МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов”

Студентка гр. 7381 Кревчик А.Б.

Преподаватель Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

Постановка задачи.

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Написать функцию, позволяющая загружать изображение пользователи и классифицировать его

Ход работы.

Набор данных MNIST уже входит в состав Keras в форме набора из четырех массивов Numpy.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()
```

Здесь `train_images` и `train_labels` — это тренировочный набор, то есть данные, необходимые для обучения. После обучения модель будет проверяться тестовым (или контрольным) набором, `test_images` и `test_labels`. Изображения хранятся в массивах Numpy, а метки — в массиве цифр от 0 до 9. Изображения и метки находятся в прямом соответствии, один к одному.

Для проверки корректности загрузки достаточно сравнить тестовое изображение с его меткой.

```
import matplotlib.pyplot as plt
plt.imshow(train_images[0],cmap=plt.cm.binary)
plt.show()
print(train_labels[0])
```

Исходные изображения представлены в виде массивов чисел в интервале [0, 255]. Перед обучением их необходимо преобразовать так, чтобы все значения оказались в интервале [0, 1].

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Также необходимо закодировать метки категорий. В данном случае прямое кодирование меток заключается в конструировании вектора с нулевыми элементами со значением 1 в элементе, индекс которого соответствует индексу метки.

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Теперь можно задать базовую архитектуру сети.

```
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential
model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Чтобы подготовить сеть к обучению, нужно настроить еще три параметра для этапа компиляции:

- функцию потерь, которая определяет, как сеть должна оценивать качество своей работы на обучающих данных и, соответственно, как корректировать ее в правильном направлении;
- оптимизатор — механизм, с помощью которого сеть будет обновлять себя, опираясь на наблюдаемые данные и функцию потерь;
- метрики для мониторинга на этапах обучения и тестирования — здесь нас будет интересовать только точность (доля правильно классифицированных изображений).

```
model.compile(optimizer='adam',loss='categorical_crossentropy',
metrics=['accuracy'])
```

Теперь можно начинать обучение сети, для чего в случае использования библиотеки Keras достаточно вызвать метод fit сети — он пытается адаптировать (fit) модель под обучающие данные.

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

В процессе обучения отображаются две величины: потери сети на обучающих данных и точность сети на обучающих данных.

Теперь проверим, как модель распознает контрольный набор:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

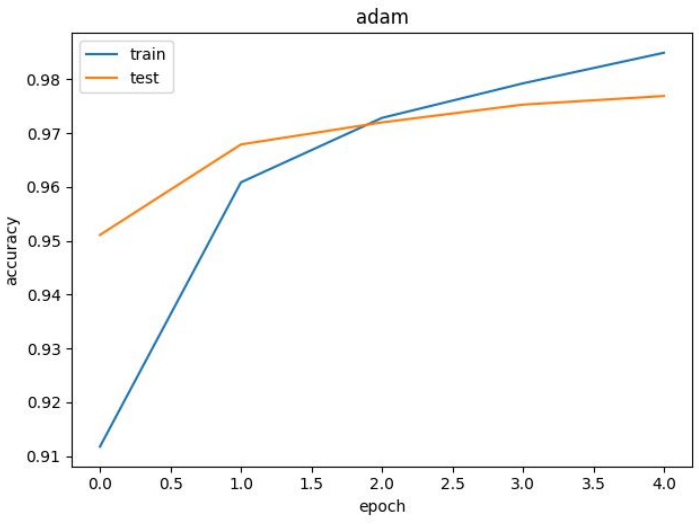
Требования

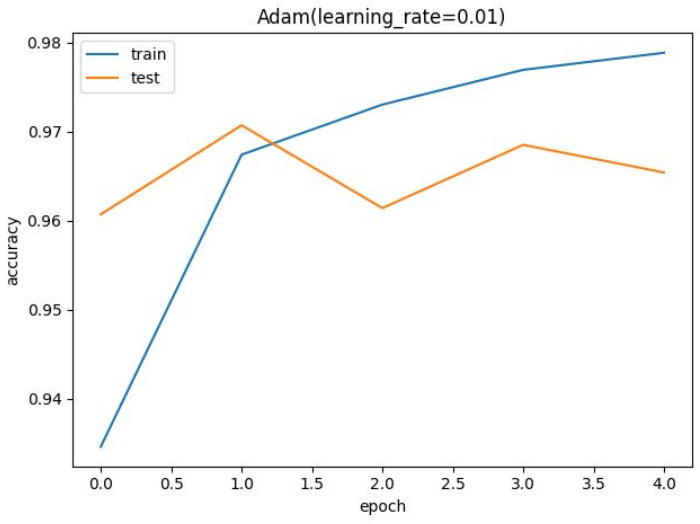
1. Найти архитектуру сети, при которой точность классификации будет не менее 95%

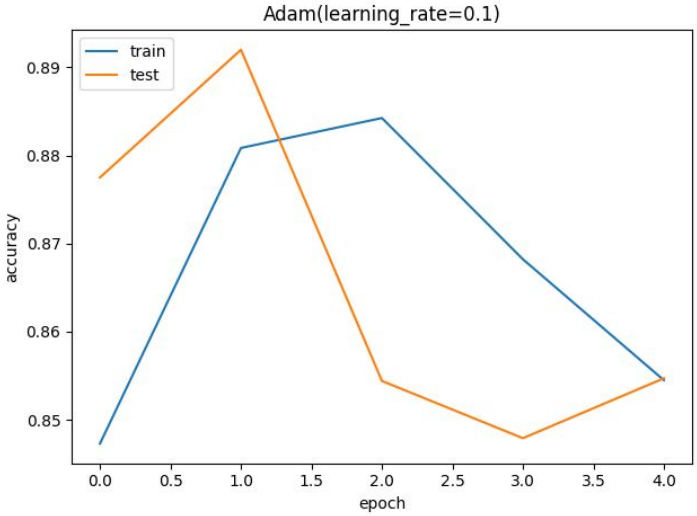
   Архитектура, данная в лабораторной работе, позволяет классифицировать изображения с точностью 0.9793.

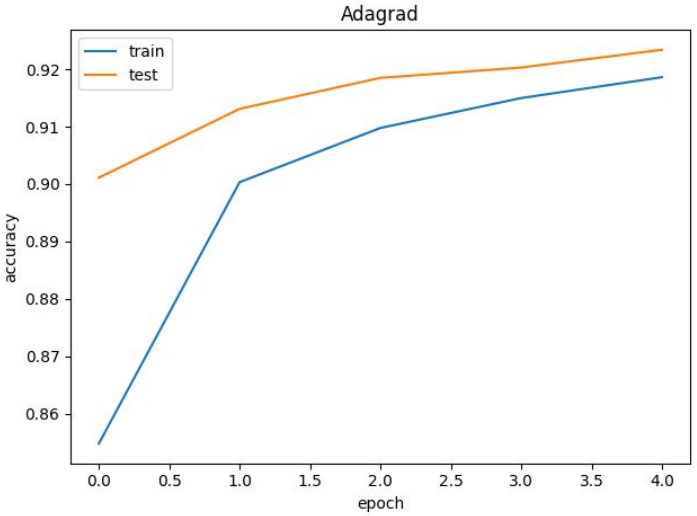
2. Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения

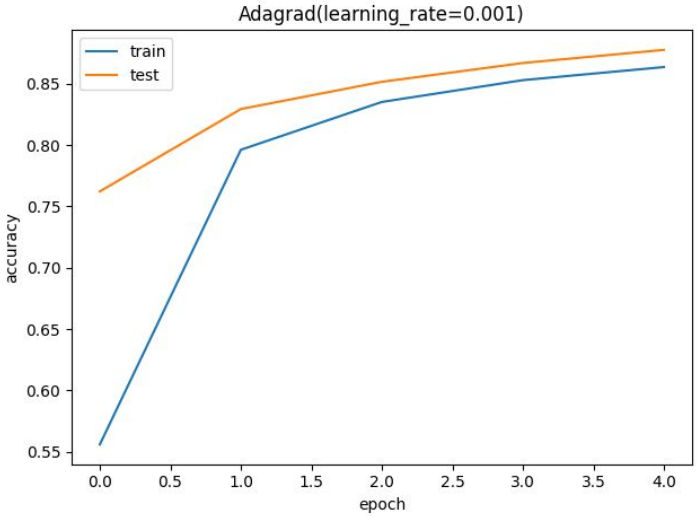
Ниже представлены графики точности при использовании различных оптимизаторов.

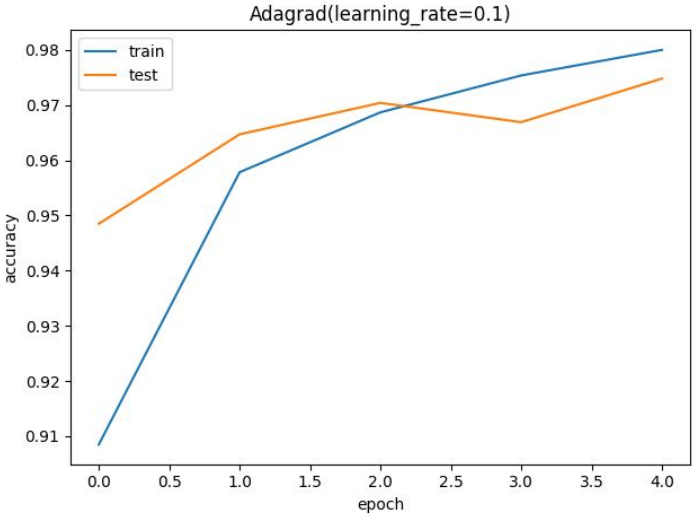


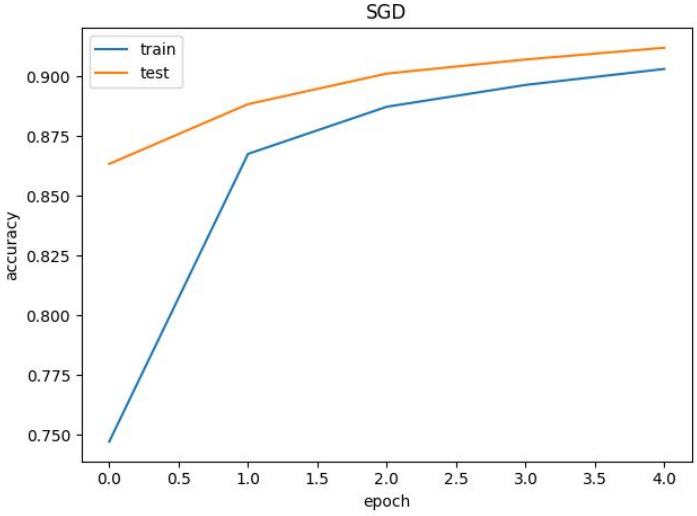


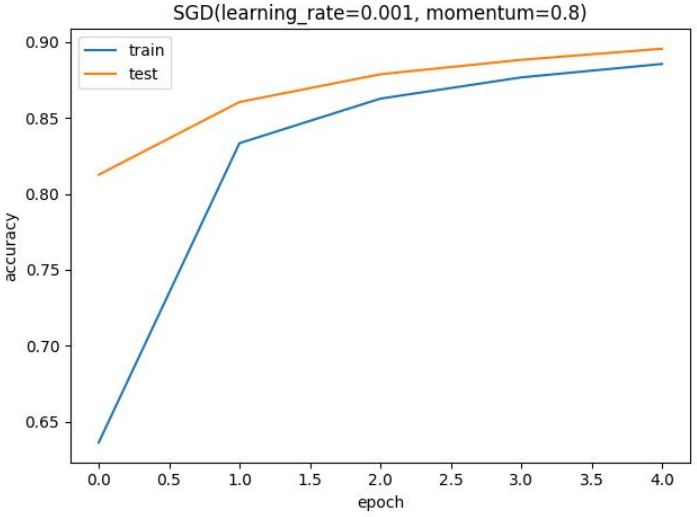

Adam(learning_rate=0.1)
train
test
accuracy
epoch

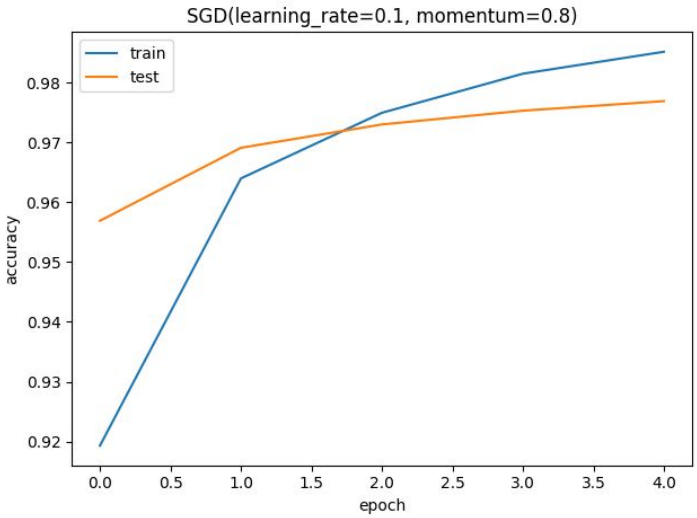

Adagrad
train
test
accuracy
epoch

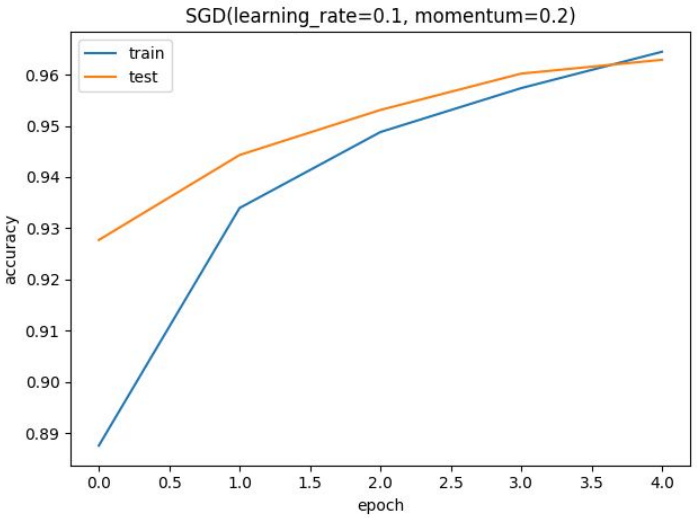

Adagrad(learning_rate=0.001)
train
test
accuracy
epoch


Adagrad(learning_rate=0.1)
train
test
accuracy
epoch


SGD
train
test
accuracy
epoch


SGD(learning_rate=0.001, momentum=0.8)
train
test
accuracy
epoch





Можно сделать вывод, что наибольшая точность получается при использовании оптимизаторов `SGD(learning_rate=0.1, momentum=0.2)` (точность 0.9629), `Adagrad(learning_rate=0.1)` (точность 0.9698) и `Adam` (точность 0.9764).

3. Написать функцию, которая позволит загружать пользовательское изображение не из датасета

   Была написана функция `test_image`:

```
def test_image(name):
    pil_im = Image.open(name)
    pil_im.thumbnail((28,28))
    image = asarray(pil_im.convert('L'))
    image= image.reshape((1, 28, 28))
    return image
```

Выводы.

Таким образом, была изучена задача классификации черно-белых изображений рукописных цифр. Также была написана функция для загрузки изображений не из представленного датасета. Изучено влияние различных оптимизаторов на точность классификации.