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

1.1. Dziļā mašīnmācīšanās

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 vienādai nullei, gan apmācības datu eksemplāriem, gan pārbaudes. Tātad apmācība cenšas kļūdu samazināt.

MAE

Mean absolute error vai vidēja absolūta kļūda:

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

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

MSE

Mean squared error vai vidēja kvadrātiskā kļūda:

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

Tā ir vidēja starpība starp pareizas un prognozētas vertībām, 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

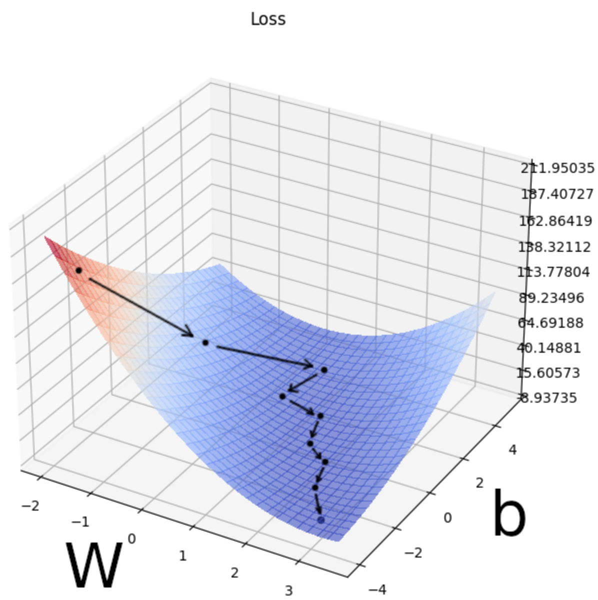
Atpakaļizplatīšanās algoritms

Šīs algoritms cenšas mainīt tīkla svarus un nobīdes tā, lai kļūda būtu 0.

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

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

Kur, piemēram, $J(\theta_0, \theta_1) = L_{MSE} = \frac{1}{N} \cdot \sum_{i=0}^{N} (h_\theta(x_i) - y_i)^2$ ir MSE kļūdas funkcija un α ir apmācības koeficients, kurš noteic, cik strauji svars W un nobīde b tiek mainīti. Kļūdas funkcijas atvasinājums noteic vai parametru ir jāpalielina, vai jāsāmazina.



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 / kofigurāciju 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)

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

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

2) Writing equations for housings:

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

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

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

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:

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

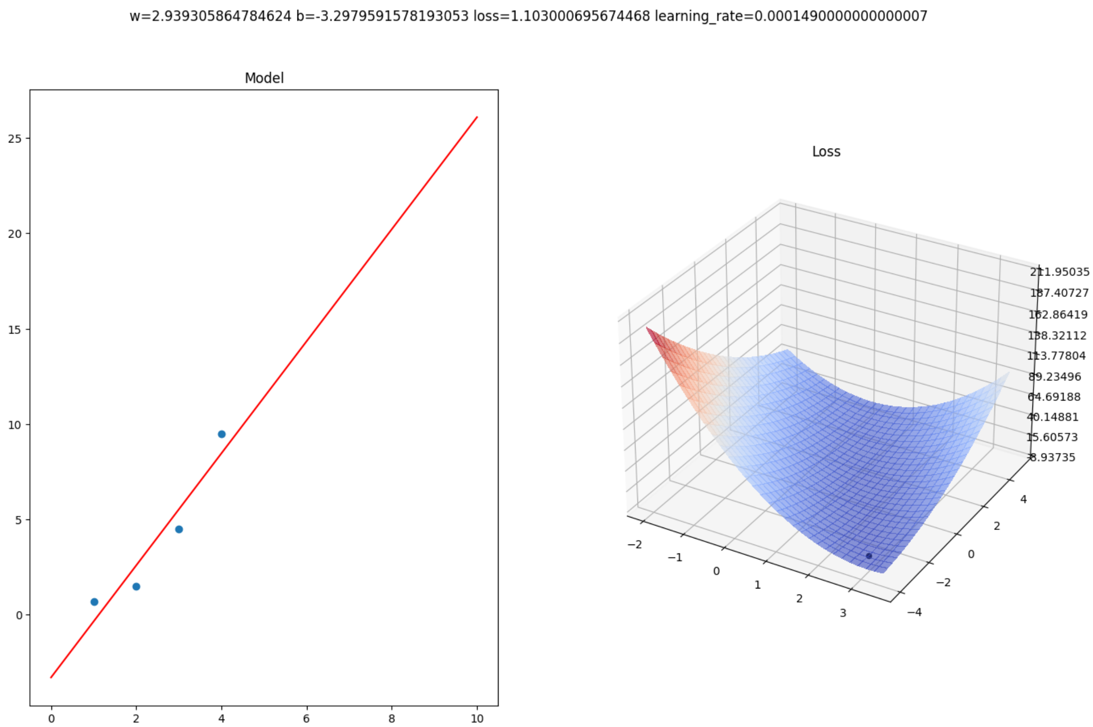
and for θ_1 or W :

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

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:



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 $h_\theta(x)=\frac{1}{1+e^{-a}}$

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

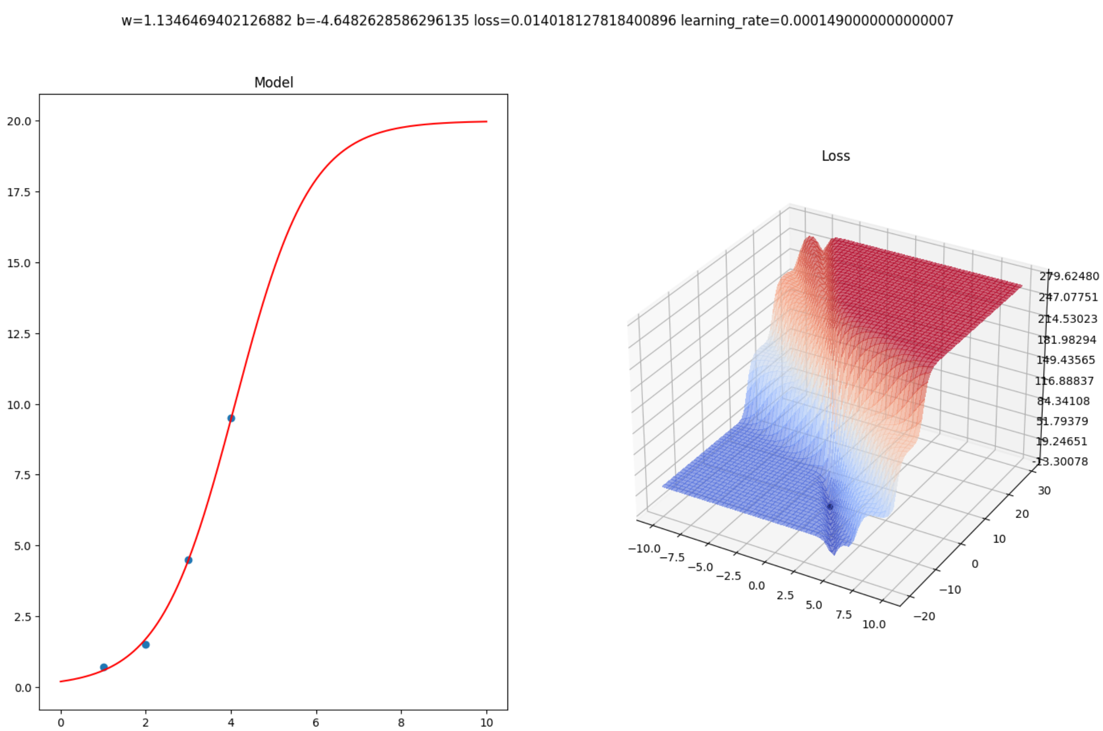
$$=-(1+e^{-a})^{-2}\cdot(0+e^{-a}\cdot\frac{\partial}{\partial a}[-a])=-(1+e^{-a})^{-2}\cdot(e^{-a}\cdot-1)=\frac{e^{-a}}{(1+e^{-a})^2}=$$

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

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

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

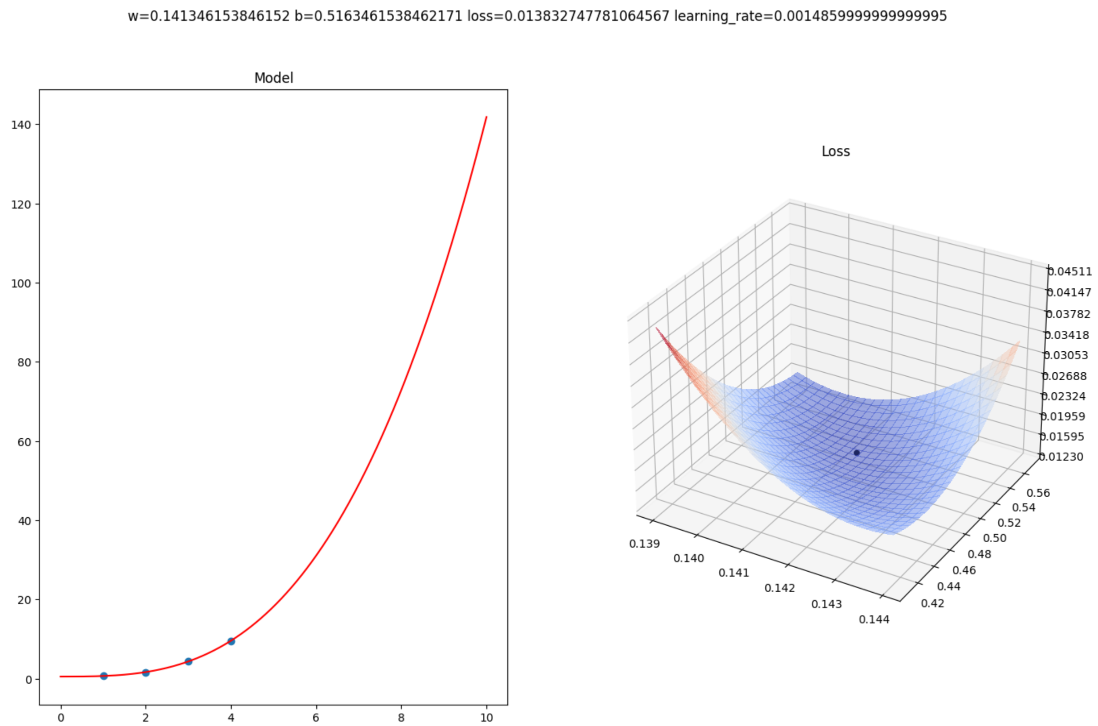
$$\frac{\partial}{\partial\theta_1}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial W}\frac{1}{1+e^{-a}}=\frac{\partial}{\partial a}\cdot\frac{\partial}{\partial W}=\frac{e^{-a}}{(1+e^{-a})^2}\cdot x=\frac{1}{1+e^{-a}}\cdot(1-\frac{1}{1+e^{-a}})\cdot x$$



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:

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



And it worked. That’s much better.

Backpropagation

$$y' = M(x) = Linear(W_1, b_1, W_2, b_2, x) = Linear(W_2, b_2, ReLU(Linear(W_1, b_1, x)))$$

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

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

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

SGD:

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

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

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

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

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

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

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

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

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

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

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

$$M(x) = Linear(W_2, b_2, ReLU(Linear(W_1, b_1, x)))$$

$$z = ReLU(q)$$

$$q = Linear(W_1, b_1, x)$$

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

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

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

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

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

LeakyReLu task

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

$$y' = M(x) = LeakyReLu(Linear(W_1, b_1, W_2, b_2, x), \alpha) =$$
$$= LeakyReLu(Linear(W_2, b_2, \tanh(Linear(W_1, b_1, x))), \alpha)$$

Where:

$$Linear(x) = W \cdot x + b$$

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

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

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

$LeakyReLU(x) = \begin{cases} x, x > 0 \\ \alpha \cdot x, x \le 0 \end{cases}$ here α is a slope, not the learning rate

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

MAE loss function:

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

SGD:

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

$$y' = M(x) = LeakyReLu(Linear(W_1, b_1, W_2, b_2, x), \alpha) =$$
$$= LeakyReLu(Linear(W_2, b_2, \tanh(Linear(W_1, b_1, x))), \alpha)$$

$$m = Linear(W_1, b_1, x)$$
$$k = Linear(W_2, b_2, \tanh(Linear(W_1, b_1, x)))$$
$$l = \tanh(Linear(W_1, b_1, x))$$

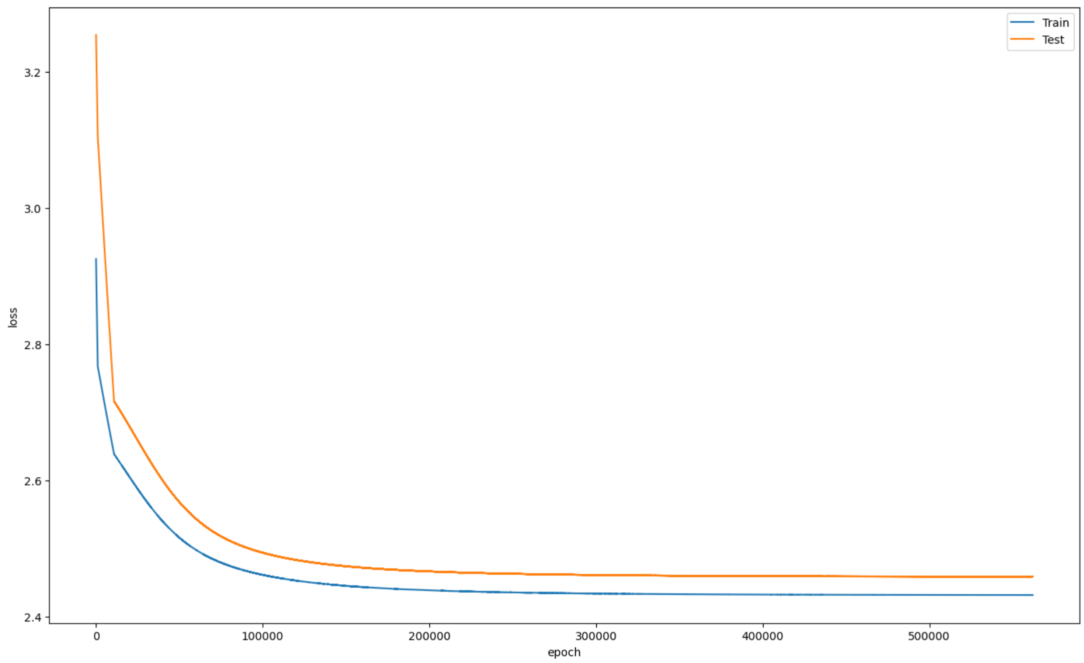
$$\frac{y'}{\partial W_2} = \frac{LeakyReLu(k)}{\partial k} \cdot \frac{k}{\partial W_2} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial W_2} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot l = \frac{LeakyReLu(k)}{\partial k} \cdot \tanh(Linear(W_1, b_1, x))$$

$$\frac{y'}{\partial b_2} = \frac{LeakyReLu(k)}{\partial k} \cdot \frac{k}{\partial b_2} = \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial b_2} = \frac{LeakyReLu(k)}{\partial k} \cdot 1$$

$$\frac{y'}{\partial W_1} = \frac{LeakyReLu(k)}{\partial k} \cdot \frac{k}{\partial W_1} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial W_1} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial l} \cdot \frac{l}{\partial W_1} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(Linear(W_1, b_1, x))}{\partial W_1} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot \frac{m}{\partial W_1} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot \frac{Linear(W_1, b_1, x)}{\partial W_1} =$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot x$$

$$\frac{y'}{\partial b_1} = \frac{LeakyReLu(k)}{\partial k} \cdot \frac{k}{\partial b_1} =$$
$$\ldots same\ as\ for\ W_1 \ldots$$
$$= \frac{LeakyReLu(k)}{\partial k} \cdot \frac{Linear(W_2, b_2, l)}{\partial l} \cdot \frac{\tanh(m)}{\partial m} \cdot 1$$

Running this model produces following result:


Train
Test
3.2
3.0
2.8
2.6
2.4
loss
0
100000
200000
300000
400000
500000
epoch

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:

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

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

- 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)
]
```

Boston house prices (PyTorch)

1) Implement pytorch based housing regression using Boston dataset(not california) and model:

$$y' = M(x) = LeakyReLU(Linear(\tanh(Linear(W \cdot x + b))))$$

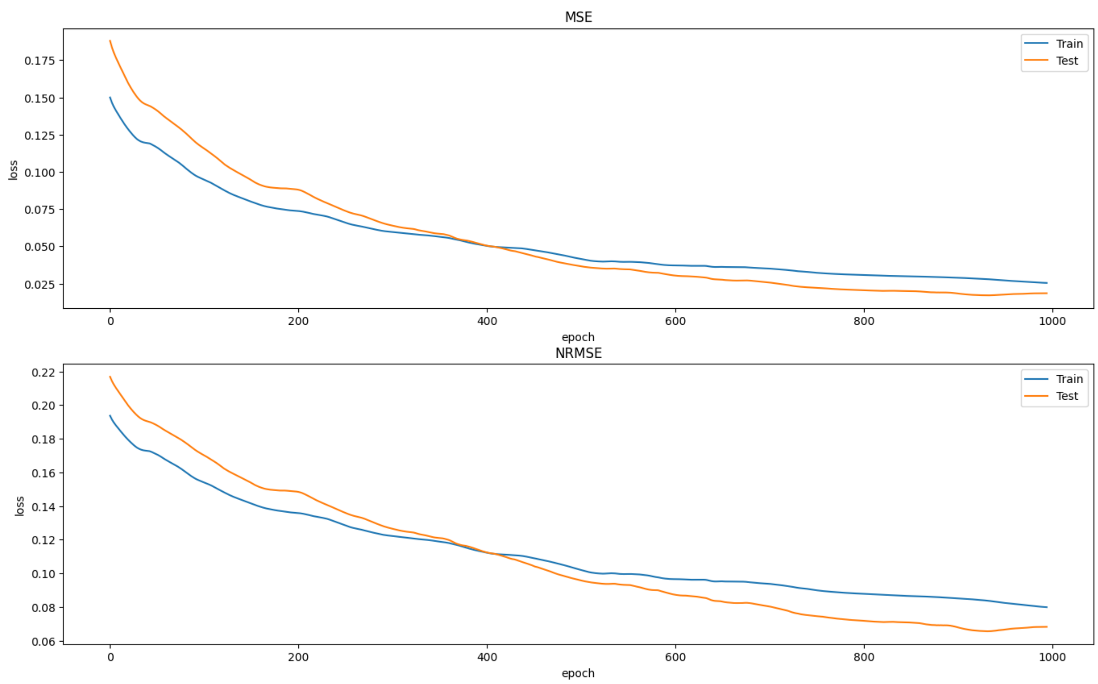
Where:

$$LeakyReLU(x) = \begin{cases} x, x > 0 \\ \alpha \cdot x, x \leq 0 \end{cases}$$

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

$$Linear(x) = W \cdot x + b$$

MSE and NRMSE loss function results (x1000 epochs):



I had to store all the data as tensors in GPU memory and decrease **.item()** which does data sync (VRAM → RAM) and decreases performance. I guess something similar happens when calling **.item()** while running on CPU. So I'm calculating loss values once per 1000 epochs. Speedup is significant.

Wine classification (NumPy, PyTorch)

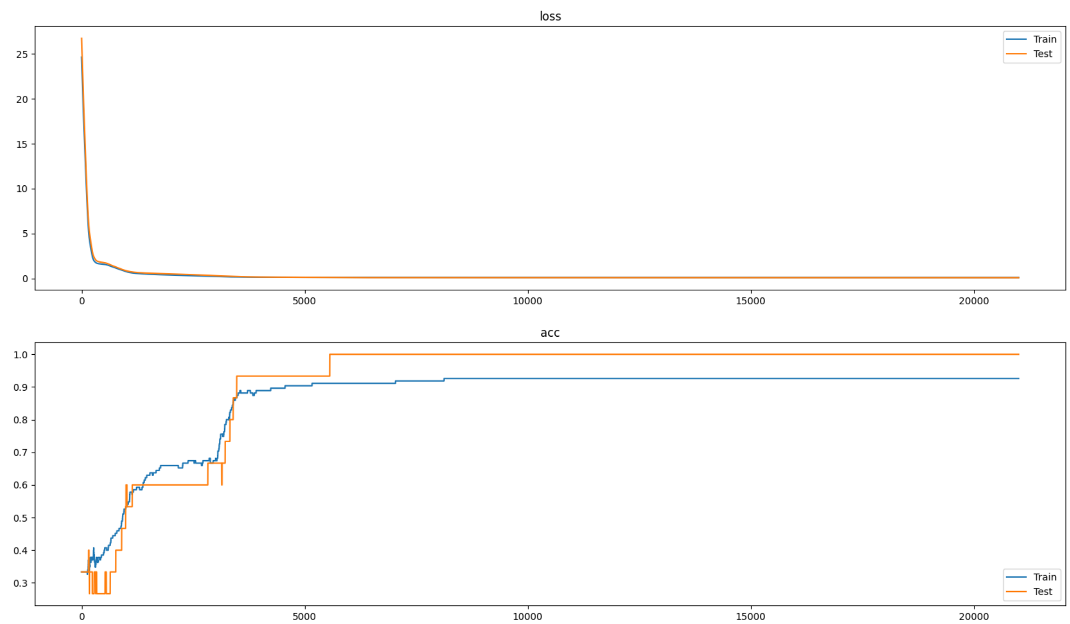
$$L(y, y') = -\frac{1}{N}\sum y \cdot \log(y')$$

$$\frac{\partial L(y, y')}{\partial y'} = -y \cdot \frac{1}{y'} = -\frac{y}{y'}$$

$$SoftMax(y=j|x) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$$

$$\begin{bmatrix} a_0(1-a_0) & -a_0 a_1 & -a_0 a_2 \\ -a_1 a_0 & a_1(1-a_1) & -a_1 a_2 \\ -a_2 a_0 & -a_2 a_1 & a_2(1-a_2) \end{bmatrix}$$

Loss/accuracy for the Iris tutorial:



Accuracy is being calculated as correct guess count divided by total guess count:

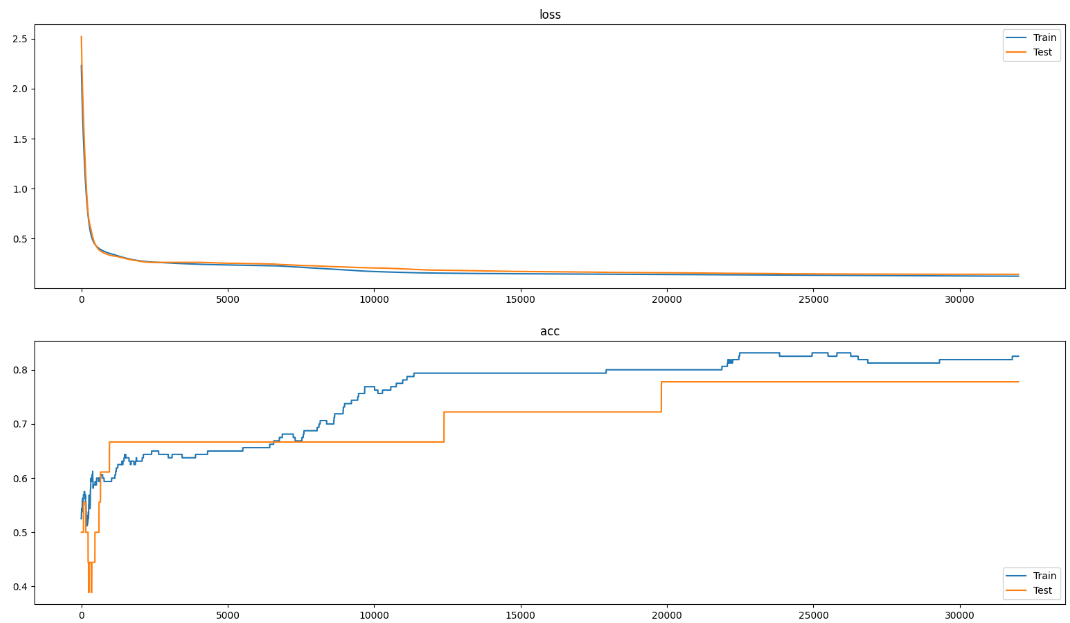
```
guess_cnt = reduce(
    lambda cnt, values: cnt + (np.argmax(values[0]) == np.argmax(values[1])),
    zip(y, y_prim.value),
    0
)
accuracy = guess_cnt / len(y_prim.value)
```

2) Implement numpy based classification using dataset – sklearn.datasets.load_wine

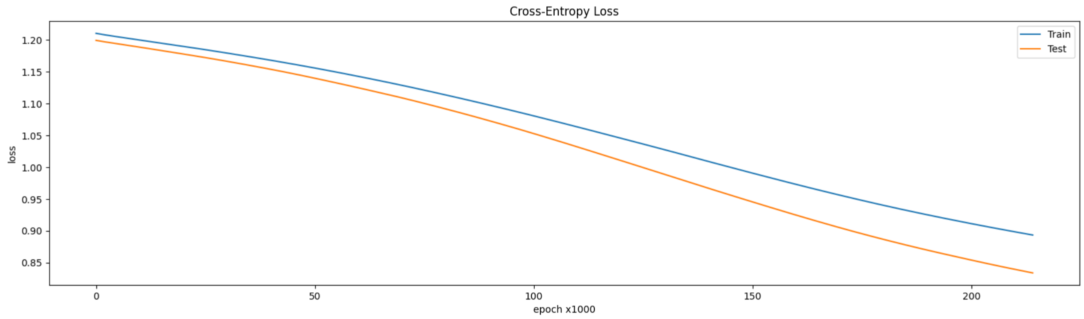
Successfully walked through Iris video tutorial, for some reason same code with different dataset and adjusted input count fails to calculate loss (results in nan).

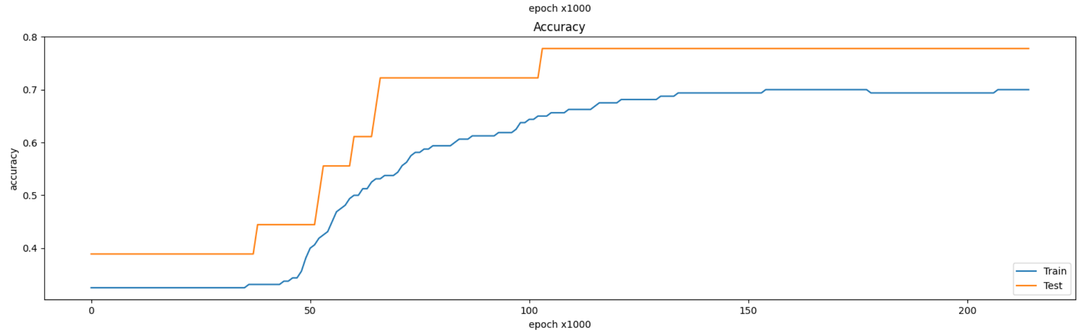
```
def forward(self, x: Variable):
    self.x = x
    np_x = np.copy(x.value)
    # numerical stability for large values
    np_x -= np.max(np_x, axis=1, keepdims=True)
    self.output = Variable(
        (np.exp(np_x + 1e-8)) / np.sum(np.exp(np_x), axis=1, keepdims=True)
    )
    return self.output
```

This is strange. But once I normalized data it started working. PyTorch works in both cases.



3) Implement pytorch based classification using dataset – sklearn.datasets.load_wine


Cross-Entropy Loss
Train
Test
loss
1.20
1.15
1.10
1.05
1.00
0.95
0.90
0.85
0
50
100
150
200
epoch x1000

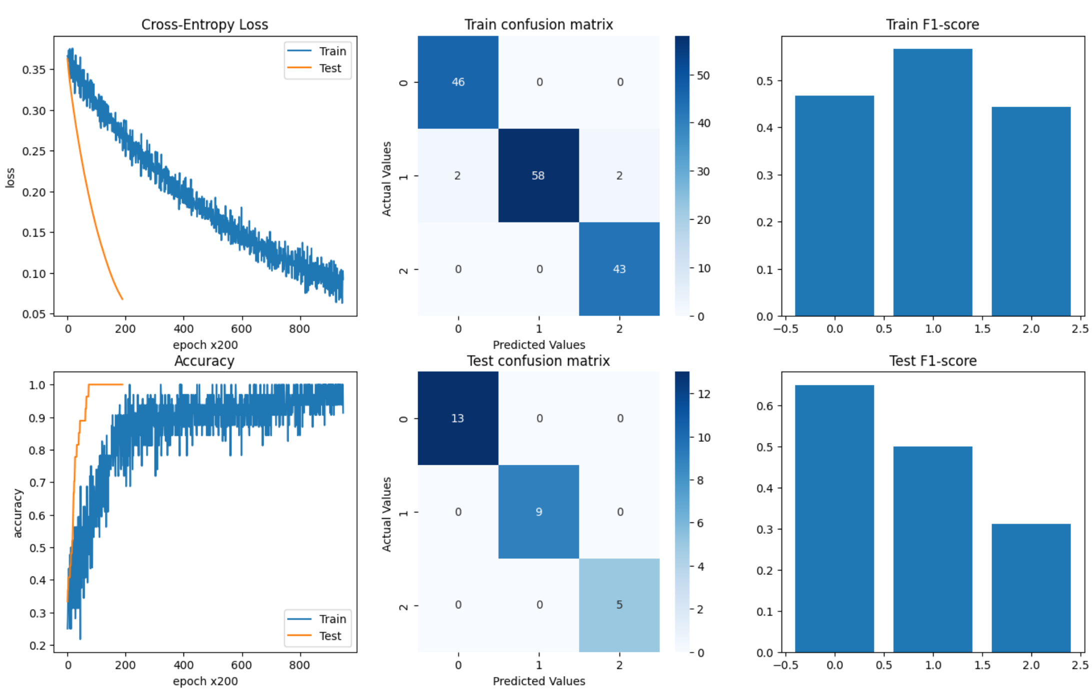

Accuracy
Train
Test
accuracy
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
epoch x1000

Wine classification

1) Implement F1-score and confusion matrix

```
def get_f1_score(matrix: np.ndarray) -> dict:
    score = {}

    for item_idx in range(matrix.shape[0]):
        tp = matrix[item_idx, item_idx]
        tn = matrix[item_idx].sum() - tp
        fn = matrix[:, item_idx].sum() - tp
        fp = matrix.sum() - tp - tn - fn

        score[item_idx] = (2 * tp) / (2 * tp + fp + fn)

    return score
```

```
def get_confusion_matrix(expected, predicted) -> np.ndarray:
    matrix = np.zeros(shape=(3, 3), dtype=np.int)

    for expected_item, predicted_item in zip(expected, predicted):
        matrix[expected_item][predicted_item] += 1

    return matrix
```

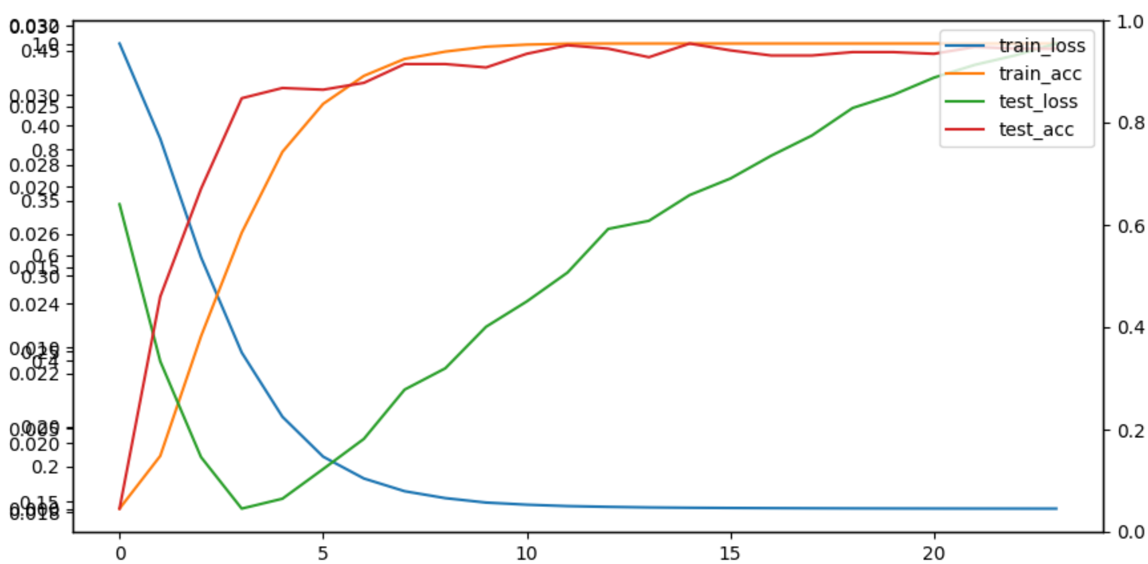


Conv2d (PyTorch)

1) Implement Conv2D task, but instead of using MNIST please change dataset and model to use LFW dataset: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_lfw_people.html#sklearn.datasets.fetch_lfw_people

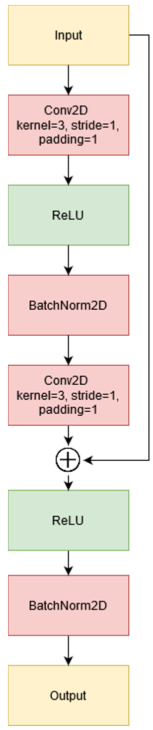
Tried to adjust the model, but never got good results for accuracy

```
self.encoder = torch.nn.Sequential(
    Conv2d(in_channels=1, out_channels=3, kernel_size=9, stride=2, padding=1),
    ReLU(),
    Conv2d(in_channels=3, out_channels=6, kernel_size=7, stride=2, padding=1),
    ReLU(),
    Conv2d(in_channels=6, out_channels=12, kernel_size=5, stride=2, padding=1),
    ReLU(),
    Conv2d(in_channels=12, out_channels=24, kernel_size=3, stride=2, padding=1),
    ReLU(),
    Conv2d(in_channels=24, out_channels=48, kernel_size=3, stride=2, padding=1)
)

o_1 = get_out_size(INPUT_SIZE, kernel_size=9, stride=2, padding=1)
o_2 = get_out_size(o_1, kernel_size=7, stride=2, padding=1)
o_3 = get_out_size(o_2, kernel_size=5, stride=2, padding=1)
o_4 = get_out_size(o_3, kernel_size=3, stride=2, padding=1)
o_5 = get_out_size(o_4, kernel_size=3, stride=2, padding=1)

self.fc = Linear(
    in_features=48*o_5*o_5,
    out_features=feature_count
)
```

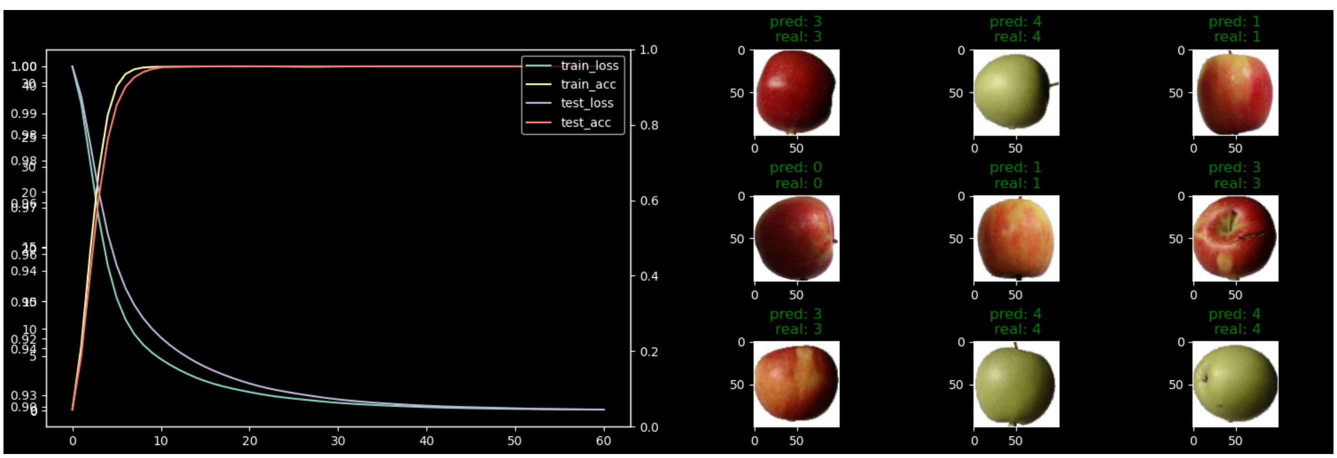


ResNet, DenseNet, InceptionNet

1) Implement ResNet according to scheme:



Performance:



2) Implement DenseNet according to scheme:

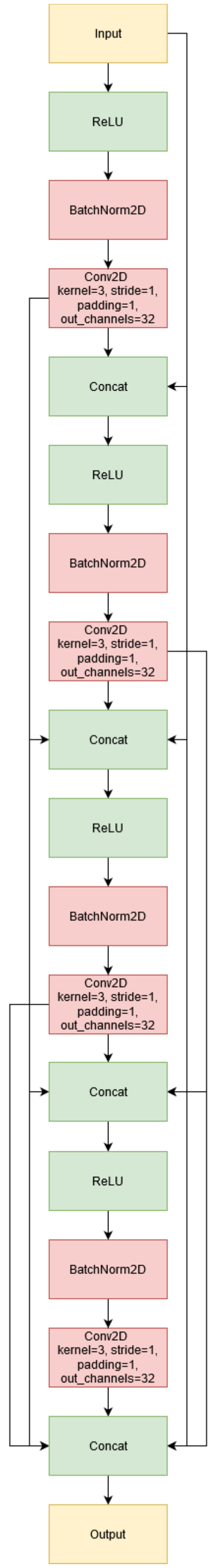


Figure 1: DenseNet block

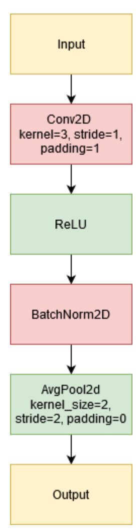
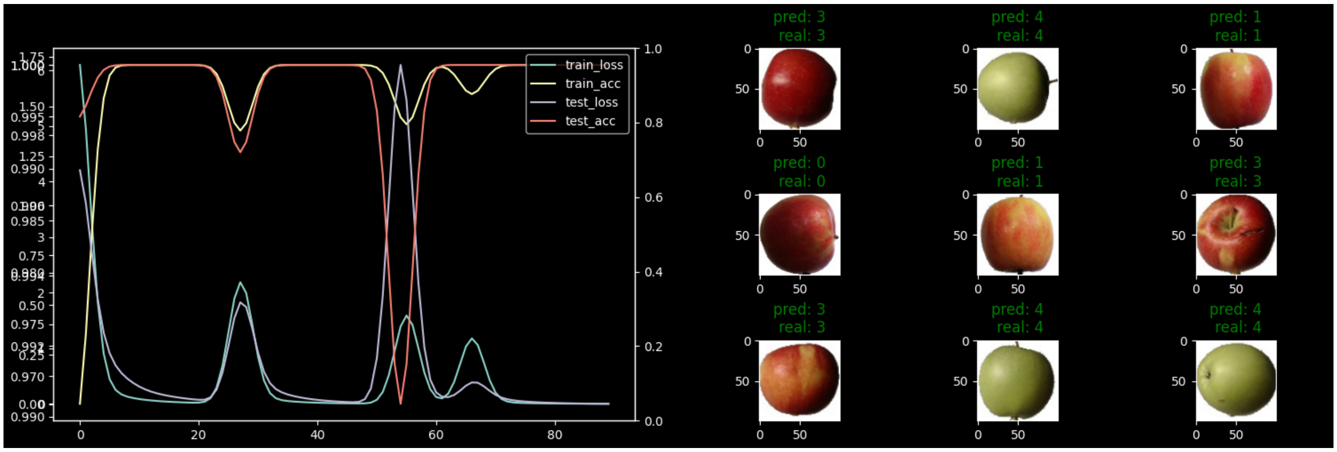


Figure 2: Transition layer

Performance:



3) Implement InceptionNet according to scheme:

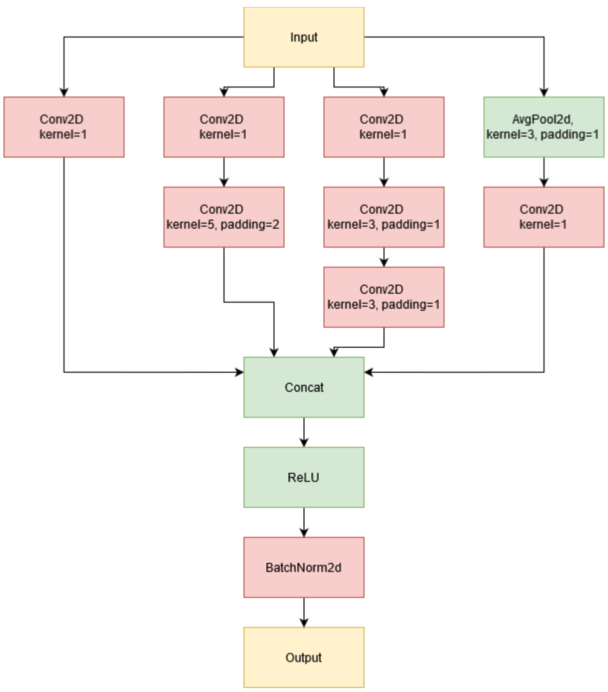


Figure 3: InceptionNet block

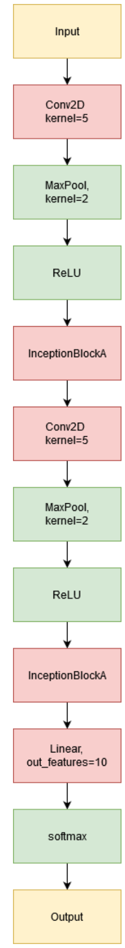
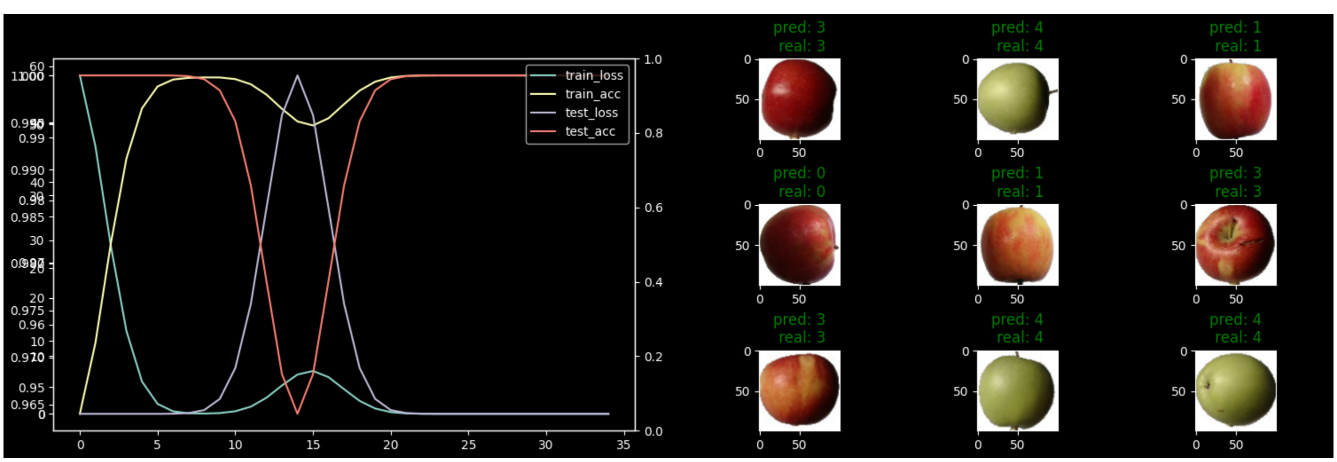


Figure 4: InceptionNet

Performance:



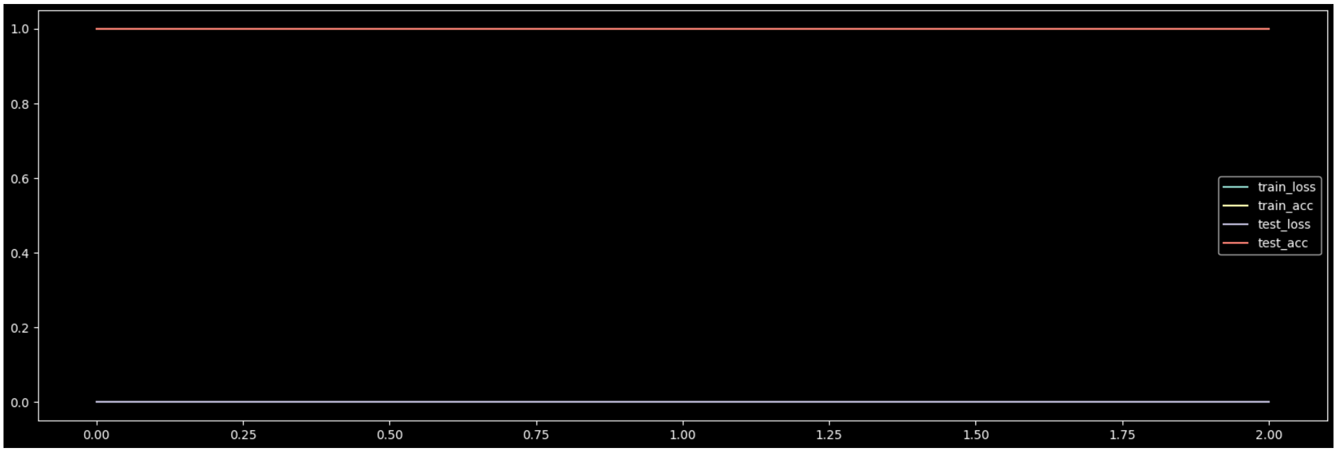
4) Implement own dataset based on NumPy memmap:

```
class DatasetFlickrImageNumpyMmap(torch.utils.data.Dataset):
    def __init__(self, root: str = 'data', force: bool = False):
        super().__init__()
        kaggle.api.authenticate()
        image_dir_name = 'flickr30k_images'
        result_file_name = 'results.csv'
        dataset_file_name = 'data.npy'
        dataset_path = Path(Path(__file__).parent, root, image_dir_name)

        dataset_file_path = Path(dataset_path, dataset_file_name)
        image_dir_path = Path(dataset_path, image_dir_name)
        result_file_path = Path(dataset_path, result_file_name)
        metadata_file_path = Path(dataset_path, 'metadata_mmap.json')

        self.y = []
        self.image_height = 256
        self.image_width = self.image_height
        self.bytes_per_value = 64 / 8

        if force or not dataset_path.exists():
            kaggle.api.dataset_download_files('hsankesara/flickr-image-dataset', path=root, quiet=False,
unzip=True,
                                              force=force)

            # Removing duplicated data
            rmtree(Path(image_dir_path, image_dir_name), ignore_errors=True)
            Path(image_dir_path, result_file_name).unlink(missing_ok=True)

        if force or not dataset_file_path.exists() or not metadata_file_path.exists():
            with open(result_file_path, mode='r', encoding='utf-8') as f:
                reader = csv.reader(f, delimiter='|')
                # Skipping header
                next(reader)

                results = tuple(reader)
                get_filename = itemgetter(0)
                filenames = set(map(get_filename, results))

            self.data_length = len(filenames)
            self.dataset_shape = (self.data_length, 3, self.image_height, self.image_width)

            self.x = open_memmap(
                str(dataset_file_path), mode='w+', dtype='float64', shape=self.dataset_shape
            )

            for idx, filename in enumerate(filenames):
                image_file_path = Path(image_dir_path, filename)
                image = Image.open(image_file_path)
                width, height = image.size  # Get dimensions
                new_size = min(width, height)

                left = int((width - new_size) / 2)
                top = int((height - new_size) / 2)
                right = int((width + new_size) / 2)
                bottom = int((height + new_size) / 2)
```

```
            # Crop the center of the image
            image = image.crop((left, top, right, bottom))

            image = image.resize((self.image_height, self.image_width), resample=Resampling.LANCZOS)
            image = np.asarray(image) / 255.0
            # Converting HxWxC to CxHxW
            image = np.transpose(image, (2, 0, 1))
            self.x[idx, :, :] = image[:, :]

            # TODO: what to use???
            self.y.append(0)

        self.x.flush()

        with open(metadata_file_path, 'w') as f:
            metadata = {
                'shape': self.dataset_shape,
                'labels': self.y
            }
            json.dump(metadata, f)
    else:
        with open(metadata_file_path) as f:
            metadata = json.load(f)

        self.dataset_shape = metadata['shape']
        self.data_length = self.dataset_shape[0]
        self.y = metadata['labels']

    self.x = open_memmap(
        str(dataset_file_path), mode='r', dtype='float64', shape=self.dataset_shape
    )
    self.y = F.one_hot(torch.LongTensor(self.y))

def __len__(self):
    if MAX_LEN:
        return MAX_LEN

    return self.data_length

def __getitem__(self, idx):
    x = torch.from_numpy(np.copy(self.x[idx]))

    return x, self.y[idx]
```

Jautājums: Varbūt būtu labāk jau no paša sākuma pārveidot datus tensorā un saglabāt?

Performance:



Accuracy is 1 and loss is 0 because I had no labels, so they always match “0”.
Takeaway: my Network had so too many channels for batch size of 64 (for 16GB RAM), I thought it was memmap which loaded complete dataset everytime, but debugging showed that it was model.forward(x). So I decreased batch size to 32 and it stopped swapping.

5) Implement own dataset using filesystem

```
class DatasetFlickrImage(torch.utils.data.Dataset):
    def __init__(self, root: str = 'data', force: bool = False, transform=None, target_transform=None):
        super().__init__()
        kaggle.api.authenticate()
        image_dir_name = 'flickr30k_images'
        result_file_name = 'results.csv'
        dataset_path = Path(Path(__file__).parent, root, image_dir_name)

        self.image_dir_path = Path(dataset_path, image_dir_name)
        result_file_path = Path(dataset_path, result_file_name)
        metadata_file_path = Path(dataset_path, 'metadata_file.json')

        self.transform = transform
        self.y = []
        self.image_height = 256
        self.image_width = self.image_height
        self.bytes_per_value = 64 / 8

        if force or not dataset_path.exists():
            kaggle.api.dataset_download_files('hsankesara/flickr-image-dataset', path=root, quiet=False,
unzip=True,
                                              force=force)

            # Removing duplicated data
            rmtree(Path(self.image_dir_path, image_dir_name), ignore_errors=True)
            Path(self.image_dir_path, result_file_name).unlink(missing_ok=True)

        if force or not metadata_file_path.exists():
            with open(result_file_path, mode='r', encoding='utf-8') as f:
                reader = csv.reader(f, delimiter='|')
                # Skipping header
                next(reader)

                results = tuple(reader)
                get_filename = itemgetter(0)
                filenames = set(map(get_filename, results))
                self.data_length = len(filenames)

                with open(metadata_file_path, 'w') as fm:
                    # TODO: what to use???
                    metadata = {filename: 0 for filename in filenames}
                    json.dump(metadata, fm)

        with open(metadata_file_path) as f:
            metadata = json.load(f)

        self.x = tuple(metadata.keys())
        self.y = tuple(metadata.values())
        self.data_length = len(self.y)

        # How do I know class count when passing transformation? idk...
        if target_transform:
            self.y = target_transform(self.y)

        # So I transform it manually inside
```

```
        self.y = F.one_hot(torch.LongTensor(self.y))

    def __len__(self):
        if MAX_LEN:
            return MAX_LEN

        return self.data_length

    def __getitem__(self, idx):
        image_path = Path(self.image_dir_path, self.x[idx])
        x = read_image(str(image_path))
        y = self.y[idx]

        if self.transform:
            x = self.transform(x)

        x = x / 255.0

        return x, y
```

6) Implement own dataset using Zarr:

```
class DatasetFlickrImageZarr(torch.utils.data.Dataset):
    def __init__(self, root: str = 'data', force: bool = False, chunks=None):
        super().__init__()
        kaggle.api.authenticate()
        image_dir_name = 'flickr30k_images'
        result_file_name = 'results.csv'
        dataset_file_name = 'data.zarr'
        dataset_path = Path(Path(__file__).parent, root, image_dir_name)

        dataset_file_path = Path(dataset_path, dataset_file_name)
        image_dir_path = Path(dataset_path, image_dir_name)
        result_file_path = Path(dataset_path, result_file_name)

        self.y = []
        self.image_height = 256
        self.image_width = self.image_height
        self.bytes_per_value = 64 / 8

        if force or not dataset_path.exists():
            kaggle.api.dataset_download_files('hsankesara/flickr-image-dataset', path=root, quiet=False,
unzip=True,
                                              force=force)

            # Removing duplicated data
            rmtree(Path(image_dir_path, image_dir_name), ignore_errors=True)
            Path(image_dir_path, result_file_name).unlink(missing_ok=True)

        if force or not dataset_file_path.exists():
            with open(result_file_path, mode='r', encoding='utf-8') as f:
                reader = csv.reader(f, delimiter='|')
                # Skipping header
                next(reader)

                results = tuple(reader)
                get_filename = itemgetter(0)
                filenames = set(map(get_filename, results))

            self.data_length = len(filenames)
            self.dataset_shape = (self.data_length, 3, self.image_height, self.image_width)

            self.dataset = zarr.open(str(dataset_file_path), mode='w')
            self.x = self.dataset.zeros('samples', shape=self.dataset_shape, chunks=chunks, dtype='float64')
            self.y = self.dataset.zeros('labels', dtype='int64', shape=self.data_length if USE_CUDA else MAX_LEN)

            y = []

            for idx, filename in enumerate(filenames):
                image_file_path = Path(image_dir_path, filename)
                image = Image.open(image_file_path)
                width, height = image.size  # Get dimensions
                new_size = min(width, height)

                left = int((width - new_size) / 2)
                top = int((height - new_size) / 2)
                right = int((width + new_size) / 2)
```

```
            bottom = int((height + new_size) / 2)

            # Crop the center of the image
            image = image.crop((left, top, right, bottom))

            image = image.resize((self.image_height, self.image_width), resample=Resampling.LANCZOS)
            image = np.asarray(image) / 255.0
            # Converting HxWxC to CxHxW
            image = np.transpose(image, (2, 0, 1))
            self.x[idx, :, :] = image

            # TODO: what to use???
            y.append(0)

            # WORKAROUND: save time
            if idx >= MAX_LEN - 1:
                break

        self.y[:] = torch.tensor(y, dtype=torch.long)

    self.root = zarr.open(str(dataset_file_path), mode='r')
    self.x = self.root['samples']
    self.y = F.one_hot(self.root['labels'])
    self.data_length = len(self.y)

def __len__(self):
    if MAX_LEN:
        return MAX_LEN

    return self.data_length

def __getitem__(self, idx):
    x = torch.from_numpy(np.copy(self.x[idx]))

    return x, self.y[idx]
```

7) Implement own dataset using HDF5

TODO

8) Implement own Dataset using CuPy
TODO