

Lecture 5. Neural networks intro, backpropagation method

Alex Avdyushenko

Kazakh-British Technical University

October 8, 2022



Five-minute questions

- What is the name of the course?
- What is the teacher's name?
- What color is the textbook?

To make long story short



2003, silver
medal

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short



2003, silver
medal

I was born and grown in faraway snowy Novosibirsk in a beautiful Soviet family: mother doctor, father engineer, older brother. Childhood in the 90s was fun: garages and new buildings as playground, the sea and the forest instilled a love of sports and tourism. In school, about the 5th grade I tried mathematics, already in the 7th I won the regional olympiad, it was almost miracle. Away we go — a physics and mathematics school in the legendary Akademgorodok, close to Obges (microdistrict)...

To make long story short



2009, master
of math



2003, silver
medal

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short

2009, master
of math2003, silver
medal

math at Novosibirsk State University

To make long story short



2009, master
of math



2014, phd

2003, silver
medal

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short

2009, master
of math2003, silver
medal

2014, phd

then science.. "You could not be a scientist, but a candidate must be!"

To make long story short



2009, master
of math



2016, YDS

2014, phd

2003, silver
medal

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short



In the graduate school, I wanted to try something else and the same time the hype about machine learning was growing, Yandex wanted to come to Novosib with the development office and, of course, School of data analysis (SHAD, YDS). This chance could not be missed and I was fortunate to simultaneously become a curator and student of the SHAD branch. It was really difficult: I took academic leave for the first time in my life! Before my graduation in the fourth semester, we had a daughter in our family, but I liked it :)

To make long story short



2009, master
of math



2003, silver
medal



2014, phd



2016, YDS



2015-2018,
analyst

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short

2009, master
of math2003, silver
medal

2014, phd



2016, YDS



2015-2018.



I already worked as an analyst at Yandex for the graduation from SHAD. It was super-useful for about three years of experience in industrial programming and analytics, but I became increasingly aware that I was drawn back to teaching, science and organizing education.

To make long story short



2009, master
of math



2003, silver
medal



2014, phd



2016, YDS



2015-2018,
analyst



2016-2022,
ed manager

To make long story short



2009, master
of math



2003, silver
medal



2014, phd



2016, YDS



2015-2018,
analyst



2016-2022,
ed manager



2019+,
docent

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short

To make long story short



2009, master
of math



2003, silver
medal



2014, phd



2016, YDS



2015-2018,
analyst



CSC
2016-2022,
ad manager



2019+,
docent

Then I moved to St.Petersburg to the new department of math and computer science at SPbU. We have such an educational startup within the university — a very strong team led by Stas Smirnov, a winner of the Fields Prize. Support from Yandex, VK, Gazprom Neft and Huawei, many ambitious tasks and one big goal to create «MIT in Russia».

To make long story short



2009, master
of math



2003, silver
medal



2014, phd



2016, YDS



2015-2018,
analyst



2016-2022,
ed manager



2022+, head
of ML ed



2019+,
docent

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short



Now I am developing data analysis and machine learning education at Yandex and partner universities

To make long story short



2009, master
of math



2003, silver
medal



2014, phd



2016, YDS



2015-2018,
analyst



2022+,
instructor



2022+, head
of ML ed



2019+,
docent

Lecture 5. Neural networks intro, backpropagation method

└ To make long story short



I am very glad to join KBTU university and I am ready to help, communicate, discuss machine learning ideas

Tasks solved by neural networks

Images: 25% → 3.5%

VS 5% errors by human

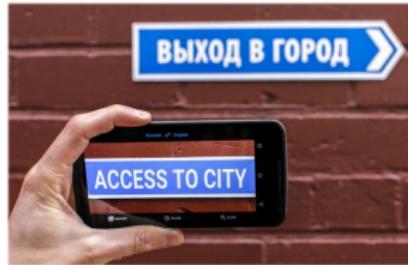


Tasks solved by neural networks

Images: 25% → 3.5%
VS 5% errors by human



Texts



Tasks solved by neural networks

Images: 25% → 3.5%
VS 5% errors by human



Texts



Voice



Siri



Google Assistant



Hey Cortana



amazon
alexa

Tasks solved by neural networks

Images: 25% → 3.5%
VS 5% errors by human



Texts



Voice



Siri



Google Assistant



Hey Cortana



amazon
alexa

Game Go, 2016



Tasks solved by neural networks

Images: 25% → 3.5%
VS 5% errors by human



Texts



Voice



Siri



Google Assistant



Hey Cortana



amazon
alexa

Game Go, 2016



StarCraft, 2019



Tasks solved by neural networks

Images: 25% → 3.5%
VS 5% errors by human



Texts



Voice



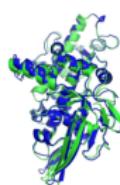
Game Go, 2016



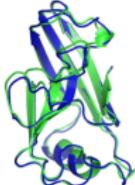
StarCraft, 2019



Protein structure, 2022

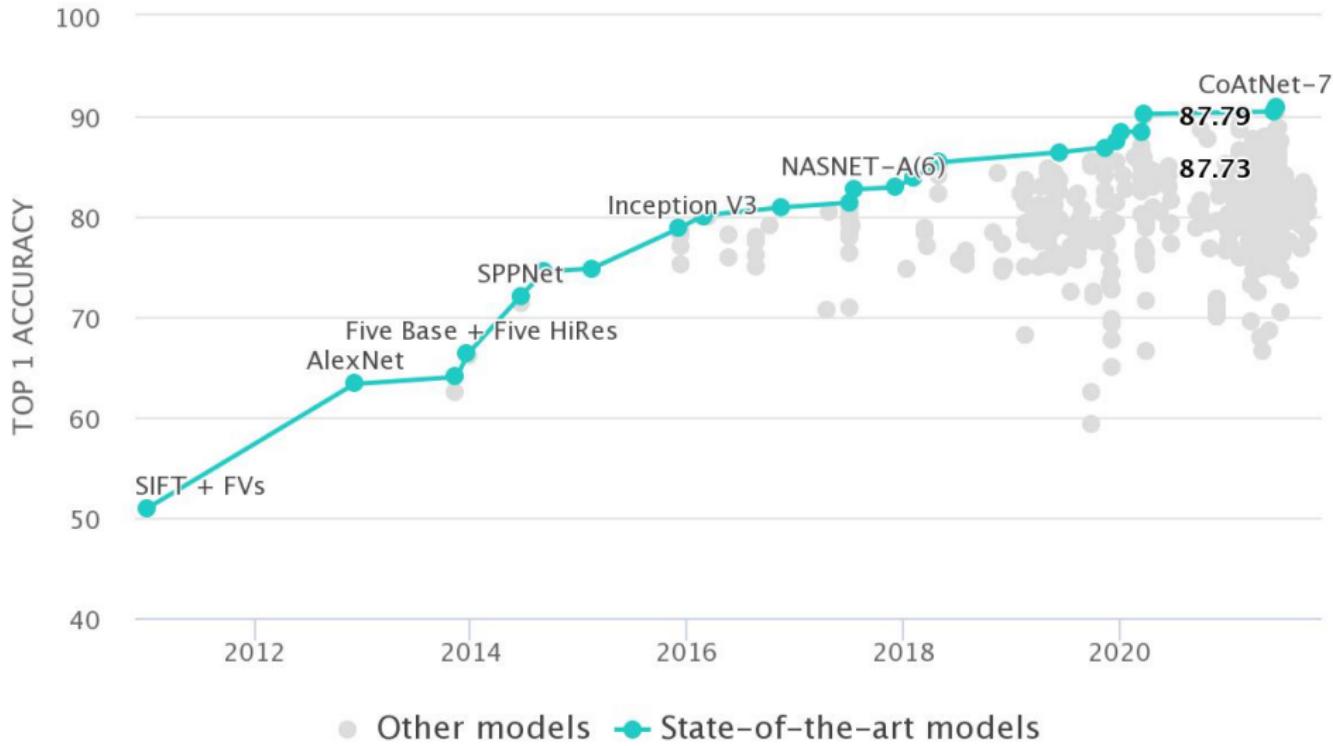


T1037 / 6vr4
90.7 GDT
(RNA polymerase domain)



T1049 / 6y4f
93.3 GDT
(adhesin tip)

- Experimental result
- Computational prediction



<https://paperswithcode.com/sota/image-classification-on-imagenet>

Linear model

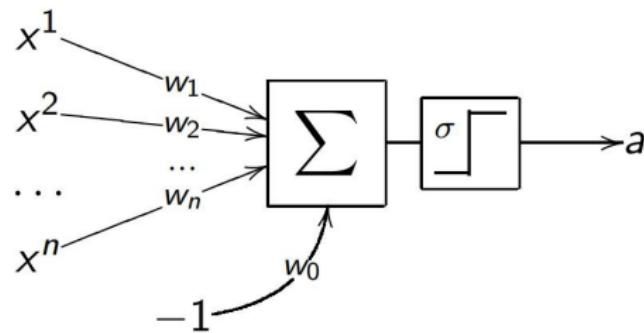
recall

$x^1, x^2, \dots, x^n \in \mathbb{R}$ — numerical features of one object x

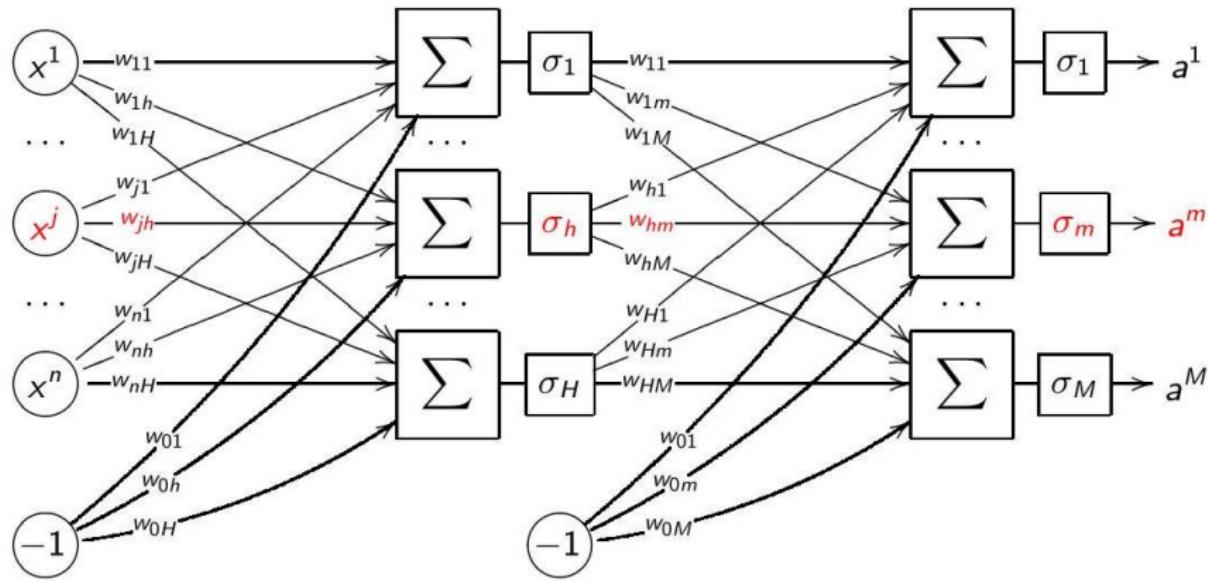
$w_0, w_1, \dots, w_n \in \mathbb{R}$ — weights of features

$$a(x, w) = \sigma(\langle w, x \rangle) = \sigma \left(\sum_{j=1}^n w_j f_j(x) - w_0 \right),$$

$\sigma(z)$ — activation function, for example one of the: $\text{sign}(z)$, $\frac{1}{1+e^{-z}}$, $(z)_+$



Neural network as combination of linear models



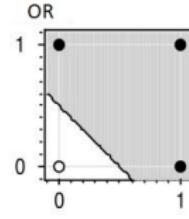
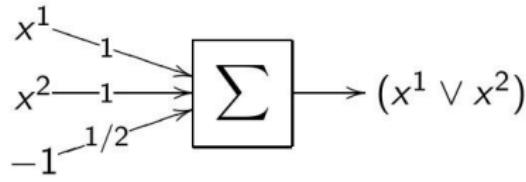
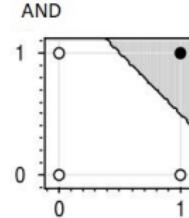
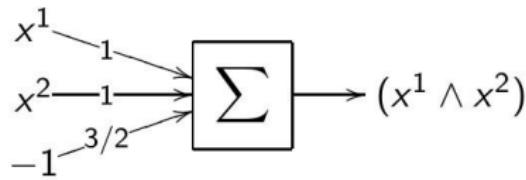
Neural implementation of logic functions

Functions AND, OR, NOT from binary variables x^1 and x^2 :

$$x^1 \wedge x^2 = [x^1 + x^2 - \frac{3}{2} > 0]$$

$$x^1 \vee x^2 = [x^1 + x^2 - \frac{1}{2} > 0]$$

$$\neg x^1 = [-x^1 + \frac{1}{2} > 0]$$



Logic function XOR

Function $x^1 \oplus x^2 = [x^1 \neq x^2]$

Is not implementable by one neuron

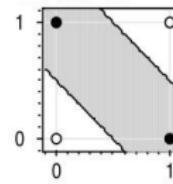
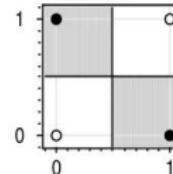
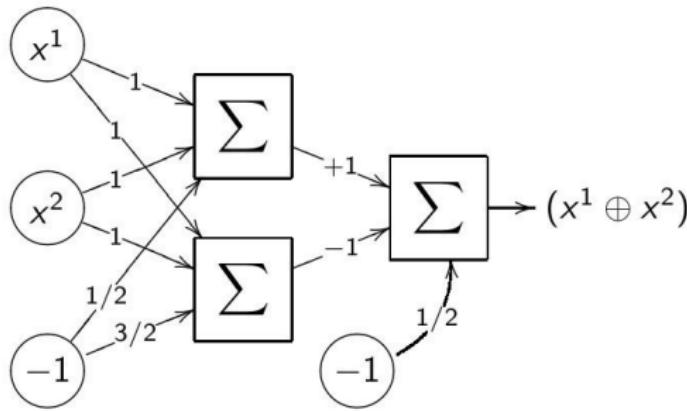
Logic function XOR

Function $x^1 \oplus x^2 = [x^1 \neq x^2]$

Is not implementable by one neuron

There are two ways to implement:

- By adding a non-linear feature $x^1 \oplus x^2 = [x^1 + x^2 - 2x^1x^2 - \frac{1}{2} > 0]$
- With a network (two-layer superposition) of AND, OR, NOT functions: $x^1 \oplus x^2 = [(x^1 \vee x^2) - (x^1 \wedge x^2) - \frac{1}{2} > 0]$.



Expressive power of a neural network

- A two-layer network in $\{0, 1\}^n$ allows us to implement an arbitrary Boolean function

Expressive power of a neural network

- A two-layer network in $\{0, 1\}^n$ allows us to implement an arbitrary Boolean function
- A two-layer network in \mathbb{R}^n allows us to separate an arbitrary convex polyhedron

Expressive power of a neural network

- A two-layer network in $\{0, 1\}^n$ allows us to implement an arbitrary Boolean function
- A two-layer network in \mathbb{R}^n allows us to separate an arbitrary convex polyhedron
- A three-layer network in \mathbb{R}^n allows you to separate an arbitrary polyhedral region (may not be convex or even connected)

Expressive power of a neural network

- A two-layer network in $\{0, 1\}^n$ allows us to implement an arbitrary Boolean function
- A two-layer network in \mathbb{R}^n allows us to separate an arbitrary convex polyhedron
- A three-layer network in \mathbb{R}^n allows you to separate an arbitrary polyhedral region (may not be convex or even connected)
- With the help of linear operations and one non-linear activation function σ any continuous function can be approximated with any desired accuracy

Expressive power of a neural network

- A two-layer network in $\{0, 1\}^n$ allows us to implement an arbitrary Boolean function
- A two-layer network in \mathbb{R}^n allows us to separate an arbitrary convex polyhedron
- A three-layer network in \mathbb{R}^n allows you to separate an arbitrary polyhedral region (may not be convex or even connected)
- With the help of linear operations and one non-linear activation function σ any continuous function can be approximated with any desired accuracy
- For some special classes of deep neural networks, they have been proven to have exponentially greater expressive power than shallow networks.

V. Khrulkov, A. Novikov, I. Oseledets. Expressive power of recurrent neural networks, Feb 2018, ICLR

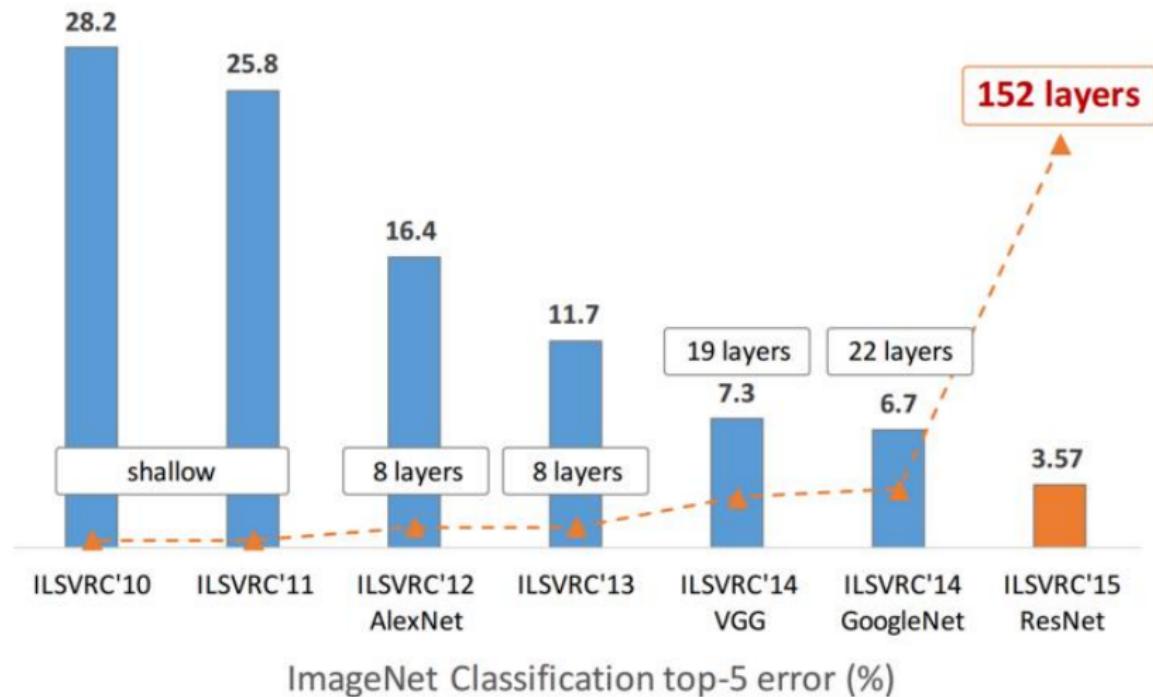
Annotated Image Dataset Project

- 14M+ images
- 20K+ classes

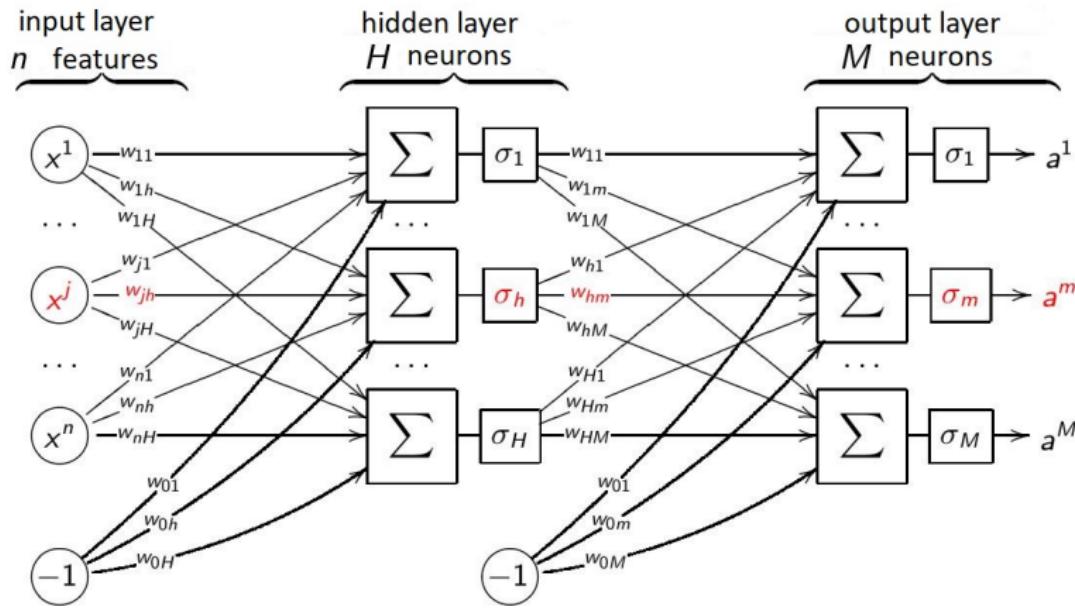


The development of convolutional networks

Or a brief history of ImageNet



Two-layer neural network with M-dimensional output



Model parameter vector $w \equiv (w_{jh}, w_{hm}) \in \mathbb{R}^{Hn+H+MH+M}$

Question 1

Where do these amounts of parameters come from?

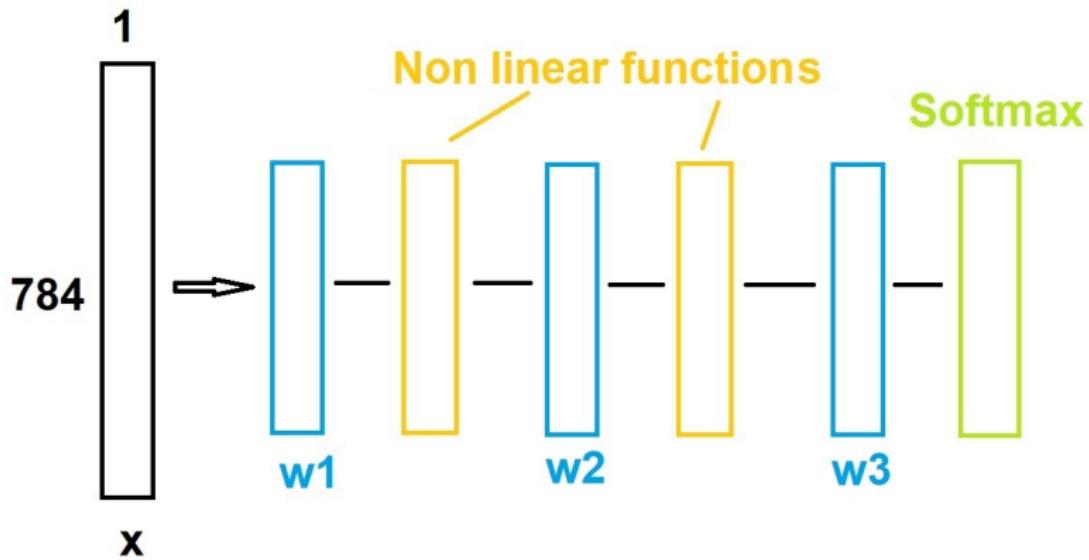
MNIST in PyTorch

Demo

Notebook at [github](#)

Source: <https://nextjournal.com/gkoehler/pytorch-mnist>

Neural network



Sigmoid Activation Functions

- Logistic sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$

Sigmoid Activation Functions

- Logistic sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Hyperbolic tangent $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$

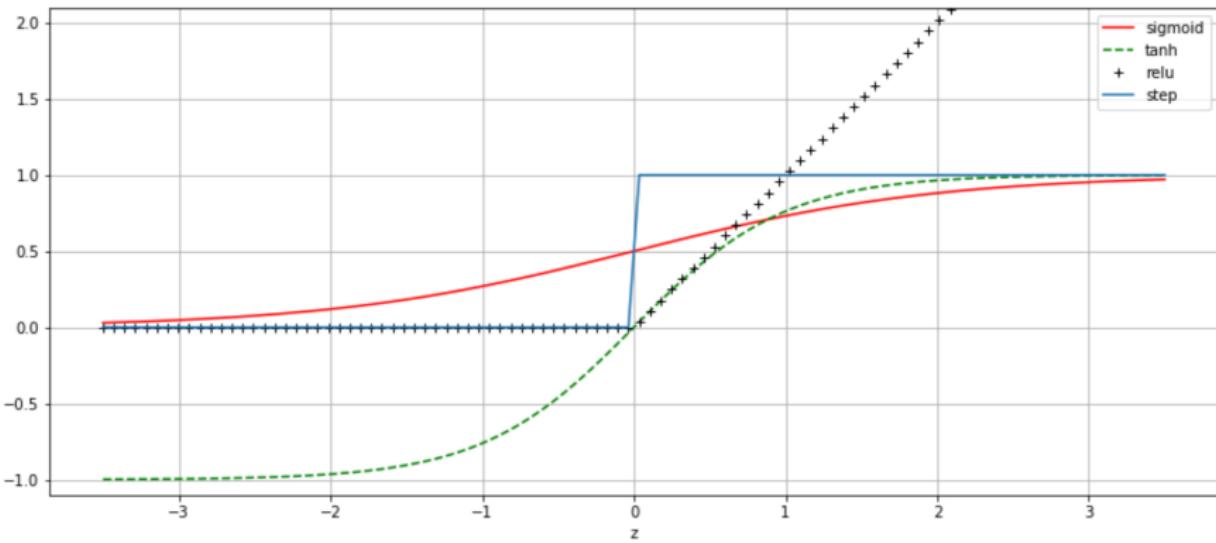
Sigmoid Activation Functions

- Logistic sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Hyperbolic tangent $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- continuous approximations of threshold function

Sigmoid Activation Functions

- Logistic sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Hyperbolic tangent $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- continuous approximations of threshold function
- can lead to vanishing gradient problem and "paralysis" of the network

Let's look at the charts



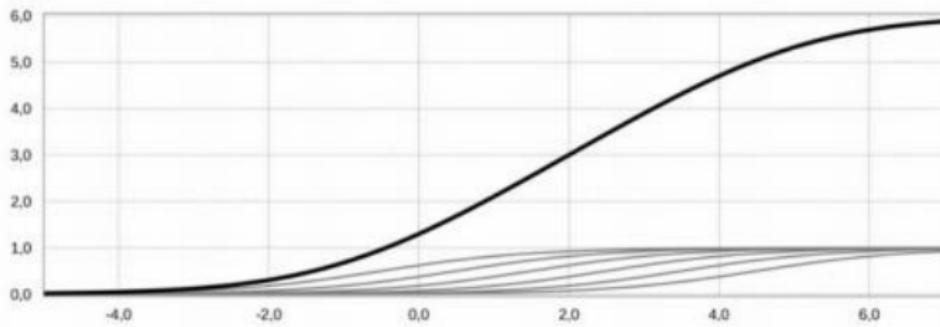
ReLU — Rectified Linear Unit

$$\text{ReLU}(z) = \max(0, z)$$

Motivation:

$$\sigma(5) \approx 0.9933, \sigma(10) \approx 0.9999$$

$$f(x) = \sigma\left(x + \frac{1}{2}\right) + \sigma\left(x - \frac{1}{2}\right) + \sigma\left(x - \frac{3}{2}\right) + \sigma\left(x - \frac{5}{2}\right) + \dots$$



(in Russian) Deep Learning: Dive into The World of Neural Networks, S.I.
Nikolenko, A.A. Kadurin, E.O. Arkhangelskaya, Piter, 2017

ReLU — Rectified Linear Unit

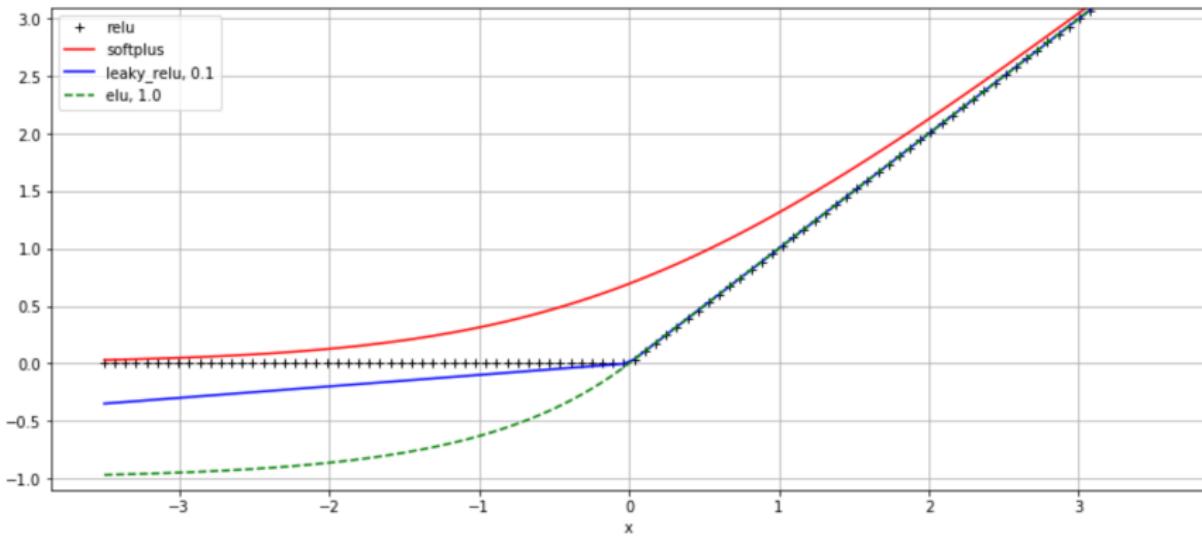
$$\int \sigma(x)dx = \log(1 + e^x) + C$$

It turns out that $f(x)$ is the Riemannian sum of such an integral:

$$\int_{1/2}^{\infty} \sigma(x + \frac{1}{2} - y)dy$$

$$\begin{aligned} f(x) &= \sum_{i=0}^{\infty} \sigma(x + \frac{1}{2} - i) \approx \int_{1/2}^{\infty} \sigma(x + \frac{1}{2} - y)dy = \\ &= [-\log(1 + \exp(x + \frac{1}{2} - y))]_{y=1/2}^{y=\infty} = \log(1 + \exp(x)) = \text{Softplus}(x) \end{aligned}$$

Let's look at the charts

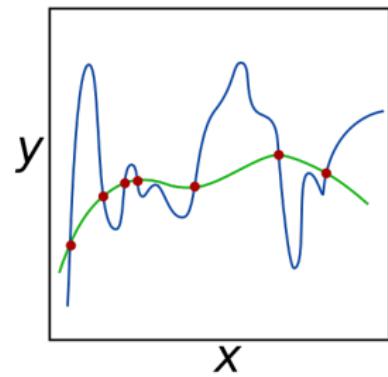


Regularization

$$L(W, b) = - \sum_j \ln \frac{e^{(x_j W + b)_{y_j}}}{\sum_i e^{(x_j W + b)_i}} + \lambda R(W, b)$$

$$R(W, b) = \|W\|_2^2 + \|b\|_2^2$$

$$\|b\|_2^2 = b_0^2 + \dots + b_k^2$$

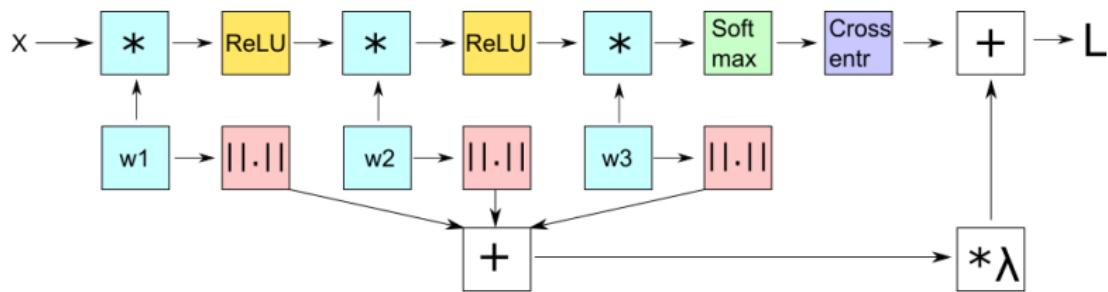


Backpropagation method

Computational graph

Let's include b into W

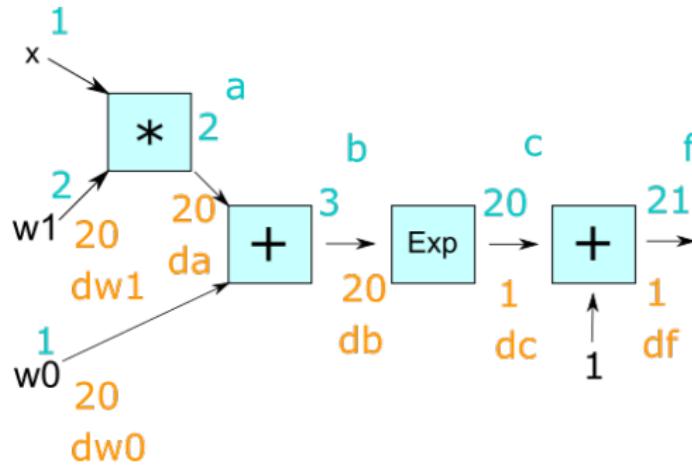
$$L(W) = - \sum_j \ln p(c = y_j | x_j) + \lambda R(W)$$



We want to find the gradients of the loss function L over all inputs of the computation graph

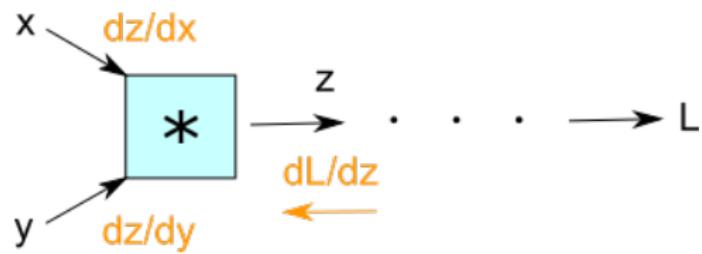
A simple example

Derivative of a function composition $f(g(x)) \rightarrow \frac{df}{dx} = \frac{df}{dg} \frac{dg}{dx}$
Let $f(x, w) = 1 + e^{w_1 x + w_0}$



$$\frac{\partial f}{\partial f} = 1, \quad f = c + 1, \quad dc = \frac{\partial f}{\partial c} = 1, \dots$$

General scheme for calculating the gradient



Backpropagation recap

As a result, we are able to calculate all the gradients with simple operations of the reverse pass through the graph and:

Backpropagation recap

As a result, we are able to calculate all the gradients with simple operations of the reverse pass through the graph and:

- we did not write out the entire derivative analytically

Backpropagation recap

As a result, we are able to calculate all the gradients with simple operations of the reverse pass through the graph and:

- we did not write out the entire derivative analytically
- at each step a simple function is differentiated

Backpropagation recap

As a result, we are able to calculate all the gradients with simple operations of the reverse pass through the graph and:

- we did not write out the entire derivative analytically
- at each step a simple function is differentiated
- we need only one pass along the calculation graph

Backpropagation recap

As a result, we are able to calculate all the gradients with simple operations of the reverse pass through the graph and:

- we did not write out the entire derivative analytically
- at each step a simple function is differentiated
- we need only one pass along the calculation graph
- parallelization is possible

Summary

Summary

- Neuron = linear classifier or regressor

Summary

- Neuron = linear classifier or regressor
- Neural network = superposition of neurons with non-linear activation functions

Summary

- Neuron = linear classifier or regressor
- Neural network = superposition of neurons with non-linear activation functions
- Backpropagation = fast differentiation of superpositions. Allows you to train networks of almost any configuration

Summary

- Neuron = linear classifier or regressor
- Neural network = superposition of neurons with non-linear activation functions
- Backpropagation = fast differentiation of superpositions. Allows you to train networks of almost any configuration
- Methods for improving convergence and quality:
 - ▶ training on mini-batches
 - ▶ various activation functions
 - ▶ regularization

Summary

- Neuron = linear classifier or regressor
- Neural network = superposition of neurons with non-linear activation functions
- Backpropagation = fast differentiation of superpositions. Allows you to train networks of almost any configuration
- Methods for improving convergence and quality:
 - ▶ training on mini-batches
 - ▶ various activation functions
 - ▶ regularization
- Was not at this lecture — will be in the next lectures of the course:
 - ▶ various optimization algorithms: adam, RMSProp
 - ▶ dropout
 - ▶ choice of initial approximation and its connection with activation functions

What else can you watch?

What else can you watch?

- In Stanford's course: <http://cs231n.github.io/optimization-2/>

What else can you watch?

- In Stanford's course: <http://cs231n.github.io/optimization-2/>
- 3blue1brown about backprop:
<https://www.youtube.com/watch?v=Ilg3gGewQ5U>

What else can you watch?

- In Stanford's course: <http://cs231n.github.io/optimization-2/>
- 3blue1brown about backprop:
<https://www.youtube.com/watch?v=Ilg3gGewQ5U>
- The third lecture of the course "Deep Learning on the fingers" from Semyon Kozlov (in Russian):
<https://www.youtube.com/watch?v=kWTC1NvL894>

In the case of a two-layer neural network

Output values of the network $a^m(x_i)$, $m = 1 \dots M$ on the object x_i :

$$a^m(x_i) = \sigma_m \left(\sum_{h=0}^H w_{hm} u^h(x_i) \right)$$

$$u^h(x_i) = \sigma_h \left(\sum_{j=0}^J w_{jh} f_j(x_i) \right)$$

Let for definiteness

$$\mathcal{L}_i(w) = \frac{1}{2} \sum_{m=1}^M (a^m(x_i) - y_i^m)^2$$

Intermediate task: Partial Derivatives $\frac{\partial \mathcal{L}_i(w)}{\partial a^m}$, $\frac{\partial \mathcal{L}_i(w)}{\partial u^h}$

Fast differentiation. Auxiliary gradients

Intermediate task: Partial Derivatives

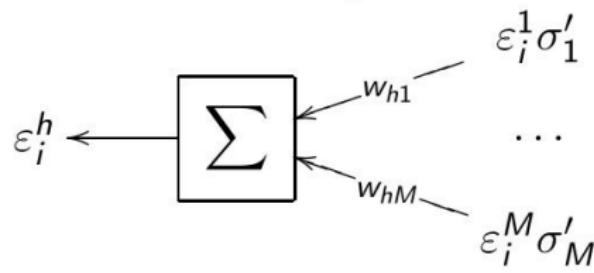
$$\frac{\partial \mathcal{L}_i(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m$$

is the error at the output layer (for quadratic losses);

$$\frac{\partial \mathcal{L}_i(w)}{\partial u^h} = \sum_{m=1}^M (a^m(x_i) - y_i^m) \sigma'_m w_{hm} = \sum_{m=1}^M \varepsilon_i^m \sigma'_m w_{hm} = \varepsilon_i^h$$

— let's call it *an error on a hidden layer*.

It turns out that ε_i^h is calculated from ε_i^m if the network is run backwards:



Fast gradient calculation

Now, given the partial derivatives of $\mathcal{L}_i(w)$ with respect to a^m and u^h , it is easy to write out the gradient of $\mathcal{L}_i(w)$ with respect to the weights of w :

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{hm}} = \frac{\partial \mathcal{L}_i(w)}{\partial a^m} \frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m \sigma'_m u^h(x_i),$$

$m = 1, \dots, M, h = 0, \dots, H$

$$\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}} = \frac{\partial \mathcal{L}_i(w)}{\partial u^h} \frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h \sigma'_h f_j(x_i),$$

$h = 1, \dots, H, j = 0, \dots, n$

Backpropagation algorithm

- ① initialize weights w_{jh}, w_{hm}

repeat

- ② select object x_i from X^ℓ (e.g. randomly)

- ③ forward pass

$$u_i^h = \sigma_h \left(\sum_{j=0}^J w_{jh} x_i^j \right), h = 1, \dots, H$$

$$a_i^m = \sigma_m \left(\sum_{h=0}^H w_{hm} u_i^h \right), \varepsilon_i^m = a_i^m - y_i^m, m = 1, \dots, M$$

$$\mathcal{L}_i = \sum_{m=1}^M (\varepsilon_i^m)^2$$

- ④ backward

$$\varepsilon_i^h = \sum_{m=1}^M \varepsilon_i^m \sigma'_m w_{hm}, h = 1 \dots H$$

- ⑤ gradient step

$$w_{hm} = w_{hm} - \eta \varepsilon_i^m \sigma'_m u_i^h, h = 0, \dots, H, m = 1 \dots M$$

$$w_{jh} = w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j, j = 0, \dots, n, h = 1 \dots H$$

- ⑥ $Q = (1 - \lambda)Q + \lambda \mathcal{L}_i$

until Q stabilizes

Expressive power of a neural network

$\sigma(z)$ is a sigmoid function if $\lim_{z \rightarrow -\infty} \sigma(z) = 0$ and $\lim_{z \rightarrow +\infty} \sigma(z) = 1$

Cybenko's theorem

(based on Kolmogorov's theorem on the representability of multidimensional functions)

If $\sigma(z)$ is a continuous sigmoid, then for any function $f(x)$ continuous on $[0, 1]^n$ there are values of the parameters $w_h \in \mathbb{R}^n$, $w_0 \in \mathbb{R}$, $\alpha_h \in \mathbb{R}$ that is a one-layer network

$$a(x) = \sum_{h=1}^H \alpha_h \sigma(\langle x, w_h \rangle - w_0)$$

approximates $f(x)$ uniformly with any accuracy ε :

$$|a(x) - f(x)| < \varepsilon, \text{ for all } x \in [0, 1]^n$$

G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems (MCSS) 2 (4): 303–314 (Dec 1, 1989)