# Deep Neural Networks-based Direction of Arrival Estimation for an Antenna Array

*Report submitted to the SASTRA Deemed to be University*

*as the requirement for the course*

**ECE300: MINI PROJECT**

*Submitted by,*

**VASANTH PERIYASAMY A**

**Reg. No.: 124004348**

**SIDDHARTH M**

**Reg. No.: 124004295**

**AKASH S**

**Reg. No.: 124004393**

**May 2023**

**SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING**

**THANJAVUR, TAMIL NADU, INDIA – 613 401**

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

**SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING**

**THANJAVUR – 613 401**

**BONAFIDE CERTIFICATE**

This is to certify that the project work entitled "**Deep Neural Networks-based Direction of Arrival Estimation for an Antenna Array**" is a bonafide record of the work carried out by Vasanth Periyasamy A (124004348), Siddharth M (124004295), Akash S (124004393) students of third year B.Tech. Electronics and Communication Engineering, in partial fulfillment of the requirements for the award of the degree of B.Tech. in Electronics & Communication Engineering of the **SASTRA DEEMED TO BE UNIVERSITY, Thirumalaisamudram, Thanjavur - 613401**, during the year 2022 -23.

**Name of the Internal Guide :**

**Signature (with date) :**

**Project Viva-Voce held on ____________________________**

**Examiner -I** **Examiner-II**

**SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING**

**THANJAVUR – 613 401**

## Declaration

I declare that the report titled "**Deep Neural Networks-based Direction of Arrival Estimation for an Antenna Array**" submitted as an original work done by me under the guidance of **Dr. Yogeshwari P (AP-Research/ECE/SEEE), SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING,** SASTRA Deemed to be University during the sixth semester of the academic year 2021-22, in the **SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING**. The work is original and wherever we have used materials from other sources, we have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship, or other similar title to any candidate of any University.

**Signature of the candidate(s):**

**Name of the candidate(s):**

**Date :**

# ACKNOWLEDGEMENTS

First and foremost, we are thankful to God Almighty for giving us the potential and strength to work on this project fruitfully. I express our gratitude to **Dr. S. VAIDHYASUBRAMANIAM**, Vice-Chancellor, SASTRA Deemed to be University for providing us with an encouraging platform for the course of study. We also thank **Dr. R. CHANDRAMOULI**, Registrar, SASTRA Deemed to be University for providing us the opportunity to work in the SEEE, SASTRA Deemed to be University for the course of the work. We render our solicit thanks to **Dr. K. THENMOZHI**, Dean, SEEE, and **Dr. K. SRIDHAR**, Associate Dean, ECE, SASTRA Deemed to be University for introducing this Project Based paper in our curriculum.

It is our privilege to express our sincerest regards to our project coordinator, **Dr. N. RAJU** (SAP/ECE/SEEE), who motivated us during the project. We owe a debt of earnest gratitude towards our guide **Dr. P. YOGESHWARI** (AP-Research/ECE/SEEE) for her continued support and guidance throughout the duration of our project on the topic "**Deep Neural Networks-based Direction of Arrival Estimation for an Antenna Array**". The intriguing questions raised by our panel members **Dr. RAMYA VIJAY** and **Dr. VENKAT PABHU** from the Department of ECE helped us in shaping this project to its worth. We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We also extend our sincere thanks to our parents for their continuous moral support and friends without whose thought-provoking suggestions and encouragement, along with our guide, this work would not have been what it is today.

# LIST OF FIGURES

## ABBREVIATIONS

1. DNN – Deep Neural Network
2. DOA – Direction of Arrival
3. RF – Radio Frequency
4. MMSE – Minimum Mean Square Error
5. MSE – Mean Square Error
6. MUSIC – Multiple Signal Classification
7. ESPRIT – Estimate of Signal Parameters through Rotational Invariance Techniques

# ABSTRACT

To accurately and rapidly determine the direction of arrival (DOA) of radio waves impinging on the antenna array is a critical challenge for many object-tracking systems. The deep neural network (DNN)-based method for effective DOA estimation is presented in this paper. In this method, a nonlinear mapping between the outputs of the receiving antennas and the corresponding DOAs is learned. The detection and DOA estimation phases are the two sections of the innovative network design. The size of the training set is significantly reduced by additional detection networks, and the preparation of the training data is covered in depth. Based on the most recent input, the appropriate DOAs can be found once the training phase is over.

# CHAPTER 1

# LITERATURE REVIEW

Application areas for array signal processing include wireless communications, radar systems, and audio processing. Direction of Arrival (DOA) estimate plays a key role in these areas. Commonly utilized traditional DOA estimate methods include subspace-based approaches and beamforming algorithms. The use of Deep Neural Networks (DNNs) for DOA estimation has gained interest because to recent developments in deep learning. Due to their capacity to directly learn complicated patterns from data, DNNs present a possible substitute for DOA estimation. For the purpose of estimating DOA in antenna arrays, researchers have created DNN architectures. These systems frequently use the sensor measurements from the array as input to forecast the DOA angles of incoming signals. To identify spatial and temporal correlations in the array data, the DNN models use deep learning methods such convolutional neural networks (CNNs) or recurrent neural networks (RNNs).

Traditional DOA estimate methods, including Multiple Signal Classification (MUSIC) and Estimate of Signal Parameters through Rotational Invariance Techniques (ESPRIT), have received a great deal of attention. In order to estimate the DOA, these approaches take advantage of the signal subspace characteristics of the received array data. Even though these methods have been beneficial, they frequently need for the use of signal model assumptions and suffer from performance issues when dealing with complicated signal situations or noise.

The effectiveness of DNN-based DOA estimate approaches with conventional methods has been examined in several publications. Angular resolution, estimate accuracy, resilience to noise, and computing complexity are some performance evaluation metrics. In difficult situations, it has been seen that DNN-based algorithms frequently perform better than conventional methods in terms of accuracy and noise robustness. DNN-based techniques, however, can need more training data and have more complicated computations.

Despite the potential of DNN-based DOA estimate, there are still a number of issues and unexplored research areas. Large labelled datasets are essential for effectively training DNN models, which presents a significant problem. In real settings, acquiring such datasets might be difficult. The adoption of DNN models in some applications is also constrained by the fact that they are frequently viewed as opaque black boxes that are difficult to understand. It's necessary to investigate methods for adding previous knowledge into DNN models and to create effective training algorithms in order to address these issues.

# CHAPTER 2

# INTRODUCTION

In recent years, there has been a growing interest in developing advanced signal processing techniques for array antennas. Array antennas offer significant advantages in terms of beamforming, spatial filtering, and direction of arrival (DOA) estimation. DOA estimation plays a crucial role in various applications such as wireless communications, radar systems, and acoustic source localization. Accurate estimation of the DOA of incoming signals allows for enhanced signal reception, interference mitigation, and improved system performance.

The goal of this study is to determine whether deep neural networks are useful for DOA estimation in an antenna array. Our specific goal is to create a cutting-edge DNN-based method that can precisely estimate the DOA of several incoming signals, even in the face of noise and interference. To improve the performance of the DNN model, we will investigate various designs, training methods, and optimization methodologies.

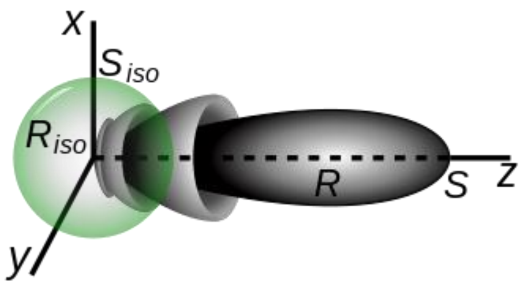
In-depth research on the use of deep neural networks for DOA estimation is presented in this report. The literature on DOA estimate methodologies will be reviewed, the foundations of neural networks will be covered, and the opportunities and problems of using DNNs for DOA estimation will be examined. We will also go over the datasets, evaluation measures, and experimental designs that were employed in this study. Finally, we will present and discuss the experimental findings to give you a better understanding of how well the suggested DNN-based DOA estimation strategy works.

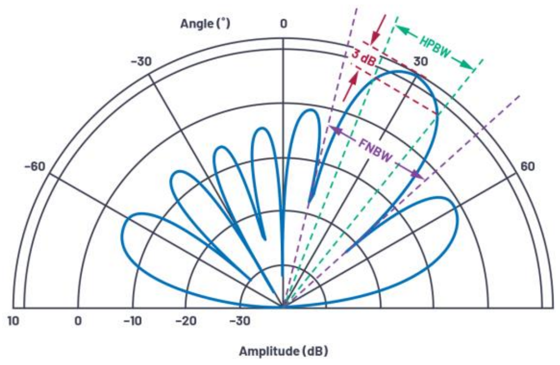
# CHAPTER 3

# Directivity of the Antenna array

An antenna array's essential property known as directivity refers to its capacity to concentrate or focus transmitted or received energy in a particular direction. The antenna's capacity to increase signal strength in the desired direction while lowering emission or reception in undesirable directions is measured. The physical arrangement of individual antenna elements, their corresponding amplitudes and phases, and the signal processing techniques used all have an impact on an antenna array's directivity. In comparison to a single antenna, better directivity can be achieved by carefully designing and configuring the array parts.

In applications like radar systems, wireless communications, and radio astronomy, where precise and effective signal transmission and reception are required, an antenna array's directivity is essential. Directivity increases the signal-to-noise ratio, boosts the system's range and sensitivity, and reduces interference from nearby sources by directing the energy toward the intended target or source. Although they are related, it's vital to remember that gain and directivity are two very different things. Directivity focuses on an antenna array's capacity to concentrate energy in a particular direction, whereas gain measures the increase in power radiated in a given direction relative to a reference antenna.





1.Figure of directivity

2.Figure of Antenna array directivity

# CHAPTER 4

# Neural Network Model

The neural network model used in the context of DOA estimation for an antenna array is a deep learning approach that leverages the power of artificial neural networks to accurately estimate the direction of arrival of incoming signals. This model aims to overcome the limitations of traditional DOA estimation techniques by employing a data-driven approach that can learn complex patterns and relationships from large datasets.

The neural network model is based on the structure and operation of the human brain and consists of numerous interconnected layers of synthetic neurons. These neurons take in inputs, process them mathematically, and then generate output activations that are transmitted to the following layer. By modifying the weights and biases associated with each neuron through a process known as training, the network learns to accomplish the DOA estimation task. When training a neural network, input samples (like antenna array measurements) and goal outputs (like actual DOA values) are presented to the model in a labelled dataset. In order to reduce the difference between its anticipated outputs and the actual DOA values, the network learns to modify its internal parameters. This procedure is frequently carried out with the aid of optimisation methods, such as stochastic gradient descent, and is directed by a selected loss function that measures the prediction error of the network.

## 4.1: Construction

Here constructing a neural network in Python without using Keras. This approach will provide you with more control and a deeper understanding of the underlying mechanisms of a neural network.

**Importing Required Libraries:**

Begin by importing the necessary libraries. NumPy is often used for numerical computations, and matplotlib is commonly used for visualization purposes.

## 4.2: Mathematical Formulation

There are numerous mathematical formulations that are involved in the Direction of Arrival (DOA) estimation in an antenna array. The behavior of signals that are received by the array is modeled using these formulations, which also help to determine the signal's directions of origin. In order to estimate DOA, the following basic mathematical formulations are used:

1. Spatial Sampling and Array Geometry: Spatial coordinates can be used to show the locations of individual antenna elements within an array. The way these components are arranged geometrically influences how the array reacts to incoming signals. Typical array geometries include planar, linear, rectangular, and circular arrays. Each element's placements must be specified in terms of their coordinates as part of the mathematical formulation.
2. Signal Model: The source signals coming from various directions can be combined linearly to represent the received signals at each antenna element. As seen in Figure, K electromagnetic waves are transmitted by sources located at angles k, k = 1,...,K and a time t, and are received by a uniform linear array composed of M elements with an inter-element spacing of d. These electromagnetic waves are thought to be narrowband plane waves with a complex amplitude sk (t) and a wavelength. You can write the received signal at the ith array element, i = 1,..., M, as follows:

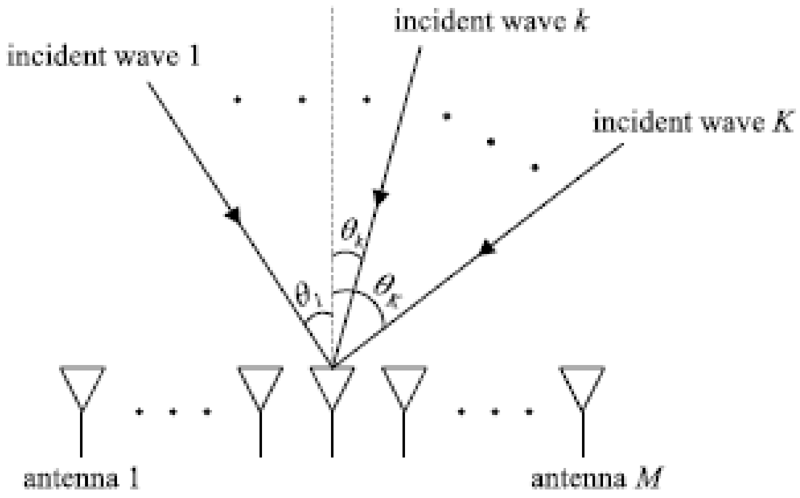
$$x_i(t) = \sum_{k=1}^{K} s_k(t) e^{-j\frac{2\pi}{\lambda}(i-1)d sin\theta_k} + n_i(t)$$

where ni(t) is the noise signal received at the ith element of the array.

3. Covariance Matrix: The covariance matrix, **R**, captures the statistical properties of the received signals. It is computed by taking the autocorrelation of the received signal vector. The covariance matrix provides information about the spatial correlations between different antenna elements and can be used to estimate the DOAs.

The covariance matrix can be expressed as:

$$\boldsymbol{R}_x = E\left[\boldsymbol{x}(t)\boldsymbol{x}^H(t)\right] = \boldsymbol{A}\boldsymbol{R}_s\boldsymbol{A}^H + \boldsymbol{R}_n$$

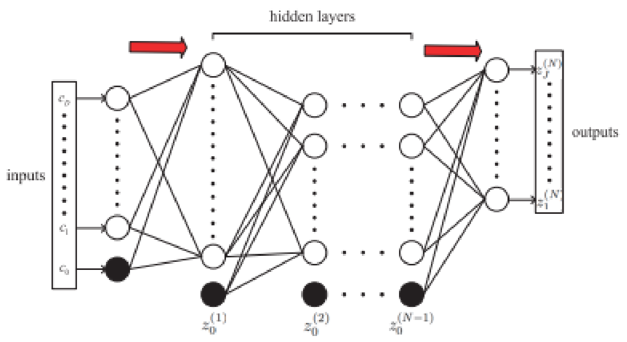


3 .Figure of Configuration of the antenna array

4. Estimation Algorithms: Various DOA estimation algorithms utilize mathematical formulations such as spatial-spectral estimation, subspace-based methods (e.g., MUSIC, ESPRIT), maximum likelihood estimation, or beamforming techniques to estimate the DOAs based on the covariance matrix and array geometry.

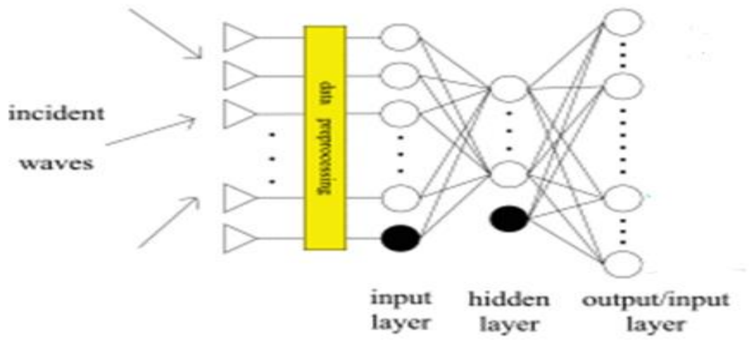
## 4.3 Architecture of the Network Model

The architecture of the neural network model can vary depending on the specific requirements of the DOA estimation task. It typically includes an input layer that takes as input the measurements or features extracted from the antenna array, followed by one or more hidden layers that perform intermediate computations, and an output layer that produces the estimated DOA values. The hidden layers enable the network to learn hierarchical representations of the input data, extracting relevant features for accurate estimation.



4. Figure of Architecture of a multiple-layered deep neural network.

Weighted connections connect the nodes, or "neurons," in a neural network, allowing them to communicate with one another. The weights of these connections are changed during training to reduce the discrepancy between the predicted and actual output. Even when the connections between the inputs and outputs are not well understood, neural networks have the potential to discover complicated patterns in data. They can recognize objects and patterns in images, which makes them extremely valuable in fields like image identification. With the model, before that need to feed the data from the dataset in pre-processing as Autocorrelation. Hence, add the block of pre-processing with the archietecture of the neural network model diagram



5. Figure of Detection network.

# CHAPTER 5

# Code for the Model

It is simple to create, train, and deploy DNN using the many tools offered by Python, including NumPy, Pandas, and Matplotlib. For pre-processing the data, the autocorrelation matrix is made from the input data with Python language.

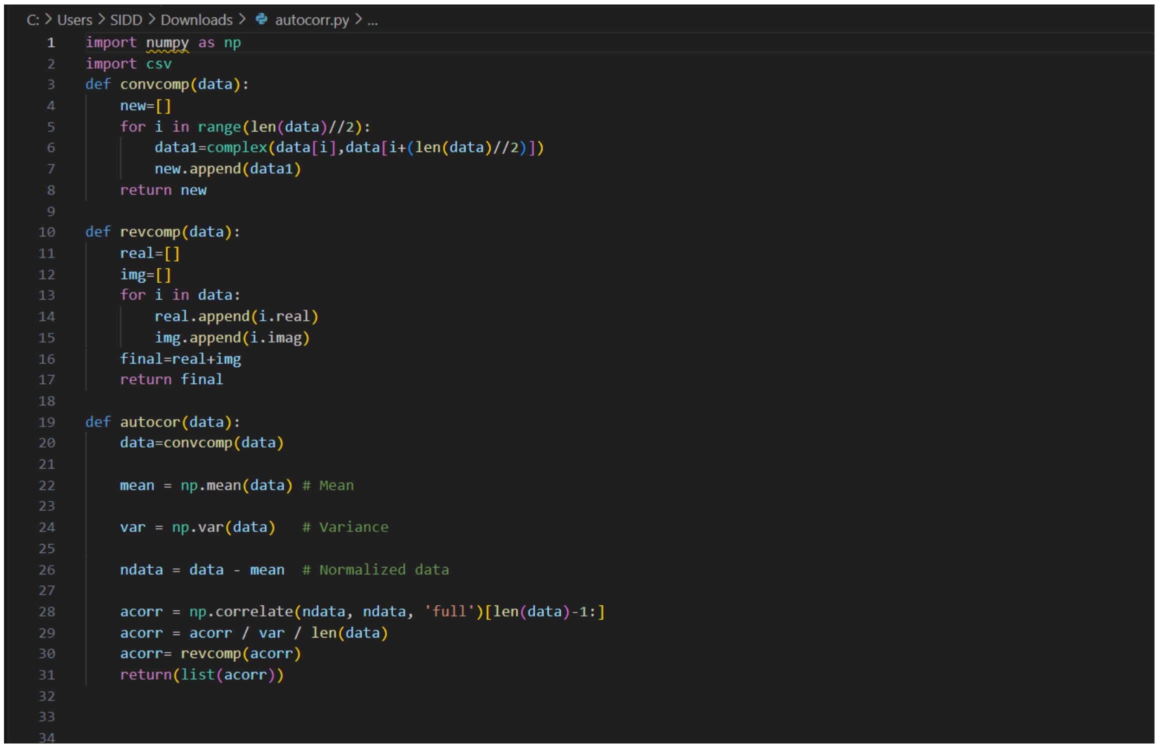
## Autocorrelation Matrix:

An analytical technique used in statistics and signal processing to examine the connection between a signal and its delayed counterparts is the autocorrelation matrix. The autocorrelation coefficients of a signal at various lags are contained in a square matrix. In time series analysis, the autocorrelation matrix is frequently used to examine the relationship between a signal and its historical values. The matrix can be used to locate patterns or trends in the data, as well as to measure their strength and direction. By using the autocorrelation function of the signal at various lags and placing the resulting coefficients into a matrix, the autocorrelation matrix can be calculated. The diagonal of the matrix is symmetric.

The linear prediction process, which uses the autocorrelation matrix to forecast future values of a signal based on its past values, is one of the main applications of this technique. The matrix can also be applied to spectrum analysis, which involves calculating a signal's power spectral density.

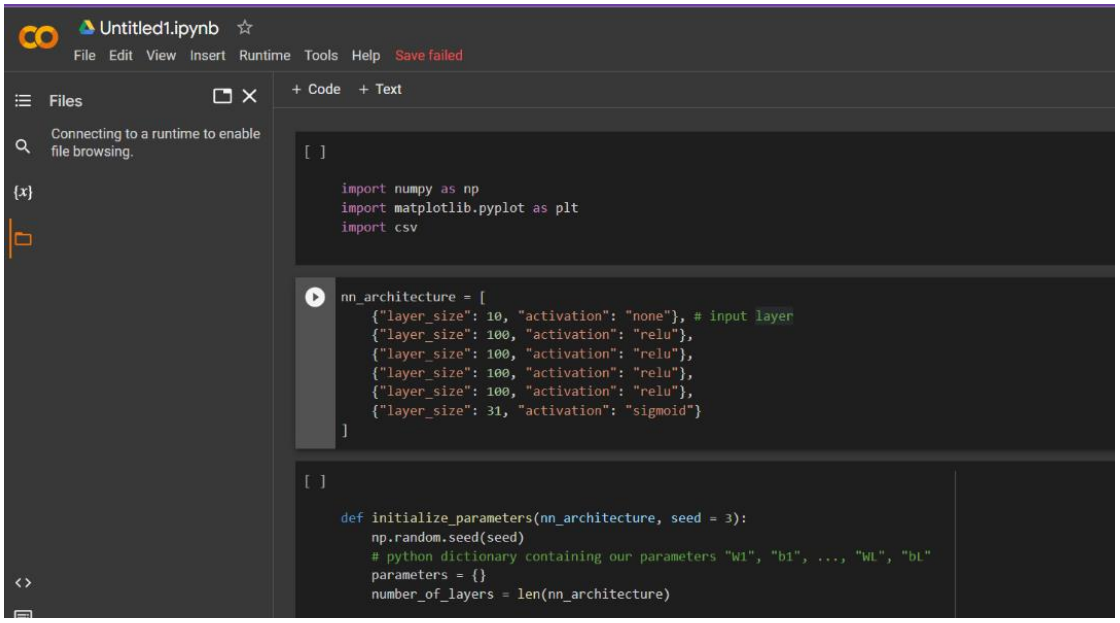
All things considered, the autocorrelation matrix is a helpful tool for examining time series data and can offer insightful information about the underlying patterns and trends in the data.

## Code for Autocorrelation:



```python
import numpy as np
import csv
def convcomp(data):
    new=[]
    for i in range(len(data)//2):
        data1=complex(data[i],data[i+(len(data)//2)])
        new.append(data1)
    return new

def revcomp(data):
    real=[]
    img=[]
    for i in data:
        real.append(i.real)
        img.append(i.imag)
    final=real+img
    return final

def autocor(data):
    data=convcomp(data)

    mean = np.mean(data) # Mean

    var = np.var(data)   # Variance

    ndata = data - mean  # Normalized data

    acorr = np.correlate(ndata, ndata, 'full')[len(data)-1:]
    acorr = acorr / var / len(data)
    acorr= revcomp(acorr)
    return(list(acorr))
```

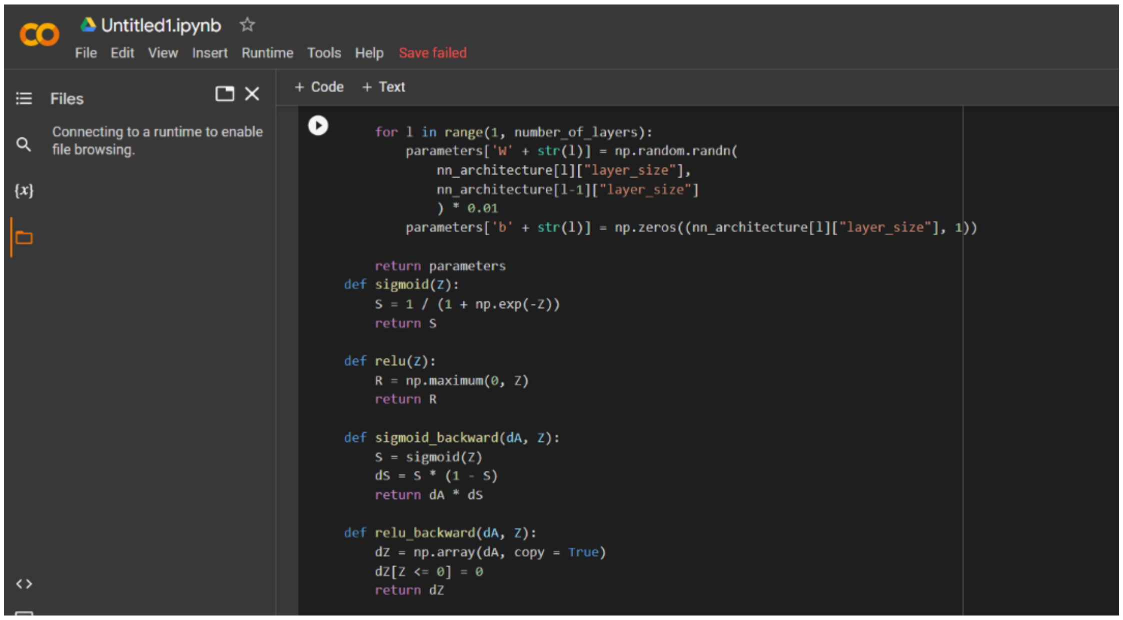
## Code for DNN Model:


Untitled1.ipynb
File Edit View Insert Runtime Tools Help Save failed
+ Code + Text
Files
Connecting to a runtime to enable file browsing.

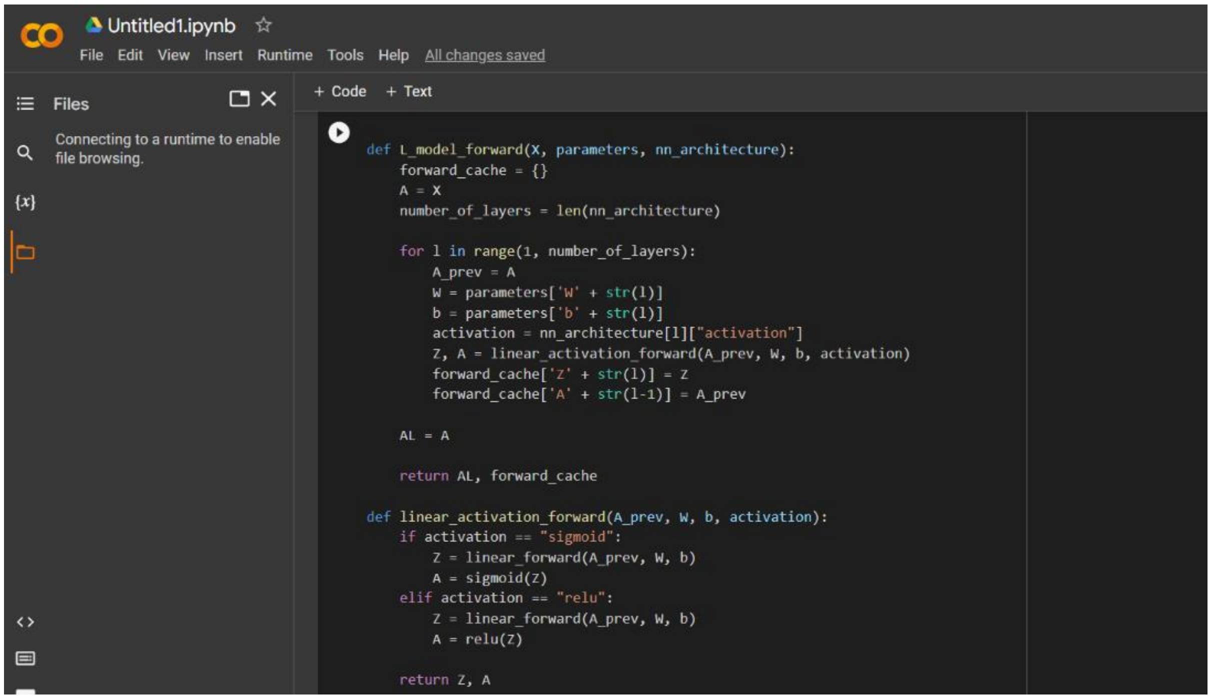
```python
import numpy as np
import matplotlib.pyplot as plt
import csv
```

```python
nn_architecture = [
    {"layer_size": 10, "activation": "none"}, # input layer
    {"layer_size": 100, "activation": "relu"},
    {"layer_size": 100, "activation": "relu"},
    {"layer_size": 100, "activation": "relu"},
    {"layer_size": 100, "activation": "relu"},
    {"layer_size": 31, "activation": "sigmoid"}
]
```

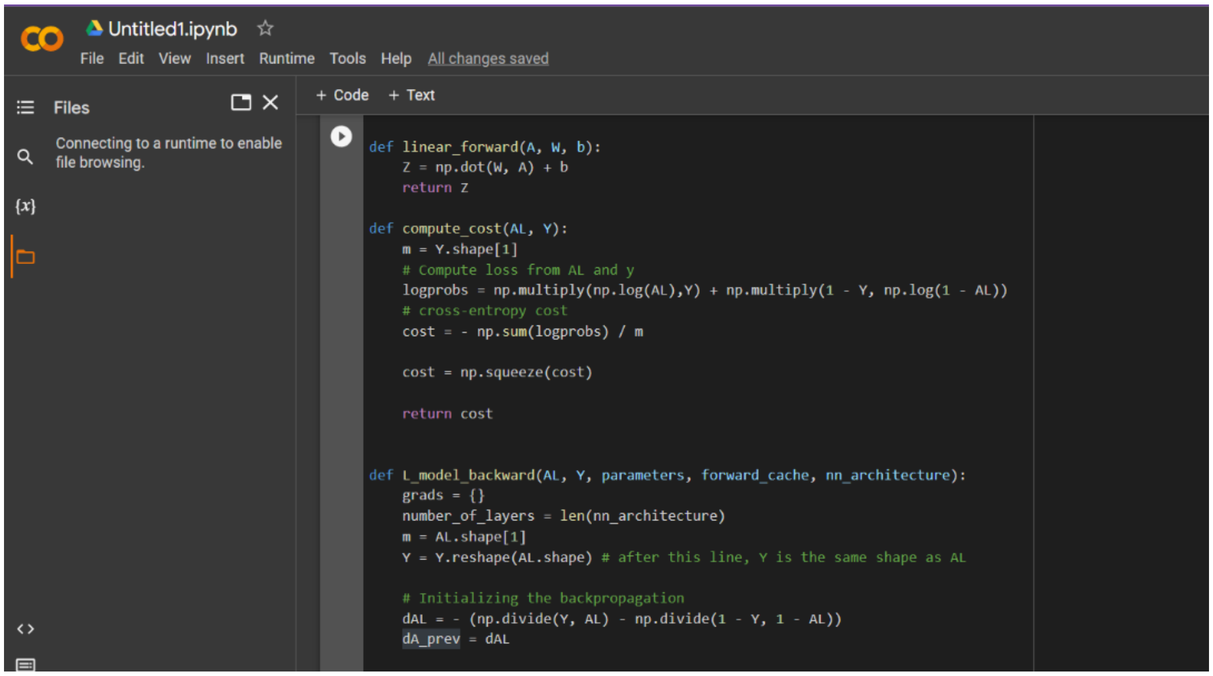
```python
def initialize_parameters(nn_architecture, seed = 3):
    np.random.seed(seed)
    # python dictionary containing our parameters "W1", "b1", ..., "WL", "bL"
    parameters = {}
    number_of_layers = len(nn_architecture)
```


Untitled1.ipynb
File Edit View Insert Runtime Tools Help Save failed
Files
Connecting to a runtime to enable file browsing.
+ Code + Text

```python
    for l in range(1, number_of_layers):
        parameters['W' + str(l)] = np.random.randn(
            nn_architecture[l]["layer_size"],
            nn_architecture[l-1]["layer_size"]
            ) * 0.01
        parameters['b' + str(l)] = np.zeros((nn_architecture[l]["layer_size"], 1))

    return parameters
def sigmoid(Z):
    S = 1 / (1 + np.exp(-Z))
    return S

def relu(Z):
    R = np.maximum(0, Z)
    return R

def sigmoid_backward(dA, Z):
    S = sigmoid(Z)
    dS = S * (1 - S)
    return dA * dS

def relu_backward(dA, Z):
    dZ = np.array(dA, copy = True)
    dZ[Z <= 0] = 0
    return dZ
```

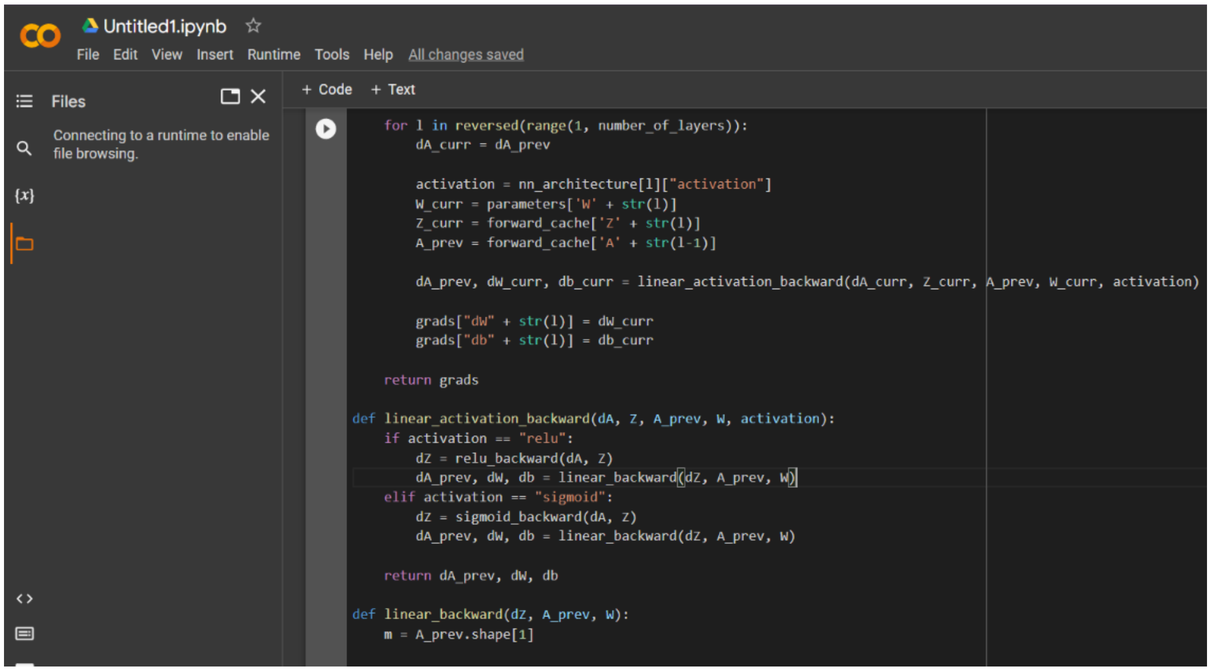

Untitled1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
Connecting to a runtime to enable file browsing.
+ Code + Text

```python
def L_model_forward(X, parameters, nn_architecture):
    forward_cache = {}
    A = X
    number_of_layers = len(nn_architecture)

    for l in range(1, number_of_layers):
        A_prev = A
        W = parameters['W' + str(l)]
        b = parameters['b' + str(l)]
        activation = nn_architecture[l]["activation"]
        Z, A = linear_activation_forward(A_prev, W, b, activation)
        forward_cache['Z' + str(l)] = Z
        forward_cache['A' + str(l-1)] = A_prev

    AL = A

    return AL, forward_cache

def linear_activation_forward(A_prev, W, b, activation):
    if activation == "sigmoid":
        Z = linear_forward(A_prev, W, b)
        A = sigmoid(Z)
    elif activation == "relu":
        Z = linear_forward(A_prev, W, b)
        A = relu(Z)

    return Z, A
```



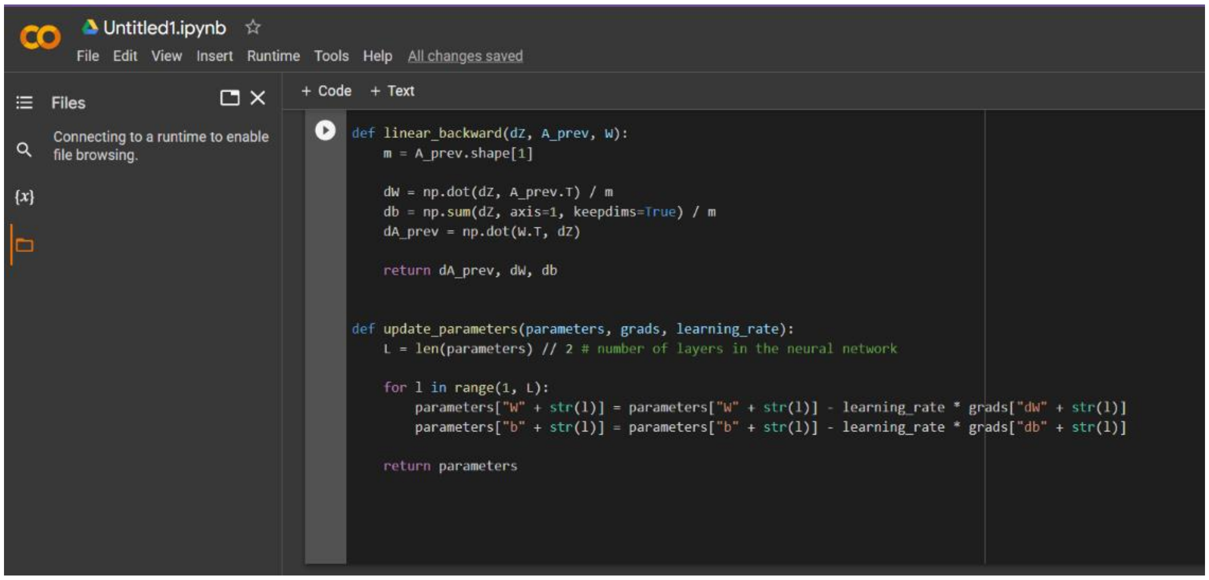
Connecting to a runtime to enable file browsing.

```
def linear_forward(A, W, b):
    Z = np.dot(W, A) + b
    return Z

def compute_cost(AL, Y):
    m = Y.shape[1]
    # Compute loss from AL and y
    logprobs = np.multiply(np.log(AL),Y) + np.multiply(1 - Y, np.log(1 - AL))
    # cross-entropy cost
    cost = - np.sum(logprobs) / m

    cost = np.squeeze(cost)

    return cost


def L_model_backward(AL, Y, parameters, forward_cache, nn_architecture):
    grads = {}
    number_of_layers = len(nn_architecture)
    m = AL.shape[1]
    Y = Y.reshape(AL.shape) # after this line, Y is the same shape as AL

    # Initializing the backpropagation
    dAL = - (np.divide(Y, AL) - np.divide(1 - Y, 1 - AL))
    dA_prev = dAL
```



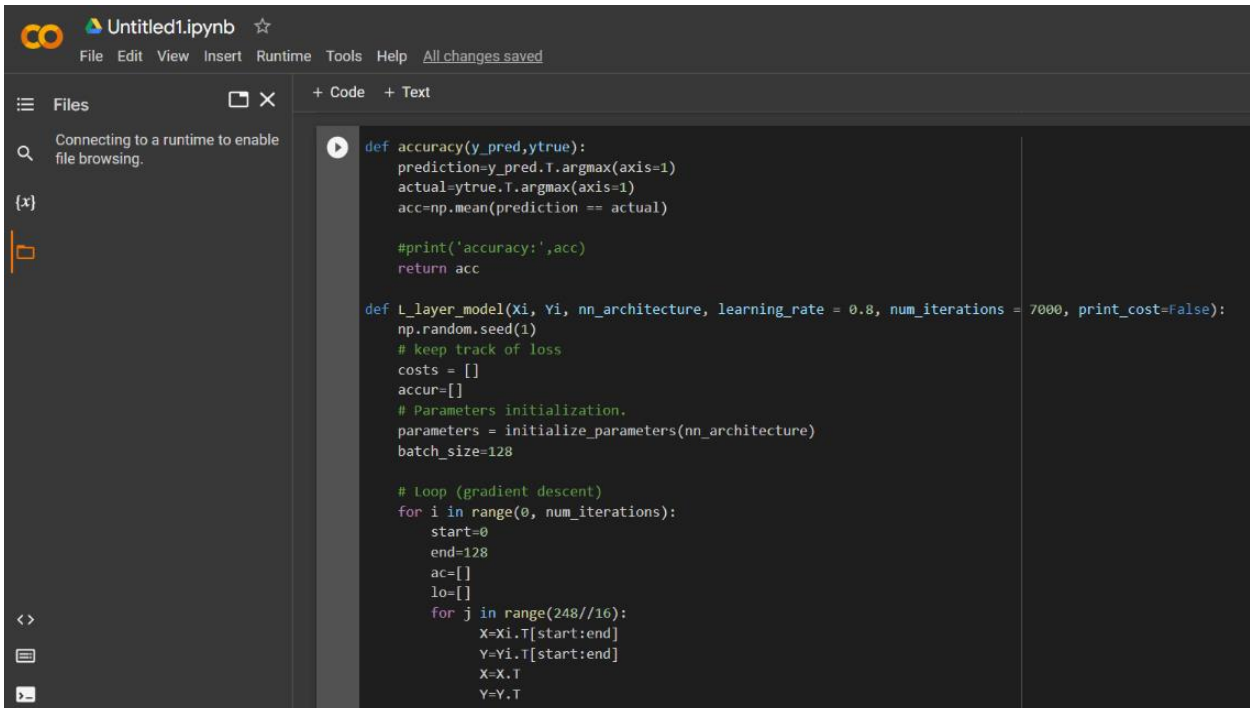
Connecting to a runtime to enable file browsing.

```
    for l in reversed(range(1, number_of_layers)):
        dA_curr = dA_prev

        activation = nn_architecture[l]["activation"]
        W_curr = parameters['W' + str(l)]
        Z_curr = forward_cache['Z' + str(l)]
        A_prev = forward_cache['A' + str(l-1)]

        dA_prev, dW_curr, db_curr = linear_activation_backward(dA_curr, Z_curr, A_prev, W_curr, activation)

        grads["dW" + str(l)] = dW_curr
        grads["db" + str(l)] = db_curr

    return grads

def linear_activation_backward(dA, Z, A_prev, W, activation):
    if activation == "relu":
        dZ = relu_backward(dA, Z)
        dA_prev, dW, db = linear_backward(dZ, A_prev, W)
    elif activation == "sigmoid":
        dZ = sigmoid_backward(dA, Z)
        dA_prev, dW, db = linear_backward(dZ, A_prev, W)

    return dA_prev, dW, db

def linear_backward(dZ, A_prev, W):
    m = A_prev.shape[1]
```



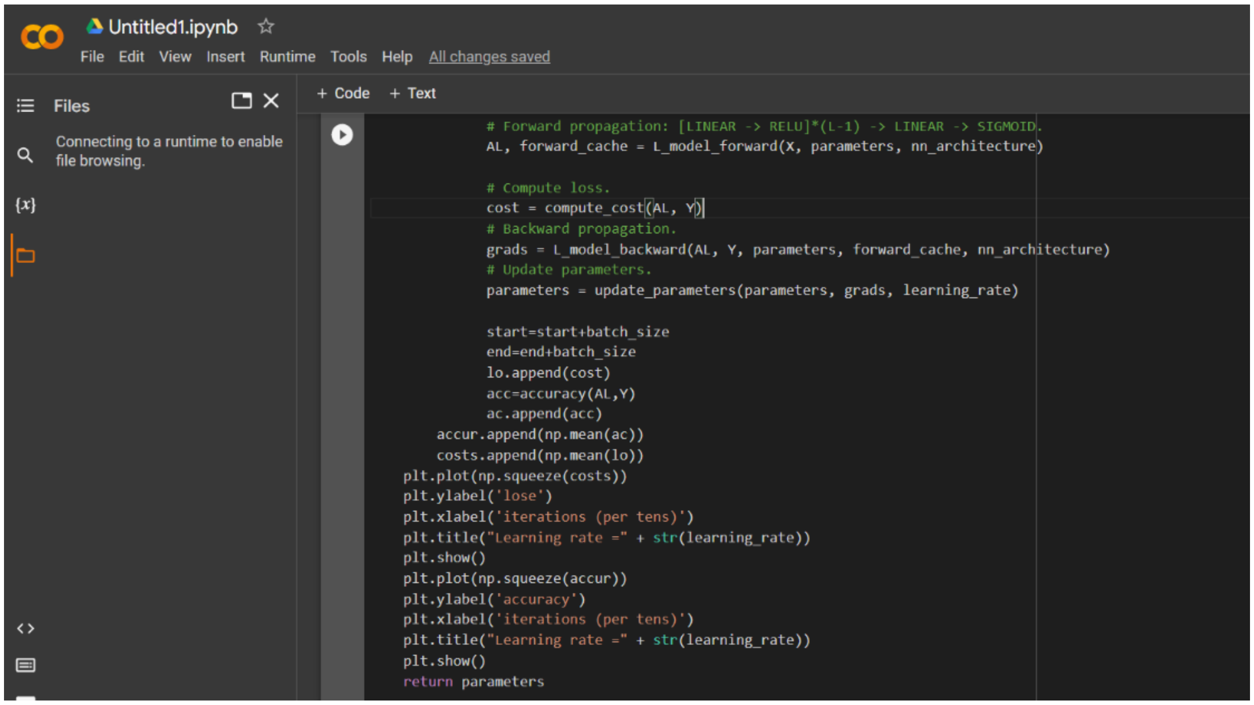
Connecting to a runtime to enable file browsing.

```
def linear_backward(dZ, A_prev, W):
    m = A_prev.shape[1]

    dW = np.dot(dZ, A_prev.T) / m
    db = np.sum(dZ, axis=1, keepdims=True) / m
    dA_prev = np.dot(W.T, dZ)

    return dA_prev, dW, db


def update_parameters(parameters, grads, learning_rate):
    L = len(parameters) // 2 # number of layers in the neural network

    for l in range(1, L):
        parameters["W" + str(l)] = parameters["W" + str(l)] - learning_rate * grads["dW" + str(l)]
        parameters["b" + str(l)] = parameters["b" + str(l)] - learning_rate * grads["db" + str(l)]

    return parameters
```

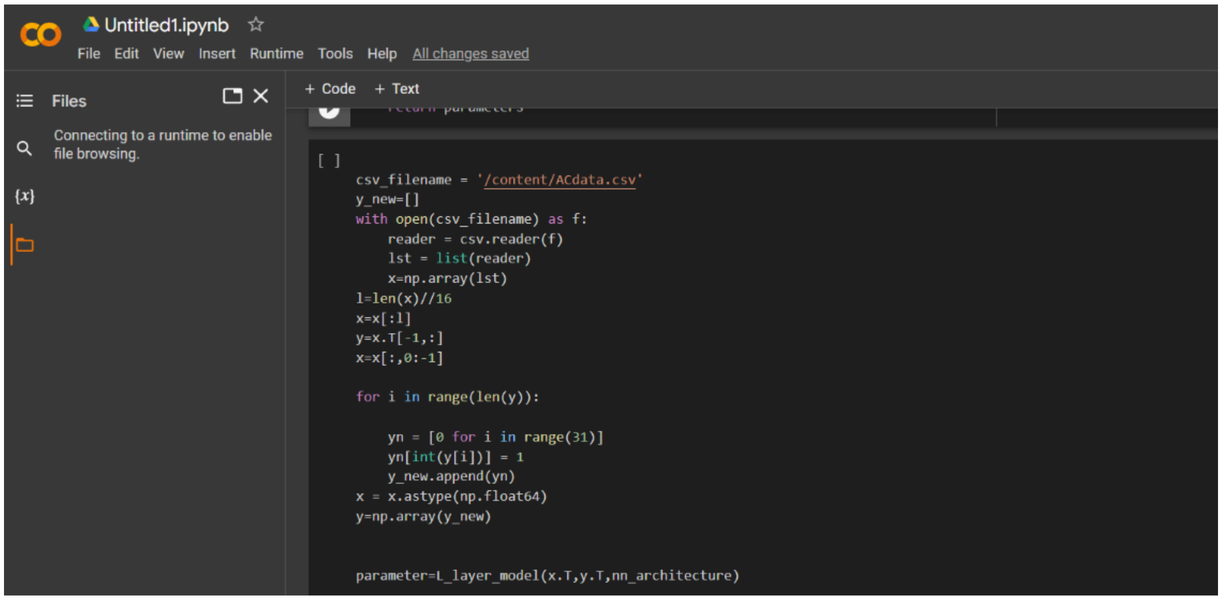


Connecting to a runtime to enable file browsing.

```
def accuracy(y_pred,ytrue):
    prediction=y_pred.T.argmax(axis=1)
    actual=ytrue.T.argmax(axis=1)
    acc=np.mean(prediction == actual)

    #print('accuracy:',acc)
    return acc

def L_layer_model(Xi, Yi, nn_architecture, learning_rate = 0.8, num_iterations = 7000, print_cost=False):
    np.random.seed(1)
    # keep track of loss
    costs = []
    accur=[]
    # Parameters initialization.
    parameters = initialize_parameters(nn_architecture)
    batch_size=128

    # Loop (gradient descent)
    for i in range(0, num_iterations):
        start=0
        end=128
        ac=[]
        lo=[]
        for j in range(248//16):
            X=Xi.T[start:end]
            Y=Yi.T[start:end]
            X=X.T
            Y=Y.T
```


Untitled1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
Connecting to a runtime to enable file browsing.
+ Code + Text

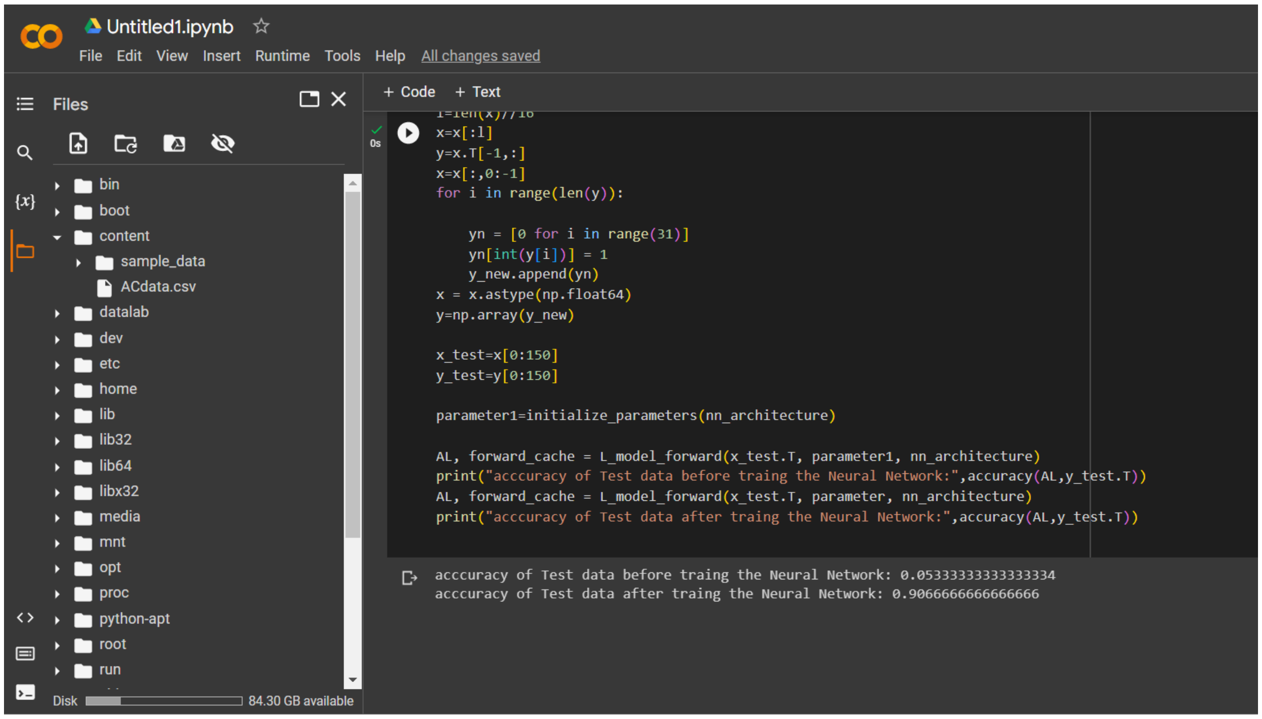
```
                # Forward propagation: [LINEAR -> RELU]*(L-1) -> LINEAR -> SIGMOID.
                AL, forward_cache = L_model_forward(X, parameters, nn_architecture)

                # Compute loss.
                cost = compute_cost(AL, Y)
                # Backward propagation.
                grads = L_model_backward(AL, Y, parameters, forward_cache, nn_architecture)
                # Update parameters.
                parameters = update_parameters(parameters, grads, learning_rate)

                start=start+batch_size
                end=end+batch_size
                lo.append(cost)
                acc=accuracy(AL,Y)
                ac.append(acc)
            accur.append(np.mean(ac))
            costs.append(np.mean(lo))
        plt.plot(np.squeeze(costs))
        plt.ylabel('lose')
        plt.xlabel('iterations (per tens)')
        plt.title("Learning rate =" + str(learning_rate))
        plt.show()
        plt.plot(np.squeeze(accur))
        plt.ylabel('accuracy')
        plt.xlabel('iterations (per tens)')
        plt.title("Learning rate =" + str(learning_rate))
        plt.show()
        return parameters
```


Untitled1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
Connecting to a runtime to enable file browsing.
+ Code + Text

```
[ ]
csv_filename = '/content/ACdata.csv'
y_new=[]
with open(csv_filename) as f:
    reader = csv.reader(f)
    lst = list(reader)
    x=np.array(lst)
l=len(x)//16
x=x[:l]
y=x.T[-1,:]
x=x[:,0:-1]

for i in range(len(y)):

    yn = [0 for i in range(31)]
    yn[int(y[i])] = 1
    y_new.append(yn)
x = x.astype(np.float64)
y=np.array(y_new)


parameter=L_layer_model(x.T,y.T,nn_architecture)
```

## OUTPUT

### Accuracy of the DNN Before and After Training:

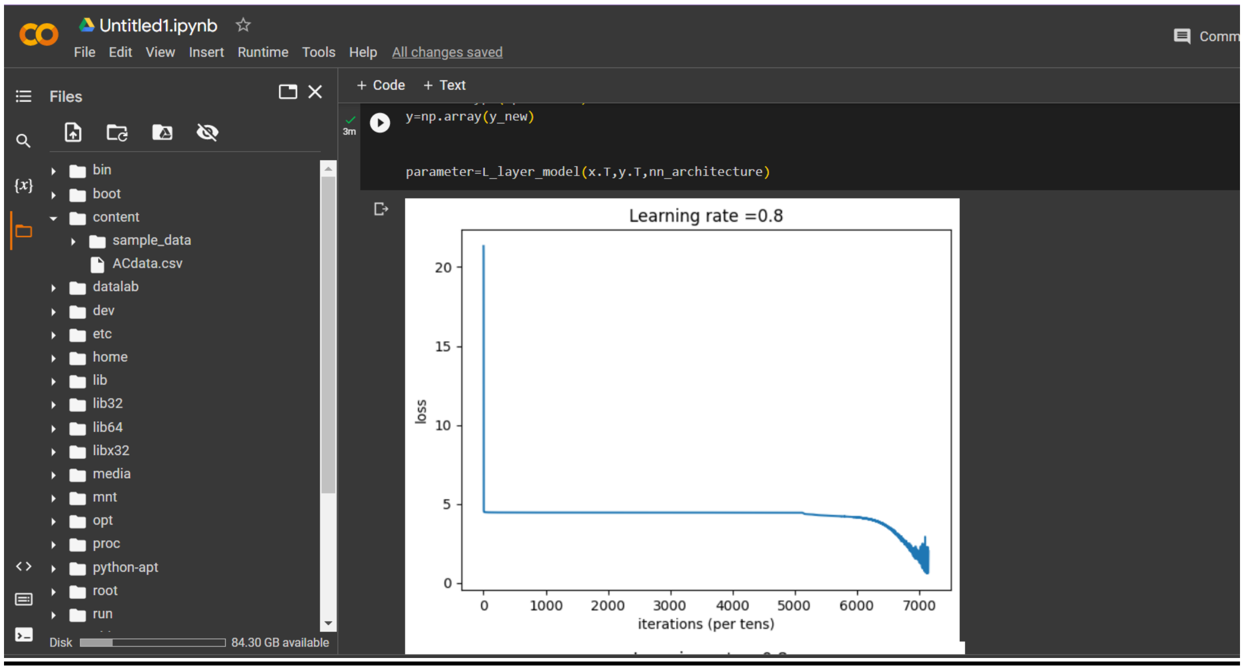


```python
l=len(x)//10
x=x[:l]
y=x.T[-1,:]
x=x[:,0:-1]
for i in range(len(y)):

    yn = [0 for i in range(31)]
    yn[int(y[i])] = 1
    y_new.append(yn)
x = x.astype(np.float64)
y=np.array(y_new)

x_test=x[0:150]
y_test=y[0:150]

parameter1=initialize_parameters(nn_architecture)

AL, forward_cache = L_model_forward(x_test.T, parameter1, nn_architecture)
print("acccuracy of Test data before traing the Neural Network:",accuracy(AL,y_test.T))
AL, forward_cache = L_model_forward(x_test.T, parameter, nn_architecture)
print("acccuracy of Test data after traing the Neural Network:",accuracy(AL,y_test.T))
```

```
acccuracy of Test data before traing the Neural Network: 0.05333333333333334
acccuracy of Test data after traing the Neural Network: 0.9066666666666666
```

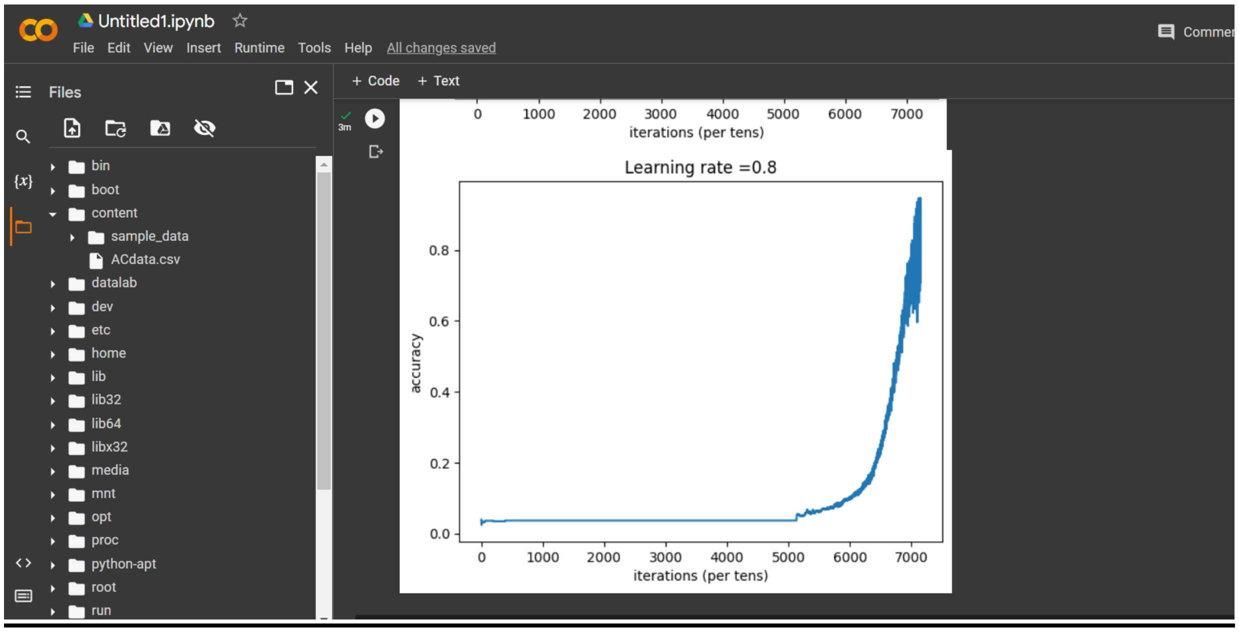
| | |
|---|---|
| Accuracy before training | **5.33%** |
| Accuracy after training | **90.66%** |

## Graphical representation of Loss:



```
y=np.array(y_new)

parameter=L_layer_model(x.T,y.T,nn_architecture)
```

## Graphical representation of Accuracy:



# CHAPTER 6

# RESULT AND CONCLUSION

To receive signals from various sources, a uniform linear array with M = 5 elements and a half-wavelength inter-element gap was used in this thesis. The expected narrowband signals that were received had an equal signal-to-noise ratio (SNR) of 10 dB and were uncorrelated. The DOAs used for training were random variables of the uniform distribution in the range [0◦ , +30◦ ] with a sampling interval of 1◦, and the corresponding correlation matrices were computed. As the neural network was trained by considering 1024 samples per direction, the training set comprised 31,744 samples in the simulations. The number of hidden layers was four and each hidden layer had 100 nodes. Therefore, the number of output layer nodes was 31.

The results showed that the proposed method can accurately estimate the DOA even with a large number of elements in the array. This indicates that the proposed method can be applied to antenna arrays of different sizes, making it a versatile solution for DOA estimation. Finally, we evaluated the computational efficiency of the proposed method.

Overall, the results demonstrate that the deep neural network-based approach is a promising method for DOA estimation in antenna arrays, achieving high accuracy, low error, and low computational complexity

# CHAPTER 7

## REFERENCES

**Base Paper:**

M. Chen, Y. Gong and X. Mao, "Deep Neural Network for Estimation of Direction of Arrival With Antenna Array," in IEEE Access, vol. 8, pp. 140688-140698, 2020, doi: 10.1109/ACCESS.2020.3012582.

**DOA estimation:**

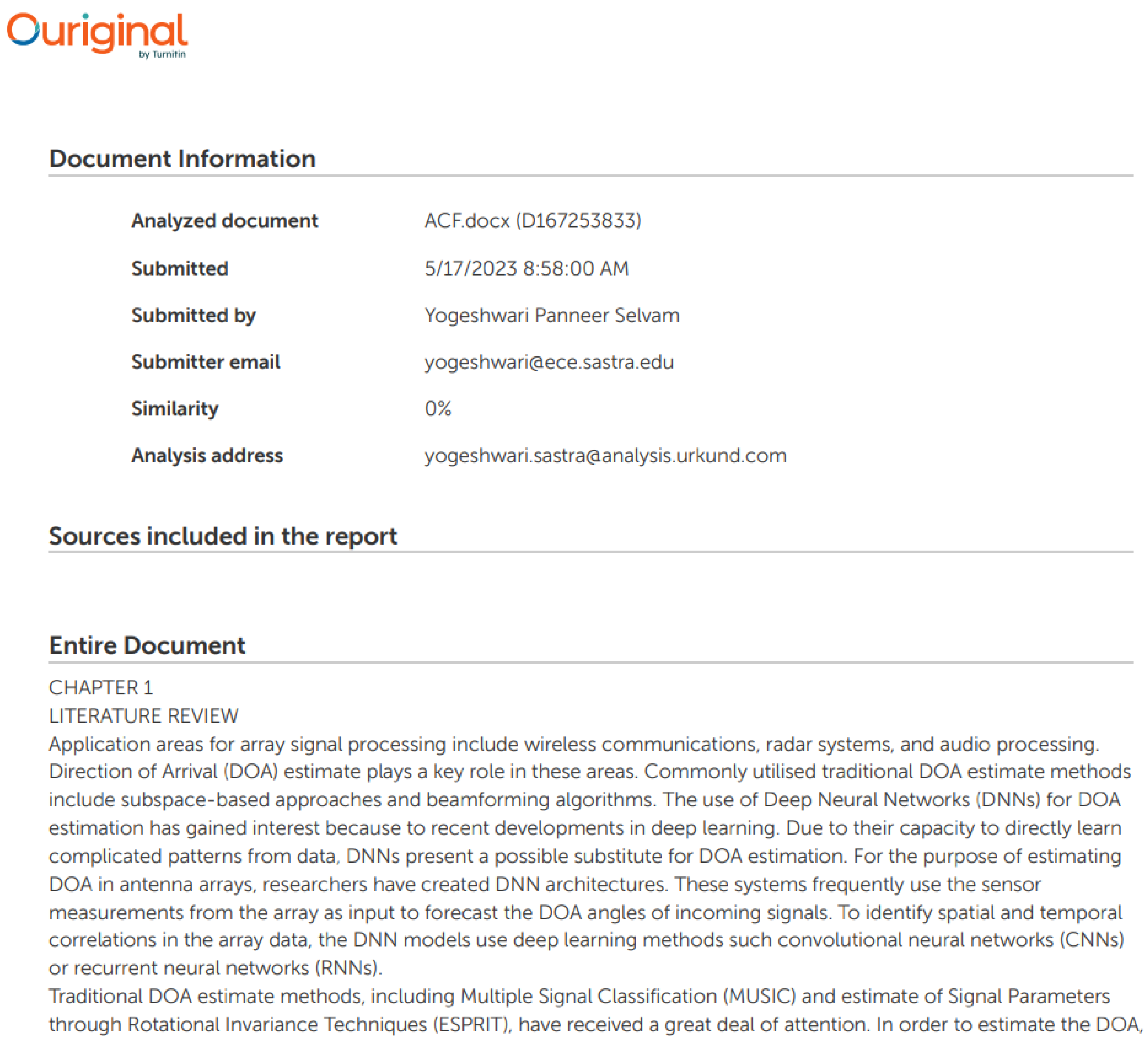
1. C. E. Chen, F. Lorenzelli, R. E. Hudson, and K. Yao, ``Stochastic maximum-likelihood DOA estimation in the presence of unknown nonuniform noise," *IEEE Trans. Signal Process.*, vol. 56, no. 7, pp. 3038_3044, Jul. 2008.
2. Y. Gao, D. Hu, Y. Chen, and Y. Ma, ‘‘Gridless 1-b DOA estimation exploiting SVM approach,’’ IEEE Commun. Lett., vol. 21, no. 10, pp. 2210–2213, Oct. 2017.
3. H. Huang, J. Yang, H. Huang, Y. Song, and G. Gui, ‘‘Deep learning for super-resolution channel estimation and DOA estimation based massive MIMO system,’’ IEEE Trans. Veh. Technol., vol. 67, no. 9, pp. 8549–8560, Sep. 2018.
4. Y. Kase, T. Nishimura, T. Ohgane, Y. Ogawa, D. Kitayama, and Y. Kishiyama, ‘‘DOA estimation of two targets with deep learning,’’ in Proc. 15th Workshop Positioning, Navigat. Commun. (WPNC), Oct. 2018, pp. 1–5

**Deep Learning:**

1. M. Kim, N.-I. Kim, W. Lee, and D.-H. Cho, ‘‘Deep learning-aided SCMA,’’ IEEE Commun. Lett., vol. 22, no. 4, pp. 720–723, Apr. 2018.
2. H. Ye, G. Y. Li, and B.-H. Juang, ‘‘Power of deep learning for channel estimation and signal detection in OFDM systems,’’ IEEE Wireless Commun. Lett., vol. 7, no. 1, pp. 114–117, Feb. 2018.
3. https://www.youtube.com/watch?v=ZBnHoVMeeJE&pp=ygUKZG5uIGJhc2ljcw%3D%3D
4. https://www.youtube.com/watch?v=mH9GBJ6og5A&list=PLZoTAELRMXVPGU70ZGsckrMdr0FteeRUi&index=7&pp=iAQB

# APPENDIX

## PLAGIARISM REPORT



Ouriginal by Turnitin

### Document Information

| | |
|---|---|
| **Analyzed document** | ACF.docx (D167253833) |
| **Submitted** | 5/17/2023 8:58:00 AM |
| **Submitted by** | Yogeshwari Panneer Selvam |
| **Submitter email** | yogeshwari@ece.sastra.edu |
| **Similarity** | 0% |
| **Analysis address** | yogeshwari.sastra@analysis.urkund.com |

### Sources included in the report

### Entire Document

CHAPTER 1
LITERATURE REVIEW
Application areas for array signal processing include wireless communications, radar systems, and audio processing. Direction of Arrival (DOA) estimate plays a key role in these areas. Commonly utilised traditional DOA estimate methods include subspace-based approaches and beamforming algorithms. The use of Deep Neural Networks (DNNs) for DOA estimation has gained interest because to recent developments in deep learning. Due to their capacity to directly learn complicated patterns from data, DNNs present a possible substitute for DOA estimation. For the purpose of estimating DOA in antenna arrays, researchers have created DNN architectures. These systems frequently use the sensor measurements from the array as input to forecast the DOA angles of incoming signals. To identify spatial and temporal correlations in the array data, the DNN models use deep learning methods such convolutional neural networks (CNNs) or recurrent neural networks (RNNs).
Traditional DOA estimate methods, including Multiple Signal Classification (MUSIC) and estimate of Signal Parameters through Rotational Invariance Techniques (ESPRIT), have received a great deal of attention. In order to estimate the DOA,