МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
 (СПБГМТУ)

| ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ  КАФЕДРА КИБЕРФИЗИЧЕСКИХ СИСТЕМ |
| --- |

ЛАБОРАТОРНАЯ РАБОТА №6

«МОДЕЛЬ КЛАССИФИКАЦИИ РУКОПИСНЫХ ЦИФР»

|  | |
| --- | --- |
|  | Выполнил  студент группы 20221 | | |
|  | Лаптев Иван Александрович | | |
|  | Проверила | | |
|  | Кайнова Татьяна Денисовна | | |
| Санкт-Петербург  2024 год | | |

**СОДЕРЖАНИЕ**

[Цели работы………………………………………………………………………. 2](#_gjdgxs)

[Результат работы………………………………………………………………….. 3](#_30j0zll)

[Заключение………………………………………………………………………...](#_3dy6vkm) 6

Листинг кода………………………………………………………………………9

# Цели работы

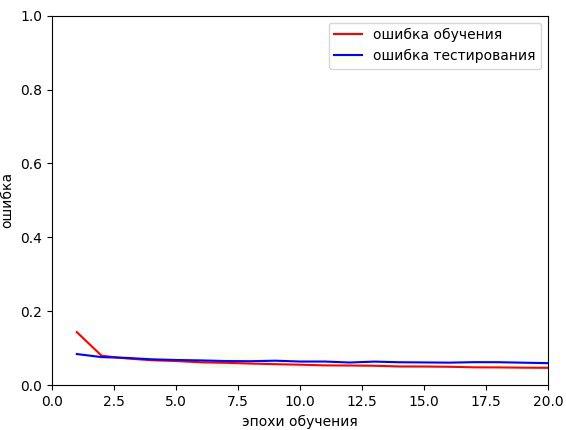
При работе над данной лабораторной работой будут затронуты следующие вопросы:

1. Импорт и преобразование данных с помощью библиотеки idx2numpy.
2. Работа с библиотеками matplotlib и numpy.
3. Разработка архитектуры нейронной сети и достижении точности более 95.5% на тестовом наборе данных.
4. Работа с данными для увеличения обучающей выборки.
5. Сравнение результатов нейронных сетей, обученных на стандартном наборе и на расширенном, а также анализ результатов.

# Результат работы

В ходе лабораторной работе было написано две программы.

Первая программа (Листинг 1) была написана путём объединния фрагментов, представленных в лекции (Рисунок 1). Изменяя количества нейронов на скрытом слое с 25 до 50 удалось достичь точности в 95.5% (Рисунок 2).

  
Рисунок 1 - Результат обучения модели с 25 нейронами на скрытом слое

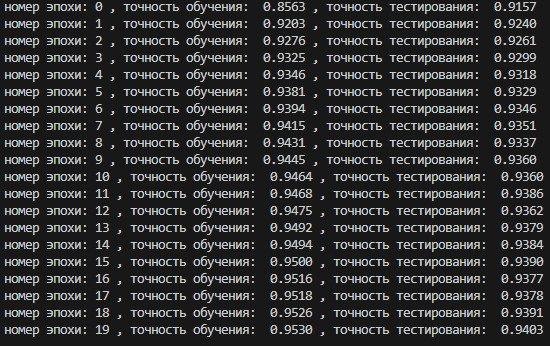
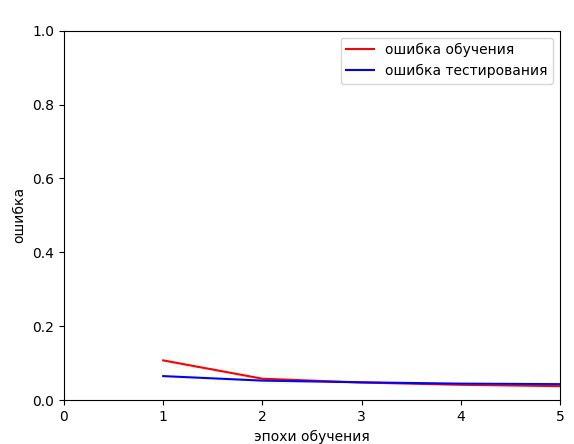


Рисунок 2 - Вывод номера эпох, точность обучения и тестирование для Рисунка 1

   
Рисунок 3 - Результат обучения модели с 50 нейронами на скрытом слое

Вторая программа (Листинг 2) была написана с целью анализа результата обучения на расширенном наборе MNIST. Для этого для каждого изображения были созданы 4 копии, сдвинутые в разные стороны: вправо, влево, вверх или вниз. Таким образом изначальнальный тренировочный набор с 60000 изображений расширился до 300000 изображений (Рисунок 3).

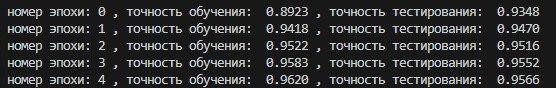
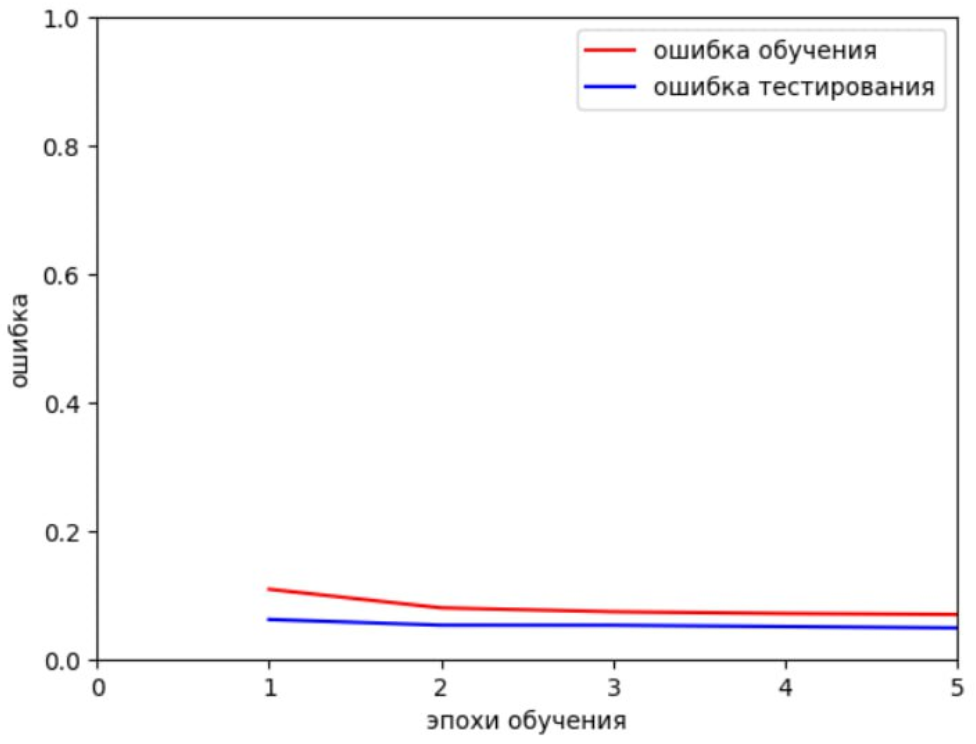


Рисунок 4 - Вывод номера эпох, точность обучения и тестирование для Рисунка 3

  
Рисунок 5 - Результат обучения модели на расширенном наборе данных MNIST с 25 нейронами на скрытом слое

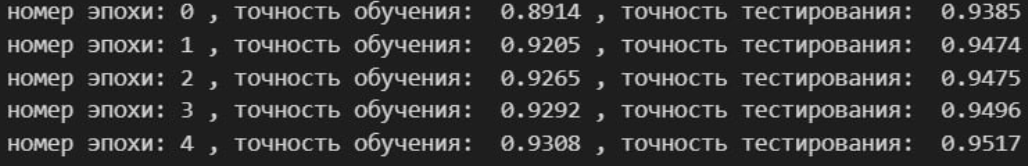
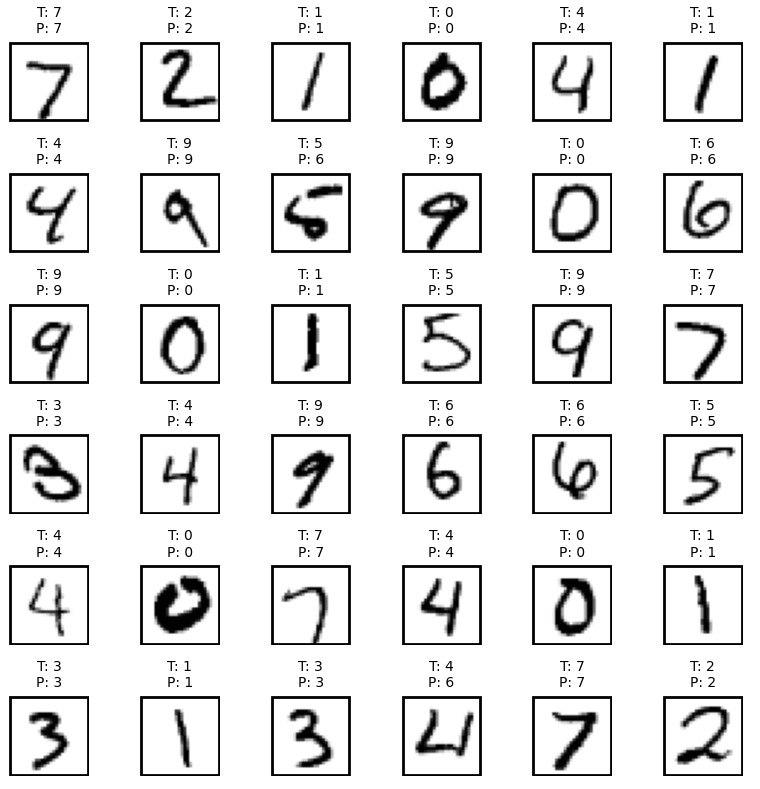


Рисунок 6 - Вывод номера эпох, точность обучения и тестирование для Рисунка 5

Анализируя точности обучения и тестирования, можно заметить, что модель, которая обучалась на расширенном наборе данных, отстаёт от модели, которая обучалась на обычном наборе данных, в точности обучения на 0.42%, но опережает в точности тестирования на 1.98%.

Так же все три модели были протестированы на базе данных MNIST. Все три модели показали один и тот же результат (Рисунок 4).

Рисунок 4 - Тестирование модели на 36 примерах набора данных MNIST

# Заключение

В ходе лабораторной работы был получен опыт работы с такими вопросами, как:

1. Импорт и преобразование данных с помощью библиотеки idx2numpy.
2. Работа с библиотеками matplotlib и numpy.
3. Разработка архитектуры нейронной сети и достижении точности более 95.5% на тестовом наборе данных.
4. Работа с данными для увеличения обучающей выборки.
5. Сравнение результатов нейронных сетей, обученных на стандартном наборе и на расширенном, а также анализ результатов.

# Листинг кода

Листинг 1 - Модель, использующая стандартный набор данных MNIST

import numpy as np

import matplotlib.pyplot as plt

import idx2numpy

np.random.seed(7)

LEARNING\_RATE = 0.01

EPOCHS = 5

# укажите пути к расположению обучающих и тестовых данных через

# TRAIN\_IMAGE\_FILENAME, TRAIN\_LABEL\_FILENAME,

# TEST\_IMAGE\_FILENAME и TEST\_LABEL\_FILENAME

TRAIN\_IMAGE\_FILENAME = 'data/mnist/train-images.idx3-ubyte'

TRAIN\_LABEL\_FILENAME = 'data/mnist/train-labels.idx1-ubyte'

TEST\_IMAGE\_FILENAME = 'data/mnist/t10k-images.idx3-ubyte'

TEST\_LABEL\_FILENAME = 'data/mnist/t10k-labels.idx1-ubyte'

def read\_mnist():

train\_images = idx2numpy.convert\_from\_file(TRAIN\_IMAGE\_FILENAME)

train\_labels = idx2numpy.convert\_from\_file(TRAIN\_LABEL\_FILENAME)

test\_images = idx2numpy.convert\_from\_file(TEST\_IMAGE\_FILENAME)

test\_labels = idx2numpy.convert\_from\_file(TEST\_LABEL\_FILENAME)

# добавьте в эту функцию чтение файлов с обучающими

# и тестовыми наборами данных, их метками в train\_images,

# train\_labels, test\_images и test\_labels

# добавьте сюда же фрагменты кода 1.2 и 1.3, приведённые ниже

#2

x\_train = train\_images.reshape(60000, 784)

mean = np.mean(x\_train)

stddev = np.std(x\_train)

x\_train = (x\_train - mean) / stddev

x\_test = test\_images.reshape(10000, 784)

x\_test = (x\_test - mean) / stddev

#3

y\_train = np.zeros((60000, 10))

y\_test = np.zeros((10000, 10))

for i, y in enumerate(train\_labels):

y\_train[i][y] = 1

for i, y in enumerate(test\_labels):

y\_test[i][y] = 1

return x\_train, y\_train, x\_test, y\_test

x\_train, y\_train, x\_test, y\_test = read\_mnist()

index\_list = list(range(len(x\_train)))

def layer\_w(neuron\_count, input\_count):

weights = np.zeros((neuron\_count, input\_count+1))

for i in range(neuron\_count):

for j in range(1, (input\_count+1)):

weights[i][j] = np.random.uniform(-0.1, 0.1)

return weights

# Объявляем матрицы и вектора, представляющие нейроны

hidden\_layer\_w = layer\_w(25, 784)

hidden\_layer\_y = np.zeros(25)

hidden\_layer\_error = np.zeros(25)

output\_layer\_w = layer\_w(10, 25)

output\_layer\_y = np.zeros(10)

output\_layer\_error = np.zeros(10)

chart\_x = []

chart\_y\_train = []

chart\_y\_test = []

def show\_learning(epoch\_no, train\_acc, test\_acc):

global chart\_x

global chart\_y\_train

global chart\_y\_test

print('номер эпохи:', epoch\_no, ', точность обучения: ', '%6.4f' % train\_acc,', точность тестирования: ', '%6.4f' % test\_acc)

chart\_x.append(epoch\_no + 1)

chart\_y\_train.append(1.0 - train\_acc)

chart\_y\_test.append(1.0 - test\_acc)

def plot\_learning():

plt.plot(chart\_x, chart\_y\_train, 'r-', label='ошибка обучения')

plt.plot(chart\_x, chart\_y\_test, 'b-', label='ошибка тестирования')

plt.axis([0, len(chart\_x), 0.0, 1.0])

plt.xlabel('эпохи обучения')

plt.ylabel('ошибка')

plt.legend()

plt.show()

def forward\_pass(x):

global hidden\_layer\_y

global output\_layer\_y

# Функция активации для нейронов скрытого слоя

for i, w in enumerate(hidden\_layer\_w):

z = np.dot(w, x)

hidden\_layer\_y[i] = np.tanh(z)

hidden\_output\_array = np.concatenate((np.array([1.0]), hidden\_layer\_y))

# Функция активации для нейронов выходного слоя

for i, w in enumerate(output\_layer\_w):

z = np.dot(w, hidden\_output\_array)

output\_layer\_y[i] = 1.0 / (1.0 + np.exp(-z))

def backward\_pass(y\_truth):

global hidden\_layer\_error

global output\_layer\_error

# Обратное распространение ошибки для каждого выходного нейрона

# и создание массива всех ошибок выходного нейрона.

for i, y in enumerate(output\_layer\_y):

error\_prime = -(y\_truth[i] - y) # Производная потерь

derivative = y \* (1.0 - y) # Производная логистической ф-ии

output\_layer\_error[i] = error\_prime \* derivative

for i, y in enumerate(hidden\_layer\_y):

# Создание массива весов, соединяющих выход скрытого

# нейрона i с с нейронами в выходном слое

error\_weights = []

for w in output\_layer\_w:

error\_weights.append(w[i+1])

error\_weight\_array = np.array(error\_weights)

# Обратное распространение для скрытых нейронов

derivative = 1.0 - y\*\*2 # производная ф-ии tanh

weighted\_error = np.dot(error\_weight\_array, output\_layer\_error)

hidden\_layer\_error[i] = weighted\_error \* derivative

def adjust\_weights(x):

global output\_layer\_w

global hidden\_layer\_w

for i, error in enumerate(hidden\_layer\_error):

hidden\_layer\_w[i] -= (x \* LEARNING\_RATE \* error) # Обновляем все веса

hidden\_output\_array = np.concatenate((np.array([1.0]), hidden\_layer\_y))

for i, error in enumerate(output\_layer\_error):

output\_layer\_w[i] -= (hidden\_output\_array \* LEARNING\_RATE \* error) # Обновляем все веса

# Цикл обучения сети

for i in range(EPOCHS): # Отсчёт эпох

np.random.shuffle(index\_list) # Случайный порядок

correct\_training\_results = 0

for j in index\_list: # Обучение на всех примерах

x = np.concatenate((np.array([1.0]), x\_train[j]))

forward\_pass(x)

if output\_layer\_y.argmax() == y\_train[j].argmax():

correct\_training\_results += 1

backward\_pass(y\_train[j])

adjust\_weights(x)

correct\_test\_results = 0

for j in range(len(x\_test)): # Оценка сети

x = np.concatenate((np.array([1.0]), x\_test[j]))

forward\_pass(x)

if output\_layer\_y.argmax() == y\_test[j].argmax():

correct\_test\_results += 1

# Вывод на экран прогресса обучения

show\_learning(i, correct\_training\_results/len(x\_train), correct\_test\_results/len(x\_test))

plot\_learning() # Вывод графика

Листинг 2 - Модель, использующая расширенный набор данных MNIST

import numpy as np

import matplotlib.pyplot as plt

import idx2numpy

np.random.seed(7)

LEARNING\_RATE = 0.01

EPOCHS = 5

TRAIN\_IMAGE\_FILENAME = 'data/mnist/train-images.idx3-ubyte'

TRAIN\_LABEL\_FILENAME = 'data/mnist/train-labels.idx1-ubyte'

TEST\_IMAGE\_FILENAME = 'data/mnist/t10k-images.idx3-ubyte'

TEST\_LABEL\_FILENAME = 'data/mnist/t10k-labels.idx1-ubyte'

def move(array, shift, axis): # сдвигает на один пиксель изображения

moved = np.roll(array, shift, axis=axis)

if axis == 1:

if shift == 1:

moved[:, 0] = 0

else:

moved[:, -1] = 0

else:

if shift == 1:

moved[0, :] = 0

else:

moved[-1, :] = 0

return moved

def read\_mnist():

train\_images = idx2numpy.convert\_from\_file(TRAIN\_IMAGE\_FILENAME)

train\_labels = idx2numpy.convert\_from\_file(TRAIN\_LABEL\_FILENAME)

test\_images = idx2numpy.convert\_from\_file(TEST\_IMAGE\_FILENAME)

test\_labels = idx2numpy.convert\_from\_file(TEST\_LABEL\_FILENAME)

for i in range(len(train\_images)):

u = move(train\_images[i], -1, 0)

d = move(train\_images[i], 1, 0)

r = move(train\_images[i], 1, 1)

l = move(train\_images[i], -1, 1)

train\_images = np.concatenate((train\_images, np.array([u, d, r, l])))

train\_labels = np.concatenate((train\_labels, np.array([train\_labels[i] for \_ in range(4)])))

x\_train = train\_images.reshape(300000, 784)

mean = np.mean(x\_train) # среднее значение элементов массива

stddev = np.std(x\_train) # стандартное отклонение

x\_train = (x\_train - mean) / stddev

x\_test = test\_images.reshape(10000, 784)

x\_test = (x\_test - mean) / stddev

y\_train = np.zeros((300000, 10))

y\_test = np.zeros((10000, 10))

for i, y in enumerate(train\_labels):

y\_train[i][y] = 1

for i, y in enumerate(test\_labels):

y\_test[i][y] = 1

return x\_train, y\_train, x\_test, y\_test

x\_train, y\_train, x\_test, y\_test = read\_mnist()

index\_list = list(range(len(x\_train)))

def layer\_w(neuron\_count, input\_count):

weights = np.zeros((neuron\_count, input\_count + 1))

for i in range(neuron\_count):

for j in range(1, (input\_count + 1)):

weights[i][j] = np.random.uniform(-0.1, 0.1)

return weights

# Объявляем матрицы и вектора, представляющие нейроны

hidden\_layer\_w = layer\_w(25, 784)

hidden\_layer\_y = np.zeros(25)

hidden\_layer\_error = np.zeros(25)

output\_layer\_w = layer\_w(10, 25)

output\_layer\_y = np.zeros(10)

output\_layer\_error = np.zeros(10)

chart\_x = []

chart\_y\_train = []

chart\_y\_test = []

def show\_learning(epoch\_no, train\_acc, test\_acc):

global chart\_x

global chart\_y\_train

global chart\_y\_test

print('номер эпохи:', epoch\_no, ', точность обучения: ',

'%6.4f' % train\_acc,

', точность тестирования: ', '%6.4f' % test\_acc)

chart\_x.append(epoch\_no + 1)

chart\_y\_train.append(1.0 - train\_acc)

chart\_y\_test.append(1.0 - test\_acc)

def plot\_learning():

plt.plot(chart\_x, chart\_y\_train, 'r-', label='ошибка обучения')

plt.plot(chart\_x, chart\_y\_test, 'b-', label='ошибка тестирования')

plt.axis([0, len(chart\_x), 0.0, 1.0])

plt.xlabel('эпохи обучения')

plt.ylabel('ошибка')

plt.legend()

plt.show()

def forward\_pass(x):

global hidden\_layer\_y

global output\_layer\_y

# Функция активации для нейронов скрытого слоя

for i, w in enumerate(hidden\_layer\_w):

z = np.dot(w, x)

hidden\_layer\_y[i] = np.tanh(z)

hidden\_output\_array = np.concatenate((np.array([1.0]), hidden\_layer\_y))

# Функция активации для нейронов выходного слоя

for i, w in enumerate(output\_layer\_w):

z = np.dot(w, hidden\_output\_array)

output\_layer\_y[i] = 1.0 / (1.0 + np.exp(-z))

def backward\_pass(y\_truth):

global hidden\_layer\_error

global output\_layer\_error

# Обратное распространение ошибки для каждого выходного нейрона

# и создание массива всех ошибок выходного нейрона.

for i, y in enumerate(output\_layer\_y):

error\_prime = -(y\_truth[i] - y) # Производная потерь

derivative = y \* (1.0 - y) # Производная логистической ф-ии

output\_layer\_error[i] = error\_prime \* derivative

for i, y in enumerate(hidden\_layer\_y):

# Создание массива весов, соединяющих выход скрытого

# нейрона i с с нейронами в выходном слое

error\_weights = []

for w in output\_layer\_w:

error\_weights.append(w[i + 1])

error\_weight\_array = np.array(error\_weights)

# Обратное распространение для скрытых нейронов

derivative = 1.0 - y \*\* 2 # производная ф-ии tanh

weighted\_error = np.dot(error\_weight\_array, output\_layer\_error)

hidden\_layer\_error[i] = weighted\_error \* derivative

def adjust\_weights(x):

global output\_layer\_w

global hidden\_layer\_w

for i, error in enumerate(hidden\_layer\_error):

hidden\_layer\_w[i] -= (x \* LEARNING\_RATE \* error) # Обновляем все веса

hidden\_output\_array = np.concatenate((np.array([1.0]), hidden\_layer\_y))

for i, error in enumerate(output\_layer\_error):

output\_layer\_w[i] -= (hidden\_output\_array \* LEARNING\_RATE \* error) # Обновляем все веса

# Цикл обучения сети

for i in range(EPOCHS): # Отсчёт эпох

np.random.shuffle(index\_list) # Случайный порядок

correct\_training\_results = 0

for j in index\_list: # Обучение на всех примерах

x = np.concatenate((np.array([1.0]), x\_train[j]))

forward\_pass(x)

if output\_layer\_y.argmax() == y\_train[j].argmax():

correct\_training\_results += 1

backward\_pass(y\_train[j])

adjust\_weights(x)

correct\_test\_results = 0

for j in range(len(x\_test)): # Оценка сети

x = np.concatenate((np.array([1.0]), x\_test[j]))

forward\_pass(x)

if output\_layer\_y.argmax() == y\_test[j].argmax():

correct\_test\_results += 1

# Вывод на экран прогресса обучения

show\_learning(i, correct\_training\_results / len(x\_train), correct\_test\_results / len(x\_test))

plot\_learning() # Вывод графика

num = 8

fig, axes = plt.subplots(6, 6, figsize=(10, 10))

for i in range(36):

x = x\_test[i]

forward\_pass(np.concatenate((np.array([1.0]), x)))

prediction = output\_layer\_y.argmax()

true\_label = y\_test[i].argmax()

color = "black"

row, col = i // 6, i % 6

image = x.reshape(28, 28)

ax = axes[row, col]

ax.imshow(image, cmap=plt.cm.binary)

ax.axis("off")

ax.set\_title(f"{true\_label} ({sign})", fontsize=12)

ax.add\_patch(plt.Rectangle((0, 0), 27, 27, fill=False, edgecolor=color, linewidth=2))

plt.tight\_layout()

plt.show()