Sofia Lima

15-686 Neural Computation

Dr. Tai-Sing Lee

17 March, 2023

Problem Set 3: Source separation and Associative Memories

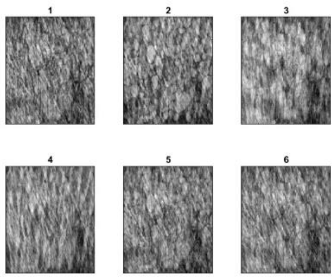
Part 1: Independent component analysis : blind source separation (5 pts)

In this assignment, we will use an optimized fast ICA package for you to understand what it can do.

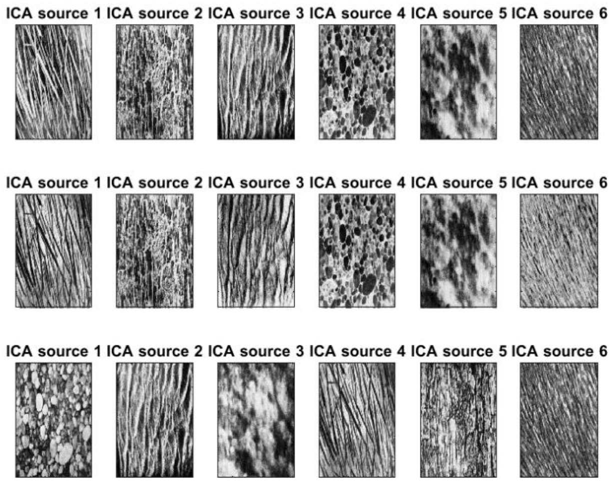
```
[icasig, A, W] = FASTICA (mixedsig); the rows of icasig contain the estimated
independent components, W contains estimated separating matrix and A contains the
corresponding mixing matrix
```

a) Discovering hidden images (1 pt)

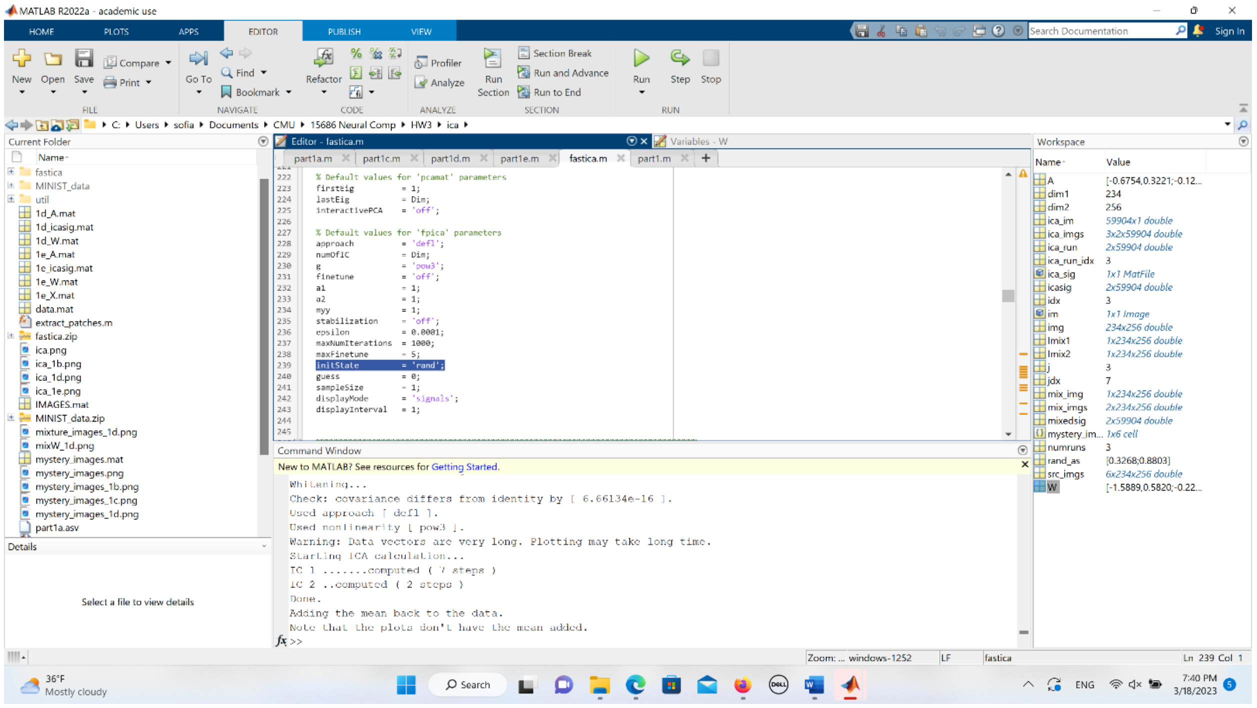
Shown below are the 6 mystery mixture images.



In the figure below, I am showing the 6 recovered source images from ICA from 3 runs in each row. Specifically, these are the ica components contained in the icasig value returned from fastica.



The results from ICA are not unique (the same every time). This is because the process begins with a random initialization for the mixing matrix A. Thus, each round varies in how model converges to the sources. Below are screenshots from the documentation.



Editor - fastica.m

```
% Default values for 'pcamat' parameters
firstEig          = 1;
lastEig           = Dim;
interactivePCA    = 'off';

% Default values for 'fpica' parameters
approach          = 'defl';
numOfIC           = Dim;
g                 = 'pow3';
finetune          = 'off';
a1                = 1;
a2                = 1;
myy               = 1;
stabilization     = 'off';
epsilon           = 0.0001;
maxNumIterations  = 1000;
maxFinetune       = 5;
initState         = 'rand';
guess             = 0;
sampleSize        = 1;
displayMode       = 'signals';
displayInterval   = 1;
```

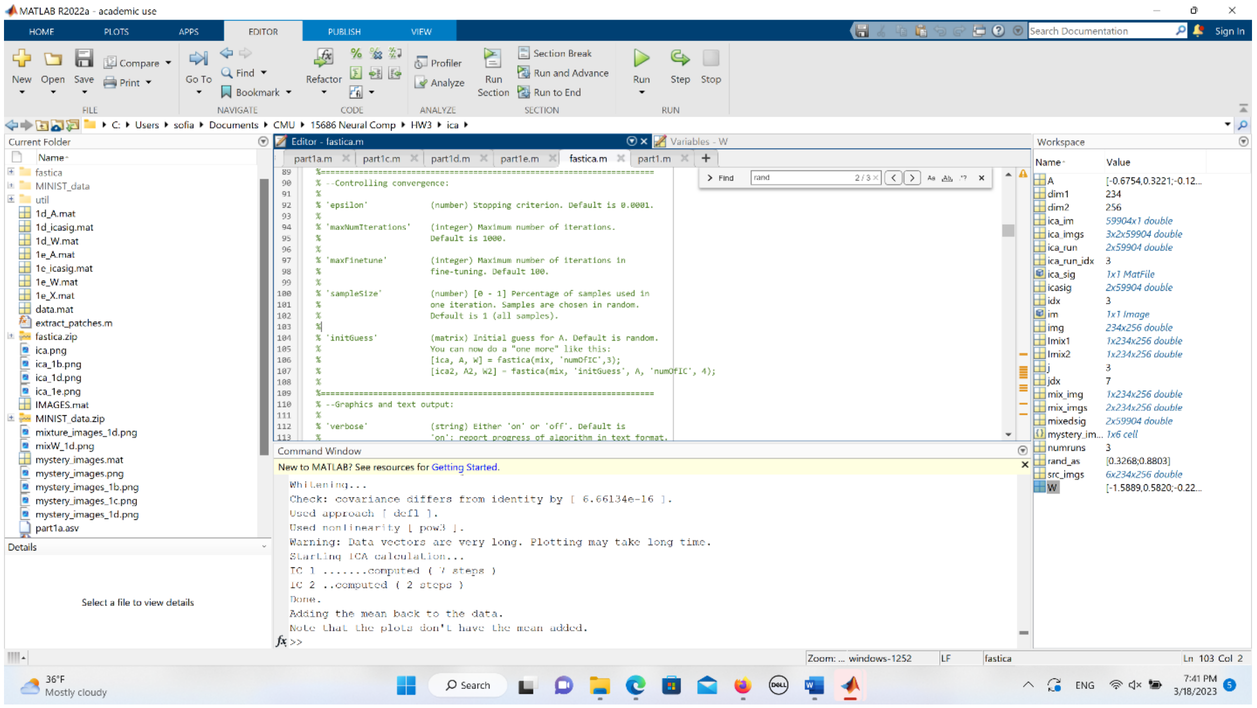
Command Window

New to MATLAB? See resources for Getting Started.

```
Whitening...
Check: covariance differs from identity by [ 6.66134e-16 ].
Used approach [ defl ].
Used nonlinearity [ pow3 ].
Warning: Data vectors are very long. Plotting may take long time.
Starting ICA calculation...
IC 1 .......computed ( 7 steps )
IC 2 ..computed ( 2 steps )
Done.
Adding the mean back to the data.
Note that the plots don't have the mean added.
>>
```

Workspace

Name	Value
A	[-0.6754,0.3221;-0.12...
dim1	234
dim2	256
ica_im	59904x1 double
ica_imgs	3x2x59904 double
ica_run	2x59904 double
ica_run_idx	3
ica_sig	1x1 MatFile
icasig	2x59904 double
idx	3
im	1x1 Image
img	234x256 double
Imix1	1x234x256 double
Imix2	1x234x256 double
j	3
jdx	7
mix_img	1x234x256 double
mix_imgs	2x234x256 double
mixedsig	2x59904 double
mystery_im...	1x6 cell
numruns	3
rand_as	[0.3268;0.8803]
src_imgs	6x234x256 double
W	[-1.5889,0.5820;-0.22...



Editor - fastica.m

```
%========================================================================
% --Controlling convergence:
%
% 'epsilon'             (number) Stopping criterion. Default is 0.0001.
%
% 'maxNumIterations'    (integer) Maximum number of iterations.
%                       Default is 1000.
%
% 'maxFinetune'         (integer) Maximum number of iterations in
%                       fine-tuning. Default 100.
%
% 'sampleSize'          (number) [0 - 1] Percentage of samples used in
%                       one iteration. Samples are chosen in random.
%                       Default is 1 (all samples).
%
% 'initGuess'           (matrix) Initial guess for A. Default is random.
%                       You can now do a "one more" like this:
%                       [ica, A, W] = fastica(mix, 'numOfIC',3);
%                       [ica2, A2, W2] = fastica(mix, 'initGuess', A, 'numOfIC', 4);
%
%========================================================================
% --Graphics and text output:
%
% 'verbose'             (string) Either 'on' or 'off'. Default is
%                       'on': report progress of algorithm in text format.
```

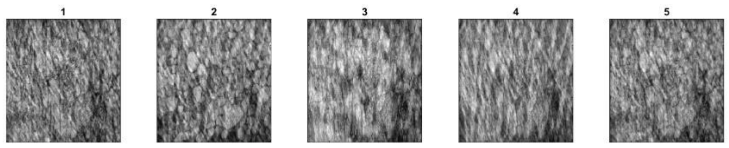
Command Window

New to MATLAB? See resources for Getting Started.

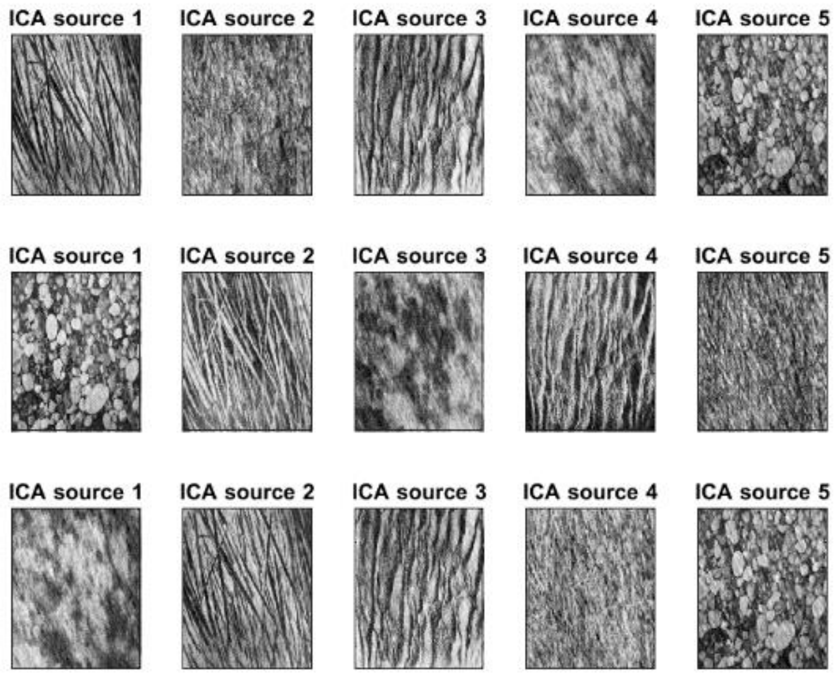
```
Whitening...
Check: covariance differs from identity by [ 6.66134e-16 ].
Used approach [ defl ].
Used nonlinearity [ pow3 ].
Warning: Data vectors are very long. Plotting may take long time.
Starting ICA calculation...
IC 1 .......computed ( 7 steps )
IC 2 ..computed ( 2 steps )
Done.
Adding the mean back to the data.
Note that the plots don't have the mean added.
>>
```

b) Limitations and Constraints: Number of Data Channels (0.5 pt)

Shown below are the first 5 of the mystery mixture images.

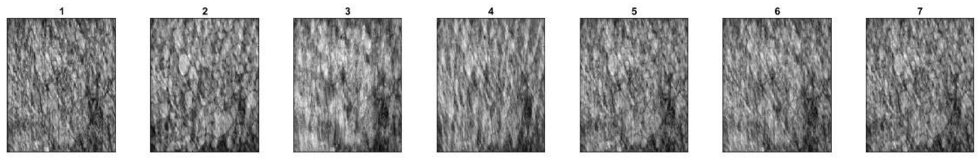


In the figure below, I am showing the 5 recovered source images from ICA from 3 runs in each row. There are 5 sources recovered because the covariance matrix only has rank 5 with only 5 images.

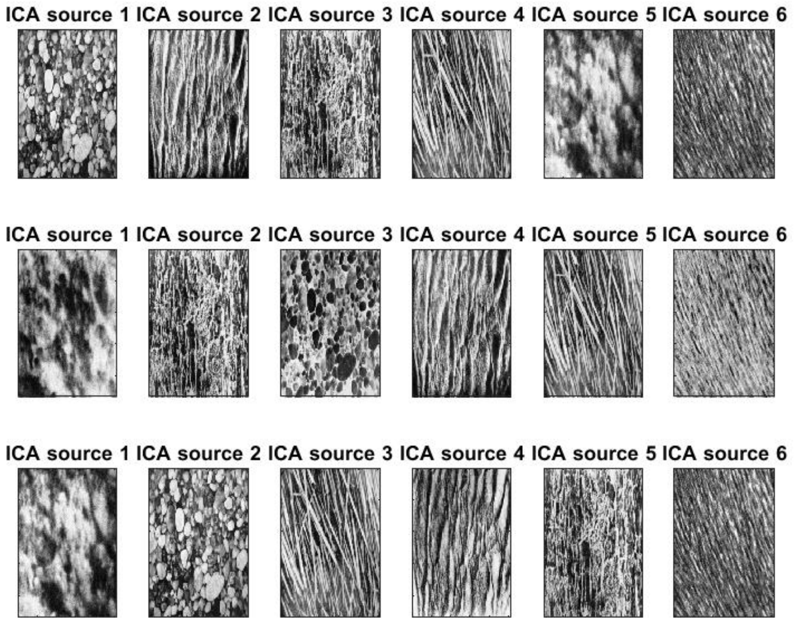


c) Limitations and Constraints: Number of Data Channels (0.5 pt)

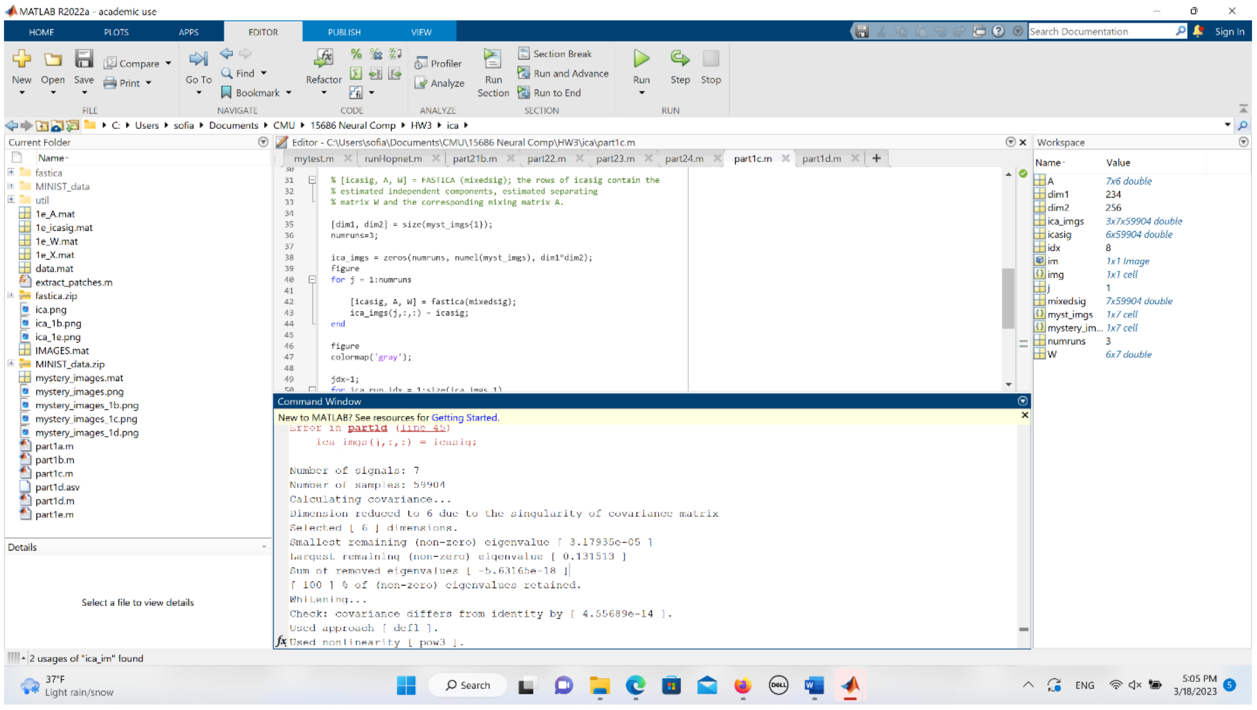
Shown below are the 6 mystery mixture images, plus a 7th mixture image generated by combining mixtures 1 and 2.



Shown below are the 6 recovered ICA components from the 7 mixtures.


ICA source 1 ICA source 2 ICA source 3 ICA source 4 ICA source 5 ICA source 6
ICA source 1 ICA source 2 ICA source 3 ICA source 4 ICA source 5 ICA source 6
ICA source 1 ICA source 2 ICA source 3 ICA source 4 ICA source 5 ICA source 6

Only 6 ICA components are recovered because the dimensions are reduced to avoid singularity in the covariance matrix. In the figure below, I am showing a screenshot of how fastica explains why only 6 ICA components are computed.


MATLAB R2022a - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Editor - C:\Users\sofia\Documents\CMU\15686 Neural Comp\HW3\ica\part1c.m
mytest.m runHopnet.m part21b.m part22.m part23.m part24.m part1c.m part1d.m
Command Window
New to MATLAB? See resources for Getting Started.

```
% [icasig, A, W] = FASTICA (mixedsig); the rows of icasig contain the
% estimated independent components, estimated separating
% matrix W and the corresponding mixing matrix A.

[dim1, dim2] = size(myst_imgs{1});
numruns=3;

ica_imgs = zeros(numruns, numel(myst_imgs), dim1*dim2);
figure
for j = 1:numruns

    [icasig, A, W] = fastica(mixedsig);
    ica_imgs(j,:,:) = icasig;
end

figure
colormap('gray');

jdx=1;
for ica_run_idx = 1:size(ica_imgs,1)
```

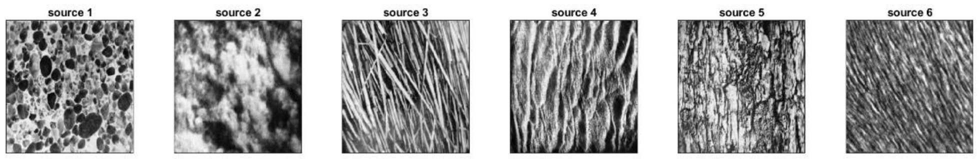
```
Error in part1d (line 45)
    ica_imgs(j,:,:) = icasig;

Number of signals: 7
Number of samples: 59904
Calculating covariance...
Dimension reduced to 6 due to the singularity of covariance matrix
Selected [ 6 ] dimensions.
Smallest remaining (non-zero) eigenvalue [ 3.17935e-05 ]
Largest remaining (non-zero) eigenvalue [ 0.131513 ]
Sum of removed eigenvalues [ -5.63165e-18 ]
[ 100 ] % of (non-zero) eigenvalues retained.
Whitening...
Check: covariance differs from identity by [ 4.55689e-14 ].
Used approach [ defl ].
Used nonlinearity [ pow3 ].
```

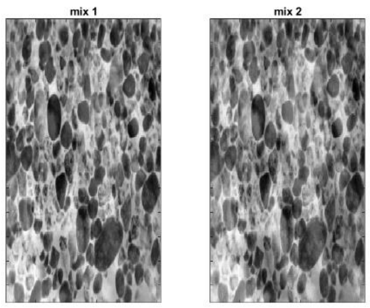
Name	Value
A	7x6 double
dim1	234
dim2	256
ica_imgs	3x7x59904 double
icasig	6x59904 double
idx	8
im	1x1 Image
img	1x1 cell
j	1
mixedsig	7x59904 double
myst_imgs	1x7 cell
mystery_im...	1x7 cell
numruns	3
W	6x7 double

d) Your own exploration (1 pt)

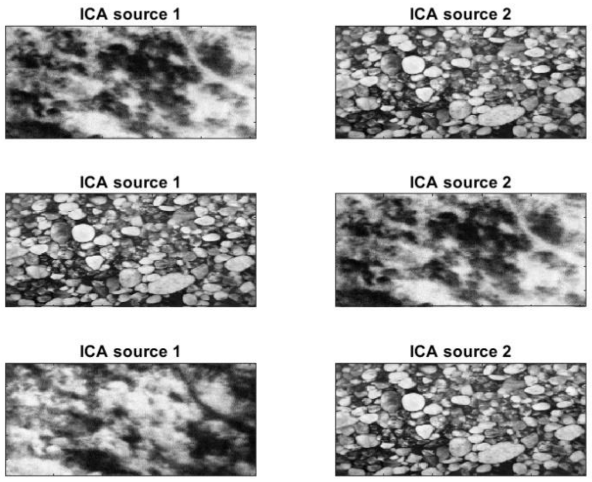
Shown below are the source images generated from ICA in part a.



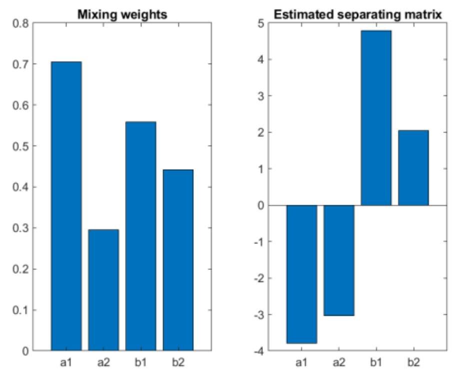
Using these sources, I create 2 mixture images from the first 2 source images. I use coefficients that sum to 1 for each mixture of the two source images. Shown below are the mixtures created.



Shown below are the recovered source images from 3 runs of ICA. Slight differences exist from the source images; mainly, the color is inverted for some cases. This was fairly successful for the model because I only used 2 source images and created 2 mixtures.



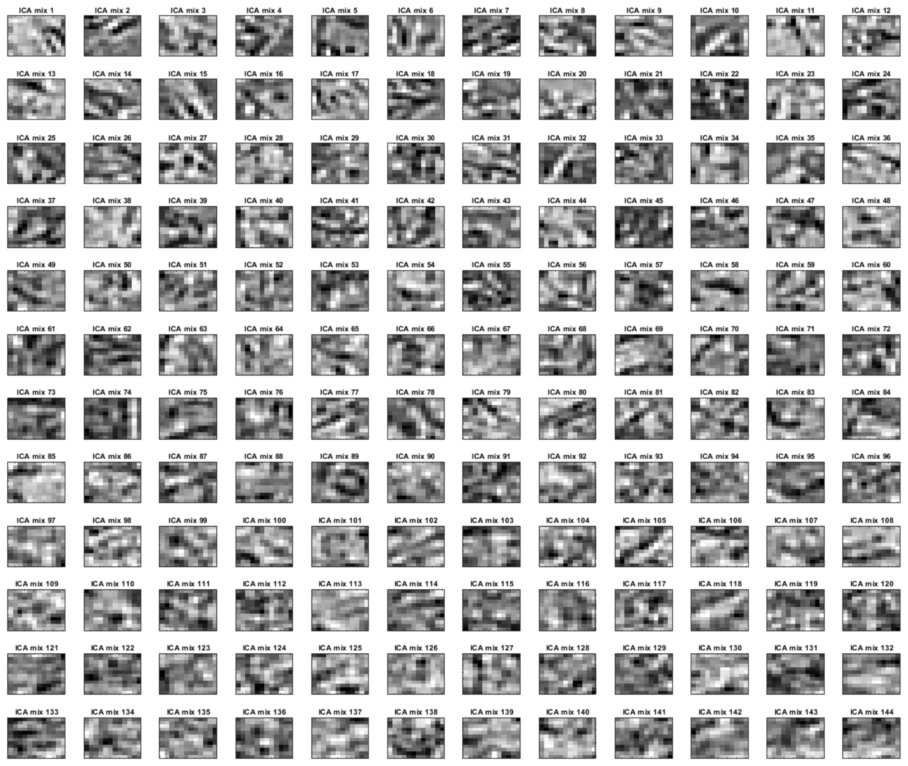
To answer how well can you recover the mixing weights a1,a2,b1,b2, I compare the mixture weights with the estimated separating weights. In the figure below, the left panel shows the values of the mixing weights used to generate the mixtures, and the right side shows the values of the W matrix returned from fastica. The fact that some values in W are negative suggests that the model is not learning the mixing weights.



Mixtures that are difficult to separate would be those with non-linear mixing. For example, if the images values are multiplied together or squared rather than combined linearly with a weighted sum.

e) Learning sparse codes of natural image using ICA (2 pt)

In the figure below, I am showing all 144 basis vectors in A, resembling the V1 simple cell receptive fields. Specifically, these are the columns of the mixing matrix A reshaped as 12 by 12 patches. These image components are sparse code of the natural images.



f) Creative Exploration with ICA (1 pt)

See part 3. Confirmed in piazza post @139 that this is a duplicate question.

Part 2: Learning Hopfield Nets (6 pts)

1. Hopfield Dynamics (2 pt)
   a. Why is the update rule doing gradient descent? (1 pt)

First let's say that before we update a unit the energy of the current configuration is E^{old}. Let us denote the unit that will be update by v_r which will have value v_r^{old}. Thus:

$$E^{old} = -\sum_{i\neq j} v_i v_j w_{i,j} = -\sum_{i\neq j\neq r} v_i v_j w_{i,j} - \sum_{i\neq r} v_i v_r^{old} w_{i,r} - \sum_{j\neq r} v_r^{old} v_j w_{r,j}$$

$$= -\sum_{i\neq j\neq r} v_i v_j w_{i,j} - 2\sum_{i\neq r} v_i v_r^{old} w_{i,r}$$

The last step is correct because the weights are symmetric. When we update v_r by the rule given by the algorithm the new v_r, v_r^{new}, will be:

$$v_r^{new} = \text{sign}\left(\sum_{i\neq r} v_i w_{i,r}\right)$$

The energy of the new configuration will then be:

$$E^{new} = -\sum_{i\neq j} v_i v_j w_{i,j} = -\sum_{i\neq j\neq r} v_i v_j w_{i,j} - \sum_{i\neq r} v_i v_r^{new} w_{i,r} - \sum_{j\neq r} v_r^{new} v_j w_{r,j}$$

$$= -\sum_{i\neq j\neq r} v_i v_j w_{i,j} - 2\sum_{i\neq r} v_i v_r^{new} w_{i,r}$$

Thus the change in energy due to the update will be:

$$\Delta E = E^{new} - E^{old} = 2\left(-v_r^{new} + v_r^{old}\right)\sum_{i\neq r} v_i w_{i,r}$$

Now we must show that our update of v_r, following the Hopfield update rule, will always make the following hold: $\Delta E \leq 0$. There are two cases:

The first case is when $v_r^{new} = v_r^{old}$. In this case the configuration does not change and thus the energy does not change. The second case is when $v_r^{new} = (-1)\, v_r^{old} = \text{sign}(\sum_{i\neq r} v_i w_{i,r})$. In this case we have the following change in energy:

$$\Delta E = 2\left(-\text{sign}\left(\sum_{i\neq r} v_i w_{i,r}\right) + (-1)\,\text{sign}\left(\sum_{i\neq r} v_i w_{i,r}\right)\right)\sum_{i\neq r} v_i w_{i,r}$$

Complete this proof by showing that with this second case the sought after property still holds, i.e. ΔE ≤ 0. We understand that not everyone in this course is familiar with proofs. Therefore it is ok to explain in words why the property will hold true.

The change in energy is zero because the only way for the statement above for ΔE to be greater than zero is if the sign of the weighted sum was not always the same, but it must be the same sign because it is the same value.

$$\Delta E = 2\left(-2\,\underline{\text{sign}\left(\sum_{i\neq r} V_i W_{i,r}\right)}\right)\underline{\sum V_i w_{i,r}}$$

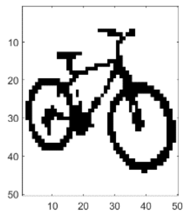
This value will never be positive because even if the weighted sum is negative, then it is multiplied again which would cancel out.

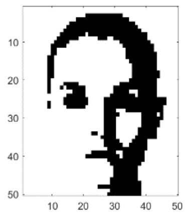
After completing the proof answer the following question: If we ran this algorithm so that it went through the while loop an infinite number of times, will the energy of the resulting configuration have lower energy than any other possible configurations? That is, does the network always settle down to the global minimum?

No, gradient descent does not always find the globally minimum energy. Based on the proof, the system can never exit the local minimum because it is always decreasing (or remain the same).

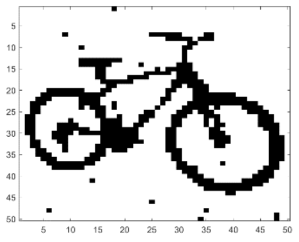
b. Implementing Hopfield dynamics (update) (1 pt)

Shown below are the loaded images to be remembered; note that the function load_image_by_name binarizes the image into black and white pixels.

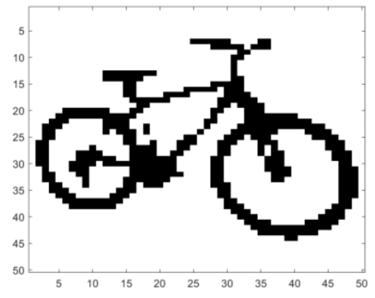




The visual source stimulus which has 20 pixels corrupted is shown below.



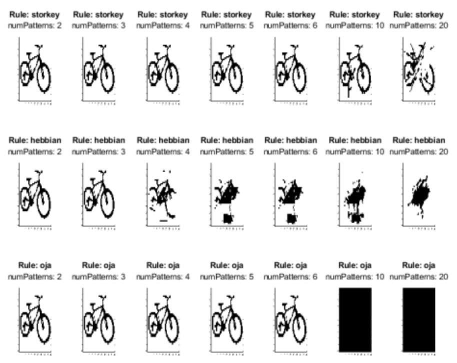
The final image retrieved is shown below. Despite corruption of 20 pixels, the pattern is learned because the input is similar to the output with such small noise. Additionally, the model parameters require only 2 patterns to remember, a relatively easier task.

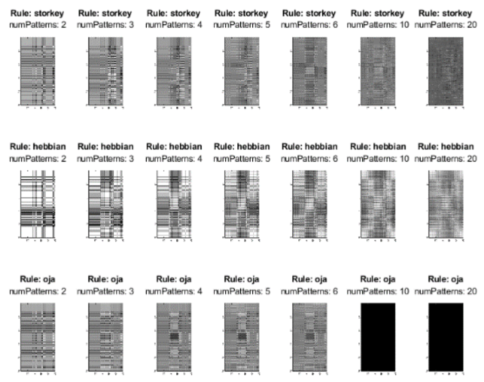


2. Implementing Outer-Product rule and Oja's Learning Rule (2 pt)

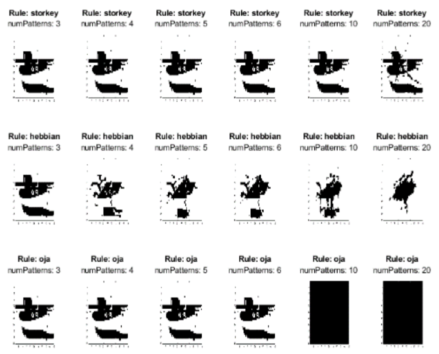
To answer how many patterns can each type of network learn, I systematically compared 2 images for testing (one with a white background and one with a non-uniform background) across different learning rules.

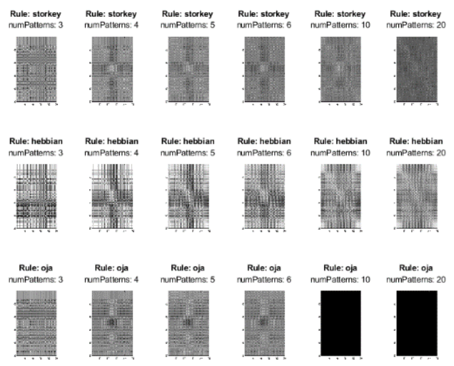
Shown on the left are the bike patterns "remembered" by the network across different types of learning rules in each row. Each column represents how many patterns the network was asked to remember. On the right, each weight matrix with the corresponding learning parameters is shown.





The same comparison is made with the boat where the background may complicate the pattern and affect the network's ability to remember. Included are the weights from these experiments.

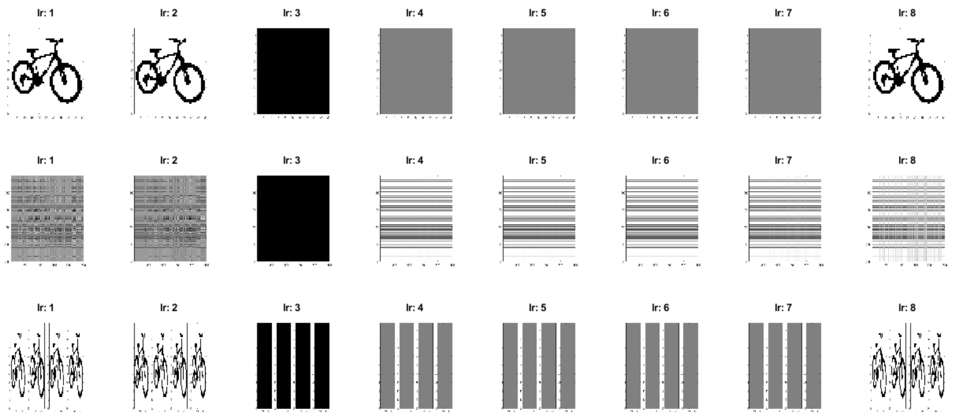




3. Effects of Learning Rates (1 pt)

To answer how memory encoding and retrieval is affected by learning rates in Oja's rule, I systematically compared testing the bike image retrieval across 8 learning rates. Shown below are the final recovered images in the top row, the weight matrix in the middle row, and 4 intermediate checkpoints in the top row. Each column is a different learning rate as follows:

- first column: learning rate = 0.7*1/numNeurons which was the provided default
- second column: 0.07*1/numNeurons which is 10 times smaller
- third column: 7*1/numNeurons which is 10 times larger
- fourth column: 0.7*1/numNeurons^2 where the denominator is squared
- fifth column: 0.7^2*1/numNeurons^2 where both the denominator and numerator are squared
- sixth column: 0.7*1/numNeurons^3 where the numerator is cubed
- seventh column: 0.0007*1/numNeurons which is 1000 times smaller than the first column
- eighth column: 0.007*1/numNeurons which is 100 times smaller than the first column

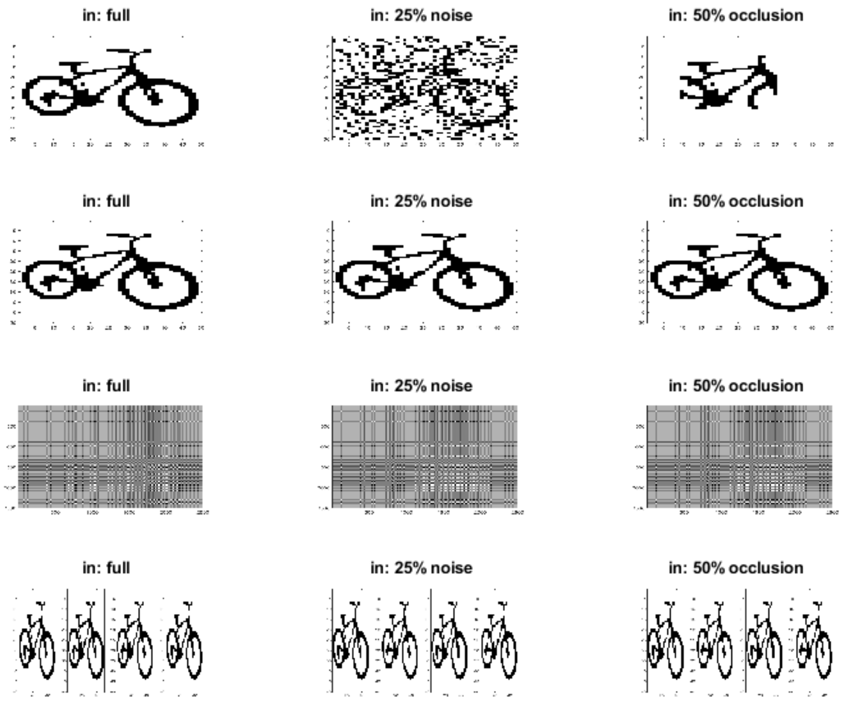


I observe an interesting observation that the learning rate can be neither too high nor too low. For example, the learning rate in case #3 is 10 times larger than case #1, suggesting that the learning rate can not be too high because the recovery was not successful. In cases #4-7, the learning rate is smaller than case #1, and the weight matrices T contain specific patterns across the columns. The interesting comparison is between cases #7 and #8, where case #8 is only 100 times smaller than case #1, but case #7 is 1000 times smaller than case #1. This demonstrates the reliability of the system on a balanced learning rate.

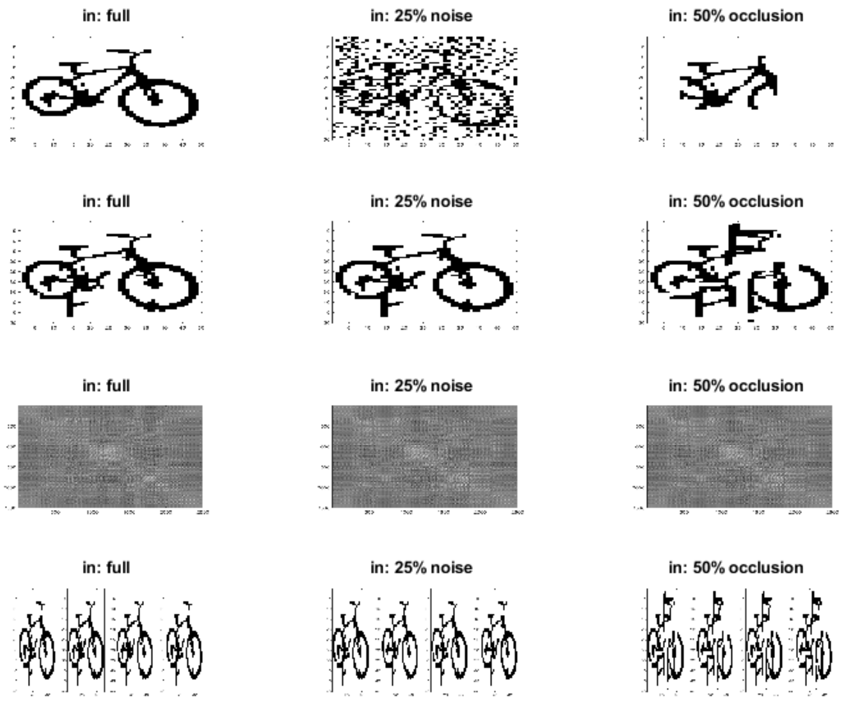
4. Retrieval Performance under Noise and Occlusion (1 pt)

To evaluate the capacity of the network when the input image is corrupted, I systematically compared testing the bike image retrieval with 25% noise and 50% occlusion. I use Storkey's learning rule given the results from part 2 where the network was able to remember patterns well across the most values of the numPatterns parameter. I use the default learning rate 0.7*1/numNeurons as set in the starter code.

Below are the results from 2 numPatterns to remember. The recovery in all conditions were fairly successful. This is not surprising given that the model was able to form a strong association between input and output even for many numPatterns.



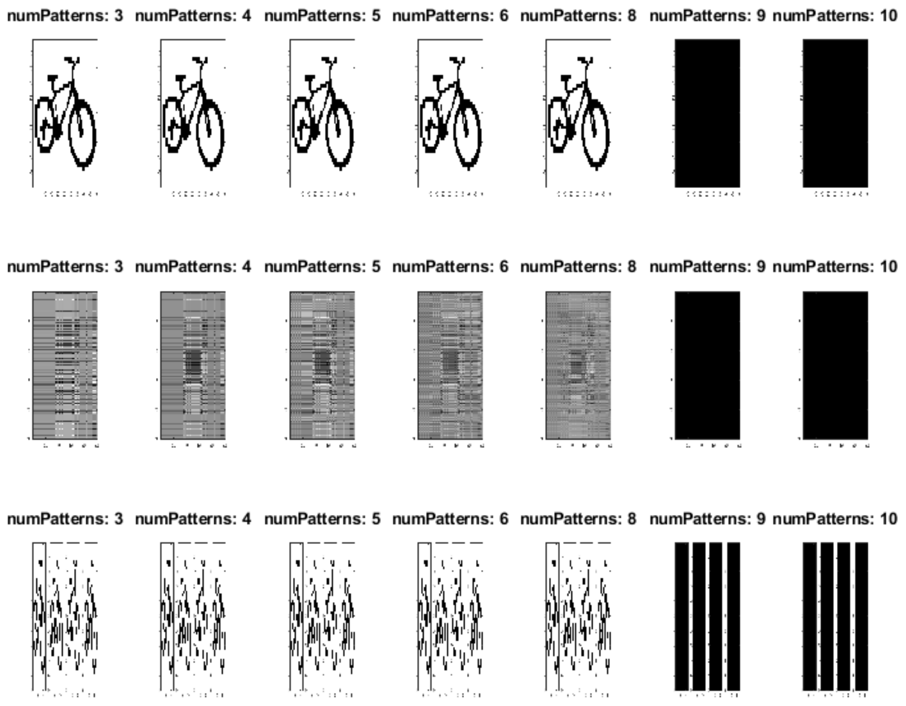
Below are the results from 10 numPatterns, which the model had some difficulty with in part 2. This setting showed that the model does struggle to retrieve correct patterns with occlusion in the input such that the center of the image was difficult to recover based on the cropped input. The results from corrupting the image with 25% noise does not hinder the recovery more than full mode.



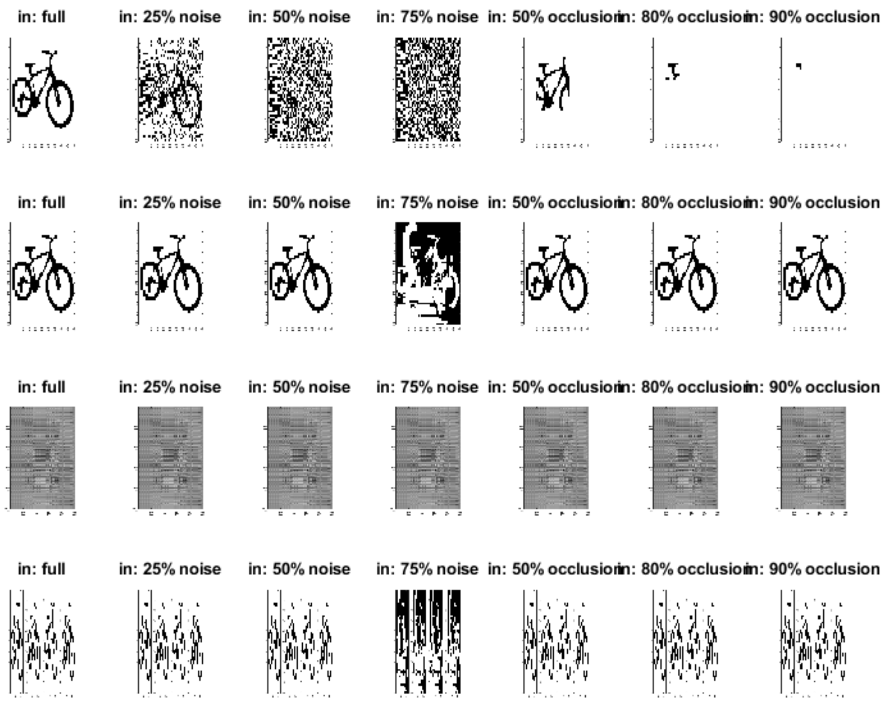
Part 3: Creative Exploration or Competition (1 pt)

For this part, you can explore either ICA or Hopfield net to answer a question or two that you pose. Interesting and outstanding answers will be rewarded up to 2 bonus points. Limit this part of the report to 4 pages maximum. One legitimate exploration is to develop strategies to maximize the number of patterns that can be encoded in the network. The winner of this competition (maximum number of patterns remembered) will surely be awarded the 2 bonus points.

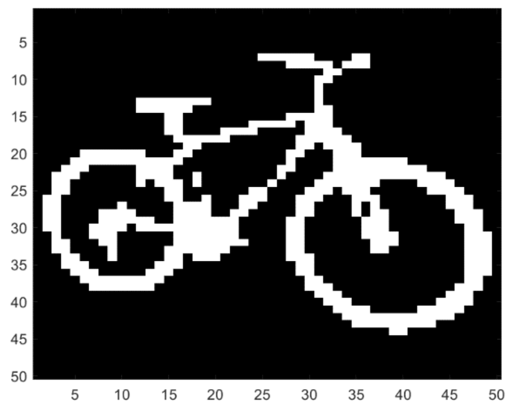
In this problem I am exploring a way to increase the number of patterns that can be learned using Oja's update rule—increasing the number of epochs. In the figures below, I am plotting the results of training a network with oja's rule with 500 epochs, 5 times the number of epochs provided in the starter code.



There seems to be no improvement to the networks ability to remember more than 8 patterns. This lead me to explore what could be recovered by increasing the number of epochs. I systematically compared various image corruption modifications to the test image to find what level of corruption would disrupt the recovery performance of the network with 8 numPatterns to remember. Shown below are the results of these runs with 100 epochs.



From this figure, you can see that ability of the network to retrieve the bike with 75% noise is disrupted. I hypothesized that by increasing the number of epochs from 100 to 500, the retrieval would be recovered. Shown below is the retrieved bike from this experiment, supporting my hypothesis. Albeit the colors are inverted from the corruption, the pattern is intact.


5
10
15
20
25
30
35
40
45
50