Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе № 7

по курсу «Основы глубокого обучения»

Выполнил:  
студент группы 22ВВИм1:

Милованов А.С.

Приняли:

Митрохин М.А  
Панков А.А.

Пенза 2023

# Порядок выполнения работы:

# Задание:

ЗАДАНИЕ 1: создать соревновательную нейросеть генерации рукописных цифр.

# Листинг программы:

import torch

import torch.nn as nn

from torch.utils.data import DataLoader

from torchvision import transforms, datasets

from matplotlib import pyplot as plt, gridspec

from torchvision import datasets, transforms as T

train\_augs = T.Compose([

    T.RandomRotation((-20,20)),

    T.ToTensor()  # converts images to pytorch tensors, also changes  channel to 0th axis (h,w,c) -> (c,h,w)

])

train\_set  = datasets.MNIST(root='D:/data/MNIST', download=True, transform=train\_augs)

mnist\_dataloader = DataLoader(train\_set, batch\_size=128, shuffle=True, num\_workers=4)

# In[11]:

image , label =train\_set[1000]

plt.imshow(image.squeeze(), cmap = 'gray')

# # Гиперпараметры

# In[12]:

device = 'cpu'   # image = image.to(device)

batch\_size = 128  # trainloader,training loop

noise\_dim = 64  # generator model

# optimizer parameters

lr = 0.0002

beta\_1  = 0.5

beta\_2  = 0.99

# training varaibles

epochs = 20

# #  Dataset в Batches

# In[13]:

from torch.utils.data import  DataLoader

from torchvision.utils import make\_grid

#len\_sliced = 12000

#tr\_sliced = torch.utils.data.random\_split(train\_set, [len\_sliced, len(train\_set)-len\_sliced])[0]

trainloader = DataLoader(train\_set, batch\_size=batch\_size, shuffle= True)

print('Total number of batches in train loader', len(trainloader))

# # Discriminator

#

# In[14]:

from torch import nn

from torchsummary import summary

# In[15]:

from torch.nn.modules.activation import LeakyReLU

def get\_disc\_block(in\_channels, out\_channels, kernel\_size, stride):

  return nn.Sequential(

      nn.Conv2d(in\_channels, out\_channels, kernel\_size, stride),

      nn.BatchNorm2d(out\_channels),

      nn.LeakyReLU(0.2)

  )

# In[16]:

class Discriminator(nn.Module):

  def \_\_init\_\_(self):

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

    self.block\_1 = get\_disc\_block(1, 16, (3,3), 2)

    self.block\_2 = get\_disc\_block(16, 32, (5,5), 2)

    self.block\_3 = get\_disc\_block(32, 64, (5,5), 2)

    self.flatten = nn.Flatten()

    self.linear = nn.Linear(in\_features=64, out\_features=1)

  def forward(self,images):

    x1 = self.block\_1(images)

    x2 = self.block\_2(x1)

    x3 = self.block\_3(x2)

    x4 = self.flatten(x3)

    x5 = self.linear(x4)

    return x5

# In[17]:

D = Discriminator()

D.to(device)

summary(D, input\_size=(1,28,28))

# # Generator

# In[18]:

from torch.nn.modules import ReLU

from torch.nn.modules.batchnorm import BatchNorm2d

def get\_gen\_block(in\_channels, out\_channels, kernel\_size, stride, final\_block = False):

  if final\_block == True:

    return nn.Sequential(

        nn.ConvTranspose2d(in\_channels, out\_channels, kernel\_size, stride),

        nn.Tanh()

    )

  return nn.Sequential(

      nn.ConvTranspose2d(in\_channels, out\_channels, kernel\_size, stride),

      nn.BatchNorm2d(out\_channels),

      nn.ReLU()

  )

# In[19]:

class Generator(nn.Module):

  def \_\_init\_\_(self,noise\_dim):

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

    self.noise\_dim = noise\_dim

    self.block\_1 = get\_gen\_block(noise\_dim, 256, (3,3), 2)

    self.block\_2 = get\_gen\_block(256, 128, (4,4), 1)

    self.block\_3 = get\_gen\_block(128, 64, (3,3), 2)

    self.block\_4= get\_gen\_block(64, 1, (4,4), 2, final\_block=True)

  def forward(self, r\_noise\_vec):

     x = r\_noise\_vec.view(-1, self.noise\_dim, 1, 1)

     X1 = self.block\_1(x)

     X2 = self.block\_2(X1)

     X3 = self.block\_3(X2)

     X4 = self.block\_4(X3)

     return X4

# In[20]:

G = Generator(noise\_dim)

G.to(device)

summary(G, input\_size=(1, noise\_dim))

# # Random Weights

# In[21]:

# Replace Random initialized weights to Normal weights

def weights\_init(m):

    if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d):

        nn.init.normal\_(m.weight, 0.0, 0.02)

    if isinstance(m, nn.BatchNorm2d):

        nn.init.normal\_(m.weight, 0.0, 0.02)

        nn.init.constant\_(m.bias, 0)

# In[22]:

D =D.apply(weights\_init)

G = G.apply(weights\_init)

# In[23]:

# 'show\_tensor\_images' : function is used to plot some of images from the batch

def show\_tensor\_images(tensor\_img, num\_images = 16, size=(1, 28, 28)):

    unflat\_img = tensor\_img.detach().cpu()

    img\_grid = make\_grid(unflat\_img[:num\_images], nrow=4)

    plt.imshow(img\_grid.permute(1, 2, 0).squeeze())

    plt.show()

# In[24]:

def real\_loss(disc\_pred):

  criterion = nn.BCEWithLogitsLoss()

  ground\_truth = torch.ones\_like(disc\_pred)

  loss = criterion(disc\_pred, ground\_truth)

  return loss

def fake\_loss(disc\_pred):

  criterion = nn.BCEWithLogitsLoss()

  ground\_truth = torch.zeros\_like(disc\_pred)

  loss = criterion(disc\_pred, ground\_truth)

  return loss

# In[25]:

D\_opt = torch.optim.Adam(D.parameters(), lr=lr, betas=(beta\_1, beta\_2))

G\_opt = torch.optim.Adam(G.parameters(), lr=lr, betas=(beta\_1, beta\_2))

# In[ ]:

from tqdm.notebook import tqdm

for i in range(epochs):

  total\_d\_loss = 0.0

  total\_g\_loss = 0.0

  for  real\_img, \_ in tqdm(trainloader):

    real\_img = real\_img.to(device)

    noise = torch.randn(batch\_size, noise\_dim, device=device)

    D\_opt.zero\_grad()

    fake\_img = G(noise)

    D\_pred = D(fake\_img)

    D\_fake\_loss = fake\_loss(D\_pred)

    D\_pred = D(real\_img)

    D\_real\_loss = real\_loss(D\_pred)

    D\_loss = (D\_fake\_loss + D\_real\_loss)/2

    total\_d\_loss += D\_loss.item()

    D\_loss.backward()

    D\_opt.step()

    G\_opt.zero\_grad()

    oise = torch.randn(batch\_size, noise\_dim, device=device)

    fake\_img = G(noise)

    D\_pred = D(fake\_img)

    # D\_fake\_loss = fake\_loss(D\_pred)

    # D\_pred = D(real\_img)

    # D\_real\_loss = real\_loss(D\_pred)

    G\_loss = real\_loss(D\_pred)

    total\_g\_loss += G\_loss.item()

    G\_loss.backward()

    G\_opt.step()

  avg\_d\_loss = total\_d\_loss/len(trainloader)

  avg\_g\_loss = total\_g\_loss/len(trainloader)

  print("Epoch ; {} | D\_loss : {} | G\_loss {}".format(i+1, avg\_d\_loss, avg\_g\_loss))

  show\_tensor\_images(fake\_img)

# In[ ]:

# In[ ]:

# In[ ]:

# Результат работы программы

# 

# Вывод: В ходе выполнения лабораторной работы была реализовано соревновательная нейросеть генерации рукописных цифр.