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

Литвинюк Т. В.

**Проверил:**

Туз И. С.

Брест 2023

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

**Ход работы**

**Вариант 1**

y = 2x - 3

from matplotlib import pyplot as plt

import numpy as np

import time

def f(x): return 2 \* x - 3

def get\_class(x, y):

return int(y > f(x))

def make\_dataset(count: int, noise\_percent: int):

train\_data = [np.array([x, y]) for x, y in [(np.random.random()\*4-1.5, np.random.random()\*4-2)

for \_ in range(count)]]

train\_test = [get\_class(x, y) for x, y in train\_data]

for \_ in range(int(count \* noise\_percent)):

train\_test[np.random.randint(count)] = not train\_data

return np.array(train\_data), np.array(train\_test)

E\_arr = []

class SingleLayerPerceptron:

def \_\_init\_\_(self, input\_size, learning\_rate=0.0001):

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):

output = np.dot(self.weights, x) - self.bias

return int(1/(1+np.exp(-output)) > 0.5)

def deritivate(self, x):

output = self.predict(x)

return output\*(1-output)

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)

if isAdapt and abs(error) > 0.1E-10 and abs(self.deritivate(X[i])) > 0.1E-10:

delta\_weights = error\*self.deritivate(X[i])\*np.array(X[i])

delta\_bias = -error\*self.deritivate(X[i])

b = 0.0

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

b += delta\_weights[j]\*X[i][j] - delta\_bias

# if b < 0.1E-6: b = 0.1E-6

self.learning\_rate = b\*error / \

(b\*\*2) if b\*error/(b\*\*2) <= 0.3 else 0.3

self.weights = self.weights - \

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

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

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 = []

for x in X\_batch:

predictions.append(self.predict(x))

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

for err in errors:

e\_arr.append(err)

if isAdapt:

b\_arr = []

for k in range(len(X\_batch)):

if abs(errors[k]) < 0.1E-5:

errors[k] = 0.1E-5 if errors[k] >= 0 else -0.1E-5

deritivate = self.deritivate(X\_batch[k])

if abs(deritivate) < 0.1E-6:

deritivate = 0.1E-6 if self.deritivate(

X\_batch[k]) >= 0 else -0.1E-6

delta\_weights = errors[k] \* \

deritivate \* np.array(X\_batch[k])

delta\_bias = -errors[k]\*deritivate

b = 0.0

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

b += delta\_weights[j]\*X\_batch[k][j] - delta\_bias

if abs(b) < 0.1E-6:

b = 0.1E-5 if b <= 0 else -0.1E-5

b\_arr.append(b)

# print("barr",b\_arr,"err",errors)

up = 0

down = 0

for k in range(len(b\_arr)):

up += b\_arr[k]\*errors[k]

down += b\_arr[k]\*\*2

self.learning\_rate = up/down/10000 if up/down/10000 <= 0.3 else 0.3

# print("lr",self.learning\_rate)

self.weights = self.weights - \

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

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

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

end\_time = time.time()

training\_time = end\_time - start\_time

return training\_time

X\_train, Y\_train = make\_dataset(100, 0.1)

X\_test, Y\_test = make\_dataset(40, 0)

epochs = 200

NN = SingleLayerPerceptron(2)

##################################################

print("CONSTANT ONLINE LEARNING")

NN.train\_constant\_learning\_rate(X\_train, Y\_train, epochs)

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

plt.show()

output = ([NN.predict(x) for x in X\_test])

correct\_points, incorrect\_points = [], []

for i, point in enumerate(output):

if point == Y\_test[i]:

correct\_points.append(X\_test[i])

else:

incorrect\_points.append(X\_test[i])

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in X\_train], [x[1] for x in X\_train], c='blue')

plt.show()

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in correct\_points], [x[1] for x in correct\_points], c='blue')

plt.scatter([x[0] for x in incorrect\_points], [x[1] for x in incorrect\_points], c='red')

plt.show()

NN.reset()

##################################################

print("BATCH ONLINE LEARNING")

NN.train\_batch\_learning(X\_train, Y\_train, epochs)

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

plt.show()

output = ([NN.predict(x) for x in X\_test])

correct\_points, incorrect\_points = [], []

for i, point in enumerate(output):

if point == Y\_test[i]:

correct\_points.append(X\_test[i])

else:

incorrect\_points.append(X\_test[i])

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in X\_train], [x[1] for x in X\_train], c='blue')

plt.show()

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in correct\_points], [x[1] for x in correct\_points], c='blue')

plt.scatter([x[0] for x in incorrect\_points], [x[1] for x in incorrect\_points], c='red')

plt.show()

NN.reset()

##################################################

print("ADAPT ONLINE LEARNING")

NN.train\_constant\_learning\_rate(X\_train, Y\_train, epochs, isAdapt=True)

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

plt.show()

output = ([NN.predict(x) for x in X\_test])

correct\_points, incorrect\_points = [], []

for i, point in enumerate(output):

if point == Y\_test[i]:

correct\_points.append(X\_test[i])

else:

incorrect\_points.append(X\_test[i])

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in X\_train], [x[1] for x in X\_train], c='blue')

plt.show()

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in correct\_points], [x[1] for x in correct\_points], c='blue')

plt.scatter([x[0] for x in incorrect\_points], [x[1] for x in incorrect\_points], c='red')

plt.show()

NN.reset()

##################################################

print("BATCH ADAPT LEARNING")

NN.train\_batch\_learning(X\_train, Y\_train, epochs, isAdapt=True)

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

plt.show()

output = ([NN.predict(x) for x in X\_test])

correct\_points, incorrect\_points = [], []

for i, point in enumerate(output):

if point == Y\_test[i]:

correct\_points.append(X\_test[i])

else:

incorrect\_points.append(X\_test[i])

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

plt.scatter([x[0] for x in X\_train], [x[1] for x in X\_train], c='blue')

plt.show()

plt.plot(range(-2, 3, 1), [f(x) for x in range(-2, 3, 1)])

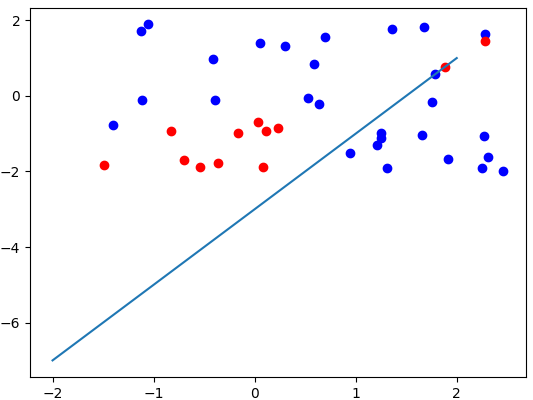
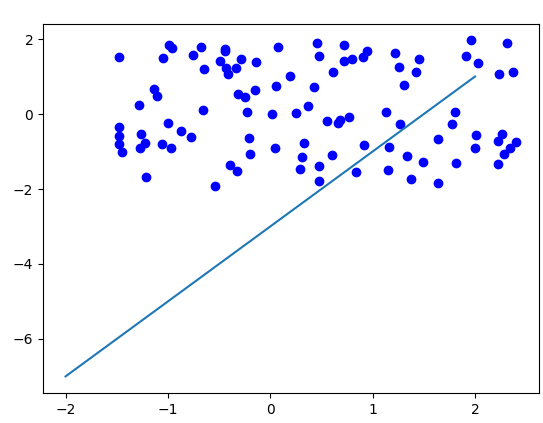
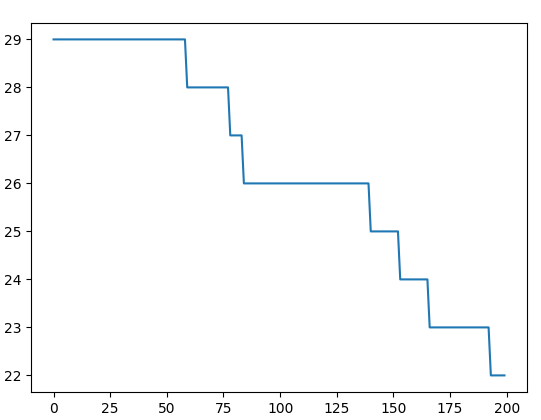
plt.scatter([x[0] for x in correct\_points], [x[1] for x in correct\_points], c='blue')

plt.scatter([x[0] for x in incorrect\_points], [x[1] for x in incorrect\_points], c='red')

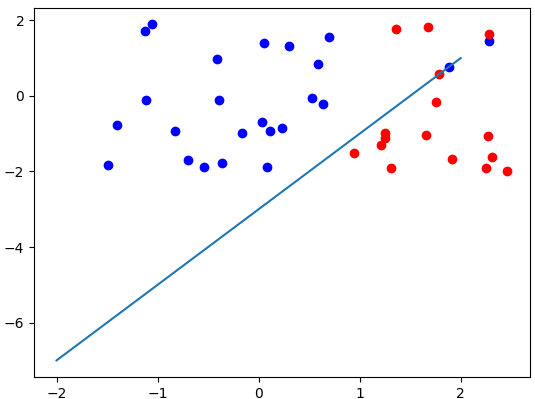
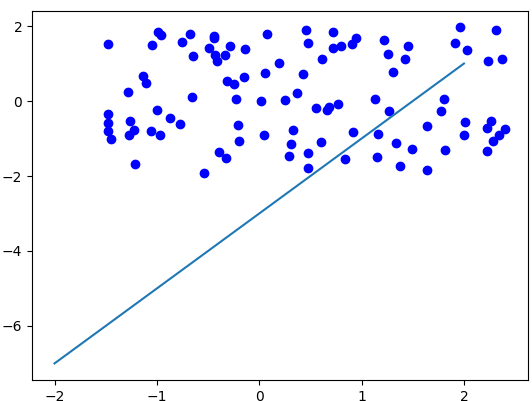
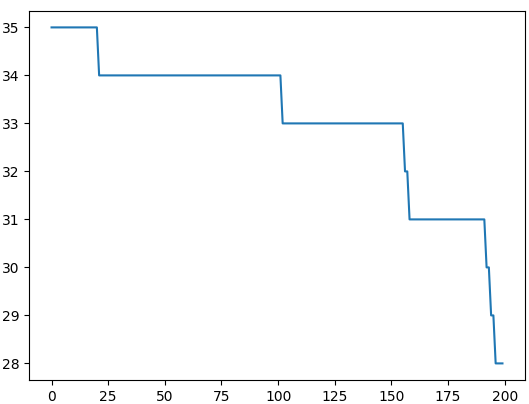
plt.show()

NN.reset()

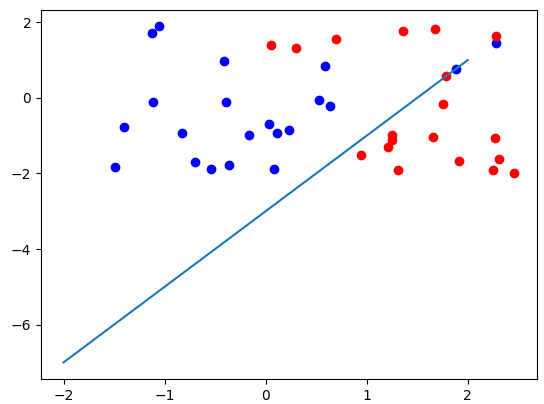
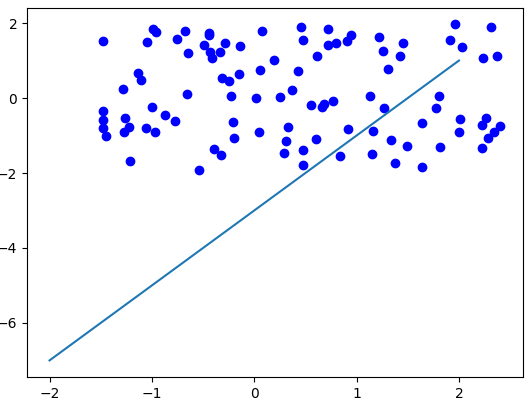
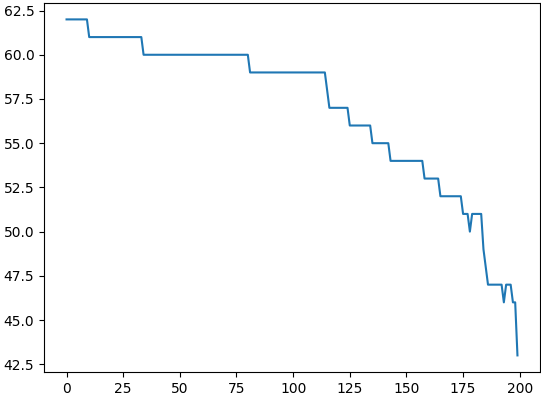
Online const:



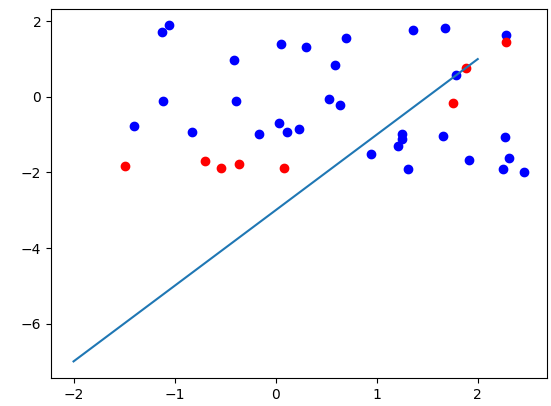
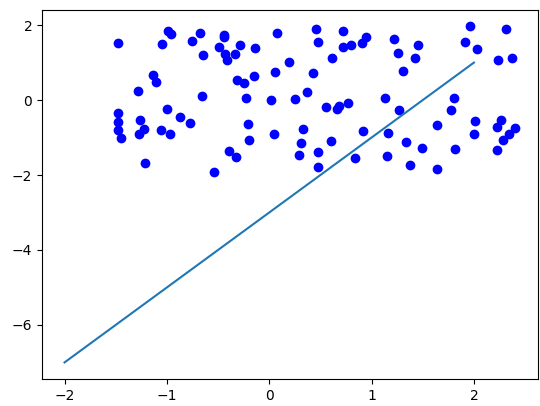
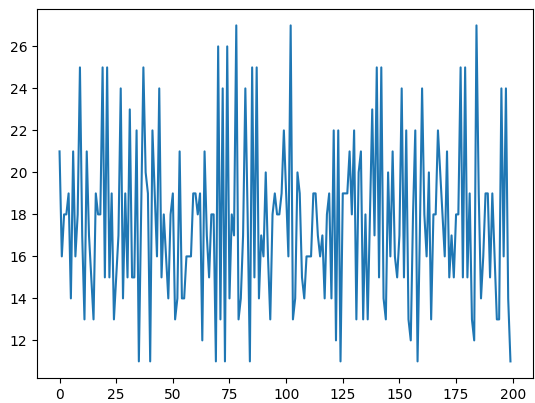
Batch const:



Online adapt:



Batch adapt:



**Вывод:** в ходе лабораторной работы я изучил функционирование ИНС при решении задач классификации.