

Московский государственный технический университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



Отчет по
лабораторной работе № 5 по курсу
«Введение в машинное обучение»

Исполнитель: ИУ5-41, Черепанов Е.

Преподаватель: Гапанюк Ю.

Москва, 2018г.

Задание

Необходимо решить задачу распознавания рукописных цифр (датасет MNIST). Задача решается в рамках платформы онлайн-конкурсов по машинному обучению Kaggle.

1. Провести предподготовку данных

В задаче каждая фотография рукописной цифры задана в виде строки из 784 (28x28) значений градации серого для каждого пикселя. Градация от 0 до 255 (0 - белый, 255 - черный). Необходимо отнормировать значения для каждой картинки и получить train и validation датасеты.

2. Обучить логистическую регрессию на scikit-learn

Здесь нужно получить модель логистической регрессии и вычислить точность предсказания на валидационном датасете. Точность должна быть ~91%.

3. Создать модель многослойной нейронной сети в keras

Используя библиотеку для глубокого обучения keras.io необходимо создать нейронную сеть из нескольких слоев. Изучить возможности библиотеки, уметь отвечать на вопросы про Dense слои, методы активации, размерности входных/выходных матриц, метод compile и fit. Провести сравнение с логистической регрессией.

4. Построить графики обучения и сделать несколько архитектурных вариаций сети

Собрать историю обучения сети (см. выход функции fit) и построить график обучения (уменьшения loss на каждой итерации). Сделать несколько вариаций архитектуры сети

5. Провести эксперименты со своими рукописными цифрами

В любом графическом редакторе нарисовать набор из 10-20 цифр необходимого размера (28x28), сохранить в папке. Реализовать функцию, которая считывает все файлы из папки и преобразует в вектор. Прогнать нарисованные цифры через полученную на предыдущем этапе лучшую нейронную сеть и подсчитать % ошибок.

Код и результаты.

```
In [2]: import pandas as pd
import numpy as np
import keras
from sklearn import metrics
import matplotlib.pyplot as plt

%matplotlib inline
```

```
In [3]: #1. Провести предподготовку данных
dtrain = pd.read_csv('./train.csv')
dtest = pd.read_csv('./test.csv')
pixels = dtrain.columns.drop('label')
y_column = 'label'
```

```
In [4]: dtrain.head()
```

```
Out[4]:
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

< >

In [5]: dtest.head()

```
Out[5]:
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pi
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

5 rows × 784 columns

```
In [6]: #Каждый пиксель задан числом от 0 до 255. Для лучшей работы сети нормируем значения  
dtrain[pixels] = dtrain[pixels].applymap(lambda x: x / 255)
```

```
In [7]: #Разделим датасеты и преобразуем к необходимому формату  
from sklearn.model_selection import train_test_split  
train, validation = train_test_split(dtrain, test_size=0.2)  
x_train = train[pixels].values  
y_train = train[y_column].values  
  
x_val = validation[pixels].values  
y_val = validation[y_column].values  
y_train = y_train.reshape((y_train.shape[0], 1))  
y_val = y_val.reshape((y_val.shape[0], 1))  
print(x_train.shape, y_train.shape)
```

(33600, 784) (33600, 1)

```
In [8]: #2. Обучить логистическую регрессию на scikit-Learn  
from sklearn.linear_model import LogisticRegression  
  
LRmodel = LogisticRegression(fit_intercept=True) # параметр multi_class по дефолту равен 'ovr' (один-против-всех), модель сама сд  
LRmodel.fit(x_train, y_train)
```

```
Out[8]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
    intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,  
    penalty='l2', random_state=None, solver='liblinear', tol=0.0001,  
    verbose=0, warm_start=False)
```

```
In [9]: y_valid_predictions = LRmodel.predict(x_val)
```

```
In [10]: metrics.accuracy_score(y_valid_predictions, y_val)
```

```
Out[10]: 0.9189285714285714
```

```
In [11]: #3. Создать модель многослойной нейронной сети в keras  
from keras.layers import Dense  
from keras.models import Sequential  
  
model = Sequential() #создание модели  
#добавление слоёв (количество нейронов, activation - функция активации, input_dim - число признаков)  
model.add(Dense(1024, activation='relu', input_dim=len(pixels))) #ReLU - преобразование max(x, 0), если x > 0, то оставляем x, а е  
model.add(Dense(600, activation="relu"))  
model.add(Dense(10, activation='softmax')) #Softmax - это обобщение логистической функции для многомерного случая
```

```
In [12]: #Перед тем, как начать тренировать модель, ее нужно скомпилировать (loss - функция потерь, optimizer - оптимизатор, metrics - спи  
model.compile(loss='sparse_categorical_crossentropy',  
    optimizer='adam',  
    metrics=['accuracy'])
```

```
In [13]: history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_size=400)
```

Train on 33600 samples, validate on 8400 samples

```
Epoch 1/10
33600/33600 [=====] - 7s 222us/step - loss: 0.3585 - acc: 0.8942 - val_loss: 0.1599 - val_acc: 0.9520
Epoch 2/10
33600/33600 [=====] - 6s 186us/step - loss: 0.1221 - acc: 0.9624 - val_loss: 0.1153 - val_acc: 0.9643
Epoch 3/10
33600/33600 [=====] - 6s 191us/step - loss: 0.0733 - acc: 0.9779 - val_loss: 0.0869 - val_acc: 0.9739
Epoch 4/10
33600/33600 [=====] - 6s 182us/step - loss: 0.0461 - acc: 0.9863 - val_loss: 0.0823 - val_acc: 0.9742
Epoch 5/10
33600/33600 [=====] - 6s 181us/step - loss: 0.0271 - acc: 0.9922 - val_loss: 0.0826 - val_acc: 0.9758
Epoch 6/10
33600/33600 [=====] - 6s 181us/step - loss: 0.0179 - acc: 0.9957 - val_loss: 0.0823 - val_acc: 0.9756
Epoch 7/10
33600/33600 [=====] - 6s 179us/step - loss: 0.0125 - acc: 0.9968 - val_loss: 0.0804 - val_acc: 0.9771
Epoch 8/10
33600/33600 [=====] - 6s 188us/step - loss: 0.0069 - acc: 0.9989 - val_loss: 0.0832 - val_acc: 0.9767
Epoch 9/10
33600/33600 [=====] - 6s 175us/step - loss: 0.0055 - acc: 0.9989 - val_loss: 0.0872 - val_acc: 0.9776
Epoch 10/10
33600/33600 [=====] - 6s 175us/step - loss: 0.0043 - acc: 0.9991 - val_loss: 0.0886 - val_acc: 0.9774
```

```
In [14]: predictions = model.predict_classes(dtest, verbose=0)
```

```
In [15]: metrics.accuracy_score(LRmodel.predict(x_val), y_val)
```

```
Out[15]: 0.9189285714285714
```

```
In [16]: #4. Построить графики обучения
```

```
#уменьшение loss на каждой итерации
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```

