Neural Network for Even-n-Parity

Code (C #) :

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace ConsoleApplication1
{
   public class Network
   {
      private double[,] weight = new double[generation, genes];
      private const int genes = 30;
      private const int generation = 100;
      private int[, ] y = new int[100, 32];
      int[,] p = new int[100, 32];
      private int[] fitness = new int[100];
      private int[] e = new int[]
      {
         -1, 1, 1, -1, 1, -1, -1, 1, 1, -1, -1, 1, -1, 1, 1, -1, 1, -1, -1,1 -1, 1, 1, -1, -1, 1, 1, -1, 1,-1,-1, 1, 1
      };
      private int[,] inputs = new int [,] { { -1, -1, -1, -1, -1 },
         { -1, -1, -1, -1, 1 },
         { -1, -1, -1, 1, -1 },
      { -1, -1, -1, 1, 1 },

      { -1, -1, 1, -1, -1 },
      { -1, -1, 1, -1, 1 },
      { -1, -1, 1, 1, -1 },
      { -1, -1, 1, 1, 1 },

      { -1, 1, -1, -1, -1 },
      { -1, 1, -1, -1, 1 },
      { -1, 1, -1, 1, -1 },
      { -1, 1, -1, 1, 1 },

      { -1, 1, 1, -1, -1 },
      { -1, 1, 1, -1, 1 },
      { -1, 1, 1, 1, -1 },
      { -1, 1, 1, 1, 1 },

      { 1, -1, -1, -1, -1 },
      { 1, -1, -1, -1, 1 },
      { 1, -1, -1, 1, -1 },
      { 1, -1, -1, 1, 1 },

      { 1, -1, 1, -1, -1 },
      { 1, -1, 1, -1, 1 },
      { 1, -1, 1, 1, -1 },
      { 1, -1, 1, 1, 1 },

      { 1, 1, -1, -1, -1 },
```

```
{ 1, 1, -1, -1, 1 },
{ 1, 1, -1, 1, -1 },
{ 1, 1, -1, 1, 1 },

{ 1, 1, 1, -1, -1 },
{ 1, 1, 1, -1, 1 },
{ 1, 1, 1, 1, -1 },
{ 1, 1, 1, 1, 1 },
};

public Network()
{
   Random rand = new Random();
   for (int i = 0; i < generation; i++)
   {
      for (int j = 0; j < genes; j++)
      {
         weight[i, j] = rand.Next(-1, 2) * rand.NextDouble();
      }
   }

   int count = 0;
   for (int i = 0; i < 100; i++)
   {
      count = 0;
      for (int j = 0; j < 32; j++)
      {
         y[i, j] = Calculate(j);
            if (y[i, j] == e[j])
               count++;
      }
      fitness[i] = count;
   }

}

public int Calculate(int h)
{
   double result = 0;
   int[] res = new int[5];

   for (int i = 0; i < 5; i++)
   {
      result = 0;
      for (int j = 0; j < 30; j++)
      {
         result += inputs[h, i] * weight[i, j];
      }
      if (result >= 0)
         res[i] = 1;
      else
         res[i] = -1;
   }
   for (int i = 0; i < 5; i++)
```

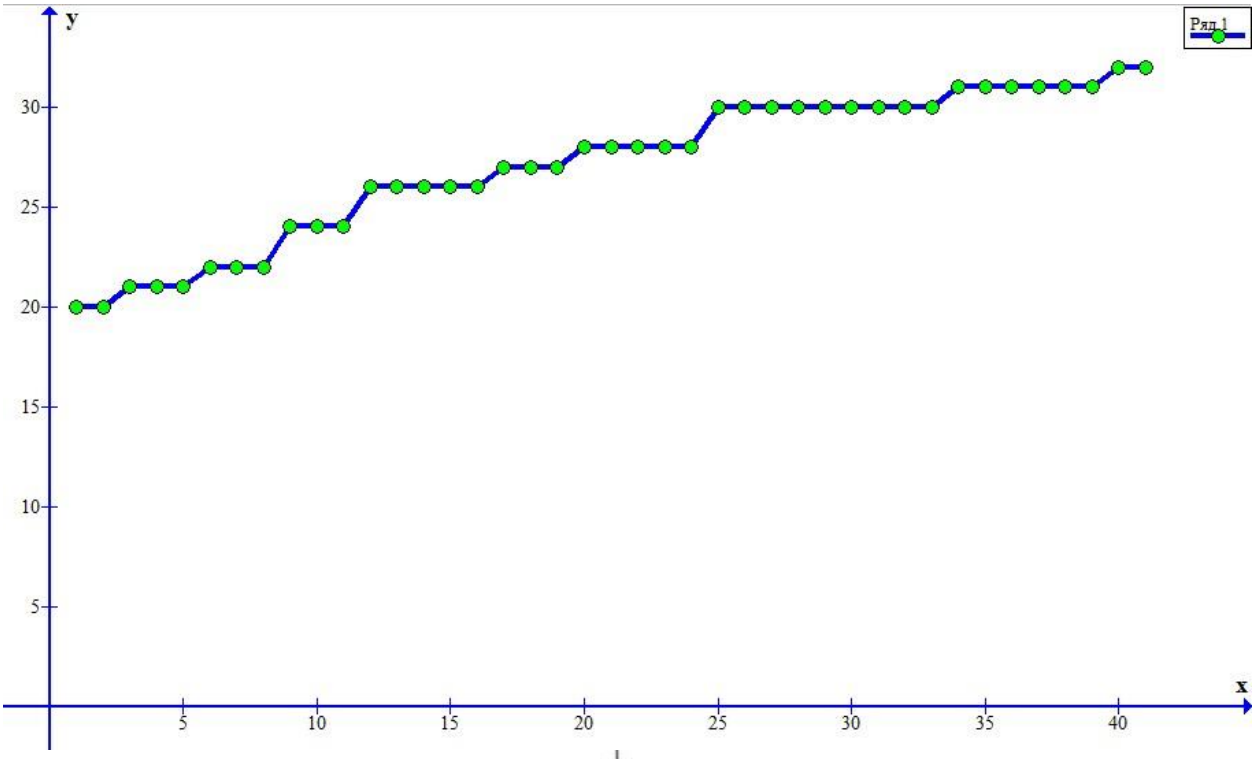
```
    {
        result = 0;
        for (int j = 0; j < 30; j++)
        {
            result += res[i] * weight[i, j];
        }
    }
    int r = 0;
    if (result >= 0)
        r= 1;
    else
        r = -1;

    return r;
}

public void Cross()
{
    int[] keys = new int[32];
    int[,] newPopulation = new int[100 / 2, 32];
    for (int i = 0; i < 32; i++)
        keys[i] = i;
    Array.Sort(fitness, keys);
    for (int i = 0; i < 100 / 2; i++)
    {
        for (int j = 0; j < 32; j++)
        {
            newPopulation[i, j] = y[keys[i], j];
        }
    }
    Random rand = new Random();
    for (int i = 0; i < 100; i++)
    {
        int a = rand.Next(0, 100/ 2);
        int b = rand.Next(0, 100 / 2);
        for (int j = 0; j < 32; j++)
        {
            p[0, j] = newPopulation[a, j];
            p[1, j] = newPopulation[b, j];
        }

        for (int j = 0; j < 32; j++)
        {
            int c = rand.Next(0, 2);
            if (c == 0)
            {
                y[i, j] = p[0, j];
            }
            else
            {
              y[i, j] = p[1, j];
            }
        }
    }
```

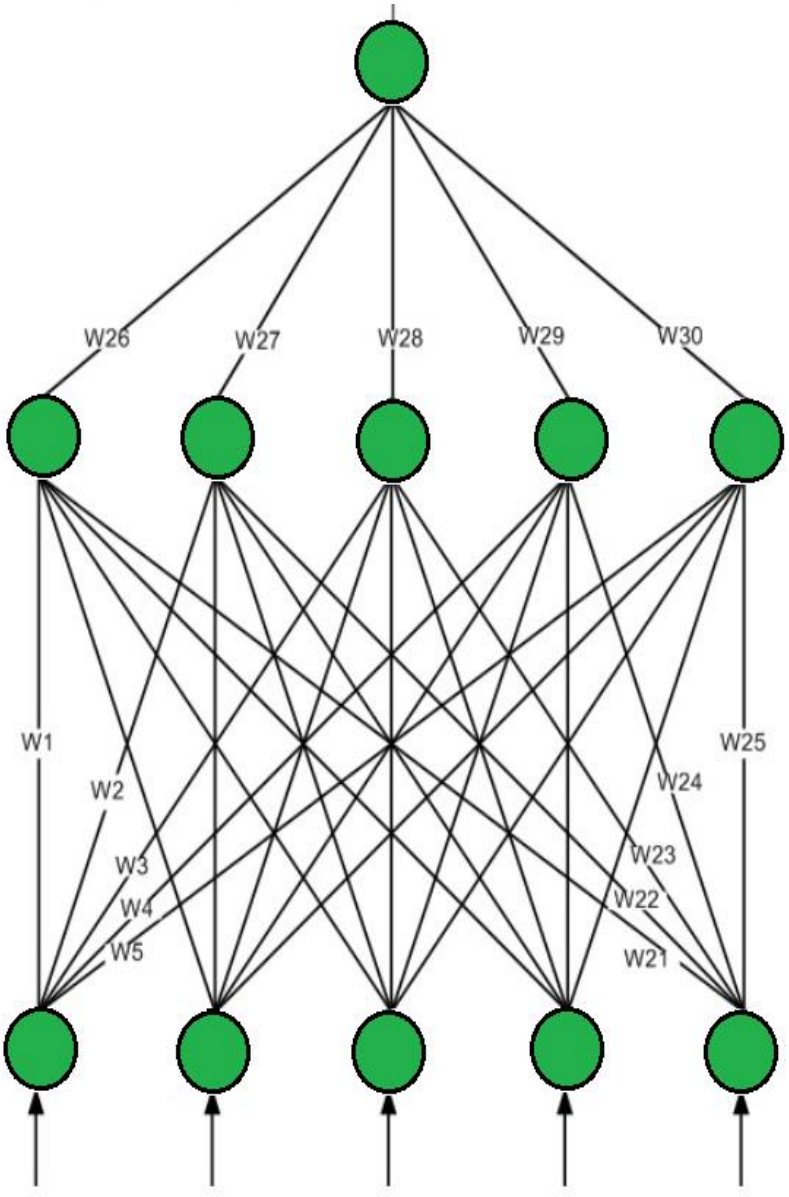
```
        }

        public int Get_fitness()
        {
            return fitness.Max();
        }

        public void Mutation()
        {
            Random rand = new Random();

            for (int i = 0; i < 100; i++)
            {
                for (int j = 0; j < 32; j++)
                {
                    int value = rand.Next(0, 32);
                    if (value == 0)
                    {
                        weight[i, j] = rand.Next(-1, 2) * rand.NextDouble();
                    }
                }
            }

            int count = 0;
            for (int i = 0; i < 100; i++)
            {
                count = 0;
                for (int j = 0; j < 32; j++)
                {
                    y[i, j] = Calculate(j);
                    if (y[i, j] == inputs[i, j])
                        count++;
                }
                fitness[i] = count;
            }
        }
    }
}


//**********************
using System;

namespace ConsoleApplication1
{
    class Program
    {
        public class Conditional
        {
            public static bool Condit(int max)
            {
                if (max == 32)
                {
                    return false;
```

```
            }
            else
               return true;
         }
      }

      static void Main(string[] args)
      {
         Network nw = new Network();
         int max = nw.Get_fitness();
         while (Conditional.Condit(max))
         {
            nw.Cross();
            nw.Mutation();
            max = nw.Get_fitness();
            Console.WriteLine(max);
         }
      }
   }
}
```

x1	x2	x3	x4	x5	y
1	1	1	1	1	-1
1	1	1	1	-1	1
1	1	1	-1	1	1
1	1	1	-1	-1	-1
1	1	-1	1	1	1
1	1	-1	1	-1	-1
1	1	-1	-1	1	-1
1	1	-1	-1	-1	1
1	-1	1	1	1	1
1	-1	1	1	-1	-1
1	-1	1	-1	1	-1
1	-1	1	-1	-1	1
1	-1	-1	1	1	-1
1	-1	-1	1	-1	1
1	-1	-1	-1	1	1
1	-1	-1	-1	-1	-1
-1	1	1	1	1	1
-1	1	1	1	-1	-1
-1	1	1	-1	1	-1
-1	1	1	-1	-1	1
-1	1	-1	1	1	-1
-1	1	-1	1	-1	1
-1	1	-1	-1	1	1
-1	1	-1	-1	-1	-1
-1	-1	1	1	1	-1
-1	-1	1	1	-1	1
-1	-1	1	-1	1	1
-1	-1	1	-1	-1	-1
-1	-1	-1	1	1	1
-1	-1	-1	1	-1	-1
-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	1

Weight	Value
w1	0,45993
w2	0,32654
w3	0,45738
w4	-0,43941
w5	-0,09348
w6	-0,32895
w7	-0,30501
w8	-0,12658
w9	0,24123
w10	0,52245
w11	-0,21497
w12	-0,72596
w13	0,10569
w14	-0,94342
w15	-0,21546
w16	0,86668
w17	-0,90866
w18	0,74833
w19	0,17638
w20	-0,25267
w21	-0,53186
w22	-0,09348
w23	-0,74685
w24	0,19476
w25	0,42447
w26	-0,89497


y
x
Ряд 1
30
25
20
15
10
5
5
10
15
20
25
30
35
40


W26
W27
W28
W29
W30
W1
W2
W3
W4
W5
W25
W24
W23
W22
W21

Generation	1	10	20	30	41
Fitness	20	24	28	30	32
	1	-1	-1	-1	-1
	1	1	1	1	1
	-1	1	-1	1	1
	1	1	-1	-1	-1
	1	-1	1	1	1
	1	1	1	1	-1
	-1	1	-1	-1	-1
	1	-1	1	1	1
	-1	1	-1	1	1
	1	-1	1	-1	-1
	-1	1	-1	1	-1
	1	1	1	1	1
	1	-1	1	-1	-1
	1	1	1	-1	1
	1	-1	-1	1	1
	-1	-1	-1	1	-1
	1	1	-1	1	1
	1	1	-1	-1	-1
	1	1	1	-1	-1
	-1	1	1	1	1
	-1	-1	1	-1	-1
	-1	1	1	1	1
	1	1	-1	1	1
	1	1	-1	-1	-1
	-1	-1	1	-1	-1
	1	1	1	1	1
	-1	-1	1	1	1
	1	-1	-1	-1	-1
	1	1	1	1	1
	1	1	1	-1	-1
	1	-1	1	1	-1
	-1	-1	1	1	1