Neural nets 101
and how to code them

This seminar is based on materials from the courses:

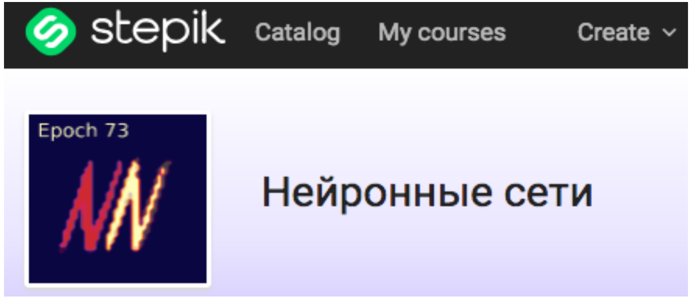
Deep learning and reinforcement learning courses:

- https://github.com/yandexdataschool/Practical_RL
- https://github.com/yandexdataschool/Practical_DL

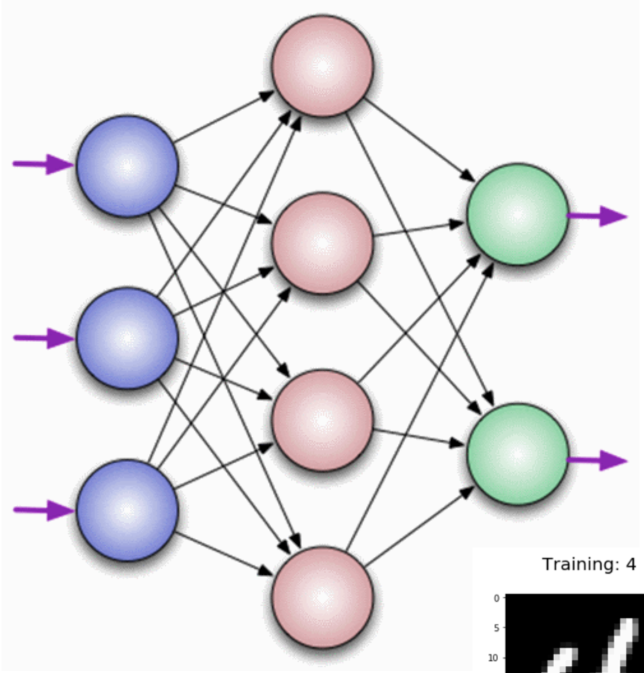
by Alexander Panin

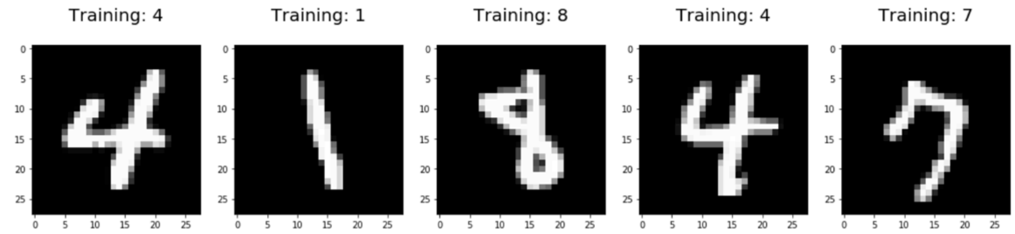
Neural Nets
by Arsenii Moskvichyov

https://stepik.org/course/401

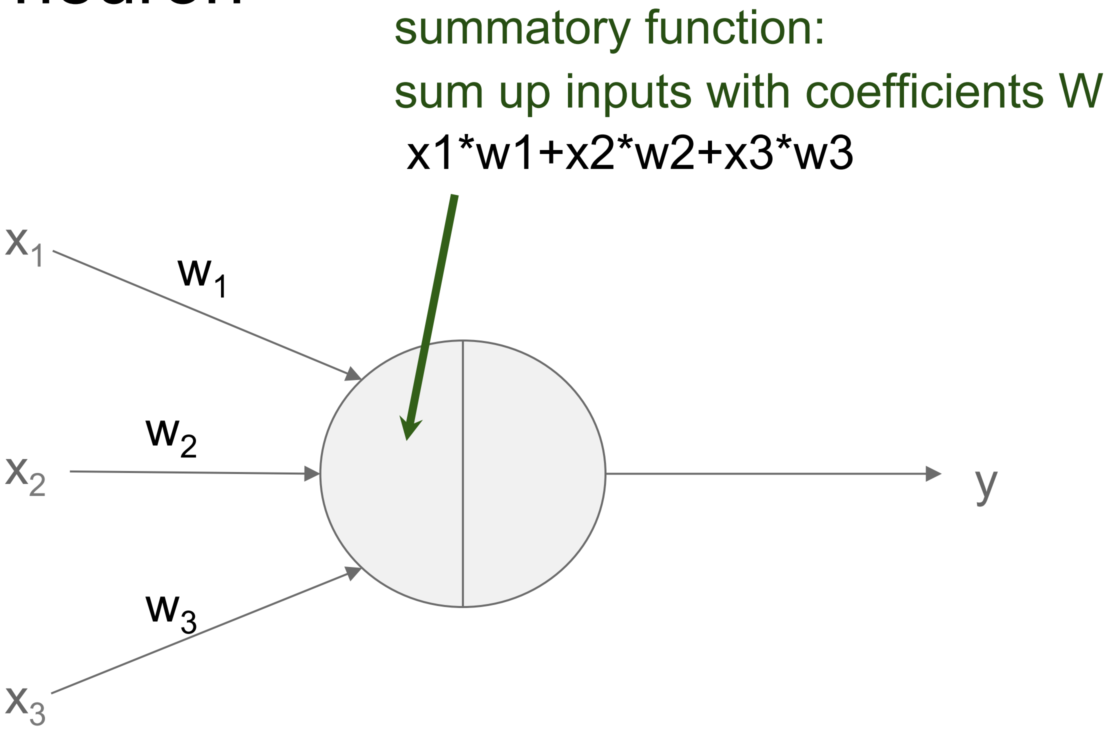


Artificial neural networks





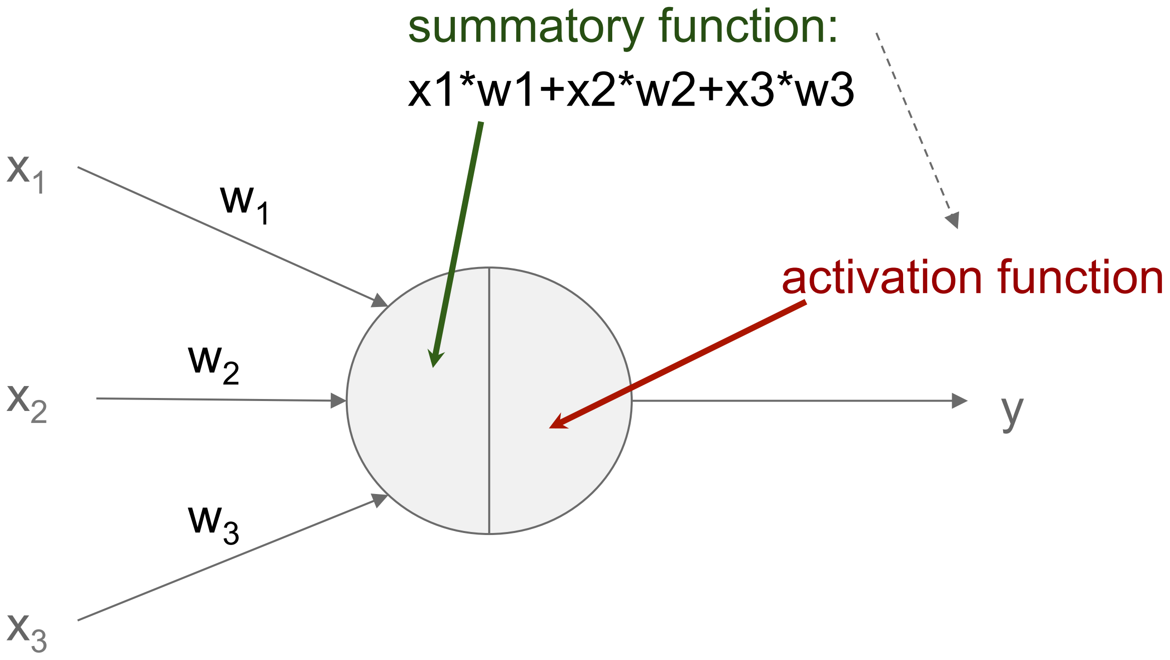
Artificial neuron



Could be rewritten in form of matrix multiplication:
X*W

- x1...xn are inputs (information from outside or activations from other neurons)
- larger weight reflects that this input is more important

Artificial neuron



output of a neuron y = f_activation(X * W)

Linear activation function

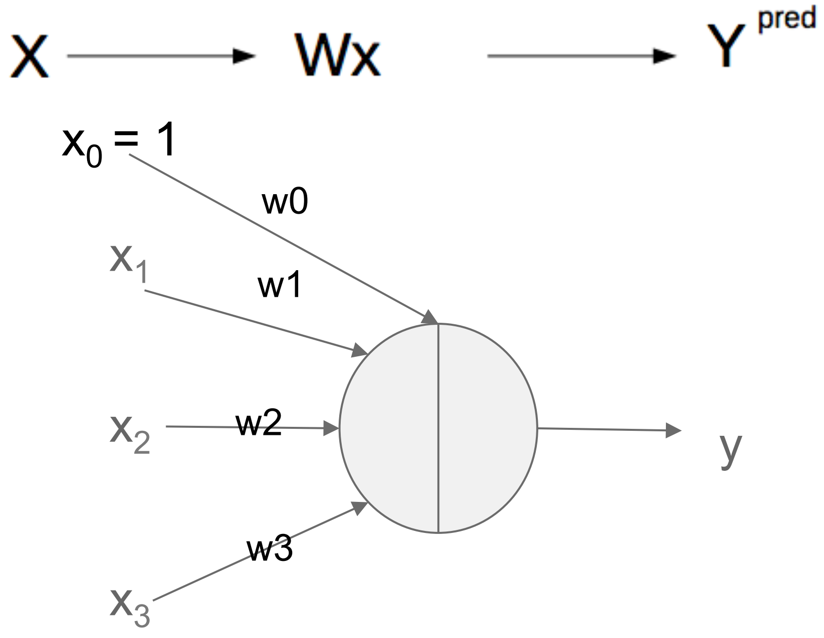
How will the output look like for this case?

y = x1*w1 + x2*w2 + x3*w3

What is it remind you?

Actually, to get a multiple linear regression we need a bias. Let's add it as dummy input

So we can keep it as matrix multiplication X*W. Or you should add bias explicitly (X*W + b)



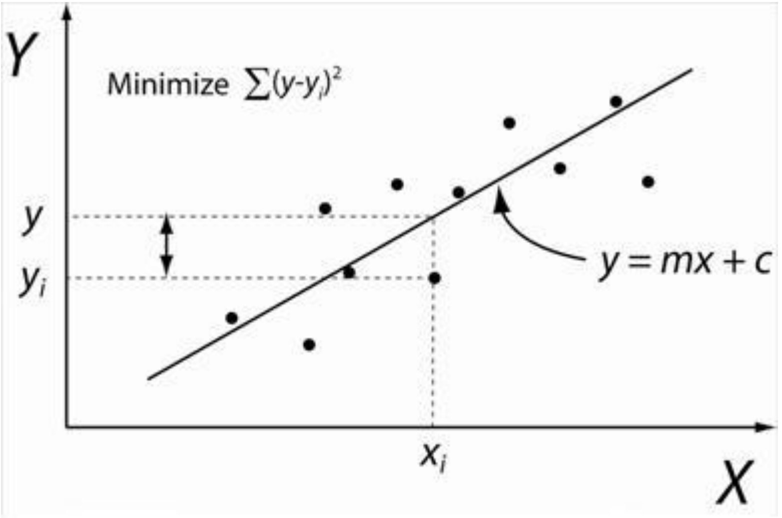
Linear neuron

$$X \longrightarrow Wx + b \longrightarrow Y^{pred}$$

For a one single linear neuron we can get optimal weights by hands...

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{3} x_i(\bar{y} - y_i)}{\sum_{i=1}^{3} x_i(\bar{x} - x_i)}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

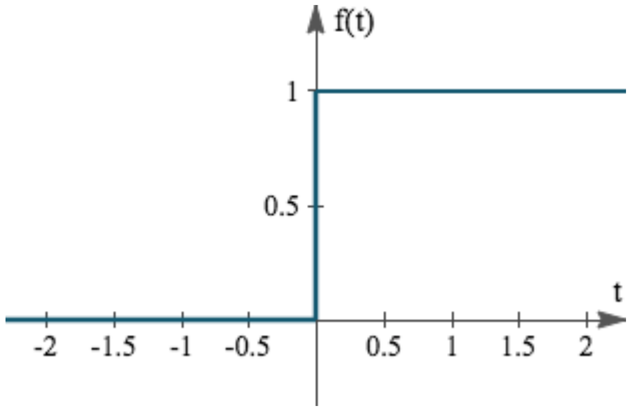


For multiple regression in matrix form:
$(X^T X)^{-1} X^T Y$

Perceptron

A different type of artificial neuron. It has different activation function:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases}$$



Perceptron as a decision-maker

Will I visit grandma?

Let’s say that input vector consists of 4 values:

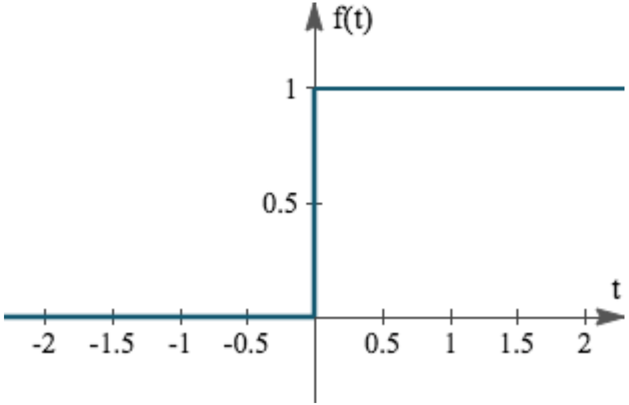
$$\begin{pmatrix} \text{bias: how much would you like to go to grandma in general} \\ \text{whether your friends have invited you for hangout} \\ \text{whether grandma have promised you to cook your favourite pie} \\ \text{whether you are on diet} \end{pmatrix}$$

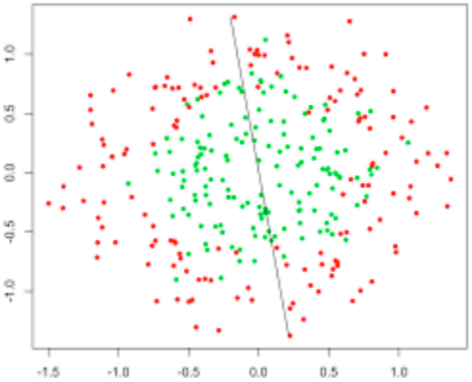
A. w=(−5, −1, 10, 0)
B. w=(4, −1, 1, 0)
C. w=(0, −10, 5, 2)
D. w=(2, −1, 15, −20)

1. a blogger, who pictures idealistic life
2. John, 6 y.o.
3. a mercantile pie lover
4. a party person

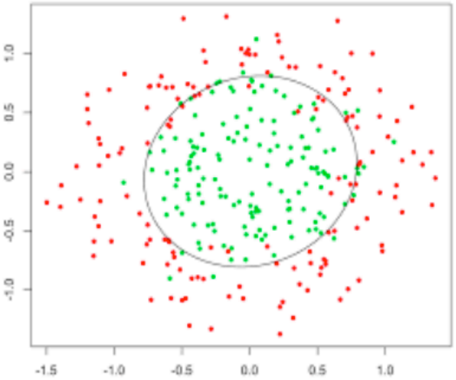
the task is taken from https://stepik.org/course/401

We need more types of neurons





What we have



What we want

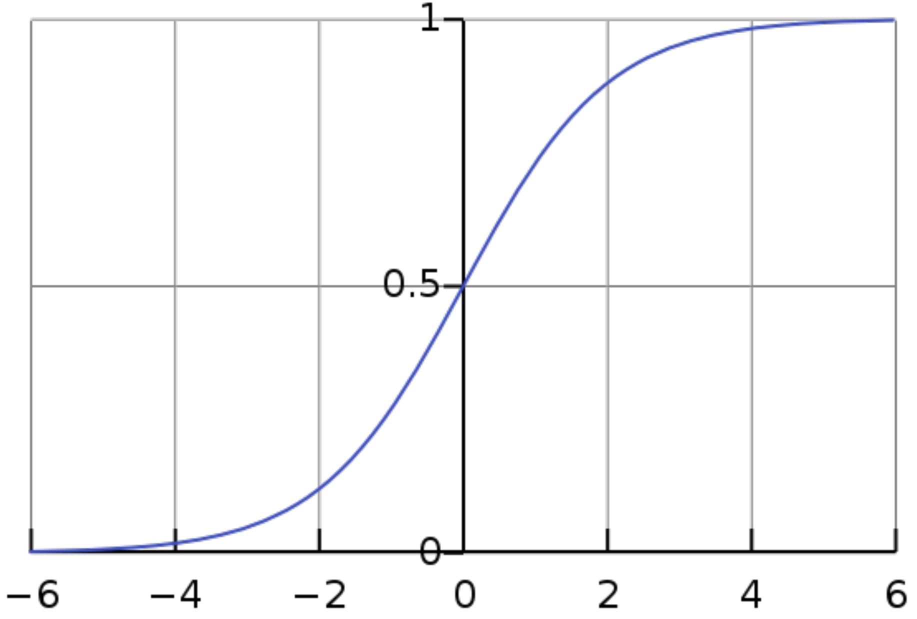
17

- How to get that?

The right picture was taken from https://github.com/yandexdataschool/Practical_DL

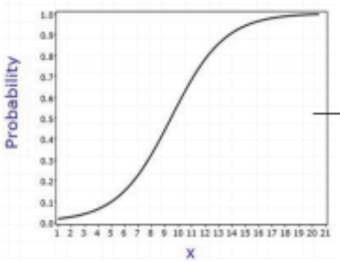
Sigmoid function

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



Sigmoid neuron

$X \longrightarrow Wx + b \longrightarrow$

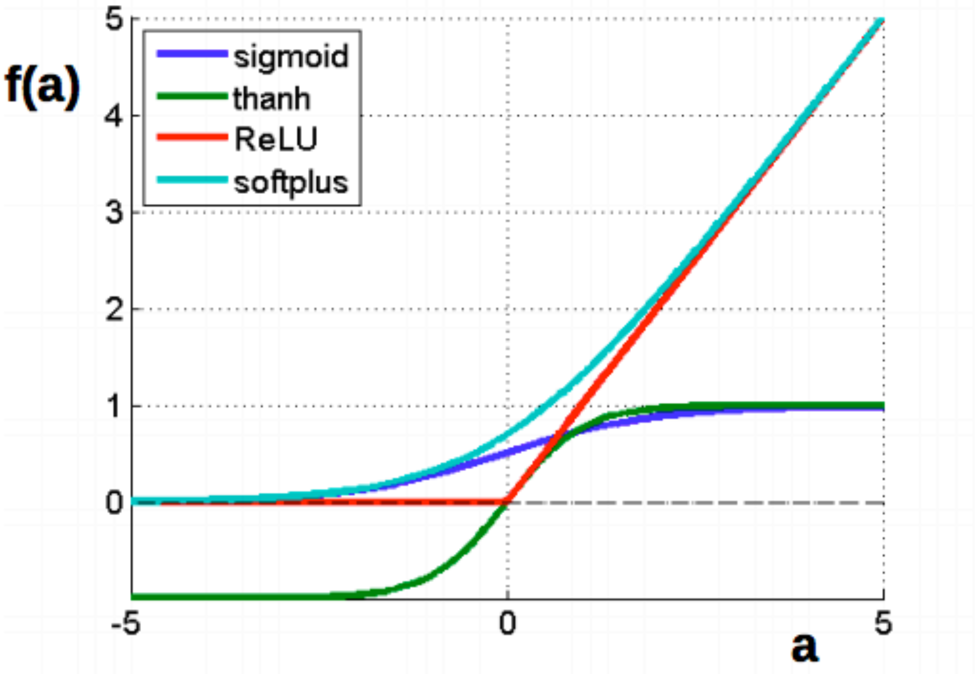


$\longrightarrow P(y)$

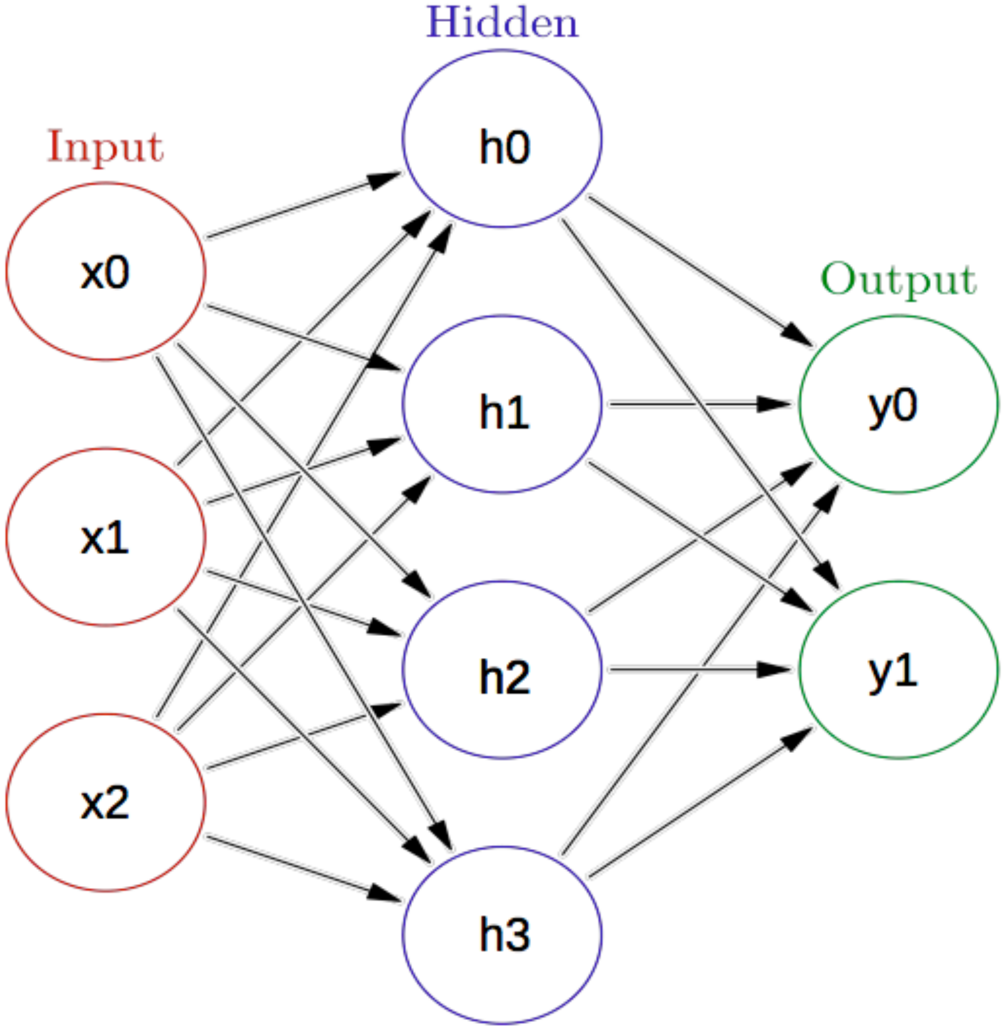
$$P(y) = \sigma(Wx + b)$$

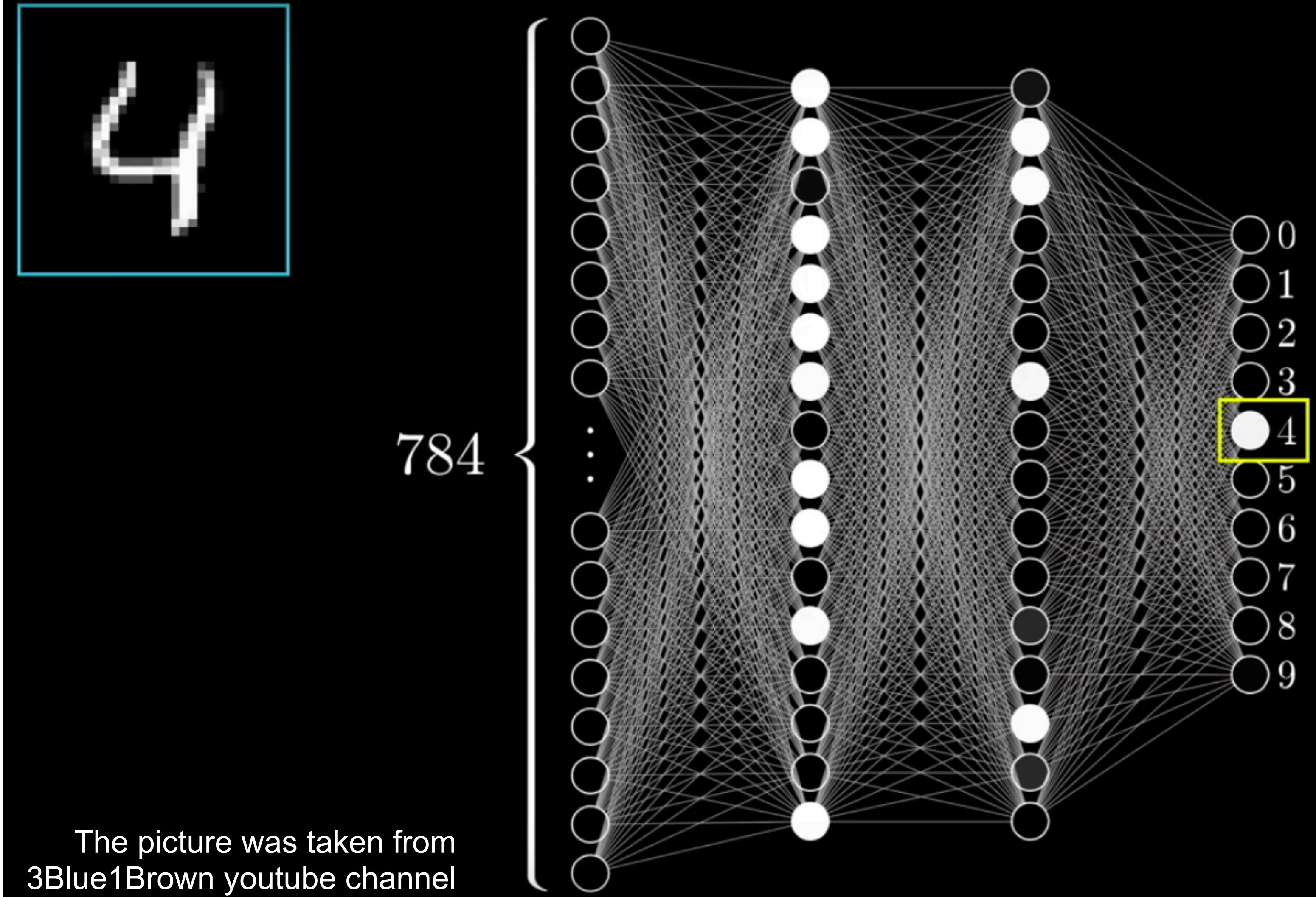
And more...

- f(a) = 1/(1+e^a)
- f(a) = tanh(a)
- f(a) = max(0,a)
- f(a) = log(1+e^a)



The slide was taken from https://github.com/yandexdataschool/Practical_DL


Hidden
Input
Output
x0
x1
x2
h0
h1
h2
h3
y0
y1

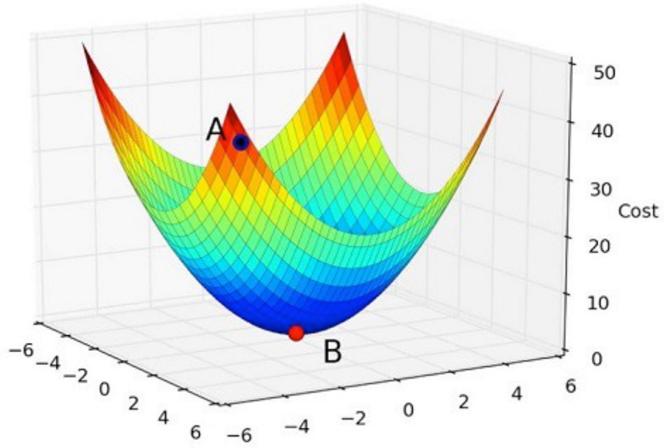


The picture was taken from
3Blue1Brown youtube channel

Let’s return to linear neuron. And train it.

1. Let’s initialize weights randomly

2. Feed forward (get prediction) : X*W = y_predi



3. How much it differ from real y?

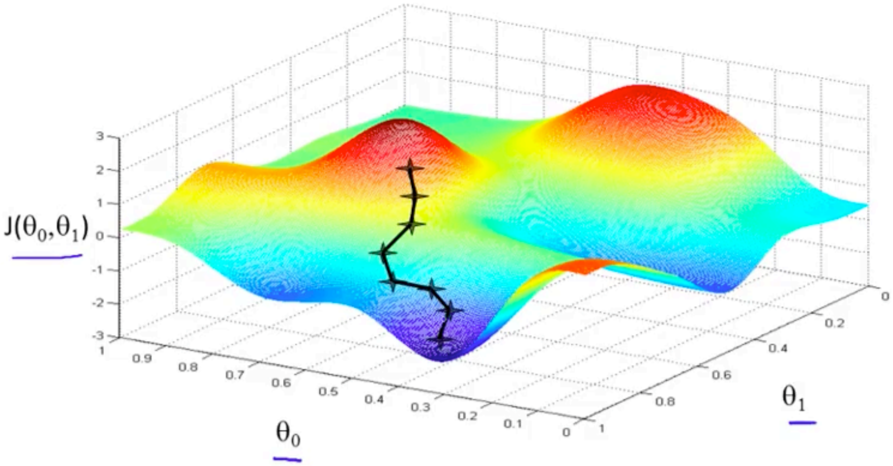
J = sum ((y_predicted(i) - y(i))^2) - a “loss function”
J = sum ((X*W - Y)^2)

Gradient is a vector, which consists form partial derivatives:

$$\nabla J = \begin{bmatrix} \frac{\partial J}{\partial w_1} \\ \dots \\ \frac{\partial J}{\partial w_m} \end{bmatrix}$$

Learning algorithm:

w_new = w - α ∇J



Back propagation

J = sum ((y_predicted(i) - y(i))^2) = sum ((X*W - Y)^2)

To get gradient, we need to take derivative dJ/dw,

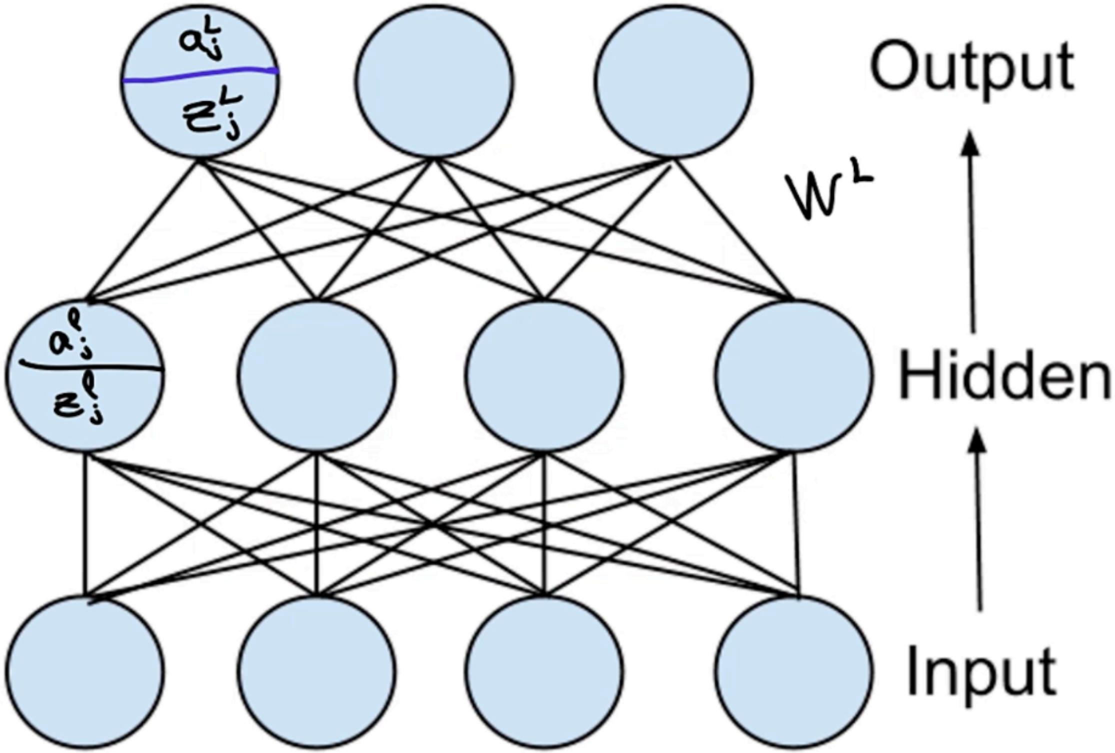
which is a derivative of the composite function, so, use chain rule:

∂J/∂w = ∂J/∂y_predicted * ∂y_predicted/∂ativation_function * ∂ativation_function/∂ W

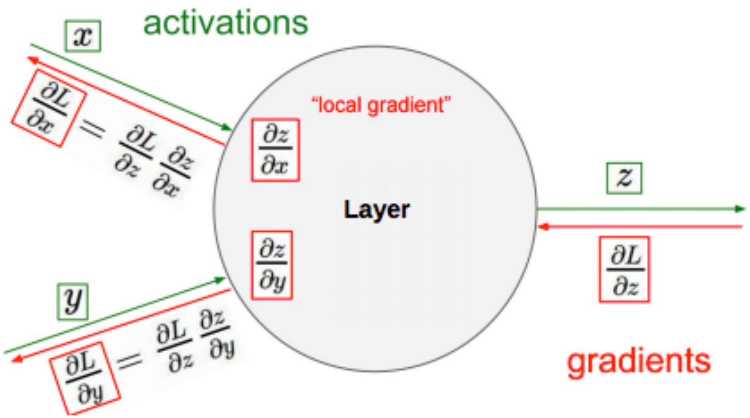
backprop = chain rule*

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

Back propagation

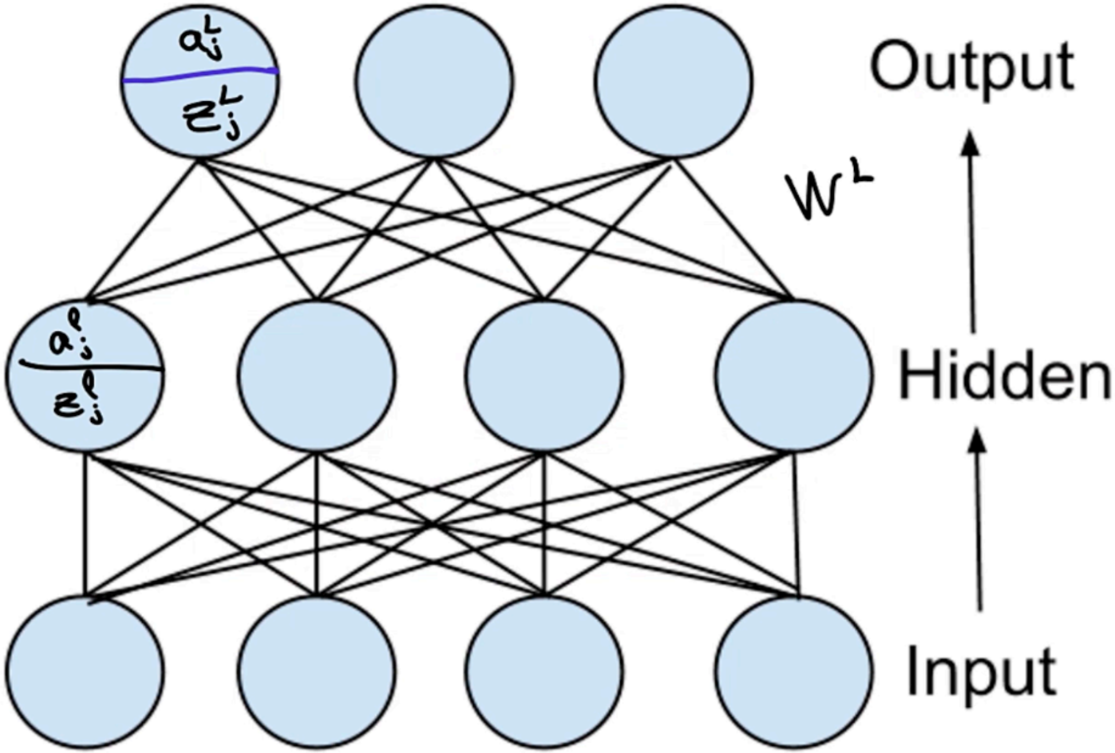


The slide was take from https://stepik.org/course/401



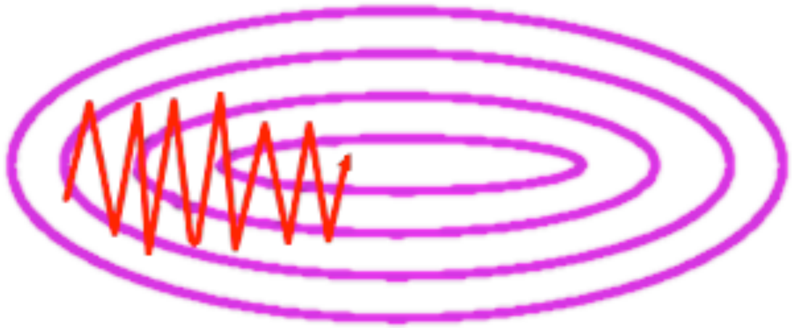
The slide was taken from https://github.com/yandexdataschool/Practical_DL

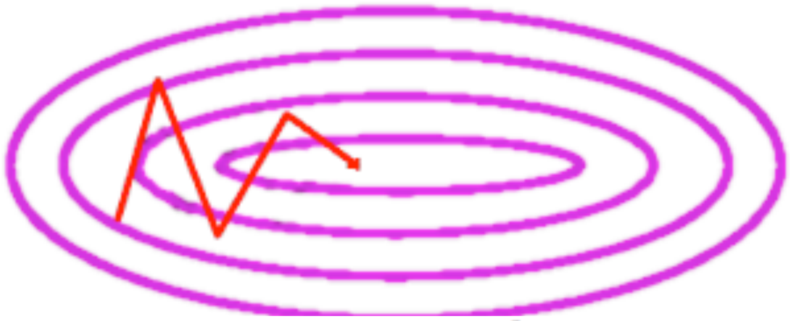
Back propagation



Different gradient descent algorithms

Let’s use momentum to train faster.
Idea: move towards “overall gradient direction”, not just current gradient.





$$v_{i+1} \leftarrow \alpha \frac{\partial L}{\partial w} + \mu v_i$$

$$w_{i+1} \leftarrow w_i - v_{i+1}$$

The slide was taken from https://github.com/yandexdataschool/Practical_DL

Loss functions

- MSE:

sum((y_predicted - y)^2) / y.size

- Cross-entropy:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

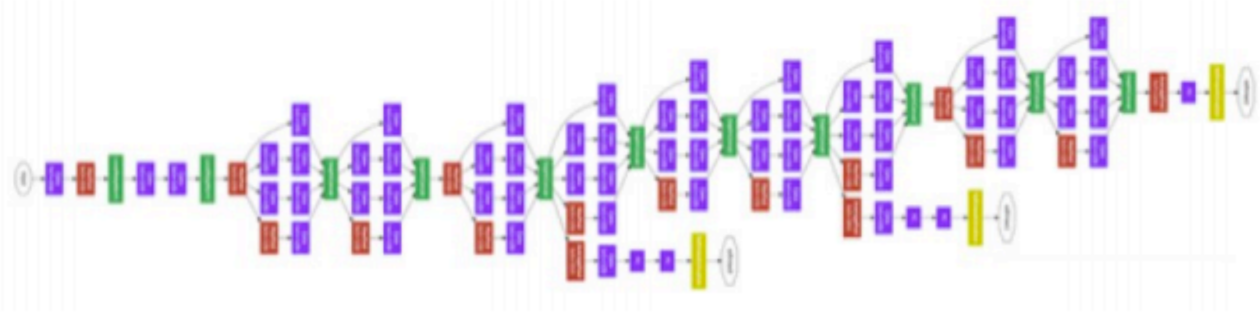
Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

and more here http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

PyTorch and other frameworks

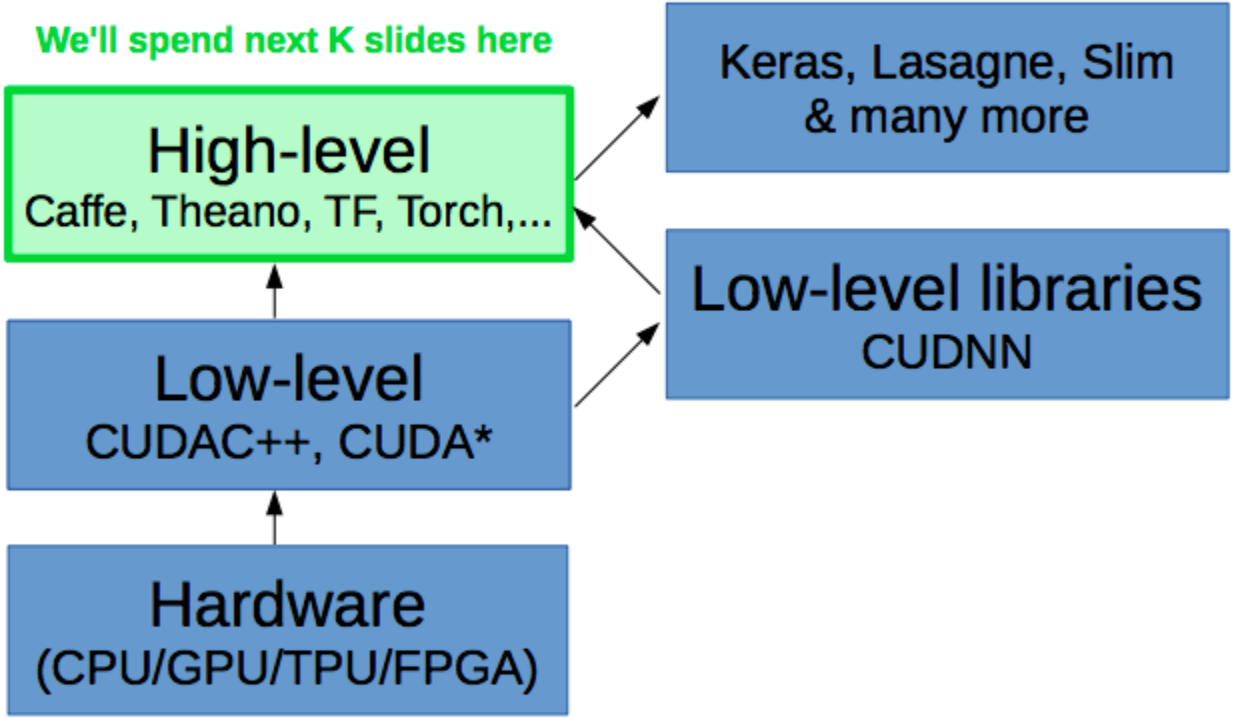
And now let's differentiate



- 5+ types of layers
- each with different dimensions
- parallel branches with independent losses
- several nonlinearities

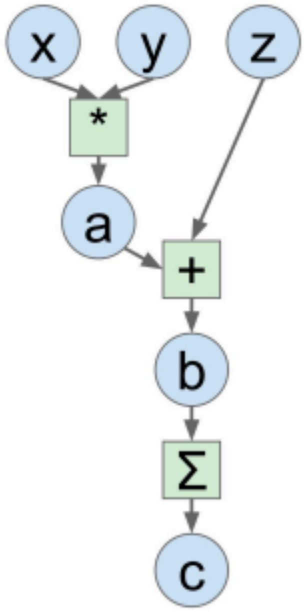
The slide was taken from https://github.com/yandexdataschool/Practical_DL

Deep learning frameworks



The slide was taken from https://github.com/yandexdataschool/Practical_DL

Symbolic graphs

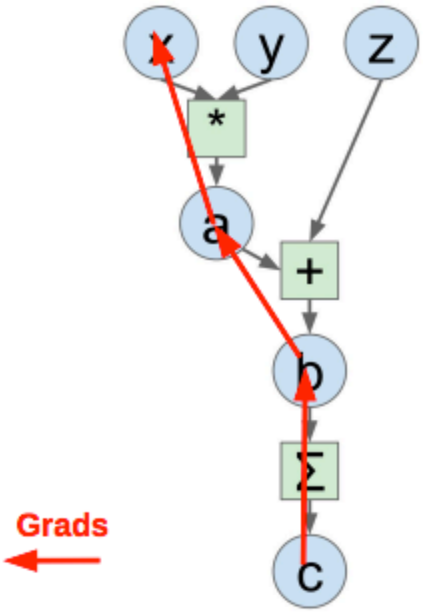



```
a = x * y
b = a + z
c = np.sum(b)
```

Idea: let's define this graph explicitly!

This many further slides taken from Ars Ashukha @deepbayes2017

Symbolic graphs




```
a = x * y
b = a + z
c = np.sum(b)
```

+ Automatic gradients!
+ Easy to build new layers
+ We can optimize the Graph
- Graph is static during training
- Need time to compile/optimize
- Hard to debug

This many further slides taken from Ars Ashukha @deepbayes2017

Dynamic graphs

Chainer, DyNet, Pytorch

+ Can change graph on the fly
+ Can get value of any tensor at any time (easy debugging)
- Hard to optimize graphs (especially large graphs)
- Still early development

Researchers love them!

The slide was taken from https://github.com/yandexdataschool/Practical_DL