# Практическая работа 4.

# Введение в проектирование нейронных сетей с помощью Python

**Обзор**

Почему нейронная сеть?

Предыдущая лабораторная работа была посвящена аналитическому подходу к анализу данных. Преимущества нейронных сетей проистекают из распараллеливания обработки информации и способности к самообучению, то есть создавать обобщения. Термин обобщение относится к способности получить достоверный результат на основе данных, которые не были обнаружены в процессе обучения. Эти свойства позволяют нейронным сетям решать сложные (крупномасштабные) задачи, которые считаются сложными для решения аналитическими методами.

Вот несколько примеров задач, решаемых нейронными сетями:

• Подгонка функции по набору точек (регрессия).

• Классификация данных по заданному набору классов.

• Кластеризация данных с выявлением ранее неизвестных классов прототипов.

• Сжатие информации.

• Восстановление потерянных данных.

• Ассоциативная память.

• Оптимизация, оптимальное управление.

*Какие знания и навыки вы приобретете, выполнив эту лабораторную работу?*

Выполнив эту лабораторную работу, вы получите следующие знания и навыки:

• Вы узнаете о структуре и функционировании нейронных сетей.

• Вы сможете создавать простые нейронные сети на Python.

• Вы сможете готовить данные для обучения и обучать простые нейронные сети на Python.

• Вы сможете решать задачи классификации данных.

Лабораторное рабочее задание

Одна из задач киберфизических систем - распознавание изображений с помощью машинного зрения. В этой практической работе мы рассмотрим решение проблемы такого типа.

Мы будем использовать два файла данных с массивом рукописных чисел: mnist\_train.csv и mnist\_test.csv. Первый файл необходимо использовать для обучения нейронной сети, второй файл необходимо использовать для проверки работы нейронной сети. Каждый файл содержит строки (60000 в первом файле и 10000 во втором файле), в каждой из которых хранится массив размером 28x28 пикселей с цифровым изображением и номером, соответствующим изображению. Пример данных показан на следующем рисунке:

Изображение выглядит как рисунок

Автоматически созданное описание

Figure 1.

Основная задача - спроектировать нейронную сеть на Python, способную распознавать изображения чисел. Необходимо:

1. Импортировать библиотеки в Python.

2. Написать функцию для установки основных сетевых параметров.

3. Создать функцию, которая устанавливает начальные значения весов нейронной сети.

4. Создать функцию, которая вычисляет выходной сигнал нейронной сети.

5. Создать функцию для обучения нейронной сети.

6. Написать функцию для обучения сети на реальных данных.

7. Написать функцию проверки сети.

8. Написать функцию, отображающую изображения чисел из набора данных.

9. Обучите сеть и рассчитайте ее эффективность.

**Теория**

**Нейрон.** Искусственные нейронные сети состоят из нейронов, поэтому знакомство с ними мы начнем с математической модели нейрона. Искусственный нейрон - это упрощенная модель естественного нейрона. У него много входов. Каждый вход имеет свой вес (wn). Внутри нейрона сумма входов S обрабатывается функцией активации F (S), результатом вычисления которой стал выход нейрона:

Изображение выглядит как карта

Автоматически созданное описание

Рисунок 2.

Таким образом, нейрон можно описать математически:



или:



где:

 вектор весов входных сигналов;

 вектор-столбец значений входного сигнала.

В качестве функции активации чаще всего используются нелинейные функции, поскольку только такие нейроны позволяют решать нетривиальные задачи с небольшим количеством узлов. Две наиболее распространенные функции:

* , который нормализован на интервал [-1, 1];
* , который нормализован на интервал [0;1].

Обе функции сигмовидные. В лабораторной работе мы будем использовать вторую функцию.

**Персептрон.** Нейроны собраны слоями: входной, скрытый, выходной. Первые сигналы отправляются на входной уровень. Затем они передаются в скрытые слои (количество скрытых слоев может быть разное). Здесь информация обрабатывается по описанному выше алгоритму. Из последнего скрытого слоя информация передается на выходной слой. Выходной сигнал - это желаемый результат обработки информации внутри нейронной сети (например, это может быть классификация объектов, распознавание и т.д.):Изображение выглядит как текст, карта

Автоматически созданное описание

Рисунок 3.

Выход любого слоя можно математически рассчитать следующим образом:



Где:

 вектор весов входного сигнала;

 вектор-столбец входных значений слоя;

 вектор-столбец выходных значений слоя;

 - матрица весов входных сигналов слоя (*n* - количество входных сигналов, m - количество нейронов в слое).

Матричное представление расчета нейронных сетей очень удобно, так как расчет нейронной сети можно записать в одну строку. Например, для сети с одним скрытым слоем расчет будет выглядеть так:



где  матрица весов скрытого слоя,  матрица весов выходного слоя.

**Обучение нейронной сети.** Теперь необходимо выяснить, как назначить веса связям, чтобы нейронная сеть давала необходимые результаты.

В зависимости от назначения нейронные сети обучаются по-разному. В нашей задаче нам нужно научить нейронную сеть распознавать изображения чисел. Мы будем использовать набор тестовых сигналов в качестве входных сигналов I нейронной сети и вычислить набор выходных сигналов O, которые мы будем сравнивать с желаемыми результатами тестирования. Ошибка выходного слоя рассчитывается следующим образом:



где

 - вектор-столбец значений желаемых результатов тестирования нейронной сети;

 вектор-столбец выходных значений слоя;

 вектор-столбец значений ошибок выходного слоя.

Ошибка скрытого слоя рассчитывается следующим образом:



Очевидно, что если имеется более одного скрытого слоя, ошибка вычисляется путем последовательного умножения на транспонированные весовые матрицы.

Веса изменяются методом градиентного спуска:





Здесь α - коэффициент обучения, который характеризует скорость изменения весов.

## Этапы выполнения практической работы

**1. Импорт библиотек.** Во-первых, нам нужно импортировать три библиотеки: numpy, matplotlib и scipy. Нам знакомы первая и вторая библиотека из предыдущей лабораторной. В третьей библиотеке нам нужна только одна функция: сигмоид .

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.special import expit as f\_act

**2. Написать функцию для установки основных сетевых параметров.** Во-первых, нам нужно установить количество нейронов во входном, выходном и скрытом слоях. Еще нам нужно установить коэффициент обучения α (с какой скоростью будет обучаться сеть от 0 до 1), он нам понадобится позже.

Для обучения сети возьмем готовую базу MNIST (аббревиатура от «Modified National Institute of Standards and Technology») - объемная база данных образцов почерка чисел. База данных — это стандарт, предложенный Национальным институтом стандартов и технологий США с целью калибровки и сравнения методов распознавания изображений с использованием машинного обучения, в основном на основе нейронных сетей.

Данные представлены в удобном текстовом формате через запятую. Учебный комплект содержит около 60 000 промаркированных образцов, меньший тестовый набор - около 10 000.

Каждый образец содержит маркер от 0 до 9 (какая цифра показана) и массив 28x28 пикселей. Всего 784 значения от 0 до 255. Следовательно, нам нужно 784 входных сигнала по количеству пикселей в массиве и 10 выходных значений по количеству цифр. Можно выбрать любое количество нейронов в скрытом слое.

Вызовем функцию init\_net ():

def init\_net():

Определение функции начинается с ключевого слова def. Тело функции (процедуры, которые должны выполняться при вызове функции) записывается с дополнительными отступами.

Установим количество входных сигналов (input\_nodes) равным 784:

input\_nodes = 784

Затем мы напишем запрос на ввод количества нейронов в скрытом слое в рабочую область и дождемся ввода значения функцией input (). Введенное значение преобразуется в целое число с помощью функции int ():

print(**'Input the number of hidden neurons: '**)  
hidden\_nodes = int(input())

Установим количество выходов (out\_nodes) равным 10 и запросим номер фактора обучения (learn\_speed). Коэффициент обучения преобразуется в число с плавающей запятой с помощью функции float ():

out\_nodes = 10  
print(**'Input the training speed (0.5): '**)  
learn\_speed = float(input())

Количество входных и выходных сигналов, количество скрытых нейронов и коэффициент обучения будут выходом нашей функции:

return input\_nodes, hidden\_nodes, out\_nodes, learn\_speed

Если мы запустим нашу функцию и установим количество скрытых нейронов равным 100 и значение коэффициента обучения равным 0,5, то в консоли мы увидим следующее:

>>>init\_net()

Введите количество скрытых нейронов:

>?100

Введите скорость тренировки (0.5):

>?0.5

(784, 100, 10, 0.5)

**3. Создать функцию, которая устанавливает начальные значения весов нейронной сети.** Начать обучение нейронной сети необходимо с некоторых начальных значений весовых коэффициентов. Чаще всего информации о том, какими они должны быть, нет, поэтому зададим им случайные числа в диапазоне [-0,5; 0,5]. Для этого воспользуемся функцией uniform (low, high, size). Здесь low и high — это границы диапазона случайных чисел, а size - это размерность массива этих случайных чисел:

def create\_net(input\_nodes, hidden\_nodes, out\_nodes):  
  
 w\_in2hidden = np.random.uniform(-0.5,0.5,(hidden\_nodes,input\_nodes))  
 w\_hidden2out = np.random.uniform(-0.5,0.5,(out\_nodes,hidden\_nodes))  
 return w\_in2hidden, w\_hidden2out

Функция получает количество входов (input\_nodes), скрытых слоев (hidden\_nodes) и выходов (out\_nodes). Функция возвращает две весовые матрицы: от входного до скрытого слоя (w\_in2hidden) и от скрытого до выходного слоя (w\_hidden2out).

**4. Создать функцию, которая вычисляет вывод нейронной сети.** Мы можем рассчитать выходы нейронной сети по следующей формуле:



Таким образом, функция должна попасть в матрицу весов нейронной сети (w\_in2hidden, w\_hidden2out) и входной сигнал (input\_signal):

def net\_output(w\_in2hidden, w\_hidden2out, input\_signal, return\_hidden):

В функцию добавлена еще одна переменная (return\_hidden). Если она не равна 0, то функция будет возвращать сигналы не только с последнего слоя, но и со всех скрытых. Это будет продемонстрировано в конце этой части практической работы.

Сначала массив входных сигналов необходимо преобразовать в вектор-столбец:

inputs = np.array(input\_signal, ndmin=2).T

Здесь параметр ndmin означает, что наш вектор-столбец представляет собой двумерную матрицу с одним столбцом.

Теперь необходимо рассчитать :

hidden\_in = np.dot(w\_in2hidden, inputs)

Для расчета выходов скрытого слоя необходимо вычислить активационную функцию суммы :

hidden\_out = f\_act(hidden\_in)

Затем те же процедуры необходимо повторить для выходного слоя. :

final\_in = np.dot(w\_hidden2out, hidden\_out)  
final\_out = f\_act(final\_in)

И, конечно же, результатом вычисления функции будет вектор-столбец выходных сигналов final\_out. При желании, если флаг return\_hidden установлен в 1, мы также будем выводить сигналы скрытого слоя hidden\_out:

if return\_hidden == 0:  
 return final\_out  
else:  
 return final\_out, hidden\_out

Вся функция выглядит так:

def net\_output(w\_in2hidden, w\_hidden2out, input\_signal, return\_hidden):  
 input = np.array(input\_signal, ndmin=2).T  
  
 hidden\_in = np.dot(w\_in2hidden, inputs)  
 hidden\_out = f\_act(hidden\_in)  
 final\_in = np.dot(w\_hidden2out, hidden\_out)  
 final\_out = f\_act(final\_in)  
  
 if return\_hidden == 0:  
 return final\_out  
 else:  
 return final\_out, hidden\_out

**5. Создать функцию для обучения нейронной сети.** Определены функции, позволяющие вычислить нейронную сеть. Перейдем к обучению нейронной сети:

def net\_train(target\_list,input\_signal, w\_in2hidden, w\_hidden2out, learn\_speed):

Входными параметрами функции являются: список желаемых сетевых выходов (target\_list); соответствующие им входные сигналы (input\_signal); матрицы весов нейронной сети (w\_in2hidden, w\_hidden2out) и коэффициент обучения (learn\_speed)

Нейронная сеть обучается по следующим формулам:





Во-первых, мы должны преобразовать targets и inputs в векторы-столбцы:

targets = np.array(target\_list, ndmin=2).T  
inputs = np.array(input\_signal, ndmin=2).T

и рассчитать выходы нейронной сети:

final\_out, hidden\_out = net\_output(w\_in2hidden, w\_hidden2out, input\_signal, 1)

Затем мы должны вычислить ошибки выходного слоя (out\_errors) по формуле :

out\_errors = targets - final\_out

и ошибки скрытого слоя (hidden\_errors) по формуле :

hidden\_errors = np.dot(w\_hidden2out.T, out\_errors)

Веса изменяются с использованием метода градиентного спуска, как указано выше.:

w\_hidden2out += learn\_speed \* np.dot((out\_errors \* final\_out \* (1 - final\_out)), hidden\_out.T)  
w\_in2hidden += learn\_speed \* np.dot((hidden\_errors \* hidden\_out \* (1 - hidden\_out)),inputs.T)

Функция возвращает новое значение весов (w\_in2hidden, w\_hidden2out):

return w\_in2hidden, w\_hidden2out

Вся функция выглядит так:

def net\_train(target\_list,input\_signal, w\_in2hidden, w\_hidden2out, learn\_speed):  
  
 targets = np.array(target\_list, ndmin=2).T  
 inputs = np.array(input\_signal, ndmin=2).T  
 final\_out, hidden\_out = net\_output(w\_in2hidden, w\_hidden2out, input\_signal, 1)  
  
 out\_errors = targets - final\_out  
 hidden\_errors = np.dot(w\_hidden2out.T, out\_errors)  
  
 w\_hidden2out += learn\_speed \* np.dot((out\_errors \* final\_out \* (1 - final\_out)), hidden\_out.T)  
 w\_in2hidden += learn\_speed \* np.dot((hidden\_errors \* hidden\_out \* (1 - hidden\_out)),inputs.T)

return w\_in2hidden, w\_hidden2out

**6.** **Написать функцию для обучения сети на реальных данных.** Уже разработаны функции для расчета нейронной сети и ее обучения. Теперь необходимо обучить нейронную сеть, используя данные, хранящиеся в файле mnist\_train.csv. Эта функция получает данные о весах нейронной сети и факторе обучения:

def train\_set(w\_in2hidden, w\_hidden2out, learn\_speed):

Для начала необходимо загрузить данные из файла:

data\_file = open(**"D:\PyThon\mnist\_train.csv"**, **'r'**)  
 training\_list = data\_file.readlines()  
 data\_file.close()

Функция open («filename», ’r’) открывает файл «filename» для чтения в переменную data\_file. Затем функция readlines () считывает все строки из data\_file в переменную training\_list. Наконец, файл, открытый по ссылке в data\_file, должен быть закрыт функцией close ().

Каждая строка содержит массив размером 28x28 пикселей с изображением цифр. Значения цвета пикселей разделяются запятыми, поэтому давайте разделим строку в каждой строке training\_list с помощью функции split () на элементы, ориентируясь на разделитель «,»:

for record in training\_list:  
 all\_values = record.split(**','**)

Теперь 784 пикселя изображения хранятся в переменной all\_values. Их нужно преобразовать из диапазона [0; 255] в диапазон входов нейронной сети [0.001; 1.00]. Почему слева нет 0? Нулевое значение может привести к тому, что некоторые веса не будут обновляться во время тренировки. Итак, давайте разделим все числа на 255 и умножим на 0,999, а затем прибавим 0,001:

inputs = (np.asfarray(all\_values[1:]) / 255.0 \* 0.999) + 0.001

Функция asfarray () преобразует числа, начиная с индекса [1], в действительные числа (элемент с индексом [0] — это номер распознаваемой цифры, поэтому нет необходимости преобразовывать его).

Пусть значения всех целей вывода сети равны небольшому числу (например, 0,001), за исключением целевого значения вывода, соответствующего цифре, которую сеть в настоящее время изучает. Пусть этот вывод будет 1:

targets = np.zeros(10) + 0.001  
targets[int(all\_values[0])] = 1.0

Функция zeros () создает массив указанной размерности, заполненный нулями.

Данные подготовлены, теперь сеть можно обучить функцией, написанной в предыдущей части лабораторной работы:

w\_in2hidden, w\_hidden2out = net\_train(targets, inputs, w\_in2hidden, w\_hidden2out, learn\_speed)

Функция возвращает новое значение весов (w\_in2hidden, w\_hidden2out):

return w\_in2hidden, w\_hidden2out

Вся функция выглядит так:

def train\_set(w\_in2hidden, w\_hidden2out, learn\_speed):  
 data\_file = open(**"D:\PyThon\mnist\_train.csv"**, **'r'**)  
 training\_list = data\_file.readlines()  
 data\_file.close()  
  
 for record in training\_list:  
 all\_values = record.split(**','**)  
  
 inputs = (np.asfarray(all\_values[1:]) / 255.0 \* 0.999) + 0.001  
  
 targets = np.zeros(10) + 0.001  
 targets[int(all\_values[0])] = 1.0  
 w\_in2hidden, w\_hidden2out = net\_train(targets, inputs, w\_in2hidden, w\_hidden2out, learn\_speed)

return w\_in2hidden, w\_hidden2out

**7.** **Написать функцию проверки сети.** Эффективность обученной нейронной сети проверяется на файле проверочных данных mnist\_test.csv. Данные для проверки должны отличаться от данных для обучения, чтобы оценка эффективности нейронной сети была адекватной.

Эта функция получает данные о весах нейронной сети:

def test\_set(w\_in2hidden, w\_hidden2out):

Для начала необходимо загрузить данные из файла:

data\_file = open(**"D:\PyThon\mnist\_test.csv"**, **'r'**)  
test\_list = data\_file.readlines()  
data\_file.close()

Введем дополнительную переменную test, в которую мы будем записывать правильные и неверные результаты работы нейронной сети:

test = []

Дадее создадим цикл для проверки данных для всего тестового образца test\_list, загруженного из файла:

for record in test\_list:

Подготовим данные, как в предыдущем абзаце:

all\_values = record.split(**','**)  
 inputs = (np.asfarray(all\_values[1:]) / 255.0 \* 0.999) + 0.001

и вычислим вывод нейронной сети:

out\_session = net\_output(w\_in2hidden, w\_hidden2out, inputs, 0)

Сравним тестовые данные с результатами нейронной сети:

if int(all\_values[0]) == np.argmax(out\_session):  
 test.append(1)  
else:  
 test.append(0)

Функция int () преобразует цифровые данные, хранящиеся в файле, в целое число. Функция argmax () возвращает выходной индекс нейронной сети с максимальным значением, которое соответствует выбору цифры нейронной сети. Добавим 1 в тестовый массив с помощью функции append (), если результат правильный, и 0, если ответ неправильный.

Давайте преобразуем тестовый массив в массив библиотеки numpy, чтобы использовать на нем функции:

test = np.asarray(test)

Рассчитайте эффективность сети как отношение суммы правильных результатов к общему количеству тестов. Сумма рассчитывается функцией sum (). Размер тестового массива записывается в его параметре size.

print(**'Net efficiency % ='**, (test.sum() / test.size) \* 100)

Вся тестовая функция выглядит так:

def test\_set(w\_in2hidden, w\_hidden2out):  
 data\_file = open(**"D:\PyThon\mnist\_test.csv"**, **'r'**)  
 test\_list = data\_file.readlines()  
 data\_file.close()  
 test = []  
 for record in test\_list:  
 all\_values = record.split(**','**)  
 inputs = (np.asfarray(all\_values[1:]) / 255.0 \* 0.999) + 0.001  
 out\_session = net\_output(w\_in2hidden, w\_hidden2out, inputs, 0)  
 if int(all\_values[0]) == np.argmax(out\_session):  
 test.append(1)  
 else:  
 test.append(0)  
 test = np.asarray(test)  
 print(**'Net efficiency % ='**, (test.sum() / test.size) \* 100)

**8. Написать функцию, которая отображает изображения чисел из набора данных.** Напишем простую функцию, которая отображает изображение цифры из данных:

def plot\_image(pixels: np.array):  
 plt.imshow(pixels.reshape((28, 28)), cmap=**'gray'**)  
 plt.show()

Здесь функция imshow () библиотеки matplotlib выводит квадратный массив точка за точкой на виртуальный экран. Параметр «cmap = 'grey'» означает, что картинка будет черно-белой. Квадратный массив получается из строки с помощью функции reshape (). Функция show библиотеки matplotlib отображает результат на экране.

**9. Обучение сети и расчет ее эффективности.** Сделаем это с помощью наших функций:

input\_nodes, hidden\_nodes, out\_nodes, learn\_speed = init\_net()  
w\_in2hidden, w\_hidden2out = create\_net(input\_nodes, hidden\_nodes, out\_nodes)  
for i in range (5):  
 print(**'Test #'**, i+1)  
 w\_in2hidden, w\_hidden2out = train\_set(w\_in2hidden, w\_hidden2out, learn\_speed)  
 test\_set(w\_in2hidden, w\_hidden2out)

Мы провели 5 тренингов и видим, как меняется эффективность сети.

Теперь посмотрим случайное изображение из тестового файла и сравним его с результатами нейронной сети:

data\_file = open(**"D:\PyThon\mnist\_test.csv"**, **'r'**)  
test\_list = data\_file.readlines()  
data\_file.close()  
all\_values = test\_list[int(np.random.uniform(0,9999))].split(**','**)  
inputs = (np.asfarray(all\_values[1:]) / 255.0 \* 0.999) + 0.001  
out\_session = net\_output(w\_in2hidden, w\_hidden2out, inputs, 0)  
print(np.argmax(out\_session))  
plot\_image(np.asfarray(all\_values[1:]))