CSS 485

02/03/2020

Jun H Park

E7.1) Consider the prototype patterns given to the left.

i. Are p1 and p2 orthogonal?

P_1 = [-1 ; -1; 1; 1]

P_2 = [1 ; 1; -1 ; 1]

The inner between P1 and P2 are not 0, but -2. Therefore, they are not orthogonal.

ii. Use the Hebb rule to design an auto associator network for these patterns.

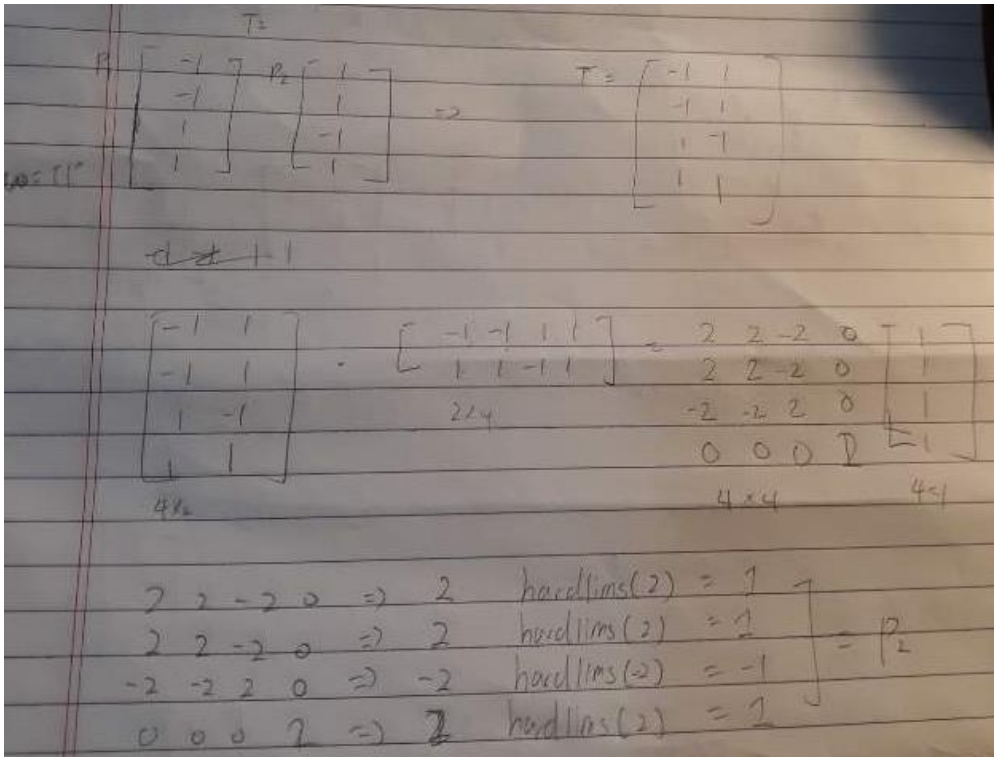
P = T = [-1 1 ; -1 1; 1 -1; 1 1]

$W_{new} = W_{old} + TP^T$ Where W_{old} is [0 0], therefore $W_{new} = TP^t$

Therefore, by following the equation

[-1 1 ; -1 1; 1 -1; 1 1] * [-1 -1 1 1 ; 1 1 -1 1] = [2 2 -2 0 ; 2 2 -2 0 ; -2 -2 2 0 ; 0 0 0 1]

iii. Test the operating of the network using the test input pattern p_t shown to the left. Does the network perform as you expected? Explain.



The Network should produce the prototype pattern that is the closest to the input pattern, which is P_2 in this case and the result is [1 ; 1 ; -1 ; 1], which matches P_2, Therefore, the network performed as I expected.

E7.2) Repeat Exercise E7.1 using the pseudoinverse rule.

i. Are p1 and p2 orthogonal?

P_1 = [-1 ; -1; 1; 1]

P_2 = [1 ; 1; -1 ; 1]

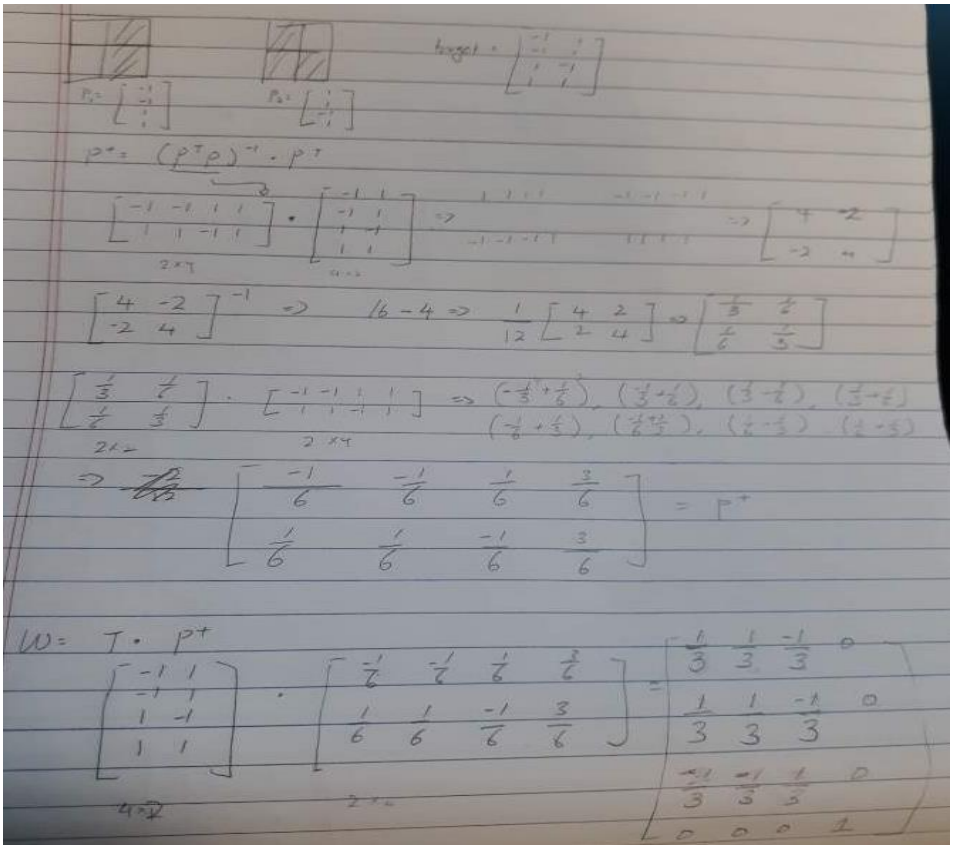
The inner between P1 and P2 are not 0, but -2. Therefore, they are not orthogonal.

ii. Use pseudoinverse rule to design

First, I had to get P^+, P^+ can be obtain by $(P^T * P)^{-1} * P^T$

To get weight, following equation had been used.

$W = T * P^+$



Therefore W = [1/3 1/3 -1/3 0 ; 1/3 1/3 -1/3 0 ; -1/3 -1/3 1/3 0 ; 0 0 0 1]

iii. Test the operating of the network using the test input pattern p_t shown to the left. Does the network perform as you expected? Explain

My expectation is the result would be more like P_2, it is because the inputs are close to P_2 compare to P_1.

By applying the equation, where a = hardlims(w*p')

myWeight =

0.3333	0.3333	-0.3333	0
0.3333	0.3333	-0.3333	0
-0.3333	-0.3333	0.3333	0
0	0	0	1.0000

summation =

0.3333

0.3333

-0.3333

1.0000

axion =

1

1

-1

1

According to the result (axion), I verified that the output is equal to P_2.

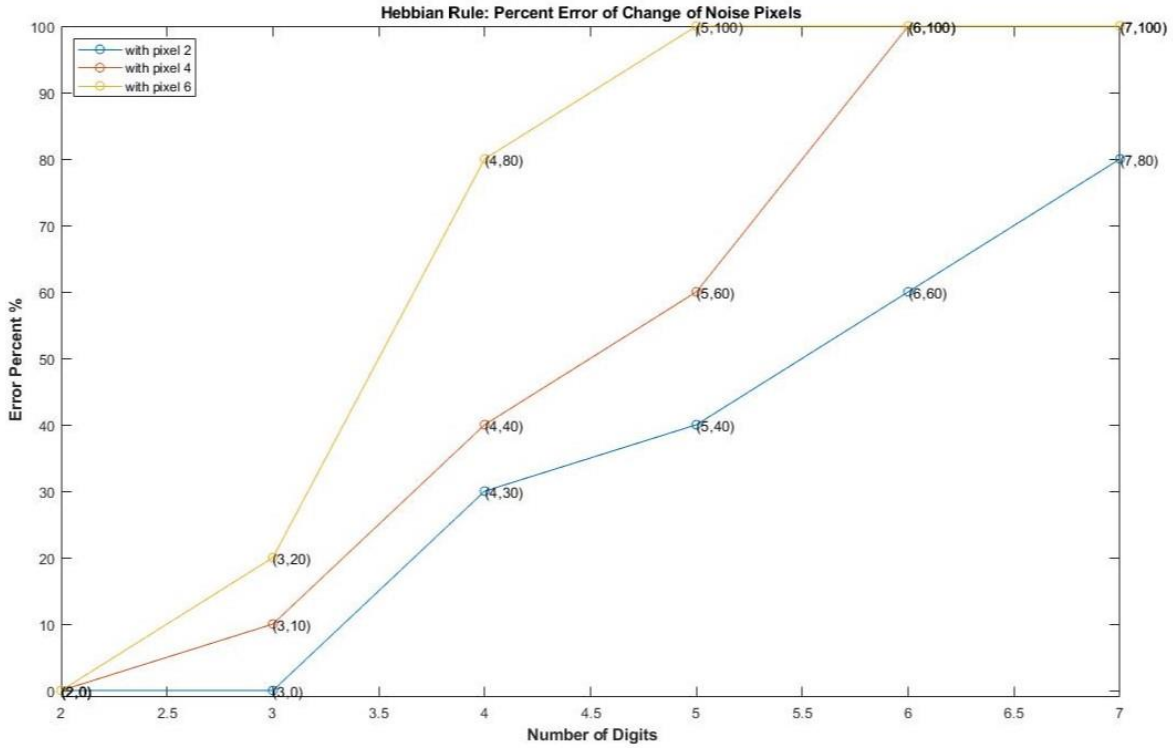
E 7.11) One question we might ask about the Hebb and pseudoinverse rules is: How many prototype patterns can be stored in one weight matrix? Test this experimentally using the digit recognition problem that was discussed on page 7-10. Begin with the digit "0" and "1". Add one digit at a time up to "6", and test how often the correct digit is reconstructed after randomly changing 2, 4 and 6 pixels.

i. First use the Hebb rule to create the weight matrix for the digits "0" and "1".

Then randomly change 2 pixels of each digit and apply the noisy digits to the network. Repeat this process 10 times and record the percentage of times in which the correct pattern (without noise) is produced at the output of the network. When you have completed all of the tests, you will be able to plot three curves showing percentage error versus number of digits stored, one curve each for 2,4, and 6pixel errors

Hebbian Rule

Figure 1.0: Graph Hebbian Rule



The figure 1.0 represents the error percentage corresponding of number of pixel change with associate number of digits. The x-axis represents the number of digits and y-axis represents percentage of error that is giving based on pixel changed of number of digits. As legend represents, the blue line represents change(noise)of 2 pixels, red line represents change(noise) of 4 pixels, and blue line represents pixel change(noise)of 6 pixels. All the error percentages are multiples of 10. Table 1.0, Table 1.1, and Table 1.2 below will show the exact error percentage associated with number of digits with pixel of noise.

Table 1.0: Percent of Error of Number Digits with 2 Pixel Noise

Digits	Pixel Changed	Error in Percentage
2 = [0, 1]	2	0%
3 = [0, 1, 2]	2	0%
4 = [0, 1, 2, 3]	2	30%
5 = [0, 1, 2, 3, 4]	2	40%
6 = [0, 1, 2, 3, 4, 5]	2	60%
7 = [0, 1, 2, 3, 4, 5, 6]	2	80%

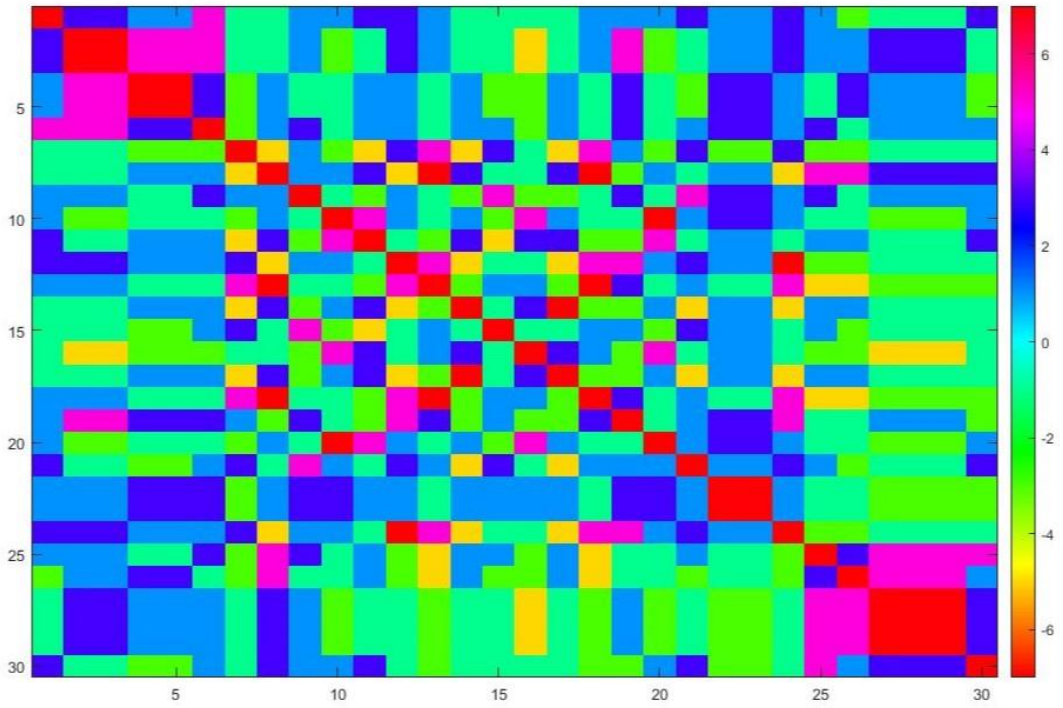
Table 1.1: Percent of Error of Number Digits with 4 Pixel Noise

Digits	Pixel Changed	Error in Percentage
2 = [0, 1]	4	0%
3 = [0, 1, 2]	4	10%
4 = [0, 1, 2, 3]	4	40%
5 = [0, 1, 2, 3, 4]	4	60%
6 = [0, 1, 2, 3, 4, 5]	4	100%
7 = [0, 1, 2, 3, 4, 5, 6]	4	100%

Table 1.2: Percent of Error of Number Digits with 6 Pixel Noise

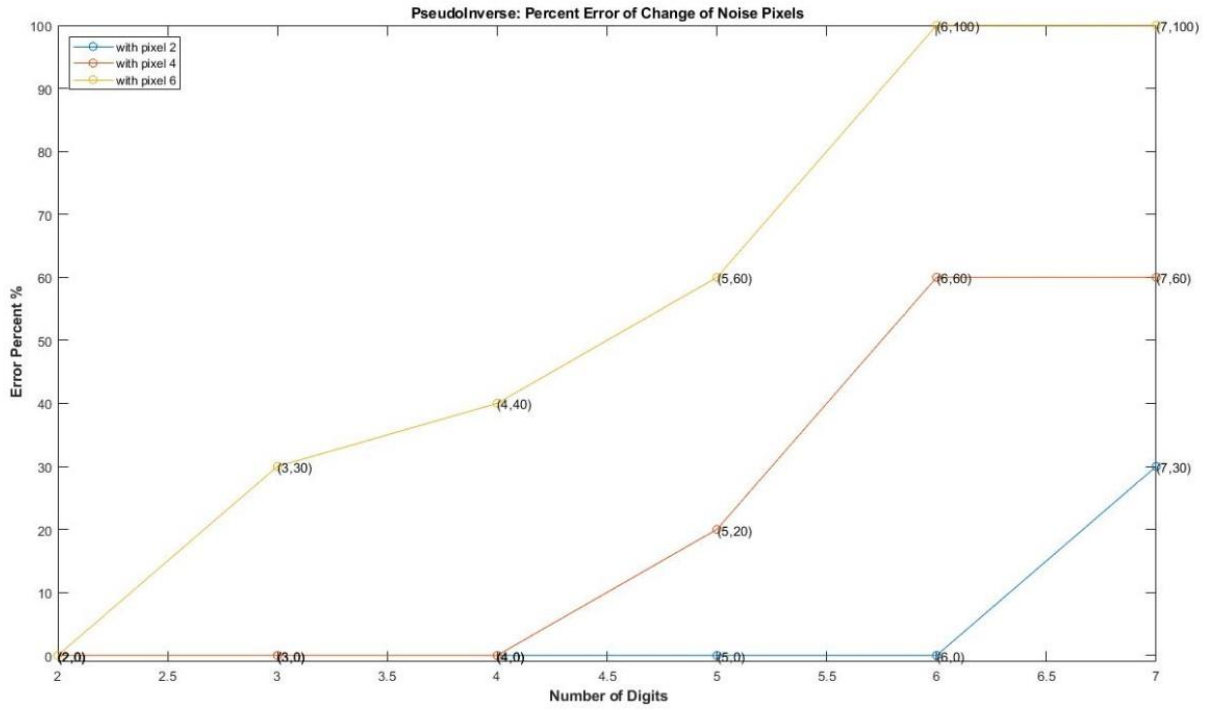
Digits	Pixel Changed	Error in Percentage
2 = [0, 1]	6	0%
3 = [0, 1, 2]	6	20%
4 = [0, 1, 2, 3]	6	80%
5 = [0, 1, 2, 3, 4]	6	100%
6 = [0, 1, 2, 3, 4, 5]	6	100%
7 = [0, 1, 2, 3, 4, 5, 6]	6	100%

Figure 1.1: Final weight of Hebbian rule



Repeat part (i) using the pseudoinverse rule, and compare the results of the two rules

Figure 2.0: Graph Pseudoinverse Rule



The figure 2.0 represents the error percentage corresponding of number of pixel change with associate number of digits. The x-axis represents the number of digits and y-axis represents percentage of error that is giving based on pixel changed of number of digits. As legend represents, the blue line represents change(noise)of 2 pixels, red line represents change(noise) of 4 pixels, and blue line represents pixel change(noise)of 6 pixels. All the error percentages are multiples of 10. Table 2.0, Table 2.1, and Table 2.2 below will show the exact error percentage associated with number of digits with pixel of noise.

Table 2.0: Percent of Error of Number Digits with 2 Pixel Noise

Digits	Pixel Changed	Error in Percentage
2 = [0, 1]	2	0%
3 = [0, 1, 2]	2	0%
4 = [0, 1, 2, 3]	2	0%
5 = [0, 1, 2, 3, 4]	2	0%
6 = [0, 1, 2, 3, 4, 5]	2	0%
7 = [0, 1, 2, 3, 4, 5, 6]	2	30%

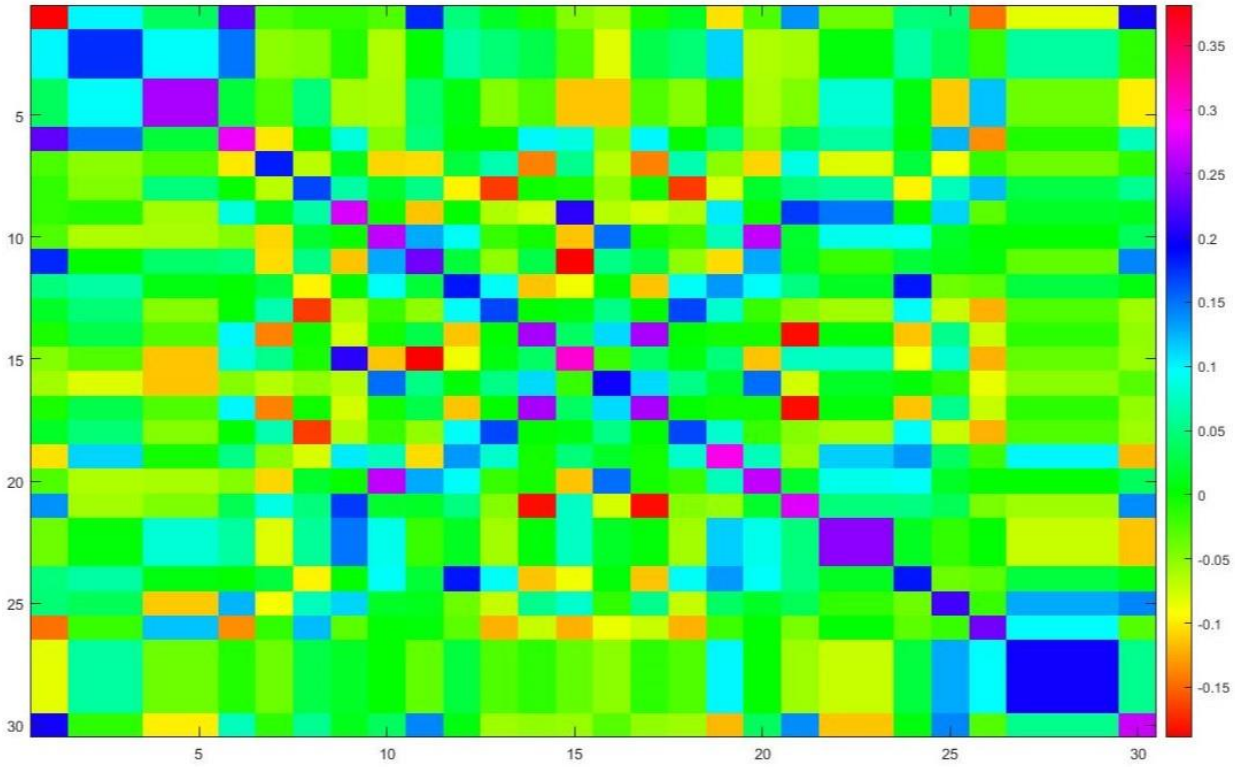
Table 2.1: Percent of Error of Number Digits with 4 Pixel Noise

Digits	Pixel Changed	Error in Percentage
2 = [0, 1]	4	0%
3 = [0, 1, 2]	4	0%
4 = [0, 1, 2, 3]	4	0%
5 = [0, 1, 2, 3, 4]	4	20%
6 = [0, 1, 2, 3, 4, 5]	4	60%
7 = [0, 1, 2, 3, 4, 5, 6]	4	60%

Table 2.2: Percent of Error of Number Digits with 6 Pixel Noise

Digits	Pixel Changed	Error in Percentage
2 = [0, 1]	6	0%
3 = [0, 1, 2]	6	30%
4 = [0, 1, 2, 3]	6	40%
5 = [0, 1, 2, 3, 4]	6	60%
6 = [0, 1, 2, 3, 4, 5]	6	100%
7 = [0, 1, 2, 3, 4, 5, 6]	6	100%

Figure 2.1: Final weight of Pseudoinverse rule



Conclusion:

As a conclusion, according to figure 1.0: Graph Hebbian Rule and figure 2.0: Graph Pseudoinverse rule, the output using Pseudoinverse rule is more accurate compare to the output that used Hebbian rule, I believe it is because it has less crazy weight figure (less error). However, both rules tend to have bigger error percentage when the inputs are increasing, it is because it is still single network and amount of data that single network can handle is limited.

Appendix

Code for Hebbian Rule

```
format compact
p0 = [-1 1 1 1 1 -1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 -1 1 1 1
1 -1]';
%length(p0)
```

```
p1 = [-1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1]';
%length(p1)
p2 = [1 -1 -1 -1 -1 -1 1 -1 -1 1 1 1 1 -1 -1 1 -1 1 -1 1 1 -1 -1 1 -1 -1 -1
-1 -1 1]';
%length(p2)
% create my 3
p3 = [-1 -1 -1 -1 -1 -1 1 -1 1 1 -1 1 1 -1 1 1 -1 1 1 1 1 1 1 1 -1 -1 -1 -1
-1 -1]';
%length(p3)
p4 = [-1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 1 1
1 1 1 1]';
%length(p4)
p5 = [1 1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 1 1
1]';
%length(p5)
p6 = [1 1 1 1 1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 1 1 1 -1 -1 -1 -1 -1
-1]';
%length(p6)

% Weight calculation
%target = [p0 p1 p2 p3];
    %p2 p3 p4 p5 p6];
%pTrans = target';
%myWeight = target * pTrans;
% test with the weight
testingWeight = p0*p0' + p1*p1' + p2*p2'+ p3*p3'+ p4* p4' + p5 * p5' + p6 *
p6';
% end of test
% my test variable for debugg purpose
showMeInput = [p0 p1 p2 p3 p4 p5 p6];
showMeTranspose = showMeInput';
showMeWeight = showMeInput * showMeTranspose;
showMeSummation = showMeWeight * showMeInput;
showMeAxion = hardlims(showMeSummation);
% End my test variable for debugg purpose

% test with the doCalculateWithInputV
inputDigit2 = [p0 p1];
inputDigit3 = [p0 p1 p2];
inputDigit4 = [p0 p1 p2 p3];
inputDigit5 = [p0 p1 p2 p3 p4];
inputDigit6 = [p0 p1 p2 p3 p4 p5];
inputDigit7 = [p0 p1 p2 p3 p4 p5 p6];

% all pixels Y value for graph and X value
xValue = [2 3 4 5 6 7]
% ============ 2 pixcel ==============
digit2Y2Pixel = doCalculateWithWeight(inputDigit2, 2)
digit3Y2Pixel = doCalculateWithWeight(inputDigit3, 2)
digit4Y2Pixel = doCalculateWithWeight(inputDigit4, 2)
digit5Y2Pixel = doCalculateWithWeight(inputDigit5, 2)
digit6Y2Pixel = doCalculateWithWeight(inputDigit6 ,2)
digit7Y2Pixel = doCalculateWithWeight(inputDigit7, 2)
yValuePix2 = [digit2Y2Pixel digit3Y2Pixel digit4Y2Pixel digit5Y2Pixel
digit6Y2Pixel digit7Y2Pixel]
```

```
% ============= 4 pixel ==============
digit2Y4Pixel = doCalculateWithWeight(inputDigit2, 4)
digit3Y4Pixel = doCalculateWithWeight(inputDigit3, 4)
digit4Y4Pixel = doCalculateWithWeight(inputDigit4, 4)
digit5Y4Pixel = doCalculateWithWeight(inputDigit5, 4)
digit6Y4Pixel = doCalculateWithWeight(inputDigit6, 4)
digit7Y4Pixel = doCalculateWithWeight(inputDigit7, 4)
yValuePix4 = [digit2Y4Pixel, digit3Y4Pixel, digit4Y4Pixel, digit5Y4Pixel,
digit6Y4Pixel, digit7Y4Pixel]
% ============= 6 pixel =============
digit2Y6Pixel = doCalculateWithWeight(inputDigit2, 6)
digit3Y6Pixel = doCalculateWithWeight(inputDigit3, 6)
digit4Y6Pixel = doCalculateWithWeight(inputDigit4, 6)
digit5Y6Pixel = doCalculateWithWeight(inputDigit5, 6)
digit6Y6Pixel = doCalculateWithWeight(inputDigit6, 6)
digit7Y6Pixel = doCalculateWithWeight(inputDigit7, 6)
yValuePix6 = [digit2Y6Pixel, digit3Y6Pixel, digit4Y6Pixel, digit5Y6Pixel,
digit6Y6Pixel, digit7Y6Pixel]
% ============= Graph ==================
plot(xValue, yValuePix2, '-o', 'DisplayName', 'with pixel 2')
hold on
plot(xValue, yValuePix4, '-o', 'DisplayName', 'with pixel 4')
hold on
plot(xValue, yValuePix6, '-o', 'DisplayName', 'with pixel 6')
legend('Location','northwest')
title('Hebbian Rule: Percent Error of Change of Noise Pixels')
xlabel(["\bf Number of Digits"])
ylabel(["\bf Error Percent %"])
ylim([-1 100])
% text for 2 px
for dis = 1:6
    text(dis+1, yValuePix2(dis), ['(' num2str(dis+1) ','
num2str(yValuePix2(dis)) ')'])
end
% end for 2 px
% text for 4 px
for dis = 1:6
    text(dis+1, yValuePix4(dis), ['(' num2str(dis+1) ','
num2str(yValuePix4(dis)) ')'])
end
% end for 4px
% text for 6px
for dis = 1:6
    text(dis+1, yValuePix6(dis), ['(' num2str(dis+1) ','
num2str(yValuePix6(dis)) ')'])
end
% end of 6px
hold off

figure()
imagesc(showMeWeight);
colormap(hsv);
colorbar;

function pvec = addNoise(pvec, num)
    % ADDNOISE Add noise to "binary" vector
```

```
    %   pvec pattern vector (-1 and 1)
    %   num  number of elements to flip randomly
    % Handle special case where there's no noise
    if num == 0
        return;
    end
    % first, generate a random permutation of all indices into pvec
    inds = randperm(length(pvec));
    % then, use the first n elements to flip
    pvec(inds(1:num)) = -pvec(inds(1:num));
end

% so now the weight calculation is dynamic
function count = doCalculateWithWeight(inputPx, numberOfPixelChange)
    targetNew = inputPx;
    pTranspose = targetNew';
    weight = targetNew * pTranspose;
    summation = weight * inputPx;
    pxAxion = hardlims(summation);
    counter = 0;
    wrongCounter = 0;
    totalCounter = 0;
    [row, col] = size(targetNew);
    %fprintf("Between\n");
        for i = 1:10
            for k = 1:col
                eachDigit = targetNew(:,k,:);
                length(eachDigit);
                % fprintf("Changing Now\n")
                tempPxho(:,k,:) = addNoise(eachDigit, numberOfPixelChange);
            end
            % fprintf("After Change\n")
            tempPxho;
            tempSummation = weight * tempPxho;
            tempAxion = hardlims(tempSummation)
            if pxAxion == tempAxion
                counter = counter + 1;
            else
                wrongCounter = wrongCounter + 1;
            end
            % reset the tempPx to original
            % somehow i have to reset the values
            % fprintf("After Reset\n")
            tempPxho = inputPx;
        end
        fprintf("Calcuation Start\n");
        wrongCounter = wrongCounter / 10;
        totalCounter =  wrongCounter + totalCounter;
        count = totalCounter * 100;
end
% end of test
```

Code for Pseudoinverse Rule

```
      format compact
```

```
% all of the inputs are ready to go
p0 = [-1 1 1 1 1 -1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 -1 1 1 1
1 -1]';
%length(p0)
p1 = [-1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1]';
%length(p1)
p2 = [1 -1 -1 -1 -1 -1 1 -1 -1 1 1 1 1 -1 -1 1 -1 1 -1 1 1 -1 -1 1 -1 -1 -1
-1 -1 1]';
%length(p2)
% create my 3
p3 = [-1 -1 -1 -1 -1 -1 1 -1 1 1 -1 1 1 -1 1 1 -1 1 1 1 1 1 1 1 -1 -1 -1 -1
-1 -1]';
%length(p3)
p4 = [-1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 1 1
1 1 1 1]';
%length(p4)
p5 = [1 1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 1 1
1]';
%length(p5)
p6 = [1 1 1 1 1 1 1 -1 1 -1 -1 1 1 -1 1 -1 -1 1 1 -1 1 1 1 1 -1 -1 -1 -1 -1
-1]';
% test if the value is same

% end of test
inputDigit2 = [p0 p1];
inputDigit3 = [p0 p1 p2];
inputDigit4 = [p0 p1 p2 p3];
inputDigit5 = [p0 p1 p2 p3 p4];
inputDigit6 = [p0 p1 p2 p3 p4 p5];
inputDigit7 = [p0 p1 p2 p3 p4 p5 p6];

xValue = [2 3 4 5 6 7]
% ============ 2 pixel ==============
digit2Y2Pixel = doCalculateWithWeight(inputDigit2, 2)
digit3Y2Pixel = doCalculateWithWeight(inputDigit3, 2)
digit4Y2Pixel = doCalculateWithWeight(inputDigit4, 2)
digit5Y2Pixel = doCalculateWithWeight(inputDigit5, 2)
digit6Y2Pixel = doCalculateWithWeight(inputDigit6 ,2)
digit7Y2Pixel = doCalculateWithWeight(inputDigit7, 2)
yValuePix2 = [digit2Y2Pixel digit3Y2Pixel digit4Y2Pixel digit5Y2Pixel
digit6Y2Pixel digit7Y2Pixel]
% ============ 4 pixel ==============
digit2Y4Pixel = doCalculateWithWeight(inputDigit2, 4)
digit3Y4Pixel = doCalculateWithWeight(inputDigit3, 4)
digit4Y4Pixel = doCalculateWithWeight(inputDigit4, 4)
digit5Y4Pixel = doCalculateWithWeight(inputDigit5, 4)
digit6Y4Pixel = doCalculateWithWeight(inputDigit6, 4)
digit7Y4Pixel = doCalculateWithWeight(inputDigit7, 4)
yValuePix4 = [digit2Y4Pixel, digit3Y4Pixel, digit4Y4Pixel, digit5Y4Pixel,
digit6Y4Pixel, digit7Y4Pixel]
% ============ 6 pixel ============
digit2Y6Pixel = doCalculateWithWeight(inputDigit2, 6)
digit3Y6Pixel = doCalculateWithWeight(inputDigit3, 6)
digit4Y6Pixel = doCalculateWithWeight(inputDigit4, 6)
digit5Y6Pixel = doCalculateWithWeight(inputDigit5, 6)
```

```
digit6Y6Pixel = doCalculateWithWeight(inputDigit6, 6)
digit7Y6Pixel = doCalculateWithWeight(inputDigit7, 6)
yValuePix6 = [digit2Y6Pixel, digit3Y6Pixel, digit4Y6Pixel, digit5Y6Pixel,
digit6Y6Pixel, digit7Y6Pixel]
% ============= Graph ==================
plot(xValue, yValuePix2, '-o', 'DisplayName', 'with pixel 2')
hold on
plot(xValue, yValuePix4, '-o', 'DisplayName', 'with pixel 4')
hold on
plot(xValue, yValuePix6, '-o', 'DisplayName', 'with pixel 6')
legend('Location','northwest')
title('PseudoInverse: Percent Error of Change of Noise Pixels')
xlabel(["\bf Number of Digits"])
ylabel(["\bf Error Percent %"])
ylim([-1 100])

% text for 2 px
for dis = 1:6
    text(dis+1, yValuePix2(dis), ['(' num2str(dis+1) ','
num2str(yValuePix2(dis)) ')'])
end
% end for 2 px
% text for 4 px
for dis = 1:6
    text(dis+1, yValuePix4(dis), ['(' num2str(dis+1) ','
num2str(yValuePix4(dis)) ')'])
end
% end for 4px
% text for 6px
for dis = 1:6
    text(dis+1, yValuePix6(dis), ['(' num2str(dis+1) ','
num2str(yValuePix6(dis)) ')'])
end
% end of 6px

hold off

% test if the value is same
target = [p0 p1 p2 p3 p4 p5 p6];
pPlus = pinv(target);
weight = target * pPlus;
figure()
imagesc(weight);
colormap(hsv);
colorbar;

function pvec = addNoise(pvec, num)
    % ADDNOISE Add noise to "binary" vector
    %   pvec pattern vector (-1 and 1)
    %   num  number of elements to flip randomly
    % Handle special case where there's no noise
    if num == 0
        return;
    end
    % first, generate a random permutation of all indices into pvec
    inds = randperm(length(pvec));
```

```
    % then, use the first n elements to flip
    pvec(inds(1:num)) = -pvec(inds(1:num));
end

% change the weight calculation to pseudoInverse
function count = doCalculateWithWeight(inputPx, numberOfPixelChange)
    targetNew = inputPx;
    pInverse = pinv(targetNew);
    weight = targetNew * pInverse;
    summation = weight * inputPx;
    pxAxion = hardlims(summation);
    counter = 0;
    wrongCounter = 0;
    totalCounter = 0;
    [row, col] = size(targetNew);
    %fprintf("Between\n");
        for i = 1:10
            for k = 1:col
                eachDigit = targetNew(:,k,:);
                length(eachDigit);
                % fprintf("Changing Now\n")
                tempPxho(:,k,:) = addNoise(eachDigit, numberOfPixelChange);
            end
            % fprintf("After Change\n")
            tempPxho;
            tempSummation = weight * tempPxho;
            tempAxion = hardlims(tempSummation);
            if pxAxion == tempAxion
                counter = counter + 1;
            else
                wrongCounter = wrongCounter + 1;
            end
            % reset the tempPx to original
            % somehow i have to reset the values
            % fprintf("After Reset\n")
            tempPxho = inputPx;
        end
        fprintf("Calcuation Start\n");
        wrongCounter = wrongCounter / 10;
        totalCounter =  wrongCounter + totalCounter;
        count = totalCounter * 100;
end
```