Angle of Arrival Estimation using Convolutional Neural Networks in Nosy and Multipath Environments

Matthew Simiele
University of California Los Angeles
mjsimiele@ucla.edu

Abstract

This paper describes the use of convolutional neural networks (CNNs) for angle of arrival (AoA) estimation in various antenna arrays. Traditional signal processing techniques such as beamforming and multiple signal classification (MUSIC) have limitations in terms of dealing with noise and interference in their environment. In contrast the CNN's can learn complex features from the raw IQ data or even the covariance matrix, which is typically used when estimating the AoA using the MUSIC algorithm. Architectures for Recurrent Neural Networks (RNNs) are also tested and compared with traditional CNNs to the MUSIC algorithm to determine the performance of the models. The results demonstrate that CNNs outperform signal processing techniques in environments with a lot of noise, interference or mutipath, since these models were able to understand and learn from their environment. However, the CNNs performed a bit worse when in clean environments following training data being used from both clean and noisy environments.

1. Introduction

Antenna arrays have been widely used in many applications such as radar and wireless communication systems such as direction finders. One of the key functions in these design spaces is estimation the angle of arrival (AoA) of a signal incident on the antenna array, Traditionally signal processing techniques such as beamforming or MUSIC (Multiple Signal Classification) have been used to estimate the AoA. However these algorithms have limitations in terms of their accuracy and computational complexity, especially in the presence of noise and interference.

Convolutional neural networks (CNNs) have emerged as a powerful tool for signal processing and pattern recognition tasks. This paper describes the use of a CNN to perform AoA estimation in antenna arrays. Unlike traditional signal processing algorithms, CNNs can learn complex features and patterns from the raw IQ data or the covariance matrix, thereby improving the accuracy of AoA estimation. CNNs can handle noisy and distorted input signals and adapt to different environments and scenarios.

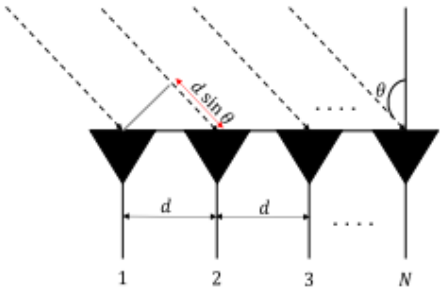

Figure 1. Incident Signals arriving at a ULA

This paper describes a detailed study of the use of CNNs to perform AoA estimation. This paper descibes a few architecture methods and data pre-processing methods used to facilitate easier learning and better AoA estimation errors. The performance of the CNNs are compared against the traditional signal processing techniques.

This introduction section has sub-parts to describe the MUSIC algorithm and CNN implementations. A subsection is added on data generation as the use of synthetic data was useful in determining the performance of all algorithms used to estimate the AoA.

1.1. Data Processing, Generation, and Signal Model

1.1.1 Signal Model

As described in more detail in the paper on Angle of Arrival Estimation in an Indoor Environment Using Machine Learning [1], the signal model of a Uniform Linear Array (ULA), which is depicted in Figure 1, is defined by the following equations:

$$\mathbf{Y} = \mathbf{A}\mathbf{x} + \mathbf{n} \tag{1}$$

Where $\mathbf{X}$ is the source signal, $\mathbf{A}$ is the steering matrix, and $\mathbf{n}$ is the noise in the environment, these can be described in vector form as:

$$\mathbf{x} = [x_1, x_2, \ldots, x_N]^T \tag{2}$$

$$\mathbf{n} = [n_1, n_2, \ldots, n_N]^T \tag{3}$$

$$\mathbf{A} = [\mathbf{a}(\theta_1), \mathbf{a}(\theta_2), \ldots, \mathbf{a}(\theta_L)]^T \tag{4}$$

The number of antenna elements in the ULA is equal to N and L is the number of propagation paths incident on the array.

One of the most common practices in AoA estimation is to convert an array of samples of $\mathbf{Y}$ into a covaiance as defined in Eq. 5

$$\mathbf{R_{YY}} = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathbf{Y}_i \mathbf{Y}_i^H. \tag{5}$$

The statistical operation, $(.)^H$, used in Eq. 5 denotes the Hermation operation.

1.1.2 Data Set Generation and Rationale

The python toolbox, doatools, was used to generate synthetic data. The author of this python tool box has used this resource to supplement their research and has used it in multiple publications, such as one on the Performance Analysis of MUSIC used in a Co-array [4].

Each data set of data generated by this tool box is labeled based on its array type and simulation run. Each simulation run contains varying AoAs for a single source as well as varying signal to noise ratios (SNRs).

The data generated is based on a ULA operating in a relatively clean environment, little multipath is present. The noise injected into the data set is modeled to be Additive Gaussian White Noise (AWGN) with its variance modified to increase or decrease the SNR.

The data set originally used to test the CNN's developed for this project were collected on the DIstributed CHAnnel Sounder by the University of Stuggart (DICHASUS) [2], which is a 32 element rectangular multi-input multi-output (MIMO) antenna array. However, this dataset did not perform well when using the MUSIC estimator and provided by the doatool python toolbox and thus I did not feel comfortable comparing the performance of my models against it. Further discussion for why the MUISC esimator did not handle this data well is in Section 4.

To match the number of elements present in the DICHASUS array the simulated data sets were generated using a ULA with 32 elements.

1.1.3 Data Processing

To trim down on size of the input to the model when using the time domain data $\mathbf{Y_S}$, with length of N_s. Pre-processing for both the DICHASUS and simulated data averaged the long time domain data in batches to produce the same input size to the model as the covariance matrix. This would allow easy transition between testing sets of data on the same models.

2. Model Architecture

2.1. MUSIC

MUISC is a method for estimation the AoA using the property that the signal subspace and noise subspace are orthogonal [3]. The formula used to implement the spectrum for MUSIC is as follows.

$$P_{MUSIC} = \frac{1}{\mathbf{A}^H U_N U_N^H \mathbf{A}} \tag{6}$$

In Eq. 6 $\mathbf{U_N}$ denotes the noise subspace.

2.2. Neural Network Architecture

The specific details on the architecture of the networks tested are included in the appendix, specifically table x, but the networks all followed the same general architecture. For the CNN the model included a few convolutional layers followed by a few fully connected layers. While the models using different RNN types included a layer of either LSTM or GRU and Simple RNN between the convolutional layers and the fully connected layers. This allowed the convolutional layers to decompose the spatial structure of the data and then allowed the RNN layer to decompose the temporal view of the input.

2.2.1 Using CNNs

One of the major perks of using the CNN is to extract spatial features from the input data. This allows the model to extract spatial patterns in the received signal.

The model using the CNN architecture is described in more detail in Table 1.

2.2.2 Adding a LSTM Layer

Long Short-Term Memory (LSTM) layers are made to process sequential data, similar to that of the raw data from the antenna array. The LSTM layer is added to the CNN model and used to extract sequential and temporal information from the input data which in turn captures the dependencies of the various input signals.

The model using the CNN+LSTM architecture is described in more detail in Table 2.

2.2.3 Using a GRU and Simple RNN Layer

Simple Recurrent Neural Networks (RNNs) and Gated Recurrent Units (GRU) were also used in the same way to process the sequential nature of the antenna array data. Compared to LSTMs the RNNs may have trouble capturing long-term memory. The GRU may have difficulty when the number of channels is large or the input signals are noisy.

The model using the CNN+GRU+RNN architecture is described in more detail in Table 3.

2.2.4 Model Parameters

For the test conducted and documented in figure 2 they used a drop out rate of 0.35 and the relu activation function for all layers except the last fully connected that used a linear activation function. Other activation functions that were tested included ELU and tanh but relu provided the best performance.

3. Results

Of the network architectures tested the CNN proved to perform the best, specifically using the covariance matrix as its input. The CNN with the best performance still performed a bit worse than MUSIC at high SNRs but still performed relatively well, a RMSE of less than 2 degrees.

The results of the best network are shown against the MUSIC algorithm in figure 3.

The results of all the networks tested are shown in figure 2.

4. Discussion

When using the covariance matrix as the input to the network the CNN+LSTM architecture proved pretty poor in performance overall. This is probably due to the fact that the covariance matrix does not represent the a sequence in time of the array but the noise and source signal sub-spaces.

The best CNN that we tested still showed when estimating one angle at a high enough power level that it performed a bit worse than the MUSIC algorithm. This may be due to the fact that the network was trained on data ranging the while spectrum of powers tested and may have become less generalized in the process. However, as mentioned before at these high power levels the CNN showed a good AoA RMSE at less than 2 degrees.

Figure 3 shows that the CNN dips below the performance of the MUSIC algorithm around -15 dB. Similar to at high powers this may be due to over-training the model and lower powers thus making performance at higher powers less accurate. Once the SNR reaches above -15dBm the CNN has a RMSE of less than 15 degrees.

As previously mentioned the DICHASUS dataset did not have great performance in terms of having a low AoA error when using the MUSIC algorithm. This is due to the environment in which the data was collected. The data was collected in an indoor environment which leads itself to a good deal of mutipath. Since the CNN was able to be trained on the noisy data it learned to more accurately predict the AoA while the MUSIC algorithm struggled to predict due to the number of mutipath signals. This is shown in the CDF plot in figure 4.

5. Future Work

As part of future work the model should have it's output layer expanded such that it can estimate multiple signals of interest [1]. Doing so the model may perform better than that of MUSIC when signals are close together in azimuth thus further justifying the use of CNNs to do AoA estimation.

Another future work option will include looking into training a neural network not only to perform AoA estimation but also to do signal classification. In certain systems, specifically passive sensor systems like a direction finder, properly classifying and producing an AoA estimation are equally hard tasks. In many industries specialized detectors are used to classify a small subset of signals of interest and typically take a long time to design. If a neural network could be designed to be easily trainable to classify new signals of interest it would be beneficial for more sensor systems to employ them.

References

[1] Aysha Alteneiji, Ubaid Ahmad, Kin Poon, Nazar Ali, and Nawaf Almoosa. Angle of arrival estimation in indoor environment using machine learning. In *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–6, 2021. 1, 3

[2] Florian Euchner, Marc Gauger, Sebastian Dörner, and Stephan ten Brink. A Distributed Massive MIMO Channel Sounder for "Big CSI Data"-driven Machine Learning. In *WSA 2021; 25th International ITG Workshop on Smart Antennas*, 2021. 2

[3] Chan-Bin Ko and Joon-Ho Lee. Performance of esprit and root-music for angle-of-arrival(aoa) estimation. In *2018 IEEE World Symposium on Communication Engineering (WSCE)*, pages 49–53, 2018. 2

[4] Mianzhi Wang, Zhen Zhang, and Arye Nehorai. Performance analysis of coarray-based music in the presence of sensor location errors. *IEEE Transactions on Signal Processing*, 66(12):3074–3085, 2018. 2

Table 1. CNN Model Architecture

Layer Type	Output Dimensions	Extra Info	Total Params
2d Convolutional	(None, 30, 30, 32)	Filters = 32 (3x3), af = relu	608
Batch Norm	(None, 30, 30 ,32)	axis=1	120
Dropout	(None, 30, 30, 32)	rate=0.35	
2d Convolutional	(None, 15, 15, 32)	Filters = 32 (3x3), af = relu	9248
2d Max Pooling	(None, 15, 15, 32)	Filters =2x2	0
Batch Norm	(None, 15, 15, 32)	axis=1	60
Dropout	(None, 15, 15, 32)	rate=0.35	0
2d Convolutional	(None, 8, 8, 32)	Filters = 32 (3x3)	9248
2d Max Pooling	(None, 8, 8, 32)	Filters =2x2	0
Batch Norm	(None, 8, 8, 32)	axis=1	60
Dropout	(None, 8, 8, 32)	rate=0.35	0
Flatten	(None, 2048)		0
FC Layer	(None, 64)	af = relu	131136
Batch Norm	(None, 64)	axis=1	256
FC Layer	(None, 32)	af = relu	131136
Batch Norm	(None, 32)	axis=1	256
FC Layer	(None, 1)	af = linear	33
Optimizer		Adam, MSE	

Table 2. CNN+LSTM Model Architecture

Layer Type	**Output Dimensions**	**Extra Info**	**Total Params**
2d Convolutional	(None, 30, 30, 32)	Filters = 32 (3x3), af = relu	608
Batch Norm	(None, 30, 30, 32)	axis=1	120
Dropout	(None, 30, 30, 32)	rate=0.35	
2d Convolutional	(None, 15, 15, 64)	Filters = 32 (3x3), af = relu	18496
2d Max Pooling	(None, 15, 15, 64)	Filters =2x2	0
Batch Norm	(None, 15, 15, 64)	axis=1	60
Dropout	(None, 15, 15, 64)	rate=0.35	0
2d Convolutional	(None, 8, 8, 64)	Filters = 32 (3x3)	36928
2d Max Pooling	(None, 8, 8, 64)	Filters =2x2	0
Batch Norm	(None, 8, 8, 64)	axis=1	32
Dropout	(None, 8, 8, 64)	rate=0.35	0
Flatten	(None, 4096)		0
FC Layer	(None, 64)	af = relu	262208
Reshape	(None, 64,1)		
LSTM	(None, 8)		320
Batch Norm	(None, 64)	axis=1	32
FC Layer	(None, 32)	af = relu	288
Batch Norm	(None, 32)	axis=1	128
FC Layer	(None, 1)	af = linear	33
Optimizer		Adam, MSE	

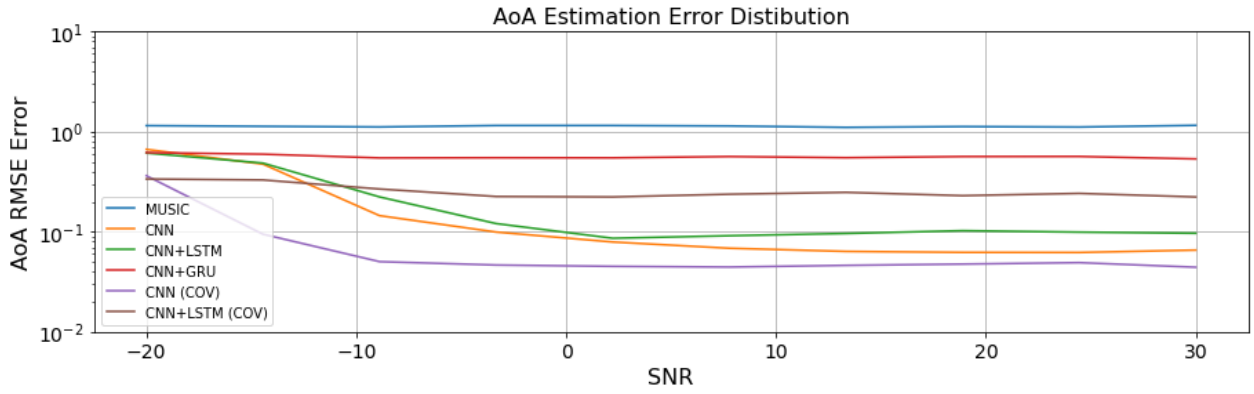

Figure 2. Model Types Performance (RMSE in Radians)

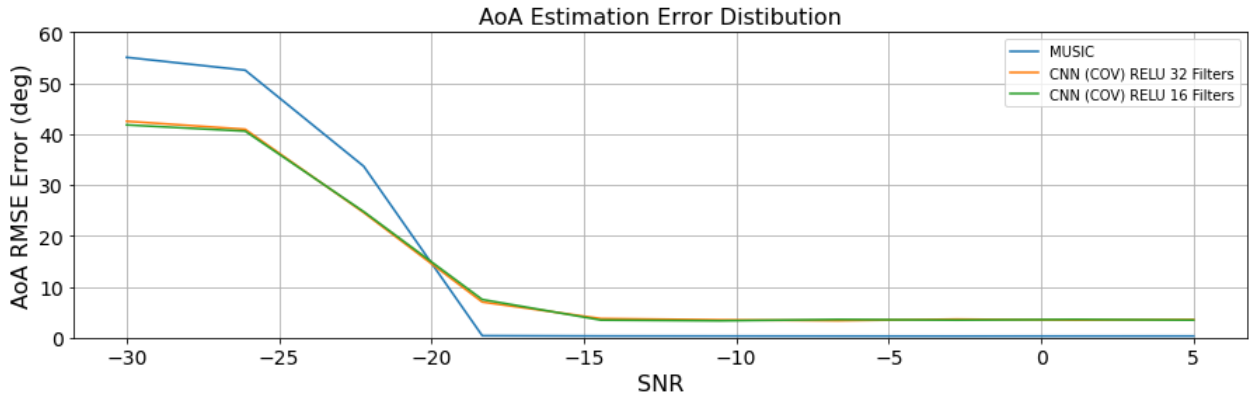

Figure 3. Best CNNs against MUSIC

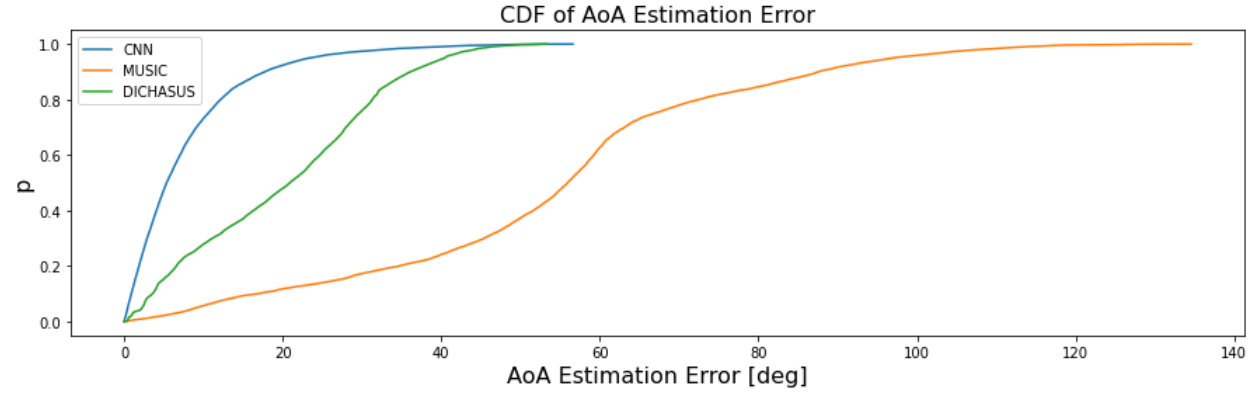

Figure 4. DICHASUS Dataset Architecture Comparison for AoA

Table 3. CNN+GRU+RNN Model Architecture

Layer Type	**Output Dimensions**	**Extra Info**	**Total Params**
2d Convolutional	(None, 30, 30, 32)	Filters = 32 (3x3), af = relu	608
Batch Norm	(None, 30, 30, 32)	axis=1	120
Dropout	(None, 30, 30, 32)	rate=0.35	
2d Convolutional	(None, 15, 15, 64)	Filters = 32 (3x3), af = relu	18496
2d Max Pooling	(None, 15, 15, 64)	Filters =2x2	0
Batch Norm	(None, 15, 15, 64)	axis=1	60
Dropout	(None, 15, 15, 64)	rate=0.35	0
2d Convolutional	(None, 8, 8, 64)	Filters = 32 (3x3)	36928
2d Max Pooling	(None, 8, 8, 64)	Filters =2x2	0
Batch Norm	(None, 8, 8, 64)	axis=1	32
Dropout	(None, 8, 8, 64)	rate=0.35	0
Flatten	(None, 4096)		0
FC Layer	(None, 64)	af = relu	262208
Reshape	(None, 64,1)		
GRU	(None, 16)		912
Reshape	(None, 16,1)		
Simple RNN	(None, 128)		16640
Batch Norm	(None, 64)	axis=1	512
FC Layer	(None, 32)	af = relu	4128
Batch Norm	(None, 32)	axis=1	128
FC Layer	(None, 1)	af = linear	33
Optimizer		Adam, MSE	