Лабораторная работа №1: Основные компоненты искусственного нейрона

Цель работы: Реализовать базовые элементы искусственного нейрона и понять принципы его работы. Получить практические навыки реализации математического аппарата нейрона на Python.

Задача: Разработать программную реализацию искусственного нейрона, включая функции взвешенной суммы и активации. Создать класс Neuron с методом прямого распространения (forward pass).

Оборудование и программное обеспечение:

- Компьютер с установленным Python 3.7+
- Рекомендуемые библиотеки: NumPy, Matplotlib (для визуализации)

Теоретическая часть: Искусственный нейрон состоит из следующих компонентов:

- **Входы (x_1, x_2, ..., x_n)** - входные сигналы
- **Веса (w_1, w_2, ..., w_n)** - коэффициенты важности входных сигналов
- **Сумматор** - вычисляет взвешенную сумму входов
- **Функция активации** - определяет выходной сигнал нейрона
- **Смещение (bias)** - дополнительный параметр для настройки порога активации

Ход работы:

1. Реализация функции взвешенной суммы:

```
def weighted_sum(inputs, weights, bias=0):
    """
    Вычисляет взвешенную сумму входных сигналов

    Параметры:
    inputs - список входных значений [x1, x2, ..., xn]
    weights - список весов [w1, w2, ..., wn]
    bias - смещение (по умолчанию 0)

    Возвращает:
    Взвешенную сумму inputs * weights + bias
    """
    if len(inputs) != len(weights):
        raise ValueError("Количество входов и весов должно
совпадать")

    total = bias
    for i in range(len(inputs)):
        total += inputs[i] * weights[i]
    return total
```

2. Реализация функций активации:

```
import math

def step_function(x, threshold=0):
```

```
    """
    Пороговая функция активации

    Параметры:
    x - входное значение
    threshold - порог срабатывания

    Возвращает:
    1 если x >= threshold, иначе 0
    """
    return 1 if x >= threshold else 0

def sigmoid(x):
    """
    Сигмоидальная функция активации

    Параметры:
    x - входное значение

    Возвращает:
    Значение в диапазоне от 0 до 1
    """
    return 1 / (1 + math.exp(-x))

def relu(x):
    """
    Функция активации ReLU (Rectified Linear Unit)

    Параметры:
    x - входное значение

    Возвращает:
    max(0, x)
    """
    return max(0, x)
```

3. Создание класса Neuron:

```
import random

class Neuron:
```

```
    def __init__(self, num_inputs, activation_function='sigmoid'):
        """
        Инициализация нейрона

        Параметры:
        num_inputs - количество входных сигналов
        activation_function - тип функции активации
        """
        self.weights = [random.uniform(-0.5, 0.5) for _ in
range(num_inputs)]
        self.bias = random.uniform(-0.1, 0.1)
        self.activation_function = activation_function

    def forward(self, inputs):
        """
        Прямое распространение сигнала

        Параметры:
        inputs - список входных значений

        Возвращает:
        Выходное значение нейрона после применения функции
активации
        """
        if len(inputs) != len(self.weights):
            raise ValueError("Количество входов не соответствует
количеству весов")

        # Вычисляем взвешенную сумму
        z = weighted_sum(inputs, self.weights, self.bias)

        # Применяем функцию активации
        if self.activation_function == 'step':
            return step_function(z)
        elif self.activation_function == 'sigmoid':
            return sigmoid(z)
        elif self.activation_function == 'relu':
            return relu(z)
        else:
            raise ValueError("Неизвестная функция активации")
```

4. Практическая проверка и тестирование:

Тест 1: Проверка функций активации

```
# Тестируем функции активации
test_values = [-2, -1, 0, 1, 2]

print("Пороговая функция:")
```

```
for x in test_values:
    print(f"step({x}) = {step_function(x)}")

print("\nСигмоида:")
for x in test_values:
    print(f"sigmoid({x}) = {sigmoid(x):.3f}")

print("\nReLU:")
for x in test_values:
    print(f"relu({x}) = {relu(x)}")
```

Тест 2: Проверка работы нейрона

```
# Создаем и тестируем нейрон
neuron = Neuron(3, 'sigmoid')
inputs = [1, 2, 3]

print(f"Веса нейрона: {[f'{w:.3f}' for w in neuron.weights]}")
print(f"Смещение нейрона: {neuron.bias:.3f}")

output = neuron.forward(inputs)
print(f"Вход: {inputs}, Выход: {output:.3f}")

# Тестируем с разными функциями активации
print("\nСравнение функций активации:")
test_inputs = [0.5, -0.2, 1.0]

neuron_step = Neuron(3, 'step')
neuron_relu = Neuron(3, 'relu')

print(f"Step: {neuron_step.forward(test_inputs)}")
print(f"ReLU: {neuron_relu.forward(test_inputs):.3f}")
print(f"Sigmoid: {neuron.forward(test_inputs):.3f}")
```

Таблица результатов тестирования:

Функция активации	Входные данные	Выходное значение
Пороговая	[1, 2, 3]	
Сигмоида	[1, 2, 3]	

ReLU	[1, 2, 3]	
Пороговая	[0.5, -0.2, 1.0]	
Сигмоида	[0.5, -0.2, 1.0]	
ReLU	[0.5, -0.2, 1.0]	

Выводы:

В данной лабораторной работе необходимо отразить:

1. Принципы работы искусственного нейрона и его основных компонентов
2. Особенности различных функций активации и их влияние на выходной сигнал
3. Результаты тестирования реализованных функций
4. Преимущества и недостатки разных функций активации
5. Практическое значение полученных результатов для построения более сложных нейронных сетей

Приложение:

- Исходный код всех реализованных функций и классов
- Результаты тестирования
- Графики функций активации (опционально)