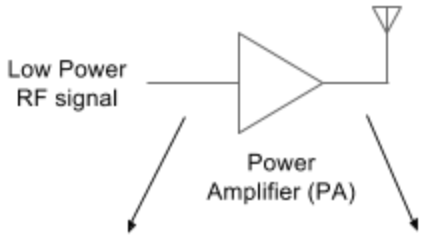
Digital Predistortion Using Machine Learning Algorithms

CS229 ; James Peroulas ; james@peroulas.com

Introduction



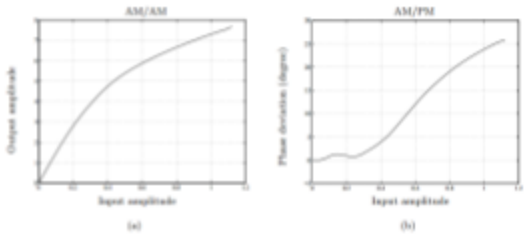
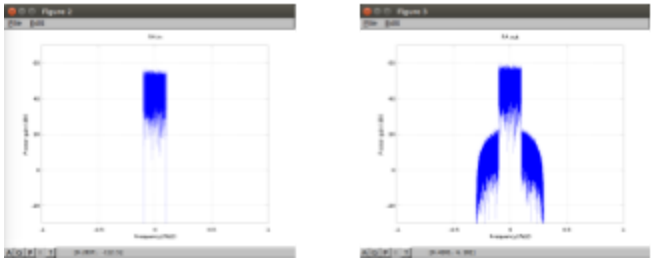


Figure 2: The AM/AM and AM/PM responses of a Class AB power amplifier.

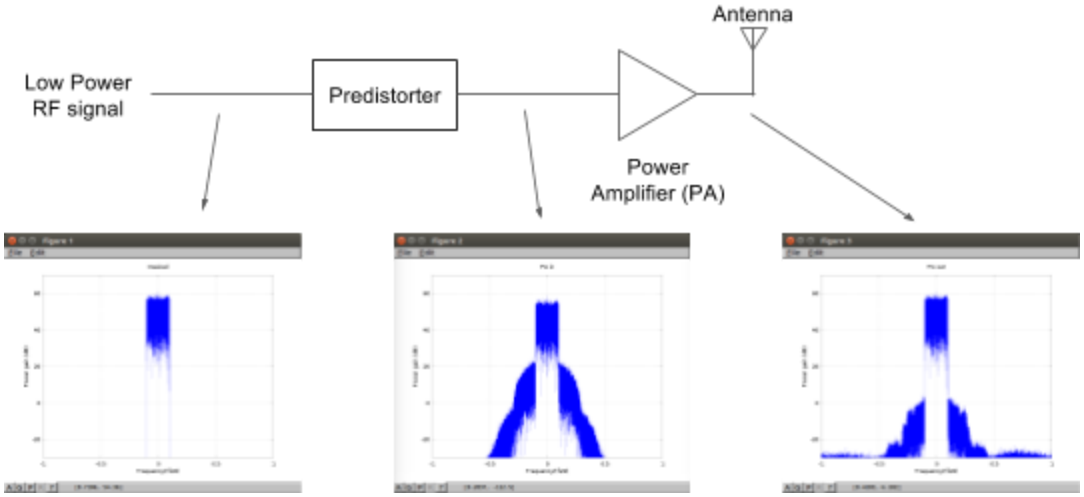


Wireless communications transmitter

The power amplifier (PA) is the last stage of a wireless communications transmitter and takes a carefully crafted low power signal and amplifies it to a sufficient level for transmission. Power amplifiers are large, expensive, and inefficient devices that have input-output curves similar to those on the right side of the figure above. If the input to the power amplifier is a sinewave with magnitude m and phase p, the output of the power amplifier will be another sinewave with magnitude AMAM(m) and phase p+AMPM(m) where AMAM(m) and AMPM(m) are the top left and right (resp.) plots above.

Historically, power amplifiers are operated in their linear region which, for the above amplifier, would be from input amplitudes from 0 to about 0.2. This, however, results in a very inefficient power hungry power amplifier.

Power amplifiers become more efficient the deeper into the nonlinear region that one drives them. This, however, distorts the carefully crafted low power RF signal and causes it to spew emissions into adjacent bands. The spectral plot on the lower left side of the above figure shows an example signal coming into the PA that is well confined and only contains energy in the transmission band. A PA driven very hard will produce a signal such as the spectral plot to the side where one sees significant emissions in the adjacent channels.



Modern wireless communications transmitter

A modern wireless transmitter adds a nonlinear predistortion block to the chain which takes the clean signal and produces a dirty signal which, when fed into the PA, causes the PA to produce a clean(er) output signal. Thus, the PA can be driven harder to (1) produce RF power and (2) waste less energy as heat.

It should be noted here that performance, in a predistortion context, is measured as the amount of power that leaks into adjacent bands $P_{leakage}$, compared to the amount of power of the desired signal $P_{desired}$. This is called the Adjacent Channel Leakage Ratio and is usually expressed in dB:

$$ACLR = 10 * log_{10}\left(P_{desired}/P_{leakage}\right) \text{ dB}$$

Digital predistortion is an iterative process in that first, a non-predistorted signal is applied to the PA and the output is observed to determine how to predistort the signal. This inevitably leads to a new signal that drives the PA harder than before and maps the PA's response for progressively higher and higher input drive levels. It typically takes 3-4 DPD iterations to reach the final performance.

Note that machine learning tries to predict the performance of the PA. However, performance is evaluated with a different metric: ACLR. This heuristic works in practice in that the better we can model the PA, the better ACLR we find.

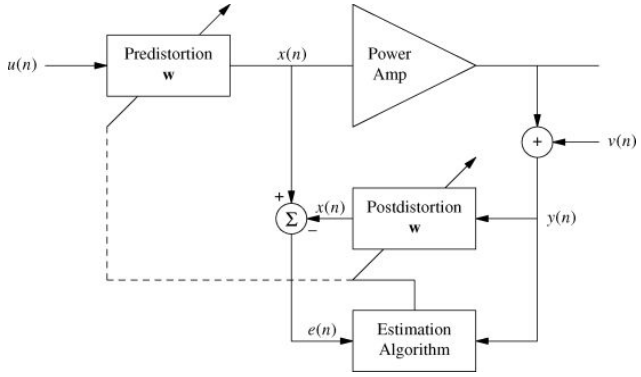
This project uses machine learning techniques to determine how to predistort the signal going to the PA to produce better performance at a lower cost. As will be explained later, the input to the ML model during training is actually the time series of *outputs* of the PA and the output of the model, during training, is a prediction of what the PA *input* was.

The ML algorithms investigated are linear regression, regularization, model selection, principal component analysis, and gradient descent directly on the cost function (explained later).

Related Work

One of the early papers on predistortion is by [Saleh] who created an AM-AM and AM-PM model of traveling wave tube (TWT) amplifiers. This was a memoryless model with only 4 parameters that, at the time, appropriately modeled TWT PA's. The paper by [Cavers] generalized this technique to model and predistort more general power amplifiers.

However, transmit bandwidths and power levels kept increasing and it became clear that memoryless predistortion was not sufficient to model PA behavior [Bsch] [Sevic] [Lajoinie]. Memory had to be considered.



Indirect Learning Architecture [Eun]

Instead of trying to invert a nonlinear PA model, [Eun] came up with the idea of solving directly for the predistortion function. Eun trained the model on the PA's actual output and produced a prediction of the input sample that caused the observed output on the PA. Thus, once the model was trained, it could be used directly as a predistorter simply by copying it over, as seen in the figure above.

[Lei] used the indirect learning architecture in combination with the "memory polynomial" to create one of the first modern digital predistortion systems that took into account the memory effects of the PA. His proposal used many polynomial terms which led to high implementation costs and stability issues. Furthermore, it required large numbers of samples to be captured going into the PA and also coming out of the PA in a so-called capture buffer. Another issue is that the memory polynomial optimizes PA modelling. It does not try to optimize ACLR directly.

Dataset and Features

For the problem at hand, data is not lacking. Samples are continuously being fed into a PA and it's possible to capture as much input/ output data as desired.

The work in this project used a Matlab model of a PA with memory created by [Ku].

The memory polynomial model of the predistorter is as follows:

$$y(n) = \sum_{d=0}^{D-1} \sum_{p=0}^{P-1} c_{d,p} |x(n-d)|^p x(n-d)$$

Where x(n) is the actual PA output and y(n) is the predicted PA input. Conceptually, y(n) is a sum of nonlinear functions of x(n-d) for delay values between 0 and D-1. Each nonlinear function is a sum of P polynomial terms from 0 to P-1. Thus, the D attributes of the model are $x(n-d)$ and the D*P features are $|x(n-d)|^p x(n-d)$.

Methods/ Experiments/ Results/ Discussion

Basic DPD

Let T represent the training size. T+D-1 PA output samples x(n) were captured along with T PA input y(n) samples:

$$x_v(n) = \left[x(n) \ |x(n)|^1 x(n) \ \ldots \ |x(n)|^{P-1} x(n) \ \ldots \ x(n-D+1) \ |x(n-D+1)|^1 x(n-D+1) \ \ldots \ |x(n-D+1)|^{P-1} x(n-D+1) \right]$$

$$X = [x_v^T(0) \ x_v^T(1) \ \ldots \ x_v^T(T-1)]^T$$

$$Y = [y(0) \ y(1) \ \ldots \ y(T-1)]^T$$

The model is represented by:

$$Y = X\theta + n$$

Where n is a noise vector. The initial investigations used the normal equations to solve the linear regression directly:

$$\hat{\theta} = \left(X^H X \right)^{-1} X^H Y$$

Multiple iterations were performed and this document will use integer subscripts to represent the iteration number. For example, for the first iteration, the pre-distorter was initialized with $\hat{\theta}_0 = [1 \ 0 \ 0 \ \ldots]$, i.e. as a passthrough device. The data captured while $\hat{\theta}_0$ was used for the predistorter function produced X_1 and Y_1 which was used to create $\hat{\theta}_1$ using the normal equations above. This continued for 5 iterations to finally produce $\hat{\theta}_5$. For all of the training iterations, the training data was randomly generated but contained roughly T samples. Performance was evaluated by creating a new, longer sequence of 100,000 samples, predistorting it using $\hat{\theta}_5$, and measuring ACLR.

As is typically done, the parameters D and P were varied so as to maximize performance which reached approximately 60 dB. The training set size was manually tuned to be 10000 and the final values were 3 and 11 respectively (33 features total). The problem was that although performance was good, it was not stable and the solution ended up bouncing around between 60 and as bad as 50 dB.

Regularization

It was theorized that the matrix being inverted was not fully rank 33 and regularization was attempted by modifying the normal equations:

$$\hat{\theta} = \left(X^H X + \lambda I \right)^{-1} X^H Y$$

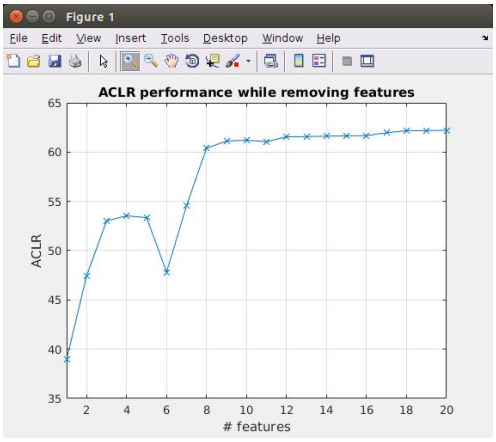
It was found that a λ value of 10^-10 produced stable performance at 60 db.

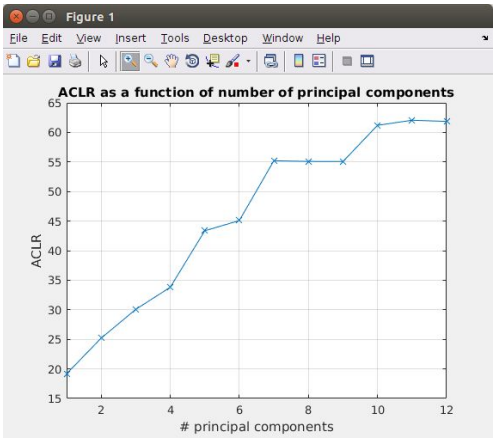
Model Selection

A formal approach to parameter selection was then tried where T, D, and P were systematically varied from 10^2, 10^3, 10^4, 10^5 (T), 1..9(D) and 1..15(P). For each T, D, P combination, λ was tuned, using cross-validation. Predistortion was trained using a training set of size T and would converge after 5 iterations. Then, a new, validation data set was created that was 10^6 samples long and performance evaluated. λ was reduced by a factor of 10 until the cross validation performance did not increase at which point it was assumed that the bias of the model was minimized.

It was found that 62 dB of performance could be achieved for [T, D, P, λ] = [10000, 4, 5, 0.01] respectively. 2dB better performance was achieved using only 20 features instead of 33!

Feature Sensitivity Analysis





Feature Sensitivity and Principal Component Analysis

The features were systematically reduced from 20 to 1 to examine how sensitive performance was to any one particular feature. As can be seen in the left figure above, several components had only a tiny incremental effect on performance. A total of 8 terms were removed and 62 dB performance was maintained. Only 12 features remained.

Principal Component Analysis

Next, principal component analysis was performed. For every DPD iteration, the 12 remaining features were normalized to have variance 1 producing X_N. Columns of X are zero mean:

$$X_N = XD = X * diag\left(\left[\sigma_1^{-1}\ \sigma_2^{-1}\ \ldots\ \sigma_{12}^{-1}\right]\right)$$

Where σ_n is the measured standard deviation of column n of X.

Singular value decomposition was used to decompose X_N:

$$X_N = USV^H$$

The first k columns of V were used to reduce the feature space from 12 to k and performance was measured to better understand the true dimension of the feature space. Specifically, let the notation V_k represent a matrix containing only the first k columns of V corresponding to the k largest singular values of X_N. Then, $\widehat{\theta}$, a vector of length k, was estimated as:

$$\widehat{\theta} = \left(V_k^H D X^H X D V_k + \lambda I\right)^{-1} V_k^H D X^H Y$$

The figure above and to the right shows the performance as a function of the number of principal components that were retained. As can be seen, although 12 columns were in X, they could be reduced down to 10 columns with minimal impact on performance. If 55 dB of performance was acceptable, we could reduce all the way down to 7 principal components.

The implementation cost of linear regression has been reduced from 33 terms to 10 while improving performance!

Gradient Descent

Up to now, the machine learning algorithms were not optimizing the cost function directly. Now, the PCA solution, with 10 columns, was modified to optimize directly for the cost function, ACLR, after converting from dB to linear values:

$$J(\theta) = 10^{ACLR(\theta)/10}$$

The update equation then becomes:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

The problem is in finding $\frac{\partial}{\partial \theta_j} J(\theta)$. $J(\theta)$ is the result of sending a random signal into the predistorter (parameterized by θ), observing the PA output, filtering by one filter that is used to measure the power of the desired signal, and filtering by another filter used to measure the adjacent bands, calculating the power of the signal coming out of these filters, and then dividing these values. This is not something that can be solved for in closed form.

Instead, $\frac{\partial}{\partial \theta_j} J(\theta)$ was estimated empirically. Specifically, j was chosen randomly and θ_j was randomly perturbed:

$\theta_j := \theta_j + 0.0001 e^{i2\pi r}$ where r is a uniformly distributed number between 0 and 1

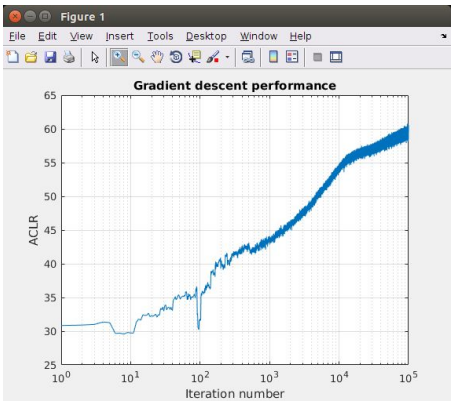
Then $J(\theta)$ was compared to $J(\theta_p)$ where θ_p represents θ after θ_j has been perturbed, to estimate $\frac{\partial}{\partial \theta_j} J(\theta)$:

$$\frac{\partial}{\partial \theta_j} J(\theta) \approx \frac{J(\theta_p) - J(\theta)}{0.0001}$$

It was found that α needed to decrease as the measured $J(\theta)$ decreased so that performance could get better and better. The final update equation was:

$$\theta_j := \theta_j - 0.02J(\theta)\left(J(\theta_p) - J(\theta)\right)e^{i2\pi r}/0.0001 = \theta_j - 200\left(J(\theta)J(\theta_p) - J^2(\theta)\right)e^{i2\pi r}$$

The coefficient 0.02 was chosen by trial and error to maximize convergence speed while minimizing instability.



Gradient Descent Performance

Gradient descent did converge and at 60 dB ACLR it did approach the performance using the normal equations, but convergence was extremely slow. As seen in the above figure, it took 100,000 iterations to reach 60 dB. This algorithm is computationally very fast, much faster than the normal equations, but 100,000 is a large number.

Conclusion

This report shows that ML techniques can be used to improve the implementation cost of using the memory polynomial for predistortion. PCA is the big winner that managed to reduce the number of features from 33 to only 10. This report also showed that it's possible to perform DPD without using a cumbersome capture buffer by simply continuously measuring ACLR. Convergence was slow but one single iteration of gradient descent executes much more quickly than an iteration of the normal equations.

Future investigations can examine ways to improve convergence speed of the gradient descent algorithm. Its simplicity of implementation is appealing, but the slow convergence speed limits its use. One possible explanation is that the estimate of the gradient is so noisy that too small of a stepsize needs to be used.

References

[Saleh] A. A. M. Saleh, J. Salz, "Adaptive linearization of power amplifiers in digital radio systems", Bell Syst. Tech. J., vol. 62, pp. 1019-1033, Apr. 1983.

[Cavers] J. K. Cavers, "Amplifier linearization using a digital predistorter with fast adaptation and low memory requirements", *IEEE Trans. Veh. Technol.*, vol. 39, pp. 374-382, Nov. 1990.

[Bsch] W. Bsch, G. Gatti, "Measurement and simulation of memory effects in predistortion linearizers", *IEEE Trans. Microw. Theory Tech.*, vol. 37, pp. 1885-1890, Dec. 1989.

[Sevic] J. F. Sevic, K. L. Burger, M. B. Steer, "A novel envelope-termination load-pull method for ACPR optimization of RF/microwave power amplifiers", *Proc. Microwave Symp. Dig.*, pp. 723-726, 1998.

[Lajoinie] J. Lajoinie, E. Ngoya, D. Barataud, J. M. Nebus, J. Sombrin, B. Rivierre, "Efficient simulation of NPR for the optimum design of satellite transponders SSPAs", *Proc. Microwave Symp. Dig.*, pp. 741-744, 1998.

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

[Lei] Lei Ding, G.T. Zhou, D.R. Morgan, Zhengxiang Ma, J.S. Kenney, Jaehyeong Kim, C.R. Giardina, “A robust digital baseband predistorter constructed using memory polynomials”, *IEEE Transactions on Communications*, Volume: 52, Issue: 1, Jan. 2004

[Ku] Hyunchul Ku; J. S. Kenney, “Behavioral modeling of nonlinear RF power amplifiers considering memory effects”, Year: 2003, Volume: 51, Issue: 12, Pages: 2495 - 2504