Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

Лабораторная работа №2

**Простые нейронные сети. Персептрон**

|  |
| --- |
| Выполнил: студент гр. 5.306М:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лаптев А. В.  Проверил: доц. каф. ВТиЭ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шайдуров А. А.  Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

**Цель:** спроектировать алгоритм и создать программу персептронной системы.

**Задачи:** основными задачами данной лабораторной работы являются:

**Ход работы:**

1. Проектирование персептронной нейронной сети для распознавания четных и нечетных чисел.

Алгоритм персептронной нейронной сети для распознавания четных и нечетных чисел

1. Загрузка данных: Программа начинает с загрузки датасета с числами с помощью функции `load\_digits` из scikit-learn.

2. Преобразование целевых меток: Целевые метки преобразуются так, чтобы отражать четность чисел: 0 для четных чисел и 1 для нечетных чисел.

3. Разделение на обучающую и тестовую выборки: Данные разделяются на обучающую и тестовую выборки с использованием функции `train\_test\_split` из scikit-learn.

4. Инициализация перцептрона: Создается перцептрон с размером входа, соответствующим размеру признаков в обучающем наборе данных.

5. Обучение перцептрона: Программа обучает перцептрон на обучающей выборке. Внутри этого процесса перцептрон корректирует веса и смещение в соответствии с правильностью предсказаний.

6. Проверка работы перцептрона на тестовой выборке: После обучения программа проверяет работу перцептрона на тестовой выборке. Она сравнивает предсказанные значения с фактическими значениями четности чисел и вычисляет точность предсказания.

7. Вывод результата: Наконец, программа выводит точность предсказания перцептрона на тестовой выборке.

Код

import numpy as np

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

class Perceptron:

    # Класс персептрона

    def \_\_init\_\_(self, input\_size):

        # Инициализация весов и смещения

        self.weights = np.random.rand(input\_size)

        self.bias = np.random.rand()

    def activate(self, x):

        # Пороговая функция активации

        return 1 if x > 0 else 0

    def forward(self, inputs):

        # Прямая передача данных со входа на выход персептрона

        weighted\_sum = np.dot(inputs, self.weights) + self.bias

        return self.activate(weighted\_sum)

    def train(self, inputs, target):

        # Обучение персептрона

        prediction = self.forward(inputs)

        if prediction == target:

            return False  # Если предсказание верное, возвращаем False

        elif prediction == 0:

            # Если предсказание неправильное и равно 0, добавляем все входы к соответствующим им весам

            self.weights += inputs

            self.bias += 1

        elif prediction == 1:

            # Если предсказание неправильное и равно 1, вычитаем каждый вход из соответствующего ему веса

            self.weights -= inputs

            self.bias -= 1

        return True  # Возвращаем True, чтобы указать, что было изменение весов

if \_\_name\_\_ == '\_\_main\_\_':

    # Загрузка датасета чисел

    digits = load\_digits()

    # Преобразование целевых меток чисел по принципу четность/нечетность

    y\_transformed = digits.target % 2

    # Разделение датасета на обучающую и тестовую выборки в соотношении 80:20

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, y\_transformed, test\_size=0.2, random\_state=42)

    # Создаем персептрон с размером входа, соответствующим размеру признаков в датасете

    perceptron = Perceptron(X\_train.shape[1])

    # Обучение перцептрона

    epochs = 1000

    for epoch in range(epochs):

        for i in range(len(X\_train)):

            if perceptron.train(X\_train[i], y\_train[i]):

                break  # Если было изменение весов, начинаем заново

    # Проверка работы персептрона на тестовой выборке

    correct = 0

    for i in range(len(X\_test)):

        prediction = perceptron.forward(X\_test[i])

        if prediction == y\_test[i]:

            correct += 1

    accuracy = correct / len(X\_test)

    print(f"Точность предсказания: {accuracy}")

Полученная при этом точность распознавания － 0.872.

Добавление дельта-правила

1. Загрузка данных: Программа начинает с загрузки датасета с числами с помощью функции `load\_digits` из scikit-learn.

2. Преобразование целевых меток: Целевые метки преобразуются так, чтобы отражать четность чисел: 0 для четных чисел и 1 для нечетных чисел.

3. Разделение на обучающую и тестовую выборки: Данные разделяются на обучающую и тестовую выборки с использованием функции `train\_test\_split` из scikit-learn.

4. Инициализация перцептрона: Создается перцептрон с размером входа, соответствующим размеру признаков в обучающем наборе данных, а также задается скорость обучения.

5. Обучение перцептрона с использованием дельта-правила:

- Для каждого обучающего примера перцептрон делает предсказание.

- Если предсказание нейрона не совпадает с целевым значением, то вычисляется изменение весов и обновляются веса и смещение перцептрона в соответствии с дельта-правилом.

6. Проверка работы перцептрона на тестовой выборке:

- После обучения программа проверяет работу перцептрона на тестовой выборке.

- Для каждого тестового примера перцептрон делает предсказание.

- Вычисляется точность предсказаний путем сравнения предсказанных значений с фактическими значениями четности чисел.

7. Вывод результата: Наконец, программа выводит точность предсказания перцептрона на тестовой выборке.

Код

import numpy as np

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

class Perceptron:

    '''Класс персептрона'''

    def \_\_init\_\_(self, input\_size, learning\_rate=0.1):

        # Инициализация весов

        self.weights = np.random.rand(input\_size)

        self.bias = np.random.rand()

        self.learning\_rate = learning\_rate

    def activate(self, x):

        # Пороговая функция активации

        return 1 if x > 0 else 0

    def forward(self, inputs):

        # Прямое распространение сигнала

        weighted\_sum = np.dot(inputs, self.weights) + self.bias

        return self.activate(weighted\_sum)

    def train(self, inputs, target):

        # Обучение перцептрона с использованием дельта-правила

        prediction = self.forward(inputs)

        if prediction != target:

            # Вычисление изменения весов

            delta\_weights = self.learning\_rate \* (target - prediction) \* inputs

            # Обновление весов

            self.weights += delta\_weights

            # Обновление смещения

            self.bias += self.learning\_rate \* (target - prediction)

            return True  # Возвращаем True, чтобы указать, что было изменение весов

        else:

            return False  # Если предсказание верное, возвращаем False

if \_\_name\_\_ == '\_\_main\_\_':

    # Загрузка датасета чисел

    digits = load\_digits()

    # Преобразование целевых меток для отражения четности чисел

    y\_transformed = digits.target % 2

    # Разделение датасета на обучающую и тестовую выборки с соотношением 80:20

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, y\_transformed, test\_size=0.2, random\_state=42)

    # Создаем персептрон с размером входа, соответствующим размеру признаков в датасете

    perceptron = Perceptron(X\_train.shape[1])

    # Обучение персептрона

    epochs = 1000

    for epoch in range(epochs):

        for i in range(len(X\_train)):

            perceptron.train(X\_train[i], y\_train[i])

    # Проверка работы персептрона на тестовой выборке

    correct = 0

    for i in range(len(X\_test)):

        prediction = perceptron.forward(X\_test[i])

        if prediction == y\_test[i]:

            correct += 1

    accuracy = correct / len(X\_test)

print(f"Точность предсказания: {accuracy}")

После добавления дельта-правила точность предсказания возросла и составляет 0.916.

1. Проектирование персептронной нейронной сети для распознавания чисел кратных и не кратных трем.

Данный функционал был добавлен к уже существующему, соответственно алгоритм обучения не был изменен.

Код

import numpy as np

from sklearn.datasets import load\_digits

from sklearn.model\_selection import train\_test\_split

class Perceptron:

    '''Класс персептрона'''

    def \_\_init\_\_(self, input\_size, learning\_rate=0.1):

        # Инициализация весов

        self.weights = np.random.rand(input\_size)

        self.bias = np.random.rand()

        self.learning\_rate = learning\_rate

    def activate(self, x):

        # Пороговая функция активации

        return 1 if x > 0 else 0

    def forward(self, inputs):

        # Прямое распространение сигнала

        weighted\_sum = np.dot(inputs, self.weights) + self.bias

        return self.activate(weighted\_sum)

    def train(self, inputs, target):

        # Обучение перцептрона с использованием дельта-правила

        prediction = self.forward(inputs)

        if prediction != target:

            # Вычисление изменения весов

            delta\_weights = self.learning\_rate \* (target - prediction) \* inputs

            # Обновление весов

            self.weights += delta\_weights

            # Обновление смещения

            self.bias += self.learning\_rate \* (target - prediction)

            return True  # Возвращаем True, чтобы указать, что было изменение весов

        else:

            return False  # Если предсказание верное, возвращаем False

if \_\_name\_\_ == '\_\_main\_\_':

    # Загрузка датасета чисел

    digits = load\_digits()

    mode  = 0

    while mode != 1 and mode != 2:

        print('Режимы распознавания:\n1. Четность/нечетность числа\n2. Кратность/не кратность числу 3')

        mode = int(input('Введите номер режима:\t'))

    if mode == 1:

        # Преобразование целевых меток для отражения четности чисел

        y\_transformed = digits.target % 2

    else:

        # Преобразование целевых меток для отражения четности чисел

        y\_transformed = (digits.target % 3 == 0).astype(int)

    # Разделение датасета на обучающую и тестовую выборки с соотношением 80:20

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data, y\_transformed, test\_size=0.2, random\_state=42)

    # Создаем персептрон с размером входа, соответствующим размеру признаков в датасете

    perceptron = Perceptron(X\_train.shape[1])

    # Обучение персептрона

    epochs = 1000

    for epoch in range(epochs):

        for i in range(len(X\_train)):

            perceptron.train(X\_train[i], y\_train[i])

    # Проверка работы персептрона на тестовой выборке

    correct = 0

    for i in range(len(X\_test)):

        prediction = perceptron.forward(X\_test[i])

        if prediction == y\_test[i]:

            correct += 1

    accuracy = correct / len(X\_test)

    print(f"Точность предсказания: {accuracy}")

Точность предсказания для чисел кратных/не кратных 3 равна 0.9027.