



Deep Learning School

Школа глубокого обучения ФПМИ МФТИ

Домашнее задание. Весна 2021

Autoencoders

- ▼ Часть 1. Vanilla Autoencoder (10 баллов)
- ▼ 1.1. Подготовка данных (0.5 балла)

```
import numpy as np
from torch.autograd import Variable
from torchvision import datasets, transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data_utils
import torch
from IPython.display import clear_output
from tqdm import tqdm
```

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import os
import pandas as pd
import skimage.io
from skimage.transform import resize
import random
from sklearn.manifold import TSNE
from sklearn.preprocessing import OneHotEncoder
%matplotlib inline

def fetch_dataset(attrs_name = "lfw_attributes.txt",
                  images_name = "lfw-deepfunneled",
                  dx=80,dy=80,
                  dimx=64,dimy=64
                 ):
    #download if not exists
    if not os.path.exists(images_name):
        print("images not found, downloading...")
        os.system("wget http://vis-www.cs.umass.edu/lfw/lfw-deepfunneled.tgz -O tmp.tgz")
        print("extracting...")
        os.system("tar xvzf tmp.tgz && rm tmp.tgz")
        print("done")
        assert os.path.exists(images_name)

    if not os.path.exists(attrs_name):
        print("attributes not found, downloading...")
        os.system("wget http://www.cs.columbia.edu/CAVE/databases/pubfig/download/%s" % attrs_name)
        print("done")

    #read attrs
    df_attrs = pd.read_csv("lfw_attributes.txt",sep='\t',skiprows=1,)
    df_attrs = pd.DataFrame(df_attrs.iloc[:, :-1].values, columns = df_attrs.columns[1:])

    #read photos
    photo_ids = []
    for dirpath, dirnames, filenames in os.walk(images_name):
        for fname in filenames:
            if fname.endswith(".jpg"):
                fpath = os.path.join(dirpath,fname)
                photo_id = fname[:-4].replace('_', ' ').split()
                person_id = ' '.join(photo_id[:-1])
                photo_number = int(photo_id[-1])
                photo_ids.append({'person':person_id,'imagenum':photo_number,'photo_path':fpath})

    photo_ids = pd.DataFrame(photo_ids)
    # print(photo_ids)
    #mass-merge
    #(photos now have same order as attributes)
    df = pd.merge(df_attrs,photo_ids,on=('person','imagenum'))

    assert len(df)==len(df_attrs),"lost some data when merging dataframes"
```

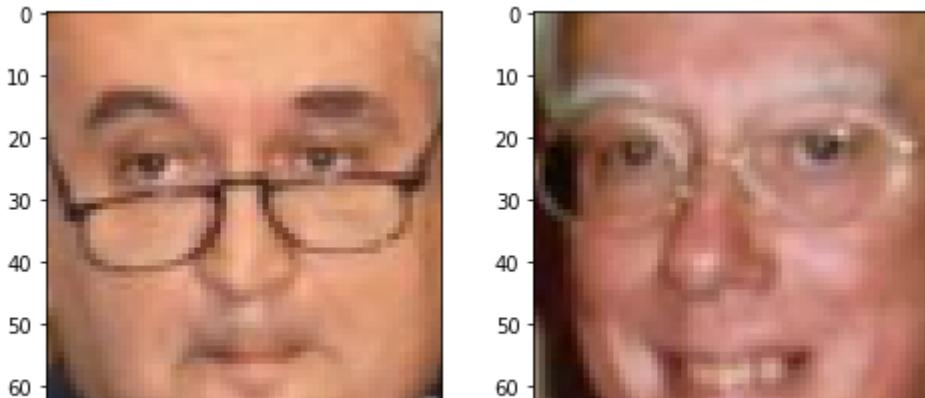
Разбейте выборку картинок на train и val, выведите несколько картинок в output, чтобы посмотреть, как они выглядят, и приведите картинки к тензорам pytorch, чтобы можно было скормить их сети:

```
train_data = torch.FloatTensor(train_data)
val_data = torch.FloatTensor(val_data)
```

```
train_data.shape[0]
```

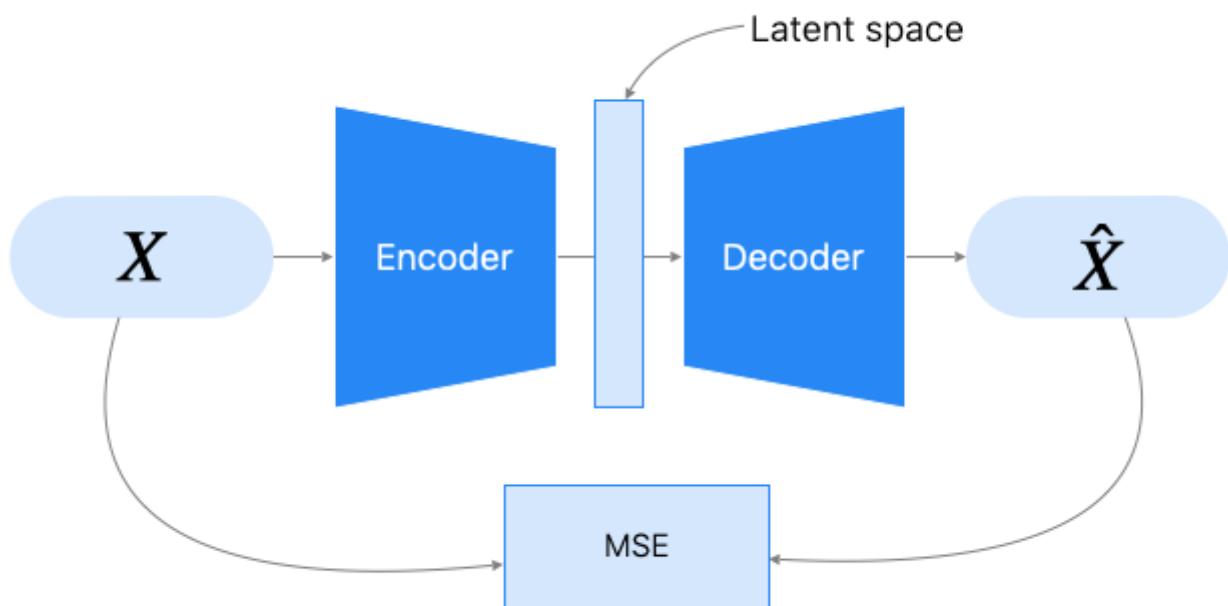
11828

```
plt.figure(figsize=(8, 8))
for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.imshow(train_data[random.randint(0,train_data.shape[0])])
```



▼ 1.2. Архитектура модели (1.5 балла)

В этом разделе мы напишем и обучем обычный автоэнкодер.



^ напомню, что автоэнкодер выглядит вот так

```
dim_code = 64 # выберите размер латентного вектора
```

```
train_data.shape
```

```
torch.Size([11828, 64, 64, 3])
```

Реализуем autoencoder. Архитектуру (conv, fully-connected, ReLu, etc) можете выбирать сами. Экспериментируйте!

```
from copy import deepcopy
```

```
class Autoencoder(nn.Module):
```

```
def __init__(self):
    super(Autoencoder, self).__init__()

    self.flatten = nn.Flatten()

    # encoder
    self.encoder = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5),
        nn.ReLU(),
        nn.Conv2d(16, 32, 5), #(32, 56, 56)
        nn.ReLU(),
        nn.Flatten(),
        nn.Linear(in_features=32*56*56, out_features=512),
        nn.ReLU(),
        nn.Linear(in_features = 512, out_features=dim_code)
    )

    # decoder
    self.decoder = nn.Sequential(
        nn.Linear(in_features=dim_code, out_features=512),
        nn.ReLU(),
        nn.Linear(512, 32*56*56),
        nn.ReLU(),
        nn.Unflatten(1, (32, 56, 56)),
        nn.ConvTranspose2d(32, 16, 5),
        nn.ReLU(),
        nn.ConvTranspose2d(16, 3, 5),
        nn.ReLU()
    )

def forward(self, x):
    #<реализуйте forward проход автоэнкодера
    #в качестве возвращаемых переменных -- латентное представление картинки (latent_code)
    #и полученная реконструкция изображения (reconstruction)>
    latent_code = self.encoder(x.view(-1, 3, 64, 64))
    reconstruction = self.decoder(latent_code).view(-1, 64, 64, 3)

    return reconstruction, latent_code

device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

criterion = F.mse_loss #log_likelihood #<loss>

autoencoder = Autoencoder().to(device)

optimizer = torch.optim.Adam(autoencoder.parameters()) #<Ваш любимый оптимизатор>
```

▼ 1.3 Обучение (2 балла)

Осталось написать код обучения автоэнкодера. При этом было бы неплохо в процессе иногда смотреть, как автоэнкодер реконструирует изображения на данном этапе обучения. Например, после каждой эпохи (прогона train выборки через автоэнкодер) можно смотреть, какие реконструкции получились для каких-то изображений val выборки.

Лучшее было бы использовать график train и val потоков в процессе тренировки –)

```
train_loader = torch.utils.data.DataLoader(train_data, batch_size=32)
```

```
val_loader = torch.utils.data.DataLoader(val_data, batch_size=32)
```

```
#<тут Ваш код тренировки автоэнкодера>
```

```
n_epochs = 17
```

```
train_losses = []
```

```
val_losses = []
```

```
for epoch in tqdm(range(n_epochs)):
```

```
    autoencoder.train()
```

```
    train_losses_per_epoch = []
```

```
    for i, X_batch in enumerate(train_loader):
```

```
        optimizer.zero_grad()
```

```
        reconstructed, latent_code = autoencoder(X_batch.to(device))
```

```
        loss = criterion(reconstructed, X_batch.to(device))
```

```
        loss.backward()
```

```
        optimizer.step()
```

```
        train_losses_per_epoch.append(loss.item())
```

```
    train_losses.append(np.mean(train_losses_per_epoch))
```

```
    autoencoder.eval()
```

```
    val_losses_per_epoch = []
```

```
    with torch.no_grad():
```

```
        for X_batch in val_loader:
```

```
            reconstructed, latent_code = autoencoder(X_batch.to(device))
```

```
            loss = criterion(reconstructed, X_batch.to(device))
```

```
            val_losses_per_epoch.append(loss.item())
```

```
    val_losses.append(np.mean(val_losses_per_epoch))
```

```
    clear_output(wait=True)
```

```
    plt.figure(figsize=(10, 7))
```

```
    plt.plot(np.arange(len(train_losses)), train_losses, label='Train')
```

```
    plt.plot(np.arange(len(val_losses)), val_losses, label='Validation')
```

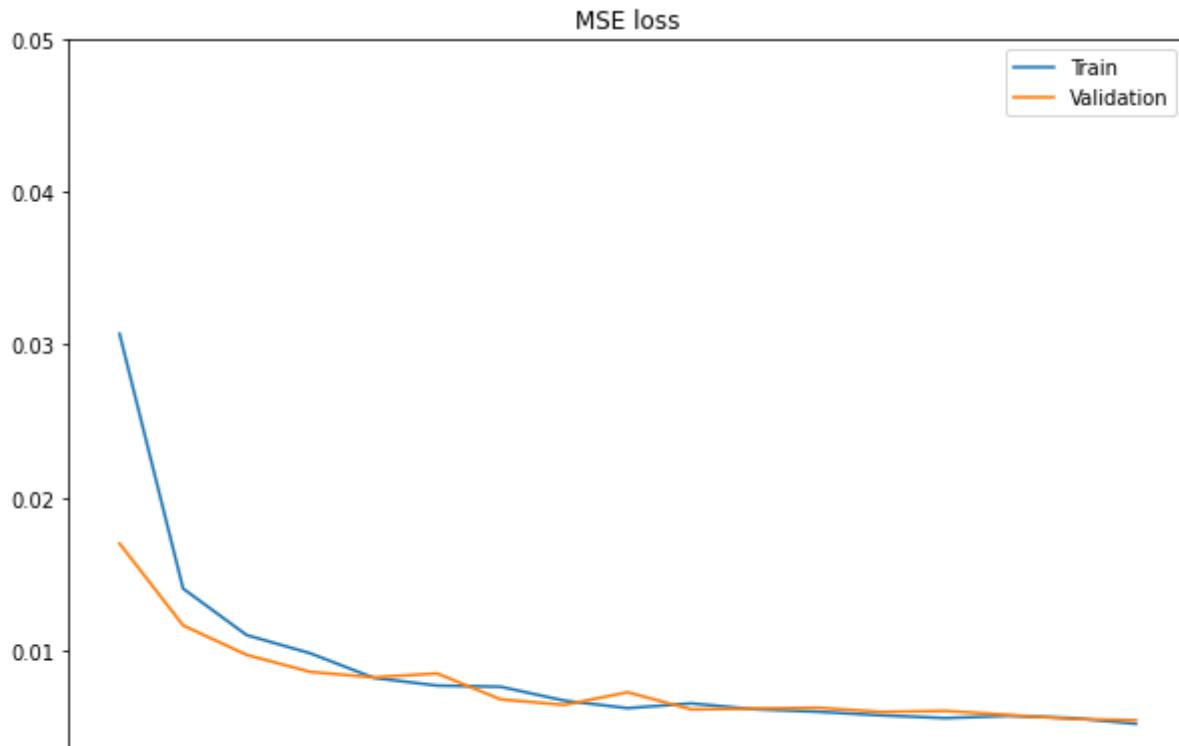
```
    plt.ylim(0, 0.05)
```

```
    plt.xlabel('Epoch')
```

```
    plt.title('MSE loss')
```

```
    plt.legend()
```

```
    plt.show()
```



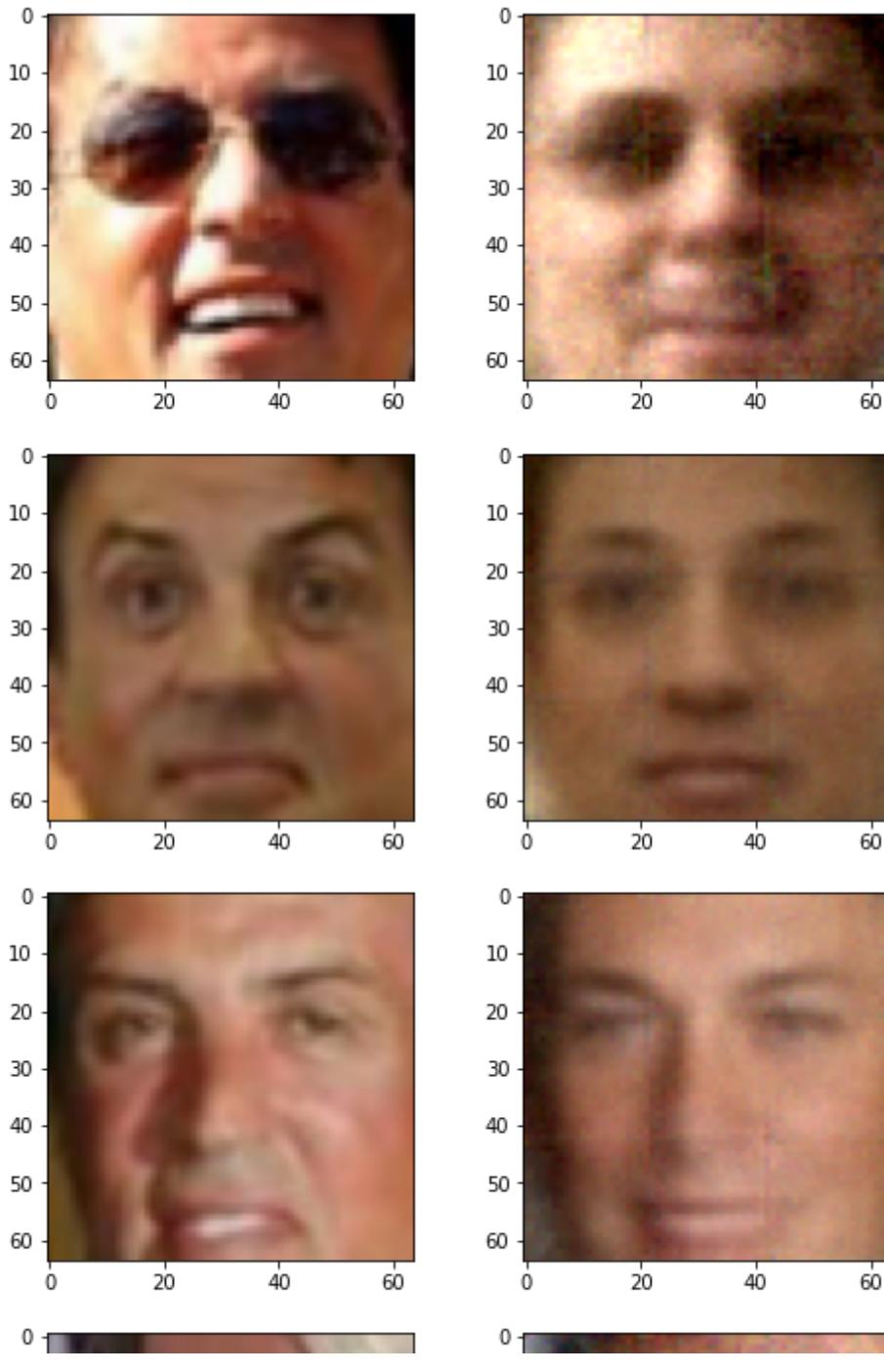
Давайте посмотрим, как наш тренированный автоэнкодер кодирует и восстанавливает картинки:

100% |██████████| 1/1 [12:37/00:00, 44.56s/it]

```
autoencoder.eval()
with torch.no_grad():
    for batch in val_loader:
        reconstruction, latent_code = autoencoder(batch.to(device))
        reconstruction = reconstruction.view(-1, 64, 64, 3)
        result = reconstruction.cpu().detach().numpy()
        ground_truth = batch.numpy()
        break
```

```
#< тут Ваш код: выведите первые X картинок и их реконструкций из val выборки на экран>
for i in range(4):
    plt.figure(figsize=(8, 20))
    for i, (gt, res) in enumerate(zip(ground_truth[:5], result[:5])):
        plt.subplot(5, 2, 2*i+1)
        plt.imshow(gt)
        plt.subplot(5, 2, 2*i+2)
        plt.imshow(res)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers) for floats o
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers) for floats o
<Figure size 576x1440 with 0 Axes>
<Figure size 576x1440 with 0 Axes>
<Figure size 576x1440 with 0 Axes>



Not bad, right?



▼ 1.4. Sampling (2 балла)



Давайте теперь будем не просто брать картинку, пропускать ее через автоэнкодер и получать реконструкцию, а попробуем создать что-то НОВОЕ

Давайте возьмем и подсунем декодеру какие-нибудь сгенерированные нами векторы (например, из нормального распределения) и посмотрим на результат реконструкции

декодера:

Подсказка: Если вместо лиц у вас выводится непонятно что, попробуйте посмотреть, как выглядят латентные векторы картинок из датасета. Так как в обучении нейронных сетей есть определенная доля рандома, векторы латентного слоя могут быть распределены НЕ как `np.random.randn(25, <latent_space_dim>)`. А чтобы у нас получались лица при запихивании вектора декодеру, вектор должен быть распределен так же, как латентные векторы реальных фоток. Так что в таком случае придется рандом немножко подогнать.

```
reconstruction, latent_code = autoencoder.forward(val_data.to(device))

mean = np.mean(latent_code.detach().cpu().numpy(), axis=0)
var = np.var(latent_code.detach().cpu().numpy(), axis=0)

print(mean, var)

[ 9.52399150e-02  1.17698312e-01 -3.32059972e-02  1.24889670e-03
 -2.74257034e-01 -4.23331186e-02 -1.04689173e-01 -4.53306250e-02
 1.49190497e+00  1.21123779e+00  7.89699927e-02  3.07183951e-01
 6.74830526e-02 -2.03955744e-04  1.67848825e-01  2.97297146e-02
 1.16656668e-01  4.76350375e-02  2.44257659e-01 -1.06215030e-01
 1.72106363e-02 -3.60329986e-01  7.62490183e-02  7.60840029e-02
 5.43511696e-02 -1.85692254e-02 -3.40742175e-03  3.26213866e-01
 -1.06369287e-01  2.92477608e-01  6.32237792e-02 -1.49282329e-02
 2.35515907e-02  6.69859290e-01 -3.48517811e-03  5.87009043e-02
 1.55491019e-02 -9.36804488e-02 -7.00640157e-02  1.92846544e-02
 -2.11698934e-01 -6.80662394e-02  8.31915736e-02  1.14518609e-02
 2.83649623e-01 -4.76794317e-03  4.67872396e-02 -8.82588699e-02
 -3.05206478e-01 -1.05528146e-01  2.97575388e-02  3.42187844e-02
 -1.56645179e-01 -1.39193912e-03 -1.29322456e-02 -8.24635699e-02
 1.74805179e-01 -1.42702207e-01 -1.33449510e-02 -2.17495114e-01
 3.07941362e-02 -1.36604682e-02  7.20575601e-02 -9.78003442e-03] [0.13028087 0.1289
 0.09029106 0.11559496 0.43659073 0.3468688 0.13721177 0.1979329
 0.22522901 0.13604216 0.10728371 0.08884541 0.09266768 0.08378241
 0.07539384 0.11143719 0.15252766 0.11588532 0.15674698 0.0935534
 0.08963233 0.08445232 0.11856072 0.06840155 0.11120622 0.11995941
 0.0769902 0.10345089 0.13264053 0.15154779 0.1119609 0.12210708
 0.07803666 0.16825898 0.08458478 0.13431881 0.11112493 0.09314383
 0.14316764 0.11065321 0.07731947 0.15548497 0.16950701 0.06974288
 0.0728558 0.10551724 0.13053198 0.12171343 0.1874642 0.12487765
 0.09188947 0.08213448 0.10701624 0.08640005 0.13877153 0.126614
 0.19795476 0.09203263 0.08851399 0.0719048 ]
```



```
# сгенерируем 25 рандомных векторов размера latent_space
z = np.random.randn(25, dim_code)*var*4.25 + mean
output = autoencoder.decoder(torch.FloatTensor(z).to(device)).view(-1, 64, 64, 3)
plt.figure(figsize=(20, 20))
for i in range(output.shape[0]):
    plt.subplot(5, 5, i + 1)
    generated = output[i].cpu().detach().numpy()
```

```
plt.imshow(generated)
```

```
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers)

▼ Time to make fun! (4 балла)

Давайте научимся пририсовывать людям улыбки =)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers)

so linear

this is you when looking at the HW for the first time



План такой:

1. Нужно выделить "вектор улыбки": для этого нужно из выборки изображений найти несколько (~15) людей с улыбками и столько же без.

Найти людей с улыбками вам поможет файл с описанием датасета, скачанный вместе с датасетом. В нем указаны имена картинок и присутствующие атрибуты (улыбки, очки...)

2. Вычислить латентный вектор для всех улыбающихся людей (прогнать их через encoder) и то же для всех грустненьких
3. Вычислить, собственно, вектор улыбки -- посчитать разность между средним латентным вектором улыбающихся людей и средним латентным вектором грустных людей
4. А теперь приделаем улыбку грустному человеку: добавим полученный в пункте 3 вектор к латентному вектору грустного человека и прогоним полученный вектор через decoder. Получим того же человека, но уже не грустненького!



```
smile = random.sample([i for i, el in enumerate(train_attrs['Smiling'].to_numpy()) > 2] if
```

```
nsmile = random.sample([i for i, el in enumerate(train_attrs['Smiling'].to_numpy()) < -2) if el], len(nsmile))
```

```
20 20
```

```
train_data[smile].shape
```

```
torch.Size([20, 64, 64, 3])
```

```
#<ваш код здесь>
```

```
latent_smile = autoencoder.encoder(train_data[smile].view(-1, 3, 64, 64).to(device))
latent_nsmile = autoencoder.encoder(train_data[nsmile].view(-1, 3, 64, 64).to(device))
```

```
latent_vector_smile = latent_smile - latent_nsmile
```

```
latent_vector_mean_smile = torch.mean(latent_vector_smile, dim = 0)
```

```
z = np.random.randn(20, dim_code)*var*4.5 + mean
```

```
output = autoencoder.decoder(torch.FloatTensor(z).to(device) + latent_vector_mean_smile).view(-1, 3, 64, 64)
```

```
plt.figure(figsize=(20, 20))
```

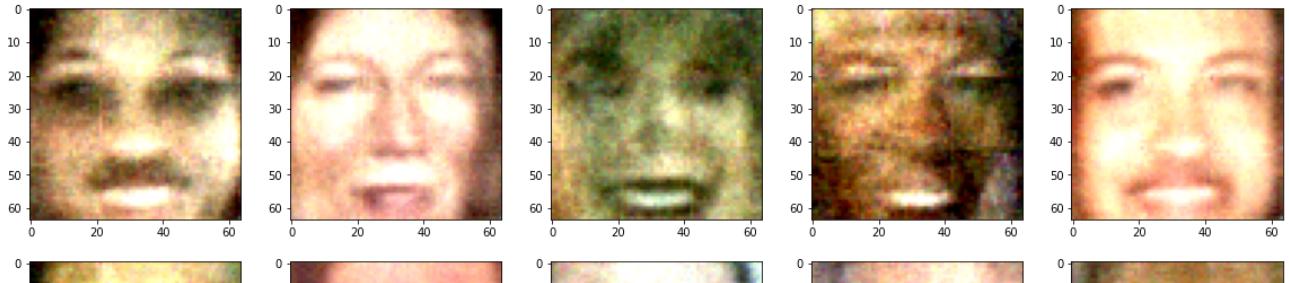
```
for i in range(output.shape[0]):
```

```
    plt.subplot(5, 5, i + 1)
```

```
    generated = output[i].cpu().detach().numpy()
```

```
    plt.imshow(generated)
```

```
plt.show()
```



```
nonsmile = [i for i, el in enumerate(train_attrs['Smiling'].to_numpy() < -2.3) if el == True]
```



```
latent_vector_smile.shape
```

```
torch.Size([20, 64])
```



```
autoencoder.eval()
```

```

latent_code = autoencoder.encoder(train_data[nonsmile].view(-1,3,64,64).to(device))
print(latent_code.shape)
reconstruction = autoencoder.decoder(latent_code).view(-1, 64, 64, 3)
reconstruction_smiling = autoencoder.decoder(latent_code + latent_vector_mean_smile).view(
result = reconstruction.cpu().detach().numpy()
result_smiling = reconstruction_smiling.cpu().detach().numpy()
ground truth = train data[nonsmile].numpy()

```

```
torch.Size([9, 64])
```

```
for i in range(4):
    plt.figure(figsize=(8, 20))
for i, (gt, res, res_smile) in enumerate(zip(ground_truth, result, result_smiling)):
    plt.subplot(9, 3, 3*i+1)
    plt.imshow(gt)
    plt.subplot(9, 3, 3*i+2)
    plt.imshow(res)
    plt.subplot(9, 3, 3*i+3)
    plt.imshow(res_smile)
```

```
plt.subplot(9, 3, 3*i+3)
plt.imshow(res_smile)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers).

Вуаля! Вы восхитительны!

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers).

Теперь вы можете пририсовывать людям не только улыбки, но и много чего другого – закрывать/открывать глаза, пририсовывать очки... в общем, все, на что хватит фантазии и на что есть атрибуты в all_attrs :)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or 0..255 for integers).

▼ Часть 2: Variational Autoencoder (10 баллов)

Займемся обучением вариационных автоэнкодеров – проапгрейженной версии AE. Обучать будем на датасете MNIST, содержащем написанные от руки цифры от 0 до 9



```
batch_size = 32
# MNIST Dataset
train_dataset = datasets.MNIST(root='./mnist_data/', train=True, transform=transforms.ToTensor())
test_dataset = datasets.MNIST(root='./mnist_data/', train=False, transform=transforms.ToTensor())

# Data Loader (Input Pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)

Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw/
9913344/? [00:00<00:00, 13313171.81it/s]

Extracting ./mnist_data/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist_data/MNIST/raw/
Downloaded 9913344 bytes in 0.000 seconds (13313171.81 kB/s)

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw/
29696/? [00:00<00:00, 5581.24it/s]

Extracting ./mnist_data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist_data/MNIST/raw/
Downloaded 29696 bytes in 0.000 seconds (5581.24 kB/s)

Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw/
1649664/? [00:00<00:00, 4311881.84it/s]

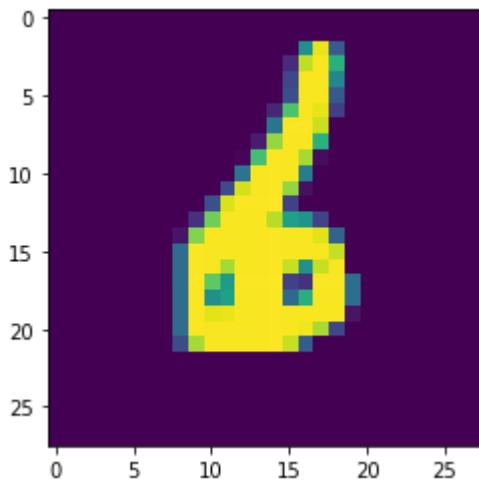
Extracting ./mnist_data/MNIST/raw/train-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw/
Downloaded 1649664 bytes in 0.000 seconds (4311881.84 kB/s)

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./mnist_data/MNIST/raw/
5120/? [00:00<00:00, 8758.30it/s]

Extracting ./mnist_data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./mnist_data/MNIST/raw/
Downloaded 5120 bytes in 0.000 seconds (8758.30 kB/s)

for i in train_loader:
    plt.imshow(i[0][30,:,:,:].view(28,28))
    break
```





▼ 2.1 Архитектура модели и обучение (2 балла)

Реализуем VAE. Архитектуру (conv, fully-connected, ReLu, etc) можете выбирать сами.

Рекомендуем пользоваться более сложными моделями, чем та, что была на семинаре:) Экспериментируйте!

```
features = 16

class VAE(nn.Module):
    def __init__(self):
        #<определите архитектуры encoder и decoder
        #помните, у encoder должны быть два "хвоста",
        #т.е. encoder должен кодировать картинку в 2 переменные -- mu и logsigma>
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 4, 2, 2),
            nn.ReLU(),
            nn.Conv2d(16, 32, 4, 2, 2),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(2048, 128),
            nn.ReLU(),
            nn.Linear(128, 2*features),
        )

        self.decoder = nn.Sequential(
            nn.Linear(features, 128),
            nn.ReLU(),
            nn.Linear(128, 2048),
            nn.ReLU(),
            nn.Unflatten(1, (32, 8, 8)),
            nn.ConvTranspose2d(32, 16, 4, 2, 2),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 6, 2, 2),
            nn.Sigmoid()
        )

    def encode(self, x):
```

```

#<реализуйте forward проход энкодера
#в качестве возвращаемых переменных -- mu и logsigma>
x = self.encoder(x.float()).view(-1, 2, features)
mu = x[:, 0, :]
logsigma = x[:, 1, :]
return mu, logsigma

def gaussian_sampler(self, mu, logsigma):
    if self.training:
        #<засемплируйте латентный вектор из нормального распределения с параметрами mu
        std = torch.exp(0.5 * logsigma) # standard deviation
        eps = torch.randn_like(std) # `randn_like` as we need the same size
        sample = mu + (eps * std) # sampling as if coming from the input space
        return sample
    else:
        # на инференсе возвращаем не случайный вектор из нормального распределения, а
        # на инференсе выход автоэнкодера должен быть детерминирован.
        return mu

def decode(self, z):
    #<реализуйте forward проход декодера
    #в качестве возвращаемой переменной -- reconstruction>
    reconstruction = self.decoder(z)
    return reconstruction

def forward(self, x):
    #<используя encode и decode, реализуйте forward проход автоэнкодера
    # в качестве возвращаемых переменных -- mu, logsigma и reconstruction>
    mu, logsigma = self.encode(x)
    reconstruction = self.decode(self.gaussian_sampler(mu, logsigma))
    return mu, logsigma, reconstruction

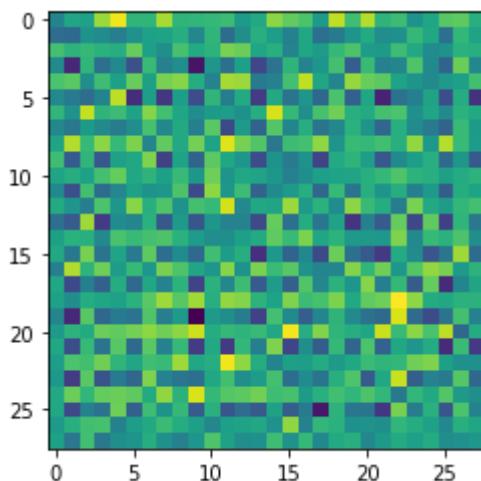
```

```

vae = VAE()
for i in train_loader:
    mu, logsigma, reconstruction = vae.forward(i[0])
    print(reconstruction.shape)
    plt.imshow(reconstruction.detach().numpy()[0,:,:,:].reshape(28,28))
    break

```

`torch.Size([32, 1, 28, 28])`



Определим лосс и его компоненты для VAE:

Надеюсь, вы уже прочитали материал в towardsdatascience (или еще где-то) про VAE и знаете, что лосс у VAE состоит из двух частей: KL и log-likelihood.

Общий лосс будет выглядеть так:

$$\mathcal{L} = -D_{KL}(q_\phi(z|x)||p(z)) + \log p_\theta(x|z)$$

Формула для KL-дивергенции:

$$D_{KL} = -\frac{1}{2} \sum_{i=1}^{\dim Z} (1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2)$$

В качестве log-likelihood возьмем привычную нам кросс-энтропию.

```
def KL_divergence(mu, logsigma):
    """
    часть функции потерь, которая отвечает за "близость" латентных представлений разных лк
    """
    loss = -0.5 * torch.sum(1 + logsigma - mu**2 - torch.exp(logsigma)) #<напишите код для
    return loss

def log_likelihood(x, reconstruction):
    """
    часть функции потерь, которая отвечает за качество реконструкции (как mse в обычном ас
    """
    loss = nn.BCELoss(reduction='sum')
    return loss(reconstruction, x)

def loss_vae(x, mu, logsigma, reconstruction):
    return KL_divergence(mu, logsigma) + log_likelihood(x, reconstruction) #<соедините тут
```

И обучим модель:

```
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

criterion = loss_vae

autoencoder = VAE().to(device)

optimizer = torch.optim.Adam(autoencoder.parameters())#<Ваш любимый оптимизатор>

device

'cuda:0'

n_epochs = 50
train_losses = []
val_losses = []
```

```
for epoch in tqdm(range(n_epochs)):
    autoencoder.train()
    train_losses_per_epoch = []
    for batch in train_loader:
        optimizer.zero_grad()
        mu, logsigma, reconstruction = autoencoder.forward(batch[0].to(device))
        #reconstruction = reconstruction.view(-1, 28, 28, 1)
        loss = criterion(batch[0].to(device).float(), mu, logsigma, reconstruction)
        loss.backward()
        optimizer.step()
        train_losses_per_epoch.append(loss.item())

    train_losses.append(np.mean(train_losses_per_epoch))

    autoencoder.eval()
    val_losses_per_epoch = []
    with torch.no_grad():
        for batch in test_loader:
            mu, logsigma, reconstruction = autoencoder.forward(batch[0].to(device))
            #reconstruction = reconstruction.view(-1, 28, 28, 1)
            loss = criterion(batch[0].to(device).float(), mu, logsigma, reconstruction)
            val_losses_per_epoch.append(loss.item())

#<обучите модель на датасете MNIST>

val_losses.append(np.mean(val_losses_per_epoch))
clear_output(wait=True)
plt.figure(figsize=(10, 7))
plt.plot(np.arange(len(train_losses)), train_losses, label='Train')
plt.plot(np.arange(len(val_losses)), val_losses, label='Validation')

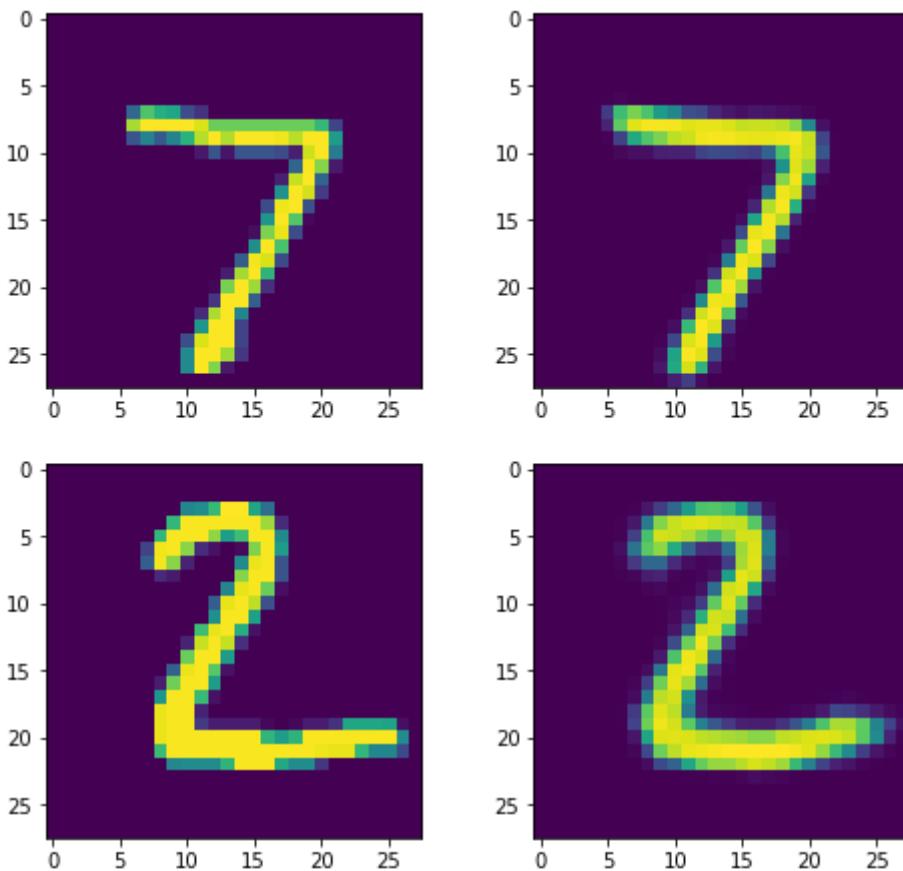
plt.xlabel('Epoch')
plt.title('VAE loss')
plt.legend()
plt.show()
```



Давайте посмотрим, как наш тренированный VAE кодирует и восстанавливает картинки:

```
```py
#< тут Ваш код: выведите первые X картинок и их реконструкций из val выборки на экран>
autoencoder.eval()
with torch.no_grad():
 for batch in test_loader:
 mu, logsigma, reconstruction = autoencoder.forward(batch[0].to(device))
 result = reconstruction.cpu().detach().numpy()
 ground_truth = batch[0].numpy()
 break

plt.figure(figsize=(8, 20))
for i, (gt, res) in enumerate(zip(ground_truth[:5], result[:5])):
 plt.subplot(5, 2, 2*i+1)
 plt.imshow(gt[0,:,:])
 plt.subplot(5, 2, 2*i+2)
 plt.imshow(res[0,:,:])
```

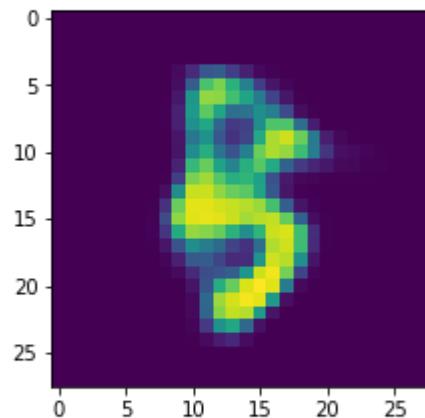
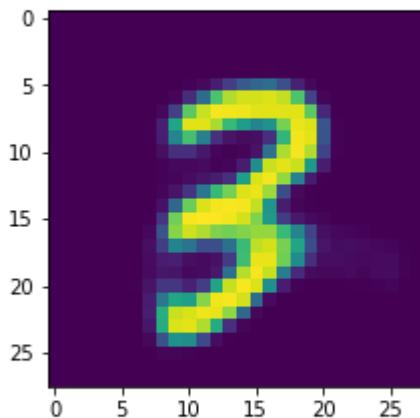
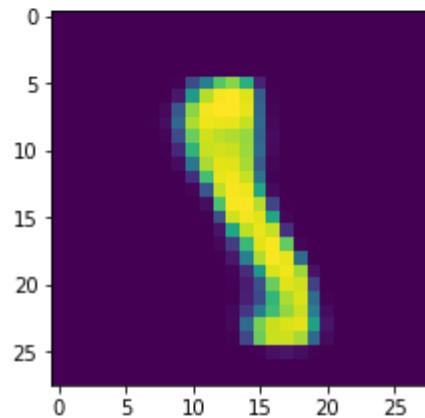
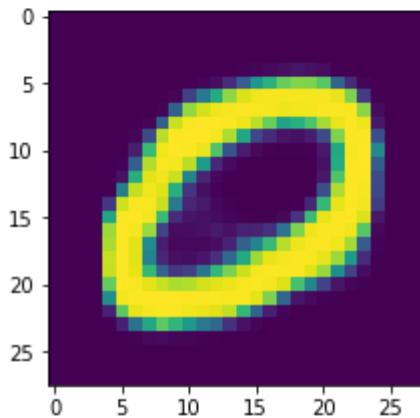
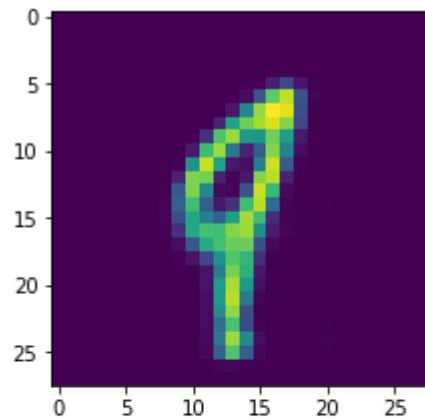
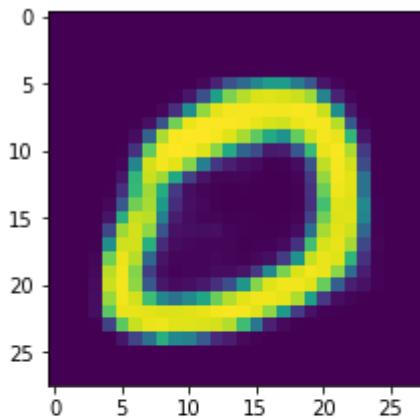


Давайте попробуем проделать для VAE то же, что и с обычным автоэнкодером -- подсунуть decoder'у из VAE случайные векторы из нормального распределения и посмотреть, какие картинки получаются:



```
вспомните про замечание из этого же пункта обычного AE про распределение латентных переменных
z = np.array([np.random.normal(0, 1, 16) for i in range(10)])
output = autoencoder.decode(torch.FloatTensor(z).to(device)) #<скормите z декодеру>
print(output.shape)
#<выведите тут полученные картинки>
plt.figure(figsize=(8, 20))
for i, image in enumerate(output.cpu().detach().numpy()):
 plt.subplot(5,2,i+1)
 plt.imshow(image[0,:,:])
```

```
torch.Size([10, 1, 28, 28])
```



## ▼ 2.2. Latent Representation (2 балла)



Давайте посмотрим, как латентные векторы картинок лиц выглядят в пространстве.  
Ваша задача – изобразить латентные векторы картинок точками в двумерном пространстве.

Это позволит оценить, насколько плотно распределены латентные векторы изображений цифр в пространстве.

Плюс давайте сделаем такую вещь: покрасим точки, которые соответствуют картинкам каждой цифры, в свой отдельный цвет

Подсказка: красить -- это просто =) У plt.scatter есть параметр с (color), см. в документации.

Итак, план:

1. Получить латентные представления картинок тестового датасета
2. С помощью TSNE (есть в sklearn) сжать эти представления до размерности 2 (чтобы можно было их визуализировать точками в пространстве)
3. Визуализировать полученные двумерные представления с помощью matplotlib.scatter, покрасить разными цветами точки, соответствующие картинкам разных цифр.

```
#<ваш код получения латентных представлений, применения TSNE и визуализации>
mu, logsigma = autoencoder.encode(train_dataset.data[:,None,:,:].to(device))
latent_code = autoencoder.gaussian_sampler(mu, logsigma)

latent_code = latent_code.cpu().detach().numpy()

latent_code_embedded = TSNE().fit_transform(latent_code)

/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning
 FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning
 FutureWarning,
```



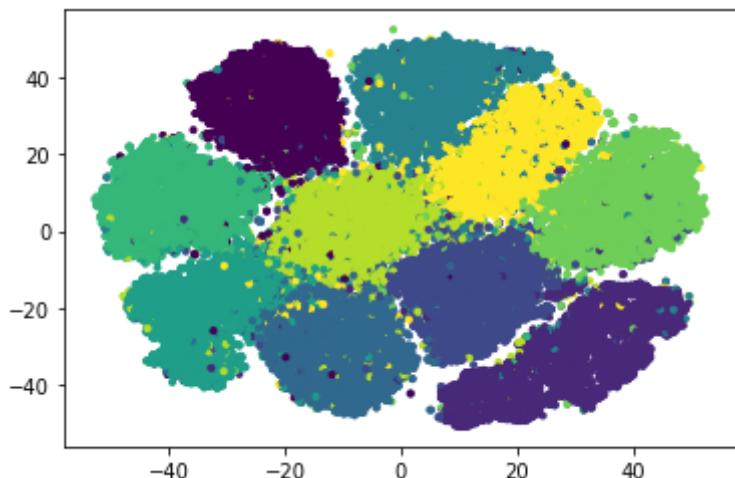
```
latent_code_embedded.shape
```

```
(60000, 2)
```

```
target = train_dataset.targets.numpy()
```

```
plt.scatter(latent_code_embedded[:,0], latent_code_embedded[:,1], c=target, marker='.'
```

```
<matplotlib.collections.PathCollection at 0x7f8cb4bef2d0>
```



Что вы думаете о виде латентного представления?

Как видим признаки вполне хорошо отделяются друг от друга, хоть кое-где и есть "выбросы", но это может быть как проблемой енкодера, так и проблемой отображения в 2д пространство, так что сказать наверняка не получится

Congrats v2.0!

## ▼ 2.3. Conditional VAE (6 баллов)

Мы уже научились обучать обычный АЕ на датасете картинок и получать новые картинки, используя генерацию шума и декодер. Давайте теперь допустим, что мы обучили АЕ на датасете MNIST и теперь хотим генерировать новые картинки с числами с помощью декодера (как выше мы генерили рандомные лица). И вот нам понадобилось сгенерировать цифру 8, и мы подставляем разные варианты шума, но восьмерка никак не генерится:(

Хотелось бы добавить к нашему АЕ функцию "выдай мне рандомное число из вот этого вот класса", где классов десять (цифры от 0 до 9 образуют десять классов). Conditional АЕ – так называется вид автоэнкодера, который предоставляет такую возможность. Ну, название "conditional" уже говорит само за себя.

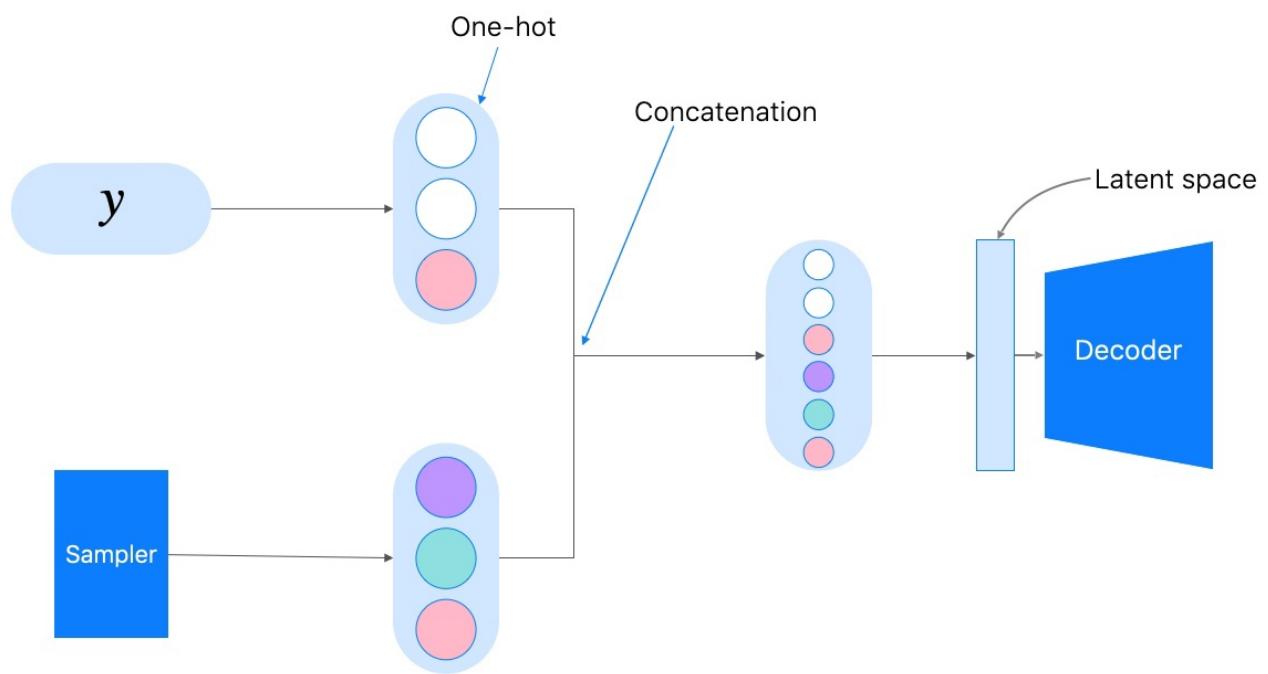
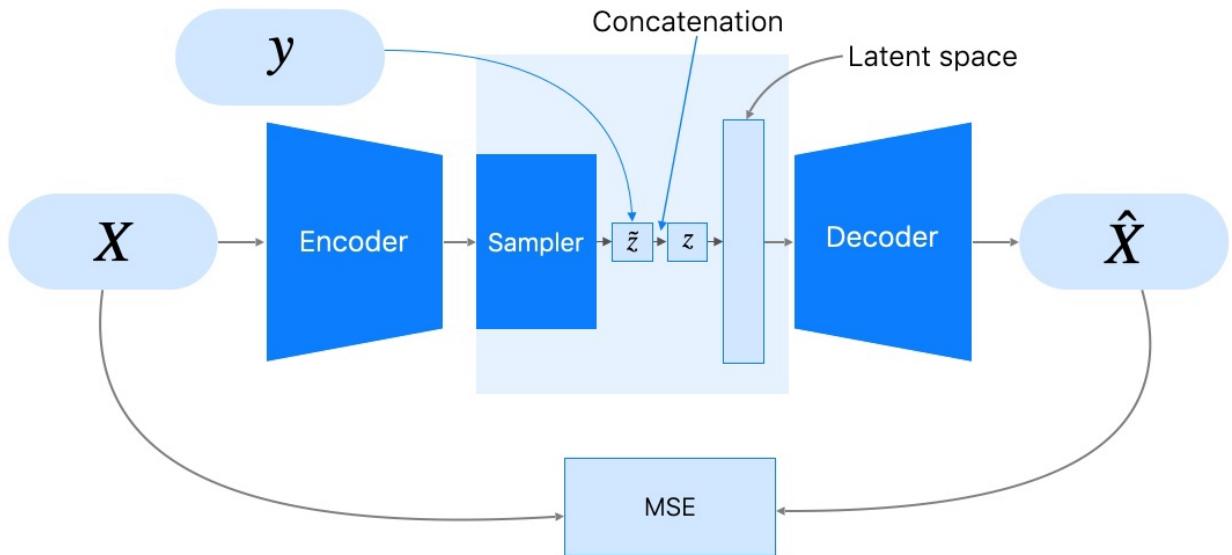
И в этой части задания мы научимся такие обучать.

## ▼ Архитектура

На картинке ниже представлена архитектура простого Conditional VAE.

По сути, единственное отличие от обычного – это то, что мы вместе с картинкой в первом слое энкодера и декодера передаем еще информацию о классе картинки.

То есть, в первый (входной) слой энкодера подается конкатенация картинки и информации о классе (например, вектора из девяти нулей и одной единицы). В первый слой декодера подается конкатенация латентного вектора и информации о классе.



На всякий случай: это VAE, то есть, latent у него все еще состоит из ти и sigma

Таким образом, при генерации новой рандомной картинки мы должны будем передать декодеру сконкатенированные латентный вектор и класс картинки.

P.S. Также можно передавать класс картинки не только в первый слой, но и в каждый слой сети. То есть на каждом слое конкетенировать выход из предыдущего слоя и информацию о классе.

```
features = 16
```

```
class CVAE(nn.Module):
 def __init__(self):
 #<определите архитектуры encoder и decoder
 #помните, у encoder должны быть два "хвоста",
 #т.е. encoder должен кодировать картинку в 2 переменные -- mu и logsigma>
 super(CVAE, self).__init__()
 self.encoder = nn.Sequential(
 nn.Conv2d(1, 16, 4, 2, 2),
 nn.ReLU(),
 nn.Conv2d(16, 32, 4, 2, 2),
 nn.ReLU(),
 nn.Flatten(),
 nn.Linear(2048, 128),
 nn.ReLU(),
 nn.Linear(128, 2*features),
)

 self.decoder = nn.Sequential(
 nn.Linear(10 + features, 128),
 nn.ReLU(),
 nn.Linear(128, 2048),
 nn.ReLU(),
 nn.Unflatten(1, (32, 8, 8)),
 nn.ConvTranspose2d(32, 16, 4, 2, 2),
 nn.ReLU(),
 nn.ConvTranspose2d(16, 1, 6, 2, 2),
 nn.Sigmoid()
)

 def encode(self, x, class_num):
 #<реализуйте forward проход энкодера
 #в качестве возвращаемых переменных -- mu, logsigma и класс картинки>
 x = self.encoder(x.float()).view(-1, 2, features)
 mu = x[:, 0, :]
 logsigma = x[:, 1, :]
 return mu, logsigma, class_num

 def gaussian_sampler(self, mu, logsigma):
 if self.training:
 #<засемплируйте латентный вектор из нормального распределения с параметрами mu
 std = torch.exp(0.5 * logsigma) # standard deviation
 eps = torch.randn_like(std) # `randn_like` as we need the same size
 sample = mu + (eps * std) # sampling as if coming from the input space
 return sample
 else:
 # на инференсе возвращаем не случайный вектор из нормального распределения, а
 # на инференсе выход автоэнкодера должен быть детерминирован.
 return mu

 def decode(self, z, class_num):
 #<реализуйте forward проход декодера
 #в качестве возвращаемой переменной -- reconstruction>
 z = torch.cat((z, class_num), 1)
 reconstruction = self.decoder(z)
 return reconstruction
```

```

def forward(self, x, class_num):#x[0] - dataset, x[1] - targets
 #используя encode и decode, реализуйте forward проход автоэнкодера
 #в качестве возвращаемых переменных -- mu, logsigma и reconstruction
 mu, logsigma, class_num = self.encode(x, class_num)
 reconstruction = self.decode(self.gaussian_sampler(mu, logsigma), class_num)
 return mu, logsigma, reconstruction

```

## ▼ Sampling

Тут мы будем сэмплировать из CVAE. Это прикольнее, чем сэмплировать из простого AE/VAE: тут можно взять один и тот же латентный вектор и попросить CVAE восстановить из него картинки разных классов! Для MNIST вы можете попросить CVAE восстановить из одного латентного вектора, например, картинки цифры 5 и 7.

```

device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

criterion = loss_vae

autoencoder = CVAE().to(device)

optimizer = torch.optim.Adam(autoencoder.parameters())#<Ваш любимый оптимизатор>

train = np.array([0,1, 2, 3 ,4, 5, 6, 7, 8, 9]).reshape(-1,1)
enc = OneHotEncoder(sparse=False)
enc.fit(train)

OneHotEncoder(sparse=False)

for batch in train_loader:
 x = batch[0].to(device)
 num_class = torch.IntTensor(enc.transform(np.array(batch[1]).reshape(-1,1))).to(device)
 print(x.shape, num_class.shape)
 autoencoder.forward(x,num_class)
 break

 torch.Size([32, 1, 28, 28]) torch.Size([32, 10])

#<тут нужно научиться сэмплировать из декодера цифры определенного класса>
n_epochs = 50
train_losses = []
val_losses = []

for epoch in tqdm(range(n_epochs)):
 autoencoder.train()
 train_losses_per_epoch = []
 for batch in train_loader:
 x = batch[0].to(device)
 num_class = torch.IntTensor(enc.transform(np.array(batch[1]).reshape(-1,1))).to(de

```

```
optimizer.zero_grad()
mu, logsigma, reconstruction = autoencoder.forward(x, num_class)
loss = criterion(x.float(), mu, logsigma, reconstruction)
loss.backward()
optimizer.step()
train_losses_per_epoch.append(loss.item())

train_losses.append(np.mean(train_losses_per_epoch))

autoencoder.eval()
val_losses_per_epoch = []
with torch.no_grad():
 for batch in test_loader:
 x = batch[0].to(device)
 num_class = torch.IntTensor(enc.transform(np.array(batch[1]).reshape(-1,1))).to(device)
 mu, logsigma, reconstruction = autoencoder.forward(x, num_class)
 loss = criterion(x.float(), mu, logsigma, reconstruction)
 val_losses_per_epoch.append(loss.item())

#<обучите модель на датасете MNIST>

val_losses.append(np.mean(val_losses_per_epoch))
clear_output(wait=True)
plt.figure(figsize=(10, 7))
plt.plot(np.arange(len(train_losses)), train_losses, label='Train')
plt.plot(np.arange(len(val_losses)), val_losses, label='Validation')

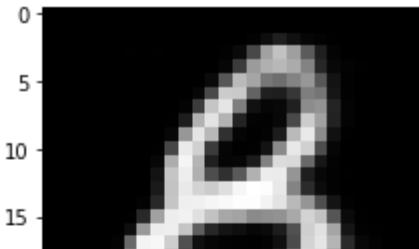
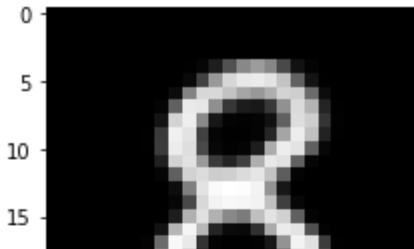
plt.xlabel('Epoch')
plt.title('VAE loss')
plt.legend()
plt.show()
```

**MSE loss**

```
z = np.array([np.random.normal(0, 1, 16) for i in range(8)])
num_class = torch.IntTensor(enc.transform(np.array([8]*8).reshape(-1,1)))
output = autoencoder.decode(torch.FloatTensor(z).to(device), num_class.to(device)) #скорр
print(output.shape)

#<выведите тут полученные картинки>
plt.figure(figsize=(8, 16))
for i, image in enumerate(output.cpu().detach().numpy()):
 plt.subplot(4,2,i+1)
 plt.imshow(image[0,:,:], cmap='gray')
```

```
torch.Size([8, 1, 28, 28])
```

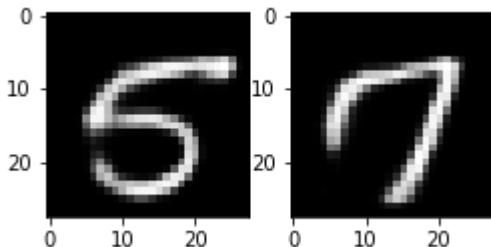


```
z = np.array([np.random.normal(0, 1, 16) for i in range(1)])
num_class_5 = torch.IntTensor(enc.transform(np.array([5]).reshape(-1,1)))
num_class_7 = torch.IntTensor(enc.transform(np.array([7]).reshape(-1,1)))
output_5 = autoencoder.decode(torch.FloatTensor(z).to(device), num_class_5.to(device)) #<<
output_7 = autoencoder.decode(torch.FloatTensor(z).to(device), num_class_7.to(device)) #<<
```

#<выведите тут полученные картинки>

```
plt.figure(figsize=(4, 8))
plt.subplot(1, 2, 1)
plt.imshow(output_5.cpu().detach().numpy()[0, 0, :, :], cmap='gray')
plt.subplot(1, 2, 2)
plt.imshow(output_7.cpu().detach().numpy()[0, 0, :, :], cmap='gray')
```

<matplotlib.image.AxesImage at 0x7f8c91690690>



Splendid! Вы великолепны!



## ▼ Latent Representations



Давайте посмотрим, как выглядит латентное пространство картинок в CVAE и сравним с картинкой для VAE =)

Опять же, нужно покрасить точки в разные цвета в зависимости от класса.



```
targets = torch.IntTensor(enc.transform(np.array(train_dataset.targets).reshape(-1,1)))
print(targets.shape, train_dataset.data.shape)
```

```
torch.Size([60000, 10]) torch.Size([60000, 28, 28])
```

#<ваш код получения латентных представлений, применения TSNE и визуализации>

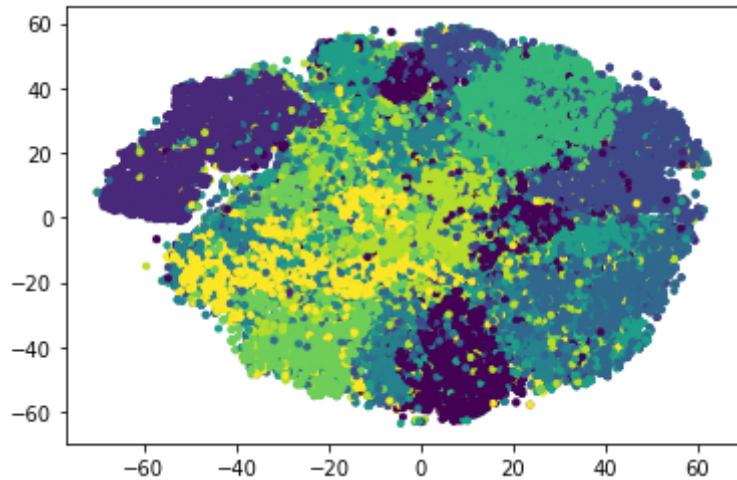
```
targets = torch.IntTensor(enc.transform(np.array(train_dataset.targets).reshape(-1,1)))
mu, logsigma, class_num = autoencoder.encode(train_dataset.data[:, None, :, :].to(device), targets)
latent_code = autoencoder.gaussian_sampler(mu, logsigma)
latent_code = latent_code.cpu().detach().numpy()
```

```
latent_code_embedded = TSNE().fit_transform(latent_code)
target = train_dataset.targets.numpy()

/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning
 FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning
 FutureWarning,
```

```
plt.scatter(latent_code_embedded[:,0], latent_code_embedded[:,1], c=target, marker='.'
```

```
<matplotlib.collections.PathCollection at 0x7f8c917a4f50>
```



Что вы думаете насчет этой картинки? Отличается от картинки для VAE?

Классы здесь менее различимы, но здесь не учтен one-hot вектор, который конкатенируется с латентным вектором CVAE. Также эта модель(как и предыдущая) явно недообучены, но обучались они одинаковое количество эпох. Если говорить конкретно об этом разделении, то на первый взгляд оно очень плохое т.к. классы сильно накладываются друг на друга, но ведь именно это и позволяют сгенерировать из одного латентного вектора разные цифры

## ‣ BONUS 1: Denoising

Внимание! За бонусы доп. баллы не ставятся, но вы можете сделать их для себя.

```
[] ↓ 7 cells hidden
```

## ‣ BONUS 2: Image Retrieval

**Внимание! За бонусы доп. баллы не ставятся, но вы можете сделать их для себя.**

[ ]  $\downarrow$  8 cells hidden

