Neural Coding

Naureen Ghani

December 2, 2017

Introduction

Neural coding is the study of how computational variables are encoded in neural activity. One early example of this is the legendary Hubel and Wiesel experiments on the cat visual system. The pair began with what was known: light stimulates light-sensing receptor cells in the retina of the eye and different receptor cells respond to stimuli in different parts of the retina's visual field. However, Hubel and Wiesel were studying neurons in higher-functioning areas of the brain not previously studied. They were frustrated by feeble responses in cortical neurons. They were using small spots of light or a black dot on a clear glass slide, projected onto a screen, as a stimulus. One day Hubel accidentally moved the glass slide a little too far, bringing its faint edge into view. Suddenly, those neurons started firing.

In this way, Hubel and Wiesel discovered that cortical neurons didn't respond to simple points of light, but rather to lines of a specific orientation. Some neurons responded to horizontal lines, others to vertical lines, and still others to orientations in between. For their work, the team won the 1981 Nobel Prize in Physiology or Medicine.

The Hubel and Wiesel experiments illustrate neural coding of visual sensory processing. A cat has been anesthetized and placed in front of a screen, with its eyelids open. The tip of a tungsten wire has been placed inside its skull, and lodged next to a neuron in the visual cortex. Even though the cat is no longer conscious, cortical neurons are still active. By connecting an amplifier to the tungsten wire, we can visualize weak electrical responses from the neuron being recorded. The amplified signal can also be connected to a loudspeaker, and we can *hear* neurons.

The technical term for a *spike* in the electrical signal of a neuron is an *action potential.* The frequency of spiking is dependent on the properties of the stimulus. In a *raster plot,* each black bar represents one *spike* or *action potential* and each row of black bars is a *spike train.*

Represent a spike train in MATLAB

In this tutorial, we will represent spike trains as MATLAB matrices. Let each element of a matrix represent a time interval of 1 ms. If there is a spike in this time interval, then we set the value of the element to 1, else we set it to 0. In other words, a spike train contains binary data. Open up your MATLAB command window, type the following command:

```
myFirstSpikeTrain = [0 1 0 0 0 1 0 0 1 0];
```

If this first spike train starts at time 0 and each element represents a 1 ms bin, then there are 3 spikes in this train. The spikes occur between 1-2 ms, 5-6 ms, and 9-10 ms. Based on this spike train, write code to identify:

1. What is the duration? {Ans: 10 ms}
2. What is the mean firing rate? {Ans: 300 Hz}

Generate a synthetic spike train in MATLAB

We use a concept in probability theory known as the *Poisson process* to simulate spike trains that have characteristics close to real neurons. For now, we will use a simple formula derived from the Poisson process. Dayan and Abbott explain how to so:

Spike sequences can be simulated by using some estimate of the firing rate, ***fr,*** *predicted from knowledge of the stimulus, to drive a Poisson process. A simple procedure for generating spikes in a computer program is based on the fact that the estimated probability of firing a spike during a short interval of duration* ***dt*** *is* ***fr*dt.*** *The program progresses through time in small steps of size dt and generates, at each time step, a random number* ***x*** *chosen uniformly in the range between 0 and 1. If* ***x*** $<$ ***fr*dt*** *at that time step, a spike is fired; otherwise it is not.*

In this technique, we are marching through time in small steps of size **dt** (i.e. 1 ms) and deciding if there should be a spike at this time point. We can break this down into three steps:

1. Compute the product of **fr*dt.** Let's stimulate a 10 ms long spike train for a neuron firing at 100 Hz. Thus, **fr** = 100 Hz, **dt** = 1 ms and **fr*dt** = 0.1 (remember that Hz is the inverse of s and 1 ms is 1/1000 s, so fr*dt is dimensionless).
2. Generate uniformly distributed random numbers between 0 and 1. In MATLAB, use the function *rand()* to do so. *rand(1, 10)* will generate 10 randon numbers.
3. Compare each random number to fr*dt. If the product is less than fr*dt (x $<$ fr*dt), then there is a spike.

We can summarize these three steps into the code below:

```
function [spikeMat, tVec] = poissonSpikeGen(fr, tSim, nTrials )

dt = 1/1000; % s
nBins = floor(tSim/dt);
spikeMat = rand(nTrials, nBins) < fr*dt;
tVec = 0:dt:tSim-dt;
end
```

An example output is:

```
myPoissonSpikeTrain =

    0 1 0 0 0 0 0 0 1 0
```

Repeat this simulation 20 times. We could just type the commands 20 times but a more efficient way is to create a 2-D matrix. Each row of the matrix will represent one simulation. Here is the code:

```
fr = 100; % Hz
dt = 1/1000; % s
nBins = 10; % 10 ms spike train
nTrials = 20; % number of simulations
spikeMat = rand(nTrials, nBins) < fr*dt;
```

Write the poissonSpikeGen function in MATLAB

A *function* is way to package code for repeated use. Each function has an input and output. In this code, we have three inputs:

1. firing rate **fr**
2. duration of simulation **tSim**
3. number of trials **nTrials**

We have introduced a new variable here– duration of simulation **tSim.** This is another way to represent **nBins.** The number of bins (**nBins**) is the same as **tSim/dt,** where we have chosen **dt** to be 1 ms. Here is the code:

```
function [spikeMat, tVec] = poissonSpikeGen(fr, tSim, nTrials )

dt = 1/1000; % s
nBins = floor(tSim/dt);
spikeMat = rand(nTrials, nBins) < fr*dt;
tVec = 0:dt:tSim-dt;
end
```

This function is automatically saved as poissonSpikeGen.m by MATLAB. It has two outputs:

1. spike matrix **spikeMat**
2. time vector **tVec**

In time vector **tVec**, each time stamp is 1 ms long (**dt**). The last time stamp will start at **tSim - dt,** which represents the time interval between **tSim - dt** and **tSim.**

Let us now simulate 20 spike trains for a neuron firing at 30 Hz for a period of 1 s. First, clear the MATLAB workspacing using the function *clc.* The, we can use our function:

```
[spikeMat, tVec] = poissonSpikeGen(30, 1, 20)
```

Plot the spike matrix

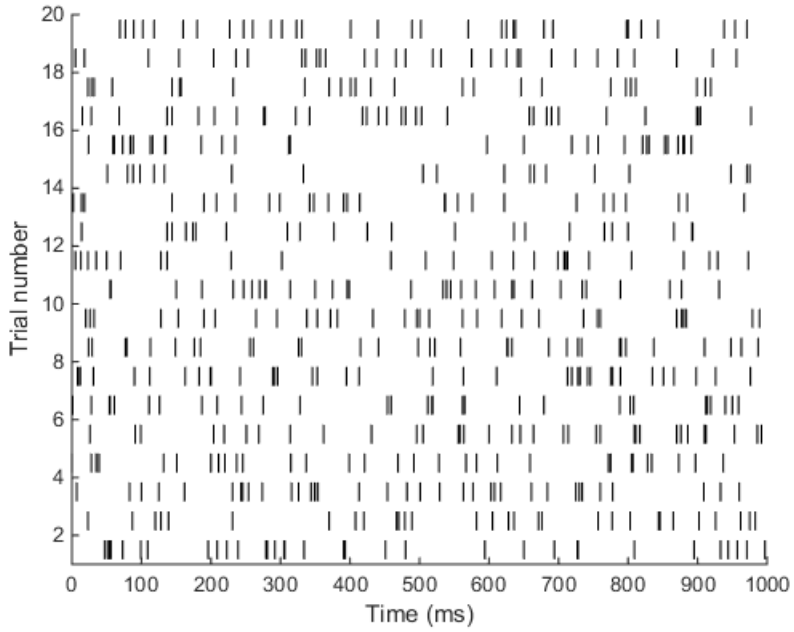
Let's visualize the data by making a raster plot. Here is the code for a function *plotRaster*:

```
function [] = plotRaster(spikeMat, tVec)
% Visualize raster plot
hold all;
for trialCount = 1:size(spikeMat,1)
    spikePos = tVec(spikeMat(trialCount,:));
    for spikeCount = 1:length(spikePos)
        plot([spikePos(spikeCount) spikePos(spikeCount)], [trialCount-0.4
            trialCount+0.4]);
    end
end

ylim([0 size(spikeMat, 1)+1]);
```

This code cycles through each trial, finds all the spike times in that trial, and draws a black line for each spike. Once you have saved this function, use it visualize the spike matrix and label the axes. Here is the code to do so:

```
[spikeMat, tVec] = poissonSpikeGen(30, 1, 20);
plotRaster(spikeMat, tVec*1000);
xlabel('Time (ms)');
ylabel('Trial Number');
```

Here is our raster plot:



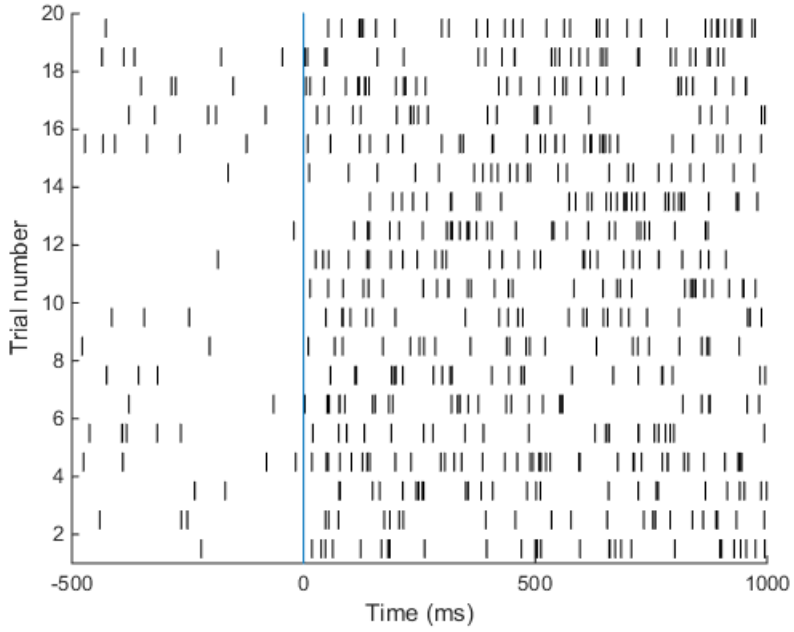
Quantify the spike matrix in MATLAB

Let's calculate the mean firing rate for each spike train in the raster plot. We can write a *for loop* to repeatedly execute code in MATLAB. To find the mean firing rate, we can calculate the number of action potentials in each trial and divide the time interval (remember to use *s* for finding *Hz*). Here is the code to do so:

```
[numTrials, timeLength] = size(spikeMat);

time_s = timeLength./1000;

numAP = cell(1,1);
for i = 1:numTrials
    numAP{i} = length(find(spikeMat(i,:)==1));
end

numAP = cell2mat(numAP');
mean_firingRate = numAP./time_s;
```

Plot a realistic spike matrix

Let us apply the functions we wrote to plot a realistic spike matrix. Imagine a neuron in in the cat visual cortex has a *baseline firing rate* of 6 Hz. Upon presentation of a vertical line, the firing rate of this neuron increases to 30 Hz. Make a raster plot representing the firing activity of this neuron. The baseline period is 500 ms and the visual stimulus was presented for 1 second. The plot should look like this:



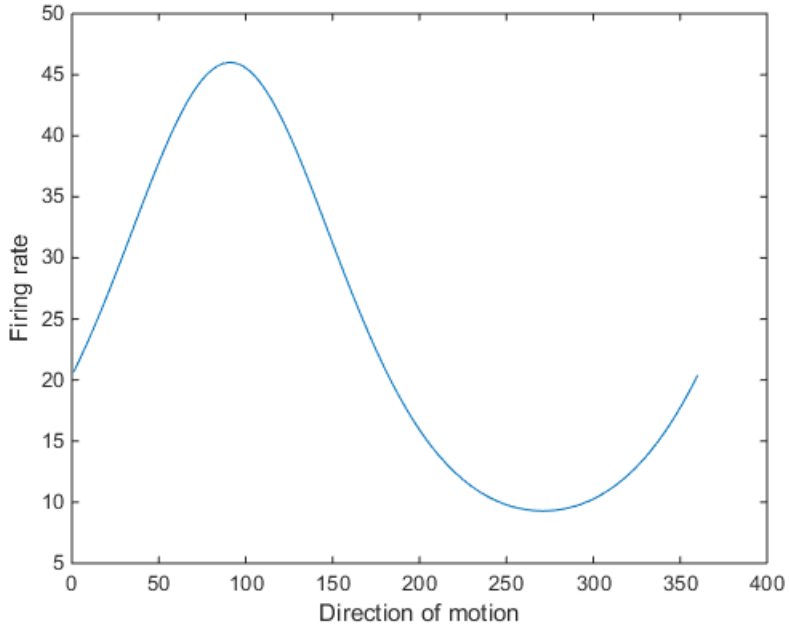
Here is the code to make this plot:

```
% simulate the baseline period
[spikeMat_base, tVec_base] = poissonSpikeGen(6, 0.5, 20);
tVec_base = (tVec_base - tVec_base(end))*1000 - 1;

% simulate the stimulus period
[spikeMat_stim, tVec_stim] = poissonSpikeGen(30, 1, 20);
tVec_stim = tVec_stim*1000;

% put the baseline and stimulus periods together
spikeMat = [spikeMat_base spikeMat_stim];
tVec = [tVec_base tVec_stim];

% plot the raster and mark stimulus onset
plotRaster(spikeMat, tVec);
hold all;
plot([0 0], [0 size(spikeMat, 1)+1]);

% label the axes
xlabel('Time (ms)');
ylabel('Trial number');
```

Simulating spike trains like the ones in the raster plot above requires only one piece of information: *the firing rate of the neuron.*

Summary

Neural encoding transforms information from the *physical space* to the *neural space.* The physical space consists of physical properties of objects (i.e. direction, shape, speed, color, and loudness). The neural space consists of properties of neurons (firing rate, single cell or population, and dendritic spine density). To understand the encoding of a neuron, we use a *tuning curve.*



A tuning curve is a simple graphical representation of neural encoding. On the *x-axis,* we represent the *physical space.* On the *y-axis,* we represent the *neural space.* Neuroscientists determine the tuning curve of a neuron by presenting the animal with various stimuli (varying along the *x-axis*) while recording the activity of that neuron.

Tuning curves translate physical quantities (direction of motion) into firing rate. The motivation for simulating spike trains is to understand the significance of the firing rate in neural encoding.