Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

за 1 семестр

По дисциплине: «МиАПР»

Тема: «Нелинейные ИНС в задачах прогнозирования»

Выполнил:

Студент 2 курса

Группы ПО-4(1)

Грибовский Д.С.

Проверил:

Крощенко А.А.

2020

**Лабораторная работа №4**

Нелинейные ИНС в задачах прогнозирования

Цель работы: Изучить обучение и функционирование нелинейной ИНС при решении задач распознавания образов.

**Вариант 6**

**Задание:**

Спрогнозировать нелинейный временной ряд, применяя параметры лабораторной работы №3. При этом необходимо использовать алгоритм обучения многослойной ИНС с адаптивным шагом.

Y = a\*cos(b\*x)+c\*sin(d\*x)

**Код программы:**

import numpy as np

#import sys

def func(x):

a, b, c, d, step = 0.2, 0.6, 0.05, 0.6, 0.1

return a \* np.cos(b \* x \* step) + c \* np.sin(d \* x \* step)

def activation(x):

return np.tanh(x)

#e = 2.718

#return 1 / (1 + e \*\* (-x))

#return 1 / (1 + np.exp(-x))

def derivative(x):

return 1 - (activation(x) \*\* 2)

#return -((activation(x) \*\* 2) - 1)

#return x \* (1 - x)

def error(y,Y):

return np.mean((y - Y) \*\* 2)

def adaptive(errors,outputs,inputs):

return np.divide(np.sum(np.dot(np.square(errors), np.subtract(1,np.square(outputs)))), np.multiply(np.add(1,np.sum(np.square(inputs))), np.sum(np.dot(np.square(errors), np.square(np.subtract(1,np.square(outputs)))))))

def training(inputs,predict,weights\_hidden,weights\_input,learning\_rate):

inputs\_hidden = np.dot(weights\_hidden,inputs)

outputs\_hidden = activation\_mapper(inputs\_hidden)

inputs\_input = np.dot(weights\_input,outputs\_hidden)

outputs\_input = activation(inputs\_input)

error\_input = np.array([outputs\_input[0] - predict])

gradient\_input = derivative(outputs\_input[0])

delta\_input = error\_input \* gradient\_input

weights\_input -= learning\_rate \* np.dot(delta\_input,outputs\_hidden.reshape(1,len(outputs\_hidden)))

error\_hidden = delta\_input \* weights\_input

gradient\_hidden = derivative(outputs\_hidden)

delta\_hidden = error\_hidden \* gradient\_hidden

weights\_hidden -= learning\_rate \* np.dot(inputs.reshape(len(inputs),1),delta\_hidden).T

learning\_rate = adaptive(error\_hidden,outputs\_hidden,inputs) # адаптивная скорость обучения

return weights\_hidden,weights\_input,learning\_rate

def prediction(inputs,weights\_hidden,weights\_input):

inputs\_hidden = np.dot(weights\_hidden,inputs)

outputs\_hidden = activation\_mapper(inputs\_hidden)

inputs\_input = np.dot(weights\_input,outputs\_hidden)

outputs\_input = activation(inputs\_input)

return outputs\_input

activation\_mapper = np.vectorize(activation)

learning = []

predictions = []

learning\_rate = 0.5

epoch = 0

epoch\_maximum = 1000

error\_minimum = 1e-5 # минимальная ошибка

n\_input = 10 # количество входов

n\_hidden = 4 # количество элементов скрытого слоя

n\_train = 30 # размер выборки для обучения

n\_predict = 15 # размер выборки для прогназированния

param = 0

w\_hidden = np.random.normal(0.0,2 \*\* -0.5,(n\_hidden,n\_input))

w\_input = np.random.normal(0.0,1,(1,n\_hidden))

#w\_hidden = np.random.normal(0.0,0.1,(n\_hidden,n\_input))

#w\_input = np.random.normal(0.0,0.1,(1,n\_hidden))

for i in range(n\_train):

inp, com = [], []

for j in range(n\_input):

inp.append(func(param))

param += 1

com.append(inp)

com.append(func(param))

learning.append(tuple(com))

while True:

inputs, predicts = [], []

for sample,predict in learning:

w\_hidden,w\_input,learning\_rate = training(np.array(sample),predict,w\_hidden,w\_input,learning\_rate)

inputs.append(np.array(sample))

predicts.append(np.array(predict))

error\_learning = error(prediction(np.array(inputs).T,w\_hidden,w\_input),np.array(predicts))

epoch += 1

#sys.stdout.write("\rОшибка: {}, Эпохи: {}".format(str(error\_learning),str(epoch)))

if error\_learning <= error\_minimum or epoch > epoch\_maximum:

break

print("Ошибка: {}, Эпохи: {}".format(str(error\_learning),str(epoch)))

for i in range(n\_train,n\_train + n\_predict):

inp, com = [], []

for j in range(n\_input):

inp.append(func(param))

param += 1

com.append(inp)

com.append(func(param))

predictions.append(tuple(com))

print("\nРЕЗУЛЬТАТЫ ОБУЧЕНИЯ:")

for sample,predict in learning:

output = prediction(sample,w\_hidden,w\_input)

print("прогноз: {:<20} ожидаемый: {:<30} погрешность: {:<20}".format(str(output),str(np.array(predict)),str(output - predict)))

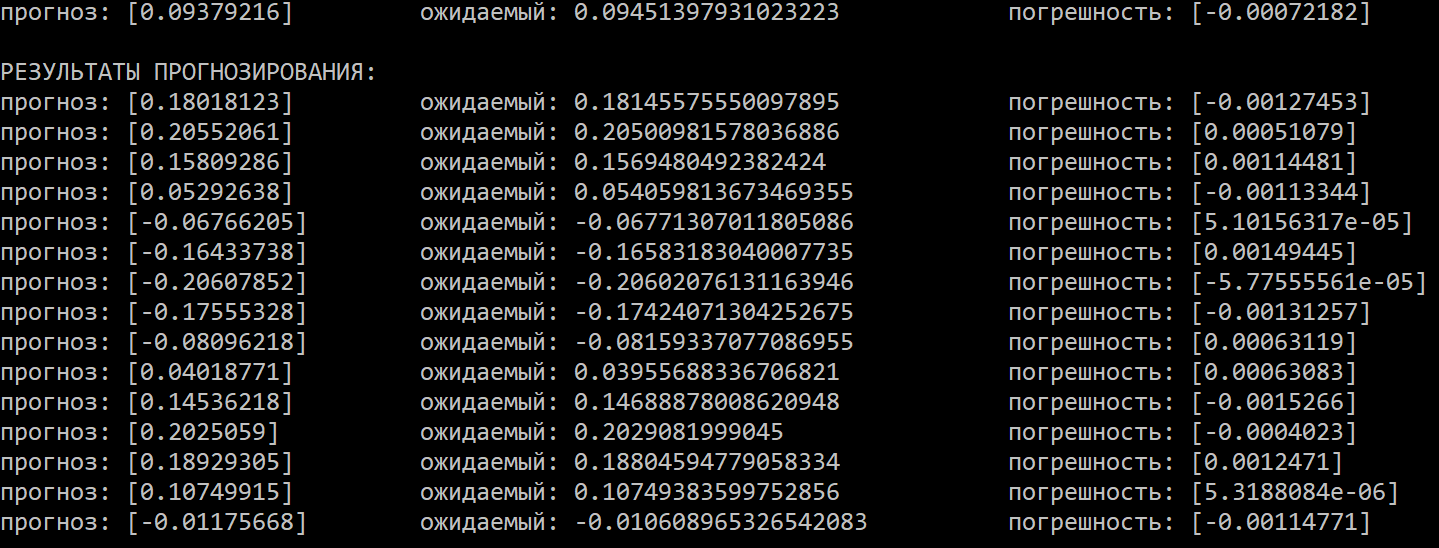
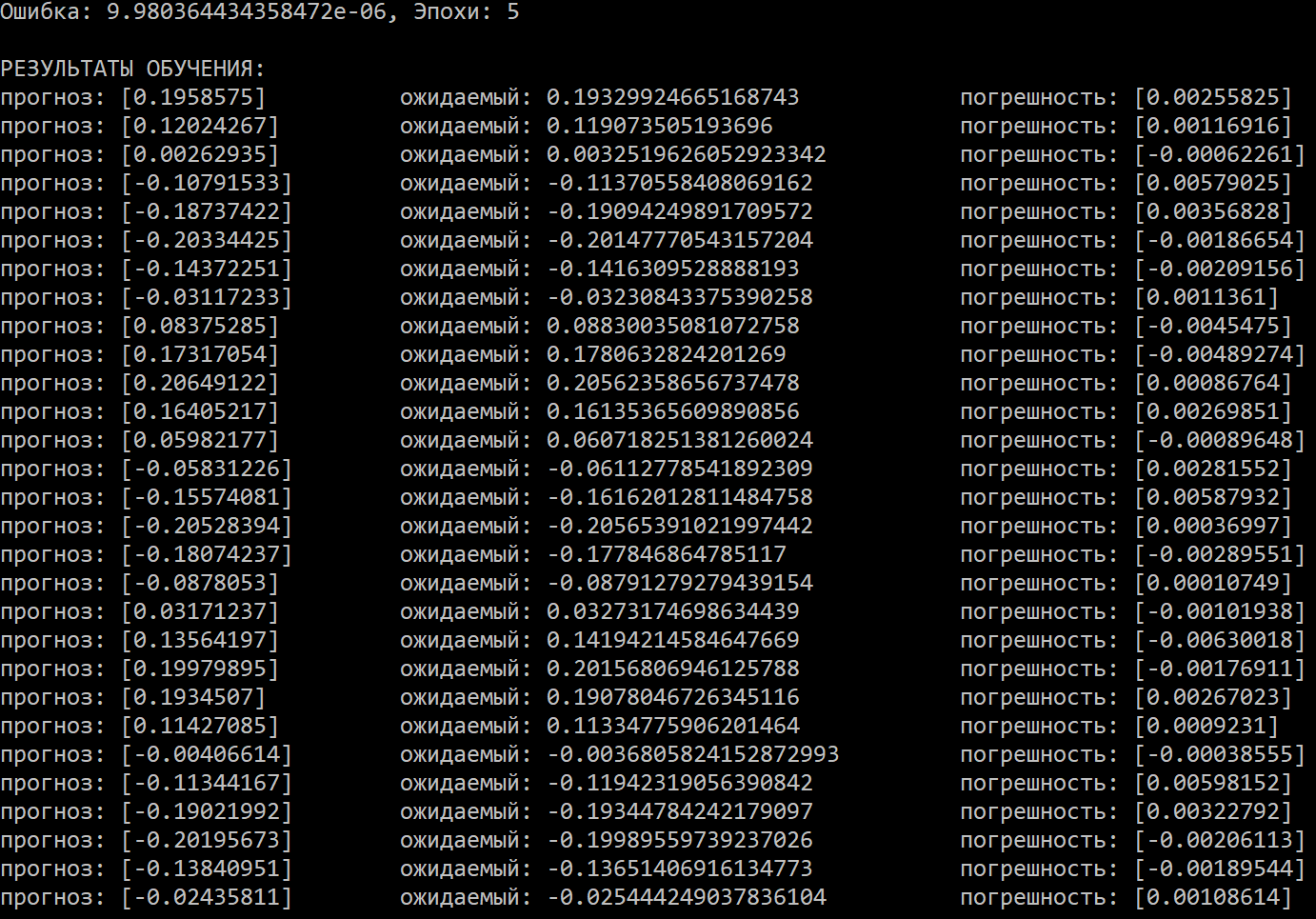
print("\nРЕЗУЛЬТАТЫ ПРОГНОЗИРОВАНИЯ:")

for sample,predict in predictions:

output = prediction(sample,w\_hidden,w\_input)

print("прогноз: {:<20} ожидаемый: {:<30} погрешность: {:<20}".format(str(output),str(np.array(predict)),str(output - predict)))

Результаты:



Вывод: В ходе выполнения работы спроектировал линейную ИНС с использованием адаптивного шага обучения.