Direction of Arrival Estimation Applied to Antenna Arrays using Convolutional Neural Networks

Giorgos Kokkinis[(1)], Zaharias D. Zaharis*[(1)], Pavlos I. Lazaridis[(2)], and Nikolaos V. Kantartzis[(1)]
(1) School of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece.
(2) School of Computing and Engineering, University of Huddersfield, Huddersfield HD1 3DH, U.K.

Abstract

In this paper, an effort is made to solve the direction of arrival (DoA) estimation problem by constructing a convolutional neural network (CNN) architecture, which estimates the angles of arrival of the incoming source signals received by a uniform linear array (ULA) antenna. The input of the CNN is the sampled correlation matrix of the signals, while the the output is a pool of the highest probabilities of the network's estimated values. The problem is modeled as a multi-label classification task, meaning that the space of angles is divided into a grid of multiple classes. To model the problem in this way, we assume that we cannot have two or more signals coming from the same angle. This also allows us to further increase the quality of our predictions, meaning that we can set an a priori minimum distance between each given output. In this way we can filter out duplicate outputs and have the desired result.

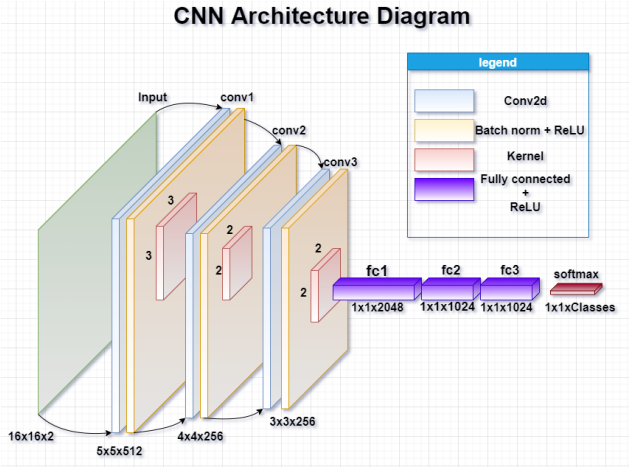

Figure 1. Basic diagram of the CNN layout.

1 Introduction

In the near future, mobile telecommunication systems will require high precision and low complexity methods in order to solve the direction of arrival (DoA) estimation problem. There have been several mathematical methods for DoA estimation [1], namely the conventional Delay-and-Sum (DS), Minimum Variance Distortionless Response (MVDR), Multiple Signal Classification (MUSIC), Estimation of Signal Parameters via Rotational Invariance Technique (ESPRIT), Unitary-ESPRIT and Fourier Transform method (FT-DoA). The above methods estimate the angles of arrival (AoAs) of incoming signals using high complexity mathematical algorithms that require both computing power and considerable processing time. Recently, neural network (NN) solutions have arisen in order to reduce these costs while providing even higher quality results. State of the art architectures exploit spatial features that can be extracted from the input signals with the use of convolutional computing. Hence, a convolutional NN (CNN) seems to be a relevant choice to solve DoA estimation problem.

2 Formulation

The CNN consists of 3 convolution layers in sequence as shown in Fig. 1, with the first one having a 3×3 convolution kernel, and the rest having a 2×2 kernel [2]. The kernel stride is a single step on the layers except the first one, where we set the stride equal to 3. This is because the network's performance does not change significantly with that small loss of information, while it greatly reduces the training time. Following each convolution layer, we have a batch normalization layer. Batch normalization [3], [4] is a common technique used in NNs. The purpose of this architecture is to reduce adaptation to the numeric values of the training set. The normalization layers improve that result by regularizing input data during training.

Following the convolution layers, the output is then flattened and passed sequentially through 3 fully connected (FC) linear layers. Note that after each convolution or FC layer, an activation function layer must be added. Activation functions are added in order to map the result within a range of desired values. Most of the activation functions are divided into linear and non-linear type. In our case, we use the Rectified Linear Unit (ReLU) activation function, which is a non linear type. Following the first two fully connected layers, we use a dropout [5] layer, which shuts down 20% of the neurons in a randomized manner, in order to prevent the network from creating a robust model based on the values of the training set. This gives worse results during training, but it improves evaluation metrics during validation.

3 Network Training Process

To perform both training and validation of the CNN, we use similar data examples. Each data record consists of various numbers of signals ranging, from 1 to 10, with AoAs between $30^\circ - 150^\circ$ with respect to the antenna array axis, and values rounded to the first decimal digit. As mentioned above, we set a minimum angle deviation between signals, i.e. $\Delta\theta > 6^\circ$. We split the data set into two subsets for training and validation with respective percentages of 90% and 10%. The correlation matrices of the signals induced at the input of the array elements are used in the both subsets. These matrices are theoretically calculated by using the well-known formula [6]:

$$\mathbf{R}_{xx} = \mathbf{A}\mathbf{R}_{ss}\mathbf{A}^H + \mathbf{R}_{nn} \,, \tag{1}$$

where $\mathbf{A}$ is the steering matrix of the ULA, $\mathbf{R}_{ss}$ is the correlation matrix of the source signals and $\mathbf{R}_{nn}$ is the correlation matrix of the noise signals received at the input of the array elements.The signal-to-noise ratio (SNR) is also taken into consideration for the construction of the matrices. It is worth mentioning that a practically useful CNN must be effective in both low (0 dB) and high (10 dB) SNR levels. For the construction of $\mathbf{R}_{xx}$, we use the sampled correlation matrix formula:

$$\mathbf{R}_{xx} = \frac{1}{T}\sum_{k=1}^{T}\mathbf{x}(k)\mathbf{x}^H(k) \,, \tag{2}$$

where T is the sample size and $\mathbf{x}$ is the input signal vector composed of all the signals induced at the inputs of the array elements. Note that the sample size depends on the SNR level and needs to be sufficient in order to approximate the theoretical value in (1). The sample correlation matrix formula gives us a more realistic view of the resulting output, as it is extracted from the actual input of the ULA antenna. The ULA consists of 16 elements at a distance $\lambda/2$, where λ is the free space wavelength.

After the correlation matrix $\mathbf{R}_{xx}$ is calculated, it is necessary to split the real and imaginary parts of the matrix and pass them as two parallel input channels to the CNN, since it performs poorly on complex number data. After the data passes through all the layers mentioned in the previous section (see Fig. 1), we get the NN output.

The training of the neuron weights is done in batches of data. The batch size is usually small ($2^5 - 2^7$). After each batch goes through the CNN, the total loss is propagated backwards into the network, shifting the weights of the neurons accordingly. The whole training process is performed iteratively, within a predetermined number of epochs. Also, the training set records are shuffled in order to prevent overfitting on numerical patterns.

The final output is a vector of size equal to the number of classes, that is, the grid resolution of the range of AoAs of the incoming signals. The index of each output value translates to an angle point of the grid. We have chosen a grid resolution of 0.25°. For example, if the angle space is $30^\circ - 150^\circ$, we get a grid vector $\mathbf{G} = \begin{bmatrix}30 & 30.25 & 30.5 & \dots & 149.5 & 149.75 & 150\end{bmatrix}$. This gives us 481 classes in total. The higher the number of classes that we set, the higher the accuracy of the output, compared to the true values of the data. However, more classes increase the probability of errors, which means that more training time is required to achieve a sufficient result.

The final step is to compare the predicted output with the truth values. The truth values are one-hot encoded into labels. The labels compose a vector of size equal to the number of classes, and its values are set equal to 0, except the elements that match with an AoA, which are set equal to 1. For example, using a grid resolution of 0.25°, the signal vector $\begin{bmatrix}30.5 & 31.25 & 31.75\end{bmatrix}$ would translate to the following label vector: $\begin{bmatrix}0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & \dots & 0\end{bmatrix}$

The actual values of the output are in a form of raw results (logits). Before we perform a loss function metric, we pass the logits through a softmax function, which converts the output values into probabilities, that is, the prediction of a signal coming from those angular values. We use the Binary Cross-Entropy Loss function (BCELoss), which is a standard metric for the multi-label classification problem [2]:

$$L = -\sum_{c=1}^{C} y_c \log(x_c) + (1-y_c)\log(1-x_c) \,, \tag{3}$$

$$Loss = \begin{cases} mean(L), \text{or} \\ sum(L) \end{cases}$$

where C is the number of classes, y_c is the label and x_c is the output prediction. Cross entropy measures the entropy between each output probability and its label. The term Binary means that it measures the entropy of each class independently of the others. This means that we can have multiple classes with a probability close to 1 in the output vectors and so we can achieve DoA estimation of multiple signals in one output.

While BCELoss is a sufficient loss function, we use the Root Mean Squared Error (RMSE) as the actual performance metric:

$$RMSE = \sqrt{\frac{1}{N}\frac{1}{K}\sum_{i=1}^{N}\sum_{j=1}^{K}(\hat{y}_{ij} - y_{ij})^2} \,, \tag{4}$$

where $\hat{y}$ denotes the ground truth and y denotes the prediction. Also, K is the number of incoming signals and N is the number of records of the data set used by the NN. This metric gives us a very good approximation of the error between prediction the ground truth values. For example, an RMSE equal to 0.1 means that we have achieved an average angular deviation of 0.1° from the actual AoAs of signals. Since

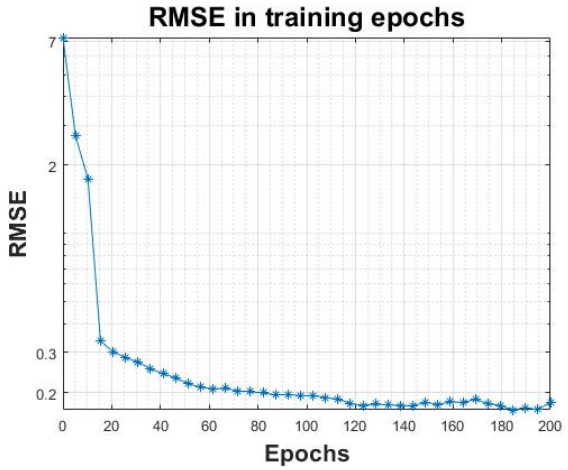

Figure 2. RMSE variation during training, when using a training set of 2.25×10^6 records, for 3 incoming signals and SNR =10dB.

the number of classes varies, the RMSE will also include the loss in accuracy due to poor grid resolution.

The code was written in Python, the environment used was Pytorch, which is a library dedicated for NNs. We also used torch.cuda(), a cuda wrapper that transmits data to the Graphics Processor Unit (GPU) in order to accelerate both the training and validation processes. The hardware used for both processes was an Intel i7-8700K CPU with a 16GB DDR4 RAM and a GPU NVidia Geforce GTX 1070Ti 8GB.

The training time varies from a few minutes to several hours. For the largest training set (i.e., 2.25×10^6), the CNN needs 11.6 hours for training.

4 Results

The performance of the proposed CNN is examined with respect to several key parameters. We parse over different SNR levels and multiple numbers of signals. The batch size during training and testing is 100. The process time of a single data record has been reduced to about $30\mu sec$, due to the use of GPU hardware acceleration.

An example of training progress is shown in Fig. 2. Here, the CNN is trained with 2.25×10^6 records for 200 epochs in order to find AoAs of 3 incoming signals in the presence of noise with SNR=10dB. During training, the RMSE gradually decreases to eventually approach very close to zero. This excellent result is due to the use of a large training set, but as mentioned earlier the training time is respectively long. If a smaller training set is used, then the training time will be reduced but a corresponding degradation will be observed in the final value of the RMSE.

In order to explore the behavior of the CNN with respect to a large number of parameters, we use every time a training set size that will provide us with a fine balance between performance and training time. When evaluating the CNN performance with respect to the SNR (Fig. 3), we use a set of 10^5 records for training. The same training set is used for the evaluation of the CNN with respect to the number of incoming signals (Fig. 4). In the end, when comparing the proposed CNN with another NN, we use a much larger training set (2.25×10^6 records), given that all parameters have a constant value.

As shown in Fig. 3, we examined the CNN over different levels of SNR. For the lowest SNR values, the CNN performs poorly (RMSE>5), because the sample size is very small for these noise levels. As the SNR increases, the RMSE seems to be reduced, while for high SNR values,

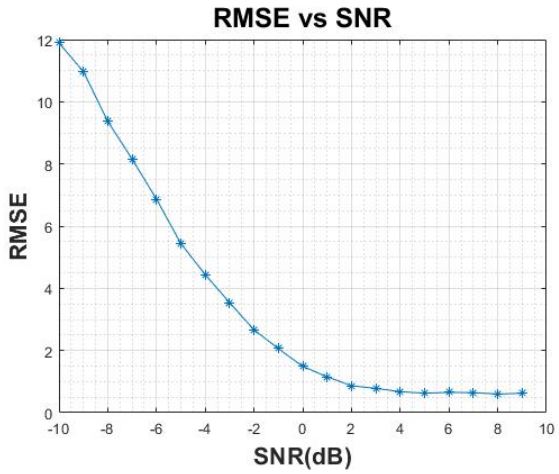

Figure 3. RMSE vs. SNR achieved by the proposed CNN trained with 10^5 records for 15 epochs. The antenna array receives 3 incoming signals ($K = 3$), while the sample size used to construct the correlation matrices is 1024 ($T = 1024$).

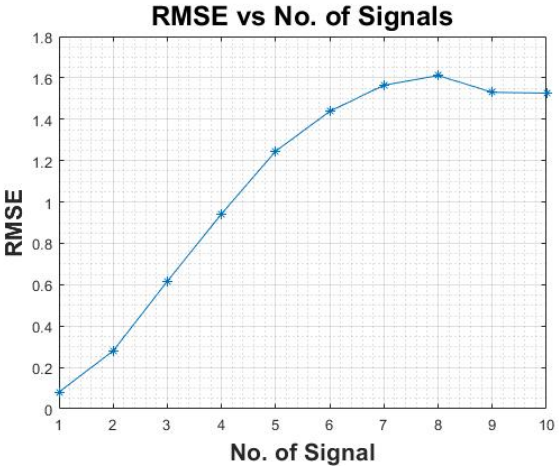

Figure 4. RMSE for various numbers of incoming signals. The proposed CNN was trained with 10^5 records for 15 epochs. SNR=10dB, while the sample size used to construct the correlation matrices is 128 ($T = 128$).

the angular deviations expressed by the RMSE are lower than half a degree. It is obvious that with a greater sample size, we will get much better results for negative values of SNR, but sample gathering and process time must also be included in the actual operational time and so the output might buffer.

Fig. 4 illustrates the CNN's performance with respect to the number of signals. Even for a large number of signals, the error is relatively small, only 1.6° at the worst case scenario.

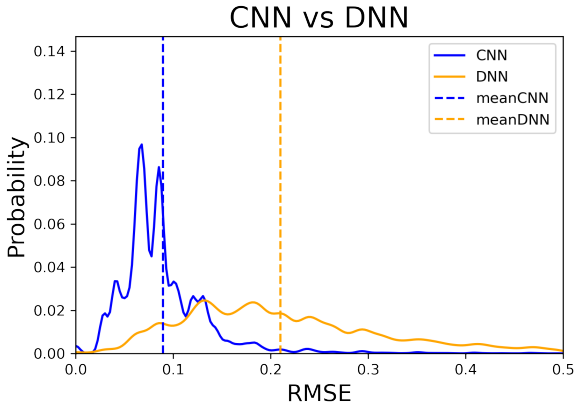

Figure 5. Comparative RMSE distributions derived by the proposed CNN and a DNN for a validation set of 2.5×10^5 records. Both NNs were trained with 2.25×10^6 records for 200 epochs. The antenna receives 3 incoming signals ($K = 3$) in the presence of noise with SNR=10dB, while the sample size used to construct the correlation matrices is 128 ($T = 128$).

In Fig. 5, we compare the robustness of the CNN over a dense NN (DNN). The latter is only composed of linear FC layers. The graphs shown in this figure are distributions of the RMSE derived respectively by the CNN and the DNN for the same validation set of 2.5×10^5 records. These distributions demonstrate the advantages of a CNN when handling spatial complexity. The convolution layers pass the feature maps into the FC layers, reducing pattern recognition based on numerical values. Since the RMSE is a quantitative metric, and NNs tend to make very few but large errors, the results that deviate from the distribution mean will inflate the total error. If we exclude the top 0.005% of the worst estimations, which is an extremely small part of the output, the proposed CNN provides a mean value of RMSE less than 0.1, as shown in Fig. 5.

5 Conclusion

In the near future, the deployment of 6G and higher generation telecommunication systems will require both swift and highly accurate computations in order to solve the DoA estimation problem. The CNN architecture proposed in this paper seems to meet future requirements. After proper training, the proposed CNN is able to effectively reduce the RMSE to values below 0.1, which provides excellent accuracy in DoA estimation, while the operational time required for calculations is quite low. By using the larger training set and increasing the training time accordingly, the CNN may achieve better accuracy (lower RMSE values) for a larger number of incoming signals and lower SNR values, without substantially affecting the operational time.

6 Acknowledgements

This research was supported by the European Union, partially through the Horizon 2020 Marie Skłodowska-Curie Innovative Training Networks Programme "Mobility and Training for beyond 5G Ecosystems (MOTOR5G)" under grant agreement no. 861219, and partially through the Horizon 2020 Marie Skłodowska-Curie Research and Innovation Staff Exchange Programme "Research Collaboration and Mobility for Beyond 5G Future Wireless Networks (RECOMBINE)" under grant agreement no. 872857.

References

[1] Gentilho E., Scalassara P. and Abrão T. "Direction-of-Arrival Estimation Methods: A Performance-Complexity Tradeoff Perspective." *Journal Of Signal Processing Systems*, vol. 92, no.2, pp. 239-256 August 2019, doi: 10.1007/s11265-019-01467-4

[2] G. K. Papageorgiou, M. Sellathurai and Y. C. Eldar, "Deep Networks for Direction-of-Arrival Estimation in Low SNR," in *IEEE Transactions on Signal Processing*, vol. 69, pp. 3714-3729, 2021, doi: 10.1109/TSP.2021.3089927.

[3] V. Thakkar, S. Tewary and C. Chakraborty, "Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data," *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pp. 1-5, 2018, doi: 10.1109/EAIT.2018.8470438.

[4] Sergey Ioffe and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ArXiv*, abs/1502.03167.

[5] Srivastava N., Hinton G., Krizhevsky A., Sutskever I. and Salakhutdinov R. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal Of Machine Learning Research*, vol.15, pp. 1929-1958, 2014, http://jmlr.org/papers/v15/srivastava14a.html

[6] Z. D. Zaharis, I. P. Gravas, P. I. Lazaridis, T. V. Yioultsis, C. S. Antonopoulos and T. D. Xenos, "An Effective Modification of Conventional Beamforming Methods Suitable for Realistic Linear Antenna Arrays," in *IEEE Transactions on Antennas and Propagation*, vol. 68, no. 7, pp. 5269-5279, July 2020, doi: 10.1109/TAP.2020.2977822.