

**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 |
| Листинг программы | 4 |
| Результаты..... | 8 |

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.

| № варианта | Тип образа | Кол-во нейронов скрытого слоя |
|------------|-----------------------|----------------------------------|
| 17 | Геометрические фигуры | 7 |

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

Норма обучения: 0.1

Количество эпох: 1000

Режим обучения: ☒ Последовательный ☐ Пакетный

Распознан Квадрат

| Класс | Вероятность |
|---------------|-------------|
| Квадрат | 0.90483 |
| Круг | 0.03454 |
| Ромб | 0.01340 |
| Треугольник | 0.01331 |
| Трапеция | 0.03079 |
| Прямоугольник | 0.00312 |

Очистить Обучить сеть Распознать

Добавить Квадрат

Добавить Круг

Добавить Ромб

Добавить Треугольник

Добавить Трапеция

Добавить Прямоугольник

Сохранить модель

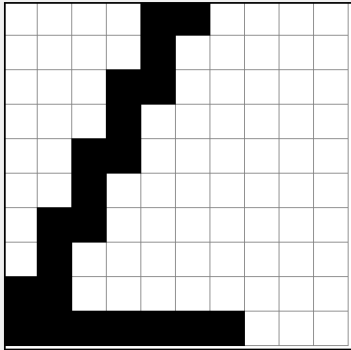
Загрузить модель

Распознавание геометрических фигур

Норма обучения: 0.1

Количество эпох: 1000

Режим обучения: ☒ Последовательный ☐ Пакетный



Распознан Треугольник

| Класс | Вероятность |
|---------------|-------------|
| Квадрат | 0.02453 |
| Круг | 0.01428 |
| Ромб | 0.16197 |
| Треугольник | 0.70334 |
| Трапеция | 0.02508 |
| Прямоугольник | 0.07080 |

Очистить Обучить сеть Распознать

Добавить Квадрат

Добавить Круг

Добавить Ромб

Добавить Треугольник

Добавить Трапеция

Добавить Прямоугольник

Сохранить модель

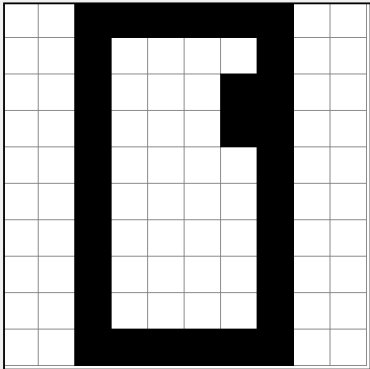
Загрузить модель

Распознавание геометрических фигур

Норма обучения: 0.1

Количество эпох: 1000

Режим обучения: ☐ Последовательный ☒ Пакетный



Распознан Прямоугольник

| Класс | Вероятность |
|---------------|-------------|
| Квадрат | 0.00085 |
| Круг | 0.00251 |
| Ромб | 0.00952 |
| Треугольник | 0.00439 |
| Трапеция | 0.02758 |
| Прямоугольник | 0.95514 |

Очистить Обучить сеть Распознать

Добавить Квадрат

Добавить Круг

Добавить Ромб

Добавить Треугольник

Добавить Трапеция

Добавить Прямоугольник

Сохранить модель

Загрузить модель

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:

- Требуется 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.