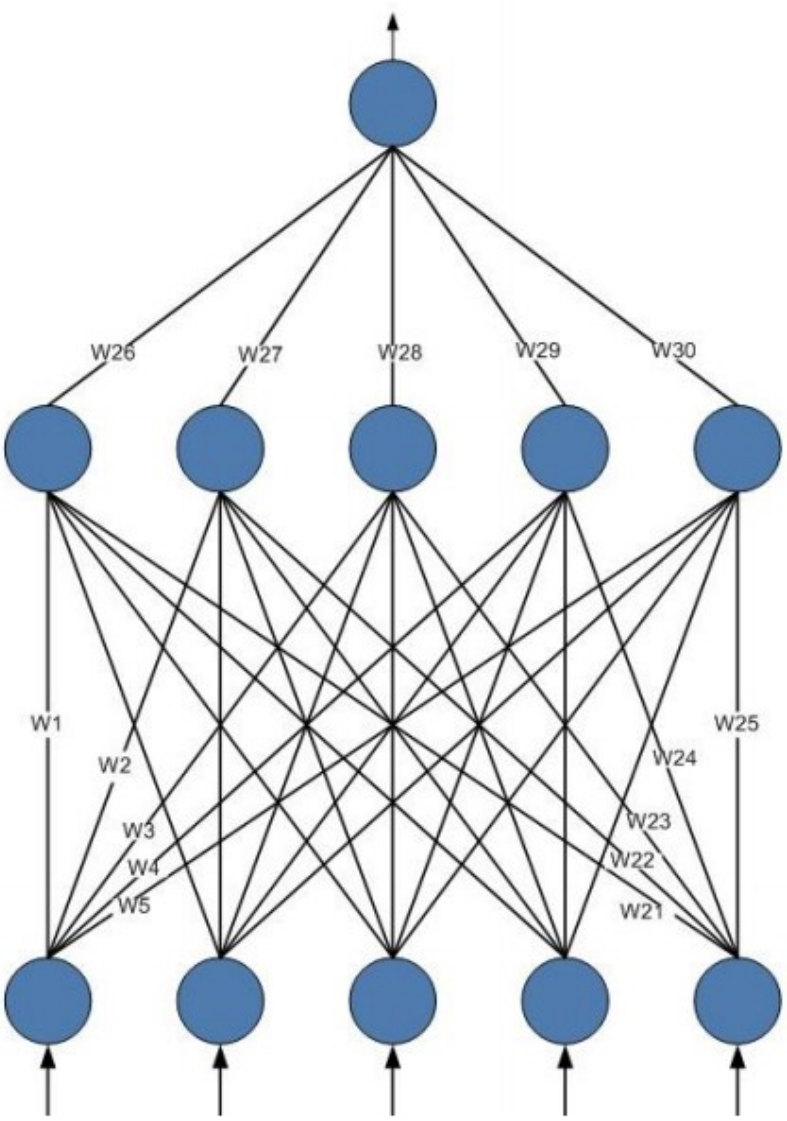
```java
import java.util.Arrays;
import java.util.Random;

public class Main
{
   static int INPUTS = 5;
   static Random rand = new Random(System.currentTimeMillis());


   static class Network
   {
      double[] values;     // hidden layer
      double[] weights;
      double result;
      int fitness;

      public Network(double[] weights)
      {
         this.values = new double[INPUTS];
         if(weights != null)
            this.weights = weights;
         else
         {
            this.weights = new double[INPUTS * (INPUTS + 1)];
            for(int i = 0; i < this.weights.length; i++)
               this.weights[i] = rand.nextDouble() * 2 - 1;
         }
         result = 0;
         fitness = 0;
      }

      int calculate(int[] inputData, int inputResult)
      {
         result = 0;
         Arrays.fill(values, 0);
         for(int i = 0; i < INPUTS; i++)
            for(int j = 0; j < INPUTS; j++)
               values[j] += inputData[i] * weights[INPUTS * i + j];
         for(int i = 0; i < INPUTS; i++)
            result += (values[i] >= 0 ? 1 : -1) * weights[INPUTS * INPUTS + i];
         if(result >= 0)
            result = 1;
         else
            result = -1;
         if((int) result == inputResult)
            fitness++;
         return (int) result;
      }
   }

   static class Generation
   {
      Network[] networks;

      public Generation(Network[] networks)
      {
         if(networks != null)
            this.networks = networks;
         else
         {
            this.networks = new Network[100];
            for(int i = 0; i < this.networks.length; i++)
               this.networks[i] = new Network(null);
         }
      }

      int calculateSortAndGetBestFitness()
```

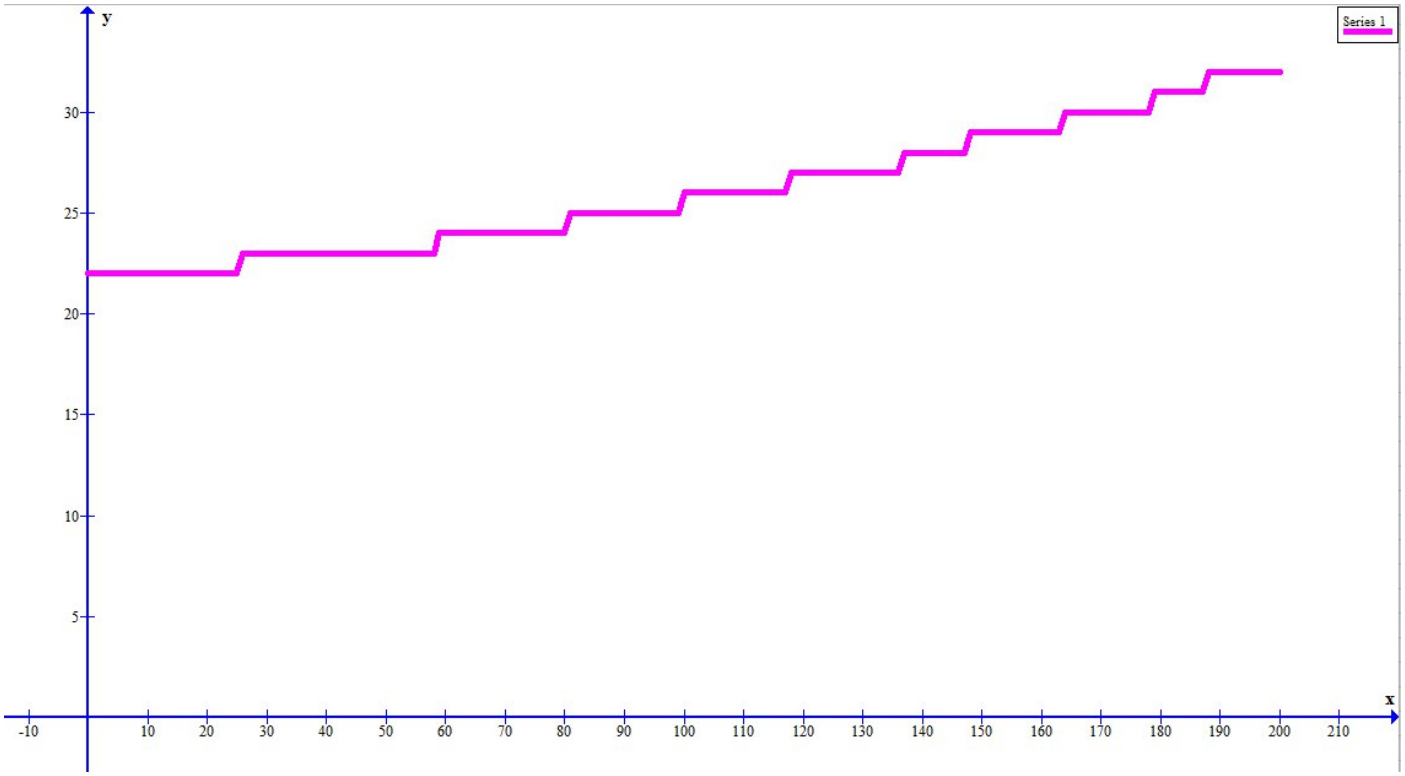
```
      {
         for(Network network : networks)
            for(int j = 0; j < 64; j++)
               network.calculate(Sample.getSample(j), Sample.getResult(j));

         Arrays.sort(networks, (o1, o2) ->
         {
            if(o1.fitness < o2.fitness)
               return 1;
            if(o1.fitness > o2.fitness)
               return -1;
            return 0;
         });
         System.out.println(networks[0].fitness);
         return networks[0].fitness;
      }

      Generation getNewGeneration()
      {
         Network[] newNetworks = new Network[networks.length];
         for(int i = 0; i < newNetworks.length; i++)
            newNetworks[i] = crossover(networks[rand.nextInt(50)],
networks[rand.nextInt(50)]);

         return new Generation(newNetworks);
      }
   }

   static Network crossover(Network first, Network second)
   {
      double[] weights = new double[first.weights.length];
      for(int i = 0; i < weights.length; i++)
         if(rand.nextInt(2) == 0)
            weights[i] = first.weights[i];
         else
            weights[i] = second.weights[i];
      for(int i = 0; i < weights.length; i++)
         if(rand.nextInt(42) == 1)
            weights[i] = rand.nextDouble() * 2 - 1;

      return new Network(weights);
   }

   public static void main(String[] args)
   {
      Generation generation = new Generation(null);
      while(generation.calculateSortAndGetBestFitness() != 64)
         generation = generation.getNewGeneration();
   }
}


package com.badmanners;


public class Sample
{
   static int[][] sample = 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}, KIRILL ZABRODSKY II-11 LAB-3 SIIT {1, 1,
-1, 1, 1}, {1, 1, 1, -1, -1}, {1, 1, 1, -1, 1}, {1, 1, 1, 1, -1}, {1, 1, 1, 1, 1}};
   static int[] result = 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};
```

```
    static int[] getSample(int i)
    {
        return sample[i];
    }
    static int getResult(int i)
    {
        return result[i];
    }
}
```

Network



1	0,69
2	0,81
3	-0,26
4	0,23
5	0,38
6	0,21
7	-0,79
8	-0,54
9	0,10
10	-0,93
11	0,02
12	-0,41
13	0,26
14	-0,80
15	0,27
16	0,38
17	0,17
18	-0,69
19	-0,64
20	1,00
21	0,53
22	-0,16
23	-0,85
24	-0,31
25	-0,37
26	-0,30
27	0,36
28	0,47
29	0,83
30	-0,67
31	0,09
32	-0,93



Fitness vs generation.

1	60	115	180	232	reference values
26	24	27	31	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	-1	-1
1	1	1	1	1	1
1	-1	-1	-1	-1	-1
-1	1	1	1	1	1
-1	1	1	1	1	1
1	-1	-1	-1	-1	-1