Assignment 5

Network modeling

Finn Joel Bjervig*, August Forsman†, and Erik Turesson‡

1MA256 - Modeling Complex Systems,
Department of Mathematics,
Uppsala university, Sweden

May 31, 2021

*Electronic address: jobj8920@student.uu.se
†Electronic address: aufo8456@student.uu.se
‡Electronic address: ertu2293@student.uu.se

Hopfield Network Model

Imagine a fully connected network/graph g, meaning every node n_i has all other nodes n_j as neighbors via the connecting edges. These edges has weights ω_{ij} assigned to them which are symmetric. Further, there are no edges and thus no weights, connecting to the same node.

$$\omega_{ij} = \omega_{ji}$$
$$\omega_{ii} = 0$$

The nodes dynamic states s_i is bipolar $s_i \in \{-1, 1\}$, and since these states are dynamic, it infers they will change in time. The updating of all nodes $s_i(t+1)$ happens synchronously by a sort of weighted majority rule from neighboring nodes states $s_j(t)$, but not by the center nodes state $s_i(t)$. The new state of a node is therefore independent of its previous state.

$$s_i(t+1) = \text{sign}\left(\sum_{j \neq i} \omega_{ij} s_j(t)\right) \tag{1}$$

The sign(x) function gives -1 if $x < 0$,

$$\text{sign}(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x = 0 \\ -1 & \text{if} \quad x < 0 \end{cases} \tag{2}$$

With this structure defined, we *hop* to what John Hopfield found, which was that by designing the initial weights of a network g, one can imprint a finite number of predefined network states $h, k, l..$ (of the same size and structure as defined above) such that the network reaches one of the states after some number of updates. These predefined states will be referred to as memories, that are imprinted onto the networks weights. The weights are constructed as follows

$$\omega_{ij,g} = \sum_k s_{i,k} s_{j,k} \tag{3}$$

This extends the theory to several memories k meaning we can imprint several memories into the weights of g.

Say we want the network g to reach the memory graph h. From randomly chosen node states and weights defined by equation 3, the network evolves from top to bottom in figure 1

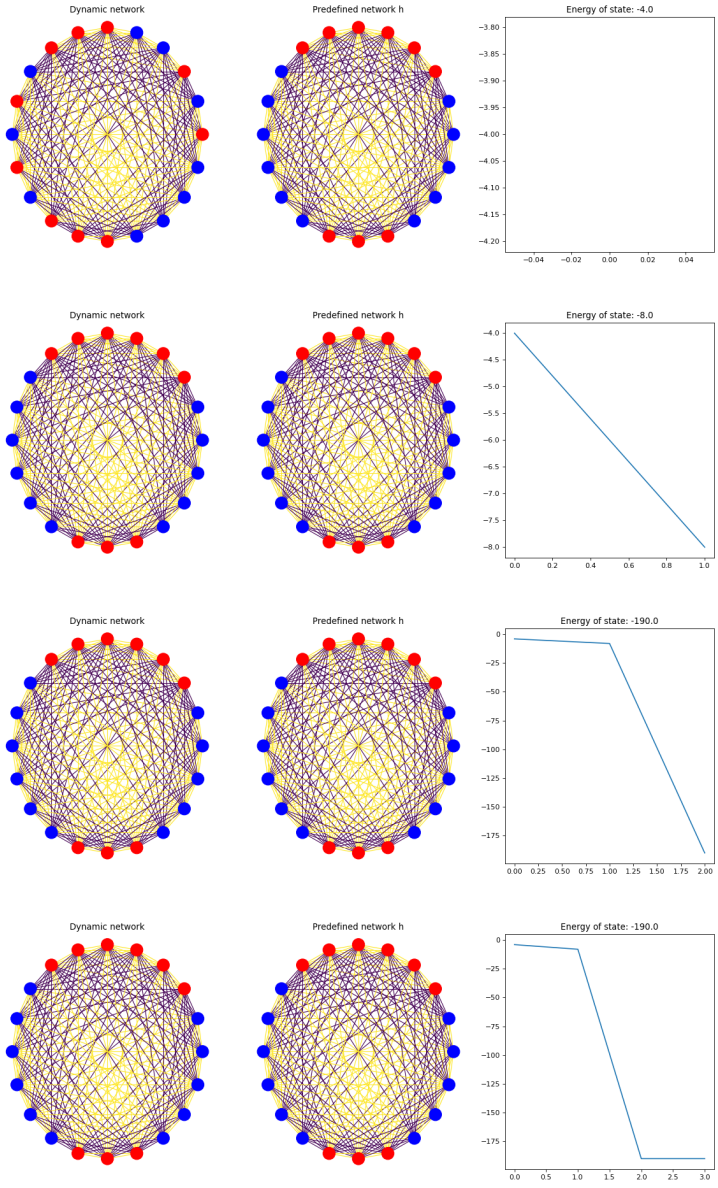

Figure 1: The left graph has randomly chosen node states, but defined weights, calculated from the nodes of the predetermined graph to the right. The weights of the memory graph is set equal to that of the graph to the left. While they will not be used, they visualize how the calculation of the weights. Looking at the memory graph, we see that there are negative weights between nodes that are different, while nodes of equal states have a positive weight.

The plot to the right shows the energy of the system which is

$$E = \frac{1}{2} \sum_{i,j} \omega_{i,j} s_i s_j \tag{4}$$

Expanding the notion to several memories we yields an *energy landscape* where the memories $h, k, l..$ will be represented as local minimums, or valley as seen in figure 2. Therefore the evolving network will converge to one of these states depending on the initial condition. Think of it as a ball being dropped to a random position on the landscape which will roll down a slope to one of the valleys. All states that will converge to the same valley are called *basins of attraction*. But this is not the only result. The network actually converges to a *spurious* steady state that contains the following cases

1. Reversed fundamental memories - if the memory is x_f then so is $-x_f$

2. Mixed fundamental memories - the equilibrium state is a linear combination of the memories

3. Spinglass state - local minima not representing any of the memories

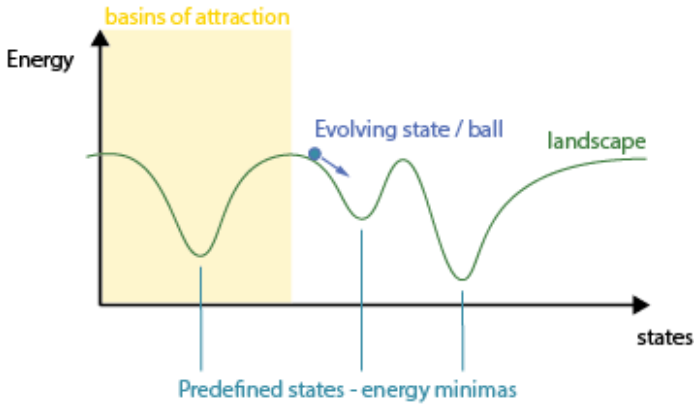

Figure 2: *All the predetermined states which are embedded into the weights of the network/ball will constitute as valleys in an energy landscape determined by equation 3*

In appendix A there are figures for three different scenarios for a graph that has three memories imprinted.

Part 2

The network was constructed using the python module `networkx`. In the creation, the user can specify values of $p_r,\ p_s,\ p_i,$, the fraction of initially infected nodes, the number of nodes, and average number of neighbours per node. Each graph is created randomly, given the number of nodes and and the probability of two nodes being connected by an edge. The user can specify whether to use update all nodes in an iteration, or update a single one chosen at random. All tests in this section will be done by updating all nodes in each iteration. Note that since both the graph and simulations are stochastic, results will be noisy and inconclusive.

For any pair $p_s, p_r > 0$, the pandemic will eventually be eradicated. The explanation for this is rather simple: if $p_s > 0$, the number of edges will decrease as long as there's infections in the system. In the extreme case, this will continue until no there are no edges remaining, making the epidemic unable to spread. Further, if $p_r > 0$, any single infection will eventually recover. Thus, in the most extreme case, each node will have no edge to another link, and each infected node will eventually recover.

As an example, the initial state of a generated graph of 30 nodes and 1.5 neighbours on average is seen in Figure 3. After 10 time steps with $p_s = 0.2$, the result is seen in Figure 4. It is evident that some links have been severed and that the number of infections have reduced.

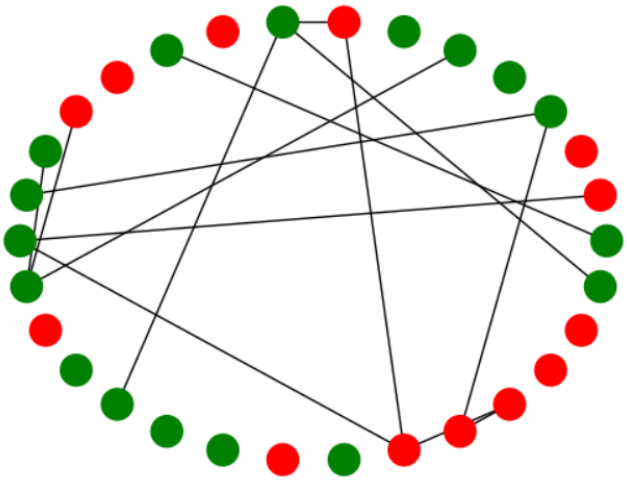

Figure 3: Initial state of 30 nodes with an average of 1.5 neighbours. Green show susceptible and red infected individuals.

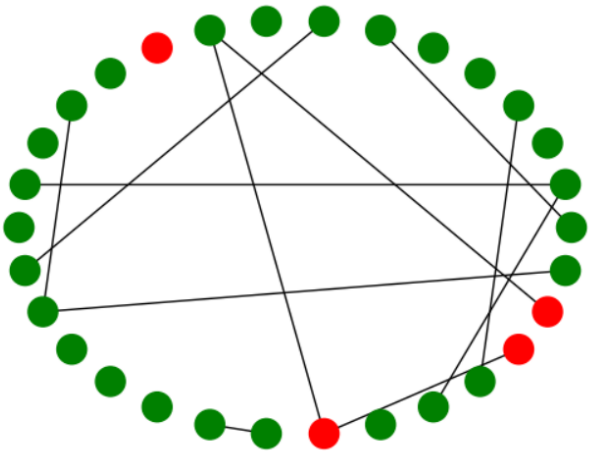

Figure 4: State after 10 iterations on a graph with 30 nodes with an average of 1.5 neighbours. Green show susceptible and red infected individuals.

If $p_s = 0$, things get more complicated as the number of edges will remain constant. Instead we need to rely on infected recovering more often than they spread to any neighbour. For the initial values of $p_i = 0.5$, $p_r = 0.2$, it seems as if the epidemic only settles for when there's a low number of neighbours for each node. After running some tests, no graph with more than 2.5 neighbours on average seemingly doesn't settle. For $1.5 <$ neighbours < 2.5 most graphs settle in a state of an eradicated epidemic, although after several hundreds or even thousands of time steps. For any lower averages of neighbours, eradication is usually seen within 100 time steps.

By running simulations for varying values of p_s, with $p_i = 0.5$, $p_r = 0.2$ and 10000 nodes with an average of 10 neighbours, and terminating once 0 infections are reached, we can see how the number of iterations needed before eradication of the epidemic is complete. The results are seen in Figure 5. A correlation of more iteration required for lower values of p_s is seen. This is expected since it means that less edges will be be severed, causing the limitation of the spread to be less severe.

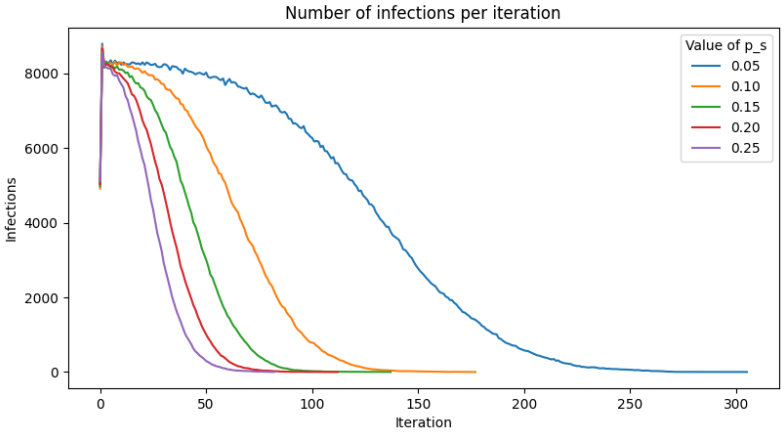

Figure 5: Number of infected people in each iteration for different values of p_s. Simulations used 10000 nodes with and average of 10 neighbours, with $p_i = 0.5$ and $p_r = 0.2$

Appendix

A Examples of Hopfield Scenarios with three memories imprinted

These scenarios are examples of the three spurious steady states that the Hopfield model may reach, as discussed at page 4. The figures are structured as follows: top row are the memories, and the other graphs below is the one network that evolves in time. The last plot shows the energy of this network over time. The network stops evolving when it reaches one of the steady states, correlated with locally having the lowest energy, i.e. being stuck in the valley.

Reversed fundamental memories

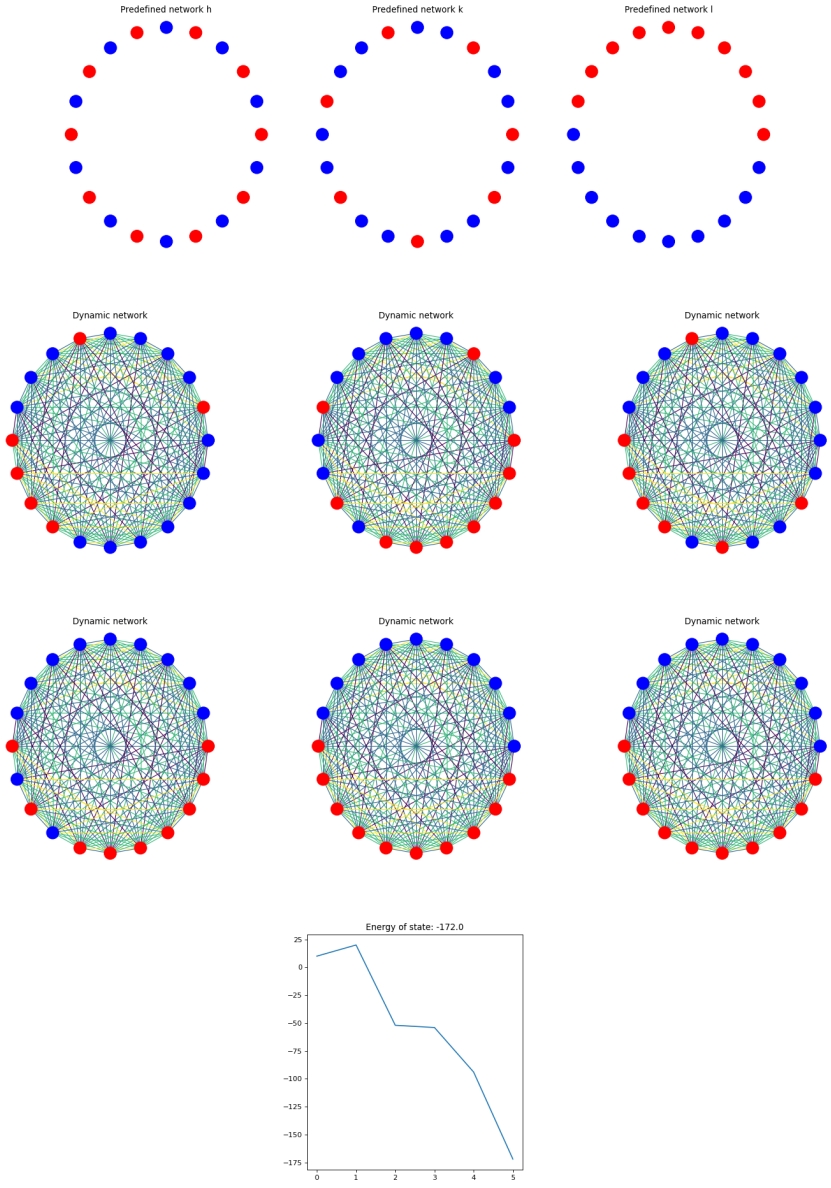

Figure 6: *Evolution of states is read as text: from left to right, and down. As stated before, if the memory state in the upper right corner is a solution, so is the inverted state*

mixed fundamental memories

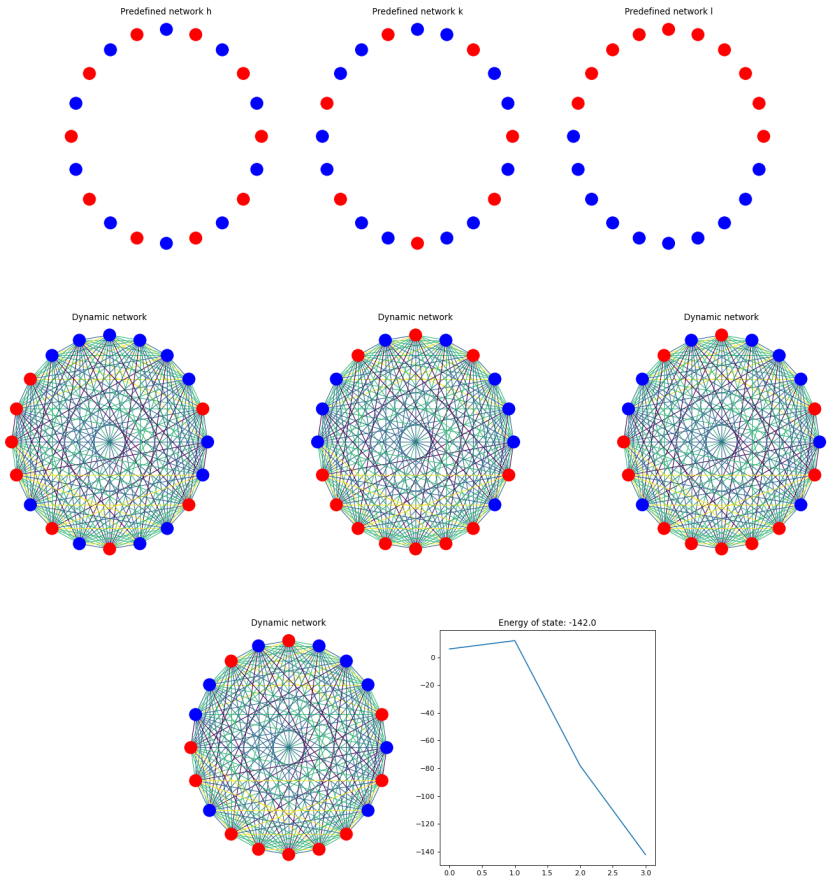

Figure 7: *Evolution of states is read as text: from left to right, and down. The steady state here is a linear combination of the memories*

Spinglass state

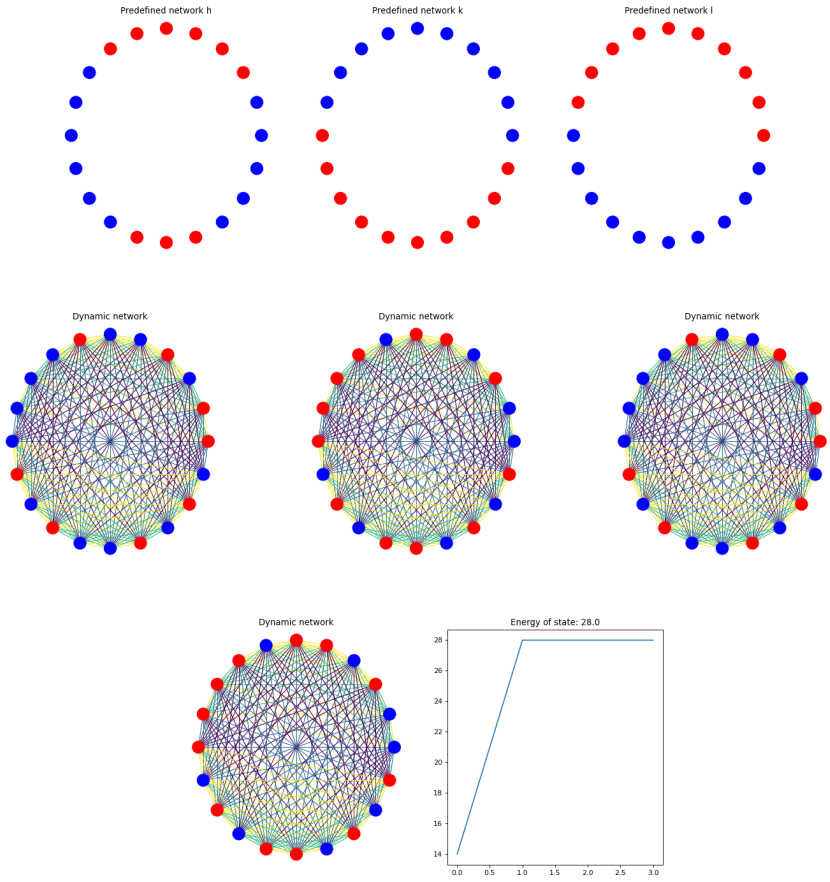

Figure 8: *Evolution of states is read as text: from left to right, and down. This particular example shows an alteration between two states that are neither of the memories. It is possible they are linear combinations of the memories, like in the mixed fundamental memory state.*

Python Code

Code files are attached with submission and/or found on GitHub