Bakunovich Ivan

Source Code
VB.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace lab3_siit_golodko
{
    class GA
    {
        Random random;
        List<Population> historyOfPopulation;
        public GA()
        {
            random = new Random();
            List<Net> Nets = new List<Net>();
            for(int i=0;i<100;i++)
            {
                List<List<double>> w1 = new List<List<double>>();
                 List<double> w2 = new List<double>();
                for(int j=0;j<5;j++)
                {
                    w2.Add(2 * random.NextDouble() - 1);
                    List<double> tempW1 = new List<double>();
                    for (int t = 0; t < 5; t++)
                        tempW1.Add(2 * random.NextDouble() - 1);
                    w1.Add(tempW1);

                }
                Nets.Add(new Net(w1,w2));
            }
            Population startPopulation = new Population(Nets);
            historyOfPopulation = new List<Population>();
            historyOfPopulation.Add(startPopulation);
            int k = 0;
            while(historyOfPopulation[k].theBestFitness!=32)
            {
                Thread.Sleep(10);
                historyOfPopulation.Add(getNextPupulation(historyOfPopulation[k]));
                k++;
            }
        }
        private List<Net> population;
        private Population getNextPupulation(Population parentPopulation)
        {
            List<Net> childrenPopulationNets = new List<Net>();

            for (int i=0;i<50;i++)
            {
                List<Net> childrenNets = getChildren(
                    parentPopulation.Nets[random.Next() % 50+50],
                    parentPopulation.Nets[random.Next() % 50+50]
                    );
                childrenPopulationNets.AddRange(childrenNets);
            }
            for (int k = 0; k < childrenPopulationNets.Count; k++)
            {
                for (int i = 0; i < childrenPopulationNets[k].w1.Count; i++)
                {
                    for (int j = 0; j < childrenPopulationNets[k].w1[i].Count; j++)
                    {
                        int prob = random.Next(0, 1000);
                        if (prob == 777)
                        {
                            childrenPopulationNets[k].w1[i][j] = 2*random.NextDouble()+ 1;
                        }
                    }
                }
                for (int j = 0; j < childrenPopulationNets[k].w2.Count; j++)
                {
                    int prob = random.Next(0, 1000);
                    if (prob == 777)
                    {
                        childrenPopulationNets[k].w2[j] = 2 * random.NextDouble() + 1;
                    }
                }
```

```
            }
            //Sorting by fitness
            for (int i = 0; i < childrenPopulationNets.Count; i++)
            {
                for (int j = childrenPopulationNets.Count - 1; j > i; j--)
                {
                    if (childrenPopulationNets[j].fitness < childrenPopulationNets[j - 1].fitness)
                    {
                        Net tempNet = childrenPopulationNets[j];
                        childrenPopulationNets[j] = childrenPopulationNets[j - 1];
                        childrenPopulationNets[j - 1] = tempNet;
                    }
                }
            }
            Population childrenPopulation = new Population(childrenPopulationNets);
            return childrenPopulation;
        }
        Boolean isReady(List<Population> historyOfPopulation)
        {
            if (historyOfPopulation[historyOfPopulation.Count - 1].theBestFitness == 32)
                return true;
            else
                return false;
        }
        private List<Net> getChildren(Net father,Net mother)
        {
            List<Net> childrenNets = new List<Net>();
                Random random = new Random();
                List<Boolean> mask = new List<bool>();
            List<List<double>> firstW1 = new List<List<double>>();
            List<List<double>> secondW1 = new List<List<double>>();
            List<double> firstW2 = new List<double>();
            List<double> secondW2 = new List<double>();
            for(int i=0;i<father.w1.Count;i++)
            {
                for(int j=0;j< father.w1[i].Count;j++)
                {
                    List<double> firstTempW1 = new List<double>();
                    List<double> secondTempW1 = new List<double>();
                    if (random.Next() % 2 == 0)
                    {
                        firstTempW1.Add(father.w1[i][j]);
                        secondTempW1.Add(mother.w1[i][j]);
                    }
                    else
                    {
                        firstTempW1.Add(mother.w1[i][j]);
                        secondTempW1.Add(father.w1[i][j]);
                    }
                    father.w1.Add(firstTempW1);
                    mother.w1.Add(secondTempW1);
                }
            }
            for (int j = 0; j < father.w2.Count; j++)
            {
                if (random.Next() % 2 == 0)
                {
                    firstW2.Add(father.w2[j]);
                    secondW2.Add(mother.w2[j]);
                }
                else
                {
                    firstW2.Add(mother.w2[j]);
                    secondW2.Add(father.w2[j]);
                }
            }
                childrenNets.Add(new Net(firstW1,firstW2));
                childrenNets.Add(new Net(secondW1,secondW2));
            return childrenNets;
        }
    }
}
```

Net.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_golodko
{
```

```
    class Net
    {
        public List<double> etalonInput2;
        public List<double> etalonResult2;
        public List<List<double>> w1;
        public List<double> w2;
        public int fitness;
        double[,] etalonInput = new double[,]
        {
        {-1, -1, -1, -1, -1 },
        {-1, -1, -1, -1, 1},
        {-1, -1, -1, 1,  -1},
        {-1, -1, -1, 1,  1},
        {-1, -1, 1,  -1, -1},
        {-1, -1, 1,  -1, 1},
        {-1, -1, 1,  1,  -1},
        {-1, -1, 1,  1,  1},
        {-1, 1,  -1, -1, -1},
        {-1, 1,  -1, -1, 1},
        {-1, 1,  -1, 1,  -1},
        {-1, 1,  -1, 1,  1},
        {-1, 1,  1,  -1, -1},
        {-1, 1,  1,  -1, 1},
        {-1, 1,  1,  1,  -1},
        {-1, 1,  1,  1,  1},
        {1, -1, -1, -1, -1},
        {1, -1, -1, -1, 1},
        {1, -1, -1, 1,  -1},
        {1, -1, -1, 1,  1},
        {1, -1, 1,  -1, -1},
        {1, -1, 1,  -1, 1},
        {1, -1, 1,  1,  -1},
        {1, -1, 1,  1,  1},
        {1, 1,  -1, -1, -1},
        {1, 1,  -1, -1, 1},
        {1, 1,  -1, 1,  -1},
        {1, 1,  -1, 1,  1},
        {1, 1,  1,  -1, -1},
        {1, 1,  1,  -1, 1},
        {1, 1,  1,  1,  -1},
        {1, 1,  1,  1,  1}};
        double[] etalonResult = new double[]
        {1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0};
        public Net(List<List<double>> w1,List<double> w2)
        {
            this.w1 = w1;
            this.w2 = w2;
            fitness = getFitness(w1,w2);
        }
        private int getFitness(List<List<double>> w1, List<double> w2)
        {
            int count = 0;
            for (int k = 0; k < 32; k++)
            {
                List<double> tempLayout = new List<double>();
                for (int i = 0; i < w1.Count; i++)
                {
                    double tempWeight = 0;
                    for (int j = 0; j < 5; j++)
                    {
                        tempWeight += etalonInput[k,i]* w1[i][j];
                    }
                    tempLayout.Add(tempWeight);
                }
                double final = 0;
                for(int i=0;i<tempLayout.Count;i++)
                {
                    final += tempLayout[i] * w2[i];
                }
                double afterActivation;
                if (final > 0)
                    afterActivation= 1;
                else
                    afterActivation= - 1;
                if (afterActivation == etalonResult[k])
                    count++;
            }
            return count;
        }
    }
}
```

Population.cs

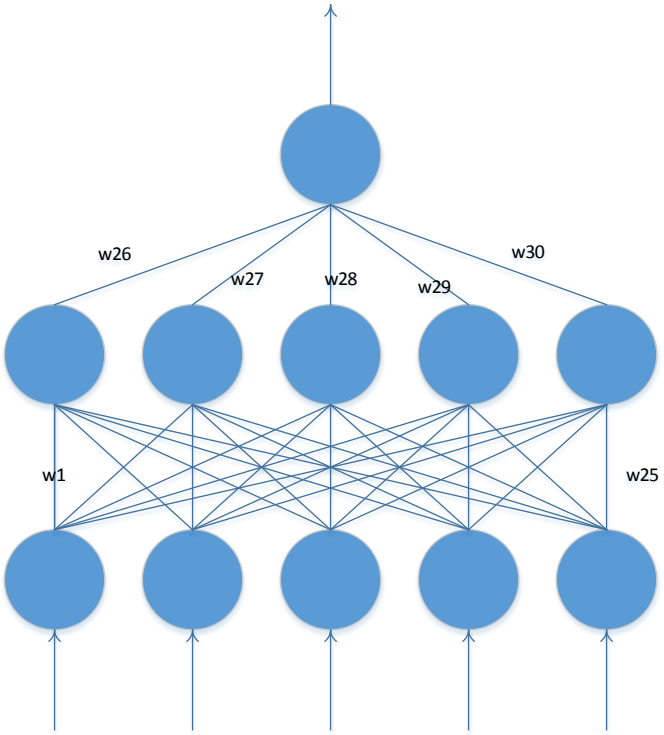
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace lab3_siit_golodko
{
    class Population
    {
        public List<Net> Nets;
        public int theBestFitness;
        public int avarageFitness;
        public Population(List<Net> Nets)
        {
            this.Nets = Nets;
            theBestFitness = getTheBestFitness(Nets);
            avarageFitness = getAvarageFitness(Nets);
        }
        public int getTheBestFitness(List<Net> Nets)
        {
            int bestFitness = 0;
            for (int i = 0; i < Nets.Count; i++)
                if (bestFitness < Nets[i].fitness)
                    bestFitness = Nets[i].fitness;
            return bestFitness;
        }
        public int getAvarageFitness(List<Net> Nets)
        {
            int avarageFitness = 0;
            for (int i = 0; i < Nets.Count; i++)
                avarageFitness += Nets[i].fitness;
            avarageFitness /= Nets.Count;
            return avarageFitness;
        }
    }
}
```

Referance values

-1	-1	-1	-1	-1	1
-1	-1	-1	-1	1	-1
-1	-1	-1	1	-1	-1
-1	-1	-1	1	1	1
-1	-1	1	-1	-1	-1
-1	-1	1	-1	1	1
-1	-1	1	1	-1	1
-1	-1	1	1	1	-1
-1	1	-1	-1	-1	-1
-1	1	-1	-1	1	1
-1	1	-1	1	-1	1
-1	1	-1	1	1	-1
-1	1	1	-1	-1	1
-1	1	1	-1	1	-1
-1	1	1	1	-1	-1
-1	1	1	1	1	1
1	-1	-1	-1	-1	-1
1	-1	-1	-1	1	1
1	-1	-1	1	-1	1
1	-1	-1	1	1	-1

1	-1	1	-1	-1	1
1	-1	1	-1	1	-1
1	-1	1	1	-1	-1
1	-1	1	1	1	1
1	1	-1	-1	-1	1
1	1	-1	-1	1	-1
1	1	-1	1	-1	-1
1	1	-1	1	1	1
1	1	1	-1	-1	-1
1	1	1	-1	1	1
1	1	1	1	-1	1
1	1	1	1	1	-1

Neural Network model



waight	value
w1	0,622
w2	0,854
w3	0,621
w4	0,912
w5	-0,899
w6	0,424
w7	0,789
w8	0,769
w9	0,913
w10	0,652
w11	0,7591

w12	0,587
w13	-0,515
w14	-0,794
w15	0,727
w16	0,037
w17	0,779
w18	0,386
w19	-0,222
w20	0,8545
w21	0,877
w22	-0,314
w23	-0,664
w24	-0,216
w25	0,327
w26	0,5881
w27	-0,521
w28	-0,731
w29	0,746
w30	0,757

generation	1	11	20	31	49
fitness	13	18	24	28	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

Graphic