Федеральное Государственное бюджетное учреждение

высшего профессионального образования

"Заполярный Государственный Университет имени Н. М. Федоровского"

Кафедра «Информационные системы и технологии»

Нейронные сети

Отчет по лабораторной работе №1

Тема: Персептрон.

Выполнил:

студент группы ИС-21

Сидельников Максим Эдуардович

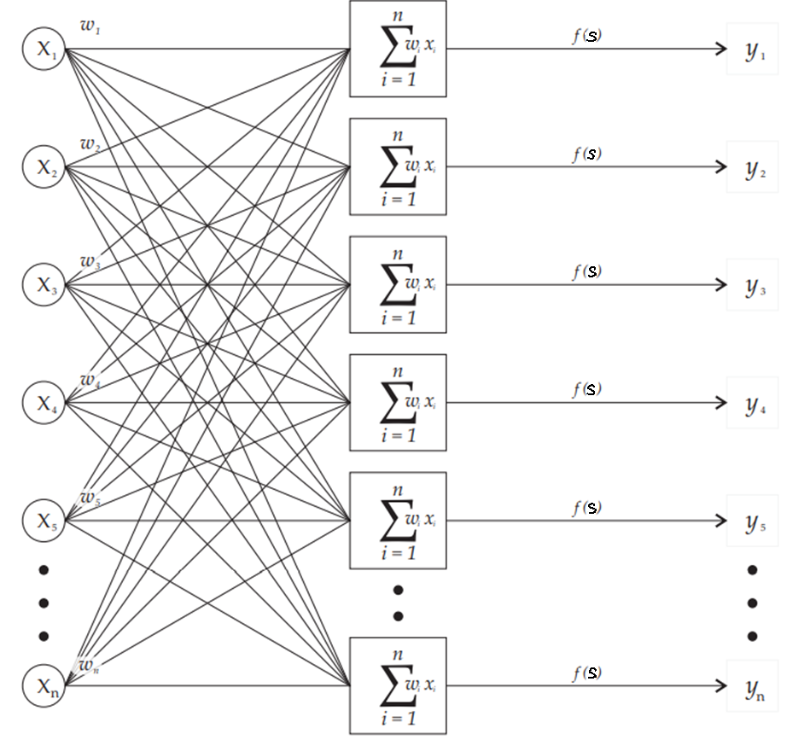
Проверила:

Дыптан Елена Арнольдовна

Норильск 2024

**Цель работы:** ознакомиться с основными понятиями нейронных сетей, принципами их работы, построением архитектуры, а также получить практические навыки в создании, настройке и обучении персептрона.

**Архитектура сети**



где **, ,**

**,**

N = 10 (0,1,2,3,4,5,6,7,8,9)

*, где порог функции*

В ходе выполнения лабораторной работы была разработана однослойная нейронная сеть. В качестве активационной функции использовалась пороговая функция с порогом 0. Весовые коэффициенты случайны (в диапазоне [-0.3;0.3]. Сеть имеет 10 нейронов, по каждому на одну арабскую цифру. Входной вектор – изображение 100x100 пикселей.

**Алгоритм обучения сети**

1. Инициализация весовых коэффициентов в диапазоне [-0.3;0.3];
2. Преобразование фотографии .bmp в массив данных;
3. Обучение нейронов. Повторяется, пока ошибки не перестанут появляться.  
   Обучение:
   1. В случайном порядке на вход подаётся образ;
   2. Задаётся счётчик ошибки, образ отправляется на проверку нейронам. В случае, если ошибок нет – обучение завершено (см. d), иначе продолжить;
   3. Отправить образ в функцию «обновления весов», имеющую вид: .
   4. Сохранить веса, перейти к режиму работы/тестирования.

Count = 0  
flag = True

Flag != False

Flag = false

I = 1, k

Перевести .bmp в вектор  
yk – ожидаемое значение

S = 0

J = 1, n

S < 0

yj = 0

yj = 1

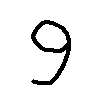
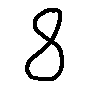
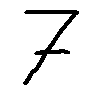
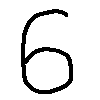
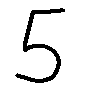
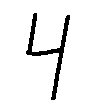
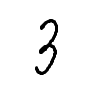
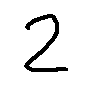
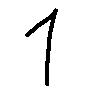
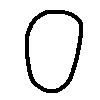
Корректировка весовых коэффициентов

m = d, Z

S += w[m] \* a[m]

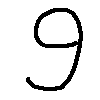
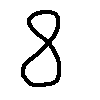
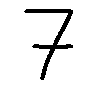
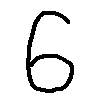
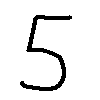
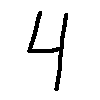
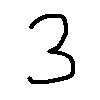
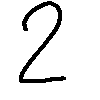
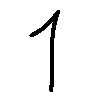
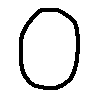
Count += 1

**Обучающая выборка**

****

Размер обучающей выборки:

**Тестовая выборка**



Размер тестовой выборки:

**Тестирование**

|  |  |  |
| --- | --- | --- |
|  | Количество эпох | Ошибки теста |
| 0,001 | 121 | 24 |
| 0,01 | 77 | 8 |
| 0,1 | 57 | 9 |
| 0,2 | 55 | 5 |
| 0,3 | 62 | 11 |
| 0,4 | 68 | 8 |
| 0,5 | 74 | 15 |
| 0,6 | 67 | 11 |
| 0,7 | 64 | 11 |
| 0,8 | 61 | 15 |
| 0,9 | 62 | 20 |
| 1 | 57 | 14 |

**Вывод:** исходя из результатов тестирования, можно сделать вывод, чтопоказательскорости обучения напрямую влияет на эффективность нейронной сети. Также на эффективность может повлиять последовательность посылаемых для обучения образов (чем сильнее они перемешаны, тем лучше). Наилучший результат наблюдается у сети с α = 0,2.

**Листинг**

import tkinter as tk

from tkinter import Canvas, messagebox

from PIL import Image, ImageDraw

import numpy as np

import matplotlib.pyplot as plt

import os

class NumberProcessor():

    def \_\_init\_\_(self, folder, learning\_rate, num\_classes=10, input\_size=10000):

        self.folder = folder

        self.learning\_rate = learning\_rate

        self.input\_size = input\_size

        self.num\_classes = num\_classes

        self.weights\_path = f"weights-{self.learning\_rate}.npy"

        if os.path.exists(self.weights\_path):

            self.weights = np.load(self.weights\_path)

            print("Веса успешно загружены")

        else:

            self.weights = np.random.uniform(-0.3, 0.3, (self.num\_classes, self.input\_size))

            print("Веса не найдены, используются случайно сгенерированные веса")

        self.images, self.lables = self.load\_images(folder=self.folder)

        self.train(self.images, self.lables)

    @staticmethod

    def get\_object\_bounds(image):

        image\_array = np.array(image)

        non\_empty\_columns = np.where(image\_array.min(axis=0) < 255)[0]

        non\_empty\_rows = np.where(image\_array.min(axis=1) < 255)[0]

        if non\_empty\_columns.any() and non\_empty\_rows.any():

            upper, lower = non\_empty\_rows[0], non\_empty\_rows[-1]

            left, right = non\_empty\_columns[0], non\_empty\_columns[-1]

            return left, upper, right, lower

        else:

            return None

    @staticmethod

    def center\_object(image):

        bounds = NumberProcessor.get\_object\_bounds(image)

        if bounds:

            left, upper, right, lower = bounds

            object\_width = right - left

            object\_height = lower - upper

            horizontal\_padding = (image.width - object\_width) // 2

            vertical\_padding = (image.height - object\_height) // 2

            cropped\_image = image.crop(bounds)

            centered\_image = Image.new("L", (image.width, image.height), "white")

            centered\_image.paste(cropped\_image, (horizontal\_padding, vertical\_padding))

            return centered\_image

        return image

    def load\_images(self, folder=None):

        if folder is None:

            folder = self.folder

        images = []

        labels = []

        for filename in os.listdir(folder):

            img\_path = os.path.join(folder, filename)

            try:

                with Image.open(img\_path) as img:

                    img = img.convert("L")

                    img = self.center\_object(img)

                    img = img.resize((100, 100))

                    images.append(np.asarray(img).flatten() / 255.0)

                    label = int(filename[0])

                    labels.append(label)

            except Exception as e:

                print(f"Ошибка загрузки {img\_path}: {e}")

        return np.array(images), np.array(labels)

    def guess(self, image):

        sum = [np.dot(w, image) for w in self.weights]

        output = [1 if s > 0 else 0 for s in sum]

        if np.array(output).sum() == 1:

            return output, output.index(1)

        else:

            return output, None

    def train(self, images, labels):

        if self.weights is None:

            self.weights = np.zeros((len(set(labels)), images.shape[1]))

        epoch = 0

        flag = True

        while flag:

            indices = np.random.permutation(len(images))

            images\_shuffled = images[indices]

            labels\_shuffled = labels[indices]

            flag = False

            for img, label in zip(images\_shuffled, labels\_shuffled):

                predictions, predicted\_label = self.guess(img)

                if predicted\_label != label:

                    flag = True

                    self.weights[label] += self.learning\_rate \* img

                    for i in range(len(self.weights)):

                        if predictions[i] == 1 and i != label:

                            self.weights[i] -= self.learning\_rate \* img

            epoch += 1

            print(f"Эпоха {epoch}")

        print(f"Обучение завершено после {epoch} эпох")

class TestCanvas(tk.Tk):

    def \_\_init\_\_(self, processor):

        super().\_\_init\_\_()

        self.processor = processor

        self.canvas = Canvas(self, width=280, height=280, bg="white")

        self.canvas.pack()

        self.bind("<B1-Motion>", self.draw)

        self.image = Image.new("L", (280, 280), "white")

        self.picture\_draw = ImageDraw.Draw(self.image)

        test\_button = tk.Button(self, text="Распознание числа", command=self.test\_image)

        test\_button.pack(side=tk.BOTTOM)

        clear\_button = tk.Button(self, text="Очистить", command=self.clear\_canvas)

        clear\_button.pack(side=tk.BOTTOM)

        save\_weights = tk.Button(self, text="Сохранить весовые коэффициенты", command=self.save\_w)

        save\_weights.pack(side=tk.BOTTOM)

        research = tk.Button(self, text="Тестирование", command=self.research)

        research.pack(side=tk.BOTTOM)

        graf = tk.Button(self, text="Визуализация", command=self.graf)

        graf.pack(side=tk.BOTTOM)

    def draw(self, event):

        x, y = event.x, event.y

        self.canvas.create\_oval(x - 7, y - 7, x + 7, y + 7, fill="black")

        self.picture\_draw.ellipse([x - 7, y - 7, x + 7, y + 7], fill="black")

    def test\_image(self):

        centered\_image = self.processor.center\_object(self.image)

        resized\_image = centered\_image.resize((100, 100))

        img\_array = np.array(resized\_image) / 255.0

        img\_array = img\_array.flatten()

        result, flag = self.processor.guess(img\_array)

        if flag == None:

            messagebox.showinfo("Результат", "Страшно, очень страшно, мы не знаем, что это такое!")

        else:

            messagebox.showinfo("Результат", f"Это похоже на цифру {flag}!")

        self.clear\_canvas()

    def save\_w(self):

        np.save(f"weights-{self.processor.learning\_rate}.npy", self.processor.weights)

        messagebox.showinfo("Результат", "Весовые коэффициенты сохранены!")

    def clear\_canvas(self):

        self.canvas.delete("all")

        self.image = Image.new("L", (280, 280), "white")

        self.picture\_draw = ImageDraw.Draw(self.image)

    def research(self):

        test\_folder = "testing"

        re\_images, re\_labels = self.processor.load\_images(test\_folder)

        wrong\_count = 0

        for img, label in zip(re\_images, re\_labels):

            pred, flag = self.processor.guess(img)

            if flag is None or flag != label:

                wrong\_count += 1

        messagebox.showinfo(

            "Результат",

            f"Количество ошибок: {wrong\_count}, процент не распознанных образов: {(wrong\_count/len(re\_labels))\*100:.2f}%"

        )

    def graf(self):

        fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 8))

        for i, ax in enumerate(axes.flat):

            im = ax.imshow(self.processor.weights[i].reshape((100, 100)), cmap="Greys")

            ax.set\_title(f"Веса для класса {i}")

        plt.tight\_layout()

        plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    folder\_path = "dataset"

    processor = NumberProcessor(folder\_path, learning\_rate=0.2)

    app = TestCanvas(processor)

    app.title("Распознавание чисел")

    app.mainloop()