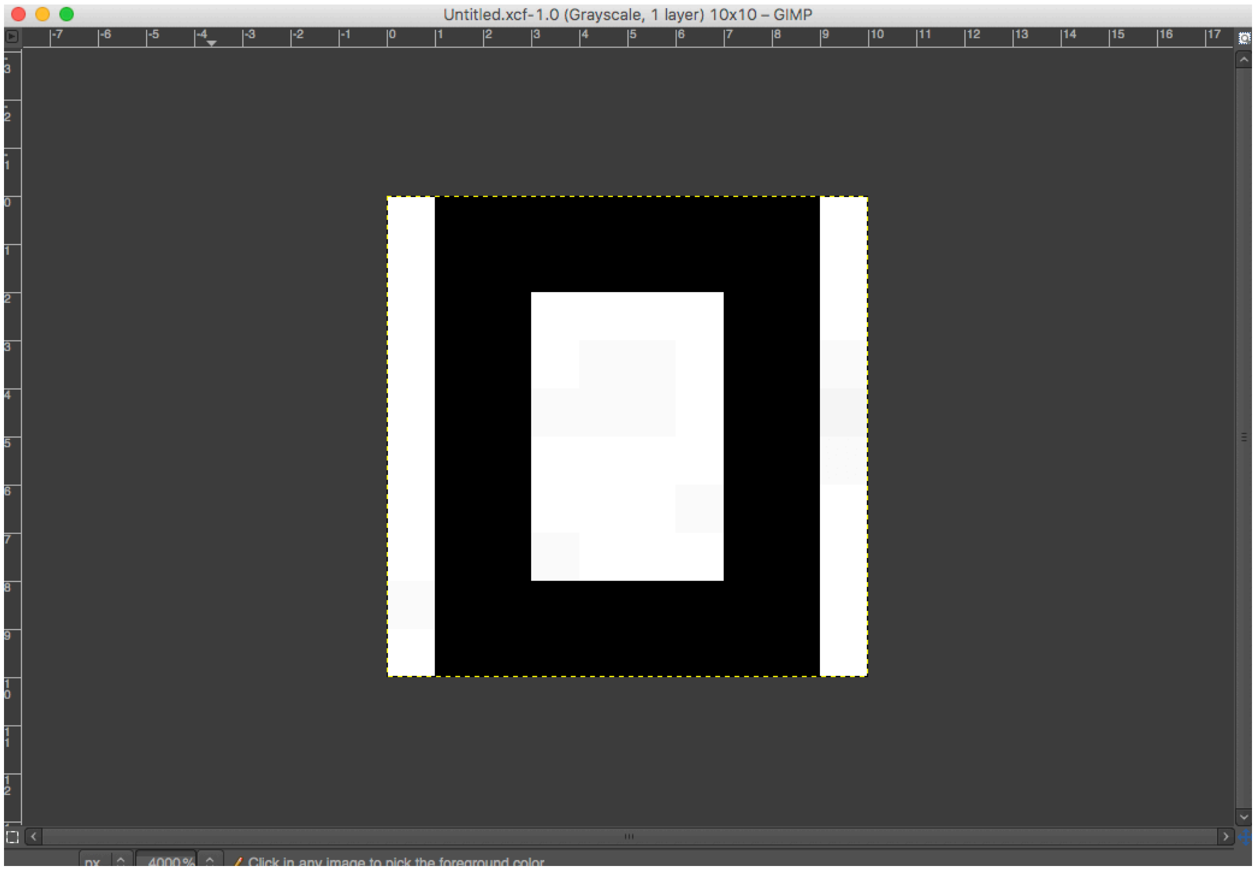
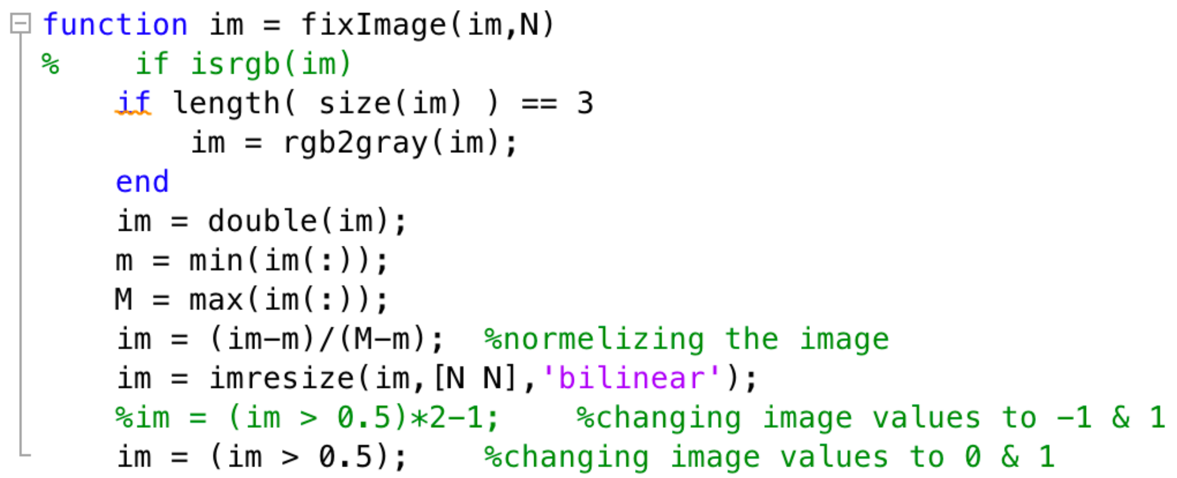
1)

a) Letters for these experiments were produced using a bitmap photo manipulation program (in this case “Gimp”). Using 10x10 and 25x25 defined grids, the letters were filled in pixel wise, saved, and then converted to binary or bipolar arrays using Matlab. Thus saved considerable time opposed to manually entering 1’s, 0’s, -1’s, and 1’s into arrays. The process is defined below:

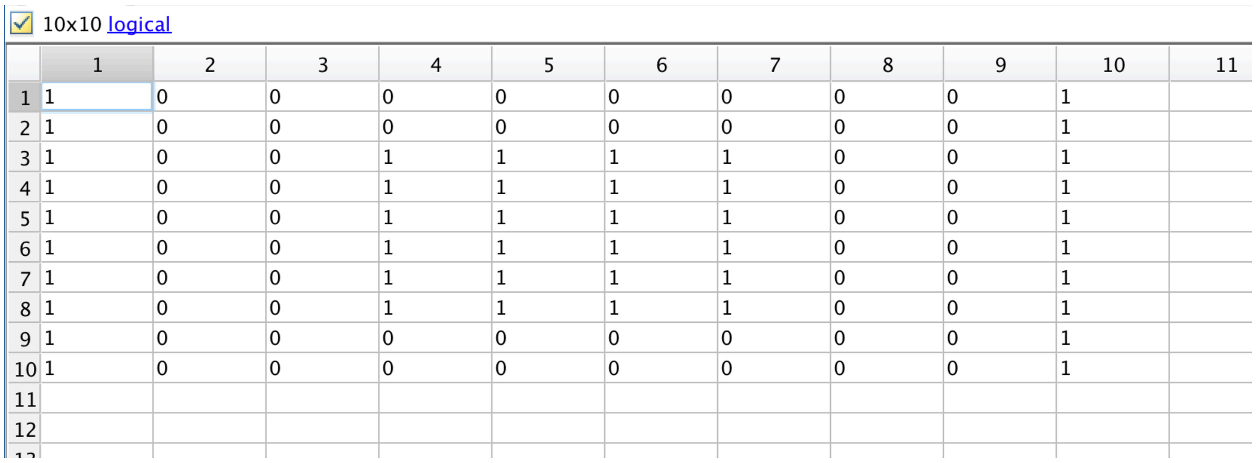

Untitled.xcf-1.0 (Grayscale, 1 layer) 10x10 – GIMP
px
4000 %
Click in any image to pick the foreground color

Here we see the bitmap canvass where the letters were “drawn,” in this case its the letter “O” in a 10x10 grid. From this step the file was saved as O.bmp and imported into Matlab. Using the following function the bitmap images were converted into bipolar and binary arrays:



```
function im = fixImage(im,N)
%     if isrgb(im)
    if length( size(im) ) == 3
        im = rgb2gray(im);
    end
    im = double(im);
    m = min(im(:));
    M = max(im(:));
    im = (im-m)/(M-m);  %normelizing the image
    im = imresize(im,[N N],'bilinear');
    %im = (im > 0.5)*2-1;    %changing image values to -1 & 1
    im = (im > 0.5);    %changing image values to 0 & 1
```

After this step we come we are left with the array (in this example binary) to run through the network:



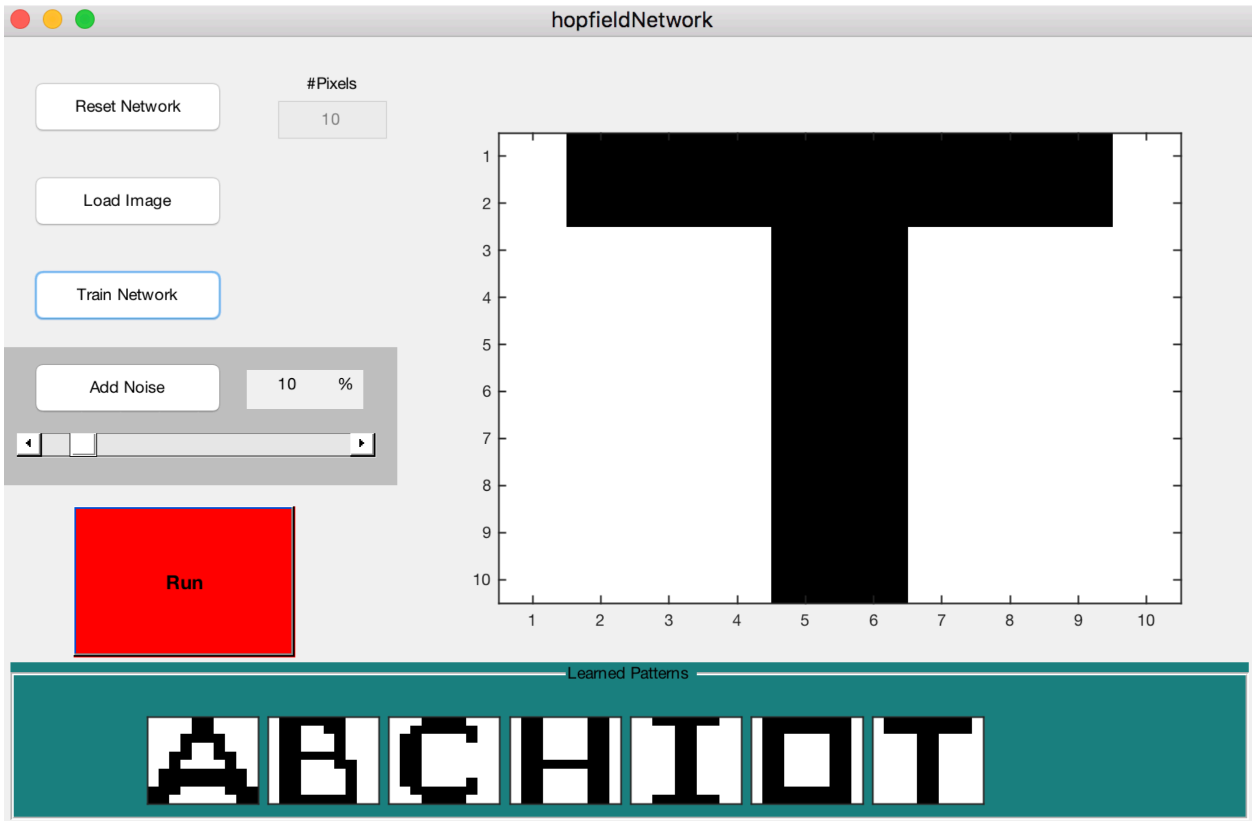
10x10 logical

	1	2	3	4	5	6	7	8	9	10	11
1	1	0	0	0	0	0	0	0	0	1	
2	1	0	0	0	0	0	0	0	0	1	
3	1	0	0	1	1	1	1	0	0	1	
4	1	0	0	1	1	1	1	0	0	1	
5	1	0	0	1	1	1	1	0	0	1	
6	1	0	0	1	1	1	1	0	0	1	
7	1	0	0	1	1	1	1	0	0	1	
8	1	0	0	1	1	1	1	0	0	1	
9	1	0	0	0	0	0	0	0	0	1	
10	1	0	0	0	0	0	0	0	0	1	
11											
12											

The array can be graphed and saved as a fig file in Matlab with the imshow() function:

This was done for letters: A, B, C, H, I, O, T in both 10x10 and 25x25 grids. **All arrays (binary/ bipolar) and graphed figure files are stored on professors drive.**

b) Next the two sets of letters were trained into the network, neurons were set to 10 and 25 respectively due to the grid size:



A GUI was created in order to organize and train the network easier, the code however is:

```
function train_Callback(hObject, eventdata, handles)

  Npattern = length(handles.hPatternsDisplay);
  if Npattern > 9
     msgbox('more then 10 paterns isn''t supported!','error');
     return
  end

  im = getimage(handles.neurons);
  disp(im);
  %save('noise10percent','im');
  %I = mat2gray(im);
  %IMg = imshow(I);
```

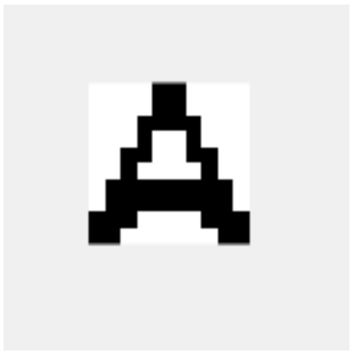
```
N = get(handles.imageSize,'string');
N = str2num(N);
W = handles.W;  %weights vector
avg = mean(im(:));   %removing the cross talk part
if ~isempty(W)
   %W = W +( kron(im,im))/(N^2);
   W = W + ( kron(im-avg,im-avg))/(N^2)/avg/(1-avg);
else
   % W = kron(im,im)/(N^2);
   W = ( kron(im-avg,im-avg))/(N^2)/avg/(1-avg);
end
% Erasing self weight
ind = 1:N^2;
f = find(mod(ind,N+1)==1);
W(ind(f),ind(f)) = 0;

handles.W = W;

% Placing the new pattern in the figure...
xStart = 0.01;
xEnd = 0.99;
height = 0.65;
width = 0.09;
xLength = xEnd-xStart;
xStep = xLength/10;
offset = 4-ceil(Npattern/2);
offset = max(offset,0);
y = 0.1;

if Npattern > 0
   for n=1 : Npattern
      x = xStart+(n+offset-1)*xStep;
      h = handles.hPatternsDisplay(n);
      set(h,'units','normalized');
      set(h,'position',[x y width height]);
   end
   x = xStart+(n+offset)*xStep;
   h = axes('units','normalized','position',[x y width height]);
   handles.hPatternsDisplay(n+1) = h;
   imagesc(im,'Parent',h);
else
   x = xStart+(offset)*xStep;
   h = axes('units','normalized','position',[x y width height]);
   handles.hPatternsDisplay = h;
end

imagesc(im,'Parent',h);
set(h, 'YTick',[],'XTick',[],'XTickMode','manual','Parent',handles.learnedPaterns);
```

```
    guidata(hObject, handles);
```

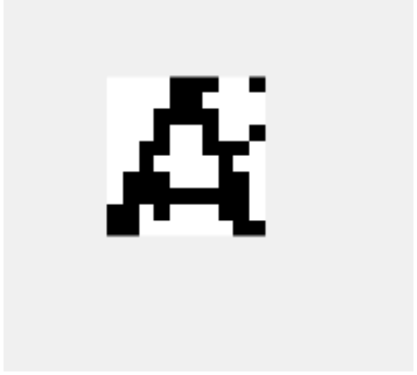
For the 25x25 letters the process was the same. Each pattern was trained into the network.

c) For each letter a certain amount of noise was generated. First 10%, then 15%, and so on till 35%. Then using these test samples full of noise based off the original letter we tested the hopfeild network to see where it will converge. We could end up with the correct letter, a different letter, or some local minimum that is neither. So these experiments will show how noise affects the hopfeild network. There are **100's** of experiments so **all the results will be recoded and stored on the professors drive**, while only a couple will be documented in this paper. Results described in this paper are cases that show the characteristics of the network.

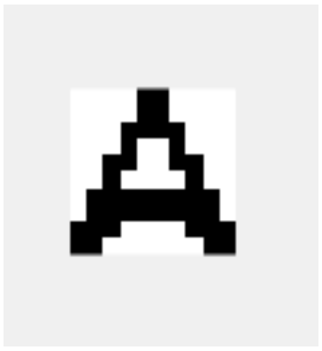
Experiment 1: 10x10 letters with 10% noise on letter A



The network has been trained with all seven letters, then the base image "A" was run through the network with 10% noise added:

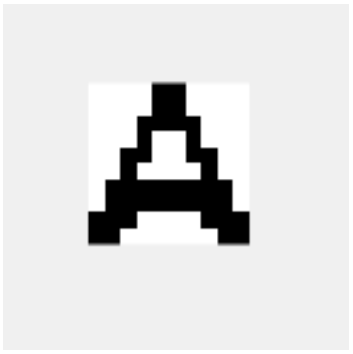


When tested, the network was able to come up with this result:

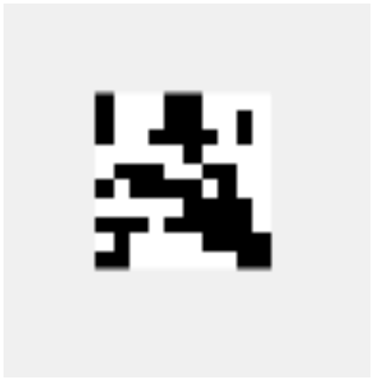


The network was able to converge on the correct pattern easily.

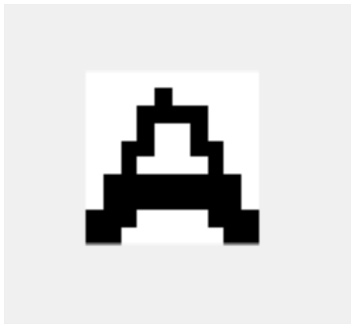
<u>Experiment 2: 10x10 letters with 25% noise on letter A</u>



The network has been trained with all seven letters, then the base image “A” was run through the network with 25% noise added:

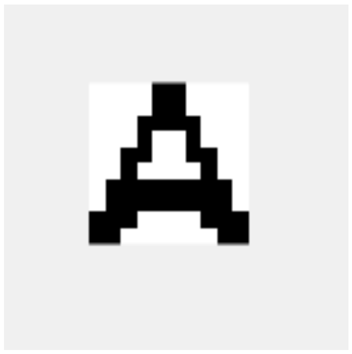


When tested, the network was able to come up with this result:

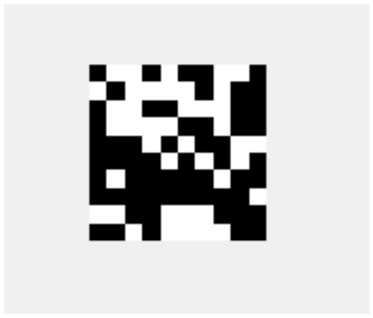


Here we see that although close the network could not match the pattern exactly. As the amount of noise increases, along with the number of learned patterns, the network can become less “sure” of a pattern.

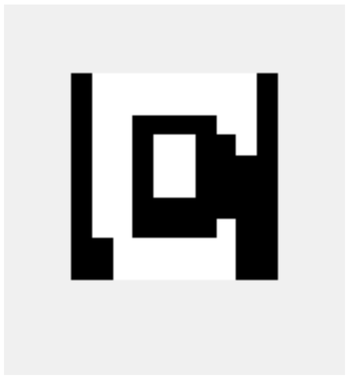
<u>Experiment 3: 10x10 letters with 35% noise on letter A</u>



The network has been trained with all seven letters, then the base image “A” was run through the network with 35% noise added:

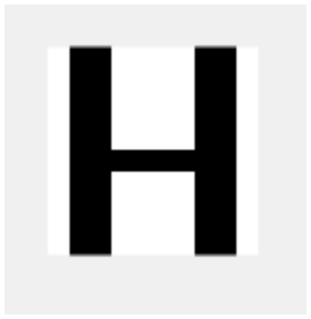


When tested, the network was able to come up with this result:

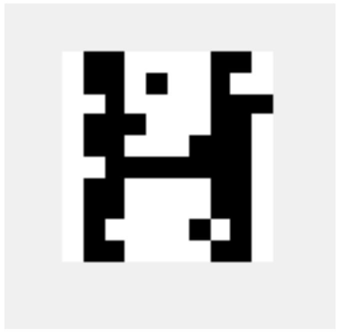


With 35% noise is clear that with the training of other patterns along with the high percentage of noise, the pattern of A is not recognizable.

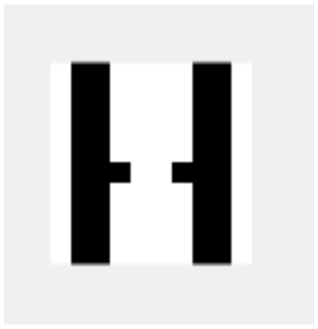
Experiment 4: 10x10 letters with 10% noise on letter H



The network has been trained with all seven letters, then the base image “H” was run through the network with 10% noise added:

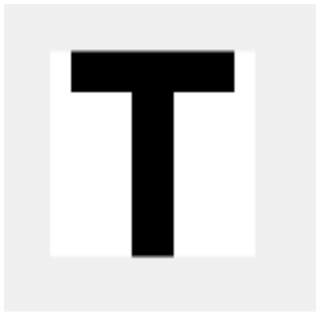


When tested, the network was able to come up with this result:

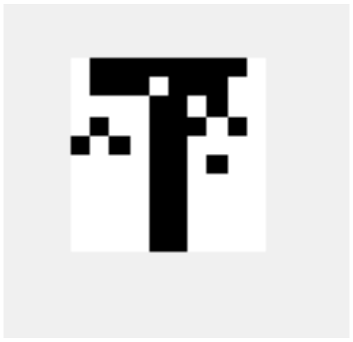


We see that the network did not converge on the ideal pattern. “H” is interesting as noise increases so more experiments were done.This showed that too much noise will give an unrecognizable pattern, but so will too little. The above pattern appeared more often than not in the results.

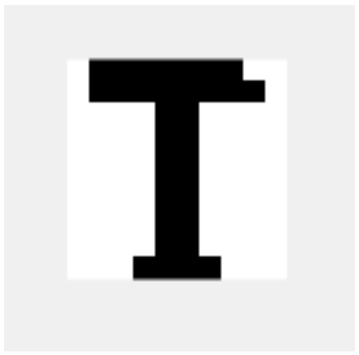
<u>Experiment 5: 10x10 letters with 10% noise on letter T</u>



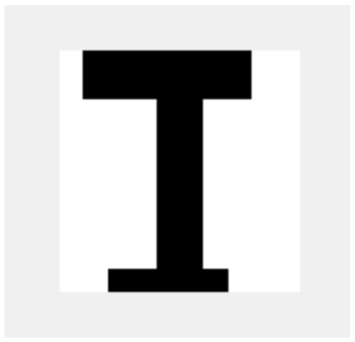
The network has been trained with all seven letters, then the base image “T” was run through the network with 10% noise added:



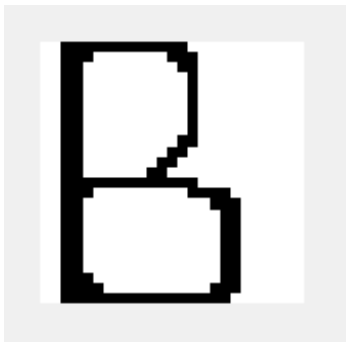
When tested this was the result:



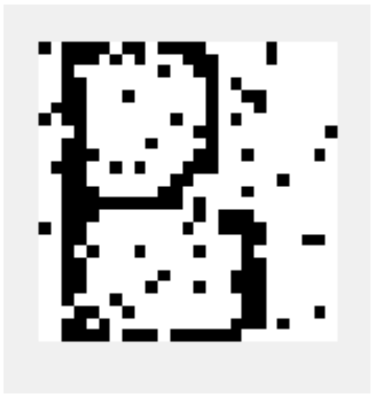
As you can see even with 10% noise the network cannot match the pattern perfectly. There are several reasons for this: First is because the learned patterns for “T” and an “I” are quite similar. Thus when noise is introduced, the network has a difficult time converging on the correct pattern in order to differentiate the two. This effect will worsen, this is the result at 35% noise:



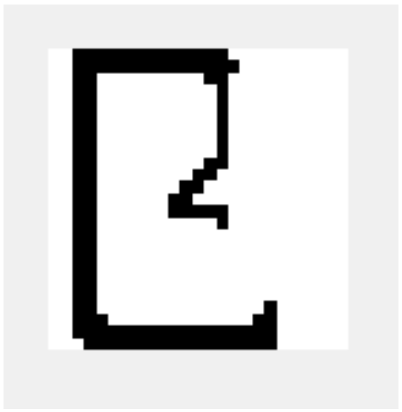
<u>Experiment 6: 25x25 letters with 10% noise on letter B</u>



These experiments now contain letters on a 25x25 grid, there are more data elements in the arrays that make up the letters. This should have an impact on the results. These experiments are being done on the network trained with the 7, 25x25 letters. This one in particular is the Letter “b” with 20% noise:

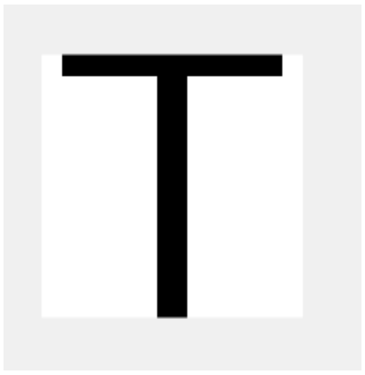


When run though the network the result was this:

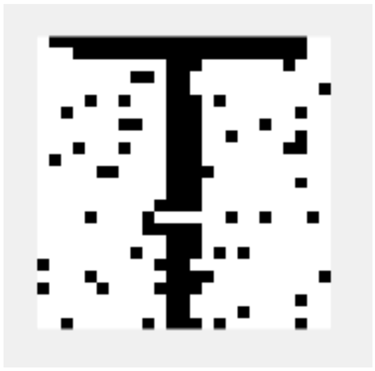


Its clear that here network was able to clear some of the noise and derive a decent pattern, but it could not accurately produce the original pattern. As the amount of noise increased this result remains essentially the same.

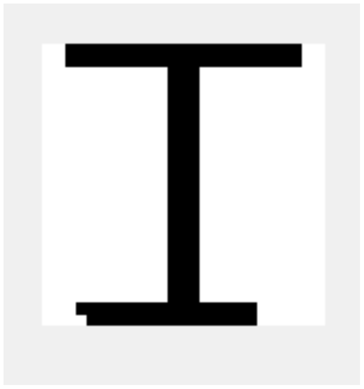
Experiment 7: 25x25 letters with 10% noise on letter T



The network has been trained with all seven letters, then the base image “T” was run through the network with 10% noise added:

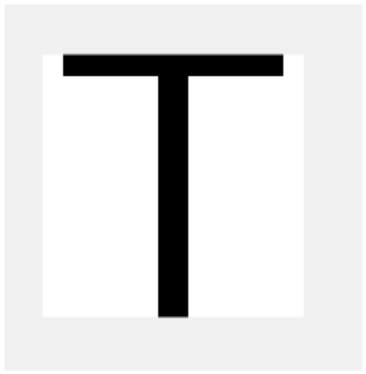


When run though the network this was the result:

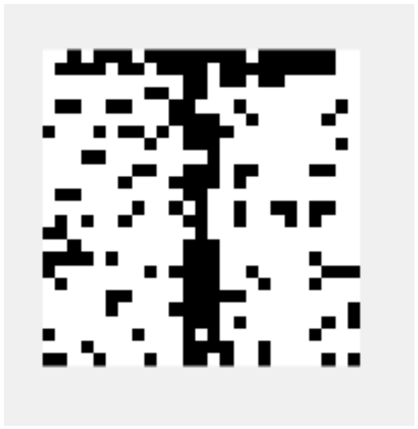


We see that we are running into he same problem as the 10x10 letters where the “T” closely resembles “I” for there to be a problem in getting the right pattern.

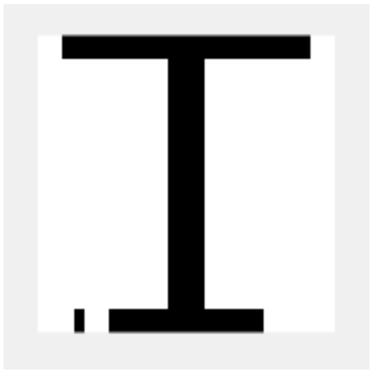
Experiment 8: 25x25 letters with 20% noise on letter T



The network has been trained with all seven letters, then the base image “T” was run through the network with 20% noise added:

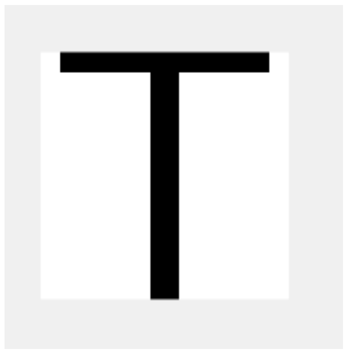


Run through the network we have a result of:

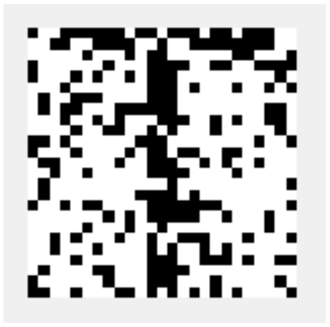


This is relatively the same result as the 10% noise.

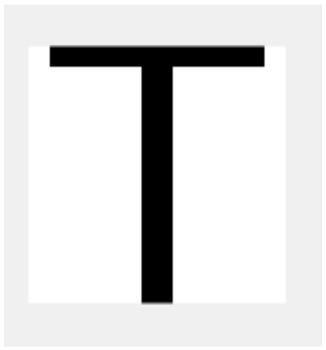
Experiment 9: 25x25 letters with 30% noise on letter T



The network has been trained with all seven letters, then the base image “T” was run through the network with 30% noise added:

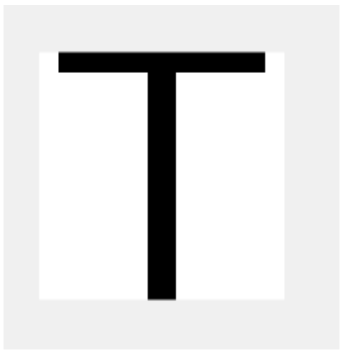


Run though the network we gain this result:

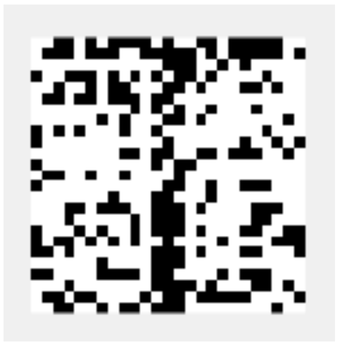


This is interesting because with more noise the recognition was better. This did not happen with the 10x1o letters. There seems to be just enough grid entries so that the network wonk get caught in another patterns local minimum. On top of that it was able to clear the noise effectively.

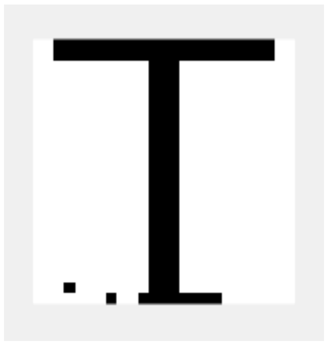
<u>Experiment 11: 25x25 letters with 35% noise on letter T</u>



The network has been trained with all seven letters, then the base image “T” was run through the network with 35% noise added:



Run though the network the result is:



Now it seems the noise is to much to overcome and the network cant get an accurate pattern.

d)

To look up all other experiments and results on professors drive:

1) Original pattern folders: “binary10x10letter”, “bianry25x25letter”, “bipolar10x10letter”, “bipolar25x25letter”, or .bmp in “origin 10x10 bitmap”, “origin 25x25 bitmap”. These folders contain the original patters in their respective labeled formats (both as arrays and and graphed figure files).
2) Noise pattern folders: “10x10 noise patterns”, “25x25 noise patterns”. These folders contain the noise patterns used for testing in there respective formats. Each subfolder is labeled by amount of noise and each file is labeled with the original letter the noise was derived from.
3) Results folders: “10x20 results”, “25x25 results”. These folders contain the arrays and graphs of the results. the subfolders are organized by amount of noise.

Example: To see the 10x10 letter H run on the network trained with the 7 letters with 30% noise on that letter you would do the following:

1) Original pattern in binary array and fig file formant:

binary10x10Letter/letter arrays/letterH.mat
binary10x10Letter/letter graphs/letterHgraph10.fig

2) Noise pattern derived from that letter:

10x10 noise patterns/ arrays /30%arrayNoise/ 30percentNoiseH.mat
10x10 noise patterns/ graphs/ 30percentNoise/ 30percentNoiseH.fig

3) Result:

10x10 results/ arrays/ 30%arrayResults/ 30noiseResultH.mat
10x10 results/ graphs/ 30percentResults/ 30noiseResultH.fig

Any questions fell free to email me.

All folders stored on Drive:

“binary10x10Letter” - contains the binary arrays (1/0) and matlab .fig graphs for the 7, 10x10 gridded letters.

“binary25x25letter” - contains the binary arrays (1/0) and matlab .fig graphs for the 7, 25x25 gridded letters.

“bipolar25x25letter” - contains the bipolar arrays (-1/1) and matlab .fig graphs for the 7, 25x25 gridded letters.

“bipolar10x10letter” - contains the bipolar arrays (-1/1) and matlab .fig graphs for the 7, 10x10 gridded letters.

“origin 10x10 bitmap” - Contains the 10x10 pixel gridded bitmap images created in the photo manipulation program “gimp” by me.

“origin 25x25 bitmap” - Contains the 25x25 pixel gridded bitmap images created in the photo manipulation program “gimp” by me.

“10x10 noise patterns” - Contains the arrays and graphs of the 7, 10x10 letters with respective amounts of noise added for testing.

“10x10 results” - Contains the results (arrays and graphs) of the respective 10x10 letters with the respective amounts of noise.

“25x25 noise patterns” - Contains the arrays and graphs of the 7, 25x25 letters with respective amounts of noise added for testing.

“25x25 results” - Contains the results (arrays and graphs) of the respective 25x25 letters with the respective amounts of noise.

“HopfieldNN” - Contains the Matlab code and GUI to run the Network

2) Implementation for Multilayer Perceptron (MLP) Feed Forward Fully Connected Neural Network with a Sigmoid activation function. Training is done using the Back-propagation algorithm with parameters for Resilient Gradient Descent, Momentum Back-propagation, and Learning Rate Decrease. The training stops when the Mean Square Error (MSE) reaches zero or a predefined maximum number of epochs is reached.

Design and Experiment plan for MLP:

Numbers of hidden layers and neurons per hidden layer- It's represented by the variable nbrOfNeuronsInEachHiddenLayer. For example: To have a neural network with 2 hidden layers with number of neurons 3, 11; that variable is set to [3 11].

Number of output layer nits- Under normal circumstances the number of output units is equal to the number of classes, but it still can be less (≤ log2(nbrOfClasses)). It's represented by the variable nbrOfOutUnits. The number of input layer units is obtained from the training samples dimension.

Sigmoid Activation Function- Sigmoid activation function is unipolar or polar. It's represented by the variable unipolarBipolarSelector.

Learning Rate- Represented as learning rate.

Maximum number of epochs- Where the training stops unless MSE reaches zero. Represented by the variable nbrOfEpochs_max.

Momentum Backpropagation rate- It's represented by the variable momentum_alpha.

Enable or disable Resilient Gradient Descent- It's represented by the variable enable_resilient_gradient_descent.

Learning Rate Decrease parameters- It's represented by the variables learningRate_decreaseValue and min_learningRate.

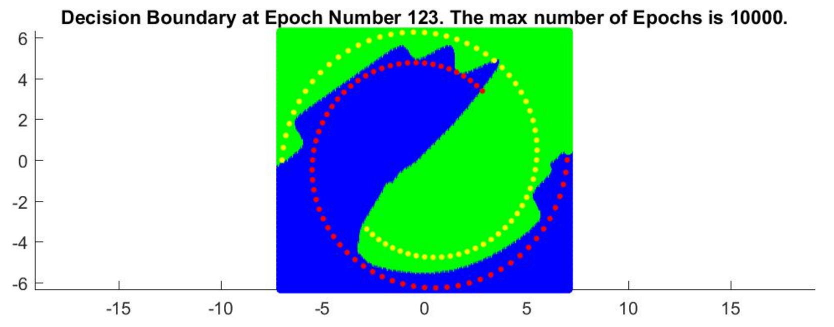
Draw Epochs- The number of epochs after which a figure is drawn and saved on the machine is specified. The figures are saved in a folder named Results besides the m files. This parameter is represented by the variable draw_each_nbrOfEpochs.

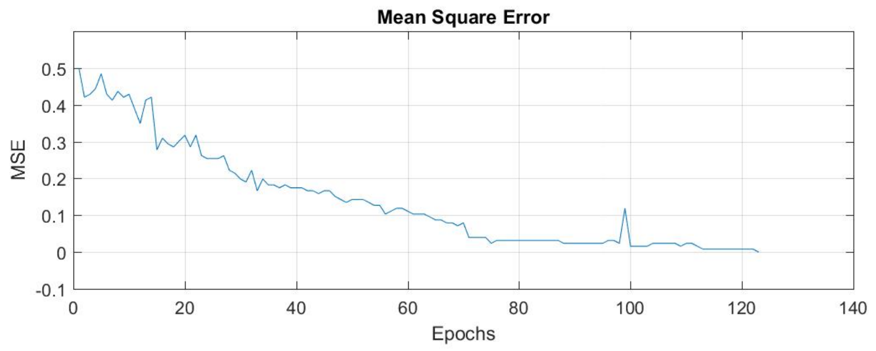
Example from Experiments:

Learning Rate: 0.15

Number of Hidden Layers: 2

Num of Neurons in each layer: [10,10]



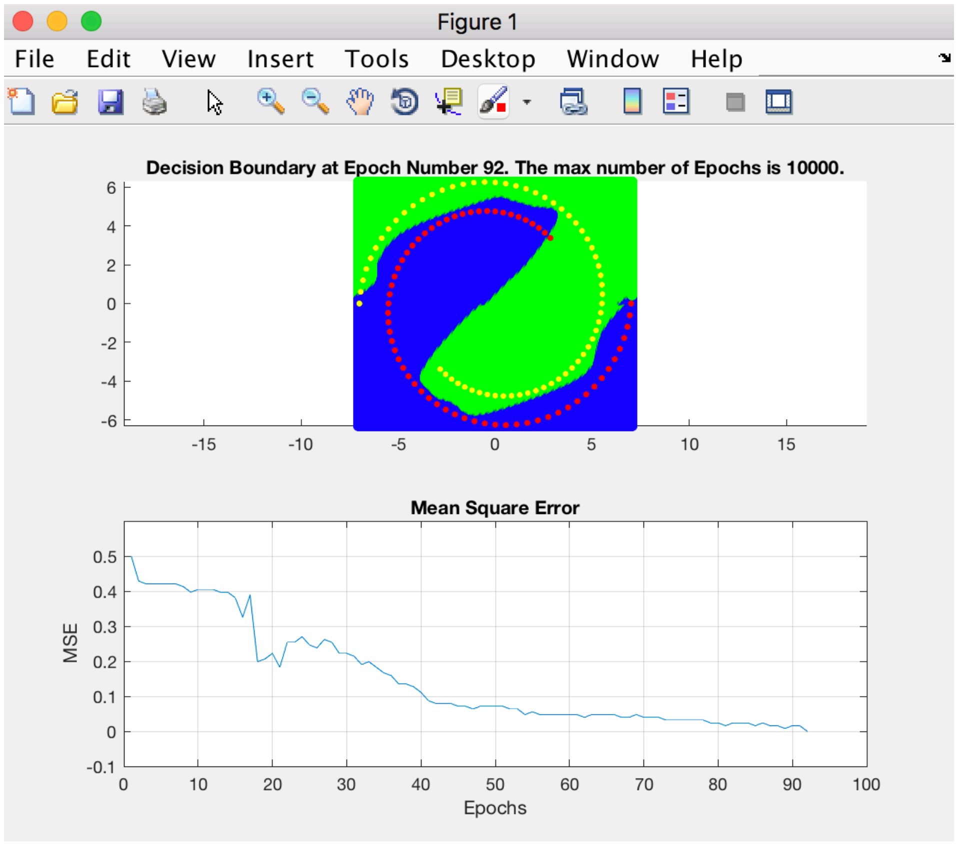


Rest of Experiment Parameters:

Learning Rate	Number of hidden neurons in layer 1	Number of hidden neuron in layer 2	Momentum	MSE=0 achieved at epoch	Stored in Folder
0.05	10	10	0	702	LearnignRate0.05_Momentum0
0.05	10	10	1	478	LearnignRate0.05_Momentum1
0.15	10	10	0	123	LearningRate0.15_Momentum0
0.15	10	10	1	145	LearningRate0.15_Momentum1
0.2	15	10	1	92	LearningRate0.2_Momentum1

Analysis:

As the number of neurons increased, the MSE would reach 0 within a smaller amount of epochs. The decision boundary would be more precise in identifying the regions of interest.



Its Easy to see by the last experiment. the boundary is much finer tuned.

All results are stored on Professors drive in their folders listed above. Any questions please contact.