Neuro 120 Homework 2: Data Analysis

William Schmitt and Will Drew

Due: Thursday 18 October 2018

1 Question 1: Auditory Neuroplasticity

1.1 Raster Plot of Single-Unit Activity

To plot the raster plot, we wrote the following code:

```
stimulus_start_times = 0:1/6:(60); % In seconds
%% Part A
% Make raster plot
figure(1);
hold on
for i = 1:(length(stimulus_start_times)-1)
    spikes_in_window = spikes_single_unit((spikes_single_unit > ...
        stimulus_start_times(i)) & (spikes_single_unit < ...
        stimulus_start_times(i + 1)));
    spikes_normalized = (spikes_in_window - stimulus_start_times(i))';
    trial_num = ones(1, length(spikes_normalized))*i;
    plot([spikes_normalized; spikes_normalized], [trial_num; trial_num-1],'k',
        'LineWidth',3)
end
xlim([0,0.167])
ylim([0,360])
yticks(0:30:360)
xlabel('Time (s)');
ylabel('Trial Number');
title('Response of a Single-Unit to the Exposure Stimulus');
```

This produces the raster plot shown in Figure 1, which clearly shows that the neuron responds consistently to the stimulus about 0.03 seconds into the start of the trial.

1.2 Gaussian Kernel Firing Rate Estimate

We calculate the estimate of the firing rate of this single-unit neuron by writing the following code:

```
%% Part B
% Create gaussian filter
figure(2);
```

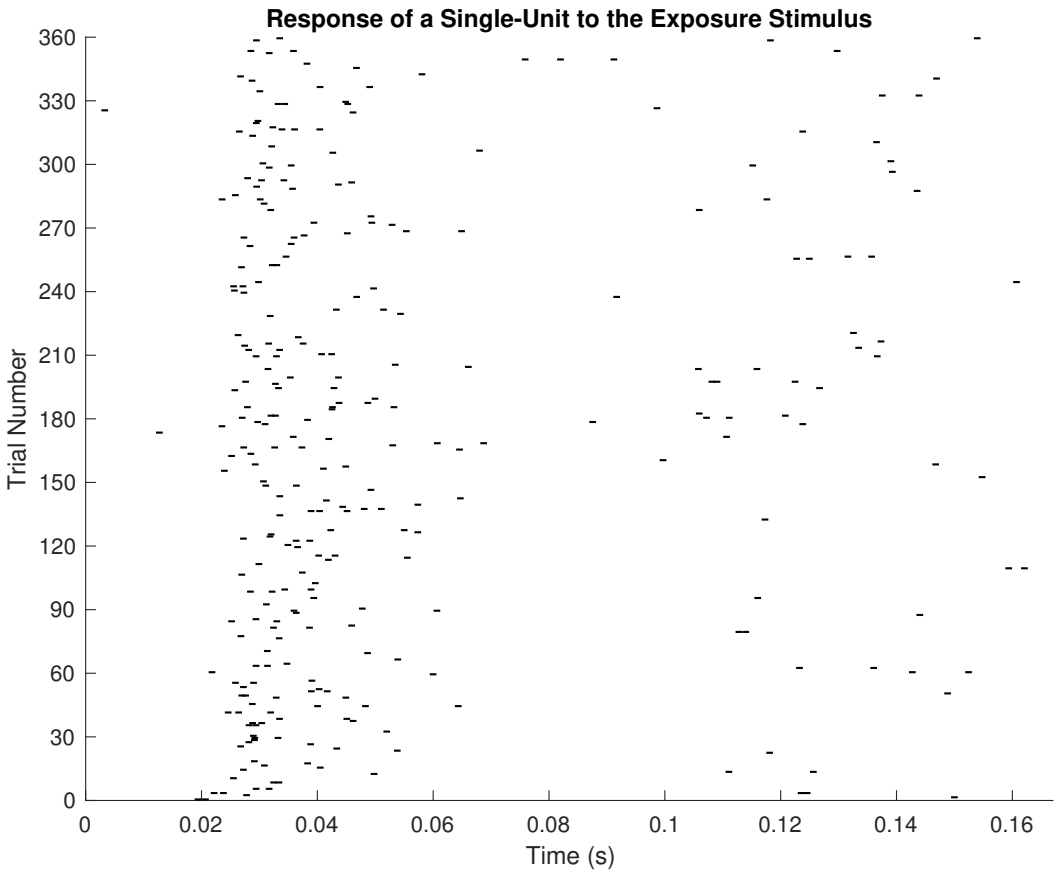

Figure 1: Raster Plot of Single-Unit Activity.

```
x=[0:0.0001:1/6];
avg_dist = zeros(1,length(x));
for i = 1:(length(stimulus_start_times)-1)
    spikes_in_window = spikes_single_unit((spikes_single_unit > ...
        stimulus_start_times(i)) & (spikes_single_unit < ...
        stimulus_start_times(i + 1)));
    spikes_normalized = (spikes_in_window - stimulus_start_times(i))';
    trial_num = ones(1, length(spikes_normalized))*i;
    norm = zeros(1,length(x));
    for j=1:length(spikes_normalized)

        norm = norm + normpdf(x,spikes_normalized(j),0.005);
    end
    avg_dist = avg_dist+norm;
end
plot(x,avg_dist./360)
xlabel('Time (s)');
ylabel('Firing Rate (Hz)');
```

This code simply places a Gaussian distribution with a standard deviation of 0.005s centered at the

location of each spike and averages these distributions over all stimulus trials. This work produces Figure 2, which shows that the firing rate increases dramatically around 0.03s into the trial, going from a baseline firing rate of about 5 Hz to a peak of just over 25 Hz.

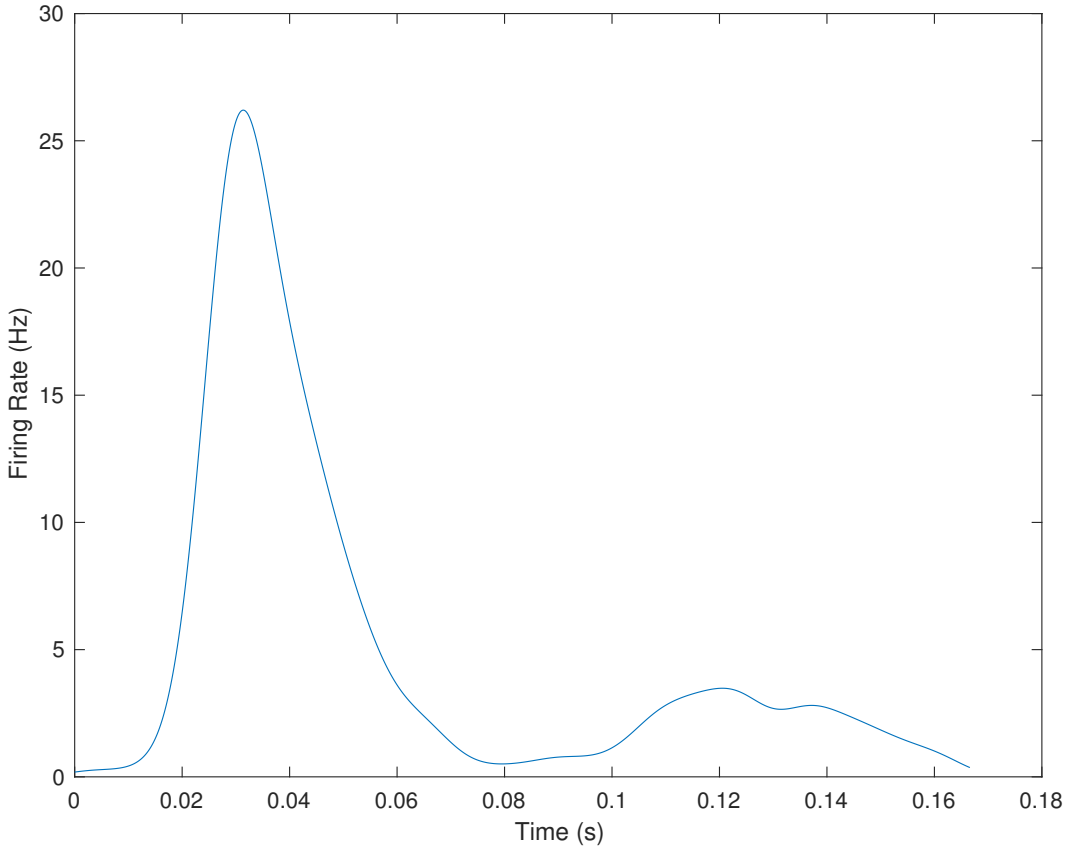

Figure 2: Gaussian Estimation of Firing Rate Using $\sigma = 0.005s$.

1.3 Gaussian Kernel Parameter Variation

We next alter the σ parameter of the Gaussian kernel from above and generate Figures 3 and 4 which shows the effect of this change. We used the following code to make these figures.

```
%% Part C
% sigma = 50ms, 0.5ms
sigma = [0.05,0.0005]
for k = 1:2
    figure(k+2);
    x=[0:0.0001:1/6];
    avg_dist = zeros(1,length(x));
    for i = 1:(length(stimulus_start_times)-1)
        spikes_in_window = spikes_single_unit((spikes_single_unit > ...
            stimulus_start_times(i)) & (spikes_single_unit < ...
            stimulus_start_times(i + 1)));
```

```
        spikes_normalized = (spikes_in_window - stimulus_start_times(i))';
        trial_num = ones(1, length(spikes_normalized))*i;
        norm = zeros(1,length(x));
        for j=1:length(spikes_normalized)
            norm = norm + normpdf(x,spikes_normalized(j),sigma(k));
        end
        avg_dist = avg_dist+norm;
    end
    plot(x,avg_dist./360)
    xlabel('Time (s)');
    ylabel('Firing Rate (Hz)');
end
```

We can see from these figures that a very small standard deviation results in a quite noisy curve with large, quick oscillations, which intuitively, does not seem to describe the behavior of the neuron well as it is unlikely the firing rate would change that quickly. Conversely, a very large standard deviation results in a very smooth, wide curve that also does not seem to describe the data well as it predicts a large firing rate at the start of the experiment, where the raster plot indicates there is very little activity.

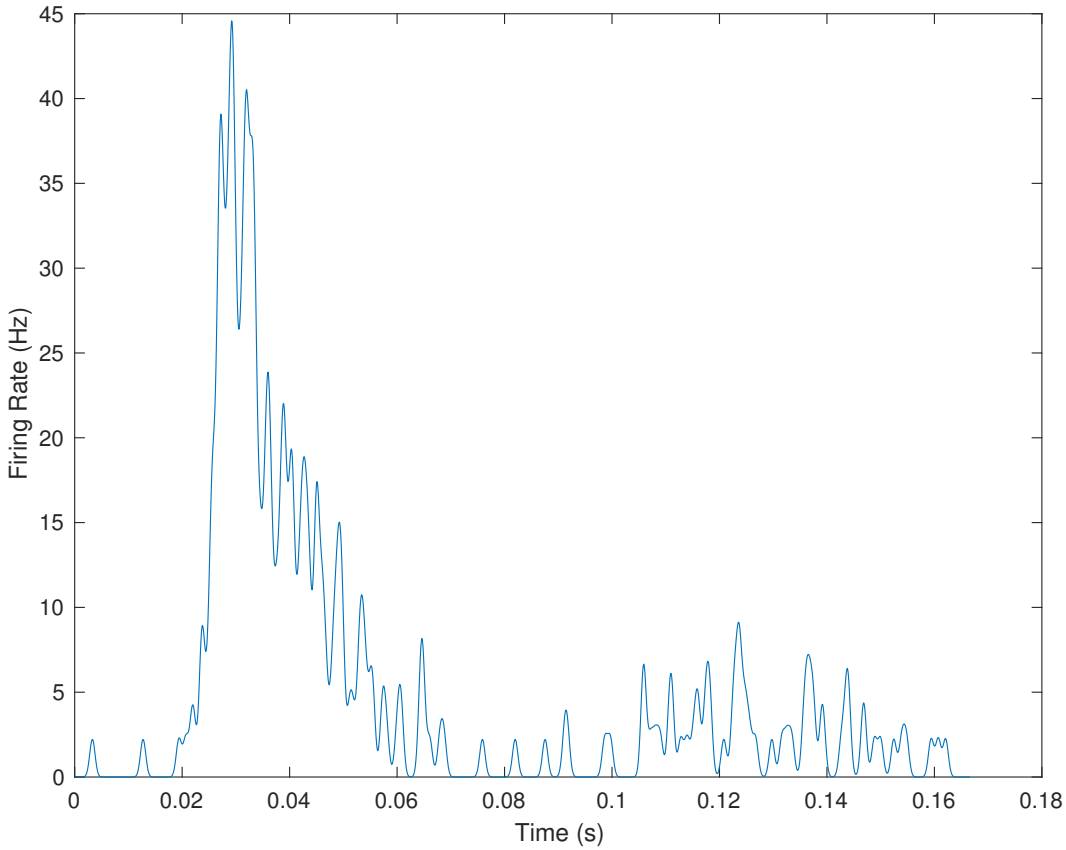

Figure 3: Gaussian Estimation of Firing Rate Using $\sigma = 0.0005s$.

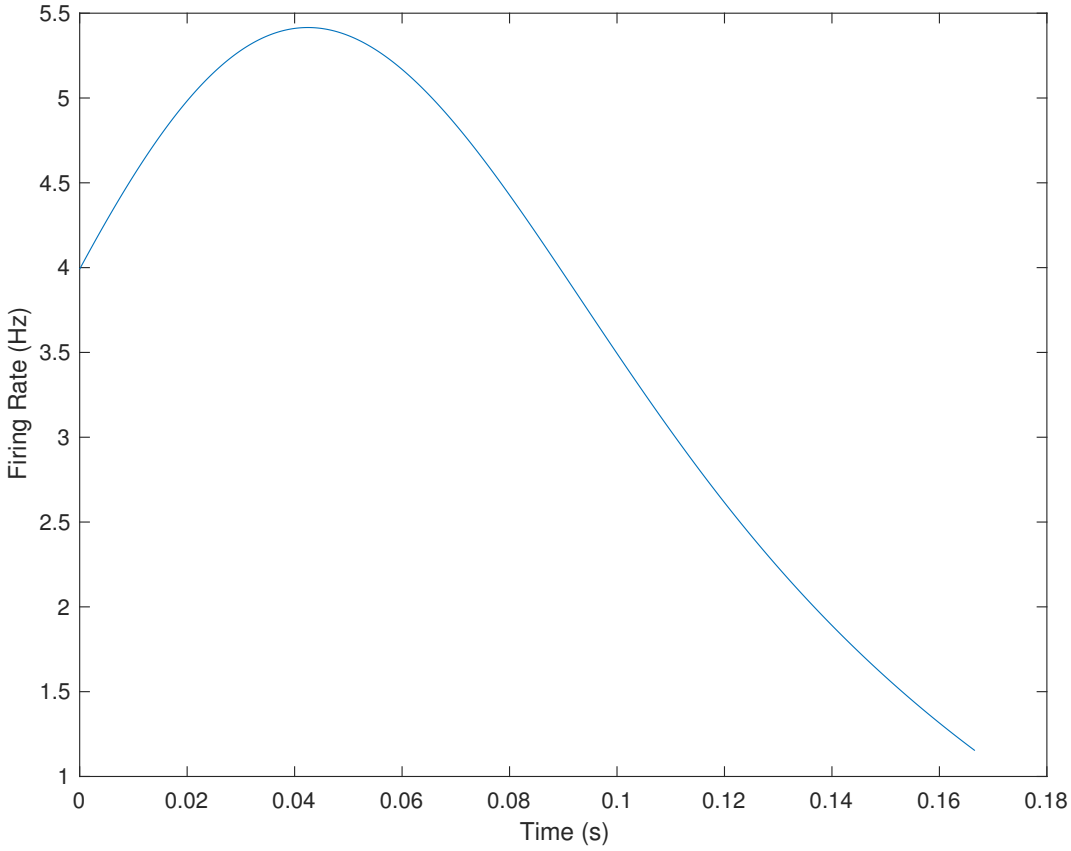

Figure 4: Gaussian Estimation of Firing Rate Using $\sigma = 0.05s$.

1.4 Grand Average Post-Stimulus Time Histogram

We compute the grand average poststimulus time histogram using the following code:

```
%% Part D
figure(5);
hold on;
nbins = length(0:0.005:(1/6))
neural_data = {spikes_control, spikes_exp};
total_counts = {zeros(1, nbins), zeros(1, nbins)};
num_neurons = {N_control, N_exp};
for condition = 1:2
    for i = 1:(length(stimulus_start_times)-1)
        spikes_in_window = neural_data{condition}((neural_data{condition} > ...
            stimulus_start_times(i)) & (neural_data{condition} < ...
            stimulus_start_times(i + 1)));
        spikes_normalized = (spikes_in_window - stimulus_start_times(i))';
        total_counts{condition} = total_counts{condition} + hist(spikes_normalized,
            nbins);
    end
    total_counts{condition} =
```

```
        (total_counts{condition}/(360*num_neurons{condition}*.005));
end
bar([total_counts{2}', total_counts{1}'])
```

This code generates Figure 5, which shows us the same trend we saw in the previous figures. The stimulus evokes a larger response from the population of neurons about 0.03 seconds after the onset of a trial.

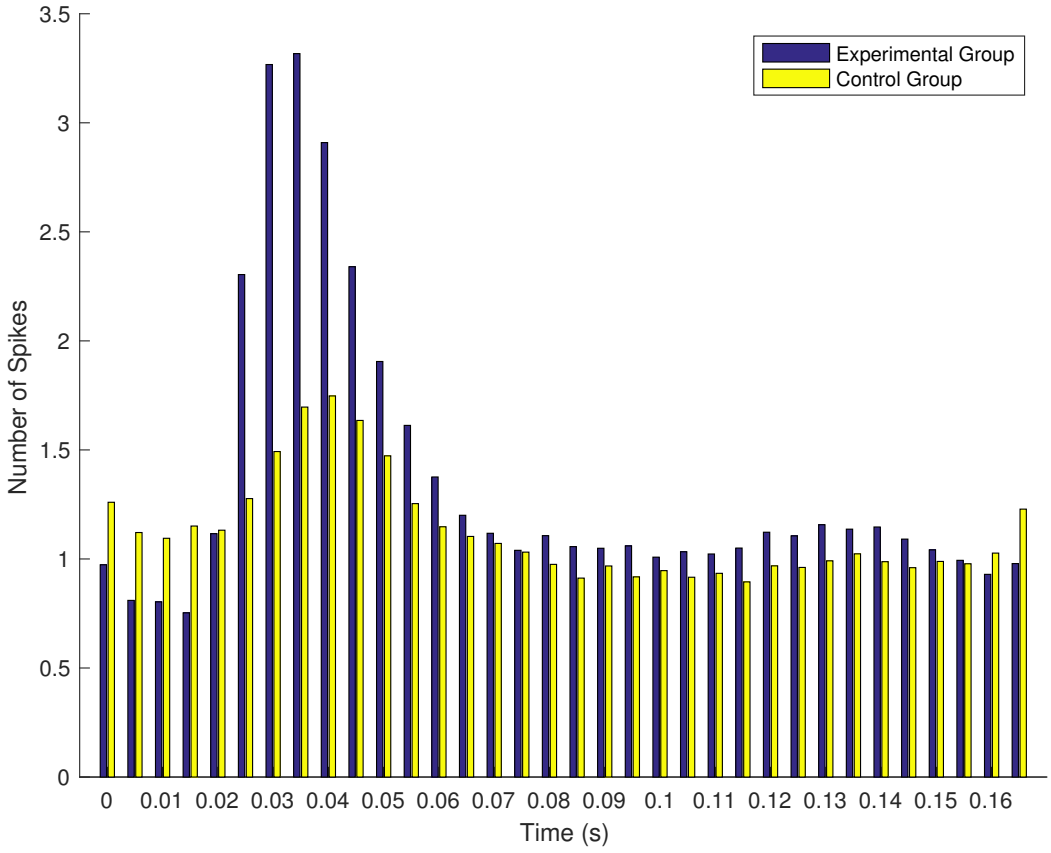

Figure 5: Grand Average Post-Stimulus Time Histogram of two Populations of Neurons. Bin width was 5ms.

1.5 Differences between Experimental and Control Groups

Looking at Figure 5, we see that the experimental group has a stronger response (i.e. more spikes) during the peak firing activity (about 0.02s to 0.05s after the onset of the trial) of the neurons. However, the control neuron population still exhibits this increase, so therefore it is just that the experimental group of neurons became more selective to the exposure stimulus while the precision of the response did not appear to change. Finally, we can see that there is very little difference in the activity of the two groups (beyond some higher activity in the control group at the start and end of a trial) outside of this peak firing range.

1.6 Spike-Triggered Average for Neuron

1.7 Frequency Selectivity of Neuron

1.8 Neuronal Stimulus Preference

1.9 Potential Stimulus Correlations

2 Random Neural Networks and Overfitting in Regression

2.1 Linear Regression with Regularization

We implement linear regression with regularization in Matlab as follows:

```
%% Part A
clear all
Nh = [2, 10, 26];
text_height = [0.7, 0.8, 0.9];


for i = 1:3

    % Set up parameters
    N = 40; % Number of training samples
    epsilon = 0.0; % Amount of label noise
    lambda = 0;

    % Make dataset
    target_fn = @(t) sin(t);
    x = linspace(-pi,pi,N);
    y = target_fn(x) + epsilon*randn(size(x));

    Ntest = 100;
    x_test = linspace(-pi,pi,Ntest);
    y_test = target_fn(x_test);

    Ni = 2;

    % Compute network activity

    J = randn(Nh(i),Ni)/Nh(i);

    h = J*[x; ones(1,N)];
    h(h<0)=0;

    h_test = J*[x_test; ones(1,Ntest)];
    h_test(h_test<0)=0;


    % Now train linear regression to map from h to y
```

```
    w = y*h'*pinv(h*h' + lambda*eye(size(h, 1)));

    y_pred = w*h_test;

    mean_squared_error = norm(y_test-y_pred).^2;



    plot(x_test,y_pred)
    hold on
    set(gca, 'YLim', [-1.5 1.5])

    text(-pi,[.1 text_height(i)]*get(gca,'YLim')',sprintf('MSE for %d Neurons: %g ',
        Nh(i), mean_squared_error))
    xlabel('Input')
    ylabel('Output')
end

plot(x,y,'ob')
hold on
plot(x_test,y_test)
legend(sprintf('Prediction %d', Nh(1)), sprintf('Prediction %d', Nh(2)), ...
    sprintf('Prediction %d', Nh(3)), 'Training data','Test data')
```

This produces Figure 6 which shows us how this network performs on test data. We see from this data that the 26 neuron model performed the best (as measured by mean squared error and visual inspection of the curve it makes), while the 10 neuron model performed reasonably well, but with some more error. Finally, the 2 neuron model performed the worst of all and has quite the large error.

2.2 Overdetermined Networks

We implement the overdetermined network scenario as follows:

```
%% Part B
clear all
figure(2);
Nh = 1:40;
mse = zeros(40, 1000);
for i = 1:length(Nh)
    for j = 1:1000
        % Set up parameters
        N = 40; % Number of training samples
        epsilon = 0.0; % Amount of label noise
        lambda = 0;

        % Make dataset
        target_fn = @(t) sin(t);
        x = linspace(-pi,pi,N);
        y = target_fn(x) + epsilon*randn(size(x));

        Ntest = 100;
```

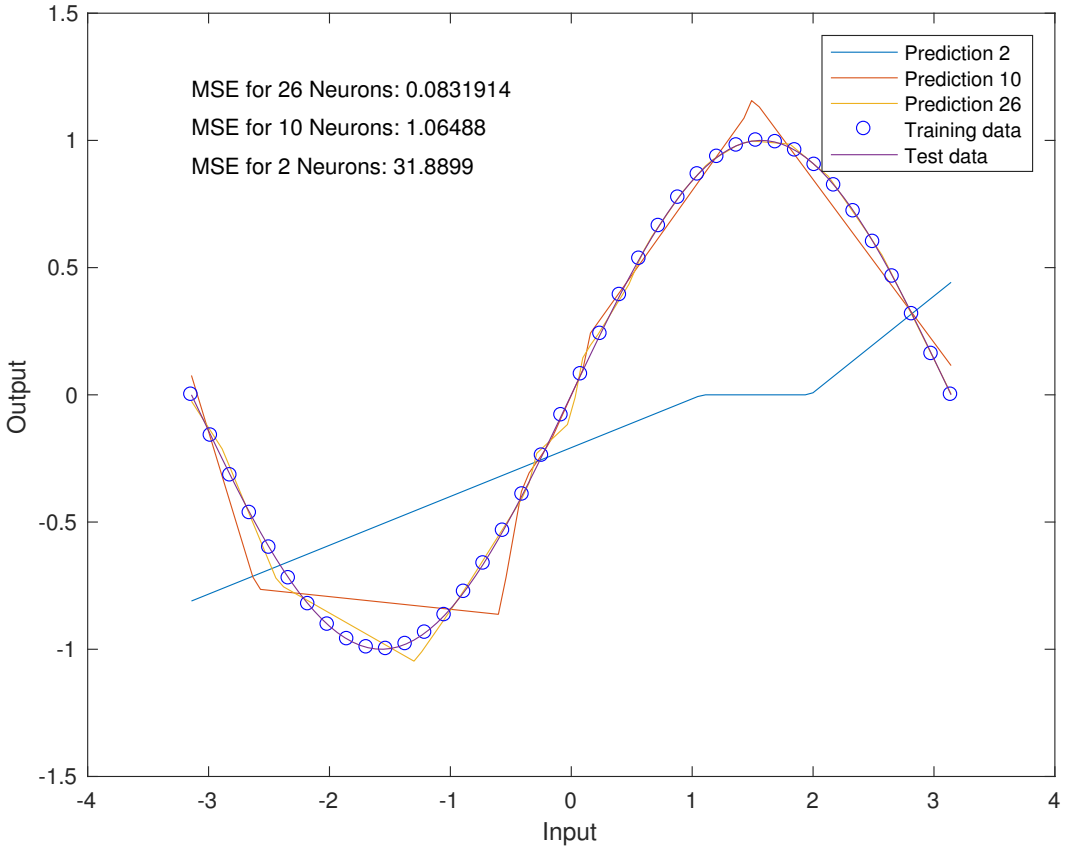

Figure 6: Performance of Simple Linear Regression with varying numbers of neurons

```
x_test = linspace(-pi,pi,Ntest);
y_test = target_fn(x_test);

Ni = 2;

% Compute network activity

J = randn(Nh(i),Ni)/Nh(i);

h = J*[x; ones(1,N)];
h(h<0)=0;

h_test = J*[x_test; ones(1,Ntest)];
h_test(h_test<0)=0;


% Now train linear regression to map from h to y

w = y*h'*pinv(h*h' + lambda*eye(size(h, 1)));

y_pred = w*h_test;
```

```
        mse(i, j) = norm(y_test-y_pred).^2;
    end

end

plot(1:40, mean(mse,2))
xlabel('Number of Neurons in Network');
ylabel('Mean Squared Error');
```

This code produces Figure 7, which shows us what happens in the overdetermined network scenario. We see from this figure that the network is quite unstable throughout this size range. Indeed, the mean squared error is quite low for networks with less than 10 neurons, 15 neurons, or 25 neurons. However, the mean squared error is astronomically high for values close to this: networks with 12, 21, and 22 neurons. Thus, we see that when the model is too small, the mean squared error might be low, but the predictions don't map neatly onto the "ideal" function we are approximating (as seen in Figure 6). However, when the model is too big, the mean squared error is quite large and the model likely fits the data too well such that it cannot generalize to new test data.

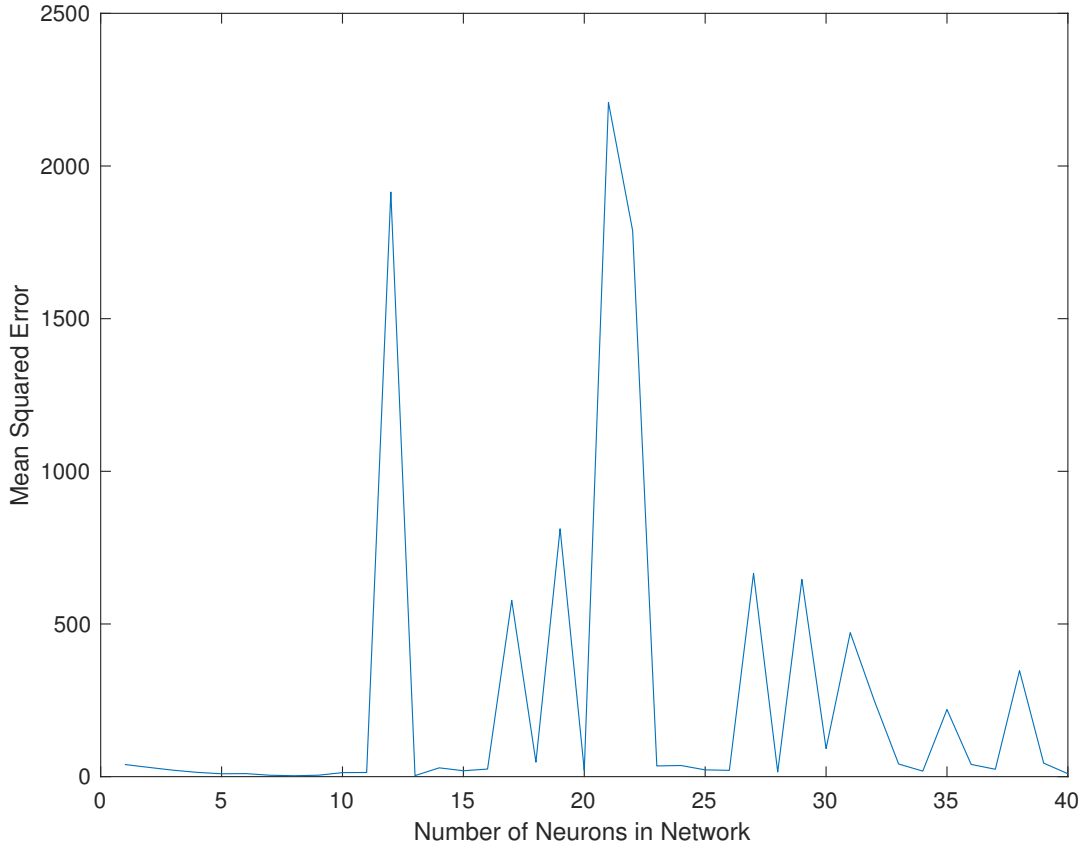

Figure 7: Mean-Squared Error as a Function of Network Size for the Over-Determined Situation.

2.3 Underdetermined Networks

We implement the underdetermined network scenario as follows:

```
%% Part C
clear all
figure(3);
Nh = 40:20:500;
mse = zeros(length(Nh), 1000);
for i = 1:length(Nh)
    for j = 1:1000
        % Set up parameters
        N = 40; % Number of training samples
        epsilon = 0.0; % Amount of label noise
        lambda = 0;

        % Make dataset
        target_fn = @(t) sin(t);
        x = linspace(-pi,pi,N);
        y = target_fn(x) + epsilon*randn(size(x));

        Ntest = 100;
        x_test = linspace(-pi,pi,Ntest);
        y_test = target_fn(x_test);

        Ni = 2;

        % Compute network activity

        J = randn(Nh(i),Ni)/Nh(i);

        h = J*[x; ones(1,N)];
        h(h<0)=0;

        h_test = J*[x_test; ones(1,Ntest)];
        h_test(h_test<0)=0;


        % Now train linear regression to map from h to y

        w = y*h'*pinv(h*h' + lambda*eye(size(h, 1)));

        y_pred = w*h_test;

        mse(i, j) = norm(y_test-y_pred).^2;
    end

end

plot(Nh, mean(mse,2))
xlabel('Number of Neurons in Network');
ylabel('Mean Squared Error');
```

This code produces Figure 8, which we see produces a low mean-squared error for large network sizes. Thus, based on this figure, we would guess that the best network size is approximately 110 neurons or so (as the mean squared error seems to plateau after this). The performance of these big networks is not as variable as the smaller networks.

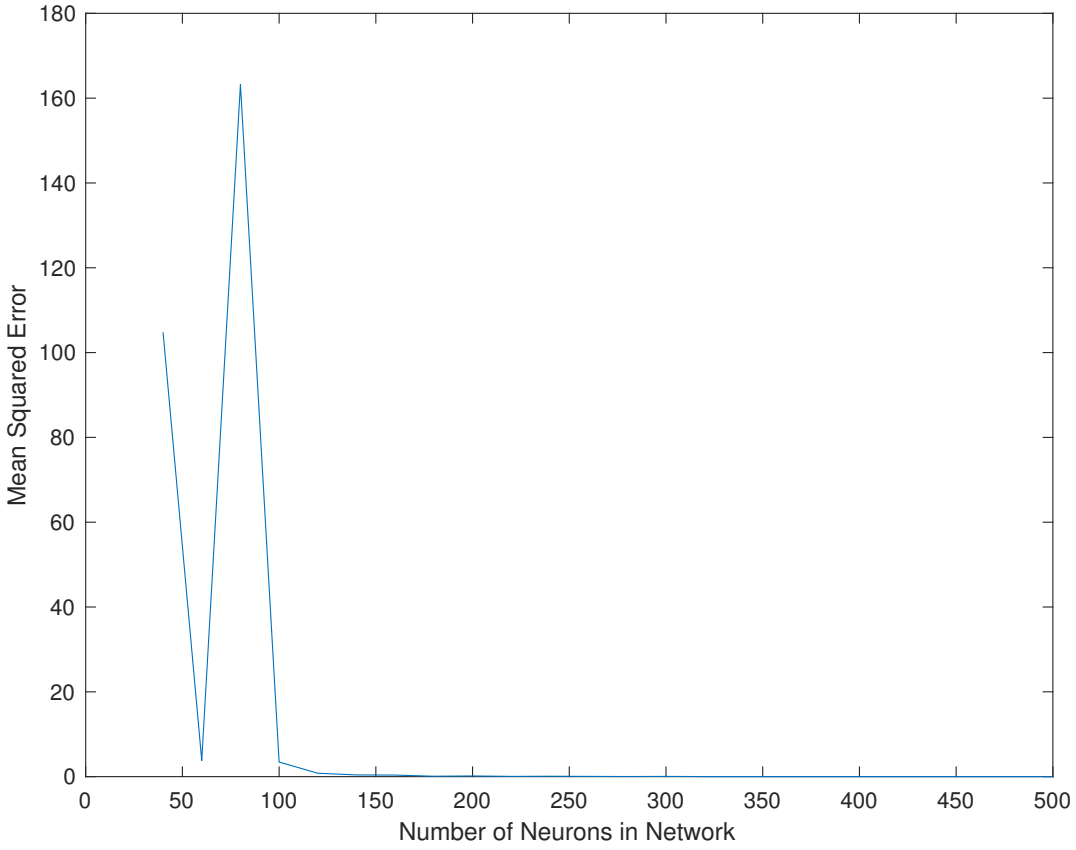

Figure 8: Mean-Squared Error as a Function of Network Size for the Under-Determined Situation.