珠海學院 CHU HAI COLLEGE OF HIGHER EDUCATION Department of Computer Science					SCORE/GRADE
STUDENT ID NO.	201436201H	STUDENT NAME	趙海騏	MARKER NAME INITIAL	
COURSE CODE	CSC167	COURSE TITLE	Neural Network	DATE	
ASSIGNMENT TITLE/PAGE NO.	Assignment 3 PAGE OF				

NO.	WORKING/STEPS	MARKS

self organizing map

Design

I have designed 6 process for SOM. There are som(main process), initialization, competitive_process, cooperative_process, adaptive_process, mapping_process.

Som: main process for som, call each process function one by one in each iteration.
Initialization: For initializing input variable and parameters
Competitive_process: Obtain the winning neuron of each sample(input)
Cooperative_process: Calculate distance between winning neuron and each neuron and topological neighborhood function
Adaptive_process: Calculate change of weight and adjust weight
Mapping_process: Show the context map after all iteration. step 1, find the neuron of strongest responses sample. step 2, fill the unoccupied neuron

parameters of my code

sig0 = 5;
t1 = 1000/(log(sig0));
r0 = 0.1;
t2 = 1000;

context map

Eagle	Eagle	Eagle	Cat	Cat	Cat	Cat	Fox	Fox	Dog
Eagle	Eagle	Eagle	Cat	Cat	Cat	Cat	Fox	Wolf	Wolf
Eagle	Eagle	Owl	Owl	Owl	Cat	Cat	Wolf	Wolf	Wolf
Owl	Owl	Owl	Owl	Owl	Owl	Lion	Lion	Wolf	Wolf
Dove	Owl	Owl	Hawk	Owl	Owl	Lion	Lion	Lion	Tiger
Dove	Dove	Owl	Owl	Owl	Owl	Lion	Lion	Lion	Tiger
Dove	Dove	Owl	Owl	Owl	Lion	Lion	Lion	Lion	Tiger
Dove	Dove	Duck	Goose	Goose	Horse	Horse	Horse	Horse	Horse
Hen	Hen	Duck	Goose	Goose	Goose	Horse	Zebra	Horse	Cow
Hen	Duck	Duck	Goose	Goose	Goose	Horse	Horse	Horse	Cow

Source code

som.m

```
%Main process of Self-organizing map
clear all
close all
% initialization
[sig0,t1,r0,t2,x,sam_name,x_nor,w,n] = initialization();

for n1=1:n % for n interations; n1: current iterations; n: the number of iterations
    %show the process precentage
    if mod(n1,10)==0
      floor((n1/n)*100)
    end

    for m=1:16 % m for each animal's type (each input)
        % competitive_process
        [i]=competitive_process(m,x_nor,w);
        % cooperative_process & adaptive_process
        [w,d,h]=cooperative_process(i,sig0,t1,w,n1,r0,t2,x_nor,m);
    end
end

%mapping_process
[map,result]=mapping_process(x_nor,w,sam_name);
```

---

initialization.m

```
function [sig0,t1,r0,t2,x,sam_name,x_nor,w,n] = initialization()
% paramter
sig0 = 5;  % exponential decay at the initiation; should be the radious of map size
t1 = 1000/(log(sig0));  % time constant, relative to sig0
r0 = 0.1;  % learning rate; 0.1 is recommended
t2 = 1000;  % tor 2; 1000 is recommended
% x: input data
x=[1 0 0 1 0 0 0 0 1 0 0 1 0; %Dove
   1 0 0 1 0 0 0 0 1 0 0 0 0; %Hen
   1 0 0 1 0 0 0 0 1 0 0 0 1; %Duck
   1 0 0 1 0 0 0 0 1 0 0 1 1; %Goose
   1 0 0 1 0 0 0 0 1 1 0 1 0; %Owl
   1 0 0 1 0 0 0 0 1 1 0 1 0; %Hawk
   0 1 0 1 0 0 0 0 1 1 0 1 0; %Eagle
   0 1 0 0 1 1 0 0 0 1 0 0 0; %Fox
   0 1 0 0 1 1 0 0 0 0 1 0 0; %Dog
   0 1 0 0 1 1 0 1 0 1 1 0 0; %Wolf
   1 0 0 0 1 1 0 0 0 1 0 0 0; %Cat
   0 0 1 0 1 1 0 0 0 1 1 0 0; %Tiger
   0 0 1 0 1 1 0 1 0 1 1 0 0; %Lion
   0 0 1 0 1 1 1 1 0 0 1 0 0; %Horse
   0 0 1 0 1 1 1 1 0 0 1 0 0; %Zebra
   0 0 1 0 1 1 1 0 0 0 0 0 0; %Cow
];
sam_name={'Dove';'Hen';'Duck';'Goose';'Owl';'Hawk';'Eagle';'Fox';'Dog';'Wolf';'Cat';'Tiger';'Li
on';'Horse';'Zebra';'Cow';};
% x_nor: normolization of input data
```

```
x_nor=[];
for m=1:16
    x_row=0;
    for n=1:13
       x_row= x_row + (x(m,n))^2;
    end
    for n=1:13
       x_nor(m,n)= x(m,n)/x_row^(1/2);
    end
end
% w: synaptic-weight vector of each neuron
w=randn(100,13);
% n: the number of iterations
n=1000;


% up to now, initialization finish
end
```

---

competitive_process.m

```
function [i]=competitive_process(m,x_nor,w)
  % i: inedx which the x_nor and w is most similiar with each other ;  winning neuron
  temp=[];
  for n=1:100
    temp(n)=0;
    for k=1:13
      temp(n) = temp(n) + abs(x_nor(m,k)-w(n,k));
    end
  end
  [M,Min_i] = min(temp);
  i=Min_i;

end
```

---

cooperative_process.m

```
function [w,d,h]=cooperative_process(i,sig0,t1,w,n1,r0,t2,x_nor,m)

  % i: winning neuron
  % j: each neuron of 10*10
  % ai: x-axis of i
  % bi: y-axis of i
  % aj: x-axis of j
  % bj: y-axis of j
  % d: distance between j and i
  % n: the number of iterations
  % n1: current iterations
  % n1: the number of iterations in current process
  % sig(n): exponential decay
  % h: topological neighborhood function
  % r0: adaptive_process usage
  % t2: adaptive_process usage
```

```
  % w: adaptive_process usage
  % x_nor: adaptive_process usage
  % m: sample number m; adaptive_process usage

  ai= mod(i-1,10);
  bi=floor((i-1)/10);
  for j=1:100
    aj= mod(j-1,10);
    bj= floor((j-1)/10);
    d(j)=((ai-aj)^2+(bi-bj)^2)^(1/2);
    sig(n1)=sig0*exp(-n1/t1);
    h(j)=exp((-(d(j))^2)/(2*(sig(n1))^2));

    % adaptcive_process
    [w(j,:)]=adaptive_process(r0,t2,i,h(j),w(j,:),x_nor,m,n1);
  end

end
```

---

adaptive_process.m

```
function [w]=adaptive_process(r0,t2,i,h,w,x_nor,m,n1)

  % deltaw: change of the weight vector in neuron j
  % r(n): learning rate at n iterations
  % n1: current iterations

  r(n1)=r0*exp(-n1/t2);
  deltaw=r(n1)*h.*(x_nor(m,:)-w);
  w=w+deltaw;

end
```

---

mapping_process.m

```
function [map,result]=mapping_process(x_nor,w,sam_name)
  % map: map of animal in number formula
  % result: map of animal in name formula
  % X: x-axis of lattice map
  % Y: y-axis of lattice map

  %plot the grid
  [X,Y]=meshgrid(0:10);
  figure; hold on;
  plot(X,Y,'k');
  plot(Y,X,'k');
  axis off;
  grid on;

  % step 1, find the neuron of strongest responses sample
  for m=1:16
     for j=1:100
```

```
    temp(j)=0;
     for k=1:13
        temp(j) = temp(j) + abs(x_nor(m,k)-w(j,k));
     end
    end
    [M,temp_win_j] = min(temp);
    result(temp_win_j,:)=sam_name(m,:);
    occupied(temp_win_j)=1; % 1 mean that neuron is occupied
  end

  % step 2, fill the unoccupied neuron
  temp=[];
  for j=1:100
   if (occupied(j)~=1)
     occupied(j)=1;
     for m=1:16
        temp(m)=0;
        for k=1:13
           temp(m) = temp(m) + abs(x_nor(m,k)-w(j,k));
        end
     end
     [M,map(j)] = min(temp);
     result(j,:)=sam_name(map(j),:);
   end
   text(mod(j-1,10)+0.1,floor((j-1)/10)+0.5,result(j,:)); % show the content of map in lattice
  end

  % save the result figure
  saveas(gcf,'Result_som.png')
end
```