Winter 2019 PHYS 178/278, Homework 3
Due 11:59 PM on February 24th

1 Two Oscillators with Exponential-decay Coupling

Consider the two weekly coupled phase oscillations,

$$\frac{d\psi_i}{dt} = \omega + \vec{Z}\left(\psi_i\right) \cdot \vec{P}\left(\psi_i, \psi_j\right), \; i, j = \{1, 2\}$$

the two perturb each other with a small phase shift, $\delta\psi_i = \psi_i - \omega t$,

$$\frac{d\delta\psi_i}{dt} = \vec{Z}\left(\delta\psi_i + \omega t\right) \cdot \vec{P}\left(\delta\psi_i + \omega t, \delta\psi_j + \omega t\right).$$

Since $\frac{d\delta\psi_{i,j}}{dt} \ll \omega$, we can average the perturbation over a full cycle,

$$\frac{d\delta\psi_1}{dt} = \Gamma\left(\delta\psi_1, \delta\psi_2\right) \tag{1}$$

$$\frac{d\delta\psi_2}{dt} = \Gamma\left(\delta\psi_2, \delta\psi_1\right) \tag{2}$$

where $\Gamma\left(\delta\psi_i, \delta\psi_j\right)$ represents the averaged perturbation of oscillator j on oscillator i over a full cycle,

$$\Gamma\left(\delta\psi_i, \delta\psi_j\right) = \frac{\epsilon}{2\pi}\int_{-\pi}^{\pi} d\theta \vec{Z}\left(\delta\psi_i + \theta\right) \cdot \vec{P}\left(\delta\psi_i + \theta, \delta\psi_j + \theta\right), \tag{3}$$

note that we have replaced $\omega t = \theta \in \{-\pi, \pi\}$. Similar to what we did in the lecture, assume that the perturbation $\vec{P}\left(\cdots\right)$ solely depends on the phase of the other oscillator, i.e., $\vec{P}\left(\delta\psi_i + \theta, \delta\psi_j + \theta\right) \to \vec{P}\left(\delta\psi_j + \theta\right)$,

$$\begin{aligned}
\Gamma\left(\delta\psi_i, \delta\psi_j\right) &= \frac{\epsilon}{2\pi}\int_{-\pi}^{\pi} d\theta\, \vec{Z}\left(\delta\psi_i + \theta\right) \cdot \vec{P}\left(\delta\psi_j + \theta\right), \text{ change of variable } \theta \to \theta - \delta\psi_j \\
&= \frac{\epsilon}{2\pi}\int_{-\pi}^{\pi} d\theta\, \vec{Z}\left(\theta + \left(\delta\psi_i - \delta\psi_j\right)\right) \cdot \vec{P}\left(\theta\right), \quad \text{let } \Delta_{ij} = \delta\psi_i - \delta\psi_j \\
\Rightarrow \Gamma\left(\delta\psi_i, \delta\psi_j\right) = \Gamma\left(\Delta_{ij}\right) &= \frac{\epsilon}{2\pi}\int_{-\pi}^{\pi} d\theta\, \vec{Z}\left(\theta + \Delta_{ij}\right) \cdot \vec{P}\left(\theta\right),
\end{aligned} \tag{4}$$

where the sensitivity of phase to the perturbation is

$$Z\left(\phi\right) = \sin\left(\phi\right), \tag{5}$$

and the perturbation $\vec{P}\left(\cdots\right)$ is given by an exponential function

$$\vec{P}\left(\phi\right) = \begin{cases} 0 & , \phi < 0 \\ \frac{g_{syn}}{c_m} e^{-\phi/\omega\tau} & , \phi \geq 0. \end{cases} \tag{6}$$

Subtracting the two equations, Eq. (1) and (2), we get the equation of motion for the phase difference between the two oscillators,

$$\begin{aligned}
\frac{d\Delta_{12}}{dt} &= \Gamma\left(\Delta_{12}\right) - \Gamma\left(\Delta_{21}\right), \\
&= \Gamma\left(\Delta_{12}\right) - \Gamma\left(-\Delta_{12}\right), \\
&= 2\Gamma_{odd}\left(\Delta_{12}\right),
\end{aligned} \tag{7}$$

where $\Gamma_{odd}(\cdots)$ is the odd part of the function $\Gamma(\cdots)$.

1.1 Calculate the averaged perturbation $\Gamma(\Delta_{ij})$ and then find the odd function $\Gamma_{odd}(\Delta_{ij})$ (*Hint*: see the foot note[1]).

1.2 Analysis the stability for the case where two oscillators have **(a)** excitatory connections $g_{syn} = g_{syn}^{excitatory} > 0$; **(b)** inhibitory connections $g_{syn} = g_{syn}^{inhibitory} < 0$. For each case, do the two oscillators tend to synchronize or anti-synchronize?

2 Hopfield model

Here we will study the energy landscapes of a Hopfield network. A Hopfield network is a network of N "binary neurons" (that can each have a value of $+1$ or -1). Each network state is a vector with the values each neuron has in that state. The neurons are connected according to the weight matrix W_{ij}. The states can be thought of as memories that the network converges into from the initial state (sometimes referred to as a cue). The convergence from the cue to the memory is done on the surface of the "energy landscape". The energy at a point $\vec{y}$ is defined as:

$$E(\vec{y}) = -\frac{1}{2}\vec{y}^T W \vec{y} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i W_{ij} y_j \tag{8}$$

Where $\vec{y}$ is any N dimensional binary vector. We will work with a network of $N = 900$ neurons. The memories can therefore be represented as a vector with 900 entries (each can be only $+1$ or -1), or a 30×30 "picture". We are given a "bank" of 60 such pictures (in the file `memories.mat`, Fig. 1), and the goal is to check how well the network can remember them. The memory "quality" is closely related to the energy function defined above. The more "rugged" this energy landscape is, the bigger the chance that the network will converge into a "wrong" memory from a given initial condition.

1. First we will build the code needed for the Hopfield network:

   (a) Given a set of memories $\vec{\boldsymbol{x}}^{(k)}$, compute the connectivity matrix according to the equation:

   $$W_{ij} = \frac{1}{N}\sum_{k=1}^{p} x_i^{(k)} x_j^{(k)} \tag{9}$$

   where p is the number of memories, and $x_i^{(k)}$ is the i-th component of the k-th memory.
   Or, in matrix notation:

   $$W_{N\times N} = \frac{1}{N} X X^T \tag{10}$$

   where X is a $N \times p$ matrix that holds the N dimensional memories,

   $$X = \begin{pmatrix} | & | & | & | \\ \vec{\boldsymbol{x}}^{(1)} & \vec{\boldsymbol{x}}^{(2)} & \cdots & \vec{\boldsymbol{x}}^{(p)} \\ | & | & | & | \end{pmatrix}_{N\times p} .$$

   Note that W is the outer product of X and should be a $N \times N$ matrix. (It is the connectivity matrix of a network that "remembers" these specific p memories).

   (b) In each step ("time t") the update is done according to the equations:

   $$a_i(t) = \sum_{j=1}^{N} W_{ij} y_j(t) \tag{11}$$

   $$y_i(t+1) = \begin{cases} +1 & a_i(t) > 0 \\ -1 & \text{otherwise} \end{cases} \tag{12}$$

[1] The integral (Eq. 4) can be done by extending the range of integration over all time, i.e., $\int_{-\infty}^{\infty}$. Since $\vec{P}(\phi) = 0$ when $\phi < 0$, the integral in Eq. 4 becomes $\int_0^{\infty}$.

where $y_i(t)$ is the "value" ($+1$ or -1) of the i-th neuron at time t ($\vec{y}$ is the current state vector of the whole network).
The first equation can be compactly written in matrix notation:

$$\vec{a}(t) = W\vec{y}(t) \tag{13}$$

This is useful when writing the MATLAB code.
The update stops when for all i:

$$y_i(t+1) = y_i(t) \tag{14}$$

Here, you are going to do the ***asynchronous updating***: only one neuron (y_i) of the state vector ($\vec{y}$) is updated at a time. This i-th neuron can be picked at random, or a pre-defined order can be imposed from the very beginning.

(c) Given $\vec{y}$ and W, compute the energy.
Hint, steps (a) (b) and (c) this could all be done in one line of code (for each step) with matrix and vector multiplication. Look at the vector equations and try to avoid loops.

2. Now, pick your favorite picture from the bank (for your convenience, a visualization of the image bank is given below), and pick an initial condition $\vec{y}\,(0)$ not too close to your image (the initial condition is a vector of length 900 with +1's and -1's).
It will be easiest run the code if you construct a vector `bnk` with the indices of the pictures you want your network to remember. If you do so, the MATLAB code that gives the correct $N \times p$ matrix is simply `X(:, bnk)`, where `X` is the 900×60 matrix given to you with all the memories.
Ex: if you want the network to memorize the selected pictures of no. 12, 18, 21, 35, 48, 55, and 60, create the vector `bnk` $=$ `[12, 18, 21, 35, 48, 55, 60]`, then `X(:, bnk)` gives you the $N \times 7$ matrix.

   (a) Initialize the network weights to "remember" the picture you have chosen.

   (b) **Plot your initial condition in a picture format** (use the function reshape to turn a vector of length 900 to a 30×30 square matrix of size).

   (c) Run the update rule until the states in two consecutive steps are identical.
   In every step, compute the energy and keep it in a vector. (the length of this vector is going to be the number of iterations until convergence)

   (d) **Plot the final state**. Is it the memory you have chosen?

   (e) **Plot the energy as a function of the iteration step**.

3. Repeat steps (a)-(e) but instead of remembering just one picture, **add more and more pictures** (to increase the number of memories p, simply add indices to the vector `bnk`) from the bank to the initialization step (a). **Start from the same initial condition every time**.

4. **Comment on your results**. Does the system converge to one of the initial memories when adding more pictures in the initialization step? Does the energy landscape change when adding more pictures?

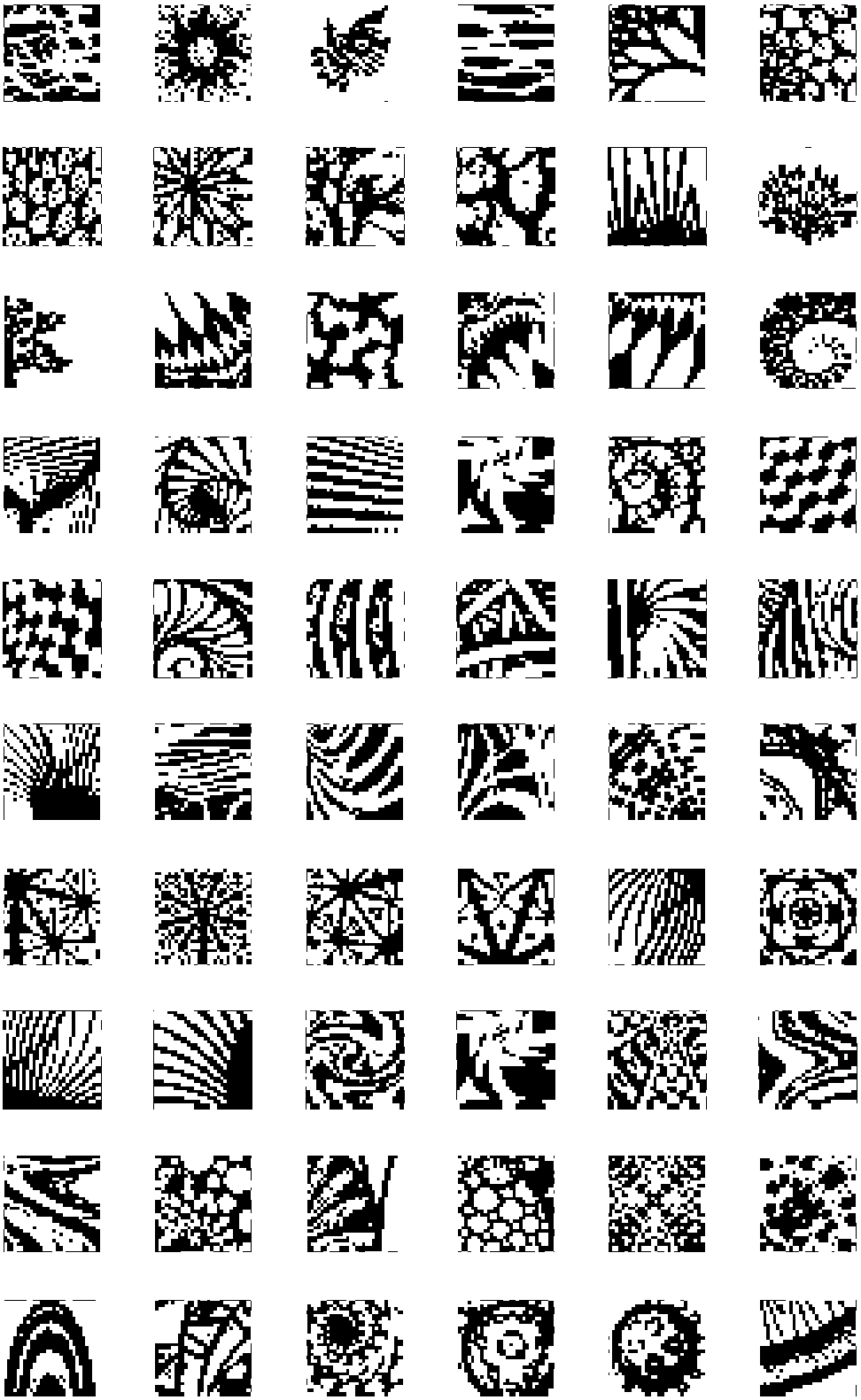

Figure 1: 60 30-by-30 binary pictures.