Tyrväinen, Juho; Lehtovuori, Anu; Ylä-Oijala, Pasi; Viikari, Ville

Applying Neural Networks for Predicting Feed Weights of an Antenna Array

Published in:
18th European Conference on Antennas and Propagation, EuCAP 2024

DOI:
10.23919/EuCAP60739.2024.10501353

Published: 01/01/2024

Document Version
Publisher's PDF, also known as Version of record

Applying Neural Networks for Predicting Feed Weights of an Antenna Array

Juho Tyrväinen*, Anu Lehtovuori*, Pasi Ylä-Oijala*, Ville Viikari*
*Department of Electronics and Nanoengineering, Aalto University, Espoo, P.O. Box 15500, 00076 Aalto, Finland,
ville.viikari@aalto.fi

***Abstract*—Artificial neural networks (ANN) are applied to find appropriate phasing for beam steering of patch antenna arrays. To obtain this goal, an ANN is trained using simulated far-field data. Two loss functions are tested and their performance is compared with a uniform and a non-uniform antenna array. The results show that in most cases both tested loss functions perform well, but are not able to find the best solution in all cases.**

***Index Terms*—antenna, antenna array, beam steering, machine learning, artificial neural network.**

I. Introduction

The use of artificial intelligence (AI) and machine learning (ML) has raised a lot of interest recently also in the antenna community [1]–[3]. Introducing higher frequencies and larger antenna arrays to common use have increased computational effort needed to determine optimal phases for multiple feeds in all use scenarios. Neural networks are expected to provide a powerful optimization tool especially in that kind of problems involving a large amount of data.

Neural networks can be used to generate different antenna structures, which decreases the need for strong knowledge of antenna design. For example, a deep neural network-based framework for designing multiband microstrip antennas given a desired impedance matching spectrum is presented in [4]. The use of neural networks might also introduce new type of antenna structures as described in [5].

Another approach is optimizing the performance of the given antenna design by utilizing neural networks e.g. to form the desired radiation pattern in terms of high gain, low sidelobes, or shaped beams. Antenna array pattern synthesis is typically time consuming and computationally heavy task. With an ANN approach a real time synthesis of antenna pattern is potentially possible as proposed in [6]. ANNs can be also used to accelerate phase calibration as presented in [7]. The goal can also be a specific sidelobe level ratio [8]. Naturally, the use of ANN is not limited only to antennas in free space. Electromagnetic power density value is determined in [9] to evaluate human exposure from millimeter wave (mmWave) mobile devices based on the phase conditions of the mmWave array antenna.

In this paper, we study the effectiveness of neural networks to solve a simple antenna array problem using generally available ANN tools. We aim to give a concrete example and introduce the details and issues faced during the study.

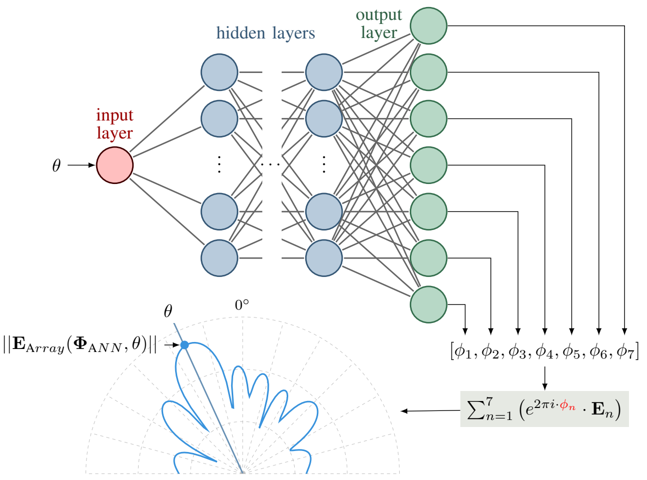

hidden layers
output layer
input layer
θ
θ
$0°$
$||\mathbf{E}_{Array}(\mathbf{\Phi}_{ANN}, \theta)||$
$[\phi_1, \phi_2, \phi_3, \phi_4, \phi_5, \phi_6, \phi_7]$
$\sum_{n=1}^{7}\left(e^{2\pi i \cdot \phi_n} \cdot \mathbf{E}_n\right)$

Fig. 1. Structure of the ANN (some layers and nodes are not shown for clarity) used to optimize the feed weights of the antenna arrays. The ANN receives a target angle θ as an input and outputs 7-phase values for each input port. The resulting far field is calculated and sampled in the target direction θ in the loss functions.

Especially, two different loss functions are tested to compare their capability to solve a known beam steering coefficients. The obtained far-field results at different beam steering angles are analyzed.

II. Method

Multiple libraries can be used to implement ML techniques such as ANN. In this study, we use Tensorflow and Keras ML libraries because they have excellent documentation and are, therefore easy to utilize. Other popular libraries include Pytorch and Matlab's Machine learning toolbox.

We create an ANN which is designed to calculate the port phases necessary to steer the beam of an antenna array to a specified direction as described in Fig. 1.

The training can be done either using a supervised training where the correct answer is provided for each input during training or using the unsupervised learning where the correct answer is not provided. We use the latter one in our study.

A. Training dataset preparation

A large amount of data has to be generated to train the ANN. To train the ANN, a dataset is created containing the complex field strength data output from a far-field simulation.

The simulation is run using CST studio suite, and the resulting far-field values are processed using a Matlab script. This data is imported into our training code, which is written in Python. For simplicity, our study is limited to steering a linear antenna array in a single target direction.

The Matlab code returns two outputs for the training code. The first one is a vector

$$\boldsymbol{\Theta} = [\theta_1, \ldots, \theta_M]^T,\ \theta_m \in \left[\frac{-180^\circ}{180}, \frac{180^\circ}{180}\right], \tag{1}$$

which contains the normalized angles for the field strength samples on the polar plane, and the second one is a matrix

$$\mathbf{E}_{\text{fields}}(\theta) = [\mathbf{E}_1(\theta), \ldots, \mathbf{E}_N(\theta)],\ \mathbf{E}_n = \begin{bmatrix} E_\theta & E_\phi \end{bmatrix} \tag{2}$$

contains the field strengths for N ports sampled in the direction θ. These outputs are used to create the Tensorflow training dataset. The code in Fig. 2 shows the commands used to generate data and randomize it for achieving improved results as explained next.

The command `.from_tensor_slices((a, b))` creates a new dataset from two arguments: a that contains the values the ANN receives as an input and b that contains the value the loss function receives. In our case, the field vector $\mathbf{E}_{\text{fields}}$, with a given angle θ, is input as parameter b for calculating the field strength achieved for the antenna array. In supervised learning, the right answer would be provided in b.

It is noticed that randomizing the data input to the ANN improves training results considerably. This can be achieved with command `.shuffle(n, )`. Shuffle-command randomizes the order of dataset, and with `reshuffle_each_iteration` set to True, also randomizes the order between training iterations. This ensures, when the dataset is divided into smaller training batches, they contain random subset of target directions and not just adjacent direction values. Parameter `n` defines in which size batches randomization is done to limit the number of datapoints needed to be loaded in memory. For a small set of possible directions used in our studies, `n` is set so that the whole dataseries can be randomized at once.

The dataset is also duplicated to improve training with a `.repeat(n)` command. This command duplicates the dataset `n` times, which can lead to improved results with small datasets. Lastly, the dataset is divided into smaller batches by running `.batch(n)`. This divides the dataset to batches with size n.

B. Proposed Loss functions

The main part of this paper focused on testing two different loss functions. These functions are compared in performance against progressive phase shift method. The loss functions should satisfy three general conditions: 1) decreasing value should correspond to a better performing ANN, 2) the function should be written using vector and matrix operations supplied by the Tensorflow library, and 3) the loss function should be differentiable because the used gradient descent method requires calculating gradient of the loss function. It is also important to keep in mind that a loss function is executed for each batch in one operation. That means that the function receives a vector with the length of a batch containing ANN prediction and datasets argument b.

```
dataset = tf.data.Dataset
    .from_tensor_slices(
        (theta, fields)
    )
    .shuffle(
        200,
        reshuffle_each_iteration=True
    )
    .repeat(2)
    .batch(32)
```

Fig. 2. Code for generating, randomizing the order, and pre-processing the simulation data to create a TensorFlow training dataset.

The loss function A, from here on called σ_A, calculates the absolute field strength in the target direction and the loss value is the inverse of the field strength. This loss function receives two input arguments. The first input is a vector

$$\boldsymbol{\Phi}_{ANN} = \begin{bmatrix} \phi_1\,, \ldots\,, \phi_N \end{bmatrix} \tag{3}$$

that contains a phase shift value for each port predicted by the ANN. These phase shift values are normalized to $[-1,\ 1]$. The second input is the vector in Eq. (2) for the target direction θ provided as an input for the ANN.

The far-field of an antenna array is the sum of individual fields of each antenna element (or port), phase shifted by the output value from the ANN. This far-field value is calculated as follows:

$$\mathbf{E}_{Array}(\boldsymbol{\Phi}_{ANN}, \theta) = \sum_{n=1}^{N} \left(e^{2\pi i \cdot \phi_n} \cdot \mathbf{E}_n(\theta)\right), \tag{4}$$

where $\mathbf{E}_{Array}(\boldsymbol{\Phi}_{ANN}, \theta)$ contains the θ and ϕ components, E_θ and E_ϕ, of the far field for direction θ. The absolute field strength is then calculated by taking the norm of (4), and the loss value is defined as the inverse of it

$$\sigma_A = \frac{1}{||\mathbf{E}_{Array}(\boldsymbol{\Phi}_{ANN}, \theta)||}. \tag{5}$$

This definition ensures that increasing the field strength in the target direction θ decreases the loss value.

For the second loss function, σ_B, maximum field achieved by the progressive phase shift is first calculated starting with the phase shift vector

$$\boldsymbol{\Phi}_{prog} = \left[\tfrac{-(N-1)p}{2}, \ldots\,, -p\,, 0\,, p\,, \ldots\,, \tfrac{(N-1)p}{2}\right], \tag{6}$$

where $p = 1/(N-1)$ is the normalized phase between the elements and N, the number of the ports, is assumed to be an odd number. With this phase shift vector, maximum field achieved is calculated

$$\mathbf{E}_{max}(\Theta) = \big[\max_p ||\mathbf{E}_{Array}\left(\boldsymbol{\Phi}_{prog}, \theta_1\right)||\,, \ldots\,,$$

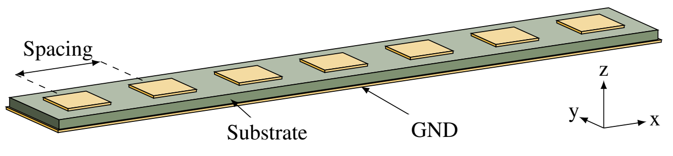


Fig. 3. 3D view of the studied uniform antenna array with seven square elements, a substrate, and ground plane (GND).

$$\max_{p} ||\mathbf{E}_{\text{Array}}\left(\mathbf{\Phi}_{prog}, \theta_M\right)||], \tag{7}$$

where p, as defined in equation (6), gets 360 values from -1 to 1. This vector contains maximum field strength achieved in each direction θ, with the progressive phase shift. This is an input for the σ_B function in addition to the same input parameters as function σ_A receives. The loss value is then calculated as

$$\sigma_B = \max_{p} ||\mathbf{E}_{Array}\left(\mathbf{\Phi}_{prog}, \theta\right)|| - ||\mathbf{E}_{\text{Array}}(\Phi_{ANN}, \theta)||, \tag{8}$$

where $\mathbf{\Phi}_{ANN}$ contains the phase shift values obtained as an output from the ANN and $\max_p ||\mathbf{E}_{Array}\left(\mathbf{\Phi}_{prog}, \theta\right)||$ is the maximum field strength achieved with the progressive phase shift in the direction θ provided as an input to the ANN.

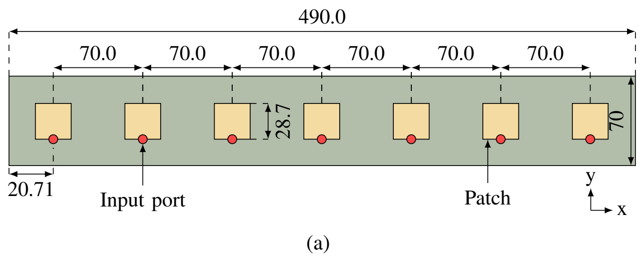
III. Results

This section first introduces the antenna arrays used in testing the loss functions. Then the procedure for creating the ANN model is shown. After that the results achieved from the trained networks are presented for four different cases. In these studies, two loss functions defined above are compared when ANN is trained on steering a uniform antenna array, and the obtained results are compared with the ANN trained with a non-uniform array.

A. Used antenna arrays

For the first part of the study, a 7-element linear antenna array is created and simulated. The array is composed of square microstrip patch elements as shown in Fig. 3. The patches are fed with discrete ports placed at the edge of the patches. Substrate FR-4 is used for its prevalence in printed circuit board (PCB) manufacturing. The patches and the ground plane are made of copper. A single patch is optimized for the frequency of 2.5 GHz. This resulted in a patch width of 28.7 mm. The simulations for a single antenna element show good matching at 2.5 GHz as S_{11}-parameter is -28 dB at that frequency.

The dimensions of antenna arrays are shown in Fig. 4. The spacing between the patches in a uniform array is 70 mm. For comparison, with $\lambda/2$ spacing, where λ is the free space wavelength, the distance between the patches is around 60 mm. For the second part of the study, an array with a non-uniform spacing is created. For this array, spacings between elements are 55, 70, 85, 60, 70, and 80 mm. The total length of the arrays (the distances between the outermost elements) is the same.



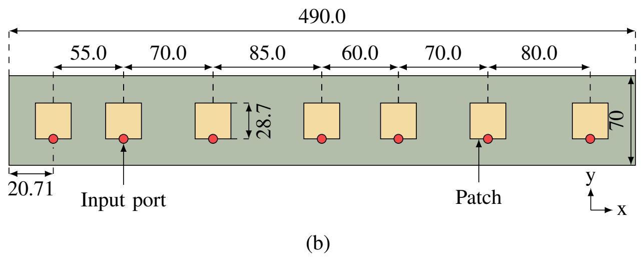


Fig. 4. Simulated antenna arrays with seven patches and discrete input ports, marked with red dots. a) Uniform antenna array b) Non-uniform antenna array. The dimensions are in mm.

B. Creating and training the ANN model

Creating the ANN model is straightforward using Tensorflow sequential function as presented in Fig. 5. First, an input layer is defined using function `tf.keras.layers.Input(n)`, where `n` is the number of inputs. For our study, ANN has just one input, that is the target beam direction. Secondly, after the input layer, fully connected layers are added using `tf.keras.layers.Dense(c, activation=<>)` function where `c` is the number of nodes in the layer and activation defines the used activation function. Between dense layers `tf.keras.layers.Dropout(r)` are used. They turn off nodes by turning their output off randomly with the rate of `r`. This is used to limit overtraining, especially in supervised training, and it can lead to better outcomes. Thirdly, the last layer should have as many nodes as outputs are needed. In our implementation, the number of nodes agrees with the number of ports of the antenna array.

A 6-layer ANN model is created containing four hidden layers, and one input and one output layer. Other important parameters for the ANN model are shown in Table I. The ANN is trained using Adam optimizer, which is an algorithm used in the gradient descent methods. This optimizer is memory efficient and computationally inexpensive to run and scalable to more complex problems [10].

C. Comparing different loss functions

First, the ANN is trained to steer the uniform antenna array. In training with the σ_A function, the training is run for 350 epochs. For comparing the results achieved with the ANN, the progressive phase shift maximum field E_{max} presented in Eq. (7) is used as a reference result. For the ANN result, the field achieved in direction $\theta \in [-180°, \ldots, 180°]$ is shown in Fig. 6a with the target direction corresponding to the same direction. The difference between the reference result and the

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(1),
    tf.keras.layers.Dense(64,
        activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128,
        activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128,
        activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(128,
        activation='relu'),
    tf.keras.layers.Dense(portCount,
        activation=None)
])
```

Fig. 5. Code for creating the ANN model with four hidden layers, and one input and one output layer.

TABLE I
PARAMETERS OF THE NEURAL NETWORKS

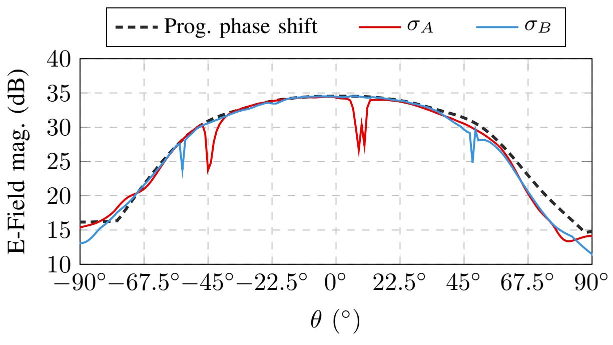
No. of hidden layers	4
Nodes in hidden layers	64, 128, 128, 128
Dropout rate	0.2
Activation function in hidden layers	ReLU
Activation function in output layer	Linear
Batch size	180
Optimizer	Adam
Learning rate	0.001

ANN field is shown in Fig. 6b. For comparison, the mean of the difference is calculated and shown as a difference graph.

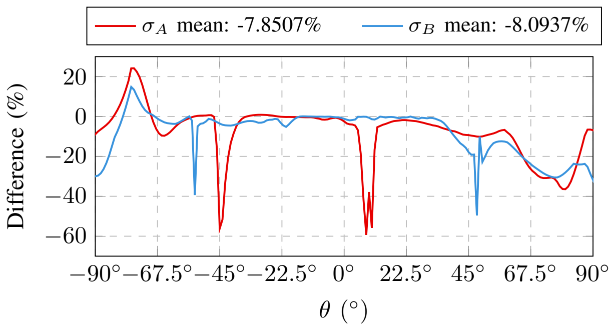
The ANN with σ_A performs mostly well, achieving a field strength comparable to the progressive phase shift method. Remarkable differences emerge at two steering angles. The reason is that the ANN aims to steer the side lobe to the target direction and maximizes it instead of the main beam. The difference grows for positive θ angles, which is caused by the main beam being slightly of the center of the target.

Next, the σ_B function is trained for 1000 epochs. The performance is similar to what is achieved with the σ_A. The difference curve is flatter for larger area in the center of the plot, achieving very good results for directions between -53–36 degrees.

Because the training of a neural network is a stochastic process, the result varies between runs, and multiple runs might be needed to achieve the best results. This can be mitigated by a better loss function and task formulation. Neural Networks work more efficiently with tasks containing one to one mapping between inputs and the outputs [6], meaning that for one input there is just one right output. This is because neural networks are shown to work as universal function approximators with enough layers [6], [11].



(a)



(b)

Fig. 6. Results for the uniform antenna array: a) Field strength achieved by the ANN trained with the two loss functions and the field achieved with the progressive phase shift. b) Difference between the progressive phase shift and the ANN results.

D. Non-uniform antenna array

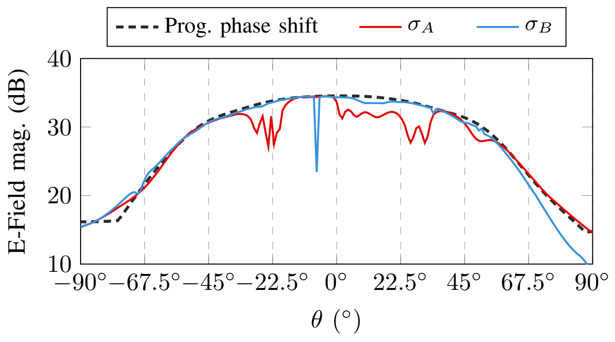
In the second study, the ANN is tested for the non-uniform antenna array presented in Fig. 4b. We compare the results with the E_{max} achieved with progressive phase shift on the uniform array.

The results with the σ_A are shown in Fig. 7a. They show similar problem with the ANN maximizing the side lobe, as mentioned in Section III-C. At some steering angles, the achieved field strength is 0%, but for many directions the resulting field is around -40% of the reference result.

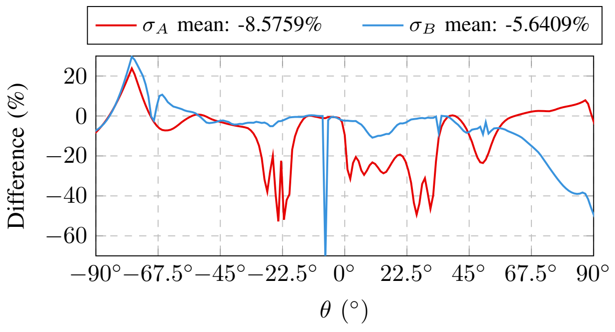
With σ_B function the results are more satisfactory and the ANN performs closely to what is achieved with the progressive phase shift. Training was run to epoch 1000, but the loss value stabilized earlier. Thus, the training could have been stopped earlier.

Figure 8 illustrates the radiation patterns for three target directions θ. The ANN performs well in most directions and the results match with the reference achieved using the progressive phase shift. At $-7°$ the ANN performance is, however, very poor. As can be seen from the results, the loss function fails to properly push the training to find the global minimum of the loss function.

Running the ANN model after the training is a very fast operation. The training times were around one and six minutes, and the inference (meaning the time taken to predict the phasing) is around 11 ns for one target angle input. The training and inference times for different loss functions are



(a)



(b)

Fig. 7. Results for the non-uniform antenna array: a) Field strength achieved by the ANN trained with the two loss functions for the non-uniform array. For the comparison case the field achieved with progressive phase shift with the uniform array b) Difference between the progressive phase shift and the ANN results.

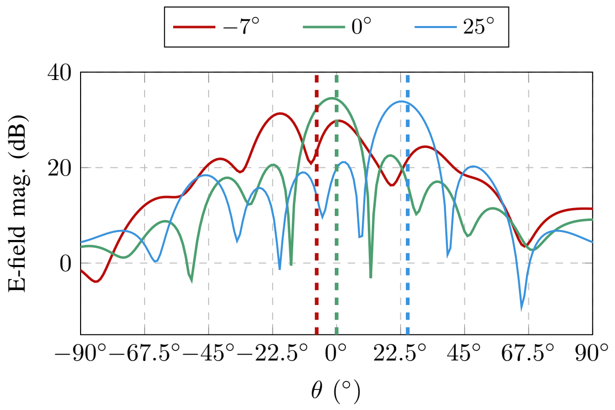


Fig. 8. Far-fields for the non-uniform array when the ANN is trained using σ_B with three different target angles (dotted lines). These target angles show a slight offset in the resulting main beam maximum direction and the target direction. The highest offset is observed with $-7°$ marked with red lines.

TABLE II
TRAINING AND INFERENCE TIME

Antenna array	Uniform array		Non-Uniform array	
Used loss function	σ_A	σ_B	σ_A	σ_B
ANN training time	42 s	6.28 min	55 s	6.25 min
Inference time / target angle	11 ns	12 ns	11 ns	11 ns

shown in Table II. All studies are run using a computer with Intel i5-12600K processor.

IV. Conclusion

We studied an ANN model with two simple loss functions to find optimal phases for the beam steering of a patch antenna arrays. The results indicate that in most cases the considered methods performed well, but are not able to find the global optimum in all cases. These results also highlighted the importance of a proper definition of the loss function used in the training. In summary, ML may not be used blindly to solve every challenging problem, rather a deep knowledge of it, and careful application dependent planning of the algorithm and training, may be required.

References

[1] H. M. El Misilmani, T. Naous, and S. K. Al Khatib, "A review on the design and optimization of antennas using machine learning algorithms and techniques," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 30, no. 10, p. e22356, 2020.

[2] F. Zardi, P. Nayeri, P. Rocca, and R. Haupt, "Artificial intelligence for adaptive and reconfigurable antenna arrays: A review," *IEEE Antennas and Propagation Magazine*, vol. 63, no. 3, pp. 28–38, 2021.

[3] S. K. Goudos, P. D. Diamantoulakis, M. A. Matin, P. Sarigiannidis, S. Wan, and G. K. Karagiannidis, "Design of antennas through artificial intelligence: State of the art and challenges," *IEEE Communications Magazine*, vol. 60, no. 12, pp. 96–102, 2022.

[4] A. Gupta, E. A. Karahan, C. Bhat, K. Sengupta, and U. K. Khankhoje, "Tandem neural network based design of multiband antennas," *IEEE Transactions on Antennas and Propagation*, vol. 71, no. 8, pp. 6308–6317, 2023.

[5] F. Mir, L. Kouhalvandi, L. Matekovits, and E. O. Gunes, "Automated optimization for broadband flat-gain antenna designs with artificial neural network," *IET Microwaves, Antennas & Propagation*, vol. 15, no. 12, pp. 1537–1544, 2021.

[6] C. Cui, W. T. Li, X. T. Ye, P. Rocca, Y. Q. Hei, and X. W. Shi, "An effective artificial neural network-based method for linear array beampattern synthesis," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 10, pp. 6431–6443, 2021.

[7] T. Iye, P. van Wyk, T. Matsumoto, Y. Susukida, S. Takaya, and Y. Fujii, "Neural network-based phase estimation for antenna array using radiation power pattern," *IEEE Antennas and Wireless Propagation Letters*, vol. 21, no. 7, pp. 1348–1352, 2022.

[8] T. Naous, A. A. Merie, S. K. A. Khatib, M. Al-Husseini, R. M. Shubair, and H. M. El Misilmani, "Machine learning-aided design of dielectric-filled slotted waveguide antennas with specified sidelobe levels," *IEEE Access*, vol. 10, pp. 30 583–30 595, 2022.

[9] J. Bang and J. H. Kim, "Predicting power density of array antenna in mmwave applications with deep learning," *IEEE Access*, vol. 9, pp. 111 030–111 038, 2021.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[11] S. Liang and R. Srikant, "Why deep neural networks for function approximation?" 2017.