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. 8382 _____ Кулачкова М.К.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы

Реализовать классификацию черно-белых изображений рукописных цифр (28×28) по 10 категориям (от 0 до 9). Набор данных содержит 60000 изображений для обучения и 10000 изображений для тестирования.

Задачи

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

Требования

1. Найти архитектуру сети, при которой точность классификации будет не менее 95%
2. Исследовать влияние различных оптимизаторов, а также их параметров на процесс обучения
3. Написать функцию, которая позволит загружать пользовательское изображение (не из датасета).

Выполнение работы

Загрузим датасет в программу. Каждое изображение представлено в виде матрицы интенсивностей, т.е. двумерного тензора размера 28×28, состоящего из чисел от 0 до 255. Для упрощения обучения масштабируем данные, чтобы все значения оказались в интервале [0, 1]. Метки категорий представим в виде векторов длины 10, в которых элемент, индекс которого соответствует метке, равен 1, а остальные – 0.

Зададим базовую архитектуру сети. В модель будет входить слой, преобразующий форму входного тензора, а именно переводящий матрицу 28×28, соответствующую изображению, в вектор длины 784. Также в модель входит скрытый полносвязный слой из 256 нейронов с функцией активации ReLU и выходной слой из 10 нейронов с функцией активации softmax.

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

Будем исследовать, как выбор оптимизатора и его параметров влияет на обучение модель. Для начала используем предложенный в методических указаниях оптимизатор Adam со значениями параметров по умолчанию. Правило обновления в этом методе оптимизации имеет вид

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t,$$

$$\text{где } \hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t};$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t;$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2.$$

Здесь θ_t – параметры сети в момент времени t, η – скорость обучения, ϵ – сглаживающий параметр, необходимый, чтобы избежать деления на 0, g_t – градиент функции потерь. Значения параметров по умолчанию равны $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 0.001$.

Обучим модель в течение 5 эпох пакетами по 128 изображений:

```
Epoch 1/5
469/469 [==============================] - 3s 4ms/step - loss: 0.52
97 - accuracy: 0.8527
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.13
95 - accuracy: 0.9608
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.09
26 - accuracy: 0.9733
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.06
80 - accuracy: 0.9801
Epoch 5/5
```

```
469/469 [==============================] - 2s 4ms/step - loss: 0.04
98 - accuracy: 0.9859
```

После этого оценим точность модели на тестовом множестве:

```
313/313 [==============================] - 1s 2ms/step - loss: 0.07
04 - accuracy: 0.9773
test_acc: 0.9772999882698059
```

Получили точность 97.73 %, что соответствует требованиям.

Проверим, как значение параметра β_1 влияет на процесс обучения. Выберем $\beta_1 = 0.5$, остальные параметры оставим неизменными.

```
Epoch 1/5
469/469 [==============================] - 3s 4ms/step - loss: 0.53
32 - accuracy: 0.8564
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.14
33 - accuracy: 0.9598
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.09
63 - accuracy: 0.9717
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.06
85 - accuracy: 0.9806
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.05
21 - accuracy: 0.9852
313/313 [==============================] - 1s 2ms/step - loss: 0.08
05 - accuracy: 0.9759
test_acc: 0.9758999943733215
```

Можно заметить, что точность модели в процессе обучения практически не изменилась. Итоговая точность модели оказалась несколько хуже, но это отклонение может быть случайным.

Теперь исследуем влияние на обучение параметра β_2. Выберем $\beta_2 = 0.5$. Значения остальных параметров оставим по умолчанию.

```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.54
68 - accuracy: 0.8434
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.11
73 - accuracy: 0.9653
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.08
29 - accuracy: 0.9753
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.06
51 - accuracy: 0.9809
Epoch 5/5
```

```
469/469 [==============================] - 2s 4ms/step - loss: 0.09
00 - accuracy: 0.9800
313/313 [==============================] - 1s 2ms/step - loss: 0.12
14 - accuracy: 0.9691
test_acc: 0.9690999984741211
```

По сравнению с предыдущими выборами параметров точность модели в процессе обучения ухудшилась, как и итоговая точность модели.

Теперь изменим параметр η, который показывает, насколько сильно будут изменяться параметры на каждом шаге метода. Выберем $\eta = 0.01$ при $\beta_1 = 0.9$ и $\beta_2 = 0.999$.

```
Epoch 1/5
469/469 [==============================] - 3s 4ms/step - loss: 0.34
87 - accuracy: 0.8888
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 0.10
83 - accuracy: 0.9676
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.09
11 - accuracy: 0.9714
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.06
45 - accuracy: 0.9798
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.06
25 - accuracy: 0.9805
313/313 [==============================] - 1s 2ms/step - loss: 0.14
13 - accuracy: 0.9688
test_acc: 0.9688000082969666
```

Точность в процессе обучения стала изменяться медленнее, и итоговая точность модели оказалась хуже, чем во всех рассмотренных ранее случаях.

Теперь рассмотрим метод оптимизации RMSProp. Формула обновления имеет вид

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t,$$

где $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$, $0 < \gamma < 1$.

Будем варьировать значения γ и η. По умолчанию $\gamma = 0.9$ и $\eta = 0.001$.

```
Epoch 1/5
469/469 [==============================] - 3s 6ms/step - loss: 0.49
18 - accuracy: 0.8637
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.13
87 - accuracy: 0.9593
Epoch 3/5
```

```
469/469 [==============================] - 2s 5ms/step - loss: 0.09
31 - accuracy: 0.9730
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.06
42 - accuracy: 0.9809
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.04
82 - accuracy: 0.9858
313/313 [==============================] - 1s 2ms/step - loss: 0.07
16 - accuracy: 0.9768
test_acc: 0.9768000245094299
```

Результат очень похож на результат работы оптимизатора Adam с параметрами по умолчанию.

Выберем $\gamma = 0.5$ при $\eta = 0.001$.

```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.53
53 - accuracy: 0.8526
Epoch 2/5
469/469 [==============================] - 3s 6ms/step - loss: 0.15
85 - accuracy: 0.9554
Epoch 3/5
469/469 [==============================] - 3s 6ms/step - loss: 0.11
40 - accuracy: 0.9664
Epoch 4/5
469/469 [==============================] - 3s 5ms/step - loss: 0.08
38 - accuracy: 0.9754
Epoch 5/5
469/469 [==============================] - 3s 6ms/step - loss: 0.06
85 - accuracy: 0.9806
313/313 [==============================] - 1s 2ms/step - loss: 0.08
59 - accuracy: 0.9759
test_acc: 0.9758999943733215
```

Точность модели в процессе обучения ухудшилась, итоговая точность модели также ухудшилась, но незначительно.

Выберем $\eta = 0.01$ при $\gamma = 0.9$.

```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.55
56 - accuracy: 0.8570
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.12
81 - accuracy: 0.9650
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.10
58 - accuracy: 0.9733
Epoch 4/5
469/469 [==============================] - 2s 5ms/step - loss: 0.09
16 - accuracy: 0.9782
Epoch 5/5
469/469 [==============================] - 2s 5ms/step - loss: 0.08
04 - accuracy: 0.9823
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.18
96 - accuracy: 0.9660
test_acc: 0.9660000205039978
```

По сравнению с параметрами по умолчанию точность модели в процессе обучения практически не изменилась, но итоговая точность модели ухудшилась.

Исследуем оптимизатор Adagrad. Правило обновления для этого метода имеет вид

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t,$$

где $G_t = G_{t-1} + g_t^2$ – сумма квадратов обновлений градиента.

Будем исследовать влияние на обучение параметров η и ϵ. По умолчанию $\eta = 0.001$ и $\epsilon = 10^{-7}$.

```
Epoch 1/5
469/469 [==============================] - 5s 4ms/step - loss: 2.07
93 - accuracy: 0.3198
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 1.24
00 - accuracy: 0.7772
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.89
05 - accuracy: 0.8207
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.71
72 - accuracy: 0.8439
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.62
36 - accuracy: 0.8582
313/313 [==============================] - 1s 2ms/step - loss: 0.55
07 - accuracy: 0.8762
test_acc: 0.8762000203132629
```

По сравнению с методами, рассмотренными ранее, Adagrad с параметрами по умолчанию дает довольно низкую точность.

Выберем $\epsilon = 0.001$.

```
Epoch 1/5
469/469 [==============================] - 4s 5ms/step - loss: 2.00
91 - accuracy: 0.3897
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 1.19
23 - accuracy: 0.7681
Epoch 3/5
469/469 [==============================] - 2s 5ms/step - loss: 0.86
40 - accuracy: 0.8237
Epoch 4/5
```

```
469/469 [==============================] - 2s 5ms/step - loss: 0.70
71 - accuracy: 0.8445
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 0.61
86 - accuracy: 0.8570
313/313 [==============================] - 1s 2ms/step - loss: 0.54
89 - accuracy: 0.8754
test_acc: 0.8754000067710876
```

Точность модели практически не изменилась.

Теперь выберем $\epsilon = 1$.

```
Epoch 1/5
469/469 [==============================] - 2s 4ms/step - loss: 2.28
24 - accuracy: 0.1000
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 1.94
01 - accuracy: 0.4624
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 1.69
23 - accuracy: 0.6382
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 1.48
76 - accuracy: 0.7099
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 1.30
75 - accuracy: 0.7546
313/313 [==============================] - 1s 2ms/step - loss: 1.18
04 - accuracy: 0.7823
test_acc: 0.7822999954223633
```

Точность модели значительно ухудшилась, так как сглаживающий параметр стал оказывать более сильное влияние на скорость изменения параметров сети. Это противоречит смыслу параметра, поэтому лучше оставлять значения параметра маленькими.

Теперь при $\epsilon = 10^{-7}$ будем изменять значение скорости обучения. Выберем $\eta = 0.1$.

```
Epoch 1/5
469/469 [==============================] - 3s 5ms/step - loss: 0.51
20 - accuracy: 0.8518
Epoch 2/5
469/469 [==============================] - 2s 5ms/step - loss: 0.16
30 - accuracy: 0.9549
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 0.11
64 - accuracy: 0.9671
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 0.08
73 - accuracy: 0.9758
Epoch 5/5
```

```
469/469 [==============================] - 2s 4ms/step - loss: 0.06
87 - accuracy: 0.9805
313/313 [==============================] - 1s 2ms/step - loss: 0.07
88 - accuracy: 0.9760
test_acc: 0.9760000109672546
```

Точность модели значительно улучшилась по сравнению с параметрами по умолчанию. При таких параметрах для данной задачи метод сходится так же хорошо, как Adam и RMSProp.

Увеличим скорость обучения до значения $\eta = 1$.

```
Epoch 1/5
469/469 [==============================] - 3s 4ms/step - loss: 3.61
49 - accuracy: 0.1927
Epoch 2/5
469/469 [==============================] - 2s 4ms/step - loss: 1.91
64 - accuracy: 0.2565
Epoch 3/5
469/469 [==============================] - 2s 4ms/step - loss: 1.49
83 - accuracy: 0.4278
Epoch 4/5
469/469 [==============================] - 2s 4ms/step - loss: 1.30
69 - accuracy: 0.5016
Epoch 5/5
469/469 [==============================] - 2s 4ms/step - loss: 1.23
28 - accuracy: 0.5354
313/313 [==============================] - 1s 2ms/step - loss: 1.20
36 - accuracy: 0.5375
test_acc: 0.5375000238418579
```

Точность модели ухудшилась. При большой скорости обучения градиентные методы могут слишком сильно менять значения параметров на каждом шаге и «перепрыгивать» минимум.

Итак, было выяснено, что в данной задаче можно использовать любой из исследованных методов, если правильно подобрать значения параметров, но далее будет использоваться Adam с параметрами по умолчанию.

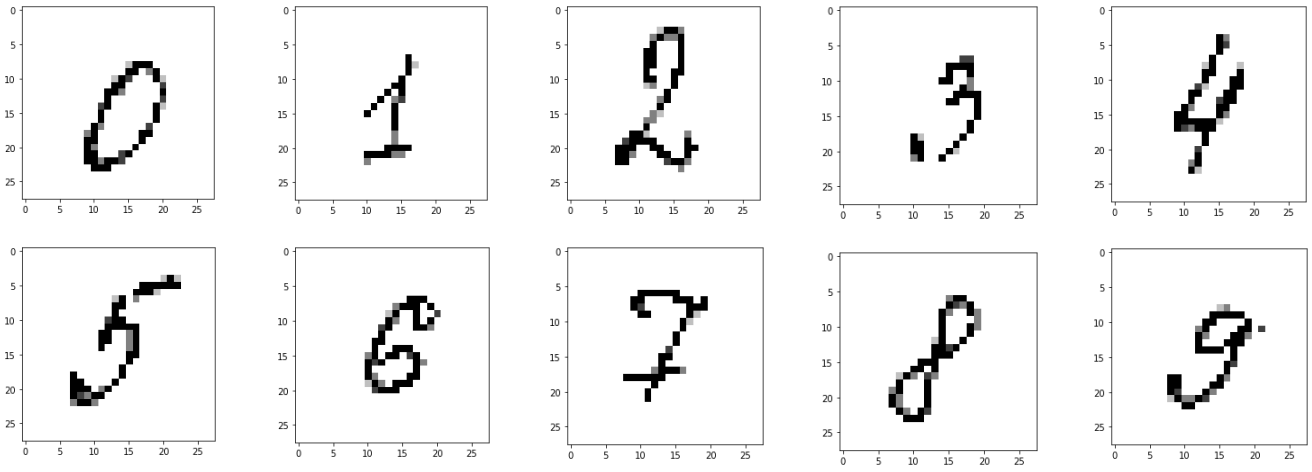
Реализуем функцию, позволяющую загружать пользовательское изображение и классифицировать его с помощью построенной модели.

```
def load_and_predict(path, model):
    img = load_img(path, color_mode="grayscale", target_size=(28, 28))
    input_arr = img_to_array(img)
    input_arr -= 255
    input_arr = input_arr / -255.0
    plt.imshow(input_arr, cmap=plt.cm.binary)
    plt.show()
    input_arr = np.asarray([input_arr])
    pr = model.predict(input_arr)
    cl = np.argmax(pr, 1)[0]
```

```
    return cl
```

Для загрузки изображения и преобразования его в массив используются функции load_img и img_to_array, импортированные из библиотеки Keras. При загрузке изображения его размер изменяется на 28×28, чтобы от соответствовал входному слою модели. Изображение загружается в черно-белом формате, чтобы оно имело только один канал. Было замечено, что изображение загружается инвертированным, поэтому после его загрузки и преобразования в массив оно инвертируется еще раз, а затем масштабируется, чтобы все значения принадлежали интервалу [0, 1]. Вывод изображения на экран необязателен и добавлен для наглядности. Так как для предсказания в модель должен передаваться массив изображений, загруженное изображение представляется в виде тензора, имеющего форму (1, 28, 28, 1). Затем осуществляется классификация изображения. Получаем массив из одного элемента, которым является вектор вероятностей принадлежности изображения к каждому из 10 классов. Для получения окончательного результата определяем индекс наибольшего элемента этого вектора и возвращаем полученное значение.

Для тестирования функции, а заодно и модели, был подготовлен ряд изображений с цифрами. Изображения приведены ниже.



В результате их классификации были получены значения 0, 1, 2, 7, 4, 5, 0, 7, 8, 0. Как видно, три из десяти изображений были классифицированы неправильно, но, так как на тестовых данных из датасета точность модели

оказалась высокой, я буду трактовать этот результат как одновременно комплимент и оскорбление в сторону своего почерка.

Выводы

В ходе выполнения работы была построена полносвязная нейронная сеть, осуществляющая классификацию изображений рукописных цифр с точностью более 97%, а также реализована функция, позволяющая загружать пользовательские изображения и классифицировать их с помощью полученной модели. Также было исследовано влияние различных методов оптимизации и их параметров на процесс обучения модели. Были рассмотрены оптимизаторы Adam, RMSProp и Adagrad. Обнаружено, что Adam и RMSProp в данной задаче дают практически одинаковые результаты, как и Adagrad при хорошо подобранной скорости обучения.