Winter 2023 COEN 6331 Assignment Report (III)

Jun Huang, Student ID: 40168167
Electrical and Computer Engineering, Concordia University
Email: jun.huang@concordia.ca

Abstract

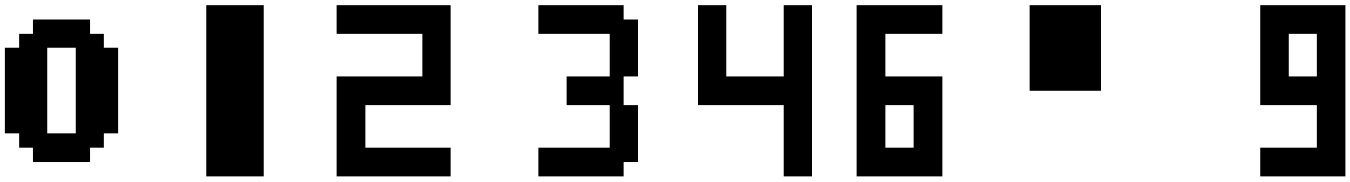
This report presents the experiments and results of building a hopfield network to recall eight given patterns from the corrupted inputs. At first, we compare the performance of the different update approaches of the hopfield network, namely, the synchronous and asynchronous update on three different levels of corrupted samples. The result shows that the synchronous update performs better than the asynchronous approaches, while they both reveal the same poor accuracy on certain patterns. We then explore the solution for avoiding the misrecalled other patterns or spurious attractors from the existing literature. One solution can distinguish the spurious attractors from the actual attractors by calculating the energy profile for each pixel. The result shows that it can avoid spurious attractors to a certain extent but can not prevent 'ghost' recalls or miscalled with other patterns. While the other solution, the discrete modern hopfield network, does an excellent job and achieves almost 100% for every pattern without any spurious attractors.

Index Terms

Associative Memory, Hopfield Network, Spurious Attractors Rectification, Discrete Modern Hopfield Network

I. Problem Description

The problem for this assignment is to build a hopfield network for a pattern association problem. The target patterns are shown at Fig. 1. Every pattern is a 12×10 matrix consists with elements in bipolar manner $\{-1, 1\}$. Based on these patterns, a set of corrupted samples is generated with a certain amount



(a) Pattern '0' (b) Pattern '1' (c) Pattern '2' (d) Pattern '3' (e) Pattern '4' (f) Pattern '6' (g) Pattern '■' (h) Pattern '9'

Fig. 1: Eight target patterns

of pixels flipped. The task of the hopfield network is to associate the corrupted sample with its original pattern and further analyze the performance of the association.

II. Clssical Hopfield Network

In this report, we first explore the classical hopfield network [1]. We build the network with Hebbian learning rule on the target patterns. Then we conduct experiments on the network with three sets of corrupted samples, with respectively, 15%, 20%, and 25% of randomly flipped pixels. Additionally, we conduct each set of corrupted samples on two different update modes (synchronous and asynchronous) of the classical hopfield network. We observe the experiment result on recalling the pattern from the corrupted sample and [TODO]

A. Building the Network

We construct the weighted matrix for the classical hopfield network with the Hebbian learning rule. We first flattern the target pattern into a vector $p_j =< p_{1,j}, p_{2,j}, \ldots, p_{d,j} >$ with $d = 12 \times 10 = 120$ dimentions. And then we put all eight patterns' flattern vectors into one matrix $P = [p_1, p_2, \ldots, p_8]^T$ and apply the Eq. 1 to implement the Hebbian rule. The second line of the Eq. 1 is for zero out the diagonal of the weight matrix for the stability of the network. The weight matrix is shown in Fig. 2.

$$\begin{aligned} W &= PP^T \\ W &= W - diag(W) \end{aligned} \tag{1}$$

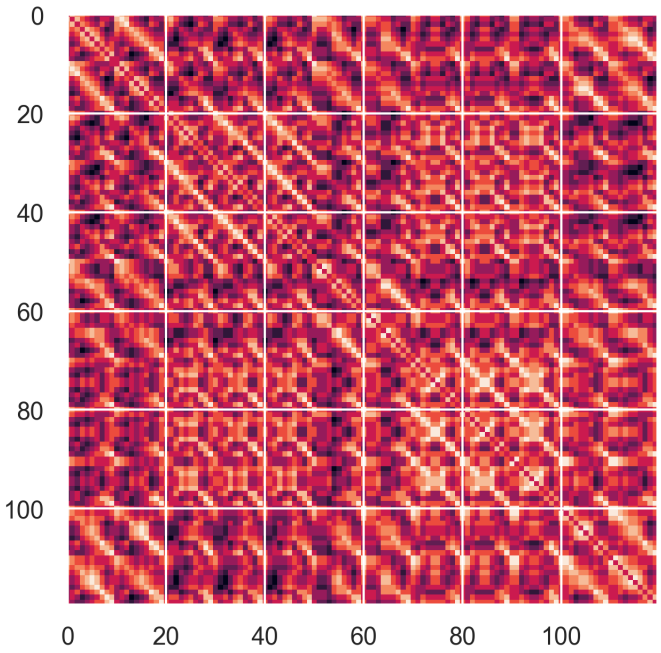


Fig. 2: Weight matrix of for the eight target patterns built by Hebbian learning rule.

B. Recall Process

Generally, the given input $X = [x_1, x_2, \ldots x_d]$ to the network is considered a network's input state where x_i represent the i^{th} pixel of the input. The initial input can be represented as $X(0) = [x_1(0), x_2(0), \ldots x_d(0)]$. The network continuously update the state with $X(t+1) = f(net_{(t)})$ where t is the current iteration, $f(\cdot)$ is the activation function, and $net_{(t)}$ is the product of the weight matrix W and the last state $X(t)$. For the hopfield network represented by bipolar value, the activation function should be the sign function $sgn(\cdot)$. The update can be represented by Eq. 2 where B is the bias vector which we consider as zeros in this report.

$$X(t+1) = sgn(WX(t) - B) \tag{2}$$

Then for each state and its next state, the network calculates its energy difference with the energy function of the two states. The recall process ends when the energy stops changing and is considered the network's convergence. We can represent the energy function bt Eq.3.

$$E = -\frac{1}{2}X^T WX - X^T B \tag{3}$$

There are two approaches to recalling the pattern from corrupted samples: synchronous update and asynchronous update. The synchronous approach updates the state for all pixels at once from $X(t)$ to $X(t+1)$. Hence every pixel's $t+1$ state is calculated by state t, while the asynchronous approach updates the state pixel by pixel with either a given order or random order. The update for i^{th} pixel with the given order affects the update for the $(i+1)^{th}$ pixel. When the last pixel is updated, the input's current shape is considered state $t+1$. The two approaches are represented by Algorithm 1 and 2.

Algorithm 1: Synchronous Update

input: X
output: X_{recall}
while *True* **do**
  /* current energy */
  $e_t \leftarrow \text{energy}(X)$
  /* update */
  $X \leftarrow \text{update}(W, X)$
  /* new energy */
  $e_{t+1} \leftarrow \text{energy}(X)$
  if $(e_{t+1} - e_t)$ is equal to 0 **then**
    break
$X_{recall} \leftarrow X$

Algorithm 2: Asynchronous Update

input: X
output: X_{recall}
while *True* **do**
  /* current energy */
  $e_t \leftarrow \text{energy}(X)$
  /* update */
  for i in randomIndex(X) **do**
    $x[i] \leftarrow \text{updatePixel}(i, W[i], X)$
  /* new energy */
  $e_{t+1} \leftarrow \text{energy}(X)$
  if $(e_{t+1} - e_t)$ is equal to 0 **then**
    break
$X_{recall} \leftarrow X$

C. Image Recall Experiments

To evaluation the hopfield network for both update approaches, we generate three sets of corrupted samples from the target pattern with 15%, 20%, and 25% pixels randomly flipped. We collect the result for the six experiments and measure the following metrics:

- Recall Accuracy: how many corrupted samples are recalled correctly?
- Average Iterations of the Convergence: how many iterations a corrupted sample needs for convergence on average?
- Percentage of Mismatched Recalls: how many mismatched attractors occur (including misrecall with other patterns and the spurious attractor)?
- Number of each Mismatched Recalls: what's the number of each mismatched attractor?
- Time of Convergence: how much time does the convergence need?

We generate 10,000 corrupted samples for each pattern and collect the metrics defined above. Table. I, II,and III show the metrics on the three levels of corrupted samples. Fig.3, Fig. 4, Fig.5, Fig. 6, Fig.7, and Fig. 8 show the top-3 spurious attractor of each pattern using the synchronous and asynchronous update on the three levels of corrupted samples.

D. Result Analysis

By observing the three tables and regarding the performance of different update approaches, the synchronous update is slightly better than the asynchronous update as their accuracy difference is within 2%. On the other hand, pattern '2', '■', and '9' perform worse than other patterns as their accuracies are lower than 80% and the others are almost 100%. Moreover, when the number of noise increases, the performance of patterns that are hard to recall drops dramatically. Regarding the time performance, the update algorithm is the only factor according to the three tables. Finally, we can learn from the table that

even though asynchronous needs more time to recall the patterns, its performance keeps aligning with the synchronous update. Furthermore, its average number of iterations of the convergence is smaller than the synchronous update. This might indicate that the energy function performs consistently no matter what approach to update the state of the network.

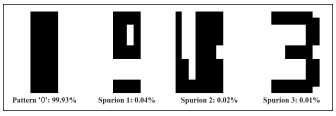
Observing the figures presenting the top-3 spurious attractors of each experiment, pattern ‘3’ has a higher chance of being misrecalled by the corrupted samples of pattern ‘1’, ‘4’, and ‘9’. Patterns ‘2’ and ‘6’ are easily misrecalled by each others’ corrupted samples. This indicates that the network is hard to distinguish amount these patterns. This misrecalled with the other attractor of the network might lead to unintentional issues when the user of the network actually do not know the corrupted samples original pattern. Because the network recalls a valid attractor for the corrupted samples but the spurious attractor, the user will easily accept the result while we know it is a mistake. On the other hand, patterns ‘2’ and ‘9’ have a high amount of a similar spurious attractor. This might indicate that the learned weight needs to memorize these patterns more clearly.

TABLE I: Evaluation metircs for hopfield network with 15% corruption on each sample

Pattern	% of Accuracy		AVG. of Iterations		% of Mismatched		Time (Seconds)	
	Sync.	Async.	Sync.	Async.	Sync.	Async.	Sync.	Async.
0	**100.0%**	99.99%	2.227	2.09	0.0%	0.01%	5.07	19.876
1	**99.93%**	99.86%	2.066	2.02	0.07%	0.14%	3.37	16.06
2	56.18%	**56.27%**	2.1948	2.10	43.82%	43.73%	5.76	16.33
3	100.00%	100.00%	2.3481	2.14	0.0%	0.0%	3.62	16.6
4	**99.94%**	99.81%	2.609	2.39	0.06%	0.19%	4.09	18.97
6	100.00%	100.00%	2.164	2.06	0.0%	0.0%	3.22	16.93
■	**91.44%**	89.08%	2.668	2.34	8.56%	10.92%	3.84	18.89
9	**51.57%**	48.02%	2.567	2.45	48.43%	51.98%	3.61	19.83

For accuracy, the bold number is the better one. Sync. indicates the synchronous update. Async. indicates the asynchronous update.

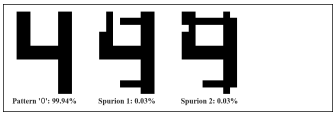
(a) Pattern ‘0’



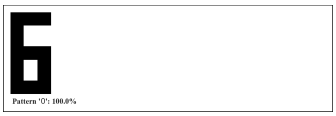
(b) Pattern ‘1’

(c) Pattern ‘2’

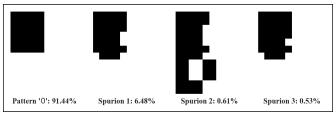
(d) Pattern ‘3’



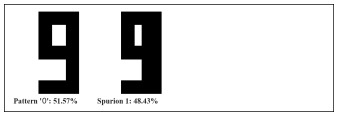
(e) Pattern ‘4’



(f) Pattern ‘6’



(g) Pattern ‘■’

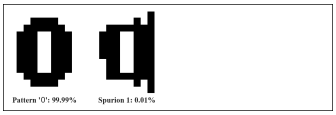


(h) Pattern ‘9’

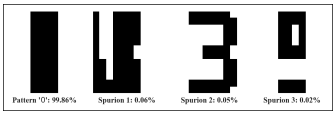
Fig. 3: Top=3 Spurious attractors of the **synchronously** updated hopfield network on 10,000 with 15% corrupted samples

III. Spurious Attractors Rectification

The bad performance of the hopfield motivates us to search for a method to reduce the spurious attractors. We could brute force compare the stable state with every pixel pattern by pixel, which is



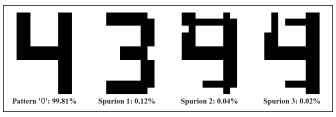
(a) Pattern ‘0’



(b) Pattern ‘1’

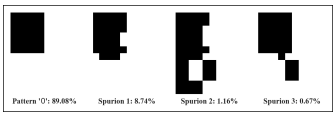
(c) Pattern ‘2’

(d) Pattern ‘3’

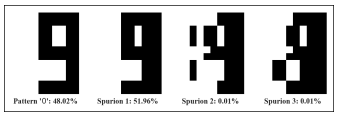


(e) Pattern ‘4’

(f) Pattern ‘6’



(g) Pattern ‘■’



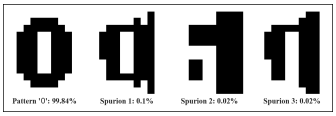
(h) Pattern ‘9’

Fig. 4: Top=3 Spurious attractors of the **asynchronously** updated hopfield network on 10,000 with 15% corrupted samples

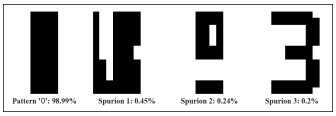
TABLE II: Evaluation metircs for hopfield network with 20% corruption on each sample

Pattern	% of Accuracy		AVG. of Iterations		% of Mismatched		Time (Seconds)	
	Sync.	Async.	Sync.	Async.	Sync.	Async.	Sync.	Async.
0	**99.84%**	99.48%	2.43	2.22	0.16%	0.52%	3.95	17.27
1	**98.99%**	98.86%	2.27	2.10	1.01%	1.14%	3.76	16.14
2	**51.91%**	49.47%	2.37	2.19	48.09%	50.53%	3.815	16.45
3	**100.00%**	99.97%	2.59	2.31	0.0%	0.03%	3.99	18.40
4	**99.37%**	98.68%	2.86	2.51	0.63%	1.32%	4.11	19.12
6	**99.99%**	99.94%	2.35	2.16	0.01%	0.06%	3.96	17.46
■	**82.15%**	79.85%	2.93	2.48	17.85%	20.15%	4.84	19.22
9	**45.55%**	42.16%	2.50	2.52	54.45%	57.84%	4.20	18.11

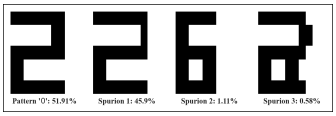
For accuracy, the bold number is the better one. Sync. indicates the synchronous update. Async. indicates the asynchronous update.



(a) Pattern ‘0’

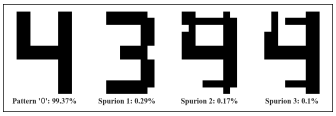


(b) Pattern ‘1’

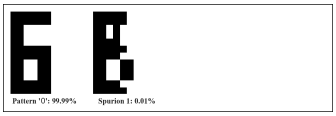


(c) Pattern ‘2’

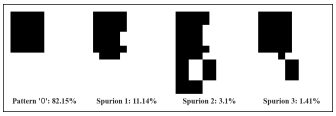
(d) Pattern ‘3’



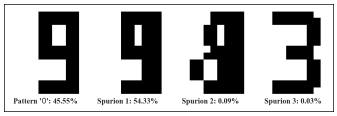
(e) Pattern ‘4’



(f) Pattern ‘6’



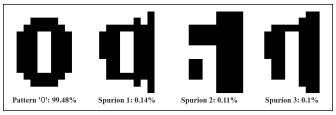
(g) Pattern ‘■’



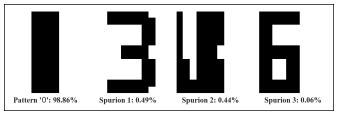
(h) Pattern ‘9’

Fig. 5: Top=3 Spurious attractors of the **synchronously** updated hopfield network on 10,000 with 20% corrupted samples

extremely expensive. One existing method [2] introduces an ‘energy ratio’ method to distinguish the pattern from most spurious attractors. The paper utilizes the local energy [3] for each state pixel and sorts those local energies as the ‘energy profile’ of the current state. It then calculates the ‘energy ratio’ from the profile by dividing the sum of the lowest 10% local energies by the sum of the highest 10% local energies. Any ‘energy ratio’ smaller than the lowest ‘energy ratio’ of the learned patterns is considered

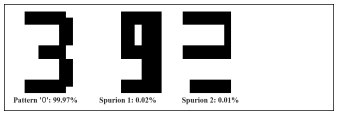


(a) Pattern ‘0’

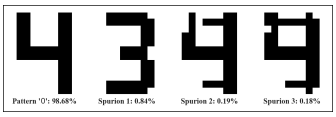


(b) Pattern ‘1’

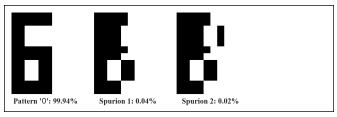
(c) Pattern ‘2’



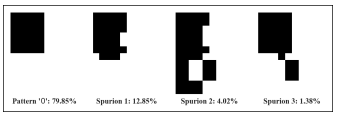
(d) Pattern ‘3’



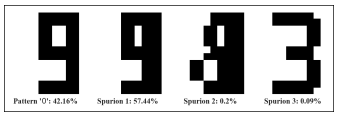
(e) Pattern ‘4’



(f) Pattern ‘6’



(g) Pattern ‘■’



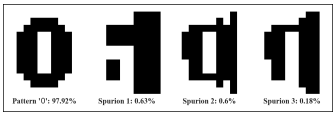
(h) Pattern ‘9’

Fig. 6: Top=3 Spurious attractors of the **asynchronously** updated hopfield network on 10,000 with 20% corrupted samples

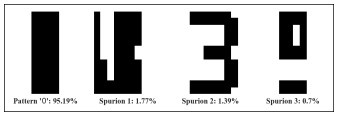
TABLE III: Evaluation metircs for hopfield network with 25% corruption on each sample

Pattern	**% of Accuracy**		**AVG. of Iterations**		**% of Mismatched**		**Time (Seconds)**	
	Sync.	Async.	Sync.	Async.	Sync.	Async.	Sync.	Async.
0	**97.92%**	96.48%	2.84	2.42	2.08%	3.52%	4.99	17.42
1	**95.19%**	93.42%	2.67	2.34	4.81%	6.58%	4.54	16.79
2	**43.87%**	42.50%	2.69	2.36	56.13%	57.5%	4.78	16.84
3	**99.80%**	99.47%	2.93	2.51	0.2%	0.53%	5.08	18.01
4	**96.62%**	95.01%	3.25	2.72	3.38%	4.99%	5.40	20.12
6	**99.76%**	99.28%	2.64	2.31	0.24%	0.72%	4.71	16.90
■	**68.56%**	67.73%	3.26	2.66	31.44%	32.27%	5.69	19.04
9	**39.82%**	38.32%	2.93	2.61	60.18%	61.68%	4.56	20.78

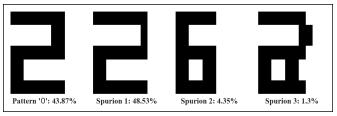
For accuracy, the bold number is the better one. Sync. indicates the synchronous update. Async. indicates the asynchronous update.



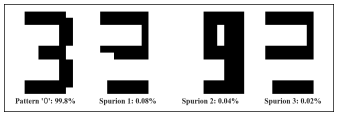
(a) Pattern ‘0’



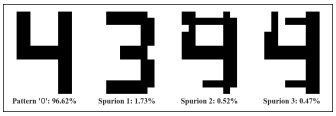
(b) Pattern ‘1’



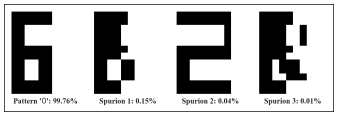
(c) Pattern ‘2’



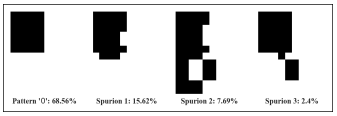
(d) Pattern ‘3’



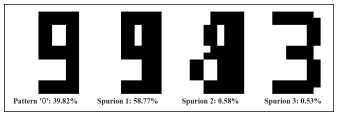
(e) Pattern ‘4’



(f) Pattern ‘6’



(g) Pattern ‘■’

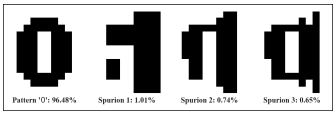


(h) Pattern ‘9’

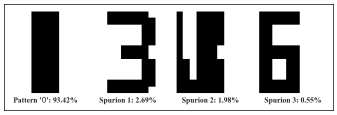
Fig. 7: Top=3 Spurious attractors of the **synchronously** updated hopfield network on 10,000 with 25% corrupted samples

a spurious attractor. In this way, we could search through every stable state without comparing pixel by pixel with the learned patterns.

We construct the rectified asynchronous update based on the original asynchronous update. Before any update, we record the input as the original state. After the network finds a stable state after the asynchronous update, we compare the current state’s ’energy ratio’ and the patterns’ lowest ‘energy ratio.’

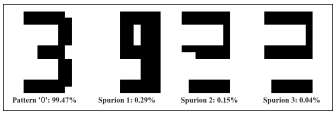


(a) Pattern ‘0’

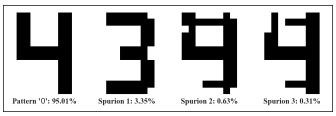


(b) Pattern ‘1’

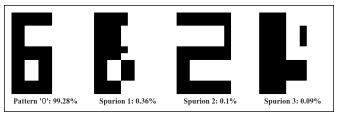
(c) Pattern ‘2’



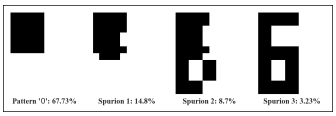
(d) Pattern ‘3’



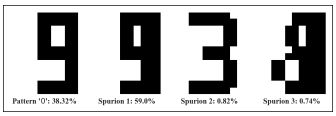
(e) Pattern ‘4’



(f) Pattern ‘6’



(g) Pattern ‘■’



(h) Pattern ‘9’

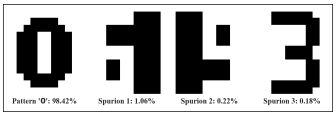
Fig. 8: Top=3 Spurious attractors of the **asynchronously** updated hopfield network on 10,000 with 25% corrupted samples

If the current state’s ‘energy ratio’ is lower than any ‘energy ratio’ of the patterns, we reset the current state to the original state and start the update again. Since every asynchronous update iteration uses different update order, it will not fall into the same basin of attraction. That’s why this approach can not be applied on the synchronous update, as the basin of attraction for the given input is fixed if we do not introduce the randomness bias term.

A. Result Analysis

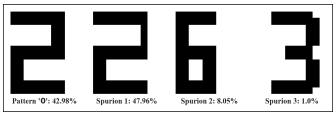
Based on such rectified asynchronous updating, we further conduct experiments on the 10,000 samples with 25% noise on each sample. Table. IV compares the rectified asynchronous update to the synchronous one. Fig. 9 presents the top-3 spurious attractor of each pattern of the rectified asynchronous update hopfield network.

From Table. IV, we observe that the rectified asynchronous update performs better than the synchronous update. Particularly, for pattern ‘■’ and ‘9’, the accuracies have significant growth with more iterations. This indicates that we can use more iteration to find the pattern even when we fall into the basin of spurious attraction without comparing the pixels. On the other hand, from Fig. 9, we observe that even though the rectified asynchronous update reduces most of the spurious attractors, it can not distinguish between the original patterns and the other patterns or the ‘ghost’ pattern of other patterns, which is their flipped pattern. For example, for pattern ‘0’, there are 1.06% of samples misrecalled to the ghost pattern of ‘6’ and ‘9’.

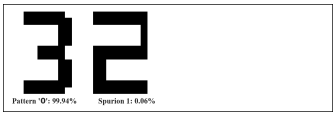


(a) Pattern ‘0’

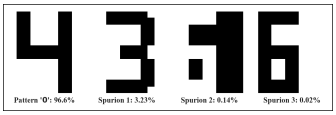
(b) Pattern ‘1’



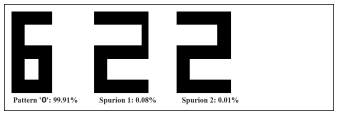
(c) Pattern ‘2’



(d) Pattern ‘3’

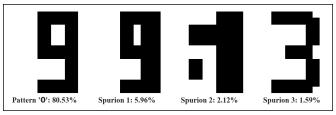


(e) Pattern ‘4’



(f) Pattern ‘6’

(g) Pattern ‘■’



(h) Pattern ‘9’

Fig. 9: Top=3 Spurious attractors of the **rectified asynchronously** updated hopfield network on 10,000 with 25% corrupted samples

TABLE IV: Evaluation metircs for rectified hopfield network with 25% corruption on each sample compared with the normal hopfield network

Pattern	% of Accuracy		AVG. of Iterations		% of Mismatched		Time (Seconds)	
	Rectified	Sync.	Rectified	Sync.	Rectified	Sync.	Rectified	Sync.
0	**98.42%**	97.92%	2.51	2.84	1.58%	2.08%	24.13	4.49
1	**96.14%**	95.19%	2.55	2.67	3.86%	4.81%	24.51	4.54
2	42.98%	**43.87%**	2.47	2.69	57.02%	56.13%	24.07	4.78
3	**99.94%**	99.80%	2.54	2.93	0.06%	0.2%	24.79	5.08
4	**96.60%**	96.62%	2.80	3.25	3.4%	3.38%	26.84	5.40
6	**99.91%**	99.76%	2.34	2.64	0.09%	0.24%	22.95	4.71
■	**91.49%**	68.56%	6.07	3.26	8.51%	31.44%	56.31	5.69
9	**80.53%**	39.82%	15.22	2.93	19.47%	60.18%	144.23	4.56

For accuracy, the bold number is the better one. Rectified indicated the recitified asynchronous update. Sync. indicates the synchronous update.

IV. Discrete Modern Hopfield Network

In 2016, the creator of the Hopfield Network, John J. Hopfield propose a paper called 'Dense Associative Memory for Pattern Recognition [4]' presents new energy function and update function for associative learning based on the classical hopfield network. In the discrete modern hopfield network, it no longer needs the learned weight matrix but focus on the patterns' attention durining each asynchronous update. It propose a new energy function as Eq.4 is shown, where p_i is the i^{th} pattern's vector and $X(t)$ is the current input state.

$$E(X(t)) = -\sum_{i=1}^{n} exp(p_i^T X(t)) \tag{4}$$

What is more, the modern hopfield network only works on asynchronous updates. It guarantees that the update will no longer need multiple iterations and converges the network with one asynchronous update for all pixels. The update function for the i^{th} pixel is done by Eq.5, where $\xi^{(i+,t)}$ indicates the current state but only set the value on i^{th} pixel to 1 and $\xi^{(i-,t)}$ indicates set the value on i^{th} pixel to -1.

$$x_{i,(t+1)} = sgn[E(\xi^{(i-,t)}) - E(\xi^{(i+,t)})] \tag{5}$$

This update function changes the i^{th} pixel to where it decreases the overall energy. It can also be interpreted as the state only updating to its flipped value when the energy drops because of the flipped state. This interpretation is more intuitive and can be described by Eq. 6.

$$x_{i,(t+1)} = \begin{cases} flip(x_{i,t}) & \text{if } E(\xi^{(flip(i),t)}) - E(\xi^{(i,t)}) < 0 \\ x_{i,t} & \text{else} \end{cases} \tag{6}$$

We experiment with the discrete modern hopfield network on the 10,000 samples with 25% corruption. The result shows that, for patterns '0', '1', '4', and '■', discrete modern achieve a 100% of accuracy. For pattern '2', 0.05% samples are misrecalled to pattern '6'. For pattern '3', 0.01% samples are misrecalled to pattern '4'. For pattern '6', 0.01% samples are misrecalled to pattern '2'. And for '9', 0.01% samples are misrecalled to pattern '3'. All in all, the result for discrete modern hopfield network is very accurate. None of the recalls ever fall into the basin of spurious attraction.

V. Conclusion

In this paper, we explore and perform the experiment on the classical hopfield network in synchronous or asynchronous updates. The result shows that the synchronous update performs better than the asynchronous update, while both have low accuracy on the same patterns. We further explore the solution for reducing the spurious attractors. We observe that the 'energy profile' and 'energy ratio methods for distinguishing the spurious and actual attractors can prevent the network from staying in the spurious attractors while leading to the ghost recall. Finally, we explore the discrete modern hopfield network and observe almost 100% accuracy for every pattern and no spurious attraction compared with the normal hopfield network.

References

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[2] A. V. Robins and S. J. McCallum, "A robust method for distinguishing between learned and spurious attractors," *Neural Networks*, vol. 17, no. 3, pp. 313–326, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608003003009

[3] J. A. Hertz, *Introduction to the theory of neural computation*. CRC Press, 2018.

[4] D. Krotov and J. J. Hopfield, "Dense associative memory for pattern recognition," *CoRR*, vol. abs/1606.01164, 2016. [Online]. Available: http://arxiv.org/abs/1606.01164