

# Hopfield Networks: Associative Memory and Energy Minimization Explained

Jack Levy — Engineering Science, University of Toronto

October 8, 2025

## Contents

<a href="#">1</a>	<a href="#">Introduction</a>	<a href="#">2</a>
<a href="#">2</a>	<a href="#">The Energy Function</a>	<a href="#">2</a>
<a href="#">3</a>	<a href="#">Python Implementation Explained</a>	<a href="#">3</a>
<a href="#">4</a>	<a href="#">Relation to Biological Memory</a>	<a href="#">4</a>
<a href="#">5</a>	<a href="#">Applications</a>	<a href="#">5</a>
<a href="#">6</a>	<a href="#">Summary</a>	<a href="#">5</a>

# 1 Introduction

A **Hopfield network** is a recurrent neural network that stores information in a distributed manner and retrieves complete memories from partial or noisy inputs by minimizing an **energy function**.

Invented by John Hopfield in 1982, it models how biological neural circuits might store and recall patterns through the dynamics of mutual neuron interactions.

Key ideas:

- Neurons are binary:  $x_i \in \{-1, +1\}$
- Connections are symmetric:  $w_{ij} = w_{ji}$
- The network's dynamics evolve toward stable low-energy states (memories)

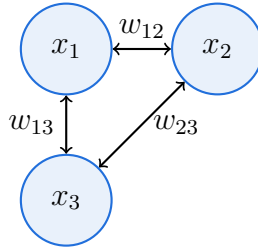


Figure 1: A simple 3-neuron Hopfield network with symmetric weights.

## 2 The Energy Function

The Hopfield network defines an energy function:

$$E = -\frac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j$$

Each neuron configuration corresponds to a point in the energy landscape. The network evolves by flipping neuron states to reduce total energy.

The minima of  $E$  correspond to the **stored memories** or **attractor states** of the network.

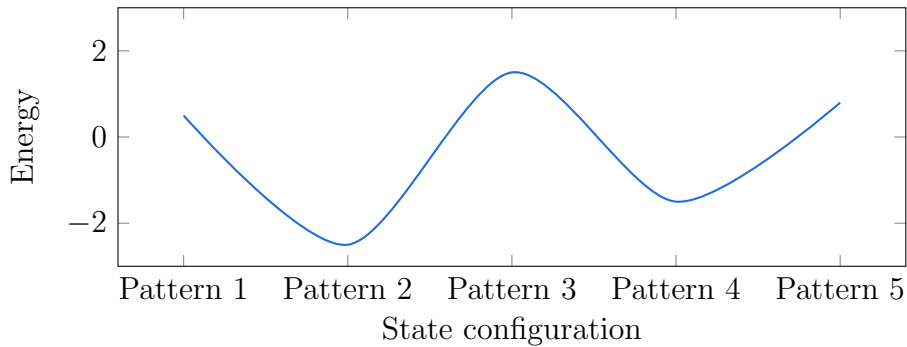


Figure 2: Conceptual energy landscape with stable attractors.

### 3 Python Implementation Explained

Below is the Hopfield energy and minima-retrieval code from class.

Listing 1: Python code for computing Hopfield energy minima

```
def energy(x, s):
    # Compute the total energy for a given state vector x
    res = 0
    for i in range(len(x)):
        for j in range(len(x)):
            if s[i][j] is not None:
                res += s[i][j] * x[i] * x[j]
    return -res

def retrieve_minima(s):
    # Evaluate all possible neuron configurations (2^3 = 8)
    cur_min = energy([-1, -1, -1], s)
    cur_loc_min = [[-1, -1, -1]]
    for x0 in [-1, 1]:
        for x1 in [-1, 1]:
            for x2 in [-1, 1]:
                x = [x0, x1, x2]
                if x == [-1, -1, -1]:
                    continue
                cur_energy = energy(x, s)
                print("x:", x, "current energy:", cur_energy)
                if cur_energy < cur_min:
                    cur_min = cur_energy
                    cur_loc_min = [x]
                elif cur_energy == cur_min:
                    cur_loc_min.append(x)
    return cur_loc_min

def loc_of_min(L):
    '''Return the location of the smallest element in L'''
    cur_loc_min = 0
    cur_min = L[0]
    for i in range(1, len(L)):
        if L[i] < cur_min:
            cur_min = L[i]
            cur_loc_min = i
    return cur_loc_min

s = [[None, 2, -1],
      [None, None, 1],
      [None, None, None]]

print(retrieve_minima(s))
```

#### Explanation

**Step 1.** The energy function calculates  $E = -\sum_{i,j} s_{ij}x_ix_j$ . If neurons  $x_i$  and  $x_j$  are aligned (both +1 or both 1) and  $s_{ij}$  is positive, the energy decreases — reinforcing stable associations.

**Step 2.** The `retrieve_minima` function enumerates all possible binary patterns of three

neurons, computes their energy, and identifies the configurations with the lowest energy.

**Step 3.** The `loc_of_min` helper simply returns the index of the smallest element in a list.

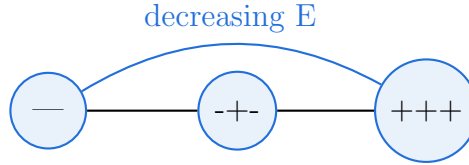


Figure 3: State transitions as the network moves toward energy minima.

The output reveals which configurations correspond to the most stable memories.

For the weight matrix:

$$s = \begin{bmatrix} - & 2 & -1 \\ - & - & 1 \\ - & - & - \end{bmatrix}$$

the minima occur at  $[1, 1, 1]$  and  $[-1, -1, -1]$ , corresponding to coherent memory states.

## 4 Relation to Biological Memory

Hopfield networks are a mathematical abstraction of how neural assemblies in the brain may collectively encode and recall information.

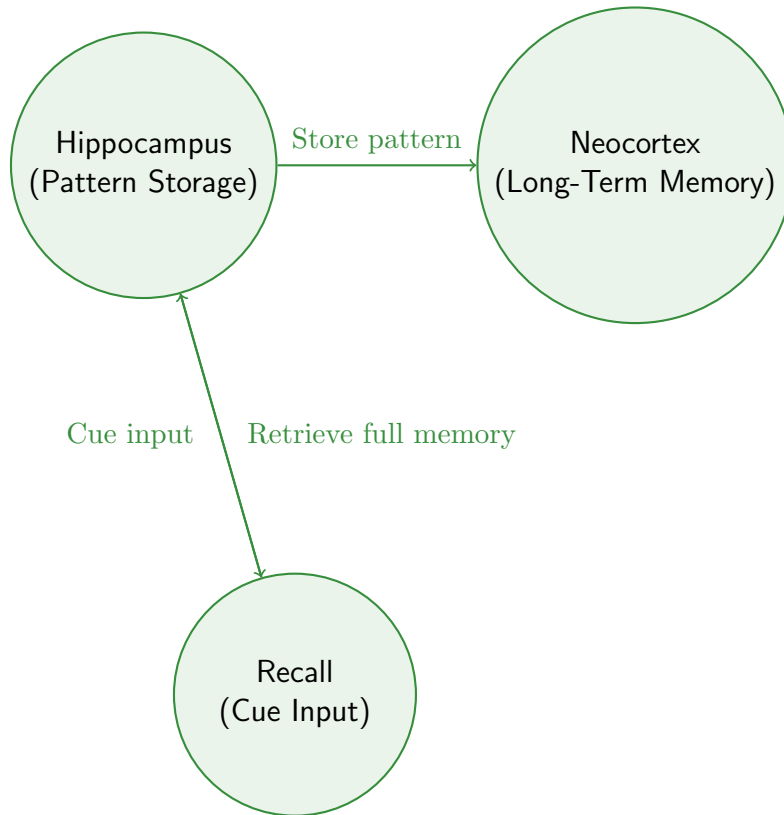


Figure 4: Biological analogy of associative memory: the hippocampus encodes patterns, transfers them to the neocortex for long-term storage, and retrieves complete memories from cues.

In the brain:

- **Neurons** correspond to units in the Hopfield model.
- **Synapses** correspond to weights  $w_{ij}$ .
- **Memory recall** emerges through attractor dynamics in cortical circuits.

Unlike biological neurons, Hopfield networks use binary activations and lack temporal dynamics or spiking behavior. They capture only a simplified associative principle.

## 5 Applications

Hopfield networks are primarily used for:

- **Associative memory retrieval** (pattern completion)
- **Error correction** (recovering noisy inputs)
- **Optimization problems** — by mapping cost functions to energy

Modern variants (e.g., *Modern Hopfield Networks*, 2020) show that transformer attention mechanisms can be derived as continuous Hopfield dynamics, linking these classical models to deep learning.

## 6 Summary

### Key Takeaways

- Hopfield networks store memories as stable energy minima.
- Dynamics evolve naturally toward these minima (pattern recall).
- The Python code simulates a 3-neuron system exploring its energy landscape.
- Biological analogy: collective neural activity in the hippocampus and cortex.