Prediction of Programmable Microwave Circuit Based on Machine Learning

Olivier Tomé [1],
[1] *Institut national de la recherche scientifique (INRS), University of Quebec, Montreal, QC, Canada*
olivier.tome@inrs.ca,

***Abstract*—In this report, we used a CNN model to predict the S-parameters of a fully reconfigurable microwave circuit. The circuit consists of 32x32 patches which can be conductive or non-conductive. The CNN model uses 5 convolutional layers. Due to the lack of training data, much of the work involved reducing overfitting using dropout layers and other regularizers. The model shows good prediction for a known function in a different position or size. It allows to obtain results in 500ms instead of several tens of minutes by simulation.**

I. Introduction

Reconfigurable systems are becoming increasingly popular, as they enable the system to be adapted to a new environment or standard. They also avoid the need to use different systems for different applications, since they can be used for multiple purposes. Reconfigurable systems are software-defined, i.e. we can select the functions the system is to perform via software. This can be as simple as a pin diode that can activate or deactivate a circuit. But for complex functions, reconfigurable systems can comprise anything from a few diodes (or other components such as varactors) to several thousand [1].

A new type of reconfigurable system is emerging: the field-programmable microwave circuit (FPMC). The FPMC, a versatile microwave device, can be customized to perform a wide range of microwave functions. For example, it can be used as a transceiver in one scenario, and as a radar in another. While field programmable gate arrays (FPGAs) have achieved a high level of reconfigurability in digital circuits, there has been a notable absence of devices with a comparable level of programmability in microwave communications. The realization of a fully reconfigurable microwave circuit promises to revolutionize microwave system design.

Reconfigurable and tunable microwave circuits have become an established field, but field-programmable microwave circuitry is significantly less developed. Reconfigurable and tunable microwave circuits encounter significant problems due to losses from the components used to realize the circuit agility which is one of the biggest problems realizing FPMC.

Recently, Lockheed Martin filed a patent for a field-programmable microwave array, which is a switch matrix to dynamically wire together microwave circuits from a plurality of predesigned microwave components such as baluns and amplifiers [2].

However, as mentioned earlier, a fully reconfigurable device will comprise several thousand pin diodes. The challenge of optimizing the configuration of such a device comes from the large number of parameters to be optimized. We can't use traditional or analytical solutions based on communication theories, mathematical models and optimization algorithms. One solution could be machine learning (ML). Indeed, an ideal algorithm would be able to control each diode according to the required function or circuit response, using S-parameters for example.

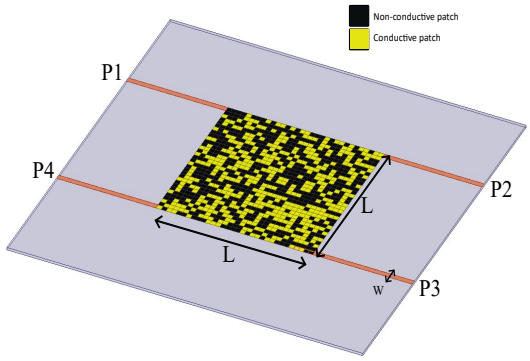

Fig. 1. 3D view of the proposed design.

In this report, we set up an ML model and trained it on a hypothetical fully reconfigurable FPMC. The simulated FPMC is shown in Fig. 1 and consists of four microstrip lines separated by 32x32 patches, which may or may not be conductive. The width of the microstrip line is set so as to obtain 50 Ω with a substrate of RO4350B with 20 mil of thickness and is equal to $w = 1.1$ mm and it is the same as the width of one patch. Thus, the total width of the matrix is $L = 35.2$ mm The number of possible configurations is equal to $2^{1024} \approx 1.7 \times 10^{308}$, Clearly, it is impossible to simulate every state, and there is no point in doing so, since a large number of configurations will have an insignificant impact (one or two conductive patches in the center won't affect the transmission line) or will serve no purpose (total conductivity). This type of device could be made using graphene and an electric field to control conductivity.

II. State of the art

The FPMC filed is actually investigated, but by hardware researcher. The few algorithms developed were usually made by the same people who have worked on FPMC. In [3] a genetic algorithm has been developed in MATLAB in order to

optimize the configuration of the cells. They use 256 unit cells and functions realized are basics. But later, the same research group has made other functions with the same FPMC [4]

A similar field, RIS, is being studied more closely for DL. RIS is like FPMS, but for radiation and transmission channel optimization. RIS can have many thousands of unit cells to control, so neuronal networks (NNs) could be really interesting for increasing speed and reconfigurability. It would be interesting to see how research uses DL in this field to adapt to FPMS.

A supervised learning based approach was proposed in [6] where a deep neural network (DNN) is trained offline to establish the implicit relationship between the measured coordinate information and RIS's phase configuration. They used as input the user location and the output is the phase configuration for each unit cell. Nevertheless, a major issue for supervised learning is how to obtain labels In [5] they use machine learning for modeling and understand the correlations between the phase shifts of RIS elements and attributes of the receiver (Rx) location. It utilizes this learned information to directly forecast achievable rates, eliminating the need for channel state information (CSI) or received pilots. Once trained, the model can effectively predict the optimal RIS configuration for new receiver locations within the same wireless network. They demonstrate that the proposed DL model can recommend near-optimal RIS configurations for tested receiver locations. Since there is a spatial correlation between each unit cell, they used a convolutional neural network (CNN) to model the RIS.

To overcome the lack of labeled data in supervised learning, the authors of [7] have used an unsupervised learning technique to determine the best phase shift configuration. The model uses 5 fully connected (FC) layers with a number of neurons multiple of the number of reflective elements. For learning, they used the adam optimizer

Another used technique is deep reinforcement learning (DRL). It can be trained with data collected online and use it for optimization problems. [8]

III. Training strategy

Since it would be too time-consuming to train an RNN, requiring numerous simulations that can take between 5 minutes and 1 hour, we decided to use ML to predict s-parameters without simulation. This way, all we need to do is simulate different configurations, start training a model with different architectures and choosing good hyper-parameters. If it was a RNN model, it would be difficult to make several attempts.

A. Dataset generation

The main problem was that we didn't have any data for training. To create a data set, we used HFSS an full-wave simulator with a Python script to automate generation. This script is available on Github with other scripts (named "Data_generatinon.py"). The first steps of this programme was to create the model on HFSS which is done in the file "To_HFSS.py" with the library PyAEDT. Once this was done, we generated a random pattern for the patch and simulated it. The results are then saved in a CSV file with all the S parameters (S11, S21, S22, S31, S32, S33, S41, S42, S43, S44). In this way, we obtain all S parameters, since we can use design symmetry (S21=S12, and the same for the others). The model is simulated between 5GHz and 15GHz with a step size of 100MHz. There are therefore 101 points recorded for each simulation. Simulation time is in the the order of 5min, but can sometimes reach almost 1 hour when there are coupling effects, reflected waves, etc...

Initially, only completely random patterns were generated Fig. 2 (a), but we soon stopped doing this as the signal could not pass through the system and all signals were reflected, so the model only had to always predict S21 = 0 and it had 100% accuracy. To improve the quality of the data set, a guided random model was therefore used. For example, we settled the first line always conductive, so that the signal could pass through port 1 to port 2, but always with the effect of the random pattern next to the line. In addition, to teach the model to recognize a filter, we randomly generated an open stub with random position and length Fig. 2 (d). For the coupling effect, we generated a random coupler between ports 1 and 3. The line between ports 1 and 2 is more or less close to the line between ports 3 and 4, also with a random length Fig. 2 (c)(e). Finally, to leave the model learn other effects, we used random again, but this time in a more intelligent way. The main problem with complete randomness is that almost once a patch is activated, the next is deactivated, the next is activated again and so on. So the idea is that if one patch is switched on, the next is more likely to be switched on too, to increase the likelihood of the signal getting through. The same effect is also used, but for a non-conductive patch Fig. 2 (e). Finally, to teach the model that if a conductive patch is far from the line, nothing or almost nothing will happen, we generated the above structure with a kind of noise Fig 2 (f).

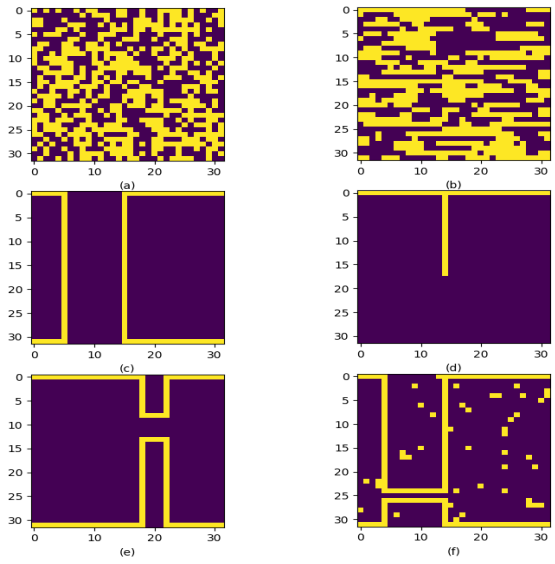

Fig. 2. Some patterns generated pseudo randomly.

With this method, almost 700 simulations were carried out, which is still very low compared to the complexity of the problem. One method used to artificially increase the data set

without simulation was therefore to use the symmetry of the model. If we apply symmetry in the YZ plane, port 1 becomes port 2 and vice versa, as do ports 3 and 4. As S21 = S12, and similarly for the other transmission parameters, we don't need to simulate this parameter again. The same applies to rotation and other axes of symmetry. An example of this method is shown in Fig. 3. With this technique, we multiply the number of data by 8. That is better, but as we will see later in this report, it is not enough.

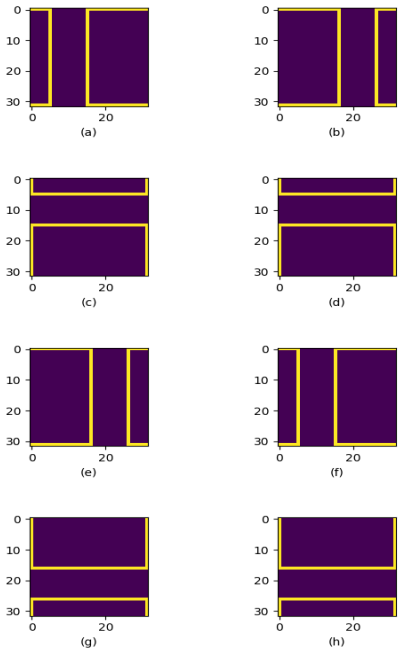

Fig. 3. Rotation and symmetry of the pattern: (a) original , (b) symmetry, (c) 90 deg rotated with the symmetry (d), (e) 180 deg rotated with the symmetry (f), (g) 270 deg rotated with the symmetry (h)

B. Architectures

Since there is a spatial correlation between each patch, the CNN is an appropriate model for this application. Many attempts have been made to find the best architecture. The input to the model is the 32x32 matrix of conductive and non-conductive patches (represented by 0 or 1). The output was firstly only the S21 (i.e. 101 points) and then all the S parameters (1010 points). Initially, we tried with 5 convolution layers with a maxpooling and batch normalization layer between each convolution layer. After a flatten layer, we added a fully connected layer. This is the initial approach, but not the final architecture. The loss function used is the Mean Squared Error function, commonly used for regression problems.

The optimizers tested were Adam and SGD, but in the end only Adam was used, as it gives fast convergence and good results. The learning rate was initially set at 0.001, then we tried to reduce it during the learning process. We also tried using a cyclical learning rate to reduce overfitting and increase accuracy as described in [9], but this didn't really work.

Batch size seems to be an important hyperparameter, since the results are completely different when it is modified. Indeed, with small batch sizes, updates tend to be noisier as each update is calculated on fewer examples. But small batch sizes can act as a form of regularization. They introduce more randomness into the optimization process, which can help prevent the model from overfitting to the training data. However, this slows down the learning process as the model is updated more frequently. In contrast, we can increase the batch size to obtain more stable estimates of the gradient of the loss function, as they are calculated from a larger number of data points. However, a larger batch size can exacerbate overfitting. Indeed, as updates are less noisy, the model can learn to fit the training data too closely. As we don't really have a lot of labeled data and after many tests, we choose to reduce the batch size to 16.

Finally, dropout layers were also used. Dropout layers are a regularization technique commonly used in deep learning models. The main advantage of dropout layers is to prevent over-fitting and improve model generalization performance. Dropout works by randomly setting to zero a fraction of the neurons in a layer during each training iteration. This means that the output of these neurons is ignored during that iteration, temporarily removing them from the network. This is only during the training phase with training data. For validation or test data, the dropout layers are ignored. We can choose the percentage of neurons set to zero by defining a group coefficient between 0 and 1. The coefficient is generally between 0 and 0.5.

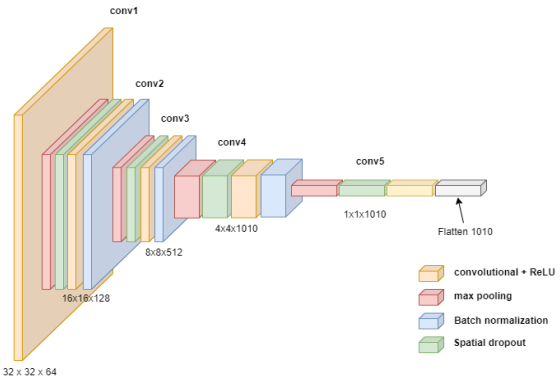

Fig. 4. Final architecture.

IV. Implementation and Results

The model was implemented in python with the Keras library (code available on Github). Initially, to simplify training, we tried to predict only S21. This way, there were only 101 points to predict. But after a few trials, we tried predicting all the S parameters and, surprisingly, results were better than with just the S21 parameter.

For the first trials, the architecture used was almost the same as the one shown in the figure 4, but with a fully connected layer instead of the last convolutional layer. During training, however, an overfitting occurred. Training loss continued to decrease, but validation loss did not. We therefore tried to reduce the complexity of the model by decreasing the number

of layer, or changing the number of filter. When the model was less complexes, there was no overfitting but the accuracy was also degraded Fig. 5.

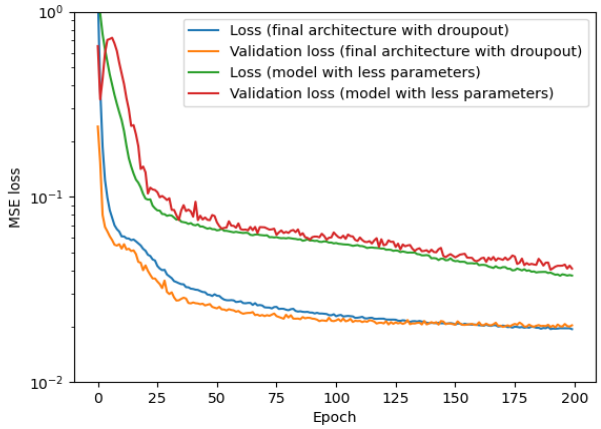

Fig. 5. MSE Loss of training and validation output between two architectures.

The final architecture used is shown in Fig. 4. The filter size for the first layer is 9x9, then 5x5,, 3x3 and 2x2 for the two last layers. The total number of learnable parameters is 3.9×10^6.

We added dropout layers to reduce overfitting. The dropout layers effectively reduced the effect, but not the loss. As can be seen in Fig. 6 for different dropout layer coefficients. We therefore tried to increase the complexity to check whether our model was not powerful enough. But once again, if we add a layer (fully connected or convolutional), it either makes no difference, or is overfitted. The only way to improve the model is to increase the amount of data.

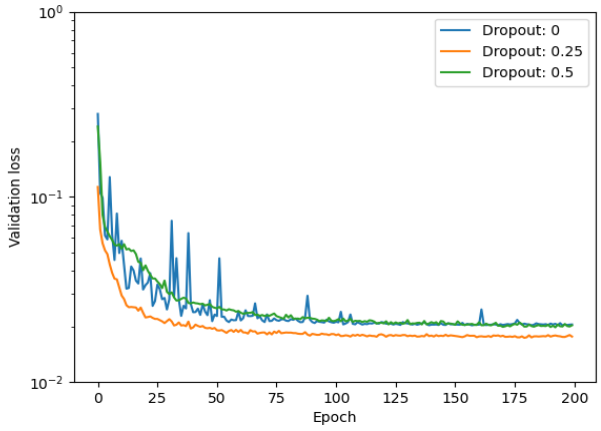

Fig. 6. Validation loss for different dropout coefficients.

We therefore tried to predict the S parameters from a new input, but not from a new function (it was a coupler, but with a different length and position from the training data). The predicted output and simulated result are shown in Fig. 7. For S21 and S11, the prediction is good, but for S31, the shape of the curve is good but slightly shifted downwards. This seems to work for an already known function, but if we try it with a completely different function (like a power divider) or a totally random model, the results can be very different.

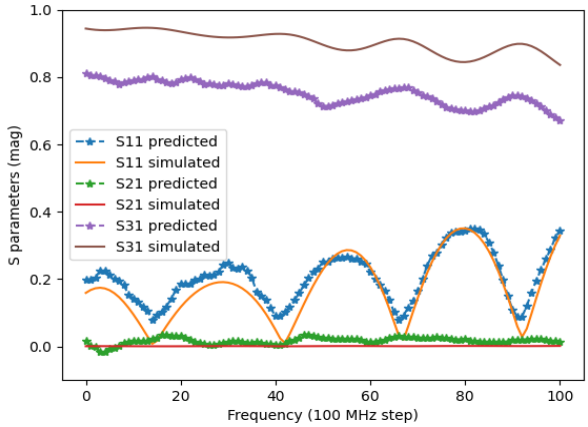

Fig. 7. Predicted output with simulated data for S11, S21 and S31.

V. Conclusion

In this project, a CNN model was implemented in Python. The model was trained with labeled data generated for this project. The importance of a representative, high-quality data set was taken into account. Much of the work involved testing different architectures, different hyperparameters such as learning rate, dropout coefficient, number of filters and their sizes. Regularizers such as the l2 penalty were also tested, as well as a learning rate programming techniques. Finally, some predictions were tested and gave good results, but not in all cases, due to the small size of the dataset. But instead of waiting between 5 minutes and an hour for simulation, here we obtained results in less than a second. One possible way of improvement is to increase the data set to obtain better predictions. It would then be possible to train an RNN model for control based on the predictions obtained with this CNN model, or to use transfer learning to train a model for a larger reconfigurable circuit. With a physical device, the work of generating data sets would be much faster, and we could have much larger data sets at our disposal. It also possible to use this model to design antenna.

References

[1] J. C. Lyke, C. G. Christodoulou, G. A. Vera and A. H. Edwards, "An Introduction to Reconfigurable Systems," in Proceedings of the IEEE, vol. 103, no. 3, pp. 291-317, March 2015, doi: 10.1109/JPROC.2015.2397832.

[2] "Field programmable microwave arrays", Jul. 2014. US8774730B1

[3] A. Saifee et al., "Reconfigurable Microwave Components Implemented using Field Programmable Microwave Substrate," 2022 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO), Limoges, France, 2022, pp. 1-4, doi: 10.1109/NEMO51452.2022.10038958.

[4] A. Saifee et al., "Reconfigurable Microwave Components Based On Optimization of Field Programmable Microwave Substrate," 2023 IEEE/MTT-S International Microwave Symposium - IMS 2023, San Diego, CA, USA, 2023, pp. 1061-1064, doi: 10.1109/IMS37964.2023.10187911.

[5] Sheen Baoling, Yang Jin, Feng Xianglong, Chowdhury Md Moin Uddin. (2021). A Deep Learning Based Modeling of Reconfigurable Intelligent Surface Assisted Wireless Communications for Phase Shift Configuration. IEEE Open Journal of the Communications Society. PP. 1-1. 10.1109/OJCOMS.2021.3050119.

[6] C. Huang, et al., "Indoor signal focusing with deep learning designed reconfigurable intelligent surfaces," 2019, [Online]. Available: https://arxiv.org/abs/1905.07726.

[7] J. Gao, C. Zhong, X. Chen, H. Lin, and Z. Zhang, "Unsupervised Learning for Passive Beamforming," IEEE Commun. Lett., vol. 24, no. 5, pp. 1052–1056, May 2020, doi: 10.1109/LCOMM.2020.2965532.

[8] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial Neural Networks-Based Machine Learning for Wireless Networks: A Tutorial," IEEE Commun. Surv. Tutorials, vol. 21, no. 4, pp. 3039–3071, Fourthquarter 2019, doi: 10.1109/COMST.2019.2926625.

[9] Cyclical Learning Rates for Training Neural Networks,Leslie N. Smith,2017,1506.01186, arXiv, cs.CV