

# Programming Tutorial

## Brain, Minds and Machines Summer Course

### Woods Hole, 2018

TA's: Karolina Marcyniak, Francisco J. Flores.

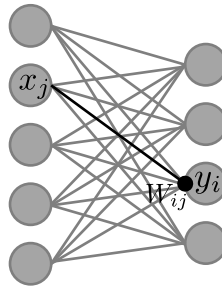
Exercises by Emily Mackevicius.

Adapted from Woods Hole and MIT Computational Neuroscience courses

*Instructions: Go through the below exercises. TAs will cover the solutions to the exercises.*

## Part 1: Matrix operations for a feedforward network

We will construct a 2 layer feedforward linear network, and use matrix operations to calculate its outputs, given its inputs and weights. We'll call the output neurons  $y_1, \dots, y_M$  and input neurons  $x_1, \dots, x_N$ .  $W_{ij}$  is the connection strength (weight) onto neuron  $y_i$  from neuron  $x_j$ . We refer to  $W$  as the weight matrix.



- 1) What does each row of the weight matrix represent? Each column?
- 2) Write an expression for calculating  $y_i$ , the response of the  $i$ th output neuron.
- 3) Write an expression for calculating the contribution of input neuron  $x_j$  to the network output (this should be a vector of length  $M$ ). Note that the total network output is the sum of the contributions from each input neuron.

Matrix notation gives us a compact way of expressing all of this information:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \\ \vdots & \vdots & & \vdots \\ W_{M1} & W_{M2} & \cdots & W_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

- 4) Convince yourself that the equation above captures what you found in parts 2) and 3).

Now we'll use MATLAB to construct an example network, with  $N = 50$  and  $M = 10$ .

- 5) Generate a weight matrix. Assume the weights are random and uniform between -1 and 1 (use `rand`).
- 6) Generate a 50-dimensional pattern of inputs consisting of Gaussian entries (use `randn`).
- 7) Calculate the network output.

## Part 2: Logical operations, for-loops, and plotting (random walk)

We will construct a biased random walk with a reflecting barrier and reset. You can think of this as a simple model of the voltage of a neuron. The voltage increases, with some fluctuations, until it reaches a threshold, at which point it spikes then resets to a resting value, and starts the process again.

Specifically, the voltage of the cell is given by:  $V(t+1) = V(t) + dV(t)$ , where  $dV(t)$  is 1 with probability  $p$ , and -1 with probability  $1 - p$ . We want the system to reset once it hits a maximum ceiling, so we will add the condition:  $V(t+1) = V_{reset}$  if  $V(t) > V_{thres}$ .

- 1) Make a function `generatevoltage(p,T,Vreset,Vthresh,V0)` that takes as input the probability of going up,  $p$ , the number of time steps to simulate,  $T$ , the reset voltage,  $V_{reset}$ , the spike threshold,  $V_{thres}$ , and the initial voltage,  $V_0$ . Your function should return a vector  $V$ , the voltage at all time steps from 1 to  $T$ . To create the function, make a new file `generatevoltage.m` with the following syntax:

```
function V = generatevoltage(p,T,Vreset,Vthresh,V0)
    % your code here
end
```

- 2) Inside the function, create the vector  $V = \text{zeros}(1,T)$ ; that will hold voltage values at each step of the process:

```
for t = 1:T-1
    % update V(t+1)
end
```

- 3) Run your function to simulate a neuron with initial voltage of -65mV, threshold of -45mV, and reset voltage of -70mV. Choose  $p$  so that your neuron has an average firing rate of approximately 10Hz, assuming each time step corresponds to 1ms (feel free to try several values of  $p$ ). Calculate the voltage values for 1 second
- 4) Plot the voltage as a function of time using `plot`. Label your axes using `xlabel` and `ylabel`.

## Part 3: Convolution to estimate voltage response to a spike train

A convolution is a mathematical operation on two functions that expresses the amount of overlap of one function as it is shifted over the other. Mathematically, it is defined as

$$\begin{aligned} [f * g](t) &= \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau \\ &= \int_{-\infty}^{+\infty} g(\tau)f(t - \tau)d\tau \end{aligned}$$

We will use convolution to estimate the voltage response of a neuron to an incoming spike train.

- 1) Generate 3 seconds of a Poisson spike train with firing rate 20Hz. Use `spiketrain = rand(1,N) > (1-p)`; where N is the total number of time steps (each time step should be 1 ms) and p is the probability of spiking in any given time step (you need to calculate N and p).
- 2) Construct a kernel, the response of the neuron to one spike at t=0. We assume the neuron is linear, that is, the response to multiple spikes is the sum of the responses to each individual spike. For the kernel, use an exponential with mean  $\mu$  of 5ms. Calculate the kernel for values between -50 and 50ms: `k = exppdf(-50:50, mu)`; Plot the kernel.
- 3) Use `conv` to estimate the voltage response of the cell to the spike train by convolving your spike train from 1) with the kernel from 2). (Hint: convert `spiketrain` to a double from a logical: `spiketrain = double(spiketrain)` ; ).
- 4) Plot the voltage and the spike train on two separate panels using `subplot`. Make sure to align them properly in time (type `doc conv` to see how `conv` works). Zoom in to see what happens to the voltage when incoming spikes occur in rapid succession. It may help to use `linkaxes` to align the two panels when you zoom.

## Part 4: Convolution to detect edges in images

In an image, edges are where the image is different from its neighbors. Convolution in two dimensions is often used for edge detection in image processing. The output of this operation is very similar to the response of cells in primary visual cortex, which respond selectively to oriented edges. The following kernel will be zero in regions of the image where neighboring pixels have the same value, and nonzero for edges: `k = [0 0 0; 0 1.125 0; 0 0 0] - .125*ones(3,3)` ;

- Load the octopus image (from <http://www.bbc.co.uk/nature/life/Octopus>). Plot it using `imagesc(octopus)`; `colormap gray`;
- 1) Use `conv2` to convolve the image with the kernel k, and plot the result using `imagesc`. Notice that edges can be darker and/or lighter than the gray background.
  - 2) Plot the absolute value of the convolution (this will show both positive and negative edges as lighter than the background). You've built a simple edge detector!
  - 3) If you have time, try modifying your filter, and try it on different images. See [http://www.mathworks.com/help/matlab/import\\_export/supported-file-formats.html](http://www.mathworks.com/help/matlab/import_export/supported-file-formats.html) for importing different data formats into MATLAB. Hint: usually, `importdata` works.

## Part 5: Correlation to analyze premotor neural data.

We'll use cross-correlation to analyze the timing relation between the song of a juvenile zebra finch and neural data recorded while he was singing.

- 1) Load the file *MackeviciusData.mat*, which includes an extracellular recording, `units`, and raw sound data, `song`, simultaneously recorded at the sampling frequency `fs`.
- 2) In two panels, plot the song data and the neural data, with the correct time axes. Listen to the song using `sound(song, fs)`;

You'll notice that both the sound data and the neural trace have a lot of fluctuations. We want to ignore the fast fluctuations, and instead focus on whether broad bursts in neural activity precede song syllables (broad bursts of sound). Therefore, we will compute the log power of each signal, so we're left with signals showing the broad changes in power of each signal.

- 3) To smooth the power of each signal by a 2.5 ms Gaussian run the following:  
`10*log10(conv(signal.^2, gausswin(ceil(fs*.025)), 'same'))`;  
 Plot these new traces. What does the `10*log10` transformation means?
- 4) Compute the cross correlation between the song and the neural activity using `xcorr`. Plot the cross correlation as a function of the time lag. Does it look like bursts of neural activity precede song syllables? Use `findpeaks` to find the exact temporal relation between spike firing and song production.

## Part 6: Singular Value Decomposition (SVD) on images of faces

We'll perform SVD on images of faces. SVD decomposes a matrix  $X$  into the product  $USV'$ , where  $U$  and  $V$  are square matrices and  $S$  is diagonal. If  $X$  is  $n \times m$ , columns of  $U$  are a basis in  $n$ -dimensional space, and columns of  $V$  are a basis in  $m$ -dimensional space. If you've mean-subtracted your data, these are the same bases you get in PCA (the eigenvectors of the covariance matrices,  $XX'$  and  $X'X$  respectively).  $S$  contains the squared eigenvalues of these covariance matrices on the diagonal ( $XX'$  and  $X'X$  have the same eigenvalues). Using SVD, we can look at the principal modes in both the pixel space (in our case  $137 \times 86$  pixels) and in the image space (in our case 213 total images).

We'll use faces from the JAFFE database, which contains images of several subjects each displaying several facial expressions (*Michael J. Lyons, Miyuki Kamachi, Jiro Gyoba. Japanese Female Facial Expressions (JAFFE), Database of digital images (1997).* [http://www.kasrl.org/jaffe\\_info.html](http://www.kasrl.org/jaffe_info.html)). The file `jaffe.mat` contains a matrix `IMS` of image data, where rows represent images and columns represent pixels. Each image has been flattened into a vector of pixels. To see the  $i$ th image, use the command:  
`imagesc(reshape(IMS(i,:), 137, 86)); colormap gray;`

- 1) Mean-subtract your data so that the mean of each pixel is 0 and the mean of each image is zero.
- 2) Use the command `svd` to compute  $U$ ,  $S$  and  $V$ . Which is a basis for the space of pixels? Which is a basis for the space of images? Plot the singular values and estimate how many dimensions are needed to capture most of the dataset.

- 3) Use `reshape`, `imagesc` and `subplot` to visualize the first 9 elements of the pixel-space basis.
- 4) Experiment with how the images look restricted to a lower dimensional basis. Remember that  $X = USV'$ , and set all but the first  $n$  columns of  $V$  to zero for  $n = 5, 20, 50, 137 \times 86$ .
- 5)  $U$  is a basis for the space of images, of which there are 213. The variable `EMind` indicates the emotion of each image, and `IDind` indicates the subject id of each image. Plot the first column of  $U$  as a function of the emotion, and also as a function of the subject id. Does this direction seem to better capture differences between subjects or between emotions?

## Part 7: Estimation of spectro-temporal receptive fields with spike-triggered averaging

In this exercise, we will demonstrate how to estimate the spectro-temporal receptive field (STRF) using spike-triggered averaging (STA). First, we will generate a spike train from a model neuron with a spectro-temporal kernel that we will provide. Whenever the stimulus is sufficiently correlated with the kernel, the neuron will fire. Next, we will compute the STA (the average stimulus that precedes a spike). We will compare the STA with the kernel used to generate the spike train, note that they look similar, and comment on the limitations that prevent us from perfectly estimating the STRF.

We begin by constructing a stimulus to present to the neuron. We know the neuron is selective to time-varying sound, but we want to know which sound best excites the neuron. Therefore, we want a stimulus that will try many different sequences of sounds with equal probability. The MATLAB file *generatestimulus.m* generates such a stimulus in the form of a 2D matrix where the rows represent 50 logarithmically spaced tone frequencies and the columns represent time bins of width 1ms. Entry  $(i, j)$  of the matrix is the amplitude of tone  $i$  at time step  $j$ . The stimulus is constructed so that tones turn on and last for 30 ms, with a slight ramping at the onset and offset to make transitions sound less abrupt. Each tone has a 10% probability of turning on in a given 30 ms window.

- 1) Run *generatestimulus.m* to construct the stimulus. The code will plot the stimulus matrix. Listen to the stimulus using the provided function *playstim.m*. Just listen to a few seconds of the stimulus, because the whole stimulus is 1000 s long, and may crash your computer if you try to play it. Make sure *playstim.m* is in your MATLAB path, and your speakers or headphones are working and execute the following:

```
freq = logspace(2,4,50);% 50 log-spaced freqs 100-10000Hz
dt=0.01;% sampling interval in seconds
PlayStim(Stimulus(:,1:(3/dt)),freq, dt);% play first 3s
```

Next we will construct an STRF for our model neuron. The file *generatekernel.m* returns an array of 100 time bins by 50 frequency bins, containing the STRF. The STRF is the sum of bivariate Gaussian distributions.

- 2) Run *generatekernel.m* to construct the STRF. The code will plot the kernel matrix. Listen to STRF using *playstim.m*. This is the stimulus that best excites our model neuron. Plot the STRF using *imagesc*. Is this receptive field separable? Briefly explain why or why not.

Now we are ready to simulate how the neuron responds to the stimulus. To do this, you will slide the kernel across the stimulus, and at each time bin  $t$  calculate the integral (sum) of the element-wise product between the kernel and the stimulus that occurred between time  $(t - 100)$  and time  $(t - 1)$ . This is our estimate of how strongly the neuron is excited by the stimulus at each time. Our simulated neuron should spike whenever this excitatory drive exceeds some threshold.

- 3) Write code to calculate the excitatory drive to the neuron at each time in the stimulus. You can start 100ms into the stimulus (the STRF is 100 ms long). Your code should record a spike whenever excitatory drive to the neuron exceeds a threshold. Choose a threshold such that the neuron spikes approximately 10,000 times over the entire stimulus.
- 4) Make a figure with the following three subplots: 5 seconds of the stimulus; the excitatory drive to the neuron during this part of the stimulus; and the spiking response of the neuron to this part of the stimulus.

Now you are ready to compute the STA from the spike train.

- 5) Write code to calculate the average 150 ms of stimulus that precedes each spike. Plot this recovered STA using 'surf' (see *generatekernel.m* for an example of how to plot a matrix using 'surf'). Listen to the STA and the stimulus with *playstim.m* and briefly comment on how they sound.
- 6) The kernel is 100ms long, and we computed a STA that is 150ms long. What do you notice in the part of the STA corresponding to 150-100 ms prior to a spike? How well did the STA recover the neuron's receptive field? What parts of the kernel did the STA fail to recover? Why do you think this is? Think of the number of spikes, the type of stimulus needed to perform this analysis, and the baseline firing rate of the neuron.