Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «SLP. Классификация»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Карагодин Д. Л.

**Проверил:**

Туз И. С.

Брест 2023

**Цель:** Изучить обучение и функционирование ИНС при решении задач классификации.

**Ход работы**

**Вариант 5**

Написать нейронную сеть(single layer perceptron) для решения задачи классификации.

Обучить сеть с использованием константного и адаптивного шага обучения, online-

learning и batch-learning. Результаты для каждого варианта сети занести в таблицу(

test error, количество эпох, время обучения и тд)

Сгенерировать датасет, разделяемый прямой на 2 класса. Добавить шумы(точки

класса 1, которые попадают на “территорию класса 0” и наоборот)

|  |  |
| --- | --- |
| № варианта | Функция |
| 5 | y = 9x - 2 |

**Код программы:**

import numpy as np

import random

import time

from matplotlib import pyplot as plt

E\_arr = []

class SingleLayerPerceptron:

    def \_\_init\_\_(self, input\_size, learning\_rate=0.046):

        self.input\_size = input\_size

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

        self.bias = np.random.randn(1)

        self.learning\_rate = learning\_rate

        self.start\_rate = learning\_rate

    def reset(self):

        global E\_arr

        E\_arr.clear()

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

        self.bias = np.random.randn(1)

        self.learning\_rate = self.start\_rate

    def predict(self, x):

        linear\_output = np.dot(self.weights, x) - self.bias

        return 1 if linear\_output >= 0 else 0

    def train\_constant\_learning\_rate(self, X, y, epochs: int,isAdapt: bool = False):

        start\_time = time.time()

        global E\_arr

        for epoch in range(epochs):

            e\_arr = []

            for i in range(len(X)):

                prediction = self.predict(X[i])

                error = prediction - y[i]

                e\_arr.append(error)

                self.weights = self.weights - self.learning\_rate \* error \* np.array(X[i])

                self.bias = self.bias + self.learning\_rate \* error

            self.learning\_rate = self.start\_rate\*(1.0 / (1.0 + epoch)) if isAdapt else self.learning\_rate

            E\_arr.append(np.sum(abs(np.array(e\_arr))))

        end\_time = time.time()

        training\_time = end\_time - start\_time

        return training\_time

    def train\_batch\_learning(self, X, y, epochs: int = 100,batch\_size: int = 10,isAdapt: bool = False ):

        start\_time = time.time()

        global E\_arr

        for epoch in range(epochs):

            e\_arr = []

            for i in range(0, len(X), batch\_size):

                X\_batch = X[i:i+batch\_size]

                y\_batch = y[i:i+batch\_size]

                predictions = [self.predict(x) for x in X\_batch]

                errors = np.array(predictions) - np.array(y\_batch)

                for err in errors:

                    e\_arr.append(err)

                self.weights = self.weights - self.learning\_rate \* np.dot(errors, X\_batch)

                self.bias = self.bias + self.learning\_rate \* np.sum(errors)

            self.learning\_rate = self.start\_rate\*(1.0 / (1.0 + epoch)) if isAdapt else self.learning\_rate

            E\_arr.append(np.sum(abs(np.array(e\_arr))))

        end\_time = time.time()

        training\_time = end\_time - start\_time

        return training\_time

def generate\_dataset():

    X = []

    Y = []

    for \_ in range(500):

        x1 = random.uniform(-10,10)

        x2 = 9 \* x1 - 2 + random.uniform(-10,10)

        X.append([x1,x2])

        Y.append(1 if x2 > 9\*x1-2 else 0)

    for \_ in range(40):

        index = int(random.uniform(0,len(Y)))

        Y[index] = 1 - Y[index]

    return X, Y

# Разделение датасета на обучающую и тестовую выборки

def split\_dataset(X, y, split\_ratio=0.8):

    split\_index = int(len(X) \* split\_ratio)

    X\_train = X[:split\_index]

    y\_train = y[:split\_index]

    X\_test = X[split\_index:]

    y\_test = y[split\_index:]

    return X\_train, y\_train, X\_test, y\_test

# Расчет ошибки на тестовой выборке

def calculate\_test\_error(model, X\_test, y\_test):

    predictions = [model.predict(x) for x in X\_test]

    errors = [1 if p != y else 0 for p, y in zip(predictions, y\_test)]

    test\_error = sum(errors) / len(errors)

    return test\_error

# Генерация датасета

X, Y = generate\_dataset()

# Разделение на обучающую и тестовую выборки

X\_test, Y\_test = [],[]

for \_ in range(500):

        x1 = random.uniform(-10,10)

        x2 = 9 \* x1 - 2 + random.uniform(-10,10)

        X\_test.append([x1,x2])

        Y\_test.append(1 if x2 > 9\*x1-2 else 0)

epochs = 1000

# Запуск обучения и оценка результатов

model = SingleLayerPerceptron(2)

training\_time = model.train\_constant\_learning\_rate(X, Y, epochs=epochs,isAdapt=False)

test\_error = calculate\_test\_error(model, X\_test, Y\_test)

print(training\_time,"ms","/\*/",test\_error,"/\*/",min(E\_arr))

plt.plot(range(epochs),E\_arr)

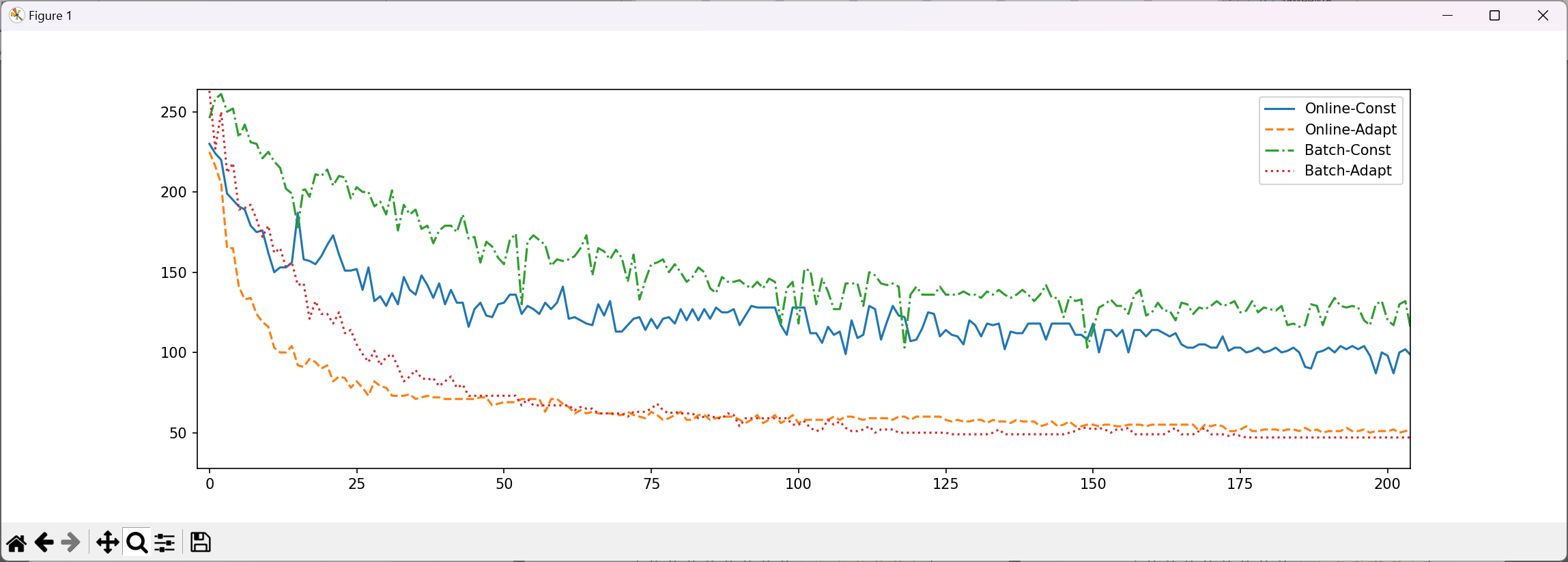
plt.show()

**Результат программы:**

**Таблица сравнения:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Эпохи | Процент ошибки (%) | Время выполнения (сек.) |
| Online-Const | 1000 | 2.2 | 3.310649871826172 |
| Online-Adapt | 1000 | 1.4 | 3.3112215995788574 |
| Batch-Const | 1000 | 18.2 | 2.5811715126037598 |
| Batch-Adapt | 1000 | 0.2 | 2.5734450817108154 |

**График изменения ошибок:**



**Вывод:** изучил обучение и функционирование ИНС при решении задач классификации.