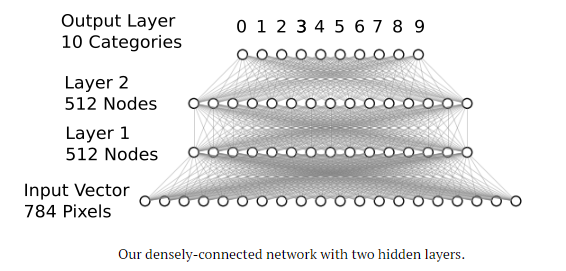
MNIST laba 2 09\_10\_2019

Простейшая MLP нейронная сеть, набор данных MNIST , вывод результатов текстом и в виде графиков



(2)Плюс визуализация первых девяти картинок

(3)Плюс визуализация распределения по интенсивности цвета ( серый) пикселей

(4) Плюс просмотр формы входных данных

(5) Плюс просмотр формы правильных выходных сигналов до категоризации и после

from keras.datasets import mnist # subroutines for fetching the MNIST dataset

from keras.models import Model # basic class for specifying and training a neural network

from keras.layers import Input, Dense # the two types of neural network layer we will be using

from keras.utils import np\_utils # utilities for one-hot encoding of ground truth value

import matplotlib.pyplot as plt

import numpy as np

batch\_size = 128 # in each iteration, we consider 128 training examples at once

num\_epochs = 20 # we iterate twenty times over the entire training set

hidden\_size = 512 # there will be 512 neurons in both hidden layers

num\_train = 60000 # there are 60000 training examples in MNIST

num\_test = 10000 # there are 10000 test examples in MNIST

height, width, depth = 28, 28, 1 # MNIST images are 28x28 and greyscale

num\_classes = 10 # there are 10 classes (1 per digit)

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data() # fetch MNIST data

# the beginning of the visualization of the first pictures

fig = plt.figure()

for i in range(9):

plt.subplot(3,3,i+1)

plt.tight\_layout()

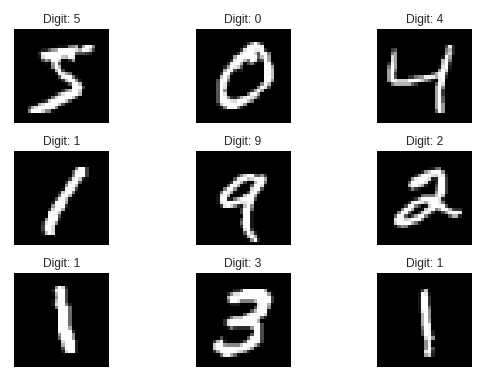
plt.imshow(X\_train[i], cmap='gray', interpolation='none')

plt.title("Digit: {}".format(y\_train[i]))

plt.xticks([])

plt.yticks([])

# the end of the visualization of the first pictures



# let's graph the distribution of our pixel values.

fig = plt.figure()

plt.subplot(2,1,1)

plt.imshow(X\_train[0], cmap='gray', interpolation='none')

plt.title("Digit: {}".format(y\_train[0]))

plt.xticks([])

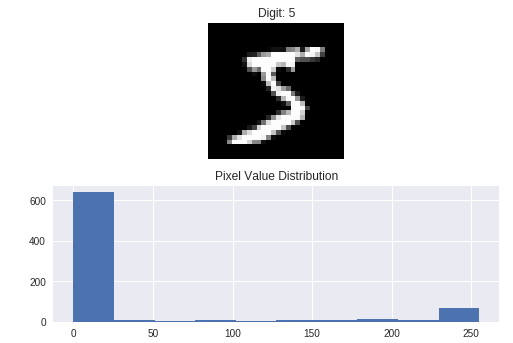
plt.yticks([])

plt.subplot(2,1,2)

plt.hist(X\_train[0].reshape(784))

plt.title("Pixel Value Distribution")

# end of graphing the distribution of our pixel values.



X\_train = X\_train.reshape(num\_train, height \* width) # Flatten data to 1D

X\_test = X\_test.reshape(num\_test, height \* width) # Flatten data to 1D

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train /= 255 # Normalise data to [0, 1] range

X\_test /= 255 # Normalise data to [0, 1] range

# print the final input shape ready for training

print("Train matrix shape", X\_train.shape)

print("Test matrix shape", X\_test.shape)

# end of printing the final input shape ready for training



#determine how many digits in the input set

print(np.unique(y\_train, return\_counts=True))

#end of determining how many digits in the input set



#(5)

print("Shape before one-hot encoding: ", y\_train.shape)

Y\_train = np\_utils.to\_categorical(y\_train, num\_classes) # One-hot encode the labels

Y\_test = np\_utils.to\_categorical(y\_test, num\_classes) # One-hot encode the labels

print("Shape after one-hot encoding: ", Y\_train.shape)

#(end of 5)



inp = Input(shape=(height \* width,)) # Our input is a 1D vector of size 784

hidden\_1 = Dense(hidden\_size, activation='relu')(inp) # First hidden ReLU layer

hidden\_2 = Dense(hidden\_size, activation='relu')(hidden\_1) # Second hidden ReLU layer

out = Dense(num\_classes, activation='softmax')(hidden\_2) # Output softmax layer

model = Model(inputs=inp, outputs=out) # To define a model, just specify its input and output layers

model.compile(loss='categorical\_crossentropy', # using the cross-entropy loss function

optimizer='adam', # using the Adam optimiser

metrics=['accuracy']) # reporting the accuracy

history = model.fit(X\_train, Y\_train, # Train the model using the training set...

batch\_size=batch\_size, epochs=num\_epochs,

verbose=1, validation\_split=0.1) # ...holding out 10% of the data for validation

ev=model.evaluate(X\_test, Y\_test, verbose=1) # Evaluate the trained model on the test set!

print(ev)

# list all data in history

print(history.history.keys())

# summarize history for accuracy

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

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

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

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

plt.show()

# summarize history for loss

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

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

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

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

plt.show()

Train on 54000 samples, validate on 6000 samples

Epoch 1/20

54000/54000 [==============================] - 3s 51us/step - loss: 0.2287 - acc: 0.9325 - val\_loss: 0.0991 - val\_acc: 0.9722

Epoch 2/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0834 - acc: 0.9737 - val\_loss: 0.0840 - val\_acc: 0.9745

Epoch 3/20

54000/54000 [==============================] - 3s 48us/step - loss: 0.0523 - acc: 0.9835 - val\_loss: 0.0756 - val\_acc: 0.9780

Epoch 4/20

54000/54000 [==============================] - 3s 48us/step - loss: 0.0383 - acc: 0.9878 - val\_loss: 0.0732 - val\_acc: 0.9775

Epoch 5/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0288 - acc: 0.9906 - val\_loss: 0.0666 - val\_acc: 0.9802

Epoch 6/20

54000/54000 [==============================] - 3s 48us/step - loss: 0.0222 - acc: 0.9928 - val\_loss: 0.0701 - val\_acc: 0.9815

Epoch 7/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0179 - acc: 0.9941 - val\_loss: 0.0718 - val\_acc: 0.9817

Epoch 8/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0153 - acc: 0.9947 - val\_loss: 0.0722 - val\_acc: 0.9823

Epoch 9/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0199 - acc: 0.9932 - val\_loss: 0.0889 - val\_acc: 0.9798

Epoch 10/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0146 - acc: 0.9954 - val\_loss: 0.0816 - val\_acc: 0.9808

Epoch 11/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0088 - acc: 0.9973 - val\_loss: 0.0876 - val\_acc: 0.9812

Epoch 12/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0116 - acc: 0.9960 - val\_loss: 0.0899 - val\_acc: 0.9798

Epoch 13/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0129 - acc: 0.9960 - val\_loss: 0.0823 - val\_acc: 0.9830

Epoch 14/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0102 - acc: 0.9968 - val\_loss: 0.0885 - val\_acc: 0.9815

Epoch 15/20

54000/54000 [==============================] - 3s 47us/step - loss: 0.0101 - acc: 0.9966 - val\_loss: 0.1207 - val\_acc: 0.9777

Epoch 16/20

54000/54000 [==============================] - 3s 48us/step - loss: 0.0108 - acc: 0.9964 - val\_loss: 0.0891 - val\_acc: 0.9823

Epoch 17/20

54000/54000 [==============================] - 3s 48us/step - loss: 0.0043 - acc: 0.9989 - val\_loss: 0.0852 - val\_acc: 0.9848

Epoch 18/20

54000/54000 [==============================] - 3s 49us/step - loss: 0.0107 - acc: 0.9964 - val\_loss: 0.1184 - val\_acc: 0.9778

Epoch 19/20

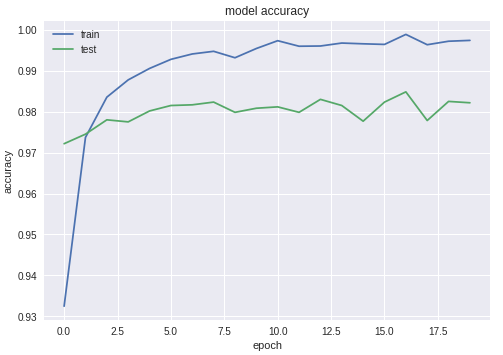
54000/54000 [==============================] - 3s 48us/step - loss: 0.0093 - acc: 0.9972 - val\_loss: 0.0912 - val\_acc: 0.9825

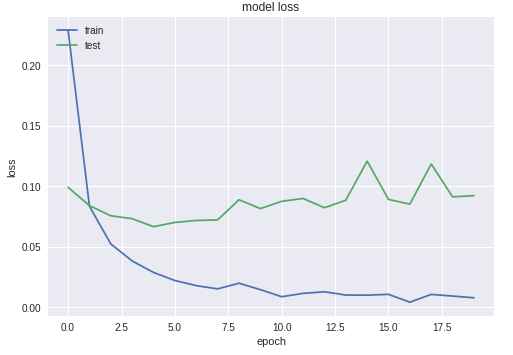
Epoch 20/20

54000/54000 [==============================] - 3s 48us/step - loss: 0.0079 - acc: 0.9974 - val\_loss: 0.0923 - val\_acc: 0.9822

10000/10000 [==============================] - 0s 47us/step

dict\_keys(['val\_loss', 'val\_acc', 'loss', 'acc'])





Задания:

1. Вывести и сравнить распределения пикселей по цветам для цифр 0, 1, …, 9
2. Вывести и сравнить распределения пикселей по цветам для различных изображений цифры 9
3. Произвести обучение нейронной сети для количества скрытых слоёв 2, 3, 4, 5. Сделать табличку для результирующей вероятности правильного распознавания цифр для тренировочных данных и для валидационных. Нарисовать график, где изображены все четыре кривых обучения
4. Сделать выводы