Anand Sekar

Professor Shea-Brown

AMATH 342

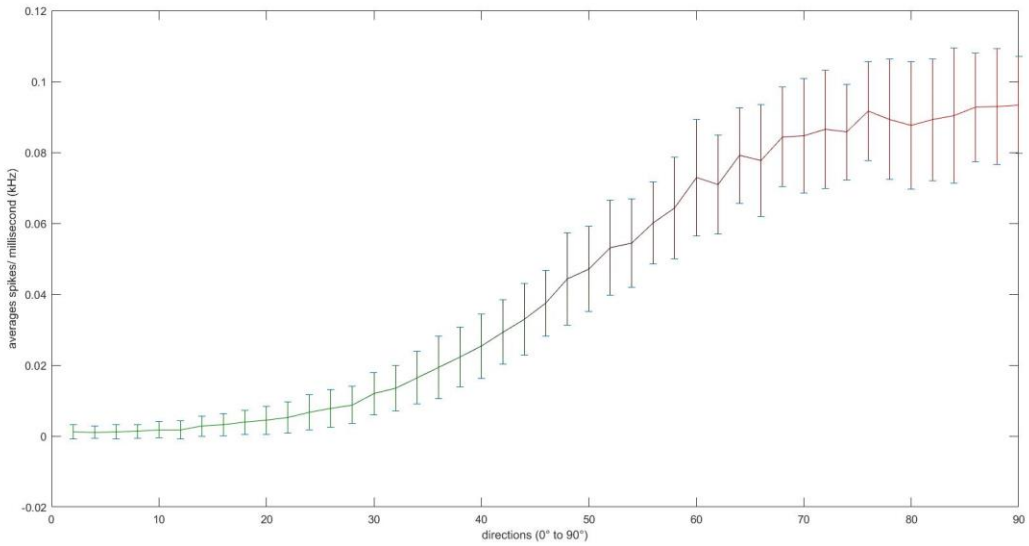
1/30/17

AMATH 342 HW 2

1. Maximum Likelihood Discrimination

Beginning Investigation

Before delving into the investigation of maximum likelihood discrimination, I was curious what the tuning curve for this noisy data looked like, so I applied last week's homework's code to this set of data, and acquired this tuning curve:



Here is the code that was used to acquire the above tuning curve:

```
clear all;
%create vector for tuning curve
%testing for every 2 degrees
tuning_curve_averages = 1:1:45;
tuning_curve_direction = 1:1:45;
tuning_curve_std = 1:1:45;

for direction = 1:45
    x1 = direction * 2;
    %input('Input your test stimulus ')
```

```
    ntrials = 100;
    %input('How many repeated trials would you like to perform? ');

    maxrate = 300; % 30 Hz max firing rate

    rate = maxrate*tuningCurve(x1);
    tau = 100;      % adaptation time constant in msec
    nmsec = 300;    % number of milliseconds to record for
    times= 1:nmsec; % time units

    spiketrain = zeros(ntrials,nmsec);      % set up output data

    ratecurve = rate*exp(-times/tau)*.001;  % adapting rate function

    for j = 1:ntrials;
        for i = 1:nmsec;
            if(rand(1)<ratecurve(i)),
                spiketrain(j,i) = 1;
            end;
        end;
    end;
%     at this point, you now have a spiketrain to analyze
%     get average firing rate and standard deviation
average_firing_rate = 1:ntrials;
    for n = 1:ntrials
        current_sum = sum(spiketrain(n,:));
        current_rate = current_sum / nmsec;
        average_firing_rate(n) = current_rate;
    end

    total_average_firing_rate = mean(average_firing_rate);
    standard_deviation = std(average_firing_rate);

%     add averages and standard deviation to vector
    tuning_curve_averages(direction) = total_average_firing_rate;
    tuning_curve_std(direction) = standard_deviation;
end;
degrees_label = tuning_curve_direction * 2;
figure;
errorbar(degrees_label, tuning_curve_averages, tuning_curve_std);
xlabel('directions (0° to 90°)')
ylabel('averages spikes/ millisecond (kHz)')
```

I do understand that this above curve and respective code is relatively irrelevant to the given objective, but it does help me visualize what I should expect in terms of how much the neuron should fire given different inputs. The higher the input, the more the neuron should fire.

COMPARING INPUTS

I first began by plotting a histogram of input '50' and '60' to visualize how much they overlapped. I modified the generateNoisyData.m file to generate two separate spiketrains (one for '50' and one for '60'). The x-axis of the histogram represents the total number of times a neuron could fire during a trial, and the y-axis of the histogram represents the frequency of the aforementioned occurrence – normalized across all the trials. Here is the code:

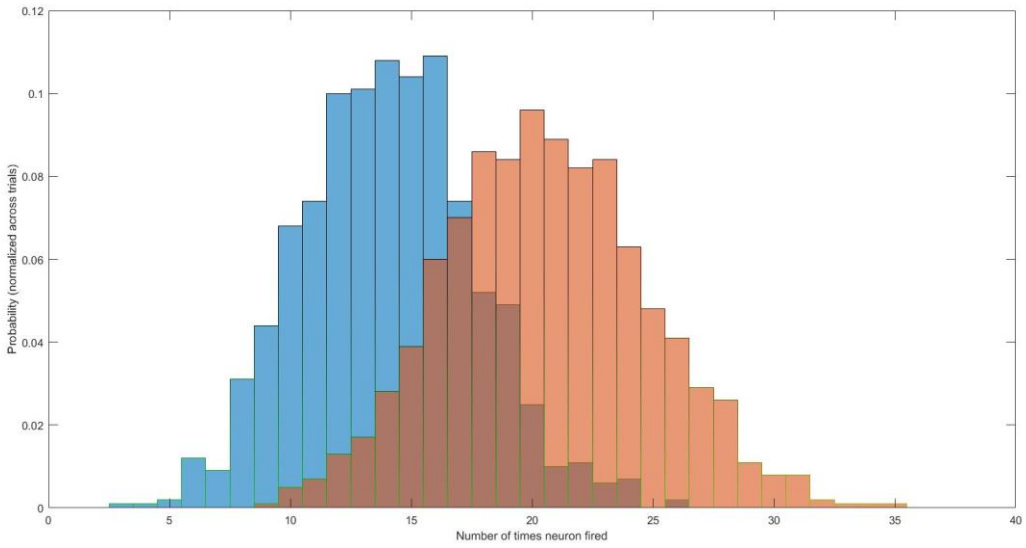
```
clear all;
x1 = 50;
ntrials = 1000;
maxrate = 300; % 30 Hz max firing rate
rate_1 = maxrate*tuningCurve(x1);
tau = 100;      % adaptation time constant in msec
nmsec = 300;    % number of milliseconds to record for
times= 1:nmsec; % time units

spiketrain = zeros(ntrials,nmsec);      % set up output data
ratecurve_1 = rate_1*exp(-times/tau)*.001;  % adapting rate function
error = 0;
for j = 1:ntrials;
    for i = 1:nmsec;
        if(rand(1)<ratecurve_1(i)),
            spiketrain(j,i) = 1;
        end;
    end;
end;

average_firing_total_1 = 1:ntrials;
average_firing_total_2 = 1:ntrials;

%for the first spiketrain
for n = 1:ntrials
    current_sum_1 = sum(spiketrain(n,:));
    average_firing_total_1(n) = current_sum_1;
end

%creating new spiketrain
x2 = 60;
rate_2 = maxrate*tuningCurve(x2);
spiketrain_2 = zeros(ntrials,nmsec);      % set up output data
ratecurve_2 = rate_2*exp(-times/tau)*.001;  % adapting rate function
for j = 1:ntrials;
    for i = 1:nmsec;
        if(rand(1)<ratecurve_2(i)),
            spiketrain_2(j,i) = 1;
        end;
    end;
end;

%for the second spiketrain
```

```
for n = 1:ntrials
    current_sum_2 = sum(spiketrain_2(n,:));
    average_firing_total_2(n) = current_sum_2;
end

total_average_firing_rate_1 = round(mean(average_firing_total_1));
disp('total average firing sum 1:');
disp(total_average_firing_rate_1);
total_average_firing_rate_2 = round(mean(average_firing_total_2));
disp('total average firing sum 2:');
disp(total_average_firing_rate_2);

%this histogram plots the number of time each spiketrain
%fires across the trials. X axis is #times fired
%and Y axis is number of trials which achieved that number
h1 = histogram(sum(spiketrain,2),'normalization','probability');
hold on;
ylabel('Probability (normalized across trials)');
xlabel('Number of times neuron fired');
h2 = histogram(sum(spiketrain_2,2),'normalization','probability');

%find where the two curves intersect on the x-axis (min and max bins)
binh1 = ceil(h1.BinEdges);
binh2 = ceil(h2.BinEdges);
binh2 = binh2(1:end-1);
valh1 = h1.Values;
valh2 = h2.Values;
%we know that bin1 < bin2 therefore min(bin2) and max(bin1) encloses
the
%region at which they intersect
intersect_min = min(binh2);
intersect_max = max(binh1);
error1 = 0;
error2 = 0;

for i=1:(numel(binh1)-1)
    k = find(binh2 == binh1(i));
    if(numel(k)>0)
        if(valh1(i) < valh2(k))
            error1 = error1 + valh1(i);
        else
            error2 = error2 + valh2(k);
        end
    end
    k = [];
end
error = (error1+error2)/2;
disp('Error 1: ');
disp(error1);
disp('Error 2: ');
disp(error2);
```

Results:

```
total average firing sum 1: 14
total average firing sum 2: 21
Error 1: 0.1620
Error 2: 0.2400
```

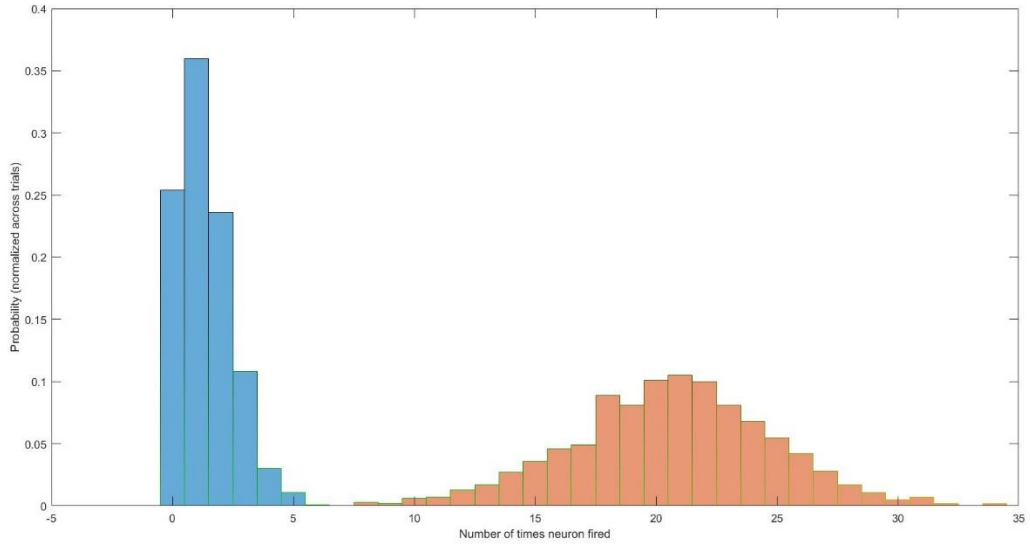
The total average firing rates were just values to make confirm my previous observation, based off the tuning curve, that the higher the input, the more times the neuron fires.



Both sets of data were taken across 1000 trials. The blue represents the histogram for input ‘50’ or x1 and the orange represents the histogram for input ’60’ or x2. Error 1 corresponds to the probability of error if I had guessed an input of ‘60’ on the right side of the centerline marked above, and Error 2 corresponds to the probability of error if I had guessed an input of ‘50’ on the left side of the centerline marked above. Averaging the two error probabilities would give you the total error probability.

To answer the question of how much higher or lower does the closeby stimulus need to be to get an error probability of 10%, we need to test the error probability from a given input, let’s say ‘50’, and incrementally increase the input until the two areas of overlap are far enough apart to achieve that low error probability. Before doing that, here are the results when the starting input is ‘20’ rather than ‘50’. This is between ‘20’ and ‘60’.

```
total average firing sum 1: 1
total average firing sum 2: 21
Error 1: 0
Error 2: 0
```



It's clear from the results that there is no overlap between these curves, and that any chance of error is 0. This is a more extreme case of what we're trying to achieve – by distancing the two inputs and lowering probability. Here is what happens when you incrementally separate the two inputs, comparing '50 + x' to '50' until a lesser than .1 probability is achieved:

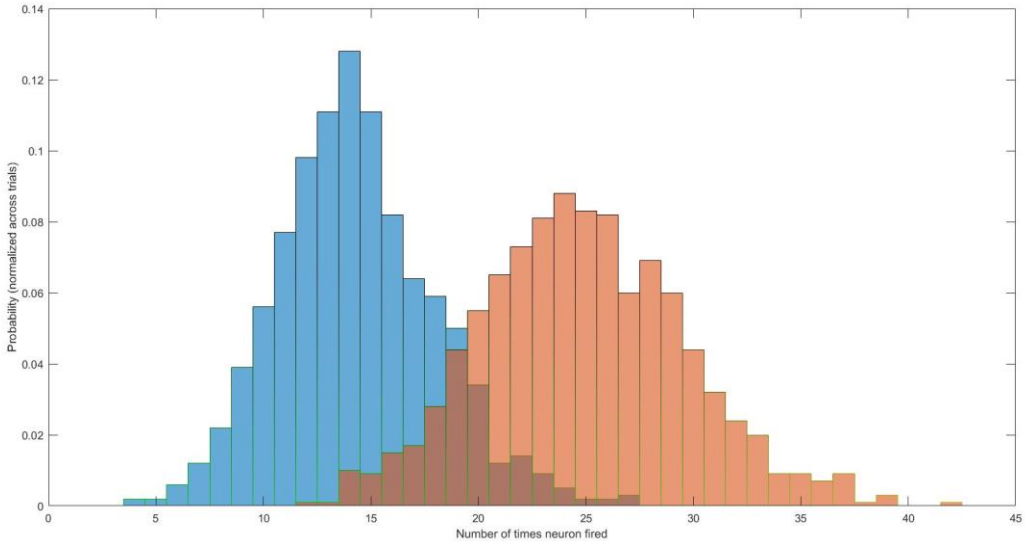
```
error = 1;
x = 0;
x2 = 0;
while error > .1
    x = x+1;
    %%constants
    x1 = 50;
    ntrials = 1000;
    maxrate = 300; % 30 Hz max firing rate
    rate_1 = maxrate*tuningCurve(x1);
    tau = 100;      % adaptation time constant in msec
    nmsec = 300;    % number of milliseconds to record for
    times= 1:nmsec; % time units

    spiketrain = zeros(ntrials,nmsec);      % set up output data
    ratecurve_1 = rate_1*exp(-times/tau)*.001;  % adapting rate
function
error = 0;
    for j = 1:ntrials;
        for i = 1:nmsec;
            if(rand(1)<ratecurve_1(i)),
                spiketrain(j,i) = 1;
            end;
        end;
    end;

    average_firing_total_1 = 1:ntrials;
```

```
    average_firing_total_2 = 1:ntrials;

    %for the first spiketrain
    for n = 1:ntrials
        current_sum_1 = sum(spiketrain(n,:));
        average_firing_total_1(n) = current_sum_1;
    end

    %creating new spiketrain
    x2 = x1 + x;
    rate_2 = maxrate*tuningCurve(x2);
    spiketrain_2 = zeros(ntrials,nmsec);      % set up output data
    ratecurve_2 = rate_2*exp(-times/tau)*.001;  % adapting rate
function
    for j = 1:ntrials;
        for i = 1:nmsec;
            if(rand(1)<ratecurve_2(i)),
                spiketrain_2(j,i) = 1;
            end;
        end;
    end;

    %for the second spiketrain
    for n = 1:ntrials
        current_sum_2 = sum(spiketrain_2(n,:));
        average_firing_total_2(n) = current_sum_2;
    end

    total_average_firing_rate_1 =
round(mean(average_firing_total_1));
    disp('total average firing sum 1:');
    disp(total_average_firing_rate_1);
    total_average_firing_rate_2 =
round(mean(average_firing_total_2));
    disp('total average firing sum 2:');
    disp(total_average_firing_rate_2);

    %this histogram plots the number of time each spiketrain
    %fires across the trials. X axis is #times fired
    %and Y axis is number of trials which achieved that number
    h1 = histogram(sum(spiketrain,2),'normalization','probability');
    hold on;
    ylabel('Probability (normalized across trials)');
    xlabel('Number of times neuron fired');
    h2 =
histogram(sum(spiketrain_2,2),'normalization','probability');

    %find where the two curves intersect on the x-axis (min and max
bins)
    binh1 = ceil(h1.BinEdges);
    binh2 = ceil(h2.BinEdges);
```

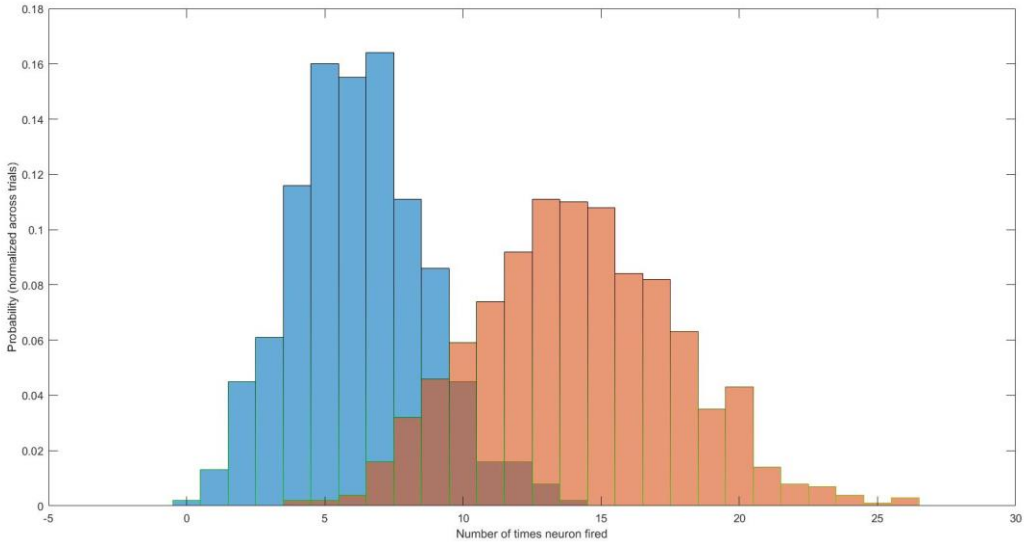
```
    binh2 = binh2(1:end-1);
    valh1 = h1.Values;
    valh2 = h2.Values;
    %we know that bin1 < bin2 therefore min(bin2) and max(bin1)
encloses the
    %region at which they intersect
    intersect_min = min(binh2);
    intersect_max = max(binh1);
    error1 = 0;
    error2 = 0;

    for i=1:(numel(binh1)-1)
        k = find(binh2 == binh1(i));
        if(numel(k)>0)
            if(valh1(i) < valh2(k))
                error1 = error1 + valh1(i);
            else
                error2 = error2 + valh2(k);
            end
        end
        k = [];
    end
    error = (error1+error2)/2;
    disp(error);
end

disp(x2);
```

It is very similar my previous code, except that the entire chunk is put into a while loop. I instantiate the error as 1 so that I can enter the loop, and set it to 0 once I enter. Note: some of the code in here may be extraneous (such as displaying the total average firing sum) and repetitive. I continue looping and incrementing x2 (as '50 + x') and end the loop as soon as the probability drops below .1, at which point I finally display the x2. The result of x2 is consistently 69. Now, if the code is modified so that the value of x1 decrements with respect to 50, the result is consistently 37 or 38. So, the answer to the question is that the stimulus needs to be 13 less or 19 more than 50 to achieve an error probability of less than 10%.

Comparison between '50' and '69':



Comparison between ‘37’ and ‘50’:



Given these results, one can visually tell that the area of intersection, which indicates the error probability, is much less in these results than the higher (approximately 20%) error probability between inputs ‘50’ and ‘60’.

MODIFYING TUNINGCURVE.M

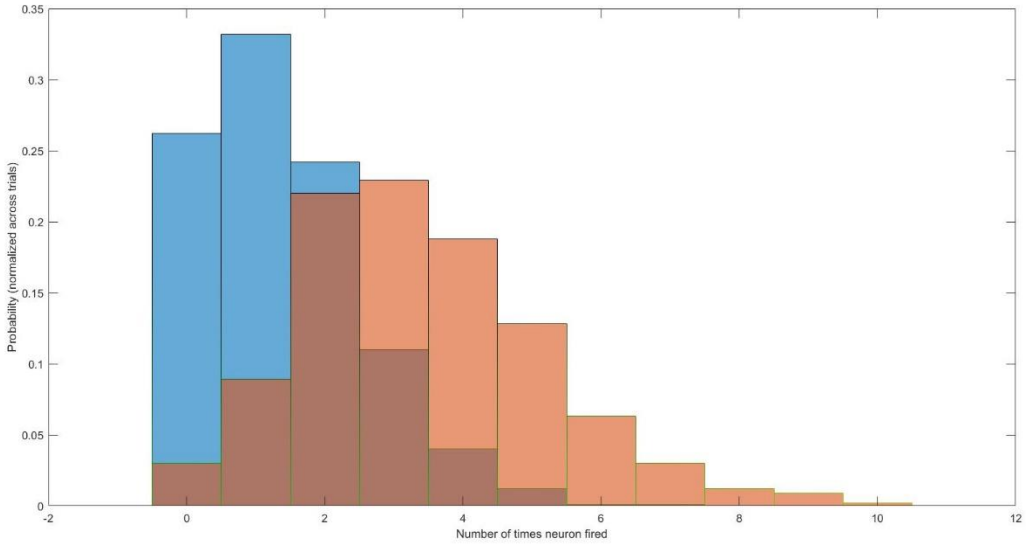
I know that tuningCurve.m is utilized in generateNoisyData.m in the statement:

```
rate = maxrate*tuningCurve(x1);
```

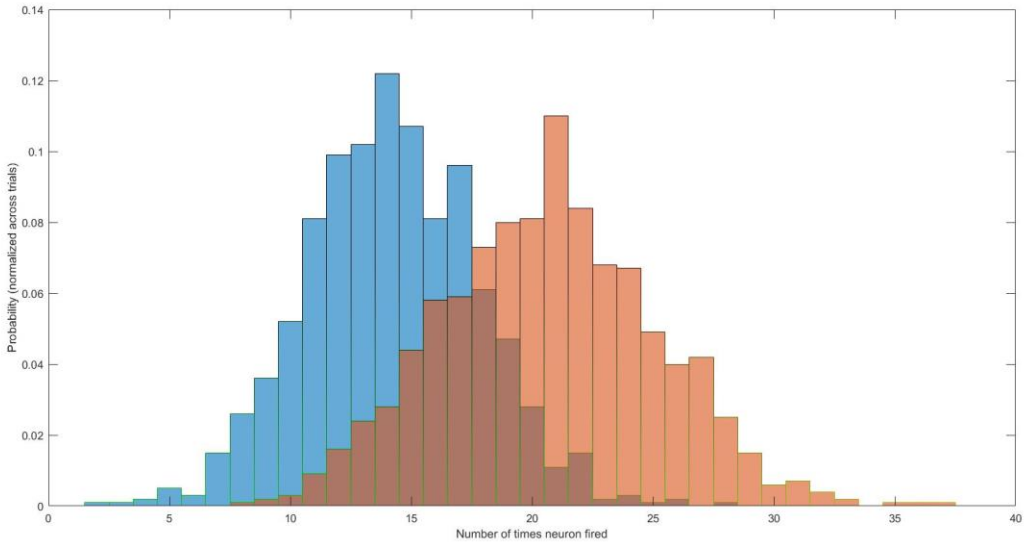
It is known that x1 is the input. The original tuningCurve code is here:

```
f = 1./(1 + exp(-(x-50)/10));
```

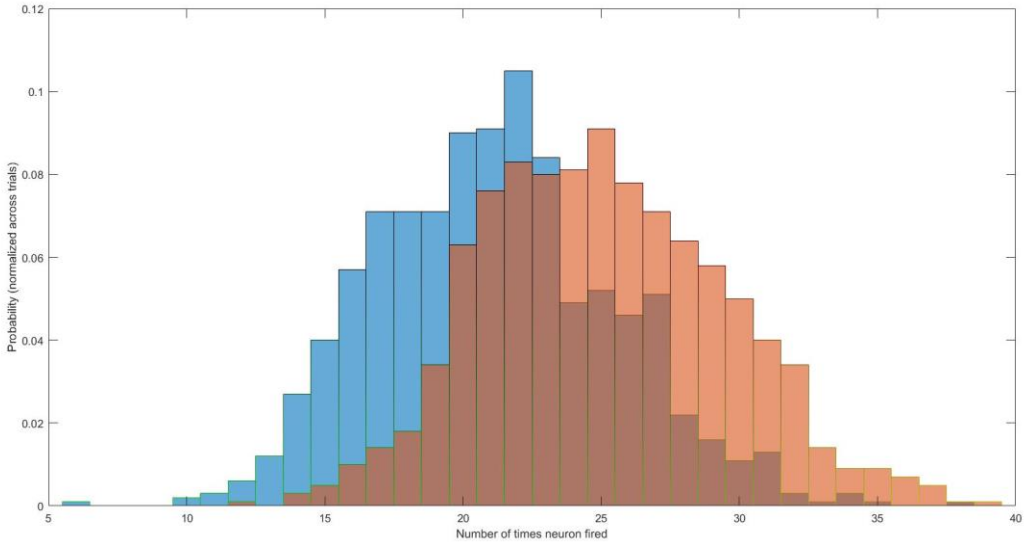
Here is what the input values of 20 and 30 look like originally:



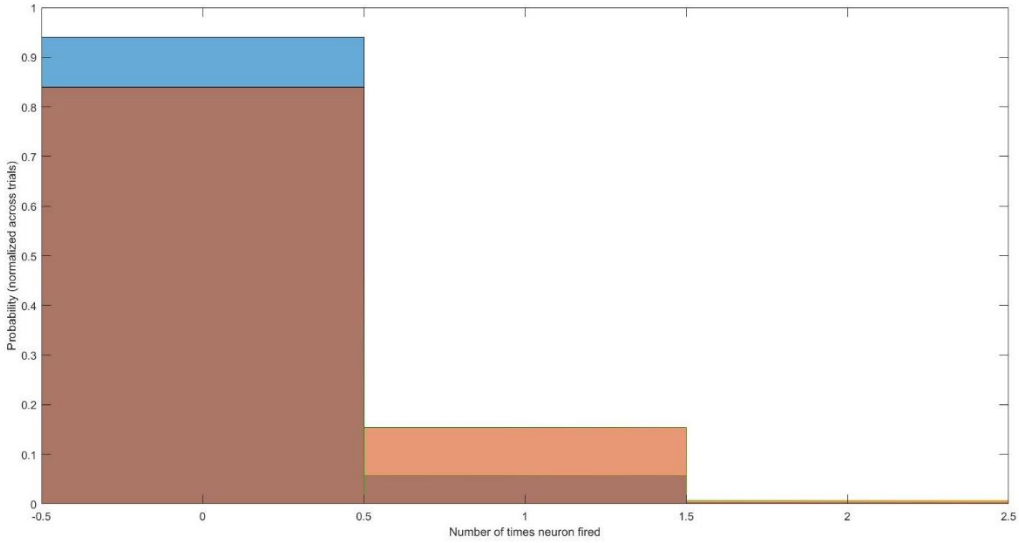
This comes with Error 1: 0.164 and Error 2: 0.339. If I modify the x-() value in tuningCurve so that it is 20 instead of 50, this is what I get:



This comes with Error 1: 0.171 and Error 2: 0.244. One thing that is clear is that the number of times the neuron fires is more, hence why the timebins are more widely spread. In any case, if we wish to achieve a more discriminable response, we must reduce the average error probability. If I reduce the value more, to 10, this is the graph I get:



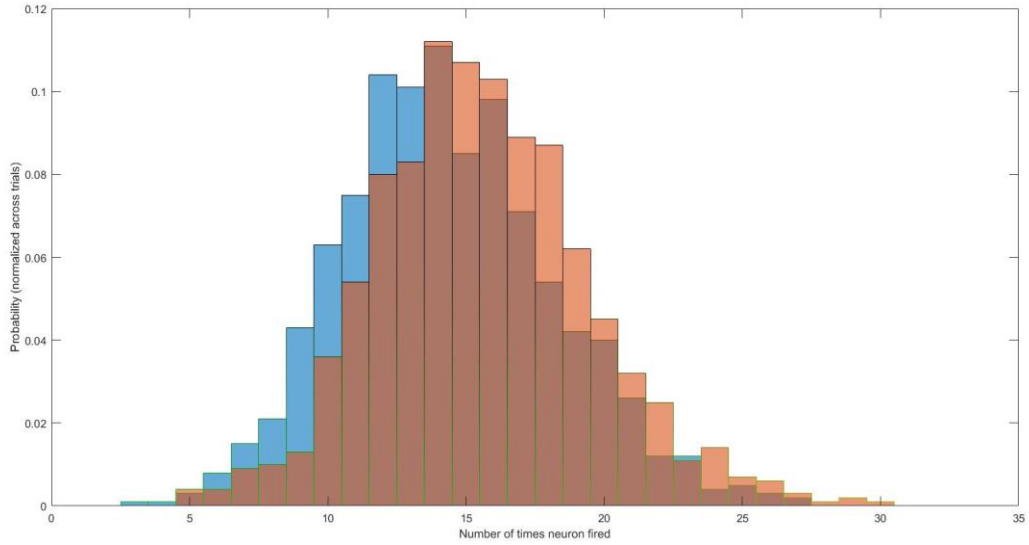
This comes with Error 1: 0.268 and Error 2: 0.388. Obviously, reducing this value far below 20 increases the overlap and is the opposite of our objective. If I increase the value to 80, I get:



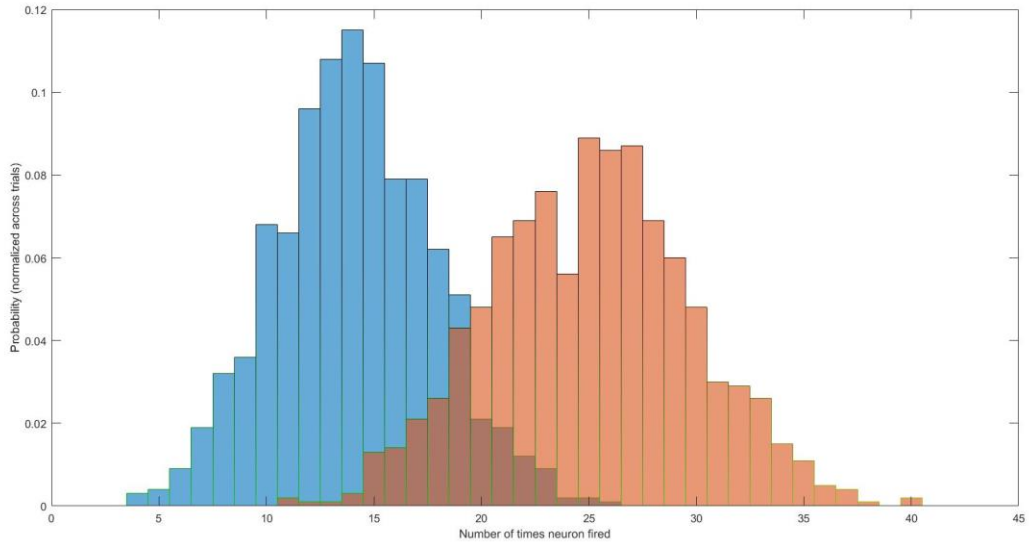
This comes with Error 1: 0.06 and Error 2: 0.839. It is apparent that decreasing or increasing that value far from 20 results in large overlap and increased error probability, with either higher or lower number of neurons fired respectively. Changing the value back to twenty, this is what my current tuningCurve looks like:

```
f = 1./(1 + exp(-(x-20)/10));
```

I shall modify the /10 to something such as /50, I get a graph such as this:



This comes with Error 1: 0.556 and Error 2: 0.3. Obviously, this tremendously increases the overlap and is the exact opposite of our objective. If I change this value to something such as /5, I get a graph that looks like this:



This comes with Error 1: 0.066 and Error 2: 0.124. These error probabilities are the smallest we've gotten by modifying such values.

```
f = 1./(1 + exp(-(x-20)/10));
```

From a functional perspective, setting the value highlighted in green closer to the given input and minimizing the value highlighted in cyan results in a more discriminable response. My hypothesis as to why this happens is that the green value shifts the number of times the neuron is

fired with respect to a relative or average value. If the average value is the given input, then it makes sense that there isn't too many or too little neurons fired to have more "normal" data. It's clear that the cyan value affects the "tightness" of the curve, therefore the smaller the value, the closer and more uniform the data is (creating less overlap and increasing discriminability).

2. White Noise Experiment

Finding the Average Response Time

I began with mapping out how I visualized the stim input (sketch below):

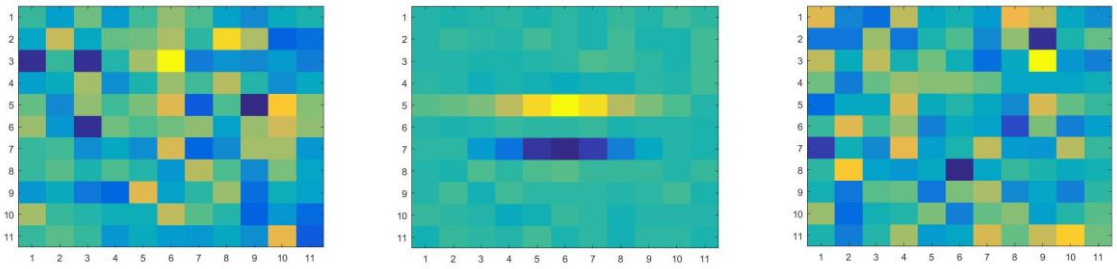
The currentFrames represents the "chunk" of frames .5 seconds (or 30 frames) before a spike occurred. These chunks are collected into the matrix called spike_triggered_stim, which is then averaged to produce the image. Here is the code:

```
clear all;
exp_sec = input('How many seconds to run the white noise
experiment?');
[ stim, spikeTrain] = generate_v1_white_noise_exp(exp_sec);
frame_rate = 60;
time_between_samples = 1/frame_rate;

%get the times at which a spike ocurred
indices = 0;
number_of_spikes = 0;
spike_triggered_stim = zeros(11,11,30,number_of_spikes);

for n = 1:60000
    if (spikeTrain(n) == 1)
        indices = [indices n];
```

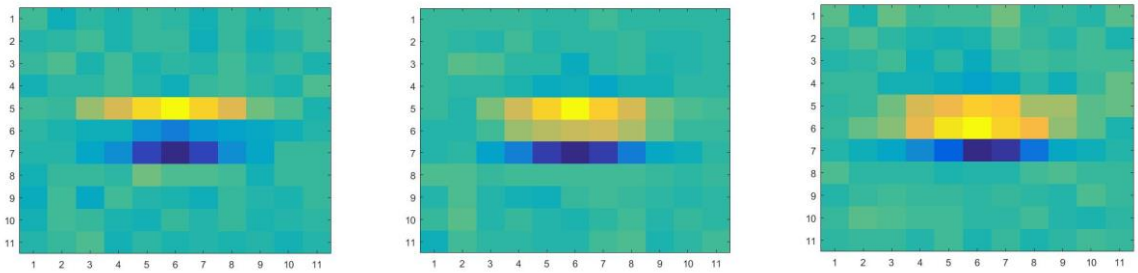
```
        number_of_spikes = number_of_spikes + 1;
        currentFrames = stim(:,:,n-29:n); % a 11x11x30 set of frames
        spike_triggered_stim(:,:,:,n) = currentFrames;
    end
end
test_frame = input('What frame before do you wanna check out?');
sta = mean(spike_triggered_stim,4);
imagesc(sta(:,:,test_frame));
```

After several attempts at trying to find the value to get a clear image (running at 1000 seconds)…

These are from frames 10, 20, and 29 before the spike respectively:



It appears that approximately 20 has the clearer image. Here is what frames 19, 21, and 22 before the spike look like respectively:

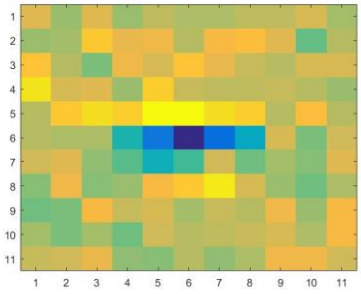


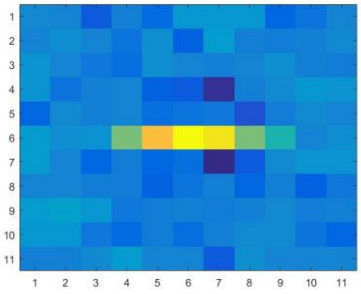
It appears that 21 frames before, or .35 seconds is the delay that resolves the clearest image.

ANALYZING RESULTS

This average image suggests that the neuron would prefer seeing objects present in the center of the receptive field. Anything in the environment that indicates a "central position," for example a candle in the middle of a dark room, or a solid black rectangle on a white sheet, or anything that involves a similar degree of a visually-distinct concentrated center could cause the neuron to fire.

At around 15-16 frames before, a "center" begins to form, which then slowly reaches the figure in frames 20-21 before. Soon afterwards, the image begins to degrade, and the center begins to disappear after 25 frames before. Images of 16 and 25 frames before are shown below respectively.





Over time, the neuron may respond to a "center-forming' image – which at around .35 seconds before the spike is fully and clearly formed/ perceived. This center image is the average preferred stimuli. For some more complex neurons – an average such as this could be misleading. An example of such a neuron would be that which recognizes the opposite of a "center-forming" image – perhaps there is a neuron which prefers responding to a checkerboard pattern (which could be very hard to average in such a grid). Another example could be a neuron which prefers responding to a cross shape in any orientation (x or t shape). Another example could be a neuron which prefers to track, over time, movements which appear to create wavelike movements (I'm not sure how a spike triggered average could ever detect something like that). However, complexity is the easy answer. With respect to invariance, a neuron could be responding to an element of the experiment which does not change – such as the presence of the monitor displaying the image. Also, the topic of neural adaptation introduced in the previous homework is very pertinent in this scenario. Multiple trials and previous images of static may "desensitize" a neuron to continuous, seemingly unchanging ("static") input. Invariance, with respect to neural adaptation, can certainly affect the whole experiment, thereby causing the average preferred stimuli to be a misleading piece of analysis.