**Московский Государственный Технический Университет имени Н. Э. Баумана**



**Машинное обучение**

**Лабораторная работа №5**

**«Нейронные сети»**

ИСПОЛНИТЕЛЬ:

ФИО Гладова Анастасия

ИУ5-41 (ИУ5Ц-61Б)

ПРЕПОДАВАТЕЛЬ:

ФИО Гапанюк Ю.Е.

**Москва 2018**

# **Задание:**

Необходимо решить задачу распознавания рукописных цифр (датасет MNIST). Задача решается в рамках платформы онлайн-конкурсов по машинному обучению Kaggle. [Ссылка на задание](https://www.kaggle.com/c/digit-recognizer).

## **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 регрессиию на scikit-learn**

Здесь нужно получить модель логистической регресии 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 (28х28), 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.

Базовый пример реализации приведен в ноутбуке example.

# **Листинг:**

# **Step 1. LogRegression:**

## **# In[1]:**

import pandas as pd

import numpy as np

import keras

from sklearn import metrics

import matplotlib.pyplot as plt

%matplotlib inline

## **# In[2]:**

#1. Проведем предподготовку данных

dtrain = pd.read\_csv('C:/Anaconda/lab5/train.csv')

dtest = pd.read\_csv('C:/Anaconda/lab5/test.csv')

pixels = dtrain.columns.drop('label')

y\_column = 'label'

## **# In[3]:**

dtrain.head()

## **# In[4]:**

dtest.head()

## **# In[5]:**

#Каждый пиксель задан числом от 0 до 255

#Для лучшей работы сети отнормируем значения

dtrain[pixels] = dtrain[pixels].applymap(lambda x: x / 255)

## **# In[6]:**

#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)

## **# In[7]:**

#2. Обучение логистической регрессиии 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 сделает one hot encoding для ответов 'label' (10 классов)

LRmodel.fit(x\_train,y\_train)

## **# In[8]:**

y\_valid\_predictions = LRmodel.predict(x\_val)

## **# In[9]:**

metrics.accuracy\_score(y\_valid\_predictions, y\_val)

## **# In[10]:**

#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(64, activation='relu', input\_dim=len(pixels))) #ReLU — преобразование max(x, 0),если x > 0, то оставляем x, а если x < 0, то заменяем на 0

model.add(Dense(10, activation='softmax')) #Softmax — это обобщение логистической функции для многомерного случая

## **# In[11]:**

#Перед тем, как начать тренировать модель, ее нужно скомпилировать (loss – функция потерь, optimizer – оптимизатор, metrics – список метрик оптимизации)

model.compile(loss='sparse\_categorical\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

## **# In[12]:**

history = model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), epochs=10, batch\_size=400)

## **# In[13]:**

predictions = model.predict\_classes(dtest, verbose=0)

## **# In[14]:**

metrics.accuracy\_score(LRmodel.predict(x\_val), y\_val)

## **# In[15]:**

#4. Построение графика обучения

#уменьшение loss на каждой итерации

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

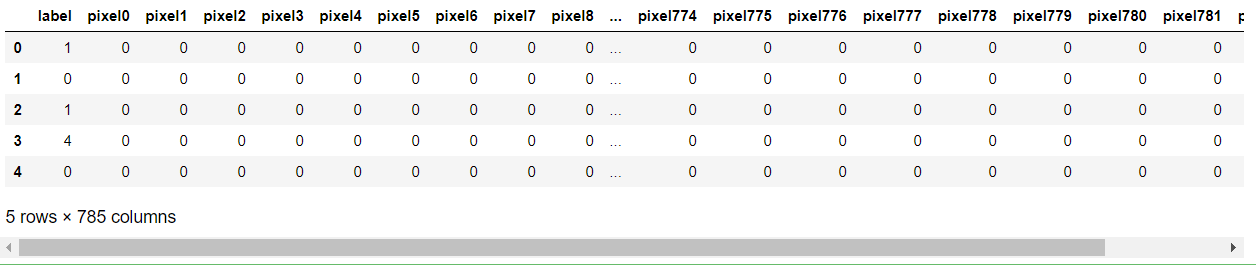
plt.xlabel('epoch')

plt.legend(['train', 'valid'], loc='upper left')

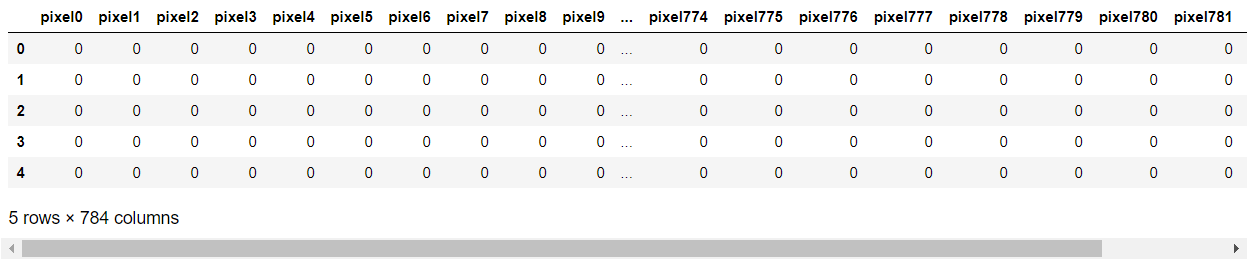
plt.show()

# **Результаты выполнения Step 1. LogRegression:**

## **# In[3]:**



## **# In[4]:**



## **# In[6]:**

(33600, 784) (33600, 1)

## **# In[7]:**

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]:**

0.11297619047619048

## **# In[12]:**

Train on 33600 samples, validate on 8400 samples

Epoch 1/10

33600/33600 [==============================] - 2s 64us/step - loss: 2.3019 - acc: 0.1105 - val\_loss: 2.3012 - val\_acc: 0.1130

Epoch 2/10

33600/33600 [==============================] - 2s 63us/step - loss: 2.3014 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 3/10

33600/33600 [==============================] - 2s 60us/step - loss: 2.3013 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 4/10

33600/33600 [==============================] - 2s 73us/step - loss: 2.3013 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 5/10

33600/33600 [==============================] - 2s 61us/step - loss: 2.3013 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 6/10

33600/33600 [==============================] - 2s 68us/step - loss: 2.3012 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 7/10

33600/33600 [==============================] - 2s 71us/step - loss: 2.3012 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 8/10

33600/33600 [==============================] - 2s 58us/step - loss: 2.3012 - acc: 0.1112 - val\_loss: 2.3010 - val\_acc: 0.1130

Epoch 9/10

33600/33600 [==============================] - 2s 58us/step - loss: 2.3011 - acc: 0.1112 - val\_loss: 2.3009 - val\_acc: 0.1130

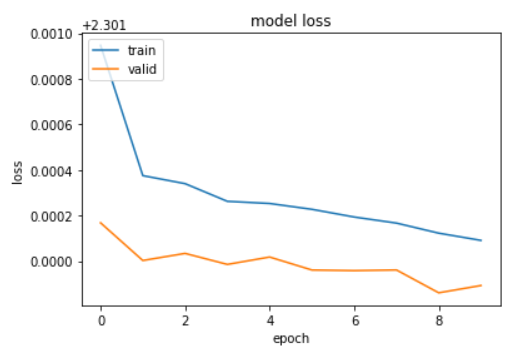
Epoch 10/10

33600/33600 [==============================] - 2s 60us/step - loss: 2.3011 - acc: 0.1112 - val\_loss: 2.3009 - val\_acc: 0.1130

## **# In[14]:**

0.11297619047619048

## **# In[15]:**



# **Step 2. Keras:**

## **# In[1]:**

import pandas as pd

import numpy as np

## **# In[2]:**

dtrain = pd.read\_csv('C:/Anaconda/lab5/train.csv')

dtest = pd.read\_csv('C:/Anaconda/lab5/test.csv')

## **# In[3]:**

from sklearn import cross\_validation

x\_train, x\_valid, y\_train, y\_valid = cross\_validation.train\_test\_split(

dtrain.drop(('label'), axis=1).astype('float32')/255.0,

dtrain['label'],

test\_size = 0.2,

random\_state = 11)

## **# In[4]:**

from keras.models import Sequential

from keras.layers import Dense

from keras import regularizers

model = Sequential()

model.add(Dense(800, input\_dim=x\_train.shape[1], activation="relu", kernel\_initializer="normal", kernel\_regularizer=regularizers.l2(0.01)))

model.add(Dense(600, activation="relu", kernel\_initializer="normal"))

model.add(Dense(10, activation="softmax", kernel\_initializer="normal"))

## **# In[5]:**

model.compile(loss='sparse\_categorical\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

## **# In[6]:**

history = model.fit(x\_train, y\_train, validation\_data=(x\_valid, y\_valid), epochs=10, batch\_size=400)

## **# In[7]:**

predictions = model.predict\_classes(dtest, verbose=0)

## **# In[8]:**

submissions=pd.DataFrame({"ImageId": list(range(1,len(predictions)+1)), "Label": predictions})

submissions.to\_csv("predictions.csv", index=False, header=True)

## **# In[9]:**

from sklearn import metrics

metrics.accuracy\_score(LRmodel.predict(x\_valid), y\_valid)

## **# In[10]:**

#сеть получилась лучше, чем логрег

import matplotlib.pyplot as plt

%matplotlib inline

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()

# **Результаты выполнения Step 2. Keras:**

## **# In[6]:**

Train on 33600 samples, validate on 8400 samples

Epoch 1/10

33600/33600 [==============================] - 14s 421us/step - loss: 6.4097 - acc: 0.8833 - val\_loss: 1.6182 - val\_acc: 0.9401

Epoch 2/10

33600/33600 [==============================] - 13s 399us/step - loss: 0.7987 - acc: 0.9455 - val\_loss: 0.4339 - val\_acc: 0.9450

Epoch 3/10

33600/33600 [==============================] - 13s 396us/step - loss: 0.3104 - acc: 0.9563 - val\_loss: 0.2856 - val\_acc: 0.9544

Epoch 4/10

33600/33600 [==============================] - 14s 403us/step - loss: 0.2527 - acc: 0.9595 - val\_loss: 0.2686 - val\_acc: 0.9554

Epoch 5/10

33600/33600 [==============================] - 13s 386us/step - loss: 0.2279 - acc: 0.9629 - val\_loss: 0.2362 - val\_acc: 0.9627

Epoch 6/10

33600/33600 [==============================] - 14s 407us/step - loss: 0.2114 - acc: 0.9664 - val\_loss: 0.2247 - val\_acc: 0.9620

Epoch 7/10

33600/33600 [==============================] - 14s 430us/step - loss: 0.2032 - acc: 0.9677 - val\_loss: 0.2218 - val\_acc: 0.9615

Epoch 8/10

33600/33600 [==============================] - 14s 402us/step - loss: 0.1918 - acc: 0.9699 - val\_loss: 0.2350 - val\_acc: 0.9565

Epoch 9/10

33600/33600 [==============================] - 13s 401us/step - loss: 0.1912 - acc: 0.9711 - val\_loss: 0.2128 - val\_acc: 0.9646

Epoch 10/10

33600/33600 [==============================] - 13s 394us/step - loss: 0.1762 - acc: 0.9731 - val\_loss: 0.2123 - val\_acc: 0.9645

## **# In[9]:**

0.6403571428571428

## **# In[10]:**

