Digital Predistortion Using Machine Learning Algorithms

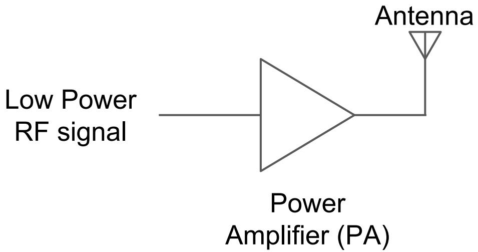
Stanford CS229

2016-12-12

James Peroulas

james@peroulas.com

Digital Predistortion (DPD) - Introduction



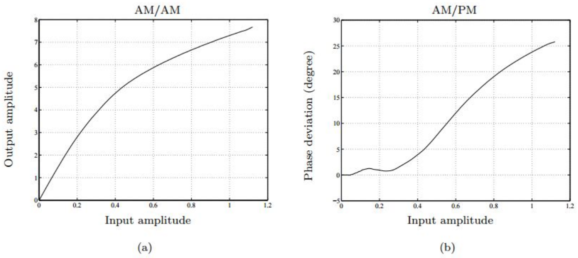
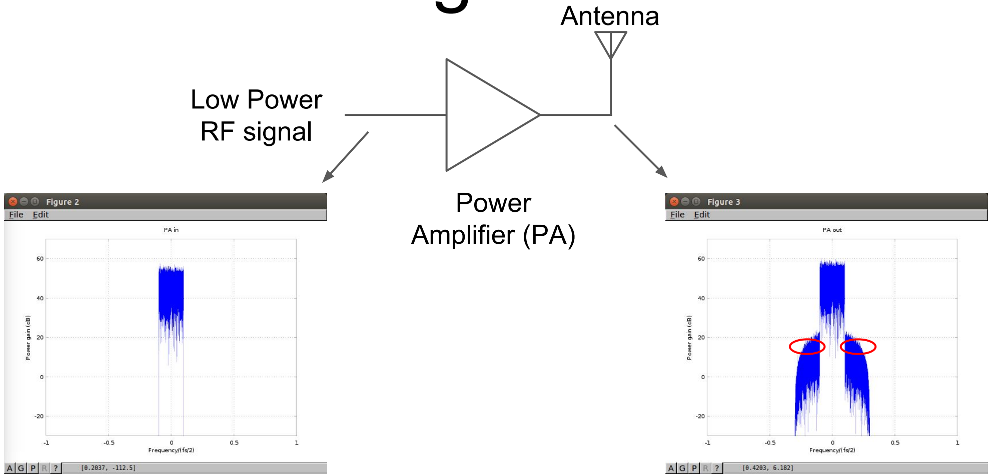


Figure 2: The AM/AM and AM/PM responses of a Class AB power amplifier.

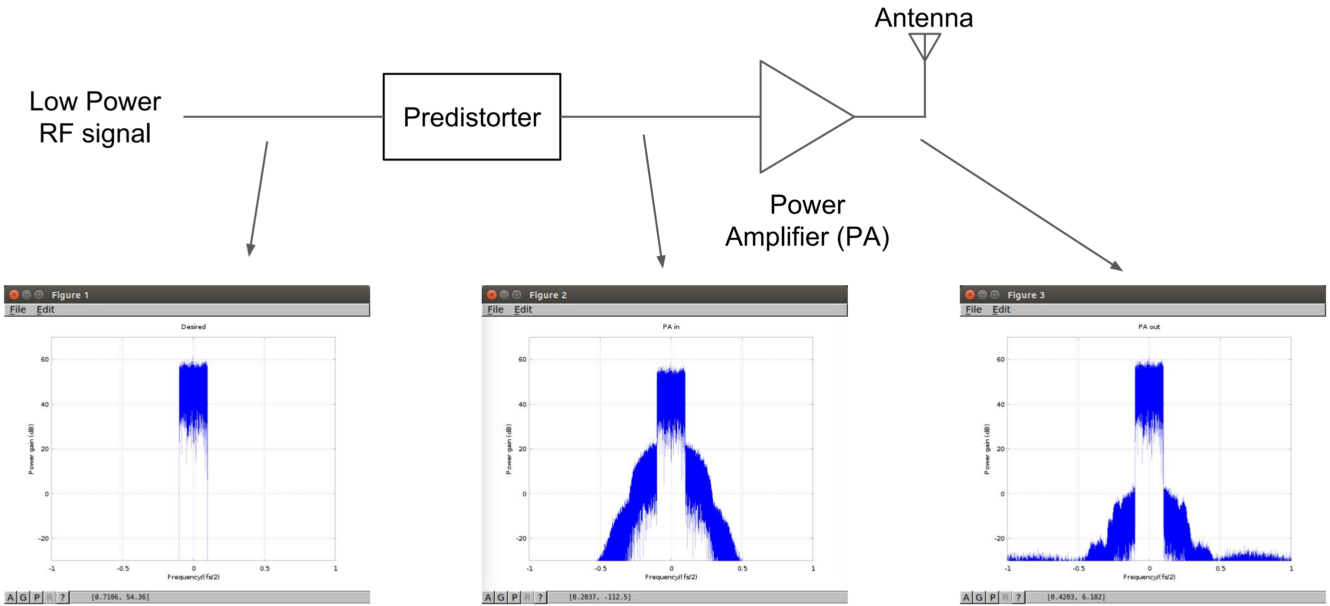
- PA is a non-linear device where output y is not simply equal to k * input x.
  - Ex: if |x|=0.4, output will be $4.7*e^{j*(4+\square x)}$
- Historically, we only used the PA in its linear region. Ex: 0<=|x|<=0.18
  - PA efficiency is very low: ~15%
- Baseband model of PA: input samples and output samples are complex values.
  - PA output is a function of current input and all past inputs!

PA driven into nonlinear region



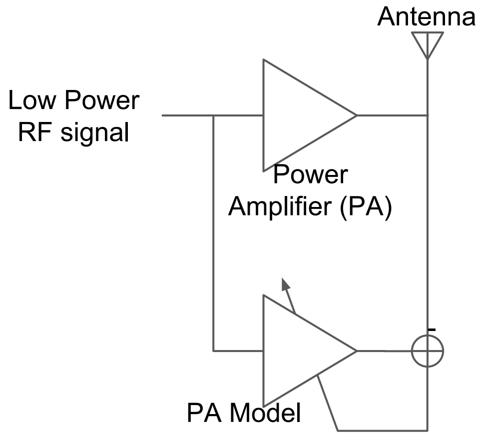
- The left power spectrum is our desired emissions and only occupies the bandwidth we are allowed to use.
- Driving a PA hard will use the PA efficiently but will produce emissions outside our assigned frequency range.
    - Examine the skirts around the desired signal in the right figure (highlighted in red)
- Performance is assessed by comparing the energy outside the desired band to the energy inside the band.
    - ACLR = ratio of desired power to power in the adjacent channel. ACLR=39dB above.

DPD



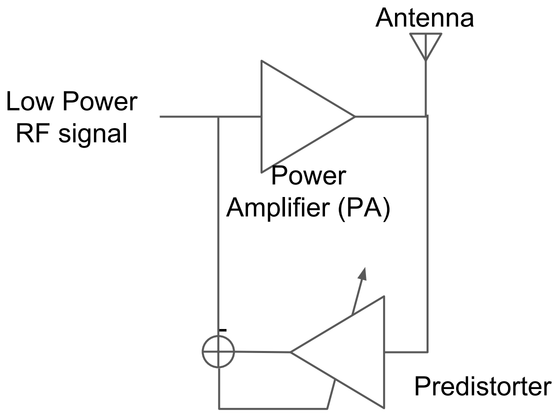
- Predistorter is a nonlinear function that produces an ugly signal (middle power spectrum) which, when fed to the PA, causes the PA to produce a clean(er) signal on its output.
- In the above case, ACLR went from 39dB (previous slide) to 60db! More than 100x less power is being leaked into the adjacent bands.

PA modelling



- Use ML to train PA model so that model output matches actual PA output
- NOT useful for predistortion because even with a perfect PA model, you now have to invert it.
    - Inverting even a “simple” PA model containing 3 memoryless terms is non-trivial and requires much more than 3 terms.
- PA modelling is <u>not</u> investigated in this project

DPD: Indirect Learning Architecture



- Directly learn the predistortion function
  - No need to invert a non-linear model
- Based on the PA outputs y(n), y(n-1), y(n-2), …, predict the value of x(n) that is currently on the input to the PA.
- When used as a predistorter, assume y(n-k)~=desired_signal(n-k).
  - Thus, based on desired_signal(n), desired_signal(n-1), desired_signal(n-2), ..., predict the value of x(n) we should use as input to the PA to cause the output of the PA, y(n) to be the same as desired_signal(n).
- Only this architecture is investigated in this project.

PA model used in this project

$y(n)=$

$c_{00}x(n-d_0)+c_{01}|x(n-d_0)|^2x(n-d_0)+$

$c_{10}x(n-d_1)+c_{11}|x(n-d_1)|^2x(n-d_1)+$

$c_{20}x(n-d_2)+c_{21}|x(n-d_2)|^2x(n-d_2)+$

$c_{30}x(n-d_3)+c_{31}|x(n-d_3)|^2x(n-d_3)+$

$c_{40}x(n-d_4)+c_{41}|x(n-d_4)|^2x(n-d_4)$

```
d=[0 1 3 37 98];

c=
[ 1.9832+0.2129j -0.6169-0.1214j
 -0.9748-0.0244j  0.2553+0.0614j
  0.1630-0.0045j -0.0211-0.0089j
 -0.0106-0.0022j  0.0244+0.0142j
  0.0065-0.0014j  0.0038+0.0144j];
```

- Note that a ‘simple’ PA model does not produce a simple predistorter.
- Ku, et al: https://smartech.gatech.edu/bitstream/handle/1853/5327/ku_hyunchul_200312_phd.pdf?sequence=1

Desired signal

- Desired signal is a bandlimited white noise signal.
- Trivial to create new signals and thus, not limited by training set, validation set, or test set size.

Predistortion Model

- Memory polynomial:
  - x(n)=sum(d=0,d=D,sum(p=0,p=P,$c_{d,p}|y(n-d)|^p y(n-d)$));
    - x(n) is the PA input (predistorter output)
    - y(n) is the desired PA output (predistorter input and, (hopefully) also the PA output)

- This is a linear regression where the single attribute is y(n) and the features are $|y(n-d)|^k y(n)$ for various d and k.

- D and P are parameters to the model
- First step, choose D and P
  - Search using wrapper where performance measurement is both ACLR and stability. Stability is measured by whether future iterations of the DPD algorithm stay at the same solution.

Hand tuned parameters

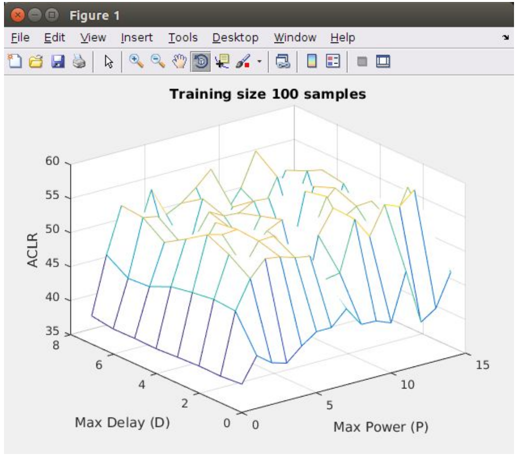
- D, P, training set size, and regularization parameter lambda were hand tuned
- Final performance:
  - D=2
  - P=10
  - (D+1)*(P+1)=33 features
  - training_length=1000
  - lambda=10^{-10}
  - Resulting ACLR on both training set and cross validation set: 60 dB

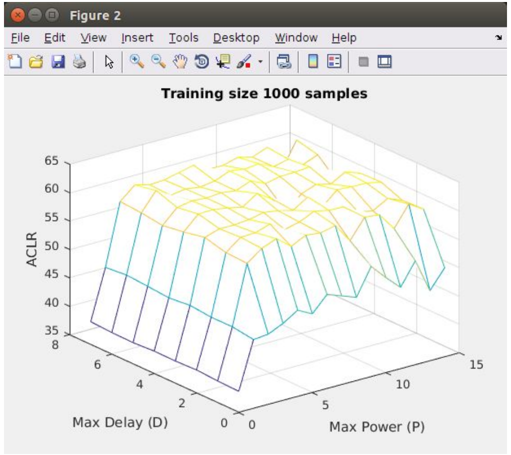
Model Selection Algorithm

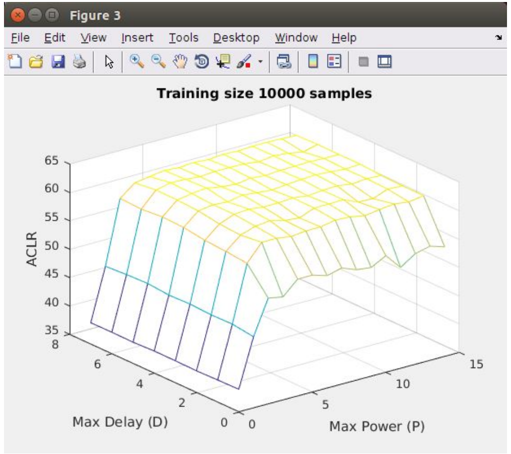
- Wrapper method.
- Try D values from 0 to 8 and P values from 0 to 14
- Vary training set size: 100, 1000, 10000, 100000
- Cross validation set always 100,000 samples.
- For each D, P, and training set size:
  - start with lambda=0.1
  - Keep decreasing lambda by a factor of 10 until performance (ACLR) no longer improves

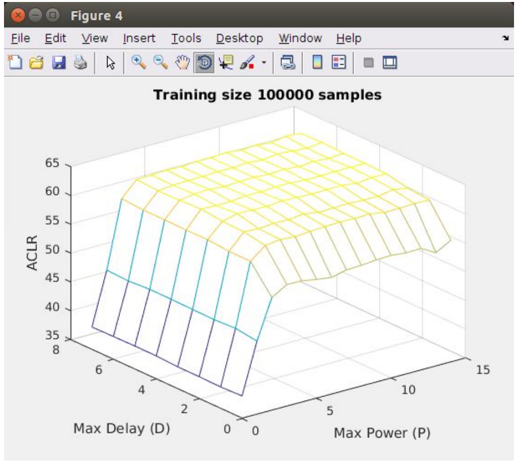
Measured results

- Best results:
- Training set size: 10000
- D=3
- P=4
- Lambda=0.01
- (D+1)*(P+1)=20 features required for near optimal performance
- 62 dB ACLR performance



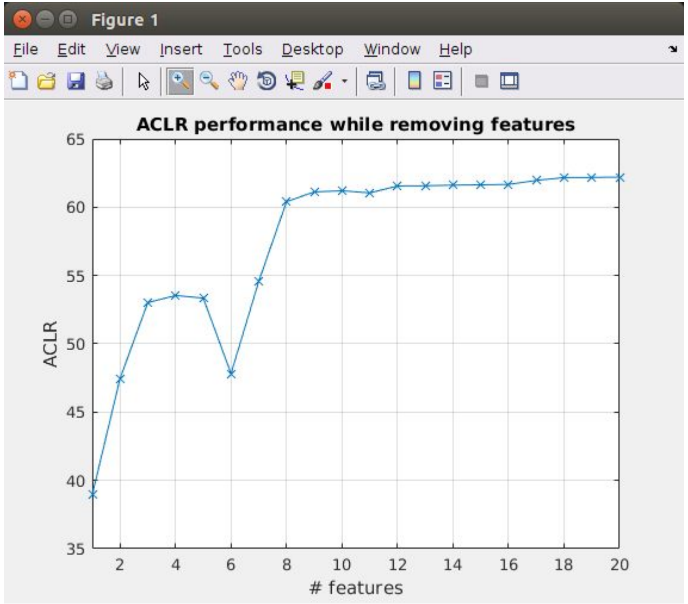






Feature sensitivity

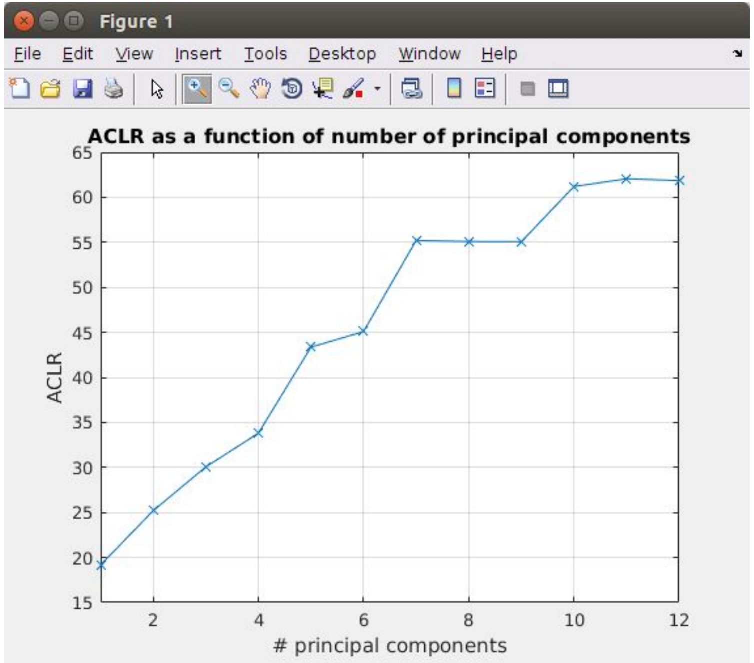
d	p	DifferentialImprovement
0	0	0.00
0	1	8.51
0	2	5.56
0	3	0.52
0	4	-0.19
1	0	-5.53
1	1	6.76
1	2	5.83
1	3	0.71
1	4	0.08
2	0	-0.16
2	1	0.51
2	2	0.02
2	3	0.05
2	4	0.02
3	0	0.03
3	1	0.29
3	2	0.21
3	3	0.01
3	4	0.02



- Notice that performance gets worse in going from 5 to 6 terms
- Removed features that did not affect performance significantly (colored in red)
- Resulted in 41 dB ACLR performance! Very bad.
- Added back the [1 0] and [2 0] (domain knowledge) terms and performance was 62 dB again
- Features reduced by 8 terms (40% reduction) with minimal impact on performance
- Completely eliminated p=4 term

Principal Component Analysis

- Two columns of the features are apparently nearly linear combinations of the other columns
- Can reduce the dimension of the space from 12 to 10 with minimal performance loss.
  - If willing to accept 55 dB performance, can go down to 7!
- Summary:
  - Hand optimized feature size: 33
  - After model selection: 20
  - After sensitivity analysis: 12
  - After PCA: 10



Gradient Descent Motivation

- DPD usually works by capturing samples going into and out of the PA
- Goal of DPD is to maximize ACLR
- ACLR can actually be measured while the PA is transmitting.
    - Requires <u>much</u> less hardware to measure ACLR than to capture and process samples.
- Use gradient descent to find the model coefficients without directly observing any samples going in and out of the PA.
- Must take into account PA output power as the simplest solution is to simply lower the power of the signal going into the PA. This will immediately cause ACLR to improve. We want both high ACLR and high transmit power.
- Minimize:
  J(theta)=-ACLR+abs(10*log10(actual_pa_power/desired_pa_power))

Gradient Descent Algorithm

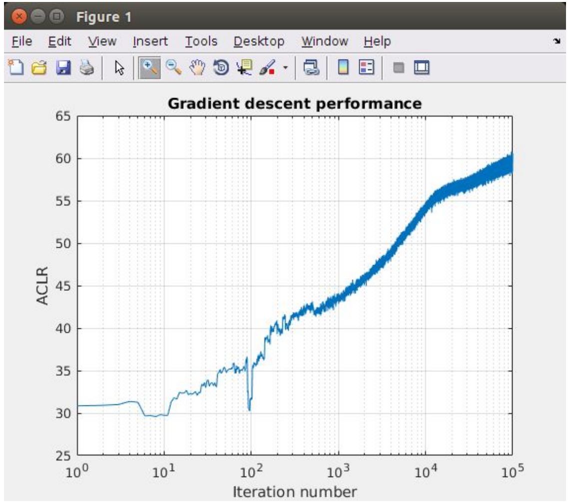
- PCA architecture used as a basis
- Measure the ACLR
- Pick a random element of theta and perturb it by a small amount
- Measure the new ACLR and calculate the gradient
- Update the coefficient:
  - theta(i)=theta(i)-gamma*gradient
- Complication: ACLR is measured on a log scale and as performance increases, stepsize needs to decrease exponentially.
- Final update equation:
  - theta(i)=theta(i)-gamma*$10^{-ACLR/10}$*gradient

Gradient Descent Algorithm

- Easy to understand
- Twiddle one of the coefficients by adding p.
- Did the ACLR get better or worse?
- If better, add a fraction of p to the coefficient
- If worse, add a fraction of -p to the coefficient
- Note that DPD coefficients are complex numbers and p is given a randomly chosen phase.

Gradient Descent Performance

- Gradient descent was highly susceptible to gamma value
- Gamma values greater than .02 quickly caused stability issues
- However, even with a gamma of .02, convergence was very slow
- Took 100,000 iterations to reach 60 dB of ACLR performance.
- Regression takes 2-3 iterations.



Conclusion

- The chosen PA can be linearized to good performance with only 7 features.
- Near perfect performance is achieved with 10 features using PCA.
- Gradient descent is not suitable for linearizing the chosen PA because of the slow rate of convergence required to maintain stability.