TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

PEDRO MIRASSOL TOMÉ
ANALOG NEURAL PREDISTORTION OF POWER AMPLIFIERS

Master of Science Thesis

ABSTRACT

Fifth-generation telecommunications networks are expected to have technical requirements which far outpace the capabilities of modern power amplifier linearization techniques such as digital predistortion. For this reason, this thesis proposes an alternative linearization method: a base band analog predistorter consisting of an artificial neural network trained using the Temporal Difference learning method. A vectorized model of the coupling of Temporal Difference learning (learning of the task) and backpropagation (structural adaptation of the neural network) is presented. While the specifics of the model may be quite complex, its formal simplicity allows for a very quick and straightforward implementation as well as its algorithmic realization, available as an appendix to this thesis. In effect, this thesis outlines a way towards the meeting of the specifications of next-generation networks.

PREFACE

I would like to express my gratitude to my thesis supervisors, Doctor Mikko Valkama, from the Tampere University of Technology, Finland, and Doctor Telmo Cunha, from the University of Aveiro, Portugal, for their always immediate support and for putting up with very self-reliant work methodology.

I would like to thank Doctor Olli-Pekka Lundén for his teachings and the interest he showed in my work, Doctor Tapio Elomaa for his contribution to my understanding of the Artificial Intelligence concepts contained within this thesis, and Mahmoud Abdelaziz, doctoral student at the Tampere University of Technology, for his resources on behavioural modeling.

I would like to acknowledge the Erasmus+ exchange program, without which I would have not had the opportunity to study at the Tampere University of Technology and to learn about the Finnish people and their customs.

My most sincere appreciation to the Teekkarikuoro. Of all the good things that happened to me in Finland, being a part of you was the best. There are no words which express how grateful I am for our time together and the feelings of us, through music, being one. I hope my playing of Rachmaninoff's *Élégie* was to your liking, and I hope it touched you as you did to me.

To my friends, Diogo Saraiva and William Richard. >Well played!

Finally, to my parents and my sister. Thank you.

Tampere, May 25, 2016
Pedro Mirassol Tomé

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

5G	Fifth generation of mobile telecommunications
AM-AM	Amplitude-to-Amplitude modulation
AM-PM	Amplitude-to-Phase modulation
ANN	Artificial Neural Network
APD	Analog Predistortion
CMOS	Complementary Metal-Oxide Semiconductor
DPD	Digital Predistortion
IMD	Intermodulation Distortion
PA	Power Amplifier
PD	Predistorter
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RC	Resistor-Capacitor
RF	Radio Frequency
TD	Temporal Difference
TD(λ)	Generalized Temporal Difference
TDMA	Time Division Multiple Access
TDNN	TD(λ) Neural Network

1. INTRODUCTION

While the requirements and specifications for fifth-generation (5G) mobile systems and services have yet to be fully defined, some goals of the next generation of mobile networks are already very clear: a tremendous increase in connection density and speed (over 1 Gbps downlink bit rate) and a similarly significant decrease in connection latency (under 1 ms roundtrip delay) [1].

However desirable, these advancements impose changes not only on the hardware that constitutes cellular networks, but on their topology as well. To be able to yield such high bit rates at such low latencies, cellular base station transmitters will need to have a wider operational bandwidth – on the order of 500 to 1000 MHz [2], in contrast to the few tens of MHz that current base stations possess –, and their center frequencies will have to be adjusted to higher regions of the spectrum – reportedly as high as 6 to 300 GHz [1].

Radiation at such high frequencies will evidently have limiting effects on the propagation of radio frequency (RF) signals through buildings and objects, thus leading to a structural change in network architectures: instead of network coverage being provided by central, hugely encompassing, high power transmitters, it will instead be done through the deployment of swarms of small, low power, distributed transmitters [1,3].

Ultimately, all of these changes, from the higher signal bandwidths to the lower power levels of the transmitting amplifiers, contribute to one critical outcome: the downfall of digital predistortion (DPD) as a viable linearization technique. Not only will the bandwidth of 5G power amplifiers (PAs) be too wide for the limited processing speed of state-of-the-art digital processors, but also their own power consumption (proportional to their switching frequency) will be too great compared to the power level of the PAs they linearize, thus defeating any sort of effort for increased power efficiency – in other words, it would not be sensible to linearize a 1 W power amplifier with a 20 W digital processor.

Naturally, the need for a means of PA linearization will remain: without it, achieving any of the next-generation (or even current-generation) goals would be impossible. New ideas must, therefore, be proposed and explored, and that is what this dissertation is all about.

1.1 The Thesis

Extraordinary needs require extraordinary measures, and thus a new line of thinking must begin. The aim of this dissertation is not to solve the problem of replacing 20 years' worth of research and technological development on digital predistortion, but to start the discussion on one way in which it might be possible to do so – eventually.

The headline of this work is the linearization of power amplifiers using the predistortion technique, performed at base band using analog implementations of artificial neural networks (ANNs), which are trained using the classical, mathematically sound machine learning method of Temporal Difference (TD).

This thesis builds upon analog predistortion (APD), the precursor to digital predistortion. Due to very significant technical advancements in digital electronics at the turn of the century, APD has been mostly put aside in favor of DPD. However, a small set of researchers have realized that the requirements for next-generation telecommunications will prove to be insurmountable for DPD, thus promoting the authoring of new literature on APD [4–6], albeit at a still relatively slow pace.

Another topic this thesis builds upon is the use of ANNs as predistortion devices, which has also been explored in the past. Most existing publications on neuronal predistortion are about DPD [7–9], since only recently has it been possible to implement ANNs as analog circuits. For this reason, the literature on this topic is still lacking [10,11].

The main contribution of this thesis is, then, the use of a formal, mathematical approach called TD(λ) to the training of the ANNs used as predistortion devices: one which, as far as we are aware, has not been used in the field of telecommunications as of yet. All of the mathematical formalization in vector form is original work.

In broad strokes, this project may be envisioned as follows:

     1. Propose a formal, mathematical description of the problem and its parts: one which is simple enough to be almost intuitive, yet powerful and complete enough to be readily implementable in an algorithmic fashion.

     2. Idealize a model of the system based on the previously defined mathematics. This entails, for instance, the establishment of the structure of the ANN: exactly how many neurons it should have, and their relation to the predistorted data (to differentiate between static and dynamic models).

     3. Synthesize the implementation of the model as an analog circuit. Extensive research will undoubtedly be required in order to build ANNs capable of meeting the specifications and requirements of 5G networks.

Naturally, this is far from a one-man job. The objective was to set up the foundation for future work, and that meant focusing on the first item of the list: building the mathematical model of a novel predistortion solution.

2. LINEARITY AND THE LACK THEREOF

Power amplifiers are some of the most fundamentally important devices in radio frequency telecommunications, since they are that which guarantees an information-carrying signal is of sufficiently high power level to be successfully transmitted by an antenna as small as a cell phone's or as large as a broadcasting radio station's.

Power amplifiers typically handle large amounts of power (for varying degrees of "large" – power ratings can vary by several orders of magnitude depending on the application), which means that power efficiency is of the highest importance: if efficiency is low, a cell phone's battery life may be severely compromised or the operational cost of a base station's cooling system may become unreasonably high.

On the other hand, if an amplifier is not perfectly linear – that is, if it does *anything* to the input signal other than to increase its power level (besides introducing a constant delay) –, the information that is supposed to be transmitted through the succeeding antenna may be corrupted.

And therein lies the problem. In general, the more linear an amplifier is, the less efficient it is [12]. For example, a class A amplifier (such as the textbook common emitter, single transistor amplifier) has very high linearity, but a theoretical (absolute maximum) efficiency limit of 50%. This isn't as unintuitive as it might seem – consider a class D amplifier, which is ideally a switch: because it is a switch, it can either be *on* or *off*, making it extremely nonlinear; but also because it is a switch, its theoretical efficiency is 100%, since "an ideal switch in its *on* state conducts all the current but has no voltage loss across it and therefore no heat is dissipated, and when it is *off* it has the full supply voltage across it but no leak current flowing through it, and again no heat is dissipated".

In short, typical applications demand high efficiency power amplifiers; because they are highly power efficient, they are very nonlinear, and because they are very nonlinear, the amplified signals – as well as the information they carry – are distorted. To solve this, these amplifiers are linearized in a variety of ways, resulting in a system that is both highly power efficient and highly linear: the best of both worlds.

2.1 Linearity: An Intuitive View

Static linearity can be formally defined through two distinct properties: superposition (2.1), and first-degree homogeneity (2.2). Essentially, this means that the net response of a linear system to a number of simultaneous inputs is the sum of the responses of the system to each individual input.

$$F(x_1 + x_2) = F(x_1) + F(x_2) \qquad (2.1)$$

$$F(\alpha x) = \alpha F(x) \qquad (2.2)$$

It is much easier, however, to think of a static linear system as one whose input/output response is, as the name implies, *linear*: a line. This line cannot have an offset, however, as there should be no output when there is no input. See Figures 2.1 and 2.2 for examples of linear and nonlinear static input/output responses.

On a more general and formal note, a linear system – be it static or dynamical –, is one whose variation of its state vector x is defined as in (2.3), where A is a constant matrix and b is a constant vector.
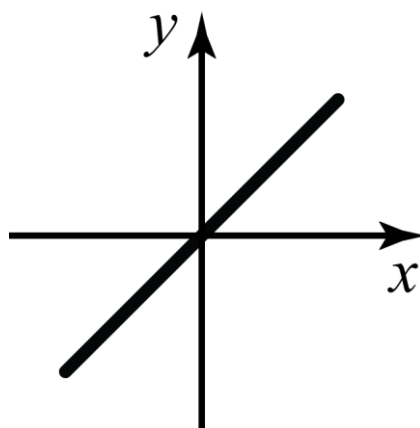
$$\dot{x} = Ax + b \qquad (2.3)$$

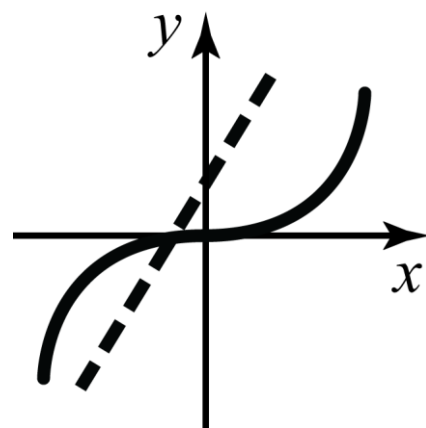**Figure 2.1.** *A linear static system.*   **Figure 2.2.** *Nonlinear static systems.*

2.2 Effects of Nonlinearity

It has been established that nonlinearity produces distortion in signals and has the potential to corrupt the information they carry. But how so? How can that be quantified?

Consider an amplifier whose behavior can be modeled by a simple third-order (nonlinear) polynomial with input $x(t)$ and output $y[\,x(t)\,]$, as in (2.4):

$$y[\,x(t)\,] = \ a_1 x(t) + a_2 x(t)^2 + a_3 x(t)^3 \tag{2.4}$$

Consider also a signal composed of two close tones, one at frequency ω_1 and amplitude X_1 and another at frequency ω_2 and amplitude X_2, defined in (2.5):

$$x(t) = \ X_1 \cos(\omega_1 t) + X_2 \cos(\omega_2 t) \tag{2.5}$$

The response of the amplifier to the signal is the sum of various tones at the following frequencies [13]:

- Base-band: $\omega_2 - \omega_1$
- Coincident with the signal: ω_1 , ω_2
- In-band distortion: ω_1 , ω_2 , $2\omega_1 - \omega_2$, $2\omega_2 - \omega_1$
- 2nd harmonic: $2\omega_1$, $\omega_1 + \omega_2$, $2\omega_2$
- 3rd harmonic: $3\omega_1$, $2\omega_1 + \omega_2$, $\omega_1 + 2\omega_2$, $3\omega_2$

Clearly, the response of the amplifier is not an amplified version of its input, otherwise the output tones would only be those coincident in frequency with the input ones; the spectrum has, therefore, expanded – see Figures 2.3 and 2.4 for a graphical example of a slightly more complex PA model (fifth-degree polynomial), showing only the fundamental frequency band.

High order harmonics and base band distortion are not exactly the problem, because they can be easily filtered out by the amplifier's output matching network. The real problem is in having to deal with spurious (unwanted) tones very near the input tones, because they would require filters with extremely high Q-factors (sharp frequency responses) to be eliminated, and those are not at all trivial to design. Also, filtering would not be reasonable for transceivers operating with multiple channels (at distinct frequency locations, although in nearby regions of the spectrum). Thus, intermodulation distortion (IMD) tones cannot be filtered – they have to be suppressed with linearization techniques.
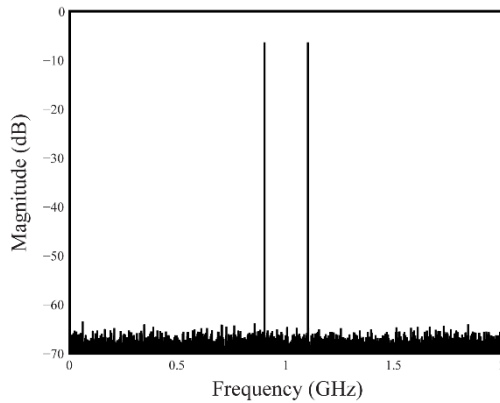
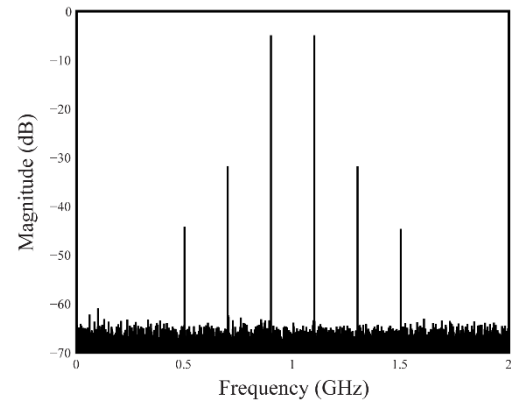***Figure 2.3.*** *The spectrum of the input signal of a nonlinear device.*

Figure 2.4. *The spectrum of the output signal of a nonlinear device.*

2.3 Linearization Techniques

Most linearization techniques fall into the four different categories explained in this section. Naturally, one can take advantage of a combination of them, producing fairly complex linearization circuits, but each of them may be used separately to great effect.

2.3.1 Power Back Off

Most power amplifiers have three operation regimes: at low powers, the amplifier is linear, with constant gain; when the amplifier approaches its saturation point, the device starts behaving nonlinearly and the gain starts decreasing; finally, when either the maximum rail voltage is reached or the maximum current is drawn, the amplifier saturates and its gain reaches its minimum.

Power back off simply consists in operating an amplifier in its linear regime, "backing off" (or "away") from the nonlinear ones; see Figure 2.5. Generally, the amount of back off power (say, 3 dB) is in respect to the device's 1 dB compression point, which is the point at which the power gain is 1 dB lower than its maximum value (the gain in the linear region, in the case of single-transistor class-A amplifiers).

The advantage of the employment of this technique is its extreme simplicity: either the input power is lowered so the amplifier operates exclusively in its linear region, or the supply voltage is increased so that the amplifier's linear region is extended. The disadvantage, however, is that the efficiency rapidly decreases with the increase of the back off power, since a linear amplifier is (usually) an inefficient one. Also, as a general rule, the higher the maximum power rating of an amplifier, the more expensive it is, so using a 200 W amplifier to produce a 100 W signal (3 dB back off) would certainly be more expensive than using a 100 W amplifier to produce the same signal.
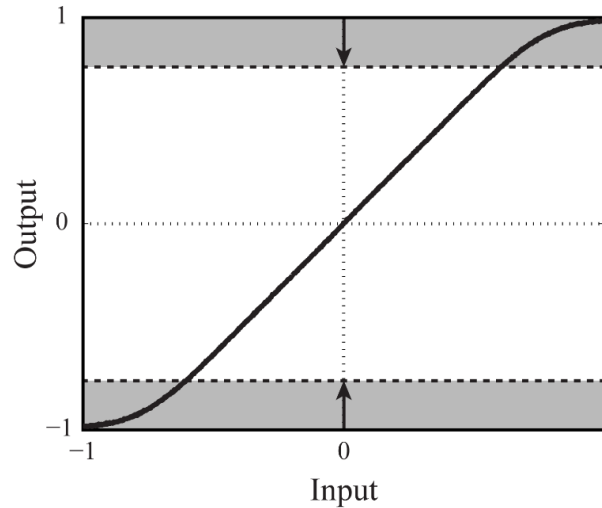
Figure 2.5. *Power back off from the perspective of an amplifier's normalized voltage input/output response.*

2.3.2 Cartesian Feedback

Most RF signals are generated through the modulation of a high frequency carrier signal using lower frequency data signals, called the in-phase (*I*) and in-quadrature (*Q*) signals. It is these *I* and *Q* components that define a system as "Cartesian", since they directly relate to a Cartesian representation of the transmitted signal (composition of two vectors, *I* and *Q*), rather than a polar one (magnitude and phase).

The most distinguishing feature of Cartesian feedback [14] – and the fundamental concept behind it – is the use of a negative feedback loop to control each of the *input I and Q components* so that the *output I and Q components* of the amplifier correspond to an output composite signal that is a linearly amplified version of the input composite signal. In Cartesian terms, a system is said to be linear if its output (*I, Q*) vector is a scaled version of its input (*I, Q*) vector – their phases should, therefore, be equal.

The output of an RF amplifier is an RF signal, so, in order to perform the feedback of its *I* and *Q* output components, these must be extracted with a demodulator which reverses the up-conversion done by the modulator that mixes the input *I* and *Q* signals with the carrier signal. After extracting the output *I* and *Q* components, *I* and *Q* error signals (the difference between the respective *I* and *Q* input and output components) are fed to control systems that guarantee the linearity of the overall system. These control systems, represented as "H(s)" blocks in Figure 2.6, may be designed with classical techniques such as dominant pole compensation [14].

The advantage of the Cartesian feedback linearization technique is, similarly to the power back off technique, its fair simplicity and reasonable IMD suppression. Feedback systems are inherently slow, though, so this technique is only reliable for low base band frequencies – up to hundreds of kHz at most [15] –, so RF feedback is not even attempted: any phase shift from the feedback path would ruin the system's stability.
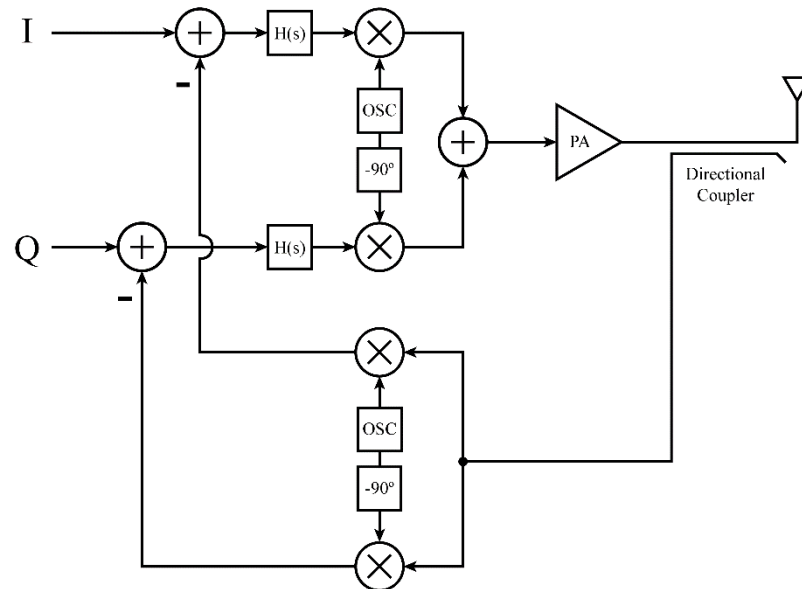
Figure 2.6. *Cartesian Feedback.*

2.3.3 Feedforward Linearization

In a feedback loop, a sample of the controlled system's output is subtracted from a reference input signal, producing an error signal. Likewise, in a feedforward scheme a sample of the controlled system's output is also subtracted from a reference input signal, producing an error signal as well. Naturally, if the system has a gain of *A* W/W then the sampled output should be attenuated by *A* W/W to achieve a proper difference or error signal; see Figure 2.7.

The difference between the two architectures – feedback and feedforward – is how they use the error signal which carries the information of how exactly the actual system output differs from the intended, target output: in a feedback topology, the error signal is used as the input of a controller which adjusts the controlled system's output so it matches the reference signal, i.e., the error signal has an indirect consequence on the system's output; in a feedforward topology, the error signal is *directly subtracted* from the system's output, producing a new, error-free signal further down the road.

Consider the following example:

- An amplifier has a power gain of 10 and introduces some spurious signals, whose power shall be named *D* ("D" for "distortion"). [e.g., $D = 0.2$ W]

- Let *X* be the input of the amplifier. Then, the output of the amplifier is $Y = 10X + D$, that is, a 10 times amplified version of the input signal plus some *D* amount of distortion. [e.g., $X = 7$ W; $Y = 70.2$ W]

- Now, to get the error signal, *E*, the input and output signals are subtracted while taking into account the gain of the amplifier (so both signals are at the same power level), so $E = X - Y/10 = X - (10X + D)/10 = -D/10$. [$E$ = -0.02 W]

- Finally, the feedforward part: the error signal is coupled (added) to the amplifier's output; again, the amplifier's gain has to be taken into consideration, so the error signal has to be multiplied by 10. The overall output of the linearized system is therefore $Y + 10E = 10X + D - D = 10X$, a perfectly amplified, distortion-free version of the input signal. $[Y + 10E = 70.2\ \text{W} + 10{\times}(\text{-}0.02\ \text{W}) = 70\ \text{W}]$

The main advantages of feedforward linearization are the wide operating bandwidth and the compensation of any sort of distortion produced by an amplifier – even that which is caused by the device's memory effects. The tradeoff, though, is the high complexity and the requirement of automatic adaptation to maintain performance specifications [15].
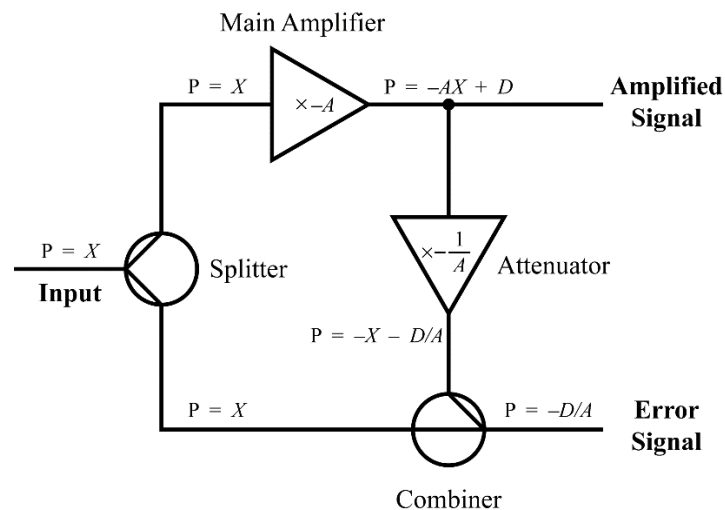

Figure 2.7. *Error signal generation through signal cancellation.*

A typical feedforward linearization system, schematized in Figure 2.8, consists of two circuits: a signal cancellation circuit and an error cancellation circuit.

The first circuit implements steps 1 to 3 of the previous example, that is, it produces a signal that *only contains the distortion* created by the power amplifier; it does this by attenuating the output of the amplifier (by an amount equal to the amplifier's gain) and combining the resulting signal with a copy of the input signal. Because these two signals have opposite phases, this essentially results in a subtraction, rather than an addition.

Finally, the second circuit implements step 4 of the previous example, that is, it amplifies the distortion signal extracted by the first circuit and couples it to the output of the amplifier. Similarly to the previous case, these two signals have opposite phases, so this essentially results in a subtraction. This means that the distortion generated by the amplifier is subtracted from the amplifier's own output signal, leaving a signal that is free of distortion and, by definition, a linearly amplified version of the input signal.
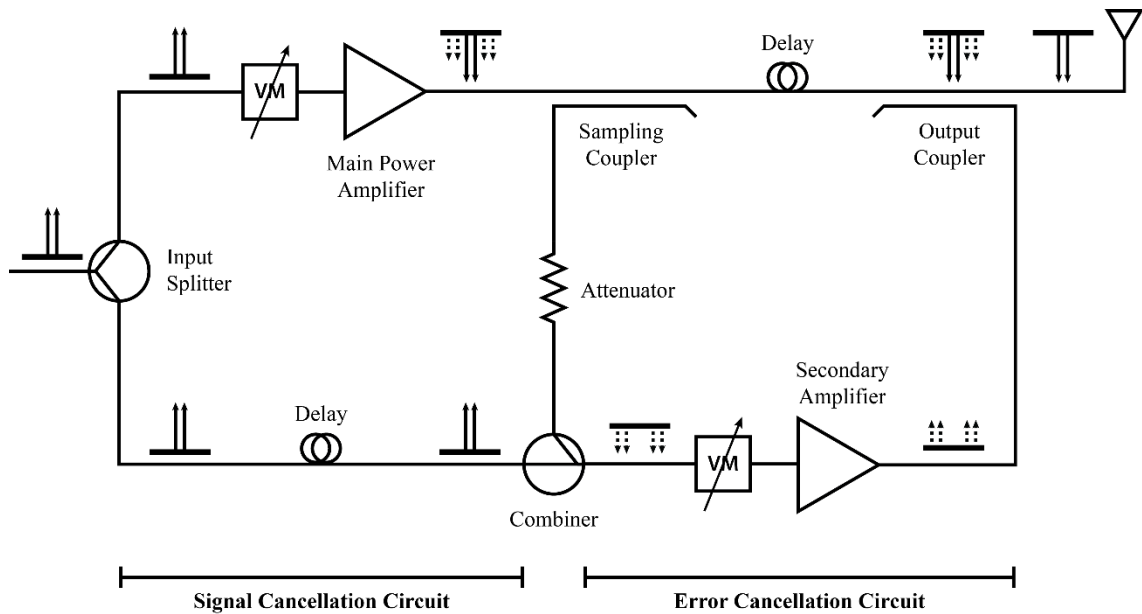
Figure 2.8. *Feedforward Linearization* [15].

2.3.4 Predistortion

Predistortion [16], illustrated in Figure 2.9, is the act of distorting a signal before it is fed to a nonlinear system in such a way that the distortion generated by the system is exactly canceled by the distortion synthesized by the predistorter (PD), resulting in an overall linear cascade of two devices. As an example, consider a system that has an input/output transfer function of $y = x^3$, which is clearly nonlinear; if a predistorter with an input/output transfer function of $y = \sqrt[3]{(x)}$ is used, then the *cascade* of the PD and the system is $y = [\sqrt[3]{(x)}]^3 = x$ and the overall system is perfectly linear.

The main advantage of predistortion is its potential to achieve fantastic intermodulation distortion suppression, i.e., very high linearity. However, predistortion usually requires the physical modeling of the amplifier, which is extremely complex, since most amplifiers exhibit memory effects, that is, their outputs depend not only on the current input, but the input at previous times as well. These models, as well as the predistortion of the input signals, are usually implemented using digital processors, which means that the bandwidth of the input signals is either limited by the sampling rate or the processing speed of the digital predistorter.

A common modification of the basic concept of predistortion is Cartesian predistortion (Figure 2.10), which is the predistortion of the base band (low frequency) in-phase and quadrature components (*I* and *Q*) instead of the predistortion of the RF (high frequency) composite signal. Among other things, this greatly reduces the required bandwidth of the predistorter. While this is a welcome relaxation of performance specifications in the case of APD, it is the very basis of DPD, since the predistortion of the RF signal would require extremely fast analog/digital conversion units and even faster processing units.

Finally, a very common way of simplifying the modeling of an amplifier and the resulting predistortion algorithm is to forgo the modeling of the amplifier's non-electrical characteristics, like temperature dependence, ageing, and other very slow phenomena. These can be compensated by recalculating the parameters of the amplifier's model based on the measurement of its response to a set of test signals. This way, the slow drifts of the input/output response of the PA due to changing temperature and other causes can be compensated. This is called "adaptive predistortion".
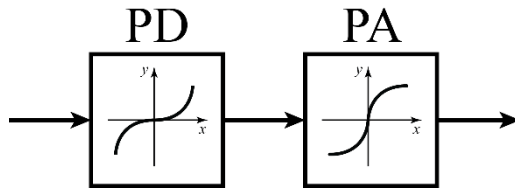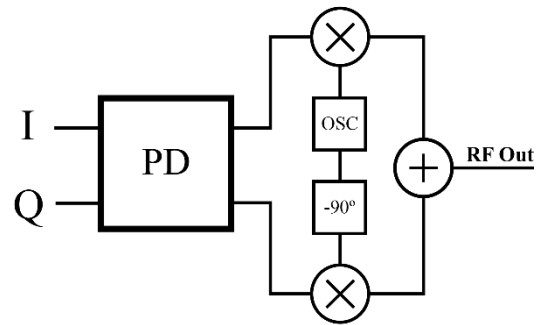

Figure 2.9. *RF Predistortion.*   ***Figure 2.10.*** *Cartesian Predistortion.*

3. ANALOG PREDISTORTION

Following Arthur C. Clarke's 1945 article on "Extra-Terrestrial Relays" [17] and John R. Pierce's 1955 article on "Orbital Radio Relays" [18], efforts towards global communications escalated along with a demand for higher transmission bandwidths at lower costs, leading to an increased interest in high order modulation techniques such as QPSK (Quadrature Phase Shift Keying) or QAM (Quadrature Amplitude Modulation) and multiple-access schemes such as TDMA (Time Division Multiple Access).

In order to achieve acceptable bit error rates and to meet the increasingly stringent spectral purity requirements of these data rate-increasing schemes, much attention was given between the late 1970s and the early 1980s to problems such as the linearization of high power microwave amplifiers used in satellite earth stations [19] and traveling wave tube amplifiers used in satellite transponders [20].

Because of the high power levels of these amplifiers, most linearization circuits consisted in the analog realization of the predistortion technique, applied not only to the microwave signals [20], but also (though less frequently) to the base band signals [19]. Regardless of the idiosyncrasy of each implementation, the great majority of the linearizers adhered to two main classes of predistortion circuits: cubic predistorters, and series diode predistorters [21].

In essence, cubic predistorters (Figure 3.1) couple the input signal to a distortion generator, a pair of antiparallel diodes which produces exclusively odd-order harmonics of the input signal while the even-order harmonics circulate inside the loop. Note that a single diode, when driven with RF, will generate both even- and odd-order distortion, and it should be biased close to the turn-on point so that sufficient distortion may be generated for low power RF signals [21].

A variable phase shifter is used to guarantee a 180º phase difference between the input and the distortion signals. This, along with a delay line that is used to equalize the group delays of the two signals, ensures that their coupling is subtractive. Finally, a variable attenuator ensures the amplitude of the generated distortion matches that of the harmonic distortion produced by the predistorted device (such as an amplifier). This amplitude matching, along with the 180º phase difference between the clean signal and the generated distortion, results in an appreciable suppression of the spurious odd-order tones produced by the nonlinear predistorted device.

Series diode predistorters (Figure 3.2) consist of a single forward-biased series diode, which may be modeled as a nonlinear resistor with a parasitic capacitance – an RC phase shift network. The S_{21} of the circuit is given in (3.1), where R is the resistance of the series diode, C is the sum of the parallel capacitance and the junction capacitance of the diode, and Z_0 is a characteristic impedance [22].

The principle of operation is fairly straightforward: as per Shockley's diode equation (3.2), an increase in forward (RF) power results in a decrease in the diode's series resistance. This, in turn, provided that the series resistance is not too high [22], results in an expanding gain and a decreasing phase shift, effectively countering the predistorted amplifier's undesired AM-AM and AM-PM characteristics: amplitude compression and phase advance.

$$S_{21} = \frac{2Z_0 Y}{1 + 2Z0Y}$$

$$\text{where } Y = \frac{1}{R} + j\omega C$$

(3.1)

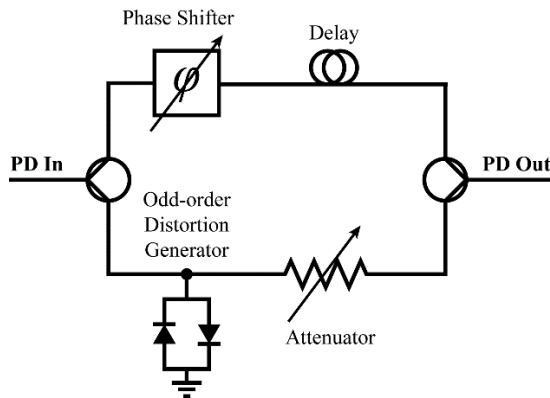$$I = I_S \left(e^{\frac{V_D}{nV_T}} - 1 \right)$$
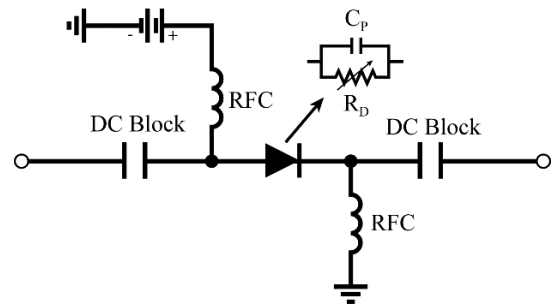
(3.2)

Figure 3.1. *Cubing Predistorter.*

Figure 3.2. *Series Diode Predistorter.*

With the advent of high speed digital computing, analog predistortion plummeted into near oblivion and was swiftly replaced by more capable and more configurable digital predistortion schemes. Still, some research was done, mainly in the early 2000s, and not only did old analog predistortion technology improve, some new interesting ideas even came to light.

The first great advancement in analog predistortion was the refinement of the cubing predistorter, which led to the development of fully configurable, independently controllable "IMD generators" [23–26]. These IMD generators are essentially branched versions of the cubing predistorter that generate 3rd- and 5th-order (and higher) intermodulation distortion tones, which can be independently scaled in magnitude and shifted in phase. See Figure 3.3 for an example of such a scheme.

The example circuit depicted in Figure 3.3 [24] is a 5th-order IMD generator predistorter. It has three main branches: the signal branch, the 3rd-order distortion branch, and the 5th-order distortion branch. The distortion components are generated using circuits based on cubic predistorters. To allow for individual control of each distortion component, the 3rd-order component on the 5th-order path is cancelled by coupling the 5th-order distortion signal with a copy of the 3rd-order distortion signal. In order for this to work, the two signals must be in anti-phase and with the correct amplitudes, so a vector modulator is used in the 5th-order path. Finally, vector modulators are used to tune each individual distortion component, and the resulting signals are coupled to the output of the complete predistorter.

The second great advancement – perhaps the most noteworthy, due to its novelty – was the realization that the AM-AM and AM-PM characteristics of a moderately nonlinear amplifier can be modelled to some extent by complex-valued polynomials of low order [27–29]. These polynomials, in turn, – or, rather, their inverse – can be approximated by transistor circuits based on the Gilbert cell [30] (Figure 3.4): a cascode circuit used as an analog four-quadrant multiplier and frequency mixer. A new class of CMOS circuits was therefore designed to implement high order polynomials (as high as 11th-order) with freely configurable coefficients and thus synthesize the inverse transfer characteristic of an amplifier – an almost ideal predistorter.

The Gilbert cell shown in Figure 3.4 consists of two pairs of transistors, Q2-Q3 and Q5-Q6, which have their collectors cross-connected. The bases of these transistors are driven by a third pair of transistors, Q1-Q4, connected as diodes, which makes the circuit linear [30].

The Gilbert cell is a circuit with two inputs, X and Y, and one output, Z. The first signal input X is the pair of currents xI_B and $(1 - x)I_B$, where $0 \leq x \leq 1$, and the second signal input Y is the pair of currents yI_E and $(1 - y)I_E$, where $0 \leq y \leq 1$. Because the ratio of the emitter currents in the Q2-Q3 and Q5-Q6 pairs is equal to that of the Q1-Q4 pair, we can write (3.3) [30]:

$$I_{C2} = xyI_E$$
$$I_{C3} = (1 - x)yI_E$$
$$I_{C5} = x(1 - y)I_E \qquad (3.3)$$
$$I_{C6} = (1 - x)(1 - y)I_E$$

Thus, the normalized differential output is (3.4) [30],

$$Z = \frac{I_{out}}{I_E} = \frac{I_{C2} + I_{C6} - I_{C3} - I_{C5}}{I_E} = 1 - 2y - 2x + 4xy \qquad (3.4)$$

Then, if both signals X and Y are bipolar and between the range [-1, +1], the normalized output is (3.5):

$$Z = XY \qquad (3.5)$$
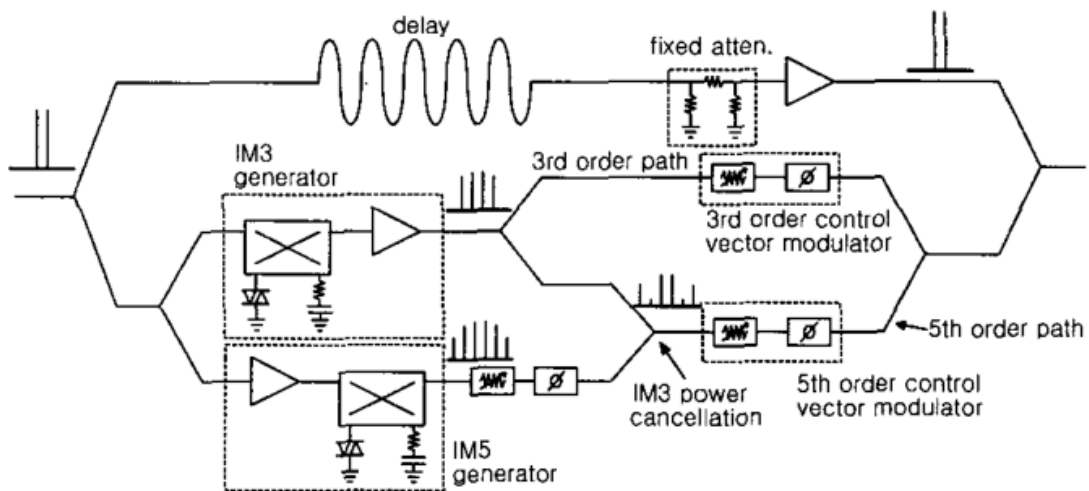
Figure 3.3. *5th-order IMD generating predistorter* [24].
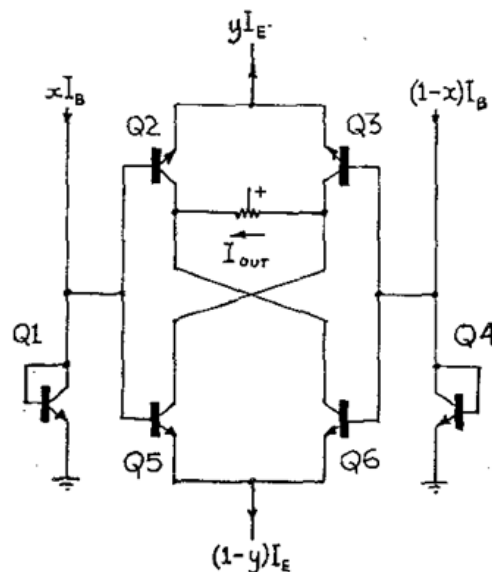
Figure 3.4. *The Gilbert cell* [30].

Finally, in the present decade, various novel analog predistortion schemes have surfaced, possibly in anticipation of the 5G networking challenges already summarized. These schemes include, among others, the bandwidth reduction of error signals [31], the use of mirror amplifiers [32], and lookup table-based, combined digital/analog predistortion systems [33].

In a mirror predistorter (Figure 3.5), a low power PA is used before the main high power PA. If the construction of the low power PA is identical to that of the main PA, then its distortion characteristics will be the same as the main PA's, though at a lower power level [32]. In effect, these characteristics mirror those of the main PA in both magnitude and phase, hence the name of the predistortion technique.

Similarly to the feedforward linearization technique, the distortion caused by the mirror PA is isolated by coupling it with an anti-phase copy of the input signal. Then, it is scaled and phase shifted in such a way that, when fed to the main PA alongside the input signal, they are canceled by the main PA's nonlinear characteristics.
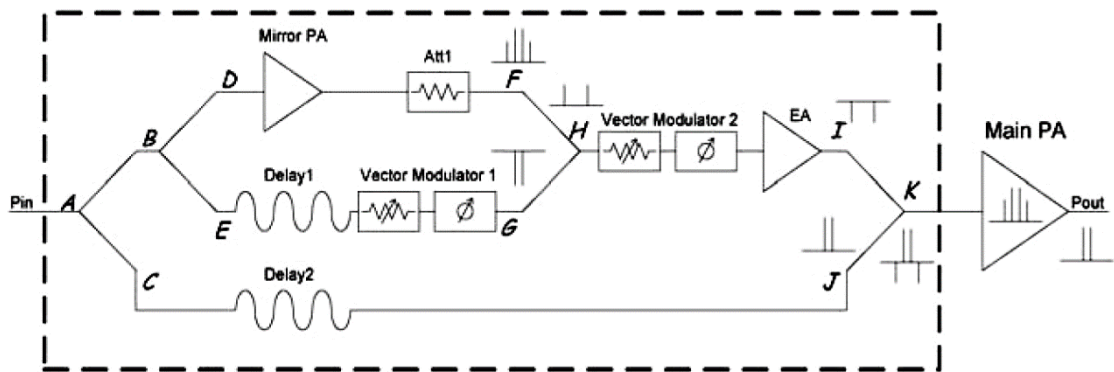
Figure 3.5. *Mirror predistorter* [32].

3.1 Proposed APD System Architecture

The system architecture of the proposed predistortion solution, schematized in Figure 3.6, consists of an analog feedforward artificial neural network that predistorts the base band *I* and *Q* components of a complex telecommunications signal. As usual, the predistorted signal is then transposed to a much higher frequency with an RF modulator and it is then fed to the power amplifier that should be linearized. Naturally, some additional components – such as filters and intermediate amplification stages – are required for the implementation of the solution, but the ones illustrated in Figure 3.6 are the main blocks of the system.

This base band architecture is ideal for an analog solution based on an artificial neural network, since the bandwidth requirements of the ANN are much lower than they would be if it were used as a radio frequency predistorter. As an additional reason for having chosen a base band solution, the predistortion of the *I* and *Q* components of the complex RF signal is a matter of amplitude scaling, which means that the function the ANN is supposed to learn is real-valued. This contributes to a relatively simple model of the ANN-based predistorter and its learning algorithm.

The ANN-based PD is trained using a reinforcement learning algorithm called Temporal Difference learning, which requires a feedback of the *I* and *Q* components output by the PA. The training, however, can be done through simulation using a forward model of the PA, leaving the effective predistortion system architecture unchanged.
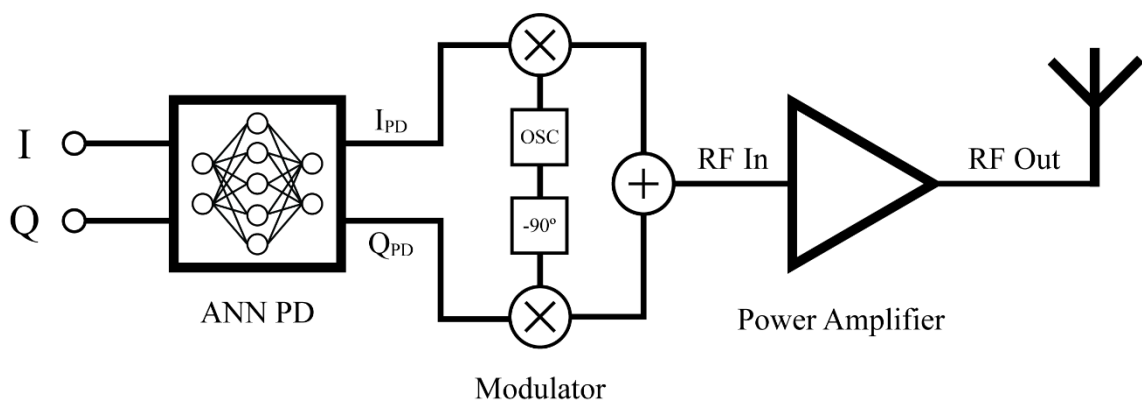
Figure 3.6. *Predistortion system architecture.*

4. ARTIFICIAL NEURAL NETWORKS

Not unlike polynomials or Volterra series, artificial neural networks are a family of nonlinear function models which consist of a series of basic computational units, the neurons (akin to polynomials' power products), that are interconnected by means of model-defining weights (akin to polynomials' coefficients). Even though there are metrics such as the Vapnik-Chervonenkis dimension, the evaluation of the complexity of an ANN (similar to a polynomial's degree) has yet to be formally and unequivocally defined [34], though it is intuitive that it is related to the number of neurons it comprises and the way they are interconnected.

The basic computational unit of an ANN is the neuron, or node, illustrated in Figure 4.1. A neuron can have an arbitrary positive number of inputs x, one of which acts as a bias, and these are processed by an activation function Φ, which is selected by the ANN designer to calculate the neuron's activation a: its output. Typical activation functions include a purely linear transfer function (4.1) and the (logistic) sigmoid function (4.2), and these can be used at will throughout an ANN. A variety of sigmoid (meaning s-shaped) functions can be used for different levels of algorithmic optimization.

$$\Phi(z) = z \tag{4.1}$$

$$\Phi(z) = \frac{1}{1 + e^{-z}} \tag{4.2}$$

Figure 4.1. *A neuron with three inputs.*

Figure 4.2. *An example feedforward network with three input nodes, one hidden layer with five nodes, and two output nodes. Displayed as well are the biasing nodes for the hidden and output layers.*

There is a nearly endless number of ways of arranging and interconnecting neurons in an ANN. There are, however, classical and established ways of doing so, such as the feedforward network illustrated in Figure 4.2. In a feedforward network, neurons are distributed between different, sequentially ordered layers: the input layer, a set of hidden l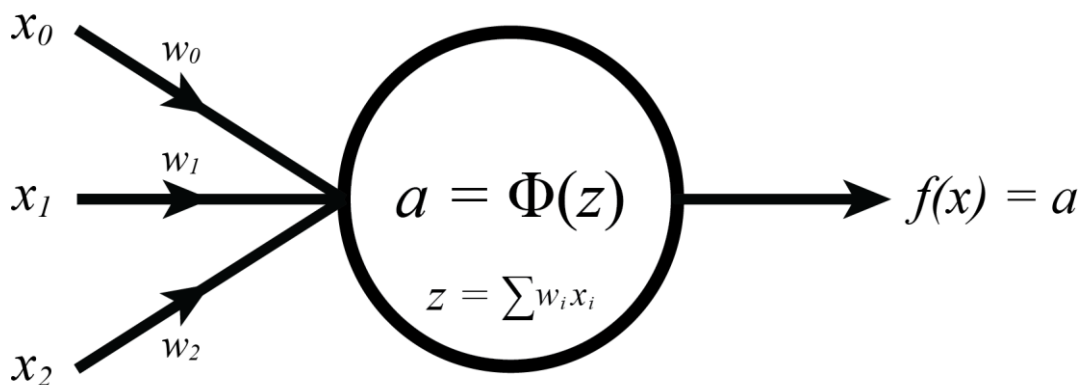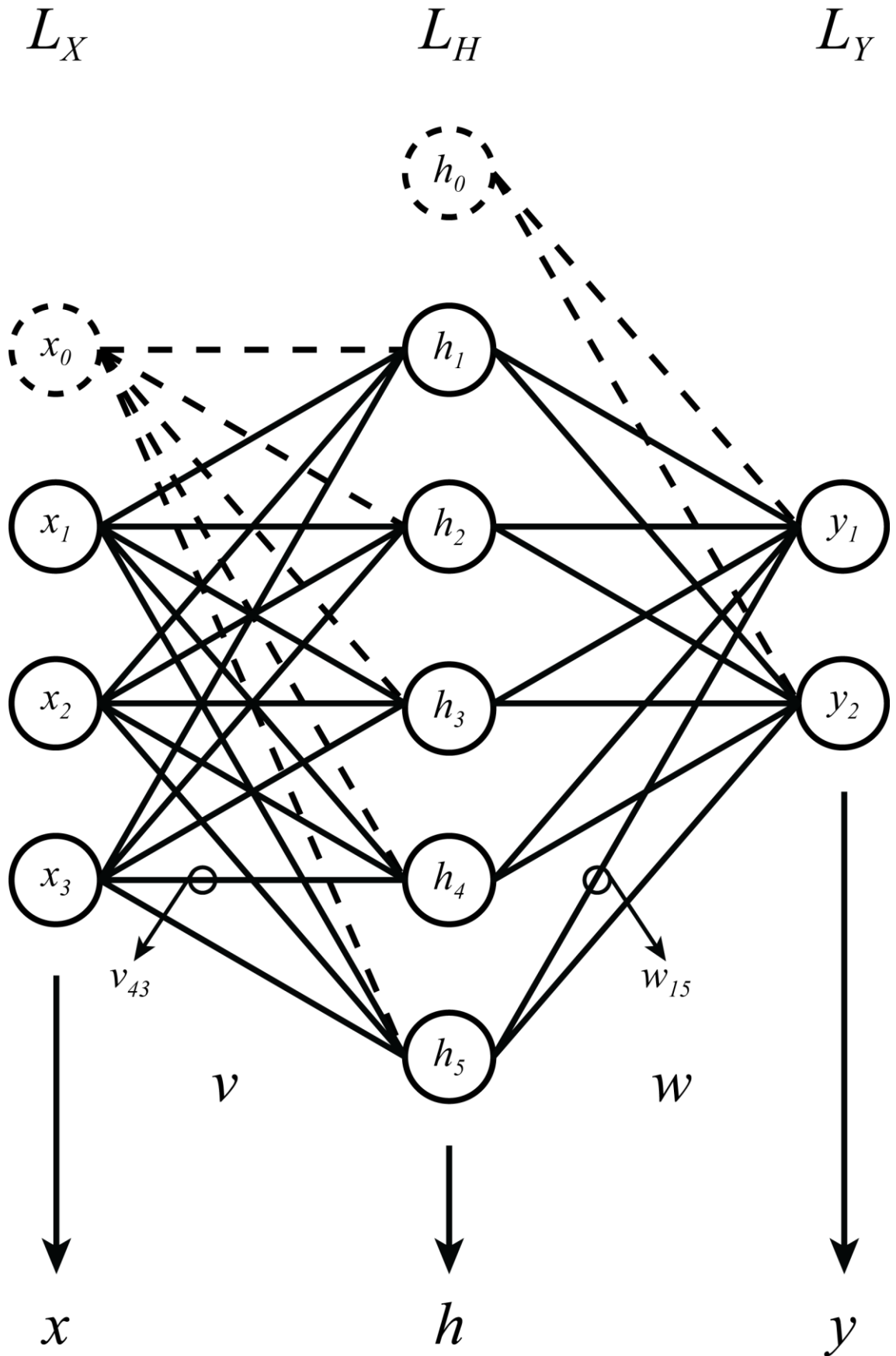ayers, and an output layer. Each neuron in each layer connects to every neuron in the immediately succeeding layer, and there are no backward or intra-layer connections – meaning that there are no cyclical connections, hence the network's designation of "feedforward".

Feedforward ANNs are universal approximators [35]. This means that for any given continuous nonlinear function, there is at least one feedforward ANN that approximates it, in a closed and bounded input range (a compact set of $\mathbf{R}^n$), with an arbitrarily small error. This was proven for feedforward networks containing a single hidden layer of neurons with sigmoidal activation functions [36,37], though it stands to reason that more expressive networks, with more hidden layers, would perform at least as well as ANNs with a single hidden layer. Naturally, the output layer should have neurons with purely linear activation functions, otherwise the range of each of the network's output neurons would be limited to (-1, 1).

4.1 ANNs as Feature-Learning Systems

While this thesis concerns the use of an ANN as a control (regression) system, ANNs can also be used as feature-learning agents. For instance, ANNs can be used for classification, which includes pattern and sequence recognition. With proper training, these networks can assign labels to data. Examples include the classification of email as "spam" or "not-spam", the identification of drawn digits and letters, and the classification of medical data as indicative of the presence of a tumor, with a stated probability.

ANNs can also be used for behavioral modeling, that is, creating arbitrarily complex models of physical devices. This has been used even in the predistortion of power amplifiers, where an ANN learns the forward model of the PA and is then used as the basis for the training of another predistorting network.

Another example of the use of ANNs as feature-learning systems is game playing. The classical example is Tesauro's Backgammon-playing ANN [38]. Backgammon is a board game in which two players alternately roll a pair of dice and move their checkers (black vs. white) in opposite directions around the playing board. On each turn, a player rolls the dice, moves one checker the number shown on one die, and then moves a second checker (or the same one) the number shown on the other die [39]. The point of the game is to be the first to move each of one's checkers all the way around and off the board.

Tesauro's Backgammon-playing network had an input layer with 198 nodes that represented the board state and a single hidden layer with 40 nodes. The output layer of the network consisted of four sigmoidal nodes which estimated the probability of white or black both achieving a regular win or a gammon (special win case) [39].

The network was trained completely by self-play: a computer simulated a dice roll and generated all board positions that were possible from the current position and the number of the die. The network was fed each of these board positions and the one that the network ranked highest was chosen as the next state. The network then did the same for the next player and repeated the process [39]. After a sufficient number of games, the network had acquired good game-playing strategies and deeply explored the state space.

4.2 ANNs as Analog Control Systems

Due to their massive expressive ability and structural simplicity, as well as ease of training, artificial neuronal networks have been used to solve board games such as backgammon [38] and Go [40], control physical systems such as inverted pendulums [41], and even predistort RF power amplifiers [7,8].

In the neuronal predistorter described in [8], a double-input and double-output design is used (just like the one proposed in this thesis). The input vector of the ANN contains the in-phase and in-quadrature components of the transmitted signal, and the output vector contains the corresponding components of the predistorted signal.

The internal structure of this ANN includes a "tapped delay line" to describe the memory effect of the PA – see Figure 4.3. While this is interesting, it could very well have been implemented as a regular ANN, but instead of having two inputs (for the I and Q signals), it would have additional inputs for the delayed signals.

As shown in Figure 4.4, the neural network was first used to perform post-distortion linearization of the PA, and then used as a predistorter. With enough repetitions, this resulted in the (simulated) compensation of the PA's nonlinearities.
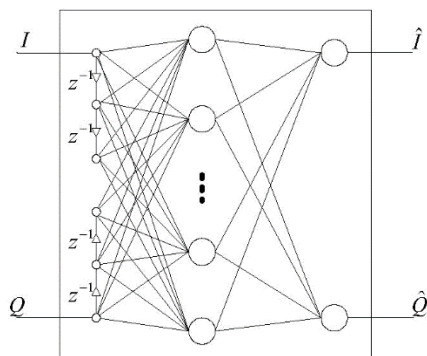

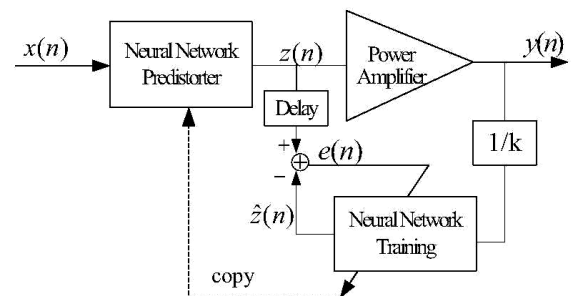***Figure 4.3.*** *Neural network with a tapped delay line* [8].

Figure 4.4. *System architecture of the neuronal predistorter in* [8].

Despite their differences, all of these applications of ANNs have one thing in common: they are digital implementations. Recent technological advances have brought the possibility of reliably implementing ANNs as analog circuits. Further advances, such as commercially-available memristors, are expected to lead to even more robust and higher-performing analog ANNs.

Compared to the analog predistortion schemes presented in section 3, analog implementations of ANNs provide very substantial advantages. Not only are relatively simple ANNs much more expressive than 11th-order polynomials (the state-of-the-art predistortion circuits until recently) in terms of function synthesis, but they also have an increased capability for generalization due to their saturating (sigmoidal) neurons, which is important when the predistorter's input range may not be clearly defined – high-order polynomials grow very quickly towards infinity outside the training sample space.

Furthermore, the bandwidth of each of an ANN's computational units (neurons) is similar to that of the predistorted signal, in contrast to the bandwidth of a polynomial's computational units (power products), which grows mostly linearly with the degree of each product (i.e., over an order of magnitude for an 11th-order polynomial predistorter).

4.3 Analog Implementations of ANNs

The class of circuits which emulate the behavior of neurons (high network connectivity, simple base processing element, and distributed memory and computation) is called neuromorphic circuits [42]. The purpose of these circuits is not to exactly replicate the processing performed by a biological brain, but to emulate some of its characteristics and do so with very different elements.

ANN chips usually offer one of three different learning capabilities of increasing complexity [42]: "off-chip learning", where the learning is done on a simulated network using software; "chip-in-the-loop learning", a combined approach where the forward pass is performed in hardware but the actual learning algorithm is implemented in software; and "on-chip learning", where all stages of learning are done by the hardware itself.

In an analog chip, the weights of the network can be stored either digitally (thus requiring digital/analog converters for the learning and chip configuration) or using analog elements, such as resistors or capacitors [42]. Additionally, they can be stored by digitally programmable, reconfigurable analog elements, though this still requires the learning to be done digitally.

Summation of the neuron inputs is rather trivial in an analog circuit, since they will either be represented as currents (which sum by joining branches in parallel) or voltages (which sum by joining branches in series). The sigmoidal activation function is usually implemented using strongly nonlinear amplifiers [42].

The implementation of ANNs using CMOS processes suffers from significant drawbacks, the most important one being the poor scalability of these circuits: a large enough ANN may prove to be impossible to be wired due to the sheer number and density of connections [42]. A new class of post-CMOS circuits seems very promising, however.

These circuits are based on the memristor, a long-anticipated fundamental circuit element on the level of resistors, capacitors and inductors [43]. In a very simplified way, memristors can be used as electrically-programmable resistances. This, along the fact that the memristive effect occurs mostly at the nanoscale, points towards the analog implementation of easily-configurable ANNs with phenomenal scaling capabilities [42].

Commercially available analog ANN chips include the Intel 80170NX and the Synaptics Silicon Retina [44]. The Intel 80170NX, also known as Electrically Trainable Analog Neural Network, was the first commercially available ANN chip. It stores the weights as electrical charges in floating gates and uses Gilbert cells for the weighting of the neuron inputs. The Synaptics Silicon Retina is a special case of hardware implementation of ANNs because it tries to emulate biological neurons as close as possible, instead of implementing a conventional, more formal architecture [44].

Commercially available hybrid implementations include the AT&T Artificial Neural Network ALU (ANNA), the Bellcore CLNN-32, the Mesa Research Neuroclassifier, and the Ricoh RN-200 [44]. The first two circuits provide a digital input interface, despite their internals being fully analog. The AT&T ANNA stores its weights as capacitor charges that are refreshed periodically, and provides a variable number of neurons from 16 to 256, though it has no built-in learning algorithm. The CLNN-32, on the other hand, implements fully connected recurrent networks and provides a built-in Boltzmann learning algorithm [44]. The Mesa Research Neuroclassifier provides an analog input and output interface, but the weights are stored digitally as 5-bit words per weight. This is the highest performing chip available, claiming a speed of 21 Giga-Connections Per Second (the number of multiply and accumulate operations per second), over an order of magnitude higher than the other enumerated chips [44]. Finally, the RN-200 implements ANNs with 16 neurons of 16 synapses each through pulse rates and widths, as well as a built-in learning algorithm. An array of 12 Ricoh RN-100 chips (the precursor to the RN-200) was used to learn how to balance a 2-D inverted pendulum in just 30 seconds [45].

4.4 Mathematical Formalization

Figure 4.2 represents a feedforward ANN with three layers: L_X, the input layer; L_H, the hidden layer; and L_Y, the output layer. Let there be the following symbols:

nX : the number of input nodes in L_X (excluding bias) – in this case, $nX = 3$;

nH : the number of hidden nodes in L_H (excluding bias) – in this case, $nH = 5$;

nY : the number of output nodes in L_Y – in this case, $nY = 2$;

x : a column vector, indexed as x_i, holding the node activations of L_X;

h : a column vector, indexed as h_j, holding the node activations of L_H;

y : a column vector, indexed as y_k, holding the node activations of L_Y;

v : a matrix, indexed as v_{ji}, holding the weights of the connections from L_X to L_H;

w : a matrix, indexed as v_{kj}, holding the weights of the connections from L_H to L_Y.

These symbols are defined as such, with example values based on Figure 4.2:

$$
\underset{((nX+1)\times 1)}{x} = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_{nX} \end{bmatrix} \qquad \text{eg:} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
$$

$$
\underset{((nH+1)\times 1)}{h} = \begin{bmatrix} h_0 \\ h_1 \\ \dots \\ h_{nH} \end{bmatrix} \qquad \text{eg:} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix}
$$

$$
\underset{(nY\times 1)}{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_{nY} \end{bmatrix} \qquad \text{eg:} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}
$$

$$
\begin{array}{cc}
v & = \\
(nH \times (nX{+}1))
\end{array}
\begin{bmatrix}
v_{10} & v_{11} & \cdots & v_{1nX} \\
v_{20} & v_{21} & \cdots & v_{2nX} \\
\cdots & \cdots & & \cdots \\
v_{nH0} & v_{nH1} & \cdots & v_{nHnX}
\end{bmatrix}
\quad \text{eg:}
\begin{bmatrix}
v_{10} & v_{11} & v_{12} & v_{13} \\
v_{20} & v_{21} & v_{22} & v_{23} \\
v_{30} & v_{31} & v_{32} & v_{33} \\
v_{40} & v_{41} & v_{42} & v_{43} \\
v_{50} & v_{51} & v_{52} & v_{53}
\end{bmatrix}
$$

$$
\begin{array}{cc}
w & = \\
(nY \times (nH{+}1))
\end{array}
\begin{bmatrix}
w_{10} & w_{11} & \cdots & w_{1nH} \\
w_{20} & w_{21} & \cdots & w_{2nH} \\
\cdots & \cdots & & \cdots \\
w_{nY0} & w_{nY1} & \cdots & w_{nYnH}
\end{bmatrix}
\quad \text{eg:}
\begin{bmatrix}
w_{10} & w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\
w_{20} & w_{21} & w_{22} & w_{23} & w_{24} & w_{25}
\end{bmatrix}
$$

Thus, x_i is the activation of the i-th input node (the i-th input value, for $i > 0$), h_j is the activation of the j-th hidden node, y_k is the activation of the k-th output node, v_{ji} is the weight of the connection between the input node i and the hidden node j, and w_{kj} is the weight of the connection between the hidden node j and the output node k. One can read the matrix v, then, as a series of columns containing the weights of the connections of each input node to every hidden node. Similarly, the matrix w can be read as a series of columns containing the weights of the connections of each hidden node to every output node.

The indexing of the v and w matrices is intentionally backwards. It would have been more aesthetic to define them as v_{ij} and w_{jk}, but this would have required the computation of their transpose matrices to perform forward propagation (explained below). The algorithmic performance gain is minimal, but it comes at essentially no cost.

To be precise, the nodes of the input layer aren't exactly neurons, but mere representations of the "input ports" of the ANN. There is no data processing or neuronal activation: input values just pass on through unchanged. This does not undermine the presented formalization, however, since it is trivial to devise neurons which would exhibit that exact behavior: a neuron, with no biasing and one data input with unitary weight, whose activation function is purely linear.

Furthermore, despite biasing being a property of the neurons and not the network architecture (even from the original, biological standpoint), it can be abstracted away as a node with constant activation (eg: $a = 1$) which connects to each neuron with weights proportional (or even equal) to the required biasing values. These biasing nodes and their connections are represented in Figure 4.2 with dashed lines, and they are referred to as the zeroth (0-th) node in each layer, if applicable. The output layer is the last layer, so, naturally, it doesn't contain bias nodes for its (nonexistent) succeeding layer.

4.5 Forward Propagation

Having defined a model for the architecture and the constituting parts of an ANN, it is now possible to model the network's operation, that is, to define how to determine its output vector. Forward propagation, the classical algorithm for doing precisely that, consists of sequentially computing the activations of each layer, from the input to the output layer.

Let the input (column) vector of the ANN – that is, the data being fed to it at a given instant – be *netInput*. Then, the vector of input node activations x is the concatenation of the activation of the input bias node, here defined as a constant 1 (the number one), and the activations of the externally-stimulated data nodes – that is, *netInput*. Similarly, the vector h is the concatenation of the hidden bias node and the activations of the hidden nodes connected to the input layer; as discussed earlier, each node's activation is a function of the weighted sum of its inputs. Finally, because there are no output bias nodes, the y vector is simply obtained by computing the activations of the output nodes.

It should be noted that the Φ function is to be applied in an element-wise fashion, and it is not necessarily the same function for every neuron (even in the same layer) – the Φ symbol is used repeatedly only to simplify the notation.

$$x \;=\; \begin{bmatrix} 1 \\ netInput \end{bmatrix}, \quad h \;=\; \begin{bmatrix} 1 \\ \Phi(v \cdot x) \end{bmatrix}, \quad y \;=\; \Phi(w \cdot h)$$

4.5.1 Example

Let us consider the ANN illustrated in Figure 4.2. The activation function of the hidden nodes is the sigmoid function (4.2), referred to as sig(), and the activation function of the output nodes is the purely linear function (4.1), referred to as purelin().

Let $v \;=\; 0.01 \times$ [10 11 12 13; 20 21 22 23; 30 31 32 33; 40 41 42 43; 50 51 52 53].
Let $w = 0.01 \times$ [10 11 12 13 14 15; 20 21 22 23 24 25].

Let *netInput* $\;=\;$ [1 2 3]$^{\mathrm{T}}$.
Then, $x \;=\;$ [1; *netInput*] $\;=\;$ [1 1 2 3]$^{\mathrm{T}}$.
Then, $h \;=\;$ [1; sig($v \cdot x$)] $\;=\;$ [1.0000 0.6985 0.8235 0.9038 0.9498 0.9744]$^{\mathrm{T}}$.
Finally, $y \;=\;$ purelin($w \cdot h$) $\;=\;$ [0.6723 1.2073]$^{\mathrm{T}}$.

4.6 Backpropagation

The Backward Propagation of Errors, or backpropagation, is the most common method of training artificial neural networks, used typically in conjunction with optimization algorithms which aim to minimize the cumulative squared error between the ANN's actual output and its target output. Such algorithms include the Nelder-Mead method [46] and the Levenberg-Marquardt algorithm [47].

Backpropagation is typically called a supervised learning algorithm, in which the target output of the ANN is explicitly specified by the modeler. This, however, is not a precise way of describing backpropagation. While it is true that it can be used (and is most often used) to perform supervised learning tasks when coupled with one of the optimization algorithms enumerated above, the true purpose of backpropagation is to solve the problem of *structural credit assignment*, that is, the problem of adjusting the weights in the network to minimize the error [39]. There is a subtle but important distinction between the two definitions – one which will be expanded upon further. Meanwhile, let us explore the formalism behind *backpropagation proper*, that is, the mechanics of weight adjustment. See [39] for this (and more) information.

Let there be an ANN whose nodes' activations have been obtained through the forward propagation of a training input vector and whose output error E has been determined according to some specific metric. For the purpose of completeness, let this metric be the sum of the square of the errors between the actual output vector y and the target output vector t of the network.

The global weight update rule is displayed in (4.3). This rule asserts that the change $\Delta\theta_{ij}$ in every weight θ_{ij} of the network (the elements of the v and w matrices) should be proportional (with constant α) to the negative of the derivative of the error with respect to the weight itself:

$$\Delta\theta_{ij} = -\alpha\frac{\partial E}{\partial\theta_{ij}} \tag{4.3}$$

Using the chain rule, the partial derivative of the error with respect to each weight between the hidden and output layers can be calculated, resulting in (4.4), where net_k is the net input of the output node k, that is, $w \cdot h$:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} \tag{4.4}$$

Simple substitutions lead to (4.5), where $\Phi'_k(net_k)$ is the derivative of the activation function of the output node k evaluated at net_k:

$$\frac{\partial E}{\partial w_{kj}} = -2(t_k - a_k) \cdot \Phi'_k(net_k) \cdot a_j \tag{4.5}$$

We can now use δ_k to represent $(t_k - a_k) \cdot \Phi'_k(net_k)$, thus leading to (4.6):

$$-\frac{\partial E}{\partial w_{kj}} \propto \delta_k \, a_j \tag{4.5}$$

Using the chain rule, the partial derivative of the error with respect to each weight between the input and hidden layers can be calculated, resulting in (4.7), where net_j is the net input of the output node j, that is, $v \cdot x$:

$$\frac{\partial E}{\partial v_{ji}} = \frac{\partial E}{\partial a_k} \cdot \frac{\partial a_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial a_j} \cdot \frac{\partial a_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial v_{ji}} \tag{4.7}$$

Simple substitutions lead to (4.8), where $\Phi'_j(net_j)$ is the derivative of the activation function of the hidden node j evaluated at net_j:

$$\frac{\partial E}{\partial v_{ji}} = \delta_k \cdot w_{kj} \cdot \Phi'_j(net_j) \cdot a_i \tag{4.8}$$

Contrary to the weights between the hidden and output layers, the weights between the input and hidden layers affect all of the output nodes simultaneously. Thus, the partial derivative of the error *across all of the output nodes* is defined in (4.9)

$$\delta_j = \Phi'_j(net_j) \sum_k \delta_k \cdot w_{kj} \tag{4.9}$$

Finally, the partial derivative of the error with respect to the weights between the input and hidden layers can be defined as in (4.10):

$$-\frac{\partial E}{\partial v_{ji}} = \delta_j \, a_i \tag{4.10}$$

5. TEMPORAL DIFFERENCE LEARNING

Temporal Difference (TD) is a reinforcement learning method, that is, a way of using past experience with an incompletely known system to predict its future behavior [48]. In a more mechanistic sense, TD is an algorithm for an agent (like a predistorter) to learn which actions to take over an environment (like a power amplifier) in order to maximize some notion of cumulative reward (like a measure of an amplifier's linearity).

TD is an unsupervised learning algorithm, which means that it does *not* require the *a priori* knowledge of the desired output of the learning agent. This is an exceptionally important detail: using a supervised learning algorithm to teach an ANN how to predistort a power amplifier does not make much sense if one does not know the amplifier's inverse transfer function to begin with.

This does not mean that it is impossible to do so, as there are a variety of papers on neuronal predistortion of power amplifiers [7–9]. These papers, however, either don't explicitly specify the learning procedure (only mentioning backpropagation, which, as is hopefully clear by now, is not a serious answer), or describe a learning procedure consisting of iteratively training an ANN to be a post-distorter, testing its performance as a predistorter, and training it again in order to gain some measure of improvement.

While this sort of methodologies may lead to acceptable results, TD provides a learning solution that is formal and serious, and it has been used in applications as diverse as solving the game of Backgammon [38], controlling quadcopter motors and inverted pendulums [41], simulating the steering of a boat across a river [49], and sensor state prediction [50].

It should be noted that TD is a general learning algorithm, that is, it does not make any assumptions regarding the learning agent. TD is not, therefore, immediately applicable to the training of structurally complex constructs such as ANNs, and that means that some sort of mathematical coupling needs to be devised. Luckily, this problem has already been solved, and it is explained further.

5.1 Mathematical Formalization

5.1.1 TD Error

Let V be the value function an agent is trying to learn. TD learning consists in adjusting V so that $V(s_t)$ – where s_t is the input state at time t – approximates the return R_t at time t, defined in (5.1) as a discounted sum of future rewards. γ is the discount constant, and it controls how far the agent should look ahead when making predictions at the current time step [39]. Equation (5.2) is derived trivially from (5.1).

$$R_t \;=\; r_{t+1} \;+\; \gamma\, r_{t+2} \;+\; \gamma^2\, r_{t+3} \;+\; \cdots \;=\; \sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1} \tag{5.1}$$

$$R_t \;=\; r_{t+1} \;+\; \gamma\, R_{t+1} \tag{5.2}$$

Thus, the TD error E_t at time t can defined as in (5.3):

$$E_t \;=\; R_t - V(s_t) \;=\; (r_{t+1} + \gamma\, R_{t+1}) - V(s_t) \tag{5.3}$$

Finally, using $V(s_{t+1})$ as an approximation of R_{t+1}, we obtain the generalized TD error in (5.4):

$$E_t \;=\; r_{t+1} + \gamma\, V(s_{t+1}) - V(s_t) \tag{5.4}$$

5.1.2 Weight Update

The derivation of the weight update rule (5.5) is rather involved, and can be found in [39].

$$\Delta w_t \;=\; \alpha[V(s_{t+1}) - V(s_t)] \sum_{k=1}^{t} \lambda^{t-k}\, \nabla_w V(s_k) \tag{5.5}$$

This is the generalized formula for TD(λ), which is the generalized form of TD itself, introduced in [25]. α is a learning-rate parameter, $V(s_{t+1}) - V(s_t)$ is the (temporal) difference between consecutive predictions, $\nabla_w V$ is the gradient of the value function with respect to its defining weights, and λ is a gradient discount parameter such that $0 \le \lambda \le 1$. λ tracks to which extent the prediction values for previous observations are eligible for updating based on current errors [39]. Therefore, the sum (5.6) is called the *eligibility trace* at time t.

$$e_t \;=\; \sum_{k=1}^{t} \lambda^{t-k}\, \nabla_w V(s_k) \tag{5.5}$$

5.2 TD(λ) Neural Networks

As discussed earlier, backpropagation solves the problem of *structural credit assignment*. On the other hand, TD solves the problem of *temporal credit assignment*, that is, the problem of attributing credit (or "blame") for error over the complete history of predictions made by the learning agent [39], and it does so through the mechanism we've just introduced: eligibility traces.

Through TD(λ) learning, an agent can determine its error based on successive predictions, and through backpropagation an agent can modify its model of prediction in order to reduce the error. Thus, combining the two algorithms results in a very powerful coupling: a universal nonlinear function approximator which learns through acquired experience.

Contrary to other neural predistortion schemes found in the literature, the one proposed in this thesis – a TD(λ) Neural Network (TDNN) – is actually capable of learning how to be a predistorter. Since the learning algorithm does not require the knowledge of the target output of the ANN, the problem of predistortion may be tackled directly, and not indirectly by training the network as a post-distorter and hoping it works as a predistorter.

5.2.1 Mathematical Formalization

5.2.1.1 Weight Update

The coupling of TD learning and backpropagation is done at the weight update stage of the algorithms. Thus, and referring back to section 4, the change in the network's weights v and w is a function of the TD error E (at each output node k) and their respective eligibility traces ev and ew:

$$\Delta w_{kj} = E_k ew_{kj} \tag{5.6}$$

$$\Delta v_{ji} = \sum_k E_k ev_{ji}^{(k)} \tag{5.7}$$

From (5.6) it is very apparent that ew should be a matrix with the same size as w: $(nY \times (nH+1))$. From (5.7) it is apparent that ev should be, however, a three-dimensional matrix of size $(nH \times (nX+1) \times nY)$ – or, rather, a set of nY matrices of size $(nH \times (nX+1))$, which is the size of w. The superscript (k) notation refers to each of the nY matrices.

5.2.1.2 Eligibility Traces

In section 4, a mathematical formalization – a model – of a generic artificial neural network was proposed. In this section, this model is expanded to include the eligibility traces introduced by the TD learning method, effectively resulting in a model of a TDNN. The basis of this work can be found in [39] and [51].

Let ew_{kj} denote the eligibility trace correspondent to the weight of the connection from the hidden node j to the output node k. Let δy_k denote $\Phi'_k(net_k).$ Then, the update rule for ew_{kj} is (5.8):

$$ew_{kj} := \lambda ew_{kj} + \Delta ew_{kj},$$

$$\text{where } \Delta ew_{kj} = \delta y_k h_j$$

(5.8)

The matrix form of (5.8) is self-evident, but the scheme in Figure 5.1 illustrates a simple way of deducing it:

$$h^{\mathrm{T}} = \begin{bmatrix} h_0 & h_1 & h_2 & h_3 & h_4 & h_5 \end{bmatrix}$$

$$\delta y = \begin{bmatrix} \delta y_1 \\ \delta y_2 \end{bmatrix} \underbrace{\begin{bmatrix} \delta y_1 h_0 & \delta y_1 h_1 & \delta y_1 h_2 & \delta y_1 h_3 & \delta y_1 h_4 & \delta y_1 h_5 \\ \delta y_2 h_0 & \delta y_2 h_1 & \delta y_2 h_2 & \delta y_2 h_3 & \delta y_2 h_4 & \delta y_2 h_5 \end{bmatrix}}_{\Delta ew}$$

Figure 5.1. *Deduction of the matrix form of Δew.*

Thus we get the update rule for the matrix form of ew:

$$ew := \lambda ew + \Delta ew,$$

$$\text{where } \Delta ew = \delta y \cdot h^T$$

(5.9)

The activation function of the output nodes of the TDNN is purely linear, so $\delta y_k = 1$ for all k.

Let $ev_{ji}^{(k)}$ denote the derivative of the output unit k with respect to the weight from the input unit i to the hidden unit j, that is, a partial eligibility trace correspondent to the weight of the connection from the input node i to the hidden node j.

Let $\overline{w}$ be the w matrix without its first column. Let $\overline{\delta h}$ be the δh vector without its first row. This removes the elements of these objects correspondent to h_0, the hidden bias node. This is necessary because there are no connections from the input nodes to the hidden bias node, which means that there are no corresponding weights or eligibility traces.

Then, the update rule for $ev_{ji}^{(k)}$ is (5.10):

$$ev_{ji}^{(k)} := \lambda ev_{ji}^{(k)} + \Delta ev_{ji}^{(k)},$$

$$\text{where } \Delta ev_{ji}^{(k)} = \delta y_k \, \overline{w_{kj}} \, \overline{h}_j \, x_i$$

(5.10)

Let us explore the Δ term of ev based on Figure 4.2:

$$\Delta ev_{10}^{(1)} = \delta y_1 \, w_{11} \, \delta h_1 \, x_0$$
$$\Delta ev_{11}^{(1)} = \delta y_1 \, w_{11} \, \delta h_1 \, x_1$$
$$\Delta ev_{12}^{(1)} = \delta y_1 \, w_{11} \, \delta h_1 \, x_2$$
$$\Delta ev_{13}^{(1)} = \delta y_1 \, w_{11} \, \delta h_1 \, x_3$$

$$\Delta ev_{20}^{(1)} = \delta y_1 \, w_{12} \, \delta h_2 \, x_0$$
$$\Delta ev_{21}^{(1)} = \delta y_1 \, w_{12} \, \delta h_2 \, x_1$$
$$\Delta ev_{22}^{(1)} = \delta y_1 \, w_{12} \, \delta h_2 \, x_2$$
$$\Delta ev_{23}^{(1)} = \delta y_1 \, w_{12} \, \delta h_2 \, x_3$$
$$\vdots$$
$$\Delta ev_{10}^{(2)} = \delta y_2 \, w_{21} \, \delta h_1 \, x_0$$

Let (5.11), where $\cdot$ is the matrix multiplication operator and $.*$ is the element-wise multiplication operator:

$$\zeta = \delta y \cdot \overline{h} \, .* \, \overline{w}$$

(5.11)

Thus,

$$\frac{\zeta}{(nY \times nH)} = \begin{bmatrix} \delta y_1 w_{11} \delta h_1 & \delta y_1 w_{12} \delta h_2 & \cdots & \delta y_1 w_{1nH} \delta h_{nH} \\ \delta y_2 w_{21} \delta h_1 & \delta y_2 w_{22} \delta h_2 & \cdots & \delta y_2 w_{2nH} \delta h_{nH} \\ \vdots & \vdots & \ddots & \vdots \\ \delta y_{nY} w_{nY1} \delta h_1 & \delta y_{nY} w_{nY2} \delta h_2 & \cdots & \delta y_{nY} w_{nYnH} \delta h_{nH} \end{bmatrix}$$

(5.12)

Substituting (5.11) in (5.10) we get (5.13):

$$\Delta ev_{ji}^{(k)} = \zeta_{kj}\, x_i \tag{5.13}$$

Let $\zeta^{(k)}$ denote the k-th row of the ζ matrix. Then, finally, we get the update rule for the matrix form of each $ev^{(k)}$:

$$ev^{(k)} := \ \lambda ev^{(k)} + \left(x \cdot \zeta^{(k)}\right)^T \tag{5.14}$$

As a final note, the derivative of the activation functions used throughout the ANN are defined in (5.15) for the sigmoid function and in (5.16) for the purely linear function.

$$\Phi_j'\left(net_j\right) = 1 \tag{5.15}$$

$$\Phi_k'(net_k) = a_k(1 - a_k) \tag{5.16}$$

5.2.2 TDNN Algorithm

The model for an artificial neural network using temporal differences as a learning method has been established. Now, let us explain how it can be used and present a class-based Matlab implementation of the vectorized TDNN model and the learning algorithm based on Sutton's (the creator of TD(λ)) own TD/Backpropagation pseudo-code [51], also used as a reference for the expansion of the model.

In a slightly simplified way, the TDNN algorithm consists of repeatedly iterating over the following set of steps:
   1. Perform the forward propagation of an input vector;
   2. Calculate the TD error at the output of the network;
   3. Update the network's weights;
   4. Perform the forward propagation of the same input vector with the new weights;
   5. Update the eligibility traits of the network.

The provided Matlab implementation allows the creation of a TDNN with an arbitrary number of input, hidden, and output nodes. For instance, the command

```
>> net = TDNN(2, 5, 2);
```

Creates, through the class's constructor and the `init` function, a TDNN object named `net` with two input nodes (plus a bias node), one layer of five hidden nodes (plus a bias node), and two output nodes. The nodes are initialized as vectors of zeros, except the bias nodes which are initialized as vectors of `BIAS = 1`.

The matrices of connection weights are initialized uniformly with random numbers. This not only helps prevent the training of the network towards a local minimum of the error, but it also nearly eliminates the possibility of the network not learning at all due to being permanently stuck at a zero state.

The main interface to the TDNN class presented below is the `train` function, which implements the learning loop described above. At first, it forward propagates the first sample of the input vector and calculates the eligibility traces of the network's weights. No learning is performed because the TD error equation (5.4) requires one skipped time step to be causal. Then, it loops through the learning algorithm for every input vector sample.

Forward propagation is done by the `forwardProp` function. In this function, the externally-connected input nodes are activated by the TDNN's input vector, and the activations of the rest of the network's nodes are calculated as explained in section 4.3.

The sigmoid activation function is implemented as the Matlab function `tansig`, which is a computationally faster version of Matlab's own hyperbolic tangent function. Although it produces some numerical errors, these are not exceptionally relevant in the case of neural networks, since they don't inhibit learning. It is a good tradeoff for speed, then.

The `updateElig` function updates the eligibility traces of the network's weights. As described in the section 5.2.1, it first computes the delta terms of the eligibility traces and then performs the update rules defined in (5.9) and (5.10). Naturally, the delta terms depend on the chosen activation functions, so both options (sigmoid and purely linear) were left for choosing by the programmer. To pick an activation function, comment out the line correspondent to the other activation function.

Finally, the `TDLearn` function updates the weights of the network according to the update rules defined in (5.6) and (5.7). Note that the matrix of eligibility traces for the weights of the connections between input and hidden nodes is three-dimensional, so the update of these weights has to be done in a loop to account for all eligibility traces.

This implementation took a very significant amount of time to develop, so it is presented as part of the body of this thesis, and not as just an appendix. In a way, it is the practical/lab part of the thesis.

```matlab
classdef TDNN
    properties (GetAccess = 'public', SetAccess = 'public')
        RAND_INIT_EPSILON;

        numInputs;
        numHidden;
        numOutputs;

        BIAS;        % Strength of the bias (constant) contribution
        ALPHA;       % 1st layer learning rate (typically 1/numInputs)
        BETA;        % 2nd layer learning rate (typically 1/numHidden)
        GAMMA;       % Discount-rate parameter (typically 0.9)
        LAMBDA;      % Trace decay parameter (should be <= GAMMA)


        x; h; y;     % Neuron activations for layers 1 to 3
        v; w;        % Weights


        oldY;        % Last output
        ev; ew;      % Hidden and output eligibility traces
        error;       % TD error
    end

    methods
        function self = TDNN(numInputs, numHidden, numOutputs)
            validateattributes(numInputs, ...
                {'numeric'}, {'scalar', 'positive', 'integer'}, ...
                '', 'numInputs');
            validateattributes(numHidden, ...
                {'numeric'}, {'scalar', 'positive', 'integer'}, ...
                '', 'numHidden');
            validateattributes(numOutputs, ...
                {'numeric'}, {'scalar', 'positive', 'integer'}, ...
                '', 'numOutputs');

            self.numInputs = numInputs;
            self.numHidden = numHidden;
            self.numOutputs = numOutputs;


            self.RAND_INIT_EPSILON = 0.5;
            self.BIAS   = 1;
            self.ALPHA  = 1/numInputs;
            self.BETA   = 1/numHidden;
            self.GAMMA  = 0.9;
            self.LAMBDA = 0.5;

            self = self.init();
        end


```

```matlab
53          function self = init(self)
54              % Neuron Activations Initialization
55              self.x = [self.BIAS ; zeros(self.numInputs, 1)];
56              self.h = [self.BIAS ; zeros(self.numHidden, 1)];
57              self.y = zeros(self.numOutputs, 1);
58              self.oldY = zeros(self.numOutputs, 1);
59
60              % Random Weight Initialization
61              self.v = rand(self.numHidden, self.numInputs + 1)  * ...
62                  (2 * self.RAND_INIT_EPSILON) - self.RAND_INIT_EPSILON;
63              self.w = rand(self.numOutputs, self.numHidden + 1) * ...
64                  (2 * self.RAND_INIT_EPSILON) - self.RAND_INIT_EPSILON;
65
66              % Eligibility Traces Initialization
67              self.ev = zeros(self.numHidden, self.numInputs + 1, ...
68                              self.numOutputs);
69              self.ew = zeros(self.numOutputs, self.numHidden + 1);
70          end
71
72
73          function [self, output] = forwardProp(self, input)
74              self.x(2:end) = input;
75              self.h(2:end) = tansig(self.v * self.x);
76              %self.y = tansig(self.w * self.h);
77              self.y = purelin(self.w * self.h);
78              output = self.y;
79          end
80
81
82          function self = TDLearn(self)
83              self.w = self.w + self.BETA * repmat(self.error, 1, ...
84                      self.numHidden + 1) .* self.ew;
85
86              dv = zeros(size(self.v));
87              for k = 1 : self.numOutputs
88                  dv = dv + self.error(k) * self.ev(:,:,k);
89              end
90              self.v = self.v + self.ALPHA * dv;
91          end
92
93
94          function self = updateElig(self)
95              %deltaY = self.y .* (1 - self.y);% Output nodes: tansig()
96              deltaY = ones(size(self.y));     % Output nodes: purelin()
97              deltaH = self.h .* (1 - self.h);
98
99              self.ew = self.LAMBDA * self.ew + deltaY * self.h';
100
101             tmp = deltaY * deltaH(2:end)' .* self.w(:,2:end);
102             for k = 1 : self.numOutputs
103                 self.ev(:,:,k) = self.LAMBDA * self.ev(:,:,k) + ...
104                                 (self.x * tmp(k,:))';
105             end
106         end
107
108
109
```

```
110        function self = train(self, netInput, reward)
111            t = 1;
112            self = self.forwardProp(netInput(:,t));
113            self.oldY = self.y;
114            self = self.updateElig();
115
116            for t = 2 : size(netInput, 2)
117                self = self.forwardProp(netInput(:,t));
118                self.error = reward(:,t) + self.GAMMA * self.y - ...
119                             self.oldY;
120                self = self.TDLearn();
121
122                self = self.forwardProp(netInput(:,t));
123                self.oldY = self.y;
124                self = self.updateElig();
125            end
126        end
127
128
129        function netOutput = output(self, netInput)
130            numSamples = size(netInput, 2);
131            netOutput = zeros(self.numOutputs, numSamples);
132
133            for t = 1:numSamples
134                [~, tmp] = self.forwardProp(netInput(:,t));
135                netOutput(:,t) = tmp;
136            end
137        end
138    end
139 end
```

Program 1. *A class-based Matlab implementation of the TDNN algorithm.*

6. CONCLUSION

A new generation of telecommunications networks requires a new generation of linearization systems for the power amplifiers they rely on. Thus, a base band analog predistorter implemented as an artificial neural network was proposed as a solution.

Traditionally, ANNs are trained in a supervised manner. This, however, goes against the very essence of the problem of predistortion: to *find* the optimal predistortion function. Roundabout ways of solving this paradox have been documented in the literature, such as training the ANN as a post-distorter and testing it as a predistorter.

In this thesis, a fundamentally different training approach is proposed: temporal difference learning. While this is a classical unsupervised learning technique, it has never been used in the field of telecommunications as far as we are aware – and definitely not as a learning procedure for power amplifier predistortion. This marks an important contribution to the state of the art, then.

The most important piece of original work in this thesis is the vectorized mathematical model for the coupling of artificial neural networks and temporal difference learning. While the specifics of the model may be quite complex, its formal simplicity allows for a very quick and straightforward implementation, exemplified in Program 1.

6.1 Future Work

It would be of value to verify the model proposed in this thesis through simulation. While this was initially part of the list of objectives to be achieved, this task was not possible to be completed due to time constrains and the workload caused by other courses.

The implementation exists – Program 1 –, but it is merely a means, not the end: the simulation of the model requires the tuning of a variety of parameters and a good selection of a reward signal. While this was attempted, success unfortunately proved to be elusive.

Notwithstanding, there is plenty of other research left to do – as was the point of proposing the solution described in this thesis. Main topics include the determination of the optimal size of the ANN to be used as a predistorter, and also the physical implementation of the ANN as an analog circuit capable of meeting the performance specifications of 5th-generation telecommunications networks.

7. REFERENCES

[1] D. Warren, C. Dewar, Understanding 5G: Perspectives on Future Technological Advancements in Mobile, GSMA Intelligence, 2014. Available: www.gsmaintelligence.com/files/analysis/?file=141208-5g.pdf

[2] H. Wang, X. Zhou, M. Reed, Coverage and Throughput Analysis with a Non-Uniform Small Cell Deployment, IEEE Transactions on Wireless Communications, 2014, Vol. 13, pp. 2047–2059.

[3] Samsung, 5G Vision, White Paper, 2015. Available: www.samsung.com/global/business-images/insights/2015/Samsung-5G-Vision-2.pdf

[4] D. Bharadia, E. McMilin, S. Katti, Full Duplex Radios, ACM SIGCOMM Computer Communication Review, 2013, Vol. 43, pp. 375–386.

[5] M. Cho, J. Kenney, Variable Phase Shifter Design for Analog Predistortion Power Amplifier Linearization System, Proceedings of Wireless and Microwave Technology Conference (WAMICON), Orlando, FL, USA, Apr 7–9, 2013, pp. 1–5.

[6] P. Fisher, S. Al-Sarawi, Analog RF Predistorter Simulation using Well-Known Behavioral Models, Proceedings of 10th Conference on Industrial Electronics and Applications (ICIEA), Auckland, New Zealand, Jun 15–17, 2015, pp. 1667–1671.

[7] N. Benvenuto, F. Piazza, A. Uncini, A Neural Network Approach to Data Predistortion with Memory in Digital Radio Systems, IEEE International Conference on Communications, Geneva, Switzerland, May 23–26, 1993, pp. 232–236.

[8] Y. Qian, F. Liu, Neural Network Predistortion Technique for Nonlinear Power Amplifiers with Memory, First International Conference on Communications and Networking in China, Beijing, China, Oct 25–27, 2006, pp. 1–5.

[9] B. Watkins, R. North, Predistortion Of Nonlinear Amplifiers Using Neural Networks, Proceedings of Military Communications Conference, McLean, VA, USA, Oct 21–24, 1996, pp. 316–320.

[10] B. Mulliez, E. Moutaye, H. Tap, L. Gatet, F. Gizard, Predistortion System Implementation Based On Analog Neural Networks For Linearizing High Power Amplifiers Transfer Characteristics, Proceedings of the 36th International Conference on Telecommunications and Signal Processing (TSP), Rome, Italy, Jul 2–4, 2013, pp. 412–416.

[11] M. Ngwar, An Analog Neural Network for Wideband Predistortion of Pico-cell Power Amplifiers, thesis, Carleton University, 2015. Available: https://curve.carleton.ca/2d167908-4475-4ab3-b192-eeb4c656e2de

[12] A. Katz, J. Wood, D. Chokola, The Evolution of PA Linearization, IEEE Microwave Magazine, 2016, Vol. 17, pp. 32–40.

[13] N. Carvalho, R. Madureira, Intermodulation Interference in the GSM/UMTS Bands, Proceedings of 3ª Conferência de Telecomunicações (ConfTele 2001), Figueira da Foz, Portugal, April 23–24, 2001.

[14] J. Dawson, T. Lee, Cartesian Feedback for RF Power Amplifier Linearization, Proceedings of 2004 American Control Conference, Boston, MA, USA, Jun 30 – Jul 2, 2004, Vol. 1, pp. 361–366.

[15] S. Stapleton, Presentation on Adaptive Feedforward Linearization for RF Power Amplifiers, Agilent Technologies, 2001. Available: http://literature.agilent.com/litweb/pdf/5989-9106EN.pdf

[16] R. Gupta, S. Ahmad, R. Ludwig, J. McNeill, Adaptive Digital Baseband Predistortion for RF Power Amplifier Linearization, High Frequency Electronics, 2006, Vol. 5, No. 9, pp. 16–25.

[17] A. Clarke, Extra-Terrestrial Relays, Wireless World, No. 10, 1945, pp. 305–308.

[18] J. Pierce, Orbital Radio Relays, Jet Propulsion, 1955, Vol. 25, pp. 153–157.

[19] H. Girard, K. Feher, A New Baseband Linearizer for More Efficient Utilization of Earth Station Amplifiers Used for QPSK Transmission, IEEE Journal on Selected Areas in Communications, 1983, Vol. 1, pp. 46–56.

[20] G. Satoh, T. Mizuno, Impact of a New TWTA Linearizer Upon QPSK/TDMA Transmission Performance, IEEE Journal on Selected Areas in Communications, 1983, Vol. 1, pp. 39–45.

[21] C. Haskins, Diode Predistortion Linearization for Power Amplifier RFICs in Digital Radios, Master's Thesis, Virginia Polytechnic Institute and State University, 2000. Available: https://theses.lib.vt.edu/theses/available/etd-04252000-16200028/unrestricted/chaskins_thesis.pdf

[22] K. Yamauchi, K. Mori, M. Nakayama, A Novel Series Diode Linearizer for Mobile Radio Power Amplifiers, IEEE International Microwave Symposium, S. Francisco, CA, USA, Jun 17–21, 1996, Vol. 2, pp. 831–834.

[23] J. Yi, M. Park, W. Kang, B. Kim, Analog Predistortion Linearizer for High-Power RF Amplifiers, IEEE Transactions on Microwave Theory and Techniques, 2000, Vol. 48, pp. 2709–2713.

[24] S. Lee, Y. Lee, S. Hong, H. Choi, Y. Jeong, Independently Controllable 3rd- and 5th-Order Analog Predistortion Linearizer for RF Power Amplifier in GSM, Proceedings of 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits, Fukuoka, Japan, Aug 4–5, 2004, pp. 146–149.

[25] Y. Lee, M. Lee, S. Jung, Y. Jeong, Analog Predistortion Power Amplifier Using IMD Sweet Spots for WCDMA Applications, Proceedings of 2007 Asia-Pacific Microwave Conference, Bangkok, Thailand, Dec 11–14, 2007, pp. 1–4.

[26] X. Sun, S. Cheung, T. Yuk, Design of a Fifth-order Analog Predistorter for Base Station HPA of Cellular Mobile Systems, Microwave Journal, 2011, Vol. 54, pp. 86–102.

[27] E. Westesson, L. Sundström, A Complex Polynomial Predistorter Chip in CMOS for Baseband or IF Linearization of RF Power Amplifiers, Proceedings of the 1999 IEEE International Symposium on Circuits and Systems, Orlando, FL, USA, May 30 – Jun 2, 1999, Vol. 1, pp. 206–209.

[28] E. Westesson, L. Sundström, Low-Power Complex Polynomial Predistorter Circuit in CMOS for RF Power Amplifier Linearization, Proceeding of the 27th Solid-State Circuits Conference, Villach, Austria, Sep 18–20, 2001, pp. 486–489.

[29] F. Roger, A 200mW 100MHz-to-4GHz 11th-Order Complex Analog Memory Polynomial Predistorter for Wireless Infrastructure RF Amplifiers, Proceedings of 2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers, San Francisco, CA, USA, Feb 17–21, 2013, pp. 94–95.

[30] B. Gilbert, A Precise Four-Quadrant Multiplier with Subnanosecond Response, IEEE Journal of Solid-State Circuits, 1968, Vol. 3, pp. 365–373.

[31] Y. Lee, M. Lee, S. Kam, Y. Jeong, Analog Predistortion Linearization of Doherty Power Amplifiers Using Bandwidth Reduction of Error Signal, Microwave and Optical Technology Letters, 2010, Vol. 52, pp. 1313–1316.

[32] K. Fayed, A. Ezzeddine, H. Huang, Linear Power Amplifier Uses Mirror Predistortion, High Frequency Electronics, 2011, Vol. 10, No. 6, pp. 18–25.

[33] K. Son, B. Koo, S. Hong, A CMOS Power Amplifier With a Built-In RF Predistorter for Handset Applications, IEEE Transactions on Microwave Theory and Techniques, 2012, Vol. 60, pp. 2571–2580.

[34] M. Bianchini, F. Scarselli, On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures, IEEE Transactions on Neural Networks and Learning Systems, 2014, Vol. 25, pp. 1553–1565.

[35] K. Hornik, M. Stinchcombe, H. White, Multilayer Feedforward Networks are Universal Approximators, Neural Networks, 1989, Vol. 2, pp. 359–366.

[36] G. Cybenko, Approximation by Superposition of a Sigmoidal Function, Mathematics of Control, Signals and Systems, 1989, Vol. 2, pp. 303–314.

[37] K. Hornik, Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 1991, Vol. 4, pp. 251–257.

[38] G. Tesauro, T. Seinowski, A Parallel Network that Learns to Play Backgammon, Artificial Intelligence, 1989, Vol. 39, pp. 357–390.

[39] J. McClelland, Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises, Second Edition, DRAFT, 2015.

[40] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, et al., Mastering the game of Go with deep neural networks and tree search, Nature, 2016, Vol. 529, pp. 484–489.

[41] C. Anderson, Learning to Control an Inverted Pendulum Using Neural Networks, IEEE Control Systems Magazine, 1989, Vol. 9, pp. 31–37.

[42] M. Forssell, Hardware Implementation of Artificial Neural Networks, 18-859E Information Flow in Networks, Carnegie Melon University. Available: http://users.ece.cmu.edu/~pgrover/teaching/files/NeuromorphicComputing.pdf

[43] L. Chua, Memristor – The Missing Circuit Element, IEEE Transactions on Circuit Theory, 1971, Vol. 18, pp. 734–738.

[44] F. Dias, A. Antunes, A. Mota, Proceedings of the 5th IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro, Portugal, 2013.

[45] C. Lindsey, T. Lindblad, Review of Hardware Neural Networks: A User's Perspective, Proceeding of the 3rd Workshop on Neural Networks: From Biology to High Energy Physics, Isola d'Elba, Italy, Sep 26–30, 1994.

[46] J. Lagarias, J. Reeds, M. Wright, P. Wright, Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions, SIAM Journal of Optimization, 1998, Vol. 9, pp. 112–147.

[47] M. Hagan, M. Menhaj, Training Feed-Forward Networks with the Marquardt Algorithm, IEEE Transactions on Neural Networks, 1994, Vol. 5, pp. 989–993.

[48] R. Sutton, Learning to Predict by the Methods of Temporal Differences, Machine Learning, 1988, Vol. 3, pp. 9–44.

[49] A. Lazaric, M. Restelli, A. Bonarini, Reinforcement Learning in Continuous Action Spaces Through Sequential Monte Carlo Methods, Advances in Neural Information Processing Systems 20 (NIPS 2007), 2007, pp. 1456–1463.

[50] J. Modayil, A. White, P. Pilarski, R. Sutton, Acquiring a Broad Range of Empirical Knowledge in Real Time by Temporal-Difference Learning, 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Seoul, South Korea, Oct 14–17, 2012, pp. 1903–1910.

[51] R. Sutton, A. Bonde, Nonlinear TD/Backprop Pseudo C-code, 1993. Available: https://webdocs.cs.ualberta.ca/~sutton/td-backprop-pseudo-code.text