Multi-Layer Perceptron for OBSS performance prediction: Predicting the throughput of IEEE 802.11 WLANs

Ramon Vallés Puig
UPF/ITU-T AI Challenge (ML5G-PS-013)

1. Introduction

One of the most common ways to evaluate network performance is by measuring its throughput. The throughput can vary depending on many factors. The physical layer in a network communication is one of those. Wireless local area networks, when placed near other wireless devices, may suffer from interferences that can disturb the communication between the Access Point and the Stations connected to it, which results in a low throughput and therefore a poor connection performance.

Another interference factor is the so-called Overlapping Basic Service Sets (OBSS), which happens when two or more BSS try to transmit on the same channel (or adjacent channel) at the same time without respecting the transmission shifts. This usually occurs in a practice commonly used by IEEE 802.11 WLANs, called Channel Bonding [1], that consist of combining the maximum number of adjacent channels in a BSS to increase transmission capacity. Thus, in these cases the initial objective of taking advantage of the maximum bandwidth is left behind, and both end up interfering with each other.

In this research we are going to exploit the power of Multi-Layer Perceptron algorithms so that we can predict the aggregate throughput of a BSS in a crowd scenario, where different Access Point may suffer from OBSS

2. Pre-Processing Data

All the data used during this research has been provided by the UPF and generated with the Komondor [2] wireless network simulator.

The dataset contained a huge amount of information, many of which was not relevant for the problem this research aims to solve. Some of the data that I have collected, or that in my opinion have seemed more relevant are, for each device: the type of device it is, the coordinates in which it is located, the minimum and maximum channel through which it is allowed to transmit. And regarding the data obtained by the simulation, I have also taken data about the transmission signal (RSSI, SINR) per station, the amount of time it has been transmitting by each channel of the AP, and the resulting throughput.

Once the information from each device was selected, the first challenge I encountered was the pre-processing data phase. This is a very delicate phase since the way we prepare the features will have a big impact when training our model.

2.1 Data cleaning

It is often the case that, when working with large amounts of information, that among the values we have in our dataset some are corrupt, or simply have no value at all. In our case, these values can be found, for example, when collecting the devices received power, where a device will receive an infinite amount of power on itself. Another case could be found when a STA receives so much interference that the SINR signal is completely obfuscated and its value is null or NaN. In this case, the NaN values must be inputted as negative SINR (between [0,-20]).

2.2 Data Transformation

Since my model aims to treat all the station devices as if they were one, and therefore one more property of each AP, the main objective is to reduce the number of stations to a single one.

To understand the idea, we will start from something very simple. Let's suppose two APs (AP1 and

AP2) with 2 and 4 devices respectively whose SINR values are AP1 [27,31], AP2 [3,4,15,26]. As we are interested in having a fixed number of features to train our model, and the number of devices may vary, we are going to encode the properties of the different devices according to the properties of their distribution (mean, standard deviation, num of STAs). In this way, the properties of the stations will be represented by a fixed number of features. (e.g. AP1:[mean:29, std:1.41, n:2] AP2:[mean:12, std:4.68, n:4])

This method has been applied to 3 main components of the Signal strength which are the previously mentioned SINR, the RSSI, and the distance from each device to the corresponding AP.

We are also interested in representing the amount of channels available, to do this, we will create a size 8 vector, where each position will correspond to a channel, and the value it contains will represent the amount of time it has been transmitting on that channel (AirTime).

Regarding the throughput: as we do not consider the stations individually, we only keep the total aggregate throughput of each AP, and discard the throughput per STA. The final result of the dataset will look like this: (for each AP in each scenario):

- **Features:**
  - [0-1] SINR [mean, std]
  - [2-3] RSSI [mean, std]
  - [4-5] Distance [mean, std]
  - [6] Number of stations
  - [7-14] Airtime per Channel
- **Target:**
  - [0] AP Throughput

3. Method

Once the dataset is ready to be used, the second phase in which we find ourselves is “how” the selected information should be used to obtain the maximum accuracy for new predictions. For this, I have considered it appropriate to design a model based on the already known Multi-Layer Perceptron.

To address the problem, we must first identify which data in the dataset are correlated, and select them carefully to avoid confusing our system.

With this in mind, we are going to divide the neural network in 3 parts: the first one will be in charge of measuring the signal quality of each AP with respect to its stations, the second one is focused on analyzing the available channels of the Access Point through which it can transmit, and the third and last part, takes both outputs from the Signal quality and AP airtime and computes the overall result.

3.1 Signal Quality

In this part, the system must understand the distribution of the stations, that is, how dispersed the devices are, what is the power that reaches these devices, and how much interference they receive from their neighbors.

For this reason, the first layer must treat these features (SINR, RSSI, DIST) separately, so that the system understands each distribution. For this, we consider 2 separate blocks consisting of a linear transformation with 2 input features, 1 output, a Parametric Rectified Linear Unit as the activation function and a 1-Dimensional Batch Normalization.

The second layer aims to study the information of the previous distributions, so the model is formed by a second linear layer with 4 neurons; the previously computed outputs and the number of STAs.

3.2 Available channels & AirtTime

This part of the Network is quite straightforward, it consists of a single linear layer, that gets as an input 8 features corresponding to each of the channels AirTime in percentage (%). This layer outputs a 3 dimensional array and is activated with a PReLU function.

3.3 Throughput Prediction:

Once the system has analyzed the signal of the AP in respect to its stations, and the airtime in which each channel has been transmitting, we can take this information and pass it through 2 fully connected hidden layers in the neural network that will combine the data to predict the final throughput of the AP. For this final layer, we will be using a ReLU activation function instead. This is because we are not interested in having negative throughputs, so all negative values will be set to zero.

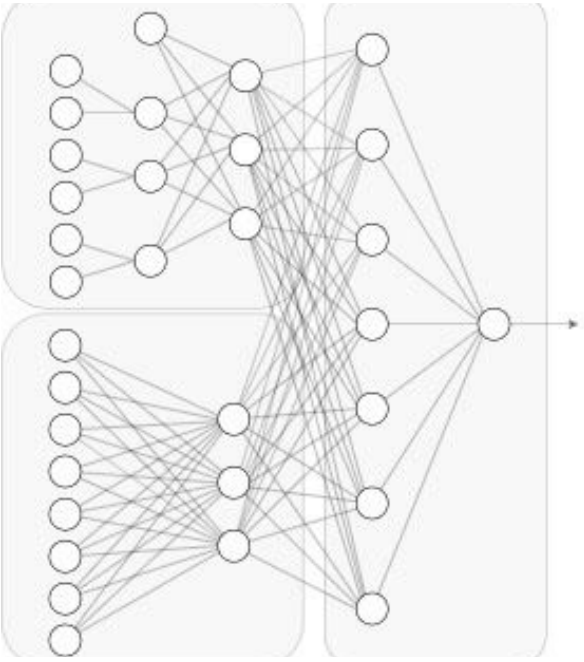


Figure 1: Architecture of the MLP divided in 3 blocks: Signal (top left), Aritime (down left), Throughput (right)

4 Training

Now that we have implemented the MLP, it is time to train it. To do so, I split the dataset into two, the first one for training (85%) and the second one for validation (15%).

The training was performed with an evaluation criterion based on the Mean Squared Error (MSE) and the optimizer chosen to optimize the training process is Adam, which has proven to be efficient for this type of ML algorithms.
For the training, several experiments were made by modulating the hyperparameters manually. The best results seemed to appear when the learning rate fell close to 0.025 and the number of epochs over 700 .

A curious fact that I found during the experiments is that adding the term weight decay, with the idea of regularizing the weights and thus avoiding a possible case of overfitting, resulted in a significant deterioration of accuracy in the vast majority of cases.

Example	LR	Epochs	Training MSE Loss	Validation MSE Loss
1	1E-2	1000	188	142
2	5E-2	500	208	158
3	5E-2	700	209	155
4	2.5E-2	700	171	135

Figure 2: Results of some experiments with different levels of Learning Rate and Number of Epochs

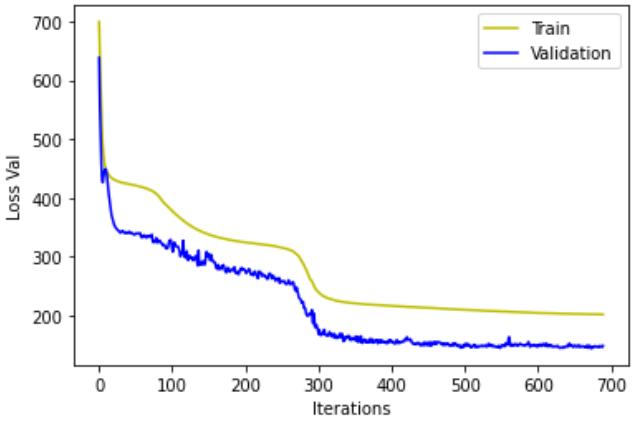


Figure 3: Learning curve of example 4

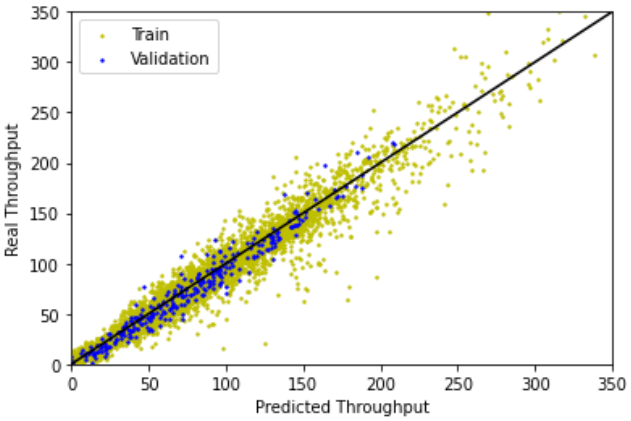


Figure 4: Graphical view of the Predicted throughput with respect to the real Throughput

5 Testing

To finally evaluate the results obtained, Pompeu Fabra University has provided us with 4 Datasets one for each different scenario according to the number of AP (4, 6, 8 and 10 AP) All mapped with a size of 80x60m.

The results of the System Testing are the following:

Scenario 1 (4 APs): TOTAL MSE: 425.51
Scenario 2 (6 APs): TOTAL MSE: 376.08
Scenario 3 (8 APs): TOTAL MSE: 270.29
Scenario 4 (10 APs): TOTAL MSE: 288.32

Figure 5: Prediction for scenario 1

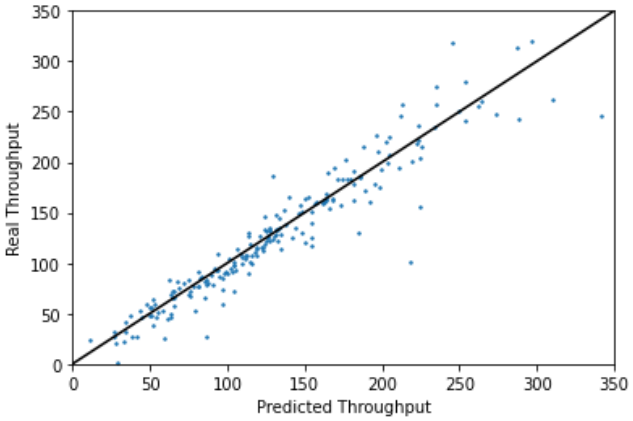


Figure 6: Prediction for scenario 2

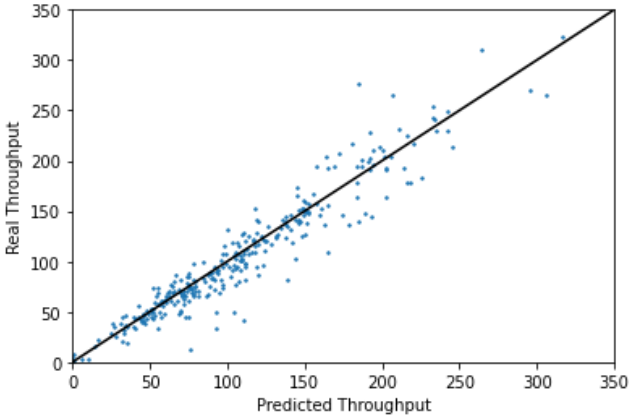


Figure 7: Prediction for scenario 3

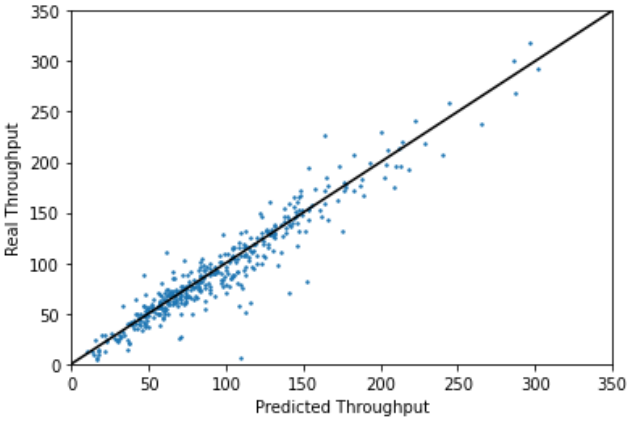
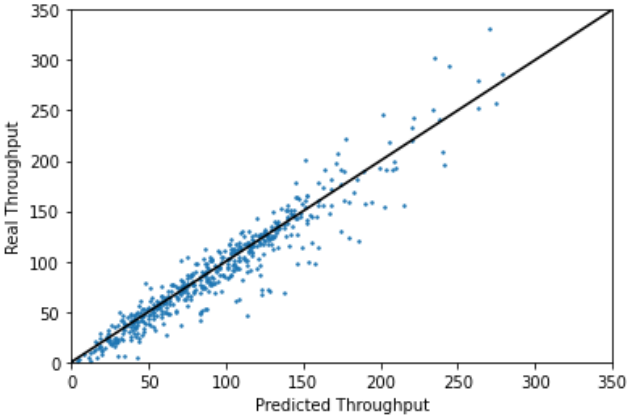


Figure 8: Prediction for scenario 4



Figures 5, 6, 7 and 8 show, for each scenario, how good the prediction is in relation to reality. These are represented around a linear function, so the further our result is from the line, the worse our prediction has been. This line also divides between those cases in which our system underestimates or overestimates the quality of the signal in each AP (if a value is above the line, it means that our prediction is below the real one, and vice versa).

6. Discussion and future work

We can therefore confirm that the designed neural network offers quite accurate predictions. However, it seems that, for some of those scenarios that our model has not previously seen (scenario 1, 2, 4), a decrease in the precision of the predictions is perceived. This may be due, among other things, to a loss of information during the Data Transformation phase previously explained, where it might happen that the distribution of the SATs and their signal are very dispersed, or the number of STAs is very small. In those cases, the average and standard deviation would not be sufficient to represent the signal of the set of devices.

A possible solution to this problem would be to add more parameters to the distribution, such as the minimum, maximum, some percentiles or even the peak of the distribution by means of the kurtosis function.
Another more complex solution, but one that would probably give more precise results and that could be implemented for future work, would be to leave distributions aside and modify the Neuronal Network so that instead of receiving a fixed number of features, it can deal with a dynamically sized input, and thus deal with the quality of all stations individually. This could be achieved with a Recursive Neural Network.

7. Conclusion:

We have described a deep learning algorithm (MLP) capable of approximating the performance of an OBSS [3]. We have analyzed the results with scenarios of different density and pointed out some limitations such as the loss of relevant information from the SINR and RSSI signal in the Data Transformation phase. Finally, we have proposed possible innovative improvements for the algorithm in question that could improve the accuracy of our predictions in a future project.

References

[1] Barrachina-Muñoz, S., Wilhelmi, F., & Bellalta, B. (2019). Dynamic channel bonding in spatially distributed high-density WLANs. IEEE Transactions on Mobile Computing.
[2] https://github.com/wn-upf/Komondor
[3] https://github.com/VPRamon/MLP-Throughput-prediction
[4] https://www.itu.int/en/ITU-T/AI/challenge/2020/Pages/default.aspx
[5] https://colab.research.google.com/notebooks/intro.ipynb#recent=true

Annotations

This project is part of a challenge promoted by Universitat Pompeu Fabra (UPF) and is part of the ITU Artificial Intelligence/Machine Learning in 5G Challenge [4].

The project has been implemented in python using the pytorch library and it has been tested in the Google Collaboratory Servers [5] to speed up the executions thanks to the free available GPUs in the cloud.