Exercises for Chapter 6, Homeostasis and Health

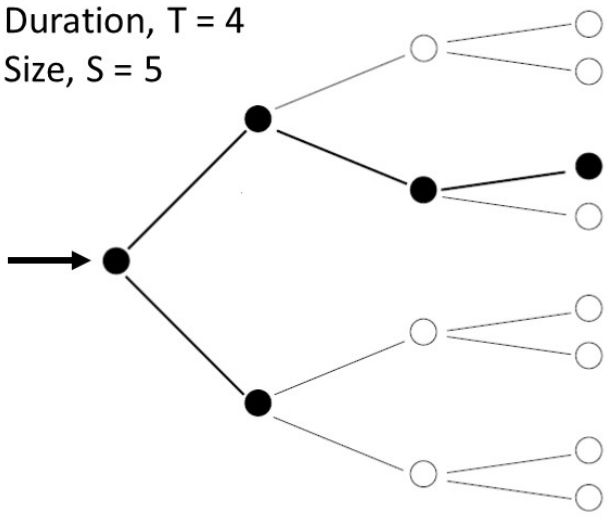
The main message of Chapter 6 is that living neural networks self-organize to operate near the critical point. In Chapter 4 we saw that being near the critical point optimized many information processing functions, but we did not address the question of *how* the network got there; Chapter 6 is concerned with this very issue. We reviewed experiments showing that even in the face of substantial perturbations, living neural networks always find ways to return toward the critical point. This process of returning is called homeostasis and is thought to be important for neurological health.

1. Description of the model

In these exercises we will look at a very simple model that self-organizes to operate near the critical point. This model is inspired by the following paper:

Zapperi, Stefano, Kent Bækgaard Lauritsen, and H. Eugene Stanley. "Self-organized branching processes: mean-field theory for avalanches." *Physical review letters* 75, no. 22 (1995): 4071.

In this paper, the authors take the simple branching model that we use in the book and they endow it with a self-adjusting branching ratio. For simplicity, the model is run on a binary tree like the one shown below.



An example of an avalanche on a binary tree, where each node has two descendants. Active nodes are in black; inactive nodes are white. A tree like this was used by Zapperi et al., 1995, and is adopted for our exercises in this chapter. The node at the left initiates activity which can then propagate to two other nodes in the next layer. Each connection from an active node has a probability of transmitting given by P_{trans}. When $P_{trans} = 0.5$, the branching ratio is exactly 1 and the network is critical ($P_{trans} \times 2$ connections; $0.5 \times 2 = 1$).

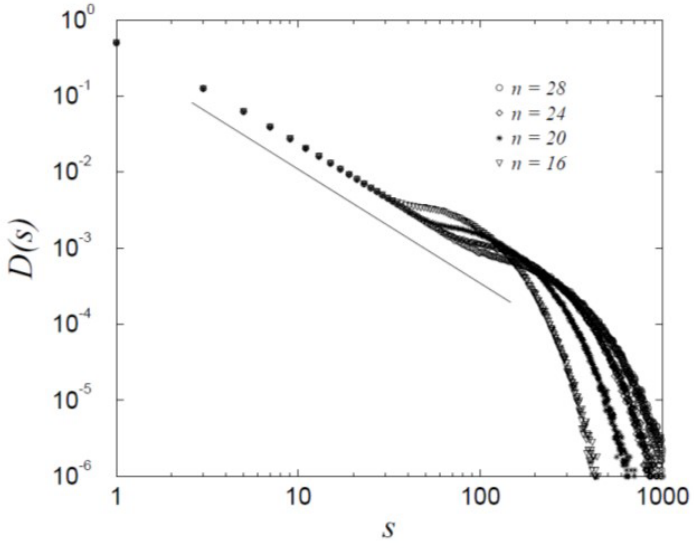
In this binary tree, each node can transmit to only two neighbors in the next layer. The probability of transmitting is given by P_{trans}. The branching ratio is just $P_{trans} \times 2$. At the start, P_{trans} is given a random value between 0 and 1, meaning that the branching ratio could be between 0 and 2. Over time, P_{trans} is adjusted in response to the number of neurons activated in the last layer of the network. If that number is zero, P_{trans} is slightly incremented for the next avalanche; if that number is one or more, P_{trans} is slightly decremented for the next avalanche. After many avalanches, P_{trans} converges toward 0.5, producing a branching ratio near 1. This negative feedback accomplishes the self-organization.

To give slightly more detail, the transmission probability at the next time step, $P_{trans}(t+1)$, is related to the transmission probability at the previous time step, $P_{trans}(t)$, by the following equation:

$$P_{trans}(t+1) = P_{trans}(t) + A\left(\frac{1 - N_{last}}{N_{all}}\right)$$

Here, N_{last} is the number of active nodes in the last layer of the tree, N_{all} is the total number of nodes in the tree, and A is a parameter that regulates how quickly the transmission probability is adjusted. When $A < 1$, adjustments are stable and slow; when $A > 1$, adjustments are faster but could become unstable.

When Zapperi and colleagues ran their model, it produced avalanche size distributions like the ones shown below.



Avalanche size distributions observed by Zapperi and colleagues using their self-organizing branching process model. The probability of an avalanche of size *s* is given by *D(s)*; results are plotted in log-log coordinates. Adapted from Zapperi et al., 1995, Physical Review Letters. Four overlaid plots are shown, for values of *n* from 16 to 28, where the number of nodes in each network is given by $N = 2^{n+1} - 1$. The straight line has a slope of -1.5 and is included for reference. Note that the distributions have humps at the end. These are the so-called exponential cutoffs and are caused by avalanches that were still expanding in size when they reached the last layer.

Clearly this is an abstract model and is not intended to have physiological realism. We do not have here mechanisms like synaptic scaling or pruning of connections, as discussed in the book. Still, this model can be helpful in building our intuitions about homeostasis and the critical point. Further, universality dictates that if any model is to apply across many scales, it should not contain details from any particular scale. Rather, it should be conceptually simple to be relevant across all scales. This branching model satisfies those requirements.

2. Determining if the tuned model is critical

A. Exercise: Before investigating how the model self-organizes, let us first see what the output of the model looks like when P_{trans} = 0.5 and the model is set to the critical point.

To explore this, use BP_Function like this:

```
[TIMERASTER] = BP_Function(Layers, timesteps);
```

Where Layers is the number of layers in the binary tree and timesteps is the number of avalanches to be run. On my laptop computer, this took about 32 seconds to run when Layers = 12 and timesteps = 50000.

The TIMERASTER output can then be fed into the analysis functions that we first used for the exercises to Chapter 3. This will allow us to view avalanche distribution plots. As before, use the AvalancheAnalysis function like this:

```
[sizeDist, durationDist, SvsT, Events] = AvalancheAnalysis(TIMERASTER);
```

You should see an avalanche size distribution that looks similar to the one shown above from (Zapperi et al., 1995). Note that the avalanches in the exponential tail may cause the avalanche duration distribution to have a corresponding hump at the end. In the same way, the avalanche size versus duration (SvsT) plot may have be distorted at the end.

With this in mind, we can proceed to check if these data approximately satisfy the exponent relation and avalanche shape collapse. To do this, you can use the output from AvalancheAnalysis to run the lines of code below, just like we did in the exercises for Chapter 3 on criticality. You will have to select LimL and LimU, the lower and upper limits, respectively. For the numbers put into BP Function above, it is reasonable to pick 4 and 10 for LimL and LimU. While this is a very limited range, this is about all you can do with such a small number of timesteps and Layers. Longer ranges of power law fitting can be obtained by running much longer and with more layers. For the function ShapeCollapse, you can give it an interval of 1 and gamma_act that was produced by the function ExponentRelation (recall we described this in the Chapter 3 exercises).

```
[CCS, CCD] = GetCCDFs(sizeDist, durationDist);
```

```
[alpha, tau, gamma_est, gamma_act, error] = ExponentRelation(CCS, CCD, SvsT,
LimL,LimU);

[Shapes] = ShapeCollapse(Events, LimL, LimU, interval, gamma_act);
```

Show these plots and comment on them. Does the model satisfy, within some limits, the criteria for operating near the critical point? Investigate whether increasing the number of layers and timesteps allows it to satisfy these criteria over a larger range.

3. Using homeostasis to watch the model self-organize

B. Exercise: Now we will move on to the self-organizing model, which uses homeostasis to approach the critical point. Here, we will use SOBPfunction, as shown below:

```
[TIMERASTER, BranchingRatio] = SOBPfunction(Layers, timesteps, A, Pstart);
```

As before, you can use values like Layers = 14 and timesteps = 50000. The parameter A is the constant that determines the magnitude of negative feedback involved. You can set A = 1 for now. Pstart is the probability of transmission at the start of the run. Recall that in this binary tree P = 0.5 will lead to a critical branching ratio of 1. Homeostasis will cause P to move from Pstart to 0.5 over time. You can pick Pstart = 0.75 or 0.25, for example; anything between 0 and 1 will be acceptable.

The function will plot the value of P over time. Verify that it converges toward P = 0.5 from both above (e.g., Pstart = 0.65) and below (e.g., Pstart = 0.35).

Experiment with different values of A. Overall, what is the effect of the value of A? Increase and decrease it, then zoom in on the plots of P over time to see how this is affected.

For a given value of A (say, A = 0.05) does the approach toward criticality from above or below differ in terms of the number of timesteps it takes? If one is generally faster, can you explain why?

4. Multiple acute perturbations

C. Exercise: The cortex may be perturbed by dramatic changes in sensory inputs, fevers, rapid swings in ionic concentrations, or traumatic injuries. Homeostasis of criticality should therefore be able to accommodate multiple perturbations from different sources. To simulate this, use the self-organized branching process function with perturbations, as given below:

```
[TIMERASTER, BranchingRatio] = SOBPfunctionPerturbed(Layers, timesteps, A, NPerts);
```

Reasonable starting parameters, for example, would be Layers = 12, timesteps = 50000, A = 0.25, and the number of perturbations, NPerts = 6. Running these parameters took about 35 seconds on my laptop computer.

Verify that the system can adapt to multiple perturbations by trying several different values of NPerts. What happens when NPerts becomes very large, say, 100? Is the average branching ratio still near 1? Does it in general become greater than 1 or less than 1 for large numbers of perturbations? Can you explain what you observe?

5. Continuous chronic perturbations

Besides the large perturbations addressed in the previous section, a given patch of cortex is constantly receiving synaptic inputs from more distant cortical areas. Such inputs may not be as abrupt as a brain injury but are more continuous. In this section, we will take the SOBP model and continuously subject it to small random perturbations. These may in some respects simulate the constant synaptic inputs arriving at a patch of cortex; these constant inputs drive the system away from equilibrium. This situation emphasizes the need for using non-equilibrium models of criticality.

D. Exercise: Use SOBP_Pspont_function to explore how the branching process network responds to increased levels of external drive, like this:

```
[TIMERASTER, BranchingRatio] = SOBP_Pspont_function(Layers, timesteps, A, Pspont);
```

Reasonable starting parameters could be Layers = 12, timesteps = 50000, and A = 0.5. Start with Pspont = 0 and then gradually increase it by increments of 0.05 to Pspont = 0.50. How does the average BranchingRatio produced by the function change as Pspont is increased? Explain what you think is causing this effect. Does this effect depend on the feedback? To find out, you could set A to zero. Keep this result in mind, as it may be helpful when we move on to quasicriticality in Chapter 7.

For every increment of Pspont above, save the TIMERASTER produced by SOBP_Pspont_function (there should be 11 of them). Later put each of them into AvalancheAnalysis to see what their avalanche size distributions look like. How do these change as Pspont is increased? Again, explain why you think this is happening?

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

BP_Function

AvalancheAnalysis

GetCCDFs

ExponentRelation

ShapeCollapse

SOBPfunction

SOBPfunctionPerturbed

SOBP_Pspont_function