CS5491 Artificial Intelligence Project 2	
WANG Yue	56359462

Description of problem:

This project implements an artificial neural network (ANN) for solving a diabetes classification problem using the training method of genetic algorithm.

Inputs and outputs of the ANN:

ANN's Input: 'diabetes.txt'
The first column of the diabetes data file is the bias input, and the second to the ninth columns are the input features. The tenth and eleventh columns of the diabetes data file are the output values. A one in the tenth column would represent diabetes positive. The 10th column is the label column which indicates whether the tuple is classified as diabetes or not. There are a total of 768 patterns in the dataset. 576 are used as training pattern and 192 are used as test pattern.

ANN's Output:
The output of the ANNs should be the predicted output according to the ANN's computation.

The inputs and outputs of the layers in the ANN model (hidden layer):
Starting from the input layer, all outputs from each layer will become the inputs for the next layer. In my experiment, there is only one hidden layer, so the input of this hidden layer is the out put of the input layer, and the output of hidden layer will becomes the input of the output layer.

How is training data gathered?

There are a total of 768 patterns in the dataset. 576 are used as training pattern and 192 are used as test pattern. Training data accounts for 75% of the total.

How is the ANN trained (or learned)?

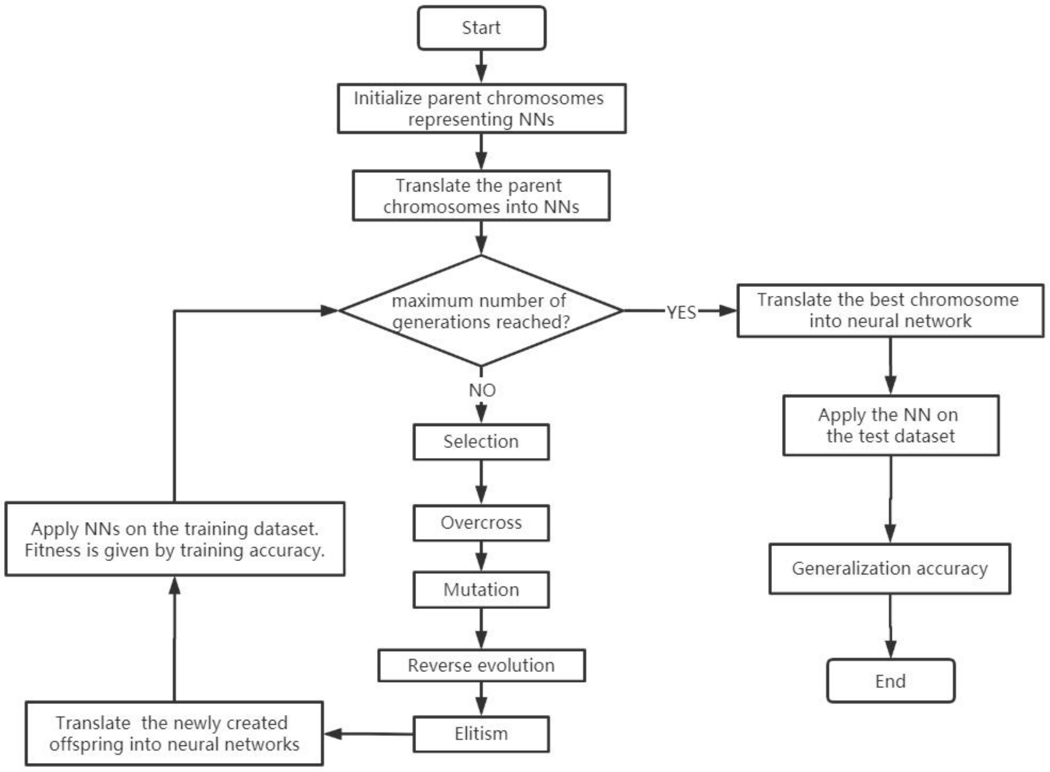
An artificial neural network using genetic algorithm is built to optimize the weights in each layer of the training model.
Firstly, count the number of genes on a chromosome: geneNum = inputNum*hiddenuNum + (hiddenuNum+1)*outputNum; (noted that the inputNum including the bias of input layer). Then the training data will be fed into the training model and the predicting process begins. In the process of predicting, the fitness of each chromosome can be calculatied by the formula

fitness(k) = C_k/N*100% = the number of correctly classified patterns by chromosome k / the total number of patterns in the training set *100%. After the fitness calculation , the program will update the maximum fitness value for the final training since its weights are the optimal ones and could likely yield a higher accuracy in the final training stage. This process will continue go on until the maximum generation is met.

◆ Modifications of code

【1】main.m

This is the main function of this project. Here is the flow chart:


Start
Initialize parent chromosomes representing NNs
Translate the parent chromosomes into NNs
maximum number of generations reached?
YES
Translate the best chromosome into neural network
Apply the NN on the test dataset
Generalization accuracy
End
NO
Selection
Overcross
Mutation
Reverse evolution
Elitism
Translate the newly created offspring into neural networks
Apply NNs on the training dataset. Fitness is given by training accuracy.

```
clc clear
t0 = clock; %The clock that calls the Windows system does the time difference calculation
global hiddenuNum
global inputNum
global outputNum
%Initialisation NN ===============================================
diabetesData = load('.\diabetes.txt'); %Change the directory to the one on your computer
%Number of nodes per layer
```

```
inputNum=9;                %8 Number of input units + 1 one for bias input
hiddenuNum = 2;         %the number of units in hidden layer
outputNum=2;              %the number of units in output layer
%Training/Testing-------------
dataNum=768; %Total data
trnDataRatio=0.75; %The ratio of data training
inputTrain=diabetesData(1:ceil(trnDataRatio*dataNum),1:inputNum); %Training input data
outputTrain=diabetesData(1:ceil(trnDataRatio*dataNum),inputNum+1:inputNum+outputNum); %Training output
data
inputTest=diabetesData(ceil(trnDataRatio*dataNum+1):dataNum,1:inputNum); %Test input data
outputTest=diabetesData(ceil(trnDataRatio*dataNum+1):dataNum,inputNum+1:inputNum+outputNum); %Test
output data
[trainNum,outputNum]=size(outputTrain);   %trainNum Training data quantity
[testNum,inputNum]=size(inputTest);          %testNum Test data quantity
% inputTrain = mapminmax(inputTrain); % inputTest = mapminmax(inputTest);
%Initialisation GA ================================================
generation = 300;         %Generations/iterations/epoch
chromosomeNum = 50;     %Initial population size
geneNum = inputNum*hiddenuNum + hiddenuNum*outputNum + outputNum; %The total number of w
fitnessStatis = zeros(generation,chromosomeNum); %Record fitness changes for each round
fitnessBest = zeros(1,generation);
fitnessAve = zeros(1,generation);
fitnessMin = zeros(1,generation);
w_evolution = zeros(generation,geneNum+1);   %(best chromosome + bestfitness)
pc = 0.5;          %Crossover probability
pm = 0.2;          %Mutation probability
GAP = 0.9;        %Generation gap (i.e., selection probability)
% Initialization of the population--------------------------------
Chrom = InitialChrom (chromosomeNum,geneNum);
fitness = zeros(1,chromosomeNum);   %初始化
iter = 0;
% GA ===============================================================
while iter < generation
    iter
    %Calculated fitness------------------------------
    for c = 1: chromosomeNum
        [trnCorrPercent,trnCorrnum] = NNforward( Chrom(c,:),inputTrain,outputTrain,trainNum);
        %[Correct rate of training data, correct number of training data]
        fitness(c) = trnCorrnum/trainNum;
    end
    %Record w Settings for maximum Fitness + average + iteration per round----------
```

```
        [fitness_best,best_index] = max(fitness);
        fitnessBest(iter+1) = fitness_best;          %Maximum fitness for each iteration
        fitnessAve(iter+1) = mean(fitness);          %Average fitness for each iteration
        fitnessMin(iter+1) = min(fitness);           %Min
        w_evolution(iter+1,:) = [Chrom(best_index,:) Chrom(best_index,1)];    %record optimal
        fitnessStatis(iter+1,:)= fitness;    %Record all fitness values for this time
        %=========================================
        % select-------------------------------
         select_num = max(floor(chromosomeNum*GAP+0.6),2);
         %The number of chromosomes selected
        Chrom1 = Select(Chrom, fitness,select_num);
        % crossover-------------------------------
        Chrom2 = Crossocer( Chrom1, pc, fitness );
        % mutation-------------------------------
        Chrom3 = Mutation( Chrom2, pm );
        %Reverse evolution--------------------------
        Chrom4 = Reverse( Chrom3 , inputTrain , outputTrain ,trainNum);
        % rein-------------------------------
        Chrom = Rein( Chrom4, Chrom, fitness );
        iter = iter+1;
end
% test data ==tstCorrPercent,tstCorr================================
[trnCorrPercent,trnCorrnum,tstCorrPercent,tstCorrnum ]...
= TestForward(w_evolution(end,1:geneNum), inputTrain, outputTrain,trainNum, inputTest,outputTest,testNum);
% test data ==tstCorrPercent,tstCorr================================
[trnCorrPercent,trnCorrnum,tstCorrPercent,tstCorrnum ]...
= TestWithBp(w_evolution(end,1:geneNum), inputTrain, outputTrain,trainNum, inputTest,outputTest,testNum);
%output======================================================
%parameters setting
disp(['The generation is:' num2str(generation)]);
% time
Time_Cost=etime(clock,t0);      %The clock that calls the Windows system does the time difference calculation
disp(['program execution time:' num2str(Time_Cost) 'seconds']);
% best fitness
disp(['Accuracy on training data set:' num2str(trnCorrPercent)]);
disp(['Accuracy on testing data set:' num2str(tstCorrPercent)]);
```

【2】InitialChrom.m

This function is used to randomly generate the initial chromosome. Every gene on every chromosome is a random number from -1 to 1.

```
function Chrom = InitialChrom(chromosomeNum,geneNum)
%    Chrom = randn(chromosomeNum,geneNum);
  for i=1:chromosomeNum
       Chrom(i,:) = 2*rand(1,geneNum)-1;      %Random initial population
  end
end
```

【3】TestForward.m

This function Apply NNs on the training dataset. Accroding to the accuracy in training data, fitness of every chromosome in this epoch are also given. In other words, use the ‘forward’ part of MLP to caculate the fitness.

```
function [trnCorrPercent,trnCorr,tstCorrPercent,tstCorr ]...
        = TestForward(chromosome,trnInData,trnOutData,trnPatternNum, tstInData,tstOutData,tstPatternNum)
global hiddenuNum
global inputNum
global outputNum
% w -------------------------
% weights of inputs and bias to hidden units
w11 = chromosome(1,1:inputNum*hiddenuNum);
w1=reshape(w11,hiddenuNum, inputNum);
% weights of hidden units and bias to output units
w22 = chromosome(1,inputNum*hiddenuNum+1:end);
w2=reshape(w22,outputNum,hiddenuNum+1);
outSubErr=zeros(1,outputNum);
tErr = 0;
trnCorr = 0; %Accuracy rate of training data
tstCorr = 0; %Accuracy rate of test data
for patternCount=1:trnPatternNum %Training each data=======================================
        %forward pass-------------------------------------------------
        for i=1:hiddenuNum %hidden layer
                huInput(i)=trnInData(patternCount,:)*w1(i,:)';
                huOutput(i)=logsig(huInput(i));
        end
        for i=1:outputNum %output layer
                ouInput(i)= w2(i,:)*[1;huOutput'];
                ouOutput(patternCount,i)= logsig(ouInput(i));
        end
        % Based on sum of squared error-----------------------
        for i=1:outputNum
                outSubErr(i)=outSubErr(i)+0.5*(trnOutData(patternCount,i)-ouOutput(patternCount,i))^2;
        end
end %all data end ==========================================================
%total error for all output during one epoch pass-----------------------
for i=1:outputNum
        tErr=tErr+outSubErr(i); %total error for all output during one epoch pass
end
Err = tErr;
%=====================================================================
% Calculate classification accuracy on Trn set---------
for patternCount=1:trnPatternNum
        for i=1:hiddenuNum %hidden layer
                huInput(i)=trnInData(patternCount,:)*w1(i,:)';
                huOutput(i)=logsig(huInput(i));
        end
        for i=1:outputNum %output layer
                ouInput(i)= w2(i,:)*[1;huOutput'];
                ouOutput(patternCount,i)= logsig(ouInput(i));
        end
        winningClassTrn=1;
        for i=2:outputNum
if(ouOutput(patternCount,i)>ouOutput(patternCount,1))&(ouOutput(patternCount,i)>ouOutput(patternCount,winni
ngClassTrn))
                        winningClassTrn=i;
                end
        end
        if trnOutData(patternCount,winningClassTrn)== 1
                trnCorr=trnCorr+1;
        end
end
trnCorrPercent = (trnCorr/trnPatternNum)*100;      %Accuracy rate obtained from training data
%=====================================================================
% Calculate generalization accuracy on Tst set---------
for patternCount=1:tstPatternNum
        for i=1:hiddenuNum %hidden layer
                huInput(i)=tstInData(patternCount,:)*w1(i,:)';
                huOutput(i)=logsig(huInput(i));
        end
        for i=1:outputNum %output layer
                ouInput(i)= w2(i,:)*[1;huOutput'];
                ouOutput(patternCount,i)= logsig(ouInput(i));
        end
        winningClass=1;
        for i=2:outputNum
if(ouOutput(patternCount,i)>ouOutput(patternCount,1))&(ouOutput(patternCount,i)>ouOutput(patternCount,winni
ngClass))
                winningClass=i;
            end
        end
        if tstOutData(patternCount,winningClass)== 1
            tstCorr=tstCorr+1;
        end
end
tstCorrPercent = (tstCorr/tstPatternNum)*100;
end
```

【4】Select.m

This function is used to select by using Roulette wheel , The portion of the roulette wheel each chromosome represents is determined by its fitness value. The higher the fitness value of the chromosome, the larger the portion of roulette wheel. Chromosomes with a higher fitness are more likely to be selected.
Another implementation method is commented out below.

```
function Chrom2 = Select(Chrom,Fit,select_num)
%Individual selection probability
sumfitness=sum(Fit);
sumf=Fit./sumfitness;
[chromosomeNum,~]=size(Chrom);
%Use roulette to select new individuals
index=[];
for i=1:select_num      %Turn the wheel select_num times
        pick=rand;
        while pick==0         %not 0
            pick=rand;
        end
        for j=1:chromosomeNum
            pick=pick-sumf(j);
            if pick<0
                index=[index j];
                break;
                %This time chromosome I is selected.
                %Note that it is possible that certain chromosomes may be selected repeatedly
                %as the select_NUM wheel rotates.
```

```
    end
end
%new
Chrom2=Chrom(index,:);
%Another way-----------------------------------------------------------------
% [~,N] = size(Fit);
% Fit = mapminmax(Fit);
% a = min(Fit);
% b = sum(Fit) + N*(-a);
% for i=1:N                                    %divide a disk into N regions
%         POP_adapt(i)=(Fit(i)-a)/b;
% end
% POPnew_adapt = cumsum(POP_adapt); %Sum to 1
% for i = 1:select_num
%         target_index = find(POPnew_adapt>rand);
%         Chrom2(i,:) = Chrom(target_index(1),:);
% end
end
```

【5】Crossover.m

This function is used to implement the crossover. Randomly generate a 0-1 number to compare with pc, if the random is smaller, exchange the second half of the two chromosomes.
(The switching points are also randomly generated).

```
function Chrom3 = Crossocer(Chrom, pc, fitness)
[row,lenchrom ]= size(Chrom);
for i=1:row
    pick=rand(1,2);
    while prod(pick)==0
        pick=rand(1,2);
    end
    index=ceil(pick.*row);
    % The crossover probability determines whether or not to cross
    pick=rand;
    while pick==0
        pick=rand;
    end
    if pick>pc
        continue;
    end
    % Randomly select the cross bit
```

```
        pick=rand;
        while pick==0
                pick=rand;
        end
        pos=ceil(pick*lenchrom); %Random selection of crossover location, that is, select the number of          variables
to cross,
        %note: the two chromosomes intersect at the same location
        pick=rand; %Cross began
        v1=Chrom(index(1),pos);
        v2=Chrom(index(2),pos);
        Chrom(index(1),pos)=pick*v2+(1-pick)*v1;
        Chrom(index(2),pos)=pick*v1+(1-pick)*v2; %Cross over
end
Chrom3 = Chrom;
end
```

【6】Mutation.m

This function is used to implemente Mutation. In each iteration there is a probability pm will be mutated on each chromosome. In the chromosome, Pick a gene at random and chang it to a random value of -1 to 1.

```
function Chrom = Mutation(Chrom, pm)
[a,b]=size(Chrom);
for i=1:a
        for j=1:b
                if rand<pm
                        Chrom(i,j)=2*rand-1;
                end
        end
end
end
```

【7】Reverse.m

This function is used to implemente reverse evolution. This is not available with standard genetic algorithms and is an operation I add to speed up evolution. Evolution here means that the reversal operation is unidirectional, that is, the individual will perform the reversal operation only after the reversal becomes better, otherwise the reversal is invalid. The specific method is to randomly generate two random Numbers r1 and r2 between [1,geneNum] (in fact, it is allowed to be the same, but when r1 and r2 are the same, it is not effective to reverse the nature, and setting cross variation is not effective, but this does not happen very often), and then reverse the sequence of genes between r1 and r2.

```
function Chrom = Reverse( Chrom, inputTrain , outputTrain ,trainNum )
[row, col] = size(Chrom);
NEWChrom = Chrom;   %NEWChorm as an intermediate variable for comparison
for i = 1:row
      [ ~ ,fitness_old   ] = NNforward ( Chrom(i,:), inputTrain , outputTrain ,trainNum);
      r = unidrnd(col,1,2);   %Generate two random numbers
      minrev = min(r);
      maxrev = max(r);
      NEWChrom(i,minrev:maxrev) = fliplr(NEWChrom(i,minrev:maxrev));
      [ ~ ,fitness_new   ] = NNforward ( Chrom(i,:), inputTrain , outputTrain ,trainNum);
      if fitness_new > fitness_old
            Chrom(i,:) = NEWChrom(i,:);   %Update if the results are better
      end
end
```

【8】Rein.m

This is the function to use Elitism in GA. This part adds the technique of Elitism, which means Re-introduce in the population previous best-so-far (elitism). Keep the best 10% of parent population and re-introduce them in the next generation by replacing the worst 10% children population. In GA, elitism tends to improve on the final results.

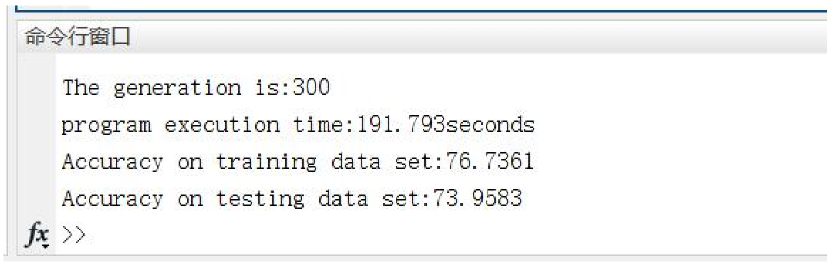
```
function Chrom4 = Rein( Chrom3, Chrom, fitness )
row_parents = size(Chrom,1);
row_son = size(Chrom3,1);   %There are POP_num*(1-GAP) differences between the mother and the
offspring
[~,index] = sort(fitness);
c=[];
for i =1:(row_parents-row_son)

       c=[c; Chrom(index(1),:)];
end
Chrom = [c;Chrom3];
Chrom4 =Chrom;
%The genes with the highest fitness values were recorded and inserted into the offspring
end
```

◆ Screenshot of an experiment

Parameter Settings for a certain experiment:

Parameter Description	Settings
Number of unit in input layer	9 (8+1bias)
Number of unit in hidden layer	3 (can change it)
Number of unit in output layer	2
Initial population number	30
Crossover rate	0.7
Mutation rate	0.05
Generations / Iterations	300
GAP	0.9

Screenshot of this experiment:



```
The generation is:300
program execution time:191.793seconds
Accuracy on training data set:76.7361
Accuracy on testing data set:73.9583
fx >>
```

◆ How do you ensure that the ANN can generalize well?

GA is used to help raise the accuracy rate by preventing the model from being trapped in a local minimum.

The main steps of GA:

1) fitness Caculation:

Caculate the fitness value of all chromosomes in the current population.

2) Selection:

Select parents for reproduction by using Roulette wheel , The portion of the roulette wheel each chromosome represents is determined by its fitness value. The higher the fitness value of the chromosome, the larger the portion of roulette wheel.

3) Crossover:

Selects two genes randomly and recombines them by exchanging their parts.

4) Mutation:

In each iteration there is a probability pm will be mutated on each chromosome. In the chromosome, Pick a gene at random and chang it to a random value of -1 to 1.

5) Reverse evolution:

This is not available with standard genetic algorithms and is an operation I add to speed up evolution. Evolution here means that the reversal operation is unidirectional, that is, the individual will perform the reversal operation only after the reversal becomes better, otherwise the reversal is invalid. The specific method is to randomly generate two random Numbers r1 and r2 between [1,geneNum] (in fact, it is allowed to be the same, but when r1 and r2 are the same, it is not effective to reverse the nature, and setting cross variation is not effective, but this does not happen very often), and then reverse the sequence of genes between r1 and r2.

6) Elitism in GA:

This part adds the technique of Elitism, which means Re-introduce in the population previous best-so-far (elitism). Keep the best 10% of parent population and re-introduce them in the next generation by replacing the worst 10% children population. In GA, elitism tends to improve on the final results.

◆ How does the ANN perform in comparison with other methods?

Gradient descent method has a problem of converging to the global minimum and therefore resulting an accuracy that is relatively low. While genetic algorithm performs well since it has selection, crossover and mutation that could improve the dilemma by producing variant kinds of genes and selecting the best among them.

In conclusion, ANN has some advantages, such as nonlinear mapping capability, the strong learning and adaptive abilities, generalization ability and Fault tolerance, but it also has some shortcoming: 1).Local minimization - From a mathematical point of view, BP algorithm is an optimization method of local search, but the problem it needs to solve is to solve the global extremum of complex nonlinear function. Therefore, the algorithm is likely to fall into the local extremum and make the training fail. On the other hand, BP neural network is very sensitive to the initial network weight. When the network is initialized with different weights, it tends to converge to different local minima. 2).The ability of network approximation and generalization is closely related to the typicality of learning samples, and it is a very difficult problem to select typical samples from the problems to form training sets. 3).The contradiction between the network's predictive ability (also known as generalization ability and generalization ability) and training ability (also known as approximation ability and learning ability). In general, when training ability is poor, prediction ability is also poor, and to a certain extent, with the improvement of training ability, prediction ability is also improved. However, there is a limit to this trend. When the limit is reached, the prediction ability will decline with the improvement of training ability, that is, the so-called "overfitting" phenomenon will appear.

◆ Experiment with the various parameters

【1】Change the num of noeds in hidden layer

This part keeps the other parameters unchanged and sets the num of noeds in hidden layer to 1, 2,3,4,5,6,7,8,9, respectively, and observe the changes in the experimental results.

Other parameter Settings：generation=300， population=30.

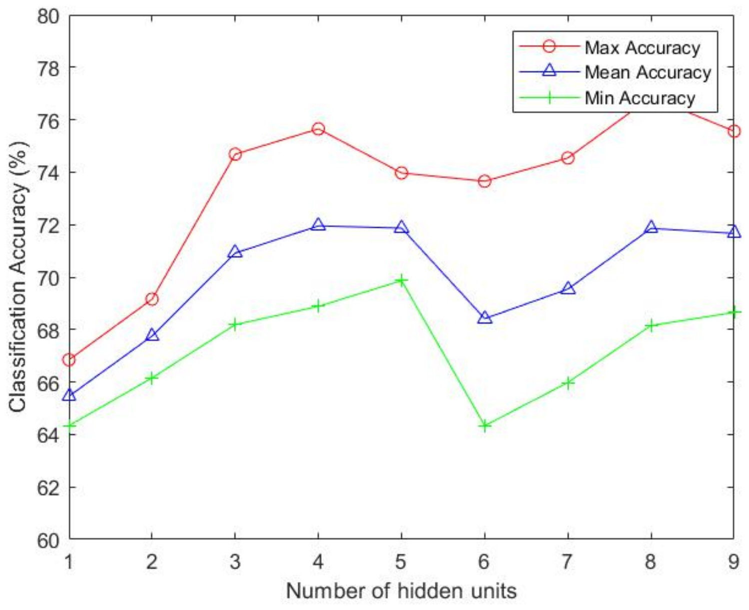


Figure 1 - accuracy varies with the mumber of hidden units

From figure1 we can see that accuracy does not change significantly with the the increase of the number in hidden units, but generally speaking, it rises gradually.
In summary, you can increase the number of hidden nodes in pursuit of more accurate results, but the increase of computational costs are also worth considering.

【2】Generation

This part's aim is to observe the change of accuracy as generation increases.
Parameter Settings： population=30, hidden units=3.

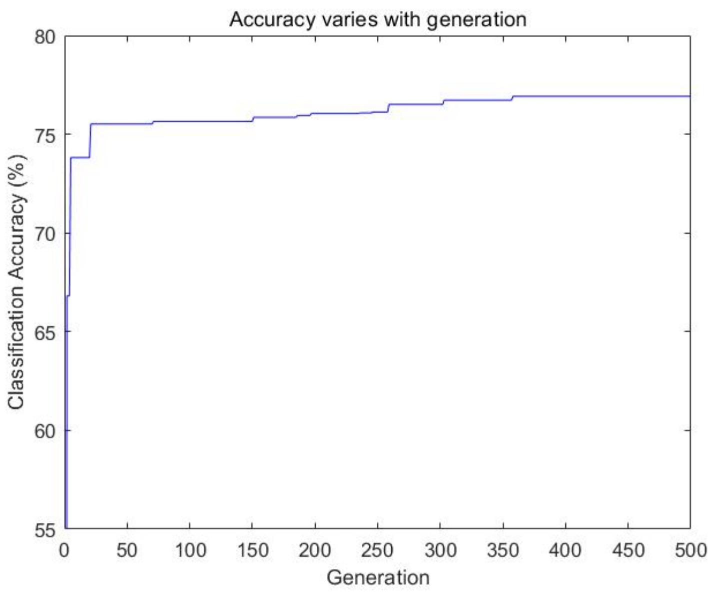


Figure 2 - accuracy varies with generation

From figure2, we can see that with the increase of the number of generation, the accuracy rate will increase, but the increase is small. Especially when generation is greater than around 350, The accuracy rate remains the same as the increase of generation. Larger generation means longer running times and computational cost, so you just pick a appropriate value for the experiment.

【3】Change the population

This part's aim is to observe the change of accuracy as population increases(30,50,70,90). Parameter Settings： population=30, hiddne units=3.

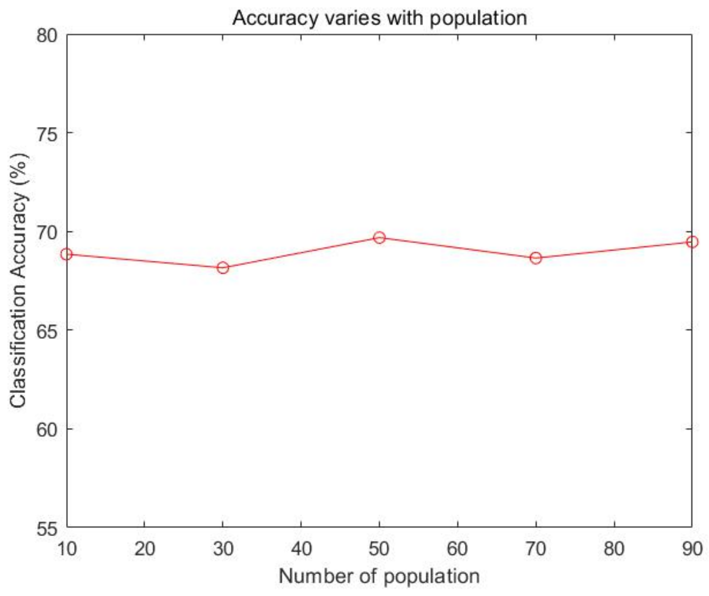


Figure 3 - accuracy varies with population

From figure3, we can see that the initial population size doesn't have significant impact on improving the accuracy.