

RĪGAS TEHNISKĀ UNIVERSITĀTE
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts
Mākslīgā intelekta un sistēmu inženierijas katedra

Oļegs Korsaks
Bakalaura studiju programmas „Datorsistēmas”
students, stud. apl. nr. 051RDB146

**SALIDZINOŠĀ ANALĪZE DATU
KOPU FORMĀTIEM PYTORCH
ATTĒLU KLASIFIKĀCIJAS
UZDEVUMIEM**

Atskaite par bakalaura darbu

Zinātniskais vadītājs
Mg.sc.ing, Pētnieks
ĒVALDS URTĀNS

Rīga 2021

Table of Contents

1. Ievads.....	3
1.1. Dziļā māšīnmācīšanās.....	3
Pamata arhitektūras.....	3
Linārie slāņi.....	3
Aktivizācijas funkcijas.....	3
Kļūdas funkcijas.....	3
MAE.....	3
MSE.....	3
CCE.....	3
BCE.....	3
Atpakaļizplatīšanās algoritms.....	3
Apmācām Parametru Optizācijas algoritmi.....	3
Metrikas.....	4
1.2. Attēlu klasifikācija.....	4
1.3. PyTorch vide.....	4
2. Metodoloģija.....	4
2.1. Datu ielādes metodes.....	4
2.1.1. Failu sistēma.....	4
2.1.2. Linux mmap.....	4
2.1.3. nVidia mmap.....	4
2.1.4. HDF5.....	4
2.1.5. PostgreSQL.....	4
2.2. Datu kopas.....	4
2.2.1. CIFAR10.....	4
2.2.2. Tiny ImageNet.....	4
2.4. Apmācības protokols.....	4
3. Rezultāti.....	5
4. Tālākie pētījumi.....	5
5. Secinājumi.....	5
Python introduction.....	6
Asteroids game.....	7
Robot arm.....	9
House prices.....	11
Backpropagation.....	15
LeakyReLU task.....	17
NumPy + OOP version.....	20

1. Ievads

1.1. Dziļā māšīnmācīšanās

Pamata arhitektūras

Linārie slāņi

Aktivizācijas funkcijas

Softmax (priekš klasifikācijas)

Kļūdas funkcijas

Kļūdas funkcija palīdz noteikt cik tālu tekoša prognozēta vērtība ir no patiesas. Un ja to pielietot visiem datu eksemplāriem – ar to var noteikt, cik labi tekošais modelis var prognozēt rezultātus kopumā. Ideālā gadījumā kļūdai jābūt vienāgai nullei, gan apmācības datu eksemplāriem, gan pārbaudes. Tātad apmācība cenšas kļūdu samazināt.

MAE

Mean absolute error vai vidēja absolūta kļūda:

$$L_{MAE} = \frac{1}{N} \cdot \sum_{i=0}^N |(h_{\theta}(x_i) - y_i)|$$

Tā ir vidēja absolūta starpība starp pareizas un prognozētas vērtībām. Kļūda pieaug lineāri un tiek pielietota regresijas uzdevumiem, kuru rezultāts ir viena vērtība.

MSE

Mean squared error vai vidēja kvadrātiskā kļūda:

$$L_{MSE} = \frac{1}{N} \cdot \sum_{i=0}^N (h_{\theta}(x_i) - y_i)^2$$

Tā ir vidēja starpība starp pareizas un prognozētas vērtībām, kas tiek pacelta kvadrātā. Kvadrāts palīdz izvairīties no negatīvām vērtībām. Kā arī kļūdas vērtība pieaug straujāk, salīdzinot ar MAE. Tiek pielietota regresijas uzdevumiem.

CCE

BCE

Atpakaļizplatīšanās algoritms

(grafiski un ar vienādojumiem)

Apmācām Parametru Optizācijas algoritmi

SGD

Metrikas

novērtēt cik labs rezultāts Accuracy, F1

1.2. Attēlu klasifikācija

ConvNets

ResNet / DenseNet

1.3. PyTorch vide

Modeļu implementāciju <> viendājumi

Datu ielādes process (DataSet)

2. Metodoloģija

2.1. Datu ielādes metodes

2.1.1. Failu sistēma

2.1.2. Linux mmap

2.1.3. nVidia mmap

2.1.4. HDF5

2.1.5. PostgreSQL

2.2. Datu kopas

2.2.1. CIFAR10

2.2.2. Tiny ImageNet

.. vel kādas

^ Atrast HiRes datu kopas priekš klasifikācijas

^ Atrast dažādu izmēru attēlu datu kopas

^ 3 dažādu izmēru / konfigurāciju datu kopas

2.4. Apmācības protokols

* Jāizdomā metrikas kā noteikt ietekmi datu ielādes metodēm

* Vidējais epocha ātrums sekundēs

3. Rezultāti

Datu ielādes metodes PRET datu kopām

4. Tālākie pētījumi

5. Secinājumi

Python introduction

My power function

```
from decimal import Decimal
from functools import reduce
from itertools import repeat
from operator import mul
from typing import Union

def my_pow(number: Union[int, float, Decimal], power: int):
    if power > 0:
        return reduce(mul, repeat(number, power))
    elif power == 0:
        return 1

    return 1 / reduce(mul, repeat(number, abs(power)))
```

Matrix dot product:

```
import numpy as np

def dot(a, b):
    try:
        a_height, a_width = np.shape(a)
    except ValueError:
        a_height, a_width = np.shape(a)[0], 1

    try:
        b_height, b_width = np.shape(b)
    except ValueError:
        b_height, b_width = np.shape(b)[0], 1

    if a_width != b_height:
        raise ValueError(f'Wrong shape of matrix: {np.shape(a)=} {np.shape(b)=}')

    c = np.array(
        tuple(
            sum(x * y for x, y in zip(a[row_idx], b[:, col_idx]))
            for row_idx in range(a_height)
            for col_idx in range(b_width)
        )
    )

    return c.reshape((a_height, b_width))
```

Asteroids game

Implemented helper functions:

```
def rotation_mat(degrees: float):
    """
    Rotating around Z axis
    :param degrees:
    :return:
    """
    theta = np.radians(degrees)
    c = np.cos(theta)
    s = np.sin(theta)

    return np.array([
        [c, -s, 0.0],
        [s, c, 0.0],
        [0.0, 0.0, 1.0],
    ])

def translation_mat(dx: float, dy: float):
    return np.array([
        [1.0, 0.0, dx],
        [0.0, 1.0, dy],
        [0.0, 0.0, 1.0],
    ])

def scale_mat(sx: float, sy: float):
    return np.array([
        [sx, 0.0, 0.0],
        [0.0, sy, 0.0],
        [0.0, 0.0, 1.0],
    ])

def dot(a, b):
    try:
        a_height, a_width = np.shape(a)
    except ValueError:
        a_height, a_width = np.shape(a)[0], 1

    try:
        b_height, b_width = np.shape(b)
    except ValueError:
        b_height, b_width = np.shape(b)[0], 1

    if a_width != b_height:
        raise ValueError(f'Wrong shape of matrix: {np.shape(a)=} {np.shape(b)=}')

    c = np.array(
        tuple(
            sum(x * y for x, y in zip(a[row_idx], b[:, col_idx]))
            for row_idx in range(a_height)
        )
    )
```

```
        for col_idx in range(b_width)
    )
)

return c.reshape((a_height, b_width))

def vec2d_to_vec3d(vec2d):
    i = np.array((
        (1.0, 0.0),
        (0.0, 1.0),
        (0.0, 0.0),
    ))

    return dot(i, vec2d[:, None]).transpose()[0] + np.array([0.0, 0.0, 1.0])

def vec3d_to_vec2d(vec3d):
    i = np.array((
        (1.0, 0.0, 0.0),
        (0.0, 1.0, 0.0),
    ))

    return dot(i, vec3d[:, None])
```


Robot arm

1) Implement 3-segment robot arm.

I've implemented a dynamic N-segment robot arm by slightly prettifying initial code.

```
def rotation(theta: float):  
    """  
    :param theta: in radians  
    :return:  
    """  
    c = np.cos(theta)  
    s = np.sin(theta)  
  
    return np.array((  
        (c, -s),  
        (s, c),  
    ))
```

```
def d_rotation(theta: float):  
    """  
    :param theta: in radians  
    :return:  
    """  
    c = np.cos(theta)  
    s = np.sin(theta)  
  
    return np.array((  
        (-s, -c),  
        (c, -s),  
    ))
```

2) Implemented multi-segment robot arm:

```
prev_r = None  
  
for segment_idx in range(SEGMENT_COUNT):  
    # getting rotation value  
    theta = thetas[segment_idx]  
    # getting rotation matrix  
    r = rotation(theta)  
    dr_theta_1 = d_rotation(theta)  
    # calculating current segment vector by adding rotated segment template to the tip of the previous segment  
    np_joints[segment_idx+1] = np.dot(r, segment) + np_joints[segment_idx]  
  
    # STILL BLACK MAGIC FOR ME  
    x = dr_theta_1 @ segment  
  
    if segment_idx:  
        x = prev_r @ x  
  
    # is this somehow related to derivative of the loss function?  
    d_theta_1 = np.sum(x * -2 * (TARGET_POINT - np_joints[-1]))
```

```
# END OF BLACK MAGIC

# updating and storing new rotation value for the current segment
thetas[segment_idx] -= d_theta_1 * LEARNING_RATE

prev_r = r

loss = np.sum((TARGET_POINT - np_joints[-1])** 2)
plt.title(f'loss: {loss:.4f} thetas: {tuple(round(np.rad2deg(theta)) for theta in thetas)}')
```

3) Loss of MSE from tip of robot arm (last vector) to target point:

```
loss = np.sum((TARGET_POINT - np_joints[-1])** 2)
```

House prices

1) Implement by hand equations of derivative using LaTeX (I will use similar format in LibreOffice/ODF, sorry)

$$f_x = \frac{\partial f}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h, y) - f(x, y)}{h}$$

$$f_y = \frac{\partial f}{\partial y} = \lim_{h \rightarrow 0} \frac{f(x, y+h) - f(x, y)}{h}$$

2) Writing equations for housings:

$$dw \text{ linear} : \frac{\partial}{\partial W} [W \cdot x + b] = x$$

$$dx \text{ linear} : \frac{\partial}{\partial x} [W \cdot x + b] = W$$

$$db \text{ linear} : \frac{\partial}{\partial b} [W \cdot x + b] = 1$$

3) Cost function

We will use mean squared error cost function:

$$J(\theta_0, \theta_1) = L_{MSE} = \frac{1}{N} \cdot \sum_{i=0}^N (h_{\theta}(x_i) - y_i)^2$$

where $h_{\theta}(x_i) = \theta_0 + \theta_1 \cdot x_i$ (is our model)

We need to minimize cost function's result.

4) Gradient descent

The goal of this is to update θ_0, b and θ_1, W to minimize cost function result.

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$b := b - \alpha \cdot \frac{\partial}{\partial b} J(b, W)$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$W := W - \alpha \cdot \frac{\partial}{\partial W} J(b, W)$$

but to simultaneously update both variables we need to assign them to temporary variables first, so b doesn't affect calculation of the W.

α - is a learning rate, which defines how fast we are going to change b and W.

By intuition what is going to happen – partial derivative of the cost function will be a positive slope (will return a positive number) if the mse will on the right side of the local minima, as a result we will subtract a slope value multiplied by learning rate from W (or b). Otherwise we will add it. In ideal situation once MSE is 0 – we don't move anymore.

So let's try to figure out partial derivatives:

For θ_0 or b :

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \cdot \frac{1}{N} \cdot \sum_{i=0}^N (h_{\theta}(x_i) - y_i)^2 = \frac{\partial}{\partial \theta_0} \cdot \frac{1}{N} \cdot \sum_{i=0}^N (\theta_0 + \theta_1 x_i - y_i)^2 = \frac{2}{N} \sum_{i=0}^N (\theta_0 + \theta_1 x_i - y_i) \cdot 1$$

equals to:

$$\frac{2}{N} \sum_{i=0}^N (\theta_0 + \theta_1 x_i - y_i) \cdot 1 = \frac{2}{N} \sum_{i=0}^N (b + W \cdot x_i - y_i) \cdot 1$$

and for θ_1 or W :

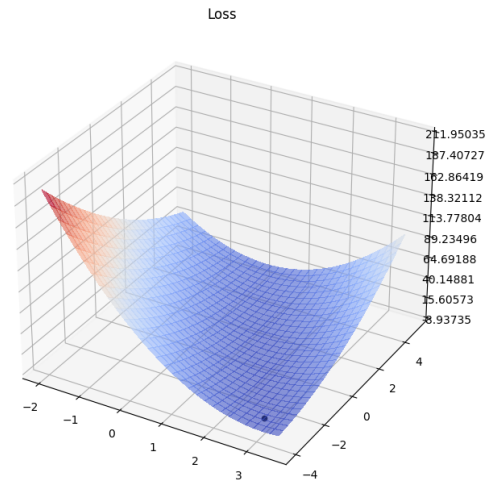
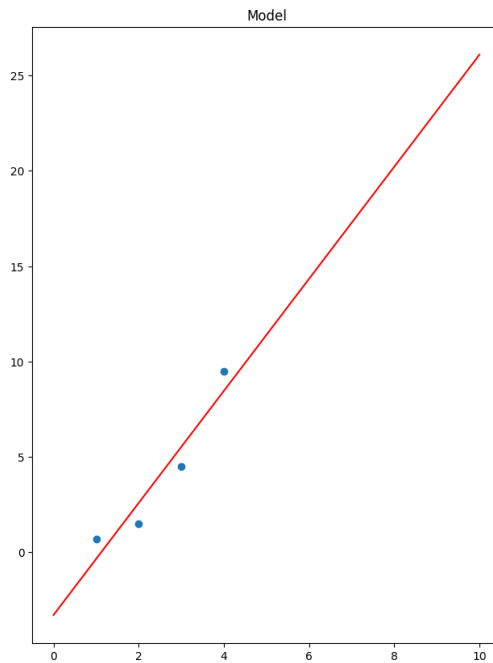
$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \cdot \frac{1}{N} \cdot \sum_{i=0}^N (h_{\theta}(x_i) - y_i)^2 = \frac{\partial}{\partial \theta_1} \cdot \frac{1}{N} \cdot \sum_{i=0}^N (\theta_0 + \theta_1 x_i - y_i)^2 = \frac{2}{N} \sum_{i=0}^N (\theta_0 + \theta_1 x_i - y_i) \cdot x_i$$

equals to:

$$\frac{2}{N} \sum_{i=0}^N (\theta_0 + \theta_1 x_i - y_i) \cdot x_i = \frac{2}{N} \sum_{i=0}^N (b + W \cdot x_i - y_i) \cdot x_i$$

For linear model:

w=2.939305864784624 b=-3.2979591578193053 loss=1.103000695674468 learning_rate=0.0001490000000000007



For sigmoid model:

$$\frac{1}{1 + e^{-x}}$$

We need to find a gradient descent:

$$\theta_0 := \theta_0 - \alpha \cdot \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$b := b - \alpha \cdot \frac{\partial}{\partial b} J(b, W)$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$W := W - \alpha \cdot \frac{\partial}{\partial W} J(b, W)$$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \cdot \frac{1}{N} \cdot \sum_{i=0}^N (h_{\theta}(x_i) - y_i)^2$$

where $h_{\theta}(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 \cdot x)}} = \frac{1}{1+e^{-(b+W \cdot x)}}$

let's substitute $a = b + W \cdot x$ so $h_{\theta}(x) = \frac{1}{1+e^{-a}}$

$$\frac{\partial}{\partial a} \frac{1}{1+e^{-a}} = \frac{\partial}{\partial a} (1+e^{-a})^{-1} = -(1+e^{-a})^{-2} \cdot \frac{\partial}{\partial a} (1+e^{-a}) = -(1+e^{-a})^{-2} \cdot \left(\frac{\partial}{\partial a} 1 + \frac{\partial}{\partial a} e^{-a} \right) =$$

$$= -(1+e^{-a})^{-2} \cdot (0 + e^{-a} \cdot \frac{\partial}{\partial a} [-a]) = -(1+e^{-a})^{-2} \cdot (e^{-a} \cdot -1) = \frac{e^{-a}}{(1+e^{-a})^2} =$$

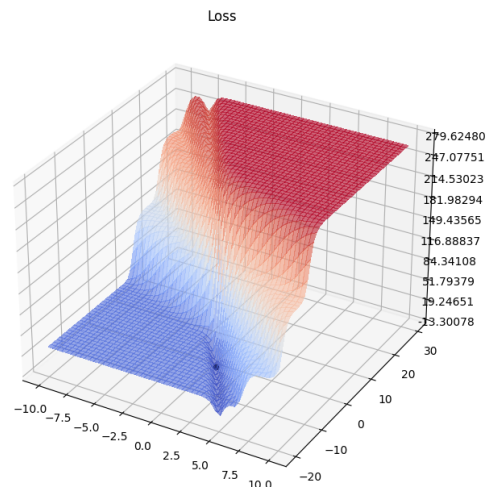
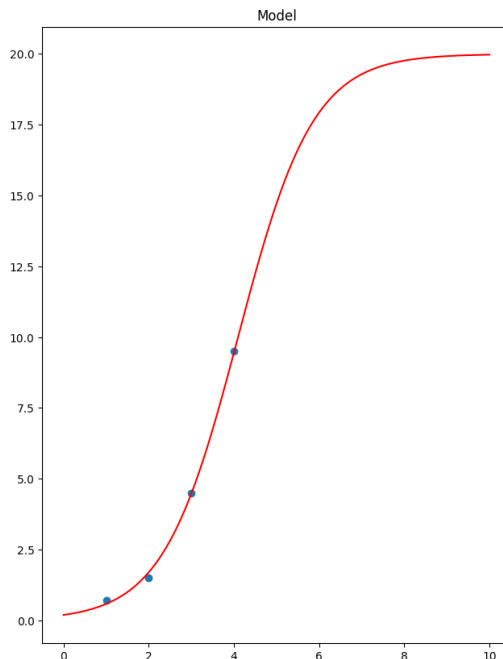
$$= \frac{e^{-a}}{(1+e^{-a}) \cdot (1+e^{-a})} = \frac{1 \cdot e^{-a}}{(1+e^{-a}) \cdot (1+e^{-a})} = \frac{1}{1+e^{-a}} \cdot \frac{e^{-a}}{1+e^{-a}} =$$

$$= \frac{1}{1+e^{-a}} \cdot \frac{e^{-a} + 1 - 1}{1+e^{-a}} = \frac{1}{1+e^{-a}} \cdot \left(\frac{1+e^{-a}}{1+e^{-a}} - \frac{1}{1+e^{-a}} \right) = \frac{1}{1+e^{-a}} \cdot \left(1 - \frac{1}{1+e^{-a}} \right)$$

$$\frac{\partial}{\partial \theta_0} \frac{1}{1+e^{-a}} = \frac{\partial}{\partial b} \frac{1}{1+e^{-a}} = \frac{\partial}{\partial a} \cdot \frac{\partial}{\partial b} = \frac{e^{-a}}{(1+e^{-a})^2} \cdot 1 = \frac{1}{1+e^{-a}} \cdot \left(1 - \frac{1}{1+e^{-a}} \right) \cdot 1$$

$$\frac{\partial}{\partial \theta_1} \frac{1}{1+e^{-a}} = \frac{\partial}{\partial W} \frac{1}{1+e^{-a}} = \frac{\partial}{\partial a} \cdot \frac{\partial}{\partial W} = \frac{e^{-a}}{(1+e^{-a})^2} \cdot x = \frac{1}{1+e^{-a}} \cdot \left(1 - \frac{1}{1+e^{-a}} \right) \cdot x$$

w=1.1346469402126882 b=-4.6482628586296135 loss=0.014018127818400896 learning_rate=0.0001490000000000007



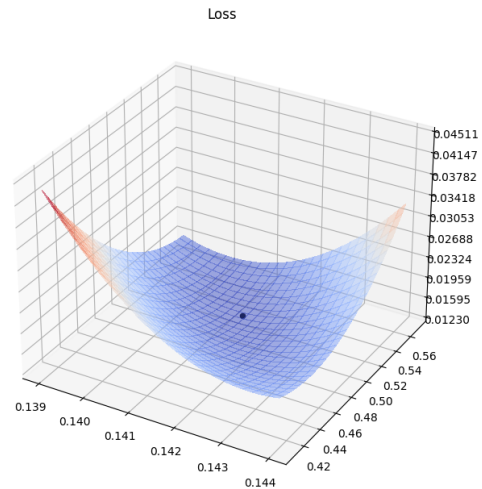
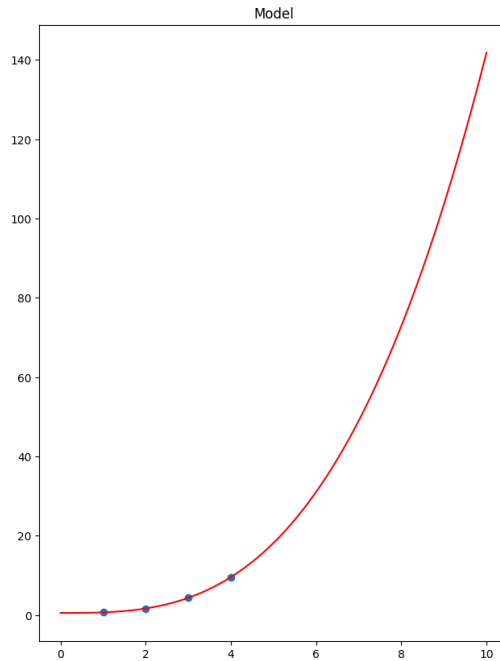
Animated version: <https://www.youtube.com/watch?v=4hFCo9tbU34>

It looks cool but I don't find sigmoid logical here because there is no chance that 10 floor building may cost same as 100 floor. What about cubic $b + W \cdot x^3$? Where b would shift the line vertically and

W would regulate its width. That was my intuition on how to fit the line. The nice part here is that loss function derivative dW, db is the same as for linear. And dx is:

$$dx_{cubic} : \frac{\partial}{\partial x} [W \cdot x^3 + b] = W \cdot 3 \cdot x^2$$

w=0.141346153846152 b=0.5163461538462171 loss=0.013832747781064567 learning_rate=0.001485999999999995



And it worked. That's much better.

Backpropagation

$$y' = M(x) = \text{Linear}(W_1, b_1, \text{Linear}(W_2, b_2, x)) = \text{Linear}(W_2, b_2, \text{ReLU}(\text{Linear}(W_1, b_1, x)))$$

$$\text{Linear}(W_i, b_i, x_i) = W_i \cdot x_i + b_i$$

$$\text{ReLU}(x_i) = \begin{cases} x_i, & x_i \geq 0 \\ 0, & x_i < 0 \end{cases}$$

$$\text{MAE}(y', y) = \frac{1}{N} \sum_{i=1}^N (y_i - y_i')$$

SGD:

$$W_i' = W_i - \alpha \cdot \frac{\text{MAE}(y, W_1, b_1, W_2, b_2, x)}{\partial W_i}$$

$$b_i' = b_i - \alpha \cdot \frac{\text{MAE}(y, W_1, b_1, W_2, b_2, x)}{\partial b_i}$$

$$\frac{\text{MAE}(y, M(x))}{\partial W_i} = \frac{|y - M(x)|}{\partial W_i}$$

Let's assume: $a = y - M(x)$

$$\text{Then: } \frac{|a|}{\partial a} = \frac{\sqrt{a}}{\partial a} = \frac{(a^2)^{\frac{1}{2}}}{\partial a} = \frac{1}{2} \cdot (a^2)^{-\frac{1}{2}} \cdot \frac{a^2}{\partial a} = \frac{1}{2} \cdot (a^2)^{-\frac{1}{2}} \cdot 2a = a \cdot (a^2)^{-\frac{1}{2}} = a \cdot \frac{1}{|a|} = \frac{a}{|a|} = \frac{y - M(x)}{|y - M(x)|}$$

$$\frac{\text{Linear}(W_i, b_i, x)}{\partial W_i} = \frac{W_i \cdot x + b_i}{\partial W_i} = x$$

$$\frac{\text{Linear}(W_i, b_i, x)}{\partial b_i} = \frac{W_i \cdot x + b_i}{\partial b_i} = 1$$

$$\frac{\text{MAE}(y, W_1, b_1, W_2, b_2, x)}{\partial W_2} = \frac{|a|}{\partial a} \cdot \frac{M(x)}{\partial W_2} = \frac{y - M(x)}{|y - M(x)|} \cdot \frac{\text{Linear}(W_2, b_2, \text{ReLU}(\text{Linear}(W_1, b_1, x)))}{\partial W_2} =$$

$$= \frac{y - M(x)}{|y - M(x)|} \cdot \text{ReLU}(\text{Linear}(W_1, b_1, x))$$

$$\frac{\text{MAE}(y, W_1, b_1, W_2, b_2, x)}{\partial b_2} = \frac{|a|}{\partial a} \cdot \frac{M(x)}{\partial b_2} = \frac{y - M(x)}{|y - M(x)|} \cdot \frac{\text{Linear}(W_2, b_2, \text{ReLU}(\text{Linear}(W_1, b_1, x)))}{\partial b_2} =$$

$$= \frac{y - M(x)}{|y - M(x)|} \cdot 1$$

$$M(x) = \text{Linear}(W_2, b_2, \text{ReLU}(\text{Linear}(W_1, b_1, x)))$$

$$z = \text{ReLU}(q)$$

$$q = \text{Linear}(W_1, b_1, x)$$

$$M(x) = \text{Linear}(W_2, b_2, z) = W_2 \cdot z + b_2$$

$$\begin{aligned}
\frac{MAE(y, W_1, b_1, W_2, b_2, x)}{\partial W_1} &= \frac{|a|}{\partial a} \cdot \frac{M(x)}{\partial W_1} = \frac{|a|}{\partial a} \cdot \frac{Linear(W_2, b_2, z)}{\partial W_1} = \frac{|a|}{\partial a} \cdot \frac{Linear(W_2, b_2, z)}{\partial z} \cdot \frac{z}{\partial W_1} = \\
&= \frac{|a|}{\partial a} \cdot \frac{W_2 \cdot z + b_2}{\partial z} \cdot \frac{z}{\partial W_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(Linear(W_1, b_1, x))}{\partial W_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot \frac{q}{\partial W_1} = \\
&= \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot \frac{Linear(W_1, b_1, x)}{\partial W_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot \frac{W_1 \cdot x + b_1}{\partial W_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot x = \\
&= \frac{y - M(x)}{|y - M(x)|} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot x
\end{aligned}$$

$$\frac{ReLU(q)}{\partial q} = \begin{cases} 1, q \geq 0 \\ 0, q < 0 \end{cases}$$

$$\frac{ReLU(W_1 \cdot x + b_1)}{\partial [W_1 \cdot x + b_1]} = \begin{cases} 1, W_1 \cdot x + b_1 \geq 0 \\ 0, W_1 \cdot x + b_1 < 0 \end{cases}$$

$$\begin{aligned}
\frac{MAE(y, W_1, b_1, W_2, b_2, x)}{\partial b_1} &= \frac{|a|}{\partial a} \cdot \frac{M(x)}{\partial b_1} = \frac{|a|}{\partial a} \cdot \frac{Linear(W_2, b_2, z)}{\partial b_1} = \frac{|a|}{\partial a} \cdot \frac{Linear(W_2, b_2, z)}{\partial z} \cdot \frac{z}{\partial b_1} = \\
&= \frac{|a|}{\partial a} \cdot \frac{W_2 \cdot z + b_2}{\partial z} \cdot \frac{z}{\partial b_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(Linear(W_1, b_1, x))}{\partial b_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot \frac{q}{\partial b_1} = \\
&= \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot \frac{Linear(W_1, b_1, x)}{\partial b_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot \frac{W_1 \cdot x + b_1}{\partial b_1} = \frac{|a|}{\partial a} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q} \cdot 1 = \\
&= \frac{y - M(x)}{|y - M(x)|} \cdot W_2 \cdot \frac{ReLU(q)}{\partial q}
\end{aligned}$$

LeakyReLU task

Model: $y' = M(x) = \text{LeakyReLU}(\text{Linear}(\tanh(\text{Linear}(W \cdot x + b))))$

$$\begin{aligned} y' = M(x) &= \text{LeakyReLU}(\text{Linear}(W_1, b_1, W_2, b_2, x), \alpha) = \\ &= \text{LeakyReLU}(\text{Linear}(W_2, b_2, \tanh(\text{Linear}(W_1, b_1, x))), \alpha) \end{aligned}$$

Where:

$$\text{Linear}(x) = W \cdot x + b$$

$$dw_{\text{linear}}: \frac{\partial}{\partial W} [W \cdot x + b] = x$$

$$dx_{\text{linear}}: \frac{\partial}{\partial x} [W \cdot x + b] = W$$

$$db_{\text{linear}}: \frac{\partial}{\partial b} [W \cdot x + b] = 1$$

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

$$\begin{aligned} \frac{\tanh(x)}{\delta x} &= \delta x \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{(e^x - e^{-x}) \cdot (e^x + e^{-x})^{-1}}{\delta x} = \frac{(e^x - e^{-x})}{\delta x} \cdot (e^x + e^{-x})^{-1} + (e^x - e^{-x}) \cdot \frac{(e^x + e^{-x})^{-1}}{\delta x} = \\ &= \frac{(e^x - e^{-x})}{\partial(e^x - e^{-x})} \cdot \frac{(e^x - e^{-x})}{\partial x} \cdot (e^x + e^{-x})^{-1} + (e^x - e^{-x}) \cdot \frac{(e^x + e^{-x})^{-1}}{\partial(e^x + e^{-x})} \cdot \frac{(e^x + e^{-x})^{-1}}{\partial x} = \\ &= (e^x + e^{-x}) \cdot (e^x + e^{-x})^{-1} - (e^x - e^{-x}) \cdot (e^x + e^{-x})^{-2} \cdot (e^x - e^{-x}) = 1 - (e^x - e^{-x})^2 \cdot (e^x + e^{-x})^{-2} \end{aligned}$$

$$\text{LeakyReLU}(x) = \begin{cases} x, & x > 0 \\ \alpha \cdot x, & x \leq 0 \end{cases} \quad \text{here } \alpha \text{ is a [slope](#), not the learning rate}$$

$$\frac{\text{LeakyReLU}(x)}{\partial x} = \begin{cases} 1, & x > 0 \\ \alpha, & x \leq 0 \end{cases}$$

MAE loss function:

$$\text{MAE}(y', y) = \frac{1}{N} \sum_{i=1}^N (y_i - y'_i)$$

SGD:

$$W'_i = W_i - \alpha \cdot \frac{\text{MAE}(y, W_1, b_1, W_2, b_2, x)}{\partial W_i}$$

$$b'_i = b_i - \alpha \cdot \frac{\text{MAE}(y, W_1, b_1, W_2, b_2, x)}{\partial b_i}$$

$$\begin{aligned} y' = M(x) &= \text{LeakyReLU}(\text{Linear}(W_1, b_1, W_2, b_2, x), \alpha) = \\ &= \text{LeakyReLU}(\text{Linear}(W_2, b_2, \tanh(\text{Linear}(W_1, b_1, x))), \alpha) \end{aligned}$$

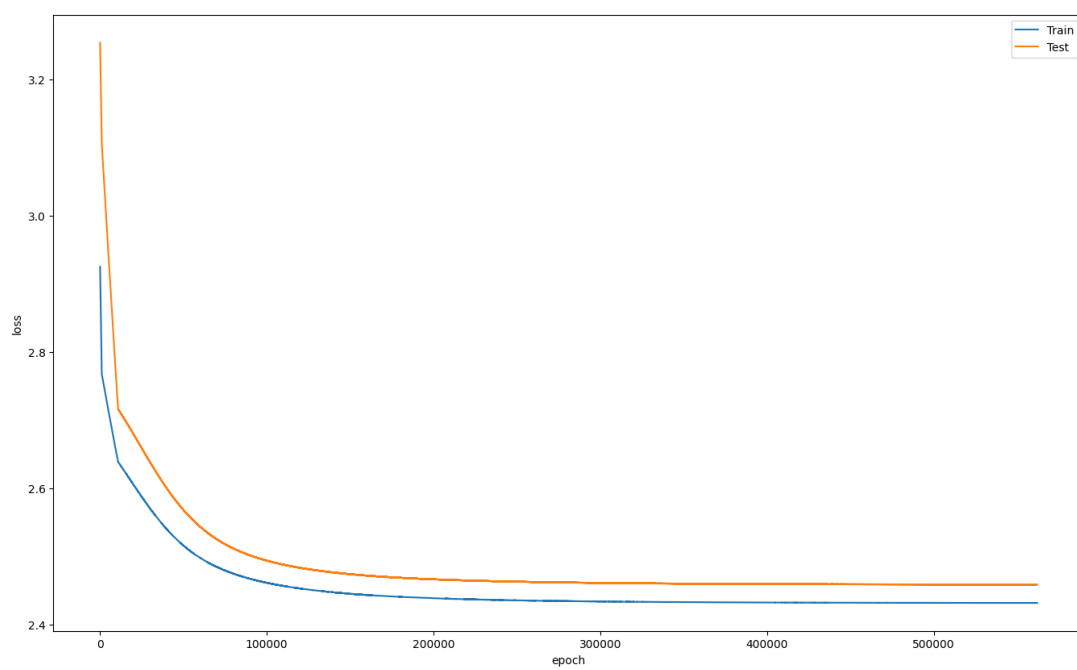
$$\begin{aligned}
m &= \text{Linear}(W_1, b_1, x) \\
k &= \text{Linear}(W_2, b_2, \tanh(\text{Linear}(W_1, b_1, x))) \\
l &= \tanh(\text{Linear}(W_1, b_1, x))
\end{aligned}$$

$$\begin{aligned}
\frac{y'}{\partial W_2} &= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{k}{\partial W_2} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial W_2} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot l = \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \tanh(\text{Linear}(W_1, b_1, x)) \\
\frac{y'}{\partial b_2} &= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{k}{\partial b_2} = \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial b_2} = \frac{\text{LeakyReLU}(k)}{\partial k} \cdot 1
\end{aligned}$$

$$\begin{aligned}
\frac{y'}{\partial W_1} &= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{k}{\partial W_1} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial W_1} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial l} \cdot \frac{l}{\partial W_1} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(\text{Linear}(W_1, b_1, x))}{\partial W_1} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot \frac{m}{\partial W_1} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot \frac{\text{Linear}(W_1, b_1, x)}{\partial W_1} = \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot x
\end{aligned}$$

$$\begin{aligned}
\frac{y'}{\partial b_1} &= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{k}{\partial b_1} = \\
&\quad \dots \text{same as for } W_1 \dots \\
&= \frac{\text{LeakyReLU}(k)}{\partial k} \cdot \frac{\text{Linear}(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot 1
\end{aligned}$$

Running this model produces following result:



NumPy + OOP version

1) Implement dataset normalization to get X and Y features in range from -1..1.

$$X_i = 2 \cdot \left(\frac{X_i - \min(X_i)}{\max(X_i) - \min(X_i)} - 0.5 \right)$$

Need also function to convert Y back to real values.

Solution: To convert values back – we need to remember min and max values of initial dataset, otherwise we don't know according to what values we have -1 and 1 boundaries.

```
def normalize(values: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    max_values = np.max(values, axis=0)
    min_values = np.min(values, axis=0)

    return 2.0 * ((values - min_values) / (max_values - min_values) - 0.5), min_values, max_values

def denormalize(values: np.ndarray, min_values: np.ndarray, max_values: np.ndarray) -> np.ndarray:
    return (values / 2.0 + 0.5) * (max_values - min_values) + min_values
```

2) Implement model with new functions

- Use code from 4. (?) task
- Add classes LossMSE (Mean square error loss function), LayerSigmoid

```
class SigmoidLayer:
    def __init__(self):
        self.x = None
        self.output = None

    def forward(self, x: Variable) -> Variable:
        self.x = x
        self.output = Variable(
            1.0 / (1.0 + np.exp(-x.value))
        )
        return self.output

    def backward(self):
        self.x.grad = -1.0 / (1.0 + np.exp(-self.x.value)) ** 2 * self.output.grad
```

```
class MSELoss:
    def __init__(self):
        self.y: Optional[Variable] = None
        self.y_prim: Optional[Variable] = None

    def forward(self, y: Variable, y_prim: Variable) -> float:
        self.y = y
        self.y_prim = y_prim
        return np.mean((y.value - y_prim.value) ** 2)

    def backward(self):
        self.y_prim.grad = 2.0 * (self.y_prim.value - self.y.value)
```

- Replace ReLU with Sigmoid in Model

We will use same Sigmoid formulas:

$$\frac{1}{1+e^{-x}}$$

$$\frac{\partial}{\partial x} \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

- Train with LossMSE

And same MSE cost function:

$$L_{MSE} = \frac{1}{N} \cdot \sum_{i=0}^N (h_{\theta}(x_i) - y_i)^2$$

$$\frac{\partial}{\partial L_{MSE}} = \frac{2}{N} \sum_{i=0}^N (b + W \cdot x_i - y_i)$$

- Fine tune Hyper parameters so you can get lowest error in 300 epochs

Resulting model layers are:

```
self.layers = [
    LinearLayer(in_features=8, out_features=4),
    SigmoidLayer(),
    LinearLayer(in_features=4, out_features=4),
    SigmoidLayer(),
    LinearLayer(in_features=4, out_features=1)
]
```