Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

за 3 семестр

По дисциплине: «МиАПР»

Тема: «Нелинейные ИНС в задачах прогнозирования»

Выполнил:

Студент 2 курса

Группы ПО-4(2)

Луд А.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.

**Условие:**

Вариант 4

a = 0.4; b = 0.4; c = 0.08; d = 0.4;

Количество входов ИНС = 6;

Количество нейронных элементов в скрытом слое = 2;

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

y= a\*cos(bx) + c\*sin(dx)

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

***1. Адаптивный шаг обучения***

В стандартном алгоритме обратного распространения ошибки существует проблема выбора подходящего шага обучения, чтобы увеличить быстродействие и обеспечить сходимость алгоритма. Для выбора адаптивного шага обучения можно использовать метод наискорейшего спуска. В соответствии с ним, на каждой итерации обучения нейронной сети, необходимо выбирать шаг обучения для каждого слоя таким, чтобы минимизировать среднеквадратичную ошибку сети:

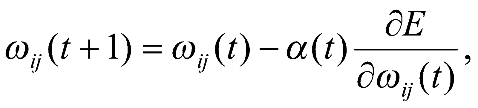
* (1)*

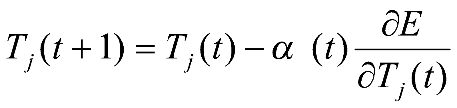
где j =  1,m, m - количество нейронных элементов последнего слоя.

Выходное значение j-го нейрона зависит от функции активации нейронных элементов и в общем случае определяется следующим образом :

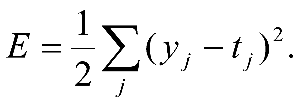
* (2)*

При этом весовые коэффициенты и пороги нейронной сети модифицируются, как

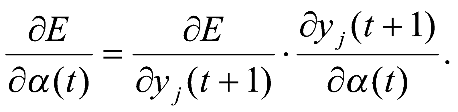
* (3)*

* (4)*

Среднеквадратичная ошибка нейронной сети равняется:

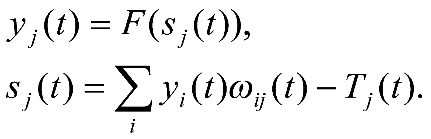
* (5)*

Тогда для нахождения α(t) необходимо решить следующее уравнение:

* (6)*

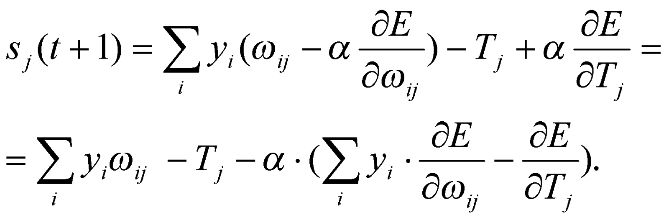
Данное уравнение невозможно решить относительно α(t) аналитическим путем. Поэтому в ряде работ для определения адаптивного шага обучения предлагается использовать методы линейного поиска. Однако это связано со значительными вычислениями. Поэтому можно предложить приближенный метод нахождения скорости обучения α(t). Он базируется на разложении функции активации нейронных элементов в ряд Тейлора. Рассмотрим это подробно.

Пусть выходное значение j-ого нейрона последнего слоя нейронной сети равняется :

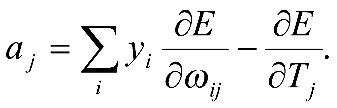
* (8)*

где yi (t) \_ выходное значение i-ого нейрона скрытого слоя.

Для определения взвешенной суммы j-ого нейрона в момент   
времени t+1 подставим в (8 ) выражения (3) и (4 ) :

* (9)*

Обозначим

* (10)*

Тогда выражение (9 ) можно представить в следующем виде :

* (11)*

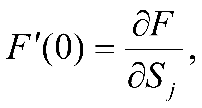
Выходное значение j-ого нейрона в момент времени t+1 равняется :

* (12)*

Разложим данное выражение по формуле Тейлора и ограничимся первыми двумя членами :

* (13)*

где



при *Sj=0.*

Подставим в (13 ) выражение (11 ). Тогда

* (14)*

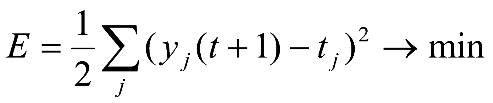
Так как

* (15)*

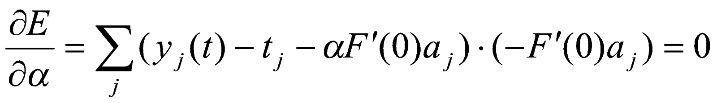
то выражение (14) можно представить в следующем виде:

* (16)*

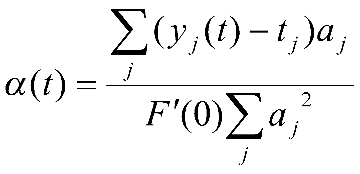
Для определения адаптивного шага обучения необходимо обеспечить :

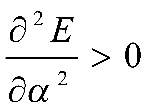
* (17)*

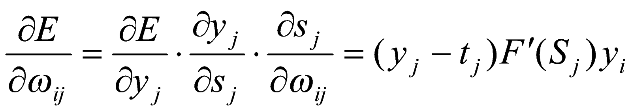
Тогда

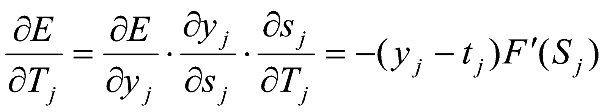


Выражая из последнего уравнения α(t) , получим :

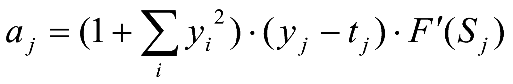
* (18)*

Так как , то при данном обеспечивается минимум   
среднеквадратичной ошибки. Найдем выражение для *аj*. Для этого определим

* (19)*

* (20)*

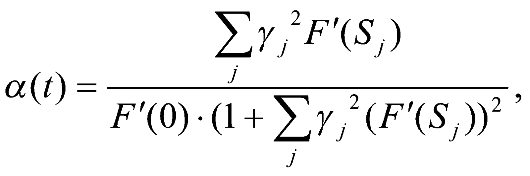
Подставляя (19) и (20) в выражение (10), получим:

* (21)*

Исходя из принципа независимости слоев, предполагаем, что

* (22)*

Подставляя выражения (22) и (21) в (18), получим приближенное выражение для вычисления адаптивного шага обучения различных слоев нейронной сети :

**

**Текст программы:**

#include <iostream>

#include <math.h>

#include <iomanip>

using namespace std;

double Func(double x);

double Sigm\_Func(double x);

double\* Hidden(double x, double w1[2][6], double T[2]);

double Print(double x, double w1[2][6], double w2[2], double T[2 + 1]);

double Adapt\_Step(double w2[], double err, double Print, double hiddens[]);

int main()

{

setlocale(LC\_ALL, "rus");

double w1[2][6],

w2[2],

T[2 + 1],

etalon,

current,

V = 0.4,

newV = 0.4,

x = 4,

Emin = 0.00002,

Emax = 0;

int era = 0,

max\_era = 1200;

for (int i = 0; i < 2; i++)

{

for (int k = 0; k < 6; k++)

{

w1[i][k] = ((double)rand() / RAND\_MAX) \* 0.005;

}

w2[i] = ((double)rand() / RAND\_MAX) \* 0.005;

T[i] = ((double)rand() / RAND\_MAX) \* 0.005;

}

T[4] = ((double)rand() / RAND\_MAX) \* 0.005;

do

{

Emax = 0;

for (int q = 0; q < 300; q++)

{

current = Print(x, w1, w2, T);

etalon = Func(x + 6 \* 0.1);

double err = current - etalon;

double\* hiddens = Hidden(x, w1, T);

for (int j = 0; j < 2; j++)

w2[j] -= V \* err \* hiddens[j];

T[4] += V \* err;

for (int k = 0; k < 2; k++)

{

for (int i = 0; i < 6; i++)

w1[k][i] -= newV \* Func(x + i \* 0.1) \* hiddens[k] \* (1 - hiddens[k]) \* w2[k] \* err;

T[k] += newV \* hiddens[k] \* (1 - hiddens[k]) \* w2[k] \* err;

}

newV = Adapt\_Step(w2, err, current, hiddens);

x += 0.1;

Emax += pow(err, 2);

}

Emax /= 2;

era++;

} while (Emax > Emin);

cout << "Emax= : " << Emax << endl;

cout << "Количество эпох: " << era << endl;

cout << setw(27) << left << "|Эталонные значения|" << setw(29) << left << "|Полученные значения|" << setw(30) << left << "|Отклонение|" << endl;

for (int i = 0; i < 100; i++)

{

double Resultat = Print(x, w1, w2, T), Etalon = Func(x + 6 \* 0.1);

cout << setw(27) << left << Etalon << setw(27) << left << Resultat << setw(30) << Resultat - Etalon << endl;

x += 0.1;

}

system("pause");

return 0;

}

double Func(double x) {

return 0.4 \* cos(0.4 \* x) + 0.08 \* sin(0.4 \* x);

}

double Sigm\_Func(double x) {

return 1 / (1 + pow(2, -x));

}

double\* Hidden(double x, double w1[2][6], double T[2]) {

double\* polychennoe = new double[2];

for (int i = 0; i < 2; i++)

polychennoe[i] = 0;

double input\_neuron[6];

for (int k = 0; k < 6; k++, x += 0.1)

input\_neuron[k] = Func(x);

for (int i = 0; i < 2; i++)

{

for (int k = 0; k < 6; k++)

polychennoe[i] += input\_neuron[k] \* w1[i][k];

polychennoe[i] -= T[i];

polychennoe[i] = Sigm\_Func(polychennoe[i]);

}

return polychennoe;

}

double Print(double x, double w1[2][6], double w2[2], double T[2 + 1])

{

double Resultat = 0;

double\* hidden\_neuron = Hidden(x, w1, T);

for (int j = 0; j < 2; j++) {

Resultat += hidden\_neuron[j] \* w2[j];

}

Resultat -= T[4];

return Resultat;

}

double Adapt\_Step(double w2[], double err, double Print, double hiddens[])

{

double V = 0, e1 = 0, e2 = 0;

for (int i = 0; i < 2; i++)

{

e1 += pow(err \* w2[i] \* (1 - hiddens[i]) \* hiddens[i], 2) \* hiddens[i] \* (1 - hiddens[i]);

e2 += pow(err \* w2[i] \* (1 - hiddens[i]) \* hiddens[i], 2) \* hiddens[i] \* hiddens[i] \* (1 - hiddens[i]) \* (1 - hiddens[i]);

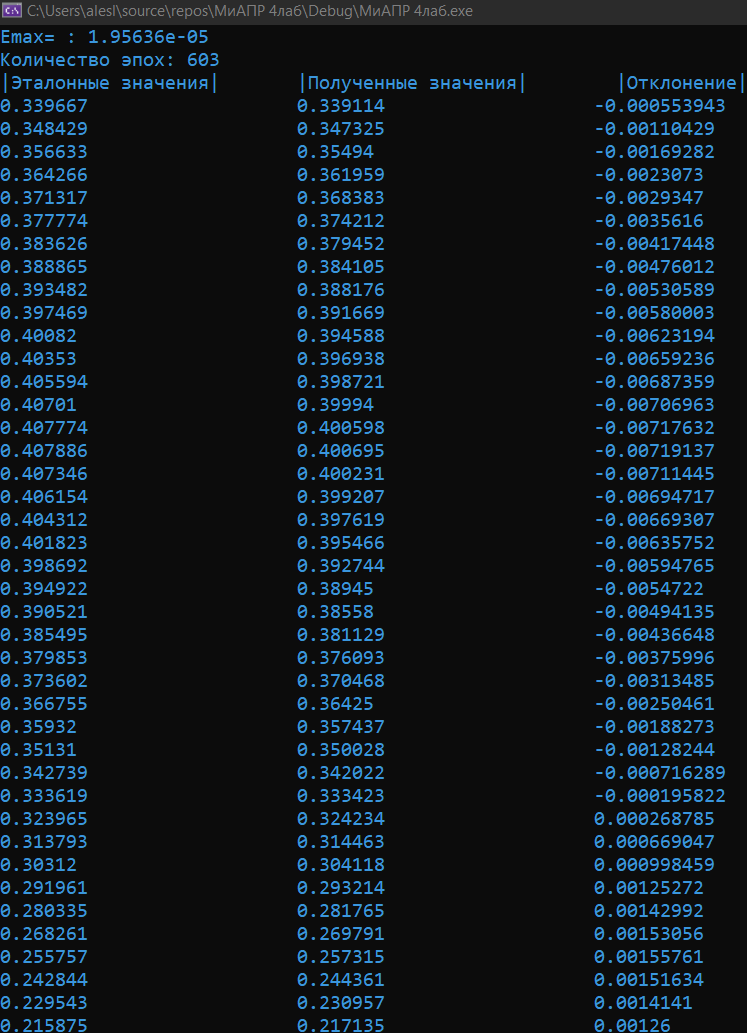
}

V = 4 \* e1 / (e2 \* (1 + Print \* Print));

return V;

}

**Результаты программы:**



**Вывод:** Изучить обучение и функционирование нелинейной ИНС при решении задач распознавания образов.