BAYESIAN LEARNING METHOD IN THE DISCRETE CELLULAR NEURAL NETWORKS APPLIED TO IMAGE PROCESSING

Nguyen Quang Hoan, Nguyen Ngoc Quynh Chau*
Hanoi Water Resources University

ARTICLE INFO

Received: 19/3/2025
Revised: 30/6/2025
Published: 30/6/2025

KEYWORDS

Cellular Neural Networks
Bayesian Learning
Metropolis-Hasting
Edge detection
Network size

ABSTRACT

This paper focuses on training the coefficients of the discrete cellular neural networks using Bayesian learning. In this method, the prior distribution, which is assumed to be a Gauss distribution, is determined based on prior information and the posterior distribution is then calculated using Bayes' theorem. A Markov Chain Monte Carlo method, specifically the Metropolis-Hastings algorithm, is used for generating random samples corresponding to the posterior distribution, thereby helping to estimate the coefficients of the network. We have modified the Metropolis-Hastings algorithm to reduce the coefficient estimation time. Some image processing experiments are implemented with estimated coefficients. In more details, we target the size of the network to be trained. Using the training data obtained from the distillation technique, we found that training a smaller network size using the method described above for image processing also gives equivalent results as compared to a larger network size. This can reduce the training time, resulting in smaller training costs and thus increasing the training efficiency of the discrete cellular neural networks.

PHƯƠNG PHÁP HỌC BAYES TRONG MẠNG NƠ RON TẾ BÀO RỜI RẠC ỨNG DỤNG CHO XỬ LÝ ẢNH

Nguyễn Quang Hoan, Nguyễn Ngọc Quỳnh Châu*
Trường Đại học Thủy Lợi Hà Nội

THÔNG TIN BÀI BÁO

Ngày nhận bài: 19/3/2025
Ngày hoàn thiện: 30/6/2025
Ngày đăng: 30/6/2025

TỪ KHÓA

Mạng nơ ron tế bào
Học Bayes
Metropolis-Hasting
Dò biên
Kích thước mạng

TÓM TẮT

Bài báo này tập trung vào việc đào tạo các hệ số của mạng nơ-ron tế bào rời rạc bằng cách sử dụng học Bayes. Trong phương pháp này, phân phối tiên nghiệm được xác định theo thông tin có từ trước. Sau khi quan sát được thực hiện và sử dụng phân phối tiên nghiệm, phân phối hậu nghiệm sẽ được tính theo định lý Bayes với giả thiết rằng phân phối tiên nghiệm tuân theo phân phối chuẩn Gauss. Phương pháp chuỗi Markov Chain Monte Carlo, cụ thể là phương pháp Metropolis-Hastings, được sử dụng để tạo ra các mẫu ngẫu nhiên tương ứng với phân phối hậu nghiệm, nhờ đó giúp đánh giá được hệ số mẫu của mạng nơ-ron tế bào rời rạc. Thuật toán Metropolis-Hastings được chúng tôi chỉnh sửa để giảm thời gian của quá trình ước tính hệ số. Sử dụng các hệ số thu được từ ước tính này, một số thí nghiệm xử lý hình ảnh được triển khai. Chi tiết hơn, chúng tôi nhắm tới kích thước của mạng cần huấn luyện. Sử dụng dữ liệu đào tạo thu được từ kỹ thuật chưng cất, chúng tôi thấy rằng việc huấn luyện một mạng nơ-ron tế bào rời rạc có kích thước nhỏ hơn bằng phương pháp vừa nêu trong xử lý ảnh cũng đem lại kết quả tương đương với việc huấn luyện một mạng có kích thước lớn hơn. Nhờ đó ta có thể giảm thời gian huấn luyện, dẫn tới giảm chi phí huấn luyện và tăng hiệu quả huấn luyện của mạng nơ-ron tế bào rời rạc.

DOI: https://doi.org/10.34238/tnu-jst.12343

* Corresponding author. *Email: chaunnq@tlu.edu.vn*

1. Introduction

Cellular neural networks (CeNN) [1], [2] have become a promising area of research, notably in image processing [3] – [5]. A CeNN is designed with a set of coefficients (synaptic weights). These coefficients need to be estimated (called training) [6]. Many methods have been proposed to estimate these coefficients such as multilayer perceptron method (MLP) [7] or recurrent perception learning algorithm (RPLA) [8] or heuristic algorithms [9], [10]. Each method has its own advantages and disadvantages. Choosing an appropriate method to train a CeNN and suitable for the type of data being processed is very important. The training method can also affect the training performance.

In [11], a CeNN is trained by Bayesian learning. This method determines the prior distribution based on prior information and the posterior distribution is then calculated using Bayes' theorem. A Markov Chain Monte Carlo (MCMC) method, specifically the Metropolis-Hastings algorithm [12], [13], is used for generating random samples corresponding to the posterior distribution, thereby helping to estimate the coefficients of the network. Unlike other sampling methods, the Metropolis–Hastings algorithm allows for the easy and fast generation of a large number of random samples from an arbitrary known probability distribution.

In this work, we target the size of a CeNN that needs to be trained. We train a CeNN following the method mentioned above using grayscale image as input data and binary image as output data. We find that training a CeNN with a small size also gives results almost equivalent to training a CeNN network with a large data. Furthermore, the training time of a small CeNN is also lower than that of a large CeNN, thus reducing the training cost and increasing the training efficiency of CeNN.

The structure of the paper is as follows. An overview of the discrete-time CeNN is briefly presented in Section 2.1. The Bayesian learning method for estimating the discrete-time CeNN coefficients is described in Section 2.2. The Metropolis-Hastings algorithm for model generation is presented in Section 2.3. Testing cases of image processing are shown in Section 2.4. And comments on the obtained results with future research are discussed in the conclusion.

2. Methodology

2.1. Architectures of the Discrete Time Cellular Neural Networks (DT-CeNN)

A DT-CeNN cell differs from a standard CeNN cell (in that it requires solving ordinary differential equations) in the following aspects [1], [2]:

$$\boldsymbol{x_{ij}(n+1) = \sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)\, v_{kl}(n) + \sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l) u_{kl} + z} \quad (1)$$

and output activation function $\boldsymbol{f\left(x_{ij}(n)\right)}$:

$$\boldsymbol{v_{ij}(n) = f\left(x_{ij}(n)\right) = \frac{1}{2}\left|x_{ij}(n)+1\right| - \frac{1}{2}\left|x_{ij}(n)-1\right|} \quad (2)$$

where *u*, *v*, and *x* represent input, output, and state, respectively.

State *x* needs to be initialized before processing input data. *A* and *B* in the equation (1) are the cloning template and control template, respectively. Each cell of a template represents the synaptic weights between a cell and its neighbors. Bias *z* is the cell bias and is constant for every cell. *Sr* denotes the neighborhood of cell *C* with radius *r* and *r* determines how many neighbor cells are interconnected with each other. The output equation states that output *v* is a function of state *x*, leading to recursive property of system.

The size of input data is the size of DT-CeNN. A stable DT-CeNN has the binary output property since the network output converges to {+1, −1}. A DT-CeNN becomes stable if the weights of templates *A* and *B* are symmetric. In order to be a symmetric and stable CeNN, the center parameter of *A* (self feedback parameter) must be greater than 1 [14].

2.2. Estimation of the DT-CeNN coefficients using Bayesian learning

Bayesian learning [15] determines the prior distribution based on prior information and the posterior distribution is then calculated using Bayes' theorem. This method helps us to estimate the unknown coefficients of the DT-CeNN network. In order to do this, we model firstly a DT-CeNN network as described in [11].

A cell in the DT-CeNN can be modelled as follows:

$$y[n] = f_{\infty}^{[n]}(u, \theta) + \omega[n] \text{ , } n = 0, 1, ..., N-1 \quad (3)$$

where u is input data and y is output. The notation $f_{\infty}^{[n]}(u, \theta)$ represents the state of the DT-CeNN that is stable with the n^{th} input sample. $\omega[n]$ represents Additive Gaussian noise adding to the n^{th} output $y[n]$. We suppose that this noise has a distribution of $N(0, \sigma_w^2)$.

Equation (3) is rewritten in vector form as:

$$y = f_{\infty}(u, \theta) + \omega \quad (4)$$

As emphasized above, a DT-CeNN is stable if the templates A and B are symmetric. Each block A and B has size of 3×3 with 5 coefficients that we need to find:

$$A = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_4 \\ \theta_3 & \theta_2 & \theta_1 \end{bmatrix}, B = \begin{bmatrix} \theta_6 & \theta_7 & \theta_8 \\ \theta_9 & \theta_{10} & \theta_9 \\ \theta_8 & \theta_7 & \theta_6 \end{bmatrix}, z = \theta_{11} \quad (5)$$

where z is the constant bias for every cell of DT-CeNN. So we need to find coefficients as a vector $\theta \in R^{11\times 1}$ defined as follows:

$$\theta = [\theta_1 \ \theta_2 \ \theta_3 \ \theta_4 \ \theta_5 \ \theta_6 \ \theta_7 \ \theta_8 \ \theta_9 \ \theta_{10} \ \theta_{11}]^T \quad (6)$$

Applied Bayes theorem, posterior distribution is defined as:

$$p(\theta|y) \propto p(y|\theta)p(\theta) \quad (7)$$

where $p(\theta)$ is prior distribution and likelihood $p(y|\theta)$ is a Gaussian distribution with mean $f_{\infty}(u, \theta)$ and variance σ_{ω}^2. Knowing that N input samples are known in advance, $p(y|\theta)$ is then rewritten as below:

$$p(y|\theta) = \prod_{n=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma_{\omega}^2}} e^{-\frac{\left(y[n]-f_{\infty}^{[n]}(u,\theta)\right)^2}{2\sigma_{\omega}^2}} \quad (8)$$

Prior distribution of θ is supposed ~ $N(\mu_{\theta}, \Sigma_{\theta})$, where μ_{θ} is the mean vector and Σ_{θ} is the covariance matrix. Therefore, prior probability density function (PDF) is computed as follows:

$$p(\theta) = \frac{1}{\sqrt{(2\pi)^{11}|\Sigma_{\theta}|}} e^{-\frac{(\theta-\mu_{\theta})^T \Sigma_{\theta}^{-1}(\theta-\mu_{\theta})}{2}} \quad (9)$$

Using Equations (7), (8) and (9) we have posterior PDF as:

$$p(\theta|y) \propto p(y|\theta)p(\theta) = \prod_{n=0}^{N-1} \frac{1}{\sqrt{2\pi\sigma_{\omega}^2}} e^{-\frac{\left(y[n]-f_{\infty}^{[n]}(u,\theta)\right)^2}{2\sigma_{\omega}^2}} \text{x} \frac{1}{\sqrt{(2\pi)^{11}|\Sigma_{\theta}|}} e^{-\frac{(\theta-\mu_{\theta})^T \Sigma_{\theta}^{-1}(\theta-\mu_{\theta})}{2}} \quad (10)$$

We cannot use derivatives to solve equation (10). This is where Markov Chain Monte Carlo (MCMC) method is used.

2.3. Metropolis algorithm

The Metropolis algorithm [12], [13], which is a MCMC method, is used for generating data samples. Before creating a sample, we must initialize an initial point to start from. This point can be predicted based on prior information. To find the vector θ, we initialize a vector as follows:

$$\theta(0) = [\theta_1(0), \theta_2(0), ..., \theta_{11}(0)]^T \quad (11)$$

with $\theta(0)$ being the initial vector corresponding to the chosen initial starting point.

A sample sequence of vector θ will be obtained in the following way. From the initial point, a new vector θ_{new} is generated according to a Gaussian distribution function [12]. Vector θ_{new} can be considered as new sample. The new sample can be accepted and inserted into the sample sequence with a rate α given in [12], [13]. The rate α is defined as follows:

$$\alpha = \frac{p(\theta_{new}|y)}{p(\theta(i-1)|y)} \quad (12)$$

where $p(\theta_{new}|y)$ is the posterior distribution of vector θ_{new} while $p(\theta(i-1)|y)$ is the posterior distribution of vector $\theta(i-1)$, which is the previous vector sample in the sequence. These posterior distributions are computed using equation (10). Note that the vector θ has 11 components, so to determine the squared error in equation (10), we have to calculate the sum of the squared errors of all the components of the vector.

The sample sequence will finally be created from K samples [16], [17]:

$$\theta(i)|y=[\theta_1(i)|y, \theta_2(i)|y, ..., \theta_{11}(i)|y]^T \text{ where } i=0, 1, ..., K-1 \quad (13)$$

The estimated value of the vector θ (so-called θ') will be approximately equal to the average of K samples in the sample sequence (due to the hypothesis that the vector θ follows a normal distribution):

$$\theta' = E[\theta|y] \sim \frac{1}{K}\sum_{i=0}^{K-1}(\theta(i)|y) \quad (14)$$

Note that equation (14) is applied for each component of $\theta(i)|y$. The obtained vector θ' are then used in equation (5) to construct the templates A, B and the bias z.

3. Image processing results

In our experiments, we target the size of the DT-CeNN to be trained. Instead of searching for coefficients (the cloning template, the control template and the bias) of a DT-CeNN of size of $n*n$ corresponding to the input data size, we try to find coefficients for a DT-CeNN of size of $k*k$ where $k < n$. After obtaining coefficients of $k*k$ DT-CeNN, we apply these coefficients to the $n*n$ DT-CeNN and then we use this $n*n$-sized network to process the input data of size of $n*n$.

As stated above, the center parameter of block A must be greater than 1. Therefore, the Metropolis algorithm is modified in the random generation step with the following rule: if a newly generated random value causes the self-feedback parameter to be less than 1, it must be regenerated. In equation (11), the vector θ has 11 components, so the aforementioned rule applies to the 5th component corresponding to the central parameter of the cloning template. With this rule, we find that the training process is faster (as demonstrated in Figure 1) while still ensuring equivalent output results. In Figure 1, we run two algorithms: the original Metropolis algorithm (referred to as Norm-Metro) and the modified Metropolis algorithm (Modified-Metro) for the image edge detection with a 16x16 DT-CeNN using the training input image number of 32. We observe that after 15,000 generated samples, the original Metropolis algorithm runs slower as compared to the modified Metropolis algorithm, while the results from both algorithms are equivalent (as detailed later in Figure 7). Based on this experiment, we decided to use the modified Metropolis algorithm in all other experiments presented in this paper.

During the training phase, we plan to train the DT-CeNN for two image processing cases: edge detection and rectangular object filling. We use equations (1), (2) and (4) to calculate $f_\infty(u,\theta)$

for each generated sample. The maximum iterations for calculation of $f_\infty(u,\theta)$ is limited to 100. The initial value of state x in equation (1) is chosen as either 1 or 0, depending on the image processing case.

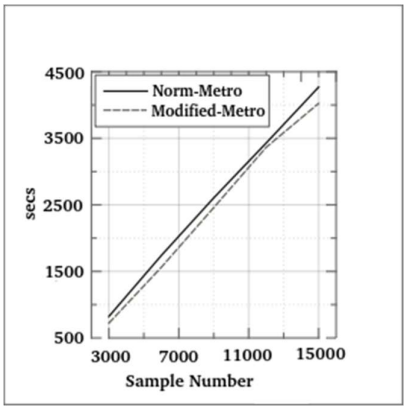


Figure 1. *Training time of a 16x16 DT-CeNN: original Metropolis vs modified Metropolis*

The initial values of vector θ in equation (11) are set to 1. During the execution of the Modified-Metro algorithm, the uniform distribution is always $U(0,1)$. The prior distribution $p(\theta)$ is a Gaussian distribution of $N(1,2)$ while the likelihood $p(y|\theta)$ follows a Gaussian distribution of $N(0,4)$. All experiments are executed on a laptop with an Intel Core i7 CPU and 8GB RAM.

3.1. Rectangle filling with different sizes of DT-CeNN

We use a set of input images and a set of output images to train a DT-CeNN to fill the rectangle object. We randomly generate 2 sets for training: 128 rectangles containing only an outline with a border width of 2 pixels and 128 solid black-filled rectangles. Each input/output image pair corresponds to a rectangle of the same size. For different input/output pairs, the rectangle size is different. These rectangles have random sizes and are stored in images of a given size. Since we have 128 input/output image pairs for training, so parameter N in equation (10) is equal to 128. This study aims to examine how the training time of the DT-CeNN changes with different network sizes. Note that the input/output images used for training has the same size of *kxk* as the DT-CeNN. We train the DT-CeNN with sizes of 20×20, 24×24 and 32×32 respectively (corresponding to k = 20, 24, 32) to estimate the coefficients. Then, we apply the obtained coefficients to the 512×512 DT-CeNN using a 512×512 testing image as input. The resulting output images are shown in Figure 2. We observe that the output results remain consistent across different values of k.

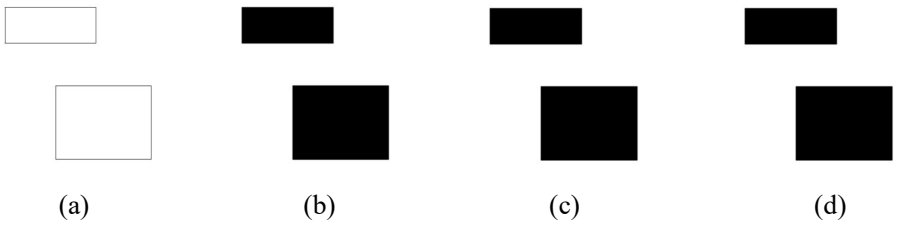


Figure 2. *512*512 testing images: (a) input and output with (b) k = 20, (c) k = 24 and (d) k = 32*

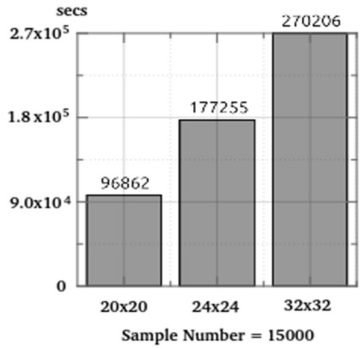


Figure 3. *Training time (in second) of the DT-CeNN with different sizes: 20x20, 24x24, 32x32*

A total of 15,000 samples are generated using the modified Metropolis algorithm. Thus, the parameter K in equation (14) is equal to 15,000. Figure 3 illustrates the training time of DT-CeNN of different sizes: 20×20, 24×24 and 32×32. We observe that the training time scales linearly with the DT-CeNN size. Theoretically, the training time of a 32x32 network will increase by about 1024/400 = 2.56 times compared to a 20x20 network. The actual measured training time is often larger than the theoretical training time because the longer the computer runs, the hotter it gets.

3.2. Edge detection

In this experiment, we use 'lena' standard image – a standard grayscale image of size of 256x256 to train the DT-CeNN. To prepare for training, we use coefficients obtained in the paper [11] applied to a 256x256 DT-CeNN with the input of the 'lena' standard image in order to obtain a 'lena' edge-filtered image as output as shown in Figure 4.

(a) (b)

Figure 4. *256x256 'lena' images prepared for training: (a) original standard image and (b) image obtained through a 256x256 DT-CeNN using coefficients in [11]*

The two images in Figure 4 are cut into two sample sets. Each image is cut into one set, so totally we have two training sets: one for input and one for output. Each set contains the images of size of 16x16. These two sets are used to train the 16x16 DT-CeNN. We choose 64 images from two sets for training (we have 32 input/output image pairs for training). The reason we choose 32 training samples, that is smaller than 128 training samples in previous experiments, in this training is because we try to minimize the training time while checking whether the DT-CeNN still operates stably. During the training process, a total of 15,000 samples will be generated by the modified Metropolis algorithm. As a result, we obtain z = -0.620913 and:

$$A = \begin{bmatrix} 2.397838 & 3.155505 & -2.143711 \\ -2.756646 & 3.189971 & -2.756646 \\ -2.143711 & 3.155505 & 2.397838 \end{bmatrix}, B = \begin{bmatrix} 0.104867 & -0.938289 & -1.269894 \\ -1.138890 & 6.923888 & -1.138890 \\ -1.269894 & -0.938289 & 0.104867 \end{bmatrix}$$

Finally, we use these obtained coefficients for the 512x512 DT-CeNN and perform on the 512x512 testing images (shown in Figure 5).

The results (Figure 6) show that we only need to train a small DT-CeNN of 16x16 size (vs 512x512) with a small number of training samples (32 samples vs 128 samples) in order to estimate the coefficients and we can still get the good results. With this method, we see that the

training time of the DT-CeNN is very small (about 4028 seconds). In theory, the training time can be reduced by 128*512*512/32*16*16 = 4096 times as compared to training the 512x512 DT-CeNN using 128 input training samples.

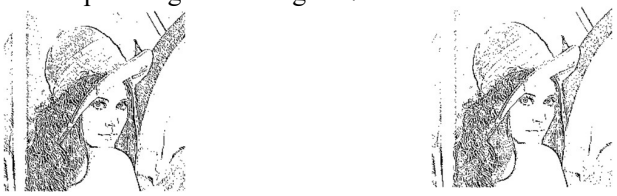
(a) *(b)* *(c)*

Figure 5. *512x512 testing images: (a) lena, (b) barbara and (c) mandril*

(a) *(b)* *(c)*

Figure 6. *512x512 resulting images: (a) lena, (b) barbara and (c) mandril*

In the final experiment, we kept the same training sets as above for two scenarios: the original Metropolis algorithm vs the modified Metropolis algorithm. Then we apply the coefficients obtained in each scenario to the 512x512 DT-CeNN for testing on 512x512 'lena' standard image as input, and we get the output images as in Figure 7.



(a) *(b)*

Figure 7. *Output lena images using: (a) modified Metropolis and (b) original Metropolis*

The output results are seemly similar (Figure 7) while the training time of the modified Metropolis algorithm is always lower than that of the original Metropolis algorithm (4028 seconds as compared to 4374 seconds respectively).

4. Conclusion

This paper focuses on estimating the coefficients of a DT-CeNN network using Bayesian learning combined with the Metropolis method. We have modified the Metropolis algorithm to reduce the training time without affecting the results of the algorithm. We found that training a smaller DT-CeNN using the above method also gives results equivalent to training a larger DT-CeNN, thereby reducing the training cost and increasing the training efficiency. However, to do this, we need to have suitable input data and in the paper we have used standard gray scale images of lena and its edge-filtered images as the training data sets for input and output, respectively. Experiments show that training a small DT-CeNN using a small number of input training samples takes only a very short training time and the obtained coefficients still meet the requirements. This is the distillation technique used by DeepSeek - a large language model [18]. In the future, we will aim to train higher-order cellular neural networks using this technique.

REFERENCES

[1] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE T. Circuits Syst.,* vol. 35, pp. 1257-1272, 1988.

[2] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE T. Circuits Syst.,* vol. 35, pp. 1273-1290, 1988.

[3] K. Kim, S. Lee, J. Y. Kim, M. Kim, and H. J. Yoo, "A Configurable Heterogeneous Multicore Architecture With Cellular Neural Network For Real-time Object Recognition," *IEEE T. Circ. Syst. Vid.*, vol. 19, no. 11, pp. 1612-1622, 2009.

[4] G. Costantini, D. Casali, and M. Carota, "CNN-Based Unsupervised Pattern Classification For Linearly And Non Linearly Separable Data Sets," *WSEAS Transactions on Circuits and Systems*, vol. 4, no. 5, pp. 448-452, 2005.

[5] A. Kananen, A. Paasio, M. Laiho, and K. Halonen, "CNN Applications From The Hardware Point Of View: Video Sequence Segmentation," *International Journal of Circuit Theory and Applications,* vol. 30, no. 2-3, pp. 117-137, 2002.

[6] J. A. Nossek, "Design And Learning With Cellular Neural Networks," in *Third IEEE International Workshop on Cellular Neural Networks and their Applications*, Rome, 1994, pp. 137-146.

[7] M. Vinyoles-Serra, S. Jankowski, and Z. Szymanski, "Cellular Neural Network Learning Using Multilayer Perceptron," in *20th European Conference on Circuit Theory and Design,* Linkping, 2011, pp. 214-217.

[8] C. Güzeliş and S. Karamahmut, "Recurrent Perceptron Learning Algorithm For Completely Stable Neural Networks," in *Third IEEE International Workshop on Cellular Neural Networks and their Applications,* Rome, 1994, pp. 177-182.

[9] T. Kozek, T. Roska, and L. O. Chua, "Genetic Algorithm For CNN Template Learning," *IEEE T. Circuits Syst.,* vol. 40, no. 6, pp. 392-402, 1993.

[10] M. Ünal, M. Onat, and A. Bal, "Cellular Neural Network Training By Ant Colony Optimization Algorithm," in *IEEE 18th Signal Processing and Communications Applications Conference,* Diyarbakır, 2010, pp. 1661-1666.

[11] H. M. Özer, A. Özmen, and H. Şenol, "Bayesian Estimation Of Discrete-time Cellular Neural Network Coefficients," *Turkish Journal of Electrical Engineering & Computer Sciences,* vol. 25, pp. 2363 – 2374, 2017.

[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations Of State Calculations By Fast Computing Machines," *J. Chem. Phys.,* vol. 21, pp. 1087-1092, 1953.

[13] W. K. Hastings, "Monte Carlo Sampling Methods Using Markov Chains And Their Applications," *Biometrika,* vol. 57, pp. 97-109, 1970.

[14] S. Arik, "Stability Analysis Of Dynamical Neural Networks," PhD. Thesis, South Bank University, UK, 1997.

[15] R. M. Neal, "Bayesian Learning For Neural Networks," PhD. Thesis, University of Toronto, Canada, 1995.

[16] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms,* UK: Cambridge University Press, Cambridge, 2003.

[17] C. M. Bishop, *Pattern Recognition and Machine Learning*, USA: Springer, 2006.

[18] DeepSeek Team, "DeepSeek-R1: Incentivizing Reasoning Capability In LLMs Via Reinforcement Learning Computation And Language," arXiv:2501.12948, 2025. [Online]. Available: https://arxiv.org/abs/2501.12948. [Accessed March 15, 2025].