Due: October 11, 2017 **Homework 2** CPSC 475/BENG 475

## 1 Artificial Neurons

For this part of the assignment you will simulate a neuron based on the leaky integrate-and-fire model. Make sure the files `nsimulator.m` and `problem1.m` are in your working directory. The simulator documentation is available in the usual way (`help nsimulator`), and the script `problem1.m` contains an example of what you can do with the simulator. We suggest that you run the sample script first and then modify it as needed.

1 a Create a simulator with a single neuron, setting the initial membrane potential value to $-60\,\mathrm{mV}$. Run the simulation for $0.2\,\mathrm{s}$ while injecting a current of $0\,\mathrm{nA}$. Repeat the simulation for $1\,\mathrm{nA}$ and $10\,\mathrm{nA}$. Describe what happens to the resting membrane potential as you vary the input current.

— b Repeat the experiments above, this time using different initial membrane potential values: $-80\,\mathrm{mV}$ and $0\,\mathrm{mV}$. Is there any difference in the neuron's behavior? Why or why not?

— c Create a simulator for two neurons. Inject $2.5\,\mathrm{nA}$ of current into Neuron 1 and $0\,\mathrm{nA}$ into Neuron 2. Set the membrane potential of Neuron 1 to $-60\,\mathrm{mV}$, and the membrane potential of Neuron 2 to $-70\,\mathrm{mV}$. Make Neuron 1 inhibit Neuron 2. Simulate for $0.1\,\mathrm{s}$. Repeat the simulation for different values of injected current into Neuron 2: $1\,\mathrm{nA}$ and $10\,\mathrm{nA}$. Compare the behavior of Neuron 2 to the behavior of the neuron in the previous part (single neuron simulator). What is the effect of Neuron 1 on Neuron 2?

— d Repeat the experiment above for an excitatory connection. What is the effect of Neuron 1 on Neuron 2?

— e Create a simulator for two neurons. Inject $2.5\,\mathrm{nA}$ into each one and initialize the membrane potential to $-60\,\mathrm{mV}$ for each one. Connect the two neurons so each excites the other. Simulate without recording for $1\,\mathrm{s}$, and then record for $0.1\,\mathrm{s}$. Do this several times. Do you observe a phase offset between the spikes of the two neurons as seen in the figure below? If not, remove the calls to `nsim.set_mpotential`, allowing the initial membrane potentials to be initialized randomly (between $-50\,\mathrm{mV}$ and $-70\,\mathrm{mV}$). Simulate several times again. Did this make a difference? Repeat for inhibitory connections. Do the neurons tend to synchronize their spikes?

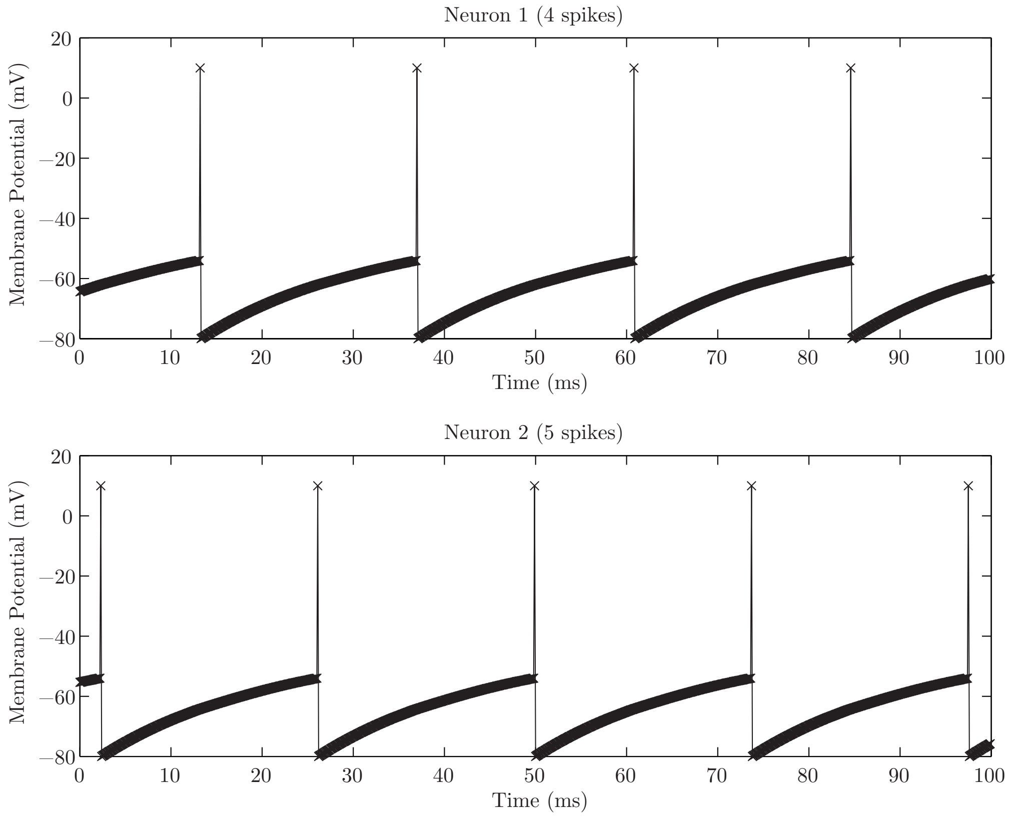


Figure 1: Example of phase offset between two neurons.

## 2 Neural Networks as Linear Systems

2a Multiply a $5 \times 5$ identity matrix by the $5 \times 1$ vector $[1\ 2\ 3\ 4\ 5]^T$. What is the result? We can think of the matrix as a linear function, or a linear transformation, being applied to an input vector, which gives us another vector as output.

— b Now change the values of the matrix along the diagonal to 2 and multiply it by the same input vector. Look at the result. What is the relation between the output and input vectors?

— c Starting with the original identity matrix, change the 0s on each side of the main diagonal (left and right, whenever possible) into 1s. What kind of operation is being done on the input vector now?

— d In Lecture 3, we learned about the *Limulus* and experimented with a simple feedforward model of processing in the lateral neural plexus. In this exercise, we want to represent these types of models in matrix form. That is, we can think of the input and output as vectors in $\mathbb{R}^n$, where $n$ is the number of cells in the layer. (For simplicity, we will consider both layers to have an equal number of cells.) Thus, any linear transformation from input to output can be represented as an $n \times n$ matrix. An appropriate matrix, when applied to the input, will result in the desired output.

Recall equation (5.4) from the notes; in its general form, it can be written as:

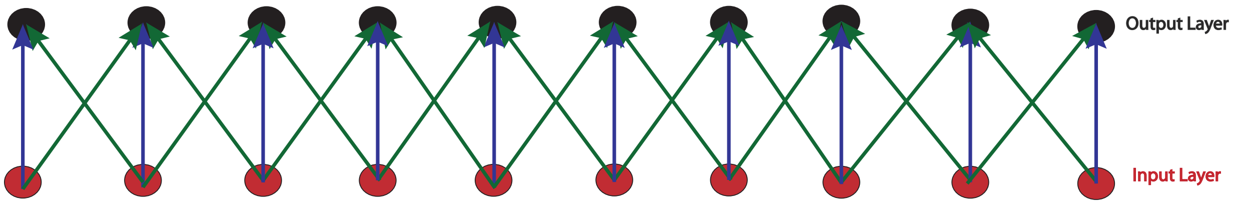
$$F_i = e_i + \sum_{j \in \mathcal{N}(i)} a_{i,j} e_j,$$

where $\mathcal{N}(i)$ is the neighborhood of influence of cell $i$ and each $a_{i,j}$ is negative for inhibitory connections, or positive for excitatory ones. Let $n = 10$ and the neighborhood of influence be one cell on each side of the current cell $i$, e.g. if $i = 5$, $\mathcal{N}(i) = \{4, 6\}$. For boundary cells, i.e. $i = 1$ or $i = 10$, let the neighborhood be $\{2\}$ and $\{9\}$, respectively. As in the lecture, let the weights $a_i = -0.2$, $1 \leq i \leq 10$.

Represent the above equation in matrix form. (*Matlab tip: to avoid having to enter each value manually, you can start with a generic* $10 \times 10$ *matrix and use for-loops to fill in the values.*)

— e Create the $10 \times 1$ input vector $[10\ 10\ 10\ 10\ 10\ 20\ 20\ 20\ 20\ 20]^T$ and plot it using Matlab's `plot` command. What kind of function does this vector look like? Apply your matrix to this input vector. What do you observe about the resulting output vector? (You can ignore the boundary values.) If we interpret the values in the vectors as light intensities, another way to visualize them is by using `imagesc` on each vector separately followed by `colormap gray` (for the output vector, you will get a better result by displaying only the values `2:9`). Remark qualitatively on the results.

— f Now consider the following feedforward model:



Green Links have weight 1/3
Blue Links have weight 1

Find the matrix representing the action of this network. Apply it to the vectors $[10\ 10\ 10\ 10\ 10\ 20\ 20\ 20\ 20\ 20]^T$ and $[10\ 8\ 10\ 8\ 10\ 8\ 10\ 8\ 10\ 8]^T$. Qualitatively, what does this network do? (Again, you can ignore the boundary values.)

— g Suppose we built a multi-level feedforward model using the model in (d) first and then used output from that network as input into the input layer of the model in (f). Remark on what you expect the combined action of this system to be. How can we build a single matrix representing it? Apply this matrix to the vector $[10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10\ 10]^T$ to verify your remark.

## 3 Like Moths to a Flame

Let's take our ability to represent a layer of a neural network in matrix form and apply it to model positive phototactic behavior in moths (that is, their tendency to fly towards a source of light!).

Our model moth has one large spherical eye, with photoreceptors clustered towards the front, but some also located towards the back, giving it a 360 degree field of view. (Its wings and

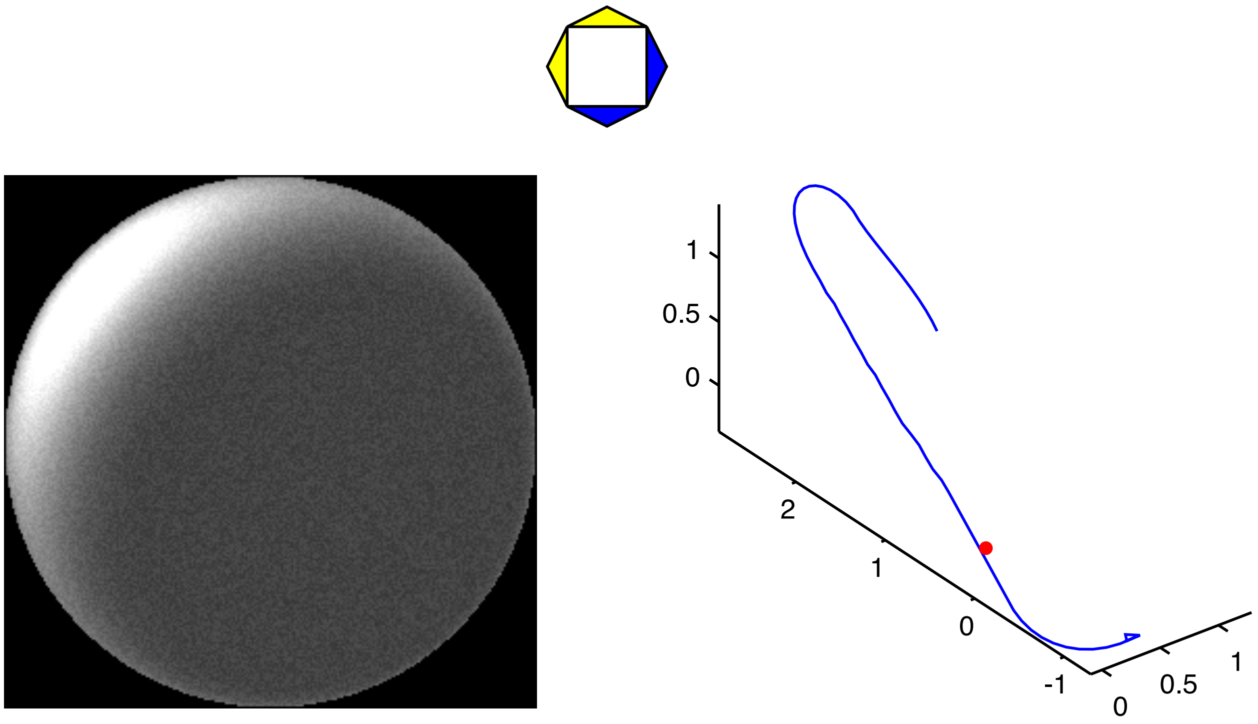


Figure 2: The moth simulator screen. On the left is the image as seen by the moth's eye, showing the light above, behind, and to the left of the moth. On the right is a plot showing the moth's flightpath. The arrows on top represent the activity of the four motor neurons. The motor neurons are currently causing the moth to rotate up and to the left.

body are perfectly transparent, so they're never visible.) The moth has four motor neurons; two controlling its pitch (up/down rotation) and two controlling its yaw (left/right rotation).

The image generated from the spherical eye is represented in two dimensions by a disk of pixels, where forward-facing photoreceptors correspond to pixels in the center of the disk, while rear-facing photoreceptors correspond to pixels near the edge of the disk. This means a ball of light straight ahead of the moth appears as a (filled in) circle in the image, while the same light directly behind the moth appears as a ring around the boundary of the image. In-between cases result in a warped representation of the light as seen in Figure 2. Note that, although the image is square, the relevant content only appears within the inscribed circle—the corners are always black.

Your task will be to design a network that processes an image from the moth's eye and activates the motor neurons to control the moth's flight, guiding it towards the light. At each time step, you will be presented with an updated image reflecting the moth's new position and orientation. You don't need to control the moth's forward flight—it will always fly forward.

To help you visualize and debug your network, you will use the `mothsim` function (in `mothsim.m`). This runs a simulator (see Figure 2) that evaluates a function you supply (`mothcontrol`) to determine the activity of the motor neurons.

3 a Design a network (by specifying the appropriate connections and a weight structure) whose input nodes are the pixels of the image from our model moth eye, and whose four output nodes control the activity of the moth's motor neurons so as to make it fly towards light. Note that the activity of the motor neurons is binary—either firing or not firing—so you must convert the outputs of your network to true or false via an appropriate set of thresholds (i.e., the value above which each output neuron should fire).

Describe your network design and its motivation, along with your choice of thresholds. You don't have to write out all the network's weights individually, as long as you provide a description with enough detail to allow their construction. If you want, you can also include a drawing with your submission, provided you display it using Matlab (don't forget to include the necessary image file when you submit it).

— b Flesh out the code in `mothcontrol.m` to implement your network for a $256 \times 256$ image. Begin by constructing a matrix representation of the network weights you designed above. Write code to apply your matrix to the input image, and test against your thresholds to generate a $4 \times 1$ boolean vector indicating which motor neurons should fire based on the input. (Hint: think about the dimensions of the input vector and matrix required for outputting a $4 \times 1$ vector.)

See `help mothsim` for detailed instructions on how the entries in the output vector should be ordered. Run the simulator with the command `mothsim(mothcontrol)`. After stopping the simulator (`Ctrl+C`), you can inspect the moth's flight path using the "Rotate 3D" tool from the figure window. **Some useful Matlab tips:** Remember that the colon operator `x(:)` can turn any 2-D array into a column vector (by stacking its entries columnwise; see `help colon`). Other functions that might come in handy are `linspace` or `meshgrid`.