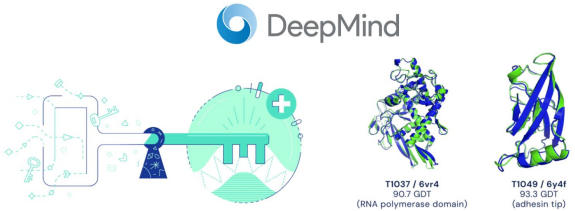
Deep Learning

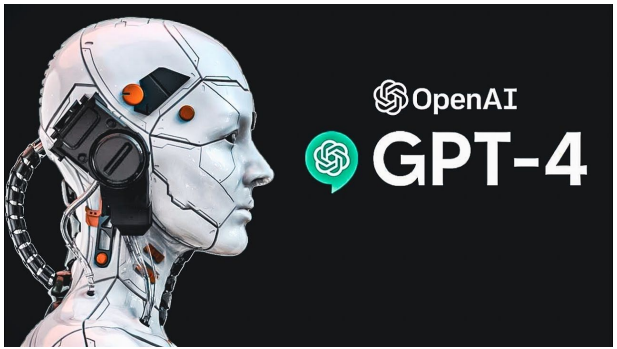
Motivation №1: Solving complex problems:

Deep learning techniques have been successfully applied to a wide range of challenging tasks, such as image recognition, natural language processing, medical diagnosis, and autonomous driving. By studying deep learning, you can acquire the knowledge and tools to tackle complex problems and contribute to cutting-edge research and applications.

Stable Diffusion



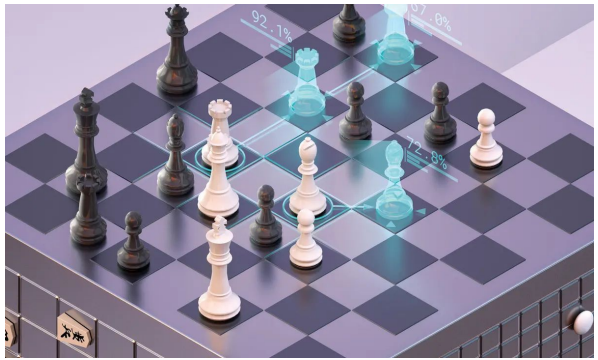
AlphaFold



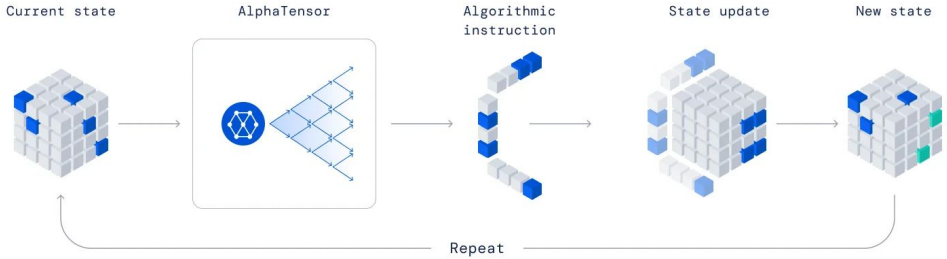
GPT-4

Motivation №2: Interdisciplinary collaboration

Deep learning transcends traditional boundaries, bringing together various disciplines like computer science, mathematics, statistics, and neuroscience. As you study this field, you will have opportunities to collaborate with experts from diverse backgrounds, fostering cross-disciplinary innovation and gaining a broader perspective on problem-solving.



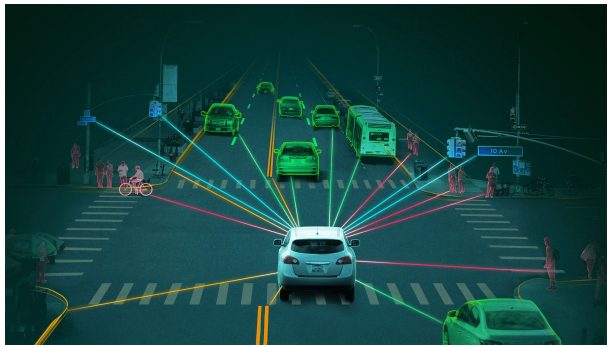
AlphaZero



AlphaTensor

Motivation №3: Fuel the technological revolution:

Deep learning is at the forefront of today's technological advancements. By acquiring knowledge in this field, you can actively contribute to shaping the future. Join the ranks of brilliant minds who are revolutionizing industries, such as healthcare, finance, transportation, and robotics, by developing cutting-edge AI solutions.



Autonomous Driving

Medical Assistance

Aim of the course

The purpose of this course is to acquaint you with the world of neural networks:

- What type of problems one can solve using ML/DL?
- What methodology should be chosen to solve the problem?
- How to implement the chosen methodology?

Team

Andrei Filatov

Olga Grebenkova

Anton Bishuk

Dmitry Kropotov

Team

Andrei Filatov

Experience

SBER AI — Research Scientist

MIPT — Lecturer on DL course

Past Experience

EPFL — Research Intern

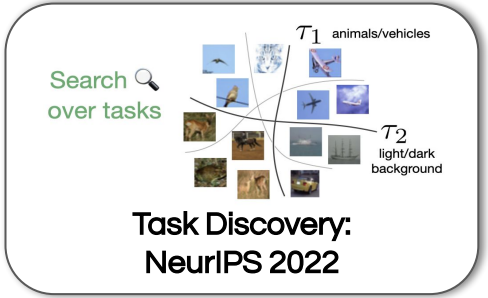
SAMSUNG — Research Intern

TINKOFF — Intern

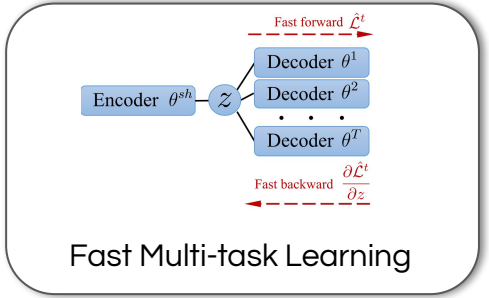
Education

- Master: MIPT + Skoltech with honours
- Bachelor: MIPT with honours

Projects



Task Discovery: NeurIPS 2022



Fast Multi-task Learning

Konstantin Yakovlev

Education

- Bachelor: MIPT with *honours*

Publications

- Yakovlev, Konstantin et al. “GEC-DePenD: Non-Autoregressive Grammatical Error Correction with Decoupled Permutation and Decoding.” *Annual Meeting of the Association for Computational Linguistics* (2023).
- Yakovlev, Konstantin et al. “Sinkhorn Transformations for Single-Query Postprocessing in Text-Video Retrieval.” *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2023): n. pag.
- Yakovlev, K.D., Grebenkova, O.S., Bakhteev, O.Y., Strijov, V.V. (2022). Neural Architecture Search with Structure Complexity Control. In: Burnaev, E., *et al.* Recent Trends in Analysis of Images, Social Networks and Texts. AIST 2021.

Eduard Vladimirov

Education

Bachelor: MIPT with *honours*

Avito Academy of Analysts

Deep Learning Systems

Experience

Junior Analyst

Projects

Diffusion Model from scratch

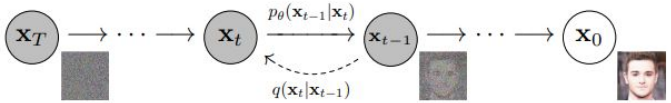


Figure 2: The directed graphical model considered in this work.

Alexander Molozhavenko

Course

Course programme

Course consists of 14 lectures & seminars:

- Lecture will take one and a half and is focused on theory.
- Then we have a break and continue with a seminar which is focused on practical details of the theory.

Discussed topics:

- Basics of deep learning (3 classes)
- NLP (2 classes)
- Computer Vision (2 classes)
- RL (3 classes)
- Generative models (2 classes)
- Other topics (2 classes)

Final mark = 0.3 x Exam + 0.7 x Homework

Homework

- Homework consists of 5 assignments: 1 theoretical, 4 programming.
- You will need GPU to complete your programming homework. Therefore, we recommend that you understand Google Colab and Kaggle.
- Possibly we have some bugs in code :(
  Don’t hesitate to contact us. If you let us know after the deadline, we can’t help you.
- Bug fixes and homework proposition (fully implemented in code) are encourages and are rewarded!

Exam

The exam is taken at the end of the course on the topics covered. It consists of answering a question on the theoretical minimum and the main ticket. It is obligatory to pass the exam to get positive mark for the course.

Collaboration policy

All submitted content should be your own content, written yourself!

However, you may (in fact are encouraged to) discuss the homework with others in the chat:

- This creates some room for undue copying, but please obey the reasonable person principle: discuss as you see fit, but don’t simply share answers

You may use snippets of code from sources like Stack Overflow, as long as you cite these properly (put a link to the source)

Lecture 1: Multi-layer perceptron

Reminder: Logistic Regression

Consider some binary classification task:

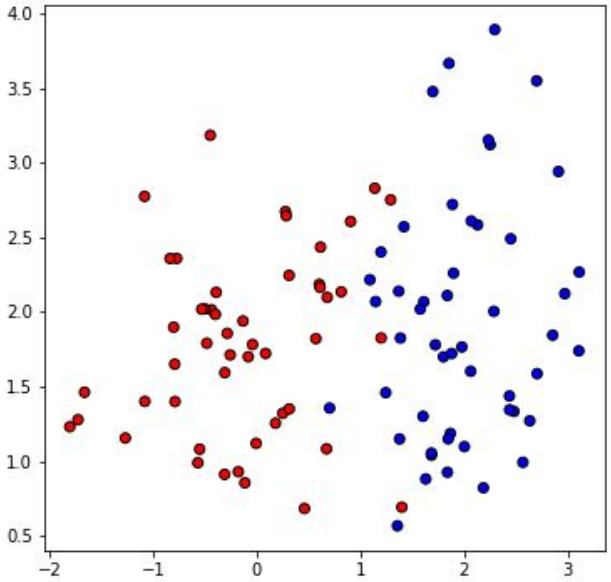
$$\{x_i, y_i\}_{i=1}^n, \; x_i \in R^d, y_i \in \{-1, 1\};$$

We want to train logistic regression models

$$p(y = 1|w, x) = \frac{1}{1 + \exp(-w^\top x)}$$

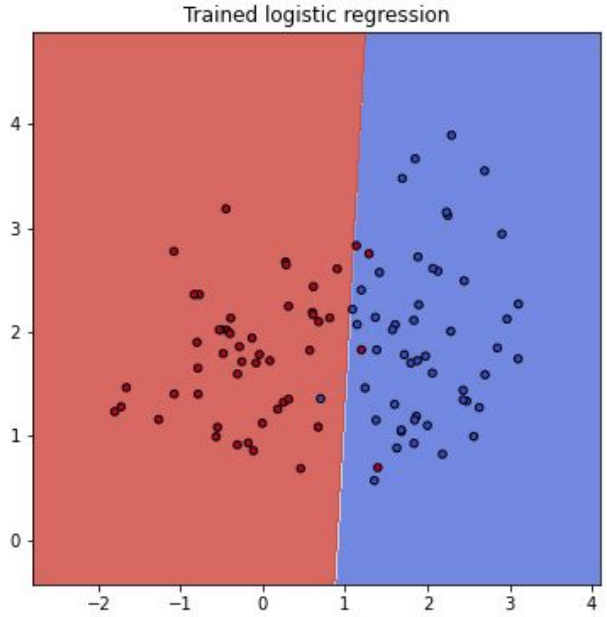
Loss function

$$-\log p(y|W, X) = -\sum_{i=1}^{N} \log p(y_i|W, x_i) \to min_W$$



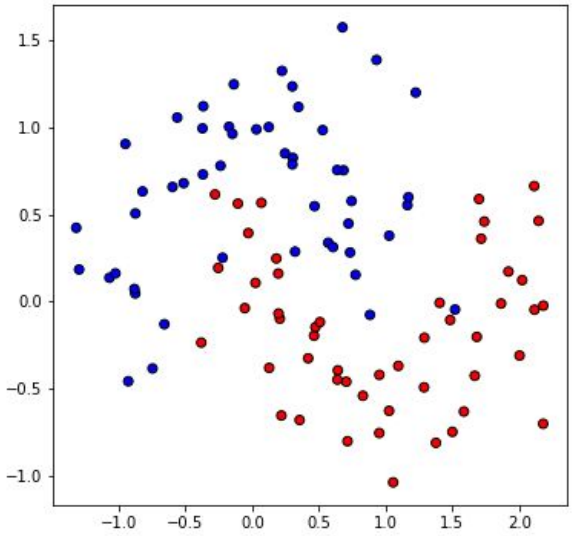
https://mlu-explain.github.io/logistic-regression/

Result

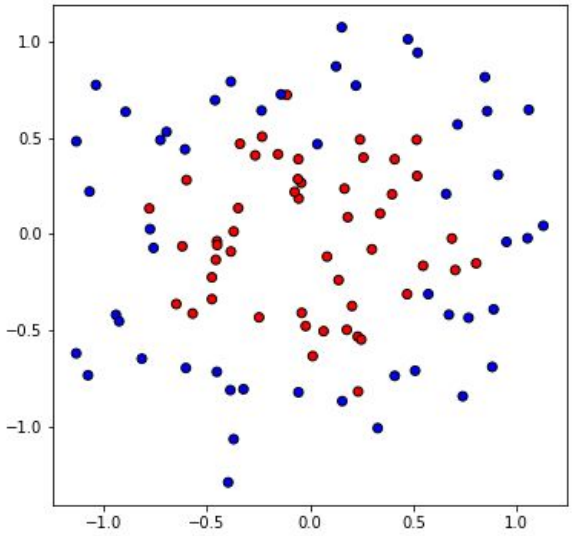


Logistic regression can solve the classification problem with a high accuracy

What does happen when our data has non-linear relationships in our data?



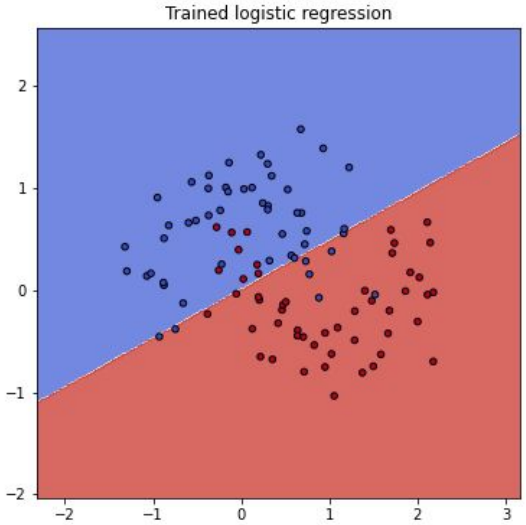
Case 1



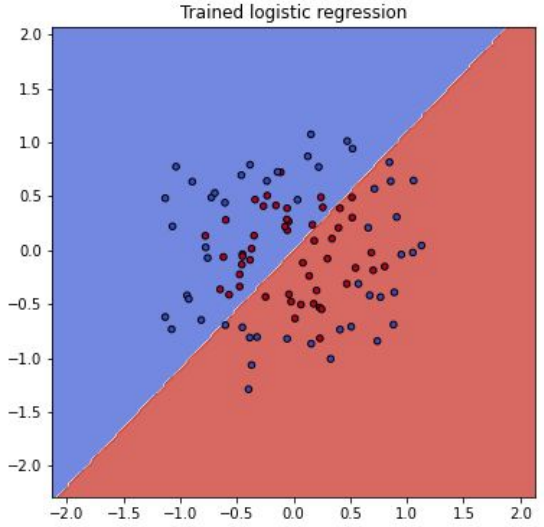
Case 2

In this case logistic regression fails :(

- Moreover, we can hardly do anything about…

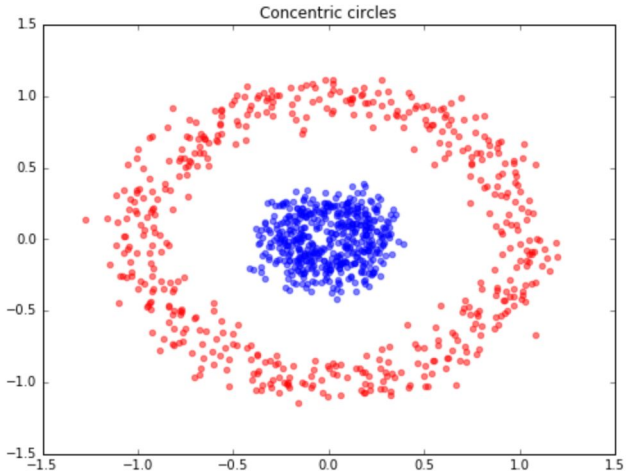


Case 1



Case 2

Some reasons perceptron a.k.a logistic regression fails



- Data can’t be splitted in linear way

- Features itself are non-informative. But combination does!

https://thiscatdoesnotexist.com

We see failure cases of logistic regression:

- To be honest, it’s just regular cases of real-world data

Audio

Graphs

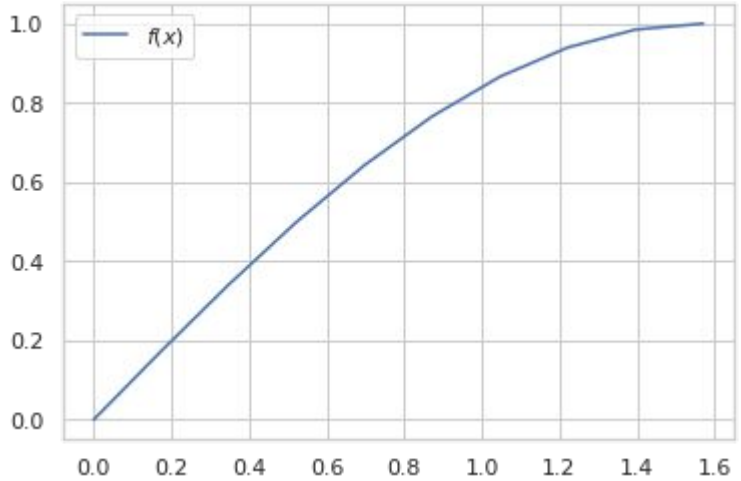
Image

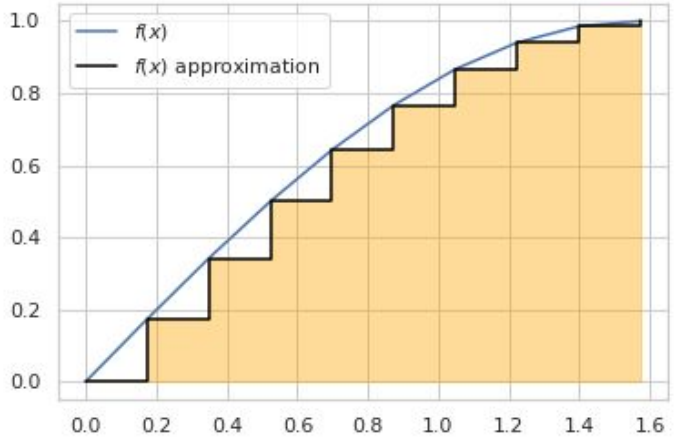
So, we should figure out some solution to overcome linear models problems:

- As always, let’s go to the basics

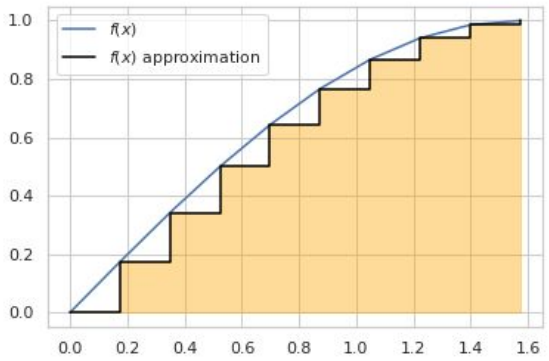
Reminder: Calculus II term

How can we approximate arbitrary continuous function?





We can use approximation with constant thresholds!



Mathematically, it can be written in the following way

- We can approximate the function with the following sum:

$$f(x) \approx \sum_{i=1}^{N} f(a_i)\underbrace{[x \in [a_i, b_i]]}$$

It is not differentiable…

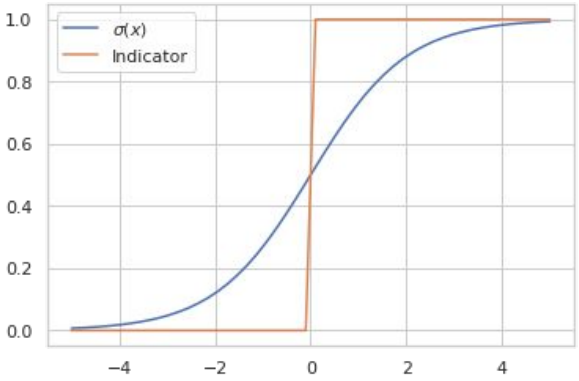
How can we approximate the following sum even further?

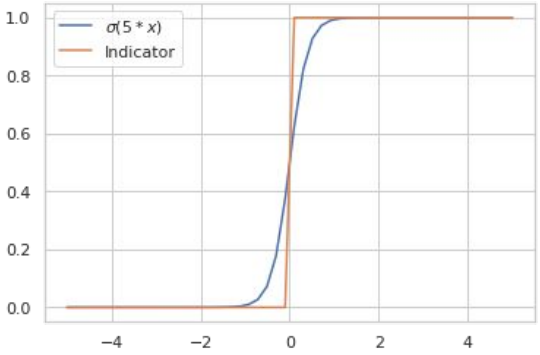
What about sigmoid?

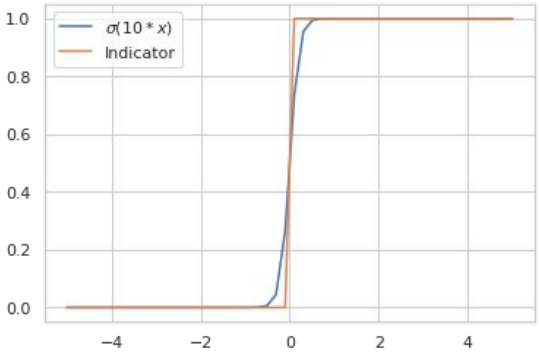
$$\sigma(w, x) = \frac{1}{1 + \exp(-w \cdot x)}$$

Let's add weight to sigmoid function and try to increase it

w







With the increase of w the sigmoid becoming very close to indicator function!

How to approximate interval indicator?

We have learnt on how to approximate the following function: $[x \geqslant 0]$

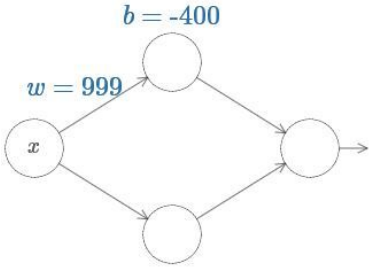
But in reality we need to approximate the following: $f(x) \approx \sum_{i=1}^{N} f(a_i)[x \in [a_i, b_i]]$

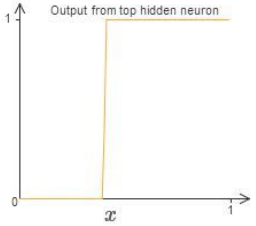
We can do a simple trick!

$$[x \in [a_i, b_i]] = [[x \geqslant a_i] - [x \geqslant b_i] > 0.5]$$

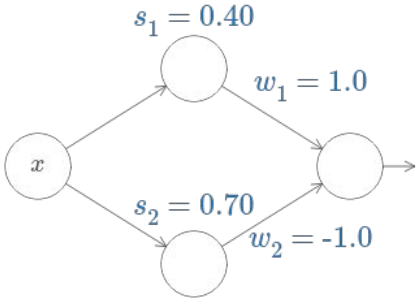
Summary!

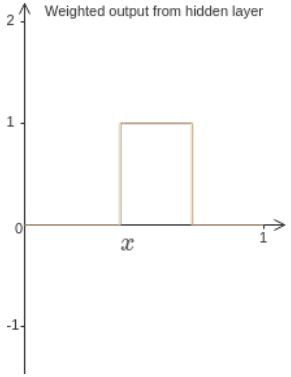
(1) Approximation of Heaviside function



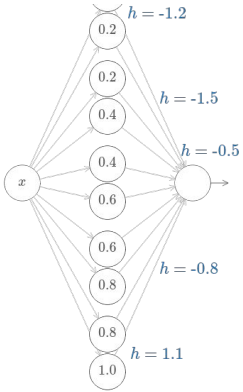


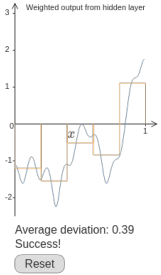
(2) Approximation of Indicator function





3) Approximation of any continuous function





Average deviation: 0.39
Success!
Reset

Universal approximation theorem

A sum of the form:

$$f(\mathbf{x}) = \sum_j w_j \sigma\left(b_j + \mathbf{v}_j \cdot \mathbf{x}\right)$$

can approximate any continuous function to any accuracy, but it might require large number of hidden neurons.

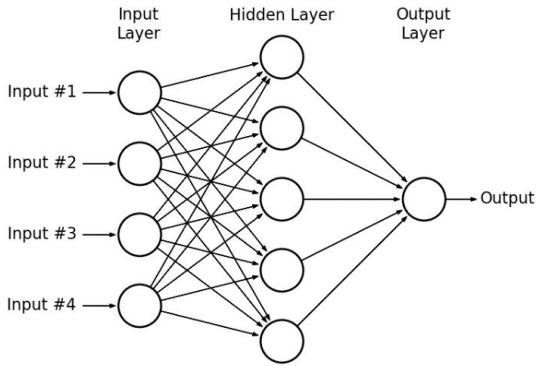
https://www.deep-mind.org/2023/03/26/the-universal-approximation-theorem/

Activation functions

Okay, we should do the following:

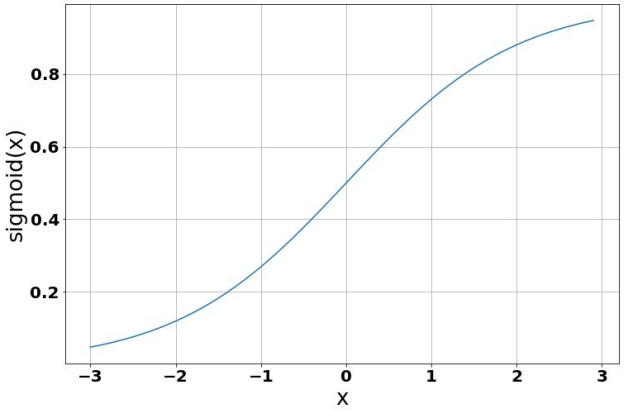
$$h_i = \sigma\left(\sum_{j=1}^{n_{\text{inputs}}} w_{ij}x_j + b_i\right), \quad i = 1, 2, \ldots, n_{\text{hidden}}$$

$$y_k = \sigma\left(\sum_{i=1}^{n_{\text{hidden}}} w'_{ki}h_i + b'k\right), \quad k = 1, 2, \ldots, n_{\text{outputs}}$$

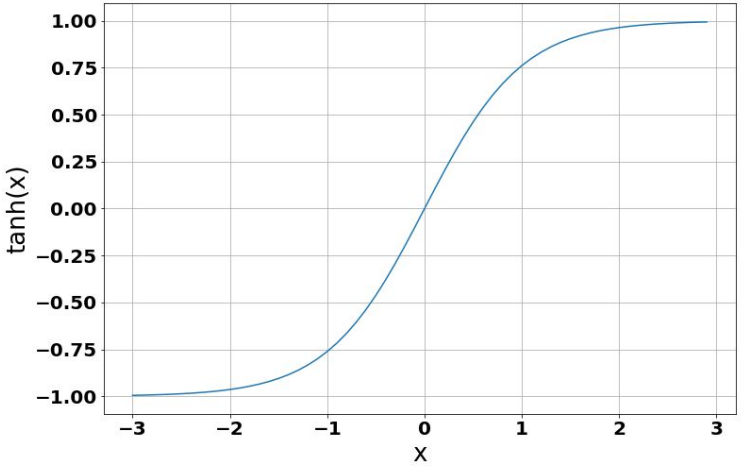


- Can we use Identity transformation instead of sigmoid?
- Can we use other functions?

Activation functions

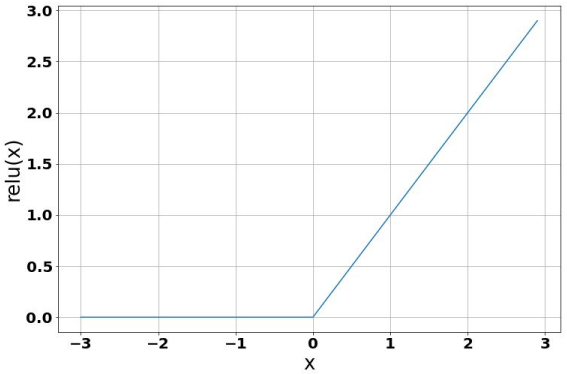


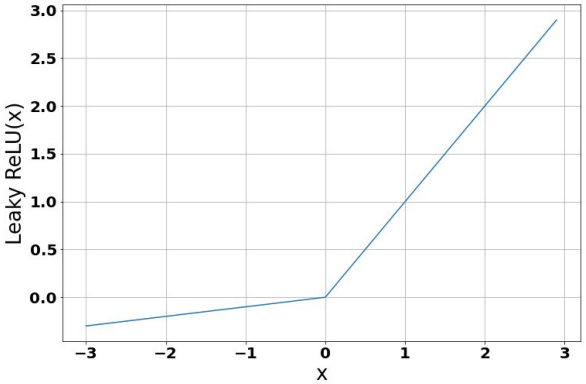
Sigmoid(x)

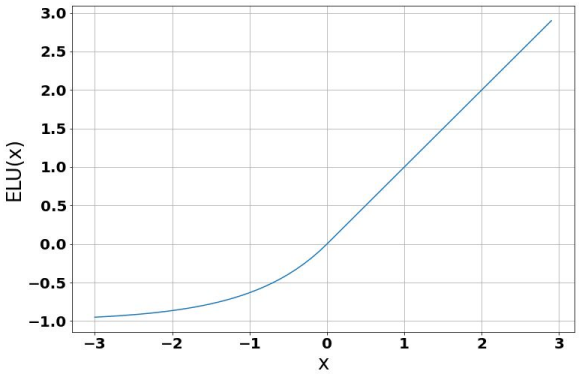


Tanh(x)

Activation functions





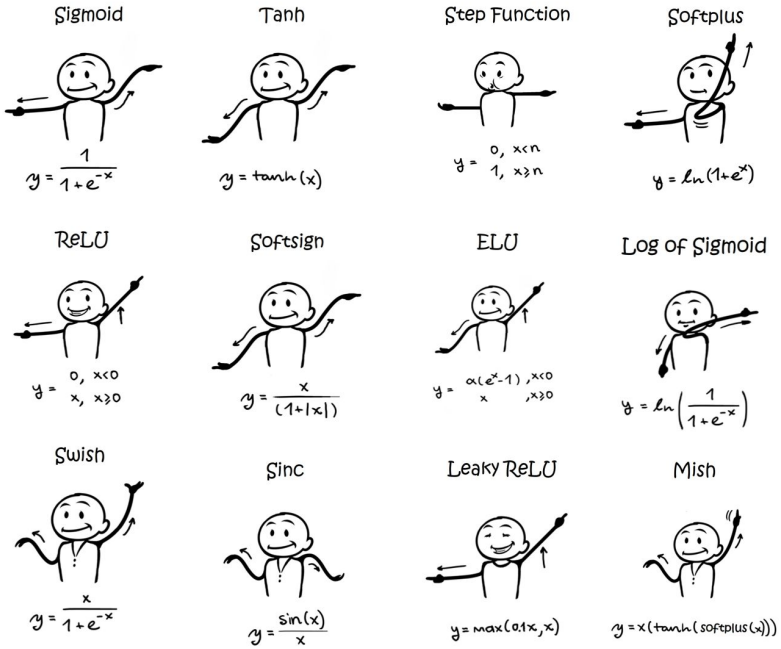


$$ReLU(x) = \begin{Bmatrix} x & x > 0 \\ 0 & x <= 0 \end{Bmatrix}$$

$$LeakyReLU(z) = \begin{Bmatrix} x & x > 0 \\ \alpha x & x <= 0 \end{Bmatrix}$$

$$ELU(x) = \begin{Bmatrix} x & x > 0 \\ \alpha(e^{\frac{x}{\alpha}} - 1) & x \leqslant 0 \end{Bmatrix}$$

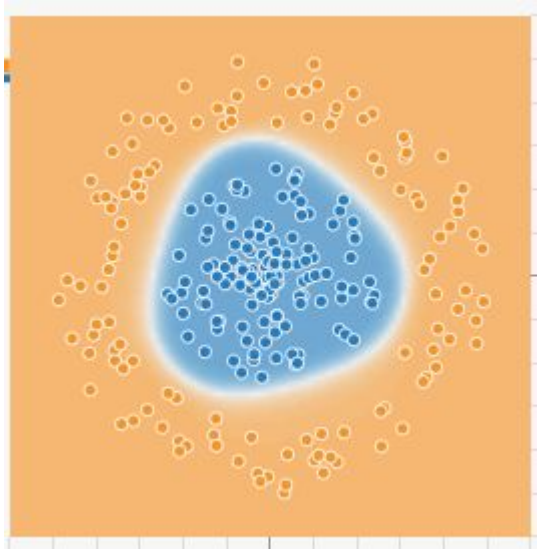
Activation functions



https://sefiks.com/2020/02/02/dance-moves-of-deep-learning-activation-functions/

https://www.vincentsitzmann.com/siren/

Conclusion



- To solve on more complex data, where there is complex relationship between features we should use neural networks.
- A simple example of neural networks isa composition of several perceptron, a.k.a multilayer perceptron
- Multilayer perceptron is a very expressive model - even **one hidden layer is enough to approximate any continuous function.**

https://playground.tensorflow.org/

Gradient calculation

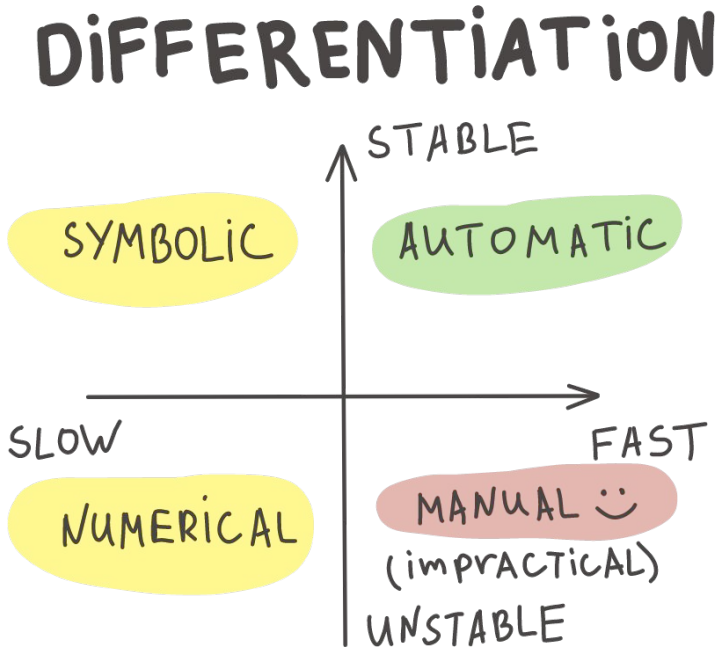
Why do we need gradient calculation?

The workhorse of optimization is gradient descent algorithm:

- Deep learning is not an exception!

We should figure out on how to calculate the gradient:

- Let’s recall our memories about differentiation!



There are four ways on how to do differentiation!

Numerical

Calculation of the gradient by definition:

$$\frac{\partial f}{\partial x_i} = \lim_{\varepsilon \to 0} \frac{f\left(x + \varepsilon e_i\right) - f(x)}{\varepsilon} \approx \frac{f\left(x + \varepsilon e_i\right) - f(x)}{\varepsilon}$$

Problems

- For large x dimension we should do a lot of function calls
- Numerically instability, optimal **eps** is equal to square root of machine precision

Hint: One can use numerical differentiation for checking automatic differentiation

Numerically with any precision

Consider f - complex-valued function

$$f(x + i\varepsilon d) = f(x) + i\varepsilon \nabla_x f^\top d + O(\varepsilon^2)$$

So, we can calculate the gradient using only one function call:

$$\nabla f(x)^\top d = \frac{Im[f(x + i\varepsilon d)]}{\varepsilon}$$

Neural networks in machine learning

Consider the following optimization problem:

$$\underset{\theta}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} \ell_{ce}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

Requires computing the gradients $\nabla_\theta \ell_{ce}\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$

Let's work through the derivation of the gradients for a simple two-layer network, written in batch matrix form, i.e.

$$\nabla_{\{W_1, W_2\}} \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)$$

The next slides are taken from https://dlsyscourse.org/lectures/

The gradient(s) of a two-layer network

$$\frac{\partial \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)}{\partial W_2} =$$

The gradient(s) of a two-layer network

$$\frac{\partial \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)}{\partial W_2} = \frac{\partial \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)}{\partial \sigma\left(XW_1\right)W_2} \cdot \frac{\partial \sigma\left(XW_1\right)W_2}{\partial W_2}$$

$$= \left(S - I_y\right) \cdot \left(\sigma\left(XW_1\right)\right)$$

$$\left[S = \text{normalize}\left(\exp\left(\sigma\left(XW_1\right)W_2\right)\right]\right.$$

so (matching sizes) the gradient is

$$\nabla_{W_2} \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right) = \sigma\left(XW_1\right)^T\left(S - I_y\right)$$

The gradient(s) of a two-layer network

Deep breath and let's do the gradient

$$\frac{\partial \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)}{\partial W_1} =$$

The gradient(s) of a two-layer network

Deep breath and let's do the gradient

$$\frac{\partial \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)}{\partial W_1} = \frac{\partial \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right)}{\partial \sigma\left(XW_1\right)W_2} \cdot \frac{\partial \sigma\left(XW_1\right)W_2}{\partial \sigma\left(XW_1\right)} \cdot \frac{\partial \sigma\left(XW_1\right)}{\partial XW_1} \cdot \frac{\partial XW_1}{\partial W_1}$$

$$= \left(S - I_y\right) \cdot \left(W_2\right) \cdot \left(\sigma'\left(XW_1\right)\right) \cdot \left(X\right)$$

and so the gradient is

$$\nabla_{W_1} \ell_{ce}\left(\sigma\left(XW_1\right)W_2, y\right) = X^T\left(\left(S - I_y\right)W_2^T \circ \sigma'\left(XW_1\right)\right)$$

Backpropagation “in general”

There *is* a method to this madness … consider our fully-connected network:

$$Z_{i+1} = \sigma_i(Z_i W_i), \qquad i = 1, \ldots, L$$

Then (now being a bit terse with notation)

$$\frac{\partial \ell(Z_{L+1}, y)}{\partial W_i} = \underbrace{\frac{\partial \ell}{\partial Z_{L+1}} \cdot \frac{\partial Z_{L+1}}{\partial Z_L} \cdot \frac{\partial Z_{L-1}}{\partial Z_{L-2}} \cdot \ldots \cdot \frac{\partial Z_{i+2}}{\partial Z_{i+1}}}_{G_{i+1} = \frac{\partial \ell(Z_{L+1}, y)}{\partial Z_{i+1}}} \cdot \frac{\partial Z_{i+1}}{\partial W_i}$$

Then we have a simple ”backward” iteration to compute the G_i’s

$$G_i = G_{i+1} \cdot \frac{\partial Z_{i+1}}{\partial Z_i} = G_{i+1} \cdot \frac{\partial \sigma_i(Z_i W_i)}{\partial Z_i W_i} \cdot \frac{\partial Z_i W_i}{\partial Z_i} = G_{i+1} \cdot \sigma'(Z_i W_i) \cdot W_i$$

Computing the real gradients

To convert these quantities to "real" gradients, consider matrix sizes

$$G_i = \frac{\partial \ell(Z_{L+1}, y)}{\partial Z_i} = \nabla_{Z_i} \ell(Z_{L+1}, y) \in \mathbb{R}^{m \times n_i}$$

so with "real" matrix operations

$$G_i = G_{i+1} \cdot \sigma'(Z_i W_i) \cdot W_i = \Big(G_{i+1} \circ \sigma'(Z_i W_i)\Big) W_i^T$$

Similar formula for actual parameter gradients $\nabla_{W_i} \ell(Z_{L+1}, y) \in \mathbb{R}^{n_i \times n_{i+1}}$

$$\frac{\partial \ell(Z_{L+1}, y)}{\partial W_i} = G_{i+1} \cdot \frac{\partial \sigma_i(Z_i W_i)}{\partial Z_i W_i} \cdot \frac{\partial Z_i W_i}{\partial W_i} = G_{i+1} \cdot \sigma'(Z_i W_i) \cdot Z_i$$

$$\Longrightarrow \nabla_{W_i} \ell(Z_{L+1}, y) = Z_i^T \Big(G_{i+1} \circ \sigma'(Z_i W_i)\Big)$$

Backpropagation

Putting it all together, we can efficiently compute *all* the gradients we need for a neural network by following the procedure below

1. Initialize: $Z_1 = X$
   Iterate: $Z_{i+1} = \sigma_i(Z_i W_i), \quad i = 1, \dots, L$ } Forward pass
2. Initialize: $G_{L+1} = \nabla_{Z_{L+1}} \ell(Z_{L+1}, y) = S - I_y$
   Iterate: $G_i = \left(G_{i+1} \circ \sigma_i'(Z_i W_i)\right) W_i^T, \quad i = L, \dots, 1$ } Backward pass

And we can compute all the needed gradients along the way

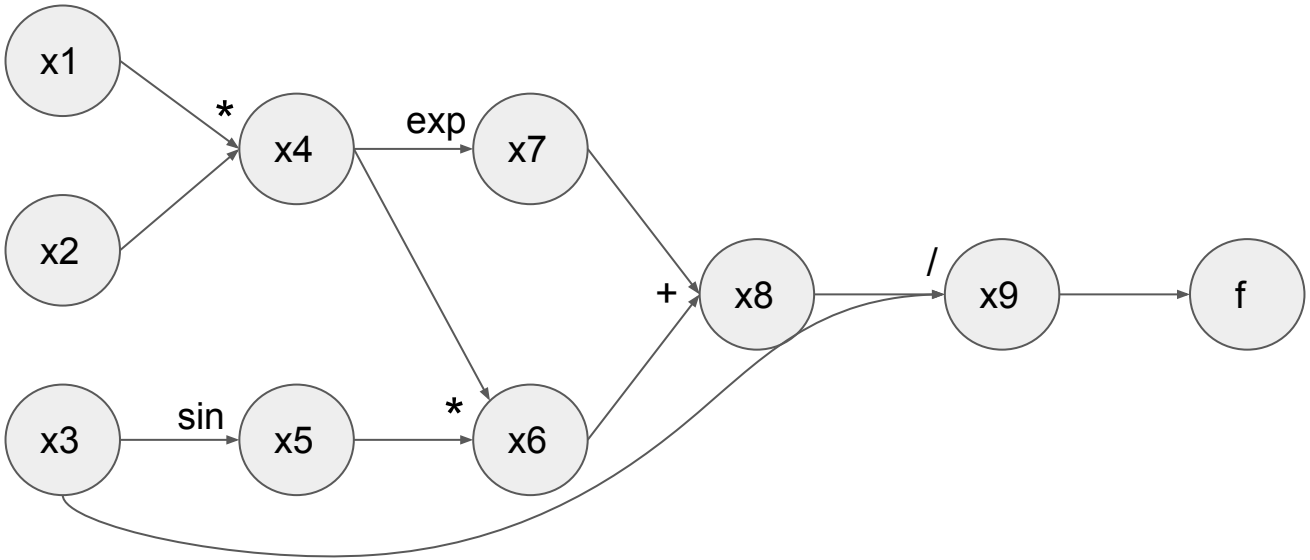
$$\nabla_{W_i} \ell(Z_{k+1}, y) = Z_i^T \left(G_{i+1} \circ \sigma_i'(Z_i W_i)\right)$$

"Backpropagation" is just chain rule + intelligent caching of intermediate results

Automatic differentiation

$$f(x_1, x_2, x_3) = \frac{x_1 x_2 \sin(x_3) + \exp(x_1 x_2)}{x_3}$$

Need: $\nabla f(x)$



Passing forward

At each step, the derivative $\frac{\partial x_j}{\partial x_i}$ will be calculated, where $i = 1, 2, 3$ are input variables, and $j = 4, 5, \ldots, 9$ are - nodes that will be sequentially calculated.

1. Initialization: $\frac{\partial x_1}{\partial x_1} = 1$, $\frac{\partial x_1}{\partial x_2} = 0$, $\frac{\partial x_1}{\partial x_3} = 0$
2. Calculate the value: $x_4 = x_1 x_2$
3. Calculate derivatives on the current layer: $\frac{\partial x_4}{\partial x_1} = \frac{\partial x_1}{\partial x_1} x_2 + x_1 \frac{\partial x_2}{\partial x_1}$, and we take the derivatives from the previous layer
4. Similarly, for the next layer, the value: $x_5 = \sin(x_3)$ and the derivative: $\frac{\partial x_5}{\partial x_1} = \cos(x_3) \frac{\partial x_3}{\partial x_1}$, where the derivative $\frac{\partial x_3}{\partial x_1}$ was calculated on the previous layer.
5. And so on.

Advantages of pass forward: No extra memory needed.
Disadvantages of pass forward: You need to traverse the calculation graph as many times as the input parameters.

Passing backwards

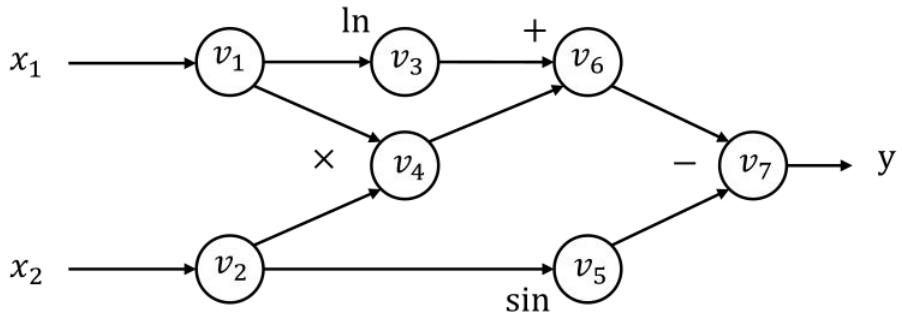
At each step, the derivative $\frac{\partial}{\partial x_j}$ will be calculated, where $j = 9, 8, \ldots, 1$ are the nodes that will be sequentially calculated.

1. First step: $\frac{\partial f}{\partial x_9} = 1$
2. Second step: $\frac{\partial f}{\partial x_8} = \frac{\partial f}{\partial x_9}\frac{\partial x_9}{\partial x_8} = 1 \cdot \frac{1}{x_3}$
3. And so on
4. In general: $\frac{\partial f}{\partial x_j} = \sum_{(j,k)\in\varepsilon} \frac{\partial f}{\partial x_k}\frac{\partial x_k}{\partial x_j}$, where $\frac{\partial f}{\partial x_k}$ is known from the previous step, and $\frac{\partial x_k}{\partial x_j}$ is computed directly from the graph .

Advantages of pass backwards: Only one backtrack is required to compute the gradient.
Disadvantages of pass backwards: You need additional memory to store all intermediate vertices.

Computational graph

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$



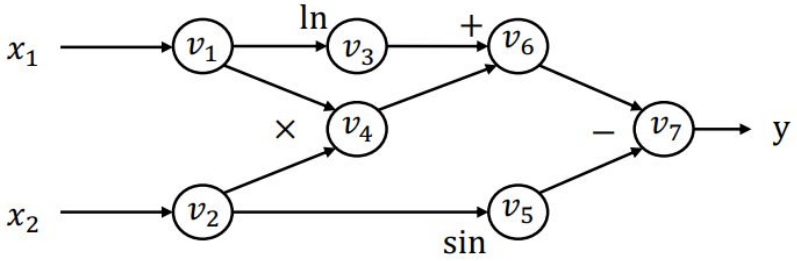
Each node represent an (intermediate) value in the computation. Edges present input output relations.

Forward evaluation trace

$$v_1 = x_1 = 2$$
$$v_2 = x_2 = 5$$
$$v_3 = \ln v_1 = \ln 2 = 0.693$$
$$v_4 = v_1 \times v_2 = 10$$
$$v_5 = \sin v_2 = \sin 5 = -0.959$$
$$v_6 = v_3 + v_4 = 10.693$$
$$v_7 = v_6 - v_5 = 10.693 + 0.959 = 11.652$$
$$y = v_7 = 11.652$$

Reverse mode AP

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$



Forward evaluation trace

$$v_1 = x_1 = 2$$
$$v_2 = x_2 = 5$$
$$v_3 = \ln v_1 = \ln 2 = 0.693$$
$$v_4 = v_1 \times v_2 = 10$$
$$v_5 = \sin v_2 = \sin 5 = -0.959$$
$$v_6 = v_3 + v_4 = 10.693$$
$$v_7 = v_6 - v_5 = 10.693 + 0.959 = 11.652$$
$$y = v_7 = 11.652$$

Define adjoint $\overline{v_i} = \frac{\partial y}{\partial v_i}$

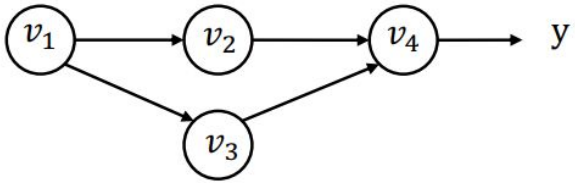
We can then compute the $\overline{v_i}$ iteratively in the **reverse** topological order of the computational graph

Reverse AD evaluation trace

$$\overline{v_7} = \frac{\partial y}{\partial v_7} = 1$$
$$\overline{v_6} = \overline{v_7}\frac{\partial v_7}{\partial v_6} = \overline{v_7} \times 1 = 1$$
$$\overline{v_5} = \overline{v_7}\frac{\partial v_7}{\partial v_5} = \overline{v_7} \times (-1) = -1$$
$$\overline{v_4} = \overline{v_6}\frac{\partial v_6}{\partial v_4} = \overline{v_6} \times 1 = 1$$
$$\overline{v_3} = \overline{v_6}\frac{\partial v_6}{\partial v_3} = \overline{v_6} \times 1 = 1$$
$$\overline{v_2} = \overline{v_5}\frac{\partial v_5}{\partial v_2} + \overline{v_4}\frac{\partial v_4}{\partial v_2} = \overline{v_5} \times \cos v_2 + \overline{v_4} \times v_1 = -0.284 + 2 = 1.716$$
$$\overline{v_1} = \overline{v_4}\frac{\partial v_4}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_4} \times v_2 + \overline{v_3}\frac{1}{v_1} = 5 + \frac{1}{2} = 5.5$$

Derivation for the multiple pathway case

v_1 is being used in multiple pathways (v_2 and v_3)



y can be written in the form of $y = f(v_2, v_3)$

$$\overline{v_1} = \frac{\partial y}{\partial v_1} = \frac{\partial f(v_2,v_3)}{\partial v_2}\frac{\partial v_2}{\partial v_1} + \frac{\partial f(v_2,v_3)}{\partial v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_2}\frac{\partial v_2}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1}$$

Define partial adjoint $\overline{v_{i\to j}} = \overline{v_j}\dfrac{\partial v_j}{\partial v_i}$ for each input output node pair i and j

$$\overline{v_i} = \sum_{j \in next(i)} \overline{v_{i\to j}}$$

We can compute partial adjoints separately then sum them together

Reverse mode AD algorithm

```
def gradient(out):
    node_to_grad = {out: [1]}

    for i in reverse_topo_order(out):
        v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])

        for k ∈ inputs(i):
            compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
            append v̄_{k→i} to node_to_grad[k]

    return adjoint of input v̄_input
```

Dictionary that records a list of partial adjoints of each node

Sum up partial adjoints

"Propagates" partial adjoint to its input

Reverse mode AD by extending computational graph

```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
    v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])
    for k ∈ inputs(i):
        compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
        append v̄_{k→i} to node_to_grad[k]
    return adjoint of input v̄_input
```

$i = 2$

node_to_grad: {

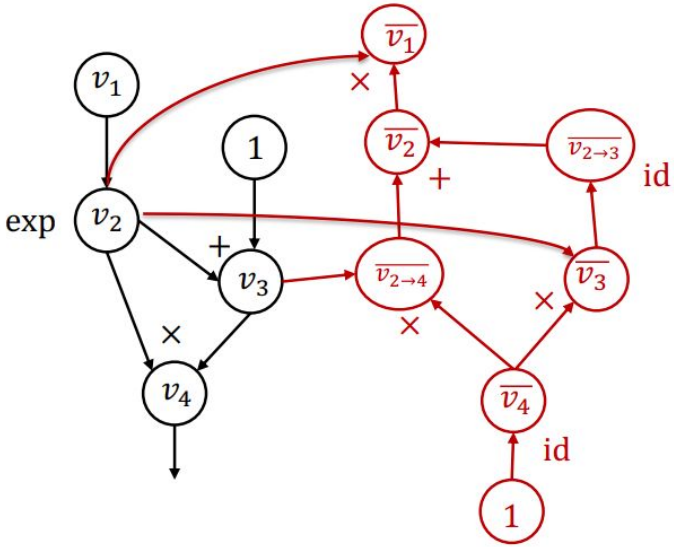
1: $[\overline{v_1}]$

2: $[\overline{v_{2\to4}}, \overline{v_{2\to3}}]$

3: $[\overline{v_3}]$

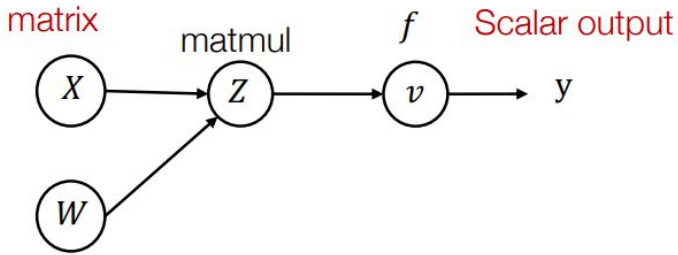
4: $[\overline{v_4}]$

}



NOTE: id is identity function

Reverse mode AD on tensors



Define adjoint for tensor values $\bar{Z} = \begin{bmatrix} \frac{\partial y}{\partial Z_{1,1}} & \cdots & \frac{\partial y}{\partial Z_{1,n}} \\ \cdots & \cdots & \cdots \\ \frac{\partial y}{\partial Z_{m,1}} & \cdots & \frac{\partial y}{\partial Z_{m,n}} \end{bmatrix}$

Forward evaluation trace

$$Z_{ij} = \sum_k X_{ik} W_{kj}$$
$$v = f(Z)$$

Reverse evaluation in scalar form

$$\overline{X_{i,k}} = \sum_j \frac{\partial Z_{i,j}}{\partial X_{i,k}} \bar{Z}_{i,j} = W_{k,j}\, \bar{Z}_{i,j}$$

Forward matrix form

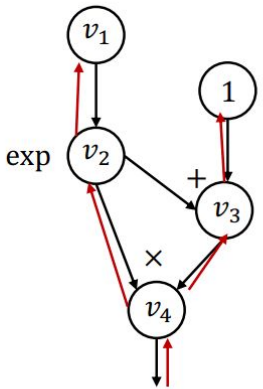
$$Z = XW$$
$$v = f(Z)$$

Reverse matrix form

$$\bar{X} = \bar{Z} W^T$$

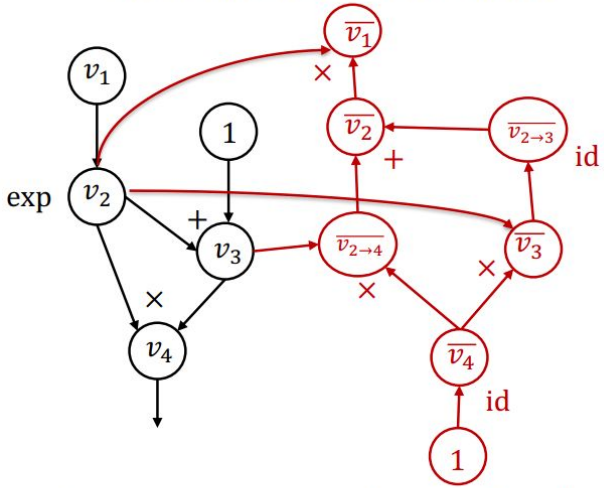
Reverse mode AD vs Backprop

Backprop



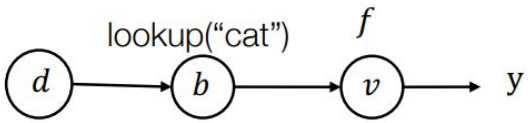
- Run backward operations the same forward graph
- Used in first generation deep learning frameworks (caffe, cuda-convnet)

Reverse mode AD by extending computational graph



- Construct separate graph nodes for adjoints
- Used by modern deep learning frameworks

Reverse mode AD on data structures



Define adjoint data structure

$\bar{d} = \{\text{“cat”}: \frac{\partial y}{\partial a_0}, \text{“dog”}: \frac{\partial y}{\partial 1}\}$

Forward evaluation trace

$d = \{\text{“cat”}: a_0, \text{“dog”}: a_1\}$
$b = d\ [\text{“cat”}]$
$v = f(b)$

Reverse evaluation

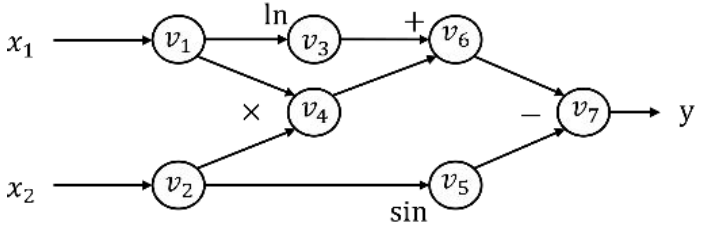
$\bar{b} = \frac{\partial v}{\partial b}\bar{v}$
$\bar{d} = \{\text{“cat”}: \bar{b}\ \}$

Key take away: Define “adjoint value” usually in the same data type as the forward value and adjoint propagation rule. Then the sample algorithm works.

Do not need to support the general form in our framework, but we may support “tuple values”

Forward Automatic mode differentiation

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$



Forward evaluation trace

$v_1 = x_1 = 2$
$v_2 = x_2 = 5$
$v_3 = \ln v_1 = \ln 2 = 0.693$
$v_4 = v_1 \times v_2 = 10$
$v_5 = \sin v_2 = \sin 5 = -0.959$
$v_6 = v_3 + v_4 = 10.693$
$v_7 = v_6 - v_5 = 10.693 + 0.959 = 11.652$
$y = v_7 = 11.652$

Define $\dot{v}_i = \frac{\partial v_i}{\partial x_1}$

We can then compute the $\dot{v}_i$ iteratively in the forward topological order of the computational graph

Forward AD trace

$\dot{v}_1 = 1$
$\dot{v}_2 = 0$
$\dot{v}_3 = \dot{v}_1 / v_1 = 0.5$
$\dot{v}_4 = \dot{v}_1 v_2 + \dot{v}_2 v_1 = 1 \times 5 + 0 \times 2 = 5$
$\dot{v}_5 = \dot{v}_2 \cos v_2 = 0 \times \cos 5 = 0$
$\dot{v}_6 = \dot{v}_3 + \dot{v}_4 = 0.5 + 5 = 5.5$
$\dot{v}_7 = \dot{v}_6 - \dot{v}_5 = 5.5 - 0 = 5.5$

Now we have $\frac{\partial y}{\partial x_1} = \dot{v}_7 = 5.5$

Automatic differentiation: summary

- There is a numerical differentiation. It’s not effective, but you can use it as a simple check
- The classical algorithm is a backpropagation - it uses chain rule to calculate the gradient
- The modern algorithm is a reverse mode AD which construct computational graph for both forward and backward passes. It’s more efficient because of possible node optimization and flexibility (allows to calculate loss w.r.t gradient for free)
- Also, there is a forward AD, but it’s efficient only in a case of low number input and large numbers of outputs.

Homework preparation

Differentials

To understand on how to calculate the gradient, we should recall what differential is.

Вход \ Выход	Скаляр	Вектор	Матрица
Скаляр	$df(x) = f'(x)dx$ ($f'(x)$: скаляр; dx: скаляр)	–	–
Вектор	$df(x) = \langle \nabla f(x), dx \rangle$ ($\nabla f(x)$: вектор; dx: вектор)	$df(x) = J_f(x)dx$ ($J_f(x)$: матрица; dx: вектор)	–
Матрица	$df(X) = \langle \nabla f(X), dX \rangle$ ($\nabla f(X)$: матрица; dX: матрица)	–	–

Note: $\langle A, B \rangle = \sum_{i,j} A_{ij} B_{i,j} = tr(A^T B)$

Matrix-vector differentiation

Conversion Rules

$$
\begin{aligned}
dA &= 0 \\
d(\alpha X) &= \alpha(dX) \\
d(AXB) &= A(dX)B \\
d(X+Y) &= dX + dY \\
d(X^T) &= (dX)^T \\
d(XY) &= (dX)Y + X(dY) \\
d\langle X, Y\rangle &= \langle dX, Y\rangle + \langle X, dY\rangle \\
d\left(\frac{X}{\phi}\right) &= \frac{\phi dX - (d\phi)X}{\phi^2}
\end{aligned}
$$

Standard derivatives table

$$
\begin{aligned}
d\langle A, X\rangle &= \langle A, dX\rangle \\
d\langle Ax, x\rangle &= \langle (A + A^T)x, dx\rangle \\
d\langle Ax, x\rangle &= 2\langle Ax, dx\rangle \qquad (\textbf{если } A = A^T) \\
d(\mathrm{Det}(X)) &= \mathrm{Det}(X)\langle X^{-T}, dX\rangle \\
d(X^{-1}) &= -X^{-1}(dX)X^{-1}
\end{aligned}
$$

Examples of calculating the differential of functions.

Example 1:

$$f(x) = (x^T Ax)(x^T Bx), \quad x \in \mathcal{R}^n, \quad A, B \in \mathcal{R}^{n \times n} \quad A = A^T, B = B^T$$

$$df = d(x^T Ax)(x^T Bx) + (x^T Ax)d(x^T Bx) =$$
$$= 2x^T Adx(x^T Bx) + (x^T Ax)2x^T Bdx$$

Examples of calculating the differential of functions.

Example 2:

$$f(x) = \|x\|_2^3, \quad x \in \mathcal{R}^n$$

$$df = \frac{3}{2}(x^T x)^{\frac{1}{2}} d(x^T x) = \frac{3}{2}(x^T x)^{\frac{1}{2}} 2x^T dx = 3\|x\|_2 x^T dx$$

Examples of calculating the differential of functions.

Example 3:

$$f(X) = det(AXB), \quad X \in \mathcal{R}^{n \times n}$$

$$df = det(AXB)tr((AXB)^{-1}d(AXB)) =$$

$$= det(AXB)tr((AXB)^{-1}AdXB)$$

Then the gradients of the functions from the examples.

Examples 1:

$$
\begin{aligned}
df &= d(x^T Ax)(x^T Bx) + (x^T Ax)d(x^T Bx) = \\
&= 2x^T Adx(x^T Bx) + (x^T Ax)2x^T Bdx = \\
&= 2(x^T Bx)x^T Adx + 2(x^T Ax)x^T Bdx = \\
&= (2(x^T Bx)x^T A + 2(x^T Ax)x^T B)dx = \nabla f(x)^T dx
\end{aligned}
$$

Then: $\nabla f(x) = 2((x^T Bx)xAx + 2(x^T Ax)Bx)$

Then the gradients of the functions from the examples.
Examples 2-3:

$$df = 3\|x\|_2 x^T dx = \nabla f(x)^T dx$$

Then:

$$\nabla f(x) = 3\|x\|_2 x$$

$$df = det(AXB)tr((AXB)^{-1}AdXB) =$$

$$= tr(det(AXB)B(AXB)^{-1}AdX) = tr(\nabla f(X)^T dX)$$

Then:

$$\nabla f(X) = det(AXB)A^T(AXB)^{-T}B^T$$

Hessian Calculation

$$f : U \to V$$

First differential: $df(x)[h]$

Second differential: $d^2 f(x)[h_1, h_2] = d(df(x)[h_1])(x)[h_2]$

If : $U = \mathcal{R}^n,\ V = \mathcal{R}$

Then: $d^2 f(x)[h_1, h_2] = h_1^T \nabla^2 f(x) h_2$

Show the calculation of the Hessian by an example

$$f(x) = \|x\|_2^3 \qquad df = 3\|x\|_2 x^T dx$$

$$\begin{aligned} d^2 f(x)[dx_1, dx_2] &= d\left(3\|x\|_2 x^T dx_1\right) = \\ &= 3d((x^T x)^{\frac{1}{2}})x^T dx_1 + 3(x^T x)^{\frac{1}{2}} d(x^T dx_1) = \\ &= 3\frac{1}{2}(x^T x)^{-\frac{1}{2}} 2x^T dx_2 x^T dx_1 + 3(x^T x)^{\frac{1}{2}} dx_2^T dx_1 = \\ &= dx_1^T x 3(x^T x)^{-\frac{1}{2}} x^T dx_2 + dx_1^T 3(x^T x)^{\frac{1}{2}} I dx_2 \end{aligned}$$

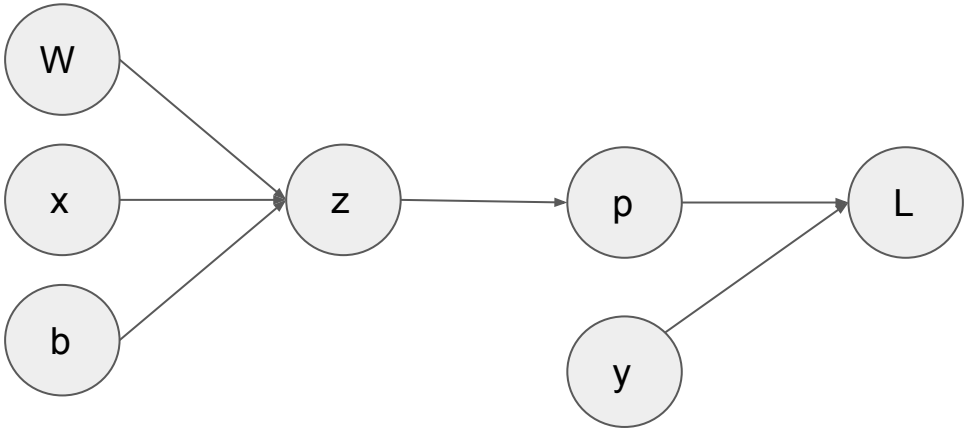
$$\boxed{\nabla^2 f(x) = 3\frac{xx^T}{\|x\|} + 3\|x\| I}$$

Gradient Calculation Examples: Logistic regression

$$z = Wx + b, \quad W \in \mathcal{R}^{K \times D}, \quad b \in \mathcal{R}^{D} \qquad\qquad x \in \mathcal{R}^{D}, \quad y \in \{1, 2, \ldots, K\}$$

$$p = softmax(z) = \frac{\exp(z)}{\sum_j \exp(z_j)}$$

$$L(p, y) = -\log p_y = -\log p^T \mathbf{1}_y, \quad \mathbf{1}_y = [0, \ldots, 0, \overset{y}{1}, 0, \ldots, 0]$$



Gradient Calculation Examples: Logistic regression

$$dL = d(-\log p^T \mathbf{1}_y) = -\frac{1}{p^T \mathbf{1}_y} d(p^T \mathbf{1}_y) = -\frac{1}{p^T \mathbf{1}_y} \mathbf{1}_y d(p)$$

then:

$$\nabla_p L = -\frac{1}{p^T \mathbf{1}_y} \mathbf{1}_y$$

Next, we need to calculate how p and z are related:

$$p = \frac{\exp(z)}{\exp(z)^T \mathbf{1}}$$

Gradient Calculation Examples: Logistic regression

$$dp = \frac{d(\exp(z))\exp(z)^T\mathbf{1} - \exp(z)d(\exp(z)^T\mathbf{1})}{(\exp(z)^T\mathbf{1})^2} =$$

$$= \frac{diag(\exp(z))dz\exp(z)^T\mathbf{1} - \exp(z)\mathbf{1}^T diag(\exp(z))dz}{(\exp(z)^T\mathbf{1})^2}$$

$$dL = -\frac{1}{p^T\mathbf{1}_y}\mathbf{1}_y dp =$$

$$= -\frac{1}{p^T\mathbf{1}_y}\frac{1}{\exp(z)^T\mathbf{1}}\mathbf{1}_y^T diag(\exp(z))dz + \frac{1}{p^T\mathbf{1}_y}\frac{\mathbf{1}_y\exp(z)\mathbf{1}_y^T diag(\exp(z))dz}{(\exp(z)^T\mathbf{1})^2}$$

Gradient Calculation Examples: Logistic regression

$$\nabla_z L = -\frac{1}{p^T \mathbf{1}_y} \frac{1}{\exp(z)^T \mathbf{1}} \mathbf{1}_y^T diag(\exp(z)) + \frac{1}{p^T \mathbf{1}_y} \frac{\mathbf{1}_y \exp(z) \mathbf{1}_y^T diag(\exp(z))}{(\exp(z)^T \mathbf{1})^2}$$

$$z = Wx + b$$

$$dz = dWx + Wdx + db$$

$$dL = \nabla_z L^T dz = \nabla_z L^T (dWx + Wdx + db) = tr(x \nabla_z L^T dW) + \nabla_z L^T W dx + \nabla_z L^T db$$

Then: $\nabla_W L = \nabla_z L x^T$ $\nabla_x L = W^T \nabla_z L$ $\nabla_b L = \nabla_z L$

Homework 1

Deadline: 19 september 23:59

Simple practicing in gradient calculation + backpropagation

Recap

- Multi-layer perceptron
  - Activations
  - Properties

- Gradient calculation
  - How to calculate gradient?