**Московский государственный технический университет им. Н.Э. Баумана**

Кафедра «Системы обработки информации и управления»

**Лабораторная работа №5**

**по курсу «Проектирование интеллектуальных систем»**

**Выполнил:**

**Хотин П.Ю.**

**ИУ5-24М**

Москва, 2020 год

**Импорт библиотек**

import tensorflow as tf  
  
import os  
import time  
import numpy as np  
import glob  
import matplotlib.pyplot as plt  
import PIL  
  
from IPython import display

**Загрузка данных MNIST**

(train\_images, \_), (test\_images, \_) = tf.keras.datasets.mnist.load\_data()

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz  
11493376/11490434 [==============================] - 5s 0us/step

**Подготовка данных и нормализация датасета**

train\_images = train\_images.reshape(train\_images.shape[0], 28, 28, 1).astype('float32')  
test\_images = test\_images.reshape(test\_images.shape[0], 28, 28, 1).astype('float32')  
  
*# Normalizing the images to the range of [0., 1.]*  
train\_images /= 255.  
test\_images /= 255.  
  
*# Binarization*  
train\_images[train\_images >= .5] = 1.  
train\_images[train\_images < .5] = 0.  
test\_images[test\_images >= .5] = 1.  
test\_images[test\_images < .5] = 0.

**Разбиение на партии и перемешивание выборок**

TRAIN\_BUF = 60000  
BATCH\_SIZE = 100  
  
TEST\_BUF = 10000  
  
train\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_images).shuffle(TRAIN\_BUF).batch(BATCH\_SIZE)  
test\_dataset = tf.data.Dataset.from\_tensor\_slices(test\_images).shuffle(TEST\_BUF).batch(BATCH\_SIZE)

**Создание класса енкодера-декодера**

**class** CVAE(tf.keras.Model):  
 **def** \_\_init\_\_(self, latent\_dim):  
 super(CVAE, self).\_\_init\_\_()  
 self.latent\_dim = latent\_dim  
 self.inference\_net = tf.keras.Sequential(  
 [  
 tf.keras.layers.InputLayer(input\_shape=(28, 28, 1)),  
 tf.keras.layers.Conv2D(  
 filters=32, kernel\_size=3, strides=(2, 2), activation='relu'),  
 tf.keras.layers.Conv2D(  
 filters=64, kernel\_size=3, strides=(2, 2), activation='relu'),  
 tf.keras.layers.Flatten(),  
 *# No activation*  
 tf.keras.layers.Dense(latent\_dim + latent\_dim),  
 ]  
 )  
  
 self.generative\_net = tf.keras.Sequential(  
 [  
 tf.keras.layers.InputLayer(input\_shape=(latent\_dim,)),  
 tf.keras.layers.Dense(units=7\*7\*32, activation=tf.nn.relu),  
 tf.keras.layers.Reshape(target\_shape=(7, 7, 32)),  
 tf.keras.layers.Conv2DTranspose(  
 filters=64,  
 kernel\_size=3,  
 strides=(2, 2),  
 padding="SAME",  
 activation='relu'),  
 tf.keras.layers.Conv2DTranspose(  
 filters=32,  
 kernel\_size=3,  
 strides=(2, 2),  
 padding="SAME",  
 activation='relu'),  
 *# No activation*  
 tf.keras.layers.Conv2DTranspose(  
 filters=1, kernel\_size=3, strides=(1, 1), padding="SAME"),  
 ]  
 )  
  
 @tf.function  
 **def** sample(self, eps=None):  
 **if** eps is None:  
 eps = tf.random.normal(shape=(100, self.latent\_dim))  
 **return** self.decode(eps, apply\_sigmoid=True)  
  
 **def** encode(self, x):  
 mean, logvar = tf.split(self.inference\_net(x), num\_or\_size\_splits=2, axis=1)  
 **return** mean, logvar  
  
 **def** reparameterize(self, mean, logvar):  
 eps = tf.random.normal(shape=mean.shape)  
 **return** eps \* tf.exp(logvar \* .5) + mean  
  
 **def** decode(self, z, apply\_sigmoid=False):  
 logits = self.generative\_net(z)  
 **if** apply\_sigmoid:  
 probs = tf.sigmoid(logits)  
 **return** probs  
  
 **return** logits

**Определение оптимизатора и функции вычисления градиента**

optimizer = tf.keras.optimizers.Adam(1e-4)  
  
**def** log\_normal\_pdf(sample, mean, logvar, raxis=1):  
 log2pi = tf.math.log(2. \* np.pi)  
 **return** tf.reduce\_sum(  
 -.5 \* ((sample - mean) \*\* 2. \* tf.exp(-logvar) + logvar + log2pi),  
 axis=raxis)  
  
@tf.function  
**def** compute\_loss(model, x):  
 mean, logvar = model.encode(x)  
 z = model.reparameterize(mean, logvar)  
 x\_logit = model.decode(z)  
  
 cross\_ent = tf.nn.sigmoid\_cross\_entropy\_with\_logits(logits=x\_logit, labels=x)  
 logpx\_z = -tf.reduce\_sum(cross\_ent, axis=[1, 2, 3])  
 logpz = log\_normal\_pdf(z, 0., 0.)  
 logqz\_x = log\_normal\_pdf(z, mean, logvar)  
 **return** -tf.reduce\_mean(logpx\_z + logpz - logqz\_x)  
  
@tf.function  
**def** compute\_apply\_gradients(model, x, optimizer):  
 **with** tf.GradientTape() as tape:  
 loss = compute\_loss(model, x)  
 gradients = tape.gradient(loss, model.trainable\_variables)  
 optimizer.apply\_gradients(zip(gradients, model.trainable\_variables))

**Задание количества эпох, размера скрытого пространства**

epochs = 30  
latent\_dim = 2  
num\_examples\_to\_generate = 16  
  
*# случайной вектор для генерации (предсказаний)*  
random\_vector\_for\_generation = tf.random.normal(  
 shape=[num\_examples\_to\_generate, latent\_dim])  
model = CVAE(latent\_dim)

**Сохранение и вывод изображений**

**def** generate\_and\_save\_images(model, epoch, test\_input):  
 predictions = model.sample(test\_input)  
 fig = plt.figure(figsize=(4,4))  
  
 **for** i in range(predictions.shape[0]):  
 plt.subplot(4, 4, i+1)  
 plt.imshow(predictions[i, :, :, 0], cmap='gray')  
 plt.axis('off')  
  
 plt.savefig('image\_at\_epoch\_{:04d}.png'.format(epoch))  
 plt.show()

**Начало обучения модели**

generate\_and\_save\_images(model, 0, random\_vector\_for\_generation)  
  
**for** epoch in range(1, epochs + 1):  
 start\_time = time.time()  
 **for** train\_x in train\_dataset:  
 compute\_apply\_gradients(model, train\_x, optimizer)  
 end\_time = time.time()  
  
 **if** epoch % 1 == 0:  
 loss = tf.keras.metrics.Mean()  
 **for** test\_x in test\_dataset:  
 loss(compute\_loss(model, test\_x))  
 elbo = -loss.result()  
 display.clear\_output(wait=False)  
 print('Epoch: {}, Test set ELBO: {}, '  
 *'time elapse for current epoch {}'*.format(epoch,  
 elbo,  
 end\_time - start\_time))  
 generate\_and\_save\_images(  
 model, epoch, random\_vector\_for\_generation)

Epoch: 30, Test set ELBO: -154.27947998046875, time elapse for current epoch 36.571539878845215

**Функция создания сетки из 25 изображений, где по оси Х изменяется значение первого элемента z, а по оси Y - второго элемента z**

n = 20  
x\_axis = np.linspace(-3, 3, n)  
y\_axis = np.linspace(-3, 3, n)  
  
canvas = np.empty((28 \* n, 28 \* n))  
**def** draw\_manifold(model):  
 **for** i, yi in enumerate(x\_axis):  
 **for** j, xi in enumerate(y\_axis):  
 z\_mu = np.array([[xi, yi]] \* BATCH\_SIZE)  
 x\_mean = model.sample(z\_mu).numpy()  
 canvas[(n - i - 1) \* 28:(n - i) \* 28, j \* 28:(j + 1) \* 28] = \  
 x\_mean[0].reshape(28, 28)  
  
 plt.figure(figsize=(8, 10))  
 Xi, Yi = np.meshgrid(x\_axis, y\_axis)  
 plt.imshow(canvas, origin="upper", cmap="gray")  
 plt.show()

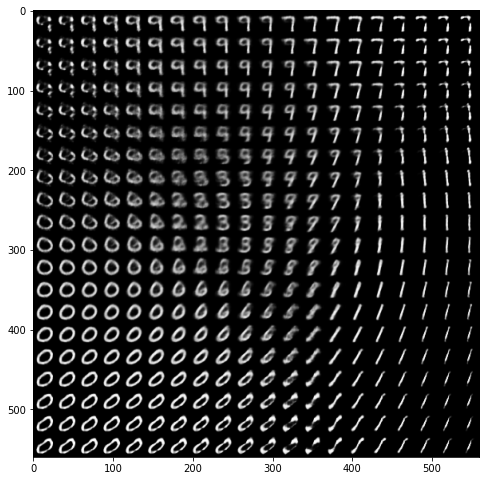
**Запуск функции**

draw\_manifold(model)

WARNING:tensorflow:Layer dense\_3 is casting an input tensor from dtype float64 to the layer's dtype of float32, which is new behavior in TensorFlow 2. The layer has dtype float32 because it's dtype defaults to floatx.  
  
If you intended to run this layer in float32, you can safely ignore this warning. If in doubt, this warning is likely only an issue if you are porting a TensorFlow 1.X model to TensorFlow 2.  
  
To change all layers to have dtype float64 by default, call `tf.keras.backend.set\_floatx('float64')`. To change just this layer, pass dtype='float64' to the layer constructor. If you are the author of this layer, you can disable autocasting by passing autocast=False to the base Layer constructor.

**Список литературы**

[1] Google. Tensorflow. 2018. Apr. url - https://www.tensorflow.org/api\_docs/python/tf/train/Saver.

[2] Google. TensorBoard. 2018. Apr. url - https://www.tensorflow.org/programmers\_guide/summaries\_and\_tensorboard.