Digital Pre-Distortion Using Deep Learning

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech.

in

ELECTRICAL ENGINEERING

By

KEERTHI SURESH

EE15B131

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY
MADRAS-600036

MAY 2019

ABSTRACT

This project is aimed at implementing digital pre-distortion using deep learning. Operating a power amplifier (PA) close to its peak power introduces non-linearities. Digital pre-distortion is a method to distort the signal before it enters the PA so that the output will be linear even when operated close to peak power. We investigate different methods to realize this using a neural network. The neural network, once trained can be used effectively as a pre-distorter. The architectures and meta parameters are decided and performance analyzed for different kinds of non-linearities and for two sets of real world data. We have used keras to implement the algorithm.

ACKNOWLEDGEMENT

I would like to thank the following people for their support and guidance without whom the completion of this project in fruition would not be possible. Dr. Devendra Jalihal, my project guide, for helping me and guiding me in the course of this project. Mr. Krishna Kumar's insights were very helpful to get a handle on the problem statement. I would also like to thank my family and friends for their constant support.

TABLE OF CONTENTS

List of Figures

Chapter 1

Introduction

An important element in a wireless communication transmitter is the power amplifer (PA). For good efficiency, the PA should be operated close to its peak power. However, this could drive the device to the non-linear region of operation. Non-linearity introduces spectral re-growth outside the allocated bandwidth thus violating the spectral mask. Further, in the case of linear modulations, in-band distortion introduced by the non-linear behaviour of the PA causes increased error vector magnitude (EVM) at the transmitted output. To achieve linearity and efficiency simultaneously, linearisation techniques are employed. For wideband waveforms with symbol periods comparable to the device memory, the non-linearity exhibits memory effects and the compensation is challenging. PA linearisation can be implemented either in the analog domain or in the digital domain. The latter is often preferred due to the flexibility of design and versatility and repeatability of implementation. Digital Pre-Distortion (DPD) is a popular linearisation technique that uses baseband digital signal processing to pre-distort the envelope such that the distortion introduced by the PA can recover the original envelope. The idea is motivated by the fact that behaviour of the PA, and hence its inverse (the pre-distorter), can be represented using baseband equivalent discrete-time models.

Deep Neural Networks (DNN) have recently gained a lot of attention in signal processing community, particularly in image and speech processing. Application of DNNs to communication problems is a recent development. We investigate the possibility of using a DNN to realize a DPD. An important aspect of the problem is that the inputs and outputs are (complex-valued) samples. Neural networks typically deal only with real

values.

Objective

This project is about investigating the effectiveness of deep learning for DPD. Two different neural network architectures are proposed. Their pros and cones are analyzed. The range of operation of the PA for which the DPD works is also investigated. First, BPSK data (string of -1s and 1s) are checked for better understanding of the system. Then we increase the complexity to memoryless and quasi-memoryless models. Finally we setup the neural network for real world data obtained from PA.

Chapter 2

Background

An understanding of a communication system and the nature of the PA is crucial in order to understand the problem. A background on data flow in wireless communication and deep learning is provided herewith.

2.1 Signal Flow

The bits generated are mapped into the respective constellations followed by spectral shaping (for example, using a Square Root Raised Cosine pulse) and oversampling. This is passed to the Power Amplifier which, ideally gives a large gain G. The gain should be large enough to make the signal detectable at the receiver side, i.e, the SNR should be high enough for good reception. The signal flow has been illustrated in figure 2.1.

2.2 Role of Power Amplifier

Ideally, the PA output y should be a scaled version of its input x, i.e., $y = gx$, where g is the gain of the amplifer. In practice, non-linearity and memory effects too need to be taken into account; hence the PA is modelled as a non-linear operator with memory, denoted as $\mathbf{G}$ such that $y = \boldsymbol{G}x$. The objective of the DPD is to identify a (scaled) non-linear *pre-inverse* $\mathbf{H}$ for $\mathbf{G}$ such that $\mathbf{GH} = g\mathbf{I}$ where I is the identity operator. Unlike linear systems, the non-linear inverse may not be well defined or unique. Figure 2.2 has a system-level representation of a typical Digital Pre-distorter.

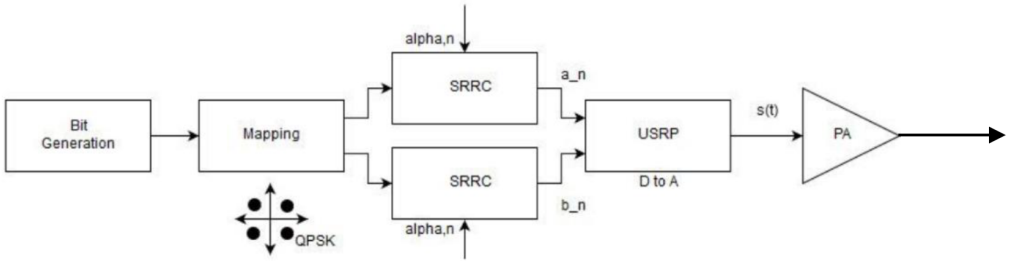


Figure 2.1: Signal Flow Diagram

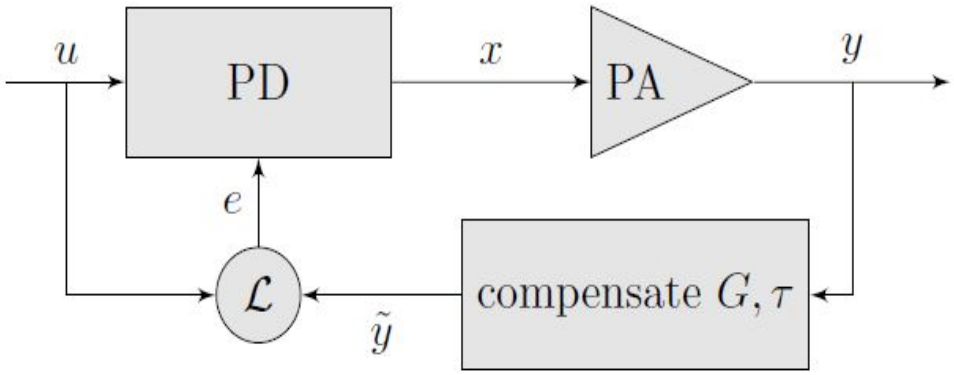


Figure 2.2: System-Level Representation of DPD

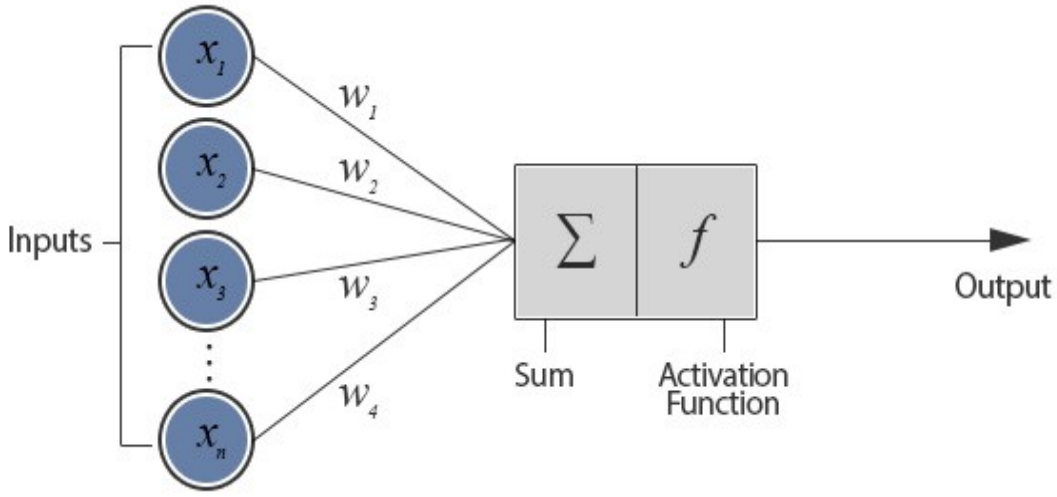


Figure 2.3: Artificial Neuron, implements $f(\sum_1^n x_i w_i)$

2.3 Machine Learning and Neural Networks

An artificial neuron is a mathematical function which operates on a set of inputs to give a single output, as shown in figure 2.3. Each input goes into a small unit called node and is multiplied with a real number called weight. These weighted values are then summed up and a function (called activation function) operates on it. This value is the output of the function. Various such neurons arranged as a network can be configured to mimic any function. When a neural network has one or more ”hidden layers” (a layer other than input and output layer), it is called a Deep Neural Network (DNN) and the machine learning associated with it is deep learning. An illustration of deep neural network is in figure 2.4.

We start with a fixed number of nodes and layers. We initialize the weights randomly and provide the neural network with some training data *x_train* and *y_train*. We specify the cost function, for example, Mean Square Error(MSE). The training data is divided into *batches* and goes through the network. After each iteration, the error is calculated and weights are updated in the direction of decreasing error using an optimizer. When this is done once for all the data points, one epoch is completed. Repeated training for more epochs reduces the error further. After training, the neural network

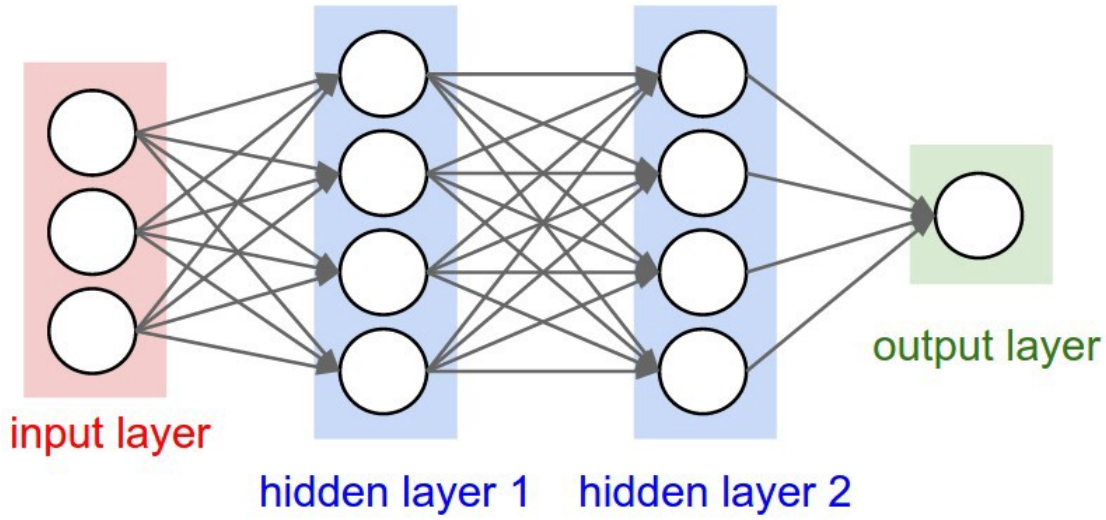


Figure 2.4: A Deep Neural Network with 2 hidden layers

can be used to predict y values for x values given as inputs.

2.4 Choosing a Platform: Keras

Keras is a high level API for building deep learning models. It has gained favor for its ease of use and syntactic simplicity facilitating fast development. Keras was chosen for the following reasons:

1. Focus on user experience
2. Research Community
3. Fast prototyping
4. Simple to get started
5. Easy production of models

Chapter 3

Preliminary Model

3.1 Dataset Specifications

A random array of 1s and -1s is created. The training data is 80% of the data generated and the test data is the remaining 20%.

3.2 Modelling the non-linearity

The PA model in the problem statement operates on complex values. These values are the complex baseband equivalent ($s(t) = acos(\omega t) + bsin(\omega t)$) of real samples. After passing through the PA, a bandpass filter cenetered at the centre frequency and with width slightly more than twice the bandwidth is applied on the signal. Even order non-linearities will give spectral components close to DC or at $2f_c$, $4f_c$ etc. These will get filtered out in the later stage and hence will not contribute to the spectral regrowth. Only the odd order non-linearities get affected, and a function like $f_1(x) = x + \alpha_1 x^3 + \alpha_2 x^5$ will be equivalent to $f_2(x) = x + \alpha_1 x \mid x \mid^2 + \alpha_2 x \mid x \mid^4$ in the passband [3].Hence, we chose a function $f_2(x) = x + \alpha_1 x \mid x \mid^2$. We set alpha = 0.1, i.e around 10% error.

3.3 Program Set-up

First data is generated, then partitioned into training and test sets. After this training is done. Finally, the test set is used to analyze the performance of the neural network. The set-up is illustrated in figure 3.1.

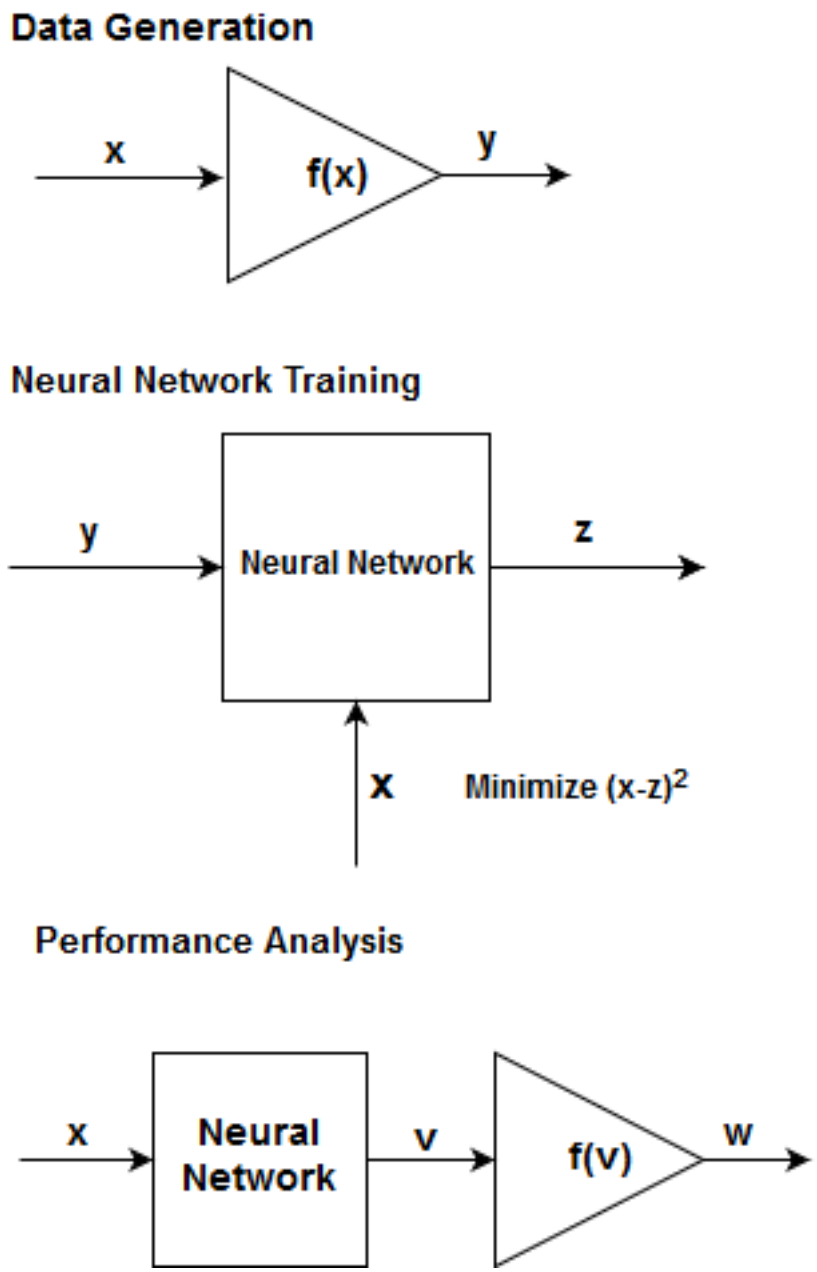


Figure 3.1: Schematic for preliminary model

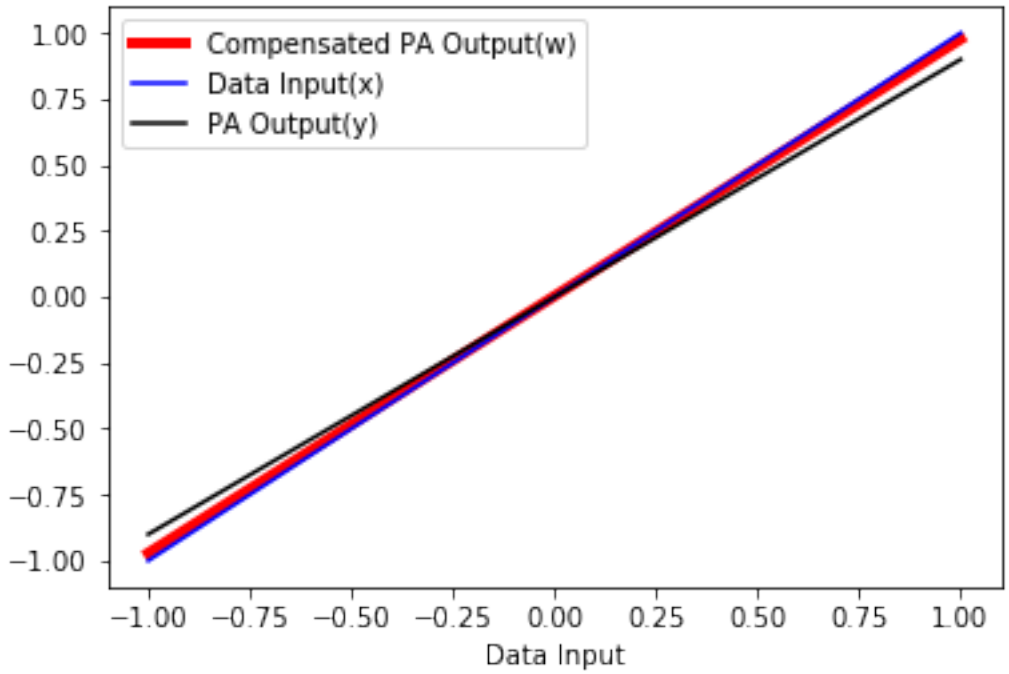


Figure 3.2: AM-AM Plot for Preliminary model

3.4 Neural Network Metaparameters

```
model = Sequential()
model.add(Dense(units=2000, input_dim=1))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=1))

model.compile(loss='mean_squared_error',
          optimizer='adam')
model.fit(y_train, x_train, epochs=75, batch_size=100,
   verbose=1)
```

3.5 Performance

The Mean Squared Error(MSE) is 0.401 in the first epoch. It goes down to 10^{-11} by the 75th epoch. Plotting x,y and w on the same graph shows that w is much linearized compared to y in figure 3.2. **Note:**For every AM-AM plot hereafter, the x-axis is the input.

Chapter 4

DPD for Two Dimensional Constellations

4.1 Usage of Complex Numbers

Most platforms for machine learning (like keras, tensorflow, PyTorch etc) operate only on real numbers. This means that input, output and weights are all real numbers. Passing the complex values as-is can result in the imaginary parts being discarded. Hence we devise a way to convert these complex values to features that can still preserve the phase dependence. By converting each complex sample to a 1X2 vector, we bypass this issue [4].

4.2 Spectral Representation

Plotting a magnitude spectrum is a better way to compare the input and output since time domain comparison may not reflect small non-linearities. Matplotlib has functions to plot magnitude spectrum. Here we have used welch method with fft size and hanning window of size 2048 or length of input, whichever is smaller [2].

Note: For every Spectral plot, the x-axis is the frequency and y-axis is the spectral magnitude in dB.

```
def plot_spec(spec_input):

    # Spectral analyses of PA input and PA output
    Fs = 15e6
    N = np.min(np.array([1024,len(spec_input)]));
    pwin = hann(N);
    f_axis,spec_input_ps = welch(spec_input, Fs, window=pwin,
```

```
    nfft=N, return_onesided=False,scaling =
    'spectrum',detrend=False)

spec_input_ps_dB = [ 10*math.log10(abs(spec_input_ps[i]))
    for i in range(len(spec_input_ps)) ];

# Plots
f_axis = [f_axis[i]/1000000 for i in range(len(f_axis))]
plt.plot(f_axis,spec_input_ps_dB,label = plot_label)
```

4.3 Regarding DPD for Non-Invertible PA Models

As mentioned in [5], in section 2.2.2 - Comments on Invertibility. A typical PA characteristic is shown in figure 4.1. Ideal predistortion is done only upto the saturation point A_{sat} letting the PD response be arbitrarily defined for inputs beyond such level. This is basically due to the limitation found in evaluating the theoretical AM/AM inverse.
The nonlinear distortion introduced by a PA can be divided into two significant categories:
- Distortions originated by the AM/AM and AM/PM nonlinear behaviour of the PA for input signal values within the 'valid' (under-saturation) input range [0,A_{sat}].
-Distortions due to the saturation imposed by the AM/AM characteristic over the input signal when its amplitude exceeds A_{sat}.

Where the former type corresponds to the portion of nonlinear effect that can be effectively counteracted by pre-distorting the input signal through the correct inverse AM/AM and AM/PM curves, while the second type refers to degradations that cannot be compensated by any efficient mean. In practice, the linearization of PAs achievable with a digital pre-distorter can be defined only within the valid input range that goes from the zero input amplitude to the saturation point.

Clearly, the PA can be operated only within its invertible range.

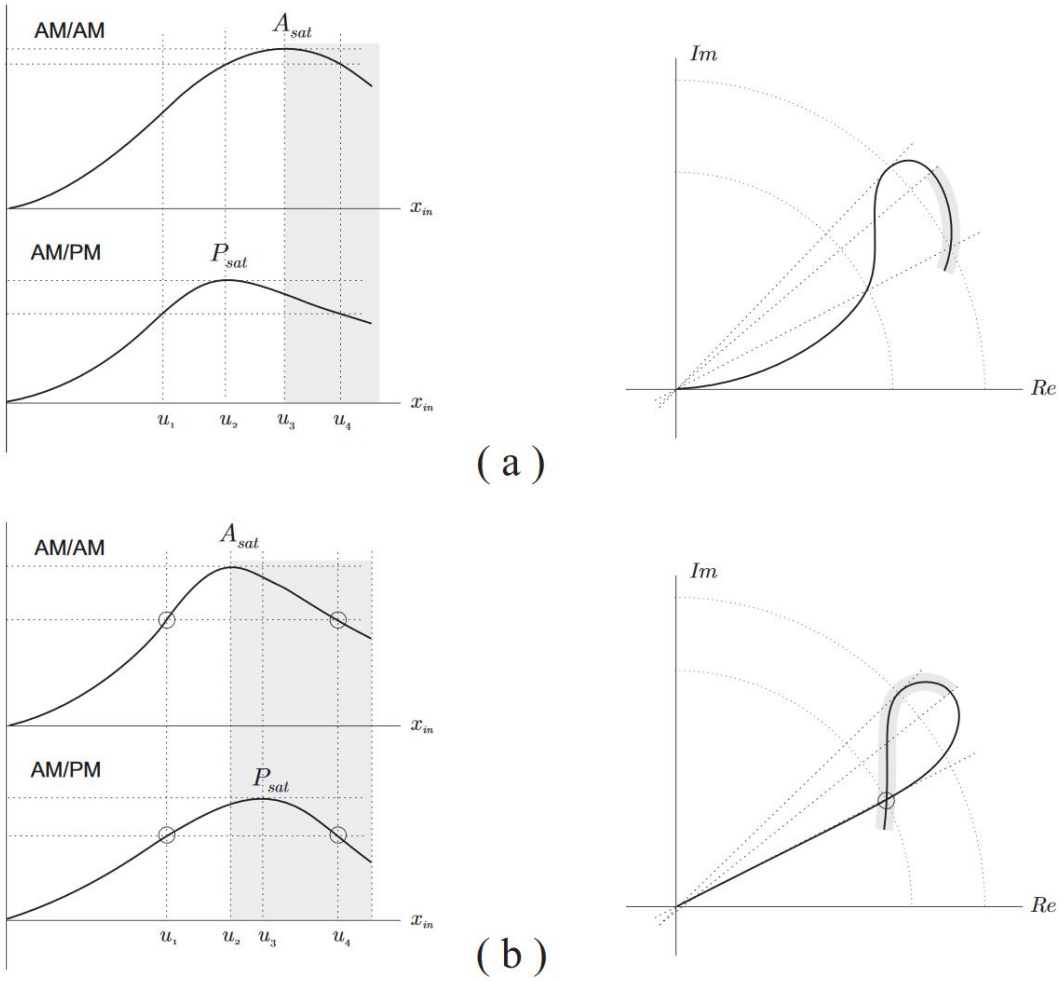


Figure 4.1: PA Model Characteristics [5]

4.4 Approach 1: Using only 1 Neural Network as Pre-Distorter

This approach is simple and intuitive. It does not take much time to train since only one neural network is involved. This works on the assumption that the post-inverse will also work as the pre-inverse. This may not be always true for complicated PA models. But provided we have enough data it is possible to get good spectral suppression as shown in the following chapters. The schematic is same as shown in figure 3.1

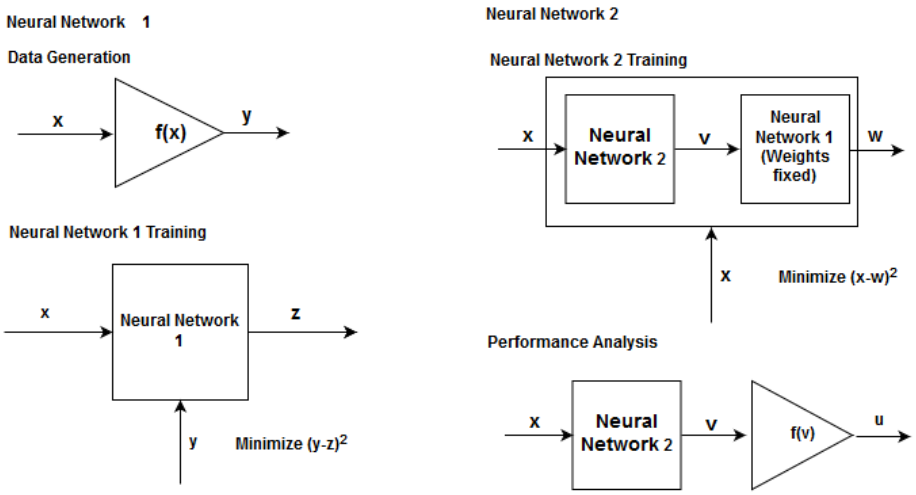


Figure 4.2: Schematic for Approach 2

4.5 Approach 2: Using 2 Neural Networks as PA and DPD

This is more difficult and time-taking to train. The error in neural network 1 limits the accuracy of the pre-distorter network (Neural Network 2). But this works for various non-linear models of PA which may not work for approach 1.

4.5.1 Simplified Method

For testing purposes when the PA function is known directly, we can remove Neural Network 1 and insert a custom layer that implements the function of the PA, as shown in figure 4.3. We shall use this for cases where the PA function is known and the function is represent-able using Lambda function, i.e in the memoryless case. For quasi-memoryless, since the coefficients of the function are imaginary, keras discards the imaginary part and hence the simplified method cannot be used.

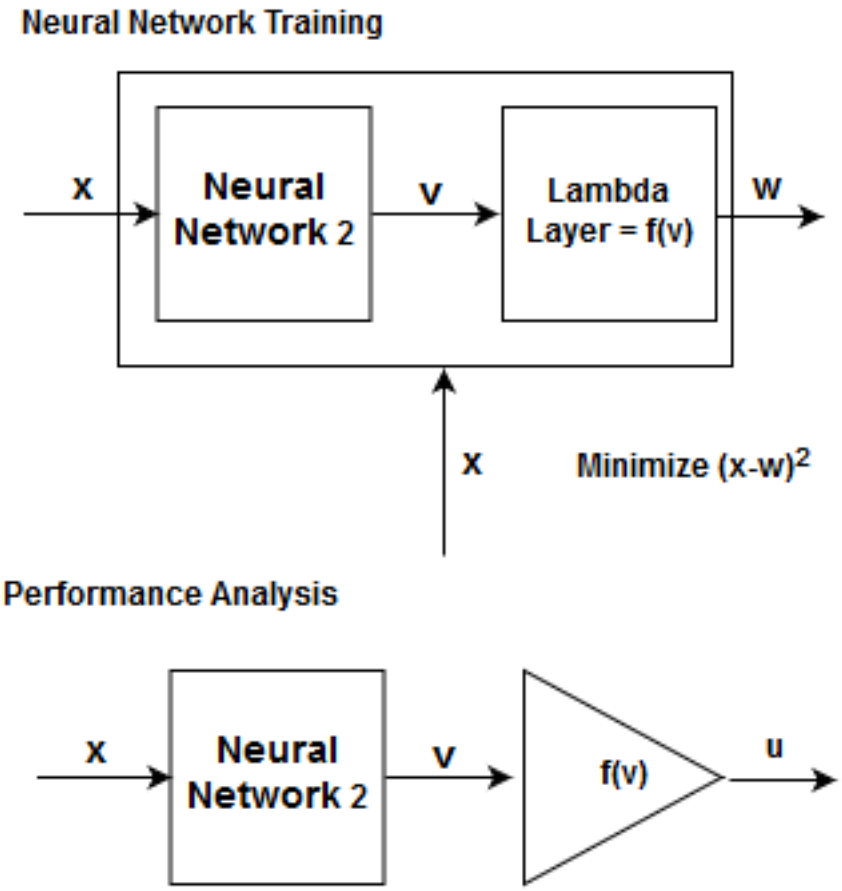


Figure 4.3: Simplified Schematic for approach 2

In the following chapters, we shall check the performance of these two approaches and the set-ups shall be referred-to as Approach 1 and Approach 2 respectively.

4.6 Nature Of QPSK Data

For the following chapters 5 and 6, we use a set of QPSK data with the specifications as in figure 4.4.

Sequence type	Pseudo Random Binary Sequence (PRBS)
Modulation	QPSK
Symbol Rate	960 K symbols per second
Pulse Shaping	Root Raised Cosine (RRC), Roll off = 0.22
Occupied Bandwidth	1.1712 MHz
Sampling Rate	7.68 MHz (8 samples/per symbol)
EVM	1.2 % rms
PAPR	4.6 dB

Figure 4.4: QPSK Signal Specifications

Chapter 5

Memoryless Model

5.1 Nature of Function used

We need to use a function that is invertible in the range of the data used. Hence we scale the QPSK data to the range -1 to 1.

5.2 Third order Non-linearity

The function chosen is

$$f(x) = x - 0.1x \mid x \mid^2 \tag{5.1}$$

Figure 5.1: 3rd Order Non-Linearity

5.2.1 Approach 1

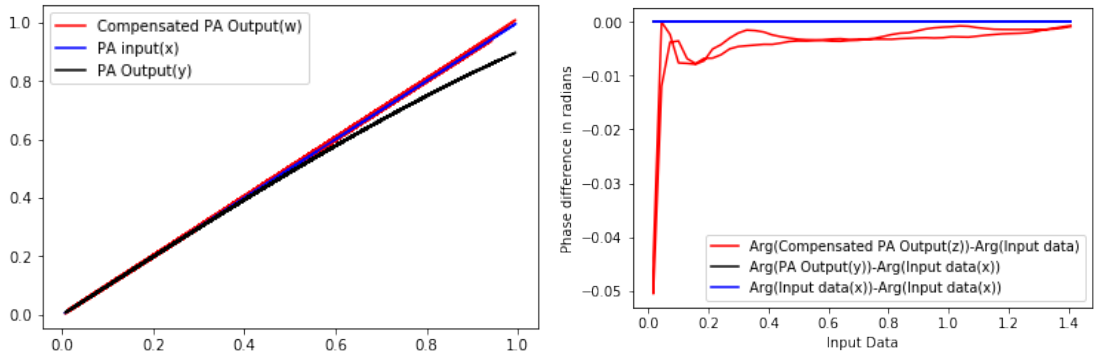
Network Meta Parameters

Training Data: uniformly distributed random complex numbers between -1.05 to 1.05. The inflection point (at which the slope starts to change sign) is around ± 1.16, hence we chose ± 1.05 as the extreme values. The range of input is taken randomly because the qpsk values are not of a reasonable enough distribution to ensure correct representation of the PA non-linearity.

Test Data: QPSK Values as explained in section 4.4.

```
model = Sequential()
model.add(Dense(units=2000, input_dim=2))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=2))

model.compile(loss='mean_squared_error',
              optimizer='adam')

model.fit(y_train, x_train, epochs=75, batch_size=100,
   verbose=1)
```

Performance



(a) AM-AM plot

(b) AM-PM plot

Figure 5.2: 3rd Order memoryless DPD Results for approach 1

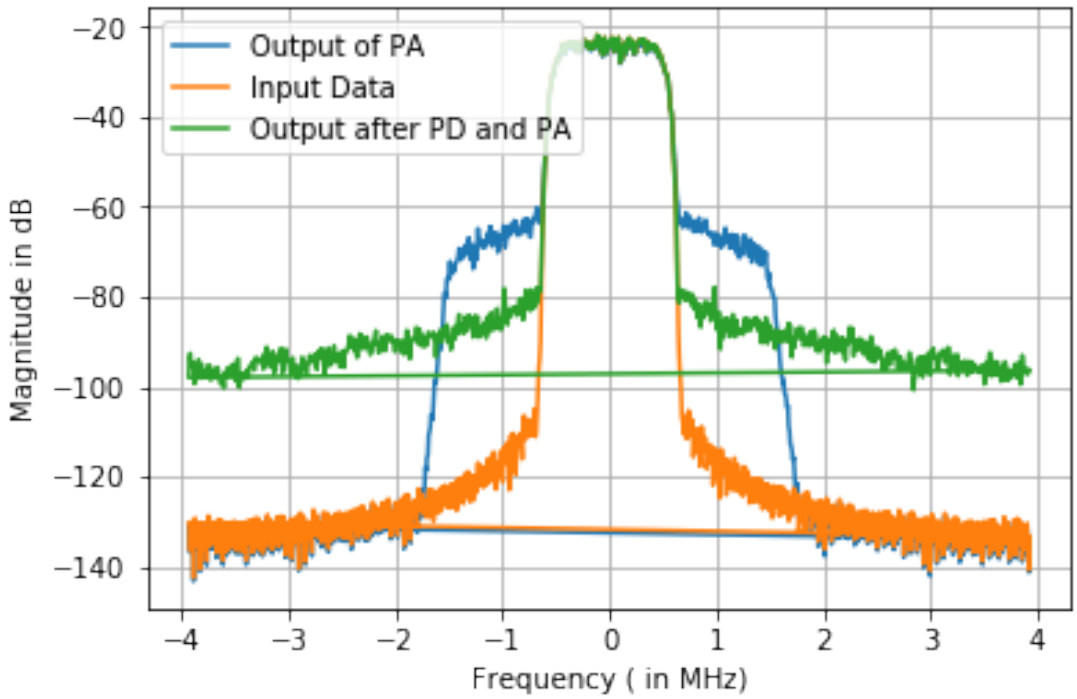


Figure 5.3: Spectral Plot for 3rd Order memoryless DPD Results for approach 1

5.2.2 Approach 2

Network Meta Parameters

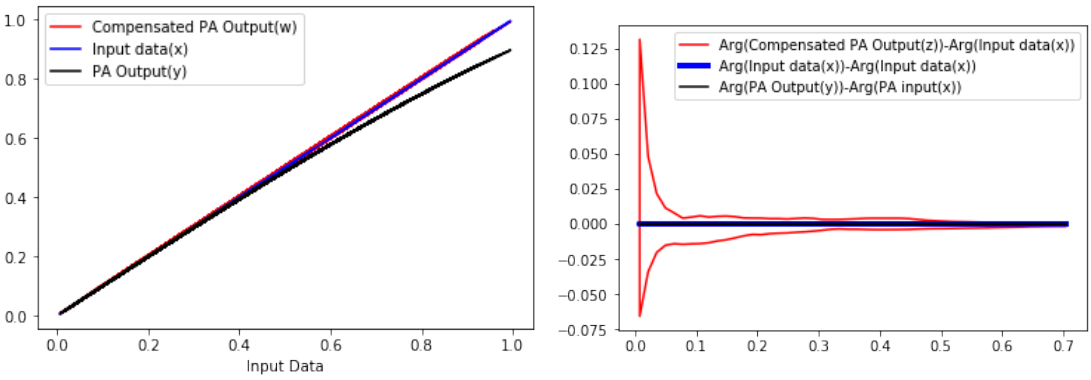
Training Data: uniformly distributed random complex numbers between -1.05 to 1.05.

Test Data: QPSK Values as explained in section 4.4.

```
model = Sequential()
model.add(Dense(units=2000, input_dim=2))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=2))

model.add(Lambda(lambda x: x - 0.1*K.sum(x*x,axis =
   1,keepdims=True)*x))
model.compile(loss='mean_squared_error', optimizer='adagrad',
   metrics=['mse'])

model.compile(loss='mean_squared_error',
          optimizer='adam')

model.fit(x_train, x_train, epochs=75, batch_size=100,
   verbose=1)
model2 = Sequential()
```

```
for layer in model.layers[:-1]:
    model2.add(layer)

classes = model2.predict(x_test, batch_size=100)
```

Performance



(a) AM-AM plot

(b) AM-PM plot

Figure 5.4: 3rd Order memoryless DPD Results for Approach 2

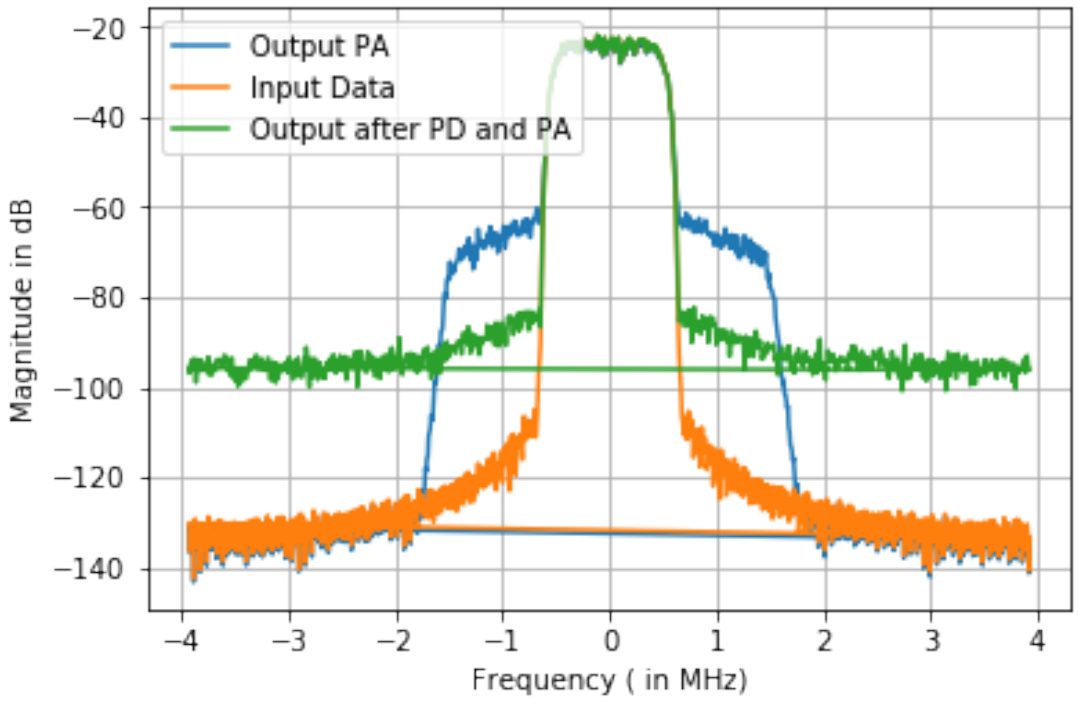


Figure 5.5: Spectral Plot for 3rd Order memoryless DPD Results for approach 1

5.3 Fifth order Non-linearity

The function chosen is

$$f(x) = x - 0.1x \mid x \mid^2 + 0.005x \mid x \mid^4 \tag{5.2}$$

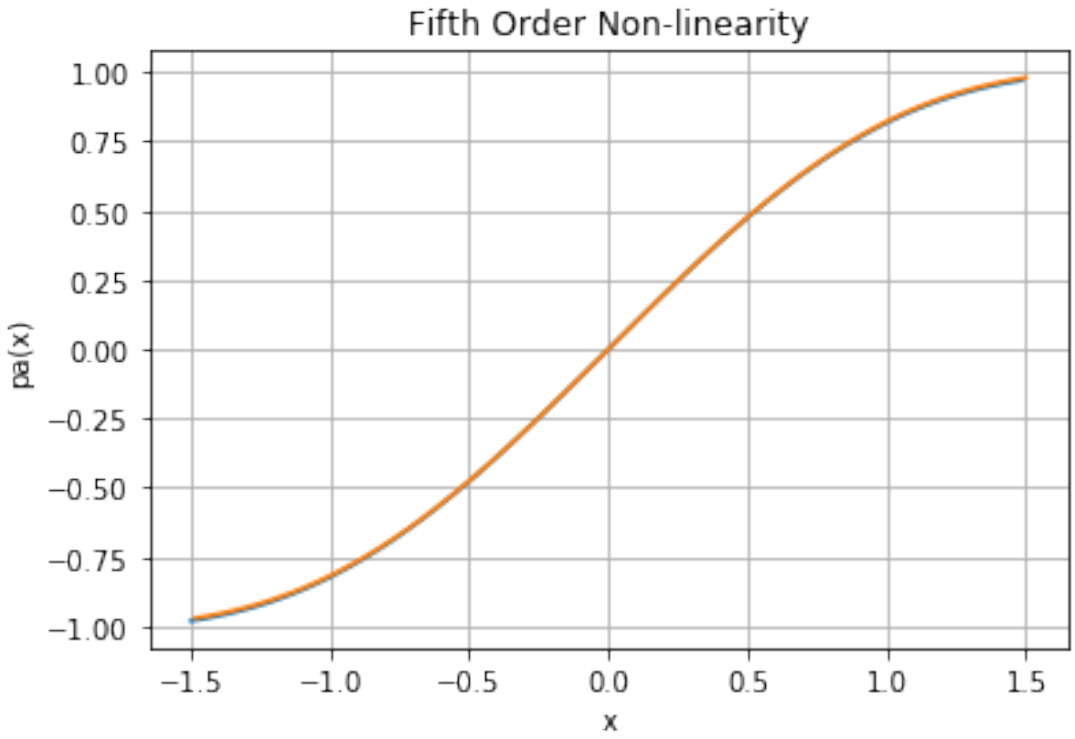


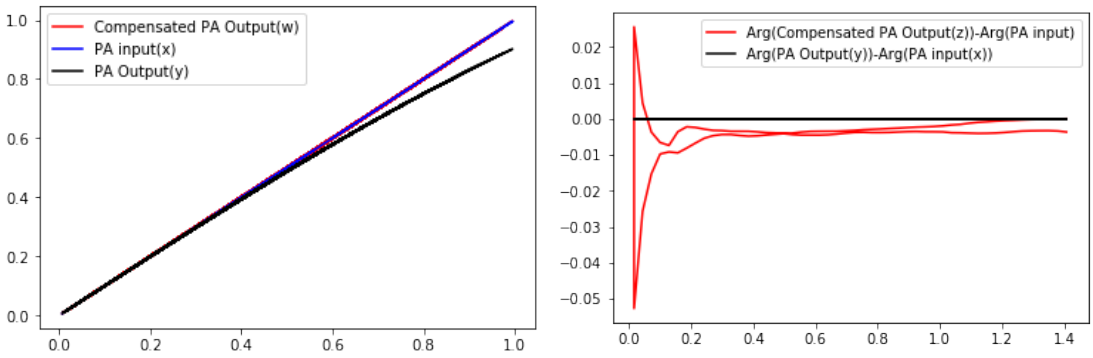
Figure 5.6: 5th Order Non-Linearity

5.3.1 Approach 1

Network Meta Parameters

Same as that of section 5.2.1.

Performance



(a) AM-AM plot

(b) AM-PM plot

Figure 5.7: 5th Order memoryless DPD Results for approach 1

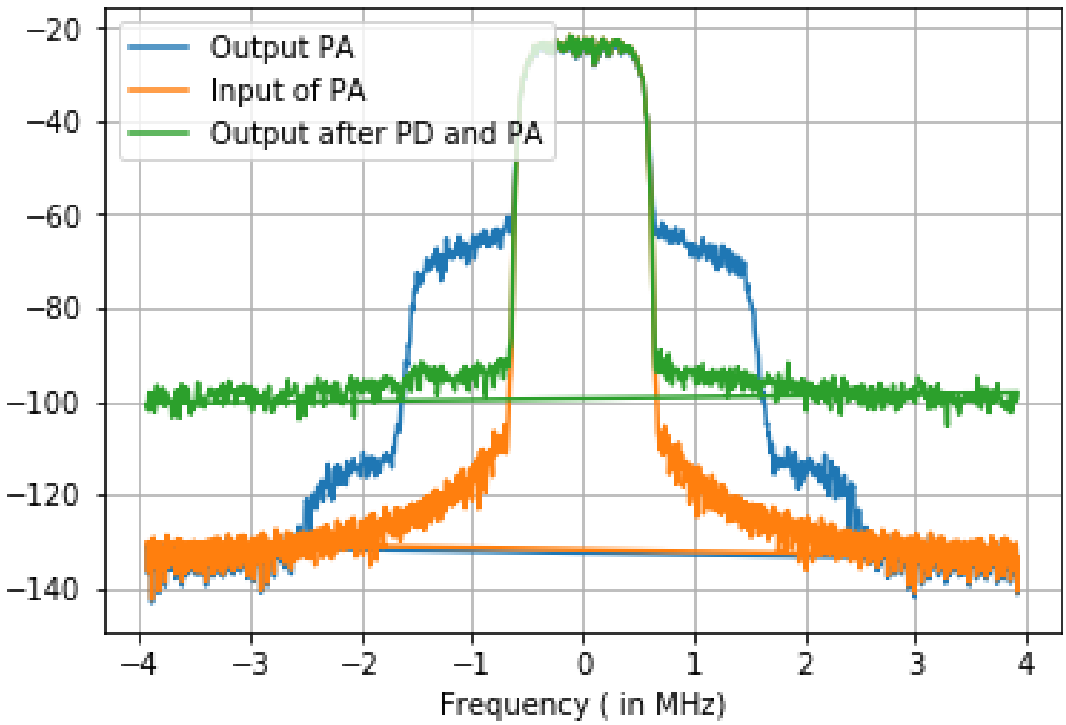


Figure 5.8: Spectral Plot for 5th Order memoryless DPD Results for approach 1

5.3.2 Approach 2

Network Meta Parameters

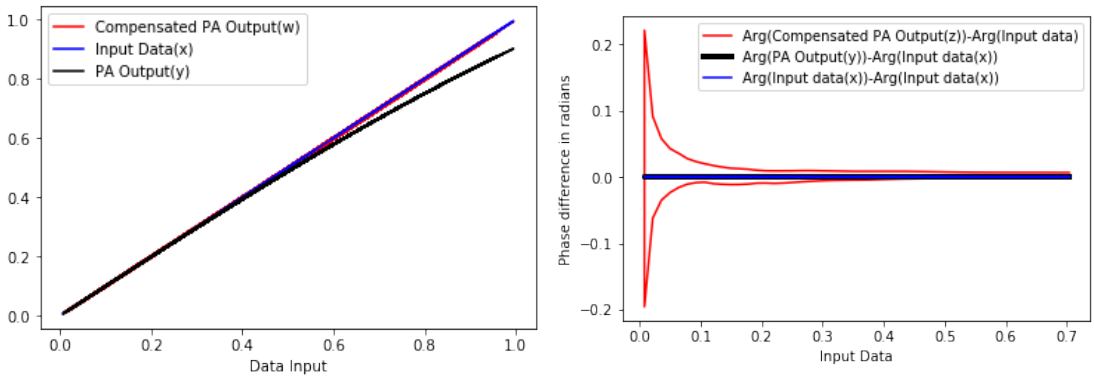
Training Data: uniformly distributed random complex numbers between -1.05 to 1.05.

Test Data: QPSK Values as explained in section 4.4.

```
x_train,y_train,x_test,y_test = Generate_Data()
```

```
model = Sequential()
model.add(Dense(units=2000, input_dim=2))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=2))

model.add(Lambda(lambda x: x - 0.1*K.sum(x*x,axis =
    1,keepdims=True)*x+0.005*tf.multiply(K.sum(x*x,axis =
    1,keepdims=True),K.sum(x*x,axis = 1,keepdims=True))*x))
model.compile(loss='mean_squared_error', optimizer='adagrad',
    metrics=['mse'])

model.compile(loss='mean_squared_error',
              optimizer='adam')

model.fit(x_train, x_train, epochs=75, batch_size=100,
    verbose=1)

model2 = Sequential()
for layer in model.layers[:-1]:
    model2.add(layer)

classes = model2.predict(x_test, batch_size=100)
```

Performance



(a) AM-AM plot

(b) AM-PM plot

Figure 5.9: 5th Order memoryless DPD Results for approach 2

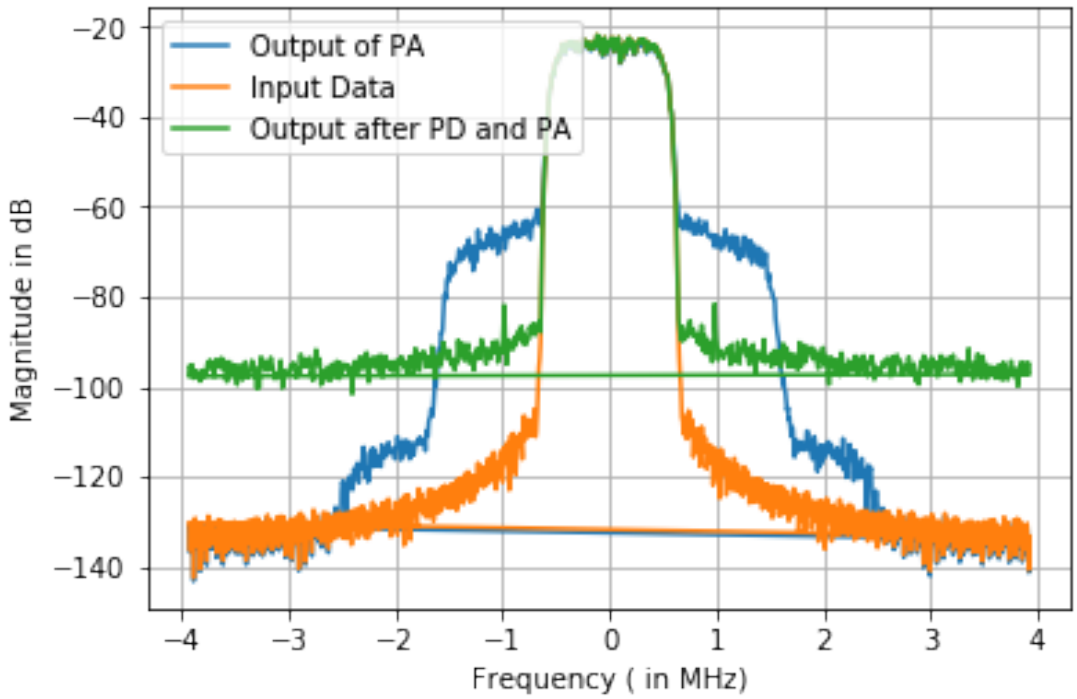


Figure 5.10: Spectral Plot for 5th Order memoryless DPD Results for approach 2

5.4 Conclusions

Both the approaches work well for both third order and fifth order non-linearities. The AM-AM plot shows that the magnitude of the compensated output perfectly matches that of the input. The AM-PM plot is between input power and output phase difference. Since the memoryless model does not introduce any phase mixing, the AM-PM plot for the PA output is a straight line at zero. The same plot for the compensated output is also very close to zero, the irregularities are due to numerical error.

Approach 1 works better for both and is more practical as well. There is a suppression of 20-40dB in the spectral regrowth. The noise floor is of around -95dB which is likely to be due to numerical error. The reason that approach 1 works better for this case might be that for these simple cases it is easier for the network to learn the post-inverse rather than the pre-inverse. If given more training data and trained for more epochs, approach 2 can also give as good a result as approach 1. But for the sake of comparability between the different models and approaches, we have used the same amount of training data and epochs for all the four cases investigated in this chapter.

Chapter 6

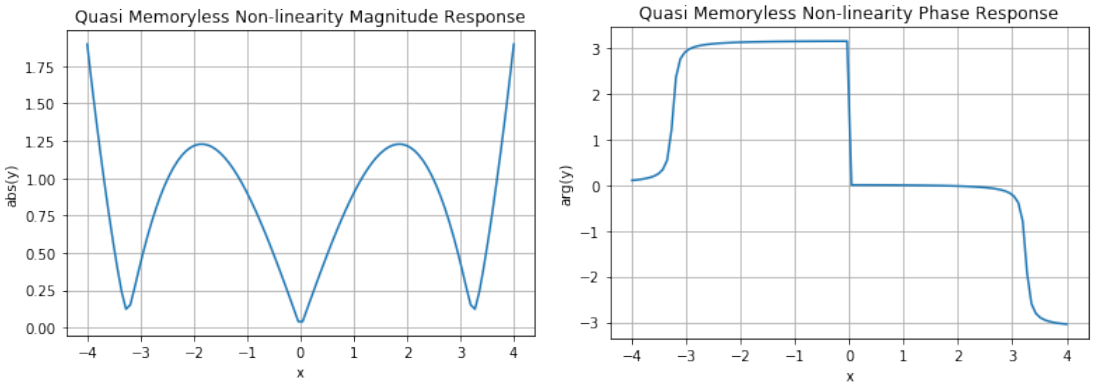
Quasi-Memoryless Model

6.1 Nature of Function used

Quasi memoryless PAs have complex numbers as coefficients. This introduces a phase mixing. Quasi memoryless models are more difficult to linearize than memoryless ones. The function chosen is

$$f(x) = x - 0.1e^{j\pi 0.05/4}|x|^2x + 0.005e^{j0.15\pi/4}|x|^4x \tag{6.1}$$

The magnitude and phase response considered together, is unique for the range considered (-1,1). Hence the DPD will be effective.



(a) Magnitude plot

(b) Phase plot

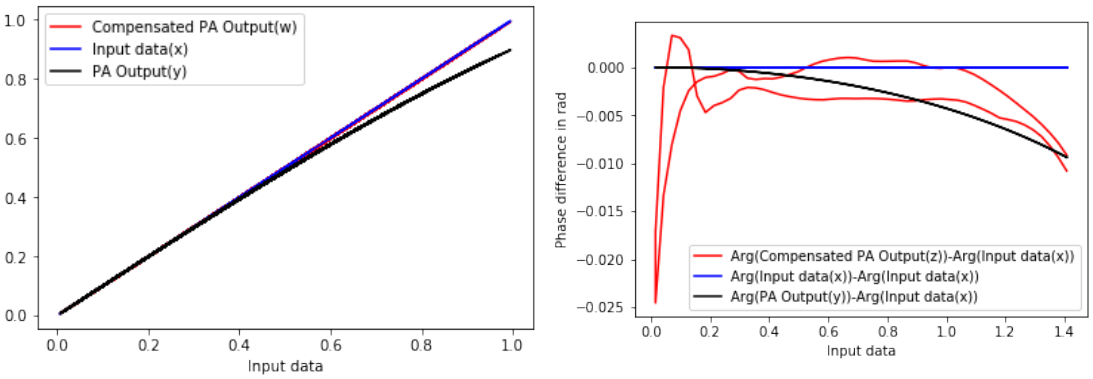
Figure 6.1: Response of Quasi Memoryless PA

6.2 Approach 1

6.2.1 Network MetaParameters

As explained in section 5.2.1.

6.2.2 Performance



(a) AM-AM plot

(b) AM-PM Plot

Figure 6.2: DPD Performance of Quasi Memoryless PA

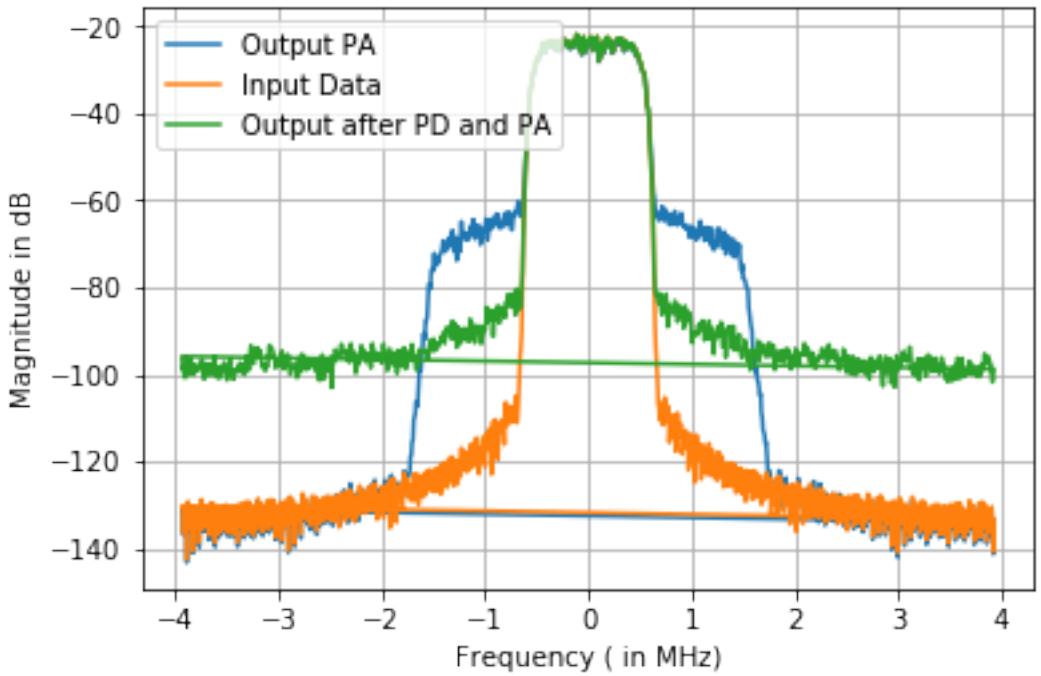


Figure 6.3: Spectral plot of DPD Performance of Quasi Memoryless PA

The AM-AM, AM-PM and spectral plots show considerable suppression of side bands.

6.3 Approach 2

6.3.1 Program Flow

We use approach 2 as explained in section 4.5. The first neural network(designed to mimic the PA) is labelled NN1. The second neural network is labelled NN2. First NN1 is trained and the performance is checked in figure 6.4. Then NN1(with the weights fixed) and NN2 are cascaded together in a neural network and trained. Finally, the trained layers of NN2 are taken separately. These layers are have the DPD functionality. The test data is passed through the DPD layers and then through the PA. Then we check how much suppression has occurred in figure 6.6.

6.3.2 Network MetaParameters

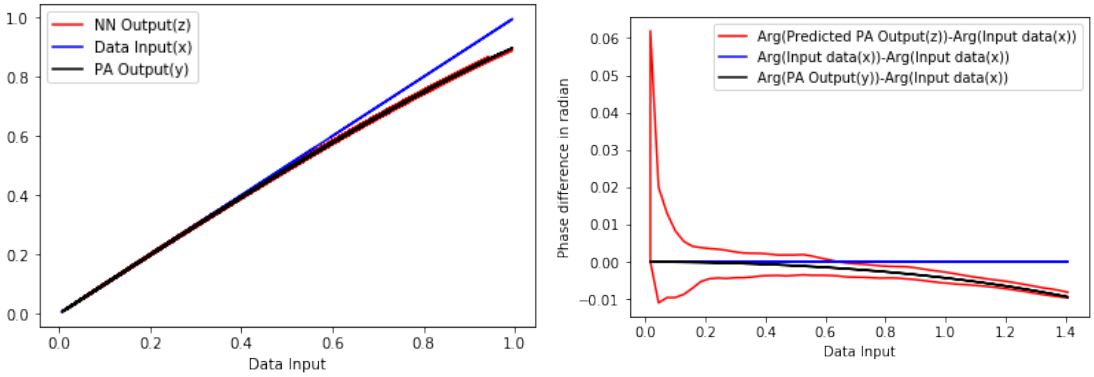
Training Data: uniformly distributed random complex numbers between -1.05 to 1.05.

Test Data: QPSK Values as explained in section 4.4.

```
#NN1
model = Sequential()
model.add(Dense(units=2000, input_dim=2))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=2))

model.compile(loss='mean_squared_error',
          optimizer='adam')

model.fit(x_train, y_train, epochs=75, batch_size=100,
   verbose=1)

# Creating a cascade of NN2 and NN1
model2 = Sequential()
model2.add(Dense(units=2000, input_dim=2))
model2.add(Activation('relu'))
model2.add(Dense(units=100))
model2.add(Activation('tanh'))
model2.add(Dense(units=10))
model2.add(Activation('linear'))
model2.add(Dense(units=2))

for layer in model.layers[:]:
```

```
    layer.trainable = False
    model2.add(layer)

model2.compile(loss='mean_squared_error',
               optimizer='adam')

model2.fit(x_train, x_train, epochs=75, batch_size=100,
    verbose=1)

#NN2 Alone functions as DPD
dpd = Sequential()
for layer in model2.layers[:-7]:
    dpd.add(layer)
```

6.3.3 Performance



(a) AM-AM Plot

(b) AM-PM Plot

Figure 6.4: NN1 Performance of Quasi Memoryless PA

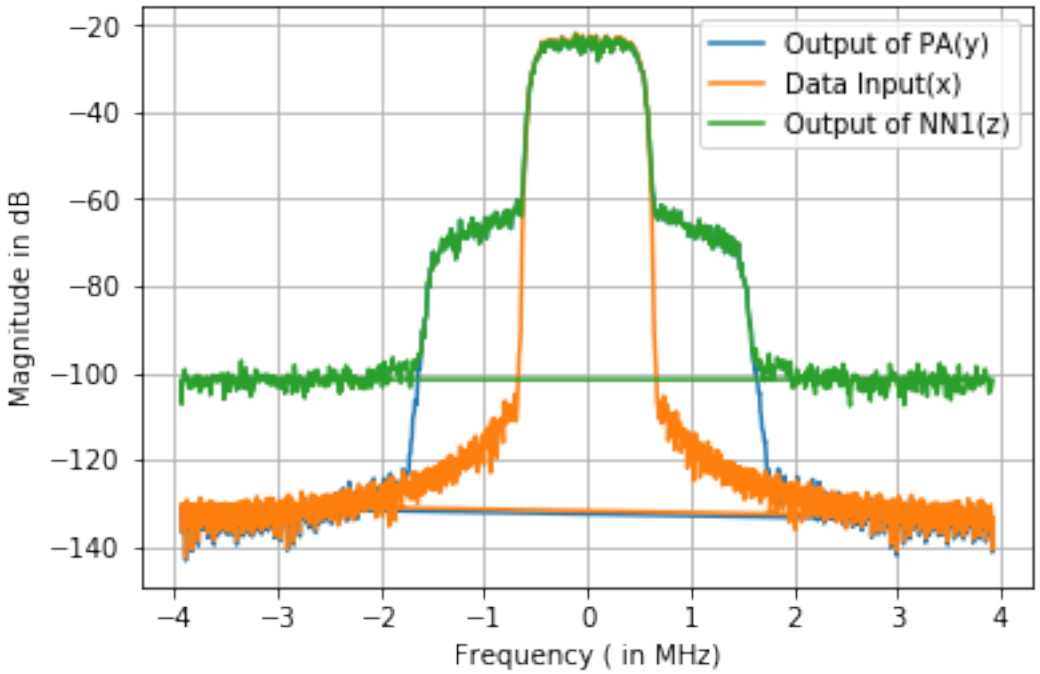
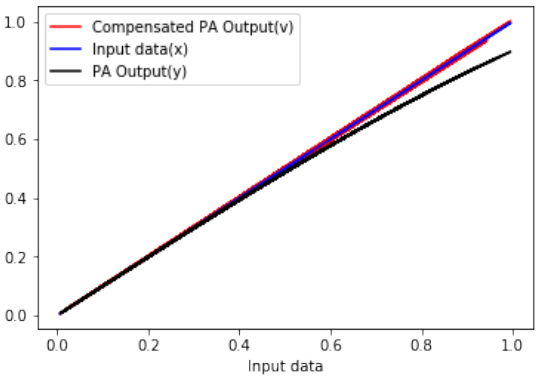
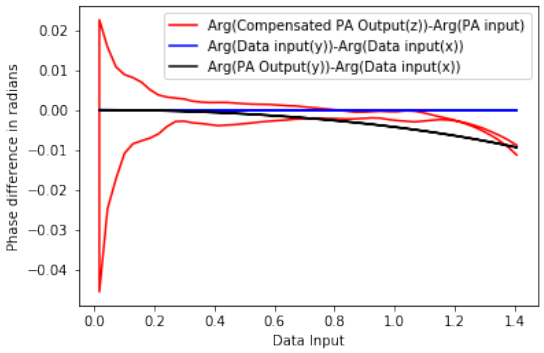


Figure 6.5: Spectral plot of NN1 Performance of Quasi Memoryless PA

NN1 is able to replicate the PA non-linearity effectively(The error at the end of 75 epochs for training data is 10^{-5}). In the AM-AM and spectral plots, the NN output follows the PA output.



(a) AM-AM Plot



(b) AM-PM Plot

Figure 6.6: NN2 Performance of Quasi Memoryless PA

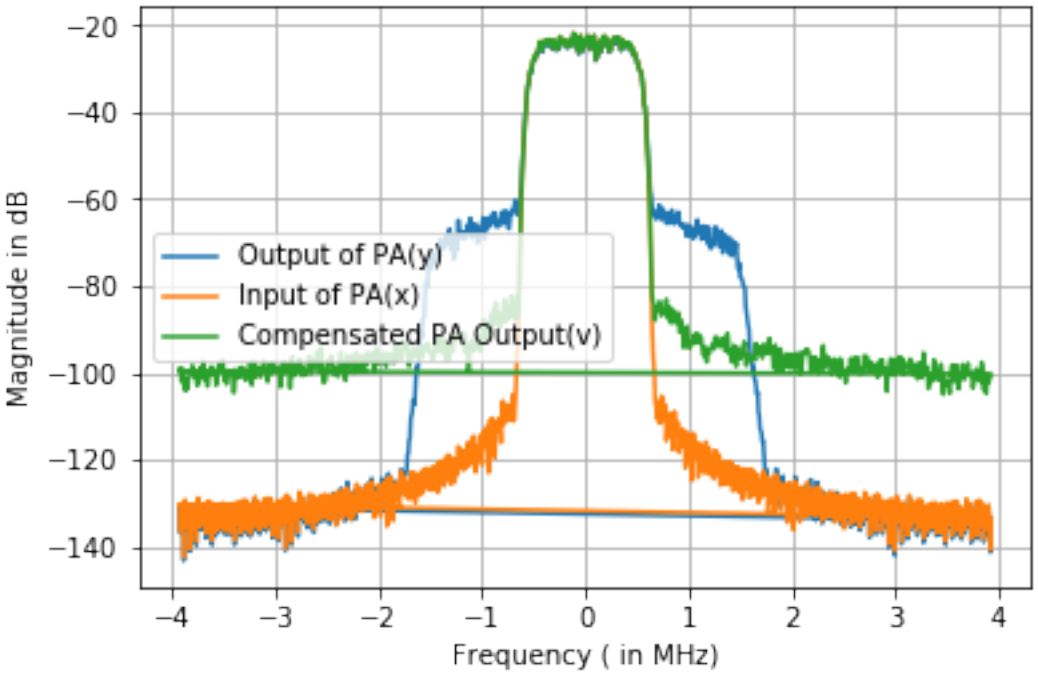


Figure 6.7: Spectral plot of NN2 Performance of Quasi Memoryless PA

The DPD network is able to supress the spectral regrowth by a little more than 20dB.

6.4 Conclusions

The AM-PM plots differ from that of memoryless case. Due to the phase mixing, AM-PM plot for the PA output is not zero. Nevertheless, the network is able to compensate this and the compensated output tends to zero in figures 6.2b and 6.6b. In figure 6.4b, the graph follows the PA Output phase curve, which shows that the network is able to replicate the phase mixing property of the PA.

For approach 1 and approach 2, the spectral suppression is 20dB.There is a noise floor of -100dB. The working of proposed set-ups for linearising quasi-memoryless PAs is verified.

Chapter 7

Real World Model

The testing so far was done on PAs modelled as mathematical functions. Now we use input and output data procured for 2 different real world PAs and check how the network performs for this. There are two main limitations for this exercise:
-The test data and train data, both come from the same pool of transmit data. It may not have a reasonable enough distribution to represent all the non-linearities of the PA.
-For testing in both the approaches, we need a PA model. The neural network output after passing through the PA is what we check for spectral suppression. To overcome the unavailability of a real world PA for the same, we use another neural network trained to mimic the PA.

7.1 Data Specifications

This data was obtained from a real world PA. The bandwidth of the signal is 15MHz and the sampling rate is 50MS/s. The input and output are scaled so that the magnitude is in the range [-1,1].

7.2 Approach 1 - Modified

Since the PA model is not available, we use a modified form of Approach 1, as illustrated in figure 7.1. We train a neural network separately to represent the the PA model. Its performance is shown in figure 7.2a. We use this to check if the output of predistorter network supresses the PA spectral regrowth.

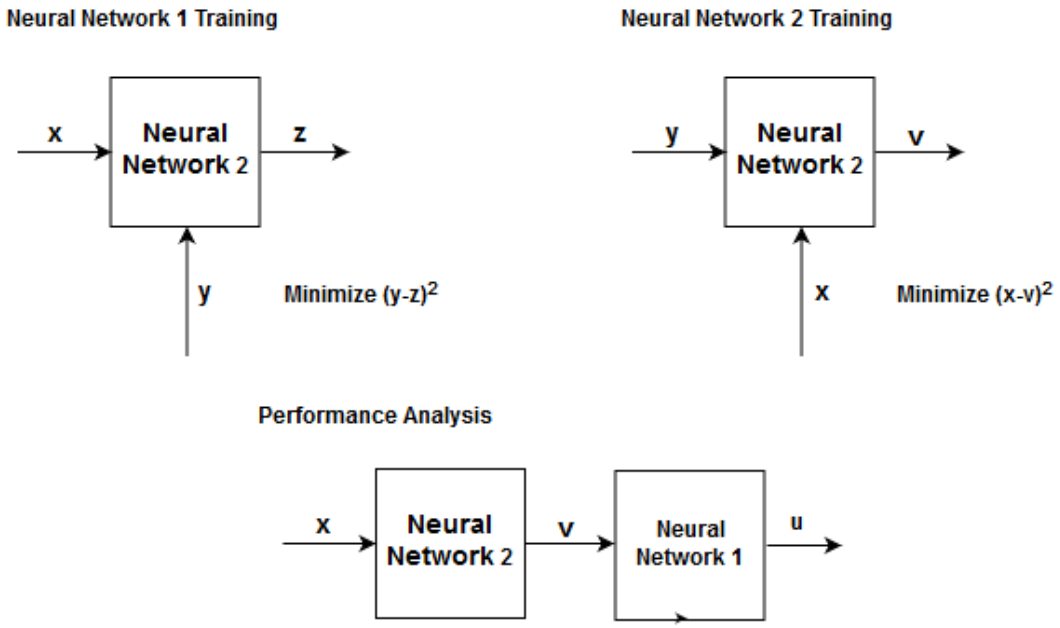


Figure 7.1: Schematic for Approach 1 for Real World PA Model

7.2.1 Network Metaparameters

Training data: 20,000 samples of the PA input and output data

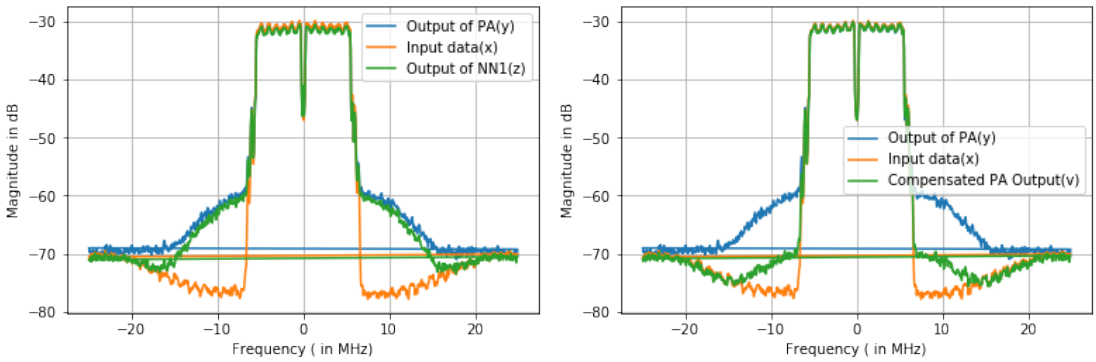
Test data: PA input and output data. We use the whole set of data available for testing so that the spectral plot is smooth and does not have any jagged lines.

```
#NN1
model = Sequential()
model.add(Dense(units=2000, input_dim=2))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=2))

model.compile(loss='mean_squared_error',
          optimizer='rmsprop')

model.fit(x_train, y_train, epochs=75, batch_size=100,
   verbose=1)

#NN2
model2 = Sequential()
model2.add(Dense(units=2000, input_dim=2))
model2.add(Activation('relu'))
model2.add(Dense(units=100))
model2.add(Activation('tanh'))
model2.add(Dense(units=10))
```

```
model2.add(Activation('linear'))
model2.add(Dense(units=2))


model2.compile(loss='mean_squared_error',
          optimizer='adam')

model2.fit(y_train, x_train, epochs=75, batch_size=100,
   verbose=1)

classes = model2.predict(x_test, batch_size=100)#output of DPD
   network
classes = model.predict(classes, batch_size=100)#Compensated
   output
```

7.2.2 Performance

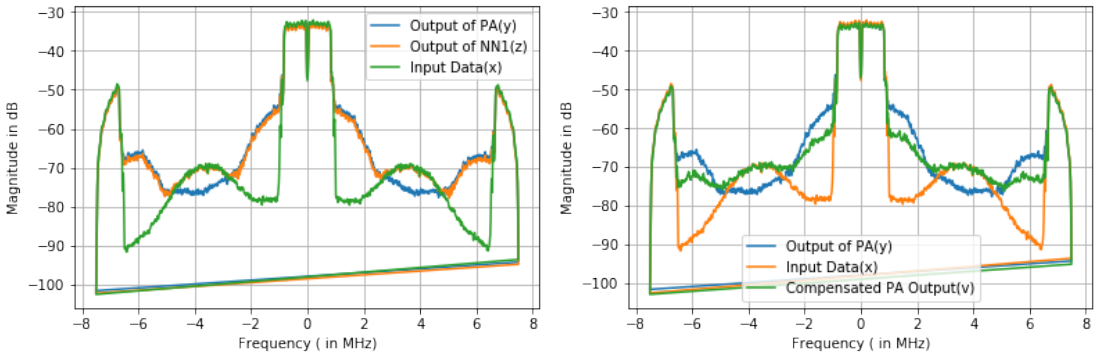
Data Set - 1



(a) Spectral Plot of NN1: PA (b) Spectral Plot of NN2: DPD

Figure 7.2: Performance of DPD with Real World data set 1: Approach1

Data Set - 2



(a) Spectral Plot of NN1: PA (b) Spectral Plot of NN2: DPD

Figure 7.3: Performance of DPD with Real World data set 2: Approach1

In figures 7.2a and 7.3a we see that the NN1 output closely matches the PA output data. In figure 7.2b and 7.3b, the compensate output shows that the spectral regrowth has been suppressed by around 8-10 dB.

7.3 Approach 2

The program flow is same as that in section 6.3.1. The single change is that since the PA model is not available, during the performance evaluation we pass the DPD output to NN1 to generate the compensated output.

7.3.1 Network Metaparameters

Training data: 20,000 samples of the PA input and output data

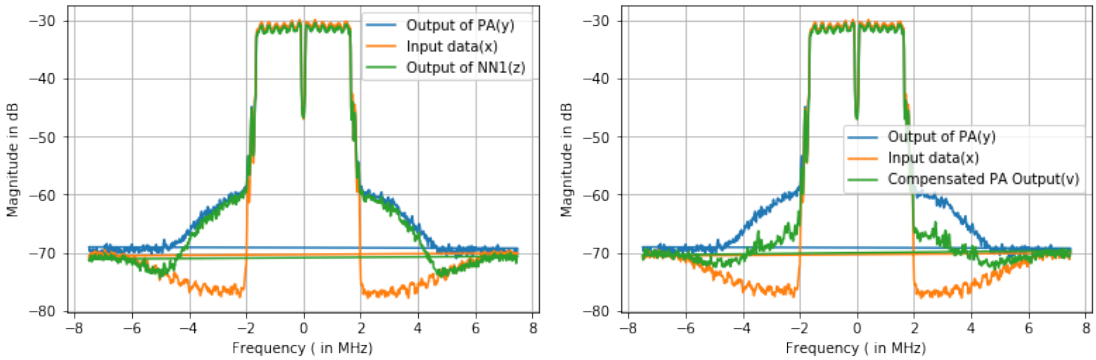
Test data: PA input and output data.

```
#NN1
model = Sequential()
model.add(Dense(units=2000, input_dim=2))
model.add(Activation('relu'))
model.add(Dense(units=100))
model.add(Activation('tanh'))
model.add(Dense(units=10))
model.add(Activation('linear'))
model.add(Dense(units=2))

model.compile(loss='mean_squared_error',
          optimizer='adam')

#Creating a cascade of NN2 and NN1
model.fit(x_train, y_train, epochs=75, batch_size=100,
   verbose=1)
model2 = Sequential()
model2.add(Dense(units=2000, input_dim=2))
model2.add(Activation('relu'))
model2.add(Dense(units=100))
model2.add(Activation('tanh'))
model2.add(Dense(units=10))
model2.add(Activation('linear'))
model2.add(Dense(units=2))

for layer in model.layers[:]:
   layer.trainable = False
   model2.add(layer)

model2.compile(loss='mean_squared_error',
```

```
            optimizer='adam')

model2.fit(x_train, x_train, epochs=75, batch_size=100,
    verbose=1)

#NN2 alone functions as dpd
dpd = Sequential()
for layer in model2.layers[:-7]:
    dpd.add(layer)
```

7.3.2 Performance

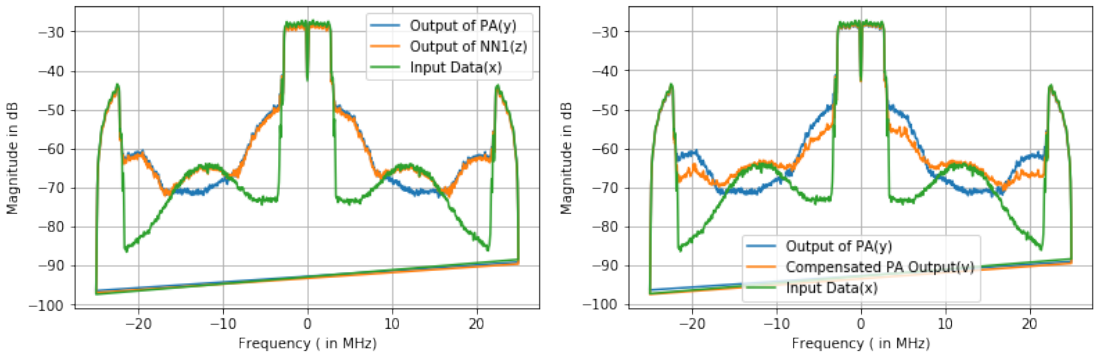
Data Set - 1



(a) Spectral Plot of NN1: PA (b) Spectral Plot of NN2: DPD

Figure 7.4: Performance of DPD with Real World data set 1: Approach2

Data Set - 2



(a) Spectral Plot of NN1: PA (b) Spectral Plot of NN2: DPD

Figure 7.5: Performance of DPD with Real World data set 2: Approach2

In figures 7.4a and 7.5a, we see that the NN1 output closely matches the PA output data. In figures 7.4b and 7.5b, the compensate output shows that the spectral regrowth

has been suppressed by around 10 dB for data set-1 and 5 dB for data set-2.

Chapter 8

Conclusions And Future Scope

In this project, we have examined how to use deep learning for digital pre-distortion of Power Amplifiers. We came up with two different approaches to do the same. First we try this on a preliminary BPSK data and a simple function as a PA model. Then we upgrade to real world QPSK data. The conditions for invertibility of the PA model is also examined. Memoryless and Quasi memoryless PA models are simulated and the spectral suppression is verified. Then we tried using input and output data from a real world PA and saw that the trained neural network is able to regenerate the spectrum (as much as 20dB suppression). Both the approaches work well and can be an effective alternative for conventional DPDs.

To take this project further, we can try implementing this on a real world communication system. The neural network will have to be coded into the transmit chain just before the PA. First, we train the neural network with some random data. After the weights are fixed, we can use the network for DPD by connecting it as a block just before the PA. Also, we are yet to explore how to compensate the memory effect of PAs.

Bibliography

[1] L. Guan and A. Zhu, "Green Communications: Digital Predistortion for Wideband RF Power Amplifiers," in IEEE Microwave Magazine, vol. 15, no. 7, pp. 84-99, Nov.-Dec. 2014.

[2] Scipy - Welch Method:
https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.welch.html

[3] S. Benedetto and E. Biglieri, Principles of Digital Transmission With Wireless Applications. New York: Kluwer, 1999.

[4] How does Keras handle complex numbers?:
https://groups.google.com/forum/!msg/keras-users/Aauby6jMy2I/n341zZ3bFwAJ

[5] Non-linear Models for High Power Amplifiers:
https://upcommons.upc.edu/bitstream/handle/2117/94197/03Hidw03de10.pdf?sequence=3&isAllowed=y

[6] Code and data:
https://drive.google.com/open?id=1F_nqzDuiu3iRHH9dLR8KMhp4RFm_fkR9

[7] Installing Jupyter Notebook:
https://jupyter.readthedocs.io/en/latest/install.html

Appendices

Appendix A

Code Attachments

Links to the code of each simulation alongwith the data used is linked in [6]

For running the code, it is highly recommended to install Jupyter Notebook, refer here for simple instructions to install the same [7]