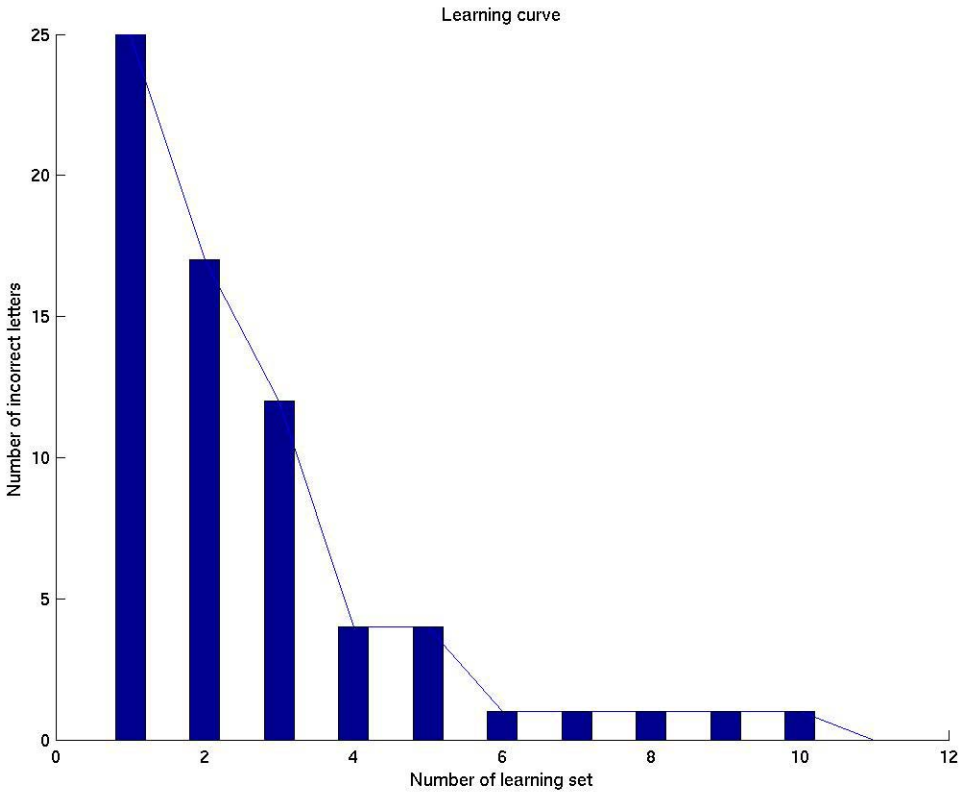
QUESTION 1:

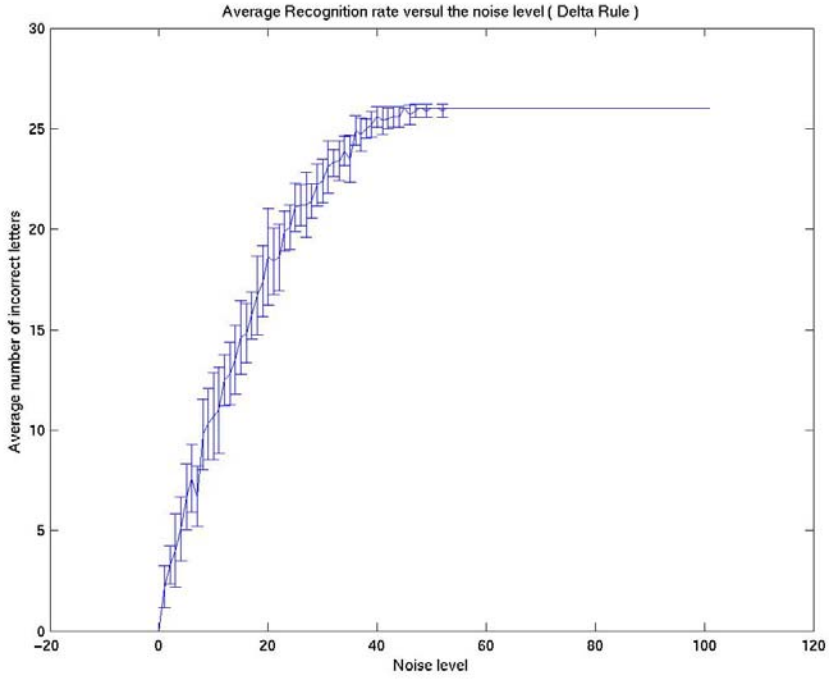


[Program: DeltaRule.m **]**

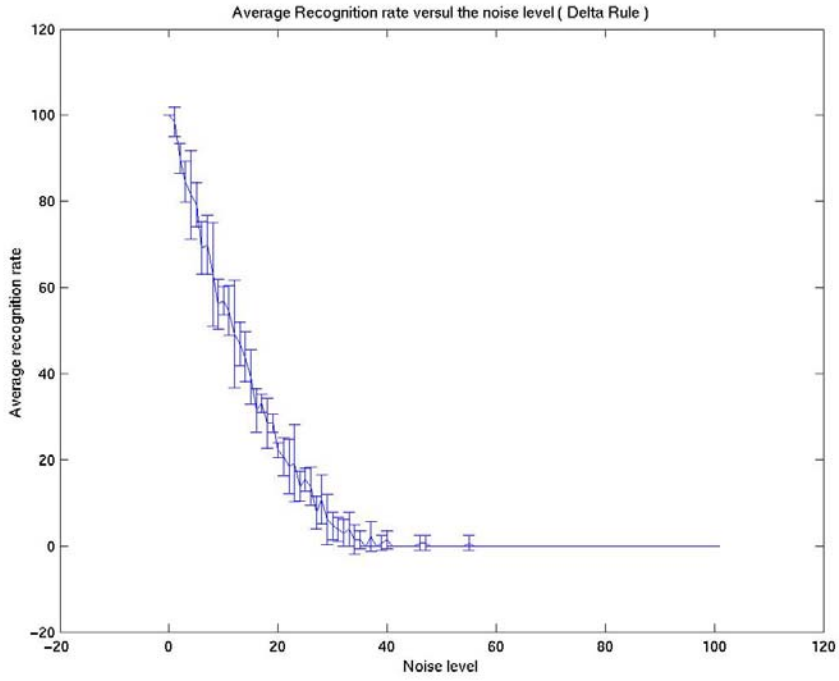
QUESTION 2:

Evaluate the robustness of the network in recognizing noisy versions of the letter patterns. Plot a curve that shows the average recognition rate versus the noise level. The plot should include errorbars.

ANSWER:



X axis means the Noise level of input
Y axis shows the average number of incorrect letters with the errorbars showing the deviation for total 50 experiments



[Program: EvaluationDeltaRule.m]

1. The curve shows the trend that incorrect recognition letters increase with the increase of input noise level.

2. At the beginning, with a noise value of zero, the correctness of recognition is 100%.

3. When the input noise value increased from 0 to 50%, the number of incorrect letters goes up sharply.

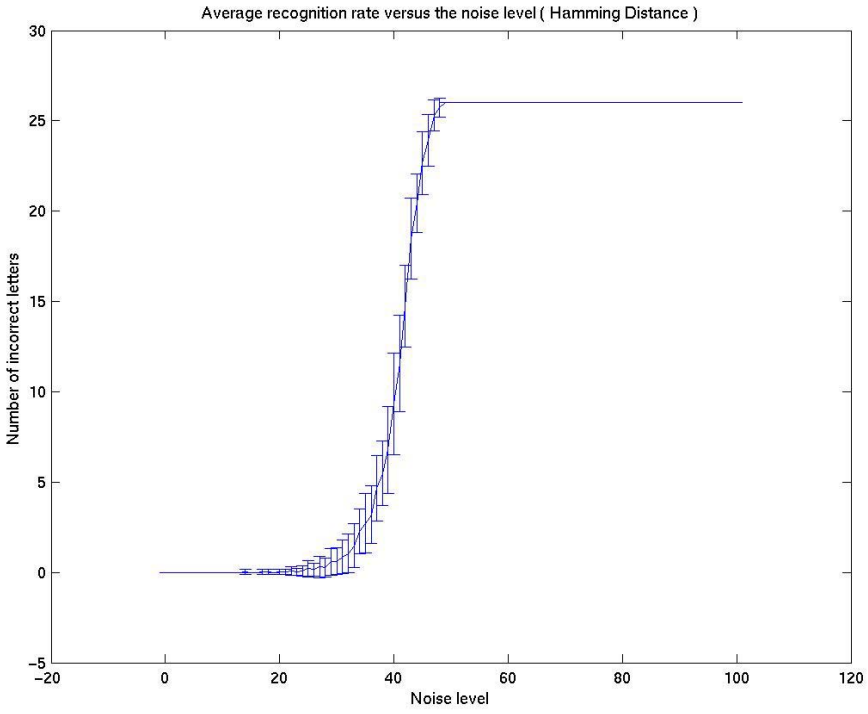
4. The network cannot recognize the input at all when the noise level is greater than 50%, in other words, it is totally an invalid network then.

5. We also try to change the recognition network with different initial weights, and to repeat the experiments, the trends of the curve are almost same.

6. Obviously, it is not a robust network in recognizing noisy versions of the letter patterns.

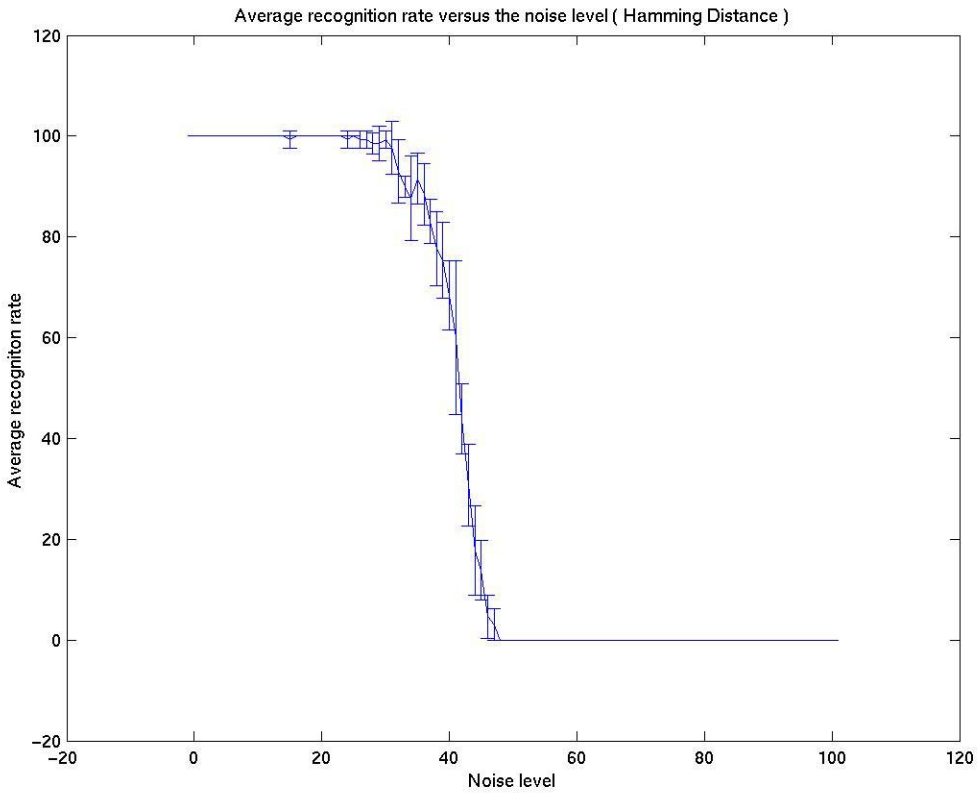
QUESTION 3:

Similarly, evaluate the robustness of your solution of the pattern recognition in tutorial 1. Discuss briefly the results of 2 and 3.



X axis means the Noise level of input

Y axis shows the average number of incorrect letters with the errorbars showing the deviation for total 50 experiments



[Program: EvaluationHammingDistance.m]

ANSWER:

1. According to the curve, the number of incorrect recognition letters increase with the increase of input noise level.

2. When the noise level is less than 20%, it works perfectly, the incorrect letter number is almost zero.

3. It is still an efficient network when the noise level is between 20% and 40%, only less than 5 letters cannot be recognized.

4. But as the noise level goes from 40% to 50%, the efficiency of the network decreases sharply, and the error recognition number goes up rapidly to the maximum.

5. The network completely does not work as the noise level is greater than 50%.

6. It is a robust network when the noise level is under 40%.

As we can see from the results of 2 and 3, “Hamming Distance” can recognize all input patterns when the noise level is less than 20%, however “Delta Rule” will get error output even with minor noise. Also, “Hamming Distance” is more robust when the noise level is less than 40%. But when the noise level is greater than 40%, they both cannot recognize all input patterns.

QUESTION4:

Discuss briefly the advantages and disadvantages of the two methods.

ANSWER:

Advantage of Delta Rule:

- The delta rule is an error-driven learning algorithm. It’s similar to Human learning process. Also it’s simple for implement and is fast to recognize input patterns.

- The neural network will learn more rapidly if the learning rate is high, but the network will bounce around if the rate is too high. The network will take more time to learn for the small learning rate, but it will get a more effective solution. In conclusion, it can be a fast learning rule.

Disadvantage of Delta Rule:

- Need training, especially, if the move in the direction of the gradient of error is very small, the Delta rule algorithm will turn to be very slow

- The exactly same initial weight matrix and input patterns could have different results

- If target values cannot be known before training, delta can’t be calculated.

- Low recognition rate for the input patterns with noise

- The delta rule has only two layers of processing units, and cannot get Delta for uncertain middle layers in multiple layer networks.

Advantage of Hamming Distance:

- High recognition rate for the input patterns within certain noise level
- The algorithm is simple and easy to implement

Disadvantage of Hamming Distance

- It need to compare outputs with all desired outputs, if the desired outputs set is large, it will be very slow.
- Not flexible and not suitable for complex recognition

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Delta rule pattern recognition %
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

% Reading pattern from pattern file

load pattern1;

% Reading desired output matrix file
load desiredOutputMatrix;

% Reshaping pattern to 156*26
letterVectors=reshape(pattern1', 12*13, 26);

% Set learning constant and number of training
learningConstant=0.1;
trainingNumber=1;

disp('Begin training, please wait!');

% Initialize weights to random values, 26*156
weights=rand(26,156)*1.5;

% repeat until the error number is zero
errorNumber=26;
while errorNumber>0

    for letterNumber=1:26

        % Apply sample patterns to the input nodes
        inputLetterVector=letterVectors(:,letterNumber); % 156*1

        % Cauculate the rate of output nodes
        outputLetterVector=weights*inputLetterVector; % 26*1

        % Apply game function to the output nodes
        for i=1:26
            outputLetterVector(i)=gameFunction(outputLetterVector(i));
        end

        % Compute the delta term for the output layer
        deltaLetterVector=desiredOutputMatrix(:,letterNumber)-
outputLetterVector; % 26*1
        deltaWeights=deltaLetterVector*inputLetterVector'; % 26*156

        % Update the weights matrix by adding the delta term
        weights=weights+deltaWeights;

    end


    % Test output with trained weight matrix
    errorNumber=0;
```

```
    for letterNumber=1:26

        % Apply sample patterns to the input nodes
        inputLetterVector=letterVectors(:,letterNumber);

        % Cauculate the rate of output nodes
        outputVector=weights*inputLetterVector;

        % Apply game function to the output nodes
        for i=1:26
            outputVector(i)= gameFunction(outputVector(i));
        end

        % Compare the output with the desired output, update error
number variable
        if ~isequal(desiredOutputMatrix(:,letterNumber),outputVector)
            % error output
            errorNumber=errorNumber+1;
        end
    end

    % Update the learning curve X Y set
    X(trainingNumber)=trainingNumber;
    Y(trainingNumber)=errorNumber;

    % Add the number of training set
    trainingNumber=trainingNumber+1;
end

% Plot the learning curve
hold on;
plot(X,Y);
bar(X,Y,0.4);
hold off;
title('Learning curve');
xlabel('Number of learning set');
ylabel('Number of incorrect letters');

disp('End of training, enjoy it.');

% Save weight matrix  to file 'w'
save('weights','weights','-ASCII');




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Game Function                 %
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function g=gameFunction(x)
% Game fuction of delta rule
% return 1 if x>0.45
% return 0 if x<=0.45
if x>0.45,
```

```
    g=1;
else
    g=0;
end



function r=ranBinVec(len,n1);

%

% r=ranBinVec(len,n1)

% returns vector of length len with n1 ones at random positions

%

r=zeros(len,1);
flag=zeros(len,1);

i=1;

while i<=n1;

    ind=floor(len*rand)+1;
    if flag(ind)==0;

        r(ind)=1;
        flag(ind)=1;
        i=i+1;

    end

end







%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Evaluation of Delta rule pattern recognition %
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

% Reading pattern from pattern file
load pattern1;

% Reading desired output matrix file
load desiredOutputMatrix;

% Reading weight matrix
```

```
load weights;

% Reshaping pattern to 156*26
letterVectors=reshape(pattern1', 12*13, 26);

disp('Begin testing input with noise!');

% Set the repeat test number of certain noise level
totalLoopNumber=50;

timeStart=now;
for noise=0:100
noise
    X(noise+1)=noise;
    for loopIndex=1:totalLoopNumber

        % Initialize the error number
        errorNumber=0;

        % Apply sample patterns to the input nodes
        inputLetterVectors=letterVectors;

        % Make nosie version
        for i=1:26
            noiseVec=ranBinVec(156,noise*156/100);

inputLetterVectors(noiseVec==1,i)=inputLetterVectors(noiseVec==1,i)==0;
        end

        % Cauculate the rate of output nodes
        outputVectors=weights*inputLetterVectors; % 26*26

        for letterNumber=1:26

            % Apply game function to the output nodes
            for i=1:26

outputVectors(i,letterNumber)=gameFunction(outputVectors(i,letterNumber
));
            end

            % Compare the output with the desired output, update error
number variable
            if
~isequal(desiredOutputMatrix(:,letterNumber),outputVectors(:,letterNumb
er))
                % Error output
                errorNumber=errorNumber+1;
            end

        end

        % Update the recognition rate versus noisie level curve X Y set

        Y(loopIndex,noise+1)=errorNumber;
    end
end;
```

```
timeEnd=now;
disp('Total time used for recognitions: ');
disp(datestr(timeEnd-timeStart,'HH:MM:SS'));

% Compute the average error number
AverageY=mean(Y);

% Comput the standard deviation
S=std(Y);

% Plot the error bar
errorbar(X,AverageY,S);
title('Average Recognition rate versul the noise level ( Delta Rule
)');
xlabel('Noise level');
ylabel('Average number of incorrect letters');

disp('End of testing input with noise.');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Evaluation of simple hamming distance pattern recognition  %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;

% Reading pattern from pattern file
load pattern1

% Reshaping pattern to 156*26
letterVectors=reshape(pattern1', 12*13, 26);

% Select one letter and display it

disp('Begin testing input with noise.');

% Set the repeat test number of certain noise level
loopNumber=50;

timeStart=now;

for noise=0:100

    X(noise+1)=noise;
    for loopIndex=1:loopNumber

        errorNumber=0;
        for letterNumber=1:26
            % Apply sample patterns to the input nodes
            thisLetterVector=letterVectors(:,letterNumber);

            % Make noisy version
```

```
            noiseVec=ranBinVec(156,noise*156/100);

thisLetterVector(noiseVec==1)=thisLetterVector(noiseVec==1)==0;

            % Calculate hamming distance between the noisy letter and
letters in the data base
            for i=1:26
                h(i)=sum((letterVectors(:,i)-thisLetterVector).^2);
            end

            % Calculate best match
            [tmp,i]=min(h);
            % Compare the output with the input, update error number
variable
            if i~=letterNumber
                errorNumber=errorNumber+1;
            end
        end
        Y(loopIndex,noise+1)=errorNumber;

    end
end

timeEnd=now;
disp('Total time used for recognitions: ');
disp(datestr(timeEnd-timeStart,'HH:MM:SS'));

% Compute the average error number
AverageY=mean(Y);

% Comput the standard deviation
S=std(Y);

% Plot the error bar
errorbar(X,AverageY,S);
title('Average recognition rate versus the noise level ( Hamming
Distance )');
xlabel('Noise level');
ylabel('Number of incorrect letters');

disp('End of testing input with noise.');
```