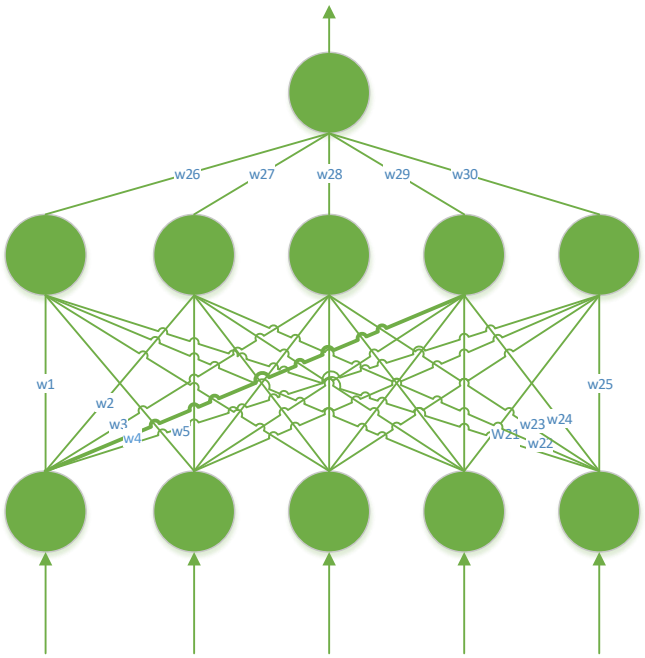
3-Ilya Babich

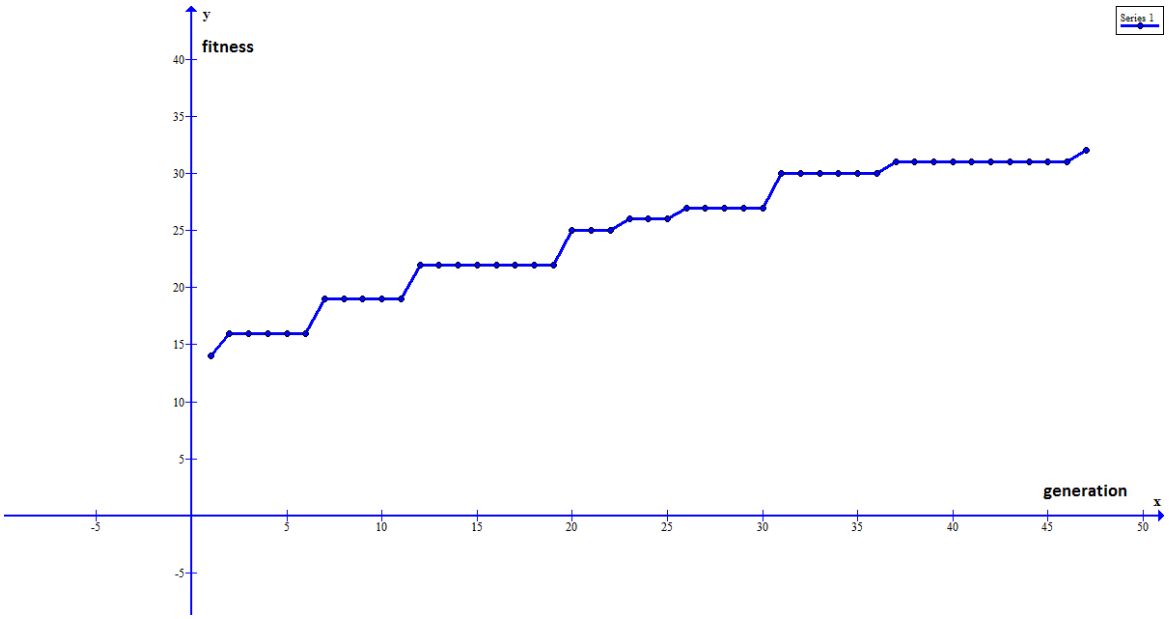
Neural network

raferance 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	0,632
w2	0,954
w3	0,611
w4	0,908
w5	-0,922
w6	0,427
w7	0,7989
w8	0,788
w9	0,91
w10	0,648
w11	0,759
w12	0,586
w13	-0,505
w14	-0,814
w15	0,726
w16	0,047
w17	0,799
w18	0,385
w19	-0,222
w20	0,855
w21	0,847
w22	-0,214
w23	-0,674
w24	-0,214
w25	0,347
w26	0,5771
w27	-0,511
w28	-0,732
w29	0,745
w30	0,856



generation	1	10	21	30	47
fitness	14	19	25	27	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

Source code:

```
namespace lab3_siit_babich
{
    class GA
    {
        Random random;
        List<Pop> histPop;
        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));
            }
```

```
        Pop startPop = new Pop(Nets);
        histPop = new List<Pop>();
        histPop.Add(startPop);
        int k = 0;
        while(histPop[k].theBestFitness!=32)
        {
            Thread.Sleep(10);
            histPop.Add(getNext(histPop[k]));
            k++;
        }
    }
    private List<Net> Pop;
    private Pop getNext(Pop parentPop)
    {
        List<Net> childrenPopNets = new List<Net>();

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

```
                childrenPopNets[k].w2[j] = 2 * random.NextDouble() + 1;
            }
        }
    }
    //Sorting by fitness
    for (int i = 0; i < childrenPopNets.Count; i++)
    {
        for (int j = childrenPopNets.Count - 1; j > i; j--)
        {
            if (childrenPopNets[j].fitness < childrenPopNets[j - 1].fitness)
            {
                Net tempNet = childrenPopNets[j];
                childrenPopNets[j] = childrenPopNets[j - 1];
                childrenPopNets[j - 1] = tempNet;
            }
        }
    }
    Pop childrenPop = new Pop(childrenPopNets);
    return childrenPop;
}
Boolean isReady(List<Pop> histPop)
{
    if (histPop[histPop.Count - 1].theBestFitness == 32)
        return true;
    else
        return false;
    //if (histPop.Count < 100)
    //    return false;
    //else
    //{
    //    for (int i = histPop.Count - 100; i < histPop.Count; i++)
    //    {
    //        if (histPop[histPop.Count - 100].avarageFitness !=
histPop[i].avarageFitness)
    //            return false;
    //    }
    //    return true;
    //}
}
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_babich
{
    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;
        }
    }
}
```

Pop.cs

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

namespace lab3_siit_babich
{
    class Pop
    {
        public List<Net> Nets;
        public int theBestFitness;
        public int avarageFitness;
        public Pop(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;
        }
    }
}
```