Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

# Evaluación de un modelo basado en aprendizaje profundo para el modelado y predistorsión de amplificadores de potencia

Autor: Antonio Corral Sierra

Tutor: Luis Javier Reina Tosina

**Dpto. Teoría de la Señal y Comunicaciones**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2020

Trabajo Fin de Máster
Ingeniería de Telecomunicación

# Evaluación de un modelo basado en aprendizaje profundo para el modelado y predistorsión de amplificadores de potencia

Autor:
Antonio Corral Sierra

Tutor:
Luis Javier Reina Tosina
Profesor titular de Universidad

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Máster: Evaluación de un modelo basado en aprendizaje profundo para el modelado y predistorsión de amplificadores de potencia

Autor: Antonio Corral Sierra

Tutor: Luis Javier Reina Tosina

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

*A mi familia*

*A mis maestros*

# Agradecimientos

En primer lugar, agradecer a mi tutor y profesor Luis Javier Reina Tosina el excelente trato mostrado durante el transcurso del proyecto, su interés en cada momento y la ayuda recibida en cada una de las tutorías, que dadas las circunstancias actuales se han visto complicadas.

En segundo lugar, a mi familia por haber confiado siempre en mí y transmitirme toda la fuerza y compañía necesaria para alcanzar mis objetivos. Sin vuestra ayuda no hubiese sido posible.

*Antonio Corral Sierra*

*Sevilla, 2020*

# Resumen

Este Trabajo de Fin de Máster tiene como objetivo estudiar la nueva tendencia existente de usar algoritmos de Machine Learning para el modelado y linealización de amplificadores de potencia. En particular, de entre estos algoritmos se destacan las oportunidades que representan las redes neuronales artificiales, cuyo uso permite obtener modelos de comportamientos estimados más precisos, ya que permiten superar algunas de las limitaciones existentes de los modelos tradicionales. La memoria de este trabajo se inicia planteando el contexto en el que nos encontramos trabajando y se definen los objetivos que se esperan alcanzar. A continuación, se exponen los dos tipos de modelado existentes, presentando algunas de las técnicas más importantes del modelado a nivel de circuito y de comportamiento para amplificadores. El tercer capítulo comienza presentando los elementos característicos de las redes neuronales artificiales. Y finaliza con el análisis de los distintos tipos de redes neuronales existentes utilizadas en el modelado de amplificadores de potencia.

Basándose en el análisis del capítulo anterior, el cuarto capítulo explica la estructura de capas que compone al modelo seleccionado para el modelado y linealización del amplificador de potencia utilizado. El quinto capítulo comienza con la presentación de la plataforma sobre la que se ha trabajado, así como del software utilizado. Finaliza realizando ciertas aclaraciones sobre el código generado. Las pruebas y resultados obtenidos tanto en el modelado de comportamiento como en la linealización basada en predistorsión serán expuestos en el sexto capítulo.

El capítulo final recoge las conclusiones obtenidas en base a las pruebas realizadas y las líneas futuras de investigación. En él queda reflejado que el uso del Machine Learning en el modelado y linealización de amplificadores de potencia es ya una realidad, ofreciendo un amplio campo de investigación con resultados muy interesantes.

# Abstract

This Master's Thesis aims to study the new existing trend of using Machine Learning algorithms for the modeling and linearization of power amplifiers. Particularly, among these algorithms, it focuses on artificial neural networks, whose use allows obtaining more precise estimated behavioral models, since they can break some of the existing limitations of traditional models. The Thesis report begins by raising the context in which we are working and defining the objectives that are expected to be achieved. The two existing types of modeling are defined next, presenting some of the most important techniques of circuit-level and behavioral modeling. The third chapter begins by presenting the characteristic elements of artificial neural networks. And it ends with the analysis of the different types of existing neural networks used in the modeling of power amplifiers.

Based on the analysis of the previous chapter, the fourth chapter explains the layer structure that makes up the selected model for modeling and linearization of the power amplifier used. The fifth chapter begins with the presentation of the platform on which it has been worked, as well as the software used. It finishes by making certain clarifications about the generated code. The tests and results obtained in both behavior modeling and predistortion-based linearization will be presented in the sixth chapter.

The final chapter collects the conclusions obtained based on the tests carried out and future lines of research. It is reflected in it that the use of Machine Learning in the modeling and linearization of power amplifiers has already been carried out, offering a wide field of research with very interesting results.

# Índice

# Índice de acrónimos

| | |
|---|---|
| 5G | Quinta generación de telefonía móvil |
| ACPR | Relación de potencia en el canal adyacente |
| BB | Banda base |
| CALLUM | Modulador universal combinado de bucle analógico bloqueado |
| DPD | Predistorsión digital |
| DSP | Procesador digital de señal |
| DUT | Dispositivo bajo prueba |
| DVB | Digital Video Broadcasting |
| EE&R | Eliminación y restauración de envolvente |
| EVM | Magnitud del vector error |
| FC | Completamente conectada |
| ICA | Interferencia en el canal adyacente |
| IF | Frecuencia intermedia |
| IMS | International Microwave Symposium |
| LINC | Amplificación lineal utilizando componentes no lineales |
| LMS | Mínimo cuadrado medio |
| LS | Mínimo cuadrado |
| MP | Memory Polynomial |
| NMSE | Error cuadrático medio normalizado |
| MSE | Error cuadrático medio |
| OFDM | Multiplexación por división de frecuencias ortogonales |
| PA | Amplificador de potencia |
| SNR | Relación señal a ruido |
| PAPR | Potencia de pico a potencia promedio |
| QAM | Modulación de amplitud en cuadratura |
| RF | Radio frecuencia |
| RMS | Media cuadrática |
| VST | Transceptor de señal vectorial |
| RVTDCNN | Real-value time-delay convolutional neural network |
| I/Q | Componentes en fase y cuadratura |
| NN | Redes neuronales |
| ANN | Redes neuronales artificiales |
| CNN | Redes neuronales convolucionales |
| RNN | Redes neuronales recurrentes |
| ■ | Símbolo de convolución |

# 1 INTRODUCCIÓN

## 1.1 Motivación

Con la llegada inminente del estándar 5G, las tasas de transmisión requeridas ascenderán a múltiples Gbps y los anchos de banda se incrementarán hasta centenares de MHz. Aunque inicialmente se abordaba únicamente el servicio de voz, las tornas han cambiado y las nuevas tendencias ofrecen una amplia variedad de servicios multimedia.

Para lograr el cometido anterior las generaciones más avanzadas consideran nuevas modulaciones basadas en la utilización de envolvente no constante en detrimento de las tradicionales modulaciones de envolvente constante que ofrecen una menor tasa de transmisión. El exponencial aumento del número de usuarios ha traído consigo la reutilización de las bandas de frecuencia existentes, así como la exploración de otras nuevas. Un ejemplo de ello es el próximo estándar 802.11ax donde, además de operar en las tradicionales bandas de 2.4 y 5 GHz, se ampliará el uso del espectro a los 6 GHz. El citado aumento en la densidad de usuarios conlleva a recurrir a esquemas de modulación de alta eficiencia espectral, en los cuales la señal transmitida sufre variaciones tanto en fase como en amplitud. La incesante búsqueda por optimizar los recursos, en este caso el espectro, lleva al desarrollo de nuevas técnicas de procesado de la señal.

Uno de los dispositivos que permite el procesado eficiente de la señal y que es indispensable en los sistemas de comunicaciones inalámbricas es el amplificador de potencia (PA, del inglés Power Amplifier). Este dispositivo proporciona la potencia necesaria a la señal para su propagación por el canal y su correcta recepción en el extremo opuesto, receptor. Sin embargo, los amplificadores de potencia, en particular los de la etapa de RF (radiofrecuencia), en búsqueda de operar con una mayor eficiencia suelen trabajar en la región no lineal de su característica entrada-salida. Dicha eficiencia trae consigo consecuencias negativas como la no linealidad en su comportamiento. El reto actual consiste en desarrollar amplificadores de potencia de RF que cumplan una relación de compromiso entre eficiencia y linealidad, pudiendo de este modo satisfacer las necesidades de los nuevos estándares de comunicación.

La no linealidad de los PA provoca un ensanchamiento espectral de la señal que lo atraviesa. En comunicaciones como las móviles el espectro de RF se encuentra dividido en canales, cada uno de estos canales es asignado a un usuario durante la comunicación. Dicho ensanchamiento provoca una disminución de la relación de potencia en el canal adyacente (ACPR), degradando de este modo la calidad de la comunicación. Otro factor importante en los nuevos sistemas de comunicaciones es la eficiencia en potencia, en particular en sistemas donde la autonomía es crucial, como en el caso de los equipos móviles y satélites.

El reto al que nos enfrentamos es intentar conciliar alta linealidad con alta eficiencia espectral en potencia, lo cual se antoja complicado. Para ello existen las técnicas de linealización de amplificadores de potencia, las cuales permiten la utilización de modulaciones de envolvente no constante y amplificadores de alta eficiencia de manera conjunta a costa de aumentar la complejidad de transmisor. Algunas de estás técnicas son el *feedforward*, predistorsión digital (DPD) o linealización mediante componentes no lineales (LINC), entre otras. El rendimiento de estás técnicas se ha visto mejorado con el uso de mecanismos adaptativos realizados por los procesadores digitales de señal (DSP).

Para poder aplicar las técnicas de linealización es necesario conocer previamente el comportamiento del amplificador de potencia. El modelado de comportamiento proporciona un método efectivo para el análisis de la no linealidad y el modelado de amplificadores de potencia. Este tipo de modelado permite una recreación de "caja negra" del sistema completo. Representa la no linealidad del sistema mediante una ecuación matemática obtenida de la observación de las respuestas de entrada y salida del mismo cuando éste se excita con señales que varían mucho en el tiempo. En este modelado, la dinámica del sistema se asocia a los efectos de memoria. Existe otro tipo de modelado basado en circuito equivalente. En él se busca establecer una relación entre las tensiones y corrientes de una estructura circuital mediante un conjunto de ecuaciones no lineales. Estos son apropiados para la simulación a nivel de circuito.

Los modelos de comportamiento se pueden clasificar en modelos con memoria o sin memoria, dependiendo de si tienen en cuenta la dependencia en la función de transferencia del amplificador de muestras correspondientes a instantes pasados de la señal de entrada. Los efectos de memoria causan una distorsión tanto en frecuencia como en amplitud, haciendo que las características AM/AM y AM/PM dejen de ser funciones estáticas y provocando asimetrías entre los niveles de los canales adyacentes al principal.

Los modelos de comportamiento tradicionales con memoria, entre los cuales destacan los modelos de Volterra, han sido ampliamente utilizados en el modelado de amplificadores de potencia de banda ancha. Sin embargo, dichos modelos presentan un factor limitante en la precisión de los modelos estimados debido a la gran correlación existente entre las bases polinómicas de los mismos. Esto unido a los excelentes resultados obtenidos por las redes neuronales artificiales (ANN) en el campo de la comunicación ha atraído la atención de los investigadores en el campo del modelado de los amplificadores de potencia.

Aunque las ANN presentan un excelente rendimiento en la aproximación de funciones no lineales, cuando se encuentran frente a un amplificador de potencia que presenta características no lineales complejas y efectos de memoria, es complicado obtener buenos resultados con modelos ANN de baja complejidad. Esto se encuentra unido al escaso avance que aún existe en el modelado de PA mediante ANN, ya que es un campo en el que se está comenzado a investigar. Por tanto, el objetivo de este proyecto es intentar indagar en esa investigación con el fin de obtener un modelo basado en redes neuronales (NN) que presente baja complejidad y buena precisión en el modelado de comportamiento.

El aprendizaje profundo (Deep Learning), en el campo de la inteligencia artificial, presenta buenas perspectivas de cara al descubrimiento de relaciones no lineales complejas usando datos etiquetados. En particular, las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN). Sin embargo, el uso del Deep Learning en el modelado de comportamiento y linealización de amplificadores de potencia es muy limitado actualmente. Uno de los motivos, en el caso de las RNN, es que el algoritmo de regresión usado está diseñado para el procesamiento del habla, presentando un aumento de la complejidad cuando se usa en el modelado y linealización de amplificadores de potencia.

Por su parte, las CNN's se usan como clasificador donde su capa de salida proporciona una decisión discreta en lugar de emitir una señal continua. Éstas son utilizadas principalmente en el reconocimiento y clasificación de imágenes, sin embargo, en este proyecto se intentan adaptar las CNN para ser usadas en el modelado de comportamiento y DPD de un amplificador de potencia. La selección de este tipo de red esta fundamentada en dos motivos. Por un lado, se ha indicado anteriormente que presenta un buen comportamiento en el descubrimiento de relaciones no lineales complejas, presente en las ecuaciones que describen el comportamiento del PA. Y en segundo lugar y más importante es que logramos reducir la complejidad computacional asociada a las redes neuronales artificiales (ANN) gracias a la característica de peso compartido propia de las estructuras de convolución.

## 1.2 Objetivos

El objetivo de este proyecto es el modelado y linealización de un PA mediante DPD haciendo uso de ANN. El amplificador que se ha utilizado para la validación de los métodos que se presentan pertenece a la plataforma RF WebLab gestionada por la Universidad Tecnológica de Chalmers (Suecia). Mediante una serie de funciones implementadas en Matlab se puede hacer uso remoto del mismo, siendo común el acceso a todos los usuarios que deseen utilizarlo. En primer lugar, se estimará el comportamiento del amplificador de potencia analizando las prestaciones del modelo en la fase de modelado. Para ello se hará uso del modelo RVTDCNN propuesto por Hu, Ghannouchi y Cols [1], presentado en detalle en secciones posteriores, el cuál estimará la respuesta del PA tras ser sometido a una fase de entrenamiento. Este modelo hace uso de CNN, dichas redes son usadas principalmente en el procesado de imagen, siendo el objetivo de este proyecto adaptar los parámetros del PA al formato de estas para poder hacer uso de su potencial. Para medir la bondad de dicha estimación se comprobarán parámetros como el MSE y el NMSE resultante entre la señal medida y la modelada. Además, se comparán los espectros reales y estimado.

Una vez modelado el PA pasaremos a la segunda fase del trabajo que consistirá en la linealización de este. La técnica utilizada será la predistorsión digital (DPD), y su implementación estará basada en el modelo inverso del RVTDCNN [1]. Para poner de manifiesto la eficacia de la DPD se compararán los resultados obtenidos con los de otros modelos implementados en el TFG del autor [2], basándonos en parámetros como el ACPR (Adyacent Channel Power Ratio). Finalmente, destacar que para la implementación del modelo RVTDCNN, así como el DPD basado en su modelo inverso, se ha utilizado el lenguaje de programación Python. La elección de este lenguaje reside en su carácter abierto y exhuberante crecimiento en los últimos años, respaldado por una comunidad desarrolladora muy activa que proporciona numerosas liberías en el campo de Machine Learning.

## 1.3 Estructura de la memoria

El documento se estructura de la siguiente manera:

- **Capítulo 1:** Introducción. En él se expone la motivación de este proyecto y se definen los objetivos que se pretenden alcanzar.
- **Capítulo 2:** Predistorsión digital mediante modelos de comportamiento. Se exponen los dos tipos de linealización existentes, presentando algunas de las técnicas más destacadas en la linealización a nivel de circuito. Se continúa con una revisión de los modelos basados en series de Volterra, particularizando para los casos Memoryless y Memory Polynomial (MP). Se exponen las diferencias entre la predistorsión con aprendizaje directo y la predistorsión con aprendizaje indirecto. Y se concluye el capítulo con la presentación de algunas de las figuras de mérito utilizadas en la evaluación del modelo presentado.
- **Capítulo 3:** Redes neuronales artificiales para el modelado de amplificadores de potencia. Se centra en la explicación de los elementos que conforman una ANN, para agilizar su comprensión se presentan incialmente los conceptos de Machine Learning y Deep Learning. Se continúa con una presentación de los diferentes tipos de redes existentes, haciendo especial hincapié en las CNN (Convolutional Neural Network), explicando sus rasgos más característicos. Finalmente, se presentan algunos de los hiperparámetros más importantes cuya elección determinará la bondad de la fase de entrenamiento además de los distintos tipos de resultados que puede arrojar dicha fase.
- **Capítulo 4:** Modelo propuesto. Como su propio nombre indica, este capítulo se centra en exponer la estructura del modelo propuesto, RVTDCNN [1]. En él se indican y justifican los parámetros e hiperparámetros que lo componen. Dicho modelo será utilizado tanto en el modelado como en la

predistorsión del PA tratado.

- **Capítulo 5:** Materiales utilizados. Comienza con la presentación de la plataforma de medidas utilizada en los experimentos realizados. A continuación, se explican los elementos software utilizados. El capítulo finaliza con la explicación del código generado.
- **Capítulo 6:** Resultados. Este capítulo se divide en dos apartados: modelado del PA y predistorsión del mismo. En ambos se presentan las medidas y resultados obtenidos, así como su posterior discusión.
- **Capítulo 7:** Conclusiones y líneas futuras de investigación.

# 2 PREDISTORSIÓN DIGITAL MEDIANTE MODELOS DE COMPORTAMIENTO

## 2.1 Generalidades

Los amplificadores de potencia son los elementos que mayor potencia consumen dentro de los sistemas de RF, en torno al 70% del total, de ahí la importancia de su eficiencia. Los sistemas de comunicaciones inalámbricas de las últimas generaciones (3G, 4G, 5G) utilizan modulaciones de envolvente no constante, con valores altos de relación de potencia de pico a potencia promedio (PAPR), lo que dificulta el aprovechamiento del margen dinámico de estos. En este ámbito es donde cobran importancia las técnicas de linealización, ya que permiten extender el rango de comportamiento lineal de PA, sin sacrificar su eficiencia al mantener el punto de trabajo en la zona de saturación.

En este capítulo se presentarán las diferencias existentes entre los dos tipos de linealización: a nivel de circuito o a nivel de sistema, presentando algunas técnicas basadas en el segundo tipo. Finalmente, se presentarán algunas de las figuras de mérito empleadas en la evaluación del modelo analizado. La redacción de este capítulo se apoya en la investigación realizada el autor en su Trabajo fin de Grado [2].

### 2.1.1 Linealización a nivel de circuito y a nivel de sistema

Atendiendo al compromiso linealidad-eficiencia, existen diversas técnicas que permiten la linealización en el proceso de amplificación sin degradar en exceso la eficiencia del PA.

La linealización puede llevarse a cabo a dos niveles: circuito y sistema. Este Trabajo Fin de Máster centra su investigación en la segunda, donde consideramos el amplificador como un modelo de "caja negra", siendo indiferente su estructura interna. Estos modelos permiten describir fenómenos cuando no se conocen todos procesos físicos involucrados. A nivel circuital, los sistemas de comunicaciones modernos e inalámbricos son a menudo demasiado complejos, dependiendo de un gran número de componentes y parámetros, lo que supone un serio problema tanto en tiempo de simulación como en recursos de memoria. Como alternativa, a menudo se reemplaza la descripción circuital de los subsistemas, como en el caso del PA, por una representación simplificada pero suficientemente precisa, denominada modelo de comportamiento. En este tipo de modelo a nivel de sistema, como hemos dicho anteriormente, los dispositivos modelados se consideran como una caja negra, obteniéndose la información del modelo a partir de las respuestas externas del dispositivo.

Los modelos de comportamiento permiten la simulación completa a un nivel de abstracción superior, pero conservando la precisión en la representación del funcionamiento. A su vez, estos modelos de comportamiento proporcionan una forma de proteger la propiedad intelectual asociada a un circuito o bloque funcional, al encapsular las características del mismo. Además, ofrecen la posibilidad de diseñar un sistema incluso antes de tener disponibles los componentes hardware de este. La linealización a nivel de sistema permite reducir los efectos del comportamiento no lineal. A continuación, se presentan algunas de las técnicas de linealización a nivel de sistema existentes.

#### 2.1.1.1 Feedback

El esquema clásico de retroalimentación o *feedback* se muestra en la figura 2-1:

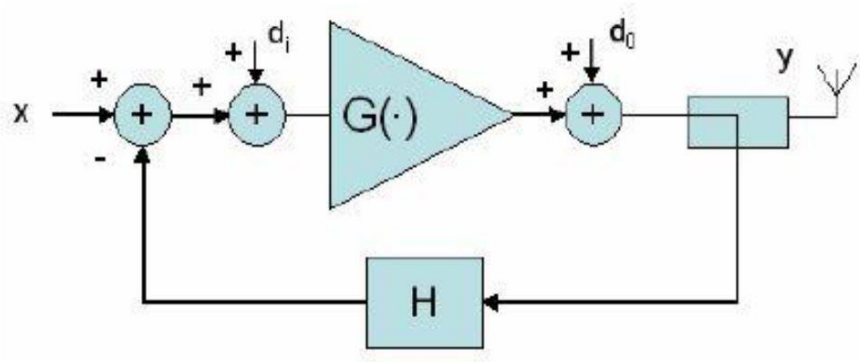


Figura 2-1. Esquema Feedback [3].

En él, las señales $d_i$ y $d_o$ representan las perturbaciones que se añaden a la entrada y salida del amplificador. La función de transferencia en bucle cerrado se considera lineal y la salida del sistema se obtiene como:

$$y = \frac{G}{1 + G \cdot H} \cdot x + \frac{G}{1 + G \cdot H} \cdot d_i + \frac{G}{1 + G \cdot H} \cdot d_o \tag{2–1}$$

donde $G$ es la función de transferencia del amplificador y $H$ es la función de transferencia del lazo de retroalimentación. Observando la función de salida se aprecia que el efecto de la perturbación de entrada, $d_i$, puede reducirse aumentando el valor de $H$, mientras que, para reducir la perturbación de salida, $d_o$, se puede aumentar $H$ o $G$. Si únicamente se considera la parte correspondiente a la entrada, $x$, se puede definir la función de transferencia del conjunto en frecuencia para las condiciones de estabilidad de la siguiente manera:

$$W = \frac{Y}{X} = \frac{G}{1 + G \cdot H} \tag{2–2}$$

La función $W$ será inestable siempre que $G \cdot H = -1$, o expresado en términos complejos, el módulo del producto sea unitario y su fase $\pm 180°$. Este producto representa la función de transferencia en bucle abierto y define las condiciones de estabilidad del sistema. Una vez analizado el concepto de *feedback*, éste es comúnmente utilizado para la linealización de amplificadores de la siguiente manera: las señales de entrada y salida son detectadas y filtradas pasobajo. Las señales resultantes en banda base son comparadas, definiendo una señal de error $V_e$, que es utilizada para controlar la ganancia del amplificador mediante un atenuador variable. Este control de la ganancia permite modificar las características del amplificador teniendo como objetivo reducir la distorsión. La figura 2-2 aclara lo explicado anteriormente. Este esquema se aplica en comunicaciones de banda estrecha ya que presenta dificultades para responder a cambios en caso de utilizar envolventes con anchos de banda de varios megahercios.

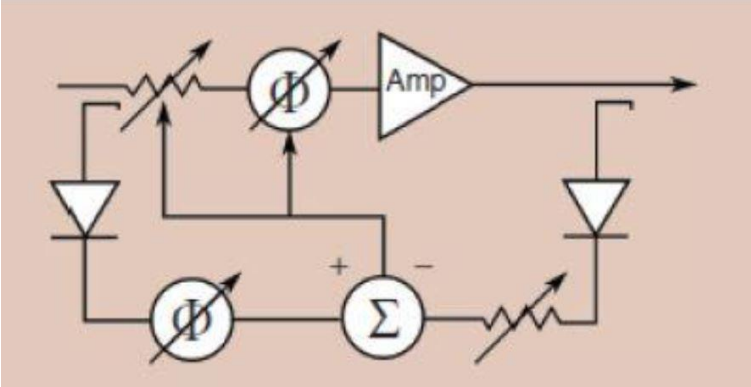


Figura 2-2. Esquema de linealizador *feedback* [4].

#### 2.1.1.2 Feedforward

El esquema típico del linealizador *feedforward* se muestra en la figura 2-3:

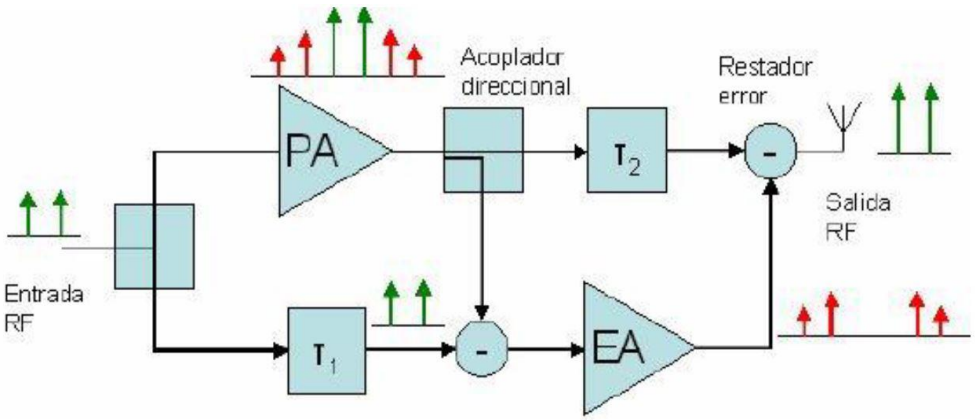


Figura 2-3. Esquema del linealizador *Feedforward* [3].

Como entrada se tiene una señal de dos tonos, que se bifurca en dos ramas, la superior y la inferior. En la rama superior la señal de entrada atraviesa el PA obteniéndose a su salida la señal amplificada más los productos de intermodulación y armónicos provocados por la no linealidad del amplificador. Una versión atenuada de la señal distorsionada por el amplificador se acopla a la rama inferior. En la rama inferior la señal de entrada ha sido retrasada un tiempo ($T_1$) equivalente al retardo inherente al amplificador, dicha señal se resta con la procedente de la rama superior, provocando que a la salida del restador se obtenga la distorsión provocada por el amplificador de potencia de la rama superior. Esta señal de distorsión es amplificada y pasada a la rama superior, mientras tanto en dicha rama superior la señal de salida del amplificador (señal amplificada + distorsión) es retrasada un retraso ($T_2$) equivalente al tiempo de operación del amplificador de la rama inferior. Ambas señales son restadas obteniéndose a la salida la señal inicial de dos tonos amplificada.

La técnica es estable, pero se deben tener en cuenta las tolerancias de los componentes, retardos, equilibrios entre los dos caminos, que deben ser compensados para evitar una mayor degradación de la señal. Existen versiones mejoradas de la estructura clásica [5].

#### 2.1.1.3 LINC (Linear Amplification Using Nonlinear Components)

Este método se basa en la separación de la señal de entrada en dos señales de envolvente constante y modulada en fase que, tras pasar cada una de ellas por sus correspondientes amplificadores de potencia no lineales y ser sumadas, dan como resultado una señal de salida lineal. La limitación de este método reside en el ancho de banda de las dos nuevas señales en las que se divide la original, que son del orden de diez veces mayor que la primera. El esquema de este linealizador se muestra en la figura 2-4:

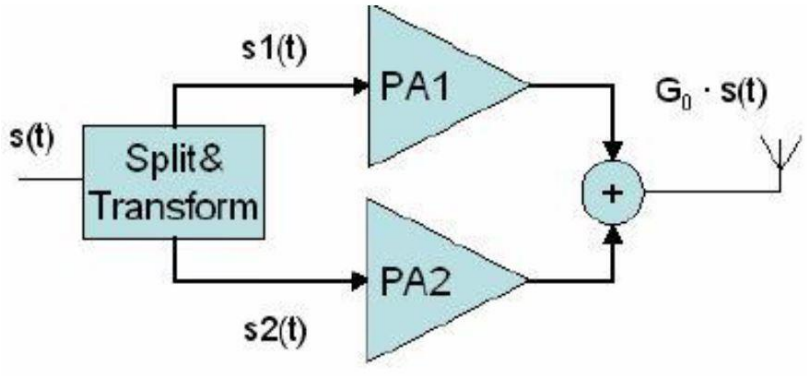


Figura 2-4. Esquema del linealizador LINC [3].

#### 2.1.1.4 CALLUM (Combined Analog-Locked Loop Universal Modulator)

Se trata de una modificación del método LINC que consiste en dividir la señal de entrada en sus componentes de fase I y cuadratura Q para ser comparadas con las componentes de fase y cuadratura retroalimentadas de la salida de los dos amplificadores de potencia del caso LINC. La señal de error de cada una de las componentes controla un oscilador controlado por tensión (VCO) cuya salida es la entrada de cada amplificador (véase la figura 2-5). Esta técnica presenta problemas de estabilidad, y de ahí que su uso quede limitado a aplicaciones de banda estrecha.

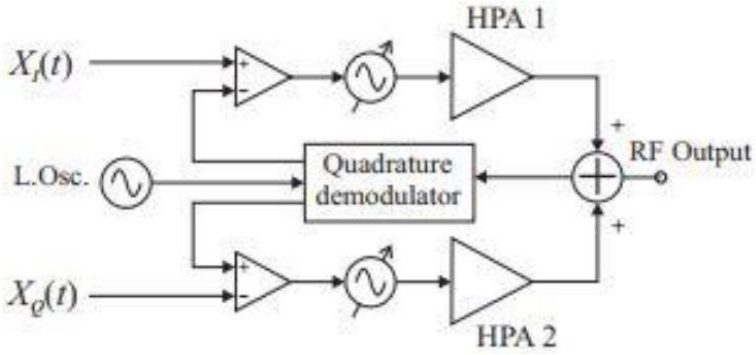


Figura 2-5. Esquema simplificado del linealizador CALLUM.

#### 2.1.1.5 EE&R (Envelope Elimination and Restoration)

La señal modulada de entrada se divide en dos señales (figura 2-6): una primera que contiene la información de

envolvente y la segunda que contiene la información de fase. La señal con la información correspondiente a la fase tiene amplitud constante y es la que se introduce en el PA, mientras que la señal que contiene la información referente a la envolvente, que es de amplitud variable, modula la tensión de alimentación del amplificador. Esta técnica ha sido utilizada en sistemas OFDM y DVB, entre otros.

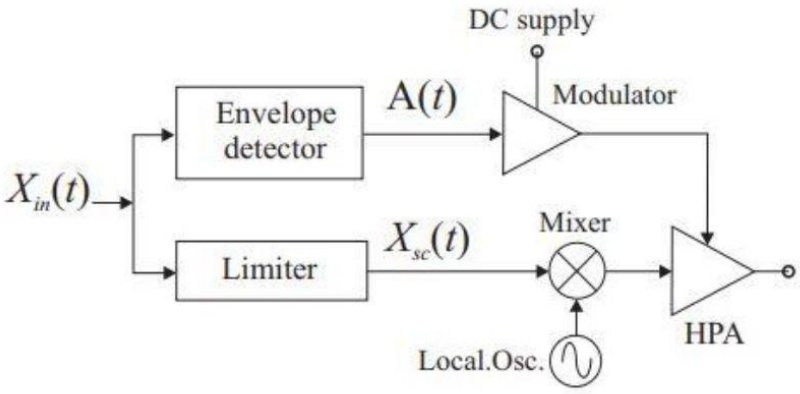


Figura 2-6. Esquema del linealizador EE&R [6].

#### 2.1.1.6 Predistorsión digital

Es la técnica de linealización aplicada en este Trabajo Fin de Máster. Consiste en modificar la señal de entrada del PA, mediante su paso previo a través del predistorsionador. Este bloque aplica sobre la señal de entrada la función inversa de la introducida por el PA (figura 2-7). De esta manera, la salida del amplificador se aproxima a la característica ideal.

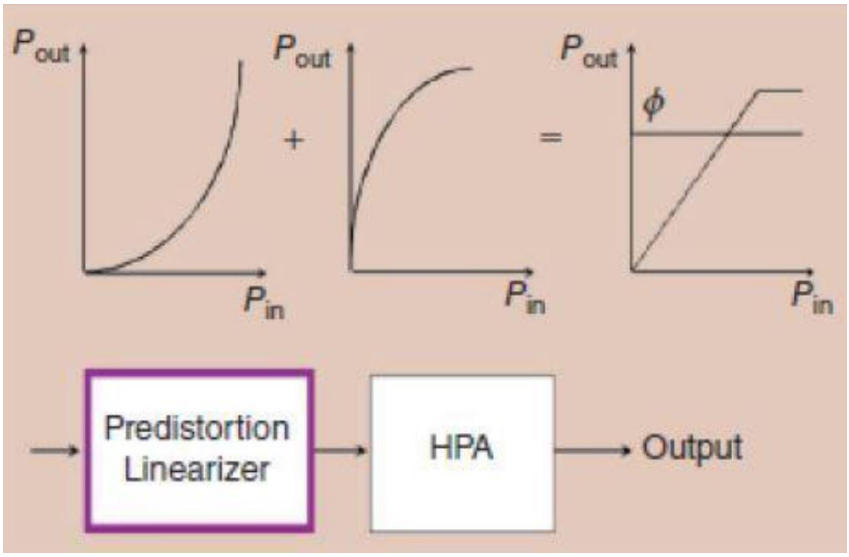


Figura 2-7. Linealización mediante predistorsión digital [4].

Las técnicas de predistorsión digital pueden aplicarse sobre la señal o los datos. La predistorsión aplicada sobre los datos es una técnica específica de cada tipo de modulación, y pretende compensar el espacio vectorial de los datos o constelación. Los coeficientes del predistorsionador se optimizan minimizando el parámetro EVM (del inglés, Error Vector Magnitude). El gran inconveniente de este tipo de predistorsión radica en que no tiene en

cuenta la distorsión fuera de banda, ya que actúa directamente sobre los datos y esto supone un problema para cumplir determinadas especificaciones de los estándares como por ejemplo los niveles de ICA. Otro de los problemas de esta técnica es que es específica para cada tipo de modulación.

Por su parte, la predistorsión digital aplicada sobre la señal genera una señal predistorsionada que, al pasar a través del PA, el cual produce sobre la señal la función inversa a la del predistorsionador, no debería tener distorsión a la salida, es decir, convertiría al amplificador de potencia en un dispositivo ideal lineal hasta el punto de saturación. Esta técnica, a diferencia de la anterior, tiene en cuenta tanto la distorsión en banda como fuera de ella.

La predistorsión de señal puede ser realizada en radiofrecuencia (RF), frecuencia intermedia (IF) o banda base (BB), siendo preferible los dos últimos casos. En IF o BB, los costes de los convertidores ADC o DCA son menores y además es más robusta la predistorsión. Su inconveniente es que en el paso a RF los mezcladores pueden introducir alguna forma de no linealidad.

Inicialmente, los predistorsionadores se diseñaban para compensar el comportamiento no lineal instantáneo del PA mediante predistorsión instantánea, es decir, no se tenían en cuenta los efectos de memoria. El comportamiento instantáneo queda determinado por las curvas estáticas AM/AM y AM/PM, debido a que la amplitud y la fase se consideraban funciones de los valores instantáneos de la señal de entrada.

Las modulaciones digitales multinivel utilizadas en los actuales estándares de comunicación requieren de un mayor ancho de banda en BB, y por lo tanto los modelos sin memoria no son adecuados para dicho propósito. Para este cometido se utilizan modelos que tengan en cuenta los efectos de memoria, y las curvas AM/AM y AM/PM presentan un comportamiento dinámico que pretende ser compensado por este tipo de modelos. Su valor se modifica dinámicamente con el tiempo y se pasa de tener una sola curva a un conjunto de éstas.

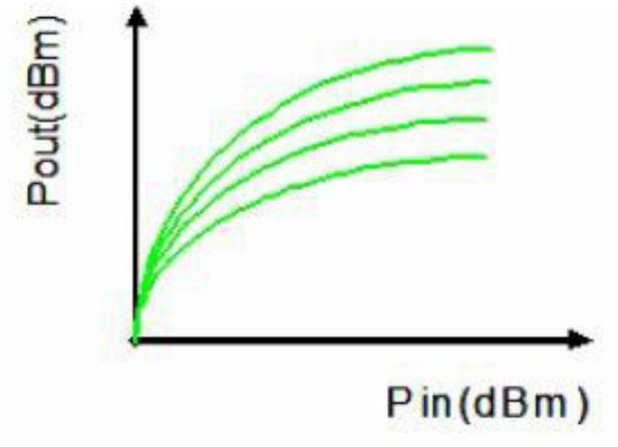


Figura 2-8. Curvas AM/AM considerando los efectos de memoria [5].

En el primer apartado de este trabajo se citaron las limitaciones que presentaban las series de Volterra en el modelado, debido a la correlación existente entre sus bases polinómicas. No obstante, el enfoque tradicional para representar una función no lineal con memoria es mediante una serie de Volterra. Por dicho motivo, en el próximo apartado se analizarán y se presentarán un par de modelos basados en éstas.

## 2.2 Modelos de Volterra

El matemático italiano Vito Volterra (1860-1940) basó sus estudios en el desarrollo de la rama de matemáticas conocida como análisis funcional. Sin embargo, la aplicación de dichas series en la ingeniería de sistemas se debe a Norbert Wiener, quien desarrolló la hipótesis de que un sistema no lineal invariante en el tiempo podía ser descrito mediante una serie de parámetros funcionales de Volterra, siempre y cuando el sistema a caracterizar no presentase no linealidades demasiado fuertes. Además de lo anterior, Wiener también propuso un método para identificar los coeficientes de las funciones de Volterra utilizando ruido blanco. En los años 60 la teoría de Volterra se aplicó a los circuitos no lineales y sistemas de comunicaciones. Sin embargo, a partir de los 70 su uso decayó debido a la aparición de dispositivos que conseguían mejoras en la linealidad de los circuitos y sistemas. La teoría de Volterra se redescubrió en los años 80 cuando la distorsión en los circuitos de comunicaciones comenzó a ser un problema de interés. La reaparición de esta teoría fue consecuencia del uso de dispositivos de bajo ruido y alta frecuencia que ofrecen unas buenas prestaciones, pero no son muy lineales. Pero no es hasta la actualidad cuando las series de Volterra alcanzan una gran dimensión y ello se debe al reciente interés por obtener modelos de comportamiento de sistemas no lineales.

Las series de Volterra son similares a las series de Taylor, hasta el punto de que se pueden ver como una serie de Taylor con memoria. Las series de Volterra deben ser convergentes, y para ello deben cumplir con algún criterio de suavidad. En la práctica, el número de términos necesarios para cumplir la condición de convergencia puede ser muy elevado.

La carga computacional requerida para el cálculo de los coeficientes de una serie de Volterra está ligada al grado del sistema polinomial que describe el sistema no lineal. Es decir, cuando el grado de éste aumenta también lo hace la carga computacional. Por este motivo, en la práctica suele truncarse el grado del sistema polinomial.

Los sistemas que se modelan utilizando las series de Volterra son cuasilineales, es decir, las no linealidades son débiles y por tanto una aproximación de bajo orden se considera apropiada.

Los modelos de Volterra llevan siendo estudiados durante mucho tiempo y poseen una amplia base teórica. Permiten modelar el comportamiento de sistemas no lineales con la particularidad de que las no linealidades de estos deben ser de carácter débil. Sin embargo, esto no representa una desventaja ya que los sistemas de comunicaciones actuales poseen un comportamiento no lineal a modelar que se encuentra situado en niveles bajos de la señal de distorsión. Lo que sí representa una desventaja para los métodos de Volterra es su complejidad computacional, pero como se ha visto anteriormente esta complejidad puede ser tratada mediante el truncamiento del grado del polinomio que describe la no linealidad.

Una serie de Volterra es la combinación de una convolución lineal y una serie de potencias no lineal que se utiliza para describir la relación entrada-salida de un sistema invariante en el tiempo causal no lineal con memoria desvaneciente. En el dominio del tiempo discreto, una serie de Volterra se expresa como:

$$y(l) = \sum_{p=1}^{\infty} \sum_{i_1=0}^{\infty} \dots \sum_{i_p=1}^{\infty} h_p(i_1, \dots, i_p) \prod_{j=1}^{p} x(l - i_j) \tag{2–3}$$

Donde $x(l)$ e $y(l)$ representa la entrada y salida respectivamente, y a $h_p(i_1, \dots, i_p)$ se le denomina kernel o núcleo de Volterra de orden *p*. Un sistema de Volterra estará caracterizado mediante el conocimiento de sus kernels. Las series de Volterra se truncan generalmente hasta un orden finito no lineal *P* y una memoria finita

*Q*. Se conoce que la componente par de la intermodulación no interfiere con la banda de señal y los coeficientes de la serie de Volterra pueden suponerse simétricos sin pérdida de generalidad. Por dicho motivo, sólo los órdenes impares de la serie de Volterra han de ser considerados en una representación de BB.

Las series de Volterra se emplean para describir la relación entre la entrada y la salida de un amplificador con memoria de forma general. La ventaja de las series de Volterra es que la salida del modelo de Volterra es lineal con respecto a sus coeficientes, como ocurre con el modelo polinomial. Bajo el supuesto de estacionaridad, si calculamos los coeficientes con el criterio de media mínima o mínimo error cuadrático, tendremos un mínimo global único. Por lo tanto, es posible extraer el modelo no lineal de forma directa mediante el uso de algoritmos de regresión lineal, por ejemplo, mínimos cuadrados. Por otra parte, la desventaja de los sistemas que emplean series de Volterra reside en la complejidad computacional y en el gran número de parámetros que hay que calcular, siendo muchos de ellos innecesarios. Las líneas actuales de investigación sobre modelado basado en series de Volterra siguen apostando por modelos basados en coeficientes para determinar la no linealidad, pero buscan omitir aquellos coeficientes que tienen una menor influencia en el comportamiento no lineal. De este modo se obtiene un modelo más sencillo que requiere de una menor carga computacional.

A continuación, se muestran dos simplificaciones para las series de Volterra en BB, el modelo sin memoria (Memoryless) y el Memory Polynomial (MP).

### 2.2.1 Modelo sin memoria (ML)

Se caracteriza porque la salida únicamente depende de instante actual (muestra actual), es decir, no se consideran muestras anteriores. La implementación de este modelo viene descrita por la siguiente ecuación:

$$y(l) = \sum_{k=0}^{K} a_k \cdot x(l) \cdot |x(l)|^{2k} \tag{2–4}$$

Donde $x(l)$ e $y(l)$ representa la entrada y la salida del sistema, $a_k$ los coeficientes del modelo y *P* representa el orden de la no linealidad expresado como $P = 2 \cdot K + 1$.

### 2.2.2 Memory Polynomial (MP)

También conocido como modelo de Volterra diagonal, introduce pares de muestras retrasadas respecto a la entrada hasta un orden *P* para describir los efectos no lineales y de memoria. La profundidad de la memoria viene determinada por el valor de *Q*. El modelo queda como sigue:

$$y_{MP} = \sum_{m=0}^{Q} \sum_{k=0}^{K} a_{m,k} \cdot x(l - \tau_m) \cdot |x(l - \tau_m)|^{2k} \tag{2–5}$$

Donde $a_{m,k}$ son los coeficientes del modelo. *P*, como se indicó anteriormente, es el orden de la no linealidad y

viene descrito por la ecuación: $P = 2 \cdot K + 1$. El retraso del modelo se asume como $\tau_m = m \cdot \tau_0$ donde $\tau_0$ es el periodo de muestreo de la señal de entrada. Este modelo puede implementarse mediante el uso de un grupo de bancos de filtros. El hecho de que todos los términos fuera de la diagonal de la serie de Volterra se igualen a cero, justifica que se le denomine modelo de Volterra diagonal. Este modelo tiene sus ventajas e inconvenientes. Por un lado, reduce enormemente la complejidad de la serie de Volterra completa, lo cual es positivo, pero por ende disminuye la fidelidad de este ya que en algunos casos los términos fuera de la diagonal, llamados términos cruzados, pueden jugar un papel importante en el comportamiento del amplificador.

## 2.3 Métodos de aprendizaje para predistorsionadores

En este apartado se presentan las dos configuraciones básicas para la estimación de los coeficientes del predistorsionador.

### 2.3.1 Predistorsión con aprendizaje directo

El funcionamiento del DPD mediante el método directo viene descrito por el diagrama de bloques de la figura 2-9:

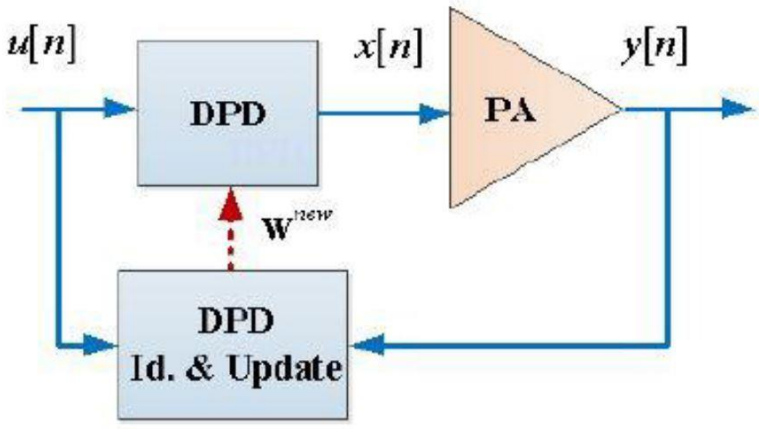


Figura 2-9. Esquema de predistorsión digital con aprendizaje directo [7].

La predistorsión realizada en el DPD conforme a la notación del diagrama de bloques (figura 2-9) pero utilizando como variable de tiempo discreto *l* en lugar de *n*, se puede expresar como:

$$x[l] = u[l] - \mathbf{U} \cdot \mathbf{w} \tag{2–6}$$

Donde **w** es el vector de coeficientes del modelo y **U** es la matriz de regresores o bases del modelo de comportamiento que se esté utilizando. El bloque de adaptación (denotado por DPD Id. & Update en la figura 2-9) es el que se encarga de la identificación y actualización de los coeficientes del DPD. La extracción de estos se realiza de manera iterativa mediante algoritmos adaptativos tipo LMS, donde en cada iteración *i* se intenta minimizar el error residual, siguiendo la ecuación (2-7):

$$\mathbf{w}^{i+1} = \mathbf{w}^{i} + \mu\Delta\mathbf{w} \tag{2–7}$$

Donde $\mu$ ($0 \leq \mu \leq 1$) es un factor de peso y $\Delta\mathbf{w}$ se obtiene como:

$$\Delta\mathbf{w} = (\mathbf{U}^{H} \cdot \mathbf{U})^{-1} \cdot \mathbf{U}^{H} \cdot \mathbf{e} \tag{2–8}$$

donde **U** es la matriz de regresores y **e** el vector de error que se define (siguiendo la notación de la figura 2-9) como

$$e[l] = \frac{y[l]}{G_0} - u[l] \tag{2–9}$$

Siendo $G_0$ la ganancia lineal deseada del PA.

El método directo es muy similar al indirecto, con la diferencia de que en el primero se busca que los coeficientes converjan hacia su valor óptimo de manera iterativa.

### 2.3.2 Predistorsión con aprendizaje indirecto

Con este método, se hace una estimación del modelo inverso del amplificador de potencia a partir de los datos de entrada y de salida. Cuando el proceso de aprendizaje finaliza, los coeficientes del post-distorsionador se copian idénticamente en el modelo del predistorsionador en la entrada.

Con esta configuración no es necesario calcular el modelo del amplificador de potencia, sino que basta con disponer de su entrada y salida para estimar el modelo del post-distorsionador. La ecuación (2-10) muestra dicho funcionamiento:

$$F_{post}\big(G(x)\big) = G_0 \cdot x \tag{2–10}$$

La figura 2-10 muestra un esquema del método de aprendizaje indirecto:

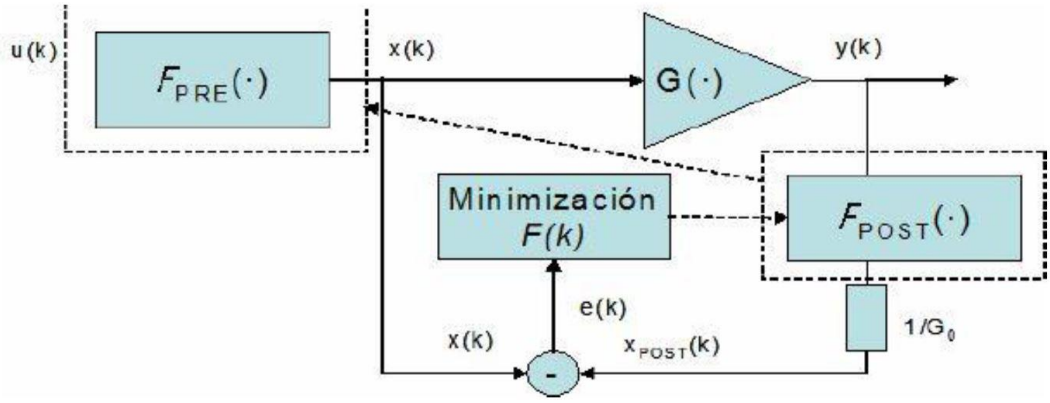


Figura 2-10. Esquema de predistorsionador con aprendizaje indirecto [3].

## 2.4 Figuras de mérito

En este apartado se presentan algunas de las figuras de mérito utilizadas para la evaluación tanto del modelado de comportamiento del PA como su linealización. Éstas son el EVM, el NMSE, el ACPR y las características AM/AM y AM/PM.

### 2.4.1 EVM (Error Vector Magnitude)

Este parámetro se determina a la salida del filtro RRC del demodulador mediante la comparación entre el valor de símbolo recibido y el valor esperado idealmente. Se puede definir como el error RMS (Root Mean Square) de la diferencia entre los valores de símbolos recibidos y esperados.

La EVM se define en la ecuación (2-11), donde I y Q son las components en fase y cuadratura de los símbolos. $\Delta I$ es la diferencia entre la componente $I$ de la señal recibida y la esperada. $N$ es el número total de símbolos en la medida. $S$ es la potencia media de los símbolos esperados o de referencia.

$$EVM = \sqrt{\frac{\frac{1}{N}\cdot\sum_{j=1}^{N}(\Delta I_j^2 + \Delta Q_j^2)}{S_{media}^2}}[\%] \tag{2–11}$$

El valor de EVM nos da una medida de cómo la no linealidad afecta al proceso de detección. Se suele expresar en porcentaje y normalizado para permitir la comparación entre las distintas modulaciones.

La figura 2-11 muestra gráficamente el parámetro EVM [8].

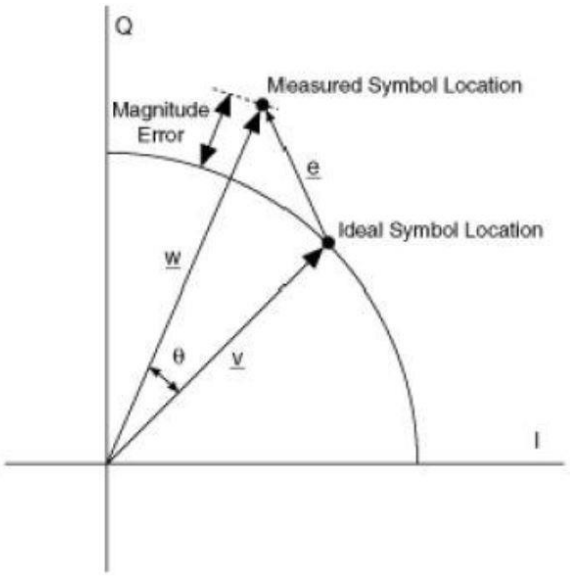


Figura 2-11. Representación gráfica del parámetro EVM.

### 2.4.2 ACPR (Adjacent Channel Power Ratio)

Este parámetro será utilizado para comparar el performance del DPD implementado con el modelo inverso de RVTDCNN frente al performance de algún modelo estándar. El ACPR nos da una medida de la potencia de la señal fuera de la banda, normalizada con respecto a la potencia de la señal dentro de la banda. Así pues, caracteriza cómo las no linealidades de la señal afectan a los canales adyacentes. La potencia en el canal adyacente debida a la señal en banda del canal considerado será vista como una señal interferente para la señal en banda del canal adyacente. El ACPR viene dado por la ecuación (2-12):

$$ACPR(dB) = 10 \cdot \log_{10} \sum_{k=1}^{K} \left| \frac{Y^{out-band}[k]}{Y^{in-band}[k]} \right|^2 \tag{2–12}$$

Donde, $Y^{out-band}$ es el espectro de la señal fuera de la banda e $Y^{in-band}$ es el espectro de la señal dentro de la banda. Cuando nos referimos al espectro, estamos hablando de la Transformada Discreta de Fourier (DFT).

### 2.4.3 NMSE (Normalised Mean Square Error)

El NMSE nos da una medida del parecido entre dos señales en el dominio del tiempo, normalizado por la potencia de la señal utilizada como referencia. En el contexto de linealización de amplificadores, se utilizará para valorar el parecido entre la salida normalizada del amplificador linealizado y su entrada. Su expression viene dada por la ecuación (2-13):

$$NMSE(dB) = 10 \cdot \log_{10} \frac{1}{N} \cdot \sum_{n=1}^{N} \left| \frac{y[n] - x[n]}{x[n]} \right|^2 \tag{2–13}$$

donde, **x** son las muestras de la señal a la entrada, **y** son las muestras de la señal a la salida, normalizada, y $N$ es el número total de muestras.

### 2.4.4 Características AM/AM y AM/PM

La curva de la característica AM/AM muestra una conversión entre la modulación de amplitud en la señal de entrada con respecto a la modulación de amplitud modificada en la señal de salida. En otras palabras, presenta la relación existente entre la potencia de entrada y la potencia de salida. Por su parte, la característica AM/PM es la conversión de la modulación de amplitud de la señal de entrada a la modulación de fase de la señal de salida.

En la curva AM/AM, encontramos que, para niveles muy elevados de potencia de entrada, el amplificador entra en la zona de saturación, de modo que la relación entre la potencia de entrada y salida deja de ser lineal. Mientras que la curva AM/PM muestra como a altos niveles de potencia de entrada la relación entre las fases deja de ser constante. Por último, destacar que la presencia de dispersión en las características AM/AM – AM/PM es consecuencia de los efectos de memoria, este tipo de características son las usuales en PA que trabajan con señales digitales de amplitud no constante. La figura 2-12 muestra un ejemplo de ambas curvas:

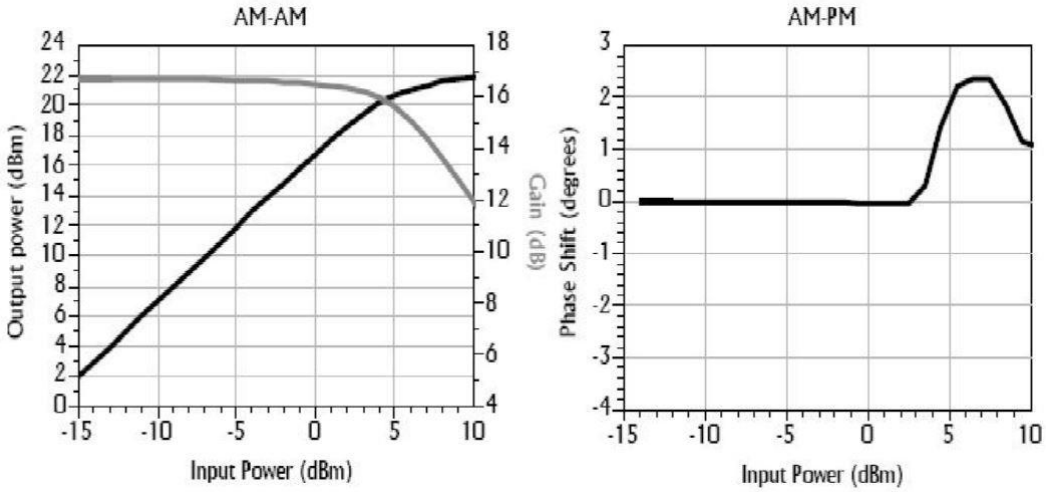


Figura 2-12. Curvas AM/AM y AM/PM [7].

# 3 ANN PARA EL MODELADO DE PA

Como se indicó en el primer capítulo, las ANN presentan un excelente comportamiento en la aproximación de funciones no lineales de ahí que se hayan comenzado a utilizar en el modelado de PA. A lo largo de este capítulo se explicarán en detalle los elementos que componen una red neuronal artificial, para facilitar la comprensión de estos se analizaran previamente los conceptos de Deep Learning y Machine Learning. Finalmente, se expondrán los distintos tipos de ANNs existentes destacando las ventajas e inconvenientes del uso de cada una de ellas en el modelado de comportamiento.

## 3.1 Deep Learning

Se define como un algoritmo automático estructurado o jerárquico que intenta emular el aprendizaje del cerebro humano, caracterizándose por un procesamiento de los datos en cascada. El Deep Learning [9] es una subparte de una de las áreas de la inteligencia artificial conocida como Aprendizaje Automático (Machine Learning).

La siguiente figura resume visualmente la idea intuitiva de que Deep Learning es solo una parte de la inteligencia artificial, aunque en estos momentos quizás la más dinámica dentro de la comunidad científica.

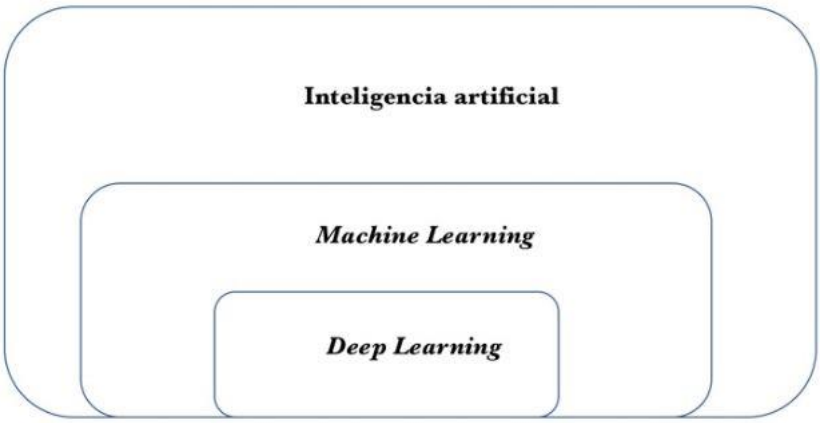


Figura 3-1. Ubicación del Deep Learning.

## 3.2 Machine Learning

El Machine Learning [10] es una disciplina científica del ámbito de la Inteligencia Artificial que crea sistemas que aprenden automáticamente. Es decir, dota a los ordenadores de la capacidad de aprender sin ser programados explícitamente para ello. Aprender en este contexto quiere decir extraer patrones de grandes volúmenes de datos. Para cada problema se desarrolla un algoritmo de predicción que analizando los datos es capaz de predecir comportamientos futuros y clasificar elementos.

Dentro del Machine Learning existen varios enfoques bien diferenciados. Cada uno de ellos utiliza una estructura algorítmica diferente con el fin de adaptarse de la manera más idónea a los datos recibidos y así poder optimizar las predicciones. Se distinguen principalmente dos categorías:

- **Aprendizaje supervisado:** Intenta deducir el algoritmo de predicción a partir de los datos de entrenamiento. Dichos datos contienen la solución, denominada "etiqueta". Consisten en pares de objetos (normalmente vectores) en los cuales una componente del par son los datos de entrada y la otra los datos deseados. Algunos de los algoritmos más populares en esta categoría son la regresión lineal, la regresión logística, Support Vector Machines (SVM), random forest y redes neuronales.
- **Aprendizaje no supervisado:** En este tipo de aprendizaje los datos utilizados en el entrenamiento no contienen las etiquetas, y es el algoritmo el encargado de clasificar la información por sí mismo. Algunos de los algoritmos más conocidos dentro de esta categoría son clustering (K-means) o Principal Component Analysis (PCA).

## 3.3 Redes Neuronales Artificiales

Como se ha visto en la clasificación del apartado anterior, un caso especial de los algoritmos de Machine Learning son las redes neuronales artificiales (ANNs) [11].

### 3.3.1 Arquitectura

Los modelos están compuestos de múltiples capas de procesamiento para aprendizaje de datos, con múltiples niveles de abstracción que realizan una serie de transformaciones lineales y no lineales que a partir de los datos de entrada proporcionados generan una salida próxima a la esperada (label). El aprendizaje supervisado consiste en la obtención de los parámetros de la red neuronal (pesos y sesgo) que proporcionen unos mejores resultados, es decir, consigan unas tranformaciones óptimas de la entrada para que la salida producida y la esperada difieran muy poco.

Un ejemplo gráfico de una red neuronal Deep Learning es:

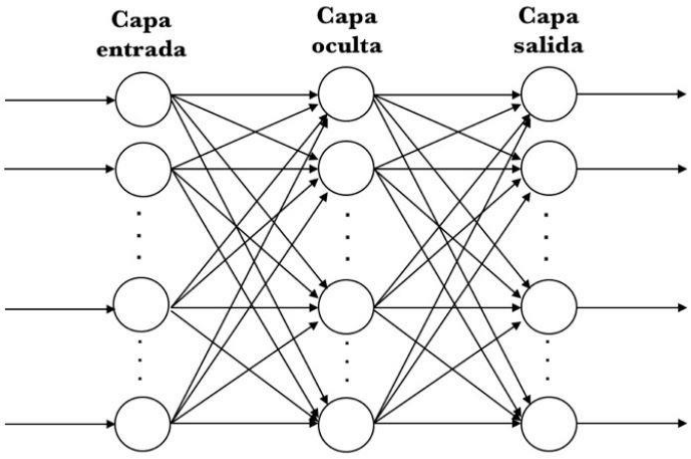


Figura 3-2. Estructura de una red neuronal.

La figura 3-2 representa una red neuronal de 3 capas: una de entrada (input layer) encargada de recibir los datos de entrada y una de salida (output layer) que devuelve la predicción realizada. La capa intermedia recibe el nombre de capa oculta (hidden layer) y pueden existir tantas como se desee, cada una de ellas con distinto número de neuronas. Las neuronas que conforman las capas pueden estar interconectadas unas con otras de distinta manera en cada una de las interfaces, entendiendo por interfaces cada par de capas que conforman la red.

Actualmente se manejan redes neuronales compuestas por muchísimas capas ocultas, de ahí el término Deep, cada una de ellas está compuesta por muchas neuronas, las cuales poseen sus propios parámetros (pesos y sesgo) que realizan una transformación simple de los datos que reciben de las neuronas de la capa anterior para pasarlos a las neuronas de la capa posterior. La unión de todas permite descrubir patrones complejos.

Para analizar más en detalle los conceptos de peso y sesgo particularizaremos para el tipo de red más simple, formada por una sola capa de una única neurona con varias entradas y una única salida. Ésta conforma la unidad básica de una red funcional y recibe el nombre de perceptrón (véase la figura 3-3).

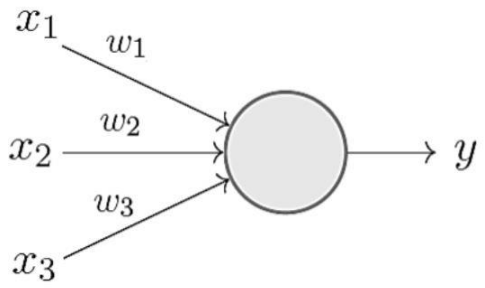


Figura 3-3. Esquema simple de un perceptrón.

Como se puede observar la neurona recibe los valores de entrada ($x_i$) los cuales tienen un peso asociado ($w_i$). El peso indica la ponderación asociada a cada una de las entradas, es decir, la "importancia" que tiene cada uno de los valores de entrada en la obtención del resultado estimado. La explicación del sesgo se expondrá en un apartado posterior. Ambos parámetros deben aprenderse durante el proceso de entrenamiento.

### 3.3.2 Función de activación

Además de las operaciones de multiplicación y suma existe una operación que realizan todas las neuronas denominada función de activación. Se utiliza para propagar hacia adelante la salida de una neurona y sirve para introducir la no linealidad en las capacidades de modelado de la red (véase la figura 3-4).

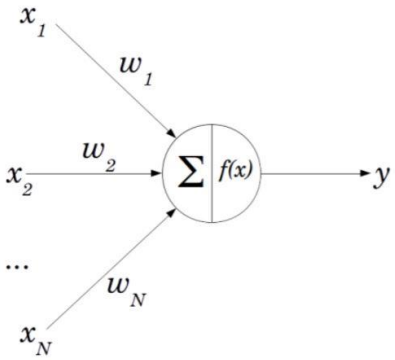


Figura 3-4. Esquema completo del perceptón.

Algunas de las funciones de activación más utilizadas se resumen en la figura 3-5.

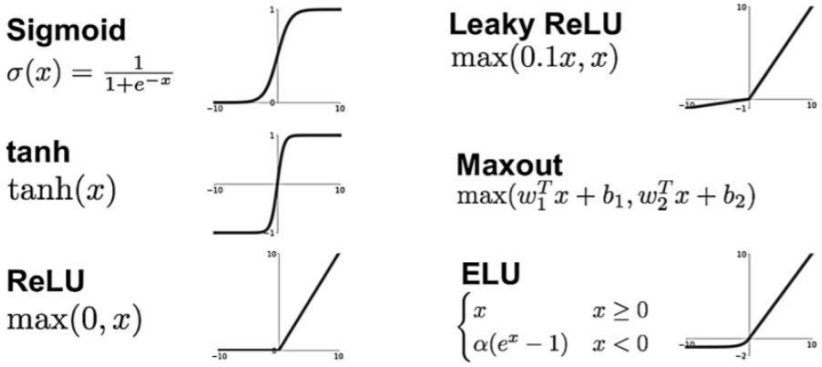


Figura 3-5. Ejemplos de funciones de activación.

De las funciones de activación presentadas en la figura 3-5, la tangente hiperbólica (tanh) será la utilizada tanto en la capa convolucional de nuestro modelo como en la capa densamente conectada (FC). Mientras que la capa de salida utilizará como función de activación la lineal. Dicha configuración se analizará con más detalle en el capítulo 4 donde se presentará el modelo.

### 3.3.3 Sesgo

Otro de los parámetros que deben ser aprendidos por el modelo junto con los pesos durante el proceso de entrenamiento es el sesgo. Este se añade a la suma de productos en cada neurona. Permite desplazar la función de activación hacia la izquierda o hacia la derecha, lo cual puede ser fundamental para un aprendizaje exitoso.

Consideremos una red de una entrada y una única salida con solamente una neurona. Dicha red no posee sesgo y tiene asociado un determinado peso para su única entrada. Supongamos que la función de activación utilizada es un Sigmoide. En este escenario la salida de la red se calcularía multiplicando la entrada por el peso asociado y pasando el resultado a través de la Sigmoide. El resultado obtenido para una misma entrada cuando se varían los pesos se representa gráficamente en la figura 3-6.

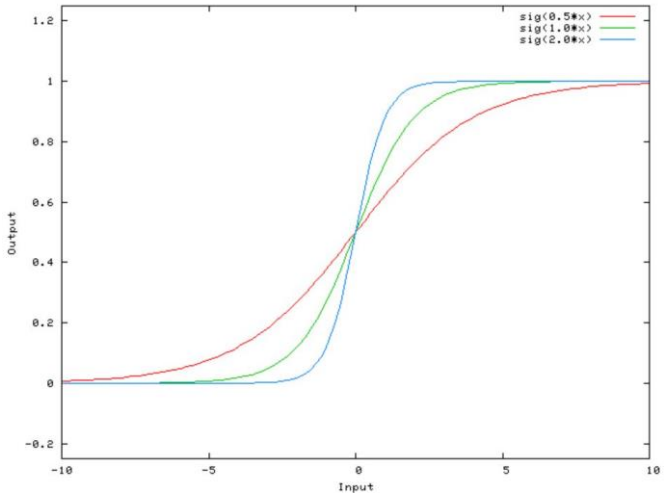


Figura 3-6. Variación del peso.

Como se puede observar en la figura 3-6, modificando el valor del sesgo lo que se consigue es variar la inclinación de la curva. Sin embargo, cuando queremos modificar la activación de la función para que se produzca a un determinado valor es necesario desplazar la curva completamente. Esto es precisamente lo que te permite hacer el sesgo y puede observarse en la figura 3-7:

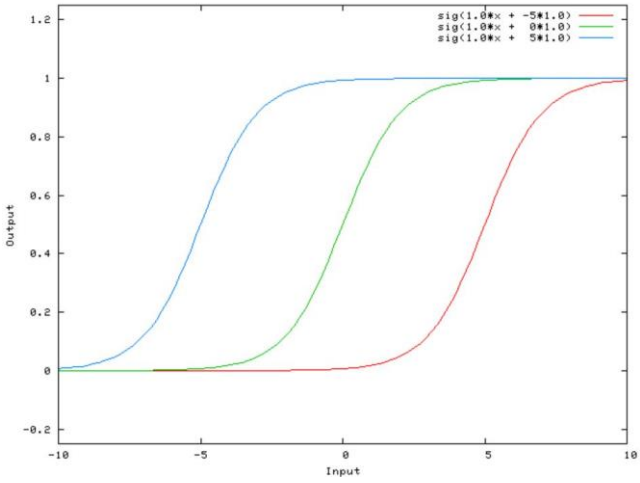


Figura 3-7. Variación del sesgo.

Finalmente, con la introducción del sesgo el esquema del perceptrón completo se puede representar mediante el diagrama de la figura 3-8.

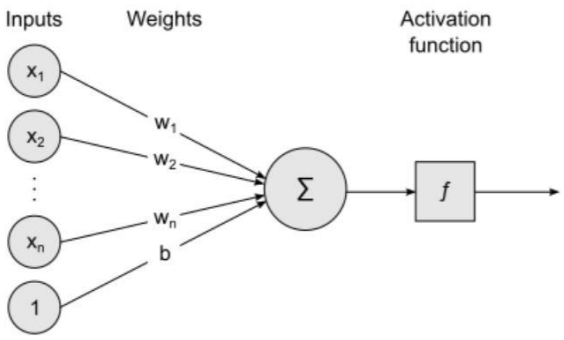


Figura 3-8. Esquema del perceptrón completo incluyendo el sesgo.

## 3.4 Redes Neuronales Artificiales para el modelado de amplificadores de potencia

Existen numerosos tipos de redes las cuales presentan distintas características, es por tanto parte fundamental del diseño analizar las funcionalidades requeridas y seleccionar la red neuronal que mejor se adapte a ellas. Aunque los principales avances ocasionados por estas se han tenido lugar en campos como el reconocimiento de voz, procesado del lenguaje o la visión por computador, su versatilidad y los buenos resultados ofrecidos en la descripción de no linealidades, han llamado la atención de los investigadores en el ámbito del modelado de comportamiento. En este apartado se realizará un recorrido por los distintos tipos de redes que se han utilizado para el modelado de comportamiento en amplificadores de potencia, resaltado los inconvenientes que se han detectado en cada una de ellas y que han llevado a seleccionar la utilizada en el modelo del presente Proyecto.

### 3.4.1 Shallow Neural Networks

Como su propio nombre indica estas redes presentan poca profundidad (shallow). En el apartado 3.2.1 se vio que el concepto de profundida hacía referencia el número de capas ocultas (hidden layers) del modelo. Las redes neuronales poco profundas se caracterizan por tener menos de tres capas ocultas y son utilizadas para expresar las características de salida del PA debido a su simple estructura y proceso de entrenamiento.

Una red neuronal poco profunda esta formada por una capa de entrada, una estructura de capas ocultas compuesta por una o dos capas, y una capa de salida. Inicialmente para el modelado de comportamiento se consideró inyectar las componentes en fase y cuadratura (I/Q) de la señal de entrada e insertar sus correspondientes valores restrasados en tiempo en la estructura de la capa de entrada de la red con el objetivo de reflejar los efectos de memoria como en el modelo RVTDNN presentado en [12]. Sin embargo, los términos dependientes de la envolvente requieren una mayor capacidad de computación de la red lo que conduce a una estructura de red con mayor complejidad y mayor número de capas ocultas. Para solventar lo anterior se recurre a la estructura mostrada en la figura 3-9, en ella se intenta inyectar además de las componentes en fase y cuadratura, los términos más importantes dependientes de la envolvente. El modelo propuesto incluye Augmented Radial Basis Function NN (ARBFNN) en [13] y Augmented Real-Value time-delay NN (ARVTDNN) en [14]. Sin embargo, con el aumento del ancho de banda de la señal, la profundidad de memoria considerada también aumentará y por ende la dimensión de entrada del modelo crecerá significativamente, dando como resultado una estructura de red compleja. En general, para proporcionar suficiente capacidad de red, las redes neuronales poco profundas hacen que el cálculo sea relativamente complejo.

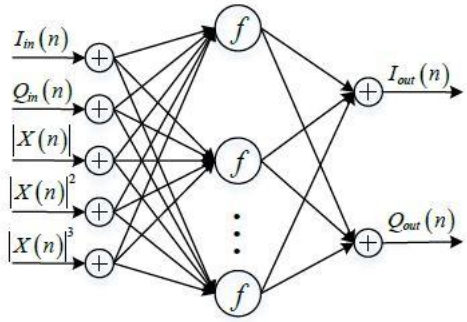


Figura 3-9. Topología de NN poco profunda con entrada de componente I/Q y términos dependientes de la envolvente.

### 3.4.2 Deep Neural Networks

Difieren de las Shallow NN en el mayor número de capas ocultas que poseen. La arquitectura de una DNN incluye más de tres capas ocultas (véase la figura 3-10) lo que le permite aproximar de manera más precisa las no linealidades y efectos de memoria del amplificador de potencia a modelar. En este tipo de redes las capacidades de ajuste y generalización incrementan de manera conjunta con el número de capas ocultas. A diferencia de las Shallow NN, las DNN pueden construir modelos más complejos con relativamente baja complejidad. Sin embargo, cuando el amplificador de potencia a modelar presenta características no lineales complejas y efectos de memoria profundos, es complicado modelarlo con DNN de baja complejidad. Además, su implementación exige recursos de procesamiento de señal excesivos a medida que el ancho de banda de la señal aumenta.

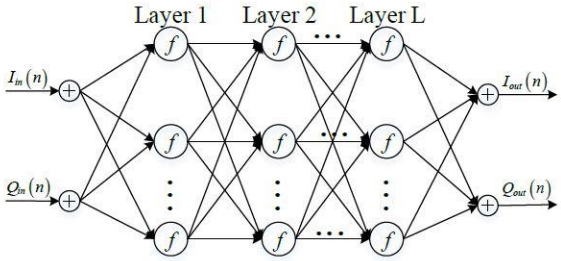


Figura 3-10. Topología de una Deep Neural Network.

### 3.4.3 Convolutional Neural Network

Para intentar solventar los problemas de complejidad existentes en las redes vistas anteriormente aparecen las redes neuronales convolucionales (CNN, del inglés Convolutional Neural Network). La estructura de distribución de pesos permite a estas redes reducir notablemente la complejidad del modelo. El modelo propuesto en este proyecto hace uso de las CNNs, dicho modelo será expuesto en detalle en el próximo capítulo, pero para su correcta comprensión en este punto se explican las características más relevantes de las CNN.

Las llamadas redes neuronales convolucionales son muy usadas en tareas de visión por ordenador y han permitido grandes avances en el campo de la inteligencia artificial. Se han popularizado mucho en los últimos años debido a los buenos resultados ofrecidos en el reconocimiento de imagen.

Son muy similares a las redes presentandas anteriormente pues están formadas por neuronas que tienen parámetros en forma de pesos y sesgos que se pueden aprender. Pero un rasgo diferencial de las CNN es que hacen la suposición explícita de que las entradas son imágenes, cosa que nos permite codificar ciertas propiedades en la arquitectura.

Su principio de funcionamiento es que cada capa va aprendiendo diferentes niveles de abstracción, es decir, cada una de ellas se encarga de analizar una característica concreta de la imagen de entrada de forma que el conjunto de las capas puede conseguir identificar estructuras más complejas en los datos de entrada.

A continuación, se muestran las dos operaciones más representativas realizadas por las neuronas que conforman las capas de una red neuronal convolucional que son: convolución y pooling.

#### 3.4.3.1 Convolución

La diferencia fundamental entre una capa completamente conectada o densamente conectada (FC) y una capa convolucional, es que la capa FC aprende patrones globales en su espacio global de entrada, mientras que la capa convolucional aprende patrones locales en pequeñas ventanas de dos dimensiones.

Se podría decir que el propósito principal de una capa convolucional es detectar características o rasgos visuales en las imágenes como aristas, líneas, etc. Esta propiedad es muy importante ya que una vez aprendida una característica en un punto concreto de la imagen puede ser reconocida posteriormente en cualquier otra parte de la misma, a diferencia de las redes neuronales densamente conectadas donde deben aprender de nuevo el patron si este aparece en una nueva localización dentro de la imagen.

Otra característica importante es que las capas convolucionales pueden aprender jerarquías espaciales de patrones preservando relaciones espaciales. Por ejemplo, una primera capa convolucional puede aprender elementos básicos como aristas, y una segunda capa convolucional puede aprender patrones compuestos por los elementos básicos de la capa anterior. Esto se puede extender a multiples capas permitiendo a las redes convolucionales aprender eficientemente conceptos visuales cada vez más complejos y abstractos.

Las capas convolucionales operan sobre los llamados feature maps los cuales están compuesto por dos ejes espaciales que son la altura y anchura (height y width), y un eje de canal también llamado profundidad (depth). Para una imagen en color RGB el valor de depth es 3, correpondiente a los colores rojo, verde y azul, mientras que para una imagen en blanco y negro el valor de depth es 1.

Para comprender con mayor claridad la opración de convolución nos apoyaremos en un ejemplo. Supongamos que la imagen analizada tiene dimesiones 28x28 y no tiene color, es en blanco y negro. En este caso nuestra capa de entrada tendrá un espacio de neuronas de dos dimensiones: height=28, width=28, depth=1. La primera capa de neuronas ocultas conectadas a las neuronas de la capa de entrada serán las encargadas de realizar las operaciones de convolución. A diferencia de las redes neuronales densamente conectadas no todas las neuronas de la capa de entrada se conectan con todas las neuronas de la primera capa oculta, sino que solo se hace por pequeñas zonas localizadas del espacio de las neuronas de entrada que almacenan los pixels de la imagen. Lo explicado se podría representar de manera visual como se muestra en la figura 3-11.

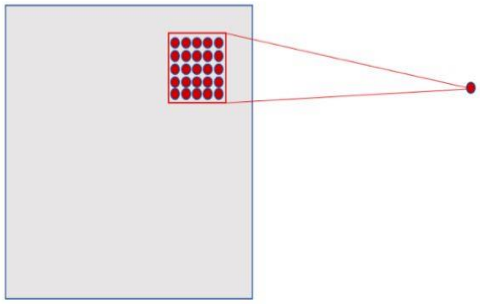

Figura 3-11. Representación de la interacción de una neurona de la capa oculta con las neuronas de la capa de entrada.

Continuando con este ejemplo, cada neurona de nuestra capa oculta será conectada a una pequeña región de 5x5 neuronas de la capa de entrada (28x28). Se puede hacer un símil con una ventana de tamaño 5x5 que va desplazándose por toda la capa de 28x28 de entrada y que por cada posición que toma la ventana hay una neurona en la capa oculta que procesa dicha información.

En nuestro caso de ejemplo, observamos que, si tenenos una entrada de 28x28 píxeles y una ventana de 5x5 esto nos define un espacio de 24x24 neuronas en la primera capa oculta, debido a que únicamente nos podemos desplazar 23 neuronas hacia la derecha y hacia abajo antes de topar con los límites de la imagen de entrada. La figura 2-12 representa esta explicación:

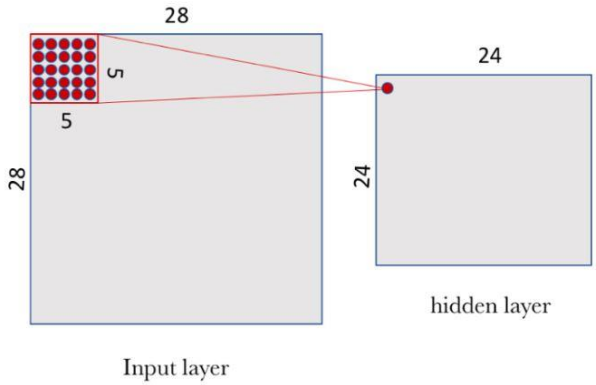


Figura 3-12. Ejemplo de convolución.

Los resultados obtenidos son fruto de la elección de un valor de stride igual a 1. Dicho parámetro define la longitud de los pasos de avance de la ventana sobre la entrada, en nuestro caso se hacen movientos de un píxel de distancia tanto en horizontal como en vertical. Para mejorar este barrido, en las CNN, existe una técnica de relleno de ceros alrededor del margen de la imagen cuyo valor viene determinado por el parámetro padding.

Para conectar cada neurona de la capa oculta con las 25 neuronas correspondientes de la capa de entrada usaremos un valor de sesgo $b$ y una matriz de pesos $W$ de tamaño 5x5 que llamaremos filtro o kernel. El valor de cada punto de la capa oculta se corresponde con el valor del producto escalar entre el filtro y las 25 neuronas de la capa de entrada.

Una de las características más importantes de las CNN es que usan el mismo filtro (la misma matriz $W$ de pesos y el mismo sesgo $b$) para todas las neuronas de la capa oculta. Esta característica permite reducir drásticamente el número de parámetros de la red. Para el caso analizado el número de parámetros que tendrían que ser ajustados

pasaría de 14.400 (5x5x24x24) a 25 (5x5) más los sesgos gracias a esta propiedad.

Cada filtro, definido como el conjunto de la matriz de pesos *W* más el sesgo *b*, únicamente permite detectar una característica concreta de la imagen. Por tanto, para poder realizar el reconocimiento completo de una imagen se utilizan varios filtros de manera simultánea, uno para cada una de las características que se quieran detectar.

Una manera visual de representar una capa de convolución se muestra en la figura 3-13. Dicha capa está compuesta por 32 filtros, donde cada uno de ellos esta definido por su matriz *W* de pesos compartida 5x5 y un sesgo *b*.

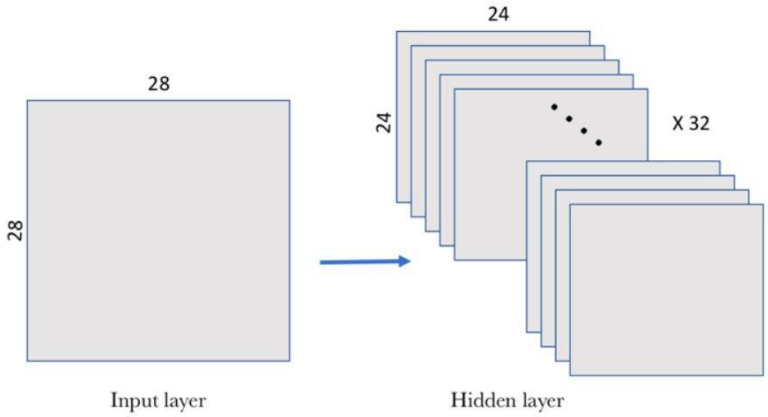


Figura 3-13. Capa de convolución compuesta por 32 filtros.

### 3.4.3.2 Pooling

Otra de las operaciones más características de las CNN es la de pooling. Suele ser aplicada justo después de la capa de convolución. La capa de pooling se encarga de hacer una simplificación de la información recogida por la capa convolucional, creando una versión condensada de la información contenida en esta.

Basándonos en el ejemplo del apartado anterior, seleccionamos una ventana de 2x2 de la capa convolucional y sintetizamos en un punto de la capa de pooling. Visualmente, la operación se puede representar mediante la figura 3-14.

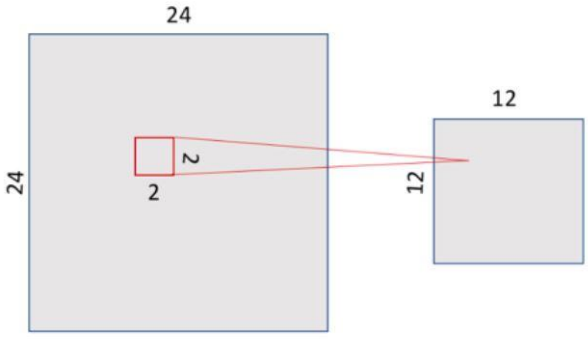


Figura 3-14. Operación de Pooling sobre capa convolucional.

Existen diversas maneras de condensar la información como el max-pooling o el average-pooling. En el primer caso nos quedamos con el valor más alto de los existentes en la ventana seleccionada. En el segundo, seleccionamos el valor promedio de los puntos que conforman la ventana. En general el max-pooling tiene a funcionar mejor. Es preciso destacar que con la transformación de pooling se mantiene la relación especial.

Como se ha mencionado anteriormente, la capa de convolución está compuesta por varios filtros, por tanto, como la aplicación del pooling es individual para cada uno de los filtros, la capa de pooling contendrá tantos filtros de pooling como filtros convolucionales (véase la figura 3-15).

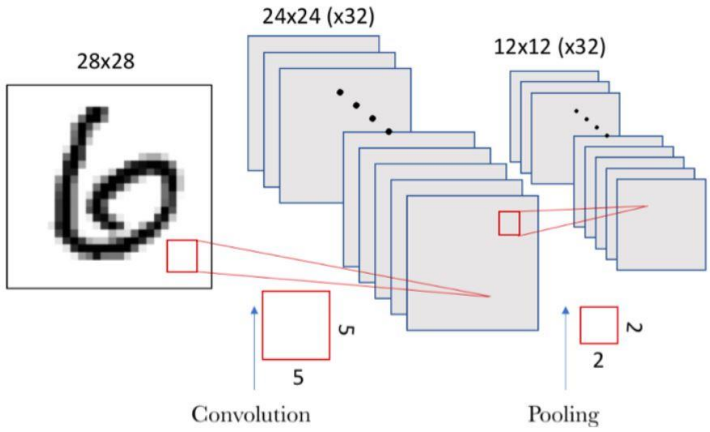


Figura 3-15. Operación de Pooling aplicada a un conjunto de filtros convolucionales.

## 3.5 Entrenamiento de una red neuronal

Entrenar una red neuronal consiste en aprender los valores de los parámetros que la conforman, pesos y sesgos. Este proceso de entrenamiento se puede ver como un proceso iterativo de "ir y venir" por las distintas capas de neuronas: *forwardpropagation* y *backpropagation* [15].

La primera fase, *forwardpropagation*, consiste en obtener las predicciones (*labels*) a partir de los datos de entrenamiento, una vez que estos atraviesan la topología de la red. Es decir, pasar los datos de entrada a través de la red de forma que todas las neuronas apliquen su transformación a la información recibida de las neuronas de la capa anterior y la envíen a las neuronas de la capa siguiente. Cuando los datos hayan cruzado todas las capas, y todas sus neuronas han realizado sus cálculos, se llegará a la capa final con un resultado de predicción para los valores de entrada empleados.

Para medir la bondad de la predicción se hace uso de la función de pérdidas (*loss*). Dicha función proporciona el error entre el resultado de la predicción y el resultado correcto, ya que como se indicó en apartados anteriores nos encontramos en un entorno de aprendizaje supervisado en el que disponemos de los valores esperados. El objetivo es que el error se haga cero, es decir, no existan divergencias entre el valor estimado y el esperado. Por eso a medida que se entrena el modelo se irán ajustando los pesos de las interconexiones entre las distintas neuronas de manera automática hasta obtener buenas predicciones.

Una vez se tiene calculado el error, se propaga hacia atrás dicha información. Comienza la segunda fase, en inglés *backpropagation*. Esta consiste en propagar la información de pérdidas desde la capa de salida hacia todas

las neuronas de la capa oculta que contribuyen a la obtención de la salida. Cada neurona de la capa oculta únicamente recibe una fracción de la señal total de pérdidas en función de la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de pérdidas que describa su contribución relativa a las pérdidas totales. Un resumen visual del proceso descrito se puede ver en la figura 3-16:

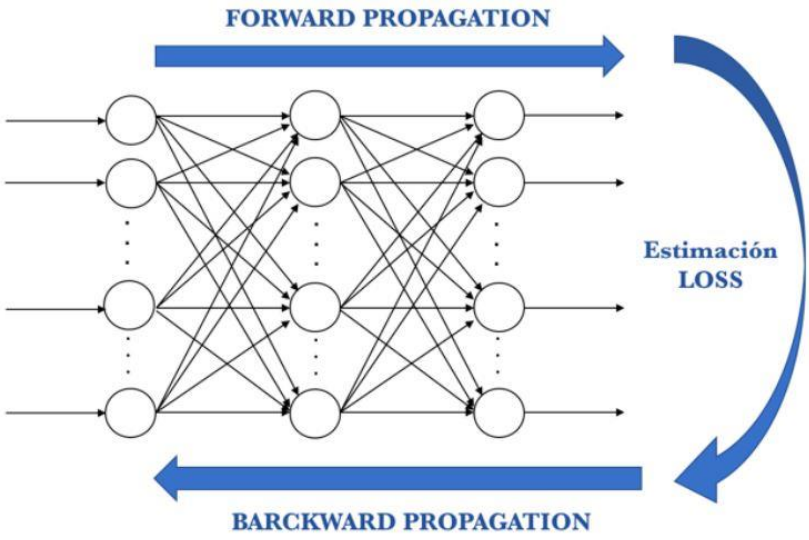


Figura 3-16. Proceso de aprendizaje de una red neuronal.

Una vez propagada hacia atrás la información de pérdidas, se pueden ajustar los pesos de las conexiones entre neuronas. Lo que se busca es obtener un error lo más cercano a cero la próxima vez que se utilice la red para una predicción. Para dicho cometido existe una amplia variedad de algoritmos de optimización: SGD, RMSprop, Adam, Adagrad, etc. La función de estos optimizadores es introducir pequeños cambios en el valor de los parámetros (pesos y sesgos) con el fin de que la función de pérdidas se minimice. Esto se va haciendo en lotes de datos (*batches*) en las sucesivas iteraciones (*epochs*) del conjunto de todos los datos que le pasamos a la red.

Para facilitar la comprensión de este proceso iterativo, nos apoyaremos en un ejemplo cotidiano en el que se trata una ANN con función de activación lineal. Supongamos que todos los días asistimos a una cafetería, en la cual tomamos un café, un zumo y una tostada, pero sólo al final de la semana nos pasan el total de la cuenta, sin indicar el precio individual de cada producto. Tras varias semanas, se estaría en disposición de estimar los precios individuales.

El algoritmo iterativo consistiría en lanzar una estimación inical aleatoria de los precios individuales, ajustándolos en cada iteración para que se ajusten a los importes facturados. Cada semana, el total nos impone una restricción lineal sobre los precios de las cantidades consumidas:

$$total = W_{café}X_{café} + W_{zumo}X_{zumo} + W_{tostada}X_{tostada} \quad (3–1)$$

Donde *W* son los distintos precios de los productos mientras que *X* representa la cantidad consumida de estos. El objetivo consiste en estimar el valor de cada *W* basándonos en los valores conocidos de *X* y *total*. Se tendrá una función objetivo (*t*) que representa el valor total de la cuenta. En base a ese valor y conocido el número de productos consumidos se realizará una estimación del costo de cada uno de los productos individuales, *W*. Esto proporcionará un total estimado (*y*), la diferencia entre *t* e *y* proporciona el error residual. El objetivo es reducir lo máximo posible este error residual mediante una tasa de aprendizaje que representamos por la siguiente

ecuación:

$$\Delta w_i = \alpha x_i (t - y) \tag{3–2}$$

Donde $\alpha$ es la tasa de aprendizaje utilizada. En definitiva, el algoritmo *backpropagation* trata de comprender como el cambio de los pesos y sesgos de una red impacta en la función de coste, lo que significa calcular las derivadas parciales de estos sobre la función de coste:

$$\frac{\partial total}{\partial w_j}, \qquad \frac{\partial total}{\partial b_j} \tag{3–3}$$

### 3.5.1 Optimizador Adam

Es el optimizador empleado en el entrenamiento del modelo que se propone utilizar en el presente Trabajo. Adam es un algoritmo para la optimización basada en el gradiente de primer orden de funciones objetivo estocásticas, basándose en estimaciones adaptativas de momentos de orden inferior [16]. El método es computacionalmente eficiente y muy adecuado para problemas que son grandes en términos de datos y/o parámetros. Calcula individualmente las tasas de aprendizaje para diferentes parámetros a partir de las estimaciones de primer y segundo momento del gradiente. Algunas de las ventajas de Adam son que las magnitudes de las actualizaciones de los parámetros son invariantes para reescalar el gradiente, sus tamaños de paso están limitados por el hiperparámetro *stepsize*, no requiere un objetivo estacionario y funciona con gradientes dispersos.

La figura 3-17 representa el pseudocódigo del algoritmo. Una buena configuración de los valores de los parámetros en entornos de Machine Learning es $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ y $\epsilon = 10^{-8}$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
**while** $\theta_t$ not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
**end while**
**return** $\theta_t$ (Resulting parameters)

Figura 3-17. Pseudocódigo del algoritmo Adam [16].

El objetivo es minimizar el valor esperado de la función objetivo, $f(\theta)$, donde $\theta$ son los parámetros de la red.

Con $g_t = \nabla_\theta f_t(\theta)$ denotamos el gradiente, que representa el vector de derivadas parciales de $f_t$ sobre cada uno de los parámetros $\theta$ evaluado para cada tiempo *t*.

El algoritmo actualiza los movimientos promedios exponenciales del gradiente ($m_t$) y el gradiente cuadrado ($v_t$) donde los hiperparámetros $\beta_1, \beta_2 \in [0,1)$ controlan las tasas de disminución exponencial de los movimientos promedios. Los movimientos promedios en sí son estimaciones del primer momento (media) y segundo momento (varianza no centralizada) del gradiente. Sin embargo, estos movimientos promedios son inicializados como vectores nulos, lo que da lugar a estimaciones de momentos sesgadas hacia cero, especialmente en los *timesteps* iniciales, y en especial cuando las tasas de disminución son pequeñas ($\beta's$ próximas a uno). La buena noticia es que el sesgo de inicialización puede ser contrarestado fácilmente, dando como resultado las estimaciones corregidas en sesgo $\widehat{m_t}$ y $\widehat{v_t}$.

Una propiedad importante de la regla de actualización de Adam's es su cuidadosa elección de los stepsizes [17]. Asumiendo $\epsilon = 0$, el paso efectivo dado en el espacio de parámetros en *t* es $\Delta_t = \alpha \cdot \widehat{m}_t / \sqrt{\widehat{v}_t}$. El *stepsize* efectivo tiene dos límites: $|\Delta_t| \leq \alpha \cdot (1-\beta_1)/\sqrt{1-\beta_2}$ en el caso de que $(1-\beta_1) > \sqrt{1-\beta_2}$, y $|\Delta_t| \leq \alpha$ en caso contrario. El primer caso únicamente ocurre en el caso más severo de disparidad: cuando el gradiente ha sido cero en todos los *timesteps* excepto en el actual. Para casos menos dispersos, el *timestep* efectivo será menor. Cuando $(1-\beta_1) = \sqrt{1-\beta_2}$ tenemos que $\left|\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}}\right| < 1$ por lo tanto $|\Delta_t| < \alpha$. En la mayoría de los casos, se tiene que $\widehat{m}_t / \sqrt{\widehat{v}_t} \approx \pm 1$. La magnitud efectiva de los pasos tomada en el espacio de parámetros de cada timestep está limitada aproximadamente por el stepsize seleccionado $\alpha$. Esto puede entenderse como el establecimiento de una region de confianza alrededor del valor del parámetro actual, más allá del cual la estimación del gradiente actual no proporciona información suficiente. Esto permite de antemano conocer la escala correcta de $\alpha$. Para muchos modelos de Machine Learning, a menudo conocemos que los valores óptimos tienen una alta probabilidad de encontrarse en una región establecida del espacio de parámetros, no siendo infrecuente disponer de una distribución previa sobre los parámetros. Dado que $\alpha$ establece un límite superior de la magnitud de los pasos en el espacio de parámetros, a menudo podemos deducir el orden correcto de la magnitud de $\alpha$ de tal manera que se pueda alcanzar un valor óptimo de $\theta_0$ dentro de un cierto número de iteraciones. Por comodidad, denominaremos a la relación $\widehat{m}_t / \sqrt{\widehat{v}_t}$ relación señal a ruido (SNR, del inglés *signal-to-noise ratio*). Un valor pequeño de SNR provocará que el *stepsize*, $\Delta_t$, se encuentre cercano a cero. Esta es una propiedad deseable, ya que una SNR más pequeña significa que existe una mayor incertidumbre sobre si la dirección de $\widehat{m}_t$ corresponde a la dirección del gradiente verdadero. Por ejemplo, el valor de la SNR tiene a cero cuando nos aproximamos a un óptimo, lo que lleva a pasos más pequeños en el espacio de parámetros.

### 3.5.2 Hiperparámetros

Por hiperparámetro nos referimos a variables de configuración que son externas al modelo en sí mismo y cuyo valor en general no puede ser estimado a partir de los datos (a diferencia de los parámetros que si son estimados a partir de estos), por lo que son especificados por el programador para ajustar los algoritmos de aprendizaje. Estos deben ser configurados antes de iniciar el proceso de entrenamiento para que los modelos entrenen mejor y más rápidamente.

En secciones anteriores se han presentado algunos hiperparámetros a nivel de estructura y topología de la red neuronal como el número de capas, el número de neuronas, funciones de activación, etc. Pero en este apartado nos centraremos en los hiperparámetros a nivel de algoritmo de aprendizaje, algunos de los más destacados son:

- **Epochs:** Indica el número de veces en las que todos los datos de entrenamiento han pasado por la red neuronal en el proceso de entrenamiento. Un indicativo en el que podemos basarnos para ajustar este

valor es incrementarlo hasta que la métrica utilizada para dimensionar la bondad de nuestra predicción comience a decrecer.

- **Batch size:** Indica el tamaño de los mini lotes en el que se particionan los datos de entrenamiento para pasarlos por la red. Cada vez que se alcance este tamaño se producirá una actualización del optimizador empleado. El tamaño óptimo dependerá de muchos factores, entre ellos la capacidad de memoria del PC que usemos para hacer los cálculos.
- **Learning rate:** También conocido como *step size* representa el tamaño de paso que el algoritmo realiza en el entrenamiento de una red, es el incremento entre dos iteraciones. Como se ilustra en la figura 3-18, el valor adecuado de este hiperparámetro es muy dependiente del problema en cuestión: si este es demasiado grande, se darán pasos grandes, lo que puede acelerar el proceso de aprendizaje, pero con la posibilidad de que se salte algún mínimo; en cambio, un learning rate pequeño provocará avances constantes y pequeños, teniéndose una mejor oportunidad de llegar a un mínimo local, pero provocando que el proceso de aprendizaje sea muy lento.

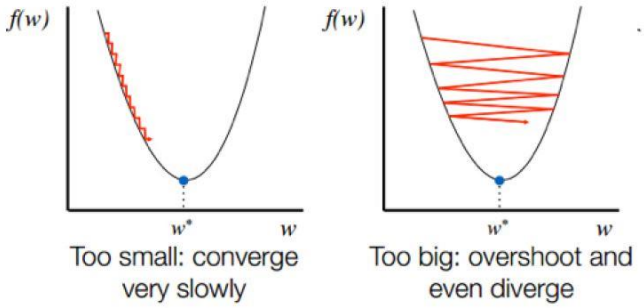


Figura 3-18. Búsqueda de la solución con Learning rate pequeño y grande.

### 3.5.3 Resultado del proceso de aprendizaje

El proceso de entrenamiento puede derivar en distintos escenarios:

- **No convergencia:** el algortimo es incapaz de encontrar una solución y continúa iterando sin fin.
- **Convergencia:** se encuentra una solución al problema; llegados a este punto se pueden dar tres posibilidades:
  - **Subajuste (underfitting):** se produce cuando el conjunto de datos de entrenamiento no es lo suficientemente representativo de la tarea a aprender. Provoca que el algoritmo no sea capaz de generalizar para ningún caso, obteniéndose un error de entrenamiento grande.
  - **Sobreajuste (overfitting):** se produce cuando se intenta minimizar demasiado el error de entrenamiento. En este caso las predicciones sobre los datos de entrenamiento son muy buenas, pero la capacidad de generalización de la red es bastante pobre, haciendo que ante nuevos datos con los que no ha sido entrenada sea incapaz de realizar una clasificación acertada.
  - **Buen entrenamiento:** la red genera buenas predicciones sobre los datos de entrenamiento, pero a su vez desarrolla la capacidad de generalización haciendo que esta ofrezca buenas predicciones sobre nuevos datos de entrada.

La figura 3-19 representa de manera gráfica los conceptos explicados anteriormente.

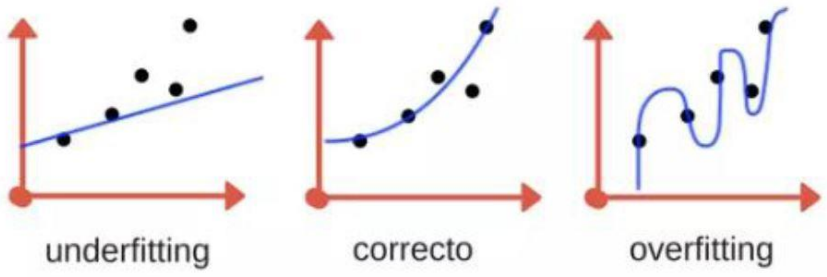


Figura 3-19. Posibles resultados de convergencia.

Existen diversas técnicas para evitar las patologías típicas de un mal entrenamiento. Sin embargo, con una correcta elección de los parámetros y una base de datos de entrada equilibrada, los algoritmos actuales son capaces de alcanzar buenas soluciones.

# 4 MODELO PROPUESTO

Una vez expuestos los parámetros e hiperparámetros que conforman las redes neuronales, así como los distintos tipos existentes, se presenta en este capítulo el modelo seleccionado para el modelado y posterior linealización del PA. Dicho modelo hace uso de las CNN, apoyándose en su característica de peso compartido para reducir su complejidad.

## 4.1 RVTDCNN (Real-Value Time-Delay Convolutional Neural Network)

El modelo RVTDCNN propuesto en [1], presenta una estructura compuesta por cuatro capas, una primera llamada capa de entrada (input layer), una capa de convolución, una completamente conectada (FC) y una capa de salida. La representación gráfica de dicha estructura se muestra en la figura 4-1:

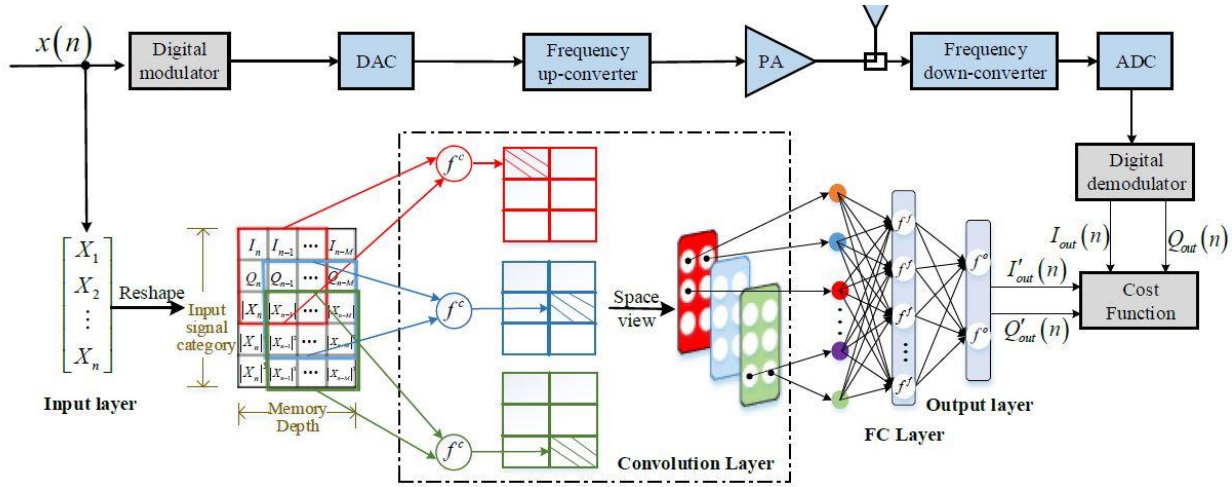


Figura 4-1. Diagrama de bloques del modelo RVTDCNN [1].

La capa de convolución es la encargada de capturar de manera efectiva las características más importantes de los datos de entrada. Estas pueden ser extraídas de forma eficiente, con baja complejidad, gracias a la característica de peso compartido y a la reducción de la dimensionalidad de los datos del kernel de convolución propias de la estructura convolucional. Las dimensiones de cada kernel de convolución son diseñadas para producir una baja complejidad computacional a la vez que se mantiene un buen comportamiento en la predicción del modelo.

La salida de la capa convolucional se conecta a la capa FC, esta es la encargada de integrar las características válidas obtenidas en la primera. Finalmente, se encuentra la capa de salida formada por dos neuronas con una función de activación lineal, encargadas de predecir las componentes I/Q de las muestras.

Como se vio en el capítulo 3, las CNN son usadas para el reconocimiento de imagen por ello que dichas redes reciben como entrada una imagen. Una imagen nos es más que un matriz donde cada una de sus entradas representa un píxel. Para poder hacer uso de la potencia de las CNN debemos adaptar nuestros datos de entrada

al formato deseado, en este caso matriz. De esta transformación se encarga la capa de entrada. Los datos de entrada se conforman en una matriz de dos dimensiones, compuesta por las componentes I/Q y los términos dependientes de la envolvente de las señales actuales y pasadas. La matriz de entrada viene dada por la siguiente expresión:

$$X_n = \begin{pmatrix} I_{in}(n) & I_{in}(n-1) & \dots & I_{in}(n-M) \\ Q_{in}(n) & Q_{in}(n-1) & \dots & Q_{in}(n-M) \\ |x(n)| & |x(n-1)| & \dots & |x(n-M)| \\ |x(n)|^2 & |x(n-1)|^2 & \dots & |x(n-M)|^2 \\ |x(n)|^3 & |x(n-1)|^3 & \dots & |x(n-M)|^3 \end{pmatrix} \tag{4–1}$$

Donde $I_{in}(n)$ y $Q_{in}(n)$ representan las componentes I/Q de la señal actual. $|x(n)|$ denota la amplitud de la señal actual. Mientras que $I_{in}(n-i), Q_{in}(n-i)\ y\ |x(n-i)|, (i = 1,2,\dots,M)$ representan los correspondientes términos de muestras pasadas. Finalmente, *M* representa la profundidad de memoria.

El formato matriz que proporciona la capa de entrada para adaptar los datos al proceso de convolución hace que los elementos de los datos de entrada correspondientes al restraso de señales adyacentes se dispongan de forma adyacente, garantizando que el kernel de convolución bidimensional empleado extraiga los términos cruzados de las diferentes señales retrasadas. La matriz de entrada $X_n$, se convoluciona con los multiples kernels de convolución (filter en la figura 4-2) y se le agrega el parámetro del sesgo para generar los distintos volúmenes de feature maps. Un ejemplo gráfico de este proceso se recoge en la figura 4-2:

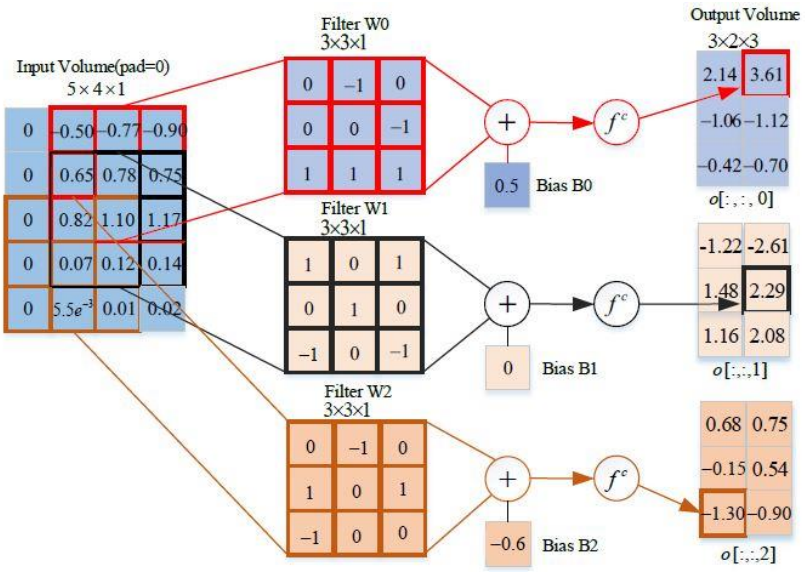


Figura 4-2. Diagrama de convolución bidimensional [1].

La operación de convolución viene dada por la ecuación (4-2):

$$h_l = \begin{pmatrix} I_{in}(n) & I_{in}(n-1) & \dots & I_{in}(n-M) \\ Q_{in}(n) & Q_{in}(n-1) & \dots & Q_{in}(n-M) \\ |x(n)| & |x(n-1)| & \dots & |x(n-M)| \\ |x(n)|^2 & |x(n-1)|^2 & \dots & |x(n-M)|^2 \\ |x(n)|^3 & |x(n-1)|^3 & \dots & |x(n-M)|^3 \end{pmatrix} \blacksquare w_l^c \tag{4–2}$$

donde $h_l$, $l = (1,2,\dots,L)$ representa la salida de la operación de convolución entre la matriz de entrada, $X_n$, y el *l*-ésimo kernel de convolución. *L* representa el número de kernel de convolución (matriz de pesos *W* en la notación empleada en el capítulo 3) y $w_l^c$ representa los coeficientes del *l*-ésimo kernel.

La salida del proceso de convolución se pasa a través de una función de activación no lineal para la obtención de la capacidad de ajuste no lineal. La salida de esta función de activación, perteneciente a los kernels de convolución, viene dada por la ecuación (4-3):

$$u_l = f^c(h_l + b_l^c) \tag{4–3}$$

donde $u_l$, $(l = 1,2,\dots,L)$ es el mapa de características del *l*-ésimo kernel de convolución, $f^c(.)$ es la función de activación y $b_l^c$ representa el sesgo del *l*-ésimo kernel de convolución.

La salida de la capa convolucional, expresada en mapas de características como se muestra en la figura 4-2, se transforma en un vector de características para ser inyectada en la capa FC. Este vector de características viene dado por la siguiente expresión:

$$m = [m_1, m_2, \dots, m_{L\times B\times C}] = [u_1(1,1), u_1(1,2), \dots, u_2(1,1), \dots, u_L(B,C)] \tag{4–4}$$

Las dimensiones del vector son $L\ x\ B\ x\ C$, mientras que la de los mapas de características $B\ x\ C$.

Tras el procesado del vector de entrada, la capa FC proporciona la siguiente salida:

$$a_t = f^f\left(\sum_{i=1}^{L\times B\times C} w_{ti}^f m_i + b_t^f\right) \tag{4–5}$$

donde $a_t$, $(t = 1,2,\dots,T)$ denota la salida de la t-ésima neurona. *T* representa el número de neuronas de la capa FC. $w_{ti}^f$ y $b_t^f$ representan los pesos y el sesgo, respectivamente. $f^f(.)$ es la función de activación de la capa FC.

Finalmente, la capa de salida pondera, mediante los pesos, y suma las características de salida de la capa FC para obtener la salida del modelo. Para garantizar valores continuos en los datos de salida, se configura la función de activación de la capa de salida, $f^o$, como una función lineal $y = x$.

$$\begin{cases} I'_{out}(n) = \sum_{t=1}^{T} w_{1t}^{o} a_t + b_1^o \\ Q'_{out}(n) = \sum_{t=1}^{T} w_{2t}^{o} a_t + b_2^o \end{cases} \tag{4–6}$$

donde $\acute{I_{out}}(n)$ y $\acute{Q_{out}}(n)$ representa las salidas de las dos neuronas de la capa de salida, que a su vez se corresponden con los valores I/Q estimados. $\{w_{1t}^o, w_{2t}^o, b_1^o, b_2^o\}$ representan los pesos y sesgos de la capa de salida.

### 4.1.1 Parámetros e hiperparámetros del modelo

Una vez presentada la estructura del modelo analizado procedemos a la configuración de los parámetros e hiperparámetros del mismo. Como se indico en el capítulo anterior, los parámetros son estimados a partir de los datos de entrada durante la fase de entrenamiento. El objetivo de dicha fase es minimizar el error entre las *labels* (solución deseada) y la salida del modelo (valor estimado), la cual es determinada en el forward path mediante la actualización de los parámetros de la red, $\theta_k = \{w_k^c, b_k^c, w_k^f, b_k^f, w_k^o, b_k^o\}$, en cada *epoch k* hasta que la red converge. En el *forward path*, se define como función de coste el error cuadrático medio (MSE, del inglés *Mean Square Error*) que viene dado por la siguiente expresión:

$$E_{mse}(\theta) = \frac{1}{2N} \sum_{n=1}^{N} [(I'_{out}(n) - I_{out}(n))^2 + (Q'_{out}(n) - Q_{out}(n))^2] \tag{4–7}$$

donde $I'_{out}(n)$ y $\acute{Q_{out}}(n)$ representan la salida estimada por el modelo RVTDCNN mientras que $I_{out}(n)$ y $Q_{out}(n)$ representa los valores esperados. *N* es el número de muestras de los datos de entrenamiento.

El optimizador utilizado para optimizar los parámetros de la red RVTDCNN, $\theta$, basandose en la función de coste es el de Adam. Finalmente, para cuantificar la presición del modelado se utiliza como valor el error cuadrátrico medio normalizado (NMSE), descrito por la siguiente ecuación:

$$NMSE = 10 \times log \frac{\frac{1}{N}\sum_{n=1}^{N}((I'_{out}(n) - I_{out}(n))^2 + (Q'_{out}(n) - Q_{out}(n))^2)}{\frac{1}{N}\sum_{n=1}^{N}((I_{out}(n))^2 + (Q_{out}(n))^2)} \tag{4–8}$$

Antes de comenzar con el proceso de entrenamiento, los parámetros del modelo RVTDCNN son aleatoriamente inicializados siguiendo una distribución normal estándar. En cada epoch, primero se calcula el valor de la función de coste y posteriormente se actualizan los parámetros mediante el algoritmo de optimización Adam. En la siguiente *epoch* se calcula el valor de la función de coste con los nuevos parámetros obtenidos en la *epoch* anterior. El proceso de entrenamiento finaliza cuando la función de coste alcanza el umbral requerido.

Tras analizar cómo se configuran los parámetros del modelo, procedemos al análisis y configuración de los hiperparámetros. Los hiperparámetros a diferencia de los parámetros, no pueden ser determinados a partir de los

datos de entrada. De modo que deben definirse adecuadamente, de forma previa a la fase de entrenamiento, para lograr unos buenos resultados en el modelado. La elección de los datos de entrada afecta a la precisión del modelado, así como a la complejidad del modelo. Una dimensión inapropiada de los datos de entrada deriva en un aumento del número de coeficientes del modelo. Por ello, y basándonos en [14], la combinación de los elementos de la señal de entrada $I$, $Q$, $|x(n)|$, $|x(n)|^2$, $|x(n)|^3$ es la mejor opción, produciendo una baja complejidad del modelo y un buen rendimiento. Este es el motivo por el que la matriz de entrada tiene un número de filas fijo, presentado dimensiones de 5x$M$, donde $M$ es la profundida de memoria.

Una vez determinados los datos de entrada, el tamaño y el número de kernels de convolución se convierte en un factor que afecta al rendimiento del modelado. Basándonos en el estudio realizado en [1], cuando la capa de convolución está compuesta por 3 kernels de tamaño 3·3·1 (expresado en el formato altura·anchura·profundidad), el número de coeficientes del modelo es relativamente pequeño, y los resultados de NMSE bastante buenos.

El número de neuronas de la capa FC es otro factor importante que afecta a la precisión del modelado y complejidad del modelo. Fijadas las dimensiones de los datos de entrada, así como el tamaño de la capa convolucional, y apoyándonos en el estudio realizado en [1], el número óptimo de neuronas en la capa FC es 6.

Finalmente, falta por definir la función de activación utilizada en la capa convolucional. El estudio realizado en [1] determina que la función de activación que mejores resultados, medidos en NMSE, ofrece es la tangente hiperbólica. Dicha función viene definida por la siguiente ecuación:

$$\tanh(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \tag{4–9}$$

Esta función de activación, junto con algunas de las más utilizadas, fueron presentadas de forma gráfica en el capítulo 3.

### 4.1.2 Modelo para DPD

Para la implementación del predistorsionador utilizamos el modelo inverso de RVTDCNN. Este modelo inverso posee las mismas dimensiones que el original, es decir: mismo tamaño de los datos de entrada, mismo número y tamaño de kernels empleados en la capa de convolución, mismo número de neuronas en la capa FC y mismo número de neuronas en la capa de salida.

La diferencia radica en que la entrada al modelo inverso (modelo DPD) son los datos de salida del PA, adaptados al formato matriz explicado anteriormente. Por su parte, los *label data*, es decir, los datos de salida esperados serán los de entrada al PA. Los datos de entrada del modelo inverso, salida del PA, antes de ser conformados en formato matriz, deben estar normalizados respecto a la ganancia de compresión. Para la selección de esta ganancia de compresión distinguimos dos criterios.

Como la superposición no se puede aplicar a un sistema no lineal (como el PA), generalmente el nivel de potencia de los datos para la extracción del modelo debe normalizarse al mismo nivel de potencia de la entrada original para que los parámetros extraídos se puedan usar directamente en el DPD después de la extracción del modelo. Sin embargo, este proceso de normalización depende en gran medida de la selección de la ganancia esperada [18].

En general, en un predistorsionador, la ganancia máxima, que es la respuesta lineal del PA, se elige como la ganancia linealizada esperada, como se muestra en la figura 4-3. Al aumentar previamente las amplitudes en el nivel de potencia más alto, la compresión de ganancia se puede compensar. Sin embargo, debido a que el predistorsionador solo puede corregir con éxito la distorsión hasta el nivel de saturación total del amplificador (después de lo cual cualquier aumento en la potencia de entrada no produce un aumento en la potencia de salida), la entrada máxima permitida para el predistorsionador solo puede alcanzar el punto donde la respuesta lineal se cruza con el límite de saturación. Por lo tanto, en este caso, la entrada original y la entrada predistorsionada, salida del DPD, tienen valores pico distintos como se muestra en la figura 4-3. Esto tiene como consecuencia que, en la extracción de parámetros, los datos de entrada y salida medidos desde el PA deben normalizarse a diferentes niveles de potencia por diferentes factores de escala, para corresponder a los diferentes valores de pico en las entradas originales y las predistorsionadas. Esto requiere un esfuerzo de calibración adicional y también aumenta la complejidad del control de potencia de entrada en la implementación final del sistema debido a los diferentes niveles de pico que ocurren antes y después del DPD.

Sin embargo, si seleccionamos la ganancia esperada en el nivel de potencia máxima objetivo, como se muestra en la figura 4-4, la ganancia del sistema general *k* viene dada por:

$$k = \frac{Max[|\tilde{y}(n)|]}{Max[|\tilde{x}(n)|]} \tag{4–10}$$

Donde $\tilde{x}(n)$ e $\tilde{y}(n)$ son las muestras de la envolvente compleja de la señal de entrada y salida del PA. En este caso asumimos que la potencia de salida del PA aumenta "linealmente" con la potencia de entrada. En otras palabras, la potencia máxima de salida solo se produce para la entrada máxima. En este caso, tanto la entrada original como la entrada predistorsionada alcanzarán el mismo nivel de potencia máxima. Por lo tanto, todas las formas de onda de entrada y salida pueden normalizarse a la unidad. Esto simplifica significativamente la extracción del modelo porque podemos normalizar directamente los datos de entrada y salida medidos desde el PA por sus respectivos valores pico, y luego utilizar estos datos como salida y entrada del predistorsionador para la extracción de parámetros. La selección de la ganancia en el nivel de potencia máximo también hace que la implementación final del sistema sea más fácil porque tanto la entrada original como la entrada predistorsionada pueden normalizarse por el mismo factor de escala, lo que facilita el control de potencia.

Aunque la ganancia general se ha reducido en este caso, la potencia de salida media y máxima después de la predistorsión sigue siendo la misma que en el caso de linealización con la ganancia máxima. En términos del rendimiento general de linealización, estos dos sistemas DPD son equivalentes.

En este trabajo el proceso seleccionado para la obtención del valor de la ganancia de compresión es el presentado en el segundo criterio, linealizado para la ganancia en el nivel de potencia máxima objetivo.

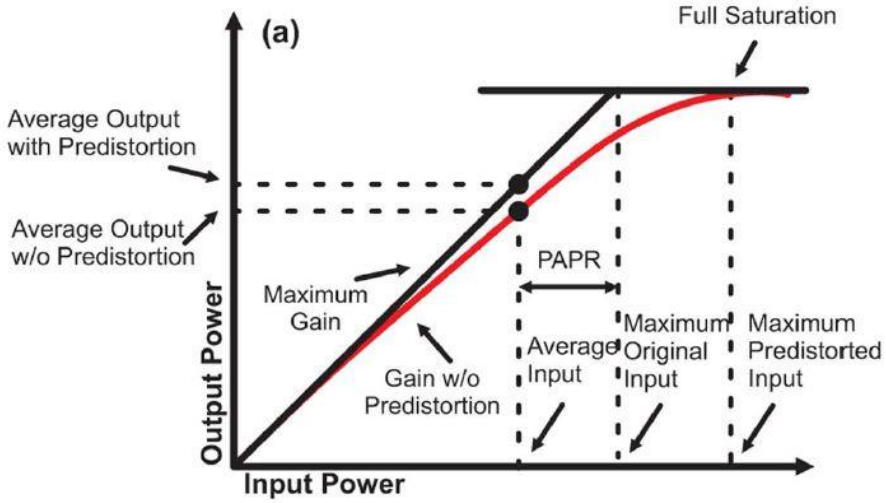


Figura 4-3. Linealizado para la ganancia máxima [18].

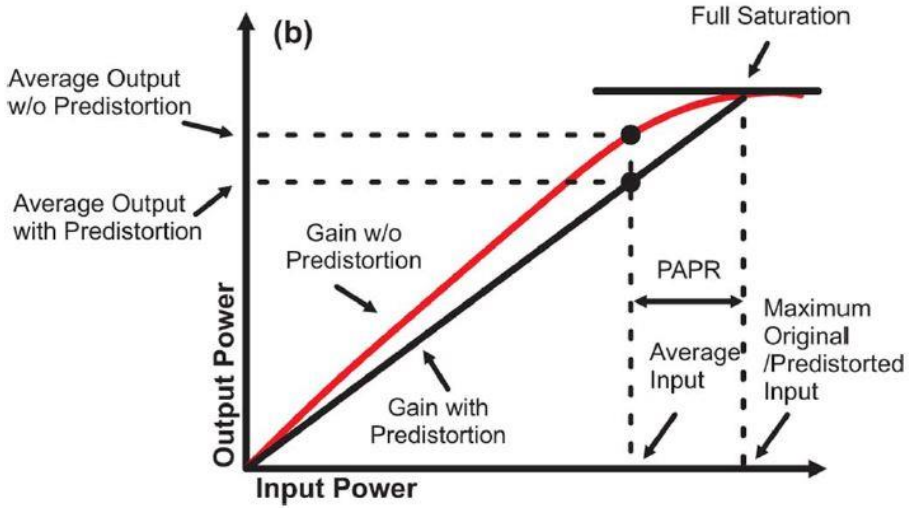


Figura 4-4. Linealizado para la ganancia en el nivel de potencia máxima objetivo [18].

# 5 MATERIALES UTILIZADOS

Una vez descrita la teoría que existe tras las redes neuronales y presentada la estructura del modelo RVTDCNN, este capítulo se centra inicialmente en exponer las herramientas a nivel software necesarias para su implementación. Mientras que la segunda parte de este estará enfocada en la explicación del código generado.

El código generado en este proyecto se divide en dos vertientes. Por una parte, el código desarrollado en Matlab es el encargado de la generación de la señal, del envío de esta al PA y de la representación de los resultados obtenidos. Por la otra parte, tenemos el código en Python encargado de la implementación del modelo RVTDCNN. Este es el motivo por el que se distingen dos entornos de trabajo.

El primer entorno de trabajo está compuesto por las siguientes herramientas:

- Plataforma WebLab.
- Software matemático Matlab.

Mientras que el segundo entorno, encargado de la implementación del modelo RVTDCNN en Python, utiliza Spyder.

## 5.1 Plataforma WebLab

La predistorsión digital (DPD) es hoy en día un método comúnmente utilizado para la linealización de transmisores de RF. Sin embargo, aún se necesitan más desarrollos. Por esta razón, en 2014 se propuso la competición de diseño de estudiantes "Linealización de un Amplificador de Potencia mediante Predistorsión Digital" en el International Microwave Symposium (IMS) del IEEE. La competición también sirve como punto de referencia entre algoritmos porque ahora es posible evaluar los diferentes algoritmos en un sistema común, por medio del acceso remoto.

Para ofrecer a los participantes la posibilidad de probar sus algoritmos de linealización por adelantado, se ha creado una página web, mantenida por la Universidad Tecnológica de Chalmers, llamada dpdcompetition.com. Esta página web es la utilizada en este proyecto para llevar a cabo las pruebas. En ella podemos acceder de forma remota al sistema de medición (WebLab) que incorpora un generador de señal, un amplificador de potencia en tecnología GaN y un analizador de señal. Esta configuración permite probar los predistorsionadores considerados. La configuración de medición está basada en un chasis PXI (PXIe-1082) con un PC host integrado de National Instruments. El chasis está equipado con un transceptor de señal vectorial (PXIe-5646R VST) que permite probar señales con un ancho de banda instantáneo de 200 MHz. La señal generada en el VST alimenta a un amplificador lineal (*driver*) antes de pasar al PA sobre el cual se realizan las medidas. A la salida del dispositivo bajo prueba (DUT) se coloca un atenuador de 30 dB y la salida de éste se conecta al receptor del VST. El PC embebido en el chasis PXI es el encargado de controlar los instrumentos y de cargar y descargar los archivos de datos a petición de los usuarios. El DUT se complementa con un módulo de fuente de alimentación (PXI-4130) que también mide el consumo de corriente del amplificador de potencia. La figura 5-1 ilustra lo explicado anteriormente:

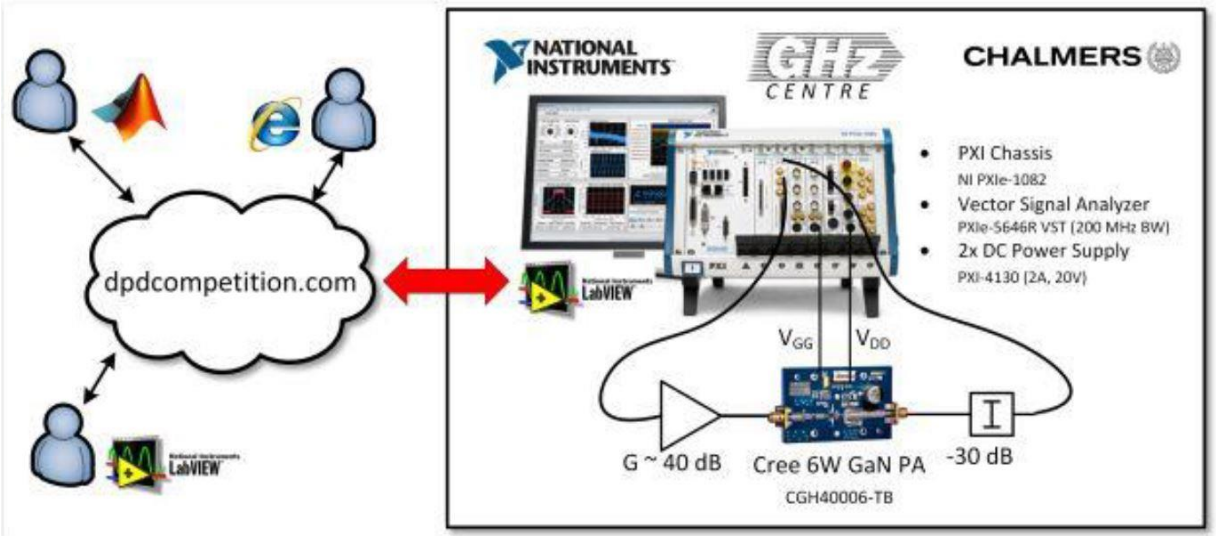


Figura 5-1. Configuración del RFWebLab [19].

El nivel máximo de potencia de salida del PA se limita aproximadamente a 6 W, al establecerse una restricción en la potencia de salida permitida del generador de señal. Los ajustes en el analizador de señal (nivel de referencia y atenuación de entrada) se configuran de tal manera que no distorsionen la señal medida, incluso para el nivel de potencia de pico más alto. Esto significa que experimentará una degradación en el rendimiento de ruido cuando se consideren niveles de potencia inferiores al máximo permitido.

Este acceso remoto a una configuración de medición de última generación se diseñó inicialmente para los participantes en la competición de estudiantes del IMS, pero gracias al patrocinio de equipos de medición propietarios de National Instruments, la plataforma se encuentra permanentemente disponible para todos. Utilizar estos recursos posibilita que aquellos grupos de investigación que no tienen acceso a equipos de medida puedan realizar mediciones utilizando una configuración experimental de última generación y, por lo tanto, aumenten la comprensión de las imperfecciones del hardware en la calidad de la señal en los transmisores modernos de los sistemas de comunicaciones.

Otra de las ventajas de este sistema de medición remoto es que no es necesario registrarse para su uso. Existen dos opciones para cargar los archivos de datos:

- Usar cualquiera de las funciones de Matlab proporcionadas por el grupo de Chalmers: PA_meas_Pin.m o PA_meas_xs.m.
- Usar LabVIEW VI.

La utilizada en este proyecto será la primera. Con el fin de proteger el sistema remoto de medición WebLab, nuestro código y el de cualquier usuario que desee acceder a él debe cumplir las siguientes restricciones:

Para proteger el amplificador y el sistema contra niveles excesivamente altos, se imponen limitaciones a la potencia máxima de salida y a la potencia de salida rms del generador de señal vectorial. Se supone que el sistema tiene una impedancia de referencia de 50 Ohm. La potencia máxima permitida del generador de señal es -8 dBm. El nivel máximo permitido de potencia rms (Pin, RMS) depende de la relación de potencia pico a promedio (PAPR) de la señal de entrada y asegurará que los picos de las señales de entrada permanezcan durante la mayor parte del tiempo por debajo de -8 dBm. En este sentido, se permite incrementar ligeramente la potencia

RMS cuando se trabaja con señales de PAPR baja. También se establece una relación máxima de potencia de pico a promedio de la señal de datos de BB de 20 dB. Si se excede cualquiera de estos límites, el sistema no realizará la medición. El resultado en estos casos es un archivo de datos que contiene un único valor "-inf".

Se acepta como entrada un máximo de 1 000 000 de muestras de datos de BB. Si se carga un archivo con más de 1 000 000 de muestras, solo se utilizan las primeras 1 000 000 de muestras. El número de muestras en el archivo de datos de salida (csv o dat) es de la misma longitud que la señal de entrada (o 1 000 000 de muestras si la señal de entrada era más larga).

La velocidad máxima de muestreo tanto en el generador como en el analizador es de 200 MHz. El ancho de banda máximo útil es de 160 MHz. Esto significa que cualquier componente de señal que se coloque fuera del rango de frecuencia [-80 80] MHz se elimina antes de ser enviado desde el generador de señal. La señal que se recibe del analizador de señal también se limita al ancho de banda de 160 MHz, es decir, utiliza solo las componentes de señal en el rango [-80 80] MHz porque todo lo que esté fuera de este rango se distorsiona por un filtrado.

## 5.2 Matlab

Es un entorno de cálculo técnico de altas prestaciones para el cálculo numérico y visualización. Entre sus prestaciones básicas destacan el análisis numérico, el cálculo matricial, el procesamiento de señales y la representación de gráficos.

Es un entorno intuitivo y fácil de usar, donde los problemas y las soluciones son expresados como se escriben matemáticamente, sin la programación tradicional. Es un sistema interactivo cuyo elemento básico de datos es una matriz que no requiere dimensionamiento. Esto permite resolver muchos problemas numéricos en una fracción más pequeña del tiempo que llevaría hacerlo en otros lenguajes de programación como C, Basic y Fortran. Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

Matlab también proporciona una serie de soluciones específicas denominadas *Toolboxes*. Estas son muy importantes para la mayoría de los usuarios ya que son conjuntos de funciones que permiten extender el entorno de Matlab para resolver determinados problemas como:

- Procesamiento de señales.
- Diseño de sistemas de control.
- Simulación de sistemas dinámicos.

Para el desarrollo de este proyecto se emplean dos *Toolboxes*: el primero de ellos es “Signal Processing ToolBox” el cual es un paquete que proporciona funciones y aplicaciones para generar, medir, transformar, filtrar y visualizar señales. Además, incluye algoritmos para remuestrear, suavizar y sincronizar señales, diseñar y analizar filtros, estimar espectros de potencia y medir picos, ancho de banda y distorsión. El segundo es el “Communications System Toolbox” caracterizado por proporcionar algoritmos y aplicaciones para el análisis, diseño, simulación extremo a extremo, y la verificación de los sistemas de comunicaciones en Matlab.

## 5.3 Python

Desde su lanzamiento en 1991 el lenguaje de programación Python ha experimentado un crecimento asombroso. Este crecimiento se debe fundamentalmente a la irrupción de las nuevas tecnologías de Inteligencia Artificial, Machine Learning y Deep Learning, donde junto con R es el lenguaje predominante. Actualmente es el lenguaje de programación de mayor crecimiento, y las tendencias hacen preveer que así será en los próximos años (véanse las figuras 5-2 y 5-3).

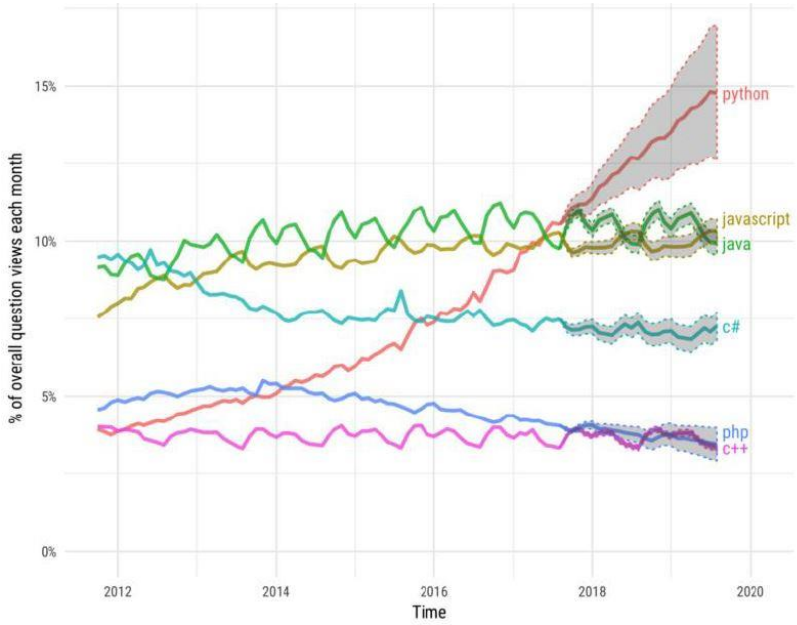


Figura 5-2. Uso del lenguaje en los últimos años [Fuente: Stackoverflow].

| Rank | Language | Type | Score |
|---|---|---|---|
| 1 | Python | | 100.0 |
| 2 | Java | | 96.3 |
| 3 | C | | 94.4 |
| 4 | C++ | | 87.5 |
| 5 | R | | 81.5 |
| 6 | JavaScript | | 79.4 |
| 7 | C# | | 74.5 |
| 8 | Matlab | | 70.6 |
| 9 | Swift | | 69.1 |
| 10 | Go | | 68.0 |

Figura 5-3. Clasificación de los lenguajes principales [20].

Es un lenguaje de programación interpretado que busca desarrollar una sintaxis que priorice la legibilidad del código. Su característica multiparadigma le hace soportar diferentes orientaciones: orientado a objetos, a

programación imperativa y funcional. Adicionalmente a esto, Python cuenta con iteraciones rápidas de datos que favorecen la concentración en los datos y en el desarrollo de los algoritmos.

Otro aspecto fundamental de su éxito es el gran número de librerías que posee. Existen miles de librerías de data science y matemáticas, pero el sistema de empaquetamiento de Python permite construir librerías custom sobre las ya existentes aumentando su potencia. Por último, destacar la capacidad de este lenguaje para combinar librerías como NumPy y ScyPi, permitiendo obtener un mayor redimiento en el procesado de datos.

## 5.4 Spyder

Es un entorno de desarrollo integrado y multiplataforma de código abierto (IDE) para programación científica basada principalmente en Python, aunque soporta otros lenguajes como R. Presenta funciones avanzadas de edición, pruebas interactivas, depuración e instrospección y un entorno informático numérico. Todo ello gracias al soporte de IPython (intérprete interactivo mejorado de Python) y numerosas bibliotecas. Está incluido en la distribución multiplataforma Anaconda [21] desarrollada por Continuum Analytics. Anaconda se caracteriza por ser una distribución libre y por incluir una gran colección de paquetes y librerías para el análisis de datos, computación científica e ingeniería. Algunas de las características más relevantes de Spyder son las siguientes:

- Plataforma cruzada, disponible en Linux, Windows y macOS.
- Consola interactiva.
- Explorador de variables.
- Visualización de gráficos y datos.

### 5.4.1 Librerías

Los paquetes y librerías empleados en la implementación del modelo RVTDCNN son los siguientes:

- **Numpy [22]:** paquete para Python de computación científica. Sus posibilidades son extensas, permitiendo trabajar con matrices N-dimensionales además de implementar álgebra lineal, transformada de Fourier y muchas funciones relacionadas con estadística.
- **Matplotlib [23]:** librería para la representación de gráficos en 2D y 3D.
- **Scipy [24]:** librería que permite leer y escribir datos en una gran variedad de formatos de archivo.

### 5.4.2 Keras

Existen númerosos *frameworks* en el mercado para la implementación de redes neuronales entre los que destacan TensorFlow, Theano, Keras y Pythorch.

Entre los anteriormente citados, se ha seleccionado Keras para la implementación de nuestro modelo. Este proporciona un proceso de experimentación rápida, ofreciendo una curva de aprendizaje rápida. Esto unido a

que es una de las herramientas más populares para la implementación de redes neuronales trás TensorFlow han propiciado su elección. Aporta una sintaxis homogénea y una interfaz sencilla, modular y ampliable para la creacion de redes neuronales.

Keras se puede usar como una librería de Python que proporciona, de una manera sencilla, la creación de una gran gama de modelos de Deep Learning usando como backend otras librerías como TensorFlow o Theano, siendo Tensorflow el elegido para trabajar bajo Keras al ser la librería más popular en Deep Learning.

## 5.5 Consideraciones sobre el código generado

Este apartado se inicia con la presentación de las características de la señal de entrada utilizada y finaliza con la descripción del funcionamiento de cada uno de los archivos que componen el trabajo.

### 5.5.1 Señal de entrada

La señal de entrada simulada para la generación de los resultados es una señal OFDM con 512 subportadoras y una constelación 128-QAM. Las suportadoras están espaciadas para producir un ancho de banda de aproximadamente 10 MHz. La señal OFDM está centrada en el origen [-5 5] MHz (BB), y es filtrada utilizando un filtro de coseno alzado. Tiene una frecuencia de muestreo fijada a 200 MHz y una duración de 500 $\mu$s. El nivel de potencia RMS de la señal de entrada es de -22 dBm; éste es el valor máximo de potencia permitido que cumple con las especificaciones de seguridad establecidas por la plataforma, pues la PAPR típica de la señal de entrada es del orden de 9 dB.

A continuación, en las figuras 5-4 y 5-5 se representan el voltaje de la parte real e imaginaria frente al tiempo y el espectro de densidad de potencia respectivamente de la señal OFDM simulada:

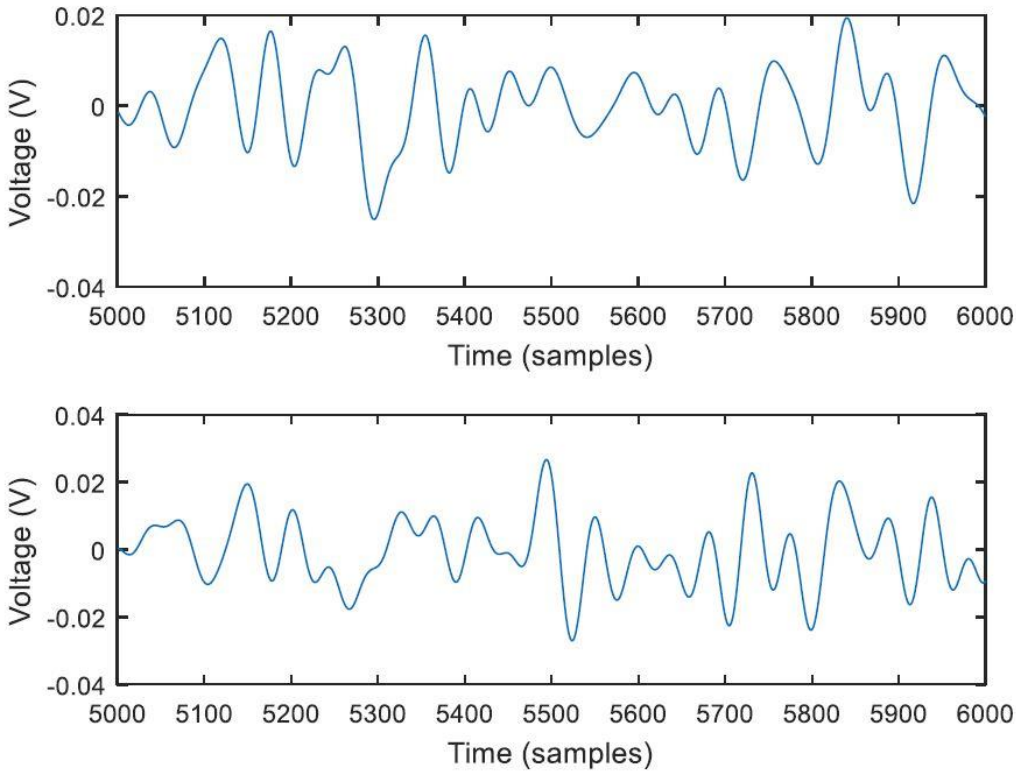


Figura 5-4. Voltaje frente a tiempo de la parte real e imaginaria de la señal de entrada.

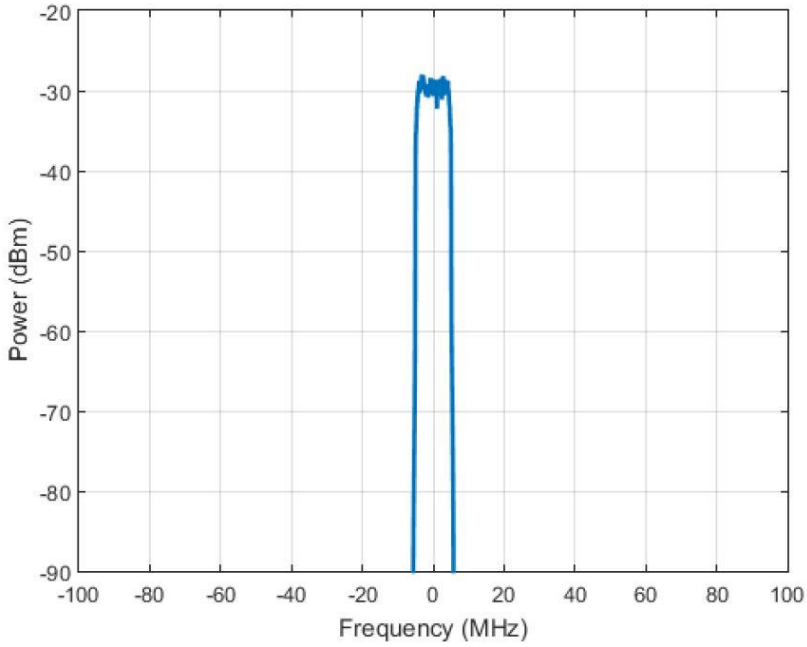


Figura 5-5. Espectro de densidad de potencia de la señal de entrada.

### 5.5.2 Descripción del código generado

A continuación, se expone las distintas funciones consideradas para la implementación del modelo:

- **IMS2016_generate_signal.m:** El objetivo de esta función es generar la señal presentada en el apartado 5.5.1, que posteriormente será enviada al PA para realizar las simulaciones. Recibe como parámetro una semilla utilizada para la generación de los datos aleatorios que conforman la señal OFDM. Devuelve el vector x_ofdm, que contiene las muestras de la envolvente compleja discreta de la señal generada, la variable fs que representa la frecuencia de muestreo y la variable f_ofdm que representa la frecuencia central de la señal OFDM. Las características de la señal generada han sido expuestas en el apartado anterior.
- **RFWebLab_PA_meas_v1_1.m:** Esta función es la encargada de enviar la señal generada por IMS2016_generate_signal al amplificador de potencia remoto. Recibe la señal OFDM generada (muestras en BB) y su potencia en dBm. Antes de enviar la señal al amplificador comprueba que cumpla con todos los requisitos de seguridad expuestos en el apartado Plataforma WebLab de este capítulo. Esta función devuelve las muestras BB de la señal a la salida del amplificador de potencia, la potencia en dBm de ésta, así como la corriente y voltaje medidos. La señal devuelta tiene la misma longitud que la enviada, pero debemos tener en cuenta que el muestreo se inicia de forma completamente aleatoria, por lo que se debe realizar una sincronización antes de utilizar los datos de salida.
- **Mysynch_IMS2016.m:** Recibe como parámetros de entrada la señal enviada al amplificador (generada por IMS2016_generate_signal) y la señal a la salida de éste. Devuelve ambas señales sincronizadas, resolviendo de este modo el problema de sincronización mencionado en la descripción de la función RFWebLab_PA_meas_v1_1. Además, devuelve la potencia media de salida, el ACPR de la señal OFDM y el NMSE entre la entrada y la salida del amplificador.
- **Espectro.m:** Recibe como parámetros de entrada la envolvente compleja discreta de una señal y su frecuencia de muestreo. Devuelve su espectro de densidad de potencia. En primer lugar, se crea una ventana de Kaiser con un ancho de banda de 8 kHz y una atenuación de lóbulo lateral de 50 dB. Utilizando la ventana anterior, se estima la densidad espectral de potencia a través del periodograma de Welch, obtenido mediante la función pwelch implementada en Matlab, utilizando 8000 puntos. Finalmente, la densidad espectral de potencia se centra en el origen utilizando la función fftshift.
- **Test_MP.m:** Este script implementa el predistorsionador basado en el modelo Memory Polynomial. Para introducir la memoria modificamos las dimensiones del vector de entrada al predistorsionador añadiéndole Q muestras. El valor Q representa la profundidad de memoria seleccionada. Como utilizamos la configuración de aprendizaje indirecto comenzamos obteniendo las señales sincronizadas a la entrada y salida del amplificador de potencia. Para ello se hace uso de las funciones expuestas anteriormente. Continuamos con el cálculo de la ganancia de compresión para poder modelar correctamente la señal de entrada al predistorsionador que será la señal de salida del amplificador de potencia dividida por la ganancia de compresión. Una vez tenemos las señales de entrada y salida bien dimensionadas calculamos los coeficientes del predistorsionador resolviendo la ecuación $\mathbf{w} = \mathbf{Y^{-1}} \cdot \mathbf{x}$, donde $\mathbf{Y^{-1}}$ representa la pseudoinversa de la matriz $\mathbf{Y}$. Esta ecuación es resuelta mediante la función pinv implementada en Matlab, la cual utiliza el algoritmo de mínimos cuadrados (LS). Tras haber obtenidos los coeficientes se comprueba el sistema completo, es decir, la cascada de predistorsionador y amplificador de potencia.
- **Input_layer.py:** Script en Python encargado de la implementación del modelo RVTDCNN. Comienza con la definición de la función input_layer, encargada de realizar la funcionalidad de la capa de entrada. Es decir, conformar los datos en el formato matriz presentado en la ecuación 4-1. El código fuente de esta función se muestra en la figura 5-6.

```
def input_layer(X,M,L):
    # X: Vector de entrada con las muestras de la envolvente compleja.
    # M: Profundidad de memoria.
    M = M + 1 # Adaptamos la variable
    X_n = np.empty((L,5,M,1)) # Creamos matriz vacia para almacenar datos de entrada.
    for h in range(L):
        for j in range(5):
            for i in range(M):
                if j == 0:
                    X_n[h][j][i][0] = np.real(X[h-i])
                elif j == 1:
                    X_n[h][j][i][0] = np.imag(X[h-i])
                elif j == 2:
                    X_n[h][j][i][0] = np.abs(X[h-i])
                elif j == 3:
                    X_n[h][j][i][0] = np.abs(X[h-i]) ** 2
                else:
                    X_n[h][j][i][0] = np.abs(X[h-i]) ** 3
    return X_n
```

Figura 5-6. Código fuente de la función input_layer.

Recibe como parámetros de entrada un vector con las muestras de la envolvente compleja de la señal de entrada, la profundidad de memoria seleccionada y el número de muestras que contiene el vector. Devuelve X_n, un array de 4 dimensiones, (L,N,M,Z) que será el input shape de la capa convolucional, es decir, la matriz de entrada. N y M representan el número de filas y columnas que poseé la matriz. Como se expuso en el capítulo 4 el número de filas es fijado a 5 mientras que el número de columnas depende de la profundidad de memoria seleccionada. L representa el número de matrices, las cuales se conforman en una tercera dimensión. Se crearán tantas matrices como elementos contenga el vector de entrada. Finalmente, Z representa el número de subcapas, el parámetro depth presentado en el capítulo 3, cuyo valor es 1 ya que consideramos la "imagen" sin color.

Una vez definida la función input_layer, se procede a la carga de los datos de entrenamiento utilizados en la fase de entrenamiento del modelo, además de definir la profundidad de memoria del mismo. Una vez cargados, se llama a la función input_layer para adaptarlos al formato definido en la ecuación 4-1.

```
M = 3 # Definimos la profundidad de memoria.
dict_x = sio.loadmat('./in_training.mat') # Cargamos la señal de entrada al amplificador
X = dict_x['x_0'] # Al cargar el archivo .mat devuelve un diccionario.
L = len(X) # Obtenemos el numero de filas, X tiene que ser un vector columna
training_data = input_layer(X,M,L)
```

Figura 5-7. Carga de los datos de entrada.

El proceso de conformación del modelo comienza cargando los datos de entrada del mismo (véase la figura 5-7). En primer lugar, definimos la profundidad de memoria. Para la carga de los datos utilizamos la función loadmat perteneciente al modulo sio de la librería Scipy, esta permite cargar la señal OFDM generada por la función IMS2016_generate_signal.m presentada anteriormente. Estos datos, procedentes de Matlab, se cargan como un diccionario, un tipo de dato muy utilizado en Python caracterizado por el formato clave-valor. Para seleccionar las muestras de la envolvente compleja de la señal procedemos como se indica en la figura 5-7. Finalmente, obtenemos el número de muestras que contiene la señal de entrada y se llama a la función input_layer para adaptar los datos de entrada al formato correcto.

A continuación, se procede a la carga de las "label" o datos objetivo. El código fuente utilizado para ello se muestra en la figura 5-8.

```
dict_y = sio.loadmat('./out_training.mat')
t_data = dict_y['y_0']
p_real = np.real(t_data)
p_imag = np.imag(t_data)
target_data = np.empty((len(p_real),2))
target_data[:,0] = p_real[:,0]
target_data[:,1] = p_imag[:,0]
```

Figura 5-8. Carga de las labels o datos objetivo.

De nuevo se utiliza loadmat para la carga de los datos procedentes de Matlab. El formato de datos devuelto es el diccionario, presentado anteriormente. Una vez cargadas las muestras de la envolvente compleja de la señal de salida, se procede a su separación en parte real e imaginaria. Esta división se realiza para adaptar los datos a la capa de salida del modelo, que como se presento en el capítulo 4, presenta dos neuronas encargadas de la estimación de la fase y cuadratura de la señal de salida. Una vez realizada la división, se conforman los datos en una matriz de dos columnas, una primera que contiene la parte real y la segunda que contiene la parte imaginaria.

Una vez cargados los datos de entrenamiento y adaptados al formato deseado se procede a la definición del modelo RVTDCNN. Para ello se hace uso de la librería Keras, presentada en el apartado 5.4.2 de este mismo capítulo. En Keras a partir del modelo *sequential* se pueden ir añadiendo capas al mismo mediante el método *add()*. En primer lugar, se añade la capa convolucional como se muestra en la figura 5-9.

```
model.add(Conv2D(filters=3, kernel_size=3, strides=1, padding='valid',
                 activation='tanh', use_bias=True,
                 kernel_initializer='random_normal',
                 bias_initializer='random_normal', input_shape=(5, M+1, 1)))
```

Figura 5-9. Definición de capa convolucional en Keras.

Está compuesta por tres kernels de tamaño 3x3. El parámetro *strides* determina el desplazamiento de los kernels por la matriz de entrada, en nuestro caso el desplazamiento será de una unidad hacia la derecha en cada iteración y de una unidad hacia abajo una vez finalizada la fila. Tras realizar el proceso de convolución, la matriz resultante puede presentar dimensiones diferentes a la matriz de entrada. Cuando el parámetro *padding* toma el valor *valid* permite que estas dimensiones se mantengan, es decir, que la matriz de salida del proceso de convolución presente una menor dimensión que la matriz de entrada, no rellenándola con ceros para igualar dimensiones. *Activation* define la función de activación, que como se vio en el capítulo 4 la seleccionada para esta capa es la tangente hiperbólica. Finalmente, se inicializan aleatoriamente siguiendo una distribución normal los valores de los kernels y sesgo, además de definir las dimensiones de la matriz de entrada.

El siguiente paso consiste en aplanar la salida de la capa convolucional. Esta debe ser aplanada en formato vector para que se adecúe al formato de entrada de la capa completamente conectada (FC). La realización de este proceso es llevada a cabo por la siguiente sentencia de código (figura 5-10).

```
# Aplanamos la salida de la capa convolucional a verctor para que
# pueda ser ingresada a la capa totalmente conectada (FC).
model.add(Flatten())
```

Figura 5-10. Aplanado de la salida de la capa convolucional.

Tras adaptar los datos al formato vector añadimos la capa FC mediante el método *add()* (figura 5-11).

```
# Capa totalmente conectada FC.
model.add(Dense(6, activation='tanh', use_bias=True,
                kernel_initializer='random_normal',
                bias_initializer='random_normal'))
```

Figura 5-11. Definición de la capa FC en Keras.

Como se vió en el cuarto capítulo está compuesta por 6 neuronas. La función de activación utlizada es la tangente hiperbólica y los parámetros de la misma son inicializados de manera aleatoria siguiendo una distribución normal.

Llegados a este punto, únicamente falta por definidir la capa de salida. Estará compuesta por dos neuronas y los parámetros de la misma estarán inicializados de igual forma que en el resto de capas. En el código de definición de esta capa no se especifica la función de activación ya que por defecto Keras utiliza una función de activación lineal (figura 5-12).

```
model.add(Dense(2, use_bias=True, kernel_initializer='random_normal',
                bias_initializer='random_normal'))
```

Figura 5-12. Definición de la capa de salida en Keras.

Definida la estructura de capas del modelo se procede a configurar el proceso de aprendizaje del mismo haciendo uso del método *compile()*, siguiendo el código que se presenta en la figura 5-13. El comportamiento de este método viene definido por el valor de sus parámetros de entrada. El primero de ellos es la función de pérdidas, usada para evaluar el grado de error entre las salidas calculadas y las salidas deseadas de los datos de entrenamiento. En nuestro caso la seleccionada es la MSE. El segundo argumento es el optimizador, definido anteriormente con los parámetros deseados, el utilizado en nuestro caso es Adam. Los parámetros de este optimizador fueron presentados en el tercer capítulo. Finalmente, el tercer parámetro define la métrica utilizada para monitorizar el proceso de aprendizaje, en nuestro caso MSE.

```
# COMPILACION DEL MODELO
# Configuracion del optimizador.
adam_opt = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)

model.compile(loss='mean_squared_error', optimizer=adam_opt,
              metrics=['mse'])
```

Figura 5-13. Definición del compilador del modelo en Keras.

El útlimo paso tras la configuración del proceso de aprendizaje es entrenar al modelo (figura 5-14). Dicho entrenamiento se realiza mediante el método *fit()*. Este método recibe como parámetros de entrada los datos de entrada del entrenamiento y los datos esperados o labels, además del hiperparámetro

epoch el cual define el número de iteraciones que se realizan sobre el total de los datos de entrenamiento.

```
training_output = model.fit(training_data, target_data, epochs=100)
```

Figura 5-14. Definición del proceso de entrenamiento en Keras.

# 6 RESULTADOS

Este capítulo recoge los resultados obtenidos con el modelo RVTDCNN tanto en el modelado como en la predistorsión del PA analizado. Estará compuesto por dos secciones. Una primera en la que se exponen los resultados correspondientes al modelado de comportamiento del PA remoto, y una segunda donde se presenta la performance del modelo inverso para la predistorsión del mismo.

Para ambos casos la señal de entrada será común, siendo ésta una señal OFDM en BB como la presentada en 5.5.1. Su potencia es de -22 dBm que es el valor máximo que puede tomar sin violar las especificaciones de seguridad establecidas por la plataforma RF WebLab.

Con el fin de poder evaluar cada uno de los casos, se presentarán las siguientes figuras:

- Comparativa del espectro real y estimado, en el caso del modelado.
- Evolución del MSE durante las diferentes Epochs, en ambos casos.
- Comparativa del espectro de la señal de salida con y sin predistorsión (caso del DPD).
- Característica AM/AM, en ambos casos.
- Ganancia frente a potencia de entrada, en ambos casos.
- Característica AM/PM, en ambos casos.

## 6.1 Modelado del PA

Este apartado tiene como objetivo intentar modelar el comportamiento del PA. Para ello en primer lugar se precisa de una fase de entrenamiento la cuál hará uso de las señales de entrada y salida del PA. Una vez finalizado este proceso, se genera una nueva señal de entrada OFDM, con distinta semilla a la utilizada en la fase de entrenamiento, la cuál es enviada al modelo RVTDCNN que se encargará de estimar la señal de salida en este caso.

Los cambios provocados por el PA en la señal de entrada presentada en 5.5.1 se muestran en la figura 6-1.

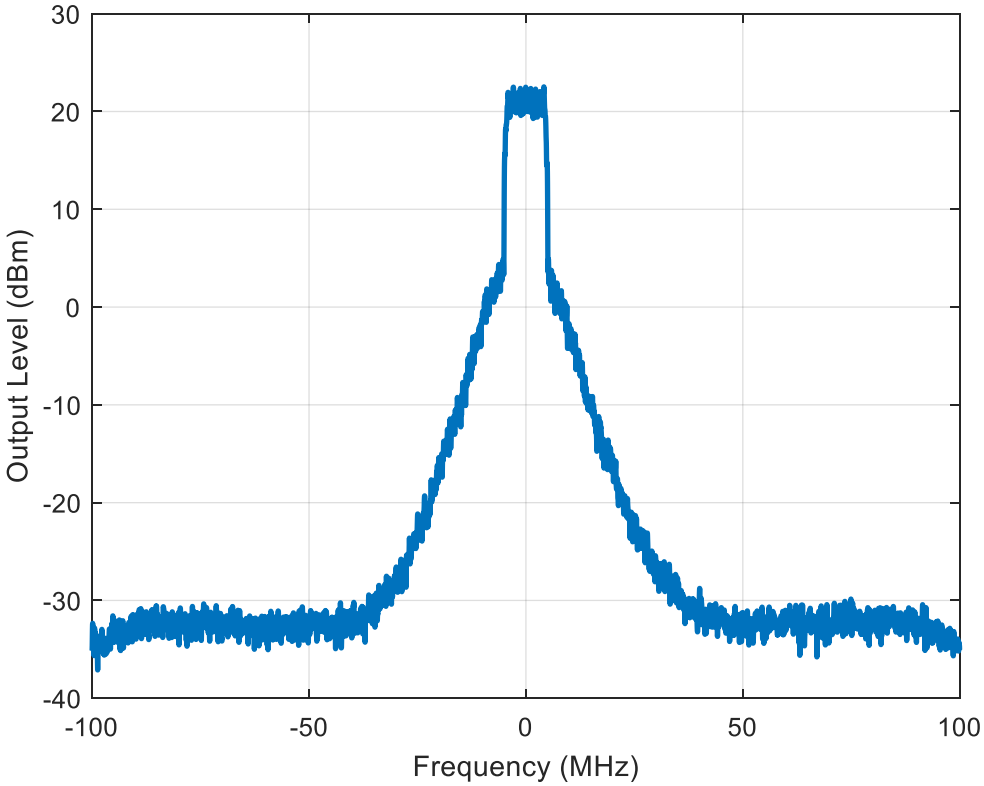


Figura 6-1. Espectro de la señal de salida del PA.

Se puede observar como el nivel de potencia de la señal ha aumentado tras su paso por el PA, sin embargo, este hecho ha provocado también un recrecimiento espectral. Este recrecimiento, como se indicó en apartados anteriores tiene una serie de consecuencias negativas y de ahí el uso de la predistorsión para mitigarlo. Este proceso de predistorsión se analizará en el apartado posterior por lo que en este se continuará con el modelado.

Una vez obtenidas las señales de entrada y salida del PA, se cargan sus muestras en Python para la realización del proceso de entrenamiento. Basándonos en la evolución de la MSE se ha fijado en cien el número de *epochs* utilizados en el proceso, previniendo de este modo el problema del *overfitting* presentado en la sección 3.5.3.

Una vez finalizada la fase de entrenamiento se genera una nueva señal OFDM con distinta semilla y se envía al PA. Esta nueva señal se carga en Python y será la señal de entrada utilizada por el modelo para la estimación del comportamiento del PA. Los datos estimados por el modelo se estructuran en una matriz de dos columnas y cien mil filas, la primera columna representa la parte real de la señal mientras que la segunda columna representa la parte imaginaria. Estos datos son almacenados con la extensión .mat para poderse cargar en Matlab. El siguiente paso consiste en cargar los datos en Matlab y corformarlos en un vector complejo. Finalmente, este nuevo vector formado por las muestras de la envolvente compleja de la señal estimada es enviado a la función “espectro.m” encargada de estimar su espectro. La figura 6-2 muestra una comparativa entre los espectros real y estimado.

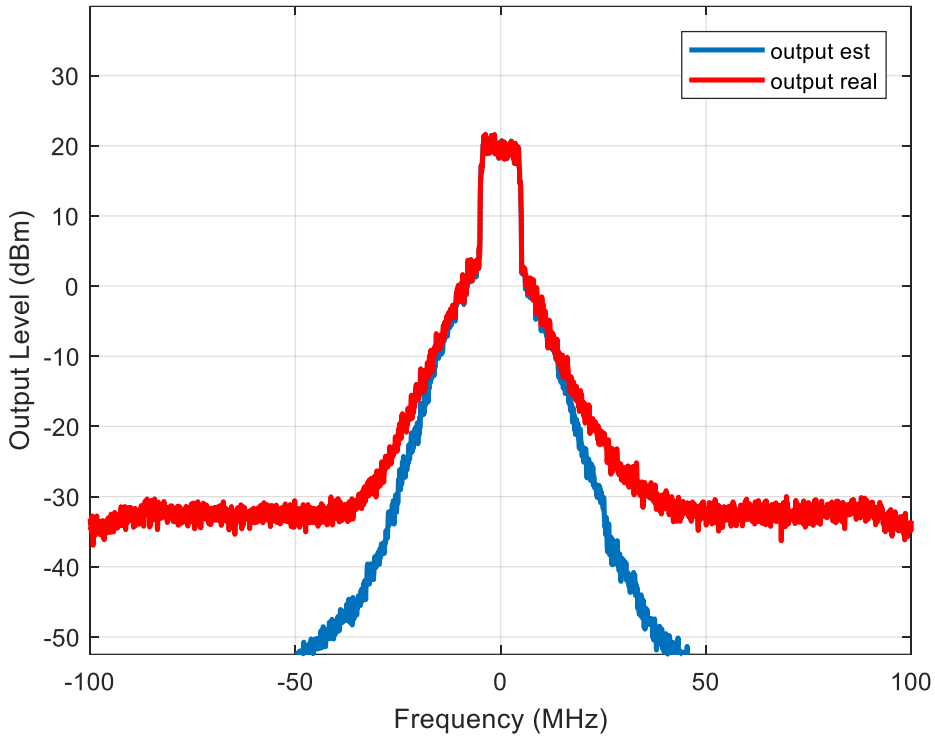


Figura 6-2. Comparativa de los espectros real y estimado de la señal de salida del PA.

Como se puede observar en la figura 6-2, la estimación del espectro en la banda de interés es muy buena ya que ambos espectros, real y estimado, son prácticamente idénticos. A partir de los 40 MHz aproximadamente estos comienzan a diferir lo cual es normal ya que a estas frecuencias el nivel de potencia de la señal es muy bajo no pudiendo nuestro modelo estimar correctamente. Por otra parte, el NMSE resultante entre la señal real ("y" real del PA) y la modelada ("y" estimada del PA) es de -23.8814 dB, el cual es un valor bastante acceptable, mientras que el valor del ACPR sin el uso de DPD es de 20.4410 dB.

La figura 6-3 representa la evolución de la MSE a lo largo de las distintas *epochs*. Se puede observar como la curva presenta una convergencia clara. Durante las primeras *epochs* el valor de MSE disminuye drásticamente hasta los 0.05, a partir de la epoch 15 el valor de MSE continúa disminuyendo, pero de una manera más suave.

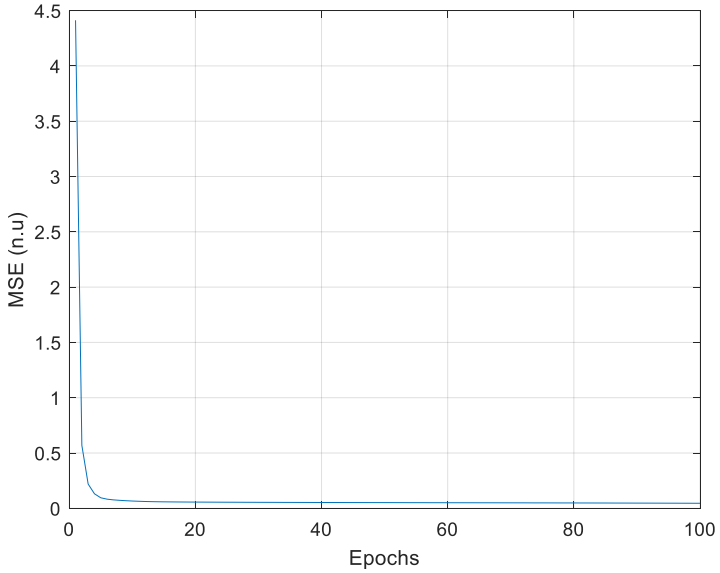


Figura 6-3. Evolución del MSE frente al número de Epochs.

Finalmente, se representan la característica AM/AM (figura 6-4), ganancia frente a potencia de entrada (figura 6-5) y característica AM/PM del PA (figura 6-6).

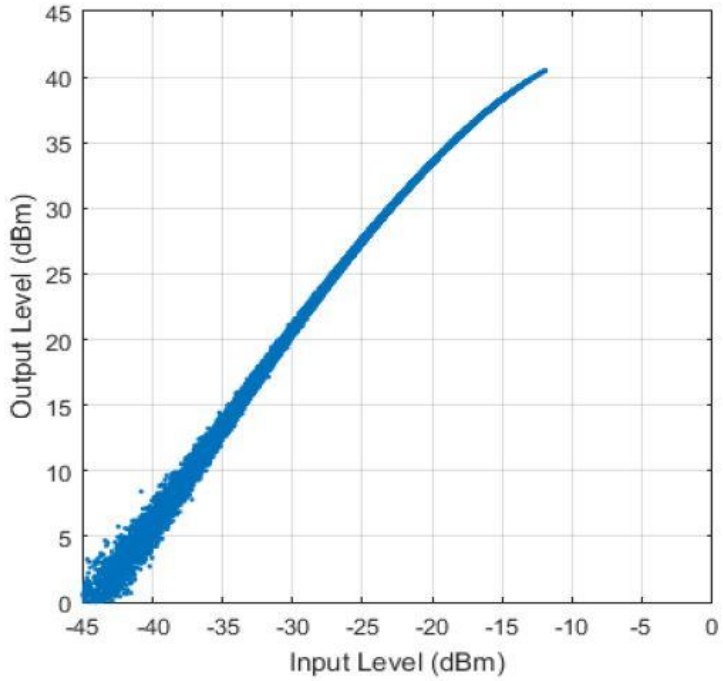


Figura 6-4. Característica AM/AM sin DPD.

En la figura 6-4 se observa una zona donde la potencia de salida crece linealmente con la potencia de entrada, esta zona es conocida como zona lineal. Está comprendida entre los -45 y -24 dBm, para potencias de entrada superiores a -24 dBm la potencia de salida ya no crece linealmente, sino que satura. Esta zona se conoce como zona de saturación y es en la que se trabajará para obtener la mayor eficiencia por parte del PA.

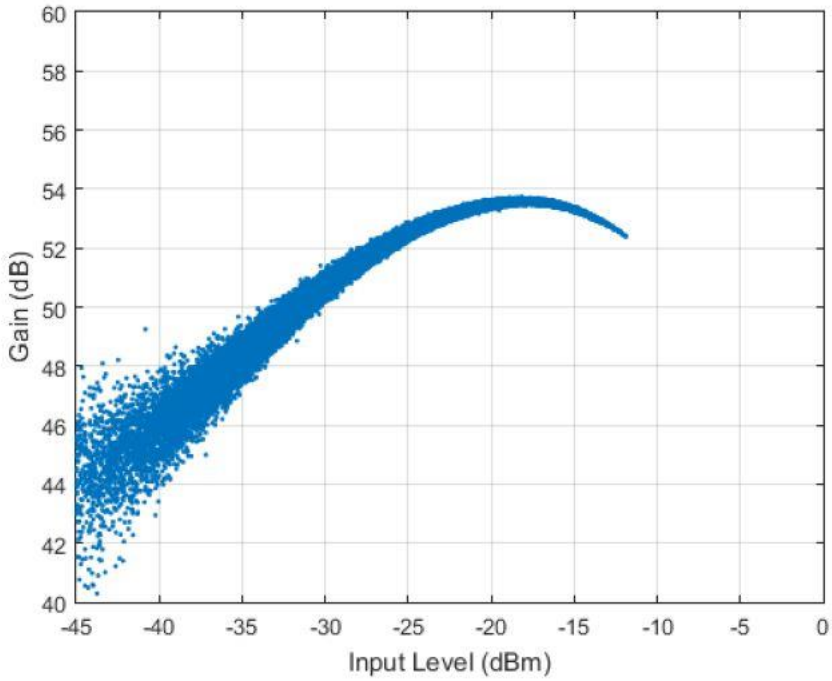


Figura 6-5. Ganancia frente a potencia de entrada sin DPD.

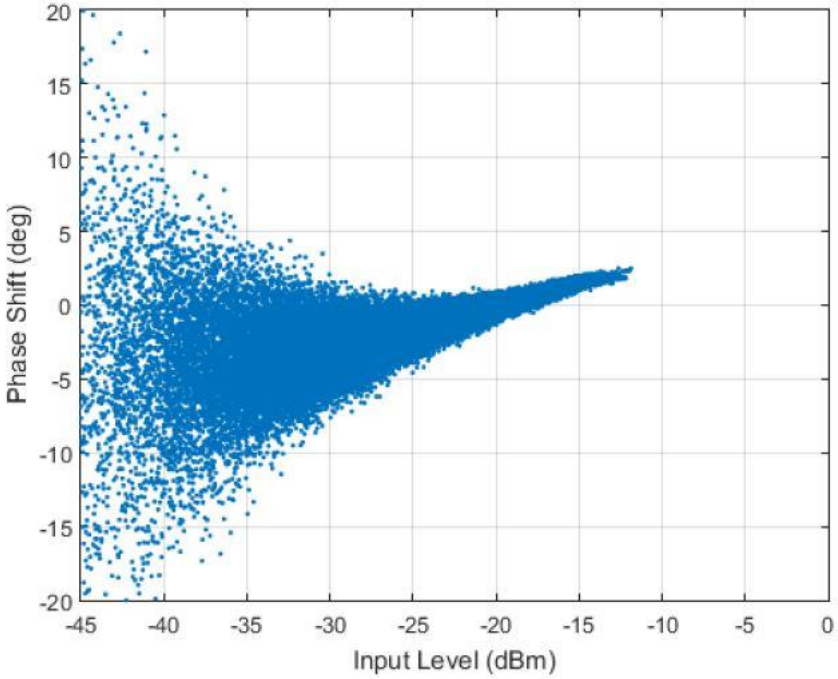


Figura 6-6. Característica AM/PM sin DPD.

## 6.2 DPD mediante modelo inverso

Como ya se indicó en el apartado 4.1.2, se utiliza el inverso de nuestro modelo para la implementación del DPD.

Con inverso nos referímos a que únicamente se modifican las señales de entrada y salida utilizadas en el proceso de entrenamiento y posterior estimación. Sin embargo, la estructura del modelo sigue siendo la misma, es decir, mismo número de capas con iguales dimensiones.

La señal de entrada de nuestro modelo será la salida del PA normilizada respecto a la ganancia de compresión. En el apartado 4.1.2 se presento el critero seleccionado para la elección de la ganancia utilizada en la normalización. Mientra que las "labels" o señal objetivo de nuestro DPD será la señal de entrada del PA.

Al igual que en el modelado de comportamiento, la estimación del DPD es un proceso que se compone de dos fases. La primera de ellas es la fase de entrenamiento, realizada de igual forma que en el modelado, misma función de coste, mismo optimizador y misma métrica. El número de epochs utilizados también es el mismo al igual que la profundidad de memoria, 3. La única diferencia con respecto a la fase de entrenamiento del modelado radica en el número de muestras de señal utilizadas, que en el caso del DPD únicamente son mil. La convergencia de la métrica en este proceso también es impecable, obteniéndose durante todo el proceso un MSE menor que en la fase de entrenamiento del modelado. Los resultados obtenidos se muestran en la figura 6-7, en ella se puede observar como durante las 4 primeras epochs disminuye drásticamente, después se estabiliza hasta las epoch 18 y finalmente vuelve a seguir disminuyendo de forma más suave.

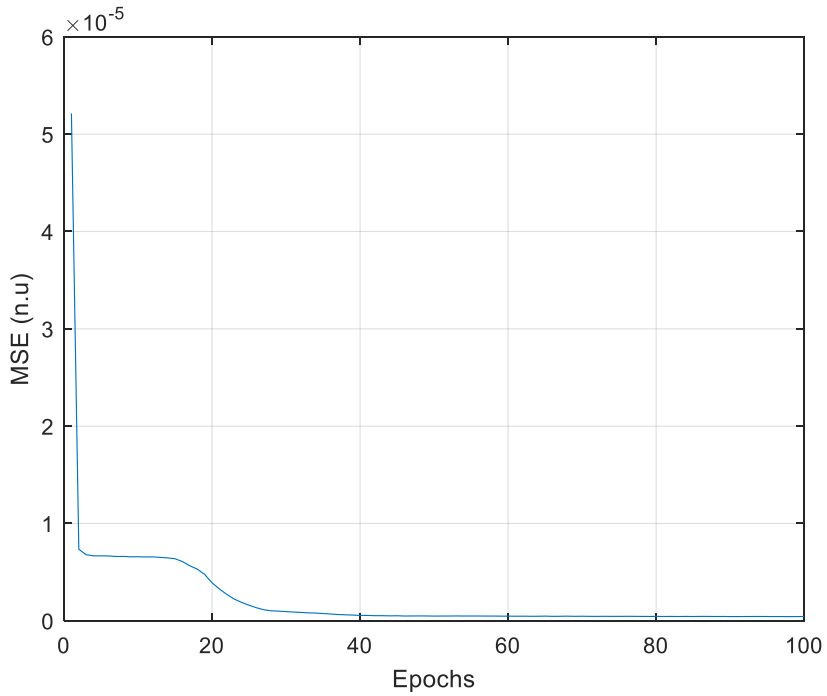


Figura 6-7. Evolución del MSE frente al número de Epochs.

Una vez finalizada la fase de entrenamiento, nuestro modelo se encuentra en condiciones de estimar la salida del DPD. Para ello, generamos una nueva señal OFDM con diferente semilla a la utilizada en la fase de entrenamiento. Esta es enviada a nuestro modelo que se encargará de aplicarle las transformaciones necesarias para producir una señal predistorsionada. Esta nueva señal predistorsionada es finalmente enviada al PA remoto. La figura 6-8 representa los resultados obtenidos, poniendo de manifiesto los beneficios introducidos por el DPD. En ella se comparan los espectros de salida obtenidos en presencia o no de DPD. Su uso ha mitigado el ensanchamiento espectral. Aunque dicho ensanchamiento no se ha erradicado, con el DPD se produce para niveles de potencia inferiores lo cual es beneficioso ya que el nivel de potencia interferente en el canal adyacente es menor.

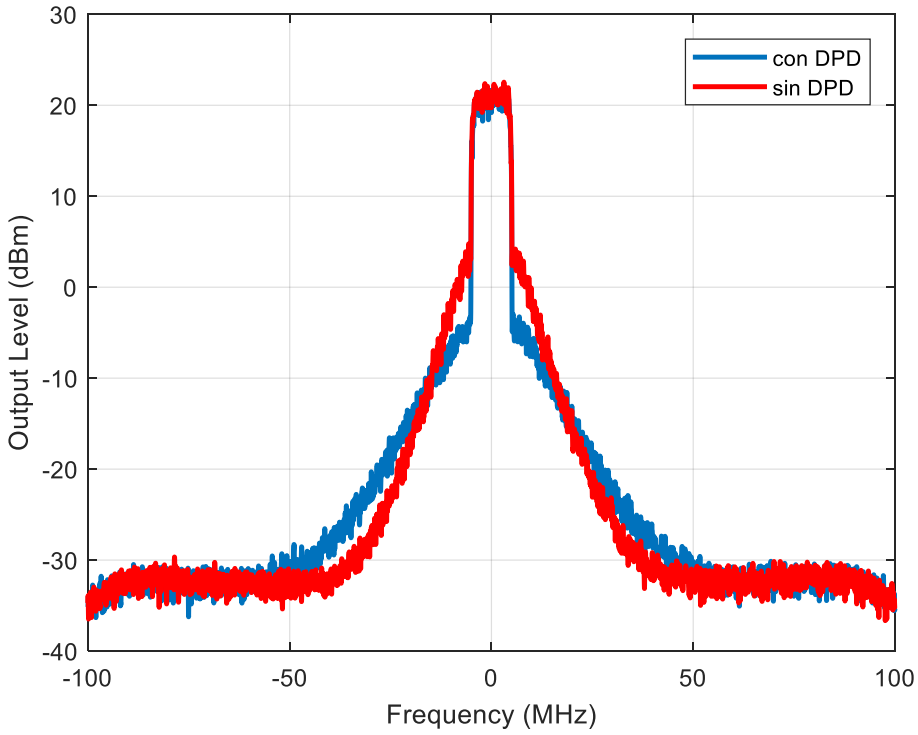


Figura 6-8. Espectro señal de salida (DPD mediante modelo inverso de RVTDCNN).

El uso del DPD ha provocado mejoras en el valor del ACPR obtenido. Inicialmente, se calculó en ausencia del DPD y se obtuvo un valor de 19.450 dB. Tras introducir el DPD el valor mejoró hasta los 28.1034 dB.

Finalmente, se representa como modifica la característica AM/AM (figura 6-9), ganancia frente a potencia de entrada (figura 6-10) y característica AM/PM (figura 6-11) la presencia de DPD.

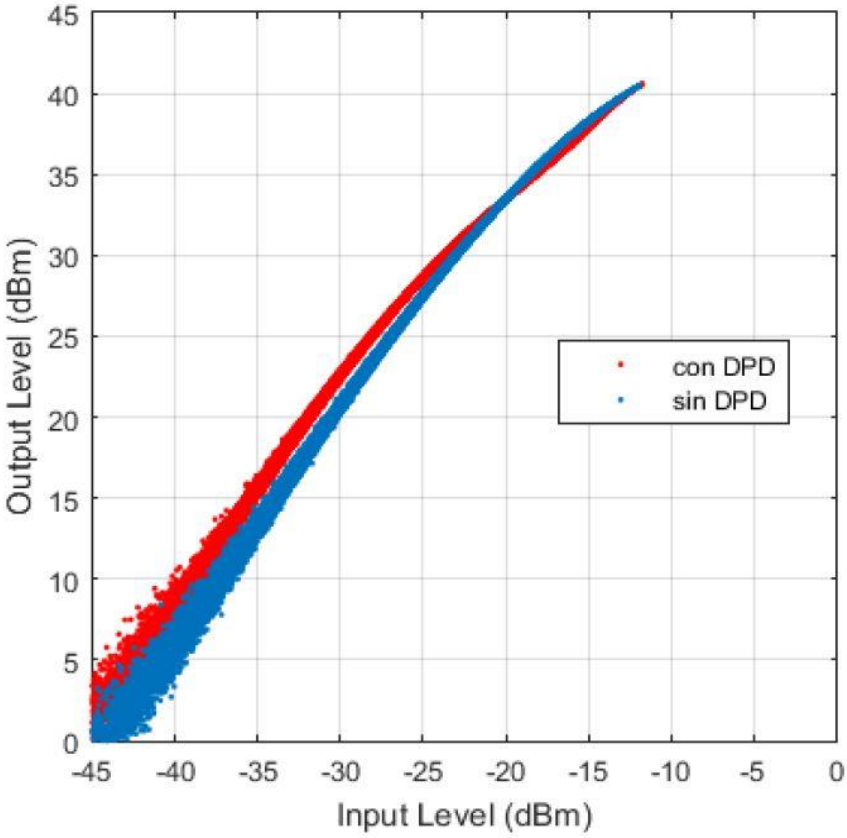


Figura 6-9. Característica AM/AM con DPD.

Con el uso del DPD se puede observar que para un mismo nivel de potencia de entrada se obtiene un nivel de potencia de salida mayor. Además, se ha aumentado en pequeña medida el rango de la zona lineal.

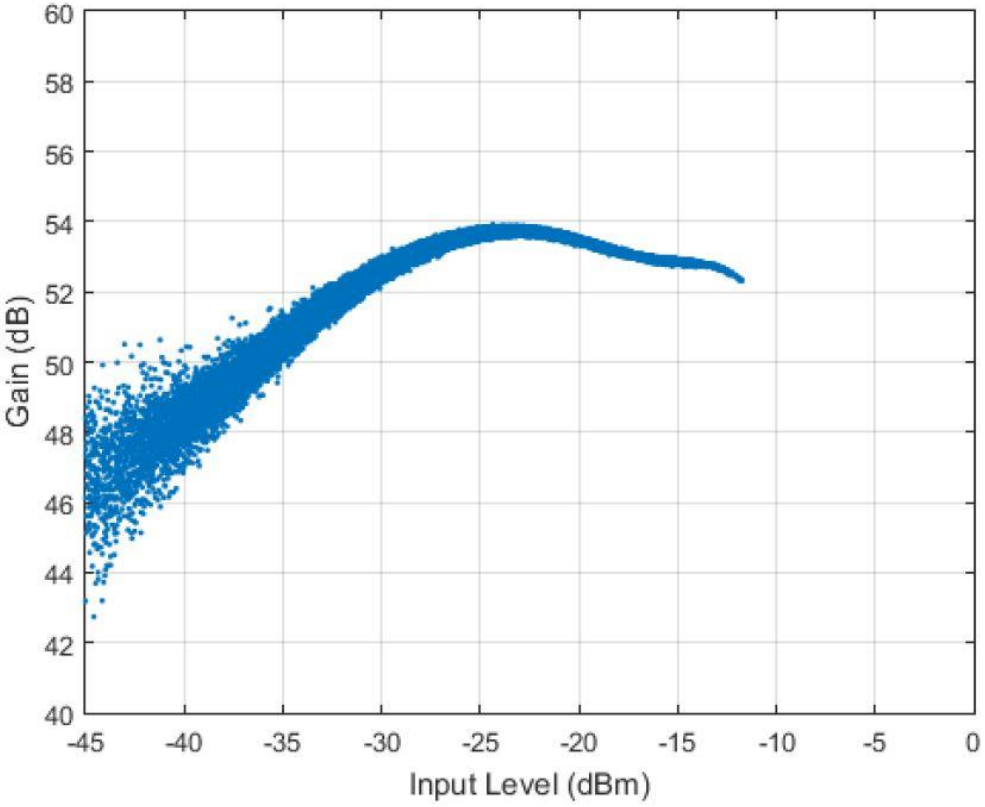


Figura 6-10. Ganancia frente a potencia de entrada con DPD.

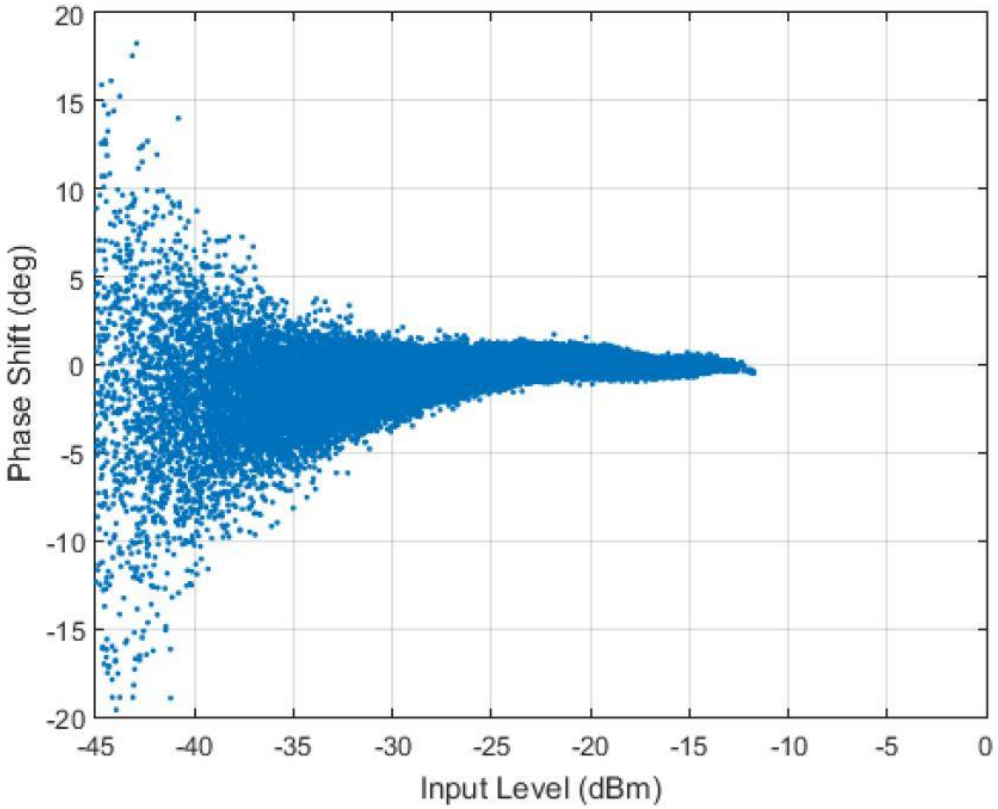


Figura 6-11. Característica AM/PM con DPD.

# 7 CONCLUSIONES Y LÍNEAS FUTURAS DE INVESTIGACIÓN

## 7.1 Conclusiones

El presente Trabajo fin de máster surgió por la nueva tendencia de utilizar algoritmos de *Machine Learning* en el modelado y linealización de uno de los dispositivos más importantes y usados en los sistemas de comunicaciones actuales como es el PA. Dentro de los algoritmos de Machine Learning, el comúmente utilizado en este cometido pertenece a la categoría del aprendizaje supervisado y son las redes neuronales.

El uso de estas surge a raíz de las limitaciones que presentaban los modelos clásicos de comportamiento, como el de Volterra, en la precisión del modelado fruto de la correlación existente entre sus bases polinómicas. Previo a la linealización del PA es necesario su modelado, y de ahí que en este Trabajo nos hayamos centrado en ambos. Para la linealización se han presentado varias técnicas a nivel de sistema como son el Feedback, Feedforward, LINC, EE&R, CALLUM y la predistorsión digital. Siendo esta última, predistorsión digital, la seleccionada.

Los efectos de las no linealidades son nocivos tanto dentro como fuera de banda por ello se han implementado modelos que realizan una predistorsión digital sobre la señal y no sobre los datos, teniendo en cuenta de este modo la distorsión fuera de banda. Algunos de los principales efectos son:

- Distorsión fuera de banda: Ensanchamiento del espectro introduciéndose en los canales adyacentes.
- Distorsión dentro de banda: Se produce una distorsión en la constelación de la señal de salida. El valor del parámetro EVM cuantifica esta distorisón.

Por tanto, para evitar los efectos anteriormente citados se ha implementado un DPD basado en el modelo inverso del modelo RVTDCNN.

Por otro lado, el modelo RVTDCNN es implementado para modelar las no linealidades y los efectos de memoria del PA remoto WebLab. Este modelo está compuesto por cuatro capas, una capa de entrada, una capa convolucional, una capa completamente contectada y una capa de salida. RVTDCNN se encarga de extraer de manera efectiva las características más importantes de los datos de entrada, organizados en una matriz bidimensional confeccionada por la capa de entrada, a través de la capa convolucional. La selección de este modelo se debe precisamente al uso de la capa convolucional, la cual gracias a la característica de peso compartido logra reducir considerablemente la complejidad computacial requerida.

Analizando en primer lugar los resultados obtenidos en el modelado, se puede observar que los espectros estimado y real son bastante similares. Además, se obtiene un valor de NMSE resultante entre la señal medida y modelada bastante bueno.

Finalmente, los resultados obtenidos tras la predistorisón muestran que se reduce la distorsión fuera de banda, provocándose un estrechamiento del espectro de la señal de salida y aumentando en casi 10 dB el valor del ACPR.

## 7.2 Líneas futuras de investigación

En un futuro sería recomendable probar el modelo RVTDCNN con otras señales de entrada para probar su validez y poder comparar los resultados obtenidos. Además, se podría extender el modelo para un amplificador multi-banda, utilizando como señal de entrada una con dos bandas.

Por otra parte, se podría realizar un estudio paramétrico del modelo con el fin de determinar si existe alguna combinación de parámetros que ofrezca mejores resultados que los obtenidos con la actual.

# REFERENCIAS


[1] X. Hu, Z. Liu, X. Yu, Y. Zhao, F. Ghannouchi y cols., "Convolutional Neural Network for Behavioral Modeling and Predistortion of Wideband Power Amplifiers," arXiv:2005.09848, 2020.

[2] A. Corral Sierra, "Identificación por mínimos cuadrados en predistorsionadores de Volterra," Trabajo fin de Grado, Universidad de Sevilla, 2018.

[3] M. S. Vázquez Rodríguez, "Implementación en DSP de un predistorsionador digital para la linealización de amplificadores de potencia," Proyecto Fin de Carrera, Universidad Politécnica de Cataluña, 2006.

[4] A. Katz, J. Wood, D. Chokola, "The evolution of PA linearization: from classic feedforward and feedback through analog and digital predistortion," IEEE Microwave Magazine, vol. 17, no. 2, pp. 32–40, 2016.

[5] A. Zozaya, "Aportación a la linealización de amplificadores de potencia mediante la teoría de la hiperestabilidad," Tesis Doctoral, Departamento de Teoría de la Señal y Comunicaciones, Universidad Politécnica de Cataluña, 2002.

[6] D. K. Su, W. J. McFarland, "An IC for linearizing RF power amplifiers using envelope elimination and restoration," IEEE Journal of Solid-State Circuits, vol. 33, no. 12, pp. 2252-2258, 1998.

[7] J. M. Mozos Ruiz, "Técnicas de reducción del modelo para linealización de amplificadores de potencia mediante DPD," Trabajo Fin de Grado, Universidad Politécnica de Cataluña, 2016.

[8] D. Hall, M. Anderson, "Understanding RF & microwave specifications – Part 2", National Instruments, 2007.

[9] I. Goodfellow, Y. Bengio y A. Corville, "Deep Learning", MIT Press, 2016.

[10] S. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach", Prentice hall, 2009.

[11] K. Gurney, "An introduction to neural networks", University of Sheffield, 1997.

[12] T. Liu, S. Boumaiza, F. M. Ghannouchi, "Dynamic behavioral modeling of 3G power amplifiers using real-valued time-delay neural networks," IEEE Trans. Microw. Theory Techn., vol. 52, no. 3, pp. 1025-1033, Mar. 2004.

[13] M. Hui, T. Liu, M. Zhang, Y. Ye, D. Shen, X. Ying, "Augmented radial basis function neural network predistorter for linearisation of wideband power ampliers," Electron. Lett., vol. 50, no. 12, pp. 877-879, Jun. 2014.

[14] D. Wang, M. Aziz, M. Helaoui, F. M. Ghannouchi, "Augmented real-valued time-delay neural network for compensation of distortions and impairments in wireless transmitters," IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 1, pp. 242-254, Jun. 2019.

[15] M. Cilimkovic, "Neural Networks and Back Propagation Algorithm", Institute of Technology

Blanchardstown, Dublin, 2015.

[16] Diederik P. Kingma, Jimmy Lei Ba, "Adam: A Method for Stochastic Optimization," Proc. 3rd International Conference for Learning Representations (ICLR 2015), San Diego, USA, 2015, pp. 7-9.

[17] Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups.", Signal Processing Magazine, IEEE, 29(6):82–97, 2012a.

[18] Anding Zhu, Paul J. Draxler, Jonmei J. Yan, Thomas J. Brazil, Donald F. Kimball, "Open-Loop Digital Predistorter for RF Power Amplifiers Using Dynamic Deviation Reduction-Based Volterra Series," IEEE transactions on microwave theory and techniques, Vol. 56, No. 7, July 2008.

[19] http://www.dpdcompetition.com/ , último acceso: agosto de 2020.

[20] https://spectrum.ieee.org/

[21] Anaconda, https://www.anaconda.com/, último acceso: agosto de 2020.

[22] Numpy, https://www.numpy.org/, último acceso: agosto de 2020.

[23] Matplotlib, https://matplotlib.org/, último acceso: agosto de 2020

[24] OS, https://docs.python.org/3/library/os.html, último acceso: agosto 2020.

[25] M. Rawat, K. Rawat, F. M. Ghannouchi, "Adaptive Digital Predistortion of Wireless Power Amplifiers/Transmitters Using Dynamic Real-Valued Focused Time-Delay Line Neural Networks," IEEE Trans. Microw. Theory Techn., vol. 58, no. 1, pp. 95-104, Jan. 2010.

[26] M. Rawat, F. M. Ghannouchi, "A Mutual Distortion and Impairment Compensator for Wideband Direct-Conversion Transmitters Using Neural Networks," IEEE Trans. Broadcast., vol. 58, no. 2, pp. 168-177, Jun. 2012.

[27] X. You, C. Zhang, X. Tan, S. Jin, H. Wu, "Ai for 5G: Research directions and paradigms," Sci. China Inf. Sci., vol. 62, no. 2, p. 21301, Feb. 2019.

[28] N. Kato, Z. M. Fadlullah, B. Mao, F. Tang, O. Akashi, T. Inoue, K. Mizutani, "The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective," IEEE Wireless Commun., vol. 24, no. 3, pp. 146-153, 2016.

[29] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi, "A survey of deep neural network architectures and their applications," Neurocomputing, vol. 234, pp. 11-26, Apr. 2017.

[30] S. Wang, M. Roger, C. Lelandais-Perrault, "Impacts of crest factor reduction and digital predistortion on linearity and power efciency of power ampliers," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 66, no. 3, pp. 407-411, Mar. 2019.

[31] J. Cai, C. Yu, L. Sun, S. Chen, J. B. King, "Dynamic behavioral modeling of RF power amplier based on time-delay support vector regression," IEEE Trans. Microw. Theory Techn., vol. 67, no. 2, pp. 533- 543, Feb. 2019.