Efficient Eigenvalue Computation of Parahermitian Matrices Using Neural Networks

Diyari A. Hassan
Faculty of Engineering & Computer Science
Qaiwan International University
Sulaymaniyah, Kurdistan Region-Iraq
diyari.hassan@uniq.edu.iq

Yunus Egi and Soydan Redif
College of Engineering and Technology
American University of the Middle East
Egaila 54200, Kuwait
{yunus.egi, soydan.redif}@aum.edu.kw

***Abstract*—In recent years, computing the eigenvalue decomposition of a polynomial matrix has become increasingly essential in many areas of adaptive signal processing. However, traditional iterative algorithms, such as sequential matrix diagonalisation (SMD), often incur high computational costs. This paper proposes two artificial neural network (ANN)-based approaches for computing polynomial eigenvalues estimated by the SMD. By utilizing feed-forward and convolutional neural network models, we significantly reduce computational costs, including CPU time and floating-point operations per second (FLOPS). The results demonstrate that ANN-based polynomial eigenvalue decomposition (PEVD) offers a more efficient solution for large matrix computations compared to traditional methods.**

***Index Terms*—Efficient polynomial Eigenvalue Decomposition, Artificial Neural Networks, Computational efficiency, floating-point operations per second (FLOPS).**

I. Introduction

The space-time covariance matrix, $\mathbf{R}[\tau]$, is crucial in addressing various issues related to broadband signal processing [1], [2], [3], [4]. For an M-element broadband array capturing signal samples $\mathbf{x}[n] \in \mathbb{C}^M$ at discrete time $n \in \mathbb{Z}$, $\mathbf{R}[\tau] = \mathbb{E}\{\mathbf{x}[n]\mathbf{x}^{\mathrm{H}}[n-\tau]\}$ describes the second-order statistical dependencies between the signals $\mathbf{x}[n]$ across different time delays $\tau \in \mathbb{Z}$. This explicit characterization of delays between array sensors using the lag parameter τ allows extraction of information regarding the direction of incoming signals. Additionally, akin to the Hermitian symmetry in the conventional covariance matrix $\mathbf{R}[0]$, the z-transform of $\mathbf{R}[\tau]$, known as the cross-spectral density (CSD) matrix, exhibits parahermitian symmetry, i.e., $\boldsymbol{\mathcal{R}}^{\mathrm{P}}(z) = \boldsymbol{\mathcal{R}}^{\mathrm{H}}(1/z^*) = \boldsymbol{\mathcal{R}}(z)$, where $[\cdot]^{\mathrm{P}}$ denotes the parahermitian operator [1].

Optimal solutions in many array-processing problems have traditionally relied on the eigenvalue decomposition (EVD) of $\mathbf{R}[0]$. For broadband-array processing, however, a polynomial EVD (PEVD) is essential to effectively diagonalize $\mathbf{R}[\tau] \rightarrow \boldsymbol{\mathcal{R}}(z)$. PEVD methods such as the second-order sequential best rotation (SBR2) [1], sequential matrix diagonalization (SMD) [5], DFT-based smooth decomposition [6], and analytic PEVD algorithms [7], [8] have been developed. These iterative, time-domain approaches, like SBR2 and SMD, have been successfully applied to various applications, including subband coding [2], broadband beamforming [9], blind source separation [3], [10], MIMO communications [11], [12], [13], [14], and speech enhancement [15], [16]. Given the high computational demands of these applications, recent research has focused on developing computationally efficient techniques for PEVD calculation [17], [18], [19], [20] to minimize CPU time and reduce computational costs.

The rise of artificial neural networks (ANNs) in machine learning has led to their application in tasks requiring advanced pattern recognition, such as speech, image, and language processing. Notably, ANNs have been applied to the EVD of scalar matrices $\mathbf{R}[0]$ [21], [22], [23], [24], [25]. These studies indicate that ANN-based EVD is more computationally efficient and offers greater utility compared to traditional algorithms. For instance, [24], [25] demonstrated significant computational efficiency when using deep neural networks for the EVD of symmetric matrices. The deep convolutional neural networks (CNNs) in [24] showed improved eigenvalue-estimation accuracy and high data efficiency, achieving high accuracy with fewer samples than traditional neural network-based methods.

Despite these benefits, the implementation of PEVD using ANNs and the associated computational advantages remain unexplored. Therefore, in this paper, we propose two ANN-based solutions for computing the polynomial eigenvalues of a parahermitian matrix and provide a detailed characterisation of their computational performance. The two proposed neural-network architectures, designed to generate the polynomial eigenvalues estimated by the SMD algorithm, are as follows: (i) a feed-forward neural network approach and (ii) a CNN-based approach.

The remainder of this paper is organized as follows: Section II describes the SMD-based PEVD. Section III presents two neural network models for computing the polynomial eigenvalues of $\mathbf{R}[\tau]$. Simulation results are shown in Section IV. Finally, conclusions are drawn in Section V.

II. Polynomial Matrix Eigenvalue Decomposition (PEVD)

In this paper, we introduce two ANN architectures for computing the polynomial eigenvalues of parahermitian matrices, as estimated by SMD in [5], and explore the resulting benefits in terms of accuracy and computational complexity.

The SMD algorithm is an iterative method in the time domain designed for estimating the PEVD of a parahermitian matrix $\boldsymbol{\mathcal{R}}(z) : \mathbb{C} \to \mathbb{C}^{M\times M}$. SMD decomposes $\boldsymbol{\mathcal{R}}(z)$ as:

$$\boldsymbol{\mathcal{Q}}(z)\boldsymbol{\mathcal{R}}(z)\boldsymbol{\mathcal{Q}}^{\mathrm{P}}(z) \approx \boldsymbol{\mathcal{D}}(z), \tag{1}$$

where $\boldsymbol{\mathcal{Q}}(z)$ and $\boldsymbol{\mathcal{D}}(z)$ contain the approximate polynomial eigenvectors and eigenvalues, respectively.

SMD starts by computing the unitary matrix $\mathbf{Q}^{(0)} \in \mathbb{C}^{M\times M}$ through a scalar EVD that diagonalizes $\mathbf{R}[0]$ and transfers energy to the main diagonal of $\boldsymbol{S}^{(0)}(z)$. Initialization sets $\boldsymbol{H}^{(0)}(z) = \mathbf{Q}^{(0)}$. In each iteration ($l = 1, 2, \ldots, L$), SMD shifts a dominant off-diagonal row (or column) at lag τ_l onto $\mathbf{S}^{(l-1)}[0]$ using a time-shift matrix $\boldsymbol{\Lambda}^{(l)}(z)$ and then diagonalizes $\boldsymbol{S}^{(l-1)}(z)$ through an ordered conventional matrix EVD. The unitary matrix $\mathbf{Q}^{(l)}$ is derived from the scalar EVD of the zero-lag coefficient matrix $\mathbf{S}[0]$ and applied to all order matrices $\mathbf{S}[\tau]$. SMD iteratively transfers cross terms of $\mathbf{S}^{(l-1)}[0]$ to its main diagonal, computing the paraunitary (PU) transformation:

$$\boldsymbol{\mathcal{S}}^{(l)}(z) = \boldsymbol{\mathcal{U}}^{(l)}(z)\boldsymbol{\mathcal{S}}^{(l-1)}(z)\boldsymbol{\mathcal{U}}^{\mathrm{P}\,(l)}(z). \tag{2}$$

In 2, $\boldsymbol{\mathcal{U}}^{(l)}(z)$ is a PU matrix satisfying $\boldsymbol{\mathcal{U}}(z)\boldsymbol{\mathcal{U}}^{\mathrm{P}}(z) = \boldsymbol{\mathcal{U}}^{\mathrm{P}}(z)\boldsymbol{\mathcal{U}}(z) = \boldsymbol{\mathcal{I}}$, formed as the product of a unitary matrix $Q^{(l)}$ and a PU time-shift matrix $\boldsymbol{\Lambda}^{(l)}(z)$, selected based on the location of the dominant off-diagonal column (row) in $\boldsymbol{\mathcal{S}}^{(l-1)}(z)$. The algorithm identifies the dominant off-diagonal column using the modified column vector $\hat{\mathbf{s}}_k^{(l-1)}[\tau]$ defined by excluding the diagonal $s_{k,k}^{(l-1)}[\tau]$:

$$\|\hat{\mathbf{s}}_k^{(l-1)}[\tau]\|_2 = \sqrt{\sum_{j=1, j\neq k}^{M} |s_{j,k}^{(l-1)}[\tau]|^2}, \tag{3}$$

where $s_{j,k}^{(l-1)}[\tau]$ denotes the scalar coefficient in the jth row and kth column of $\mathbf{S}^{(l-1)}[\tau]$ at lag τ.

The SMD algorithm iterates until $\boldsymbol{\mathcal{S}}^{(L)}(z)$ is sufficiently diagonalized, ensuring the largest norm of the off-diagonal column satisfies:

$$\max_{k,\tau} \|\hat{\mathbf{s}}_k^{(L)}[\tau]\|_2 \leq \rho, \tag{4}$$

where ρ is a small value. After L iterations, SMD provides an approximate PEVD.

For convergence proofs of the SMD algorithm, refer to [5].

III. Neural Network for Computing PEVD

This section introduces two architectures of artificial neural networks (ANNs) designed to compute polynomial eigenvalues of parahermitian matrices. Specifically, we explore feed-forward neural networks (FNN) and convolutional neural networks (CNN). The setup and parameters for both FNN and CNN models will be detailed.

A. Feed-Forward Neural Network (FNN)

A feed-forward neural network (FNN) is a type of artificial neural network where connections between nodes do not form cycles, unlike recurrent neural networks. It processes information in a single direction, from input to output. In FNNs, weights are adjusted through training using methods like gradient descent and back-propagation, which refine the network's ability to produce accurate outputs. Multi-layer perceptrons utilize back-propagation to adjust weights in each hidden layer based on output from the final layer [26].

B. Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) are deep learning models particularly suited for analyzing visual data such as images and videos. They employ layers that automatically learn and detect patterns and features within the input data. CNNs are pivotal in tasks like image classification, object detection, and facial recognition, leveraging convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for making predictions. CNNs have revolutionized computer vision and are foundational in deep learning [27].

C. ANN-based PEVD

Our ANN architecture takes as input the polynomial eigenvalues extracted from the diagonalized matrix $\boldsymbol{\mathcal{D}}(z)$, estimated using the SMD algorithm on the parahermitian matrix $\boldsymbol{\mathcal{R}}(z)$ (see Eq. 1). These eigenvalues are fed into both our FNN and CNN models.

The CNN employs hierarchical feature extraction through convolutional and pooling layers. Specifically, we use two convolutional layers with ReLU activation functions, each employing 64 and 32 filters respectively. A max-pooling layer of size (2×2) with a stride of (2×2) follows to downsample features, enhance computational efficiency, and prevent overfitting. For the FNN, which employs multiple hidden layers for complex decision-making, we configure two fully connected layers with 64 and 32 neurons to capture intricate relationships within the polynomial matrices. ReLU activation functions in the hidden layers facilitate nonlinear transformations, while linear activation is applied in the output layer for regression and eigenvalue estimation. The model's effectiveness is evaluated iteratively using Mean Square Error (MSE) to minimize prediction errors. Figure 1 depicts a block diagram of our ANN-based SMD architecture used for training, and ***Algorithm 1*** provides the pseudocode for implementing this architecture.

D. Performance Evaluation Metrics

Performance metrics provide measurable criteria for assessing the effectiveness of an ANN model. Key evaluation metrics used in this study include mean squared error (MSE) and mean absolute error (MAE). These metrics are crucial for quantitatively evaluating how well ANN models predict PEVD values, guiding iterative improvements and optimization efforts in regression tasks.

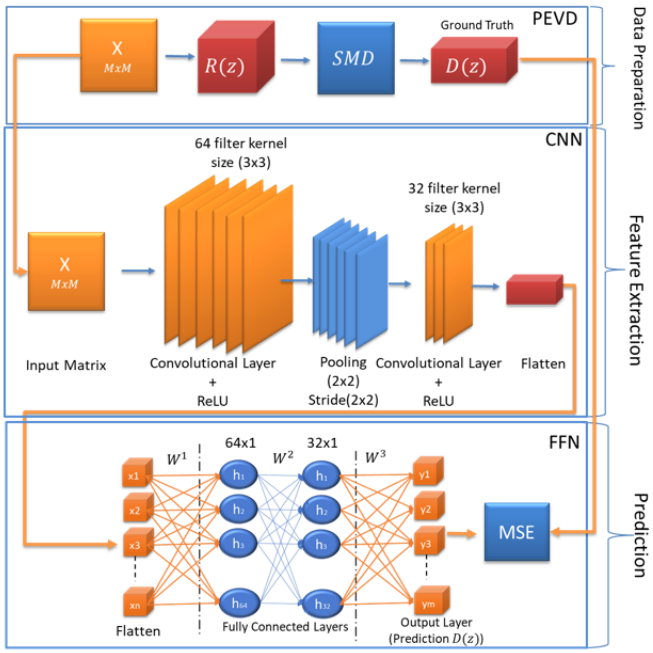

Fig. 1. Proposed ANN- based SMD architectures

MSE is computed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{5}$$

where n denotes the number of samples, y_i represents the actual PEVD values, and $\hat{y}_i$ denotes the predicted PEVD values. MSE values are calculated during the training phase.

Mean absolute error (MAE) is calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|, \tag{6}$$

where $\bar{y}$ represents the mean of the observed data. MAE is used to assess the agreement between actual and predicted polynomial eigenvalues.

E. Computational Efficiency: CPU time for Algorithm Computation

The computational efficiency of algorithms is crucial for their practical applicability, especially in resource-constrained environments like those relying on traditional CPUs. This study evaluates the CPU time required to execute the proposed algorithm compared to iterative time-domain approaches such as the SMD algorithm. CPU time, measured in seconds, quantifies the total processing time required by the processor to complete the algorithm's execution on a given dataset or input size. It encompasses not only the algorithm's computational complexity but also the efficiency of its implementation and hardware utilization[28], [29].

To accurately assess CPU time, we conducted extensive experiments across various datasets and input configurations.

Algorithm 1 ANN-based SMD for calculating polynomial eigenvalues of a parahermitian matrix.

NET: Choose Neural Network Model (FFN, CNN)
$num\_samples = N$
$matrix\_size = M$
$Lag = L$
$D\_eff\_length = Lag + 2$
Initialise D as a matrix of zeros
Generate N random matrices with dimension $M \times M$ denoted as $X_{M\times M}$.
Step 1: Prepare the train data
for $i = 1$ to $num\_samples$ **do**
  $\mathcal{R}(z) =$ Compute the space-time covariance matrix(X)
  $[\mathcal{U}(z), \mathcal{D}\rangle(z)] =$ Apply SMD$(\mathcal{R}(z))$
  $D\_eff =$ Extract Effective Polynomial Coefficients(Di)
  Store D_eff in the D matrix
end for
Step 2: Define the architecture
if NET = FFN **then**
  Define a feed-forward neural network 'net' with input and output sizes
  $input\_size = matrix\_size \times matrix\_size \times Lag$
  $output\_size = matrix\_size \times D\_eff\_length$
  $hidden\_size = H$ where H is an arbitrary number $> (2 \times input\_size)$
else
  Define a convolutional neural network 'net' with input and output sizes
  Input layer: Matrix Input Layer with size $[matrix\_size \times matrix\_size \times Lag]$
  Convolutional layer with MxM filters and K output channels, followed by ReLU activation
  Max-pooling layer with 2x2 pooling and a 2x2 stride
  Convolutional layer with MxM filters and 32 output channels, followed by ReLU activation
  Fully connected layer with 64 Neurons, followed by ReLU activation
  Fully connected layer with 32 Neurons, followed by ReLU activation
  Output layer with $(matrix\_size \times D\_eff\_length)$ units
end if
Step 3: Train the network
Train the network 'net' using the input data X and target data D
Step 4: Test the network
$num\_test\_samples = Tn$
Initialise $predicted\_eigenvalues$ as a matrix of zeros
Initialise $actual\_eigenvalues$ as a matrix of zeros
Generate a random test matrix X_test
Calculate $actual\_eigenvalues$ for the test data using SMD
Predict the polynomial eigenvalue with the ANN model
Determine the precision of predicted polynomial eigenvalues in relation to the actual values.

F. Floating-Point Operations per Second (FLOPS)

FLOPS serve as a fundamental metric for evaluating the computational performance of algorithms, particularly in numerical computation and machine learning. FLOPS quantifies the rate at which a processor performs arithmetic operations involving floating-point numbers, providing crucial insights into an algorithm's efficiency in utilising computational resources[30], [31]

From [12], it is determined that the number of FLOPS required to perform an SMD on the matrix $\boldsymbol{\mathcal{R}}(z)$ is estimated as approximately

$$SMD_{FLOPS} = M^3 \sum_{l=0}^{L} \left(2\text{len}\{\boldsymbol{R}^{(l)}\} + \text{len}\{\boldsymbol{U}^{(l)}\} \right), \quad (7)$$

where M denotes the size of the input matrix, and len represents the length of a matrix.

To calculate the FLOPs for a feedforward neural network, follow these steps:

1. Calculate FLOPs for Each Layer

For fully connected (dense) layers, the FLOPs can be calculated using the formula:

$$FNN_{FLOPs} = 2 \times N \times M, \quad (8)$$

where N is the number of input neurons and M is the number of output neurons. The factor of 2 accounts for both multiplication and addition operations.

2. Sum FLOPs for All Layers

Sum the FLOPs for all layers in the network to obtain the total FLOPs.

To calculate the FLOPs for a convolutional neural network (CNN), determine the FLOPs for each layer and sum them up as follows:

1. Calculate FLOPs for a Convolutional Layer

$$CNN_{FLOPs} = 2 \times H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}} \times (K \times K \times C_{\text{in}} + 1), \quad (9)$$

where:

- H_{out} and W_{out} are the height and width of the output feature map.
- C_{out} is the number of output channels.
- K is the kernel size.
- C_{in} is the number of input channels.
- The term $(K \times K \times C_{\text{in}} + 1)$ includes the bias term.

2. Calculate FLOPs for Fully Connected Layers

$$CNN_{FLOPs} = 2 \times (N \times M + M), \quad (10)$$

where N and M are the number of input and output neurons, respectively.

3. Calculate FLOPs for Other Layers

Pooling and activation functions contribute minimally to the total FLOPs.

4. Sum FLOPs for All Layers

The total FLOPs across all layers determine the computational load.

This paper analyses the FLOPs achieved by our proposed algorithm compared to the SMD algorithm. By measuring the total number of FLOPS executed per second during algorithm execution, we assess its computational efficiency.

IV. Results and Discussion

To demonstrate the benefits of the proposed ANN architecture, we present results pertaining to the efficacy of the polynomial eigenvalues and the computational demand in generating these polynomial eigenvalues.

In constructing and analysing the performance of the proposed ANN architecture, we used the Deep Learning Toolbox made available in Matlab. To train and test the proposed models, 10,000 samples of a synthetic dataset containing 3×3 matrices are generated from zero-mean, i.i.d. random processes. After computing the space-time covariance matrix for each data matrix, the SMD algorithm is applied to obtain the polynomial eigenvalues for testing the proposed model. SMD is allowed to run until a normalised off-diagonal energy of $E_{norm} \leq -20$ dB is achieved, with a truncation parameter set to $\mu = 10^{-7}$. Following this, the order of polynomial eigenvalues is truncated to an arbitrary effective length that enforces constraints on the property of $\boldsymbol{D}_l(z)$. This truncation is necessary because the size of the output array of the neural network needs to be fixed over all iterations. For adaptive learning, the Adam optimiser was selected. The initial learning rate, number of epochs, and maximum iteration were set to 0.001, 100, and 500, respectively. Training and testing were completed after 100 epochs with an average MSE value of 0.18.

Fig. 2 depicts the relationship between the MAE accuracy of the estimated polynomial eigenvalues and the number of data matrices used to train the neural network model. The maximum accuracy for CNN and FNN is 85.6% and 74.9%, respectively. It is clear that, overall, the CNN outperforms the FNN. This superiority can be attributed to the CNN's ability to process input as a scalar matrix and better feature extraction through convolution. In our approach, we feed slices (lags) of the polynomial matrix as input data into the CNN. In contrast, the FNN requires a vector of data, necessitating the reshaping of the polynomial matrix into a vector matrix before feeding it into the system. This process disrupts the spatial relationships between the lags, diminishing the effectiveness of the FNN in capturing the inherent structure of the polynomial matrix.

In Figure 3, the time complexity is evaluated for calculating the polynomial eigenvalues of a matrix concerning varying sizes of input matrices. It is observable that increasing the size of the matrix necessitates more time for computation. Moreover, it is evident that the SMD algorithm requires more time to calculate the eigenvalues compared to the neural network approaches. Among the ANN methods, the FFN is noted to be faster in computation compared to CNN.

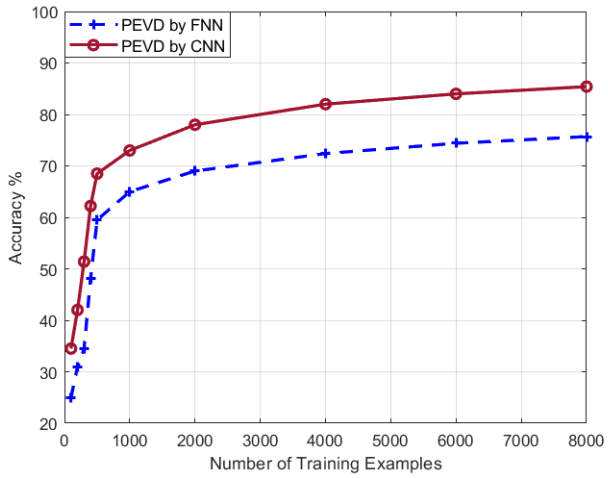

Fig. 2. The MAE accuracy of the PEVD calculation is assessed using both FFN and CNN, as a function of the number of data matrices utilized for training the system.

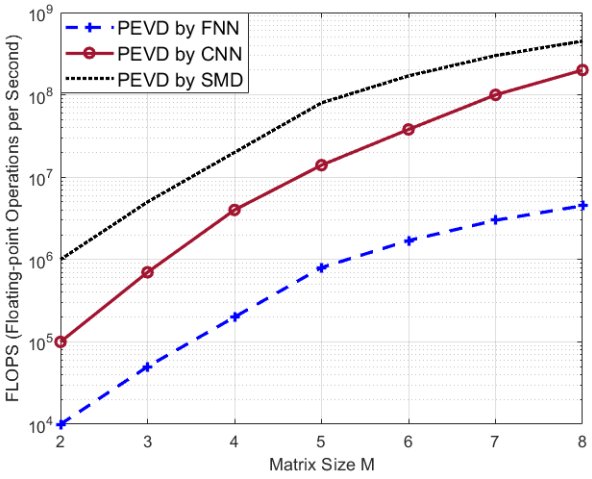

Fig. 4. Floating-point operations per second (FLOPS) analysis to calculate the eigenvalues across FFN, CNN, and SMD algorithm relative to varying input matrix size

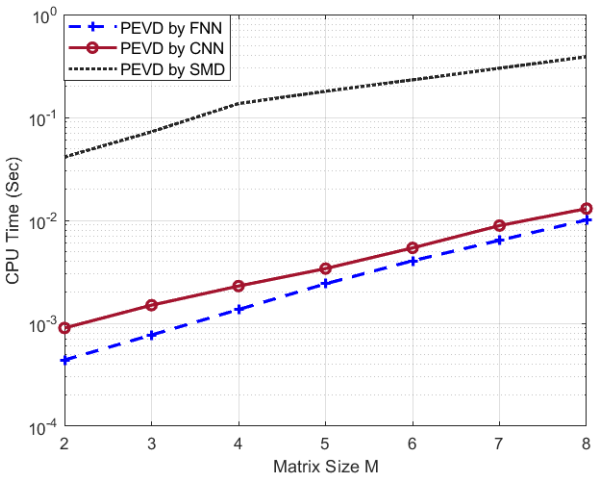

Fig. 3. Time complexity analysis to calculate the eigenvalues across FFN, CNN, and SMD algorithm relative to varying input matrix sizes

In Figure 4, the complexity in terms of FLOPS for calculating the polynomial eigenvalues of a matrix is evaluated with respect to varying input matrix sizes. It is evident that as the size of the matrix increases, the number of FLOPS required for computation also increases. Additionally, the SMD algorithm demands significantly more operations to compute the eigenvalues compared to the neural network approaches. Among the ANN methods, the FNN demonstrates faster computation times compared to the CNN. On the other hand, the ANN approach demonstrates lower accuracy in predicting eigenvalues compared to the SMD algorithm.

V. Conclusion and Future Work

This paper introduces two innovative artificial neural network (ANN)-based methods for computing polynomial eigenvalues: the feed-forward neural network (FNN) and the convolutional neural network (CNN). These models are trained using polynomial eigenvalues obtained from the sequential matrix diagonalisation (SMD) algorithm. Our results indicate that ANN-based methods, particularly the CNN model, provide a promising alternative to traditional iterative PEVD algorithms, achieving high accuracy in polynomial eigenvalue estimation while maintaining favorable computational efficiency. The CNN model significantly surpasses the FNN in accuracy due to its enhanced capability to process input data as scalar matrices and effectively extract features through convolution, preserving the spatial relationships inherent in the polynomial matrix. While the FNN demonstrates marginally faster computation times, it compromises significantly on accuracy, underscoring the trade-off between computational speed and precision in polynomial eigenvalue estimation. Moreover, the proposed ANN-based methods require fewer operations for polynomial-eigenvalue computation compared to the more traditional SMD algorithm.

References

[1] J. G. McWhirter, P. D. Baxter, T. Cooper, S. Redif, and J. Foster, "An evd algorithm for para-hermitian polynomial matrices," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 2158–2169, 2007.

[2] S. Redif, J. G. McWhirter, and S. Weiss, "Design of fir paraunitary filter banks for subband coding using a polynomial eigenvalue decomposition," *IEEE Transactions on Signal Processing*, vol. 59, no. 11, pp. 5253–5264, 2011.

[3] S. Redif, S. Weiss, and J. G. McWhirter, "Relevance of polynomial matrix decompositions to broadband blind signal separation," *Signal Processing*, vol. 134, pp. 76–86, 2017.

[4] V. Neo, S. Redif, J. G. McWhirter, J. Pestana, I. K. Proudler, S. Weiss, and P. A. Naylor, "Polynomial eigenvalue decomposition for multichannel broadband signal processing," *Signal Processing Magazine*, vol. 40, no. 7, pp. 18–37, 2023.

[5] S. Redif, S. Weiss, and J. G. McWhirter, "Sequential matrix diagonalization algorithms for polynomial evd of parahermitian matrices," *IEEE Transactions on Signal Processing*, vol. 63, no. 1, pp. 81–89, 2014.

[6] M. Tohidian, H. Amindavar, and R. A. M., "A dft-based approximate eigenvalue and singular value decomposition of polynomial matrices,"

Journal on Advances in Signal Processing, vol. 2013, no. 1, pp. 1–16, 2013.

[7] S. Weiss, I. K. Proudler, and F. K. Coutts, "Eigenvalue decomposition of a parahermitian matrix: Extraction of analytic eigenvalues," *IEEE Transactions on Signal Processing*, vol. 69, no. 1–5, pp. 722–737, 2021.

[8] S. Weiss, I. K. Proudler, F. K. Coutts, and F. Khattak, "Eigenvalue decomposition of a parahermitian matrix: Extraction of analytic eigenvectors," *IEEE Transactions on Signal Processing*, vol. 71, pp. 1642–1656, 2023.

[9] S. Weiss, S. Bendoukha, A. Alzin, F. Coutts, I. Proudler, and J. Chambers, "Mvdr broadband beamforming using polynomial matrix techniques," in *European Signal Processing Conference (EUSIPCO)*. EUSIPCO, 2015, pp. 839–843.

[10] S. Redif, "Convolutive blind signal separation via polynomial matrix generalised eigenvalue decomposition," *Electronics Letters*, vol. 53, pp. 87–89, 2017.

[11] C. H. Ta and S. Weiss, "A design of precoding and equalisation for broadband mimo systems," in *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*. IEEE, 2007, pp. 1616–1620.

[12] D. Hassan, S. Redif, and S. Lambotharan, "Polynomial matrix decompositions and semi-blind channel estimation for mimo frequency-selective channels," *IET Signal Processing*, vol. 13, no. 3, pp. 356–366, 2019.

[13] N. Moret, A. Tonello, and S. Weiss, "Mimo precoding for filter bank modulation systems based on psvd," in *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, 2011, pp. 1–5.

[14] D. Hassan, S. Redif, and S. Lambotharan, "Polynomial GSVD beamforming for two-user frequency-selective MIMO channels," *IEEE Transactions on Signal Processing*, vol. 69, pp. 948–959, 2021.

[15] V. W. Neo, C. Evers, and P. A. Naylor, "Speech enhancement using polynomial eigenvalue decomposition," in *Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*. IEEE, 2019, pp. 125–129.

[16] V. W. Neo, S. Weiss, and P. A. Naylor, "A polynomial subspace projection approach for the detection of weak voice activity," in *2022 Sensor Signal Processing for Defence Conference (SSPD)*, 2022, pp. 1–5.

[17] S. Kasap and S. Redif, "Novel field-programmable gate array architecture for computing the eigenvalue decomposition of para-hermitian polynomial matrices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 522–536, 2014.

[18] S. Redif and S. Kasap, "Novel reconfigurable hardware architecture for polynomial matrix multiplications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 3, pp. 454–465, 2015.

[19] F. Coutts, J. Corr, K. Thompson, I. Proudler, and S. Weiss, "Divide-and-conquer sequential matrix diagonalisation for parahermitian matrices," in *2017 Sensor Signal Processing for Defence Conference (SSPD)*, 2017, pp. 1–5.

[20] F. K. Coutts, I. K. Proudler, and S. Weiss, "Efficient implementation of iterative polynomial matrix evd algorithms exploiting structural redundancy and parallelisation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 12, pp. 4753–4766, 2019.

[21] N. Samardzija and R. Waterland, "A neural network for computing eigenvectors and eigenvalues," *Biological Cybernetics*, vol. 65, no. 4, pp. 211–214, 1991.

[22] A. Cichocki and R. Unbehauen, "Neural networks for computing eigenvalues and eigenvectors," *Biological Cybernetics*, vol. 68, pp. 155–164, 1992.

[23] Z. Yi, Y. Fu, and H. J. Tang, "Neural networks based approach for computing eigenvectors and eigenvalues of symmetric matrix," *Computers & Mathematics with Applications*, vol. 47, no. 8-9, pp. 1155–1164, 2004.

[24] D. Finol, Y. Lu, V. Mahadevan, and A. Srivastava, "Deep convolutional neural networks for eigenvalue problems in mechanics," *International Journal for Numerical Methods in Engineering*, vol. 118, no. 5, pp. 258–275, 2019.

[25] Z. Hu, "Estimation and application of matrix eigenvalues based on deep neural network," *Journal of Intelligent Systems*, vol. 31, no. 1, pp. 1246–1261, 2022.

[26] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[27] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[28] C.-H. Hsu and U. Kremer, "The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003, pp. 38–48.

[29] N. Mohammadi, "Evaluation of gpu performance compared to cpu for implementing algorithms with high time complexity," *American Journal of Software Engineering and Applications*, vol. 9, no. 2, pp. 10–14, 2016.

[30] S.-C. Hsia, S.-H. Wang, and C.-Y. Chang, "Convolution neural network with low operation flops and high accuracy for image recognition," *Journal of Real-Time Image Processing*, vol. 18, no. 4, pp. 1309–1319, 2021.

[31] M. E. Paoletti, J. M. Haut, X. Tao, J. Plaza, and A. Plaza, "Flop-reduction through memory allocations within cnn for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 7, pp. 5938–5952, 2020.