Digital Pre-Distortion using Deep Learning

A Project Report

submitted by

PUVI ARASU N P

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2022

ACKNOWLEDGEMENTS

I would like to express my sincere thanks of gratitute to Prof.Devendra Jalihal for giving me the opportunity to work under him and for guiding me throughout my B.Tech project. I'm thankful to Mr. Krishna Kumar for taking his time out of his busy schedule to clear my doubts, correct my mistakes and guide me in the correct direction. I extend my gratitude to all my professors, without whom I would not have acquired the knowledge necessary to pursue this project. Special thanks to my family and friends for being my constant pillar of support.

ABSTRACT

The objective of this project is to implement Digital Pre-distortion(DPD) for different Power Amplifiers using deep learning and analyse the performances of Neural network based DPD models in Linearization. Power amplifiers when operated below a certain power level amplify the input signal linearly but beyond that, the power amplifiers introduce non-linearities. Pre-distortion is a technique used to improve the linearity of the power amplifiers. In this method, the signal is distorted before it enters the Power amplifier so that the output will be linear even when operated at higher power. In the past, pre-distorter models were based on the Volterra series, memory polynomials and Wiener-Hammerstein models. Then machine learning-based Predistorters were proposed and proved useful. Since then, so many ML-PD linearizers have been developed. PD linearizers using shallow neural networks map the inverse transfer function well in the absence of system characteristics (IQ imbalance, DC offset). On the other hand, deep neural networks can even map the complex inverse transfer function under different system conditions.

In this project, I use different neural networks to implement pre-distortion for different types of power amplifiers. I start by implementing DPD for memoryless power amplifier models using Multilayer Feedforward Neural networks, then use LSTM networks for realizing DPD for power amplifier models with memory, and finally deal with DPD for a real-world Power Amplifier. Two methods of training the neural networks are used in the project and also discussed in this report. We analyse the performances of the NN-based DPD models using AM-AM, AM-PM and spectral plots.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Power amplifiers(PA) are electronic amplifiers used to increase the magnitude of power of the input signal. They are an important part of wireless communication transmitters. At low levels of power, power amplifiers amplify input signals in a linear fashion, but beyond a certain level of power, the power amplifier introduces nonlinearities. Non-linearity introduces spectral re-growth outside the allocated bandwidth thus violating the spectral mask. Power Amplifiers should be operated at close to their peak power to achieve high efficiency. Hence, we have to operate PAs in their non-linear region of operation for maximum efficiency.

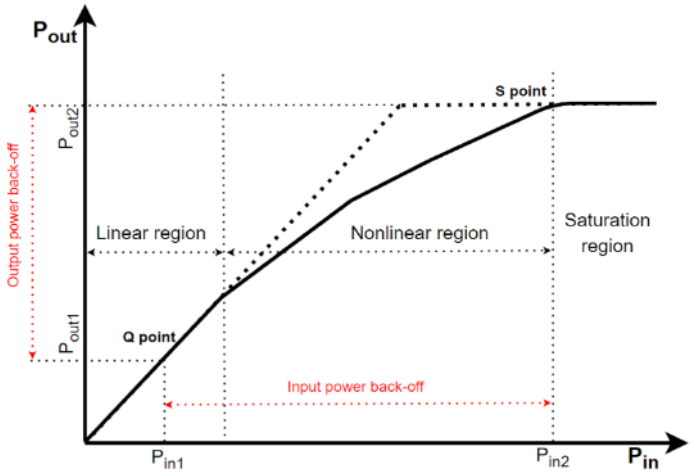


Figure 1.1: **Non-linearity in Power Amplifier Output**

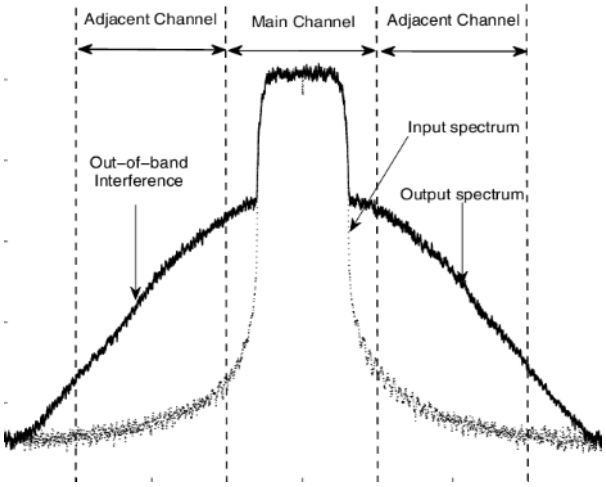


Figure 1.2: **Spectral Regrowth due to non-linearity**

To achieve linearity and efficiency simultaneously, Pre-distortion techniques are employed. In this method, the signal is distorted before it enters the power amplifier so that even at higher output power the output remains linear. Basically, the PD linearizers generate a non-linear transfer function which is inverse to the transfer function of PA.

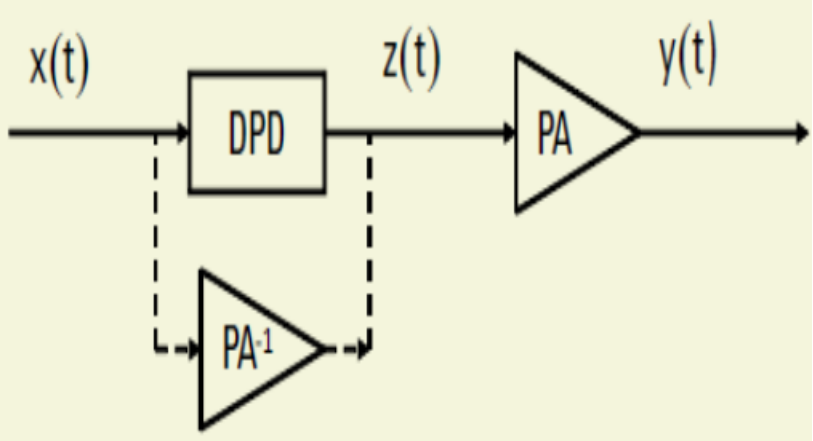


Figure 1.3: Pre-Distortion

The initial working range of the Power Amplifier is the input power range where the output is linear. The objective of implementing Pre-distortion is to increase this range. We'll try to increase the working range of PA models to the entire non-saturating region but when it's not possible we'll try to extend it to at least 1-dB compression point. 1 dB compression point is the input power at which the linear output power and the output power of the amplifier differ by 1 dB.

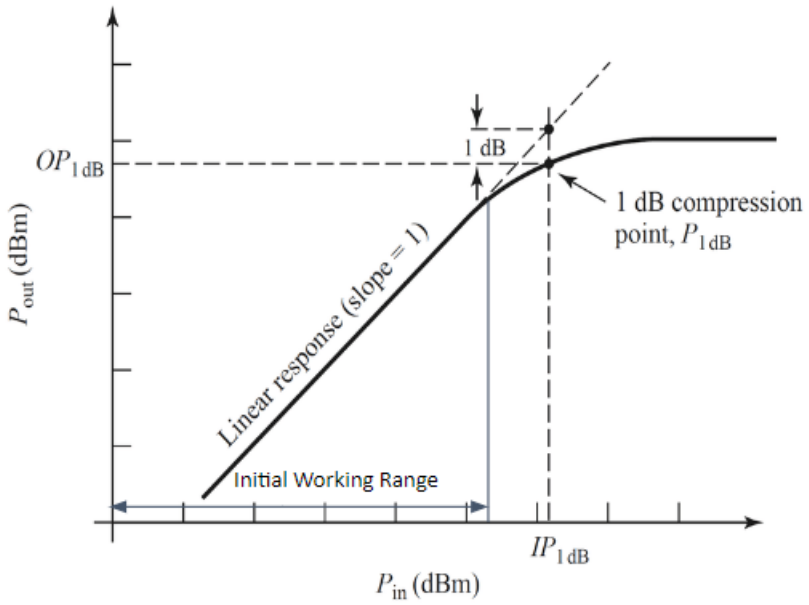


Figure 1.4: 1-dB compression point

Linearizing power amplifiers can be done in both analog and digital domains. But the digital domain is often preferred due to the repeatability of implementation, versatility, and flexibility of design. The behavior of the PA and its inverse (i.e pre-distorter) can be represented using baseband equivalent discrete-time models. Therefore Digital Pre-Distortion(DPD) uses baseband digital signal processing to pre-distort the envelope so that the distortion introduced by the PA is able to recover the original envelope.

Various conventional models like wiener, hammerstein and memory polynomial were used to implement Digital Pre-distortion for PAs. In the recent years, with the success of Deep learning in solving communication problems, DPD researchers have shifted their attention to Neural Network based Pre-distorter models due to their excellent modelling ablilities. Hence, I have also implemented DPD for different Power Amplifier models using neural networks.

1.1 Historical approaches & NN-based approach to DPD

Different DPD models have been proposed to compensate for the nonlinearities of power amplifiers. In the past, Volterra series-based models like memory polynomial model, and generalized memory polynomial model were most commonly used. Volterra-based models use polynomials as basis functions, resulting in high correlation between different basis functions. Thus, even with a large number of basis functions, Volterra-based models would be limited in performance and may deteriorate numerical stability. Signals employed in 5G communication systems have wider bandwidth and higher Peak-to-Average Power Ratio(PAPR), hence the static nonlinearity and memory effects of RF PAs become much more severe and complex, which results in the limited linearization performance with the conventional DPD models. Therefore, it is urgent to develop new DPD models with better nonlinear fitting capability.

Neural networks have recently attracted attention due to their excellent modeling capabilities and are considered promising modeling methods for DPD. Various neural network-based models have been developed for DPD. In this study, we have implemented different NN DPD models for different Power Amplifiers. I have used the Multilayer Feedforward Neural Network(MLFFNN) to perform predistortion for Mem-

oryless power amplifier models because deep neural networks can map the inverse characteristics of power amplifiers in the presence of system conditions(IQ imbalance, DC offset) unlike shallow neural networks. Feedforward networks can only process single data points but not entire sequences of data. Hence they are not well suitable for implementing DPD of power amplifiers with memory. Long short-term memory networks(LSTMs) are recurrent neural networks (RNNs) that are used for time-series data modeling and prediction. LSTM units can capture and exploit the memory effects of PA. Hence we have used LSTM-based neural networks to model DPD for PAs with strong memory effects.

CHAPTER 2

BACKGROUND

2.1 Choosing the Platform: Matlab

Due to practical constraints, I could not work with power amplifier hardware. So I had to simulate power amplifiers and implement DPD for the PA models. Matlab Simulink has RF block set library using which one can model Memory polynomial, Generalized Hammerstein, Cross-term Memory, and Cross-term Hammerstein power amplifiers. It also has a deep learning library that contains LSTM layers, fully connected layers and activation layers which are enough for this project. Because of these reasons I chose Matlab as the platform to implement our project instead of others like Keras even though they have more and better neural network models.

2.2 Matlab-Simulink Model of Power Amplifier

Memoryless Polynomial, Memory Polynomial, Parallel Wiener models used in this project are simulated using the following Matlab Simulink design.

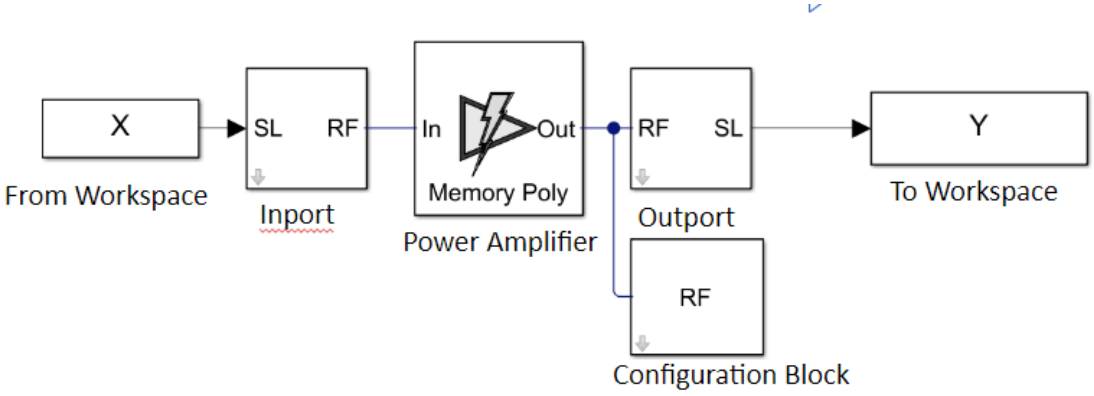


Figure 2.1: Simulink model

From workspace block takes the discrete input data from matlab workspace and generates analog simulink signal(by linear interpolation). Inport block converts the simulink

signal to RF signal. Power Amplifier block amplifies block amplifies the signal. Outport block converts the RF signal to simulink signal. To workspace block converts the simulink signal to discrete data and sends the output of power amplifier to matlab workspace. Configuration block is used to control the simulation settings.

Memory Polynomial Power Amplifier is characterized by its coefficient matrix. If the input to the power amplifier is x(n) and the coefficient matrix of the PA is

$$C = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1K} \\ \vdots & \vdots & \vdots & \vdots \\ C_{M1} & C_{M2} & \dots & C_{MK} \end{bmatrix}$$

Then the output of PA is y(n) = $\sum_{m=1}^{M} \sum_{k=1}^{K} C_{mn}$ x(n-m+1) $|x(n-m+1)|^{k-1}$.
If the coefficient matrix of the power amplifier is a row matrix

$$C = \begin{bmatrix} C_1 & C_2 & \dots & C_K \end{bmatrix}$$

Then it is a memoryless power amplifier and the output of PA is
$y(n) = C_1x(n) + C_2x(n)|x(n)| + C_3x(n)|x(n)|^2 + \cdots + C_Kx(n)|x(n)|^{K-1}$
If the coefficient matrix of the power amplifier is a column matrix

$$C = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_M \end{bmatrix}$$

Then the output of PA is $y(n) = C_1x(n) + C_2x(n-1) + C_3x(n-2) + \ldots + C_Mx(n-M+1)$
If the coefficient matrix of the power amplifier is a diagonal matrix

$$C = \begin{bmatrix} C_1 & 0 & \dots & 0 \\ 0 & C_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & C_K \end{bmatrix}$$

Then the output of PA is $y(n) = C_1x(n) + C_2x(n-1)|x(n-1)| + C_3x(n-2)|x(n-2)|^2 + \cdots + C_Kx(n-k)|x(n-k)|^{K-1}$

2.3 Input signals

QPSK signal is used as input for Memoryless Power Amplifier models,Memory Polynomial model and Parallel Wiener model. The specifications of the QPSK signal is given below

Symbol Rate	1 MHz
Spectral shaping Pulse features	SRRC Pulse with roll off factor = 0.22, Bandwidth = 1.22 MHz, Time duration = (-4T,4T) => 65 samples
Sampling Rate	8 MHz
Oversampling factor	8 samples/symbol

Figure 2.2: QPSK signal characteristics

OFDM signal is used as input for NXP Airfast Power Amplifier model. The specifications of the OFDM signal is given below

Modulation	64 QAM
Bandwidth	100 MHz
Number of subcarriers	3276
Sampling Rate	122.88 MHz

Figure 2.3: OFDM signal characteristics

2.4 Training and implementing NN-based DPD-1

First we send the training input(QPSK or OFDM data) to power amplifier and get the training output. Then we train a neural network with training output as input to the neural network and training input as target output of the neural network. This ensures that the trained neural network acts like the inverse of the power amplifier. After training the neural network, we send the test input to it and pass the output of the NN to PA. The output we get at the end is the compensated output. We compare the output before implementing DPD with the compensated output using AM/AM, AM/PM and spectral plots.

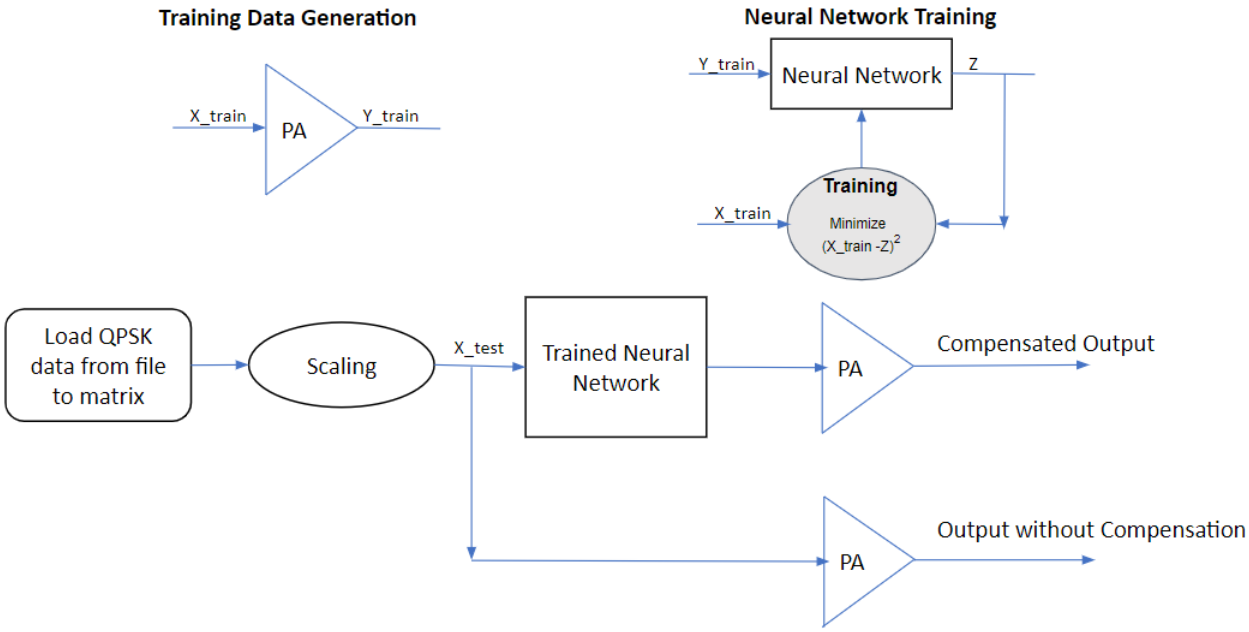
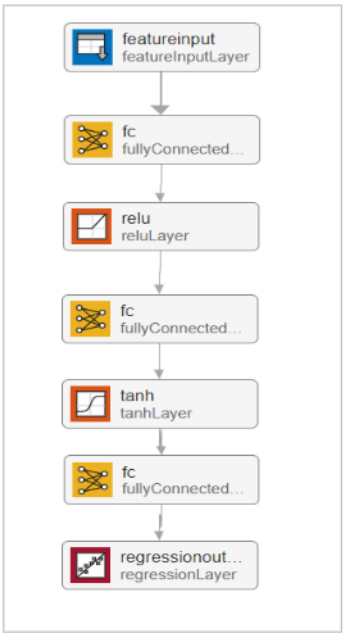


Figure 2.4: Training and Implementation of DPD - Method 1

2.5 Neural Network-1

I used the following Multilayer Feedforward Neural Network to implement DPD for Memoryless Power Amplifier models.




```
7×1 Layer array with layers:

   1   ''   Feature Input       2 features
   2   ''   Fully Connected     200 fully connected layer
   3   ''   ReLU                ReLU
   4   ''   Fully Connected     100 fully connected layer
   5   ''   Tanh                Hyperbolic tangent
   6   ''   Fully Connected     2 fully connected layer
   7   ''   Regression Output   mean-squared-error
```

Figure 2.5: Neural Network-1(MLFFNN)

CHAPTER 3

Memoryless Power Amplifier Models

3.1 Memoryless Polynomial Model

3.1.1 Input and Model Specifications

Input: QPSK Signal

Power Amplifier Model: Memoryless Polynomial model whose output is given by

$$f(x) = x - 0.1x \mid x \mid^2 + 0.005x \mid x \mid^4$$

where x is the input to the PA

Neural-network model for DPD : Neural Network-1(MLFFNN)

Important part of Matlab code to implement DPD for Memoryless Polynomial model

```
%Generating training input and output
[x_train,y_train] = Generate_Data();
%Modeling the MLFFNN Pre-distorter
layers = [ ...
    featureInputLayer(2)
    fullyConnectedLayer(200)
    reluLayer
    fullyConnectedLayer(100)
    tanhLayer
    fullyConnectedLayer(2)
    regressionLayer
    ]
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',75, ...
```

```
        'MiniBatchSize', 100, ...
        'InitialLearnRate', 0.001, ...
        'ExecutionEnvironment',"auto",...
        'plots','training-progress', ...
        'Verbose',false);
%Training the MLFFNN
net = trainNetwork(y_train,x_train,layers,options)
%Generating the test input and output
[x_test,y_test] =  load_data(data_total);
%Getting the output from DPD
classes = predict(net,x_test,'MiniBatchSize',100);
```

3.1.2 Performances

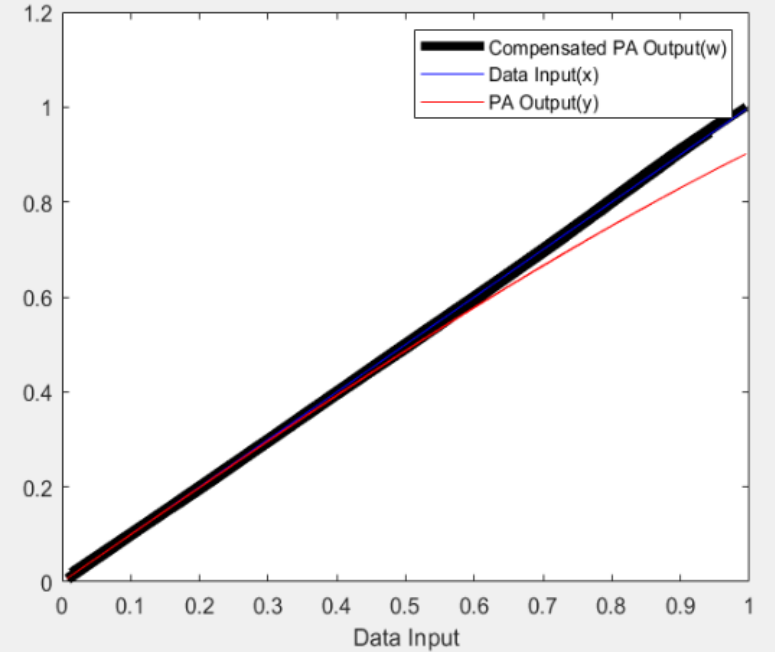


Figure 3.1: Magnitude Plot

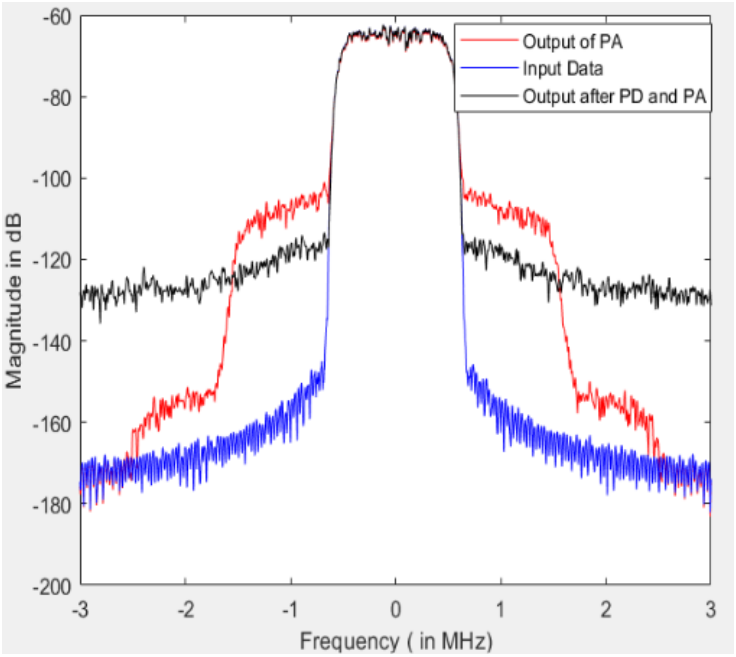


Figure 3.2: Spectral plot

3.1.3 Conclusion

There is around 20dB suppression in the spectral regrowth after the implementation of DPD. The magnitude plot of the compensated output is linear and coincides with the magnitude plot of input as expected. Hence, we have successfully implemented Digital Pre-distortion for Memoryless Polynomial Power Amplifier model.

3.2 Quasi Memoryless Polynomial Model

3.2.1 Input and Model Specifications

Input: QPSK Signal

Power Amplifier Model: Quasi-memoryless Polynomial model whose output is given by

$$f(x) = x - 0.1e^{j\frac{0.05\pi}{4}}x|x|^2 + 0.005e^{j\frac{0.15\pi}{4}}x|x|^4$$

where x is the input to the PA

Neural-network model for DPD : Neural Network-1(MLFFNN)

Important part of Matlab code to implement DPD for Quasi-memoryless model

```
%Generating training input and output
[x_train,y_train] =  Generate_Data();
%Modeling the MLFFNN Pre-distorter
layers = [ ...
    featureInputLayer(2)
    fullyConnectedLayer(200)
    reluLayer
    fullyConnectedLayer(100)
    tanhLayer
    fullyConnectedLayer(2)
    regressionLayer
    ]
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',75, ...
    'MiniBatchSize', 100, ...
    'InitialLearnRate', 0.001, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);
%Training the MLFFNN
net = trainNetwork(y_train,x_train,layers,options)
%Generating the test input and output
[x_test,y_test] =  load_data(data_total);
%Getting the output from DPD
classes = predict(net,x_test,'MiniBatchSize',100);
```

3.2.2 Performances

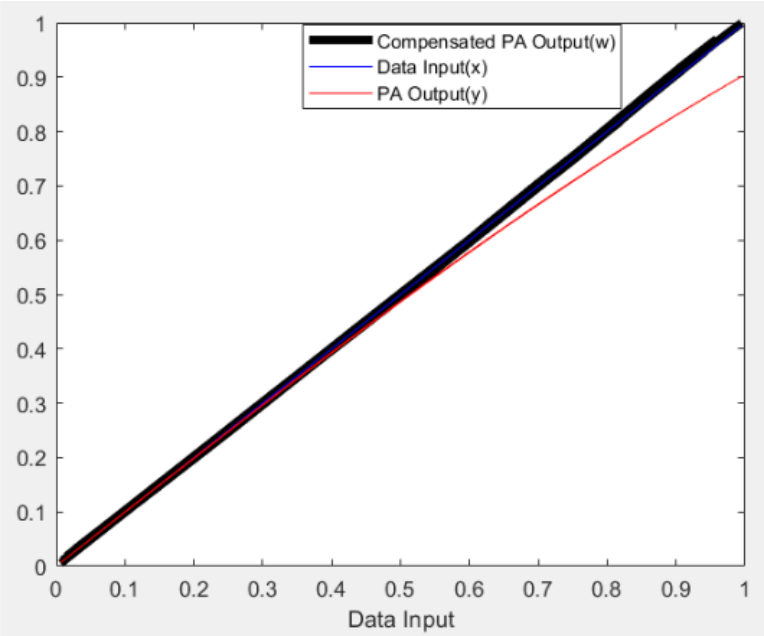


Figure 3.3: Magnitude Plot

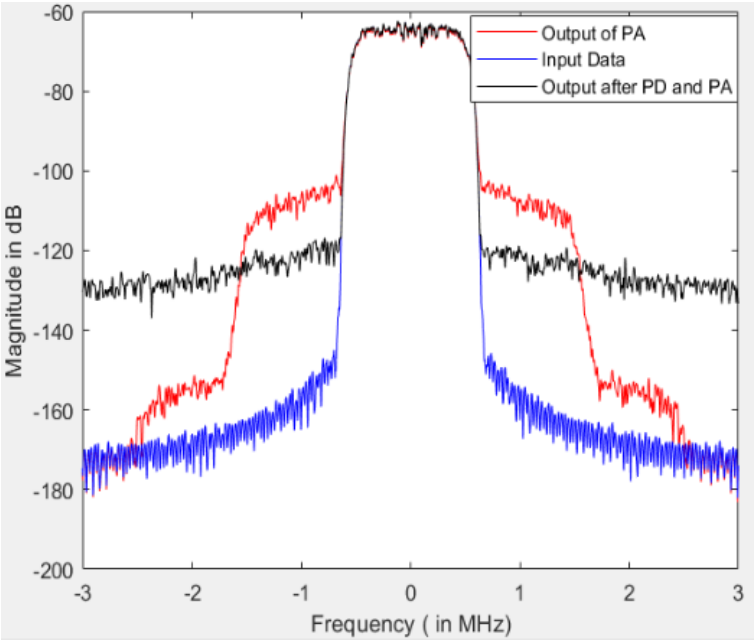


Figure 3.4: Spectral plot

3.2.3 Conclusion

There is greater than 20dB suppression in the spectral regrowth after the implementation of DPD. The magnitude plot of the compensated output is linear and coincides with the

magnitude plot of input as expected. Hence, we have successfully implemented Digital Pre-distortion for Quasi-memoryless Polynomial Power Amplifier model.

CHAPTER 4

Long short-term memory neural networks for DPD

Following the successful implementation of digital pre-distortion for memoryless PA models, the next step is to implement digital pre-distortion for power amplifiers with memory. The same trained MLFFNN cannot be used here as a Pre-distorter because it can only process single points of data, not a sequence of data. So we introduce a lstm layer in the neural network. LSTM units store information over a period of time. In order words, they have a memory capacity. Hence, I have trained the following LSTM-based neural network model and used it as a pre-distorter for power amplifiers with memory in literature as well as the NXP Airfast PA model.

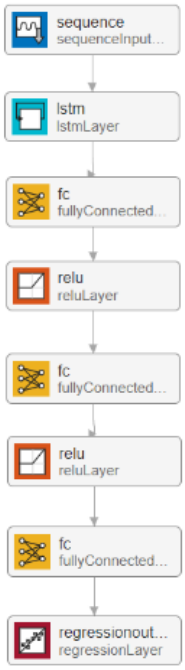
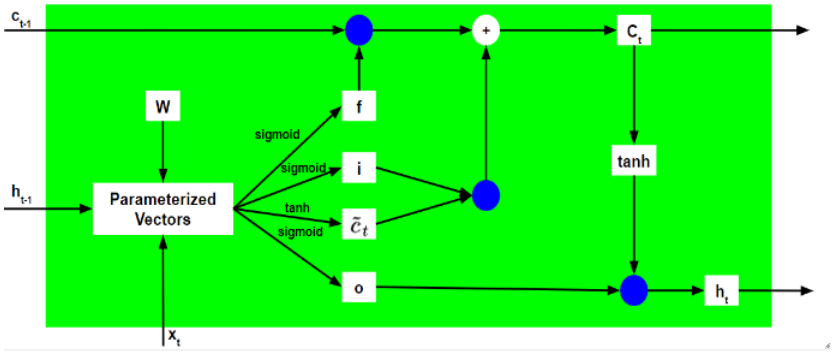


Figure 4.1: LSTM DNN-1

Reference for LSTM DNN-1 : “LSTM-Deep Neural Networks based Predistortion Linearizer for High Power Amplifiers” by Meenakshi Rawat Deepmala Phartiyal.

4.1 LSTM - An Overview

LSTM has a chain structure that contains four neural networks and different memory blocks called cells. Information is retained by the cells and the memory manipulations are done by the gates.



$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \sigma_h(c_t)$$

where the initial values are $c_0 = 0$ and $h_0 = 0$ and o denotes element wise product

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in (0,1)^h$: forget gate's activation vector
- $i_t \in (0,1)^h$: input/update gate's activation vector
- $o_t \in (0,1)^h$: output gate's activation vector
- $h_t \in (-1,1)^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in (-1,1)^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training

where the superscripts d and h refer to the number of input features and number of hidden units, respectively.

Figure 4.2: LSTM unit

A LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. The working of the three gates to produce output sequence from a sequence of input data is explained as follows:

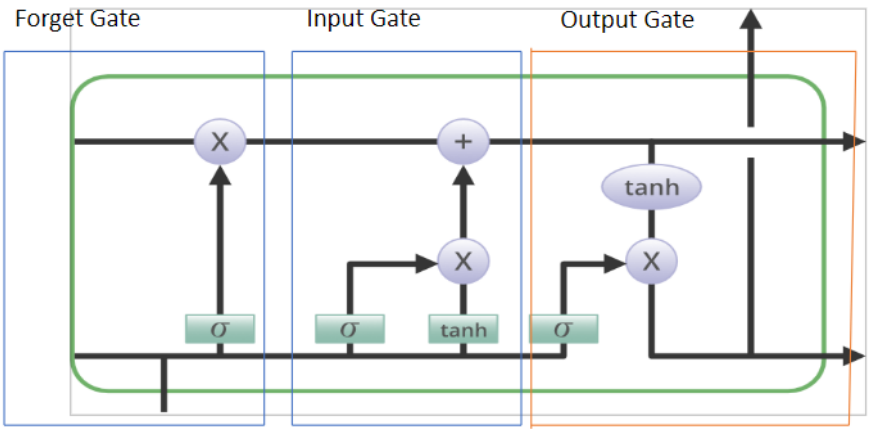


Figure 4.3: LSTM Structure

Forget Gate: The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_t(input at the particular time) and h_{t-1}(previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

Input gate: The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h_{t-1} and x_t. Then, a vector is created using tanh function that gives an output from -1 to +1, which contains all the possible values from h_{t-1} and x_t. At last, the values of the vector and the regulated values are multiplied to obtain the useful information.

Output gate: The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs h_{t-1} and x_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

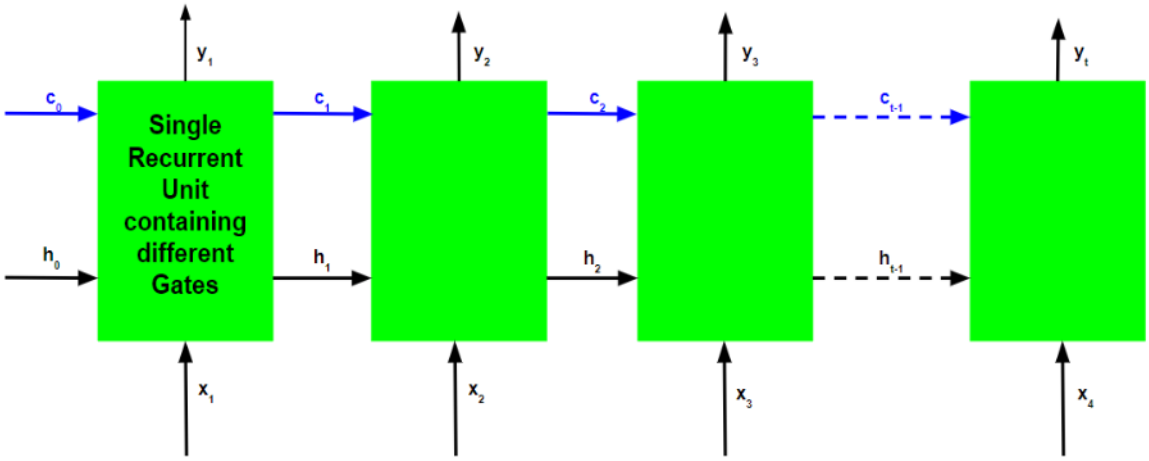


Figure 4.4: Working of LSTM units

CHAPTER 5

Power Amplifiers with memory

Using the AM/AM and AM/PM plots, I investigated different Memory power amplifier models in the literature(Reference for literature: Thesis of Ding Lei on "Digital Predistortion of Power Amplifiers for Wireless Applications"). From this literature, PA models like the Wiener model, Hammerstein model, Perturbed wiener model, and Wiener-Hammerstein model were ignored as their AM/AM plots were not linear and showed memory effects even for low input power. The AM/AM plots of Memory polynomial and Parallel wiener models were as desired, so I decided to implement DPD for these two models.

5.1 Memory Polynomial Model

5.1.1 Input and Model Specifications

Input: QPSK Signal

Power Amplifier Model: Memory Polynomial model whose coefficient matrix is given by

$$C = \begin{bmatrix} 1.0513 + 0.0904j & 0 & -0.0542 - 0.2900j & 0 & -0.0542 - 0.2900j \\ -0.0680 - 0.0023j & 0 & 0.2234 + 0.2317j & 0 & -0.2451 - 0.3735j \\ 0.0289 - 0.0054j & 0 & -0.0621 - 0.0932j & 0 & 0.1229 + 0.1508j \end{bmatrix}$$

Neural-network model for DPD : LSTM-DNN 1

Important part of Matlab code to implement DPD for Memory Polynomial model

```
%Generating training input and output
[x_train,y_train] = Generate_Data(data_total);
%Normalizing the training output
gamma = sqrt(sum(abs(Convert2Complex(y_train)).^2)/
sum(abs(Convert2Complex(x_train)).^2));
```

```
gamma_phase = exp(-1*1j*atan(dot(Convert2Complex(x_train),
conj(Convert2Complex(y_train)))));
y_train=Convert2Feature((Convert2Complex(y_train)*gamma_phase)/gamma
%Modeling the LSTM based Neural network
layers = [ ...
    sequenceInputLayer(2)
    lstmLayer(12,'OutputMode','sequence')
    fullyConnectedLayer(7)
    reluLayer
    fullyConnectedLayer(5)
    reluLayer
    fullyConnectedLayer(2)
    regressionLayer];
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',6001, ...
    'MiniBatchSize', 100, ...
    'InitialLearnRate', 0.005, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);
%Training the LSTM-DNN
net = trainNetwork(y_train',x_train',layers,options)
%Generating the test input and output
[x_test,y_test] =  load_data(data_total);
%Normalizing the test output(Max normalization)
delta = max(abs(Convert2Complex(y_test)));
y_test = (y_test)/delta;
%Getting the output from DPD
classes = predict(net,x_test','MiniBatchSize',100);
classes = classes';
%Getting PA output after compensation
output = Convert2Feature(pa(Convert2Complex(classes)));
```

```
%Normalizing the compensated output
beta = sqrt(sum(Convert2Complex(output).^2)/
sum(abs(Convert2Complex(x_test)).^2));
beta_phase = exp(-1*1j*atan(dot(Convert2Complex(x_test),
conj(Convert2Complex(output)))));
output = Convert2Feature((Convert2Complex(output)*beta_phase)/beta);
```

5.1.2 Performances

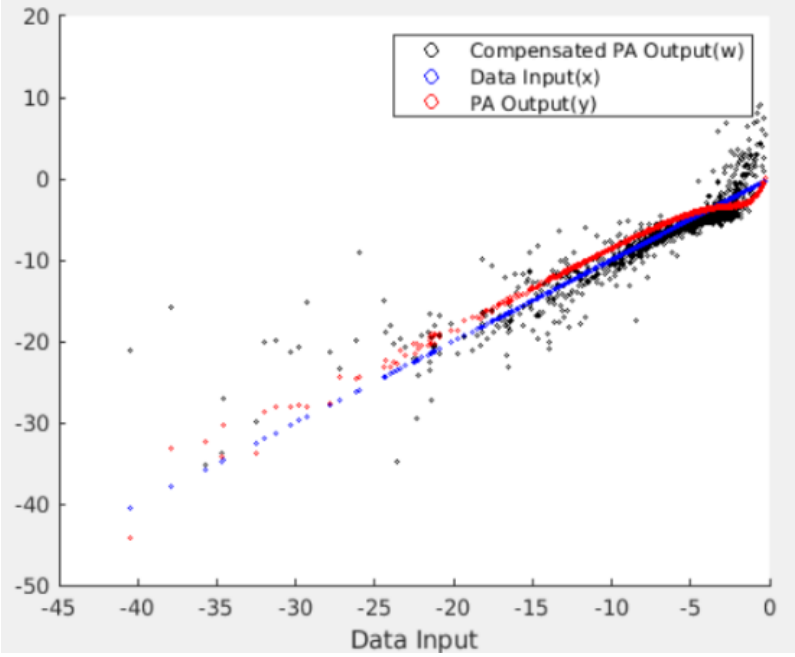


Figure 5.1: AM/AM Plot

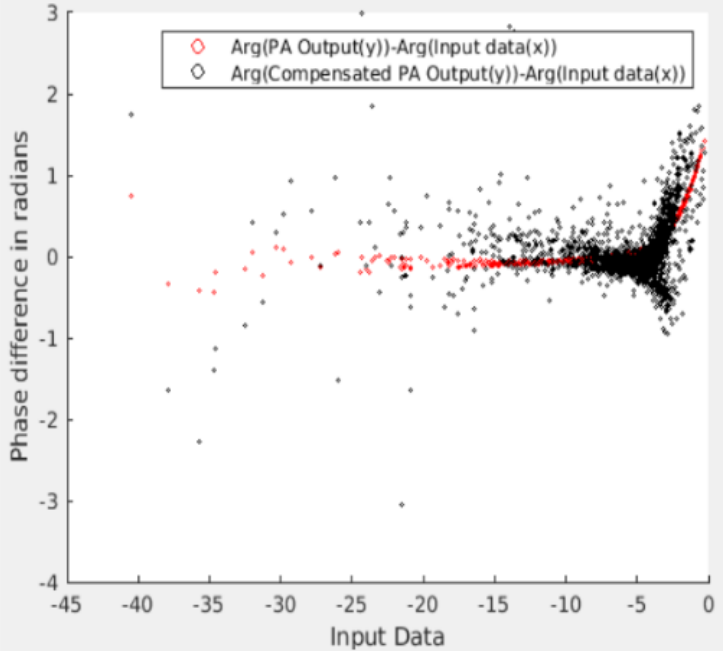


Figure 5.2: AM/PM Plot

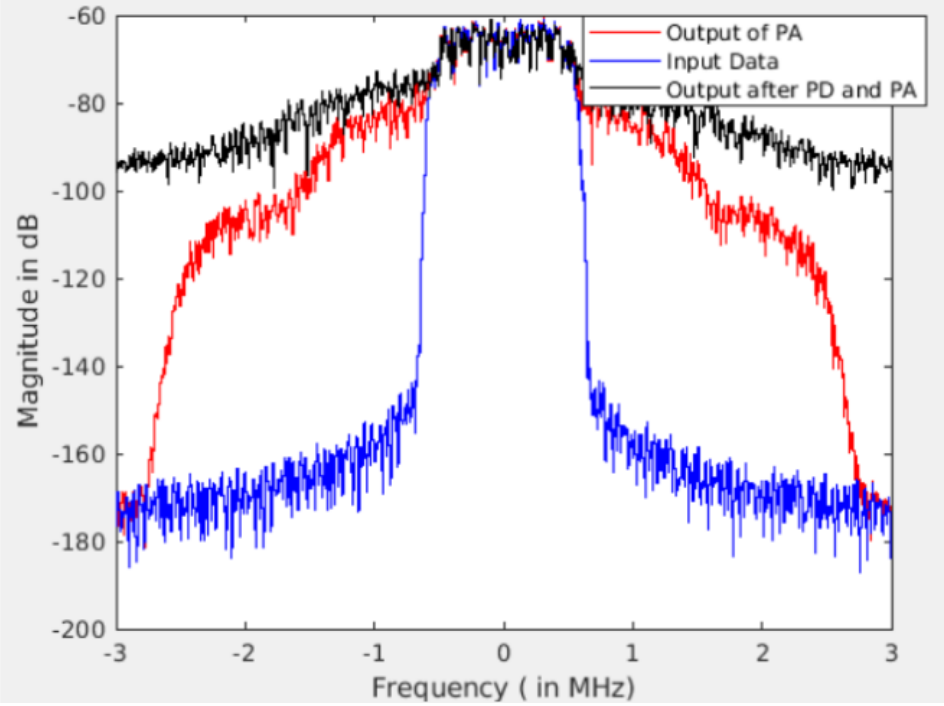


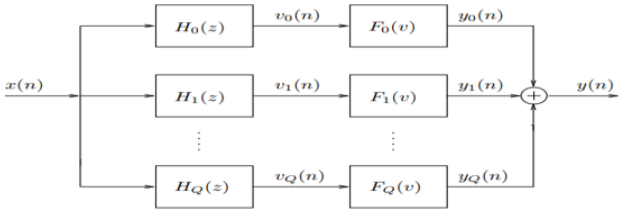
Figure 5.3: Spectral plot

5.2 Parallel wiener Model

5.2.1 Input and Model Specifications

Input: QPSK Signal

Power Amplifier Model: Parallel wiener model



The transfer functions of the LTI systems in the model are

$$H_1(z) = 1,$$
$$H_2(z) = \frac{1 + 0.3z^{-1}}{1 - 0.1z^{-1}},$$
$$H_3(z) = \frac{1 - 0.2z^{-1}}{1 - 0.4z^{-1}}.$$

The coefficient matrices of the Memoryless polynomial systems in the model are

[0.5054+0.0429j 0 0.04395-0.07915j 0 -0.5496-0.44455j]

[0.05895+0.0002j 0 -0.0909+0.01955j 0 0.0842+0.0017j]

[0.02365-0.0029j 0 0.01975+0.01415j 0 -0.05075-0.0098j]

Neural-network model for DPD : LSTM-DNN 1

Important part of Matlab code to implement DPD for Parallel wiener model

```
%Generating training input and output
[x_train,y_train] = Generate_Data(data_total);
%Normalizing the training output
gamma = sqrt(sum(abs(Convert2Complex(y_train)).^2)/
sum(abs(Convert2Complex(x_train)).^2));
gamma_phase = exp(-1*1j*atan(dot(Convert2Complex(x_train),
```

```
conj(Convert2Complex(y_train)))));
y_train=Convert2Feature((Convert2Complex(y_train)*gamma_phase)/gamma
%Modeling the LSTM based Neural network
layers = [ ...
    sequenceInputLayer(2)
    lstmLayer(12,'OutputMode','sequence')
    fullyConnectedLayer(7)
    reluLayer
    fullyConnectedLayer(5)
    reluLayer
    fullyConnectedLayer(2)
    regressionLayer];
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',6001, ...
    'MiniBatchSize', 100, ...
    'InitialLearnRate', 0.005, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);
%Training the LSTM-DNN
net = trainNetwork(y_train',x_train',layers,options)
%Generating the test input and output
[x_test,y_test] =  load_data(data_total);
%Normalizing the test output(Max normalization)
delta = max(abs(Convert2Complex(y_test)));
y_test = (y_test)/delta;
%Getting the output from DPD
classes = predict(net,x_test',...
'MiniBatchSize',100);
classes = classes';
%Getting PA output after compensation
output = Convert2Feature(pa(Convert2Complex(classes)));
%Normalizing the compensated output
```

```
beta = sqrt(sum(Convert2Complex(output).^2)/
sum(abs(Convert2Complex(x_test)).^2));
beta_phase = exp(-1*1j*atan(dot(Convert2Complex(x_test),
conj(Convert2Complex(output)))));
output = Convert2Feature((Convert2Complex(output)*beta_phase)/beta);
```

5.2.2 Performances

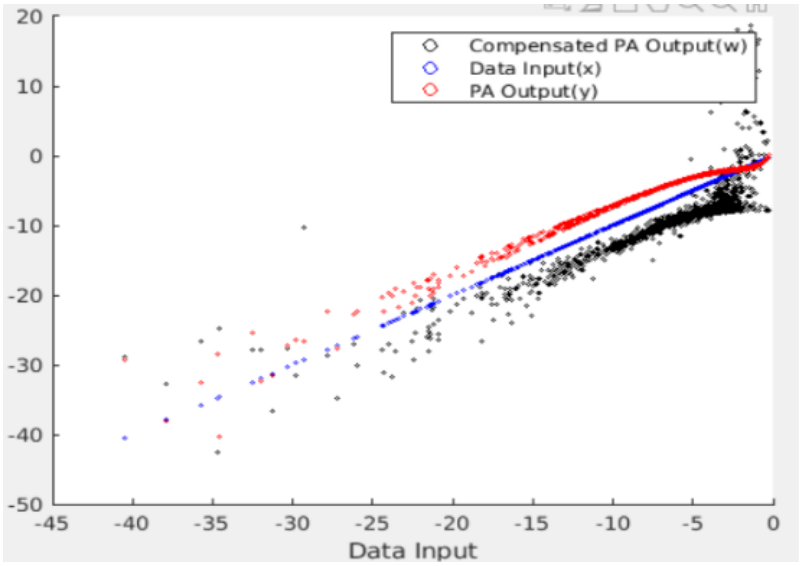


Figure 5.4: AM/AM Plot

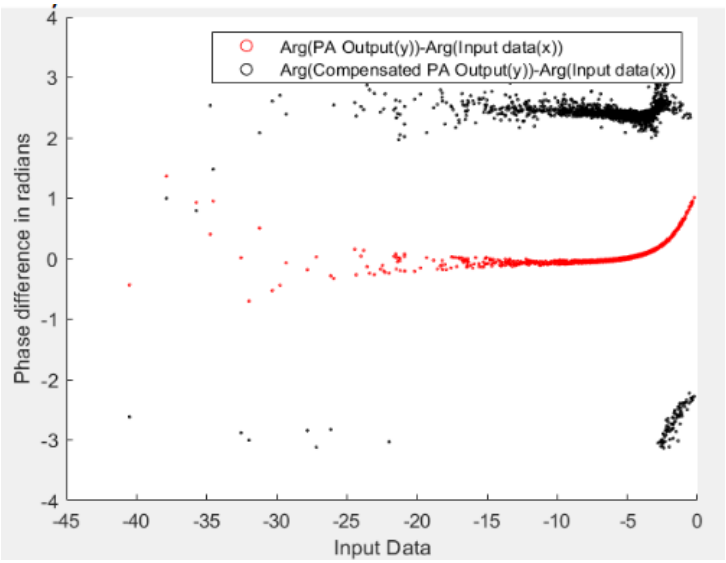


Figure 5.5: AM/PM Plot

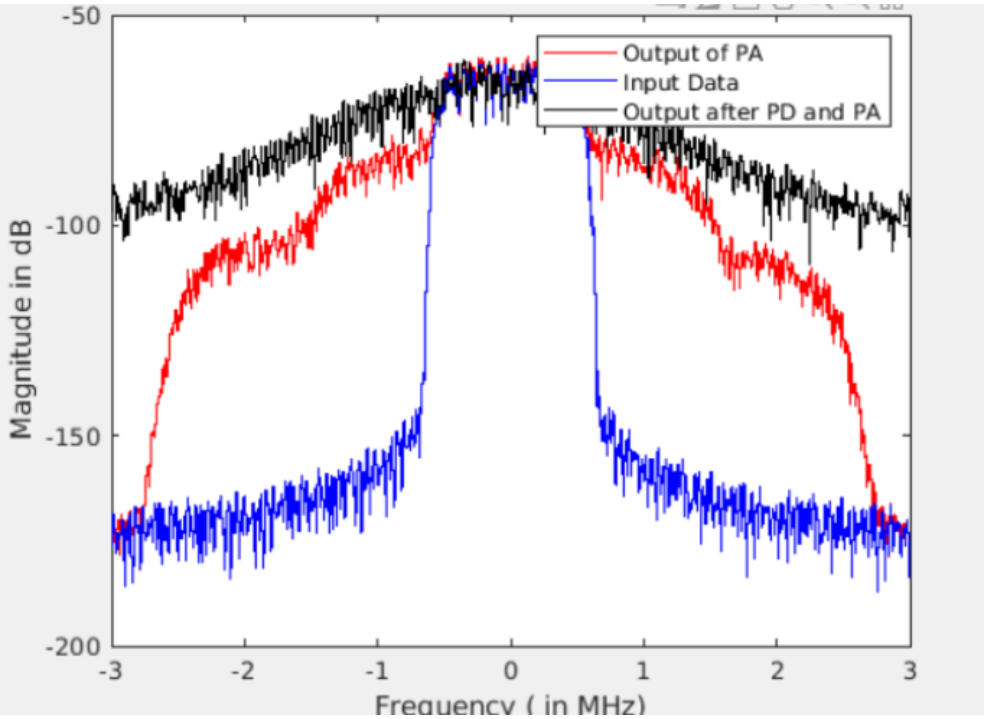


Figure 5.6: Spectral plot

5.3 Conclusion for both the models

The spectral regrowth is actually worse after implementing DPD. The AM/AM plot of the compensated output is non-linear for high input power and doesn't coincide with the AM/AM plot of input. The AM/PM plot of the compensated output is not at all close to zero for all input.

We can see that implementing DPD to increase the working range of these two PA models to entire non-saturating region is actually worsening the performance. So I decided to extend the working range till the 1-dB compression points of the models.

5.4 1 dB compression points

In the plot below, Uniformly distributed data is used as input to the Memory Polynomial power amplifier. I fitted a quadratic polynomial curve for PA output corresponding to input data in power range(-5.5 dB,-1.5dB). Using the the fitted curve I got the 1 dB(-3.72 dB) and 2 dB(-2.4 dB)compression points.

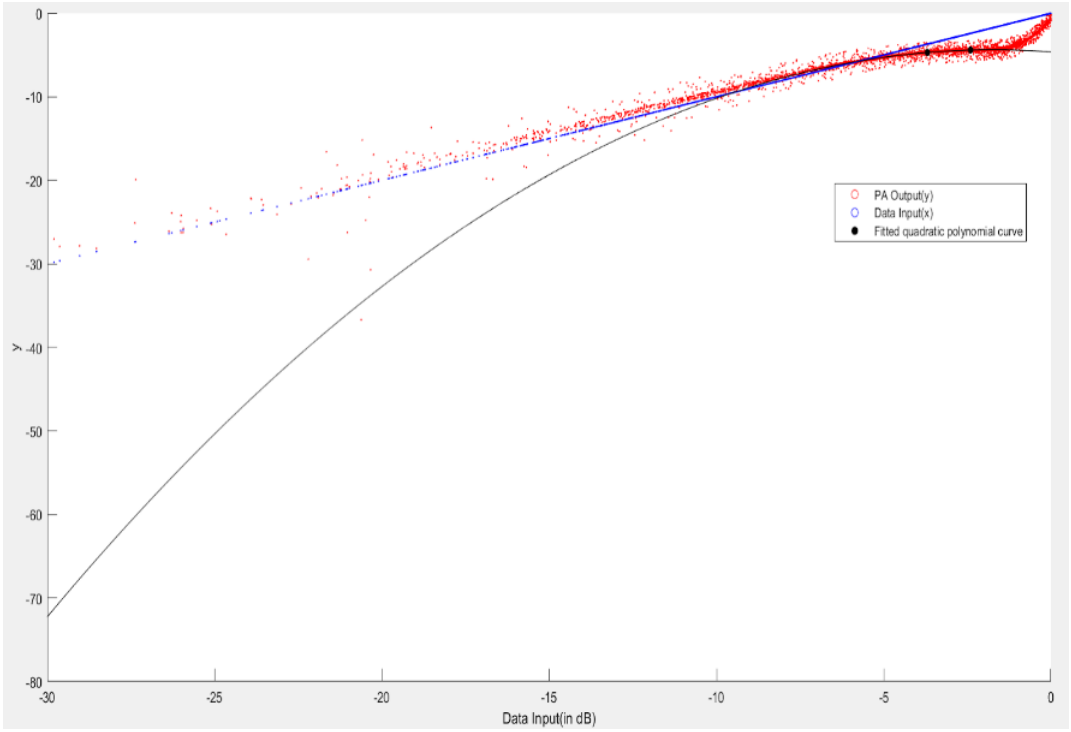


Figure 5.7: 1-dB and 2-dB compression points of Memory Polynomial model

In the plot below, Uniformly distributed data is used as input to the Parallel wiener power amplifier. I fitted a quadratic polynomial curve for PA output corresponding to input data in power range (-4dB,-1dB). Using the the fitted curve I got the 1 dB(-2.31 dB) and 2 dB(-1.17 dB)compression points.

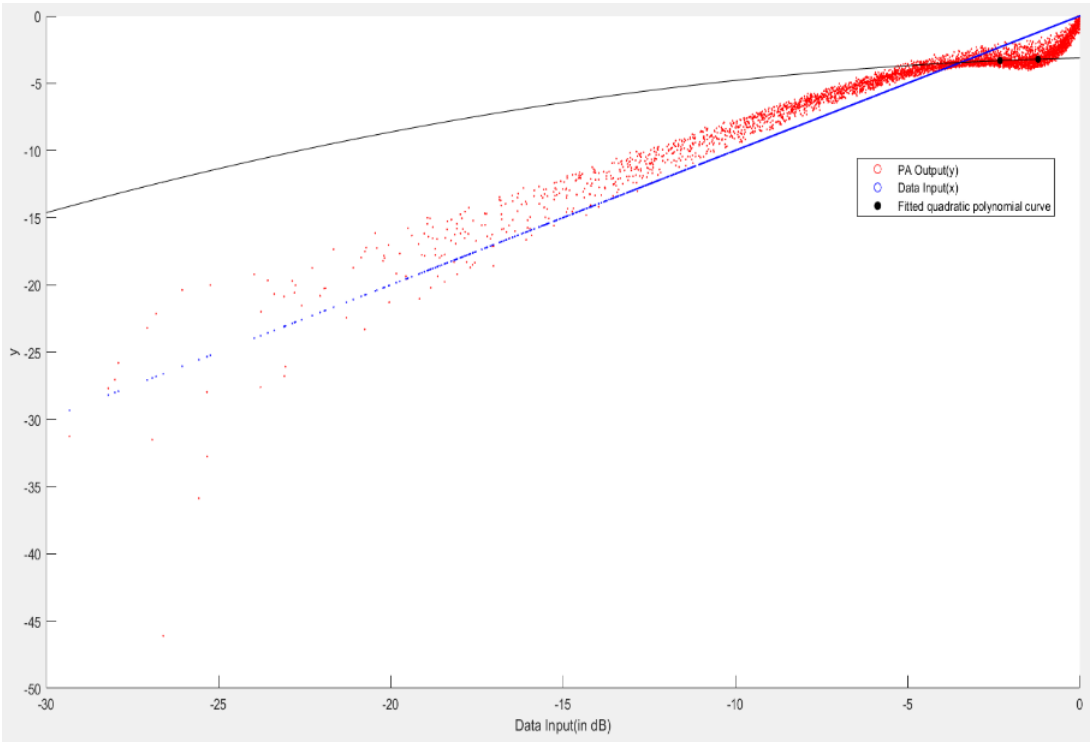


Figure 5.8: 1-dB and 2-dB compression points of Parallel wiener model

After finding the 1-dB compression point, we scale the input in such a way that 99% of the input points fall below the 1-dB compression point.

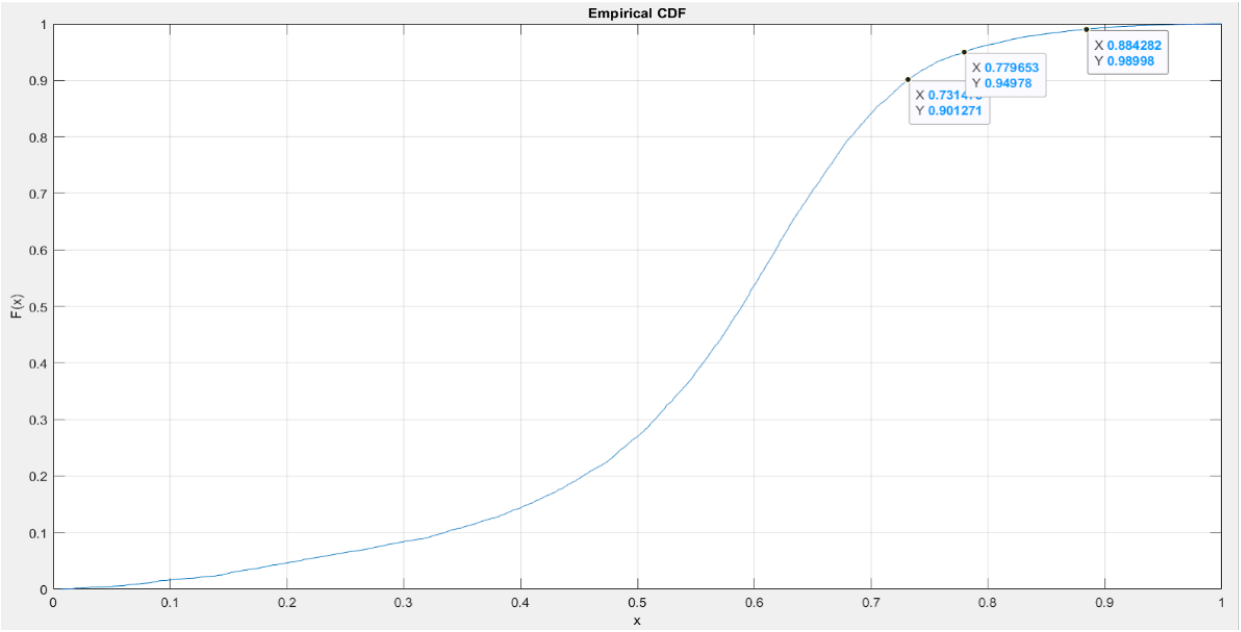


Figure 5.9: CDF plot of the Memory Polynomial PA input before scaling

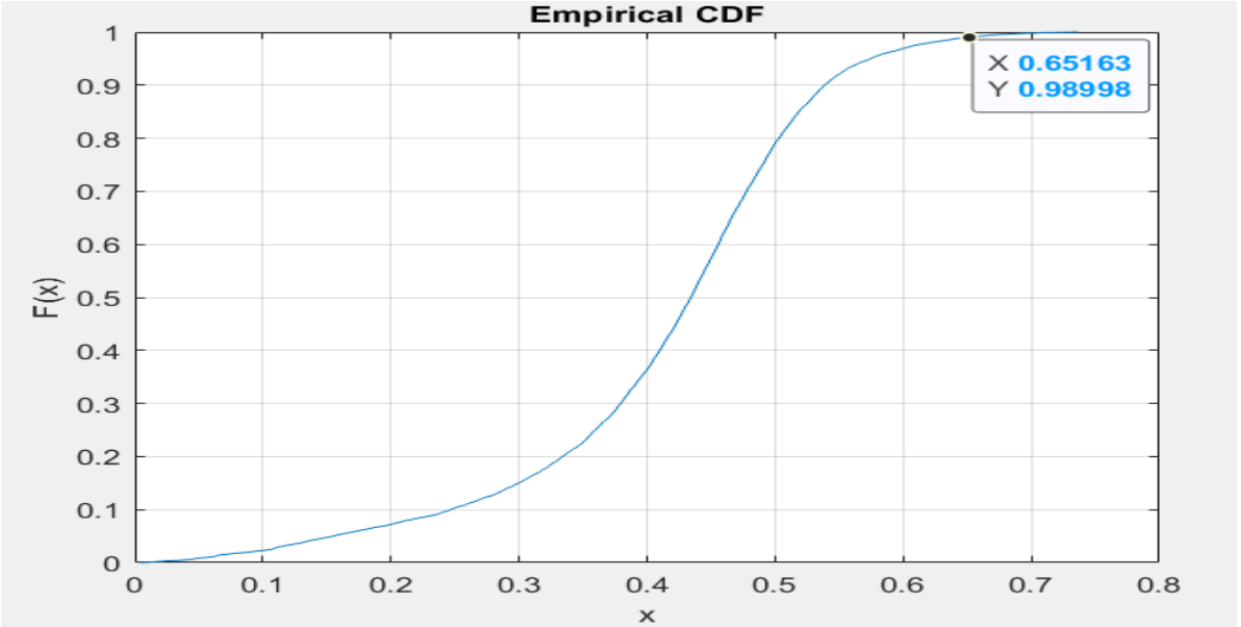


Figure 5.10: CDF plot of the Memory Polynomial PA input after scaling

5.5 Final Performances

Important part of Matlab Code to implement DPD for Power Amplifiers in literature(extending working range to 1-dB compression point):

```
%Generating the training input and output
[x_train,y_train] =  Generate_Data(data_total);
%Normalizing the training output(Max normalization)
gamma = max(abs(Convert2Complex(y_train)));
y_train = (y_train)/gamma;
%Modeling the LSTM based Neural network
layers = [ ...
    sequenceInputLayer(2)
    lstmLayer(12,'OutputMode','sequence')
    fullyConnectedLayer(7)
    reluLayer
    fullyConnectedLayer(5)
    reluLayer
    fullyConnectedLayer(2)
    regressionLayer];
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',6001, ...
    'MiniBatchSize', 100, ...
    'InitialLearnRate', 0.005, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);
%Training the LSTM-DNN
net = trainNetwork(y_train',x_train',layers,options)
%Generating the test input and output
[x_test,y_test] =  load_data(data_total);
%Normalizing the test output(Max normalization)
delta = max(abs(Convert2Complex(y_test)));
y_test = (y_test)/delta;
%Getting the output from DPD
classes = predict(net,x_test','MiniBatchSize',100);
classes = classes';
```

```
%Getting PA output after compensation
output = Convert2Feature(pa(Convert2Complex(classes)));
%Normalizing the compensated output(Max normalization)
beta = max(abs(Convert2Complex(output)));
output = (output)/beta;
```

5.5.1 Memory Polynomial

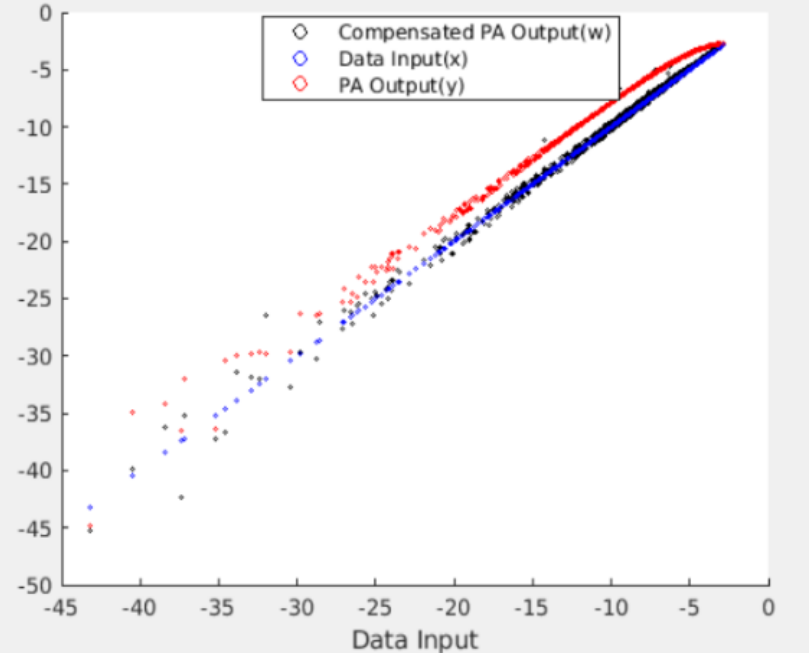


Figure 5.11: AM/AM Plot

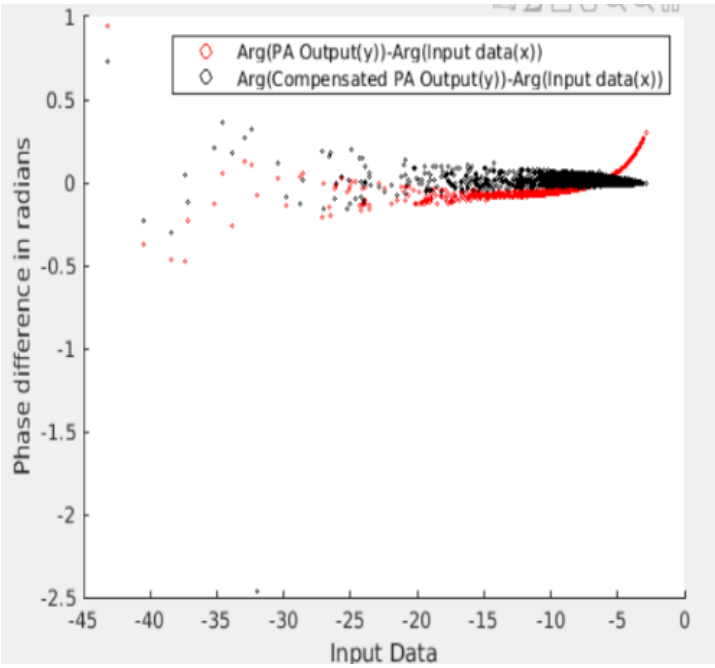


Figure 5.12: AM/PM Plot

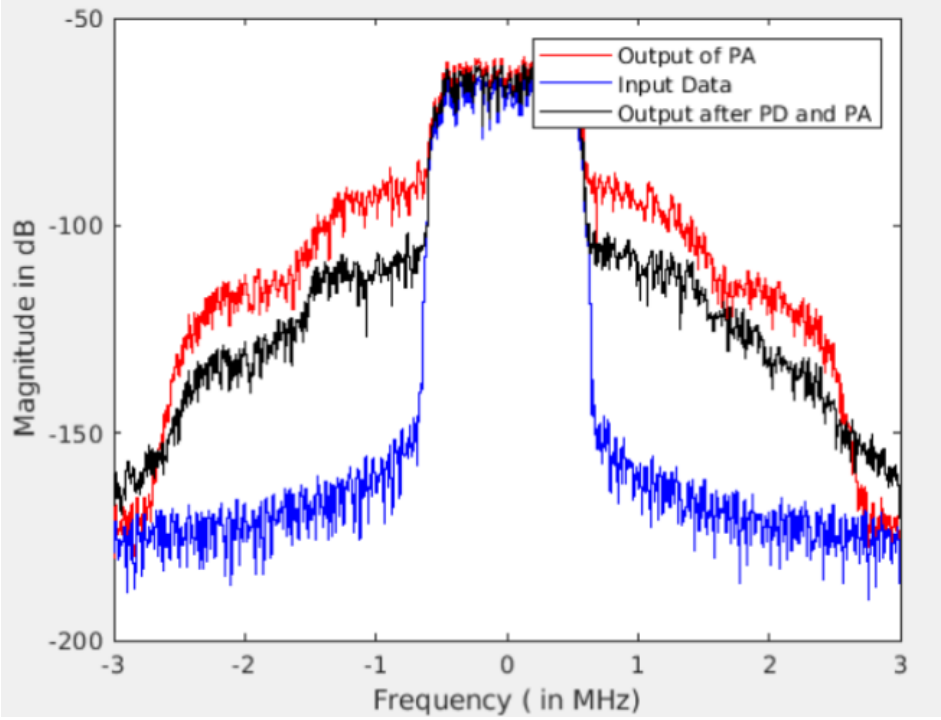


Figure 5.13: Spectral plot

5.5.2 Parallel wiener

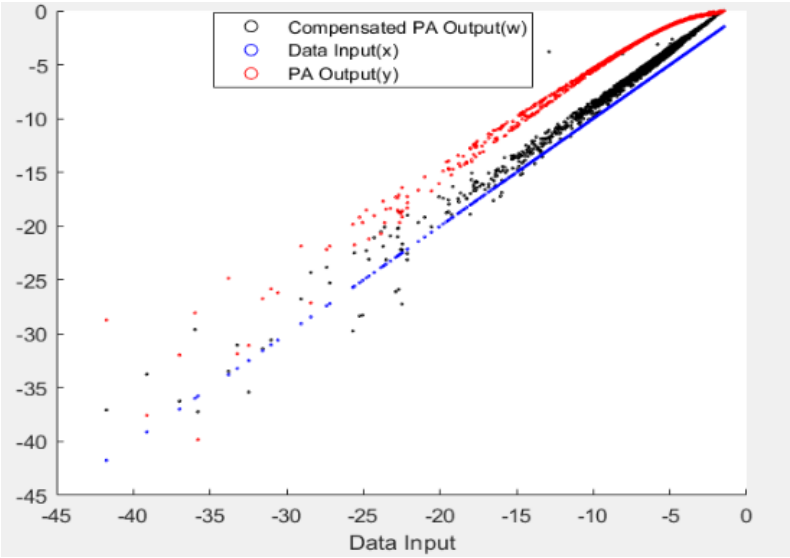


Figure 5.14: AM/AM Plot

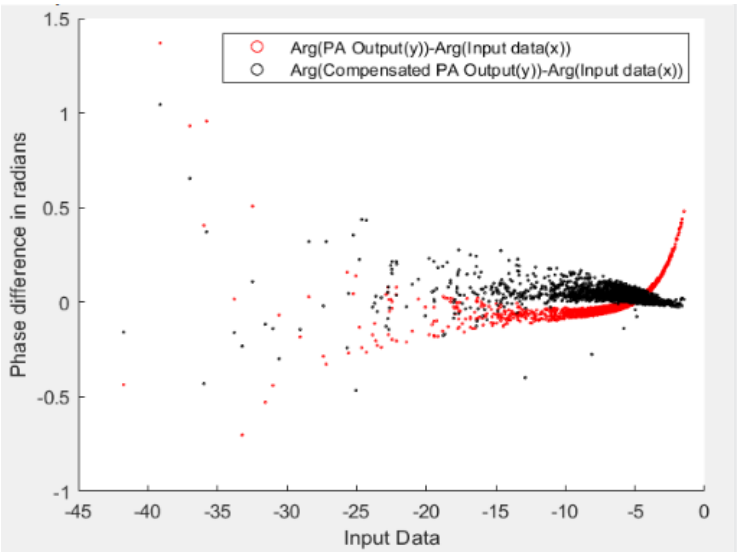


Figure 5.15: AM/PM Plot

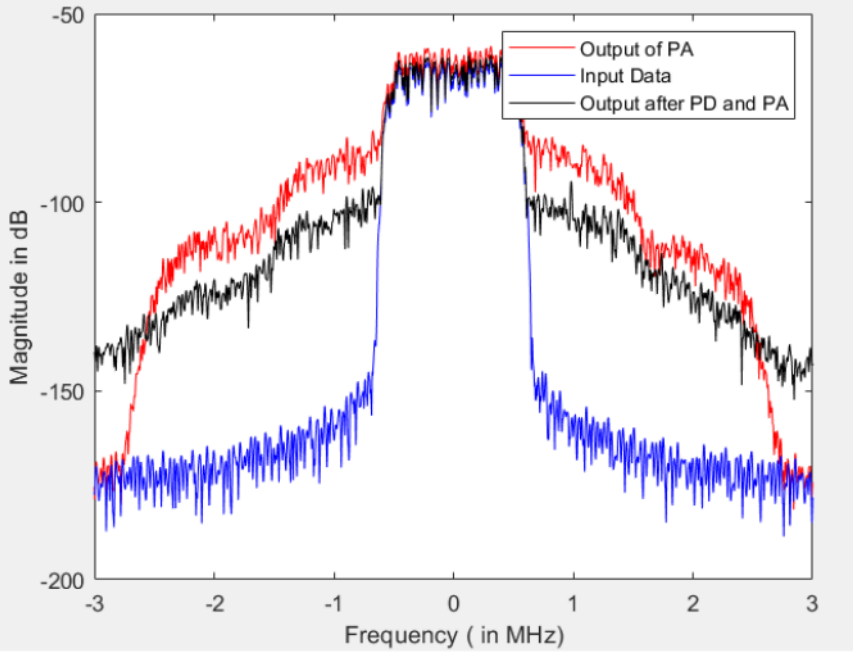


Figure 5.16: Spectral plot

5.5.3 Conclusion

After restricting the input power below 1dB compression point, good suppression in spectral regrowth is achieved after implementing DPD. The AM/AM plot of the compensated output is linear and coincides with the AM/AM plot of input. The AM/PM plot of the compensated output is close to zero for all input as expected. Hence we have successfully implemented DPD and extended the working input power range of Memory polynomial and Parallel wiener models.

CHAPTER 6

Real world Power Amplifier

Due to practical difficulties, I couldn't work with real power amplifier hardware. So I used the measured input and output data of the NXP Airfast Power Amplifier from Matlab central(https://in.mathworks.com/help/comm/ug/power-amplifier-characterization.html) to fit a memory polynomial power amplifier model with memory length 2 and degree 5. The coefficient matrix of the fitted PA is

```
21.6584+3.2644i  10.9776+2.8696i  -21.0558-17.5067i  14.8153+13.0797i  -3.418-2.9818i
-0.9117+2.9878i   4.5515-6.4774i   -8.3123+3.8890i   5.2385+1.0406i   -1.0285-0.6839i
```

Once we validate this model,we can use this model as Airfast PA and implement DPD.

6.1 Validating the model

I simulated the Memory Polynomial PA with the derived coefficient matrix using a matlab function. It is necessary to validate the matlab function model. I used ofdm signal as input to the PAs and got the AM/AM,AM/PM and spectral plots of actual NXP Airfast PA output and matlab function output.

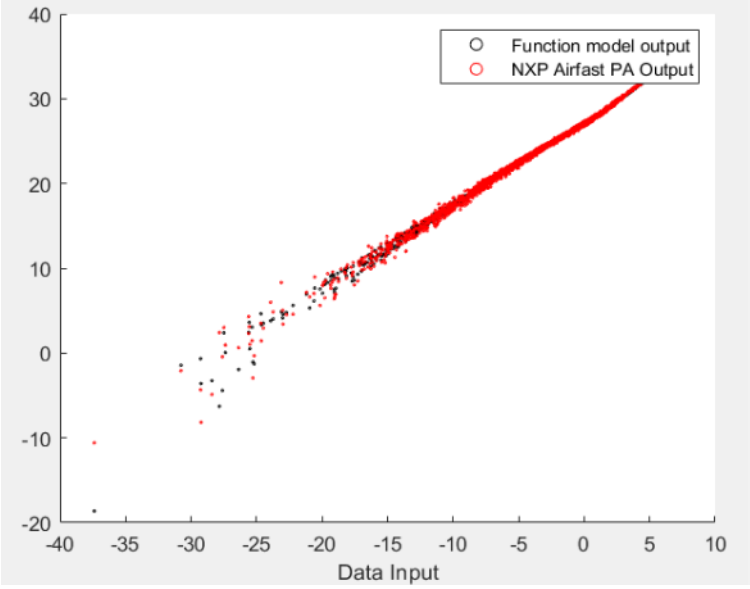


Figure 6.1: AM/AM Plot

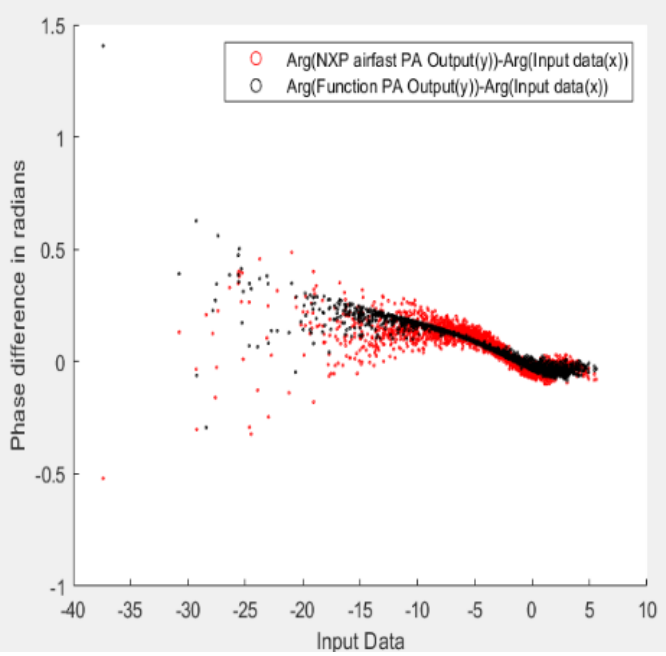


Figure 6.2: AM/PM plot

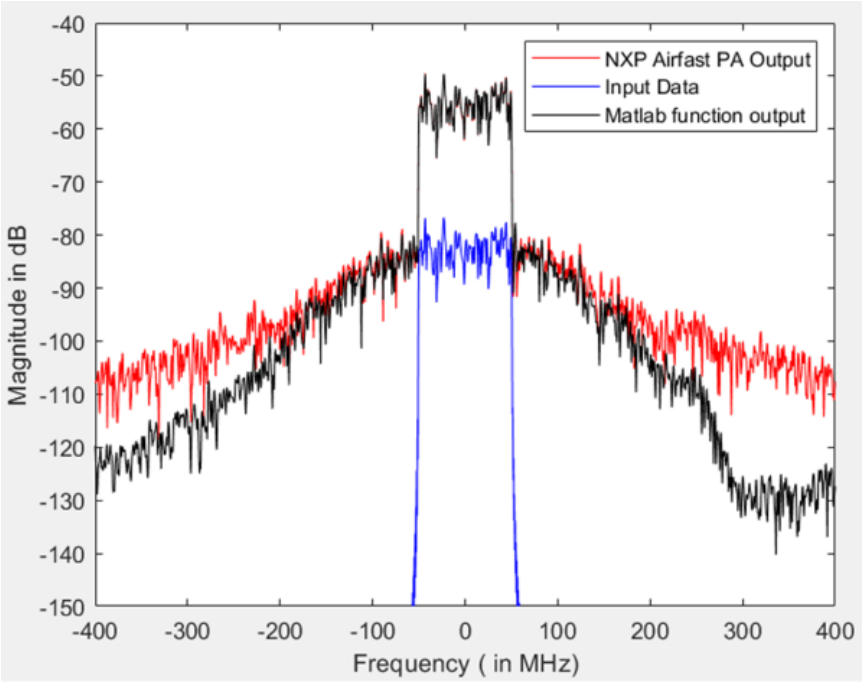


Figure 6.3: Spectral plot

We can observe that the plots of Memory Polynomial PA output and NXP Airfast PA output are very similar. Hence we proceed to implement DPD for this model.

6.1.1 Input and Model Specifications

Input: OFDM Signal

Power Amplifier Model: NXP Airfast PA modeled as Memory Polynomial model with coefficient matrix

```
21.6584+3.2644i  10.9776+2.8696i  -21.0558-17.5067i  14.8153+13.0797i  -3.418-2.9818i
-0.9117+2.9878i   4.5515-6.4774i   -8.3123+3.8890i   5.2385+1.0406i   -1.0285-0.6839i
```

Neural-network model for DPD : LSTM-DNN 1

Important part of Matlab Code to implement DPD for Modeled NXP Airfast PA(using first method of training the neural network)

```
%Generating the training input and output
[x_train,y_train] =  Generate_Data(data_total);
%Modeling the LSTM based Neural network
layers = [ ...
    sequenceInputLayer(2)
    lstmLayer(12,'OutputMode','sequence')
    fullyConnectedLayer(7)
    reluLayer
    fullyConnectedLayer(5)
    reluLayer
    fullyConnectedLayer(2)
    regressionLayer];
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',8000, ...
    'MiniBatchSize', 100, ...
    'InitialLearnRate', 0.002, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);
%Training the LSTM-DNN
net = trainNetwork(y_train',x_train',layers,options)
```

```
%Generating the test input and output
[x_test,y_test] =  load_data(data_total1);
%Normalizing the test output
delta = sqrt(sum(abs(Convert2Complex(y_test)).^2)/
sum(abs(Convert2Complex(x_test)).^2));
delta_phase = exp(-1*1j*atan(dot(Convert2Complex(x_test),
conj(Convert2Complex(y_test)))));
y_test=Convert2Feature((Convert2Complex(y_test)*delta_phase)/delta);
%Getting the output from DPD
classes = predict(net,x_test','MiniBatchSize',100);
output = classes';
%Getting PA output after compensation
output = pa(output);
```

6.1.2 Performance

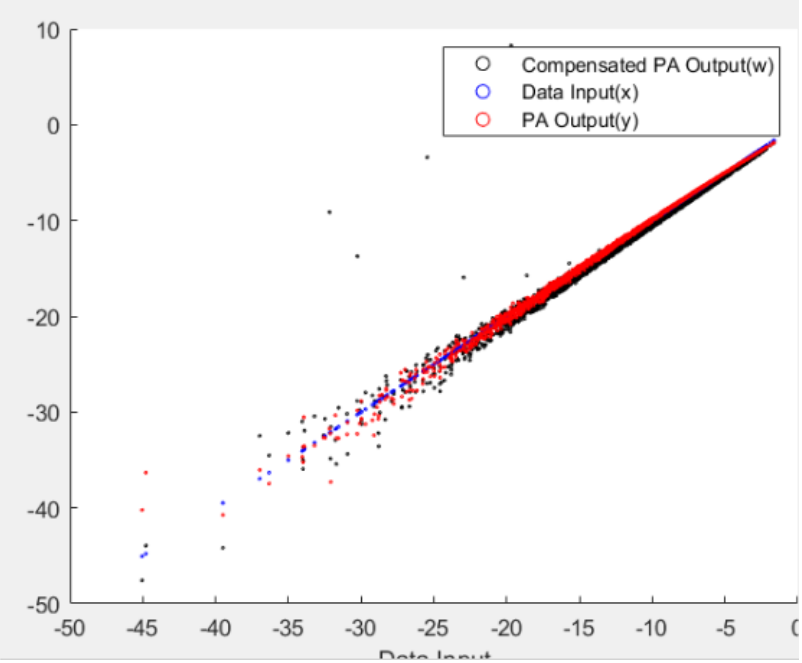


Figure 6.4: AM/AM Plot

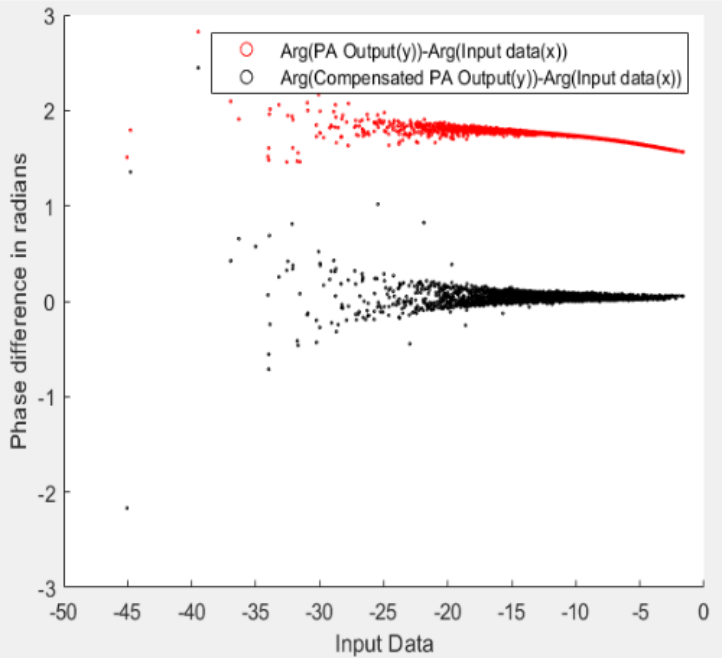


Figure 6.5: AM/PM plot

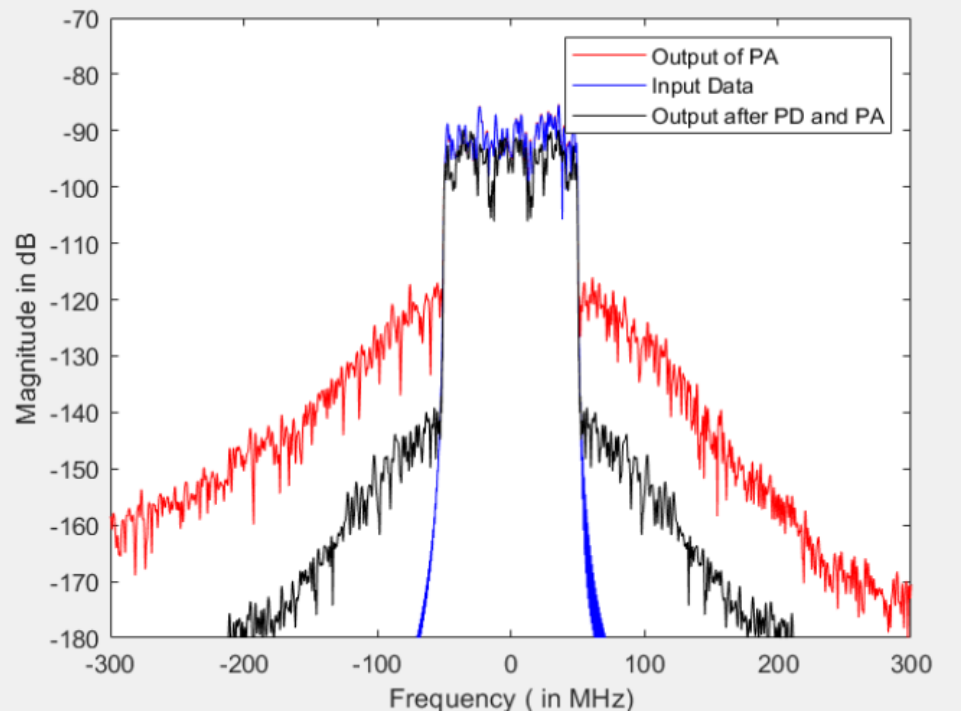


Figure 6.6: Spectral plot

6.1.3 Conclusion

DPD appears to be very effective in suppressing spectral regrowth. The AM/AM plot of compensated output is linear and matches the AM/AM plot of input as expected. For all inputs, the AM/PM plot of the compensated output is near zero.

6.2 Another way of training the neural network

I also tried closed-loop method of training the neural network. In this technique, we train the neural network in such a way that the training output of the cascaded system of the neural network and the power amplifier is same as training input signal. After training the neural network, we send the test input to cascaded system of trained neural network and PA and get the compensated output. We use AM/AM, AM/PM and spectral plots to compare the output before and after implementing DPD.

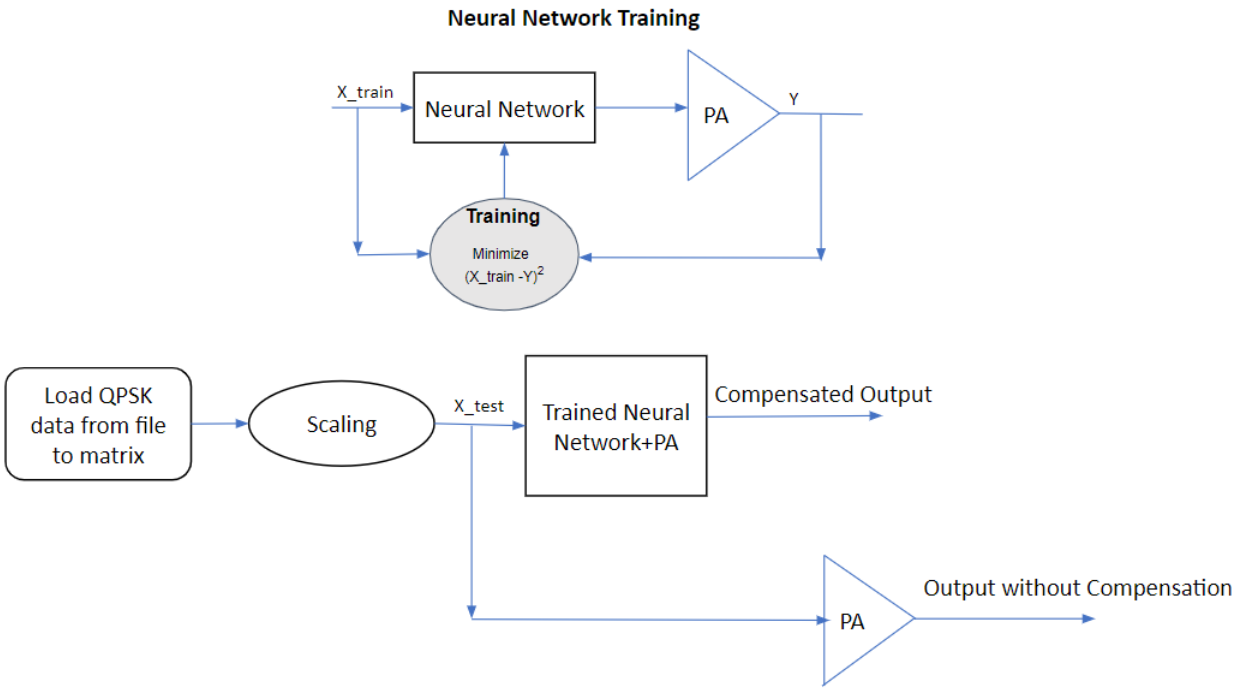


Figure 6.7: LSTM-DNN 2

I implemented DPD for NXP Airfast PA using this method to analyse the performance of this technique.

6.2.1 Input and Model Specifications

Input: OFDM Signal

Power Amplifier Model: NXP Airfast PA modeled as Memory Polynomial model with coefficient matrix

```
21.6584+3.2644i  10.9776+2.8696i  -21.0558-17.5067i  14.8153+13.0797i  -3.418-2.9818i
-0.9117+2.9878i   4.5515-6.4774i   -8.3123+3.8890i   5.2385+1.0406i   -1.0285-0.6839i
```

Neural-network model for DPD : LSTM-DNN 2

Important part of Matlab Code to implement DPD for Modeled NXP Airfast PA(using second method of training the neural network)

```
%Generating the training input and output
[x_train,y_train] =  Generate_Data(data_total);
%Modeling the LSTM based Neural network+Power Amplifier system
layers = [ ...
    sequenceInputLayer(2)
    lstmLayer(12,'OutputMode','sequence')
    fullyConnectedLayer(7)
    tanhLayer
    fullyConnectedLayer(5)
    tanhLayer
    fullyConnectedLayer(2)
    tanhLayer
    custLayer(1,"new_layer")
    regressionLayer];
%Setting the hyperparameters
options = trainingOptions('adam', ...
    'MaxEpochs',1400, ...
    'MiniBatchSize', 100, ...
    'InitialLearnRate', 0.004, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);
%Training the LSTM-DNN
net = trainNetwork(x_train',y_train',layers,options)
%Generating the test input and output
[x_test,y_test] =  load_data(data_total1);
%Normalizing the test output
delta = sqrt(sum(abs(Convert2Complex(y_test)).^2)/
sum(abs(Convert2Complex(x_test)).^2));
delta_phase = exp(-1*1j*atan(dot(Convert2Complex(x_test),
```

```
conj(Convert2Complex(y_test)))));
y_test=Convert2Feature((Convert2Complex(y_test)*delta_phase)/delta);
%Getting PA output after compensation
classes = predict(net,x_test','MiniBatchSize',100);
output = classes';
```

6.2.2 Performance

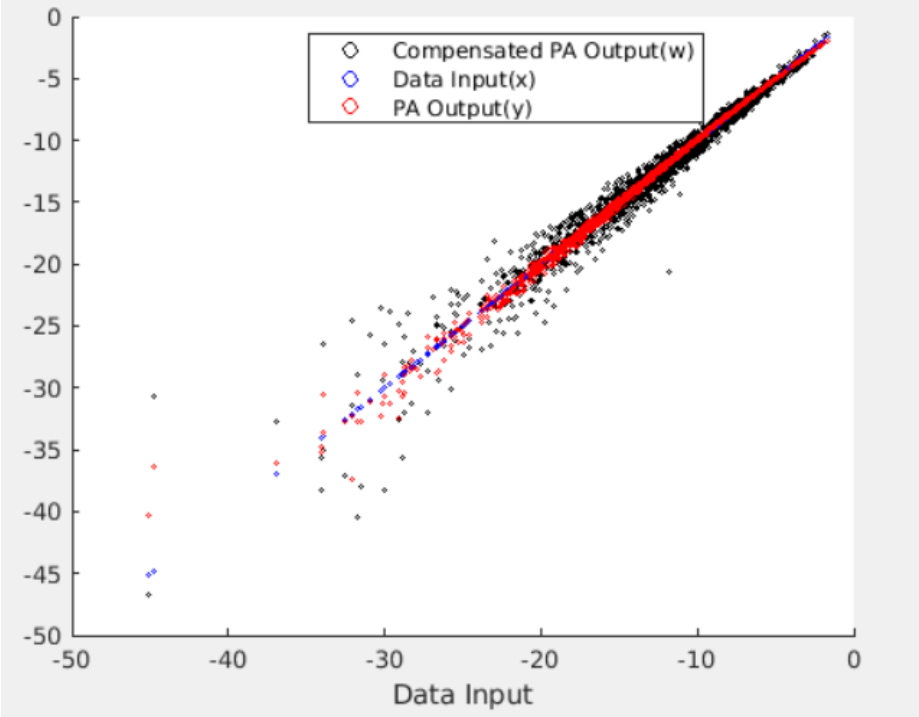


Figure 6.8: AM/AM Plot

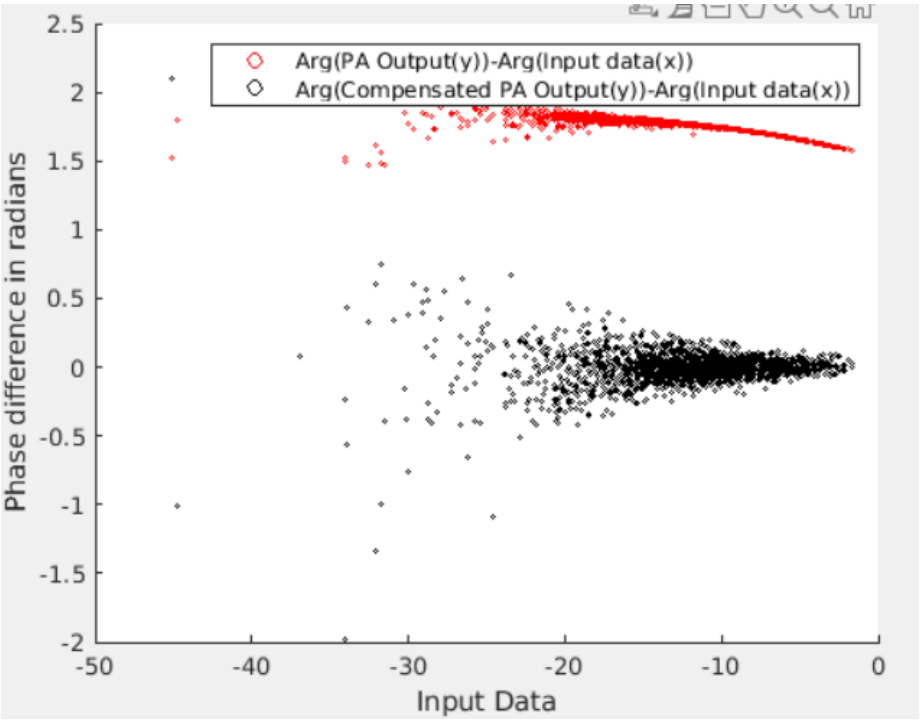


Figure 6.9: AM/PM plot

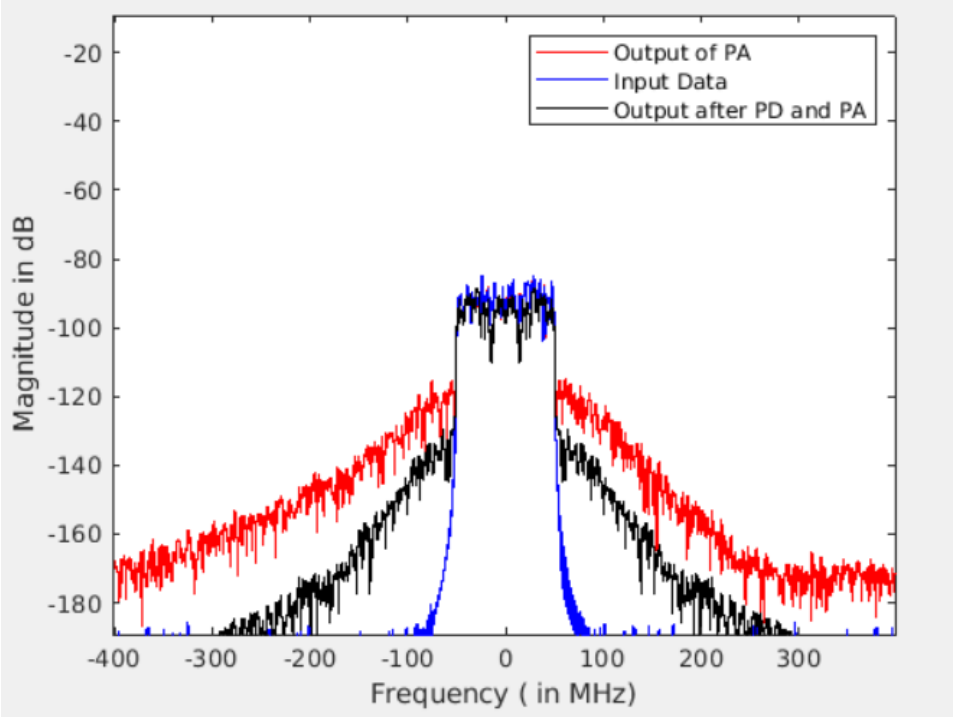


Figure 6.10: Spectral plot

6.2.3 Conclusion

This way of training the neural network is very slow compared to the previous method as the power amplifier has to be simulated in every training epoch. Hence I used just

1200 epochs to train the LSTM-DNN. Despite this, we can observe decent suppression in spectral regrowth and linearity in AM/AM plot. The AM/PM plot is around zero for all input as desired. Thus we get better performance using this technique compared to the previous way of training for same number of epochs.

CHAPTER 7

Conclusion & Future work

I simulated different Power Amplifier models using Matlab Simulink and then implemented DPD using different neural network models to increase the working range of input power for the PA models.

Feedforward networks linearize the memoryless power amplifiers very well. LSTM-based neural networks are well suitable for implementing DPD for Power Amplifiers with memory.

Both the approaches to train the neural network discussed in the paper result in good linearization of PAs.

We can implement DPD on real communication hardware in the future. The neural network will have to pre-trained using a set of power amplifier input and output. Once trained, we can use the network for DPD by connecting it as a block just before the PA.