Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «МиАПР»

Тема: «Нелинейные ИНС в задачах прогнозирования. Адаптивный шаг обучения»

Выполнил:

Студент 2 курса

Группы ПО-7(2)

Якимчик Д.В.

Проверил:

Крощенко А.А.

2021

Вариант 1

**Цель работы:**

изучить обучение и функционирование линейной ИНС при решении

задач прогнозирования.

**Задание:**

Написать на любом ЯВУ программу моделирования прогнозирующей нелинейной ИНС. Для тестирования использовать функцию

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

a=0.1, b=0.1, c=0.05, d=0.1, кол-во входов ИНС = 6, кол-во НЭ в скрытом слое = 2.

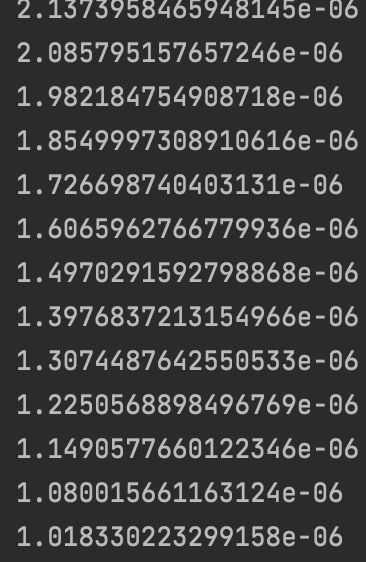
Для прогнозирования использовать многослойную ИНС с одним скрытым слоем. В качестве функций активации для скрытого слоя использовать сигмоидную функцию, для выходного - линейную.

**Код программы:**

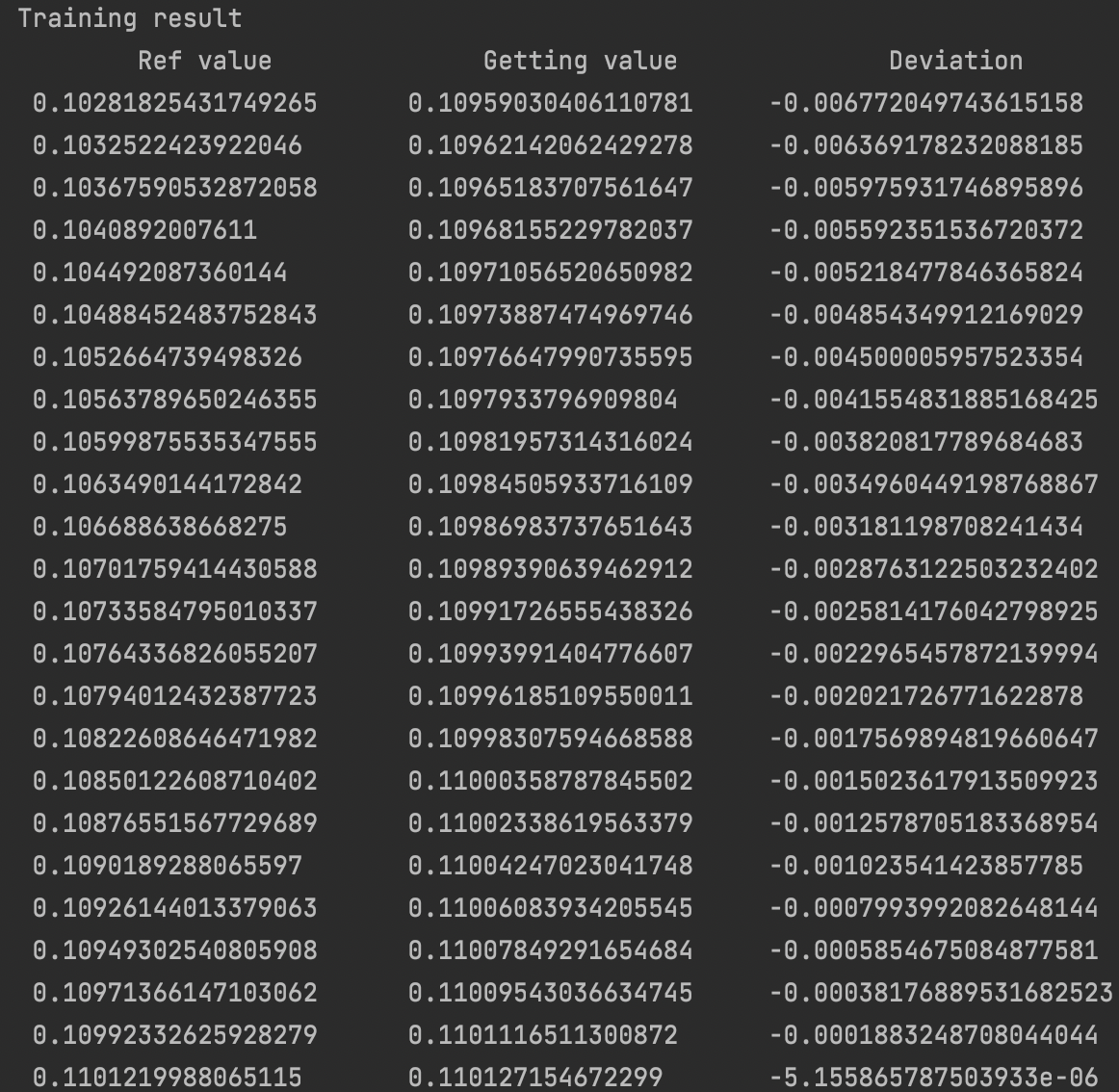
import random  
from math import sin, cos, exp  
  
  
def ref(x):  
 a, b, c, d = 0.1, 0.1, 0.05, 0.1  
 return a \* cos(b \* x) + c \* sin(d \* x)  
  
  
def calc\_S\_hid(y\_hid\_arr, w\_hid\_arr, ref\_arr, T\_hid\_arr, input\_nn, hidden\_nn, iter):  
 for i in range(len(y\_hid\_arr)):  
 y\_hid\_arr[i] = 0  
  
 for i in range(hidden\_nn):  
 for j in range(input\_nn):  
 y\_hid\_arr[i] += w\_hid\_arr[i][j] \* ref\_arr[iter + j]  
 y\_hid\_arr[i] -= T\_hid\_arr[i]  
 return y\_hid\_arr  
  
  
def calc\_y\_hid(S\_hid):  
 return 1 / 1 + exp(-S\_hid)  
  
  
def calc\_y(y\_hid\_arr, w\_out\_arr, T\_out, hidden\_nn):  
 y = 0  
 for i in range(hidden\_nn):  
 y += y\_hid\_arr[i] \* w\_out\_arr[i]  
 return y - T\_out  
  
  
def change\_w\_out(w\_out\_arr, gamma\_out, y\_hid\_arr, hidden\_nn, alpha\_out):  
 for i in range(hidden\_nn):  
 w\_out\_arr[i] -= alpha\_out \* gamma\_out \* y\_hid\_arr[i]  
 return w\_out\_arr  
  
  
def change\_T\_out(T\_out, gamma\_out, alpha\_out):  
 T\_out += alpha\_out \* gamma\_out  
 return T\_out  
  
  
def change\_w\_hid(  
 w\_hid\_arr, gamma\_hidden, y\_hid\_arr, ref\_arr, input\_nn, hidden\_nn, iter, alpha\_hid  
):  
 for i in range(hidden\_nn):  
 for j in range(input\_nn):  
 w\_hid\_arr[i][j] -= (  
 alpha\_hid  
 \* gamma\_hidden[i]  
 \* y\_hid\_arr[i]  
 \* (1 - y\_hid\_arr[i])  
 \* ref\_arr[j + iter]  
 )  
 return w\_hid\_arr  
  
  
def change\_T\_hid(T\_hid\_arr, gamma\_hidden, y\_hid\_arr, hidden\_nn, alpha\_hid):  
 for i in range(hidden\_nn):  
 T\_hid\_arr[i] += alpha\_hid \* gamma\_hidden[i] \* y\_hid\_arr[i] \* (1 - y\_hid\_arr[i])  
 return T\_hid\_arr  
  
  
def new\_alpha\_hid(y\_hid\_arr, y, gamma\_hidden):  
 numerator, denum, new\_alpha = 0, 0, 0  
 for i in range(len(y\_hid\_arr)):  
 numerator += (gamma\_hidden[i] \*\* 2) \* y\_hid\_arr[i] \* (1 - y\_hid\_arr[i])  
 denum += (  
 (gamma\_hidden[i] \*\* 2) \* (y\_hid\_arr[i] \*\* 2) \* ((1 - y\_hid\_arr[i]) \*\* 2)  
 )  
 new\_alpha = (4 \* numerator) / ((1 + y \*\* 2) \* denum)  
 return new\_alpha  
  
  
def main():  
 step = 0.1  
 input\_nn = 6  
 hidden\_nn = 2  
 train\_value = 30  
 test\_value = 15  
 ref\_arr = [ref(x \* step) for x in range(train\_value)]  
  
 w\_hid\_arr = []  
 for i in range(hidden\_nn):  
 row = []  
 for j in range(input\_nn):  
 row.append(random.uniform(0, 1))  
 w\_hid\_arr.append(row)  
 print(w\_hid\_arr)  
 T\_hid\_arr = [0 for i in range(hidden\_nn)]  
 y\_hid\_arr = [0 for i in range(hidden\_nn)]  
  
 w\_out\_arr = [random.uniform(0, 1) for i in range(hidden\_nn)]  
 T\_out = random.uniform(0, 1)  
  
 gamma\_out = 0  
 gamma\_hidden = [0 for i in range(hidden\_nn)]  
  
 epoch\_counter = 0  
 y = 0  
 err = 1  
 min\_err = 1e-6  
 while err > min\_err:  
 err = 0  
 for iter in range(train\_value - input\_nn):  
 y\_hid\_arr = calc\_S\_hid(  
 y\_hid\_arr, w\_hid\_arr, ref\_arr, T\_hid\_arr, input\_nn, hidden\_nn, iter  
 )  
 for i in range(hidden\_nn):  
 y\_hid\_arr[i] = calc\_y\_hid(y\_hid\_arr[i])  
  
 y = calc\_y(y\_hid\_arr, w\_out\_arr, T\_out, hidden\_nn)  
  
 gamma\_out = y - ref\_arr[iter + input\_nn]  
  
 for i in range(hidden\_nn):  
 gamma\_hidden[i] = gamma\_out \* y \* (1 - y) \* w\_out\_arr[i]  
  
 y\_hid\_arr\_sum\_square = 0  
 for i in range(len(y\_hid\_arr)):  
 y\_hid\_arr\_sum\_square += y\_hid\_arr[i] \*\* 2  
 alpha\_out = 1 / y\_hid\_arr\_sum\_square  
  
 alpha\_hid = new\_alpha\_hid(y\_hid\_arr, y, gamma\_hidden)  
  
 w\_out\_arr = change\_w\_out(  
 w\_out\_arr, gamma\_out, y\_hid\_arr, hidden\_nn, alpha\_out  
 )  
 T\_out = change\_T\_out(T\_out, gamma\_out, alpha\_out)  
 w\_hid\_arr = change\_w\_hid(  
 w\_hid\_arr,  
 gamma\_hidden,  
 y\_hid\_arr,  
 ref\_arr,  
 input\_nn,  
 hidden\_nn,  
 iter,  
 alpha\_hid,  
 )  
 T\_hid\_arr = change\_T\_hid(  
 T\_hid\_arr, gamma\_hidden, y\_hid\_arr, hidden\_nn, alpha\_hid  
 )  
  
 err += ((y - ref\_arr[iter + input\_nn]) \*\* 2) / 2  
 epoch\_counter += 1  
 err /= train\_value - input\_nn  
 if epoch\_counter % 10000 == 0:  
 print(err)  
  
 print("Training end\nTraining result")  
 print("{:^25}{:^25}{:^25}".format("Ref value", "Getting value", "Deviation"))  
 for iter in range(train\_value - input\_nn):  
  
 y\_hid\_arr = calc\_S\_hid(  
 y\_hid\_arr, w\_hid\_arr, ref\_arr, T\_hid\_arr, input\_nn, hidden\_nn, iter  
 )  
 for i in range(hidden\_nn):  
 y\_hid\_arr[i] = calc\_y\_hid(y\_hid\_arr[i])  
  
 y = calc\_y(y\_hid\_arr, w\_out\_arr, T\_out, hidden\_nn)  
 print(  
 "{:< 25}{:< 25}{:< 25}".format(  
 ref\_arr[iter + input\_nn],  
 y,  
 ref\_arr[iter + input\_nn] - y,  
 )  
 )  
  
 ref\_test\_arr = [  
 ref(x \* step) for x in range(train\_value - input\_nn, test\_value + train\_value)  
 ]  
 print("Testing result")  
 print("{:^25}{:^25}{:^25}".format("Ref value", "Getting value", "Deviation"))  
 for iter in range(test\_value):  
  
 y\_hid\_arr = calc\_S\_hid(  
 y\_hid\_arr, w\_hid\_arr, ref\_test\_arr, T\_hid\_arr, input\_nn, hidden\_nn, iter  
 )  
 for i in range(hidden\_nn):  
 y\_hid\_arr[i] = calc\_y\_hid(y\_hid\_arr[i])  
  
 y = calc\_y(y\_hid\_arr, w\_out\_arr, T\_out, hidden\_nn)  
 print(  
 "{:< 25}{:< 25}{:< 25}".format(  
 ref\_test\_arr[iter + input\_nn],  
 y,  
 ref\_test\_arr[iter + input\_nn] - y,  
 )  
 )  
  
  
main()

**Результат:**

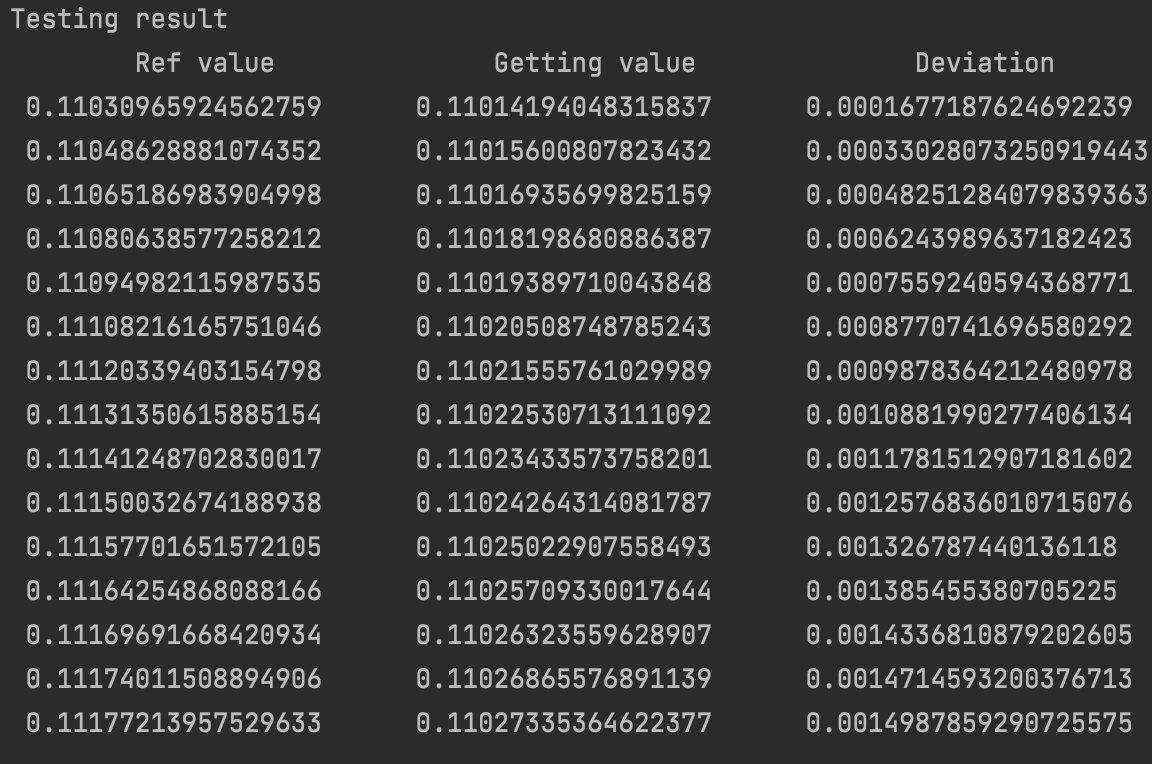
Вывод ошибки каждые 10000 эпох в процессе обучения:



**Результаты на обучающей выборке**

****

**Результаты на тестирующей выборке**

****

**Вывод:** В сравнении с константным шагом нейронная сеть с адаптивным шагом обучается немного быстрее. В среднем, адаптивный шаг обучения должен давать лучший результат(сеть с адаптивным шагом должна обучаться намного быстрее), однако, учитывая коэффициенты в исходной функции (a=0.1, b=0.1, c=0.05, d=0.1) и соответствующее изменение функции, это лучший полученный мною вариант.