ML with NetSim: Training a DNN for predicting SINR for 5G scenarios

Maitri Saraf*

Abstract: [1] Any general 5G scenario with gNBs can be modeled in NetSim's 5G library. UEs can be placed anywhere and received SINR can be obtained. We build, train, and validate a Deep Neural Network (DNN) using NetSim SINR output at different locations. Once the DNN is trained, we use it to predict the SINR - at locations not in the training data. These predictions are compared with Netsim simulation output. Results show an excellent fit. This methodology can be applied to numerous use cases and can be used to train real-world DNNs with NetSim's data.[2].

1 Introduction and Approach

Machine learning is a powerful tool for developing predictive models from data. Deep learning is a specialized type of machine learning that uses neural networks with multiple layers to learn hierarchical representations of data. DNNs have been highly successful in various applications.

Our objective is to demonstrate how Deep Neural Networks (DNNs) deployed in the real world can be trained using synthetic data from NetSim.

We begin by training a DNN to approximate the log distance mean path loss model. Distances and transmit power are input "features" while received

*School of Computer Science and Electrical Engineering, IIT Mandi

[1]Project download link: See Appendix-1. The Github link contains ipython files and necessary data for running the code.
Applicable release: v13.3 or higher
Applicable versions: All (Academic, Standard and Pro)

[2]This paper is meant for readers reasonably knowledgeable in machine learning and communication networking

power is the output "label". We start with this toy example keeping in mind readers who may be new to DNNs.

We then progress to training a DNN using NetSim's 5G simulation output data. For this, we chose to predict the downlink signal-to-interference-plus-noise ratio (SINR) for the classical 7-cell network. The base station (BS) to BS inter-site distance, termed ISD, is fixed and SINR measurements were collected at various (X, Y) points using the distance (d) and angle (θ) of the UE with respect to the central gNB within the central cell. Once the DNN is trained and validated, we chose a set of random points in the scenario which are not part of the training set. At these points, we use the DNN to predict the SINR. These predictions are compared to NetSim simulation results.

2 The NetSim Network Simulator

NetSim is a GUI-based, easy-to-use, high-performance network simulator and emulator. Through its user interface users can design networks by simply "clicking and dropping" network elements. Protocol and device properties can be configured by "right-clicking and editing attributes". Underneath the GUI is a high-performance C language-based simulation engine, network stack, and protocol library set. Simulations can be run for thousands of nodes, millions of packets, and billions of events. Post-simulation, NetSim displays performance measures as tables and charts for intuitive visualization. Launched in 2005, NetSim now is used by 500+ organizations across 25+ countries for network R & D. [6].

3 The Log Distance Mean Pathloss Model

The log distance mean path loss model is given by:

$$P_r = P_t + 20log_{10}(\frac{\lambda}{4\pi d_o}) + 10\eta log_{10}(\frac{d_o}{d})$$

where, P_r is the received power, P_t, is transmit power, d, is the transmitter-receiver distance, and η, is the path-loss exponent, which we take as 3. We take $f = 3.5GHz$, which is the frequency of the 5G C-Band, $c = 3 \times 10^8 m/s$, as the speed of light, and $d_o = 1m$, the reference distance. The wavelength λ can be calculated as $\lambda = \frac{c}{f}$.

3.1 Data Generation and Test-Train-Validate Split

We experiment with different values of two features, namely d (ranging from $30m$ to $500m$) and P_t (ranging from $20dBm$ to $43dBm$), to generate the label P_r using a given formula, which serves as the training data for the DNN. Before feeding the data into the DNN, we normalize the features to facilitate training.

The dataset comprises of 10,810 samples, which are divided into three sets: 70% for training, 20% for testing, and 10% for validation. The training set is utilized to train the DNN's parameters, while the validation set is employed for hyperparameter tuning and model selection. By using the validation error metric, we can fine-tune the model's hyperparameters such as the number of layers and learning rate, to achieve optimal bias and variance trade-off. This can be achieved through regression techniques in both overfitting and underfitting scenarios [4]. Lastly, the testing set is used to assess the model's performance on new, unseen data points and ensure its ability to generalize well. Refer Appendix 1 for more details.

3.2 DNN Architecture, Parameters, and Hyper Parameters

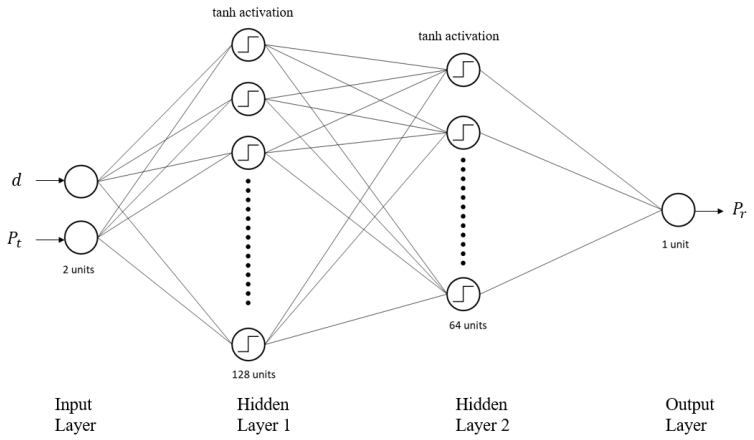


Figure 1: The DNN architecture used to approximate the Log Distance Mean Path Loss function

Fully connected deep neural networks (DNN) are a type of artificial neural

network where the architecture is such that all the neurons, in one layer, are connected to the neurons in the next layer [1]. We built such a DNN *sequential* model using Python's Keras library [5]. The model consists of 1 input layer with 2 neurons, 2 hidden layers comprising 128 and 64 neurons with *tanh* activation functions, and 1 output layer (Figure 1).

We use the Mean Squared Error (MSE) loss function with Stochastic Gradient Descent (SGD) as the optimizer. We choose a batch size of 4 with 50 epochs. The choice of batch size and epochs merit additional discussion.

The number of epochs is a hyperparameter that defines the number of times the learning algorithm will work through the entire training data set. We need to choose an optimum batch size considering the number of samples in the training data set and computation time. Small values give a learning process that converges quickly at the cost of noise in the training process. Large values give a learning process that converges slowly with accurate estimates of the error gradient. SGD with a specified batch size basically computes the gradients on small batches of data in order to reduce the variance of the updates.

3.3 Results

Distance	$d = 30.5m$	$d = 300.5m$	$d = 495.5m$
MSE (m^2)	0.0220	0.3927	0.4101
R^2	0.9993	0.9880	0.9875

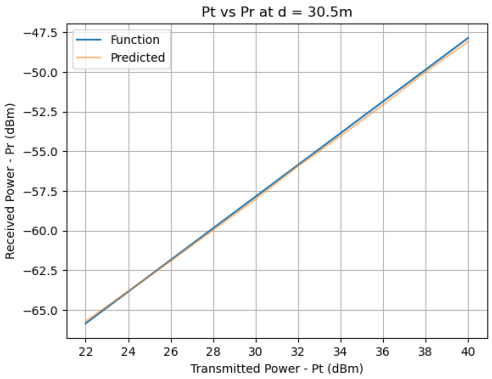
Table 1: Measures for Log Distance Path loss for various distances

We use MSE (mean squared error) and R^2 (coefficient of determination that provides information about the goodness of fit of a model) [3] for evaluating the model performance.

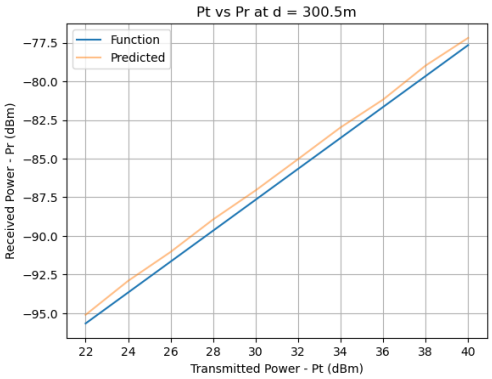
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

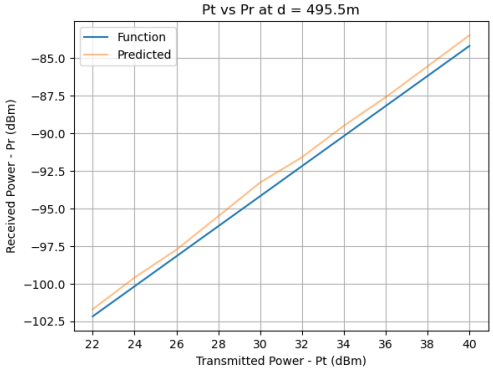
where,



(a) $d = 30.5m$



(b) $d = 300.5m$



(c) $d = 495.5m$

Figure 2: Plots: P_t vs P_r for varying d

N is the number of samples
y_i is the actual value of i^{th} sample
$\hat{y}_i$ is the predicted value of i^{th} sample
$\bar{y}$ is the sample mean

We plot P_t vs P_r to compare the actual(given by formula) and predicted (obtained from DNN) for 3 different values of d as shown in Figure 2 and obtain MSE and R^2 values as given by Table 1. The training MSE was obtained as 0.1661 m^2.

4 Approximating SINR in a 5G NR 7-cell scenario

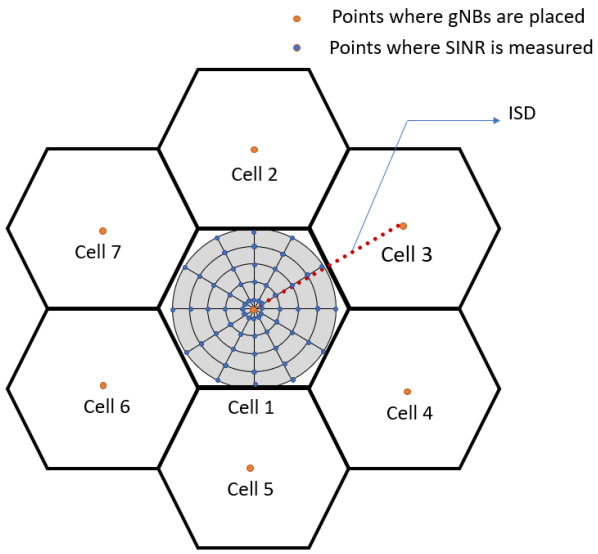


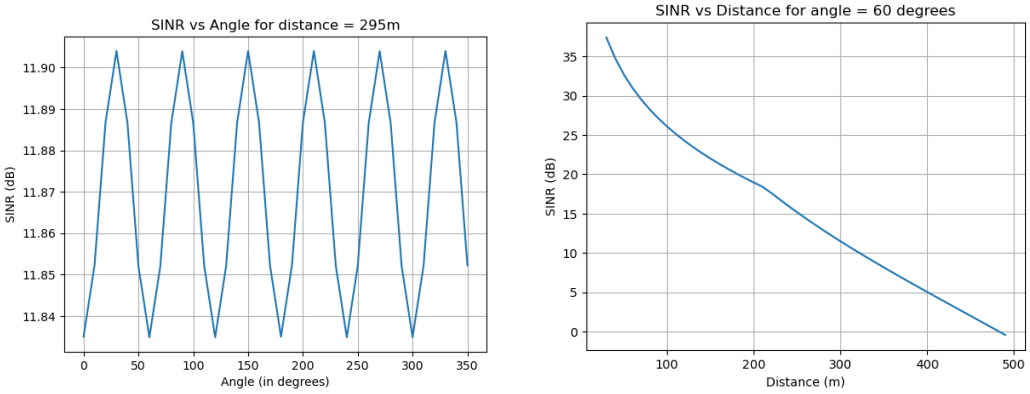
Figure 3: The standard hexagonal 7-cell network. Each cell is equipped with a BS or gNB at its center. Our goal is to approximate the SINR within the central cell using a DNN. We grid the area within the central cell in a polar fashion. Concentric circles are spaced at intervals of 10m, and the radial lines are drawn every 10 degrees, spanning the grey-shaded area. The SINR measurements are taken at the intersection points marked by blue dots. The distance between the center of Cell-1 and the centers of the other six cells is constant, and it is referred to as ISD.

We create a NetSim scenario consisting of a central gNB placed at position $(1500, 1500)$, with 6 gNBs surrounding it, in accordance with the assumption of a hexagonal cell structure. The SINR measurements are taken by placing UEs on a polar grid - UEs are distributed in a circular fashion around the central gNB at distances ranging from $30m$ to $ISD/2$, i.e. upto the edge of the central hexagon at intervals of $10m$, and at diverse angles from 0 to 360 degrees, as depicted in Figure 3. The UEs see a direct signal from the central gNB and interfering signals from the surrounding 6 gNBs. For details regarding creating scenario and running simulations in NetSim, refer

Appendix 2

We measure SINR at all the UEs by running NetSim simulations. Then we train the DNN to approximate this SINR based on the input features. The DNN would thus be able to predict the SINR at *any* point within the central cell. In the first case we keep the ISD fixed and in the second case vary ISD thereby making it an additional input feature. Fugure 4a shows variation in SINR with angle for a fixed distance of 295m and and Figure 4b shows SINR vs distance for a fixed angle of 60 degrees.

4.1 SINR calculation in NetSim as per 3GPP propagation models



(a) SINR vs Angle

(b) SINR vs distance

Figure 4: Plots: Variation in SINR with respect to angle and distance

The 3GPP pathloss equations defined in 39.901 standards, for a rural scenario assuming line-of-sight (LOS) communication between a UE and a gNB is

$$PL_{RMA_{LOS}} = \begin{cases} \text{PL}_\text{a} & \text{if 10m} <= \text{d}_\text{2D} <= \text{d}_\text{BP} \\ \text{PL}_\text{b} & \text{if d}_\text{BP} <= \text{d}_\text{2D} <= \text{10Km} \end{cases}$$

$$PL_a = 20log_{10}(40\pi d_{3D} f_c/3) + min(0.03h^{1.72}, 10)log_{10}(d_{3D}) - min(0.044h^{1.72}, 14.77) + 0.002log_{10}(h)d_{3D}$$

$$PL_b = PL_a(d_{BP}) + 40log_{10}(\frac{d_{3D}}{d_{BP}})$$

where,

$d_{BP} = 2\pi h_{BS} h_{UT} f_c / c$ is the breakpoint distance

$h_{BS} = 35m$ is the height of hBS antenna height

$h_{UT} = 1.5m$ is the height of hUT antenna height

$c = 3 \times 10^8$ is the speed of light

f_c is the centre frequency

$h =$ height of the gNB

$d_{2D} =$2D distance between the gNB and the UE

$d_{3D} =$3D distance between the gNB and the UE [2]

Now, the P_r the received power is P_t the transmit power minus the path loss. This is applicable to both the direct signal and interfering signals. The signal-to-noise ratio is defined as

$$SINR = \frac{\frac{P_t}{PL_1}}{\sum_{k=2}^{7} \frac{P_t}{PL_k} + BW \times N_0}$$

Where PL_1 represents the direct signal pathloss from gNB1, while PL_k represents the interfering signal pathloss from gNBs 2, 3, ..., 7, BW is the total bandwidth and N_0 is the total Noise.

4.2 Model with fixed ISD

To approximate the label, SINR, we vary the features namely (i) the distance of the UE from the central gNB and (ii) its polar angle. The reason for choosing the radial distance and polar angle rather than Cartesian X, Y, is because of the symmetry that is evident. Given that there are 6 surrounding gNBs we can expect the SINR measurements to exhibit a $(\frac{360}{6}) = 60$ degree periodicity. Specifically, we set the distance from the central gNB to range from $30m$ to $500m$ in increments of $10m$, and the angle to range from 0 to 360 degrees in increments of 10 degrees, with ISD set to $1000m$.

4.2.1 UE and gNB placement

Recall that, NetSim is a Discrete Event Simulator (DES) and not a radio planning tool, and hence measurements are recorded only where UEs are

present. We use NetSim's *Rapid Scenario Generator* to create this scenario with 7 gNBs and 1692 UEs. The gNB coordinates are given in the Table 2. We write python code for generating UE coordinates (x, y) taking distances (d in range (30, 500, 10)) and angles (θ in range (0, 360, 10)), as $(d\cos\theta, d\sin\theta)$, thus getting $(\frac{500-30}{10}) \times (\frac{360}{10}) = 1692$ co-ordinates. UEs are placed at each point where SINR is to be measured. Simulation is run (i) without configuring any network traffic since the focus is not on the network performance but on obtaining SINR measurements, and (ii) for the minimum duration required for UEs to get associated with gNBs. The SINR measurements in the downlink channel (PDSCH) are considered only after the UEs get associated with their respective gNBs.

gNB	X	Y
gNB_1	1500	1500
gNB_2	500	1500
gNB_3	1000	634
gNB_4	2000	634
gNB_5	2500	1500
gNB_6	2000	2366
gNB_7	1000	2366

(a) ISD=1000m

gNB	X	Y
gNB_1	1500	1500
gNB_2	1500-ISD	1500
gNB_3	1500-ISD$cos60$	1500-ISD$sin60$
gNB_4	1500-ISD$cos60$	1500-ISD$sin60$
gNB_5	1500+ISD	1500
gNB_6	1500+ISD$cos60$	1500+ISD$sin60$
gNB_7	1500-ISD$cos60$	1500+ISD$sin60$

(b) Varying ISDs

Table 2: gNB placement coordinates

4.2.2 Data Collection and Split

When running NetSim simulation, we enable radio measurements. We then record the following with the output value in a separate comma-separated value (CSV) file: (i) UE distance (ii) UE polar angle (θ) and (iii) measured SINR. We normalize the input features. Then we split this data as follows: 70% training data, 20% for testing, and 10% for validation.

4.2.3 Model Architecture, Parameters, and Hyper Parameters

The neural network model depicted in Figure 5 has 1 input layer with 2 neurons, 4 hidden layers with 16, 32, 64, and 128 neurons. The first 3 hidden

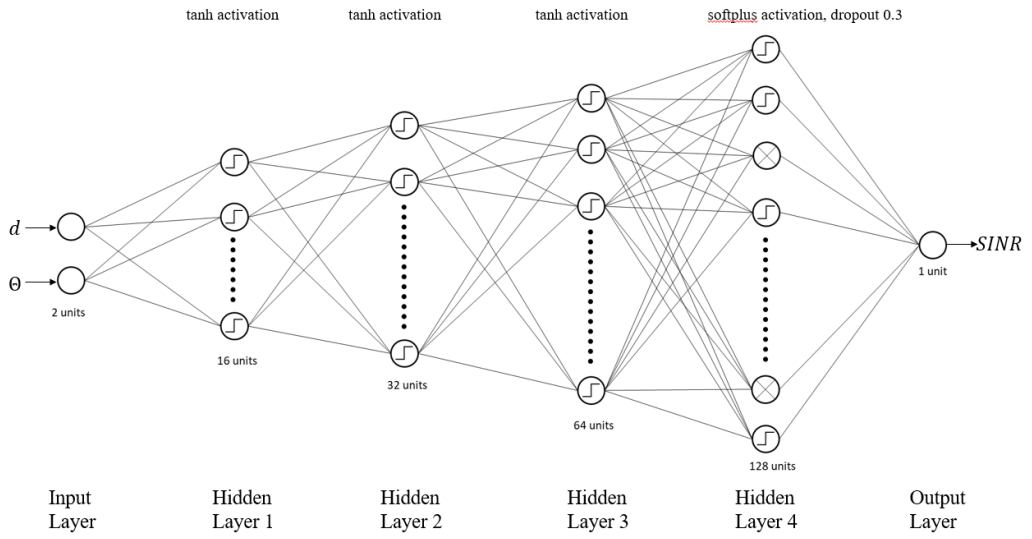


Figure 5: Neural Network Diagram for SINR approximation

layers use tanh activation and the last hidden layer uses the softplus activation. Softplus activation function ($y = log(1 + e^x)$) is a smooth continuous version of ReLU which gives a positive output and is also differentiable at all points unlike ReLU. Additionally, we apply a dropout with a probability of 0.3 on the layer with 128 neurons. Then we employ the mean squared error loss function with Adam optimizer running a batch size of 8 for 400 epochs. We tried SGD and Adam optimizers and found the Adam optimizer to converge quickly. Stochastic gradient descent algorithm takes a lot of iterations and makes N (number of samples) updates in one epoch. It moves very slowly in error curve having gentle gradient therefore it takes a lot of time to converge. Mini batch SGD just gives better approximate as compared to SGD with batch size 1. Adam on the other hand takes into consideration the momentum parameter which helps it to move quickly in the regions of gentle slope and also it takes different/adaptive learning rates for each parameter to make sure the updates are made inversely proportional to update history, hence it is optimal and yields faster convergence The plot of mean squared error with the number of epochs is shown in Figure 6.

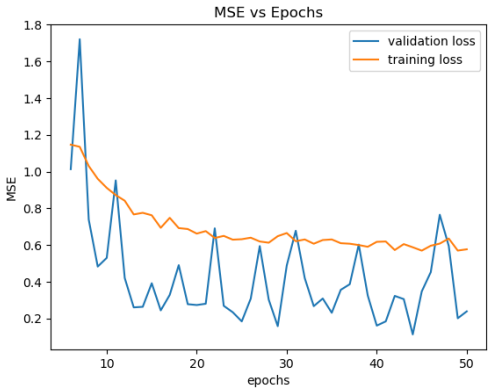


Figure 6: Plot: MSE vs Epochs - fixed ISD

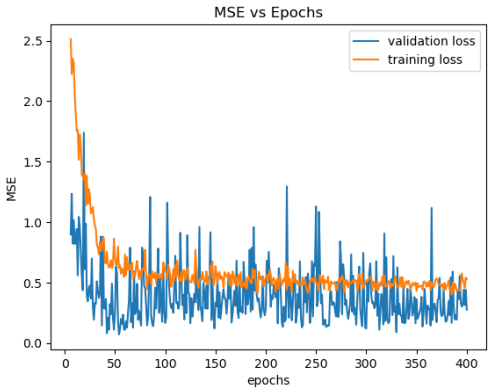
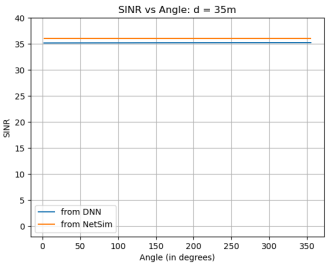
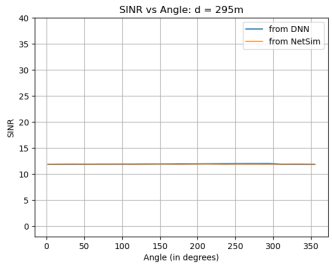


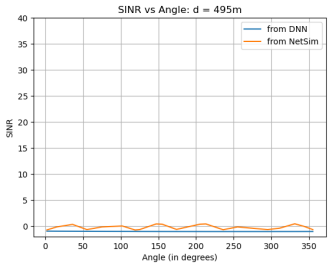
Figure 7: Plot: MSE vs Epochs - varying ISD



(a) $d = 35m$



(b) $d = 295m$



(c) $d = 495m$

Figure 8: Plots: SINR vs Angle for varying distances

Loss	MSE (dB^2)
Training Loss	0.4584
Validation Loss	0.2762
Test Loss (20% test data)	0.1567
Test Loss (plot)	0.0766

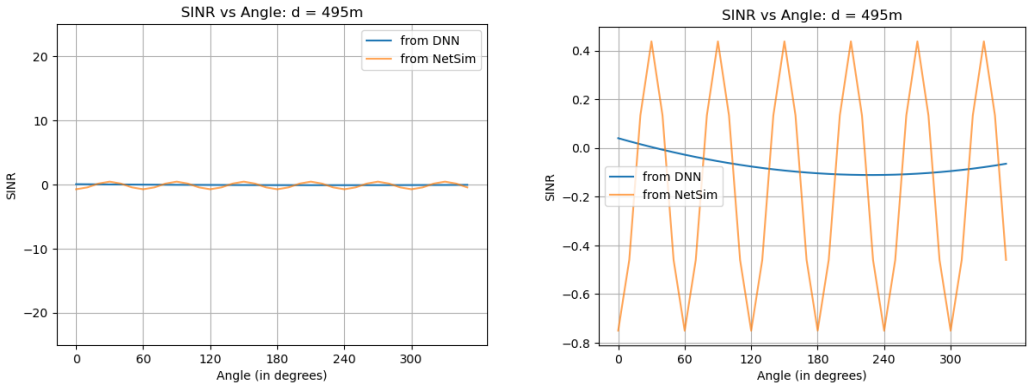
Table 3: Loss values for Model with fixed ISD

4.2.4 Results

To evaluate the model, we create a NetSim scenario with 60 UEs, 20 placed at a distance of $35m$, 20 at $295m$, and 20 at $495m$ with some random angles not in the training set.. We plot SINR vs distance, comparing the values

obtained from NetSim and values obtained from DNN as shown in Figure 8. The measures obtained for the DNN Model: MSE=0.0766 and R^2=0.9992. Refer Table 3 for loss values.

4.2.5 Limitations



(a) Plot depicting values obtained from NetSim close to DNN

(b) Plot zoomed in to show variation in SINR values

Figure 9: SINR vs Angle by taking angles from 0 to 360 degrees at interval of 10 degrees for distance = 495m

As mentioned earlier, the SINR measurements exhibit polar symmetry. This arises from the scenario geometry. There are 6 identical gNBs placed such that the centers of the gNBs lie on a circle whose center is the first gNB and each gNB in the surrounding *ring* is separated in polar angle by 60 degrees. Therefore, we can theoretically expect a 60-degree periodicity in the SINR measurements. This is indeed what we see from NetSim measurements shown in orange in Figure 9. However, the DNN is unable to capture this periodicity in its approximation - as seen in the blue plots in the same figure. We hypothesize that the reason the DNN is unable to capture this is because of how *small* the periodic variations are. If one were to measure the coefficient of variation, we see that it is equal to $\frac{\sigma}{\mu} = 0.95$, where σ is the standard deviation and μ is the mean. When we calculate the value of R^2 for the test data we get overall value as 0.999 but if we consider for individual plots like Figure 9b, we get the value as -102.397. In practice, R^2 will be negative whenever the model's predictions are worse than a constant function that

always predicts the mean of the data. In case of a periodic function, the mean lies in its equilibrium position whereas the DNN predicts value close to the actual value that can be greater than or less than the mean.

4.3 Understanding the periodicity of the SINR

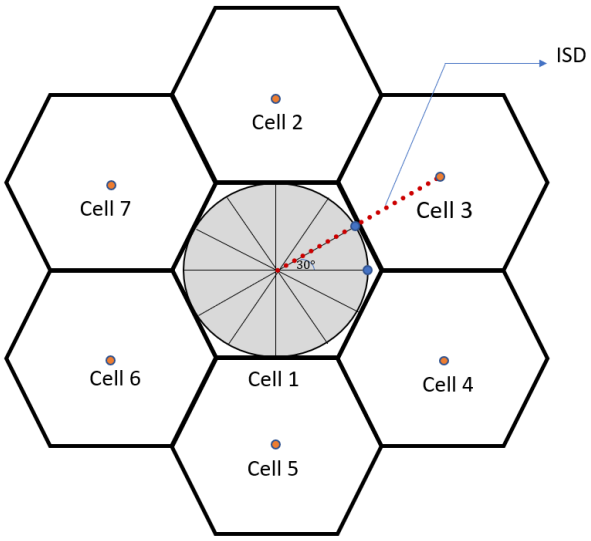


Figure 10: Approximating Log Distance Pathloss to understand periodic varioations in SINR values by calculating it for 0 degrees and 30 degrees

We try to understand the periodic variations by approximating SINR for log distance pathloss model for two points at 0 and 30 degrees. We assume that noise is very small. So, the relation between the received power and transmitted power is given as $P_r = P_t(\frac{d}{d_0})^{-\eta}$. We calculate the value for two points represented by blue dots in Figure 10. Taking $d_0 = 1$ and $\eta = 3$, the formula for SINR will be as follows:

$$SINR = \frac{P_{t_1}}{\sum_{k=2}^{7} P_{t_k}} = \frac{d_1^{-3}}{\sum_{k=2}^{7} d_k^{-3}}$$

We write a python code to obtain SINR for the two points by calculating the values of d_k which is the distance of the point where SINR is calculated from the k^{th} gNB where k varies from 1 to 7. The euclidean distances are

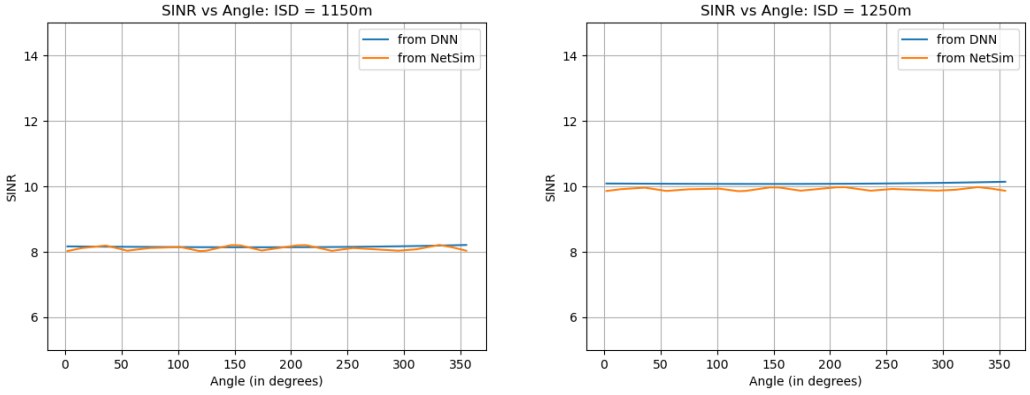
obtained by taking the co-ordinates of UE and gNB. The SINR values for 0 degrees and 30 degrees are 0.78 and 0.85 as expected (peak at 0 degrees and minimum at 30 degrees).

4.4 Model with varying ISD

We now also add ISD as an input feature along with the UE distance and angle from the central gNB. So, now we have 3 neurons in the input layer instead of 2. We vary the ISD from $1000m$ to $1500m$ in steps of $100m$, so we have 6 values of ISDs. We take UEs in the shaded area at distances $30m$ to $\frac{ISD}{2}$ from the central gNB in steps of $10m$ at angles from 0 to 360 degrees, so in total and 12744 data samples and we split for train, test and validation as the previous case. Figure 7 shows the plot of mean squared error with the number of epochs while training the data.

4.4.1 Results

We take two different values of ISD (not in training set), a mid range distance value and some random angles (same as in previous case) , and plot SINR vs angle (Figure 11). The measures of losses are given in Table 4.



(a) $ISD = 1150m$ (b) $ISD = 1250m$

Figure 11: Plots: SINR vs Angle for varying ISD at distance = 405m

Loss	MSE (dB^2)
Training Loss	0.5765
Validation Loss	0.2384
Test Loss (20% test data)	0.2591
Test Loss (plot)	0.0215

Table 4: Loss values for Model with varying ISD

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[2] *5G NR (3GPP) pathloss models*. Accessed: March 12, 2023. URL: https://www.tetcos.com/pdf/Simulation-Study-3GPP-5G-Pathloss-Models.pdf.

[3] *Coefficient of Determination, R-squared*. URL: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html.

[4] *Regularization in Machine Learning*. URL: https://www.simplilearn.com/tutorials/machine-learning-tutorial/regularization-in-machine-learning#:~:text=Regularization.

[5] *Tensorflow Keras Sequential Model*. URL: https://www.tensorflow.org/guide/keras/sequential_model.

[6] *TETCOS*. Accessed: March 4, 2023. URL: http://www.tetcos.com.

Appendix 1: Python DNN code

The entire DNN code is written in python. We use Keras which is a high-level API of tensorflow to build a sequential model. In the sequential API, data goes from one layer to another until it reaches the ouput layer. We use Jupyter notebook to write the code for training the DNN as Jupyter Notebook allows us to record code and analyze steps in a single document.

We extract the features from a Comma-Separated (csv) file using pandas dataframe and normalize them to input to the DNN. The data is then split into train test and validation using *train_test_split*. The model is constructed

using the keras *sequential* API adding layers successively. Then we add the dropout rate and activation function as layers. The model is then compiled using a suitable optimzer by choosing the optimum batch size and epochs through experimentation. The validation data is useful in adjusting hyper-parameters such as the number of layers, learning rate, batch size, epochs, etc as it validates the model on an unseen dataset. We then test the model on the test data and obtain suitable plots using *matplotlib* and obtain the measures using *sklearn.metrics*.

The entire code can be viewed here: `https://github.com/NetSim-TETCOS/DNN-for-NetSim-s-5G-Library-data/archive/refs/heads/main.zip`

Appendix 2: Procedure for running simulations in NetSim for the different cases

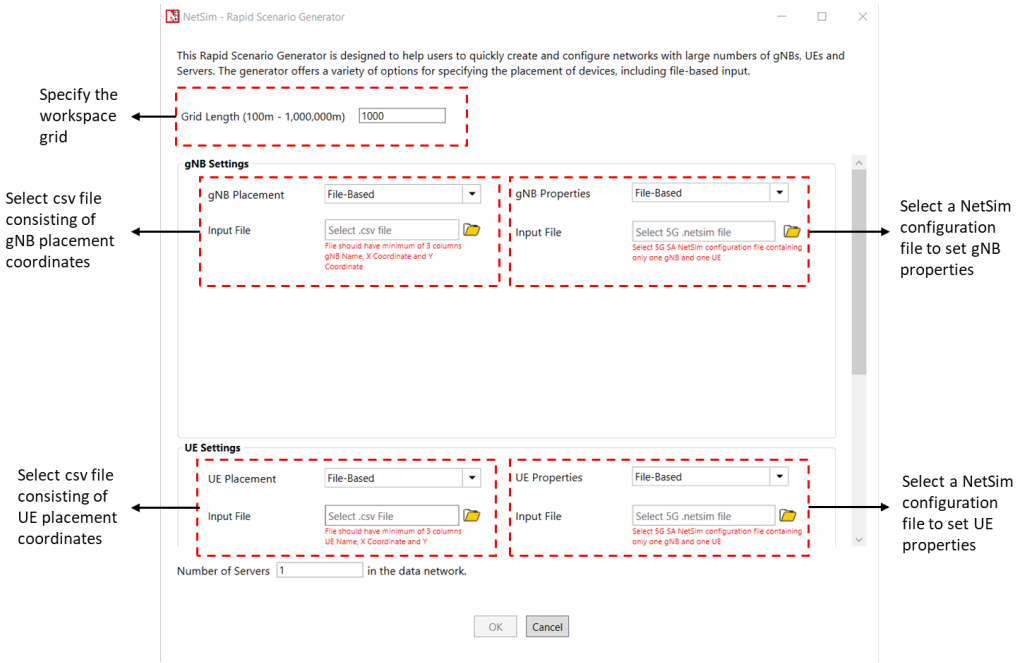


Figure 12: NetSim Rapid Scenario Generator for file-based placement

In NetSim v13.3.11, we open the Scenario Generator from NetSim file location by specifying four arguments namely, *Output path*, *Configuration helper files path where 'scenario_generator' folder will be present*, *Version_Name*

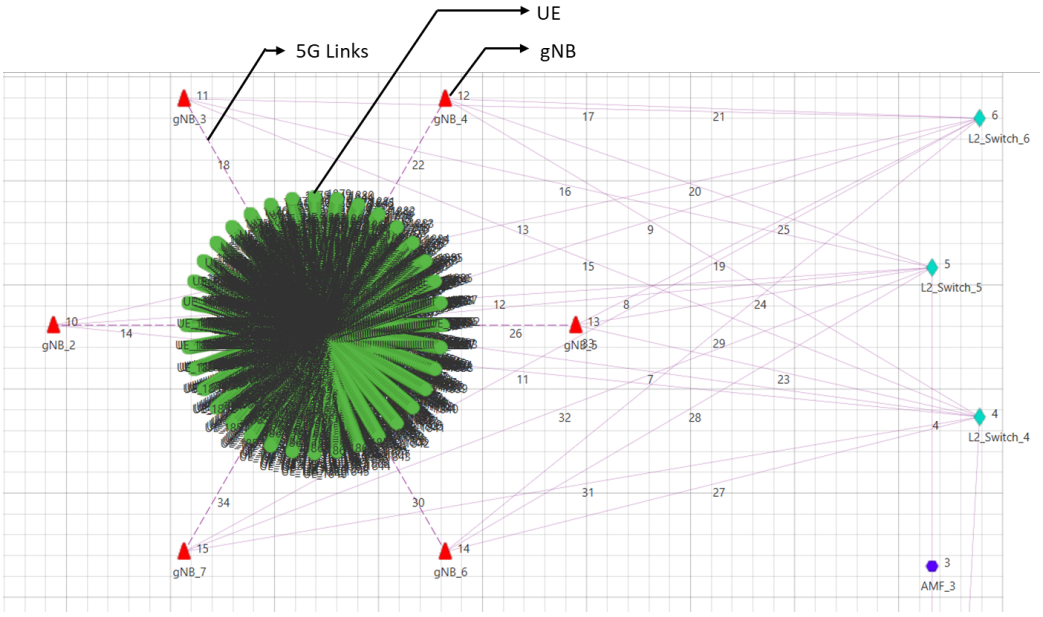


Figure 13: The scenario obtained using NetSim

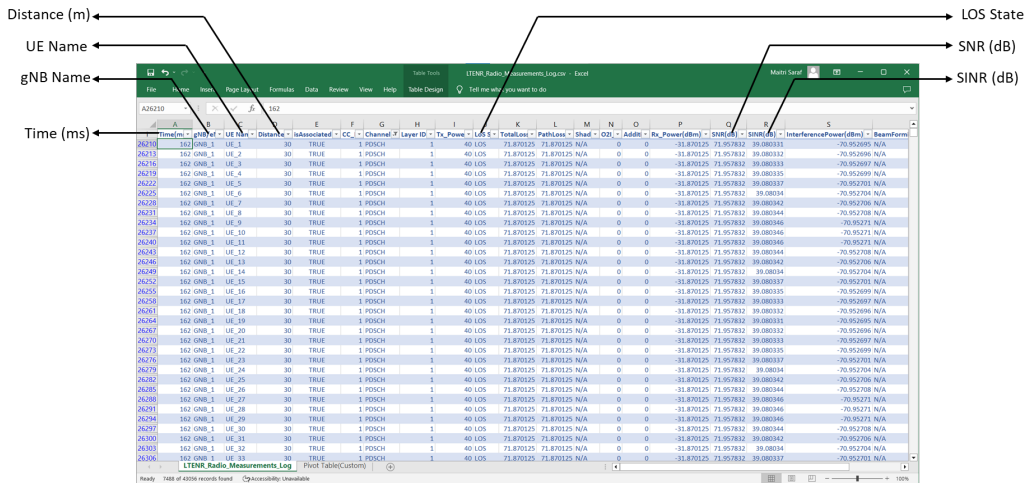


Figure 14: Radio Measurements Log File obtained by running the simulation in NetSim

and *Version_Number*. Using the Rapid Scenario Generator, (Refer Figure 12) we specify UE and gNB coordinates, and their respective properties are set

using a reference file consisting of only 1 UE and 1 gNB. We then import the configuration file to NetSim and obtain the configuration as shown in Figure 13 and run the application by enabling Radio Measurement Logs. We run the simulation without configuring any network traffic since the focus is not on the network performance but on obtaining SINR measurements. Simulation is run for the minimum duration required for UEs to get associated with gNBs. The SINR measurements in downlink channel (PDSCH) are taken only after the UEs get associated to their respective gNB. The result obtained is a csv file consisting of UE name, gNB with which it is associated, pathloss, SNR, SINR measurements as shown in Figure 14. From this, we extract the SINR values for training the DNN. Similarly, we obtain the configuration for test data.