Neural Network DPD via Backpropagation through a Neural Network Model of the PA

Chance Tarver*, Liwen Jiang, Aryan Sefidi, and Joseph R. Cavallaro§
Department of Electrical and Computer Engineering, Rice University
Houston, TX, USA
Email: *tarver@rice.edu, §cavallar@rice.edu

***Abstract*—We demonstrate digital predistortion (DPD) using a novel, neural-network (NN) method to combat the nonlinearities in power amplifiers (PAs), which limit the power efficiency of mobile devices, increase the error vector magnitude, and cause inadequate spectral containment. DPD is commonly done with polynomial-based methods that use an indirect-learning architecture (ILA) which can be computationally intensive, especially for mobile devices, and overly sensitive to noise. Our approach using NNs avoids the problems associated with ILAs by first training a NN to model the PA then training a predistorter by backpropagating through the PA NN model. The NN DPD effectively learns the unique PA distortions, which may not easily fit a polynomial-based model, and hence may offer a favorable tradeoff between computation overhead and DPD performance. We demonstrate the performance of our NN method using two different power amplifier systems and investigate the complexity tradeoffs.**

***Index Terms*—DPD, neural networks**

I. Introduction

With the deployment of 5G New Radio (NR), the challenges on the physical layer have never been higher. One of these challenges is power amplifier (PA) nonlinearities. Two characteristics of 5G exacerbate this challenge. The 5G waveform is OFDM based and hence suffers from a high peak-to-average power ratio (PAPR) [1]. Moreover, NR supports bandwidths as large as 400 MHz for mmWave[2], which may lead to severe memory effects in the PA. The high PAPR and large bandwidths can degrade the error vector magnitude (EVM) and cause adjacent channel leakage in the frequency domain. Moreover, with massive MIMO, these corrections may need to be done for many antennas, drastically increasing the computational burden on systems.

A common way to correct for nonlinearities in PAs is through digital predistortion (DPD), where the PA nonlinearities are learned so that an inverse can be applied in the digital baseband. Most DPD works rely on some form of memory polynomials (MPs) to create the predistorter [3], [4]. Although their performance can be adequate, the computational complexity needed to achieve such performance can be extreme with generalized memory polynomials (GMPs), sometimes needing hundreds of coefficients.

This work was supported in part by the Rice Office of Undergraduate Research and Inquiry and by the US NSF under grants CNS-1717218 and CNS-1827940 for the "PAWR Platform POWDER-RENEW: A Platform for Open Wireless Data-driven Experimental Research with Massive MIMO Capabilities."

Neural networks (NNs) are well known to be able to learn any arbitrary nonlinear function according to the universal approximation theorem [5]. Considering that DPD is a nonlinear function, it is natural to consider NNs for predistortion, and many works have already applied NNs to DPD [6].

These NN implementations and most other DPD works rely on an indirect learning architecture (ILA) to train the predistorter. The ILA, which was originally designed for NNs [7], was first applied to DPD in [8] and has since been widely used. In an ILA, a postdistorter is used to learn an inverse model, and then the postdistorter is copied to the predistorter. However, learning a postdistorter is known to be susceptible to bias due to noise in the feedback signal and does not necessarily converge to the best possible predistorter [9], [10].

In this paper, we seek to combat the nonlinearities in PAs with a novel predistortion technique that utilizes (NNs) while avoiding the ILA. However, we cannot immediately train a NN based predistorter since the ideal PA input signal is unknown. Instead, we choose to use a NN to model the PA. We then connect a NN for DPD with the NN PA model. Given that we know the ideal PA output, we can then calculate the linearization error of the cascaded NN DPD and PA model and then backpropagate this error through the NNs to update the DPD NN. This DPD NN can then be used in front of the actual PA and achieve performance that rivals MP-based predistorters.

The NN based predistorter also has the advantage of being more flexible than polynomial based methods. For large, GMP-based solutions, the DPD designer will often fit hundreds of coefficients then prune any that have negligible effect. A NN based solution does not have this issue in that it will learn whatever effects contribute most to a specified error function without the DPD designer having to consider all possible causes of distortion in a model explicitly. For example, a simple memory polynomial cannot correct for IQ imbalance. To correct for IQ imbalance, the authors of [11] add a conjugate branch and DC term to the predistorter which significantly increases the model complexity. By the nature of the fully connected NN, the NN can learn such impairments and correct them.

The main contributions of this work are as follows: (1) We propose a novel, NN-based DPD algorithm. (2) We introduce a new NN topology for DPD by including skip connections to form a linear bypass so that the hidden layers focus on

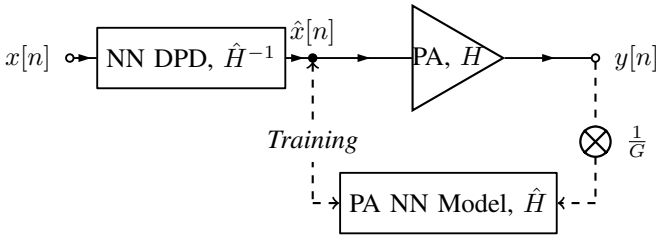

Figure 1. Architecture of the NN DPD system. The DPD NN, $\hat{H}^{-1}$ and PA NN model, $\hat{H}$, have structures resembling Fig. 2. After broadcasting through the PA, the input/output PA data is used to train $\hat{H}$. By backpropagating an error through $\hat{H}$, we train $\hat{H}^{-1}$. This DPD NN is then used with the actual PA. Training and predistortion are done in the digital baseband, so we omit the DAC, up/downconverters, and ADC in this figure.

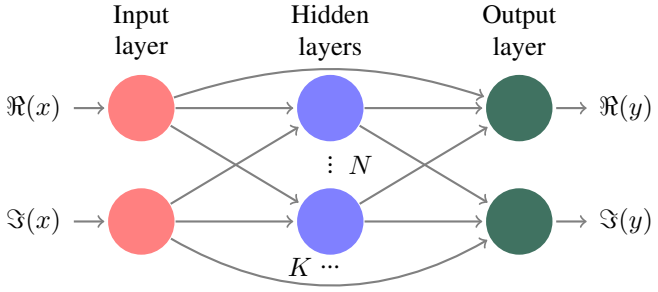

Figure 2. General structure of the NNs with K hidden layers and N neurons per hidden layer. There are always two input and output neurons for the real and imaginary parts of the signal. The inputs are connected to the output neurons via a skip connection so that the hidden layers concentrate on the nonlinear portion of the signal.

the cancellation of nonlinearities. (3) We present a novel training method for a NN controller connected to a black box (in this case a PA). (4) We evaluate the complexity of the proposed algorithms. (5) We test the proposed NN algorithm's predistortion capabilities on multiple real PA platforms. (6) We investigate what the NNs are learning compared to the MPs.

The rest of this paper is organized as follows. In Section II, we overview the full NN DPD system and the architecture of the NNs. We then investigate the complexity of a MP and a NN-based predistorter in Section III. In Section IV we present predistortion results using two PAs. In Section V we begin investigating the features learned by the NNs. We then conclude by discussing possible extensions to this work.

II. Architecture

A. NN DPD System Overview

Consider a complex signal at baseband, $x(n)$, to be broadcast through a PA. Let a PA with discrete time-domain transfer function $H(n)$ have output signal $y(n)$. It is the goal of DPD to find an approximate inverse transfer function of the PA, $\hat{H}^{-1}$, with output $\hat{x}(n)$ as shown in (1), so that the output of the PA is an amplified version of the original input as shown in (2) where G is a scalar representing the gain of the PA.

$$\hat{x}(n) = \hat{H}^{-1}(x(n)) \tag{1}$$

$$y(n) = Gx(n) = H(\hat{x}(n)) \tag{2}$$

In this work, we use a NN to find $\hat{H}^{-1}$ for predistortion. To train a NN, typically example input and output training data are required. However, the ideal $\hat{x}(n)$ is unknown, so we cannot directly train to create a NN for DPD. Many other works overcome this through the ILA. In this work, we avoid the ILA by creating a second NN to model the PA. Given input data, $\hat{x}(n)$, and output data, $\frac{y(n)}{G}$, for any PA, we can train the PA NN model for regression on this data so that it learns an approximate transfer function of the PA, $\hat{H}$.

Once a PA NN Model is created, we freeze its weights and connect the NN DPD to the PA NN Model. We then can use the original input data $x(n)$ as the input and the output training data for this system to calculate an error via a loss function, which can then be backpropagated through $\hat{H}$ to update $\hat{H}^{-1}$. For example, when using a mean square error (MSE) loss function over a window of N samples, the output error, e, is given as

$$e = \sum_{i=1}^{N} \left\| \hat{H}(\hat{H}^{-1}x(i)) - x(i) \right\|^2. \tag{3}$$

Once e is sufficiently close to 0, then we have learned an approximate inverse of the PA. This NN can be used with the actual PA to provide predistortion, as was shown in (1). Fig. 1 illustrates the complete DPD system.

B. NN Design

There are two separate NNs in this project: a DPD NN, which will be used to predistort the actual PA, and a PA NN model, which is used to train the DPD NN. For the sake of simplicity, we keep the design of these NNs the same. However, since the PA NN is only used during the DPD training phase, it can be made arbitrarily complex to improve accuracy in the inverse model.

Since the baseband signal is complex, we design our NNs to have two input neurons and two output neurons: one neuron for the real part of the signal, and one for the imaginary part. We then have at least one fully connected hidden layer with rectified linear unit (ReLU) activation functions. We chose ReLU activation functions due to their low implementation complexity.

Due to the dominant linear component in any PA model or DPD model, we chose to connect the input neurons directly to the output neurons bypassing the hidden layer. By doing so, the expressive power of the NN can be used exclusively for the nonlinear portion of the signal. This skip connection is similar to what is seen in well-performing image classification convolutional neural networks such as Resnet-50. Our NN architecture is illustrated in Fig. 2.

The output of the NN is mathematically provided in the following equations. Here, f is any nonlinear activation function, W_i and b_i are weight matrices and bias vectors corresponding to the ith layer in the NN. The output of the first hidden layer is given by (4). The output of hidden layer i for

$i \in \mathbb{N} : 2 \leq i \leq T$ is given in (5). The final output of the network after hidden layer T is given by (6) where the first element represents the real part of the signal, and the second element represents the imaginary part,

$$h_1(n) = f\left(W_1 \begin{bmatrix} \Re(x) \\ \Im(x) \end{bmatrix} + b_1\right), \tag{4}$$

$$h_i(n) = f\left(W_i h_{i-1}(n) + b_i\right), \tag{5}$$

$$\hat{x}(n) = W_{T+1} h_T + b_{T+1} + W_{\text{linear}} \begin{bmatrix} \Re(x) \\ \Im(x) \end{bmatrix}. \tag{6}$$

In (6), W_{linear} is a 2×2 matrix of the weights corresponding to the linear bypass. In practice, it is expected that it will nearly be the identity matrix, $\mathbb{I}_2$. However, in systems with strong IQ imbalance, this can deviate from that identity matrix.

C. Training

An overview of the training algorithm for the NN DPD system is shown in Algorithm 1. For this algorithm, we use the MSE of the time domain signals as our loss function. The weights of the NNs are updated via backpropagation and any of the standard optimizers, such as stochastic gradient descent or Adam [12]. After training the DPD NN, the procedure can be repeated to further fine-tune the models and improve the performance for a total of D training iterations.

Algorithm 1 DPD Training

$\hat{x}[n] \leftarrow x[n]$
for $i \leq D$ **do**
  $y[n] \leftarrow H\left(\hat{x}[n]\right)$: // PA Broadcast
  $\hat{H} \leftarrow$ Train on $\hat{x}[n], \frac{y[n]}{G}$ // Update PA NN Model
  // Freeze NN weights of $\hat{H}$
  $\hat{H} \leftarrow$ Train on $x[n]$. // Use $\hat{H}^{-1}(\hat{H}(x(n)))$
  $\hat{x}[n] \leftarrow \hat{H}^{-1}\left(x[n]\right)$: // Predistort
end for

D. Software

We use Keras [13] with the TensorFlow backend to design and train the NNs. The software is written in Python and uses the Matlab engine for Python to communicate to WARPLab or RFWebLab for experimental testing with physical PAs.

III. Complexity Considerations

To be able to compare the complexity/performance tradeoffs for the NN with MPs, we develop expressions for the number of parameters and multiplications for each. We only consider the running complexity of the predistorter, which corresponds to inference of the DPD NN.

A. Memory Polynomial

The MP is implemented in (7) with a conjugate branch [11]. The conjugate branch allows for the predistorter to also correct for IQ imbalance. Although this increases complexity, we include a conjugate branch here to make for a more fair comparison with the NN DPD since it is also able to correct this impairment. In (7), α and β are the complex coefficients for the predistorter, $x(n)$ is the original signal to be predistorted, P and Q are the highest degree polynomial considered for the primary and conjugate branches respectively, M and L are the memory depth per polynomial order for the primary and conjugate branches, and c is a DC offset correction term,

$$\begin{aligned} \hat{x}(n) = &\sum_{\substack{p=1,\\ p \text{ odd}}}^{P} \sum_{m=0}^{M} \alpha_{p,m} x(n-m)|x(n-m)|^{p-1} + \\ &\sum_{\substack{q=1,\\ q \text{ odd}}}^{Q} \sum_{l=0}^{L} \beta_{q,l} x^*(n-l)|x^*(n-l)|^{q-1} + c. \end{aligned} \tag{7}$$

The total number of complex parameters is given as

$$n_{\text{poly}} = M\left(\frac{P+1}{2}\right) L\left(\frac{Q+1}{2}\right) + 1. \tag{8}$$

Assuming three real multiplications per complex multiply, we get the following number of multiplies from the complex coefficients multiplied by the basis function generation plus the actual generation of the basis functions,

$$n_{\text{MUL, poly}} = 3n_{\text{poly}} + 3\left(\frac{P-1}{2} - 1\right) + 3\left(\frac{Q-1}{2} - 1\right). \tag{9}$$

B. Neural Network

We implement the NN shown in Fig. 1 with the linear bypass. Let K be the number of hidden layers and let N be the number of neurons per hidden layer. The complexity can then be calculated in terms of number of parameters, n_{NN}; number of multiplies, $n_{\text{MUL, NN}}$; and number of additions, $n_{\text{ADD, NN}}$. The number of parameters in the NN model is equal to the number of weights and biases,

$$n_{\text{NN}} = (K-1)N^2 + (4+K)N + 6. \tag{10}$$

The number of multiplies is equal to the sum of the sizes of the weight matrices for all layers,

$$n_{\text{MULT, NN}} = (K-1)N^2 + 4N + 4. \tag{11}$$

The additions come from the inner products of multiplying each weight matrix by the layer output,

$$n_{\text{ADD, NN}} = (K-1)N^2 + (4-K)N. \tag{12}$$

The number of activation functions, $n_{\text{act. func}}$, is given by the number of hidden layer neurons. Here we use a ReLU activation function which can be implemented with a simple multiplexer,

$$n_{\text{act. func}} = KN. \tag{13}$$

Moreover, feedforward NNs also have a regular structure consisting primarily of matrix-vector multiplications. The regular structure of the NN is attractive for implementation purposes and is easily parallelizable and pipelinable leading to high-throughput, low-latency custom accelerators.

From these equations, we can see that complexity scales quadratically with the number of neurons if there is more than one hidden layer. For a single hidden layer, the complexity grows linearly with the number of neurons. The universal approximation theorem states that a single hidden layer can be used for arbitrary function approximation, so for implementations where complexity may be a significant concern, a single hidden layer may be desirable.

IV. Experimental Results

In this section, we present experimental results for using the NN based DPD. We test on two separate power amplifiers. The first is the RFWebLab system and the second is the PA from the WARPv3 board corresponding to devices that are similar to what would be seen in base stations and user equipments (UEs), respectively.

1) RFWebLab: RFWebLab is a web-connected PA at Chalmers University [14]. This system uses a Cree CGH40006-TB GaN PA with a peak output power of 6 W. Using their Matlab API, we test the NN predistorter. We broadcast a 10 MHz LTE-like OFDM signal through the PA. We train on 10 symbols then validate on 10 different symbols with $N = 16$ and $K = 1$. The Adam optimizer is used with an MSE loss function. ReLU activation functions are used at the hidden layer neurons.

In Fig. 3, we show the MSE training loss for the NNs. The minimum MSE achieved by the PA NN model was 0.0004. The residual error in the NN model can be attributed to memory effects that the current NN architecture cannot account for (adding recurrent neural networks (RNNs) to improve this is left for future work) and noise in the data. The DPD NN then learns how to predistort for the PA NN model. For the learned PA model, the DPD NN is nearly able to learn a perfect inverse with a minimum MSE achieved being 4.3×10^{-6}. Although the MSE for the DPD NN is near zero, it is important to remember that this is only a training loss for predistorting against the PA NN model and not the actual PA.

We then use the DPD NN with the actual power amplifier to measure the frequency domain result shown in Fig. 4. This result is shown after the training procedure shown in Fig. 3. For the result, the RMS PA output of RFWeblab is set to -24 dBm where it is in saturation. The PAPR of the OFDM signal used for training is 10.24 dB. We can see significant spectral regrowth around the main carrier when DPD is not used shown by the red curve. After applying DPD, the spectral regrowth is significantly reduced shown by the green curve.

2) WARPv3: The WARPv3 SDR is a complete wireless prototyping system developed at Rice University [15]. It includes an Anadigics AWL6951 dual-band power amplifier, which is designed for a maximum output power of 23 dBm. For this platform, we use a coaxial cable with a 30dB attenuator to connect the TX output port to the RX port.

We perform training on the DPD NN similarly to the previous section. In Fig. 5 we show a result after training with this PA. Since the WARPv3 board is limited to a 40 MHz sampling rate, we use a 3 MHz OFDM signal to demonstrate the performance. The NN is set to $N = 10$ and $K = 2$. This result shows another example of the NN-DPD suppressing the adjacent channel leakage.

V. Neural Network vs. Memory Polynomial

In this section, we compare the outputs of the NN and MP DPDs to understand better what the NN does differently. Here, we focus on RFWeblab. We train a NN DPD with $K = 1; N = 20$ and a MP DPD with $P = 9; M = 1$. After training, the NN DPD has a better adjacent channel leakage ratio (ACLR) than the MP DPD, as can be seen in Fig. 4, and we seek to understand the reason. To do so, we create a test vector to broadcast through the DPDs with inputs over the complex plane from $-1-1j$ to $1+1j$. We plot the difference between the DPD outputs compared to the input for each input point in Figs. 6 and 7. The plotted color shows the change in the input magnitude of the signal after predistortion for the corresponding complex input sample.

For both Figs. 6 and 7, the red color indicates that the DPDs increased the magnitude of the corresponding input sample. We see that over the majority of the IQ plane, the magnitude of the input samples was increased. This increase is to be expected in that DPD is primarily used to overcome compression in the PA outputs due to saturation. In Fig. 6, we see that the MP DPD is phase invariant while in Fig. 7, the NN DPD effect varies with the phase.

The difference between the MP and NN DPDs are highlighted in Fig. 8. Here we can see that the NN learned to apply DPD differently than the MP. For positive real values, the NN is applying more expansion of the input samples compared to the MP. For negative real values, the NN is applying less expansion than the MP. This effect is persistent across multiple runs. Although this effect is still under investigation, we see that the flexibility to apply DPD differently depending on the phase of the input allows the NN to outperform MP based solutions.

VI. Conclusions

In this paper, we show a novel NN structure and training algorithm for digital predistortion. The complexity was derived in terms of the number of multiplies and parameters for the NN and a comparable MP implementation. We experimentally validated the DPD method with two separate power amplifiers. In future work, we will extend the NN to include memory effects via RNNs and investigate in more detail the specific physical features that the NNs are learning. We will also examine the complexity–performance tradeoffs for the NNs and compare this to the MP-based DPD by measuring the ACLR to examine performance per the number of multiplies.

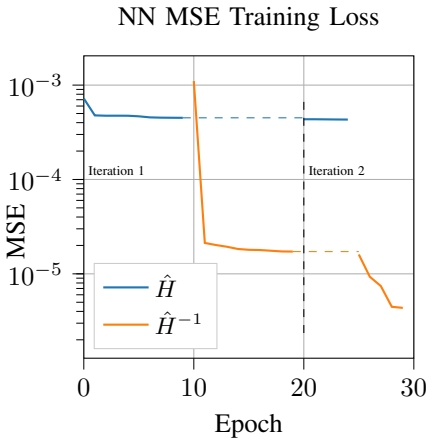

Figure 3. DPD training loss of the NNs for Algorithm 1 using 2 iterations. The PA NN model, $\hat{H}$ is trained based on PA input/output data. Using this, we train the NN DPD, $\hat{H}^{-1}$. This process is repeated as needed.

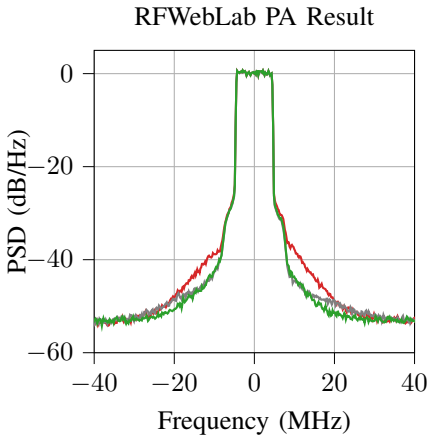

Figure 4. RFWebLab PA output with and without DPD shown in the frequency domain. The red curve shows the PA output without DPD, and the green curve shows the PA output on the same signal when applying the NN DPD. The gray curve shows MP DPD with $P = 9; M = 1$.

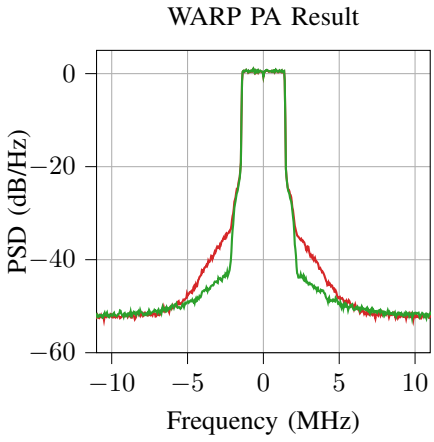

Figure 5. WARPv3 PA output with and without DPD shown in the frequency domain. The red curve shows the PA output without DPD and the green curve shows the PA output on the same signal when appying the NN DPD.

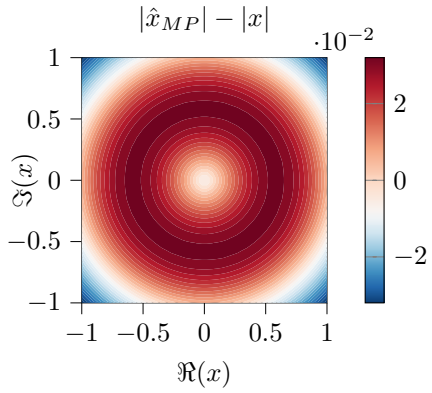

Figure 6. Output of MP DPD compared to the input over the space of IQ input samples.

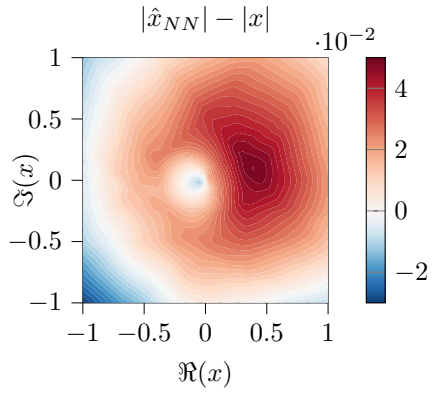

Figure 7. Output of NN DPD compared to the input over the space of IQ input samples.

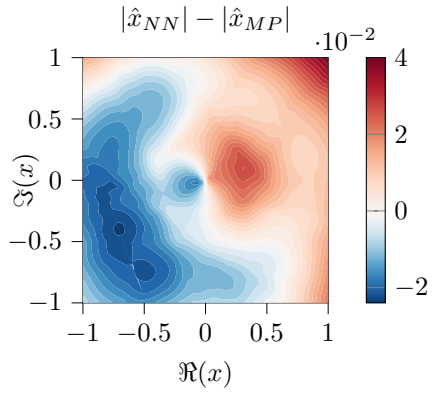

Figure 8. Difference between NN and MP DPDs for input samples over the IQ plane.

References

[1] S. H. Han and J. H. Lee, "An overview of peak-to-average power ratio reduction techniques for multicarrier transmission," *IEEE Wireless Commun.*, vol. 12, no. 2, pp. 56–65, April 2005.

[2] 3GPP, "NR; base station (BS) radio transmission and reception," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.104, 04 2019, version 15.5.0. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3202

[3] L. Ding *et al.*, "A robust digital baseband predistorter constructed using memory polynomials," *IEEE Trans. Commun*, vol. 52, no. 1, pp. 159–165, Jan 2004.

[4] P. L. Gilabert, G. Montoro, D. Vegas, N. Ruiz, and J. A. Garcia, "Digital predistorters go multidimensional: DPD for concurrent multi-band envelope tracking and outphasing power amplifiers," *IEEE Microw. Magazine*, vol. 20, no. 5, pp. 50–61, May 2019.

[5] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190009T

[6] N. Benvenuto, F. Piazza, and A. Uncini, "A neural network approach to data predistortion with memory in digital radio systems," in *Proc. 1993 IEEE Int. Conf. on Commun.*, vol. 1, May 1993, pp. 232–236 vol.1.

[7] D. Psaltis, A. Sideris, and A. A. Yamamura, "A multilayered neural network controller," *IEEE Control Syst. Mag.*, vol. 8, no. 2, pp. 17–21, April 1988.

[8] C. Eun and E. J. Powers, "A new volterra predistorter based on the indirect learning architecture," *IEEE Trans. on Signal Process.*, vol. 45, no. 1, pp. 223–227, Jan 1997.

[9] D. R. Morgan, Z. Ma, and L. Ding, "Reducing measurement noise effects in digital predistortion of RF power amplifiers," in *Proc. 2003 IEEE Int. Conf. on Commun.*, vol. 4, May 2003, pp. 2436–2439 vol.4.

[10] H. Paaso and A. Mammela, "Comparison of Direct Learning and Indirect Learning Predistortion Architectures," in *2008 IEEE Int. Symp. on Wireless Commun. Syst.*, Oct 2008, pp. 309–313.

[11] L. Anttila, P. Handel, and M. Valkama, "Joint mitigation of power amplifier and I/Q modulator impairments in broadband direct-conversion transmitters," *IEEE Trans. Microw. Theory Tech.*, vol. 58, no. 4, pp. 730–739, April 2010.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[13] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[14] "RF WebLab." [Online]. Available: http://dpdcompetition.com/rfweblab/

[15] "WARP Project." [Online]. Available: http://warpproject.org