Exercises for Chapter 4, Optimality

The theme of Chapter 4 is that many information processing functions will be optimized in neural networks when they operate near the critical point. Thus, functions like the susceptibility, the dynamic range, and the mutual information should show a peak near the critical point. Optimality and universality are the two main consequences of being nearly critical.

Recall that the exercises in Chapter 1 were intuitive; now they will be more quantitative. We will be examining each of these functions and looking for peaks. More specifically, we will look at how the locations of those peaks move as the simulations are made larger or run longer. We will be interested in extrapolating to estimate the behavior of an infinitely large network, in what is called the "thermodynamic limit." While doing a very thorough job of this would often take a supercomputer, it is possible to get glimmers of the thermodynamic limit even with a laptop. In doing so, we will be exploring the finite size effects discussed in pages 79-80 of the book.

1. **Susceptibility**
   The sensitivity of a network to changes in input is roughly captured by the susceptibility, as explained in the book on pages 80-81. The prediction of the criticality hypothesis is that living neural networks will have a narrow peak in susceptibility near the critical point. This would optimize their ability to detect slight changes in inputs.

   First, let's look at the variability of a network's output to a constant input. We will use a branching model network with a feed-forward architecture. This network will have only one active neuron in the first layer, and the activity from that neuron may propagate into subsequent layers. We will stimulate the network many times, say 500, and on each occasion, we will take the activity in the last layer of the network as the output. We can create a TIMERASTER by concatenating all the 500 outputs into one matrix. To do this, use the FeedForwardBranchingModelFunction, like this:

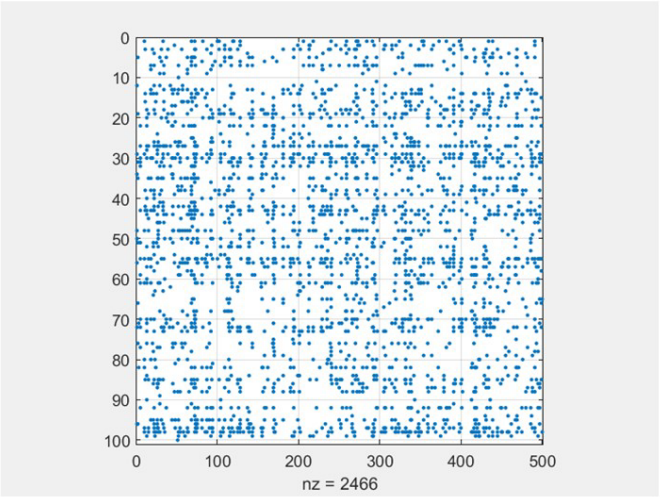
   ```
   [TIMERASTER] = FeedForwardBranchingModelFunction(neurons, numStim, layers,…
   connects, BR, timesteps);
   ```

   To start, try these parameters: "neurons" = 100, "numStim" = 1, "layers" = 11, "connects" = 10, "BR" = 1.2, "timesteps" = 500:

   ```
   [TIMERASTER] = FeedForwardBranchingModelFunction(100, 1, 11, 10, 1.2, 500);
   ```

   Below is a screenshot of a TIMERASTER produced by this function, which you can plot if you just type this:

   ```
   figure; spy2(TIMERASTER);
   ```

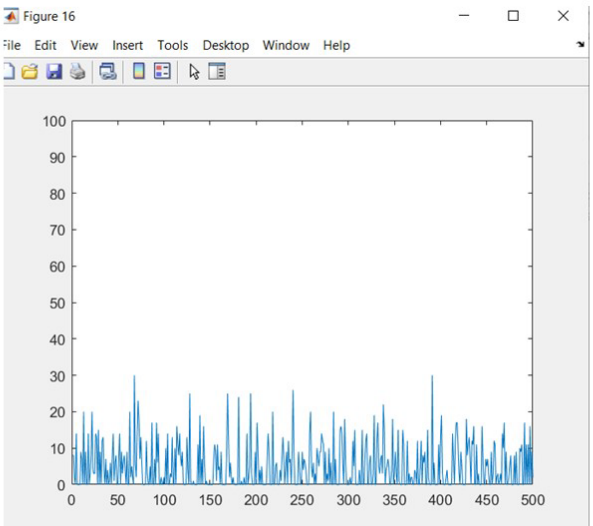


Output from a feed-forward branching network. The network had 100 neurons per layer and 11 layers. To initiate activity, only one neuron was activated in the first layer. If this then propagated to the last layer, the resulting activity was entered into one of the columns in this TIMERASTER. Here, the network was stimulated 500 times, once for each column. Note that sometimes no activity propagated, while other times many neurons became active.

It should be apparent that there is great variability in the output, even though the input was consistently just one neuron stimulated in the first layer. To get an idea of how the branching ratio impacts this variability, we can plot the sum of the columns in the TIMERASTER:

```
figure; plot(sum(TIMERASTER)); ylim([0 100]); ylabel("activity");
xlabel("time step");
```

When you do that, you should see something like this:



When the activity along the columns of the TIMERASTER are summed, they show variability like this. The variance of this signal is the susceptibility.

A. Exercise: Input branching ratios from 1 to 5, in increments of 1, to FeedForwardBranchingModelFunction. Then plot each of the TIMRASTERs by using this line of code:

```
figure; plot(sum(TIMERASTER)); ylim([0 100]);
```

By inspection, can you tell the relationship between the branching ratio and the variability? Is it linear? In other words, does the variability of the output always increase as the branching ratio is increased? To be more quantitative, you can calculate the variance of the output with this function:

```
[Chi] = SusceptibilityCalc(TIMERASTER)
```

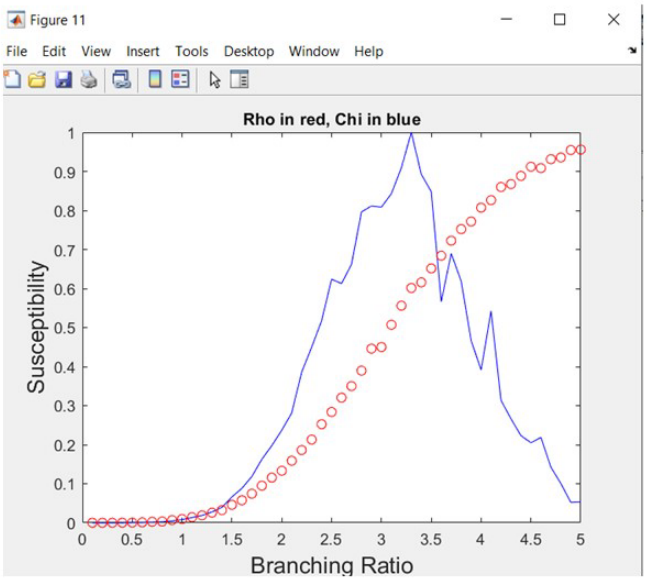
Here, the Greek letter Chi is the susceptibility. Now, to do this systematically we will use the SusceptibilityDemoFunction:

```
[Chi, Rho, BR] = SusceptibilityDemoFunction(neurons, layers, connects,
timesteps, numSamples);
```

To start, try these parameters: “neurons” = 100, “layers” = 5, “connects” = 10, “timesteps” = 500, “numSamples” = 50:

```
[Chi, Rho, BR] = SusceptibilityDemoFunction(100, 5, 10, 500, 50);
```

This should take less than 5 minutes to run and will give you a plot that looks something like this:



Sample plot of the output of SusceptibilityDemoFunction after simulating a feed-forward network with 100 neurons per layer and 5 layers. The blue curve is the normalized (max set to 1) susceptibility and the red curve is the density of active sites.

Recall that the Greek letter Rho represents the density of sites, or the fraction of neurons that are active. Here we see a peak in the susceptibility at a branching ratio of about 3.3.

B. Exercise: Gradually increase the number of layers in the network while holding other input parameters constant (e.g., try layers = 5 to 25 by increments of 5). Note the branching ratio at which the peak occurs for each network and then plot the peak branching ratio against the number of network layers. If possible, also record the width of the susceptibility curve at half its amplitude. In a separate graph, plot the width against the number of layers.

C. Exercise: You will probably realize that running very many layers takes longer. Instead of trying to run very large networks, try to extrapolate based on the data you have. Where would the susceptibility curve for a network with 1000 layers have its peak? What would be the width of its susceptibility curve at half amplitude?

2. Dynamic range

In addition to being sensitive to slight changes in inputs (susceptibility) neural networks also meaningfully encode a wide range of different input strengths. This is known as dynamic range, discussed in the book on pages 78-79. Our auditory system, for example, can process sound intensities that differ by ten orders of magnitude. We will again use a feed-forward branching model to explore how the dynamic range is affected by the branching ratio.

Now, instead of stimulating the network with only one neuron in the first layer, we will cycle through all possible stimulation values. For example, if a network has 100 neurons per layer, we will stimulate with values from 1 to 100. The activity will then propagate through the layers and the output response will merely be the number of neurons that are active in the last layer of the network. This will be done many times so we can get a distribution of response values for each input value. As we'll see, the dynamic range is a way of measuring the breadth of the response distribution.

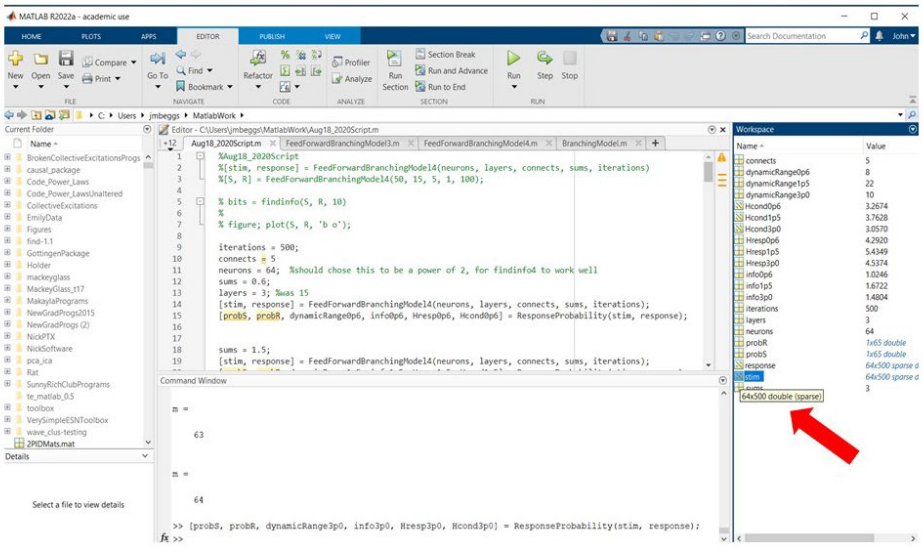
Let's start by just looking at the output of a feed-forward network after it has been given a wide range of inputs. Use the function FeedForwardBranchingModel4 as shown below:

```
[stim, response] = FeedForwardBranchingModel4(neurons, layers,…
connects, BR, iterations);
```

Here, pick "neurons" = 64, "layers" = 3, "connects" = 5, "BR" = 1.5, "iterations" = 500:

```
[stim,response] = FeedForwardBranchingModel4(64, 3, 5, 1.5, 500);
```

If you run it, it will produce the two output variables “stim” and “response” – do not delete these just yet, as we will use them in the next steps. In the Matlab Workspace window, you should be able to see the variable “stim.” Click on it to see the different stimulation values that were given. After that, click on “response” to see what it looks like. From this, you should be able to get an idea of the variety of responses. Still, it is not easy to read such a large array.

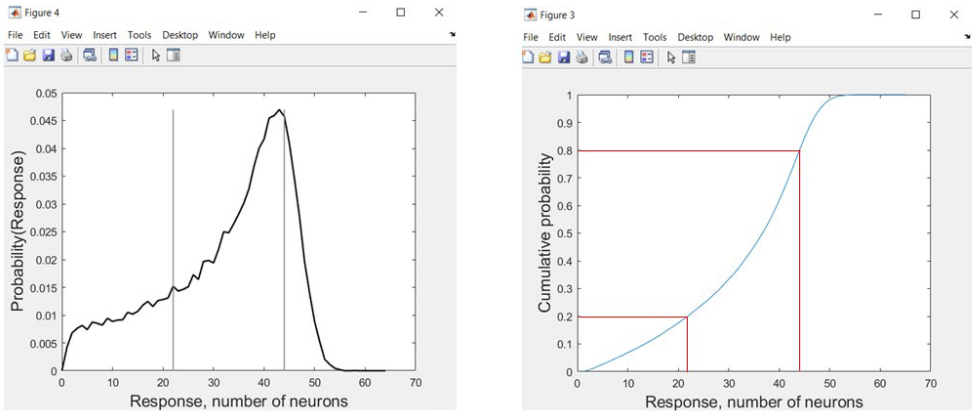


A screenshot of Matlab, showing the variable “stim” in the workspace. If you click on it, you can see all the values it contains. You can also just type stim in the command window.

Next, we will plot the distribution of response values so it can be visualized. Use the function ResponseProbability like this:

```
[probS, probR, dynamicRange, info, Hresp, Hcond] = …
ResponseProbability(stim, response);
```

There are many variables here, but for our purposes right now we will only be concerned with “probS,” “probR” and “dynamicRange.” These give the probability of the stimulus, the probability of the response and the dynamic range, respectively. If you take the output variables “stim” and “response” that were produced by your previous run of FeedForwardBranchingModel4, and feed them into ResponseProbability, you should then see a plot of the distribution of the response values. It should look something like the left plot in this figure below.



Left, A response distribution produced by the function ResponseProbability, after it was fed the output from FeedForwardNetworkModel4. Here, each layer had 64 neurons, there were 3 layers, each neuron had 5 connections, and the branching ratio was 1.5. The response was just the number of neurons active in the last layer, plotted along the x-axis. The probability of each response is plotted along the y-axis. The two vertical gray lines are drawn at the points where 20% and 80% of the cumulative area under the distribution occur. The distance between these lines is one way to measure the dynamic range. The cumulative probability of the response distribution is shown in the plot on the right; red lines are at 20% and 80% of cumulative probability.

As you can see, when the branching ratio is 1.5, in this network the output layer never produces more than 58 active neurons in the last layer. This limits the range of responses the network can give, so it cannot encode very large stimulation values. However, it does do a fairly good job of producing different response probabilities when it is stimulated in the range of 1 to about 43. Each time the stimulus is increased, the response probability generally increases also.

To quantify the ability of the network to encode this range of values, we can measure the width of the response distribution in the following way: take its cumulative sum and mark when it reaches 20% and 80% of the total (see the right plot in the figure above for the cumulative sum of the probability). We can now find the response values that correspond to those percentiles. These are just the red lines in the left panel of the figure, dropped down to the x-axis. There, we can see that the corresponding response values are about 22 and 44. The distance between those values we can call the dynamic range. Note that one could have also picked 10% and 90% cumulative probabilities, for example – this is another common way to measure the dynamic range. The function ResponseProbability automatically picks 20% and 80% markers and draws vertical gray lines, as shown in the plot on the left. Given this background, you should be able to do the set of exercises for this section.

D. Exercise: Using the same set of values for “neurons,” “layers,” “connects” and “iterations,” now vary the branching ratio, “BR,” from 0.5 to 2.5 in increments of 0.25 and measure the dynamic range. Plot the dynamic range against the branching ratio. You should see a function that has a peak; show the plot.

E. Exercise: Repeat exercise D again, but this time vary the number of layers in the network while keeping the number of neurons per layer fixed at 64. For

example, try 10 layers now instead of 3. How does this affect the dynamic range? How does it affect the shape of the plot of dynamic range vs branching ratio?

F. Exercise: Repeat exercise D again, but this time vary the number of neurons in the layers while keeping the layers fixed at 3. For example, try 128 neurons per layer instead of 64. How does this affect the dynamic range? How does it affect the shape of the plot of dynamic range vs branching ratio?

G. Exercise: Repeat exercise D again, but this time increase both the number of layers and the number of neurons in the layers. Do these changes interact in some way? If so, how? From this, can you estimate what would happen in a network with very many neurons and layers?

H. Exercise: One of the output variables of ResponseProbability that we did not look at previously is "Hresp," which stands for the entropy of the response distribution. How does "Hresp" relate to the dynamic range? Does it peak at the same BR? Pages 75-76 of the book offer a brief description of entropy and how it could be applied to a distribution. See also Figure 4.2. One further note: the entropy of the response, Hresp, is sometimes also called the information capacity. We are now moving towards mutual information – see more below.

3. Mutual information

The book explains the intuition behind mutual information on pages 10-12. Very briefly, it can be described as a guessing game of sorts: if you know the output of a network, to what extent are you now able to correctly guess what the input was? Mutual information is a principled way to quantify this precisely. Pages 74-77 explain this in more detail. The remarks below are intended to describe mutual information in a way that builds on the previous set of exercises measuring dynamic range in feed-forward networks.

We have been working with feed-forward networks so far. To apply mutual information in these circumstances, one would take the activity in the first layer to be the input and the activity in the last layer to be the output. By seeing activity in the last layer, could you narrow down the possible inputs that caused it?

Interestingly, this approach can also be used in recurrent networks, where there is no clear input layer or output layer. Here, instead of looking at layers, we could look at the activity in the network at some initial time and compare it to the activity at some later time. In this situation, time would play the role that layers played in the feed-forward network. By seeing the network activity at time t = 10, could you even partially reconstruct what the activity in the network was at time t = 0?

One other remark is in order – this concerns the difference between magnitude and configuration. In the case of dynamic range, we asked whether the magnitude of the output could tell us something about the magnitude of the input. There was a simple type of mapping that we were looking for: as the inputs got larger, we expected the outputs to get larger too. A large dynamic range would allow this mapping to extend widely, so that more of the input space could be encoded. We found that broad distributions had a larger dynamic range than narrow ones. In the last exercise of the previous section, you should have noticed a relationship between the response entropy, Hresp, and the breadth of the response distribution. The broader the distribution, the higher the value of Hresp. Roughly speaking, entropy measures how flat a distribution is.

However, it is easy to see that information could be carried not just by magnitude but by configuration also. For example, imagine that the initial input configuration to a network was a pattern of active neurons that formed either an “X” or an “O.” This might occur in a retina viewing two different symbols. Further imagine that the number of active neurons in both cases was exactly the same. Here, we probably would not get very far in guessing the input symbol if we only counted the number of active neurons 10 time steps later. We would need to know something about the neurons’ arrangement to guess which input pattern occurred at time t = 0.

Fortunately, mutual information can take configuration into account as well as magnitude. It does this by looking at the conditional entropy, Hcond, in addition to the previously mentioned response entropy Hresp. Intuitively, Hcond is related to the breadth of the output distribution *when the same input is presented again and again*. If a network is to reliably transmit information, we would expect it to have a very narrow range of outputs when it is presented with the same input repeatedly. Ideally, it would only have one output when the same input was presented. This would lead to a very narrow distribution, causing Hcond to be small or even zero. Hcond is sometimes called the “equivocation” because it measures the uncertainty in the response to the same input.

The mutual information involves both Hresp and Hcond. Hresp quantifies the breadth of the entire output distribution, while Hcond quantifies the breadth of the output distributions for each input. For mutual information to be large, we need a large Hresp for all the inputs taken together and a small Hcond for each individual input. The equation for mutual information between a set of stimuli, S, and a set of responses, R, can be given by:

$$MI(S;R) = H(R) - H(R|S)$$

Here, H(R) is Hresp and H(R|S) is Hcond (see page 76 of the book). As you can see, the upper bound of MI is set by Hresp; for this reason it is also called the “information capacity.” Hcond only detracts from this information capacity; it is also called the “equivocation.”

Now that we have given this background, you should be prepared to do the following exercises.

I. Exercise: Using RecurrentInfoFunction, produce the output variables “TotalMI,” “Hresp” and “Hcond” for a recurrent network. You can use it like this:

```
[TotalMI, Hresp, Hcond, BRs] = RecurrentInfoFunction(N, r, c,…
connects, reps, timesteps, increments, increment, numNets);
```

Where “N” is the number of neurons selected from the recurrent network to be used as input. The same number, but different neurons, will be used for output. The variables “r” and “c” determine the total number of neurons in the network (total number = r x c). “Connects” is the number of connections from each neuron to other neurons. “Reps” is the number of times the activity will be repeatedly run through the network. “Timesteps” is the number of time steps the recurrent network will be run for. “Increments” is the number of different branching ratios to be tried. “Increment” is the difference between each branching ratio, given that they will start out at 0.1 for the first branching ratio (notice that the output variable “BRs” gives the sequence of branching ratios tried). “numNets” is the number of networks to be run and averaged. To start, use these input values:

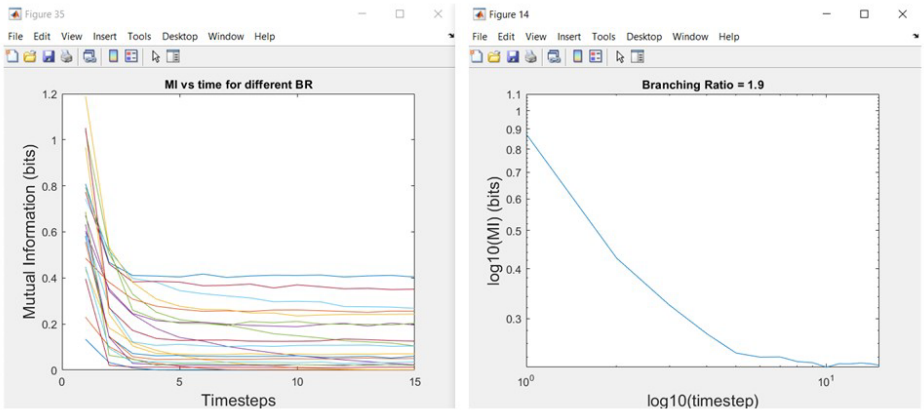
```
[TotalMI, Hresp, Hcond, BRs] = …
RecurrentInfoFunction(6,4,4,6,100,15,21,0.2,2);
```

This should produce output in about 7 - 10 minutes and will be good enough to see some plots with the general features you will be looking for. Mutual information calculations can be intensive and take longer than the exercises we have done up to this point. If there are 6 neurons in this case being stimulated, how many different configurations could they have (assume each could be either on (1) or off (0)). Keep the output variables “TotalMI,” “Hresp,” “Hcond,” and “BRs” because we will use them later as inputs to other functions.

J. Exercise: Plot the mutual information against time by using the function InfoVsTime:

```
InfoVsTime(TotalMI, BRs);
```

You should see something like the figure below.



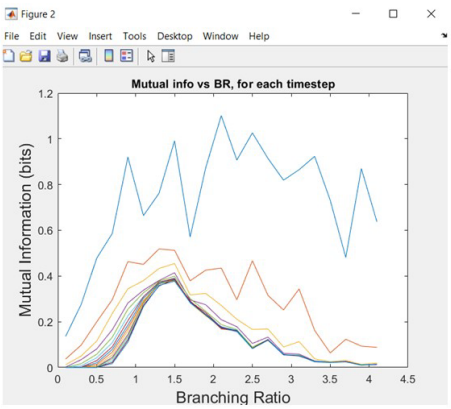
Left, Information plotted against time in a recurrent network for branching ratios ranging from 0.1 to 4.1; each color represents a different branching ratio. Right, Information plotted against time in log-log coordinates for one branching ratio (BR = 1.9) from the same network. Network had 16 neurons; a subset of 6 were used for input and a different set of 6 were used for output. Each curve is the average for 6 such networks.

By looking through all the plots for the individual branching ratios (right side of figure above), you should be able to identify the curve that maintains the highest information by the last time step. Which branching ratio corresponds to that curve? Now plot the final information value (at the last time step) for each of these curves against the branching ratio.

K. Exercise: Next plot the mutual information against the branching ratio. Do this by using the function RecurrentMIplotter as shown below:

```
RecurrentMIplotter(TotalMI, BRs);
```

If you used the suggested parameters above, you should get a plot that looks like this:



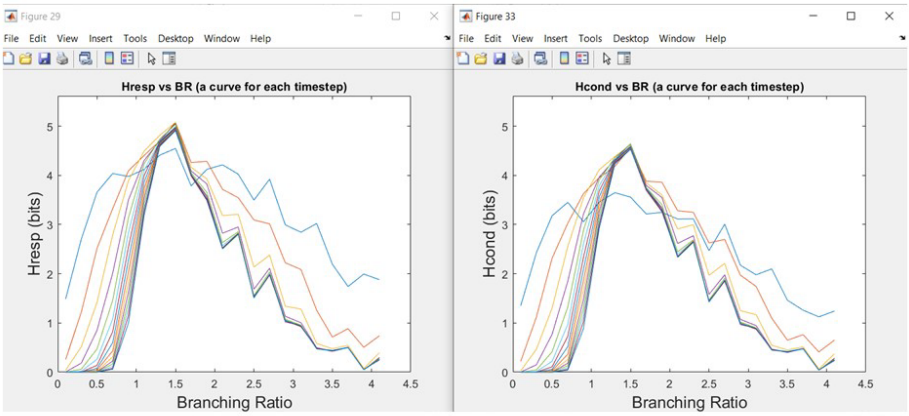
Mutual information plotted against branching ratio in a recurrent network. Each curve is taken at a different time step ranging from 1 to 15. Early curves have more Information but are highly variable. Later curves have less information but have a more consistent peak. Same network as in the previous figure. Each curve is the average of 6 networks.

Notice that each curve represents a snapshot of the mutual information present at each time step. The first time step has the most mutual information, while the last time step has the least. Why are the early curves broad and why are the later curves more sharply peaked?

L. Exercise: Plot the response entropy and the conditional entropy separately. You can do that by using EntropyPlotter, like this:

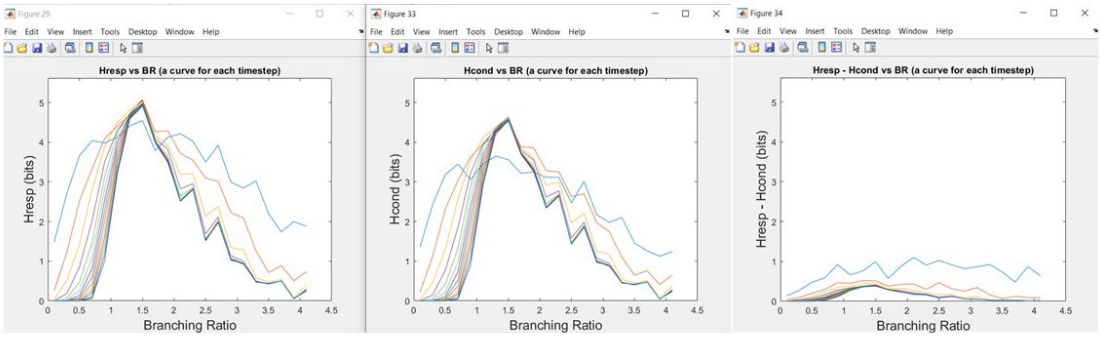
```
EntropyPlotter(Hresp, Hcond, BRs);
```

You should get plots that look like this:



Left, Response entropy (Hresp) plotted against branching ratio for a recurrent network. Each curve represents Hresp at a different timestep, ranging from 1 to 15. Right, conditional entropy (Hcond) plotted against branching ratio, with curves representing different times. Same network as used in the previous two figures.

Given the earlier exercise on the susceptibility, does it make sense that the Hresp curve should also show a peak? Can this reasoning be applied equally well to the Hcond curve? Why or why not?



The two output entropies, Hresp (left plot) and Hcond (middle plot), shown along with the difference between them (rightmost plot). Same network as in previous three figures. For proper comparison, all plots have the same scales.

Why is the plot of (Hresp – Hcond) so small? Zoom in on it - is it the same as the mutual information plot you made previously?

From the Hresp and Hcond curves, as well as the susceptibility curves seen earlier, it looks like output variability is maximal near the critical point. This would seem to be bad for reliable transmission of information. Yet the mutual information curves peak near the critical point. Explain why this is the case.

M. Exercise: As with the susceptibility and the dynamic range, we will now look at how these results scale with increased network size. Starting with the parameters used above (N = 6, r = 4, c = 4, timesteps = 15, numNets = 2), we will systematically vary them to see their effects.

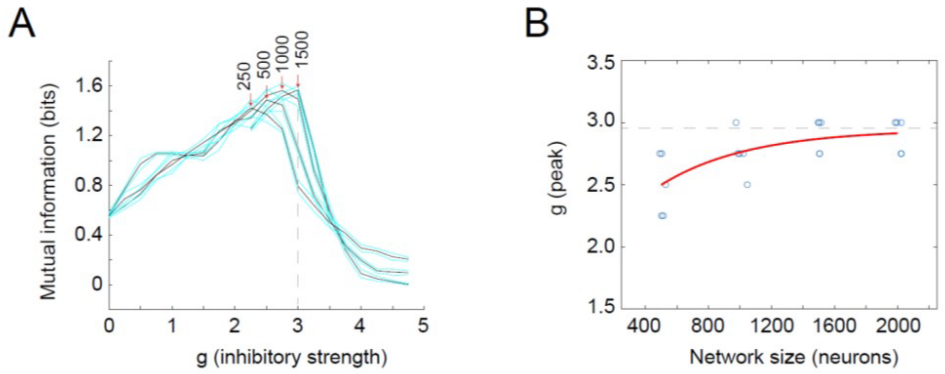
First, what are the effects of increasing the number of networks used? Increase numNets from 2 to 20. How does the curve look different? Note that this will take ten times longer to run than your previous run, so plan accordingly.

Second, what are the effects of decreasing the size of the network (go from 4 × 4 to 3 × 3) on the mutual information curve? What are the effects of increasing the size (go to 6 × 6)?

Third, what are the effects of decreasing the number of neurons stimulated (go from 6 to 4)? What are the effects of increasing this number (go from 6 to 8)?

Fourth, what are the effects of increasing the number of timesteps used? For example, go from timesteps = 15 to timesteps = 30?

Fifth, if you really have access to lots of computing power, you could run these networks with very large sizes and for very long times. Below is a figure showing a Brunel model network, like the one used at the beginning of the exercises for Chapter 3, when it is scaled up (figure from Beggs, 2022, Frontiers in Computational Neuroscience). Notice that the peaks for the mutual information curves move toward the point where g = 3 (g is the control parameter in this model). To the right is an asymptotic plot for multiple runs of the model at various sizes. Fitting an exponential curve to this plot produced a best estimate of the critical value of g. Attempt a similar analysis for the recurrent branching model here.



A, Mutual information between the response and the stimulus was measured in Brunel models of different sizes (250, 500, 1000, 1500 neurons shown) as the inhibitory strength, g, was varied. Stimuli consisted of eight different numbers of neurons (e.g., 0, 125, 250, 375, … for N = 1000 neuron model) randomly activated at one time; the average number of neurons active at time steps 3 through 5 after the stimulus was taken as the response. Black curves show mutual information for each model; cyan curves show one standard deviation. Five models of a given size were run 30 times each to produce each data point; more details are in the supplementary material. Peak mutual information values for each model size are indicated by red arrows. Note that as model size increases, peaks become taller and move toward g = 3. Dashed vertical line is at g = 3 for reference. **B**, The value of g at which mutual information peaks is plotted against model sizes (N = 500, 1000, 1500, 2000 shown). Blue circle tokens were jittered slightly to improve visibility. An exponential fit to these data gives an asymptotic value of g = 2.952, with 95% confidence bounds at 2.637 and 3.268.

Matlab code used for exercises in this chapter, listed in order of use:

FeedForwardBranchingModelFunction
weights2
spy2
SusceptibilityCalc
SusceptibilityDemoFunction
ResponseProbability
remove
RecurrentInfoFunction
RecurrentModel
RecurrentInfoTester
bin2dec2
FindInformation
InfoVsTime
RecurrentMIplotter
EntropyPlotter