580.422 System Bioengineering II: Neurosciences (Spring 2015)

Homework #9: Perceptrons, Optimal Linear Weighting, and Hopfield Networks

Dr. Kechen Zhang

The following files are needed for the exercises:

X.mat
HOP.mat
randsign.m
randzero.m

1. Computer experiment with a perceptron

The perceptron is basically a linear model of a single neuron that receives multiple inputs. For MATLAB implementation, the perceptron equation can be written in a vector form:

```
y = w'*x
```

where the input `x` and weight `w` are taken as column vectors, and the prime indicates transpose**.** The threshold parameter is taken as zero and ignored. Before training, set

```
w = 0
```

After input `x` is presented, the increment of the weight vector should be

```
delta_w = eta*(Y-y)*x
```

where `Y` is the desired output, and the learning rate `eta` is a small number. In this homework we set

```
eta = 1e-4
```

After training, the weight vector becomes:

```
w = w + delta_w
```

The input data are 10 black-and-white National Geographic photos of natural scenes (228-by-320 pixels). Load the data file `X.mat` data into Matlab using:

```
load X
```

Matrix `X` has 10 columns, each column is an input vector (a photo), and each entry in the vector is a real number that represents the gray level of a pixel. To view the first column vector

```
x=X(:,1);
```

as an image, for instance, use

```
m=reshape(x, 228, 320);
imagesc(m)
colormap(gray)
```

Questions:

(a) Teach a perceptron to tell whether there is an animal in a scene: Photos 1, 3, 4, and 9 contain an animal while the rest don't. Present each photo only once, from photo 1 to photo 10 in sequential order, and update the weights according to the learning rule. The desired output `Y=1` when an animal is present and `Y=-1` otherwise. After training, test the photos one by one (without changing the weight) and see how the perceptron responds. List the actual outputs for these photos.

(b) The response is considered correct if the output is positive when there is an animal or negative otherwise. After the first training session, the responses to which photos are correct?

(c) Repeat the training session a second time. Are the responses to all the photos correct now?

(d) Repeat the training session for a total of 100 times (including the two in (a) and (c)). It may take a while to finish. Does the actual output appear to approach the desired output (1 or –1)? List the actual outputs after the training.

(e) You may suspect that a perceptron cannot really tell animals from non-animals. Let's define the two classes of photos arbitrarily. Call photos 4, 6, 8, 10 positive ones and let the desired output `Y=1`. Call the rest of the photos negative ones and let `Y=-1`. Start from scratch again by setting the weights to zero. Repeat the same experiment in (a) and train the perceptron with each photo once. Using the same criterion as in (b), how accurately can the perceptron distinguish the two classes of photos after the first training session?

Photos stored in matrix `X`:

Photo 1

Photo 2

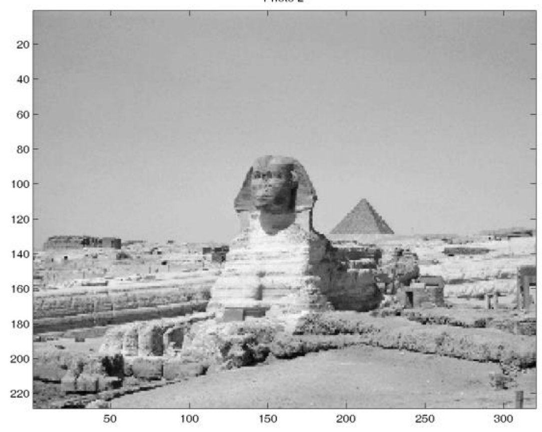


Photo 3

Photo 4

Photo 5

Photo 6

Photo 7

Photo 8

Photo 9

Photo 10

2. Optimal linear mapping

The data matrix $\mathbf{X}$ in the last exercise has 10 columns and 72960 rows, and each column corresponds to a data image with $228 \times 320 = 72960$ pixels. We want to find a linear network with weight matrix $\mathbf{W}$ so that each of the 10 images is mapped to itself. In other words, we want

$$\mathbf{X} = \mathbf{W}\mathbf{X}\,. \tag{1}$$

An exact solution to this problem is

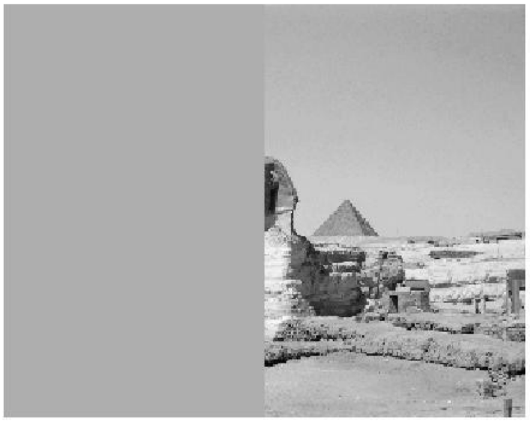
$$\mathbf{W} = \mathbf{X}\mathbf{X}^{\dagger}, \tag{2}$$

where the Moore-Penrose pseudoinverse is

$$\mathbf{X}^{\dagger} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\,. \tag{3}$$

Questions:

a. Verify that $\mathbf{X}^{\dagger}\mathbf{X} = \mathbf{I}$ and that solution (2) does satisfy equation (1).
b. Given that the data images are linearly independent, what is the rank of the data matrix $\mathbf{X}$?
c. What is the size of the weight matrix $\mathbf{W}$?
d. Numerical verification of equation (1): the pseudoinverse can be obtained by the MATLAB command `P=pinv(X)` or by equation (3): `P=inv(X'*X)*X'`. The weight matrix in equation (2) is too large to be stored in MATLAB. To compute $\mathbf{Y} = \mathbf{WX} = \mathbf{XX}^{\dagger}\mathbf{X}$, we use `Y=X*(P*X)`. Now visualize the output `Y` as in the last exercise and report whether the images look correct. What is the relative error as quantified by the ratio: `sum(abs(X(:)-Y(:)))/sum(abs(X(:)))`?
e. Let's clip the images by half by doing `C=X` followed by `C(1:36480,:)=0`. For example, the second image now looks like



Now use `Y=X*(P*C)` and visualize the output `Y`. How do the output images look like? What is the value of the relative error as defined in the last question?

3. Computer experiment with a Hopfield network

The purpose of this exercise is to gain direct experience with the behaviors of a recurrent network. The key MATLAB codes are described below, so the computer work should be simple and straightforward. Two MATLAB m-files `randzero.m` and `randsign.m` and one data file `HOP.mat` are needed for this exercise. Use `load HOP` to load `HOP.mat` into MATLAB. It contains the following vectors

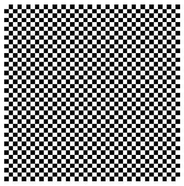
`A, B, C, D, Test1, Test2, Test3,`

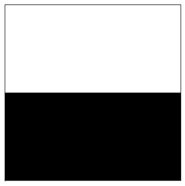
We'll build a Hopfield network with 1600 binary neurons that are fully connected with one another. The state of the network or the pattern of activities of these neurons will be described by a column vector `S`, which has 1600 entries, each for the activity of one neuron. For easy visualization, we can always plot this pattern of activities as a 40-by-40 image using

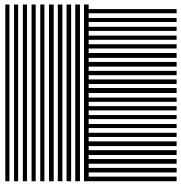
```
        imagesc(reshape(S,40,40))
        colormap(gray)
```

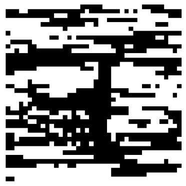
as in the perceptron exercise.

We want to store the following patterns of activities in the Hopfield network.


pattern A


pattern B


pattern C


pattern D

We'll start by storing the first three patterns `A`, `B` and `C`. The synaptic weight matrix should be

```
W = A*A′ + B*B′ + C*C′;
```
(1)

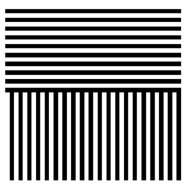
Here `A`, `B` and `C` are the columns vectors that you have loaded from HOP.mat. Each entry of these vectors are either 1 or –1 with equal chance. Convince yourself that Eq. (1) is equivalent to the weight rule given in the lecture note.

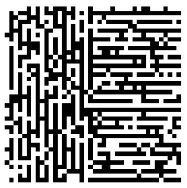
The command `S=A` sets the intial state of the network as the memory pattern `A`. To update the state of the network, use:

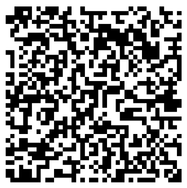
```
S = sign(W*S)
```
(2)

If `S` changes, do this again until `S` becomes stationary. Start also from `B` and `C` and do the iteration. This network converges to a stationary state very fast, often after only one iteration.

Questions:
1. *Stored memories:* Start from each of the stored patterns `A`, `B` or `C`. Are all these states stationary states of the network?
2. *Noise in initial state:* Set the initial state `S=Test1`, and do the iteration in Eq. (2). What is the final stationary state? Answer the same question using `Test2` and `Test3` as the initial state.


pattern `Test1`


pattern `Test2`


pattern `Test3`

For your information, `Test2` was derived from pattern `C` by flipping the signs of 15% of pixels that are randomly selected. `Test3` was derived similarly but with 40% flipping.

(If you want do to more tests, the included function `randflip.m` is handy. For example, `s=randsign(A, 0.2)` flips 20% of randomly selected entries in `A`.)

3. *Unrelated pattern:* Now set the initial state as pattern `D` (a binary version of the Loch Ness monster photo) and do the iteration. What is the final stationary state?

4. *Learning new memory pattern:* Add the storage of pattern D using

```
W = W + D*D′
```
(3)

This formula is entirely consistent with the weight rule in Eq. (1). Now do the same as in question 3 and what's the result?

5. *Are old memories affected?* Start from Test3 and iterate. Do you get the same final state as before (question 2)?

6. *Spurious memories:* Use S=`sign(randn(1600,1))` to set a random initial state where each neuron is either on or off (1 or –1) with equal probability. Try this several times. Do you get any final stationary state that is different from the stored memory patterns?

7. *Symmetric connections:* To see how the neurons in the network are connected, use `imagesc(W)`, or `imagesc(W(1:100,1:100))` for easier view. (`colormap(jet)` and `colorbar` add color and scale.) Notice the symmetry about the diagonal, which means `W′=W` or $w_{ij} = w_{ji}$. That is, every pair of neurons must be reciprocally connected with equal strength. Does the weight rule in Eqs. (1) and (3) imply that this symmetry must always be true?

8. *Damage to synaptic connections:* Cut some of the connections in the network and see how that affects the memory storage. Use `W=randzero(W, 0.7)`. The effect is to randomly select 70% of the entries of matrix `W` and then set them all to 0. That is, the majority of the connections in the network are destroyed. Use `Test3` as the initial state. Do you get the same final result as in question 2?