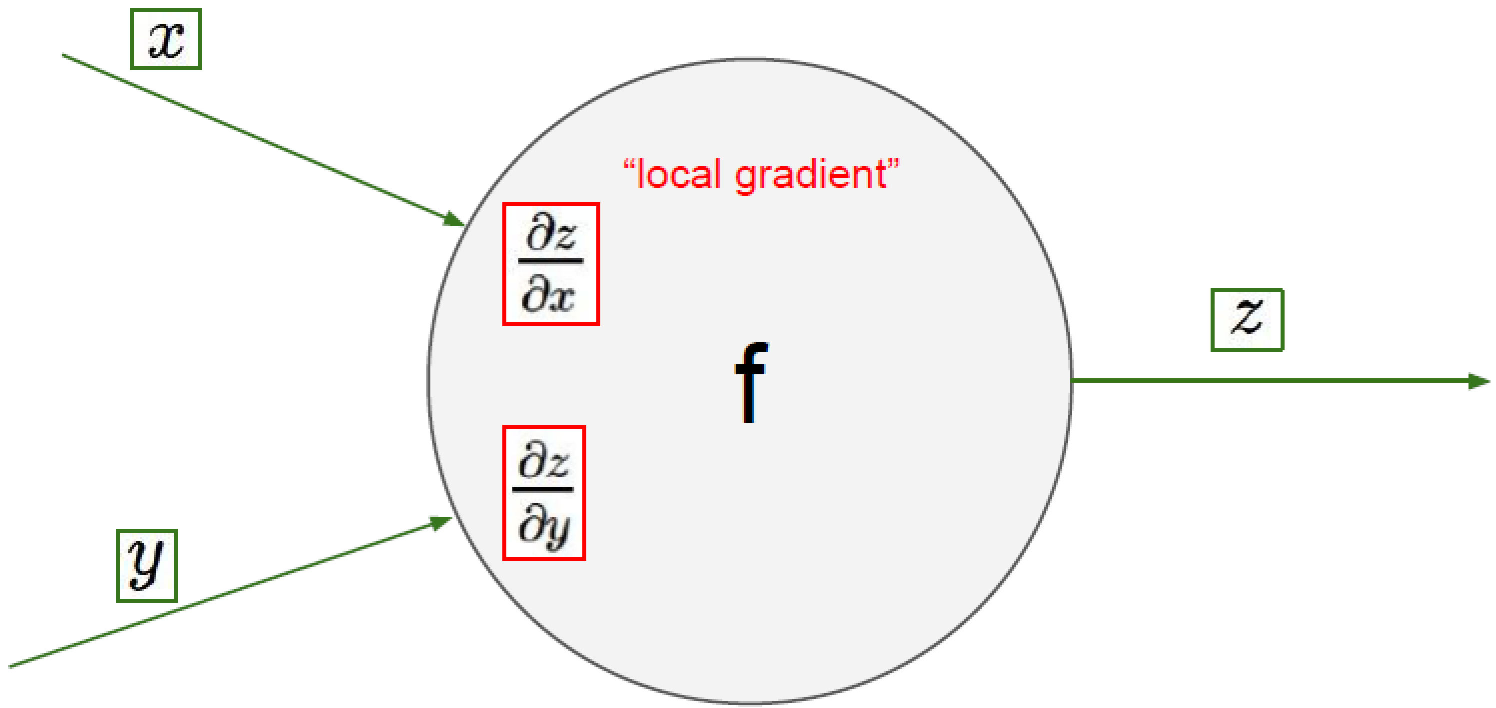
Обратное распространение градиента (Backpropagation)



http://cs231n.stanford.edu/2017/index.html

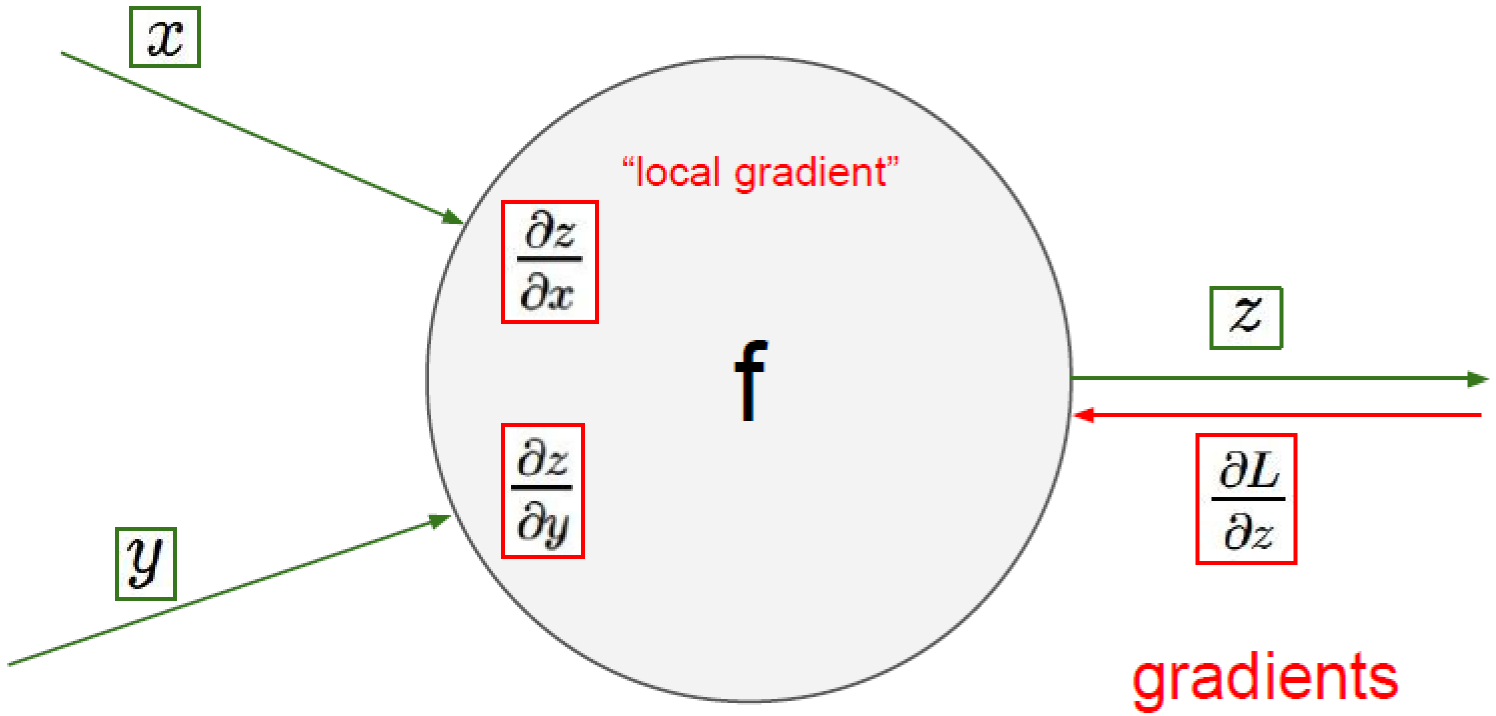
Обратное распространение = SGD + дифференцирование сложных функций

Обратное распространение градиента (Backpropagation)



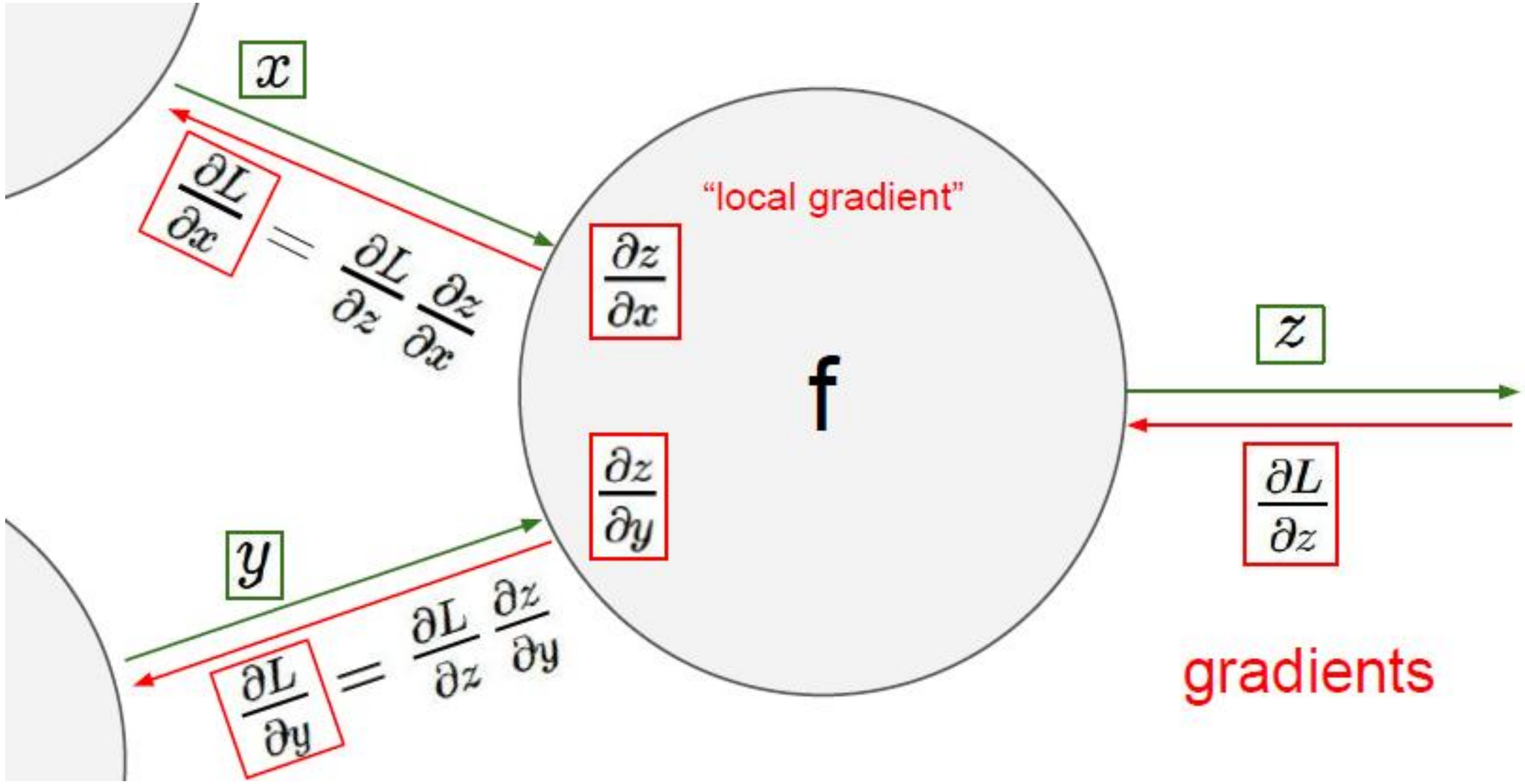
http://cs231n.stanford.edu/2017/index.html

Обратное распространение градиента (Backpropagation)



http://cs231n.stanford.edu/2017/index.html

Обратное распространение градиента (Backpropagation)

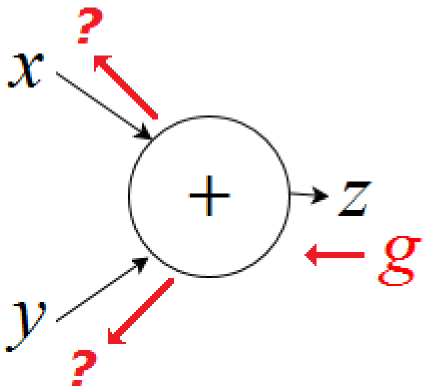


http://cs231n.stanford.edu/2017/index.html

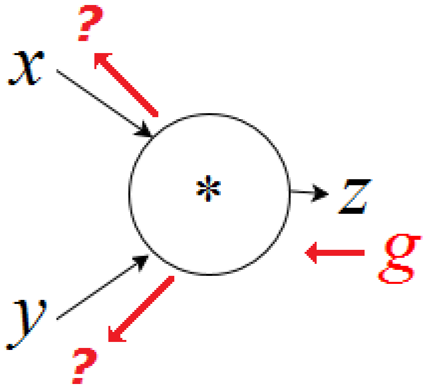
Прохождение градиента через гейты

Как здесь проходит градиент?

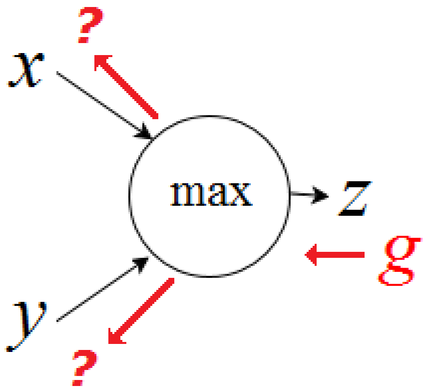
$$z = x + y$$



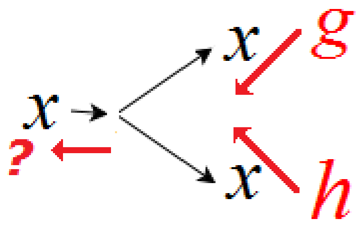
$$z = x * y$$



$$z = \max(x, y)$$



$$z_1 = z_2 = x$$

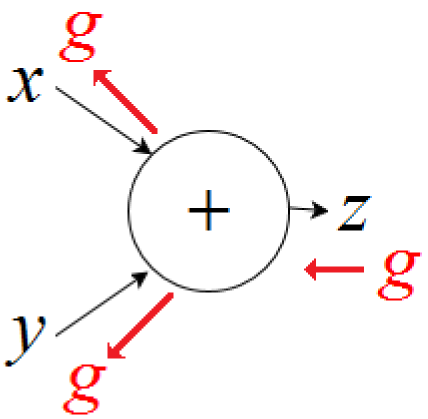


Прохождение градиента через гейты

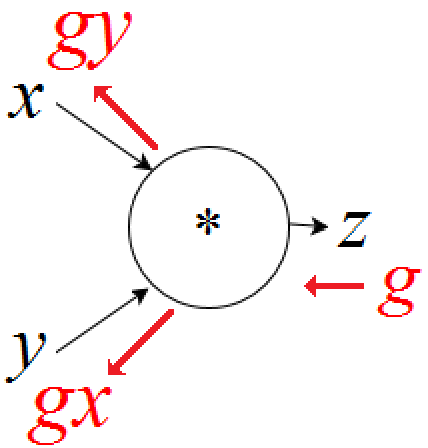
Как здесь проходит градиент?

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} = g\frac{\partial (x+y)}{\partial x} = g$$

$$z = x + y$$



$$z = x * y$$

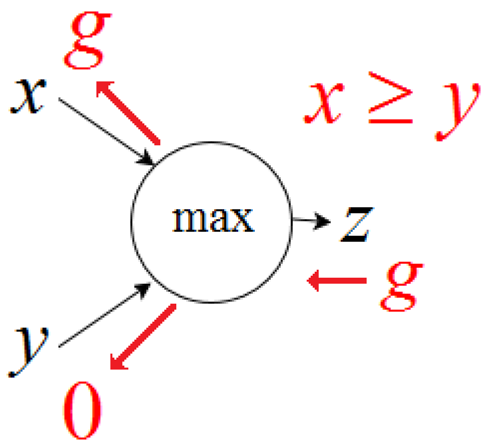


$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} = g\frac{\partial (xy)}{\partial x} = gy$$

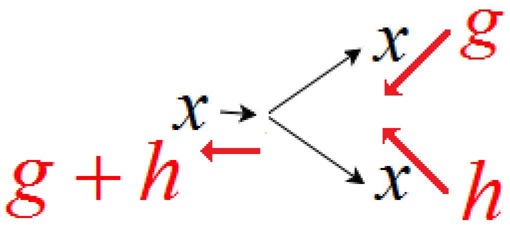
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} = g\frac{\partial \max(x,y)}{\partial x} = \begin{cases} g, & x > y \\ 0, & y \geq x \end{cases}$$

А что при x=y?!

$$z = \max(x, y)$$



$$z_1 = z_2 = x$$



Подумать, почему так? Потом станет яснее…

Прохождение градиента через гейты

Можно и в векторном виде. Напомним

производная

$$x \in \mathbb{R}, z \in \mathbb{R} \Rightarrow$$
$$\frac{\partial z}{\partial x} \in \mathbb{R}$$

Градиент

$$x \in \mathbb{R}^n, z \in \mathbb{R} \Rightarrow$$
$$\frac{\partial z}{\partial x} = \left(\frac{\partial z}{\partial x_1}, \ldots, \frac{\partial z}{\partial x_n} \right)^{\mathrm{т}} \in \mathbb{R}^n$$

Якобиан

$$x \in \mathbb{R}^n, z \in \mathbb{R}^m \Rightarrow$$
$$\frac{\partial z}{\partial x} = \left\| \frac{\partial z_j}{\partial x_i} \right\|_{n \times m} \in \mathbb{R}^{n \times m}$$

в вики матрица Якоби по-другому (транспонирована)

$$\frac{\partial z}{\partial x} = \left\| \begin{matrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_m}{\partial x_1} \\ \cdots & \cdots & \cdots \\ \frac{\partial z_1}{\partial x_n} & \cdots & \frac{\partial z_m}{\partial x_n} \end{matrix} \right\|_{n \times m} \in \mathbb{R}^{n \times m}$$

Прохождение градиента через гейты

Пример (меняем порядок множителей, чтобы произведения совпадали по размерам)

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

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

Для линейного слоя

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

$$\frac{\partial z}{\partial x} = \frac{\partial Wx}{\partial x} = \frac{\partial \begin{bmatrix} w_{11}x_1 + \ldots + w_{1n}x_n \\ \ldots \\ w_{m1}x_1 + \ldots + w_{mn}x_n \end{bmatrix}}{\partial x} = \begin{bmatrix} w_{11} & \ldots & w_{m1} \\ \ldots & \ldots & \ldots \\ w_{1n} & \ldots & w_{mn} \end{bmatrix} = W^{\mathrm{T}}$$

Прохождение градиента через гейты

Линейный слой + активация

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

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

Несколько слоёв...

$$z = \mathrm{ReLU}(W_k(\ldots W_2\,\mathrm{ReLU}(W_1 x)))$$

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

Если бы все матрицы совпадали и функция активации была бы тождественной, то

$$\frac{\partial z}{\partial x} = (W^{\mathrm{T}})^k$$

Прохождение градиента через гейты

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

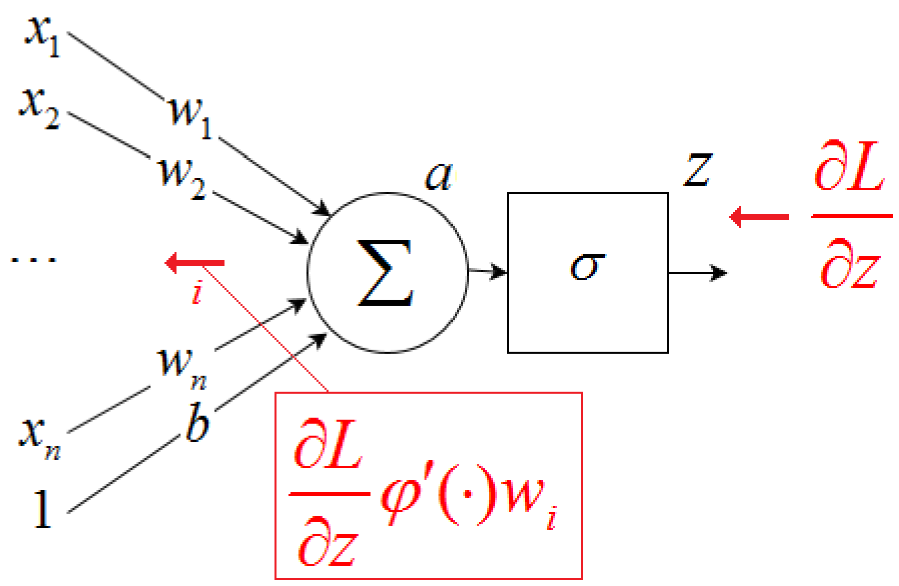
Берём производные по матрице весов

$$\frac{\partial L}{\partial W} = \underbrace{\frac{\partial L}{\partial z}}_{m\times 1}\frac{\partial z}{\partial W} = \frac{\partial L}{\partial z}\underbrace{x^{\mathrm{T}}}_{1\times n} = \nabla_z L \cdot x^{\mathrm{T}}$$

Можно убедиться, что это так...

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial z}{\partial w_{ij}}\frac{\partial L}{\partial z} = (\underbrace{0,\ldots,0,x_j}_{i},0,\ldots,0)\cdot\nabla_z L = (\nabla_z L)_{[i]}\cdot x_j$$

Прохождение градиента через нейрон



$$z = \varphi(w_0 + w_1x_1 + \ldots + w_nx_n)$$

для распространения градиента

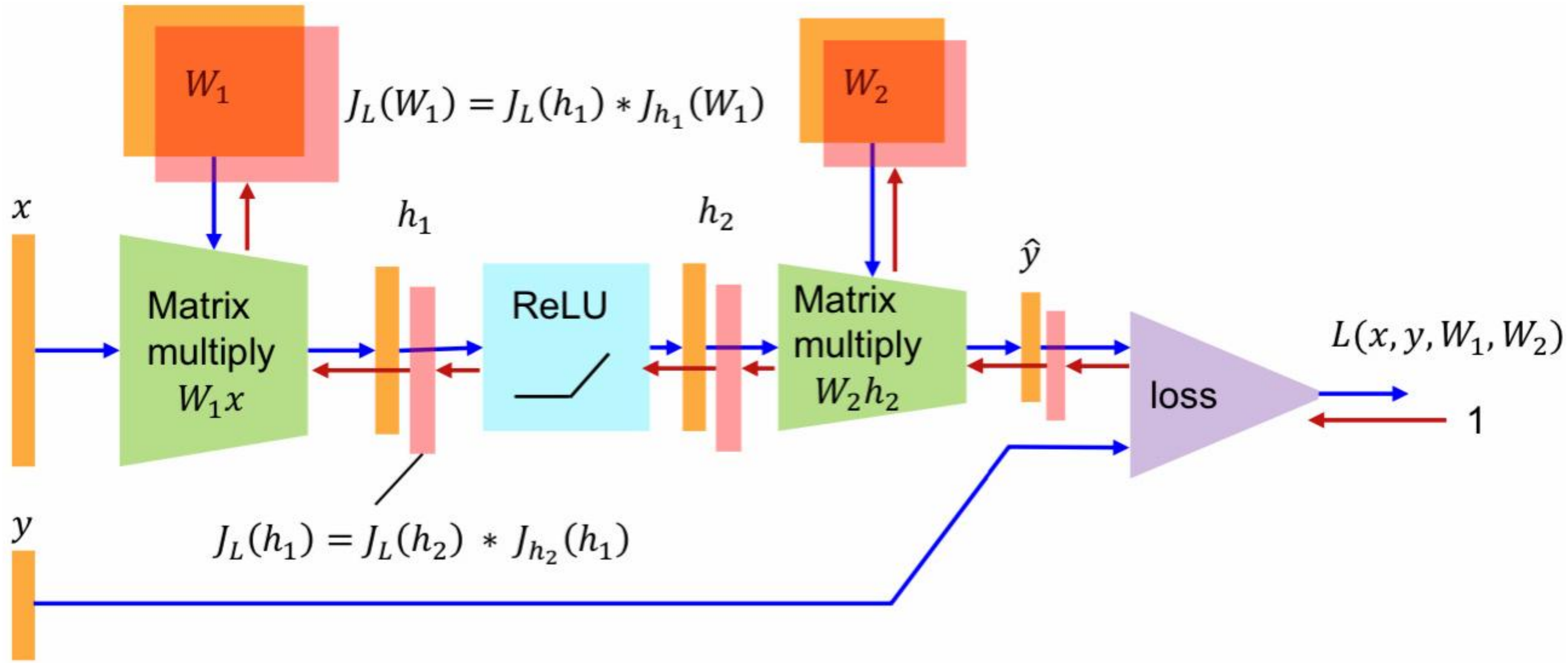
$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x_i} = \frac{\partial L}{\partial z}\varphi'(\cdot)w_i$$

для корректировки весов

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

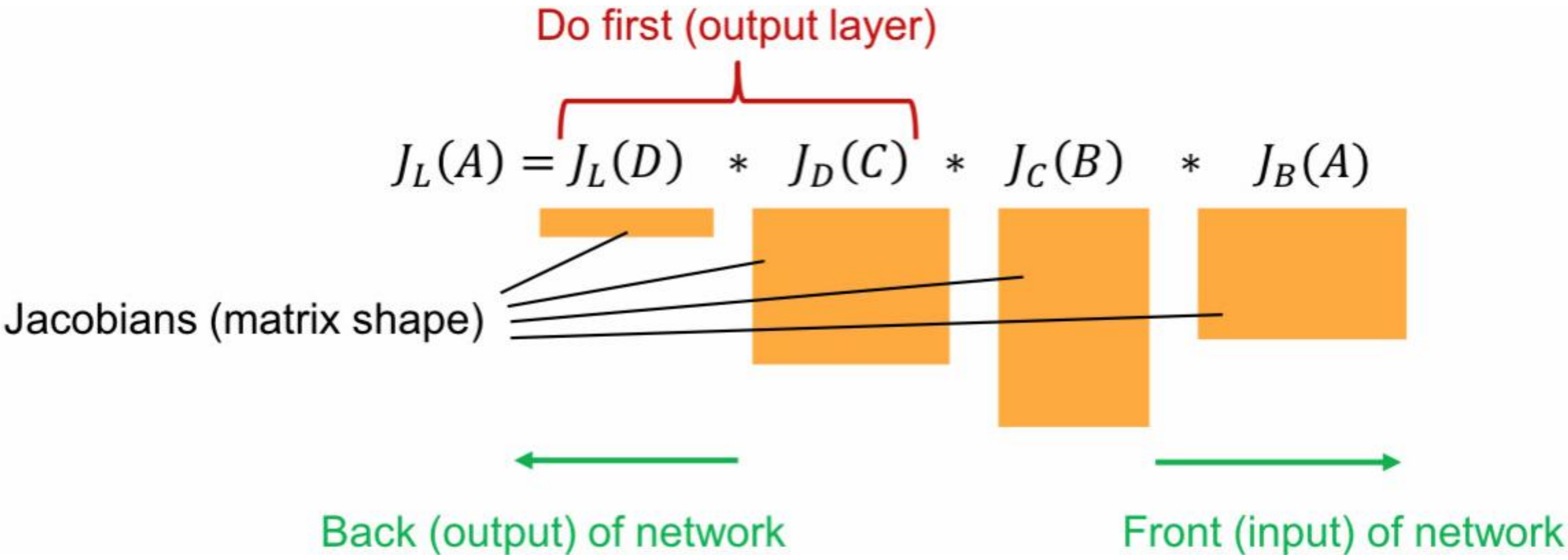
при прохождении градиента через нейрон
происходит умножение на производную функции активации

Обратное распространение градиента (Backpropagation)



https://bcourses.berkeley.edu/courses/1487769/pages/cs-l-w-182-slash-282a-designing-visualizing-and-understanding-deep-neural-networks-spring-2020

Обратное распространение градиента (Backpropagation)



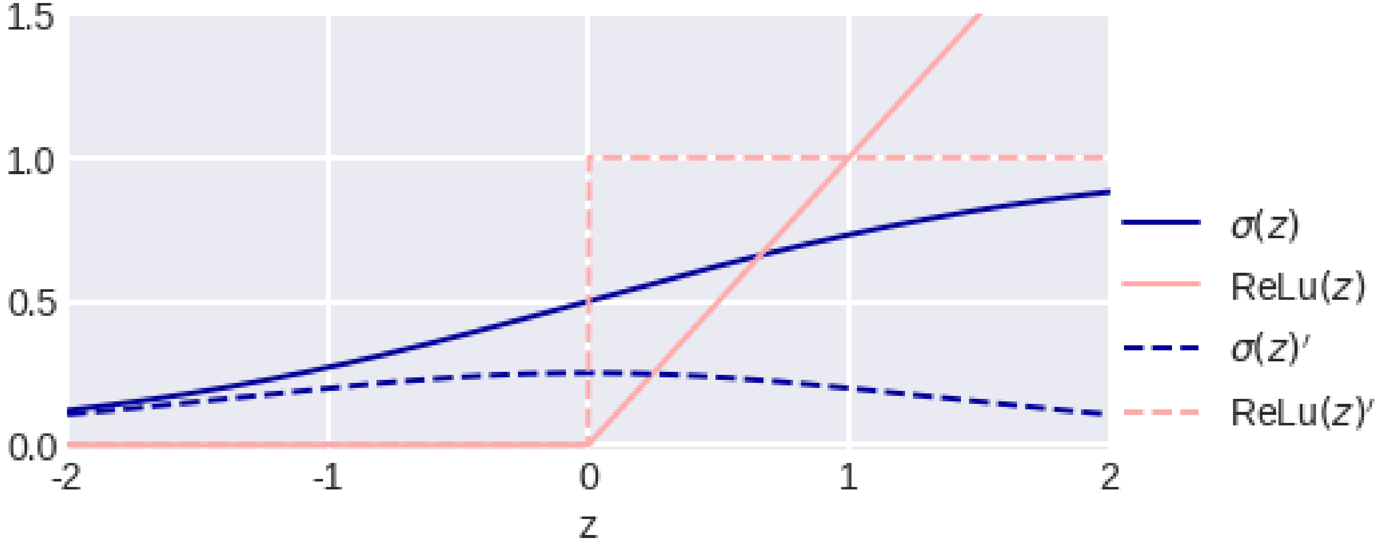
слева – строка, т.к. ошибка – скаляр

Проблема затухания градиента

Производная сигмоиды при «насыщенном сигнале» близка к нулю

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

$$\mathrm{ReLU}(z) = \max(0, z) \qquad \frac{\partial\,\mathrm{ReLU}(z)}{\partial z} = I[z>0]$$



ReLU = Rectified Linear Unit

Что плохого в сигмоиде

- «убивает» градиенты
- выходы не отцентрированы (легко устранить $\to$ tanh)
- вычисление экспоненты всё-таки дорого…

Что хорошего в ReLU

- быстро вычисляется
- есть теоретические / биологические обоснования
- зоны константного градиента и обнуления (разреженное решение)

«Dead Neurons»

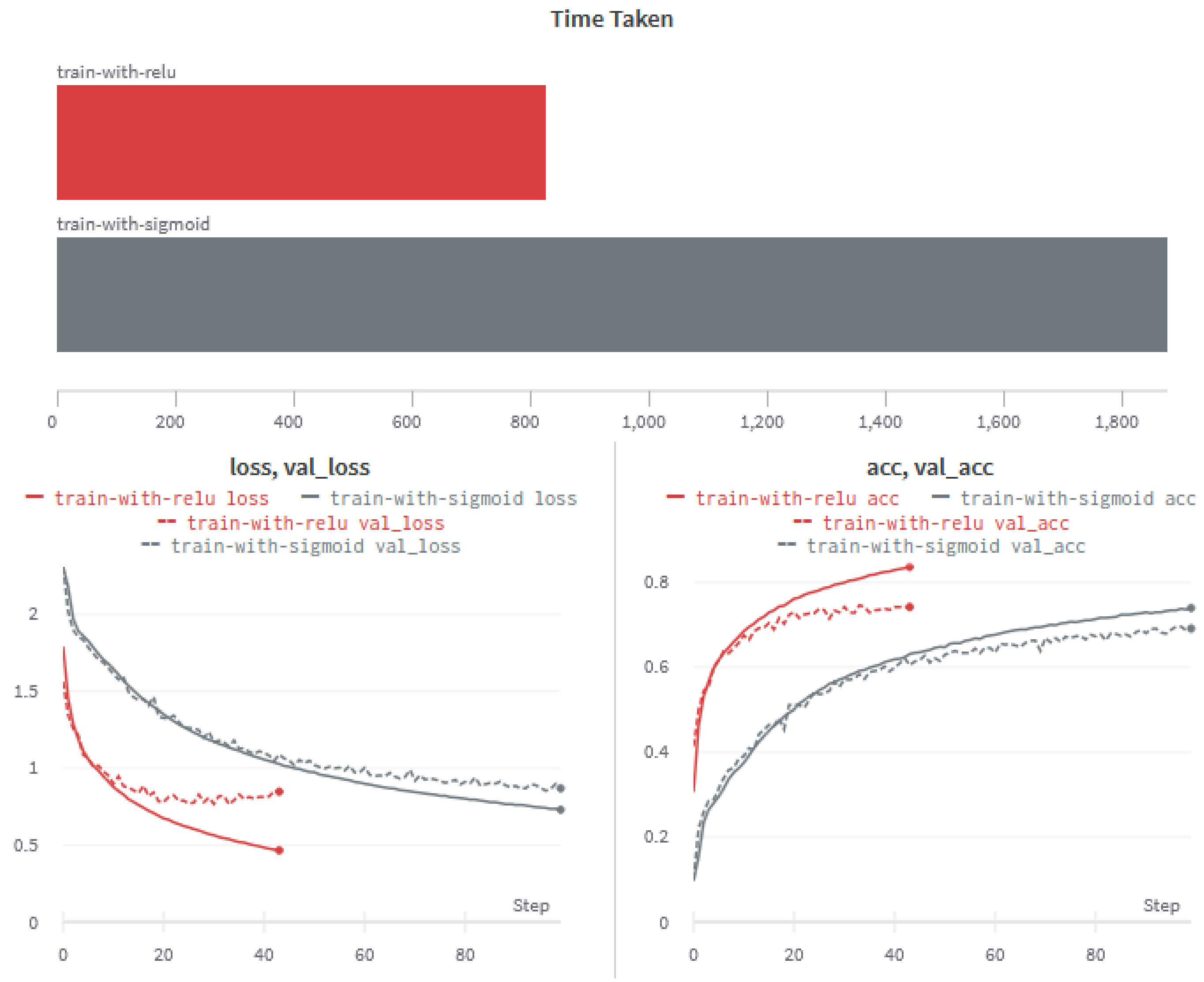
Мёртвые нейроны – выдают ноль на любом объекте выборки

Их можно найти

В ReLU есть опасность их появления

- **Но вероятность мала!**
- **При обучении их число уменьшается**
- **В глубоких сетях с ReLU они, действительно, могут стать проблемой**

Sigmoid vs ReLU



https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks-Why-ReLU-is-so-Prevalent--VmlldzoyMDk0MzI

Ещё функции активации

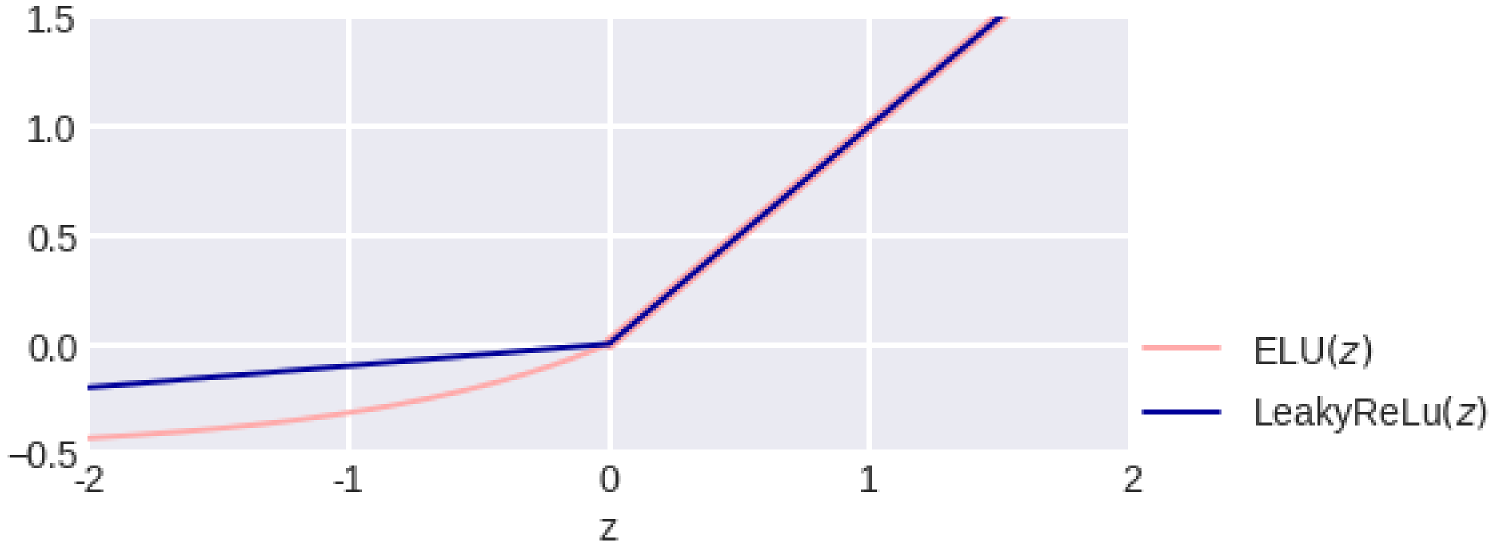
$$\mathrm{LeakyReLU}(z) = \max(0.1z, z)$$

Exponential Linear Unit

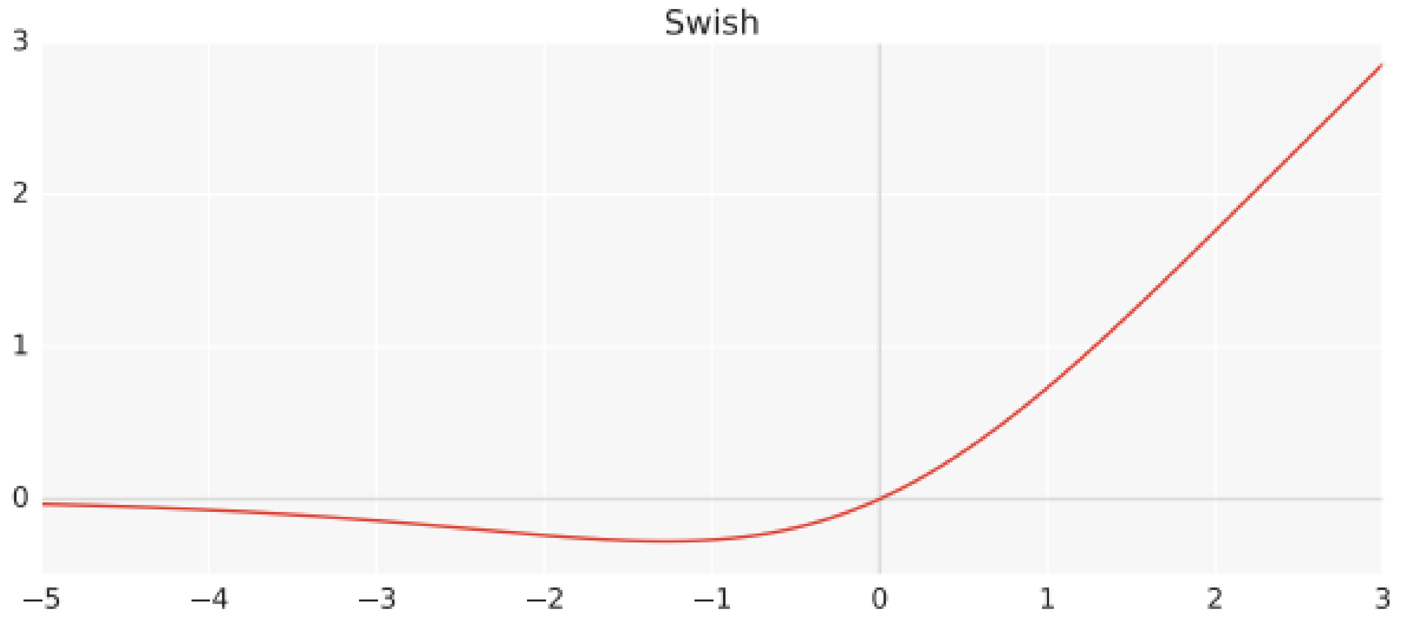
$$\mathrm{ELU}(z) = \begin{cases} z, & z \geq 0, \\ \alpha(e^z - 1), & z < 0. \end{cases}$$

Scaled Exponential Linear Unit

$$\mathrm{SELU}(z) = \lambda\,\mathrm{ELU}(z)$$



Ещё функции активации



Swish

$$\mathrm{swish}(z) = x \cdot \sigma(\alpha x)$$

При замене ReLu на swish в глубоких сетях немного (<1%) улучшается качество

SILU

CLASS `torch.nn.SiLU(inplace=False)` [SOURCE]

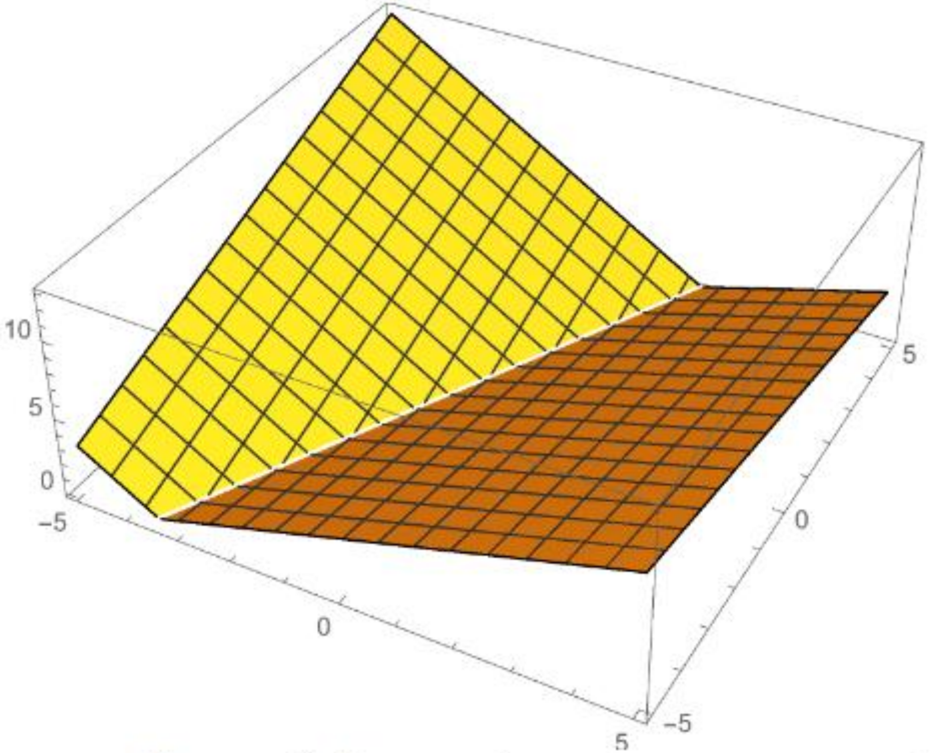
Applies the Sigmoid Linear Unit (SiLU) function, element-wise. The SiLU function is also known as the swish function.

$$\mathrm{silu}(x) = x * \sigma(x), \text{where } \sigma(x) \text{ is the logistic sigmoid.}$$

• NOTE

See Gaussian Error Linear Units (GELUs) where the SiLU (Sigmoid Linear Unit) was originally coined, and see Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning and Swish: a Self-Gated Activation Function where the SiLU was experimented with later.

Не совсем функции активации



$$\max\{x_1 - 0.5x_2 + 1, -2x_1 + x_2 - 2\}$$

Maxout

$$\mathrm{Maxout}(z) = \max(w^{\mathrm{T}} z + w_0, v^{\mathrm{T}} z + v_0)$$

не совсем, т.к. есть параметры

GLU: Gated Linear Unit

$$\mathrm{GLU}(z) = \sigma(w^{\mathrm{T}} z + w_0) \cdot (v^{\mathrm{T}} z + v_0)$$

$$\mathrm{GLU}(z) = \sigma(W^{\mathrm{T}} z + w_0) \cdot (V^{\mathrm{T}} z + v_0)$$

https://raw.githubusercontent.com/epfml/ML_course/master/lectures/08/lecture08d_neural_nets.pdf

Ещё функции активации

Gaussian Error Linear Unit

(Google's BERT, OpenAI's GPT-2)

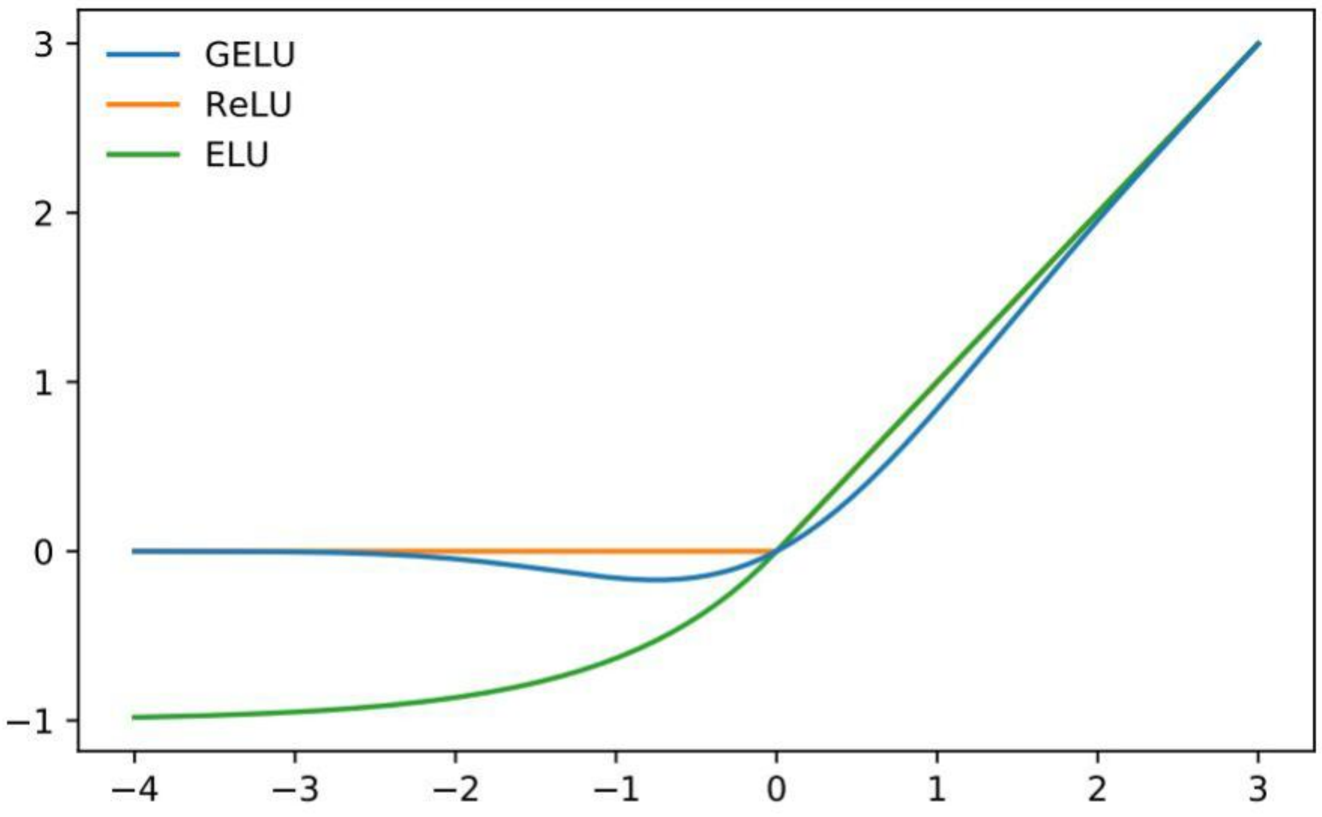
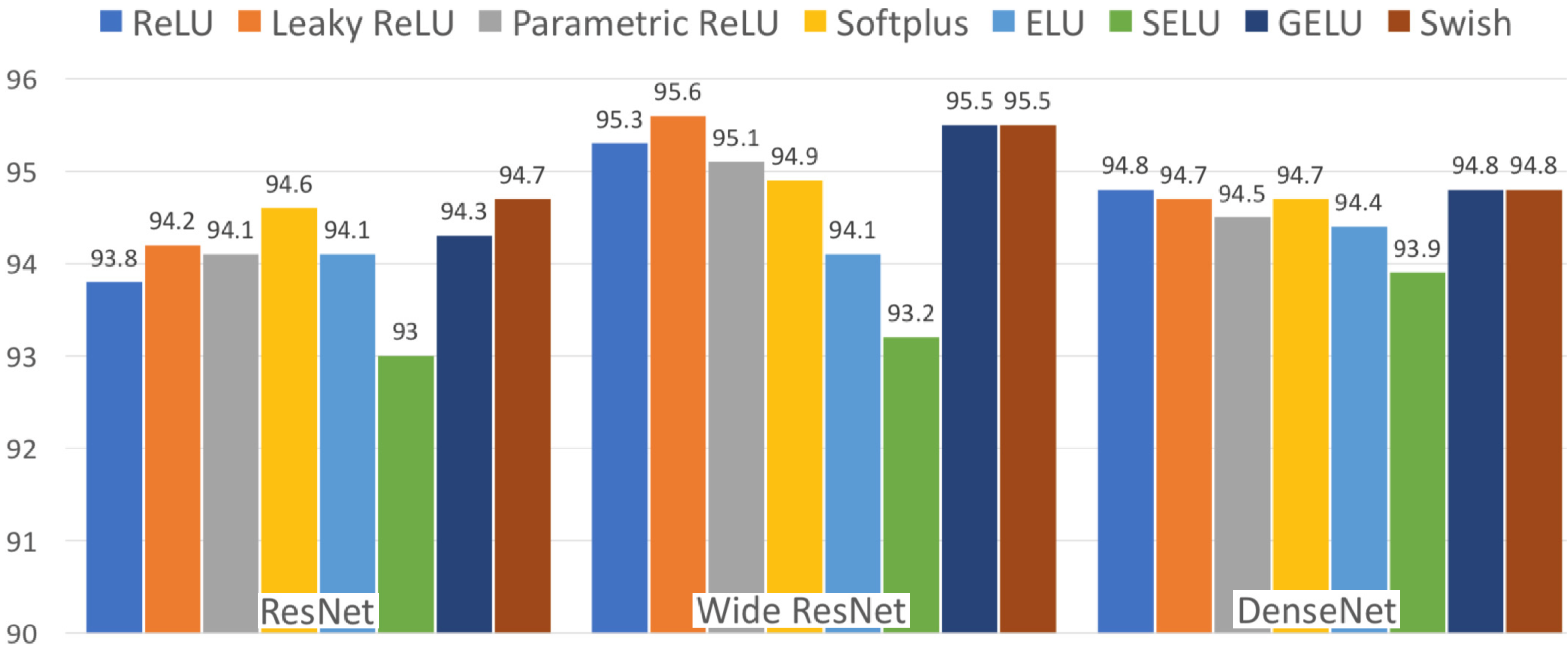


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

$$\mathrm{GELU}(z) = \frac{z}{2}\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(z + \alpha z^3)\right)\right)$$

$$\alpha = 0.044715$$

Сравнение на CIFAR10



[Juhstin Johnson] / https://arxiv.org/pdf/1710.05941.pdf

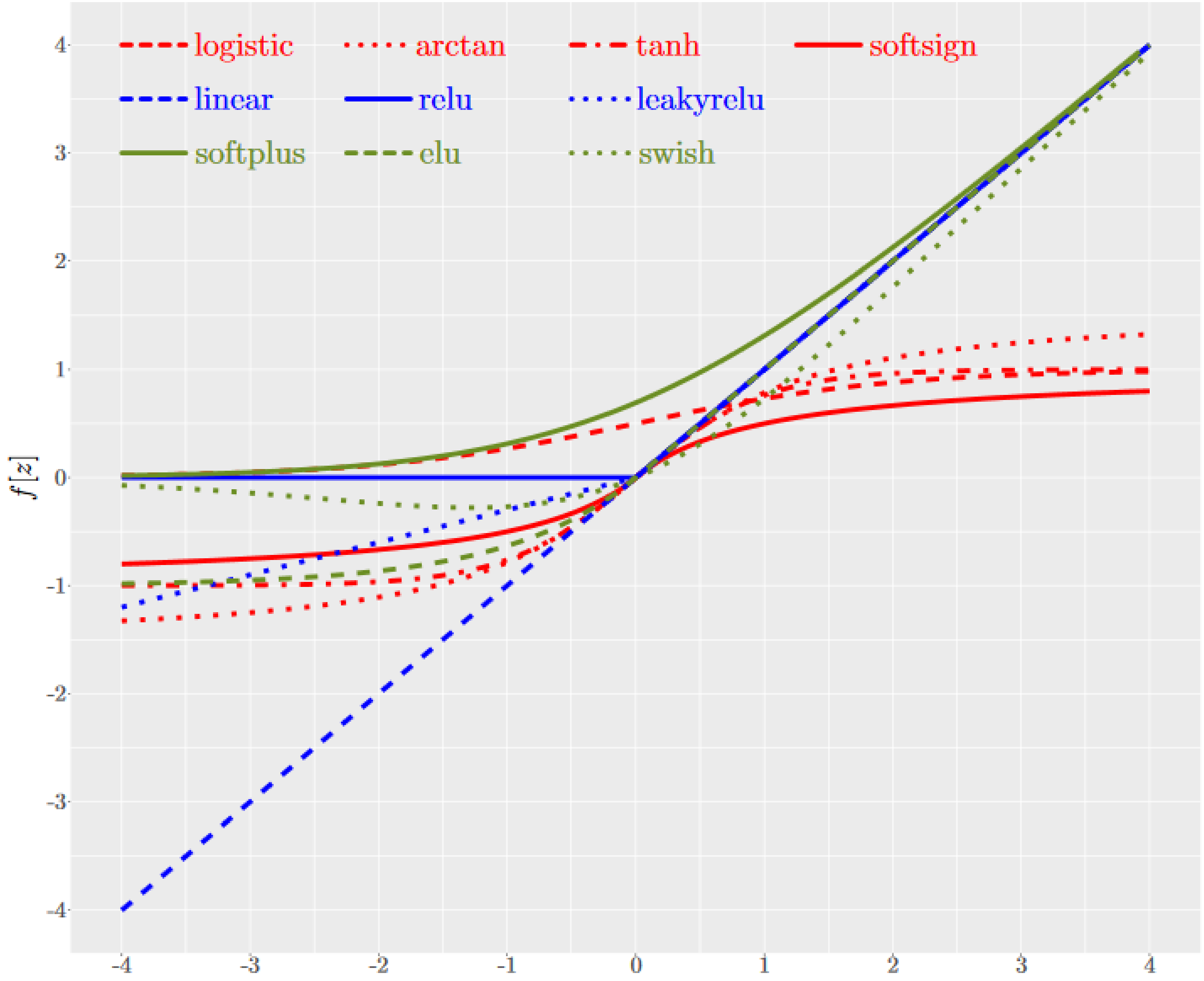
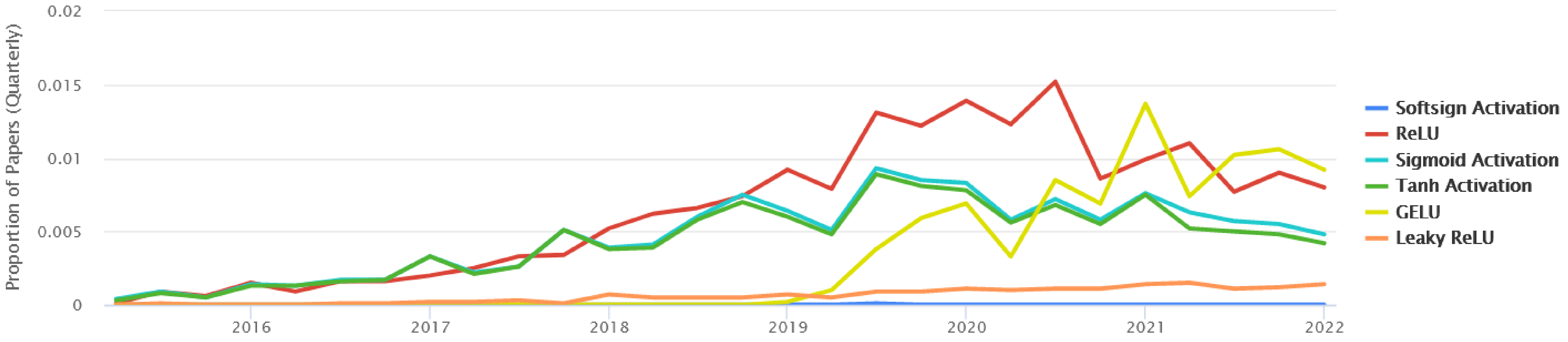


Figure 8: Overview of sigmoid functions (red, see Section 2.1), piecewise-linear functions (blue, see Section 2.2), and other functions (green, see Section 2.3). (Best seen in color.)

https://arxiv.org/pdf/2101.09957v1.pdf

«Популярность» функций активации



https://paperswithcode.com/method/softsign-activation

Минутка кода

```
# рекомендация для своей функции активации
@torch.jit.script
def fused_gelu(x):
    return x * 0.5 * (1.0 + torch.erf(x / 1.41421))

# своя активация в виде модуля
class ActivationFunction(nn.Module):

    def __init__(self):
        super().__init__()
        self.name = self.__class__.__name__
        self.config = {"name": self.name}

class MySigmoid(ActivationFunction):

    def forward(self, x):
        return 1 / (1 + torch.exp(-x))
```

https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial3/Activation_Functions.html

Минутка кода

```
class LeakyReLU(ActivationFunction):

    def __init__(self, alpha=0.1):
        super().__init__()
        self.config["alpha"] = alpha

    def forward(self, x):
        return torch.where(x > 0, x, self.config["alpha"] * x)

def get_grads(act_fn, x):
    # Вычисляем производные
    x = x.clone().requires_grad_() # для этого тензора берём производные
    out = act_fn(x)
    out.sum().backward() # sum()!!!
    return x.grad # градиент

x = torch.tensor([-1., 0., 1.])
f = MySigmoid()
f(x), get_grads(f, x)

(tensor([0.2689, 0.5000, 0.7311]), tensor([0.1966, 0.2500, 0.1966]))
```

Проблемы НС

Большая сложность модели → переобучение

следующая лекция

Нестабильность обучения

по всему курсу

Большое время обучение, много данных

GPU, TPU и т.д.

собирание данных, аугментации и т.п.

эффективные современные алгоритмы настройки

Итог

НС – нелинейное обобщение линейных алгоритмов

• последовательное преобразование признакового пространства

• ансамбль алгоритмов

• суперпозиция «логистических регрессий»

• граф вычислений

высокая функциональная выразимость

обучение градиентными методами

Дальше: как делать эффективное обучение?

Попробуйте поиграться здесь:

http://playground.tensorflow.org/

Литература

Производные на компьютере

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, Jeffrey Mark Siskind «Automatic differentiation in machine learning: a survey» 2015-2018
https://arxiv.org/abs/1502.05767

лучшая книга по DL

Ian Goodfellow, Yoshua Bengio, Aaron Courville «Deep Learning»
http://www.deeplearningbook.org/

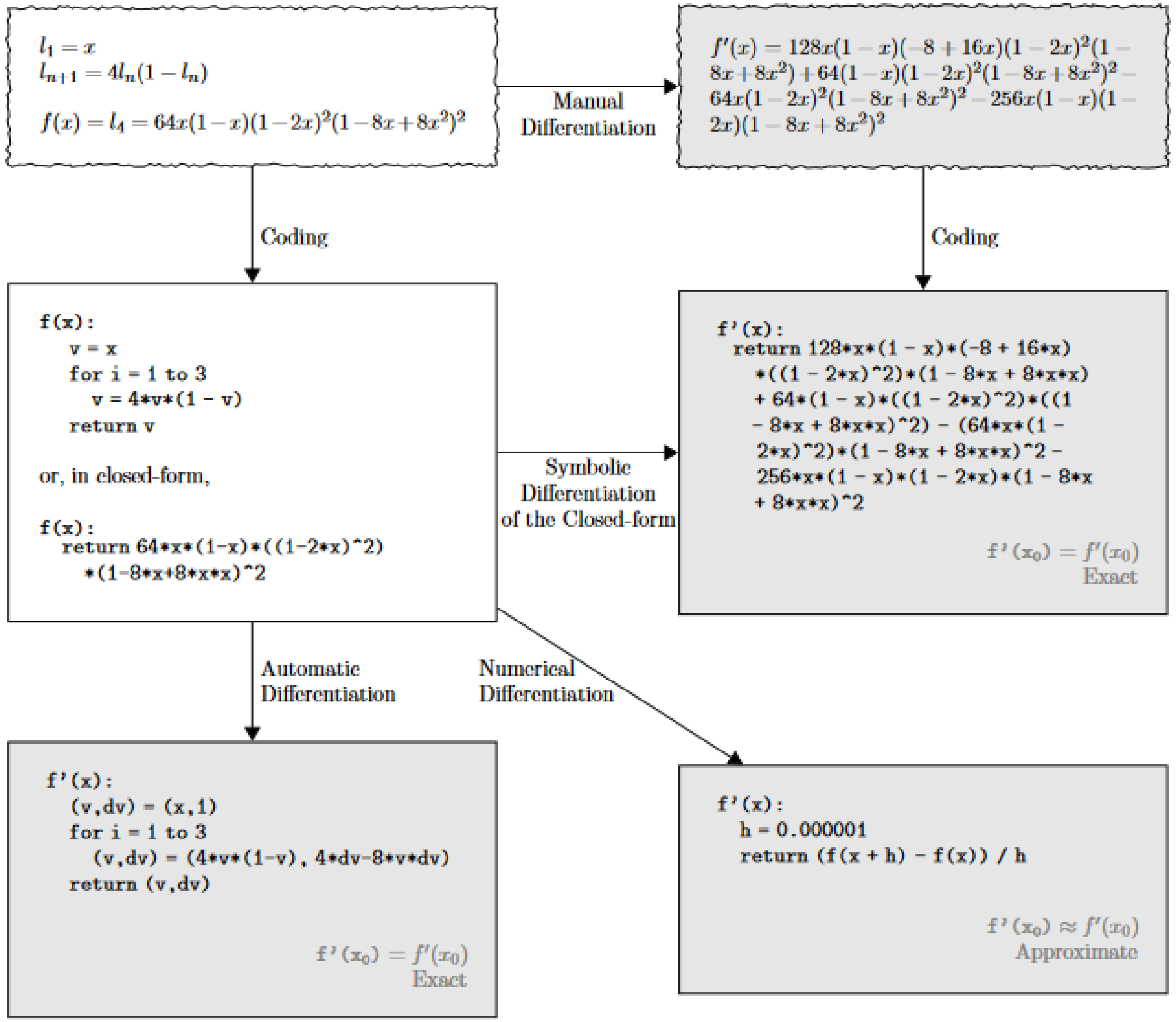


Figure 2: The range of approaches for differentiating mathematical expressions and computer code, looking at the example of a truncated logistic map (upper left). Symbolic differentiation (center right) gives exact results but requires closed-form input and suffers from expression swell; numerical differentiation (lower right) has problems of accuracy due to round-off and truncation errors; automatic differentiation (lower left) is as accurate as symbolic differentiation with only a constant factor of overhead and support for control flow.

Производные на компьютере:

Аналитическая формула

Численный градиент

Символьное дифференцирование

Алгоритмическое дифференцирование