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

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 7383 _____ Лосев М.Л.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель.

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

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

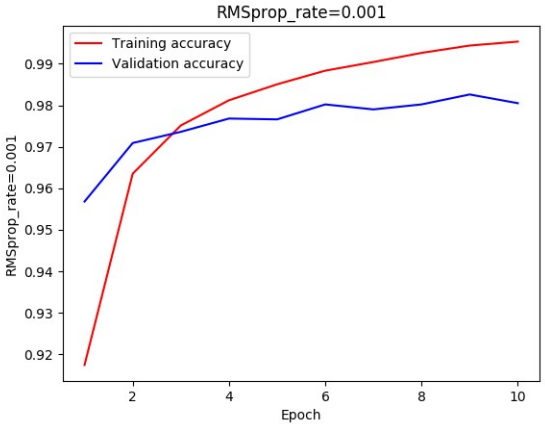
Выполнение работы.

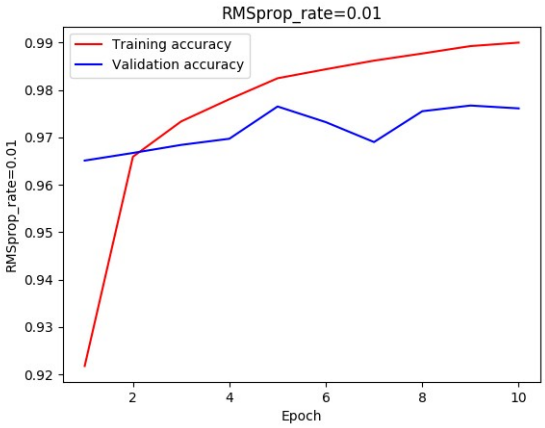
1. Архитектура с одним внутренним слоем из 256 нейронов позволяет достичь точности 0.97-0.98 (если использовать оптимизатор adam):

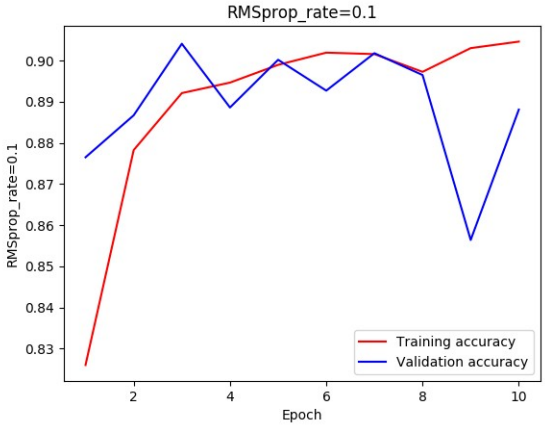
```
model=Sequential()
model.add(Flatten())
model.add(Dense(256,activation='relu'))
model.add(Dense(10, activation='softmax'))
```

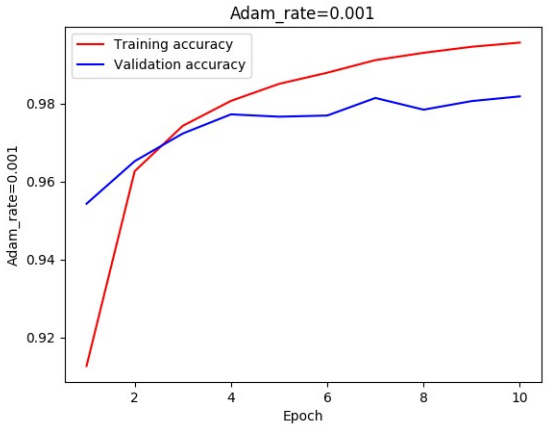
2. Чтобы исследовать влияние различных оптимизаторов и их параметров на обучение, попробуем SGD, RMSprop, Adam и Adadelta. В качестве метрики возьмем точность, сохраним ее графики.

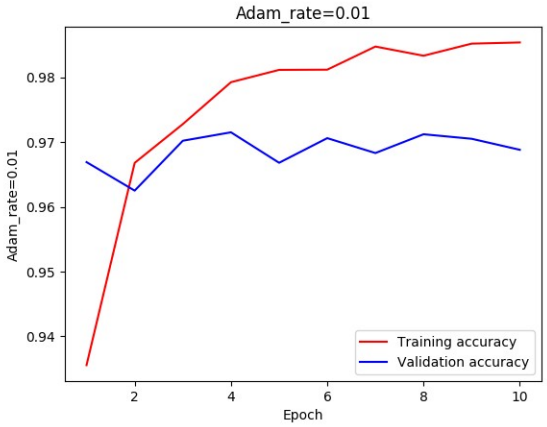
Для RMSprop в документации keras не рекомендуется менять никакие параметры, кроме learning_rate. Опыт показал, что значение этого параметра должно быть маленьким. Например, из 0.1, 0.01 и 0.001 лучшая точность достигается при learning_rate = 0.001. То же самое справедливо Adam. Следующие шесть графиков показвыают это.

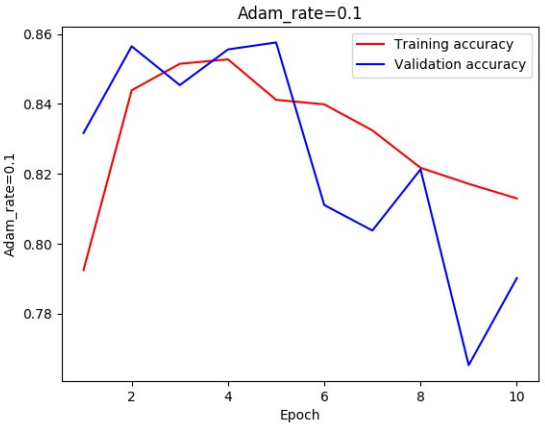




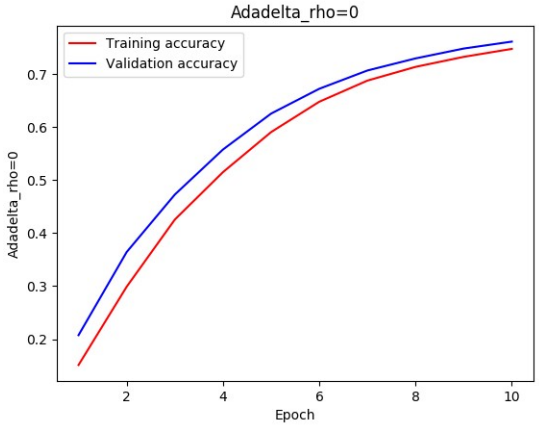


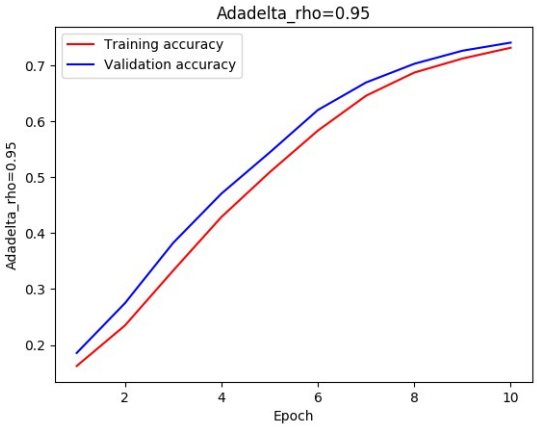


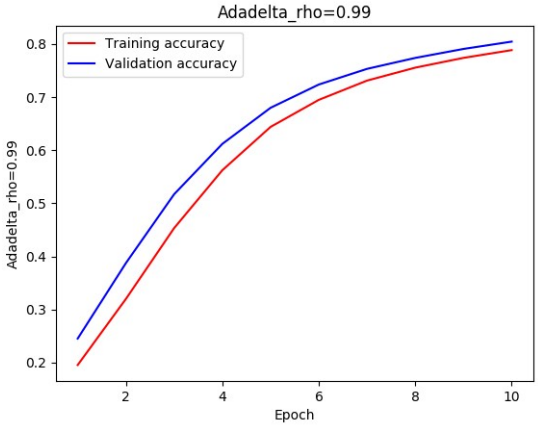




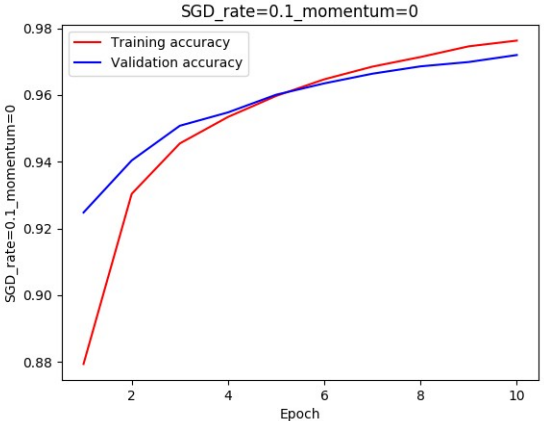
Для Adadelta рекомендуется изменять только параметр rho. Оптимальное его значение оказалось 0.99, хотя точность была немного ниже 0.8, что меньше, чем позволяет добиться Adam. Ниже представлены графики.

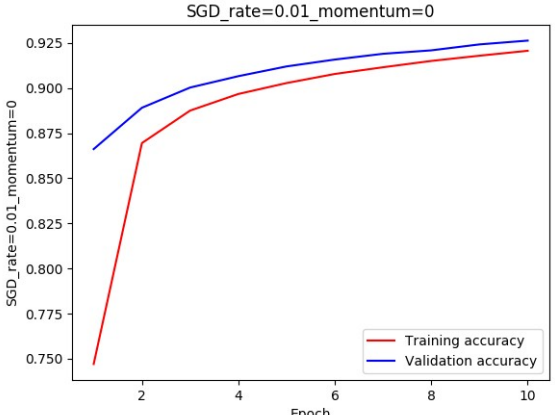


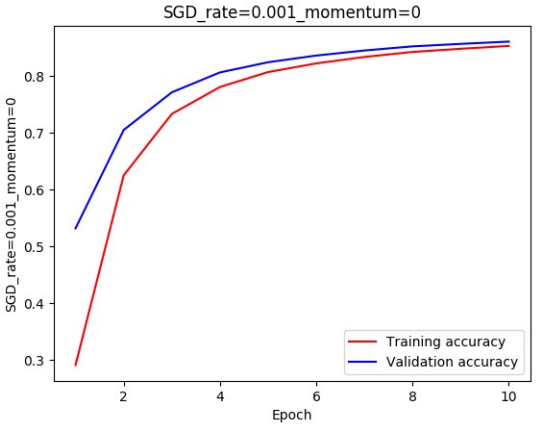


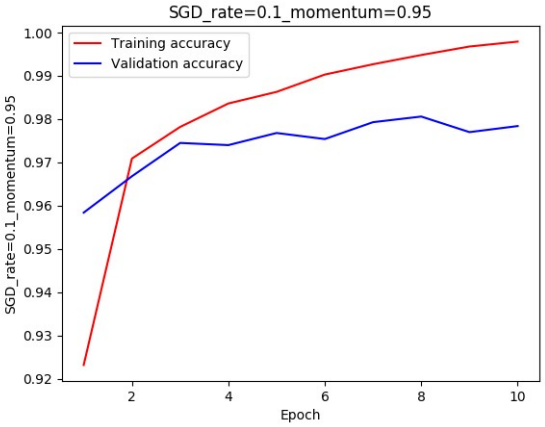


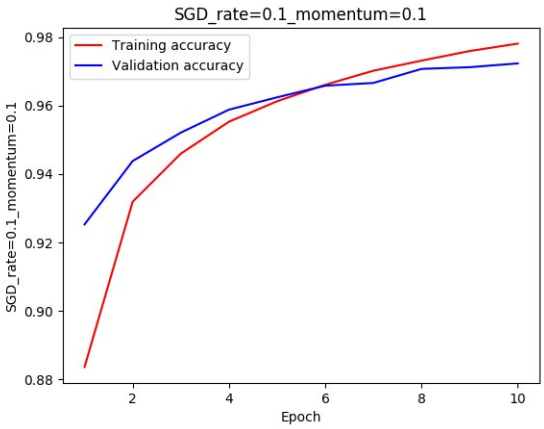
Для SGD исследовались параметры learning_rate и momentum. Оказалось, что увеличение learning_rate увеличивает точность. Значение learning_rate=0.1 позволяет добиться точности 0.97 — 0.98. Момент немного уменьшает скорость обучения, но при его значении, близком к 1, обучение невозможно, так как точность падает с каждой эпохой. Таким образом для этой задачи момент может быть строго меньше 1, и лучше ближе к 0. Ниже приведено несколько графиков, показывающих это.

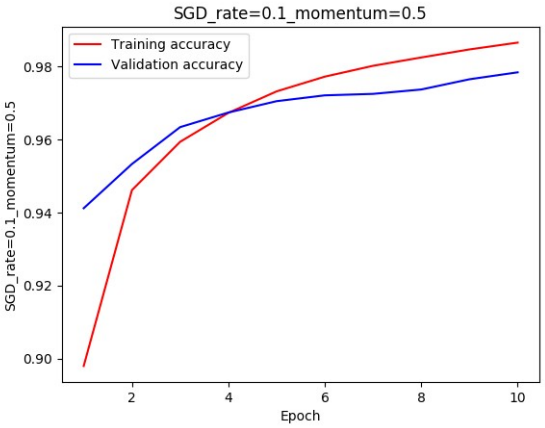

SGD_rate=0.1_momentum=0
Training accuracy
Validation accuracy
SGD_rate=0.1_momentum=0
Epoch

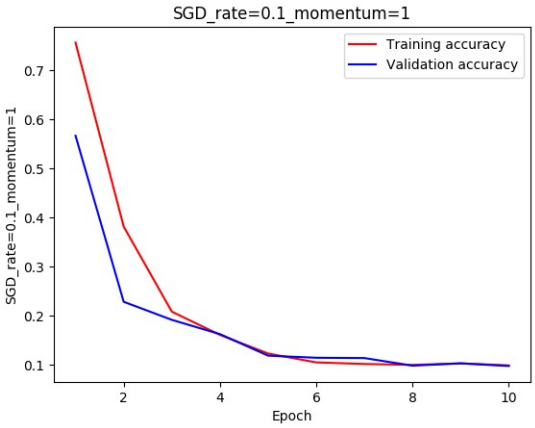

SGD_rate=0.01_momentum=0
Training accuracy
Validation accuracy
SGD_rate=0.01_momentum=0
Epoch


SGD_rate=0.001_momentum=0
Training accuracy
Validation accuracy
SGD_rate=0.001_momentum=0
Epoch


SGD_rate=0.1_momentum=0.95
Training accuracy
Validation accuracy
SGD_rate=0.1_momentum=0.95
Epoch


SGD_rate=0.1_momentum=0.1
Training accuracy
Validation accuracy
SGD_rate=0.1_momentum=0.1
Epoch


SGD_rate=0.1_momentum=0.5
Training accuracy
Validation accuracy
SGD_rate=0.1_momentum=0.5
Epoch


SGD_rate=0.1_momentum=1
Training accuracy
Validation accuracy
SGD_rate=0.1_momentum=1
Epoch

3. Попробуем загрузить пользовательское изображение. Приведем его к размеру 28*28 и усредним цвета (RGB), чтобы получить черно-белое изображение. Чтобы было интереснее, попробуем распознать изображение цифры 2 (Код в приложении А). Нейросеть не распознает изображение: чаще она идентифицирует его как пятерку.

Вывод.

В ходе выполнения данной работы была создана простейшая сеть, распознающая рукописные символы. Было исследовано влияние оптимизаторов и их параметров на обучение. Лучше всего подошел Adam со скоростью обучения 0.001. Было загружено пользовательское изображение, но результат его распознавания окался неверен. Это можно объяснить тем, что загруженное изображение отличается стилистикой от изображений обучающего датасета.

Приложение А

```
import tensorflow as tf
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras import optimizers
from PIL import Image
import numpy as np

def save_plot(label, history):
    picdir = './pics/'
    plt.clf()
    arg = range(1, len(history['loss']) + 1)
    # only save accuracy cuz its the metrics
    plt.clf()
    plt.plot(arg, history['acc'], 'r', label='Training accuracy')
    plt.plot(arg, history['val_acc'], 'b', label='Validation accuracy')
    plt.title(label)
    plt.xlabel('Epoch')
    plt.ylabel(label)
    plt.legend()
    plt.savefig(picdir + label + '_accuracy' + '.png')

def openUserImage(path): # returns black&white image of size 28x28
    image = Image.open(path)
    image = image.resize((28,28), Image.ANTIALIAS)
    data = np.asarray(image)
    data = np.reshape(data, (data.size // data.shape[-1], data.shape[-1]))
    retval = []
    for i, p in enumerate(data):
        # find the mean of (R, G, B) in order to make the image black and white
        r = 0
        for j, c in enumerate(p):
            if j < 3: # RGB
                r += c
            if j == 4: # Alpha
                r *= c
            if j > 4: # unexpected case
                break
        r //= 3
        retval.append(r)
    return np.reshape(retval, (28,28))

def build_model():
    model = Sequential()
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    return model

def explore_optimizer(optimizer, label):
    model = build_model()
    model.compile(optimizer=optimizer, learning_rate=0.01, momentum=1000,
loss='categorical_crossentropy', metrics=['accuracy'])
    H = model.fit(train_images, train_labels,
                        epochs=10,
                        batch_size=128,
                        verbose=1,
                        validation_split = 0.1, validation_data = (test_images,
test_labels))
    val_acc = H.history['val_acc'][-1]
    print('test_acc:', val_acc)
```

```
    save_plot(label, H.history)

mnist = tf.keras.datasets.mnist
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

rate = [0.1, 0.01, 0.001]
momentum = [0, 0.1, 0.5, 0.95]
rho = [0, 0.9, 0.95, 0.99, 10]
for r in rate:
    for m in momentum:
        explore_optimizer(optimizers.SGD(learning_rate=r, momentum=m),
'SGD_rate='+str(r)+'_momentum='+str(m))
    explore_optimizer(optimizers.Adam(learning_rate=r), 'Adam_rate='+str(r))
    # It is recommended to leave the parameters of this optimizer at their
default values
    # (except the learning rate, which can be freely tuned).
    explore_optimizer(optimizers.RMSprop(learning_rate=r),
'RMSprop_rate='+str(r))
for r in rho:
    # Initial learning rate is recommended to leave at the default value.
    explore_optimizer(optimizers.Adadelta(rho=r), 'Adadelta_rho='+str(r))

# load user image and try to recognize it
data = openUserImage('./2.jpg')
print(data.shape)

model = build_model()
model.compile(optimizer='adam', learning_rate=0.01, momentum=1000,
loss='categorical_crossentropy', metrics=['accuracy'])
H = model.fit(train_images, train_labels,
                    epochs=10,
                    batch_size=128,
                    verbose=1,
                    validation_split = 0.1, validation_data = (test_images,
test_labels))

categorical_prediction = model.predict(np.reshape(data, (1, 28, 28)))[0]
digits = np.asarray([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], int)
print(int(np.dot(categorical_prediction, digits)))
```