HOPFIELD NETWORK HYPERPARAMETER STUDY (PLAIN-LANGUAGE NOTES)

WHAT WE WANT TO KNOW

- How the code boosts or shrinks the strength of the outside input (‘w_external‘).
Outside input = the â pushâ coming from stimuli.
- How the code boosts or shrinks the internal feedback (‘w_s‘).
Internal feedback = neurons reinforcing each other to stay in a memory.
- Where random noise enters the maths.
Noise = random wiggles that can shake the system into a new state.
- Where we could plug in controls for these three knobs and later add scoring.

Line numbers below refer to the files in this repository snapshot.

WHERE THE IMPORTANT PIECES LIVE

MEMORY GENERATOR (‘HNN_GEN‘)

‘HNN_Gen.HNN‘ (‘HNN_Gen.py:12-37‘) builds the stored patterns (â memoriesâ) and sets the initial neuron activity. The network size is ‘self.N = 2**6‘ (64 neurons). The number of memories is ‘self.P = 10‘. ‘self.eps‘ sets the size of the random kick around each stored memory. Building the Hebbian matrix (‘mems @ memsÆµ‘) sets the baseline internal feedback.

STOCHASTIC INTEGRATOR (‘EUL_MAY.EM‘)

‘EM.Eu_Ma_Test‘ (‘Eul_May.py:37-81‘) steps the activity forward in time using the Eulerâ Maruyama method.
Eulerâ Maruyama = a recipe for simulating noisy differential equations.
Each step combines:
- Recurrent drive ‘np.dot(W, z)‘ where ‘z = tanh(delta * y)‘ (‘Eul_May.py:87-107‘).
‘tanh‘ squashes values between â 1 and 1 so the neurons saturate gently.
- Leak ‘-y‘, which pulls neurons back toward zero.
- External input ‘u‘, included when we pick additive mode (‘C == ’A’‘).
- Noise term ‘sigma * random‘, where ‘sigma‘ controls the noise strength.

Noise enters through the Wiener increment (‘Eul_May.py:79‘).
Wiener increment = the small random jump that models Brownian motion.

EXPERIMENT DRIVER (‘MAINHNN_SHB‘)

‘MainHNN_SHB‘ (‘MainHNN_SHB.py:40-125‘) sets time span, generates memories, and builds the external input.
‘u_proto‘ (‘MainHNN_SHB.py:62-84‘) contains amplitudes for each memory and each stimulus slot. The diagonal entries (2.0â 3.5) push the intended memory; small off-diagonal terms (0.05â 0.2) create gentle cross-talk. Combining ‘u_proto‘ with the memory patterns yields

‘u_tran‘, the actual input sent to the integrator. The list ‘sigma = [0.5]‘ sets noise levels for repeats.

AUTAPSE TOY MODEL (‘PARETO/FIGURES/FIG_MODEL.PY‘)

This self-feedback toy system introduces clear knobs ‘w_in‘ (external weight) and ‘w_self‘ (internal weight) (‘pareto/figures/fig_model.py:33-68‘).
Autapse = a neuron that talks to itself, used here as a simple test bed.
The update rule:

[CODE BLOCK]

```
x â  x + dt * (-x - leak + w_in * h + w_self * activation(x)) + â dt * sigma_n * random
```

[CODE BLOCK]

It then sweeps these knobs to show a trade-off: stronger input speeds up responses but can shorten memory (‘pareto/figures/fig_model.py:183-369‘).

HOW THE THREE KNOBS APPEAR

EXTERNAL INPUT STRENGTH (‘W_EXTERNAL‘)

- Right now the strength lives inside ‘u_proto‘ (‘MainHNN_SHB.py:62-84‘). There is no single scalar knob; each stimulus draw sets its own value.
- ‘EM.hop_field_test‘ adds the vector ‘u‘ directly (‘Eul_May.py:107‘). To mirror Danâ s ‘w_external‘, we can wrap it as ‘w_external * u‘.
- The autapse model already exposes ‘w_in‘ (‘pareto/figures/fig_model.py:33-46‘), and later plots assume the total input weight is ‘w_self + w_in‘.

Takeaway: add a global multiplier, either when computing ‘u_tran‘ or inside ‘hop_field_test‘, so experiments can scale outside input with one parameter.

INTERNAL COUPLING STRENGTH (‘W_S‘)

- The recurrent drive comes from the Hebbian matrix ‘W = (1/N) * M â (mems @ memsÆµ)‘ (‘Eul_May.py:104-105‘).
Hebbian = neurons that fire together wire together; this matrix encodes that rule.
- A new knob ‘w_s‘ could multiply ‘np.dot(W, z)‘ to deepen or flatten the attractor wells.
Attractor well = a stable state the system tends to fall into.
- In the autapse toy, ‘w_self‘ already plays this role.

NOISE LEVEL (‘SIGMA‘)

- ‘sigma‘ passes through every call to ‘Eu_Ma_Test‘ and multiplies the random part of the step (‘Eul_May.py:79‘).
- Adjusting the ‘sigma‘ list in ‘MainHNN_SHB‘ lets you run several noise settings back to back.

HOW THESE KNOBS INTERACT

1. **External input vs. stability:** Larger diagonal entries in ‘u_proto‘ make the network snap faster to the target memory, but can overpower the internal matrix if cross-talk is too big. The autapse sweeps show the same story: higher ‘w_in‘ raises responsiveness but shrinks memory duration (‘pareto/figures/fig_model.py:183-260‘).
2. **Internal gain vs. flexibility:** A bigger ‘w_s‘ would make attractor wells deeper, giving longer recall but slower switching. Autapse plots confirm that higher ‘w_self‘ stretches memory but slows response (‘pareto/figures/fig_model.py:307-365‘).
3. **Noise vs. control:** ‘sigma‘ helps the system leave a memory when the input changes, but too much noise destroys stored patterns. Sweeping ‘sigma‘ will highlight where the model shifts from â stuckâ to â chaoticâ .

PRACTICAL SUGGESTIONS

1. Add explicit knobs ‘w_external‘ and ‘w_s‘ in ‘EM.hop_field_test‘ so every experiment can tune them without changing random seeds or inputs.
2. When building ‘u_proto‘, multiply by the global ‘w_external‘ so different runs stay comparable.
3. Borrow the simple metrics from the autapse scripts (response rate, memory length, reaction time) and apply them to the overlap traces produced in ‘HNPlot‘.

LINK TO THE SCIENCE ADVANCES PAPER

The repository follows *Stimulus-Driven Dynamics for Robust Memory Retrieval in Hopfield Networks* (Science Advances, doi:10.1126/sciadv.adu6991). The paper discusses balancing quick stimulus-driven retrieval against stable memory storage. Our code reflects that balance: stimuli (‘MainHNN_SHB.py:62-125‘) drive switching, recurrent connections and noise (‘Eul_May.py:79-107‘) control how long memories stick. Putting the ‘w_external‘, ‘w_s‘, and ‘sigma‘ knobs on the surface will let us explore the same trade-off in a controlled way.

NEXT STEPS

- Wire in the new scaling knobs and expose them through configuration so sweeps are simple.
- Reuse the autapse scoring logic to measure responsiveness and memory retention directly on the Hopfield outputs.

- Run structured sweeps (for example, a grid over ‘w_external‘ and ‘w_s‘) and log results into the time-stamped folders in ‘outputs/‘.