**Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

**Лабораторная работа №4**

Введение в искусственные нейронные сети.

Выполнил:

Студент группы КБ-211

Коренев Д.Н.

Принял:

Твердохлеб В.В

Оглавление

[Задание 3](#_Toc179595081)

[Код алгоритма **Ошибка! Закладка не определена.**](#_Toc179595082)

[Интерфейс приложения **Ошибка! Закладка не определена.**](#_Toc179595083)

[Результаты исследования влияния параметров алгоритма **Ошибка! Закладка не определена.**](#_Toc179595084)

[Вывод **Ошибка! Закладка не определена.**](#_Toc179595085)

[Приложения **Ошибка! Закладка не определена.**](#_Toc179595086)

*Цель работы:* Разработка и исследование алгоритмов обучения искусственных нейронных сетей.

Задание

Разработать программу «Распознавание образов», отвечающую следующим требованиям.

А) Распознавание образов должно выполняться искусственной нейронной сетью, обучаемой по алгоритму с обратным распространением ошибки. Общие требования к сети и программе:

* связи между нейронами – прямые;
* количество скрытых слоев – 1;
* количество нейронов выходного слоя (классов образов) – не менее 4;
* количество обучающих образов – не менее 1 на каждый класс образов;
* функция активации – сигмоидальная .
* распознаваемые (тестовые) образы – формируются пользователем;
* норма обучения и количество эпох обучения – задаются пользователем;
* режим обучения или распознавания – задается пользователем.

Б) Индивидуальный вариант выбрать согласно таблице 1.

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание



**Выполнение**

Листинг программы

Python 3.12

import tkinter as tk

from tkinter import messagebox, Canvas, ttk

import numpy as np

from keras.models import Sequential

from keras.layers import Dense

from keras.optimizers import SGD

import threading

import os

input\_size = 100  # Размер изображения

output\_size = 6  # Количество классов

hidden\_size = 7  # Количество нейронов в скрытом слое

training\_data = []  # Список для хранения обучающих образов

training\_labels = []  # Список для хранения меток классов

model = None  # Переменная для хранения модели

epochs = 1000  # Количество эпох

learning\_rate = 0.1  # Норма обучения

def create\_model(learning\_rate):

    # Создание модели

    global model

    model = Sequential()

    model.add(Dense(hidden\_size, input\_dim=input\_size, activation='sigmoid'))

    model.add(Dense(output\_size, activation='softmax'))

    optimizer = SGD(learning\_rate=learning\_rate)

    model.compile(optimizer=optimizer,

                  loss='categorical\_crossentropy', metrics=['accuracy'])

    return model

def train\_model():

    # Функция для обучения нейронной сети

    if not training\_data or not training\_labels:

        messagebox.showerror("Ошибка", "Недостаточно обучающих данных!")

        return

    model = create\_model(float(learning\_rate\_entry.get()))

    X\_train = np.array(training\_data)

    y\_train = np.array(training\_labels)

    progress\_var = tk.IntVar()

    progress\_bar = ttk.Progressbar(root, maximum=int(

        epochs\_entry.get()), variable=progress\_var)

    progress\_bar.grid(row=8, column=1, columnspan=2, sticky="ew")

    def training\_thread():

        for epoch in range(int(epochs\_entry.get())):

            if training\_mode.get() == "batch":

                model.fit(X\_train, y\_train, epochs=1, verbose=0, batch\_size=32)

            else:

                model.fit(X\_train, y\_train, epochs=1, verbose=0)

            progress\_var.set(epoch + 1)

            root.update\_idletasks()

        messagebox.showinfo("Обучение завершено",

                            "Нейронная сеть успешно обучена!")

        progress\_bar.grid\_forget()

    threading.Thread(target=training\_thread).start()

def add\_training\_data(image, label):

    # Функция для добавления образов для обучения

    training\_data.append(image)

    training\_labels.append(label)

    messagebox.showinfo("Добавлено", f"Образ добавлен в класс {label}")

def recognize():

    # Функция для распознавания образа

    if not model:

        messagebox.showerror("Ошибка", "Модель не обучена!")

        return

    image = np.array(user\_image).flatten()

    image = np.expand\_dims(image, axis=0)

    prediction = model.predict(image)

    class\_index = np.argmax(prediction)

    recognized\_class = classes[class\_index]

    result\_label.config(text=f"Распознан {recognized\_class}")

    # Обновление таблицы вероятностей

    for i, prob in enumerate(prediction[0]):

        probability\_table.item(probability\_table.get\_children()[

                               i], values=(classes[i], f"{prob:.5f}"))

def save\_model():

    # Функция для сохранения обученной модели

    if not model:

        messagebox.showerror("Ошибка", "Модель не обучена!")

        return

    model.save('trained\_model.h5')

    messagebox.showinfo("Сохранено", "Модель успешно сохранена!")

def load\_model():

    # Функция для загрузки модели

    global model

    if os.path.exists('trained\_model.h5'):

        from keras.models import load\_model

        model = load\_model('trained\_model.h5')

        messagebox.showinfo("Загружено", "Модель успешно загружена!")

    else:

        messagebox.showerror("Ошибка", "Файл модели не найден!")

# Создание окна программы

root = tk.Tk()

root.title("Распознавание геометрических фигур")

root.geometry("870x950")

root.configure(bg='#f0f0f0')

classes = ['Квадрат', 'Круг', 'Ромб',

           'Треугольник', 'Трапеция', 'Прямоугольник']

# Ввод нормы обучения и количества эпох

tk.Label(root, text="Норма обучения", bg='#f0f0f0', font=('Arial', 12)).grid(

    row=0, column=0, pady=10, padx=10, sticky='w')

learning\_rate\_entry = tk.Entry(root, font=('Arial', 12))

learning\_rate\_entry.grid(row=0, column=1, pady=10, padx=10, sticky='e')

learning\_rate\_entry.insert(0, "0.1")

tk.Label(root, text="Количество эпох", bg='#f0f0f0', font=('Arial', 12)).grid(

    row=1, column=0, pady=10, padx=10, sticky='w')

epochs\_entry = tk.Entry(root, font=('Arial', 12))

epochs\_entry.grid(row=1, column=1, pady=10, padx=10, sticky='e')

epochs\_entry.insert(0, "1000")

# Выбор режима обучения

training\_mode = tk.StringVar(value="sequential")

tk.Label(

    root,

    text="Режим обучения",

    bg='#f0f0f0',

    font=('Arial', 12)

).grid(row=2, column=0, pady=10, padx=10, sticky='w')

tk.Radiobutton(

    root,

    text="Последовательный",

    variable=training\_mode,

    value="sequential",

    bg='#f0f0f0',

    font=('Arial', 12)

).grid(row=2, column=1, pady=10, padx=10, sticky='w')

tk.Radiobutton(

    root,

    text="Пакетный",

    variable=training\_mode,

    value="batch",

    bg='#f0f0f0',

    font=('Arial', 12)

).grid(row=2, column=1, pady=10, padx=10, sticky='e')

canvas\_size = 400

grid\_size = 10

cell\_size = canvas\_size // grid\_size

user\_image = np.zeros((grid\_size, grid\_size))

def paint(event):

    x, y = event.x // cell\_size, event.y // cell\_size

    if x < grid\_size and y < grid\_size:

        user\_image[y, x] = 1

        canvas.create\_rectangle(

            x \* cell\_size, y \* cell\_size,

            (x + 1) \* cell\_size, (y + 1) \* cell\_size,

            fill='black'

        )

def clear\_canvas():

    global user\_image

    user\_image = np.zeros((grid\_size, grid\_size))

    canvas.delete('all')

    draw\_grid()

def draw\_grid():

    for i in range(grid\_size):

        for j in range(grid\_size):

            canvas.create\_rectangle(

                i \* cell\_size, j \* cell\_size,

                (i + 1) \* cell\_size, (j + 1) \* cell\_size,

                outline='gray'

            )

canvas = Canvas(

    root, width=canvas\_size, height=canvas\_size,

    bg='white', bd=2, relief='solid'

)

canvas.grid(row=3, column=0, pady=10, padx=10, rowspan=5)

canvas.bind("<B1-Motion>", paint)

draw\_grid()

# Метки классов и кнопки для добавления образов

for i, figure\_class in enumerate(classes):

    tk.Button(

        root, text=f"Добавить {figure\_class}",

        command=lambda i=i: add\_training\_data(

            user\_image.flatten(), np.eye(output\_size)[i]

        ),

        font=('Arial', 12), bg='#2196F3', fg='white'

    ).grid(row=9 + i, column=0, pady=5, padx=10, sticky='ew')

# Результат распознавания

result\_label = tk.Label(root, text="Распознан",

                        bg='#f0f0f0', font=('Arial', 12))

result\_label.grid(row=3, column=1, pady=10, padx=10)

# Таблица вероятностей

probability\_table = ttk.Treeview(root, columns=(

    "Класс", "Вероятность"), show='headings')

probability\_table.heading("Класс", text="Класс")

probability\_table.heading("Вероятность", text="Вероятность")

probability\_table.grid(row=3, column=1, pady=10, padx=10, rowspan=5)

for figure\_class in classes:

    probability\_table.insert("", "end", values=(figure\_class, "0.00000"))

def center\_text(treeview):

    for col in treeview['columns']:

        treeview.column(col, anchor='center')

        treeview.heading(col, anchor='center')

center\_text(probability\_table)

train\_button = tk.Button(root, text="Обучить сеть", command=train\_model, font=(

    'Arial', 12), bg='#A2C2E8', fg='black')

train\_button.grid(row=8, column=0, pady=10, padx=10)

recognize\_button = tk.Button(root, text="Распознать", command=recognize, font=(

    'Arial', 12), bg='#A2C2E8', fg='black')

recognize\_button.grid(row=8, column=0, pady=10, padx=10, sticky='e')

clear\_button = tk.Button(root, text="Очистить", command=clear\_canvas, font=(

    'Arial', 12), bg='#F44336', fg='white')

clear\_button.grid(row=8, column=0, pady=10, padx=10, sticky='w')

save\_button = tk.Button(

    root, text="Сохранить модель", command=save\_model,

    font=('Arial', 12), bg='#A2C2E8', fg='black'

)

save\_button.grid(row=10, column=1, pady=10, padx=10, sticky='ew')

load\_button = tk.Button(

    root, text="Загрузить модель", command=load\_model,

    font=('Arial', 12), bg='#A2C2E8', fg='black'

)

load\_button.grid(row=11, column=1, pady=10, padx=10, sticky='ew')

root.mainloop()

Результаты

Изображение выглядит как текст, снимок экрана, программное обеспечение, Параллельный

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Параллельный

Автоматически созданное описание

Реализованы следующие режимы:

1)Последовательный режим - веса модели обновляются после каждого примера обучающих данных. То есть, модель проходит через каждую обучающую выборку по одному, и после обработки каждого примера выполняется шаг обратного распространения ошибки для обновления весов.

* Преимущества:
  + Может приводить к более точному обучению, особенно на небольших наборах данных, поскольку модель обновляет свои параметры более часто.
  + Хорошо подходит для данных, которые могут быть обработаны в потоковом режиме или в реальном времени.
* Недостатки:
  + Может быть менее эффективным с точки зрения времени на обучение, особенно при использовании больших наборов данных, так как каждое обновление весов происходит слишком часто.
  + Более высокая вероятность колебаний в процессе обучения из-за шумных данных, поскольку каждое обновление основано только на одном примере.

2. Пакетный режим - обучающая выборка делится на небольшие группы (пакеты), и веса модели обновляются после обработки каждого пакета. Преимущества:

* + Более эффективное использование ресурсов (например, память и процессорное время), так как операция обновления происходит реже, и можно использовать векторизованные операции для обработки пакетов данных.
  + Обычно ведет к более стабильному и плавному процессу обучения, так как обновления основаны на среднем значении ошибки по пакету.
* Недостатки:
  + Требует больше памяти, поскольку необходимо хранить все примеры в пакете одновременно.
  + Может привести к ухудшению обучения, если размер пакета слишком мал или слишком велик. Слишком маленькие пакеты могут привести к шумным обновлениям, в то время как слишком большие могут затруднить нахождение локальных минимумов.

Таким образом, выбор между последовательным и пакетным режимами обучения зависит от конкретной задачи, объема данных и доступных вычислительных ресурсов. В большинстве случаев пакетный режим является более распространенным и эффективным способом обучения нейронных сетей, особенно на больших наборах данных. Однако последовательный режим может быть предпочтительным в случаях, когда данные поступают в режиме реального времени или когда набор данных небольшой.

**Вывод:** в результате выполнения лабораторной работы был разработан и исследован алгоритм обучения искусственных нейронных сетей.