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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Искусственные нейронные сети»
Тема: «Распознавание рукописных символов»

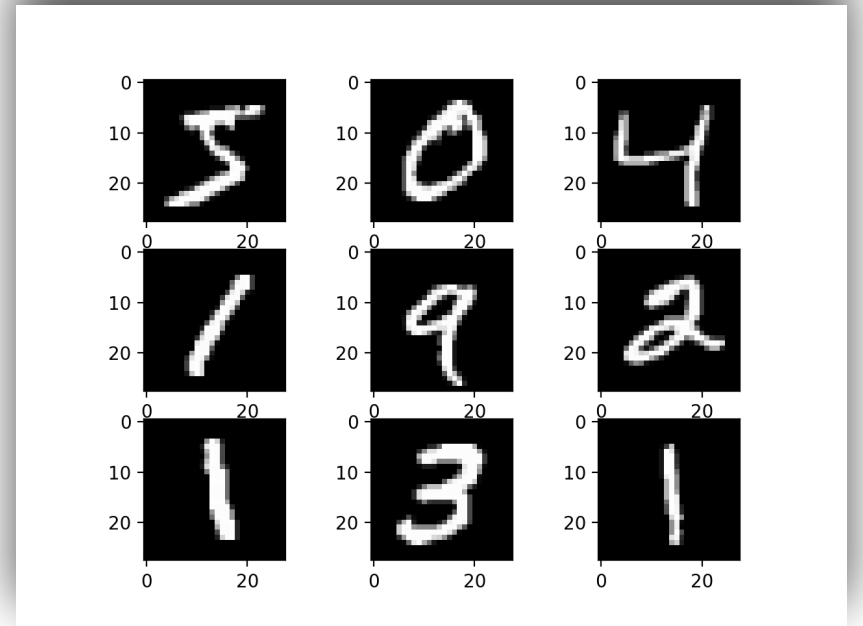
Студентка гр. 7381 _____ Судакова П.С.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург
2020

Цель работы.

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



Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

Порядок выполнения работы.

- 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

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.

Ход работы.

1. Выберем модель сети с 256 нейронами на входном слое, оптимизатором adam. Данная модель обучается на 5 эпохах. Точность и ошибок во время обучения показаны на рис. 1 – 2 соответственно.

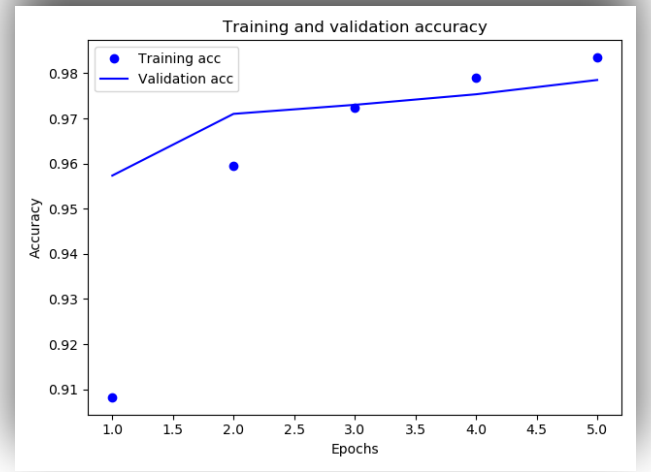


Рисунок 1 – График точности во время обучения

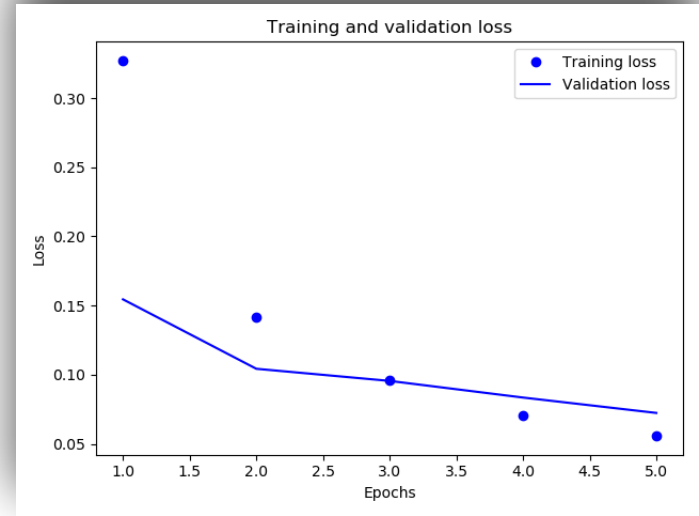


Рисунок 2 – График ошибок по время обучения

Точность данной модели на тестовых данных равна 97.55%.

2. Изучим влияние RMSprop, Adam, Adagard, Adadelta, SGD оптимизаторов на обучение модели с различными значениями скорости обучения. Результат представлены на рис. 3 - 7.

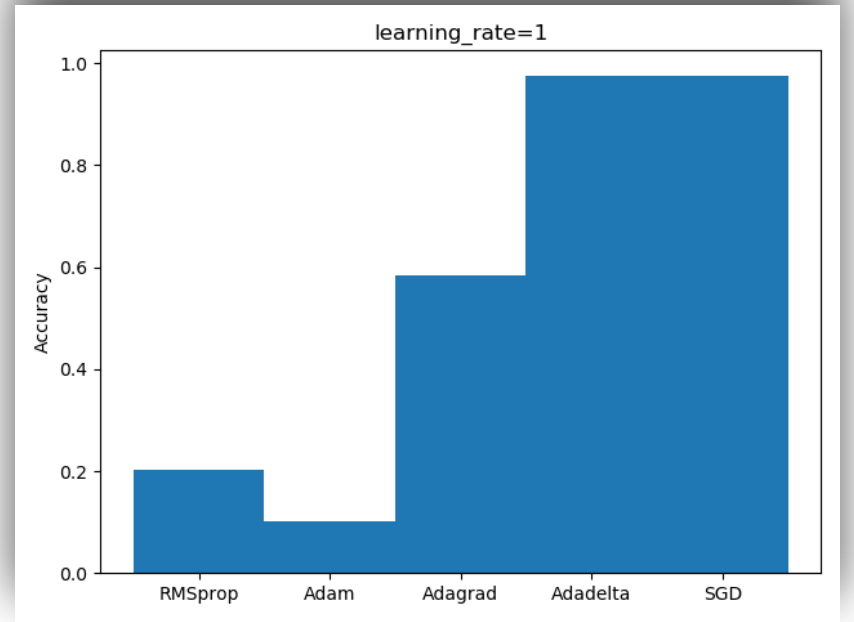


Рисунок 3 – График точности со скоростью обучения = 1

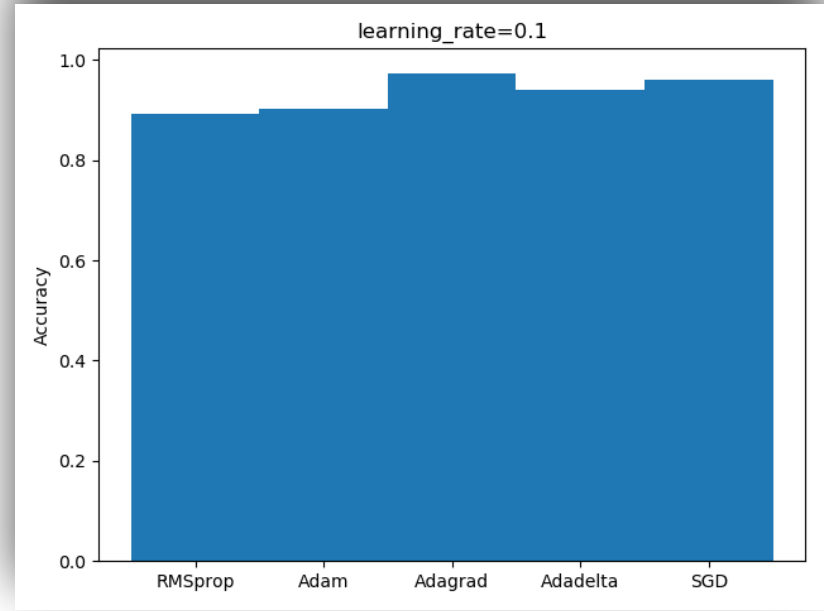


Рисунок 4 – График точности со скоростью обучения = 0.1

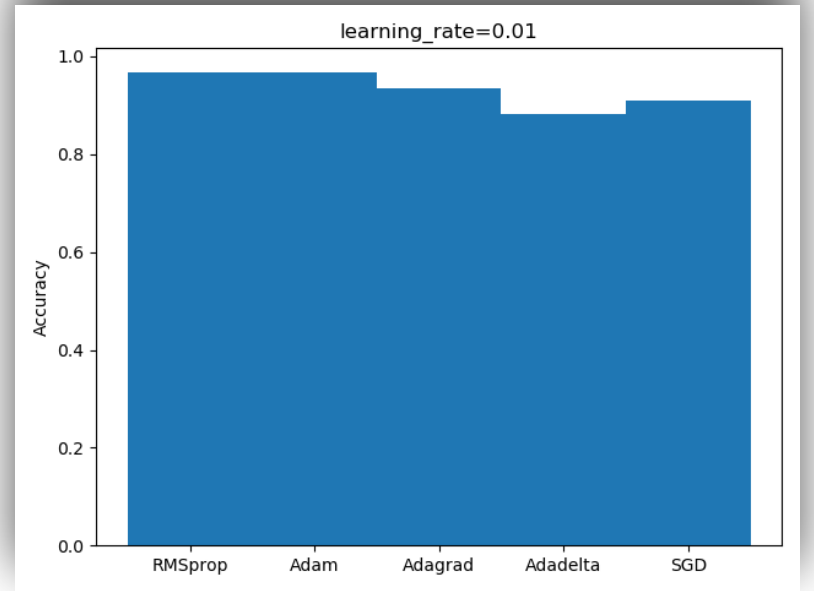


Рисунок 5 – График точности со скоростью обучения = 0.01

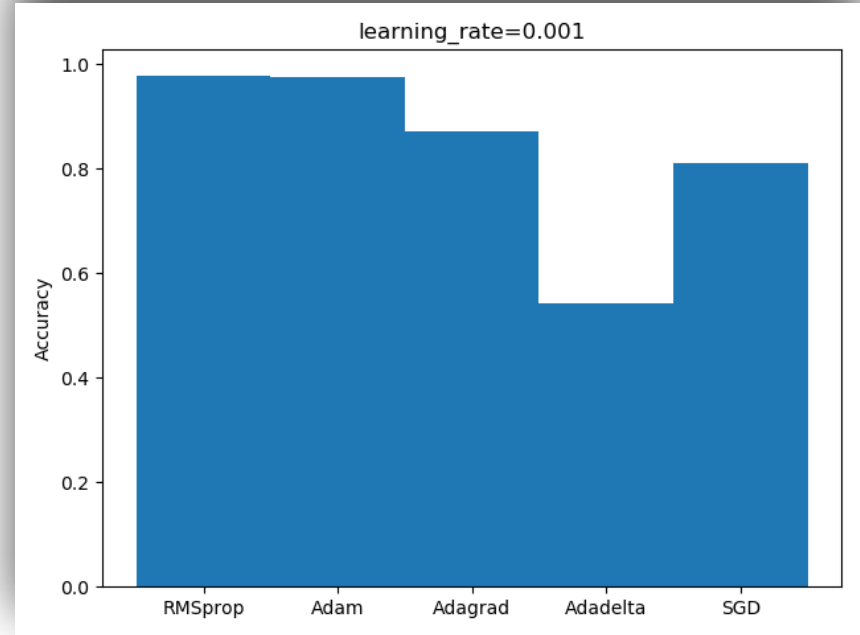


Рисунок 6 – График точности со скоростью обучения = 0.001

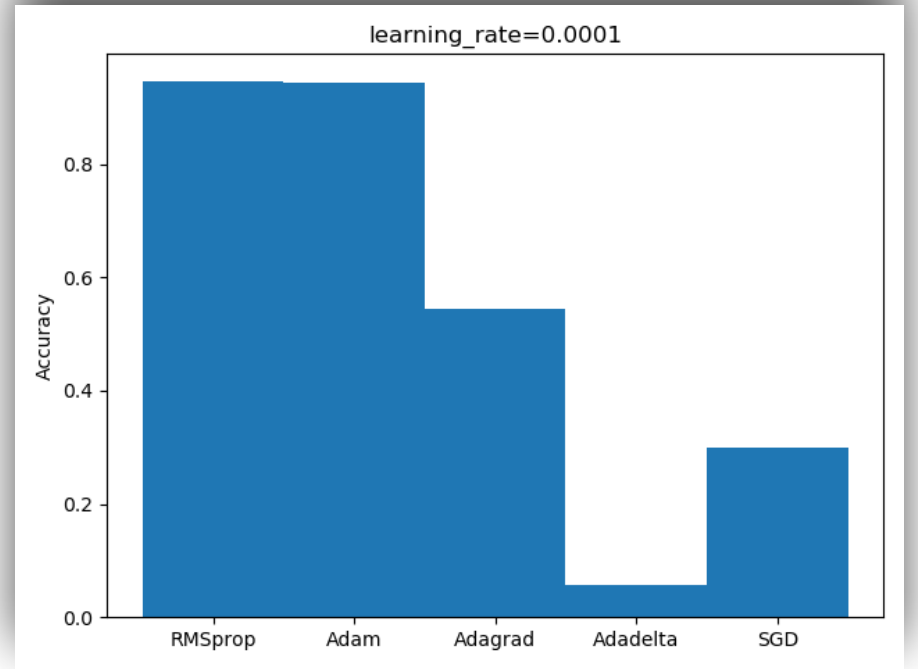


Рисунок 7 – График точности со скоростью обучения = 0.0001

Adadelta и SGD показывают наилучшие результаты при больших значениях скорости обучения, Adagrad – при средних значениях, а Adam и RMSprop – при малых.

Выводы.

Было изучено влияние различных оптимизаторов. Приобретен навык распознания рукописных символов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
import numpy as np
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import boston_housing
import matplotlib.pyplot as plt

(train_data, train_targets), (test_data, test_targets) =
boston_housing.load_data()


def build_model():
    model = Sequential()
    model.add(Dense(64, activation='relu',
input_shape=(train_data.shape[1],)))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='rmsprop', loss='mse',
metrics=['mae'])
    return model


def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point *
(1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std

k = 4
num_val_samples = len(train_data) // k
num_epochs = 100

all_val_mae_history = []
all_mae_history = []
```

```
all_val_loss_history = []
all_loss_history = []

for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) *
num_val_samples]
    val_targets = train_targets[i * num_val_samples:(i + 1) *
num_val_samples]
    partial_train_data = np.concatenate([train_data[:i *
num_val_samples], train_data[(i + 1) * num_val_samples:]],
axis=0)
    partial_train_targets = np.concatenate([train_targets[:i *
num_val_samples],train_targets[(i + 1) *
num_val_samples:]],axis=0)
    model = build_model()
    history = model.fit(partial_train_data,
partial_train_targets, validation_data=(val_data, val_targets),
epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    loss_history = history.history['val_loss']
    all_mae_history.append(mae_history)
    all_val_loss_history.append(loss_history)

    plt.plot(range(num_epochs), mae_history, label='Val mae')
    plt.title('k=' + str(i))
    plt.xlabel('Epochs')
    plt.ylabel('Mean absolute error')
    plt.show()

    plt.clf()
    plt.plot(range(num_epochs), loss_history, label='Val loss')
    plt.title('k=' + str(i))
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.show()
    plt.clf()

average_mae_history = [np.mean([x[i] for x in all_mae_history])
for i in range(num_epochs)]
smooth_mae_history = smooth_curve(average_mae_history[10:])
plt.plot(range(1, len(smooth_mae_history) + 1),
smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()

model = build_model()
model.fit(train_data, train_targets, epochs=80, batch_size=16,
verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data,
test_targets)
```