Минобрнауки России

ФГБОУ ВПО «НИУ МЭИ» АВТИ

Кафедра математического и компьютерного моделирования

**Лабораторная работа №10**

**Компьютерная графика**

**Работу выполнил:**

Солонин Егор А-14-19

**Работу принял:**

Бартеньев О.В.

Москва 2021

1. **Задание.**

1. Пополнить MNIST 11-м классом "не цифры", поместив в него случайно выбранные буквы EMNIST.

11-й класс формируется из представителей всех классов EMNIST.

В обучающую порцию данных 11-го класса добавляются по 231 примеру из первых 20-и классов EMNIST

и по 230 из последующих (231 \* 20 + 230 \* 6 = 6000).

Аналогично создается и проверочная порция данных 11-го класса MNIST.

2. Используя PyTorch, создать и обучить модель нейронной сети для классификации примеров сформированного в п. 1 набора данных.

Максимально приблизить точность классификации к 99,6%.

3. Обученную модель сохранить в файл.

4. При обучении, кроме потерь, выводить точность.

5. Построить графики обучения (по аналогии с ЛР6).

6. В режиме проверки (прогнозирования) загружать модель из файла.

7. Вывести точность классификации примеров проверочного множества по классам.

8. Вывести оценки качества модели посредством classification\_report из sklearn.metrics

9. Вывести рисунки неверно классифицированных изображений (примеры).

10. Напечатать список неверно классифицированных изображений проверочного множества (в виде таблицы),

указывая в нем номер класса и индекс изображения. Список упорядочить по числу ошибок в классе.

11. Результат оформить в виде отчета.

1. **Входные данные**

* MNIST – 70'000 рукописных цифр; из них 60'000 входят в обучающую выборку, а 10'000 – в тестовую; размер каждого образа – 28\*28 пикселей; рисунки выполнены в оттенках серого цвета;
* EMNIST-letters – 145'600 рукописных букв английского алфавита; из них 124'800 входят в обучающую выборку, а 20'800 – в тестовую; размер каждого образа – 28\*28 пикселей; рисунки выполнены в оттенках серого цвета

1. **Выходные данные**
2. **Потери и точность на каждой эпохе:**

Загрузка данных из двоичных файлов

Число эпох 300

Эпоха: 1

Потери: обучение: 0.341508 проверка: 0.086834

Точность: обучение: 89.50% проверка: 97.18%

Эпоха: 2

Потери: обучение: 0.131197 проверка: 0.068422

Точность: обучение: 95.93% проверка: 97.84%

Эпоха: 3

Потери: обучение: 0.105589 проверка: 0.053288

Точность: обучение: 96.72% проверка: 98.28%

Эпоха: 4

Потери: обучение: 0.091308 проверка: 0.053520

Точность: обучение: 97.11% проверка: 98.32%

Эпоха: 5

Потери: обучение: 0.090020 проверка: 0.046114

Точность: обучение: 97.19% проверка: 98.44%

Эпоха: 6

Потери: обучение: 0.079506 проверка: 0.050863

Точность: обучение: 97.62% проверка: 98.29%

Эпоха: 7

Потери: обучение: 0.079834 проверка: 0.046519

Точность: обучение: 97.54% проверка: 98.60%

Эпоха: 8

Потери: обучение: 0.078348 проверка: 0.047256

Точность: обучение: 97.55% проверка: 98.49%

Эпоха: 9

Потери: обучение: 0.074032 проверка: 0.041572

Точность: обучение: 97.72% проверка: 98.70%

Эпоха: 10

Потери: обучение: 0.069822 проверка: 0.042921

Точность: обучение: 97.75% проверка: 98.62%

Эпоха: 11

Потери: обучение: 0.071801 проверка: 0.040368

Точность: обучение: 97.83% проверка: 98.80%

Эпоха: 12

Потери: обучение: 0.068820 проверка: 0.038749

Точность: обучение: 97.92% проверка: 98.67%

Эпоха: 13

Потери: обучение: 0.066475 проверка: 0.041786

Точность: обучение: 97.96% проверка: 98.71%

Эпоха: 14

Потери: обучение: 0.068003 проверка: 0.041631

Точность: обучение: 97.90% проверка: 98.64%

Эпоха: 15

Потери: обучение: 0.066869 проверка: 0.037635

Точность: обучение: 97.86% проверка: 98.83%

Эпоха: 16

Потери: обучение: 0.065239 проверка: 0.038603

Точность: обучение: 97.96% проверка: 98.85%

Эпоха: 17

Потери: обучение: 0.063076 проверка: 0.043836

Точность: обучение: 98.04% проверка: 98.67%

Эпоха: 18

Потери: обучение: 0.069158 проверка: 0.051173

Точность: обучение: 97.85% проверка: 98.42%

Эпоха: 19

Потери: обучение: 0.066560 проверка: 0.044280

Точность: обучение: 97.92% проверка: 98.69%

Эпоха: 20

Потери: обучение: 0.069900 проверка: 0.035380

Точность: обучение: 97.86% проверка: 99.01%

...

...

Эпоха: 290

Потери: обучение: 0.057692 проверка: 0.047977

Точность: обучение: 98.38% проверка: 98.91%

Эпоха: 291

Потери: обучение: 0.058716 проверка: 0.044604

Точность: обучение: 98.36% проверка: 98.85%

Эпоха: 292

Потери: обучение: 0.058673 проверка: 0.051999

Точность: обучение: 98.42% проверка: 98.79%

Эпоха: 293

Потери: обучение: 0.061541 проверка: 0.042542

Точность: обучение: 98.28% проверка: 98.85%

Эпоха: 294

Потери: обучение: 0.060643 проверка: 0.049039

Точность: обучение: 98.32% проверка: 98.80%

Эпоха: 295

Потери: обучение: 0.056402 проверка: 0.044593

Точность: обучение: 98.43% проверка: 98.94%

Эпоха: 296

Потери: обучение: 0.059177 проверка: 0.046926

Точность: обучение: 98.38% проверка: 98.84%

Эпоха: 297

Потери: обучение: 0.056255 проверка: 0.043664

Точность: обучение: 98.43% проверка: 98.83%

Эпоха: 298

Потери: обучение: 0.059922 проверка: 0.040943

Точность: обучение: 98.38% проверка: 98.89%

Эпоха: 299

Потери: обучение: 0.060688 проверка: 0.038961

Точность: обучение: 98.35% проверка: 98.93%

Эпоха: 300

Потери: обучение: 0.056554 проверка: 0.044418

Точность: обучение: 98.42% проверка: 98.84%

1. **Результаты обучения**

Потери: 0.044418

Точность: 0.9884

Точность по классам:

0 - 99.7959%

1 - 99.2952%

2 - 98.7403%

3 - 97.9208%

4 - 98.8798%

5 - 98.6547%

6 - 99.0605%

7 - 98.7354%

8 - 99.0760%

9 - 97.7205%

10 - 99.3000%

**Classification report:**

precision recall f1-score support

0 1.00 0.93 0.97 30

1 0.96 1.00 0.98 24

2 1.00 0.91 0.95 23

3 1.00 1.00 1.00 27

4 1.00 1.00 1.00 26

5 1.00 0.95 0.97 20

6 0.88 1.00 0.94 15

7 1.00 1.00 1.00 30

8 1.00 1.00 1.00 24

9 0.95 1.00 0.98 21

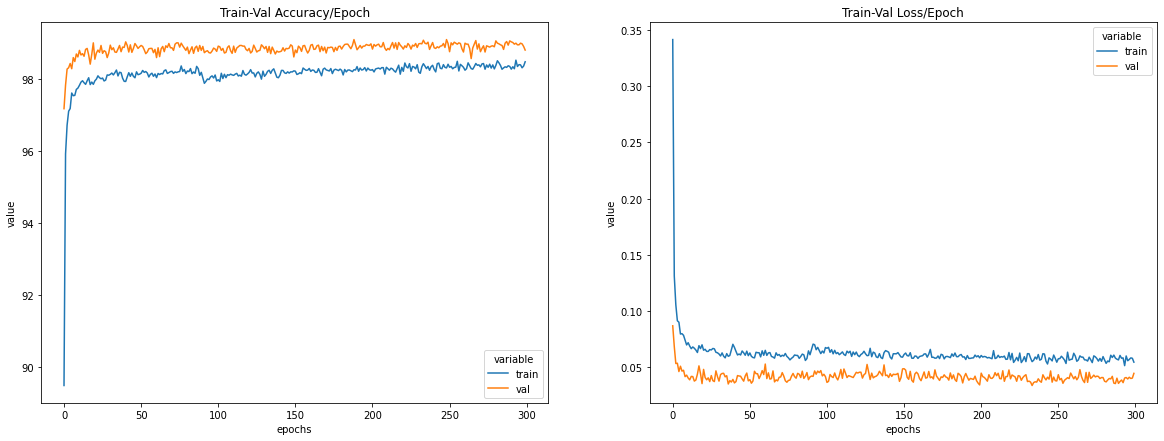
10 0.94 1.00 0.97 16

accuracy 0.98 256

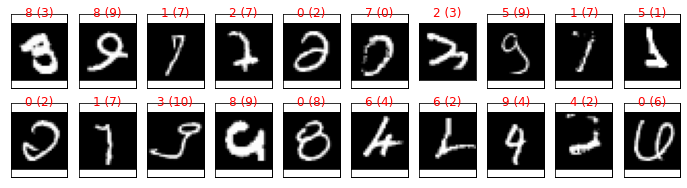
macro avg 0.98 0.98 0.98 256

weighted avg 0.98 0.98 0.98 256

1. **Графики обучения**

****

1. **Примеры неверно классифицированных изображений**

****

1. **Код**

from sys import exit

import matplotlib.pyplot as plt

import numpy as np

import torch

import pandas as pd

import seaborn as sns

import torch.nn as nn

import torch.nn.functional as F

from torch.utils.data import DataLoader

from sklearn.metrics import classification\_report

import warnings

warnings.filterwarnings('ignore')

#

pathToData = "D:/mpei/LabsCGI/А14\_СолонинЕгор\_ЛР10/mnistBinData/"

pathToEmnistData = "D:/mpei/LabsCGI/А14\_СолонинЕгор\_ЛР10/emnistBinData/"

img\_rows = img\_cols = 28

num\_classes = 11

show\_img = not True

batch\_size = 256  # Размер обучающего (проверочного) пакета

train\_model = not True

save\_model = not True

load\_model = True

show\_model = True

fn\_w = 'model.wei'  # Файл с весами НС

n\_epochs = 300  # Число эпох обучения

criterion = nn.CrossEntropyLoss()  # Функция потерь

#

accuracy\_stats = {

    'train': [],

    "val": []

}

loss\_stats = {

    'train': [],

    "val": []

}

class Net(nn.Module):

    def \_\_init\_\_(self):

        super(Net, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(1, 16, kernel\_size=3)

        self.conv2 = nn.Conv2d(16, 32, kernel\_size=3)

        self.max\_pool2d1 = nn.MaxPool2d(2)

        self.max\_pool2d2 = nn.MaxPool2d(2)

        self.conv2\_drop = nn.Dropout2d(p=0.3)

        self.dropout = nn.Dropout(p=0.3)

        self.fc1 = nn.Linear(800, 32)  # 800 - считаем сами

        self.fc2 = nn.Linear(32, num\_classes)

    def forward(self, x):

        x = self.conv1(x)

        x = F.relu(x)

        x = self.max\_pool2d1(x)

        x = self.conv2\_drop(x)

        x = self.conv2(x)

        x = F.relu(x)

        x = self.max\_pool2d2(x)

        x = x.view(-1, 800)  # 800 - считаем сами (flatten)

        x = self.dropout(x)

        x = F.relu(self.fc1(x))

        x = self.fc2(x)  # torch.Size([256, 10])

        return F.log\_softmax(x, dim=-1)

model = Net()  # Формируем модель НС

if show\_model:

    print(model)

def load\_bin\_data(pathToData, img\_rows, img\_cols, show\_img):

    print('Загрузка данных из двоичных файлов')

    with open(pathToData + 'imagesTrain.bin', 'rb') as read\_binary:

        x\_trn = np.fromfile(read\_binary, dtype=np.uint8)

    with open(pathToData + 'labelsTrain.bin', 'rb') as read\_binary:

        y\_trn = np.fromfile(read\_binary, dtype=np.uint8)

    with open(pathToData + 'imagesTest.bin', 'rb') as read\_binary:

        x\_tst = np.fromfile(read\_binary, dtype=np.uint8)

    with open(pathToData + 'labelsTest.bin', 'rb') as read\_binary:

        y\_tst = np.fromfile(read\_binary, dtype=np.uint8)

    pathToData = pathToData.replace("mnist", "emnist")

    with open(pathToData + 'imagesTrain.bin', 'rb') as read\_binary:

        x\_trn\_e = np.fromfile(read\_binary, dtype=np.uint8)

    with open(pathToData + 'labelsTrain.bin', 'rb') as read\_binary:

        y\_trn\_e = np.fromfile(read\_binary, dtype=np.uint8)

    with open(pathToData + 'imagesTest.bin', 'rb') as read\_binary:

        x\_tst\_e = np.fromfile(read\_binary, dtype=np.uint8)

    with open(pathToData + 'labelsTest.bin', 'rb') as read\_binary:

        y\_tst\_e = np.fromfile(read\_binary, dtype=np.uint8)

    x\_trn = x\_trn.reshape(-1, 1, img\_rows, img\_cols)

    x\_tst = x\_tst.reshape(-1, 1, img\_rows, img\_cols)

    x\_trn\_e = x\_trn\_e.reshape(-1, 1, img\_rows, img\_cols)

    x\_tst\_e = x\_tst\_e.reshape(-1, 1, img\_rows, img\_cols)

    x\_tst\_e, y\_tst\_e = build\_11th\_class(x\_tst\_e, y\_tst\_e, 1000)

    x\_trn\_e, y\_trn\_e = build\_11th\_class(x\_trn\_e, y\_trn\_e, 6000)

    x\_trn = np.concatenate((x\_trn, x\_trn\_e), axis=0)

    y\_trn = np.concatenate((y\_trn, y\_trn\_e), axis=0)

    x\_tst = np.concatenate((x\_tst, x\_tst\_e), axis=0)

    y\_tst = np.concatenate((y\_tst, y\_tst\_e), axis=0)

    if show\_img:

        for i in range(9):

            plt.subplot(3, 3, i + 1)

            img = x\_trn[i].reshape(img\_rows, img\_cols)

            plt.imshow(img, cmap=plt.get\_cmap('gray'))

            plt.title(y\_trn[i])

            plt.axis('off')

        plt.subplots\_adjust(hspace=0.5)

        plt.show()

    x\_trn = np.array(x\_trn, dtype='float32') / 255

    x\_tst = np.array(x\_tst, dtype='float32') / 255

    return x\_trn, y\_trn, x\_tst, y\_tst

def build\_11th\_class(x, y, ammount):

    repres\_count = np.zeros(27)

    repres\_1\_max = 231 if ammount == 6000 else 39

    repres\_2\_max = 230 if ammount == 6000 else 38

    counter = 0

    border = 20 if ammount == 6000 else 12

    x\_res = x

    for i in range(len(y)):

        if counter == ammount:

            break

        if y[i] > border and repres\_count[y[i]] < repres\_2\_max:

            x\_res[counter] = x[i]

            repres\_count[y[i]] += 1

            counter += 1

        if y[i] <= border and repres\_count[y[i]] < repres\_1\_max:

            x\_res[counter] = x[i]

            repres\_count[y[i]] += 1

            counter += 1

    x\_res = x\_res[:ammount]

    x\_res = x\_res.transpose(0, 1, 3, 2)

    y\_res = np.ones(ammount) \* 10

    return x\_res, y\_res

x\_trn, y\_trn, x\_tst, y\_tst = load\_bin\_data(

    pathToData, img\_rows, img\_cols, show\_img)

trn\_data = [[x, int(y)] for x, y in zip(x\_trn, y\_trn)]

tst\_data = [[x, int(y)] for x, y in zip(x\_tst, y\_tst)]

# Формирование обучающих и проверочных пакетов

trn\_loader = DataLoader(trn\_data, batch\_size=batch\_size, shuffle=True)

tst\_loader = DataLoader(tst\_data, batch\_size=batch\_size, shuffle=True)

# Число примеров в обучающем и проверочном множествах

in\_trn = len(trn\_loader.sampler)  # 66000

in\_tst = len(tst\_loader.sampler)  # 11000

if show\_img:

    trn\_features, trn\_labels = next(iter(trn\_loader))

    img = trn\_features[0].squeeze()

    ind = int(trn\_labels[0])

    plt.imshow(img, cmap='gray')

    plt.title(ind)

    plt.axis('off')

    plt.show()

def train(epoch):

    trn\_loss = tst\_loss = 0

    trn\_correct = tst\_correct = 0

    model.train()  # Режим обучения

    for batch\_no, (data, target) in enumerate(trn\_loader):

        optimizer.zero\_grad()  # Обнуляем градиенты

        output = model(data)  # forward

        loss = criterion(output, target)  # <class 'torch.Tensor'>

        loss.backward()  # Подготовка потерь для обратного шага

        optimizer.step()  # Обратное распространение ошибки, обновляем веса модели

        trn\_loss += loss.item() \* data.size(0)

        \_, pred = torch.max(output, 1)

        trn\_correct += (pred == target).sum().item()

    model.eval()  # Режим оценки

    for data, target in tst\_loader:

        output = model(data)

        loss = criterion(output, target)

        tst\_loss += loss.item() \* data.size(0)

        \_, pred = torch.max(output, 1)

        tst\_correct += (pred == target).sum().item()

    trn\_loss = trn\_loss / in\_trn

    tst\_loss = tst\_loss / in\_tst

    trn\_prec = trn\_correct / in\_trn \* 100

    tst\_prec = tst\_correct / in\_tst \* 100

    loss\_stats['train'].append(trn\_loss)

    loss\_stats['val'].append(tst\_loss)

    accuracy\_stats['train'].append(trn\_prec)

    accuracy\_stats['val'].append(tst\_prec)

    print('\nЭпоха: {}\n\t Потери: обучение: {:.6f} \tпроверка: {:.6f}\n\t Точность: обучение: {:.2f}% \tпроверка: {:.2f}%'.format(

        epoch + 1, trn\_loss, tst\_loss, trn\_prec, tst\_prec))

if load\_model:

    print('Загрузка весов из файла', fn\_w)

    device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

    model.load\_state\_dict(torch.load(fn\_w, map\_location=torch.device(device)))

if train\_model:

    print('Число эпох', n\_epochs)

    # betas = (0.9, 0.98), eps = 1e-9

    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

    print('Обучение')

    for ep in range(n\_epochs):

        train(ep)

    if save\_model:

        print('Сохранение весов модели')

        torch.save(model.state\_dict(), fn\_w)

def test():

    print('Проверка')

    tst\_loss = 0  # Потери на проверочном множестве (ПМ)

    y\_pred\_list = []

    # Число верно классифицированных цифр в каждом классе (ПМ)

    cls\_correct = [0]\*num\_classes

    # Число цифр в каждом классе (ПМ)

    cls\_total = [0]\*num\_classes

    model.eval()  # Режим оценки

    for data, target in tst\_loader:

        output = model(data)

        loss = criterion(output, target)

        tst\_loss += loss.item() \* data.size(0)

        \_, pred = torch.max(output, 1)

        y\_pred\_list.append(pred.cpu().numpy())

        correct = np.squeeze(pred.eq(target.data.view\_as(pred)))

        for i in range(len(target)):

            label = target.data[i]

            cls\_correct[label] += correct[i].item()

            cls\_total[label] += 1

    # Средние потери

    tst\_loss /= len(tst\_loader.sampler)

    print('Потери: {:.6f}'.format(tst\_loss))

    print('Точность: {:.4f}'.format(sum(cls\_correct) / sum(cls\_total)))

    print('Точность по классам:')

    cls = 0

    for cc, ct in zip(cls\_correct, cls\_total):

        print('{} - {:.4f}%'.format(cls, cc / ct \* 100))

        cls += 1

def show\_res():

    print('Смотрим начальный кусок первого пакета')

    data\_iter = iter(tst\_loader)  # Первый пакет

    images, labels = data\_iter.next()

    output = model(images)

    \_, pred = torch.max(output, 1)

    images = images.numpy()

    fig = plt.figure(figsize=(12, 3))

    N = 20

    for i in range(N):

        ax = fig.add\_subplot(2, int(N / 2), i + 1, xticks=[], yticks=[])

        img = images[i]  # (1, 28, 28)

        img = np.squeeze(img)  # или: img = img.reshape(img\_rows, img\_cols)

        ax.imshow(img, cmap='gray')

        ttl = str(int(pred[i].item())) + ' (' + str(labels[i].item()) + ')'

        clr = 'green' if pred[i] == labels[i] else 'red'

        ax.set\_title(ttl, color=clr)

    plt.show()

    show\_graph()

    print(

        f"Classification report:\n"

        f"{classification\_report(pred, labels)}\n"

    )

def show\_err():

    data\_iter = iter(tst\_loader)  # Первый пакет

    images, labels = data\_iter.next()

    output = model(images)

    \_, pred = torch.max(output, 1)

    images = images.numpy()

    fig = plt.figure(figsize=(12, 3))

    N = 20

    i = count = 0

    while count < N:

        ax = fig.add\_subplot(2, int(N / 2), count + 1, xticks=[], yticks=[])

        img = images[i]  # (1, 28, 28)

        img = np.squeeze(img)  # или: img = img.reshape(img\_rows, img\_cols)

        if pred[i] != labels[i]:

            ax.imshow(img, cmap='gray')

            ttl = str(int(pred[i].item())) + ' (' + str(labels[i].item()) + ')'

            ax.set\_title(ttl, color='red')

            count += 1

            print(count)

        i += 1

        if i > 255:

            images, labels = data\_iter.next()

            output = model(images)

            \_, pred = torch.max(output, 1)

            images = images.numpy()

            i = 0

    plt.show()

    show\_graph()

    print(

        f"Classification report:\n"

        f"{classification\_report(pred, labels)}\n"

    )

def show\_graph():

    # Create dataframes

    train\_val\_acc\_df = pd.DataFrame.from\_dict(accuracy\_stats).reset\_index().melt(

        id\_vars=['index']).rename(columns={"index": "epochs"})

    train\_val\_loss\_df = pd.DataFrame.from\_dict(loss\_stats).reset\_index().melt(

        id\_vars=['index']).rename(columns={"index": "epochs"})

    # Plot the dataframes

    fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 7))

    sns.lineplot(data=train\_val\_acc\_df, x="epochs", y="value",

                 hue="variable",  ax=axes[0]).set\_title('Train-Val Accuracy/Epoch')

    sns.lineplot(data=train\_val\_loss\_df, x="epochs", y="value",

                 hue="variable", ax=axes[1]).set\_title('Train-Val Loss/Epoch')

test()  # Проверка

# show\_res()  # Смотрим начальный кусок первого пакета

show\_err()