Simple Neural Network Simulation in C++

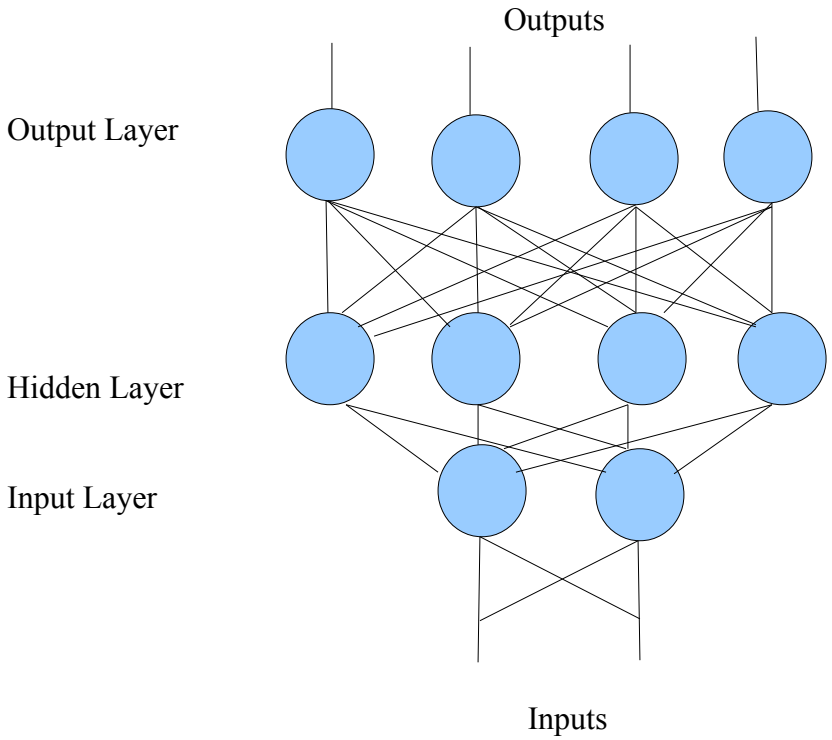
1.0 Implemented problems

The problems implemented in here are found from the link, *http://www.aijunkie.com/ann/evolved/nnt1.html*

1.1 Neural network

I used a nural network with two inputs and four outputs. Inputs are x and y lengths between the bot(big square) and the nearest food(small square). And the outputs which is in binary is decoded in to x and y speeds of the bot. Because of that the bots can get ether -2, -1, 1 or 2 as its x or y speed.

Number of hidden layers in a network can be change by the user. But there is only four bodes in each node. Output layer also has four nodes and input layer has two nodes.



example 1.0 : structure of a network with one hidden layer.

The structure node is used to hold information of a single node.

```
typedef struct node{
        int num_inputs;
        float *weights;
} node;
```

And the layer struct has the information on a layer with an array of nodes.

```
typedef struct layer{
        int num_nodes;
        node *chr;
} layer;
```

the class neural has all the functions and variables of an entire network.

```
class neural{
        private:
                int num_inputs;
                int num_outputs;
                int num_layers;
                int num_weights;
                layer *layers;
                int fitness;
                int total_fitness();
        public:
                nural(int in, int out, int num);
                int get_fitness();
                int inc_fitness();
                int get_num_weights();
                float *get_weights();
                void put_weights(float *weights);
                float* update(float *inputs);
                void dic_fitness();
                float sigmoid(float netinput, float response);
} ;
```

What a node in a neural network basically do add all the input after they are multiplied by a unice weight. Then a threshold value is reduced from it and convert that value in to a 0 or one using

activation function. Function used to convert sums to 1 or 0 is,

```
float nural::convert(float input){
        return ( 1 / ( 1 + exp(-input)));
}
```

In this simulation first thing done is to create a set of neural networks with random weights and runing them. The total number of food which one bot catch in given number of cycles. is passed as its fitness into a “genetic algorithm” and create next generation of bots. Then they are test as the first set. The simulation is repeating this process over and over.

1.2 Genetic Algorithm

The genetic algorithm class used is this,

```
class population{

    private:
        int size;
        int new_count;
        int b_fit;
        int w_fit;
        float avg_fit;
        int mutation_rate;
        int crossover_rate;
        int num;
        int *sums;
        int *fitness;
        float **chromosoms;
        float **chromosoms_new;
        int get_total();

    public:
        population(int s, int n, int cross, int mutation);
        float **genarate();
        int choose();
        void mutate(int i1, int rate);
        void cross_over(int i1, int i2, int rate);
        float** new_gen(int *fit);
        void operation(int *fit);
```

```
        int cal_b_fit();
        void cal_w_fit();
        float cal_avg_fit(int *fit);
};
```

First the genetic algorithm creates a set of random chromosomes (set of weights which can be used in a neural network). Then after running them in a simulation the fitnesses of them are passed as an array to the algorithm class. Two bots with highest fitnesses are copy without change to the new generation chromosome array. Then sets of two chromosomes are selected according to there fitnesses from the old set. Then they are crossed over and mutated according to the crossover and mutation rates and placed them in new generation array. Then that array is tested. This process keeps repeating in the simulation.

The bots which has more the average fitness in a generation is colored in red and others are colored in white.

2.0 Compiling source and running

You need X11 libraries to compile second problem. Run ,

sudo apt-get install libx11-dev

Then use make commend to compile each source. Then copy the “config” file in to the same directory which the source file compiled into.

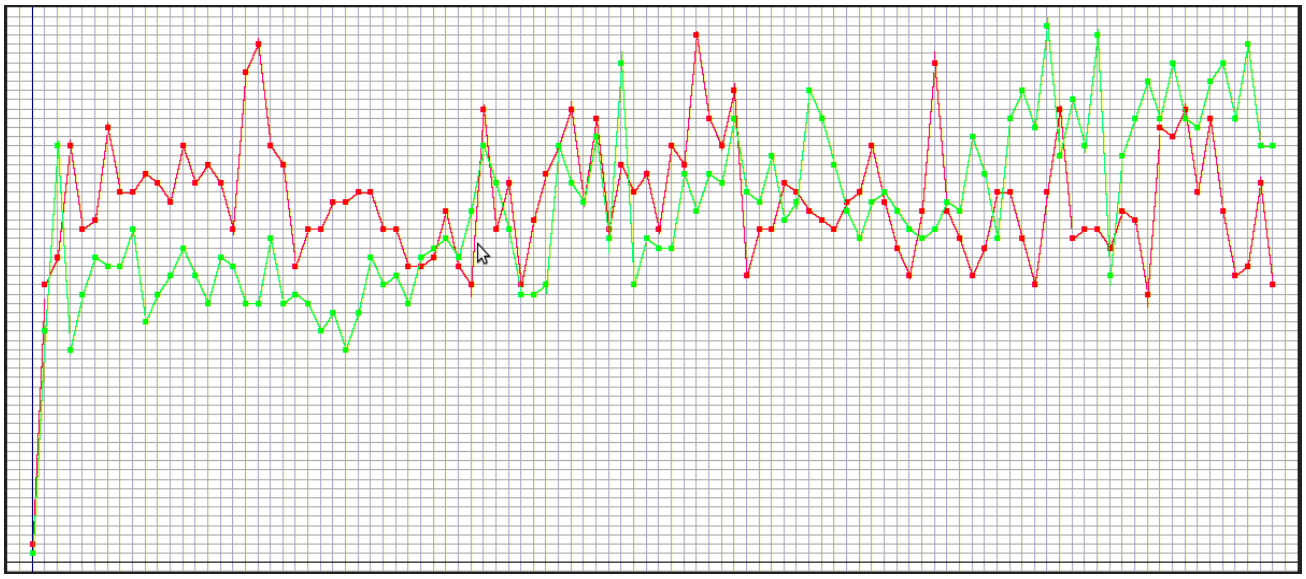
To run use command line.

./[executable]

You can edit the settings to run the simulation in different conditions. And the program will generate two text files containing best fitness in each generation and the total grabs done in a generation.

3.0 Example Results

graph 3.1 shows the best fitness in 0 to 100 generations. Red graph is with 3 layers and green graph is with 4 layers.

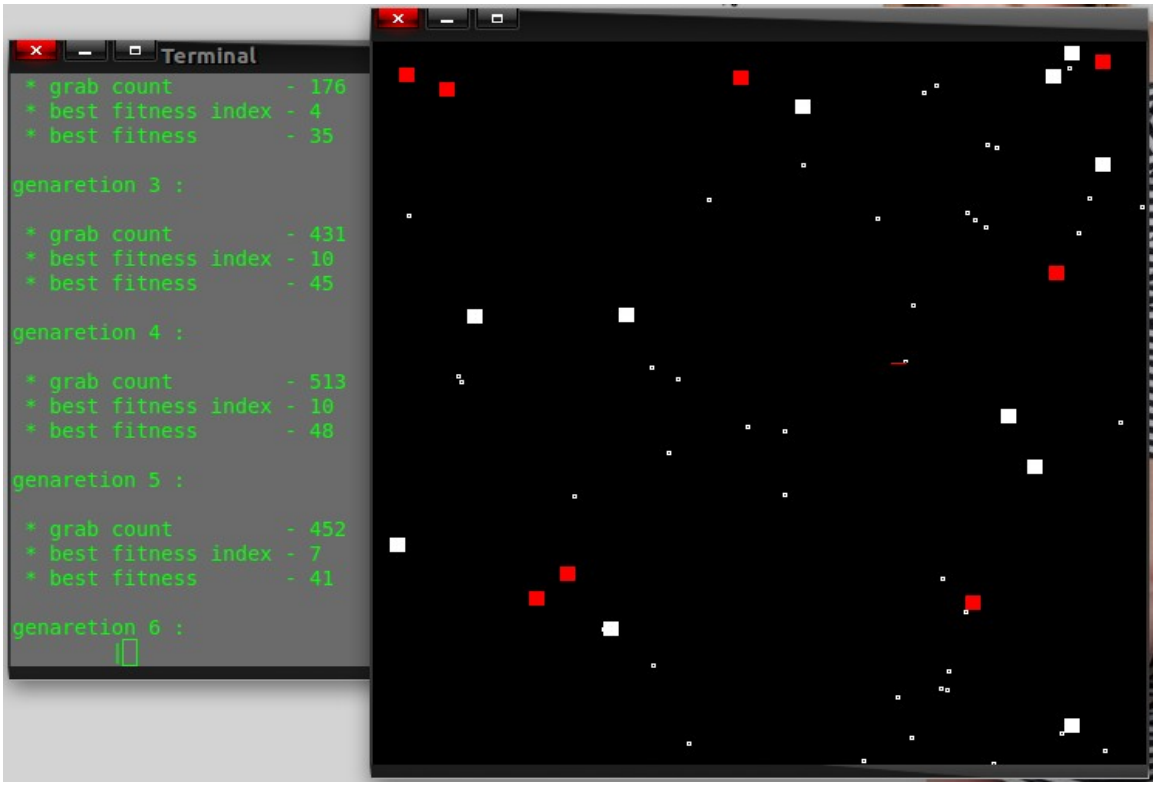


graph 3.1
(y unit = 1, x unit = 1, x axes is on y = 5, y axes is on x = 0)

graph 3.2 show the total number of food catch by bots in 100 generations. Red graph is with 3 layers and green graph is with 4 layers.

graph 3.2
(y unit = 10, x unit = 1, x axes is on y = 30, y axes is on x = 0)

4.0 Screen Shots



screen shot 4.1

5.0 Reference

- *http://www.ai-junkie.com/ann/evolved/nnt1.html*
- *http://www.obitko.com/tutorials/genetic-algorithms/index.php*

6.0 More information

- tcg.galahena@gmail.com
- http://www.inf0warri0r.blogspot.com