```
%% Q1 Using MATLAB, generate a matrix called pn that encodes the 4 patterns on slide 35 as a 4
% data obtained on slide 51
pn = [1 1 -1 -1 -1 -1 -1 1;
    -1 1 1 1 -1 -1 1 -1;
    -1 -1 1 1 1 -1 -1 -1;
    1 -1 -1 1 -1 1 -1 1];

%% Q2  Call the function w = memstor(pn) to create a Hopfield network (an 8x8 matrix) from the
w = memstor(pn)
```

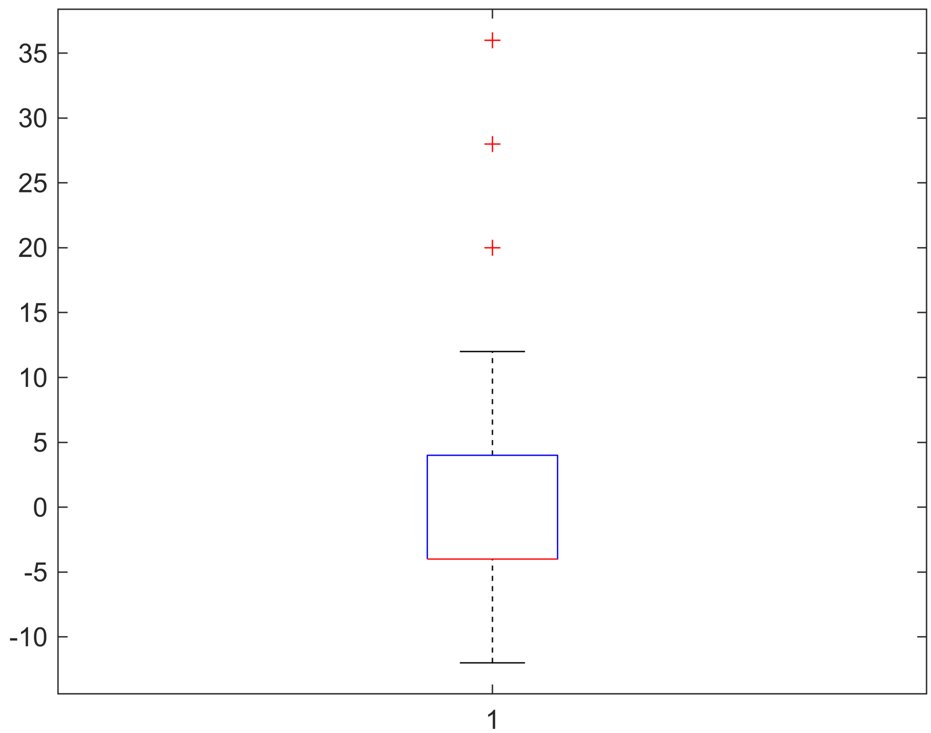
```
w = 8×8
     0     0    -4    -2    -2     2    -2     4
     0     0     0    -2    -2    -2     2     0
    -4     0     0     2     2    -2     2    -4
    -2    -2     2     0     0     0     0    -2
    -2    -2     2     0     0     0     0    -2
     2    -2    -2     0     0     0     0     2
    -2     2     2     0     0     0     0    -2
     4     0    -4    -2    -2     2    -2     0
```

```
%% Q3 Compute the Goodness Value for for all possible patterns (goodness.m is in the slide set)
g = goodness(w)
```

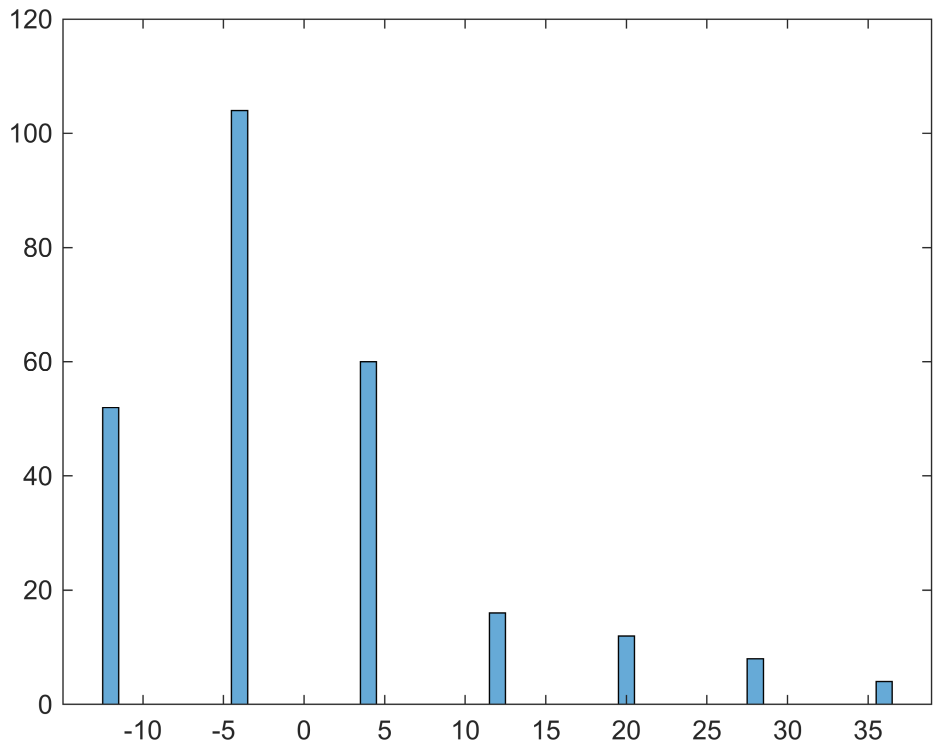
```
ans = 256×9
    -1    -1    -1    -1    -1    -1    -1    -1   -12
     1    -1    -1    -1    -1    -1    -1    -1    -4
    -1     1    -1    -1    -1    -1    -1    -1    -4
     1     1    -1    -1    -1    -1    -1    -1     4
    -1    -1     1    -1    -1    -1    -1    -1    -4
     1    -1     1    -1    -1    -1    -1    -1   -12
    -1     1     1    -1    -1    -1    -1    -1     4
     1     1     1    -1    -1    -1    -1    -1    -4
    -1    -1    -1     1    -1    -1    -1    -1    -4
     1    -1    -1     1    -1    -1    -1    -1    -4
     ⋮

g = 256×1
   -12
    -4
    -4
     4
    -4
   -12
     4
    -4
    -4
    -4
     ⋮
```

```
%% Q4 Generate a boxplot or histogram that shows the distribution of Goodness values across al
figure
boxplot(g)
```




```
figure
histogram(g)
```



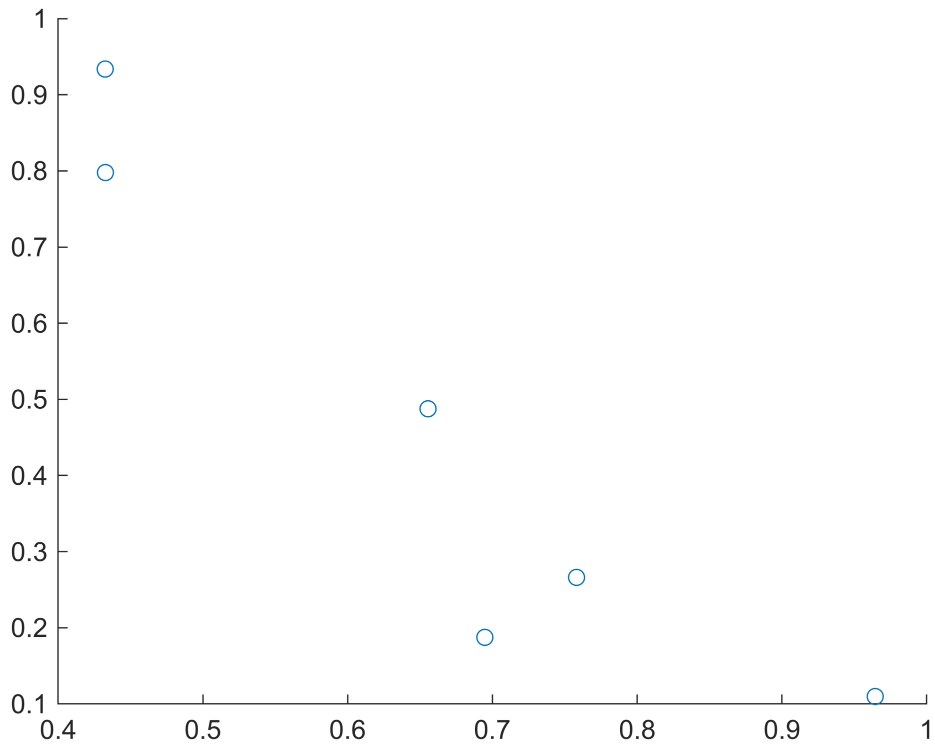
Q5 Using hopupdate.m and the Necker cube weights (both from the slide set), run a Hopfield net for many iterations and interpret the result. (show a printout of the final activities -- a plot is great, just numbers is sufficient)

```
clear;
% Necker cube obtained from slide 17
w=[0.0  1.0  1.0  0.0  1.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  0.0  0.0;
    1.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  0.0;
    1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  0.0  0.0;
    0.0  1.0  1.0  0.0  0.0  0.0  0.0  1.0  0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  0.0;
    1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0;
    0.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0 -1.5  0.0 -1.5;
    0.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0;
    0.0  0.0  0.0  1.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0 -1.5  0.0 -1.5;
    -1.5  0.0 -1.5  0.0  0.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  1.0  0.0  0.0  0.0;
    0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0;
    -1.5  0.0 -1.5  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0;
    0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0  0.0  1.0;
    0.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0;
    0.0  0.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0  1.0  0.0  0.0  1.0  0.0  0.0  1.0;
    0.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  1.0;
    0.0  0.0  0.0  0.0  0.0 -1.5  0.0 -1.5  0.0  0.0  0.0  1.0  0.0  1.0  1.0  0.0];

iv = rand(16,16)-0.5; % generate a initial value of a random matrix of [-1, 1]
h = hopupdate(w,iv,100)
```

```
h = 16×16
   -1.0000   -0.1819   -0.0954   -0.1065    0.1834   -0.1226   -0.3001    0.3507 ···
   -1.0000   -0.3808   -0.0516    0.1714    0.2040   -0.2840   -0.0930    0.0606
   -1.0000    0.4398   -0.1342    0.2413   -0.0577    0.2904    0.2487    0.4296
   -1.0000    0.1456    0.2635    0.0201   -0.4804    0.4493    0.3256    0.1967
   -1.0000   -0.0205    0.1279   -0.1523   -0.1691   -0.1724    0.2900    0.0828
   -1.0000    0.1393    0.2720   -0.3500   -0.0757    0.1713   -0.1815    0.3154
   -1.0000    0.0447    0.4329    0.0861   -0.2297   -0.0614    0.0341    0.3790
   -1.0000    0.1473    0.4727   -0.2379   -0.3029    0.3335   -0.4100    0.4889
   -1.0000    0.0439   -0.3080   -0.4555    0.3217    0.2689   -0.3883   -0.4995
   -1.0000    0.2210   -0.3611    0.2549   -0.0701   -0.3327   -0.3637    0.3654
      ⋮
```

```
%% Q6 Replicate the TSP example from the slides using a set of 6 random cities (instead of 7)
locations=rand(6,2); %creates 6 random locations
scatter(locations(:,1),locations(:,2));
```



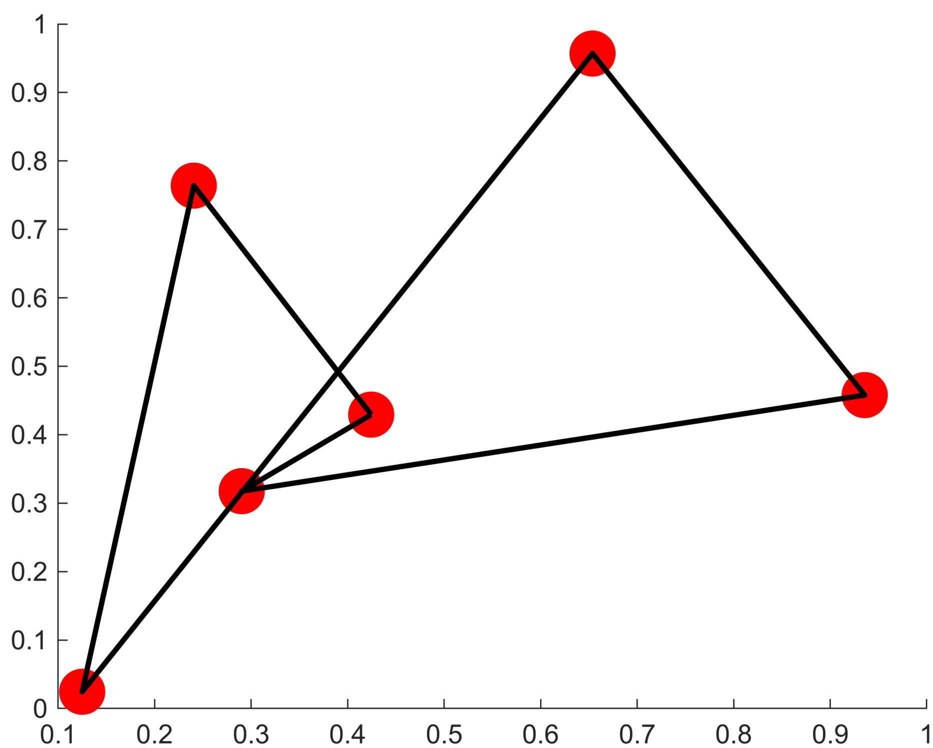

```
dmat=squareform(pdist(locations)); %computes the distance matrix
tspmat=hopfieldwts(6,45,dmat);
iacn(.05*rand(6,6),0.2*rand(6,6),tspmat,.05,100000)
```

```
ans = 6×6
    0.0000    0.0000    0.0000    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000    0.0000    0.0000    0.0000
    0.0000    1.0000    0.0000    0.0000    0.0000    0.0000
    1.0000    0.0000    0.0000    0.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    1.0000    0.0000    0.0000
    0.0000    0.0000    0.0000    0.0000    0.0000    1.0000
```

```
%% Q7  Download the matlab programs in the module.  Try running tsp.m which runs a constraint s
%{
- tsp.m generates a set of random locations and finds a path through them -- the network does n
- The user (you) can set the number of locations and the magnitude of the inhibition in the fir
- After running it, you can run it subsequent trials to find a different route for the same set
- If the programs does not converge on several trials, try increasing the "inhib" parameter (li
%}
clear;
% We'll need the professor provided .m files to run below
tsp;
```

```
loc = 2×6
    0.4243    0.1249    0.2902    0.6537    0.9357    0.2405
    0.4294    0.0244    0.3175    0.9569    0.4579    0.7639
dm = 6×6
         0    0.5036    0.1747    0.5753    0.5122    0.3817
```

```
    0.5036         0    0.3365    1.0720    0.9194    0.7484
    0.1747    0.3365         0    0.7355    0.6606    0.4491
    0.5753    1.0720    0.7355         0    0.5732    0.4561
    0.5122    0.9194    0.6606    0.5732         0    0.7596
    0.3817    0.7484    0.4491    0.4561    0.7596         0
revdist = 6×6
    0.0111    0.0840    2.5000    0.0645    0.0811    0.1658
    0.0840    0.0111    0.2419    0.0272    0.0321    0.0419
    2.5000    0.2419    0.0111    0.0430    0.0507    0.1088
    0.0645    0.0272    0.0430    0.0111    0.0649    0.1049
    0.0811    0.0321    0.0507    0.0649    0.0111    0.0410
    0.1658    0.0419    0.1088    0.1049    0.0410    0.0111
apattern = 6×6
    1.0000   -0.0000    0.0000    0.0000    0.0000   -0.0000
    0.0000   -0.0000    0.0000   -0.0000    1.0000    0.0000
    0.0000    1.0000    0.0000    0.0000   -0.0000    0.0000
   -0.0000   -0.0000   -0.0000    1.0000    0.0000    0.0000
    0.0000   -0.0000    1.0000   -0.0000   -0.0000   -0.0000
   -0.0000    0.0000    0.0000    0.0000   -0.0000    1.0000
s = 6×1
     1
     3
     5
     4
     2
     6
t = 3.6107
```




```
%% Attachments
% scraped from lecture slides
```

```
function mem=memstor(pats)
% each row of the matrix pats is a pattern
[np nd]= size(pats) ;
mem=zeros(nd) ;
for i=1:nd
    for j=1:nd
        if (i~=j)
            for k=1:np
                mem(i,j)=mem(i,j)+pats(k,i)*pats(k,j) ;
            end
        end
    end
end
end


function gvals = goodness( hopnet )
%calculates goodness for all patterns in a Hopfield Network
gvals=[];
pmat=[] ;
netsize=size(hopnet,1) ;
for k=0:(2^netsize-1)
    pvec=2*de2bi(k,netsize)-1; % need package
    %pvec=pvec([end:-1:1]) ;
    pmat=[pmat;pvec];
    g=0;
    for i=1:(netsize-1)
        for j=(i+1):netsize
            g=g+hopnet(i,j)*pvec(i)*pvec(j) ;
        end
    end
    gvals=[gvals, g];

end
gvals=gvals';
[pmat,gvals]
end

function hmtx=hopfieldwts(nc,mag,distances)
% nc = number of cities
% mag = magnitude of constraint (usually between 10 and 50)
% matrix is [target (city, position), source (target, position) ]
hmtx=zeros(nc,nc,nc,nc) ;
mdist=max(max(distances)) ;
nsd=10.0 ;
revdist=10*(mdist-distances+1).*(mdist-distances+1)/(mdist*mdist) ;
%revdist=0.1+10*exp(-distances.*distances/(mdist*mdist/(nsd*nsd))) ;
%revdist=0.1./(0.1+distances.*distances) ;
for j=1:nc
    hmtx(j,:,j,:)=mag*(eye(nc)-1) ;
```

```
    hmtx(:,j,:,j)=mag*(eye(nc)-1) ;
end
for j=2:nc
    for k=1:(j-1)
        % for each distance in the matrix distances
        for m1=1:nc
            m2=1+mod(m1,nc) ;
            hmtx(j,m1,k,m2) = revdist(j,k) ;
            hmtx(k,m1,j,m2) = revdist(j,k) ;
            hmtx(j,m2,k,m1) = revdist(j,k) ;
            hmtx(k,m2,j,m1) = revdist(j,k) ;
        end
    end
end
end

function finalact=iacn(extin,initact,conmat,dt,niter)
finalact=initact;
for k=1:niter
    finalact=iaciter(extin,finalact,conmat,dt);
end
end

function newact=iaciter(extinp,oldact,cmat,lr)
% Interactive activation model
% extinp -- ext input to each node
% oldact -- prev activity matrix (courses,times)
% cmat is the constraint matrix
% newact is the activity after a single iteration
% lr is the learning rate -- MUST BE < 1
del=mul4d2d(cmat,oldact) + extinp ; % netinputs to each hypothesis node
ldel = 2./exp(-del) - 1.0 ; % squashes input to [-1 +1]
newact = oldact + lr*(ldel>0).*(1-oldact) - lr*(ldel<0).*oldact ;
end

function outm=mul4d2d(m4d,m2d)
newdim=prod(size(m2d));
m4d2=reshape(m4d,newdim,newdim) ;
m2d1=reshape(m2d,newdim,1);
penout=m4d2*m2d1 ;
outm=reshape(penout,size(m2d)) ;
end

function newact = hopupdate( hmat,oldact,niter )
% hopfield updates for a fixed number of iterations
% (not the traditional stopping criterion)
newact=oldact ;
for ii=1:niter
    rrownum=randi(size(hmat,1),1); % fixed irand()
    % FIXME There's something terrible wrong in this asynchronous correction
```

```
    % But I'll leave it as the slide provided for now
    if (hmat(rrownum,:)*newact'>0)
        newact(rrownum)=1 ;
    else
        newact(rrownum)=-1;
    end
end
end
```