Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

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 №5**

Нелинейные ИНС в задачах прогнозирования

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

**Вариант 6**

**Задание:**

Написать на любом ЯВУ программу моделирования нелинейной ИНС для распознавания образов. Рекомендуется использовать сигмоидную функцию, но это не является обязательным. Количество НЭ в скрытом слое взять согласно варианту работы №3. Его можно варьировать, если сеть не обучается или некорректно функционирует.

**Код программы:**

import itertools

import numpy as np

def relu(x):

return np.where(x <= 0,0,1)

return np.maximum(0,x)

def drelu(x):

return np.where(x <= 0,0,1)

def sigmoid(x):

return np.tanh(x)

def dsigmoid(x):

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

def error(y,Y):

yy = y.ravel()

YY = Y.ravel()

return np.mean((yy.reshape(1,len(yy)) - YY.reshape(1,len(YY))) \*\* 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 = sigmoid\_mapper(inputs\_hidden)

#outputs\_hidden = relu\_mapper(inputs\_hidden)

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

outputs\_input = sigmoid\_mapper(inputs\_input)

#outputs\_input = relu\_mapper(inputs\_input)

error\_input = np.subtract(outputs\_input,predict)

gradient\_input = dsigmoid(outputs\_input)

#gradient\_input = drelu(outputs\_input)

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

for w,d in zip(weights\_input,delta\_input):

ww, dd = [], []

ww = w.reshape(1,len(w))

dd.append(d)

ww -= learning\_rate \* np.dot(dd,outputs\_hidden.reshape(1,len(outputs\_hidden)))

for w,d in zip(weights\_input,delta\_input):

ww, dd = [], []

ww = w.reshape(1,len(w))

dd.append(d)

error\_hidden = dd \* ww

gradient\_hidden = dsigmoid(outputs\_hidden)

#gradient\_hidden = drelu(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 = sigmoid\_mapper(inputs\_hidden)

#outputs\_hidden = relu\_mapper(inputs\_hidden)

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

outputs\_input = sigmoid\_mapper(inputs\_input)

#outputs\_input = relu\_mapper(inputs\_input)

return outputs\_input

sigmoid\_mapper = np.vectorize(sigmoid)

relu\_mapper = np.vectorize(relu)

learning = []

predictions = []

learning\_rate = 0.5

epoch = 0

epoch\_maximum = 15000

error\_minimum = 1e-5 # минимальная ошибка

n\_input = 20 # количество входов

n\_hidden = 10 # количество элементов скрытого слоя

n\_output = 8 # количество выходов

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

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

vectors = np.array([[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0],[1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0],[1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,1]])

codes = np.array([[0,0,0,0,0,1,0,0],[0,0,0,0,1,0,0,0],[0,0,0,0,0,0,0,1]])

for vector, code in zip(vectors, codes):

com = []

com.append(vector)

com.append(code)

learning.append(tuple(com))

while True:

inputs, predicts = [], []

for sample,predict in learning:

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

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

break

vvectors = np.array([[0,1,0,0,1,1,0,1,0,0,0,0,1,0,1,0,1,0,0,0],[0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0],[1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1],[1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0],[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1]])

ccodes = np.array([[1,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0],[0,0,1,0,0,0,0,0],[0,0,0,1,0,0,0,0],[0,0,0,0,0,0,1,0]])

for vector, code in zip(vvectors, ccodes):

com = []

com.append(vector)

com.append(code)

predictions.append(tuple(com))

print("\nРЕЗУЛЬТАТЫ ОБУЧЕНИЯ:")

for sample,predict in learning:

print(sample)

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

print("прогноз : {:<30}\nожидаемый: {:<30}\n".format(str(output),str(np.array(predict))))

print("\nРЕЗУЛЬТАТЫ ПРОГНОЗИРОВАНИЯ:")

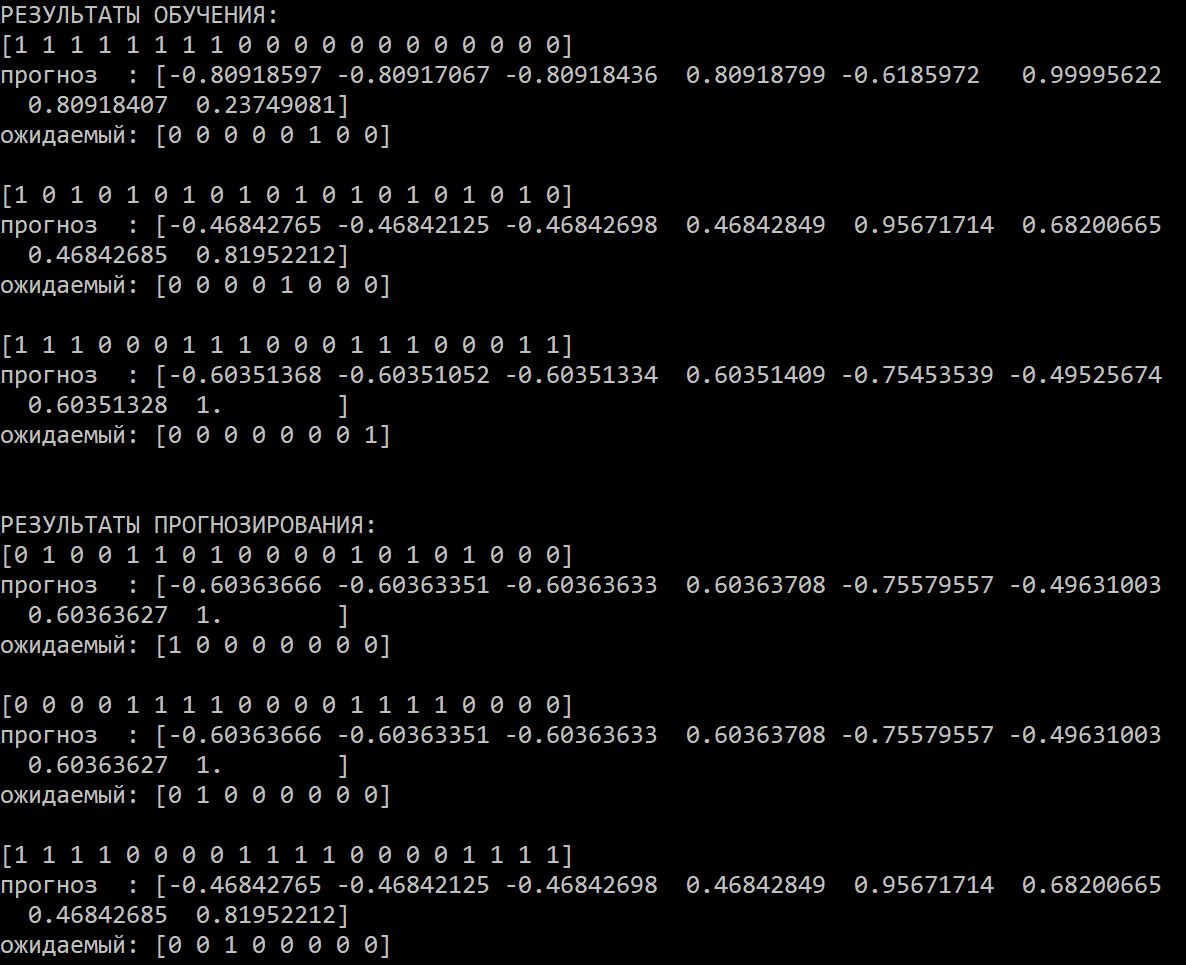
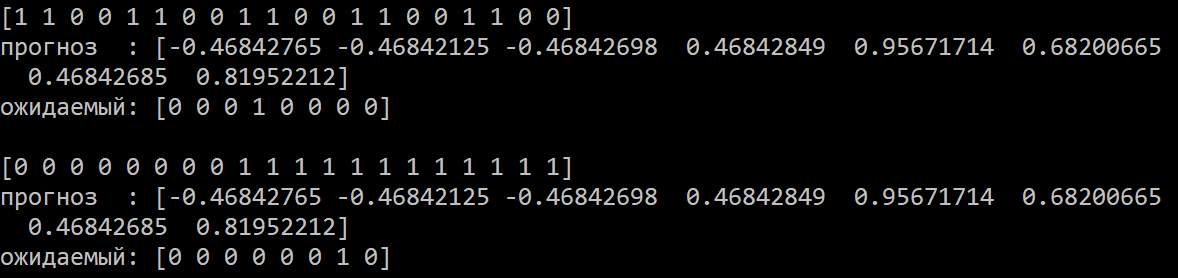
for sample,predict in predictions:

print(sample)

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

print("прогноз : {:<30}\nожидаемый: {:<30}\n".format(str(output),str(np.array(predict))))

Результаты:

Вывод: В ходе выполнения работы спроектировал нелинейную ИНС в задачах распознавания образов.