Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «МиАПР»

Тема: «Нелинейные ИНС в задачах прогнозирования»

Выполнила:

Студентка 2 курса

Группы ПО-7

Олехнович В. Н.

Проверил:

Крощенко А.А.

2021

Нелинейные ИНС в задачах прогнозирования

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

Вариант 2

Задание:

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| вариант | a | b | с | d | Кол-во входов ИНС | Кол-во НЭ в скрытом слое |
| 2 | 0.2 | 0.2 | 0.06 | 0.2 | 8 | 3 |

Код программы:

#include <iostream>

#include <cmath>

using namespace std;

double get\_random (double LO = -1.0, double HI = 1.0);

int main ()

{

const int random\_seed =1347652890;// Сюда пишем любое число, чтобы был разный рандом

const double a = 0.2; // Параметр y = a \* cos(b\*x) + c \* sin(d\*x)

const double b = 0.2; // Параметр y = a \* cos(b\*x) + c \* sin(d\*x)

const double c = 0.06; // Параметр y = a \* cos(b\*x) + c \* sin(d\*x)

const double d = 0.2; // Параметр y = a \* cos(b\*x) + c \* sin(d\*x)

const double step = 0.01; // x = step \* index

double x;

double y;

const int inputs = 8; // Количество нейронов входного слоя

const int hiddens = 3; // Количество нейронов скрытого слоя

const int outputs = 1;

double Ee = 1e-6; // Желаемая средне квадратичная ошибка

double E;

double alpha\_ki = 0.001; // Скорость обучения от входного слоя до скрытого

double alpha\_ij = 0.001; // Скорость обучения от скрытого слоя до выходного

double alpha\_sum1;

double alpha\_sum2;

double alpha\_sum3;

double alpha\_x2;

const int number\_learning = 30; // Количество для обучения

const int number\_test = 15; // Количество для тестов

double\* e = new double [number\_learning + number\_test + inputs]; // Эталоны

double\*\* w\_ki = new double\* [inputs]; // Веса от входного до скрытого слоя

for (int k = 0; k < inputs; ++k) w\_ki[k]= new double [hiddens];

double\*\* w\_ij = new double\* [hiddens]; // Веса от скрытого до выходного слоя

for (int i = 0; i < inputs; ++i) w\_ij[i]= new double [outputs];

double\* T\_i = new double [hiddens]; // Пороги скрытого слоя

double\* T\_j = new double [outputs]; // Пороги скрытого слоя

int era; // Количество эпох

int q;

int k; // Итератор для входного слоя

int i; // Итератор для скрытого слоя

int j; // Итератор для выходного слоя

double\* y\_k = new double [hiddens]; // Значения на входном слое (эталоны)

double\* S\_i = new double [hiddens]; // Взвешенная сумма для скрытого слоя

double\* y\_i = new double [hiddens]; // Значение на скрытом слое (сигмоидная функция)

double\* S\_j = new double [outputs]; // Взвешенная сумма для выходного слоя

double\* y\_j = new double [outputs]; // Значение на скрытом слое (линейная функция)

double\* j\_j = new double [outputs]; // Обратная ошибка выходного слоя

double\* j\_i = new double [hiddens]; // Обратная ошибка скрытого слоя

double dF\_i; // Переменная для вычисления обратной ошибки выходного слоя

double dF\_j; // Переменная для вычисления обратной ошибки скрытого слоя

// = = = = = = = = ~ ~ ~ ~ ~ ~ ~ ~ = = = = = = = = ~ ~ ~ ~ ~ ~ ~ ~ = = = = = = = =

system("chcp 65001");

srand(random\_seed);

printf (" Привет, мир! \n \n");

// = = = = = Создаем эталонные значения = = = = =

printf (" e = { \n");

for (i = 0; i < number\_learning + number\_test + inputs; ++i)

{

x = step \* i;

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

e[i] = y;

printf (" %4d: %20.16f , \n", i+1, e[i]);

}

printf (" } \n \n");

// = = = = = Генерируем веса от входного до скрытого слоя = = = = =

printf (" w\_ki = { \n");

for (k = 0; k < inputs; ++k)

{

printf (" %4d: { ", k+1);

for (i = 0; i < hiddens; ++i)

{

w\_ki[k][i] = get\_random();

printf (" %4d: %20.16f ,", i+1, w\_ki[k][i]);

}

printf (" } , \n");

}

printf (" } \n \n");

// = = = = = Генерируем веса от скрытого до выходного слоя = = = = =

printf (" w\_ij = { \n");

for (i = 0; i < hiddens; ++i)

{

printf (" %4d: { ", i+1);

for (j = 0; j < outputs; ++j)

{

w\_ij[i][j] = get\_random();

printf (" %4d: %20.16f ,", j+1, w\_ij[i][j]);

}

printf (" } , \n");

}

printf (" } \n \n");

// = = = = = Генерируем пороги для скрытого слоя = = = = =

printf (" T\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

T\_i[i] = get\_random();

printf (" %4d: %20.16f , \n", i+1, T\_i[i]);

}

printf (" } \n \n");

// = = = = = Генерируем пороги для выходного слоя = = = = =

printf (" T\_j = { \n");

for (j = 0; j < outputs; ++j)

{

T\_j[j] = get\_random();

printf (" %4d: %20.16f , \n", j+1, T\_j[j]);

}

printf (" } \n \n");

// = = = = = Правило Видроу-Хоффа = = = = =

for (era = 1; ; ++era)

{

for (q = 0; q < number\_learning - inputs; ++q)

{

// = = = = = Вычисляем значения на входном слое (взяли эталоны) = = = = =

// printf (" y\_k = { \n");

for (k = 0; k < inputs; ++k)

{

y\_k[k] = e[q + k];

// printf (" %4d: %20.16f , \n", k+1, y\_k[k]);

}

// printf (" } \n \n");

// = = = = = Вычисляем взвешенную сумму скрытого слоя = = = = =

// printf (" S\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

S\_i[i] = 0;

for (k = 0; k < inputs; ++k)

{

S\_i[i] += y\_k[k] \* w\_ki[k][i];

}

// printf (" %4d: %20.16f , \n", i+1, S\_i[i]);

S\_i[i] -= T\_i[i];

// printf (" %4d: %20.16f , \n", i+1, S\_i[i]);

}

// printf (" } \n \n");

// = = = = = Вычисляем значения на скрытом слое (сигмоидная функция) = = = = =

// printf (" y\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

y\_i[i] = 1 / (1 + pow(-S\_i[i] ,2));

// printf (" %4d: %20.16f , \n", i+1, y\_i[i]);

}

// printf (" } \n \n");

// = = = = = Вычисляем взвешенную сумму выходного слоя = = = = =

// printf (" S\_j = { \n");

for (j = 0; j < outputs; ++j)

{

S\_j[j] = 0;

for (i = 0; i < hiddens; ++i)

{

S\_j[j] += y\_i[i] \* w\_ij[i][j];

}

// printf (" %4d: %20.16f , \n", j+1, S\_j[j]);

S\_j[j] -= T\_j[j];

// printf (" %4d: %20.16f , \n", j+1, S\_j[j]);

}

// printf (" } \n \n");

// = = = = = Вычисляем значения на выходном слое (линейная функция) = = = = =

// printf (" y\_j = { \n");

for (j = 0; j < outputs; ++j)

{

y\_j[j] = S\_j[j];

// printf (" %4d: %20.16f , \n", j+1, y\_j[j]);

}

// printf (" } \n \n");

// = = = = = Вычисляем обратную ошибку для выходного слоя = = = = =

// printf (" j\_j = { \n");

for (j = 0; j < outputs; ++j)

{

j\_j[j] = y\_j[j] - e[q + inputs + j];

// printf (" %4d: %20.16f , \n", j+1, j\_j[j]);

}

// printf (" } \n \n");

// = = = = = Вычисляем обратную ошибку для скрытого слоя = = = = =

dF\_j = 1;

// printf (" j\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

j\_i[i] = 0;

for (j = 0; j < outputs; ++j)

{

j\_i[i] += j\_j[j] \* dF\_j \* w\_ij[i][j];

}

// printf (" %4d: %20.16f , \n", i+1, j\_i[i]);

}

// printf (" } \n \n");

// 190 эпох - без alpha\_ki

// 182 эпохи - с alpha\_ki

// 92 эпохи - с alpha\_ki \* (-1)

// < < < < < Адаптивный шаг от входного до скрытого слоя (для сигмоидной функции)

alpha\_sum1 = 0;

alpha\_sum3 = 0;

for (j = 0; j < outputs; ++j)

{

alpha\_sum1 += pow(j\_j[j], 2) \* y\_j[j] \* (1 - y\_j[j]);

alpha\_sum3 += pow(j\_j[j], 2) \* pow(y\_j[j], 2) \* pow(1 - y\_j[j], 2);

}

// printf(" alpha\_sum1 = %20.16f \n", alpha\_sum1);

// printf(" alpha\_sum3 = %20.16f \n", alpha\_sum3);

alpha\_sum2 = 0;

for (i = 0; i < hiddens; ++i)

{

alpha\_sum2 += pow(y\_i[i], 2);

}

// printf(" alpha\_sum2 = %20.16f \n", alpha\_sum2);

alpha\_ki = 4 \* alpha\_sum1 / ( (1 + alpha\_sum2) \* alpha\_sum3 ); // адаптивный шаг для сигмоидной функции

// printf(" alpha\_ki = %20.16f \n", alpha\_ki);

// > > > > >

// < < < < < Адаптивнй шаг от скрытого до выходного слоя (для линейной функции)

alpha\_x2 = 0;

for (i = 0; i < hiddens; ++i)

{

alpha\_x2 += pow(e[q + i], 2);

}

alpha\_ij = 1 / (1 + alpha\_x2);

// > > > > >

// = = = = = Новые веса между скрытым и выходным слоем = = = = =

// printf (" w\_ij = { \n");

for (i = 0; i < hiddens; ++i)

{

// printf (" %4d: { ", i+1);

for (j = 0; j < outputs; ++j)

{

w\_ij[i][j] -= alpha\_ij \* j\_j[j] \* dF\_j \* y\_j[j];

// printf (" %4d: %20.16f ,", j+1, w\_ij[i][j]);

}

// printf (" } , \n");

}

// printf (" } \n \n");

// = = = = = Новые пороги для выходного слоя = = = = =

// printf (" T\_j = { \n");

for (j = 0; j < outputs; ++j)

{

T\_j[j] += alpha\_ij \* j\_j[j] \* dF\_j;

// printf (" %4d: %20.16f , \n", j+1, T\_j[j]);

}

// printf (" } \n \n");

// = = = = = Новые веса между входным и скрытым слоем = = = = =

// printf (" w\_ki = { \n");

for (k = 0; k < inputs; ++k)

{

// printf (" %4d: { ", k+1);

for (i = 0; i < hiddens; ++i)

{

dF\_i = y\_i[i] \* (1 - y\_i[i]);

w\_ki[k][i] -= alpha\_ki \* j\_i[i] \* dF\_i \* y\_i[i];

// printf (" %4d: %20.16f ,", i+1, w\_ki[k][i]);

}

// printf (" } , \n");

}

// printf (" } \n \n");

// = = = = = Новые пороги для скрытого слоя = = = = =

// printf (" T\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

dF\_i = y\_i[i] \* (1 - y\_i[i]);

T\_i[i] += alpha\_ki \* j\_i[i] \* dF\_i;

// printf (" %4d: %20.16f , \n", i+1, T\_i[i]);

}

// printf (" } \n \n");

// break;

}

// break;

// = = = = = Ошибка сети = = = = =

E = 0;

for (j = 0; j < outputs; ++j)

{

E += 0.5 \* pow(y\_j[j] - e[q + inputs + j], 2);

}

if (E < Ee)

{

printf (" era: %8d Error: %20.16f < %-20.16f \n", era, E, Ee);

break;

}

printf (" era: %8d Error: %20.16f > %-20.16f \n", era, E, Ee);

}

printf(" \n \n");

// return 0;

// = = = = = Веса от входного до скрытого слоя после обучения = = = = =

printf (" w\_ki = { \n");

for (k = 0; k < inputs; ++k)

{

printf (" %4d: { ", k+1);

for (i = 0; i < hiddens; ++i)

{

printf (" %4d: %20.16f ,", i+1, w\_ki[k][i]);

}

printf (" } , \n");

}

printf (" } \n \n");

// = = = = = Веса от скрытого до выходного слоя после обучения = = = = =

printf (" w\_ij = { \n");

for (i = 0; i < hiddens; ++i)

{

printf (" %4d: { ", i+1);

for (j = 0; j < outputs; ++j)

{

printf (" %4d: %20.16f ,", j+1, w\_ij[i][j]);

}

printf (" } , \n");

}

printf (" } \n \n");

// = = = = = Пороги для скрытого слоя после обучения = = = = =

printf (" T\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

printf (" %4d: %20.16f , \n", i+1, T\_i[i]);

}

printf (" } \n \n");

// = = = = = Пороги для выходного слоя после обучения = = = = =

printf (" T\_j = { \n");

for (j = 0; j < outputs; ++j)

{

printf (" %4d: %20.16f , \n", j+1, T\_j[j]);

}

printf (" } \n \n");

// = = = = = Результаты после обучения = = = = =

printf(" Результат после обучения на обученной выборке \n");

printf(" %-8s %-20s %-20s %-20s %-20s \n",

"#",

"etalon",

"prognoz",

"otclonenie",

"otclonenie^2"

);

for (q = 0; q < number\_learning; ++q)

{

// = = = = = Вычисляем значения на входном слое (взяли эталоны) = = = = =

// printf (" y\_k = { \n");

for (k = 0; k < inputs; ++k)

{

y\_k[k] = e[q + k];

// printf (" %4d: %20.16f , \n", k+1, y\_k[k]);

}

// printf (" } \n \n");

// = = = = = Вычисляем взвешенную сумму скрытого слоя = = = = =

// printf (" S\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

S\_i[i] = 0;

for (k = 0; k < inputs; ++k)

{

S\_i[i] += y\_k[k] \* w\_ki[k][i];

}

// printf (" %4d: %20.16f , \n", i+1, S\_i[i]);

S\_i[i] -= T\_i[i];

// printf (" %4d: %20.16f , \n", i+1, S\_i[i]);

}

// printf (" } \n \n");

// = = = = = Вычисляем значения на скрытом слое (сигмоидная функция) = = = = =

// printf (" y\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

y\_i[i] = 1 / (1 + pow(-S\_i[i] ,2));

// printf (" %4d: %20.16f , \n", i+1, y\_i[i]);

}

// printf (" } \n \n");

// = = = = = Вычисляем взвешенную сумму выходного слоя = = = = =

// printf (" S\_j = { \n");

for (j = 0; j < outputs; ++j)

{

S\_j[j] = 0;

for (i = 0; i < hiddens; ++i)

{

S\_j[j] += y\_i[i] \* w\_ij[i][j];

}

// printf (" %4d: %20.16f , \n", j+1, S\_j[j]);

S\_j[j] -= T\_j[j];

// printf (" %4d: %20.16f , \n", j+1, S\_j[j]);

}

// printf (" } \n \n");

// = = = = = Вычисляем значения на выходном слое (линейная функция) = = = = =

// printf (" y\_j = { \n");

for (j = 0; j < outputs; ++j)

{

y\_j[j] = S\_j[j];

// printf (" %4d: %20.16f , \n", j+1, y\_j[j]);

}

// printf (" } \n \n");

printf(" %8d %20.16f %20.16f %20.16f %20.16f \n",

q + 1,

e[q + inputs],

y\_j[0],

e[q + inputs] - y\_j[0],

pow(e[q + inputs] - y\_j[0] ,2)

);

}

// = = = = = Результаты тестирования = = = = =

printf(" Результат после обучения на неизвесной выборке (выборка не участвовала в обучении) \n");

printf(" %-8s %-20s %-20s %-20s %-20s \n",

"#",

"etalon",

"y\_j",

"etalon - y\_j",

"(etalon - y\_j)^2"

);

for (q = 0; q < number\_test; ++q)

{

// = = = = = Вычисляем значения на входном слое (взяли эталоны) = = = = =

// printf (" y\_k = { \n");

for (k = 0; k < inputs; ++k)

{

y\_k[k] = e[q + number\_learning + k];

// printf (" %4d: %20.16f , \n", k+1, y\_k[k]);

}

// printf (" } \n \n");

// = = = = = Вычисляем взвешенную сумму скрытого слоя = = = = =

// printf (" S\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

S\_i[i] = 0;

for (k = 0; k < inputs; ++k)

{

S\_i[i] += y\_k[k] \* w\_ki[k][i];

}

// printf (" %4d: %20.16f , \n", i+1, S\_i[i]);

S\_i[i] -= T\_i[i];

// printf (" %4d: %20.16f , \n", i+1, S\_i[i]);

}

// printf (" } \n \n");

// = = = = = Вычисляем значения на скрытом слое (сигмоидная функция) = = = = =

// printf (" y\_i = { \n");

for (i = 0; i < hiddens; ++i)

{

y\_i[i] = 1 / (1 + pow(-S\_i[i] ,2));

// printf (" %4d: %20.16f , \n", i+1, y\_i[i]);

}

// printf (" } \n \n");

// = = = = = Вычисляем взвешенную сумму выходного слоя = = = = =

// printf (" S\_j = { \n");

for (j = 0; j < outputs; ++j)

{

S\_j[j] = 0;

for (i = 0; i < hiddens; ++i)

{

S\_j[j] += y\_i[i] \* w\_ij[i][j];

}

// printf (" %4d: %20.16f , \n", j+1, S\_j[j]);

S\_j[j] -= T\_j[j];

// printf (" %4d: %20.16f , \n", j+1, S\_j[j]);

}

// printf (" } \n \n");

// = = = = = Вычисляем значения на выходном слое (линейная функция) = = = = =

// printf (" y\_j = { \n");

for (j = 0; j < outputs; ++j)

{

y\_j[j] = S\_j[j];

// printf (" %4d: %20.16f , \n", j+1, y\_j[j]);

}

// printf (" } \n \n");

printf(" %8d %20.16f %20.16f %20.16f %20.16f \n",

q + number\_learning + 1,

e[q + number\_learning],

y\_j[0],

e[q + number\_learning] - y\_j[0],

pow(e[q + number\_learning] - y\_j[0] ,2)

);

}

printf(" \n");

printf(" %8s : %d \n", "seed", random\_seed);

printf(" %8s : %d \n", "era", era);

printf(" %8s : %-20.16f \n", "y\_j", y\_j[0]);

return (0);

}

double get\_random (double LO, double HI)

{

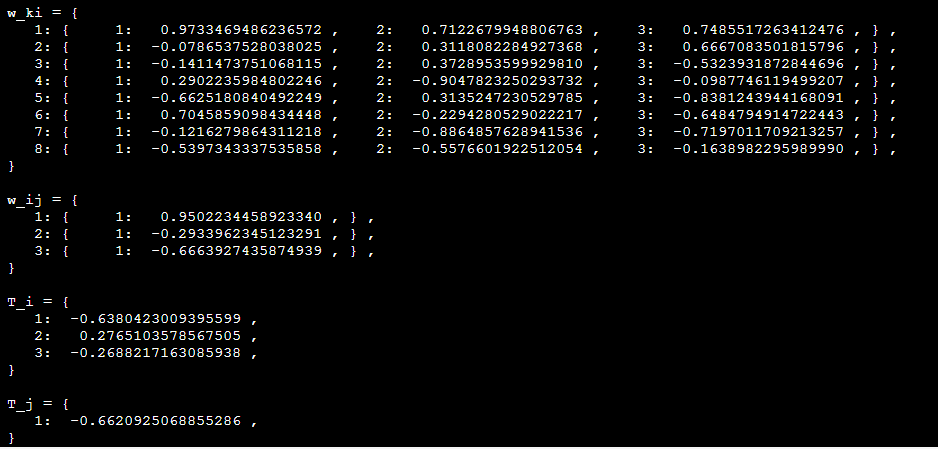
double rand\_number;

// = = = = = = = = ~ ~ ~ ~ ~ ~ ~ ~ = = = = = = = = ~ ~ ~ ~ ~ ~ ~ ~ = = = = = = = =

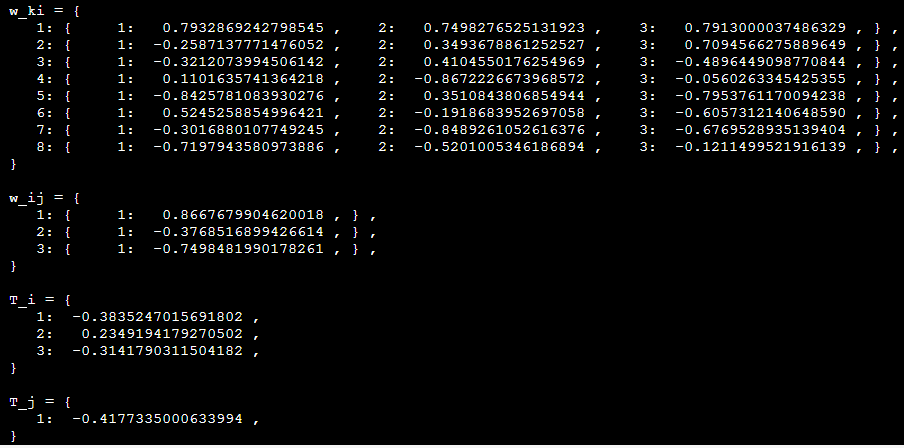
rand\_number = LO + static\_cast <float> (rand()) /( static\_cast <float> (RAND\_MAX/(HI-LO)));

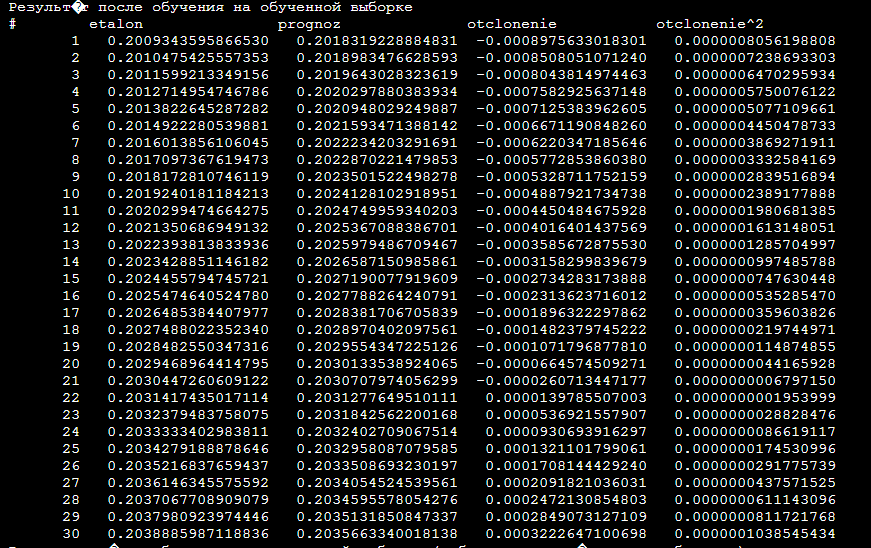
return (rand\_number);

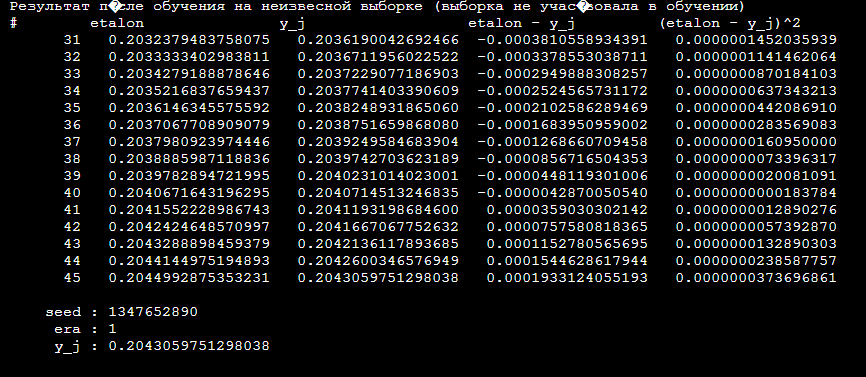
}



После обучения:







Вывод: Изучила алгоритм обучения многослойной ИНС с адаптивным шагом.