Домашнее задание № 1: Реализация однослойного персептрона

17 сентября 2024 г.

Сергей Виленский, ИУ9-72Б

Цель работы

1. Реализовать на языке высокого уровня однослойный персептрон и проверить его работоспособность на примере искусственных данных типа цифр от 0 до 9 и букв русского алфавита. Размер поля 5x4.
2. Исследовать работу персептрона на основе использования различных функций активации. (Линейной, сигмоиды, гиперболического тангенса, ReLU).

Индивидуальный вариант

Персептрон должен распознавать следующие символы: `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `A`, `E`, `T`, `K` и `Y`.

Реализация

В ходе выполения домашнего задания был реализован класс `Perceptron` с методами `__init__`, `softmax`, `predict`, `fit`, `evaluate`: * метод `__init__` принимает размер входных данных, количество результирующих классов, функцию активации, ее производную, константу скорости обучения и количество эпох обучения, и инициализируются веса модели и производная функции потерь; * метод `softmax` принимает данные вывода функции активации для каждого класса и нормализирует их в формат распределения вероятностей для каждого класса; * `predict` принимает один экземпляр входных данных и возвращает вероятности для каждого класса; * `fit` принимает набор входных данных и меток для них и обучает модель на основе метода обратного распространения ошибки; * `evalute` так же принимает размеченный набор входных данных и возвращает точность обучения модели, оценивая погрешность между фактическими метками и результатами предсказаний модели на тестовых данных.

Для визуализации результатов обучения модели при разном количестве эпох

была написана функция `find_min_epochs_count`, которая строит графики зависимости точности обучения модели от количества эпох обучения.

В качестве базы данных хранения входных данных с метками использовался формат файлов `JSON`.

Объем выборки составляет 39 размеченных образцов входных данных.

Исходный код

```
import json
import numpy as np
import matplotlib.pyplot as plt

class Perceptron:
    def __init__(
        self,
        input_size,
        num_classes,
        activation_function,
        activation_function_derivative,
        learning_rate=0.01,
        epochs=1000
    ):
        """
        Инициализация персептрона
        :param input_size: количество входов (размерность входных данных)
        :param num_classes: количество классов для классификации
        :param learning_rate: скорость обучения
        :param epochs: количество итераций обучения
        :param activation_function: функция активации
        """
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.zeros((num_classes, input_size + 1))  # Веса для каждого класса + 
        self.activation_function = lambda x: list(map(activation_function, x))
        self.activation_function_derivative = lambda x: list(map(activation_function_derivat
        self.mse = lambda target, output: (output - target) ** 2 / len(target)
        self.mse_derivative = lambda target, output: 2 * (output - target) / len(target)

    def softmax(self, x):
        """
        Softmax для получения вероятностей
        :param x: входной вектор
        :return: вектор вероятностей
        """
        exp_x = np.exp(x - np.max(x))  # Для стабильности вычислений вычитаем максимум
```

```
        return exp_x / exp_x.sum(axis=0)

    def predict(self, X):
        """
        Прогноз на основе текущих весов
        :param X: Входные данные (без смещения)
        :return: Вектор вероятностей для каждого класса
        """
        linear_output = np.dot(self.weights[:, 1:], X) + self.weights[:, 0]
        return self.softmax(self.activation_function(linear_output))

    def fit(self, X, y):
        """
        Обучение персептрона с использованием MSE
        :param X: Массив входных данных, размерность (n_samples, input_size)
        :param y: Целевые метки (n_samples,) в формате индексов классов
        """
        for epoch in range(self.epochs):
            for i in range(len(X)):
                # Прямой проход: получаем прогноз (вероятности)
                probabilities = self.predict(X[i])
                # Преобразуем целевую метку в one-hot вектор
                target = np.zeros(len(probabilities))
                target[y[i]] = 1
                # Вычисляем производную MSE по выходу
                mse_grad = self.mse_derivative(target, probabilities)
                # Вычисляем градиент активационной функции
                activation_grad = self.activation_function_derivative(np.dot(self.weights[:,
                # Общий градиент
                gradient = mse_grad * activation_grad
                # Обновляем веса для каждого класса
                self.weights[:, 1:] -= self.learning_rate * np.outer(gradient, X[i])
                self.weights[:, 0] -= self.learning_rate * gradient  # Обновляем смещение (b

    def evaluate(self, X, y):
        """
        Оценка точности модели на тестовых данных
        :param X: Входные данные
        :param y: Истинные метки
        :return: Точность модели
        """
        correct_predictions = 0
        for i in range(len(X)):
            probabilities = self.predict(X[i])
            prediction = np.argmax(probabilities)
            if prediction == y[i]:
```

```
                correct_predictions += 1
        return correct_predictions / len(X)


    def evaluate_mse(self, X, y):
        """
        Оценка точности модели на тестовых данных по функции потерь
        :param X: Входные данные
        :param y: Истинные метки
        :return: Среднее значение функции потерь MSE
        """
        mse_accuracy = 0
        for i in range(len(X)):
            probabilities = self.predict(X[i])
            target = np.zeros(len(probabilities))
            target[y[i]] = 1
            mse_accuracy += self.mse(target, probabilities).sum()
        return mse_accuracy / len(X)

def find_min_epochs_count(
    X, y,
    max_epochs,
    activation_function,
    activation_function_derivative,
    activation_name,
    return_mse_accuracy=False
):
    input_size = len(X[0])
    num_classes = max(y) + 1
    accuracies = []
    epoch_list = []

    for epochs in range(1, max_epochs + 1):
        perceptron = Perceptron(
            input_size=input_size,
            num_classes=num_classes,
            activation_function=activation_function,
            activation_function_derivative=activation_function_derivative,
            learning_rate=0.01,
            epochs=epochs,
        )
        perceptron.fit(X, y)
        if return_mse_accuracy:
            accuracy = perceptron.evaluate_mse(X, y)
        else:
            accuracy = perceptron.evaluate(X, y)
```

```
            accuracies.append(accuracy)
            epoch_list.append(epochs)

        plt.plot(epoch_list, accuracies, marker=".", label=activation_name)

        max_accurace_i = np.argmax(accuracies)
        return epoch_list[max_accurace_i], accuracies[max_accurace_i]

data = json.load(open("data.json"))

keys = {key: i for i, key in enumerate(set(next(zip(*data))))}
alphabet = {key: i for i, key in keys.items()}
X = np.array([
        [
            (1 if draw_cell == "#" else 0)
            for draw_cell in ''.join(draw)
        ] for _, draw in data
    ])
y = np.array([keys[key] for key, _ in data])

EPOCHS = 25

k = 9
print("Линейная функция активации:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x:
        (lambda x, k: k * x)(x, k=k),
    activation_function_derivative=lambda x: k,
    activation_name="Линейная функция активации"
))

k = 10
print("ReLu:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x:
        (lambda x, k: k * x if x > 0 else 0)(x, k=k),
    activation_function_derivative=lambda x: k,
    activation_name="ReLu"
))

sigmoida = lambda x, k: 1 / (1 + np.exp(-k * x))
sigmoida_derivative = lambda x, k: k * sigmoida(x, k) * (1 - sigmoida(x, k))
k = 24
print("Сигмоида:", find_min_epochs_count(
    X, y, EPOCHS,
```

```
    activation_function=lambda x: sigmoida(x, k=k),
    activation_function_derivative=lambda x: sigmoida_derivative(x, k=k),
    activation_name="Сигмоида"
))

tanh_derivative = lambda x, k: k * (1 - np.tanh(k * x) ** 2)
k = 8
print("Гиперболический тангенс:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x: np.tanh(k * x),
    activation_function_derivative=lambda x: tanh_derivative(x, k=k),
    activation_name="Гиперболический тангенс"
))

plt.title("Зависимость точности от количества эпох")
plt.xlabel("Количество эпох")
plt.ylabel("Точность")
plt.grid(True)
plt.legend()
plt.show()

EPOCHS = 25

k = 9
print("Линейная функция активации:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x:
        (lambda x, k: k * x)(x, k=k),
    activation_function_derivative=lambda x: k,
    activation_name="Линейная функция активации",
    return_mse_accuracy=True,
))

k = 10
print("ReLu:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x:
        (lambda x, k: k * x if x > 0 else 0)(x, k=k),
    activation_function_derivative=lambda x: k,
    activation_name="ReLu",
    return_mse_accuracy=True,
))

sigmoida = lambda x, k: 1 / (1 + np.exp(-k * x))
sigmoida_derivative = lambda x, k: k * sigmoida(x, k) * (1 - sigmoida(x, k))
k = 24
```

```
print("Сигмоида:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x: sigmoida(x, k=k),
    activation_function_derivative=lambda x: sigmoida_derivative(x, k=k),
    activation_name="Сигмоида",
    return_mse_accuracy=True,
))

tanh_derivative = lambda x, k: k * (1 - np.tanh(k * x) ** 2)
k = 8
print("Гиперболический тангенс:", find_min_epochs_count(
    X, y, EPOCHS,
    activation_function=lambda x: np.tanh(k * x),
    activation_function_derivative=lambda x: tanh_derivative(x, k=k),
    activation_name="Гиперболический тангенс",
    return_mse_accuracy=True,
))

plt.title("Зависимость точности от количества эпох")
plt.xlabel("Количество эпох")
plt.ylabel("MSE")
plt.grid(True)
plt.legend()
plt.show()
```

Входные данные

```
[
    ["0", [
        "####",
        "#__#",
        "#__#",
        "#__#",
        "####"
    ]],
    ["0", [
        "_##_",
        "#__#",
        "#__#",
        "#__#",
        "_##_"
    ]],
    ["1", [
        "__#_",
        "__#_",
        "__#_",
```

```
    "__#_",
    "__#_"
]],
["1", [
    "___#",
    "___#",
    "___#",
    "___#",
    "___#"
]],
["2", [
    "####",
    "___#",
    "####",
    "#___",
    "####"
]],
["2", [
    "####",
    "___#",
    "_##_",
    "#___",
    "####"
]],
["2", [
    "###_",
    "___#",
    "__#_",
    "##__",
    "####"
]],
["3", [
    "####",
    "___#",
    "####",
    "___#",
    "####"
]],
["3", [
    "###_",
    "___#",
    "###_",
    "___#",
    "###_"
]],
["3", [
```

```
        "###_",
        "___#",
        "####",
        "___#",
        "###_"
    ]],

    ["3", [
        "_##_",
        "___#",
        "_##_",
        "___#",
        "_##_"
    ]],
    ["4", [
        "#__#",
        "#__#",
        "####",
        "___#",
        "___#"
    ]],
    ["4", [
        "__##",
        "_#_#",
        "####",
        "___#",
        "___#"
    ]],
    ["5", [
        "####",
        "#___",
        "####",
        "___#",
        "####"
    ]],
    ["5", [
        "####",
        "#___",
        "###_",
        "___#",
        "###_"
    ]],
    ["6", [
        "####",
        "#___",
        "####",
```

```
    "#__#",
    "####"
]],
["6", [
    "_###",
    "#___",
    "###_",
    "#__#",
    "_##_"
]],
["7", [
    "####",
    "___#",
    "___#",
    "___#",
    "___#"
]],
["7", [
    "####",
    "___#",
    "__#_",
    "_#__",
    "#___"
]],
["7", [
    "####",
    "___#",
    "####",
    "_#__",
    "#___"
]],
["8", [
    "####",
    "#__#",
    "####",
    "#__#",
    "####"
]],
["8", [
    "_##_",
    "#__#",
    "_##_",
    "#__#",
    "_##_"
]],
["9", [
```

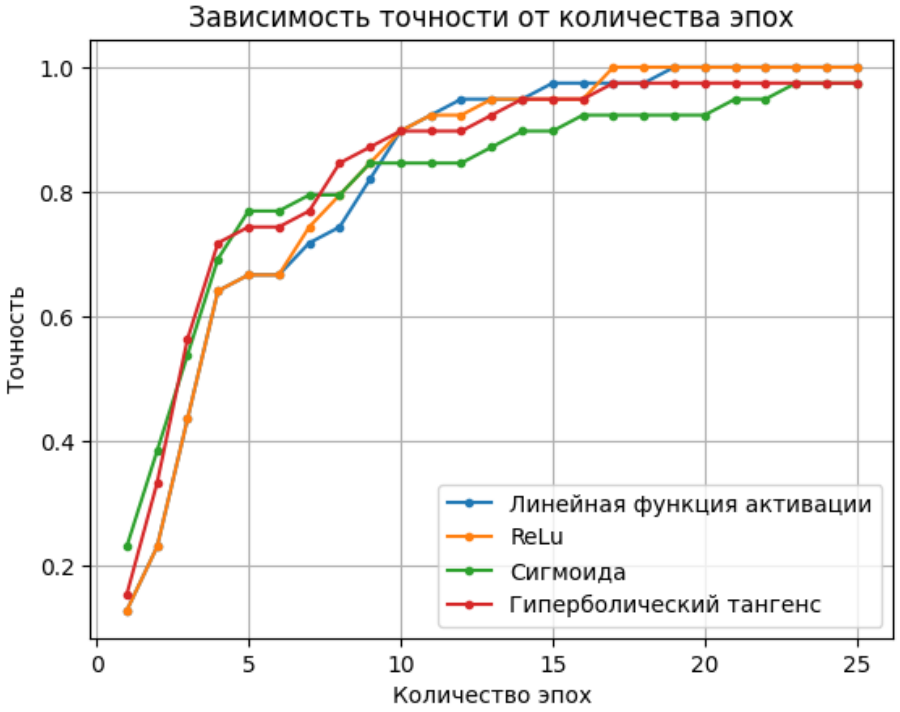
```
        "####",
        "#__#",
        "####",
        "___#",
        "####"
    ]],
    ["9", [
        "_##_",
        "#__#",
        "_##_",
        "___#",
        "_##_"
    ]],
    ["9", [
        "_##_",
        "#__#",
        "_###",
        "___#",
        "###_"
    ]],
    ["A", [
        "_##_",
        "#__#",
        "####",
        "#__#",
        "#__#"
    ]],
    ["A", [
        "####",
        "#__#",
        "####",
        "#__#",
        "#__#"
    ]],
    ["E", [
        "####",
        "#___",
        "####",
        "#___",
        "####"
    ]],
    ["E", [
        "_###",
        "#___",
        "_###",
        "#___",
```

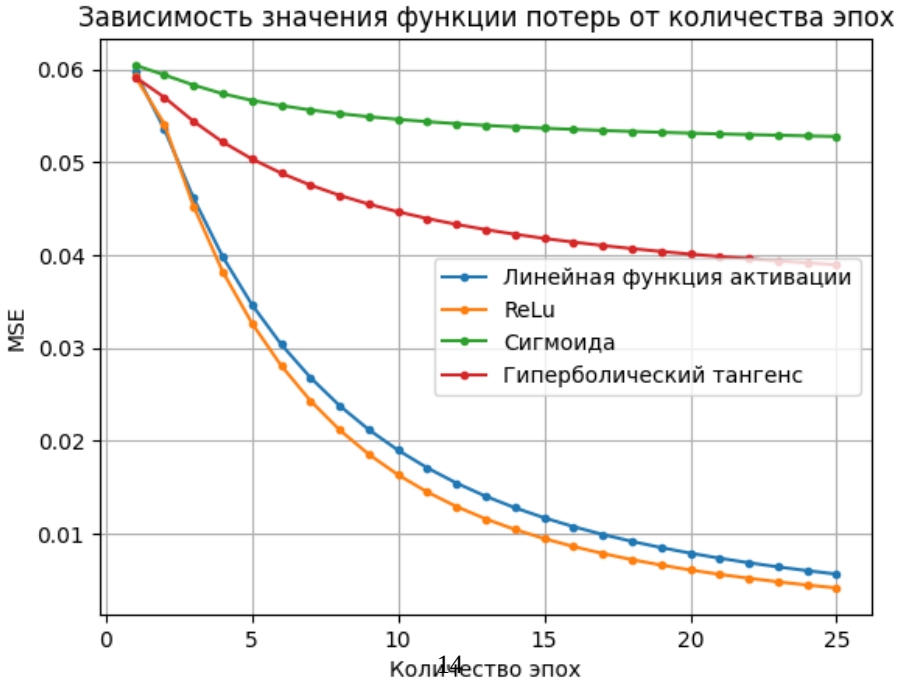
```
    "_###"
]],
["T", [
    "###_",
    "_#__",
    "_#__",
    "_#__",
    "_#__"
]],
["T", [
    "_###",
    "__#_",
    "__#_",
    "__#_",
    "__#_"
]],
["K", [
    "#__#",
    "#_#_",
    "##__",
    "#_#_",
    "#__#"
]],
["K", [
    "#_#_",
    "#_#_",
    "##__",
    "#_#_",
    "#_#_"
]],
["K", [
    "_#_#",
    "_#_#",
    "_##_",
    "_#_#",
    "_#_#"
]],
["Y", [
    "#__#",
    "#__#",
    "_##_",
    "_##_",
    "_##_"
]],
["Y", [
    "#_#_",
```

```
            "#_#_",
            "_#__",
            "_#__",
            "_#__"
        ]],
        ["Y", [
            "#_#_",
            "#_#_",
            "#_#_",
            "_#__",
            "_#__"
        ]],
        ["Y", [
            "_#_#",
            "_#_#",
            "__#_",
            "__#_",
            "__#_"
        ]],
        ["Y", [
            "_#_#",
            "_#_#",
            "_#_#",
            "__#_",
            "__#_"
        ]]
    ]
```

Результаты

Графики зависимостей





Вывод программы

Результаты использования разных функций активации в формате `{название}: ({минимальное количесвто эпох, за которые был достигнут лучший результат}, {лучший результат оценки точности обученной модели})`

```
Линейная функция активации: (19, 1.0)
ReLu: (17, 1.0)
Сигмоида: (23, 0.9743589743589743)
Гиперболический тангенс: (17, 0.9743589743589743)
```

Вывод

До 7-ой эпохи точность обучения моделей, в которых использовались сигмоида и гиперболический тангенс в качестве функций активации, растет заметно быстрее в сравнении с точностью обучения моеделей, использующих в качестве функций активации линейную и ReLU, однако последние достигают 100%-ой точности, в то время как первые стремятся к ней, но достигают лишь точности до 98%. По этой причине наиболее эффективной в данном случае можно считать функцию активации ReLU, так как при ее использовании модель достигает максимально возможной точности распознавания всего лишь за 17 эпох, в то время как использование прочих функций вынуждает выделять большее количество эпох для достижения наилучшей точности модели.