Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторная работа №5 по дисциплине

«Проектирование интеллектуальных систем»

ИСПОЛНИТЕЛЬ:

Чечнев А.А.

Группа ИУ5-23М

_____

"__"_____ 2020 г.

Москва 2020

1. Задание

1.1. Создать вариационный автоэнкодер с использованием сверток (Conv2d) в энкодере (слои отвечающие за среднее и отклонение остаются полносвязными), и с развертками (Conv2dTranspose) в декодере. Размерность скрытого вектора равна двум

1.2. Создать сетку из 25 изображений, где по оси X изменяется значение первого элемента **z**, а по оси Y - второго элемента **z**

2. Решение

Импорт библиотек

```
from packaging import version

import tensorflow as tf
from tensorflow import keras
import datetime as dt

print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >= 2, \
    "This notebook requires TensorFlow 2.0 or above."
```

```
from packaging import version
import tensorflow as tf
from tensorflow import keras
import datetime as dt

print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >= 2, \
    "This notebook requires TensorFlow 2.0 or above."
```

Загрузка данных MNIST и создание Dataset.

```
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()


train_dataset = tf.data.Dataset.from_tensor_slices((x_train)).batch(32).shuff
le(10000)
train_dataset = train_dataset.map(lambda x: (tf.math.divide(tf.cast(x, tf.flo
at32), 255.0)))
#Бинаризация
train_dataset = train_dataset.map(lambda x: (tf.round(x)))
train_dataset = train_dataset.map(lambda x: (tf.image.random_flip_left_right(
x)))

test_dataset = tf.data.Dataset.from_tensor_slices((x_test)).batch(32)
test_dataset = test_dataset.map(lambda x: (tf.math.divide(tf.cast(x, tf.float
32), 255.0)))
#Бинаризация
test_dataset = test_dataset.map(lambda x: (tf.round(x)))
```

Класс энкодер

```
class MnistEncoder(keras.Model):
    def __init__(self):
        super(MnistEncoder, self).__init__(name='mnist_encoder')
        with tf.device('/device:GPU:0'):
            self.flatten = keras.layers.Reshape(target_shape=(28, 28, 1))
            self.conv2d_1 = keras.layers.Conv2D(filters=32,
                                                kernel_size = (3,3),
                                                padding='same',
                                                activation='relu')
            self.mxpool = keras.layers.MaxPool2D(pool_size=(2, 2))
            self.flatten2 = keras.layers.Flatten()
            self.fc2 = keras.layers.Dense(64,
                                          activation=tf.nn.relu,
                                          kernel_initializer=tf.initializer
s.RandomNormal,
                                          kernel_regularizer=keras.regulari
zers.l2(l=0.001))
            self.dropout = keras.layers.Dropout(0.5)
            self.fc_mu = keras.layers.Dense(2,
                                            activation=None,
                                            kernel_initializer=tf.initiali
zers.RandomNormal,
                                            kernel_regularizer=keras.regul
arizers.l2(l=0.001))
            self.fc_var = keras.layers.Dense(2,
                                             activation=None,
                                             kernel_initializer=tf.initial
izers.RandomNormal,
                                             kernel_regularizer=keras.regu
larizers.l2(l=0.001))


    def call(self, x, training=False):
        if training:
            x = self.dropout(
                  self.fc2(
                      self.flatten2(
                          self.mxpool(
                              self.conv2d_1(
                                 self.flatten(x) ) ) ) ) )

        else:
            x = self.fc2(
                  self.flatten2(
                      self.mxpool(
                          self.conv2d_1(
                              self.flatten(x) ) ) ) )
        mu = self.fc_mu(x)
        var = self.fc_var(x)
        return mu, var
```

Класс декодер

```
class MnistDecoder(keras.Model):
  def __init__(self):
    super(MnistDecoder, self).__init__(name='mnist_decoder')
    with tf.device('/device:GPU:0'):
      self.decoder_input = keras.layers.InputLayer(input_shape=(2,))
      self.fc1 = keras.layers.Dense(units=7*7*32,
                                    activation="relu")
      self.reshape = keras.layers.Reshape(target_shape=(7, 7, 32))
      self.conv2trd1 = keras.layers.Conv2DTranspose(filters=64,
                                                    kernel_size=3,
                                                    strides = (2,2),
                                                    padding="same",
                                                    activation="relu")
      self.dropout = keras.layers.Dropout(rate=0.5)
      self.flatten = keras.layers.Flatten()
      self.fc2 = keras.layers.Dense(784)
      self.resh = keras.layers.Reshape(target_shape=(28, 28))

  def call(self, x, training=False):
    x = self.decoder_input(x)
    x = self.fc1(x)
    x = self.reshape(x)
    x = self.conv2trd1(x)
    x = self.flatten(x)
    x = self.fc2(x)
    #x = self.conv2dtr2(x)
    if training:
      x = self.dropout(x)
    x = self.resh(x)
    return x
```

Класс вариационный автоэнкодер

```
class VAE(keras.Model):
    # Функции от keras.Model
        def __init__(self):
            super(VAE, self).__init__(name='mnist_vae')
            with tf.device('/device:GPU:0'):
                self.encoder = MnistEncoder()
                self.decoder = MnistDecoder()

        def call(self, x, training=False):
            mu, var = self.encoder(x,training)
            z = self.reparametrize(mu, var)
            x_hat = self.decoder(z,training)
            return x_hat, mu, var

    # Функция выполняющая репараметризацию
```

```
        def reparametrize(self, mu, var):
            eps = tf.random.normal(shape=mu.shape)
            return eps * var + mu

    # Функция выполняющая получение изображение из вектора z. Заметим, что

    # здесь присутствует сигмоида
        def sample(self, eps=None):
            if eps is None:
                eps = tf.random.normal(shape=(100, self.latent_dim))
            return tf.sigmoid(self.decoder(eps,False))
```

Обучение

```
from datetime import datetime
import numpy as np

optimizer = tf.keras.optimizers.Adam(1e-4)

summary_writer = tf.summary.create_file_writer('logs', flush_millis=10000)
summary_writer.set_as_default()
@tf.function
def log_normal_pdf(mean, var, raxis=1):
    arg1 = tf.multiply(tf.pow(mean,2) + tf.pow(var,2) - tf.math.log(tf.pow
(var,2)) - 1,0.5)
    return tf.reduce_sum(arg1,axis=1)

@tf.function
def compute_loss(model, x):
    x_hat, mu, var = model(x,False)
    ## сигмоида встроена в функцию потери
    cross_ent = tf.nn.sigmoid_cross_entropy_with_logits(logits=x_hat, labe
ls=x)
    logpx_z = -tf.reduce_sum(cross_ent, axis=[1, 2])
    logqz_x = log_normal_pdf(mu, var)
    return -tf.reduce_mean(logpx_z  - logqz_x)

@tf.function
def compute_gradients(model, x):
    with tf.device('/gpu:0'):
        with tf.GradientTape() as tape:
            loss = compute_loss(model, x)
        return tape.gradient(loss, model.trainable_variables), loss
@tf.function
def apply_gradients(optimizer, gradients, variables):
    optimizer.apply_gradients(zip(gradients, variables))

def generate_and_save_images(model, epoch, test_input, file_writer):
    '''
    Генерирует примеры на основе test_input и записывает их в file_writer
для
```

```
    визуализации в tensorboard
    '''
    predictions = model.sample(test_input)
    with file_writer.as_default():
        images = np.reshape(predictions, (-1, 28, 28, 1)) * 255
        images[images < 0] = 0
        images[images > 255] = 255
        tf.summary.image("16 generated data examples", images, max_outputs
=16, step=epoch)

epochs = 40
latent_dim = 2
num_examples_to_generate = 16

# Создадим случайный вектор
random_vector_for_generation = tf.random.normal(
    shape=[num_examples_to_generate, latent_dim])


model = VAE()


import time

current_time = datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/gradient_tape/' + current_time + '/train'
test_log_dir = 'logs/gradient_tape/' + current_time + '/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
test_summary_writer = tf.summary.create_file_writer(test_log_dir)


generate_and_save_images(model, 0, random_vector_for_generation, test_summ
ary_writer)
# Создаем метрики для вычисления.
train_loss = tf.keras.metrics.Mean('train_loss', dtype=tf.float32)
test_loss = tf.keras.metrics.Mean('test_loss', dtype=tf.float32)

%tensorboard --logdir logs/gradient_tape

for epoch in range(1, epochs + 1):
    for train_x in train_dataset:
        gradients, loss = compute_gradients(model, train_x)
        train_loss(loss)
        apply_gradients(optimizer, gradients, model.trainable_variables)
    with train_summary_writer.as_default():
        tf.summary.scalar('loss', train_loss.result(), step=epoch)
    print(epoch,loss)

    if epoch % 1 == 0:
#         for test_x in test_dataset:
#             test_loss(compute_loss(model, test_x))
```

```
#         with test_summary_writer.as_default():
#             tf.summary.scalar('loss', test_loss.result(), step=epoch)

        generate_and_save_images(model, epoch, random_vector_for_generatio
n, test_summary_writer)
    # Обнуляем метрики на каждой эпохе
    train_loss.reset_states()
    test_loss.reset_states()
```

```
30 tf.Tensor(153.52444, shape=(), dtype=float32)
31 tf.Tensor(156.3598, shape=(), dtype=float32)
32 tf.Tensor(157.7155, shape=(), dtype=float32)
33 tf.Tensor(147.23096, shape=(), dtype=float32)
34 tf.Tensor(145.8035, shape=(), dtype=float32)
35 tf.Tensor(180.13516, shape=(), dtype=float32)
36 tf.Tensor(171.5007, shape=(), dtype=float32)
37 tf.Tensor(169.98785, shape=(), dtype=float32)
38 tf.Tensor(155.88887, shape=(), dtype=float32)
39 tf.Tensor(150.18681, shape=(), dtype=float32)
40 tf.Tensor(128.99367, shape=(), dtype=float32)
```

Создание сетки изображений

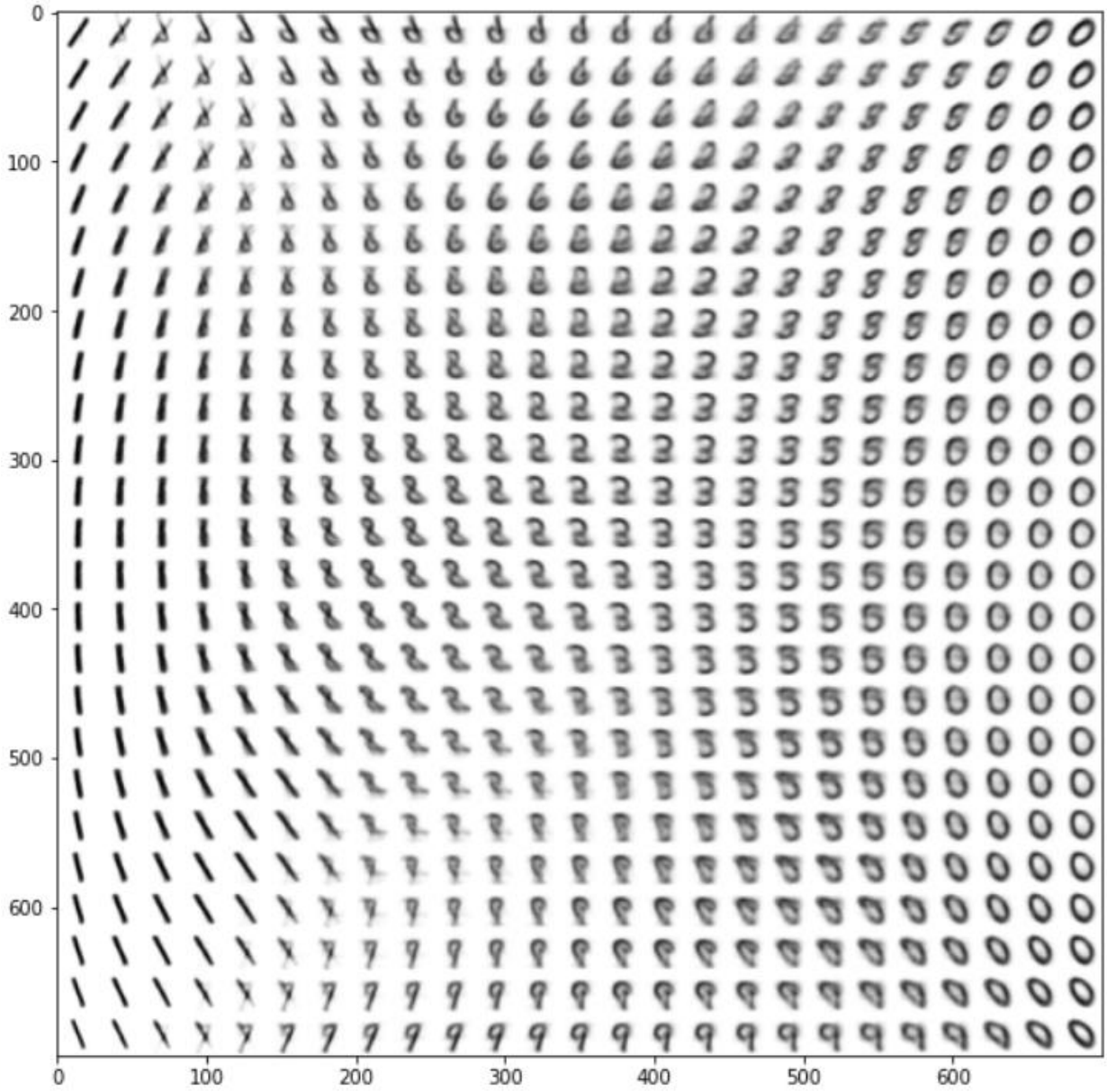
```
from scipy.stats import norm
import matplotlib.pyplot as plt
# Display a 2D manifold of the digits
n = 25  # figure with 25x25 digits
digit_size = 28
batch_size = 32
figure = np.zeros((digit_size * n, digit_size * n))

# Construct grid of latent variable values
grid_x = norm.ppf(np.linspace(0.05, 0.95, n))
grid_y = norm.ppf(np.linspace(0.05, 0.95, n))

# decode for each square in the grid
for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        z_sample = np.tile(z_sample, batch_size).reshape(batch_size, 2)
        x_decoded = model.sample(z_sample)
        digit = tf.reshape(x_decoded[0], (digit_size, digit_size))
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit
```

```
plt.figure(figsize=(10, 10))
plt.imshow(figure, cmap='binary')
plt.show()
```

Результат



Список литературы

1. Терехов В.И., Черненький И.М., Методические указания к лабораторной работе №5 - М, 2020
2. TensorFlow [Электронный ресурс] - Режим доступа - https://www.tensorflow.org/