Simulating A Hopfield Network

Yashila Ramesh
ECE 3803 Fall 2025
Georgia-Tech Europe
Metz, France
yramesh6@gatech.edu

***Abstract*—This project intends to do the following: simulate a Hopfield network solution to store patterns and calculate a MaxCut solution for a small graph.**

I. Introduction To Hopfield Networks

A. Neural Network Models and Associative Memory

Neural networks can be understood as dynamical systems that evolve toward stable states. Among the earliest examples, the Hopfield network (Hopfield 1982, 1984) demonstrates how a fully connected recurrent network can function as an associative memory. In such networks, information is not stored at individual nodes but rather distributed across the connections. This structure enables pattern completion: given a noisy or incomplete input, the system evolves toward the nearest stored memory, known as an attractor.

Hopfield networks are a canonical example of autoassociative memory models, where the retrieval cue and the target memory have the same form. This makes them a natural framework for studying memory robustness, stability, and optimization.

B. The Hopfield Model

C. Energy Functions and Dynamics

One of the most important aspects of an implementation of a Hopfield network is the choice of energy function. For the purposes of this project, I stayed pretty close to the traditional implementation, pulling from Hopfield's original 1982 paper. $E(s) = -\frac{1}{2}s^{\top}Ws - b^{\top}s$. (3)

The energy function formalizes the idea of attractors: each stored pattern corresponds to a local minimum. During recall, the system naturally moves downhill in this energy landscape, converging to the attractor closest to the initial condition. For continuous Hopfield networks, the ODE dynamics described above can be shown to decrease E over time, ensuring stability. The nonlinearity, tanh, constrains outputs between –1 and +1, mimicking biological neurons' saturation and preventing unbounded growth.

D. Applications of Hopfield Networks

Hopfield networks have two applications relevant to this project:

1) Pattern Storage and Recall.

Networks can store a set of patterns through Hebbian learning. When presented with a corrupted version of a stored pattern, the network dynamics correct the errors and converge to the nearest attractor.

2) Optimization Problems.

Many combinatorial optimization problems can be reformulated as energy minimization tasks. One of the most common is the Max-Cut problem, where the goal is to partition a graph into two sets of vertices such that the sum of edge weights across the cut is maximized. By defining weights appropriately, the Hopfield network dynamics perform gradient descent on the equivalent energy function, converging to approximate solutions

II. Methods and Results

A. Simulation Setup

First, it is worth noting the governing equations for a Hopfield network. From the 1984 paper (1):

$$\tau\frac{du}{dt} = -u + Ws + b\,,\; s = \tanh(\beta u)$$

u: internal state vector. s: neuron outputs. τ: time constant. β: gain. b: bias vector. And the Lyapunov function or energy function is:

$$E(s) = -\frac{1}{2}s^{\top}Ws - b^{\top}s$$

When implemented in the code, the first thing that is performed is creating the weight matrix:

$$W = \frac{1}{n}\sum_{m=1}^{k} s^{(m)}\, s^{(m)\top}$$

Then the network object is created which stores the weight matrix, fixes the parameters, sets a random generator for random initial values, and initializes u and s. From here we run the dynamics of the systme and integrate the ODE with RK4 for a set total time T (5.0). With this Hopfield network setup we can use it for the use cases pertinent to this project.

B. Pattern Storage

To begin using our Hopfield Network, we stored 3 patterns in our network. These patterns were formed to cover 3 cases with a network dimension of 25. One where the first half of the indices are 1 and the second half are 0. The second pattern has all even indices as a 0 and odd as 1. The third pattern randomly assigns indices with either a 0 or 1.

Once the patterns are created the Hebbian weights are computed with the patterns stacked vertically passed into the Hebbian weight computation. The weight computation first maps the patterns to {-1, 1} from {0, 1}. From there a classic Hebbian weight computation is used:

$W = \frac{1}{n}\sum_{m=1}^{k} s^{(m)}\, s^{(m)\top},\; s^{(m)} \in \{-1,1\}^n$ [Hopfield 1984]

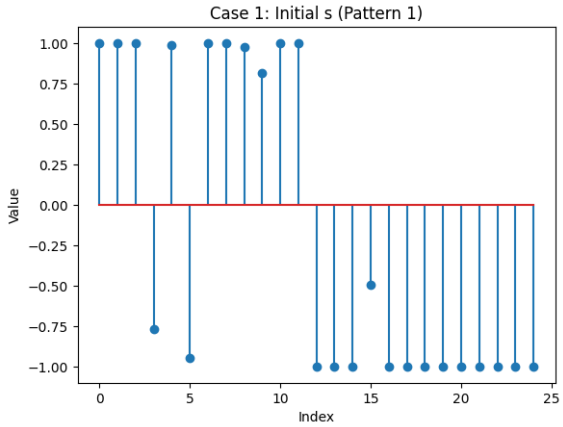
Furthermore, since neurons cannot have any self-connections, the diagonal is set to 0. The 1/n term ensures the normalization of the W matrix so that the weights do not grow as the number of neurons grow. This keeps the dynamics more stable. Then the Hopfield network is built according to the aforementioned implementation. Then using the network, we set the state from the pattern. So first, the pattern is converted into bipolar coding, which is then clipped since arctanh(±1) = ±∞. Then we calculate the corresponding internal state using $u = \frac{1}{\beta}\tanh^{-1}(s)$. Then to test the robustness of the algorithm, Gaussian noise is added represented by:

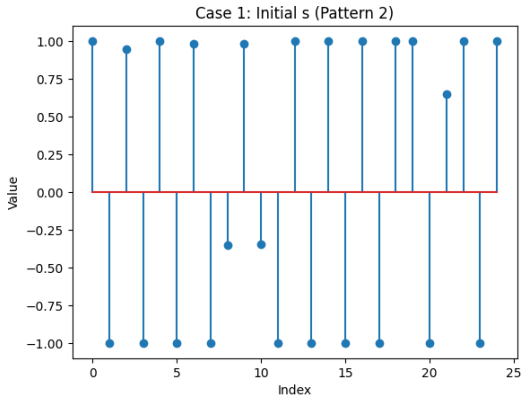
$$u_0 = u + \eta,\; \eta \sim \mathrm{N}(0, \sigma^2 I)$$

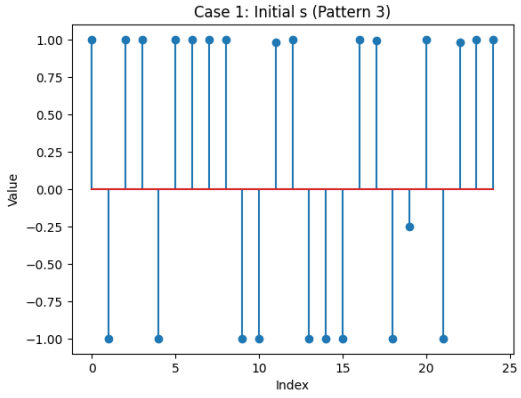
Where σ represents the noise inputted. Then we recompute the output s . This gives us our starting point, which is a noisy, corrupted version of the true memory. Now, we run the dynamics integrating the ODE for time T with steps dt. With the following parameters:

$$\tau = 1.0,\ \beta = 5.0,\ dt = 0.01,\ \sigma = 2$$

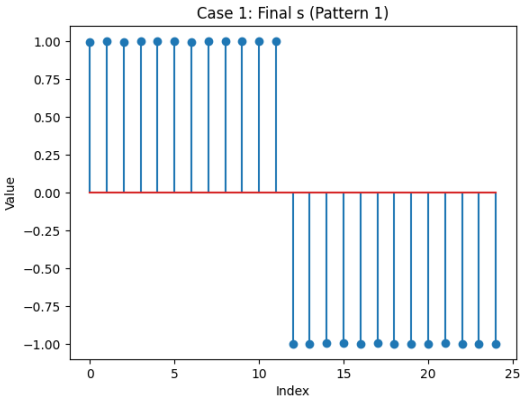
The three patterns initial s with noise look like the following:

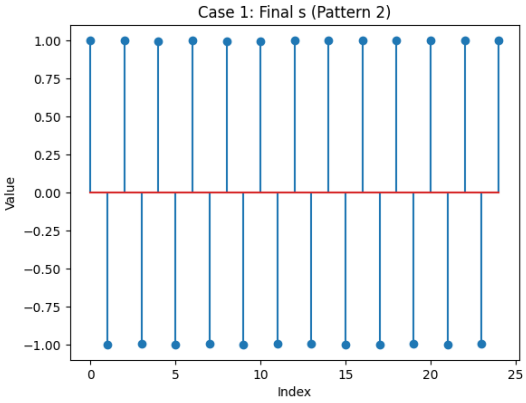


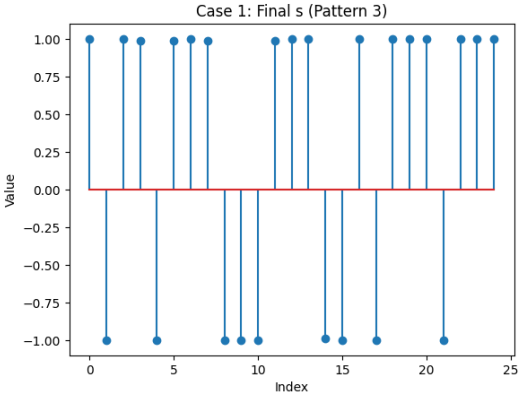




These converged to the following respectively







C. *Max-Cut*

The second portion of this project requires the use of our Hopfield network to calculate a maximum cut solution. To begin, it is useful to review what the Max-Cut problem is and the mathematics behind the solution. To begin we are given an undirected weighted graph with edge weights greater than or equal to 0. The Max-Cut problem is to Partition the vertices V into two disjoint sets S and V \ S such that the sum of the weights of the edges crossing the cut is maximized. The cut can then be mathematically summarized in the following

$$Cut(s) = \frac{1}{2}\sum_{i<j} w_{ij} - \frac{1}{2}\sum_{i<j} w_{ij}\, s_i s_j$$

Seeing as the first term is constant maximizing the cut is equivalent to minimizing the second term. Which we can rewrite as:

$$A = [w_{ij}],\ E(s) = -\frac{1}{2}s^{\top}As$$

So, The Max-Cut problem is an Ising energy minimization problem. It may become apparent here that this matrix looks pretty similar to the Hopfield energy equation. Recall:

$$E(s) = -\frac{1}{2}s^{\top}Ws - b^{\top}s$$

If we set W = -A and b = 0

$$E(s) = -\frac{1}{2}s^{\top}(-A)s = \frac{1}{2}s^{\top}As$$

So the Hopfield network naturally becomes an approximate solver for Max-Cut.

It runs dynamics until it lands in some attractor (a local minimum), which corresponds to a partition with a large cut value.

With the following graph:

edges = [

(0,1,1.0), (0,2,0.8), (1,2,0.5), (1,3,1.2), (2,4,1.1),

(3,4,0.7), (3,5,1.0), (4,5,0.9), (4,6,0.6), (5,7,1.3),

(6,7,0.4), (2,6,0.5), (1,7,0.6)

]

The maximum cuts were found:

Init: 0.2

Final spins: [1 -1 1 1 -1 -1 -1 1]

Cut value: 8.300

Init: 0.8

Final spins: [-1 1 -1 -1 1 -1 -1 1]

Cut value: 7.700

Init: 1.2

Final spins: [1 -1 1 1 -1 -1 -1 1]

Cut value: 8.300

III. Conclusion

In this project, we implemented and evaluated a continuous Hopfield network in two distinct contexts: associative memory and combinatorial optimization.

For the pattern storage task, the network weights were constructed using Hebbian learning from three binary patterns. The simulations showed that when initialized near a stored pattern—even with added noise—the network dynamics reliably converged to the clean attractor. In several cases, when the initialization was already close to the true attractor, the initial and final states were identical, confirming that the stored patterns are indeed stable fixed points of the energy landscape. When higher levels of noise were added, the trajectories and energy plots illustrated the network's ability to denoise and recover the original pattern.

For the Max-Cut problem, we encoded the graph's adjacency into a Hopfield weight matrix with W = -A Multiple runs from different random initial states converged to feasible partitions of the graph. The final solutions corresponded to distinct local minima of the energy, with cut values that, while slightly variable, consistently reflected strong partitioning of the graph's heaviest edges. This illustrates both the promise and the limitation of the Hopfield approach: it naturally approximates NP-hard problems by descending the energy surface, but convergence depends on initialization and may yield local optima rather than the global maximum cut.

Overall, the experiments confirm the nature of Hopfield networks they demonstrate stable attractors and error correction, and as heuristic solvers, they provide a biologically inspired way to approach hard optimization tasks.

References

[1] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proceedings of the National Academy of Sciences of the United States of America, vol. 79, no. 8, pp. 2554–2558, 1982, doi: 10.1073/pnas.79.8.2554.

[2] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," Proceedings of the National Academy of Sciences of the United States of America, vol. 81, no. 10, pp. 3088–3092, 1984, doi: 10.1073/pnas.81.10.3088.

[3] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," Biological Cybernetics, vol. 52, no. 3, pp. 141–152, 1985, doi: 10.1007/BF00339943.

[4]