Learning Sequence Attractors in Hopfield Networks with Hidden Neurons

Yao Lu **Si Wu**
Peking University
{yao.lu,siwu}@pku.edu.cn

Abstract

The brain is targeted for processing temporal sequence information. It remains largely unclear how the brain learns to store and retrieve sequence memories. Here, we study how networks of Hopfield type learn sequence attractors to store predefined pattern sequences and retrieve them robustly. We show that to store arbitrary pattern sequences, it is necessary for the network to include hidden neurons even though their role in displaying sequence memories is indirect. We develop a local learning algorithm to learn sequence attractors in the networks with hidden neurons. The algorithm is proven to converge and lead to sequence attractors. We demonstrate that our model can store and retrieve sequences robustly on synthetic and real-world datasets. We hope that this study provides new insights in understanding sequence memory and temporal information processing in the brain.

1 Introduction

The brain is targeted for processing temporal sequence information but computational modeling study on this issue lags far behind that on static information processing. Attractor neural networks are promising computational models for elucidating the mechanisms of the brain representing, memorizing and processing information [Amari, 1972; Hopfield, 1982; Amit, 1989]. By considering simplified neuron model and threshold dynamics, the classical Hopfield network has successfully elucidated how a recurrent neural network learns to store static memory patterns [Hopfield, 1982]. However, the classical Hopfield network and related works typically only consider static attractors, which can not explain the sequential neural activities widely observed in the brain (e.g., in memory retrieval in the hippocampus). In this paper, we follow and extend the standard form of the Hopfield model by considering binary neurons and threshold dynamics, as this enables us to pursue theoretical analysis, and we investigate how a recurrent neural network learns to store sequence attractors.

2 Limitation of Classical Hopfield Networks

We first consider the classical Hopfield networks of N visible binary neurons [Amari, 1972; Hopfield, 1982]. All the neurons are bidirectionally connected and their weight matrix is $\mathbf{W}$ of which W_{ij} denotes the synaptic weight from the j-th neuron to the i-th neuron. Let $\boldsymbol{\xi}(t) = (\xi_1(t), ..., \xi_N(t)) \in \{-1, 1\}^N$ be the states of the neurons at time t. These states are synchronously updated according to the threshold dynamics, for $i = 1, ..., N$,

$$\xi_i(t+1) = \text{sign}\Big(\sum_{j=1}^{N} W_{ij}\xi_j(t) - \theta_i\Big) = \text{sign}\Big(\sum_{j=0}^{N} W_{ij}\xi_j(t)\Big), \tag{1}$$

where $\text{sign}(x) = 1$ if $x > 0$ and $\text{sign}(x) = 0$ otherwise. Hereafter, the threshold parameter θ_i is omitted as it can be absorbed by adding an extra neuron $\xi_0(t) = 1$ and $W_{i0} = -\theta_i$. Given a pair of

Associative Memory & Hopfield Networks in 2023. NeurIPS 2023 workshop.

successive network states $\boldsymbol{\xi}(t)$ and $\boldsymbol{\xi}(t+1)$, the dynamics of the recurrent network can be unfolded in time and viewed as a feedforward network, in which each output neuron is a perceptron of the inputs.

Given a sequence in the form of $\mathbf{x}(1), ..., \mathbf{x}(T) \in \{-1, 1\}^N$, one can use a learning algorithm to adjust $\mathbf{W}$ such that the evolving of the network state matches the pattern sequence. Although recurrent networks without hidden neurons can generate some sequences of maximal length 2^N [Muscinelli et al., 2017], they are fundamentally limited in the class of sequences that can be generated. Since each neuron can be regarded as a perceptron, the condition that sequence $\mathbf{x}(1), ..., \mathbf{x}(T)$ can be generated by the network is, for each i, the dataset $\{(\mathbf{x}(t), x_i(t+1))\}_{t=1}^{T-1}$ is linearly separable [LeCun, 1986; Bressloff & Taylor, 1992; Brea et al., 2013; Muscinelli et al., 2017].

3 Hopfield Networks with Hidden Neurons

To overcome the limitation of the classical Hopfield networks, we consider a group of hidden neurons in the network, in addition to visible ones. The visible and hidden neurons are bidirectionally connected, and there is no intra-connection within visible neurons or hidden neurons. Let $\mathbf{U}$ be the weight matrix from visible neurons to hidden neurons, of which U_{ij} denotes the synaptic weight from the j-th visible neuron to the i-th hidden neuron, and $\mathbf{V}$ be the weight matrix from hidden neurons to visible neurons, of which V_{ji} denotes the synaptic weight from the i-th hidden neuron to the j-th visible neuron. Let $\boldsymbol{\xi}(t) = (\xi_1(t), ..., \xi_N(t)) \in \{-1, 1\}^N$ be the states of visible neurons and $\boldsymbol{\zeta}(t) = (\zeta_1(t), ..., \zeta_M(t)) \in \{-1, 1\}^M$ be the states of hidden neurons at time t. These states are synchronously updated according to, for $i = 1, ..., M$ and $j = 1, ..., N$,

$$\zeta_i(t) = \text{sign}\Big(\sum_{k=1}^{N} U_{ik}\xi_k(t)\Big), \tag{2}$$

$$\xi_j(t+1) = \text{sign}\Big(\sum_{k=1}^{M} V_{jk}\zeta_k(t)\Big), \tag{3}$$

where we omit the threshold parameters as they can be absorbed into the equations. As illustrated in Figure 1, given a pair of successive network states $\boldsymbol{\xi}(t)$ and $\boldsymbol{\xi}(t+1)$, the dynamics of the network can be unfolded in time and viewed as a feedforward network with a hidden layer of neurons.

The networks of M hidden neurons can generate arbitrary sequences with Markov property and of length at least M, as stated in Theorem 1. We provide a constructive proof based on one-hot encoding by the hidden neurons in the Appendix.

Theorem 1 *Let $\mathbf{x}(1), ..., \mathbf{x}(T) \in \{-1, 1\}^N$ such that $\mathbf{x}(i) \neq \mathbf{x}(j)$ for $i \neq j$ except that $\mathbf{x}(1) = \mathbf{x}(T)$. Then $\mathbf{x}(1), ..., \mathbf{x}(T)$ can be generated by the network defined in (2)(3) for $M = T - 1$.*

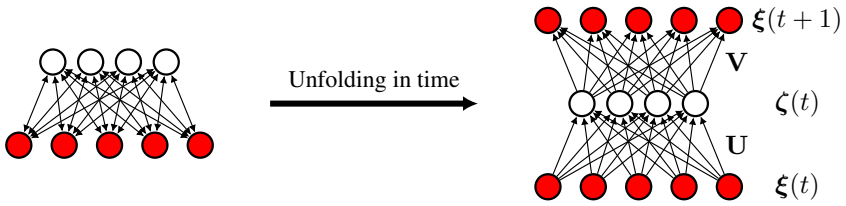


Figure 1: A recurrent network with hidden neurons. The red circles denote visible neurons and the white circles denote hidden neurons.

4 Learning

To learn the weight matrices, one can first unfold the Hopfield network with hidden neurons in time such that it becomes a feedforward network with a hidden layer, and the pairs of successive patterns in the sequence constitute the training examples. However, learning in the unfolded feedforward network is difficult since the backpropagation algorithm cannot be applied as the neurons are not differentiable.

We propose a new learning algorithm to learn the weight matrices in the unfolded feedforward networks, which draws inspirations from three ideas: feedback alignment [Lillicrap et al., 2016], target propagation [LeCun, 1987; Bengio, 2014; Litwin-Kumar et al., 2017] and three-factor rules [Frémaux & Gerstner, 2016; Kuśmierz et al., 2017]. As in feedback alignment, it requires a random matrix $\mathbf{P}$, which is fixed during the learning process, to backpropagate signals. As in target propagation, it does not propagate errors but targets to create surrogate targets for the hidden neurons. Each weight parameter is updated by a three-factor rule, in which the presynaptic activation, the postsynaptic activation and an error term as neuromodulation are multiplied. The three-factor rule is similar to the one for the classical Hopfield networks [Bressloff & Taylor, 1992] and known as margin perceptron in the machine learning literature [Collobert & Bengio, 2004].

The algorithm works as follows. Given a pair of successive patterns $\mathbf{x}(t)$ and $\mathbf{x}(t+1)$, for $i = 1, ..., M$ and $j = 1, ..., N$ in parallel,

1. Update $\mathbf{U}$ by

$$z_i(t+1) = \text{sign}\Big(\sum_{k=1}^{N} P_{ik} x_k(t+1)\Big), \tag{4}$$

$$\mu_i(t) = H\Big(\kappa - z_i(t+1)\sum_{k=1}^{N} U_{ik} x_k(t)\Big), \tag{5}$$

$$U_{ij} \leftarrow U_{ij} + \eta \mu_i(t) z_i(t+1) x_j(t). \tag{6}$$

2. Update $\mathbf{V}$ by

$$y_i(t) = \text{sign}\Big(\sum_{k=1}^{N} U_{ik} x_k(t)\Big), \tag{7}$$

$$\nu_j(t) = H\Big(\kappa - x_j(t+1)\sum_{k=1}^{M} V_{jk} y_k(t)\Big), \tag{8}$$

$$V_{ji} \leftarrow V_{ji} + \eta \nu_j(t) x_j(t+1) y_i(t), \tag{9}$$

where P_{ik} denotes the (i,k) entry of the fixed random matrix $\mathbf{P}$, $H(\cdot)$ is the Heaviside function ($H(x) = 1$ if $x > 0$ and $H(x) = 0$ otherwise), $\kappa > 0$ is the robustness hyperparameter and $\eta > 0$ is the learning rate hyperparameter. $\mu_i(t)$ and $\nu_j(t)$ can be interpreted as the error terms for the hidden and the visible neurons, respectively. $z_i(t+1)$ can be interpreted as the synaptic input from an external neuron. The above procedure is then repeated for each t.

4.1 Analysis

In this section, we provide theoretical analysis of the algorithm. The proofs are left to the Appendix. First, we provide convergence guarantee of the algorithm.

Theorem 2 *Given the definitions in (4)(5)(7)(8), for all i, j and t, if a solution exists such that $\mu_i(t) = 0$ and $\nu_j(t) = 0$, then the algorithm (4)-(9) converges in finite steps given U_{ij} and V_{ji} are initialized to zero.*

Next, we show the algorithm can reduce error $\mu_i(t)$ for a single step of updating $\mathbf{U}$. The theorem can be trivially extended for $\nu_j(t)$ and $\mathbf{V}$ by a similar proof.

Theorem 3 *Given the definitions in (4)(5), let*

$$U'_{ik} = U_{ik} + \eta \mu_i(t) z_i(t+1) x_k(t), \tag{10}$$

$$\mu'_i(t) = H\Big(\kappa - z_i(t+1)\sum_{k=1}^{N} U'_{ik} x_k(t)\Big). \tag{11}$$

Then $\mu'_i(t) = 0$ for sufficiently large $\eta > 0$.

To understand why reducing the errors $\mu_i(t)$ and $\nu_j(t)$ leads to sequence attractors, we present the following result.

Theorem 4 *Given the definitions in (7)(8), let* $\hat{\mathbf{y}}(t) = (\hat{y}_1(t), ..., \hat{y}_M(t)) \in \{-1, 1\}^M$ *such that* $\sum_k |\hat{y}_k(t) - y_k(t)| < \epsilon$. *If* $\nu_j(t) = 0$ *and*

$$\epsilon \cdot \max_k |V_{jk}| < \kappa, \tag{12}$$

then

$$x_j(t+1) = \text{sign}\Big(\sum_{k=1}^{M} V_{jk}\hat{y}_k(t)\Big). \tag{13}$$

Theorem 4 shows that when the errors are zero, given perturbed hidden neuron states $\hat{\mathbf{y}}(t)$, we have $\mathbf{x}(t+1) = \text{sign}(\mathbf{V}\hat{\mathbf{y}}(t))$. The result can be trivially extended to show that given perturbed visible neuron states $\hat{\mathbf{x}}(t)$ we have $\mathbf{y}(t) = \text{sign}(\mathbf{U}\hat{\mathbf{x}}(t))$ by a similar proof. Therefore, the network can generate sequence $\mathbf{x}(1), ..., \mathbf{x}(T)$ as an attractor. From Theorem 4, we can also see that κ acts as the robustness hyperparameter as it controls the level of perturbation ϵ for inequality (12) to hold.

We show experiments in the Appendix.

References

Shun-ichi Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on Computers*, 1972.

Daniel J Amit. *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, 1989.

Yoshua Bengio. How auto-encoders could provide credit assignment in deep networks via target propagation. *arXiv preprint arXiv:1407.7906*, 2014.

Johanni Brea, Walter Senn, and Jean-Pascal Pfister. Matching recall and storage in sequence learning with spiking neural networks. *Journal of Neuroscience*, 2013.

Paul C Bressloff and John G Taylor. Perceptron-like learning in time-summating neural networks. *Journal of Physics A: Mathematical and General*, 1992.

Ronan Collobert and Samy Bengio. Links between perceptrons, mlps and svms. *International Conference on Machine learning*, 2004.

Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, 2016.

Elizabeth Gardner. The space of interactions in neural network models. *Journal of Physics A: Mathematical and General*, 1988.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 1982.

Haruyuki Iwama, Mayu Okumura, Yasushi Makihara, and Yasushi Yagi. The ou-isir gait database comprising the large population dataset and performance evaluation of gait recognition. *IEEE Transactions on Information Forensics and Security*, 2012.

Łukasz Kuśmierz, Takuya Isomura, and Taro Toyoizumi. Learning with three factors: modulating hebbian plasticity with errors. *Current Opinion in Neurobiology*, 2017.

Yann LeCun. Learning process in an asymmetric threshold network. *Disordered Systems and Biological Organization*, 1986.

Yann LeCun. *Modeles connexionnistes de lapprentissage*. PhD thesis, These de Doctorat, Universite Paris, 1987.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 2016.

Ashok Litwin-Kumar, Kameron Decker Harris, Richard Axel, Haim Sompolinsky, and LF Abbott. Optimal degrees of synaptic connectivity. *Neuron*, 2017.

Marvin Minsky and A Papert Seymour. *Perceptrons*. MIT Press, 1969.

Samuel P Muscinelli, Wulfram Gerstner, and Johanni Brea. Exponentially long orbits in hopfield neural networks. *Neural Computation*, 2017.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. *International Conference on Machine Learning*, 2015.

A Experiments

We run experiments on synthetic and real-world sequence datasets for Hopfield networks with hidden neurons by the algorithm proposed in the previous section to learn sequence attractors. All the experiments are carried out in MATLAB and PyTorch. In all the experiments, each weight parameter of $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{P}$ is sampled i.i.d. from Gaussian distribution with mean zero and variance 1×10^{-6}, learning rate $\eta = 1 \times 10^{-3}$ and robustness $\kappa = 1$. In each experiment, we run the algorithm for 500 epochs. In each epoch, the algorithm runs on $(\mathbf{x}(t), \mathbf{x}(t+1))$ from the start to the end of each sequence. No noise is added during learning. Noise is added only at retrieval.

A.1 Toy Examples

In Figure 2, we show examples of sequences that cannot be generated by the network. The sequences are synthetically constructed. We then test if the perceptron learning algorithm can learn the sequences. Since the algorithm converges if the linear separability condition is met [Minsky & Seymour, 1969], the divergence of the algorithm implies that the sequences cannot be generated by the networks.

To show Hopfield networks with hidden neurons can overcome the limitation of classical Hopfield networks, we conduct experiments on the examples in Figure 2. We construct a network of visible neurons $N = 10$ and hidden neurons $M = 50$ for each example. After learning, we test the robustness of the networks in retrieval by adding two salt-and-pepper noises (flipping the states of two out of ten neurons) to the first pattern of a sequence and set it to be the initial network state. The results are shown in Figure 3, from which we can see that the networks with hidden neurons can generate sequences which cannot be generated by the classical Hopfield networks and retrieve them robustly under moderate level of noise.

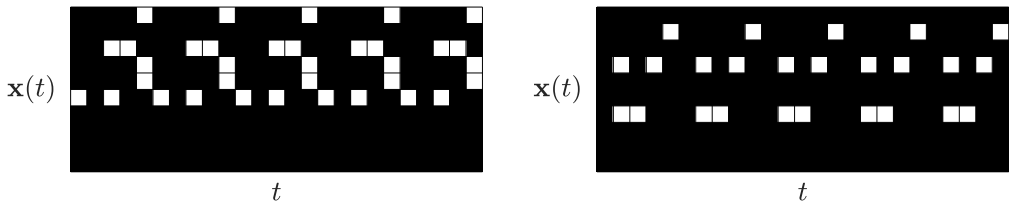


Figure 2: Two example sequences which cannot be generated by the classical Hopfield networks. White squares denote positive ones and black squares denote negative ones.

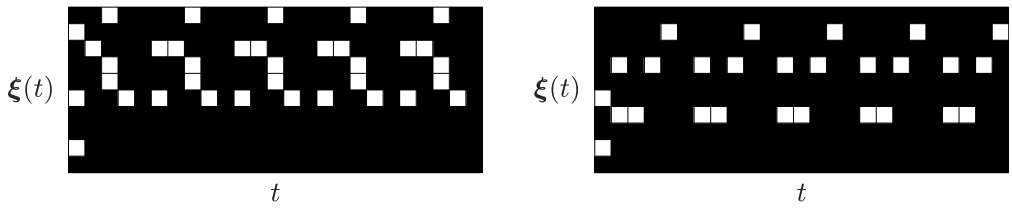
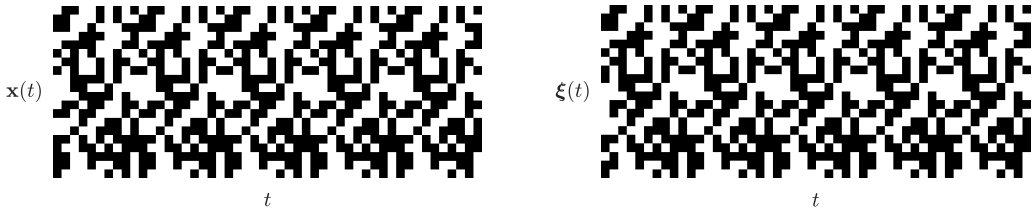


Figure 3: Hopfield networks with hidden neurons can generate the two sequences in Figure 2, which cannot be generated without hidden neurons, despite noisy initial states. Note that in the first column of each diagram two salt-and-pepper noises are added to test the robustness of the retrieval.

A.2 Random Sequences

We generate periodic sequences of random patterns $\mathbf{x}(1), ..., \mathbf{x}(T) \in \{-1, 1\}^N$. In each sequence, $\mathbf{x}(i) \neq \mathbf{x}(j)$ for $i \neq j$ except that $\mathbf{x}(1) = \mathbf{x}(T)$ for the periodicity. We set $N = 100$ and vary period length T. We sample each $\mathbf{x}(t)$ independently from the uniform distribution of $\{-1, 1\}^N$ for $t = 1, ..., T-1$ and then resample it if it is identical to a previous pattern. Finally, we set $\mathbf{x}(T) = \mathbf{x}(1)$. For each random sequence, we construct a network with hidden neurons and apply the proposed learning algorithm. To evaluate the effectiveness of the learning algorithm, we compare learning only $\mathbf{V}$ (with $\mathbf{U}$ fixed during learning) and learning both $\mathbf{U}$ and $\mathbf{V}$. Once the learning is done, we test if the network can retrieve the sequence robustly given perturbed $\mathbf{x}(1)$ with 10 salt-and-pepper noises as the initial network state $\boldsymbol{\xi}(1)$. We define that the retrieval is successful if $\boldsymbol{\xi}(\tau + t) = \mathbf{x}(t)$ for some τ and all $t = 1, ..., T$. We run 100 trials for each T or M setting and count the successful retrievals.

In Figure 4, we show the visualization of a result. In Figure 5, we show the results with various period lengths T for $M = 500$. In Figure 6, we show the results with various numbers of hidden neurons M for $T = 70$. We can see learning both $\mathbf{U}$ and $\mathbf{V}$ is more effective than learning only $\mathbf{V}$. However, in both cases, the algorithm fails for large T, even if we increase the number of hidden neurons, which might be due to the suboptimality of the algorithm.



(a) Ground truth (b) Network states

Figure 4: Learning random periodic sequences in Hopfield networks with hidden neurons. Only the first 20 neurons of 100 visible neurons are selected for visualization due to space limitation. Note that in the first column of (b) salt-and-pepper noises are added to test the robustness of the retrieval.

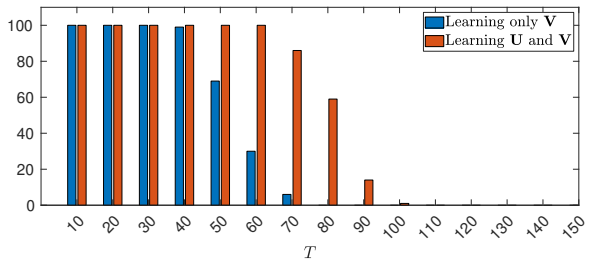


Figure 5: Successful retrievals out of 100 trials with different sequence period lengths.

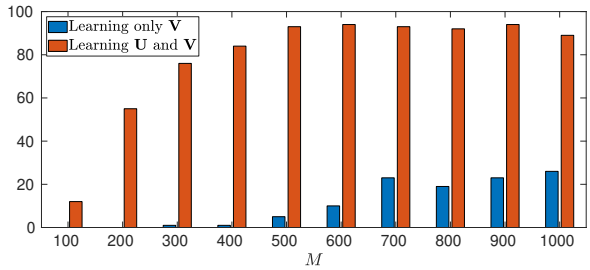


Figure 6: Successful retrievals out of 100 trials with different numbers of hidden neurons.

A.3 Real-World Sequences

We test Hopfield networks with hidden neurons by our algorithm in learning real-world sequences on a silhouette sequence dataset (OU-ISIR gait database large population [Iwama et al., 2012]) and a handwriting sequence dataset (Moving MNIST [Srivastava et al., 2015]). The patterns in the sequences are rather correlated since adjacent image frames are similar. To adopt the datasets for the networks to learn, we convert the image intensity values to ± 1. For the silhouette dataset, we use a network with hidden neuron number $M = 200$ to learn a single image sequence of length 103, in which each image has size 88×128. The images are flatten to vectors of size $88 \times 128 = 11264$. For the handwriting dataset, we use a network with hidden neuron number $M = 1000$ to learn 20 image sequences of length 20, in which each image has size 64×64. The images are flatten to vectors of size $64 \times 64 = 4096$. In Figure 7 and 8, we show the visualization results of the learned networks for robust retrieval, in which the first image of a sequence is corrupted and set to be the initial state of a network. In Figure 9, we show the average errors $\frac{1}{M}\sum_t \sum_i \mu_i(t)$ and $\frac{1}{N}\sum_t \sum_j \nu_j(t)$ during the learning process, from which we can see that both errors reduce to zero.

(a) Ground truth $\mathbf{x}(t)$

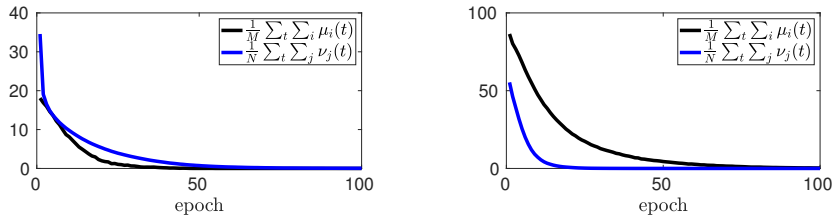
(b) Network states $\boldsymbol{\xi}(t)$

Figure 7: Retrieval of sequences under noise on the silhouette sequence dataset. An image sequence of length 136 is learned. Each image has size 88×128. In (a) and (b), $\mathbf{x}(t)$ and $\boldsymbol{\xi}(t)$ are shown respectively for $t = 1, ..., 10$. In (b), 2000 salt-and-pepper noises are added to the first image. The corrupted image is set to be the initial state of the network.

(a) Ground truth $\mathbf{x}(t)$

(b) Network states $\boldsymbol{\xi}(t)$

Figure 8: Retrieval of sequences under noise on the handwriting sequence dataset. 20 image sequences of length 20 are learned. Due to space limitation, only one image sequence is displayed in here. Each image has size 64×64. In (a) and (b), $\mathbf{x}(t)$ and $\boldsymbol{\xi}(t)$ are shown respectively for $t = 1, ..., 8$. In (b), 300 salt-and-pepper noises are added to the first image. The corrupted image is set to be the initial state of the network.



(a) Silhouette (b) Handwriting

Figure 9: Errors during learning. $\frac{1}{M}\sum_t\sum_i \mu_i(t)$ is the average error for the hidden neurons. $\frac{1}{N}\sum_t\sum_j \nu_j(t)$ is the average error for the visible neurons.

B Proof of Theorem 1

We construct a network such that, given $\boldsymbol{\xi}(t) = \mathbf{x}(i)$ for $i = 1, ..., T-1$, the hidden neurons provide an one-hot encoding of the successive pattern $\mathbf{x}(i+1)$, which is then decoded to be $\boldsymbol{\xi}(t+1)$.

To store $\mathbf{x}(1), ..., \mathbf{x}(T) \in \{-1, 1\}^N$ in (2)(3), assuming $\mathbf{x}(i) \neq \mathbf{x}(j)$ for $i \neq j$ except that $\mathbf{x}(1) = \mathbf{x}(T)$, let $M = T - 1$ and construct weight matrix $\mathbf{U}$ as

$$\mathbf{U} = (\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(T-1))^\top \tag{14}$$

and hidden neurons $\boldsymbol{\zeta}(t) = (\zeta_1(t), ..., \zeta_M(t))$ as

$$\zeta_i(t) = \text{sign}\Big(\sum_{k=1}^{N} U_{ik}\xi_k(t) - N\Big) \tag{15}$$

$$= \text{sign}\big(\mathbf{x}(i)^\top \boldsymbol{\xi}(t) - N\big) \tag{16}$$

such that given $\boldsymbol{\xi}(t) = \mathbf{x}(i)$ for $i = 1, ..., T-1$, we have

$$\zeta_j(t) = \begin{cases} +1, & \text{if } j = i, \\ -1, & \text{otherwise.} \end{cases} \tag{17}$$

Next, we construct the weight matrix $\mathbf{V}$ as

$$\mathbf{V} = (\mathbf{x}(2), \mathbf{x}(3), ..., \mathbf{x}(T)) \tag{18}$$

and visible neurons $\boldsymbol{\xi}(t+1) = (\xi_1(t+1), ..., \xi_N(t+1))$ as

$$\boldsymbol{\xi}(t+1) = \text{sign}(\mathbf{V}\boldsymbol{\zeta}(t) + \boldsymbol{\theta}) \tag{19}$$

where $\boldsymbol{\theta} = \sum_{j=2}^{T} \mathbf{x}(j)$ such that given the one-hot vector $\boldsymbol{\zeta}(t)$ we have

$$\boldsymbol{\xi}(t+1) = \text{sign}\Big(\mathbf{x}(i+1) - \sum_{j \neq i+1} \mathbf{x}(j) + \sum_{j=2}^{T} \mathbf{x}(j)\Big) \tag{20}$$

$$= \text{sign}(2 \cdot \mathbf{x}(i+1)) \tag{21}$$

$$= \mathbf{x}(i+1) \tag{22}$$

C Proof of Theorem 2

Note that the update of $\mathbf{U}$ (4)(5)(6) in Section 5 does not depend on $\mathbf{V}$. Therefore, we first prove the convergence of updating $\mathbf{U}$ for $\eta > 0$ and $\kappa > 0$. The proof follows from [Gardner, 1988]. Assume $\mathbf{U}^*$ exists such that, for all t and i,

$$z_i(t+1) \sum_k U^*_{ik} x_k(t) \geq \kappa. \tag{23}$$

Define the p-th update of $\mathbf{U}$ with $\mu_i(t_p) = 1$ by

$$U_{ij}^{(p+1)} = U_{ij}^{(p)} + \eta z_i(t_p + 1)x_j(t_p) \tag{24}$$

for some $t_p \in \{1, ..., T-1\}$ and all j in parallel. We assume zero-initialization, that is, $U_{ij}^{(1)} = 0$ for simplicity but the result holds if $|U_{ij}^{(1)}|$ is sufficiently small. Let

$$X_i^{(p+1)} = \frac{\sum_j U_{ij}^{(p+1)} U_{ij}^*}{\sqrt{\sum_j \left(U_{ij}^{(p+1)}\right)^2}\sqrt{\sum_j \left(U_{ij}^*\right)^2}}. \tag{25}$$

The Cauchy-Schwarz inequality, we have $X_i^{(p+1)} \leq 1$. Now we prove the convergence of updating $\mathbf{U}$ by contradiction. Assuming the update of $\mathbf{U}$ does not converge, we will show that $X_i^{(p+1)} > 1$ as $p \to \infty$. First, we have

$$\sum_j U_{ij}^{(p+1)} U_{ij}^* - \sum_j U_{ij}^{(p)} U_{ij}^* = \eta \sum_j z_i(t_p + 1) U_{ij}^* x_j(t_p) \geq \eta\kappa \tag{26}$$

due to (1) and therefore

$$\sum_j U_{ij}^{(p+1)} U_{ij}^* = \sum_j U_{ij}^{(p+1)} U_{ij}^* - \sum_j U_{ij}^{(p)} U_{ij}^* + ... + \sum_j U_{ij}^{(2)} U_{ij}^* - \sum_j U_{ij}^{(1)} U_{ij}^* + \sum_j U_{ij}^{(1)} U_{ij}^* \tag{27}$$

$$\geq \eta\kappa p \tag{28}$$

since we assumed $U_{ij}^{(1)} = 0$. Next, we have

$$\sum_j \left(U_{ij}^{(p+1)}\right)^2 - \sum_j \left(U_{ij}^{(p)}\right)^2 = \sum_j \left(U_{ij}^{(p)} + \eta z_i(t_p + 1)x_j(t_p)\right)^2 - \sum_j \left(U_{ij}^{(p)}\right)^2 \tag{29}$$

$$= 2\eta \sum_j U_{ij}^{(p)} z_i(t_p + 1)x_j(t_p) + N\eta^2 \tag{30}$$

$$= 2\eta z_i(t_p + 1) \sum_j U_{ij}^{(p)} x_j(t_p) + N\eta^2 \tag{31}$$

$$< 2\eta\kappa + N\eta^2 \tag{32}$$

since we assumed $\mu_i(t_p) = 1$ and therefore $z_i(t_p + 1) \sum_j U_{ij}^{(p)} x_j(t_p) < \kappa$. Then, we have

$$\sqrt{\sum_j (U_{ij}^{(p+1)})^2} - \sqrt{\sum_j (U_{ij}^{(p)})^2} \tag{33}$$

$$= \Big(\sum_j \left(U_{ij}^{(p+1)}\right)^2 - \sum_j \left(U_{ij}^{(p)}\right)^2\Big) \Big/ \Big(\sqrt{\sum_j \left(U_{ij}^{(p+1)}\right)^2} + \sqrt{\sum_j \left(U_{ij}^{(p)}\right)^2}\Big) \tag{34}$$

$$< (2\eta\kappa + N\eta^2) \Big/ \Big(\sqrt{\sum_j \left(U_{ij}^{(p+1)}\right)^2} + \sqrt{\sum_j \left(U_{ij}^{(p)}\right)^2}\Big). \tag{35}$$

By Cauchy-Schwarz inequality, we have

$$\sqrt{\sum_j (U_{ij}^{(p+1)})^2}\sqrt{\sum_j (U_{ij}^*)^2} \geq \sum_j U_{ij}^{(p+1)} U_{ij}^* \geq \eta\kappa p \tag{36}$$

and therefore

$$\sqrt{\sum_j (U_{ij}^{(p+1)})^2} \geq \frac{\eta\kappa p}{\sqrt{\sum_j (U_{ij}^*)^2}}. \tag{37}$$

Also,

$$\sqrt{\sum_j (U_{ij}^{(p+1)})^2} = \sqrt{\sum_j (U_{ij}^{(p+1)})^2} - \sqrt{\sum_j (U_{ij}^{(p)})^2} + ... + \sqrt{\sum_j (U_{ij}^{(2)})^2} - \sqrt{\sum_j (U_{ij}^{(1)})^2} \quad (38)$$

$$+ \sqrt{\sum_j (U_{ij}^{(1)})^2} \quad (39)$$

$$< \sum_{q=1}^{p} (2\eta\kappa + N\eta^2) \Big/ \Big(\sqrt{\sum_j \big(U_{ij}^{(q+1)}\big)^2} + \sqrt{\sum_j \big(U_{ij}^{(q)}\big)^2} \Big) \quad (40)$$

$$< \sum_{q=1}^{p} (2\eta\kappa + N\eta^2) \sqrt{\sum_j (U_{ij}^{*})^2} \frac{1}{\eta\kappa(2q-1)} \quad (41)$$

$$= \frac{\eta\kappa + N\eta^2/2}{\eta\kappa} \sqrt{\sum_j (U_{ij}^{*})^2} \sum_{q=1}^{p} \frac{1}{q-1/2} \quad (42)$$

Note that for $q > 1$

$$\frac{1}{q-1/2} \leq \int_{q-3/2}^{q-1/2} \frac{1}{x} dx = \log(q-1/2) - \log(q-3/2) \quad (43)$$

and

$$\sum_{q=1}^{p} \frac{1}{q-1/2} = \frac{1}{2} + \sum_{q=2}^{p} \frac{1}{q-1/2} \leq \frac{1}{2} + \int_{1/2}^{p-1/2} \frac{1}{x} dx = 2 + \log(p-1/2) - \log(1/2). \quad (44)$$

Therefore,

$$\sqrt{\sum_j (U_{ij}^{(p+1)})^2} = O(\log(p)) \quad (45)$$

and

$$\sum_j U_{ij}^{(p+1)} U_{ij}^{*} = \Omega(p) \quad (46)$$

as $p \to \infty$. We have,

$$X_i^{(p+1)} = \frac{\sum_j U_{ij}^{(p+1)} U_{ij}^{*}}{\sqrt{\sum_j (U_{ij}^{(p+1)})^2} \sqrt{\sum_j (U_{ij}^{*})^2}} > 1 \quad (47)$$

for some p. This contradicts that $X_i^{(p+1)} \leq 1$. Thus, the updating $\mathbf{U}$ converges.

Upon the convergence of updating $\mathbf{U}$, we can prove the convergence of $\mathbf{V}$ if there exists $\mathbf{V}^*$ such that for all t and i,

$$x_i(t+1) \sum_k V_{ik}^{*} y_k(t) \geq \kappa \quad (48)$$

by a similar proof.

D Proof of Theorem 3

If $\mu_i(t) = 0$, then $U'_{ik} = U_{ik}$ and $\mu'_i(t) = \mu_i(t) = 0$. If $\mu_i(t) = 1$, then

$$\mu'_i(t) = H\Big(\kappa - z_i(t+1) \sum_{k=1}^{N} \big(U_{ik} + \eta z_i(t+1) x_k(t)\big) x_k(t)\Big) \quad (49)$$

$$= H\Big(\kappa - z_i(t+1) \sum_{k=1}^{N} U_{ik} x_k(t) - \eta \big(z_i(t+1)\big)^2 \sum_{k=1}^{N} \big(x_k(t)\big)^2\Big) \quad (50)$$

$$= H\Big(\kappa - z_i(t+1) \sum_{k=1}^{N} U_{ik} x_k(t) - \eta N\Big) = 0 \quad (51)$$

for sufficiently large $\eta > 0$ given $x_k(t) = \pm 1$, $z_i(t+1) = \pm 1$ and the property of Heaviside function.

E Proof of Theorem 4

If $\nu_j(t) = 0$, then we have

$$x_j(t+1) \sum_{k=1}^{M} V_{jk} y_k(t) \geq \kappa. \tag{52}$$

Next,

$$\begin{aligned}
x_j(t+1) \sum_{k=1}^{M} V_{jk} \hat{y}_k(t) &= x_j(t+1) \sum_{k=1}^{M} V_{jk} \big(y_k(t) + \hat{y}_k(t) - y_k(t) \big) && (53) \\
&= x_j(t+1) \sum_{k=1}^{M} V_{jk} y_k(t) + x_j(t+1) \sum_{k=1}^{M} V_{jk} \big(\hat{y}_k(t) - y_k(t) \big) && (54) \\
&\geq \kappa + x_j(t+1) \sum_{k=1}^{M} V_{jk} \big(\hat{y}_k(t) - y_k(t) \big) && (55) \\
&\geq \kappa - \Big| \sum_{k=1}^{M} V_{jk} \big(\hat{y}_k(t) - y_k(t) \big) \Big| && (56) \\
&\geq \kappa - \max_k |V_{jk}| \sum_{k=1}^{M} |\hat{y}_k(t) - y_k(t)| && (57) \\
&> \kappa - \max_k |V_{jk}| \cdot \epsilon > 0 && (58)
\end{aligned}$$

since $x_j(t+1) = \pm 1$, which implies

$$x_j(t+1) = \text{sign}\Big(\sum_{k=1}^{M} V_{jk} \hat{y}_k(t) \Big). \tag{59}$$