Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Modelado y linealización de amplificadores de potencia mediante redes neuronales

Autor: María Vázquez de Górgolas

Tutor: Maria José Madero Ayora y Elías Marqués Valderrama

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

Sevilla, 2024

Trabajo Fin de Máster
Ingeniería de Telecomunicación

Modelado y linealización de amplificadores de potencia mediante redes neuronales

Autor:
María Vázquez de Górgolas

Tutor:
María José Madero Ayora
Elías Marqués Valderrama

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Máster: Modelado y linealización de amplificadores de potencia mediante redes neuronales

Autor: María Vázquez de Górgolas

Tutor: María José Madero Ayora

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, gracias a mis tutores María José Madero y Elías Marqués, por su dedicación, por estar disponibles para la resolución de dudas y por la gran labor que han hecho en este trabajo.

Gracias a mamá, por tener un escáner en los ojos y saber que me pasaba algo cuando ni yo misma lo sabía; a papá, por sus consejos y por confiar siempre en mí; y a Sonia, por ser el espejo en el que fijarme cada día (y no me refiero solo a la ropa).

Gracias a Enrique, Mariana y Manuel, por haberme aguantado en mis momentos más pesimistas y nunca permitirme abandonar. La clave está en rodearte de gente que te sume. También a los cariñosos, por hacer del máster algo divertido, por las noches en la sureña y las tardes en la biblioteca. Juntos es más fácil.

Gracias a mis amigos desde adolescente, 'sabéis quiénes sois', como dice la canción de El Kanka. Crecer con vosotros ha sido un regalo.

Y, por último, como cuando echar de menos duele, recordar es un acto de amor, no puedo olvidarme de mis abuelos Lala, Antonio y Tere, y tampoco de Jose, que no era familia, pero casi. Sé que estáis orgullosos de mí.

María Vázquez de Górgolas

Sevilla, 2024

Resumen

Este trabajo trata de linealizar el comportamiento de un amplificador de potencia mediante predistorsión digital utilizando una red neuronal del tipo ANN, concretamente denominada ETDNN. Para ello, se utilizará una señal de entrada al predistorsionador 5G-NR con un ancho de banda de 30 MHz. También se ha utilizado el algoritmo ILC (Control de Aprendizaje Iterativo) para poder entrenar la red neuronal, con los datos de entrada y salida de este algoritmo.

Abstract

This work tries to linearize the behavior of a power amplifier through digital predistortion using a neural network of the ANN type, specifically called ETDNN. To do this, an input signal will be used to the 5G-NR predistorter with a bandwidth of 30 MHz. The ILC (Iterative Learning Control) algorithm has also been used to find the optimal output of the neural network.

Índice

Índice abreviado

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El ser humano es considerado un ser social, lo que provoca que las comunicaciones tengan un papel crucial en nuestra vida. Esto implica considerar el nacimiento de las comunicaciones móviles como uno de los mayores avances que se han dado en nuestra historia.

Estas comunicaciones, que nos permiten comunicarnos aún a distancia, han pasado (por ahora) por 5 generaciones, y vamos encauzados hacia la sexta. Este camino va desde la primera generación (1G), con telefonía analógica, donde solo se podían realizar llamadas, hasta la quinta generación (5G), donde se está llegando a una comunicación máquina-máquina y un espacio de conectividad total donde todos nuestros dispositivos están conectados entre sí. Las señales que se utilizan en esta última generación son OFDM (multiplexación por division en frecuencia orthogonal, del inglés Orthogonal Frequency Division Multiplexing), caracterizadas por tener un PAPR muy elevado.

La posibilidad de este avance cada vez más exponencial se da gracias a la investigación para la eficiencia de estas comunicaciones. Uno de los problemas más estudiados en esta investigación es la distorsión de los amplificadores de potencia (PA de aquí en adelante), producida porque es más eficiente energéticamente trabajar cerca de la zona de saturación de la curva del amplificador. Además, el hecho de trabajar con señales con un PAPR elevado facilita que entremos en esta zona de saturación, lo que implica grandes desventajas, como productos de intermodulación o calentamiento excesivo.

Una de las soluciones más extendidas para conseguir frenar estos comportamientos es la predistorsión digital (DPD): un bloque no lineal justo antes del PA cuya curva de potencia es inversa a la del PA, buscando que el resultado de los dos elementos en cascada tenga un comportamiento lineal con la ganancia deseada. La incorporación de este bloque requiere modelar el comportamiento del amplificador, y para ello existen técnicas como las series de Volterra.

En este caso, se diseñará un predistorsionador usando redes neuronales, concretamente una red denominada Envelope Time-Delay Neural Network (ETDNN) [1] [2]. Para diseñar el predistorsionador óptimo con el que poder entrenar la red, se empleará el algoritmo Iterative Learning Control (ILC) [3]. Una vez obtengamos buenos resultados con este algoritmo, entrenaremos la red con la señal original (entrada del bloque DPD) y la señal de entrada al PA óptima (salida del bloque DPD proporcionada por ILC). Este paso intermedio es necesario, ya que se precisa conocer la señal de salida óptima del bloque DPD para que la red pueda predecir su comportamiento correctamente.

Para comprender si la red ha predicho bien y, por tanto, está teniendo un comportamiento adecuado, recurriremos a figuras de mérito como el error cuadrático medio normalizado (NMSE), y a comparaciones entre la salida del algoritmo ILC y la salida de la red neuronal. Estas comparaciones podrán ser tanto visuales como numéricas, a fin de poder juzgar la validez de la propuesta presentada.

De esta manera, el capítulo 2 de este trabajo se centra en técnicas para el modelado del amplificador y en figuras de mérito para medir la linealidad de estos, además de hablar de la técnica del algoritmo ILC como solución para estos problemas de linealidad. El capítulo 3 resume conceptos clave de las redes neuronales y explica detalladamente en qué consiste la red utilizada en este caso, la ETDNN. El capítulo 4 explica cómo se ha llevado a cabo la implementación de este trabajo y comenta los resultados obtenidos. Por último, en el capítulo 5 se presentan las conclusiones obtenidas y posibles líneas de mejora.

2 ALGORITMOS Y CRITERIOS PARA EL MODELADO Y LA LINEALIZACIÓN DE AMPLIFICADORES DE POTENCIA

Uno de los mayores retos al trabajar con amplificadores de potencia es intentar predecir su comportamiento mediante un modelado del sistema. Este reto se debe a que los PA tienen un comportamiento no lineal, es decir, la potencia de salida del amplificador no es durante todo el rango de trabajo de valores de potencia a la entrada $P_{out} = g \cdot P_{in}$, siendo g la ganancia del PA. Esto se debe principalmente a dos motivos: el aumento de la potencia de entrada, que implica trabajar cerca del punto de saturación del amplificador, y su dependencia con la temperatura, ya que el dispositivo tiende a calentarse cuando su punto de polarización es elevado. Este es el motivo por el cual aparecen nuevos términos retrasados y no lineales en la salida del PA, convirtiendo así el dispositivo en un sistema no lineal y con memoria.

Una vez modelado el amplificador, el siguiente desafío es intentar linealizar su comportamiento. Tanto para el modelado como para la linealización, a lo largo de los años se han ido desarrollando distintos modelos y arquitecturas de predistorsión que intentan darles respuesta a estos problemas, así como figuras de mérito que nos permiten evaluar el sistema. En este capítulo se mostrarán algunos de ellos.

2.1. Sistemas no lineales con memoria

Al contrario que un sistema lineal, un sistema no lineal se caracteriza por tener más de una componente en la señal de salida, que no será simplemente una muestra retardada y aumentada (por el factor ganacia) de la señal de entrada.

Un sistema no lineal sin memoria cumple la siguiente ecuación:

$$x(t) \rightarrow \mathrm{y(t)} = a_1 x(t-\tau) + a_2 x(t-\tau)^2 + \cdots + a_n x(t-\tau)^n \quad (2.1),$$

siendo n el orden no lineal del sistema y τ el retardo del mismo. En este caso, el primer término $a_1 x(t-\tau)$ sería el componente lineal y deseable, mientras que el resto de componentes introducen distorsiones adicionales a nuestra señal.

Sin embargo, los PA son sistemas lineales con memoria, lo que implica que la salida no depende de un retardo fijo, sino del instante actual y otros instantes previos. Así, la salida de un sistema no lineasl con memoria cumpliría la siguiente ecuación:

$$x(t) \rightarrow \mathrm{y(t)} = f_{\mathrm{NL}}[x(t), x(t-\tau_1), x(t-\tau_2), \ldots, x(t-\tau_n)] \quad (2.2),$$

donde $[\tau_1, \tau_n]$ son los retrasos asociados a cada instante de la señal y f_{NL} es una función no lineal, por ejemplo, la serie de potencia de (2.1). La no linealidad en estos dispositivos puede conllevar varios problemas, como por ejemplo la intermodulación, que introduce nuevas frecuencias no deseables en la señal de salida. En los casos en los que $x(t)$ tuviese una magnitud pequeña podrían despreciarse el resto de términos salvo el primero, y obtendríamos un sistema lineal, pero esta idea no es aplicable cuando se pretenden conseguir potencias de salida elevadas.

2.2. Señal OFDM

Las señales que se utilizan en 5G-New Radio (5G-NR), es decir, en la capa física de la quinta generación de comunicaciones móviles, son de tipo OFDM con prefijo cíclico (CP).

OFDM [4] basa su funcionamiento en la división del canal de transmisión en múltiples subportadoras que son ortogonales entre sí en el dominio de la frecuencia, evitando de esta foirma interferencias entre las subportadoras y permitiendo una transmisión de datos más eficiente. Este concepto se muestra en la Figura 2.1, donde observamos que el máximo de una subportadora se produce en la misma frecuencia que los ceros de sus adyacentes.

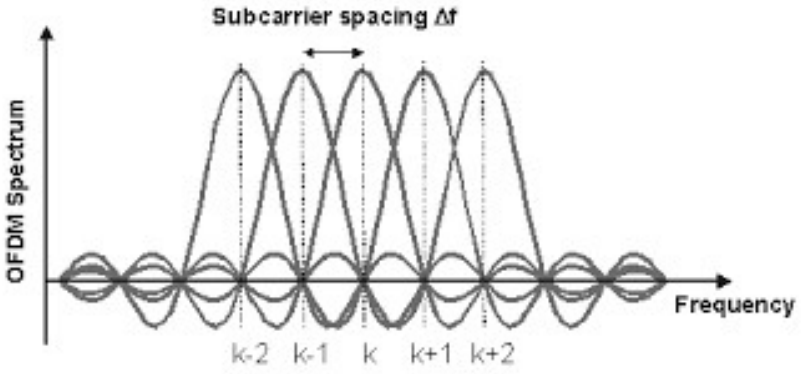


Figura 2.1. Señal OFDM

Por otro lado, el CP implica añadir una copia del final de un símbolo OFDM al principio de este, eliminando así la interferenia entre portadoras (ICI).

Además, en 5G-NR la duración de los símbolos OFDM no es fija, sino que puede tener distintos valores en función de la numerología $\mu \in [0,5]$, teniendo así mayor flexibilidad:

$$\Delta f = 2^{\mu} \cdot 15\ kHz \quad (2.3).$$

Sin embargo, trabajar con este tipo de señales, aunque tiene grandes ventajas, presenta un problema, debido a que la diferencia entre el máximo y el promedio de la señal (relación potencia pico a potencia promedio, PAPR, definida en (2.4) es muy grande.

$$PAPR(dB) = 20\log\left(\frac{máx[x[k]]}{\sqrt{\frac{1}{N}\sum_{n=1}^{N}|x[k]|^2}}\right) \quad (2.4)$$

Esto implicará que el amplificador tenga que estar constantemente trabajando cerca de la zona de saturación, por tener la señal potencias altas. Esto producirá distorsión, como se ha explicado en el capítulo introductorio de este trabajo, y es aquí donde nace la necesidad de un predistorsionador que mitigue estos efectos nocivos.

2.3. Modelado del PA mediante series de Volterra

Las series de Volterra son la aproximación más cercana hasta la fecha para modelar el verdadero comportamiento de un PA. Éstas tienen su punto de partida en las aproximaciones de Taylor, que, como se muestra en la ecuación (2.5), descomponen la señal de salida en una suma de polinomios de la señal de entrada con un retraso fijo:

$$y(t) = f[x(t)] = a_0 + a_1 x(t - \tau) + \cdots + a_N x^N (t - \tau) \quad (2.5),$$

donde N es el orden máximo y τ el retraso.

El problema es que las aproximaciones de Taylor no presentan memoria, es decir, la señal $y(t)$ en un instante τ_1 solo depende de la señal de entrada en ese mismo instante τ_1 . Sin embargo, numerosos estudios demuestran que este no es el caso de los PA, cuya señal $y(t)$ en un instante dependerá de la señal de entrada en ese instante

y en instantes pasados. Así, nacen las series de Volterra, que se definen como:

$$y(t) = \sum_{p=1}^{\infty} y_p(t) \qquad (2.6),$$

teniendo $y_p(t)$ la siguiente forma:

$$y_p(t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\dots\int_{-\infty}^{\infty} h_p(\tau_1, \tau_2, \dots, \tau_n)\cdot\prod_{s=1}^{n} x(t-\tau_s)\cdot\delta\tau_s \quad (2.7).$$

Aquí aparece el término regresor, que son las combinaciones entre las versiones retrasadas de $x(t)$ y $h_p(\tau_1,\tau_2,\dots,\tau_n)$ es el vector de coeficientes o kernel.

A la hora de manejar las series de Volterra computacionalmente, es necesario que las variables sean discretas y que la serie esté truncada, ya que la memoria de un ordenador no es infinita. Así, es imprescindible reescribir la serie de Volterra como:

$$y[k] = \sum_{n=1}^{N}\sum_{q_1=0}^{Q}\dots\sum_{q_n=0}^{Q} h_n(q_1,\dots,q_n)\times\prod_{s=1}^{n} x(k-q_s) \quad (2.8),$$

donde N es el orden máximo del sistema y Q la profundidad de la memoria, es decir, la cantidad de retrasos de la señal que se tienen en cuenta. Este modelo ya es válido para trabajar de forma práctica con él, sin embargo, solo sirve para muestras de señales y[k] y x[k] reales, es por ello por lo que se denomina serie de Volterra en RF (radiofrecuencia). En este campo de trabajo, sin embargo, es preferible trabajar con muestras de la envolvente compleja de la señal en banda base. Así, la ecuación (2.8) queda reescrita como [5]:

$$\tilde{y}[k] = \sum_{n\ impar}^{n}\sum_{q_1=0}^{Q}\dots\sum_{q_{2p+1}=0}^{Q_{2p+1}} h_{2p+1}(q_1,\dots,q_{2p+1})\times\prod_{s=1}^{p+1}\tilde{x}\,[k-q_s]\times\prod_{r=p+2}^{2p+1}\tilde{x}^*[k-q_r], \qquad (2.9),$$

siendo 2P+1 el orden no lineal máximo y Q_{2p+1} la profundidad de memoria, y $\tilde{x}$ e $\tilde{y}$ representan la envolvente compleja en banda base. Por simplicidad de escritura, a partir de aquí, la entrada $x[k]$ y la salida $y[k]$ representarán en todo momento la envolvente compleja en banda base. Este modelo se denomina Full Volterra (FV) por incorporar todos los coeficientes, siendo unicamente una señal discreta y truncada.

No obstante, en la práctica es común trabajar con simplificaciones de este modelo FV, ya que éste es muy complejo computacionalmente por la cantidad de regresores a tener en cuenta. A continuación, se presentan algunas de estas simplificaciones que han sido ampliamente utilizadas en la literatura.

2.3.1. Memory Polinomial (MP)

Este modelo es el más sencillo, donde a señal de salida y_{MP} [k] se expresa como una suma de productos de las versiones retrasadas de la entrada x con sus respectivas magnitudes elevadas a diferentes potencias. La diferencia con el modelo FV es que en este caso todas las entradas tendrán un retraso común q, obviando así el vector de memorias del modelo FV. La expresión del modelo MP quedaría como [6]:

$$y_{MP}[k] = \sum_{p=1}^{P}\sum_{q=0}^{Q} h_p[q]x[k-q]x[k-q]^{p-1} \qquad (2.10)$$

2.3.2. Generalized Memory Polinomial (GMP)

Este modelo es una ampliación del MP que tiene en cuenta términos de memoria cruzada. Su expresión es [7]:

$$\begin{aligned} y_{GMP}[k] = &\sum_{p=1}^{P_a}\sum_{q=0}^{Q_a} h_a[q]x[k-q]|x[k-q]|^{2p} \quad + \\ &+\sum_{p=1}^{P_b}\sum_{q=0}^{Q_b}\sum_{m=0}^{M_b} h_b[q,m]x[k-q]|x[k-q-m]|^{2p} + \\ &+\sum_{p=1}^{P_c}\sum_{q=0}^{Q_c}\sum_{m=0}^{M_c} h_c[q,m]x[k-q]|x[k-q+m]|^{2p} \qquad (2.11). \end{aligned}$$

Este modelo se divide en tres partes, donde la primera (parte A) es igual que el modelo MP, la segunda (parte B) son combinaciones de la envolvente retrasada y la tercera (parte C) son combinaciones de la envolvente adelantada, ambas *m* muestras con respecto a la señal.

2.4. Expresión en forma matricial

El hecho de expresar los modelos anteriores en forma matricial implica menos carga computacional, y además ayuda a no perder las propiedades algebraicas de las ecuaciones del modelo. Si nos fijamos en la ecuación (2.11), observamos que se trata de un sumatorio cuya serie es simplemente la multiplicación de un vector por una matriz. Así, se definen el vector de coeficientes y la matriz de regresores en las ecuaciones (2.12) y (2.13), respectivamente:

$$\boldsymbol{h}_p = \left[h_1[0], \dots, h_1[Q], h_p[0], \dots, h_p[Q], \dots h_1[0,0], h_1[0,Q], h_P[0,0], \dots h_P[Q,M]\right] \quad (2.12)$$

$$\mathbf{X} = \begin{bmatrix} x[0] & x[-1] & \dots & x[-Q] & x[0]|x[0]|^2 & \dots & x[-Q]|x[-Q]|^2 & \dots \\ x[1] & x[0] & \dots & \vdots & x[1]|x[1]|^2 & \dots & \vdots & \dots \\ x[2] & x[1] & & x[-1] & x[2]|x[2]|^2 & \dots & x[-1]|x[-1]|^2 & \dots \\ \dots & \dots & & & \dots & & \vdots & \\ \vdots & \vdots & \dots & & \vdots & \dots & \vdots & \dots \\ x[N] & x[N-1] & & & x[N]|x[N]|^2 & & x[N-Q]|x[N-Q]|^2 & \end{bmatrix} \quad (2.13),$$

donde N corresponde al número de muestras, y los índices negativos a las muestras de la señal tomadas antes del instante de inicio, representando así los efectos de la memoria

De esta forma, cualquiera de los modelos anteriores podría ahora definirse matricialmente como:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{h}_{\boldsymbol{p}} \quad (2.14),$$

con las ventajas que ello implica, expresadas al inicio de este mismo apartado.

2.5. Predistorsión digital

La predistorsión digital es una técnica para mejorar el funcionamiento de los amplificadores de potencia, contrarrestando el efecto no lineal con memoria del PA para que, en conjunto, resulte en un sistema lineal.

La idea de esta técnica es aplicar una señal "pre-distorsionada" antes de que pase por el amplificador. Esta señal modificada está diseñada para contrarrestar la distorsión que el PA va a introducir. Para hacer esto, primero se crea un modelo matemático del comportamiento del amplificador. Este modelo predice cómo se distorsionará la señal y permite ajustar la señal de entrada de manera que la salida final sea lo más cercana posible a la original con la ganancia deseada.

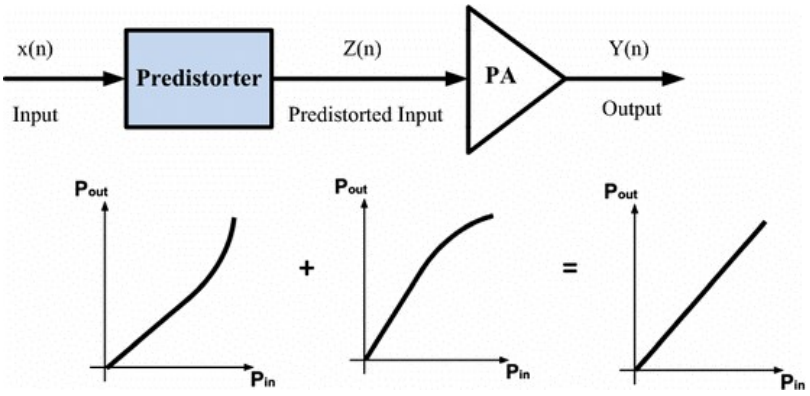


Figura 2.2. Esquema del predistorsionador digital.

El bloque de predistorsión digital tiene dos componentes clave, que se muestran en la Figura 2.2. El primero es el predistorsionador digital, que utiliza algoritmos para ajustar la señal de entrada. Luego está el PA, que es el que amplifica la señal "pre-distorsionada" y cuya salida, si la señal ha estado bien predistorsionada, debería parecerse lo máximo posible a la de un sistema lineal.

De las distintas formas que existen de llevar a cabo la predistorsión, este trabajo se centrará en la arquitectura de aprendizaje directo (DLA), explicada a continuación y cuyo esquemático se presenta en la Figura 2.3. La arquitectura DLA basa su funcionamiento en encontrar iterativamente el vector error e, que muestra el efecto negativo que está teniendo el amplificador sobre la señal de entrada, y que se calcula teniendo en cuenta que, si el PA tuviese un comportamiento totalmente lineal, la señal de salida sería la señal de entrada multiplicada por la ganancia del amplificador. En este caso se emplea la misma matriz de regresores que para el modelado, pero lo que se pretende estimar ahora es directamente la señal de distorsión $d = Uw$, donde ahora w hace en la etapa de adaptación de vector de coeficientes que se va calculando iterativamente con el factor de aprendizaje $\mu \in (0,1]$.

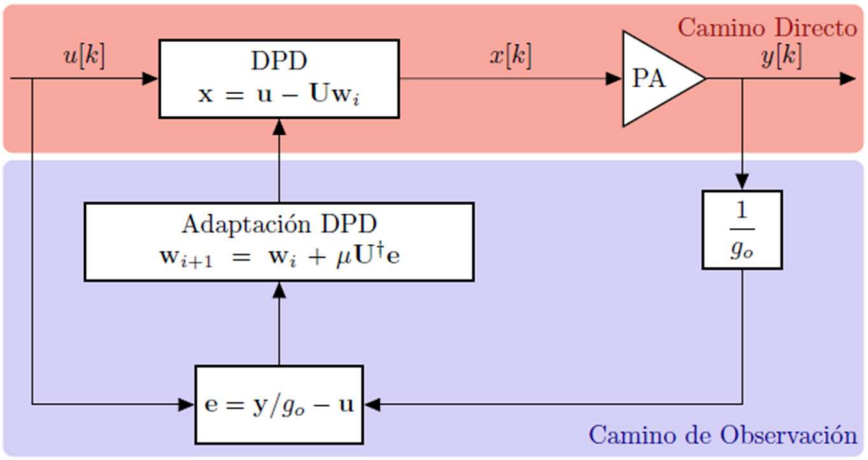


Figura 2.3. Esquema de la arquitectura DLA

2.6. Algoritmo ILC

El algoritmo ILC [3] es un algoritmo de aprendizaje automático de k iteraciones y que, en este contexto, puede servir para realizar predistorsión. Este algoritmo calcula, para cada iteración k, la señal error tal como se muestra en la fórmula (2.15), donde $y_k(n)$ es la señal de salida del amplificador para cada iteración k, y $y_d(n)$ es la señal de salida deseada, en nuestro caso, la señal linealizada a la salida del PA. Este error $e_k(n)$ se utiliza para calcular la nueva señal de entrada al amplificador $u(k+1)$, que se utilizará en la siguiente iteración.

$$e_k(n) = y_d(n) - y_k(n) \qquad (2.15)$$

Se trata pues, de ir modificando la señal de entrada del amplificador en función del error calculado, según la siguiente fórmula:

$$u_{k+1} = u_k + \tau e_k \qquad (2.16)$$

Donde τ denota la matriz de aprendizaje que controla la velocidad de convergencia del algoritmo, $u_k = [u_k(0), u_k(1), \ldots, u_k(N-1)]^T$ y $e_k = [e_k(0), e_k(1), \ldots, e_k(N-1)]^T$

Esto se muestra también en la Figura 2.4, donde se esquematiza lo explicado anteriormente.

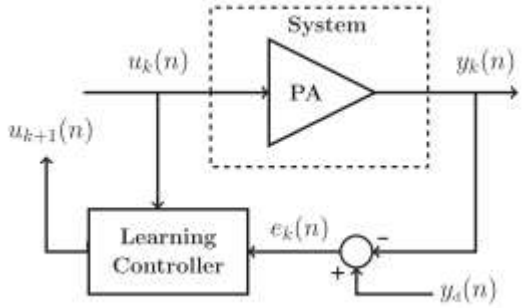


Figura 2.4. Esquema del algoritmo ILC.

La principal dificultad del diseño es encontrar el algoritmo que haga que $u_k(n)$ converja a un valor fijo y óptimo $u^*(n)$ que minimice el error $e_k(n)$. Para ello, hay que prestar especial atención en el diseño de τ, que puede ser de distintos tipos. En este caso, se explicará el utilizado para la toma de datos de este trabajo, que es el denominado "Linear"

En este algoritmo, la señal de entrada del amplificador en la iteración k+1 se calcula como:

$$u_{k+1}(n) = u_k + \gamma e_k \qquad (2.17)$$

Donde γ es denotado por la ganancia de aprendizaje y γI sería la matriz de aprendizaje, con I la matriz identidad. Tal y como se puede observar, en este caso la señal de entrada del amplificador para cada iteración se actualizará usando una ganancia constante. Este algoritmo también se conoce como tipo lineal de primer orden.

Las ventajas que presenta este tipo de algoritmo ILC son que es menos denso computacionalmente, y que la información que requiere acerca de las características del PA es mínima. Es decir, de este algoritmo destaca provechosamente su simpleza

La ejecución del algoritmo ILC se puede resumir en los siguientes pasos clave:

1. Calcular la salida $y_d(n)$ del amplificador que haga que éste tenga un comportamiento lineal.
2. Para $k = 1$, calcular $u_1 = y_d/g_{avg}$, donde g_{avg} es la ganancia media del amplificador para la potencia de salida promedio deseada.
3. Aplicar la entrada $u_k(n)$ al PA y medir la salida y_k
4. Calcular el error $e_k(n)$ según la fórmula (2.15)
5. Analizar si el error cumple con los requisitos. Si la respuesta es sí, detener el proceso; de lo contrario, continuar con el siguiente paso.
6. Calcular la siguiente entrada u_{k+1} según lo establecido en la fórmula (2.17) para el caso de "Linear".
7. Incrementar la iteración $k = k + 1$ y volver al paso 3.

Al ejecutar estos pasos durante varias iteraciones, encontraremos la señal de entrada óptima $u^*(n)$ que haga que la salida del amplificador sea muy parecida a $y_d(n)$.

Una vez explicado el funcionamiento del algoritmo, y para darle contexto en este trabajo, el algoritmo ILC es una técnica que puede ser utilizada para identificar los parámetros de un predistorsionador digital. En concreto, existe la técnica de predistorsión digital basada en el control iterativo de aprendizaje (ILC-DPD), mostrada en el esquema 2.5.

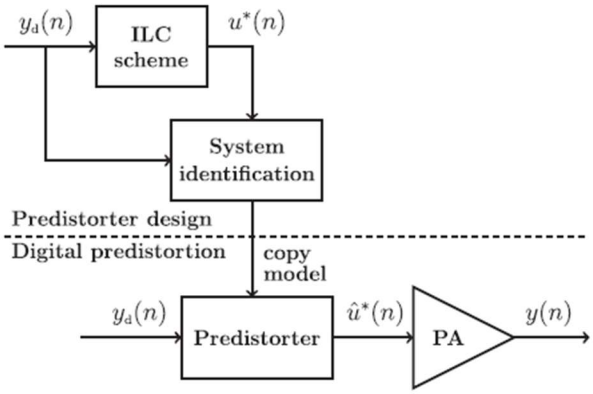


Figura 2.5. Esquema de la técnica ILC-DPD

Este esquema utiliza primero el ILC para encontrar la señal óptima de entrada del PA $u^*(n)$ que genere la respuesta deseada $y_d(n)$. Una vez identificada esta señal, se emplea para modelar el predistorsionador, utilizando la señal deseada $y_d(n)$, que coincide con la señal original salvo en un factor de escala, como entrada y la señal óptima $u^*(n)$ como salida del modelo. La principal ventaja de este método es que permite probar diferentes estructuras de modelos y configuraciones sin necesidad de iterar sobre el PA, lo que resulta en un proceso más eficiente y robusto para diseñar predistorsionadores digitales.

En el artículo de investigación utilizado para este trabajo, se opta por el modelo GMP, explicado en este trabajo en el apartado 2.3.2.

2.7. Figuras de mérito

Como se ha dicho anteriormente, las figuras de mérito nos ayudan a medir cómo de cerca estamos de que la señal de salida de un amplificador sea lineal. Concretamente, estos indicadores de rendimiento calculan la diferencia entre la señal de salida deseada y la real. Cada figura de mérito tiene unos rangos en los cuales se considera que el comportamiento del amplificador es el deseado. Por tanto, es una forma evaluar si la predistorsión realizada es buena o no.

A continuación, se explican algunas de estas figuras, como son la relación de potencia en el canal adyacente (ACPR), la magnitud del vector error (EVM) y el error cuadrático medio normalizado (NMSE).

- **ACPR**

El valor de ACPR indica la cantidad de potencia que el PA emite fuera del ancho de banda de la señal de entrada.

Esto lo convierte en un indicador de la distorsión que ocurre fuera de la banda, siendo así una figura de mérito fundamental, ya que es la forma de controlar que no se está emitiendo demasiada potencia fuera de la banda trabajo. De esta forma, midiendo el ACPR, nos estamos asegurando de no interferir en otros sistemas que trabajen a frecuencias cercanas. Además, en 5G-NR hay impuestas unas máscaras de emisión fuera de banda que regulan la máxima potencia que se puede transmitir fuera de banda por normativa. Esto convierte, dentro del contexto del 5G, al ACPR en una medida primordial para poder transmitir cumpliendo con la regulación.

En otras palabras, este indicador compara la potencia generada en los canales adyacente con la potencia generada en el canal principal de la señal y viene dado tanto para el canal adyacente inferior como superior, de la siguiente forma:

$$ACPR_{-1}(dB) = 10log_{10}\frac{\int_{f_{L1}}^{f_{L2}} S_y(f)df}{\int_{f_{c1}}^{f_{c2}} S_y(f)df} \qquad (2.18)$$

$$ACPR_{+1}(dB) = 10log_{10}\frac{\int_{f_{U1}}^{f_{U2}} S_y(f)df}{\int_{f_{c1}}^{f_{c2}} S_y(f)df} \qquad (2.19),$$

donde Sy(f) es la densidad espectral de potencia de la señal a la salida del amplificador; $[f_{L1}, f_{L2}]$ y $[f_{U1}, f_{U2}]$

determinan el ancho de banda de los canales adyacentes inferior y superior, respectivamente, y [f_{C1}, f_{C2}] es el ancho de banda de canalización.

- **EVM**

Esta figura está bastante relacionada con la constelación de la señal, ya que compara los símbolos de la señal demodulada con los transmitidos en su constelación. Por ello, es deseable que esta medida sea lo más cercana a 0 posible, ya que indicará que los simbolos transmitidos y recibidos son similares. Al contrario que en el caso anterior, se trata de una medida de distorión en banda. Se caracteriza de la siguiente forma:

$$EVM[\%] = \sqrt{\frac{\sum_{i=1}^{N}|\hat{x}_i - x_i|^2}{\sum_{i=1}^{N}|x_i|^2}} * 100 \quad (2.20),$$

donde x_i y $\hat{x}_i$ representan los símbolos transmitidos y recibidos, respectivamente, y N es el número de símbolos transmitidos

- **NMSE**

Con esta medida se pretende calcular la diferencia entre la señal medida que se obtiene a la salida de un sistema y la señal modelada, es decir, la estimación de la señal.

El NMSE se diferencia de las otras métricas vistas en que no se enfoca únicamente en la distorsión fuera de banda o la distorsión en banda, si no que busca proporcionar una evaluación global de la calidad de la señal. Es, por tanto, una medida integral que tiene en cuenta todos los tipos de distorsión, ruido o error de modelado que pueden afectar a la señal. Esto proporciona una visión completa del rendimiento del sistema, abarcando todos los posibles problemas que puedan influir en la calidad de la señal.

El cálculo de esta figura de mérito, para N muestras, es:

$$NMSE\ (dB) = 20log\frac{\sum_{k=1}^{N}|y_{mod}(k) - y_{med}(k)|}{\sum_{k=1}^{N}|y_{med}(k)|} \quad (2.21),$$

donde $y_{mod(k)}$ y $y_{med}(k)$ son la señal de salida modelada y medida, respectivamente. Esta medida será la más utilizada en este trabajo. Un valor bajo de NMSE (idealmente cercano a -40 dB) nos indicará que la señal modelada y la real son bastante parecidas, lo que sugiere que el modelo es eficiente y se ha entrenado correctamente. Por el contrario, un valor alto de NMSE indica que la diferencia entre las señales real y modelada es significativamente grande, y que el modelo no es válido.

3 APLICACIÓN DE REDES NEURONALES AL PROBLEMA DE LINEALIDAD

Este capítulo se centra en explicar una implementación de red neuronal, que, al entrenarla, propocione una señal de entrada al PA parecida a la resultante de aplicar el algoritmo ILC, explicado en el capítulo anterior. En este caso, siguiendo el artículo de investigación *Efficient Digital Predistortion Using Sparse Neural Network* de Masaaki Tanio [1]. En concreto, se propone una ANN denominada Envelope Time-Delay Neural Network (ETDNN) que actúe como predistorsionador, y que se entrenará con los datos resultantes de aplicar el algoritmo ILC.

Primeramente, este capítulo, y con la idea de poner en contexto sobre redes neuronales, constará de una introducción sobre éstas, hasta entrar progresivamente en la red utilizada en este trabajo.

3.1 Redes neuronales

Las redes neuronales son modelos computacionales inspirados en el funcionamiento del cerebro humano, diseñados para reconocer patrones y realizar tareas de procesamiento de datos. A nivel técnico, consisten en una serie de unidades interconectadas llamadas neuronas, organizadas en capas. Cada neurona recibe señales de múltiples neuronas de la capa anterior, las procesa mediante una función de activación y transmite el resultado a las neuronas de la capa siguiente. El aprendizaje en las redes neuronales se realiza mediante un proceso de ajuste iterativo de los pesos de las conexiones entre neuronas, utilizando algoritmos de optimización. Este ajuste permite que la red minimice el error en la predicción o clasificación de datos, mejorando su precisión y eficiencia.

A continuación, se explicará con más detalle algunos de estos conceptos, como las neuronas y el proceso de aprendizaje.

3.1.1 Estructura de una red neuronal

3.1.1.1 Neuronas

Las redes neuronales están formadas, como su nombre deja entrever, por neuronas, que se consideran la unidad básica de procesamiento.

Las partes de una neurona se pueden observar en la Figura 3.1.

En primer lugar, tenemos las entradas x_i, que son los datos o, en este caso, señales que la neurona recibe desde otras neuronas o desde el entorno exterior.

Cada entrada está asociada a un peso w_i, que son parámetros que determinan la relevancia de cada entrada x_i en el cálculo de la señal de salida y. Estos pesos se van ajustando durante el aprendizaje, es decir, que no tienen un valor fijo e inamovible desde el principio.

Después, la neurona realiza una suma ponderada de cada entrada multiplicada por su respectivo peso, añadiendo el parámetro aditivo sesgo o bias, que permite a la neurona ajustar su umbral de activación, es decir, saber cuándo activarse. Este sesgo funciona como un umbral que ayuda a desplazar la función de activación. Esta suma ponerada se muestra en (3.1), donde b es el sesgo.

$$Z = \sum_{i=1}^{n} x_i w_i + b \qquad (3.1)$$

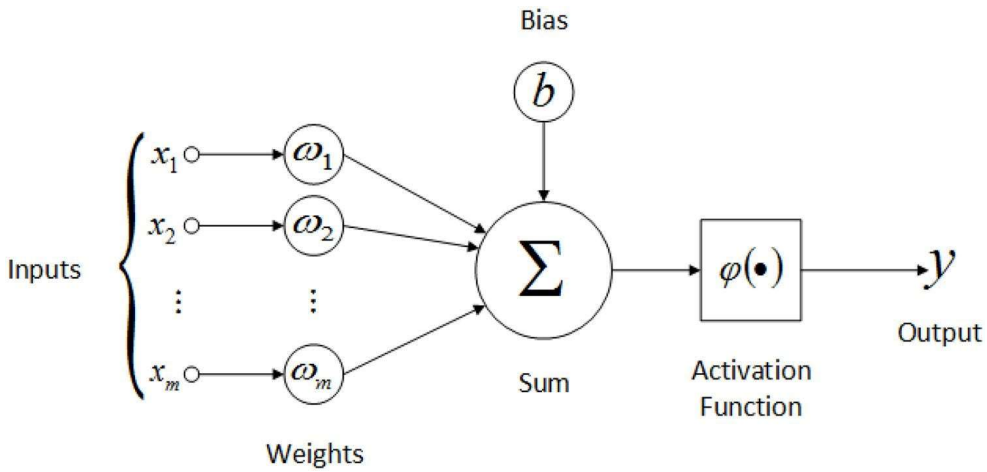


Figura 3.1. Concepto de neurona

Por último, esta suma ponderada pasará por la función de activación, cuyo objetivo será introducir no linealidad en el modelo, ya que podemos observar que Z es una función lineal. Esta última parte de la neurona permite que la red pueda aprender relaciones no lineales. Algunas funciones de activación comunes son sigmoide (sigmoid), tangente hiperbólica (tanh) o rectificador lineal (ReLU).

El resultado de aplicar la función de activación a la suma ponderada será la salida de la neurona "y". Esta salida se puede utilizar como entrada para otras neuronas en capas posteriores de la red o como la predicción final, si nos encontramos en la capa de salida.

Este proceso se repite en cada neurona de cada capa de la red.

3.1.1.2 Capas

Una capa en una red neuronal representa un conjunto de neuronas que procesan simultáneamente unas entradas y producen unas salidas. Las capas son los bloques estructurales fundamentales de una red neuronal, y cada una cumple una función específica en el procesamiento de la información, es decir, en cada una de las capas se aprenderán diferentes características de nuestros datos.

El número de neuronas y la cantidad de capas son hiperparámetros que elegimos cuando decidimos implementar una red. Cada capa puede utilizar un número de neuronas distintas.

3.1.1.3 Arquitectura de una red neuronal

La arquitectura típica de una red neuronal, mostrada en la Figura 3.2, define cómo se conectan las neuronas entre sí, cuántas capas hay, y cómo fluye la información a través de la red.

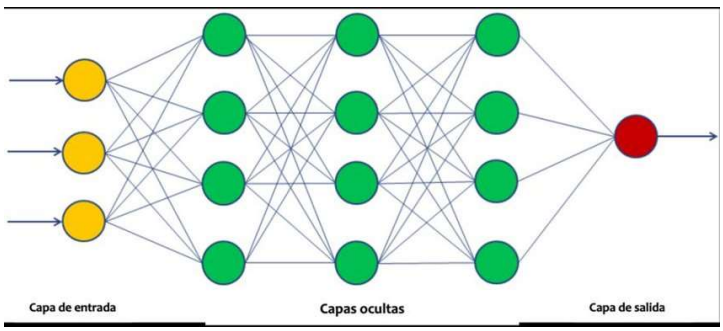


Figura 3.2. Arquitectura de una red neuronal

En primer lugar, la capa de entrada es la encargada de recibir los datos originales que se van a procesar en la red. Cada neurona en esta capa corresponde a una característica inicial del conjunto de datos.

Las salidas de esta capa de entrada pasan por las capas ocultas (hidden layers). Estas capas realizan

transformaciones intermedias de los datos.

Una red neuronal puede tener desde una hasta varias capas ocultas, en función del objetivo. Cada capa representa un nivel de abstracción y a medida que profundizamos en la red cada una de estas capas ocultas atenderá a características más complejas de nuestro conjunto de datos.

Por último, la salida de la última capa oculta pasará por la última capa de la red, la capa de salida. Esta capa es la utilizada para producir la salida final de la red, que puede ser una predicción o una clasificación.

3.1.2 Aprendizaje de una red neuronal

Como se ha explicado en el apartado 3.1.1, el aprendizaje de una red neuronal se basa en un proceso iterativo mediante el cual la red ajusta tanto los pesos como los sesgos para hacer predicciones precisas. El número de iteraciones que la red realiza se conoce como "epoch", y es un hiperparámetro que se decide al ejecutar una red. El objetivo final de este aprendizaje es minimizar el error, es decir, la diferencia entre la salida real esperada y la salida de la capa de salida de la red. Este aprendizaje tiene dos procesos diferenciados: forward propagation y backward propagation, tal como se muestra en la Figura 3.3.

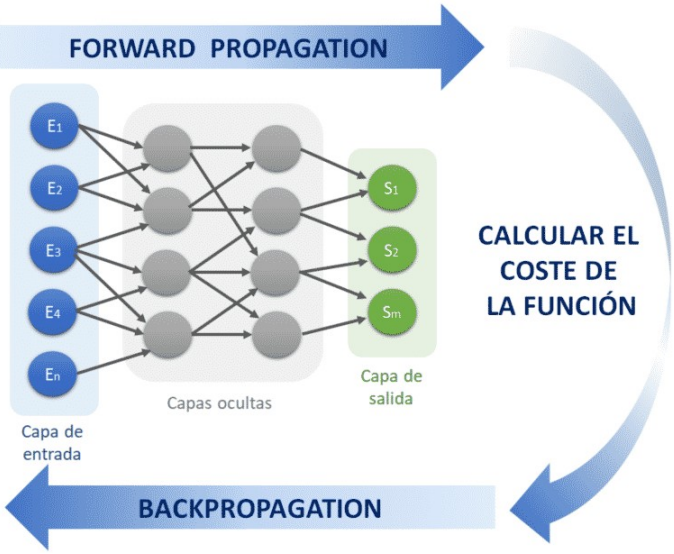


Figura 3.3. Proceso de aprendizaje de una red neuronal

3.1.2.1 Forward propagation

Este paso, que se realiza en primer lugar, implica calcular la salida de la red neuronal para un conjunto de datos de entrada dados.

Cada neurona recibe unos datos de entrada, realiza las operaciones matemáticas vistas en el apartado 3.1.1.1, y la salida obtenida de esta operación la pasa a la siguiente capa, hasta llegar a la capa de salida, donde el valor de salida de la capa será la predicción del algoritmo.

Con esta predicción se calcula el error, que se utilizará en el proceso backward propagation. El error cuantifica la diferencia entre la salida predicha por la red y la salida real. Para calcular el error, se utiliza una función de pérdidas L, como la entropía cruzada (utilizada sobre todo en clasificación), o el error cuadrático medio (utilizada para regresión).

Con la función de pérdidas podremos calcular la función de costes, que se utilizará en el ajuste de pesos en backward propagation. La función de costes se calcula en (3.2) como el promedio de todos los resultados de la función de pérdidas para m ejecuciones del proceso.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{Y}_i, Y_i\right), \qquad (3.2)$$

3.1.2.2 Backward propagation

Este proceso basa su funcionamiento en ajustar los pesos y sesgos que se utilizarán en forward propagation, para así minimizar la diferencia entre la predicción de la iteración del algoritmo y la salida real del sistema.

El algoritmo más utilizado y conocido para esta etapa es el descenso por gradiente, que se explica a continuación.

En primer lugar, se calcula el gradiente de la función de coste respecto a cada peso y sesgo, utilizando la regla de la cadena como se indica en (3.3) y (3.4). Esta derivada nos indica la dirección óptima en la que los parámetros deben actualizarse para conseguir el objetivo de minimizar el error.

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_i} \qquad (3.3)$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b} \qquad (3.4)$$

Las fórmulas para la actualización de pesos y sesgos, respectivamente, son (3.5) y (3.6), donde η es la tasa de aprendizaje y determina la magnitud de los cambios de los parámetros. Este también es un hiperparámetro que decidimos nosotros al entrenar la red.

$$w_i = w_i - \eta \cdot \frac{\partial J}{\partial w_i} \qquad (3.5)$$

$$b = b - \eta \cdot \frac{\partial J}{\partial b} \qquad (3.6)$$

El objetivo del descenso por gradiente es alcanzar el mínimo óptimo de la función que relaciona J, w y b desplazándose cada paso de aprendizaje, pasando de A (inicio del algoritmo) a B (mínimo óptimo) según la Figura 3.4.

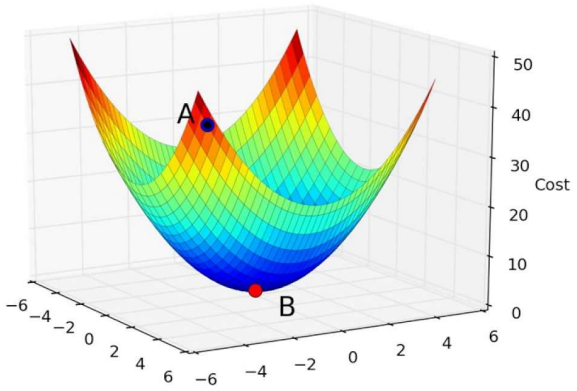


Figura 3.4. Descenso por gradiente

3.1.2.3 División de datos en el aprendizaje

Para el aprendizaje de una red neuronal, es crucial dividir nuestro conjunto de datos en tres partes: datos de entrenamiento, datos de validación y datos de prueba.

Los datos de entrenamiento forman el subconjunto más grande de los tres. Estos datos serán los utilizados por la red en todas las iteraciones para realizar el proceso de aprendizaje, es decir, los datos con los que se ajustarán los pesos y sesgos según lo anteriormente explicado.

Los datos de validación validan periódicamente el modelo y proporcionan una medida de cómo predice el modelo con datos no vistos durante el entrenamiento al terminar cada etapa. Este subconjunto de datos no es obligatorio.

Los datos de test, que se utilizan al final del entrenamiento, evalúan el rendimiento del modelo una vez entrenado. Estos datos solo se utilizan una vez, y es importante que estén completamente separados de los datos de validación y entrenamiento, para asegurarnos que son datos que el modelo no ha visto antes y está prediciendo

por primera vez.

Algunos valores típicos para estas particiones son: entrenamiento (70-90%), test (10-20%) y validación (10-20%).

3.1.3 Problemas comunes en el entrenamiento y soluciones

A continuación, y una vez explicado todo el funcionamiento de una red neuronal, se explican algunos problemas usuales que podemos encontrarnos al ejecutar una red, y algunas soluciones para estos.

Uno de los problemas más sonados es el sobreajuste u "overfitting", que ocurre cuando el modelo aprende tan bien los detalles del conjunto de datos de entrenamiento que no es capaz de generalizar a nuevos datos. De esta forma, se conseguirá un rendimiento muy alto para los datos de entrenamiento, pero cuando pase a la parte de prueba el rendimiento de la red pasará a ser muy bajo.

El overfitting puede darse por utilizar modelos demasiado complejos, es decir, con demasiados parámetros (capas o neuronas), por utilizar un conjunto de datos de entrenamiento pequeño o por entrenar el modelo durante demasiadas épocas. Algunas soluciones al overfitting son:

- Aumentar los datos de entrenamiento, bien recopilando más datos o bien haciéndole modificaciones a los datos ya existentes (data augmentation)
- Dropout: Apagar aleatoriamente neuronas durante el entrenamiento para prevenir la co-adaptación de las mismas.
- Cross-Validation: Utilizar técnicas de validación cruzada para asegurar que el modelo generalice bien.

Por el contrario, el underfitting o subajuste ocurre cuando el modelo es demasiado simple para la complejidad de los datos de entrenamiento, dando así un mal resultado tanto con los datos de entrenamiento como con los datos de validación y test. La mayor solución a este problema es utilizar una arquitectura del modelo más compleja, es decir, con más capas y neuronas. Este problema también puede deberse a no estar entrenando el modelo durante las épocas suficientes, no permitiendo así que el modelo llegue al mínimo óptimo de la función de costes. La solución a esto es, claramente, entrenar más épocas.

Estos problemas se muestran gráficamente en la Figura 3.5.

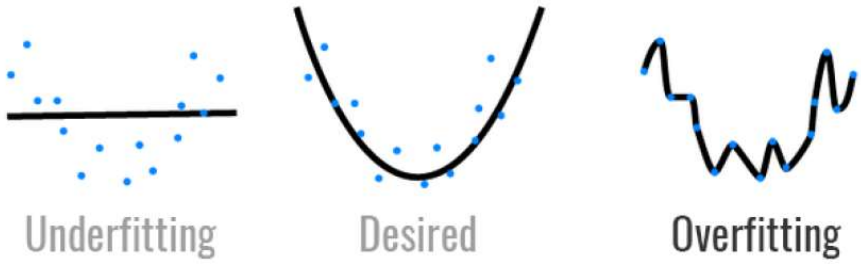


Figura 3.5. Overfitting y underfitting

3.2 Tipos de redes neuronales

Existen diversos tipos de redes neuronales según su arquitectura. A continuación, se describen algunos de los tipos más comunes de redes neuronales.

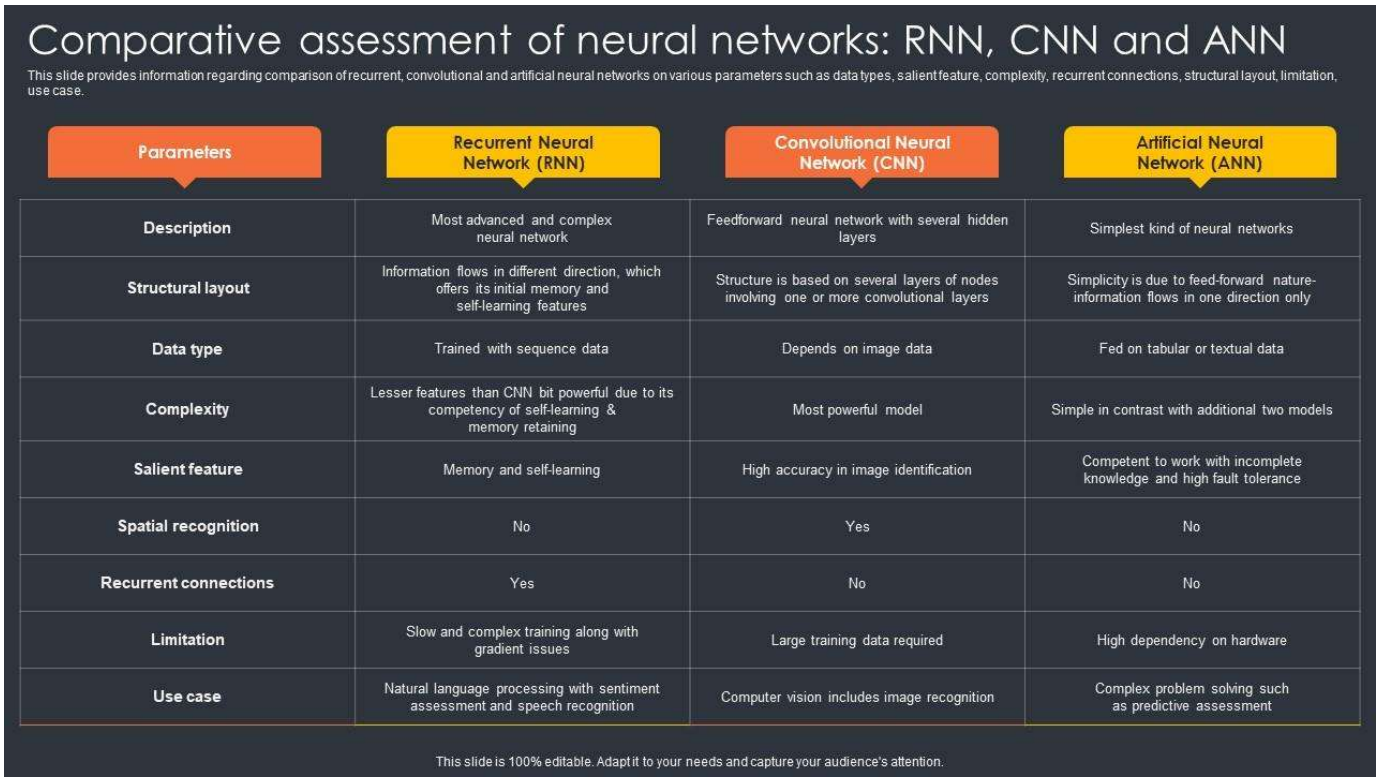
En primer lugar, el tipo más básico de red neuronal son las redes neuronales artificiales (ANN). Las ANN son genéricas y pueden usarse para una amplia gama de problemas, desde clasificación hasta regresión. Se caracterizan por su versatilidad, ya que pueden aprender representaciones complejas de los datos. Sin embargo, no tienen en cuenta la estructura espacial o temporal de los datos, lo que las vuelve más limitantes para soluciones con imágenes o con series temporales, donde se utilizan en su lugar redes CNN y RNN, respectivamente.

En nuestro caso práctico que envuelve este trabajo, esta será la red utilizada.

Las redes neuronales convolucionales (CNN) son unas de las redes neuronales más populares, especializadas en el procesamiento de datos con estructura de cuadrícula, como por ejemplo, imágenes. Utilizan capas convolucionales que aplican filtros para detectar características locales de los datos y capas de pooling para reducir la dimensionalidad. La capacidad de las CNN para capturar características espaciales las hace superiores a las ANN para tareas de visión artificial, es decir, para el procesamiento de imagen y video.

Por último, las redes neuronales recurrentes (RNN) tienen sus neuronas conectadas en forma de bucle, permitiendo así que la información vaya hacia atrás, manteniendo una memoria de estados anteriores. Esto permite que la red tenga en cuenta el contexto temporal o secuencial de la información, lo que es importante para tareas como el procesamiento del lenguaje natural y la predicción de series temporales.

En la Figura 3.6. observamos una comparativa de estos tres tipos de redes neuronales.



Comparative assessment of neural networks: RNN, CNN and ANN

This slide provides information regarding comparison of recurrent, convolutional and artificial neural networks on various parameters such as data types, salient feature, complexity, recurrent connections, structural layout, limitation, use case.

Parameters	Recurrent Neural Network (RNN)	Convolutional Neural Network (CNN)	Artificial Neural Network (ANN)
Description	Most advanced and complex neural network	Feedforward neural network with several hidden layers	Simplest kind of neural networks
Structural layout	Information flows in different direction, which offers its initial memory and self-learning features	Structure is based on several layers of nodes involving one or more convolutional layers	Simplicity is due to feed-forward nature-information flows in one direction only
Data type	Trained with sequence data	Depends on image data	Fed on tabular or textual data
Complexity	Lesser features than CNN bit powerful due to its competency of self-learning & memory retaining	Most powerful model	Simple in contrast with additional two models
Salient feature	Memory and self-learning	High accuracy in image identification	Competent to work with incomplete knowledge and high fault tolerance
Spatial recognition	No	Yes	No
Recurrent connections	Yes	No	No
Limitation	Slow and complex training along with gradient issues	Large training data required	High dependency on hardware
Use case	Natural language processing with sentiment assessment and speech recognition	Computer vision includes image recognition	Complex problem solving such as predictive assessment

This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

Figura 3.6. Comparativa de tipos de redes neuronales

3.3 Envelope Time-Delay Neural Network (ETDNN)

La red neuronal Envelope Time-Delay Neural Network (ETDNN) es una red neuronal utilizada en el campo de investigación de los amplificadores de potencia para aplicar DPD a la señal de entrada del PA.

Esta red está formada en primer lugar por una ANN con una capa de entrada, una capa oculta y una capa de salida. Esto lo convierte en una red muy sencilla, lo cual es una ventaja de cara al poco peso computacional que requiere. El número de neuronas de cada capa será un hiperparámetro que configuraremos nosotros al hacer una implementación práctica de esta red, que se explica en el capítulo 4. En segundo lugar, la capa de salida de la ANN pasa a un filtro de fase, lo que permite tener en cuenta la estructura temporal de los datos que en principio una ANN normal no contempla. Esto puede observarse en la Figura 3.7.

Como se observa, las señales de salida de la ANN solo dependen de la amplitud de las señales de entrada $|x(k)|$ y se envían al filtro de fase como coeficientes. El filtro de fase realiza la rotación de fase multiplicando los coeficientes de la ANN, cuyos valores son complejos ya que los pesos de entrenamiento de esta red han de ser complejos, por las señales de entrada de valores complejos $x(k)$, ya que esta vez no se aplica el módulo a la

señal.

El número de señales de entrada tanto en la ANN como en el filtro de fase, que han de ser iguales, será la cantidad de retrasos m de la señal que tengamos en cuenta para obtener un DPD óptimo.

Por otro lado, en la implementación de esta red, y para que el concepto de filtro de fase tenga sentido, es necesario que el número de neuronas de la capa de salida y el número de entradas en la capa de entrada sean iguales, ya que el filtro de fase será la suma de multiplicar cada coeficiente por la señal de entrada correspondiente.

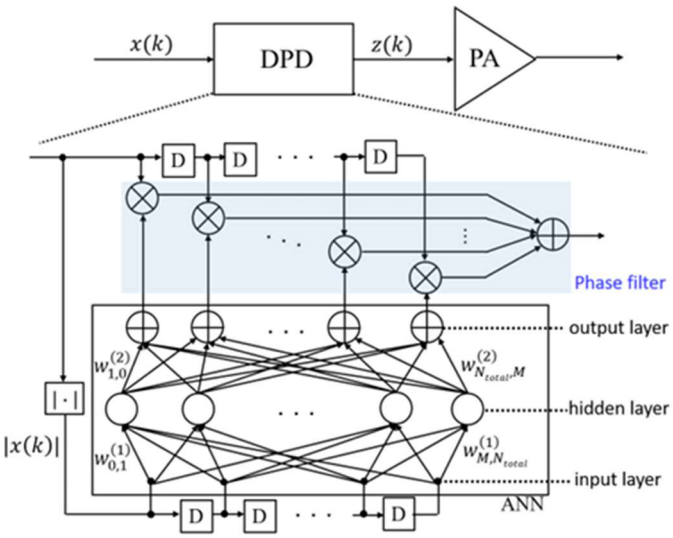


Figura 3.7. Envelope Time-Delay Neural Network (ETDNN)

Este mismo esquema se corresponde matemáticamente con la ecuación planteada en (3.6), siendo $z(k)$ la señal predistrsionada que se utiliza como entrada al PA.

$$z(k) = \sum_{m=0}^{M} \left\{ \sum_{j=1}^{N_{total}} w_{j,m^{\emptyset}}^{(2)} \left(\sum_{l=0}^{M} w_{l,j}^{(1)} \, |x(k-l)| + b_j^{(1)} \right) + b_m^{(2)} \right\} x(k-m) \qquad (3.6)$$

donde $w_{l,j,m}^{(1)}$ y $b_{j,m}^{(1)}$ son los pesos y los sesgos reales entre la capa de entrada y la capa oculta, y $w_{j,m^{\emptyset}}^{(2)}$ y $b_m^{(2)}$ son los pesos y sesgos complejos entre la capa oculta y la capa de salida, y M y N_{total} son la cantidad de neuronas de la capa de entrada/salida y de la capa oculta, respectivamente.

Esta red cumple con todas las restricciones del modelado físico de los PA, tanto de parte impar como de unidad de fase.

4 IMPLEMENTACIÓN DE LA SOLUCIÓN Y RESULTADOS OBTENIDOS

En este capítulo se va a tratar la solución que se ha implementado para resolver el problema de modelado y linealización del amplificador.

Esta solución, como se ha ido viendo a lo largo de los capítulos 2 y 3, consta de dos partes: en primer lugar, aplicar el algoritmo ILC al PA para una señal de entrada $u(n)$, obteniendo la señal predistorsionada óptima $x(n)$. Una vez aplicado el algoritmo, se procederá a implementar y ejecutar una red neuronal ETDNN con las entradas $u(n)$ y salidas $x(n)$ obtenidas del algoritmo.

Se buscará obtener el máximo parecido posible entre la señal $x(n)$ obtenida del algoritmo ILC y la señal de salida de la red neuronal, denominada $x_{estimado}(n)$.

4.1 Algoritmo ILC

Este apartado se divide en la implementación del algoritmo y los resultados obtenidos por el mismo.

4.1.1 Implementación

Esta parte ha sido implementada por el Grupo de Sistemas de Radiocomunicación de la Escuela Técnica Superior de Ingeniería. Para esta implementación se utilizó el setup mostrado en la Figura 4.1. y descrito a continuación.

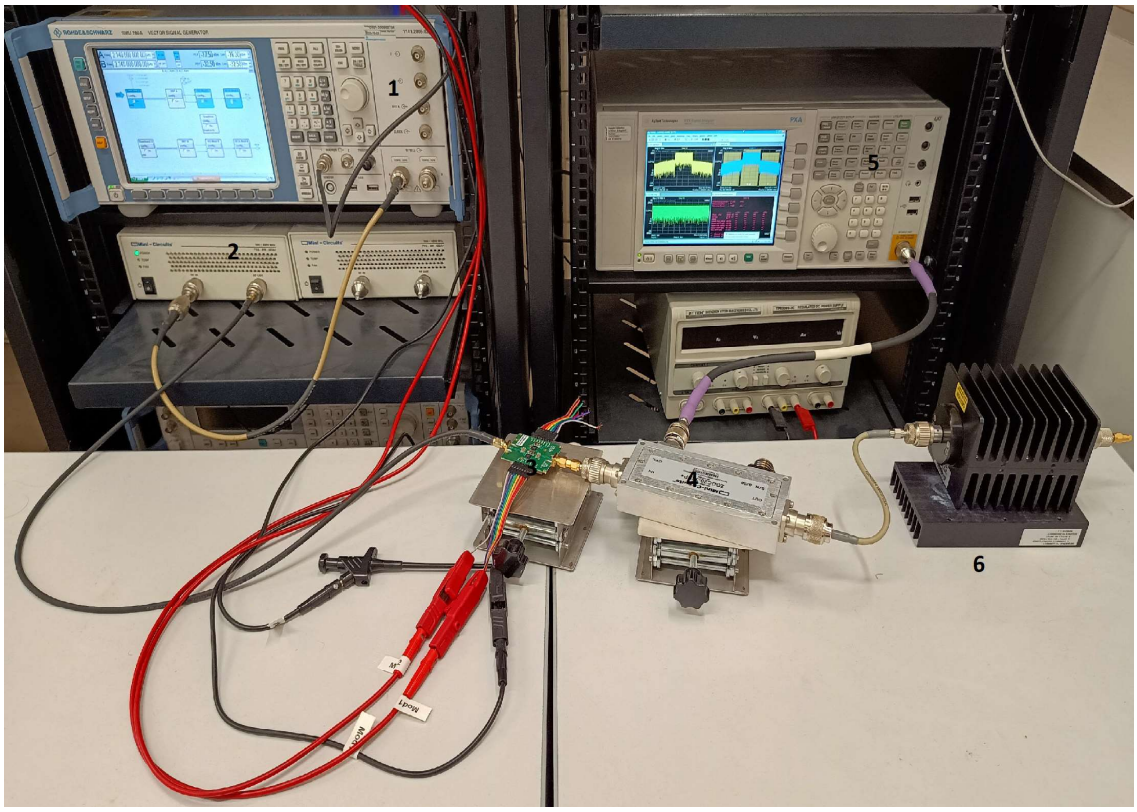


Figura 4.1. Setup de laboratorio para algoritmo ILC

En la imagen se han enumerado los distintos dispositivos utilizados. El número 1 corresponde a un generador de señal Rohde & Schwarz SMU200A y el número 2 a un preamplificador Minicircuits TVA-4W-422A+ trabajando en zona lineal. La señal de salida del preamplificador es la entrada del PA (número 3), que en este caso fue un SKY66394-11 de Skyworks con un amplio ancho de banda instantáneo, que operaba a una frecuencia de 2.14 GHz. La salida del PA pasa al analizador vectorial de señales (VSA) PXA-N9030A de Keysight Technologies (número 5), pasando antes por un acoplador direccional (número 4) que termina su puerto de salida con un atenuador seguido de una carga adaptada (número 6) para proteger al VSA de señales demasiado grandes.

Los datos que se han utilizado para esta implementación son n =10 iteraciones, ganancia objetivo del PA 26.85 dB, potencia en el generador de -30.5 dBm, lo que implica una potencia a la entrada del amplificador de -8 dBm, y potencia a la salida del amplificador de 23.2 dBm.

La señal de sondeo era una OFDM siguiendo el formato de forma de onda 5G-NR con un ancho de banda de 30 MHz y una numerología μ = 1, correspondiente a una separación entre subportadoras de 30 kHz. La señal OFDM tenía un PAPR de aproximadamente 11 dB y se generó con una frecuencia de muestreo de 92,16 MHz. Posteriormente, el analizador de señal adquirió muestras de la envolvente compleja a la salida del PA.

4.1.2 Resultados

El algoritmo ILC, como se ha explicado en el marco teórico de este trabajo, va calculando en cada iteración del algoritmo la señal $x(n)$ que, al actuar como entrada del PA, consigue una salida lineal.

Para comprobar que este algoritmo ha linealizado correctamente, se presentan a continuación las gráficas de las características AM/AM y AM/PM de la primera y la última iteración.

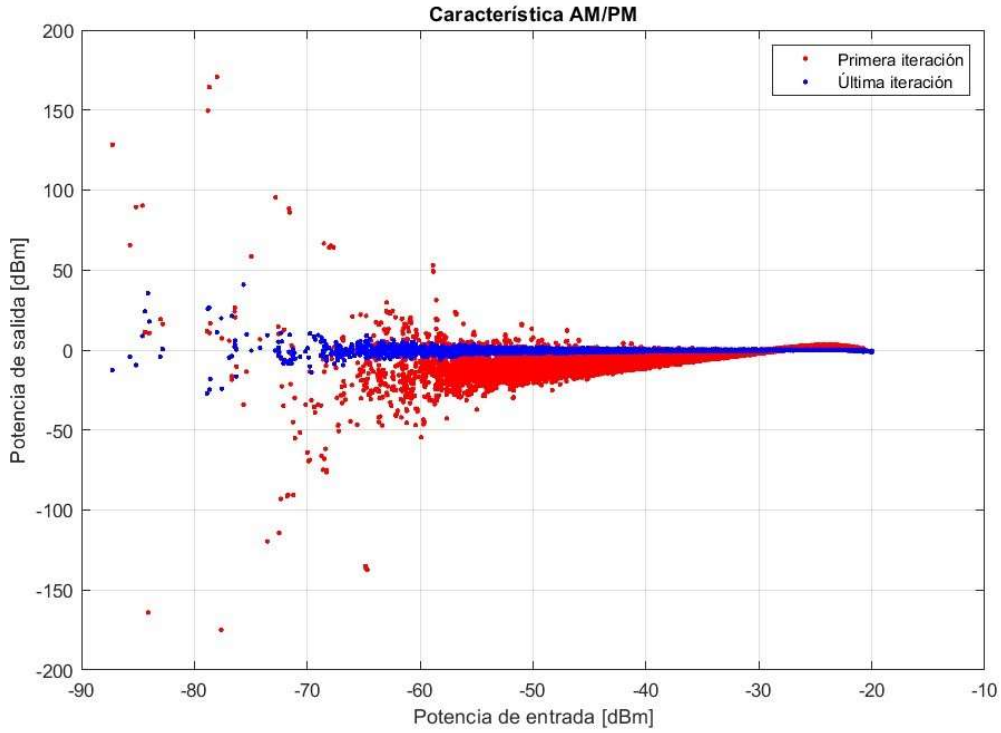


Figura 4.2. Característica AM/PM

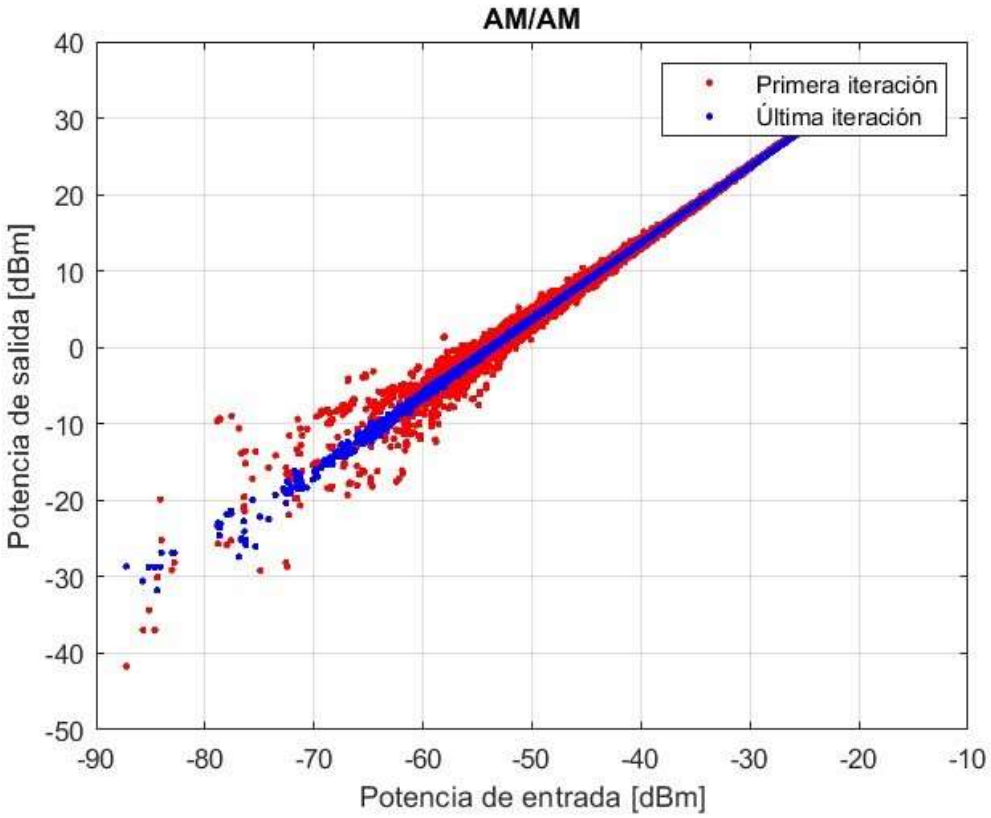


Figura 4.3. Característica AM/AM

Se observa en ambas gráficas que el comportamiento a la salida del amplificador se linealiza, ya que en la última iteración los resultados se asemejan más a una línea recta que en la primera.

Además, como se vió en el capítulo 2 de este trabajo, una figura de mérito muy útil para evaluar tanto la distorsión en banda como fuera de banda de una señal es el NMSE entre la señal de entrada y la de salida. Por ello, vamos a calcularlo tanto para la primera iteración como para la última, obteniendo estos resultados:

```
El resultado de NMSE para la primera iteración es: -27.4039 dB
El resultado de NMSE para la última iteración es: -51.0787 dB
```

Figura 4.4. Resultados de NMSE para la primera y la última iteración

Se observa claramente como el algoritmo consigue linealizar en la última iteración, pasando de un NMSE bastante alto a uno menor que -40 dB, que es el límite usado en modelado y linealización de amplificadores para concluir que el parecido entre la salida y la entrada escalada es bastante bueno.

Además, si calculamos el NMSE en las diez iteraciones, vemos como éste va bajando progresivamente, con lo cual, concluimos que el algoritmo funciona tal y como esperábamos.

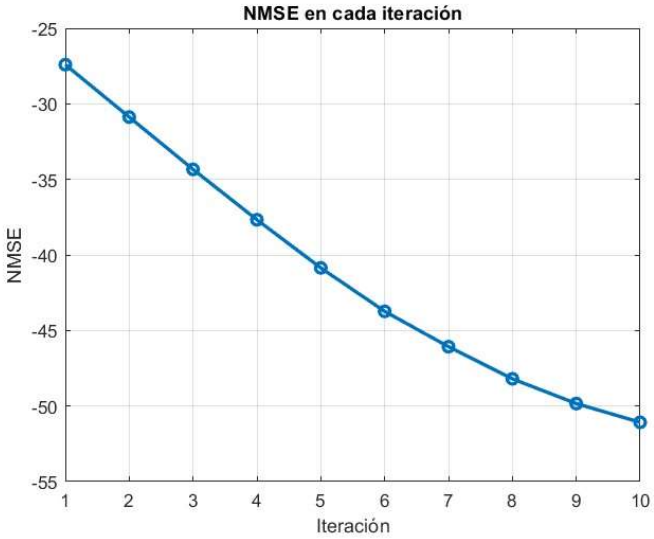


Figura 4.5. NMSE del algoritmo ILC por iteración

Otra forma de corroborar que el algoritmo está funcionando correctamente es observando las diferencias del espectro entre la primera y la última iteración. Esto se muestra en la Figura 4.6.

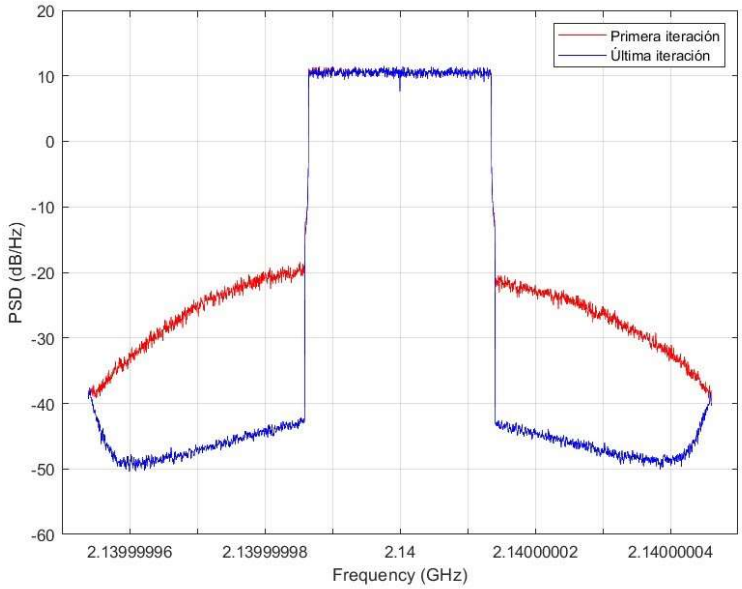


Figura 4.6. Espectro

Se observa como el espectro mejora significativamente, reduciéndose de la primera a la última iteración el recrecimiento espectral, que es mucho más plano en la última iteración en comparación con la primera.

4.2 ETDNN

4.2.1 Implementación

La implementación de esta red neuronal se realizará utilizando el lenguaje de programación Python, que es el lenguaje más usual y extendido cuando hablamos de redes neuronales. Su popularidad se debe a que tiene la posibilidad de implementar infinidad de librerías, entre las que se encuentra Numpy, para realizar operaciones matemáticas con nuestros datos, u Os, que permite cargar directamente los datos a utilizar desde Google Drive. Esta última es una ventaja frente a otros lenguajes como Matlab, que no permiten el uso de este tipo de librerías. Además, su popularidad también permite que la documentación sobre implementar redes neuronales con Python sea mucho más amplia que para otros lenguajes, lo que me ha convencido para utilizar este lenguaje.

Por otro lado, Python cuenta con varios frameworks tanto para el manejo y visualización de datos, como para la implementación de modelos de Machine Learning. En la Figura 4.7. se pueden ver los más utilizados.

TensorFlow y PyTorch son dos de los frameworks más populares para el desarrollo de modelos de aprendizaje profundo. En el desarrollo de este trabajo se ha utilizado TensorFlow por los siguientes motivos principalmente:

- TensorFlow es ampliamente utilizado en producción debido a su soporte para despliegue en varias plataformas, como servidores, dispositivos móviles y navegadores.
- TensorFlow integra Keras como su API de alto nivel. Keras, ejecutándose sobre Tensorflow, permite construir y entrenar modelos de manera rápida y sencilla sin tener que preocuparnos por los detalles complejos del backend de TensorFlow. Esto acelera el proceso de desarrollo y permite centrarse más en experimentar con los modelos que en la programación en sí.

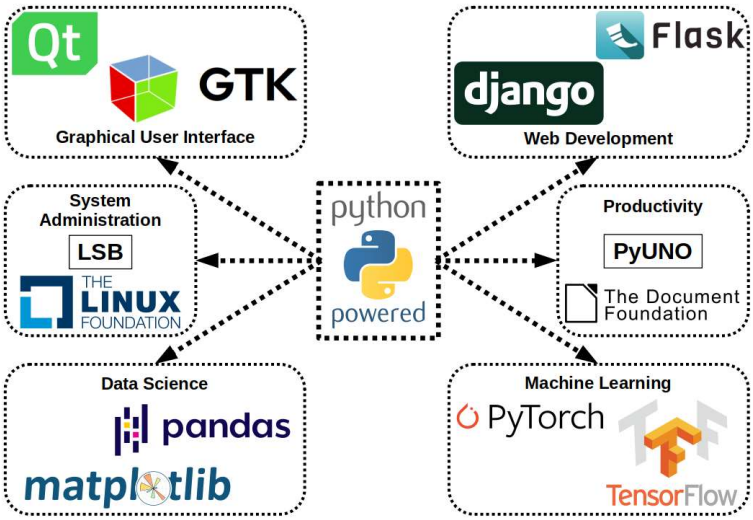


Figura 4.7. Frameworks en Python

El entrenamiento de las redes neuronales es complejo computacionalmente hablando en la mayoría de los casos, y se necesita un entorno hardware potente que permita la ejecución en un tiempo razonable. En el caso de este trabajo, este entorno no ha sido local sino en la nube, utilizando Google Colab.

Este entorno en la nube permite, en su versión gratuita, ejecuciones con GPU o TPU de 12 horas máximo y hasta 23 GB de RAM. En este caso, se ha utilizado la GPU ya que ofrecía mejores prestaciones que la TPU.

Una vez sabemos qué entorno y framework vamos a utilizar para el entrenamiento de esta red, vamos ahora a describir los puntos más importantes de la implementación de esta solución.

En primer lugar, se muestran en la Figura 4.8 los imports necesarios para este trabajo.

Las librerías drive y os se utilizan para cargar los datos desde Google Drive directamente, es decir, en este caso tampoco es necesario cargar los datos en local, sino que se indica que se va a utilizar Drive y la ruta de acceso al fichero con los datos se especifica utilizando os.chdir().

```
#%%#%%IMPORTS
import scipy.io
from google.colab import drive
import os
import numpy as np
import matplotlib.pyplot as plt
```

Figura 4.8. Librerías utilizadas

Por un lado, se carga la estructura .mat que contiene los datos que utilizaremos, siendo u y x la entrada y la salida del DPD implementado mediante el algoritmo ILC, respectivamente. Por otro lado, u_test y x_test son los datos pertenecientes a la validación del algoritmo ILC que, en el caso de la red neuronal, se utilizarán para testear la solución.

Esas son las cuatro columnas que contiene la estructura "datos.mat". Cada columna se pasa a un array independiente, que además nos aseguramos que sea unidimensional utilizando el método flatten(). También debemos asegurarnos de que los datos sean de tipo complejo, en este caso, con precisión doble (64 bits para la parte real y 64 bits para la parte imaginaria, totalizando 128 bits).

```
# Cargar el archivo .mat
mat_data = scipy.io.loadmat('datos.mat')

# Extraer los datos relevantes
u = mat_data['u_ILC'].flatten()
u_test = mat_data['u_ILC_test'].flatten()
x = mat_data['x_ILC'].flatten()
x_test = mat_data['x_ILC_test'].flatten()

u_data = u.reshape(-1, 1).astype(np.complex128)
x_data = x.reshape(-1, 1).astype(np.complex128)
utest_data = u_test.reshape(-1, 1).astype(np.complex128)
xtest_data = x_test.reshape(-1, 1).astype(np.complex128)
```

Figura 4.9. Carga de los datos.

Antes de generar la red, debemos asegurarnos de que los datos tienen la estructura con la que necesitamos que entren en la red. Para ello, se hacen dos transformaciones sobre los mismos, que se muestran en la Figura 4.10.

```
#Normalización:
u_data = (u_data - min(u_data)) / (max(u_data) - min(u_data))
x_data = (x_data - min(x_data)) / (max(x_data) - min(x_data))
utest_data = (utest_data - min(utest_data)) / (max(utest_data) - min(utest_data))
xtest_data = (xtest_data - min(xtest_data)) / (max(xtest_data) - min(xtest_data))


U_data = np.column_stack([u_data[window_size-i-1:len(u_data)-i] for i in range(window_size)])
x_data_adjusted = x_data[window_size - 1:]
```

Figura 4.10. Transformaciones sobre los datos de entrada.

En primer lugar, en aplicaciones de inteligencia artificial es común normalizar los datos. Esto se debe a que así el algoritmo de optimización puede ajustarse más uniformemente a todas las características, lo que conlleva una convergencia más rápida y eficiente. Además, se reduce el riesgo de problemas de estabilidad numérica. Se puede optar por varios tipos de normalización pero en este caso, por ser común en el tratamiento de señales, se ha optado por una normalización entre 0 y 1. En este punto, los datos tienen la siguiente estructura:

```
Forma de u_data: (368640, 1)
Forma de x_data: (368640, 1)
Forma de utest_data: (368640, 1)
Forma de xtest_data: (368640, 1)
```

Figura 4.11. Estructura de los datos antes de las transformaciones.

Por otro lado, ya se explicó en el marco teórico de la ETDNN que por la red pasan m muestras retrasadas de la señal. Este número m se denominará window_size en esta implementación, y será un hiperparámetro que elegiremos nosotros e iremos mejorando para obtener mejores prestaciones de la red.

Esto implica que u_data, es decir, la entrada a la red, tendrá que tener m columnas, donde cada una de ellas será una muestra retrasada de la señal e incluirá un número de valores que será el tamaño total del vector – m muestras. Las muestras irán para cada columna desde el valor “m” hasta el final del vector, de forma cíclica, tal como se muestra en la Tabla 1.

u_data(m)	u_data(m-1)	…	u_data(1)
u_data(m+1)	…	…	u_data(2)
…	u_data(end-2)	…	
u_data(end)	u_data(end-1)	…	u_data(end-m)

Tabla 4.1. Estructura de los datos de entrada a la red.

Por último, para que los datos de entrada y salida de la red tenga el mismo tamaño, se quitan las primeras m muestras de la señal de salida, quedando así la misma estructura que para la primera muestra de la entrada, es decir, la señal sin retardo.

Así, se conseguirá una estructura de los datos que será:

```
Forma de U_data: (368541, 100)
Forma de x_data_adjusted: (368541, 1)
```

Figura 4.12. Tamaño de los datos con las transformaciones.

Una vez los datos tienen la estructura deseada, podemos pasar a la implementación de la propia red.

En este caso, se ha utilizado el siguiente modelo de red neuronal implementado manualmente:

```
class NeuralNetwork(Model):
    def __init__(self, input_dim, hidden_units, output_units, **kwargs):
        super(NeuralNetwork, self).__init__(**kwargs)
        self.input_layer = CustomLayer(hidden_units, complex_weights=False)
        self.hidden_layer = CustomLayer(output_units, complex_weights=True)
        self.phase_filter = PhaseFilter()

    def call(self, inputs):
        print(f"NeuralNetwork input shape: {inputs.shape}")
        abs_inputs = tf.abs(inputs)
        x = self.input_layer(abs_inputs)
        x = tf.keras.activations.sigmoid(x)
        x = self.hidden_layer(x)
        output = self.phase_filter(x, inputs)
        print(f"NeuralNetwork output shape: {output.shape}")
        return output
```

Figura 4.13. Modelo de red neuronal implementado.

Esta clase está dividida en dos partes.

Por un lado, el constructor ('__init__') define las tres capas mencionadas de la ETDNN: una capa de entrada que no utiliza pesos complejos, una capa oculta que sí utiliza pesos complejos y un filtro de fase.

Por otro lado, el método call es donde se define el "forward pass" de la capa o el modelo, es decir, define la lógica computacional que sigue una entrada que pasa por una capa o modelo. Se puede observar que, en este caso, las transformaciones son las siguientes: primero, los datos de entrada se convierten de valores complejos a absolutos, y pasan por la "input layer" definida en el constructor; después, a la salida de esta capa se le aplica la función sigmoide para convertir cada valor de x en un rango entre 0 y 1; esta salida se pasa a través de la capa oculta de la red; y por último pasa por el filtro de fase.

Para que este modelo funcione, también se han tenido que definir los modelos de capa y filtro de fase, indicados en la Figura 4.14.

En primer lugar, la clase que define la capa que se utiliza tanto en la capa de entrada como en la oculta tiene dos argumentos especificados en el constructor: units, que indica el número de neuronas de la capa, y complex_weigths, un argumento de tipo booleano que indica si los pesos serán complejos. Si es *True*, se generan pesos reales e imaginarios por separado.

El método build se llama cuando se conoce la forma de la entrada. Si se aceptan pesos complejos, se crean dos variables para los pesos reales e imaginarios, y si no, solo se crea un conjunto de pesos.

El método 'call' procesa la entrada y realiza una multiplicación matricial. Si los pesos son complejos, se calculan las partes real e imaginaria por separado y luego se combinan en un número complejo. Si los pesos no son complejos, se realiza la multiplicación estándar con los pesos.

Por último, la capa está diseñada para aplicar un filtro de fase sobre la salida de las capas anteriores. Calcula un producto entre la salida de la capa anterior ($inputs$) y la entrada original a la red ($original\_inputs$), seguido de una reducción sumando los valores a lo largo de la última dimensión, manteniendo la dimensionalidad (keepdims=True).

```
class CustomLayer(Layer):
    def __init__(self, units, complex_weights=False, **kwargs):
        super(CustomLayer, self).__init__(**kwargs)
        self.units = units
        self.complex_weights = complex_weights

    def build(self, input_shape):
        if self.complex_weights:
            self.kernel_real = self.add_weight(shape=(input_shape[-1], self.units),
                                               initializer='random_normal',
                                               trainable=True)
            self.kernel_imag = self.add_weight(shape=(input_shape[-1], self.units),
                                               initializer='random_normal',
                                               trainable=True)
        else:
            self.kernel = self.add_weight(shape=(input_shape[-1], self.units),
                                          initializer='random_normal',
                                          trainable=True)

    def call(self, inputs):
        print(f"CustomLayer input shape: {inputs.shape}")
        if self.complex_weights:
            real_part = tf.matmul(inputs, self.kernel_real)
            imag_part = tf.matmul(inputs, self.kernel_imag)
            output = tf.complex(real_part, imag_part)
        else:
            output = tf.matmul(inputs, self.kernel)
        return output

class PhaseFilter(Layer):
    def __init__(self, **kwargs):
        super(PhaseFilter, self).__init__(**kwargs)

    def call(self, inputs, original_inputs):
        print(f"PhaseFilter input shape: {inputs.shape}")
        print(f"PhaseFilter original input shape: {original_inputs.shape}")
        original_x = tf.cast(original_inputs, dtype=tf.complex64)
        output = tf.reduce_sum(tf.multiply(inputs, original_x), axis=-1, keepdims=True)
        print(f"PhaseFilter output shape: {output.shape}")
        return output
```

Figura 4.14. Definición de capa y filtro de fase.

Una vez definido el modelo de red neuronal que se va a utilizar, se definen las neuronas que se utilizarán en la capa de entrada, en la capa oculta y en la capa de salida. Con estas dimensiones, se llama al modelo y se compila. La compilación se realizará utilizando un optimizador Adam, que es un método de optimización adaptativa basado en el descenso por gradiente explicado en el capítulo 3 de este trabajo, y una función de pérdida que mide el Error Cuadrático Medio (MSE), muy utilizada en problemas de regresión.

Después, se entrena el modelo con los hiperparámetros épocas, batch_size y validation_split, que muestra el porcentaje de valores aleatorios de entrada y salida que se van a utilizar para la validación del modelo (en este caso, el 10%).

Por último, se predice el modelo con los datos de entrada de test, obteniendo una salida predicha $x\_data\_estimado$.

```
#Llamada al modelo
model = NeuralNetwork(input_dim, hidden_units, output_units)

# Compilación del modelo
model.compile(optimizer='adam', loss='mean_squared_error')

# Entrenamiento del modelo
history = model.fit(U_data, x_data_adjusted, epochs=15, batch_size=10, validation_split=0.1)

# Predicción
x_data_estimado = model.predict(U_data_test)
```

Figura 4.15. Entrenamiento del modelo.

4.2.2 Resultados

Para este apartado, se mostrarán los resultados obtenidos para distintas configuraciones de hiperparámetros.

En primer lugar, se muestra una tabla donde se calcula tanto el NMSE de validación como el número de pesos que se han entrenado en la red para distintas clasificaciones.

El número de pesos se ha calculado siguiendo (4.1).

$$Nºpesos = Épocas\ x\ (Neuronas_{entrada} + Neuronas_{oculta} + Neuronas_{salida}) \quad (4.1)$$

ÉPOCAS	Neuronas en la capa de entrada y salida	Neuronas en la capa oculta	BATCH_SIZE	NMSE validación(dB)	Número de pesos
10	10	64	10	-34.305	840 pesos
10	15	64	10	-35.23	940 pesos
10	15	64	32	-35.005	940 pesos
15	15	64	32	-34.88	1410 pesos
15	20	128	32	-35.67	2520 pesos
15	25	128	32	-35.82	2670 pesos
15	100	128	32	-36.15	4920 pesos
15	100	64	10	-36.52	2685 pesos
15	100	128	10	-36.74	4920 pesos

Tabla 4.2. NMSE y número de pesos para cada configuración.

Algunos detalles observados en el entrenamiento de la red son: al aumentar el tamaño de los hiperparámetros, disminuye el valor de NMSE significativamente. Sobre todo, el hiperparámetro que más afecta es el número de neuronas a la entrada y salida de la red o, lo que es lo mismo, la cantidad de retrasos de la señal que se van a

tener en cuenta. También cabe destacar que, al aumentar estos parámetros, obtenemos mejores resultados prácticos, pero el número de pesos también aumenta considerablemente, lo que implica mayor carga computacional, ejecución más lenta, y la necesidad de mayores recursos hardware.

Por tanto, muchas veces es mejor optar por una solución intermedia, que no suponga demasiado computacionalmente, pero a la vez obtenga buenos resultados de NMSE. A continuación, se presentan los resultados de la comparación de la entrada al PA (señal x) del algoritmo ILC con la predicha por la red para tres casos distintos: uno con un NMSE malo, otro intermedio, y el mejor. Como la señal tiene muchas muestras, se han cogido únicamente 1000 muestras, concretamente de la 10000 a la 11000. Esta elección ha sido aleatoria.

En primer lugar, se presenta el peor resultado de todos: 10 épocas, 10 retrasos, 64 neuronas en la capa oculta, batch_size de 10. Con estos datos, se obtiene un NMSE de validación de -34.305 dB, y los resultados de la comparación del módulo de la señal, la comparación de la parte real de la señal, y de la parte imaginaria de la señal, se muestran en las Figuras 4.16, 4.17 y 4.18 respectivamente.

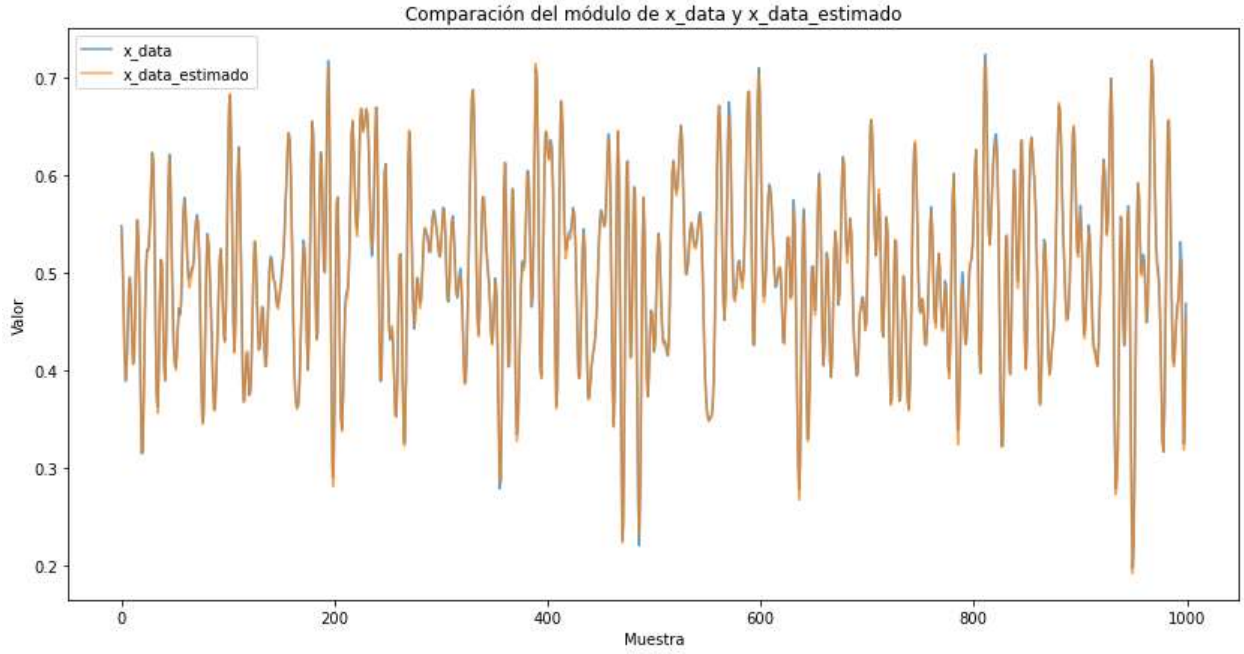


Figura 4.16. Comparación del módulo de x_data y x_data estimado para el peor caso.

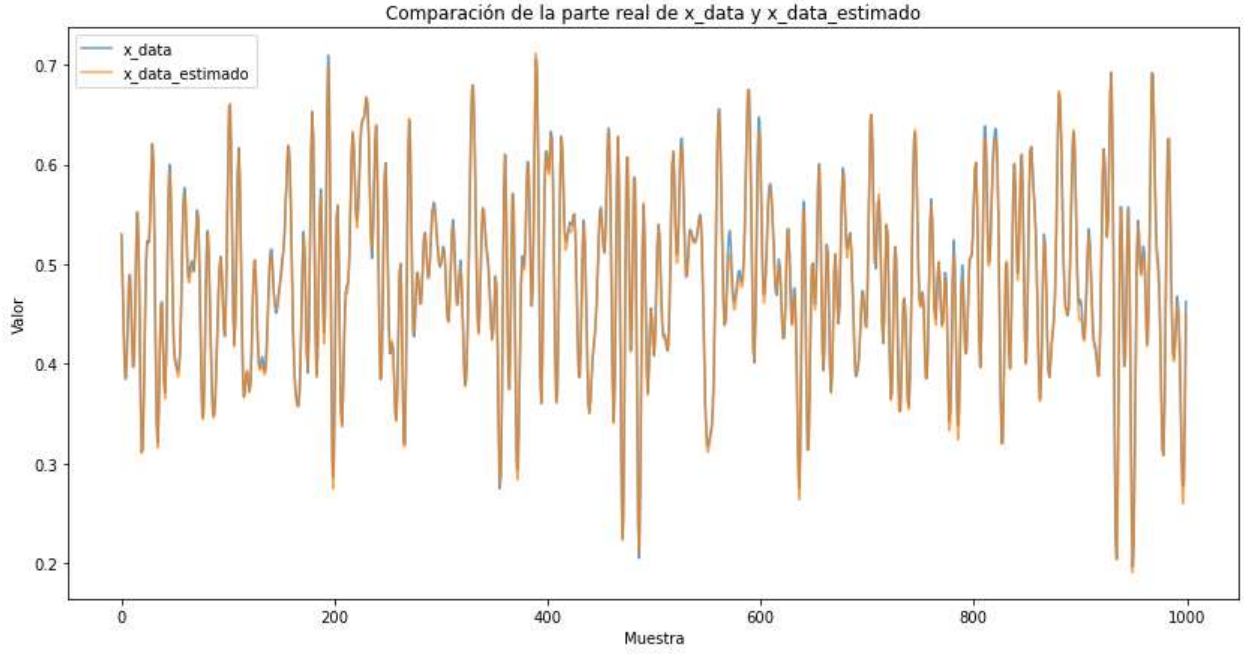


Figura 4.17. Comparación de la parte real de x_data y x_data_estimado para el peor caso

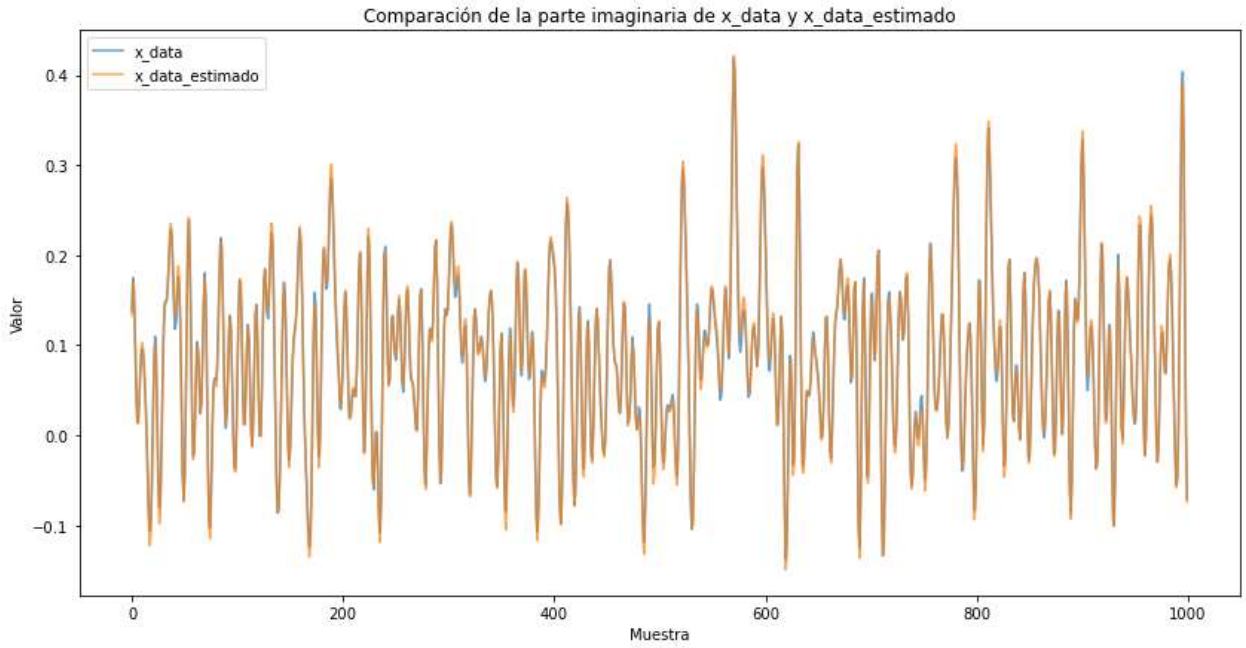


Figura 4.18. Comparación de la parte imaginaria de x_data y x_data_estimado para el peor caso

En segundo lugar, se presenta un resultado intermedio: 15 epocas, 20 retrasos, 128 neuronas en la capa ocultta, 32 de batch_size. Con estos datos, se obtiene un NMSE de validación de -35.67 dB, y los resultados de la comparación del módulo de la señal, la comparación de la parte real de la señal, y de la parte imaginaria de la señal, se muestran en las Figuras 4.19, 4.20 y 4.21 respectivamente.

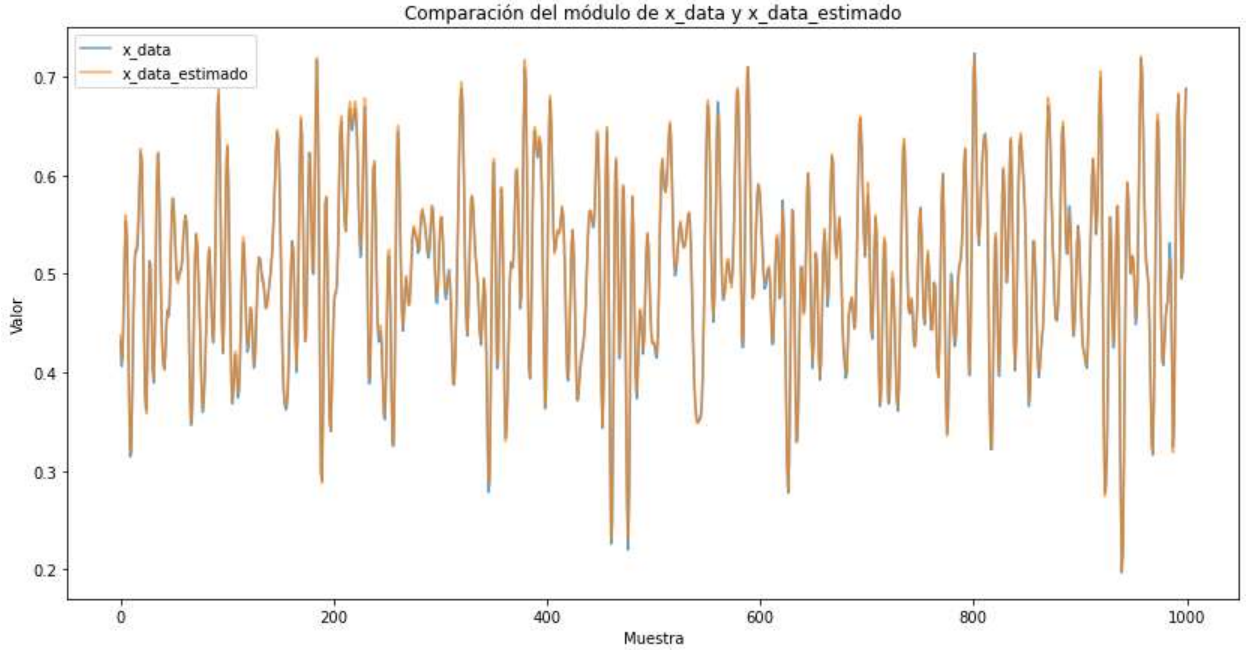


Figura 4.19. Comparación del módulo de x_data y x_data_estimado para un caso intermedio

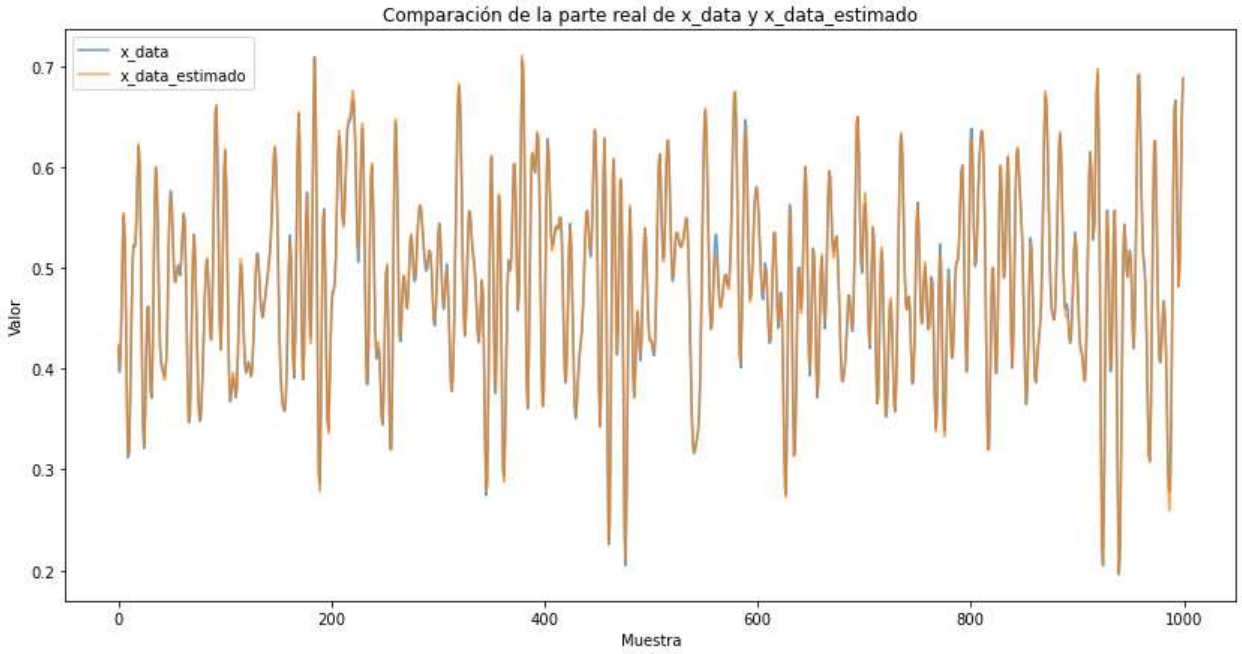


Figura 4.20. Comparación de la parte real de x_data y x_data_estimado para un caso intermedio

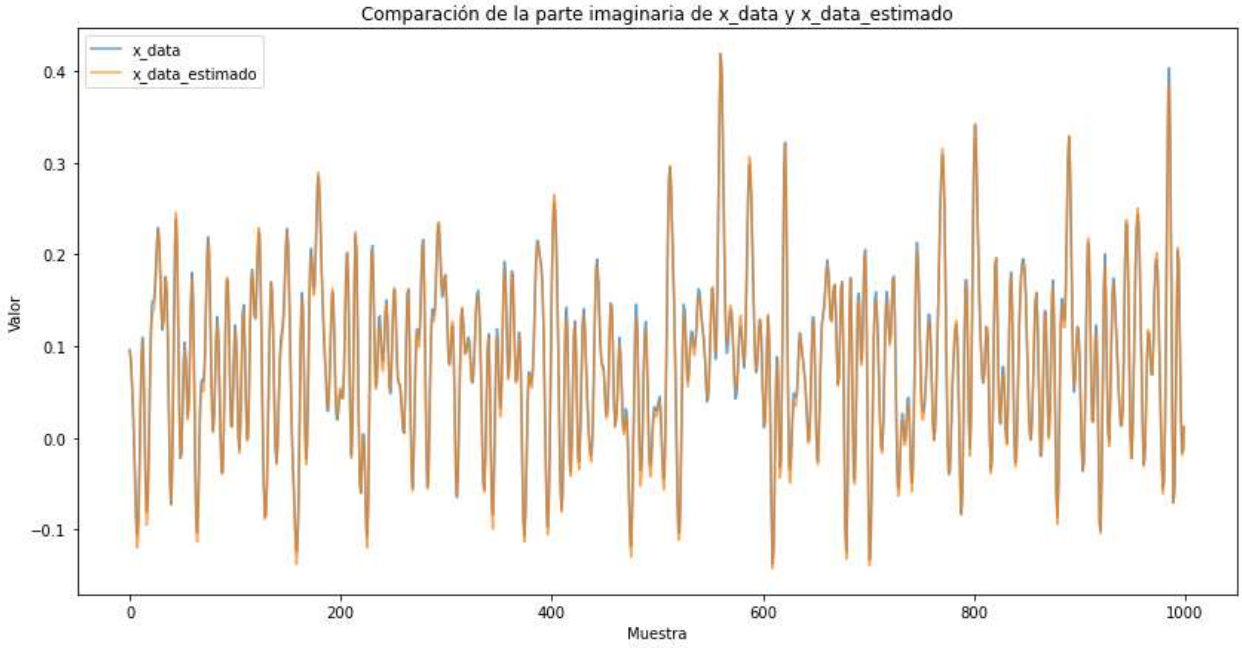


Figura 4.21. Comparación de la parte imaginaria de x_data y x_data_estimado para un caso intermedio

Por último, se presenta el mejor resultado de todos: 100 retrasos, 15 epochs, 128 capas y batch size de 10. Con estos datos, se obtiene un NMSE de validación de – 36.75 dB, y los resultados de la comparación del módulo de la señal, la comparación de la parte real de la señal, y de la parte imaginaria de la señal, se muestran en las Figuras 4.22, 4.23 y 4.24 respectivamente.

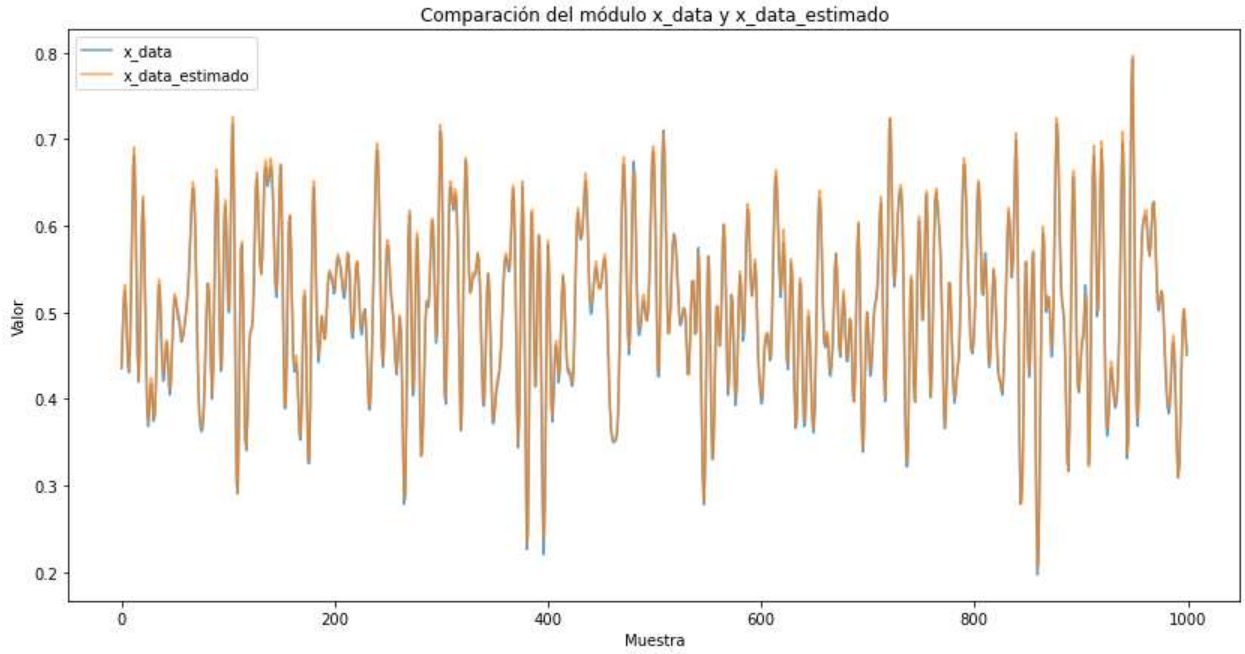


Figura 4.22. Comparación del módulo de x_data y x_data_estimado para el mejor caso

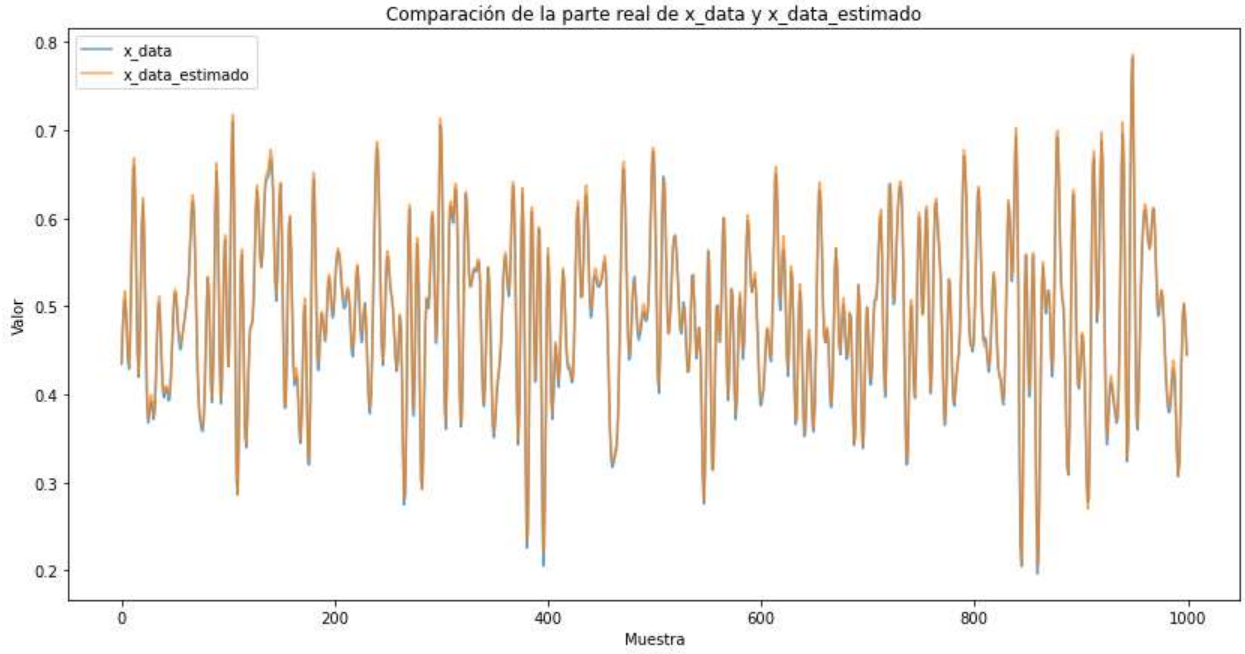


Figura 4.23. Comparación de la parte real de x_data y x_data_estimado para el mejor caso

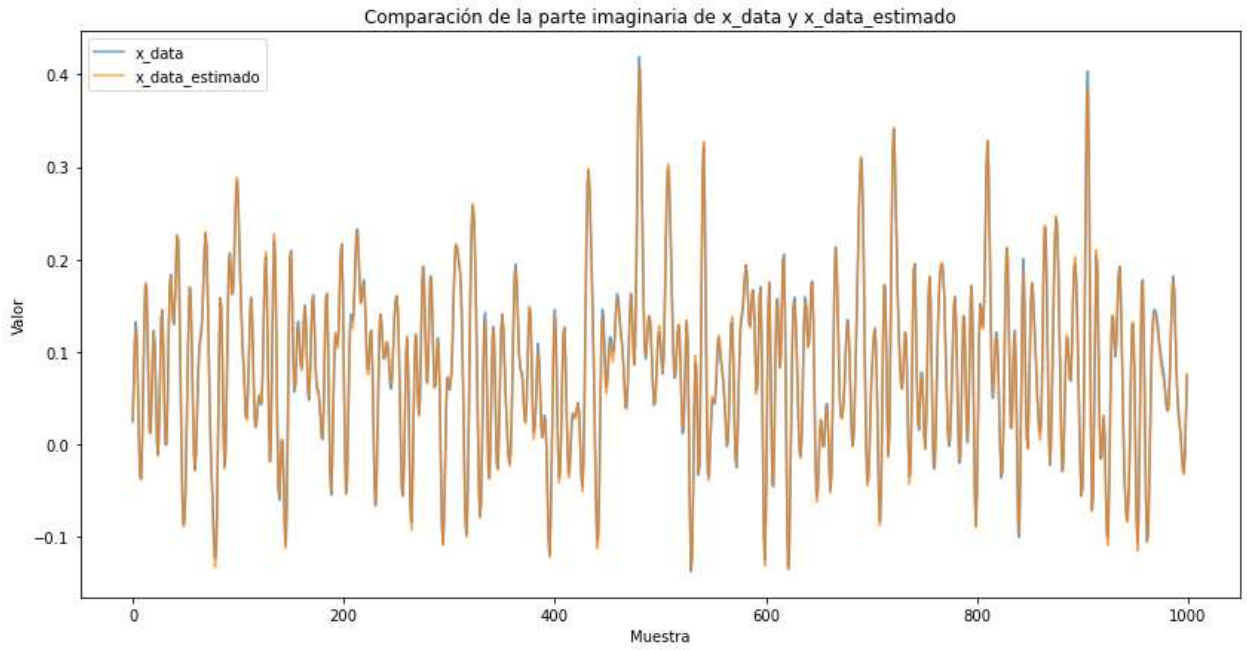


Figura 4.24. Comparación de la parte imaginaria de x_data y x_data_estimado para el mejor caso

Se puede observar que, en cualquiera de los casos, las señales están completamente alineadas, y las únicas diferencias observables son en los picos. Podemos concluir entonces que la predicción es bastante buena, aunque claramente observamos como las señales son mucho más parecidas en el mejor caso que en el peor.

Sin embargo, al intentar seguir subiendo de valor los hiperparámetros para conseguir resultados mejores, más cercanos a -40 dB, la señal de pérdidas en el entrenamiento era menor, pero los resultados de test peores, lo que quiere decir que el modelo estaba sobreentrenado y estaba aplicando overfitting.

La decisión entre elegir un modelo más liviano computacionalmente pero con peores resultados o uno con más carga computacional pero más correcto, dependerá de la aplicación y también del hardware del que disponga el dispositivo que se va a utilizar como predistorsionador, en este caso. Resulta aliviador observar como con un pequeño número de pesos, el modelo predice la señal de una forma eficiente, lo que conlleva que no se necesitan grandes recursos para obtener buenos resultados.

En este caso, al estar utilizando Google Colab para ejecutarlo, optamos por el mejor resultado, ya que los recursos hardware no son un problema. Para este caso se presenta la señal error en la Figura 4.25.

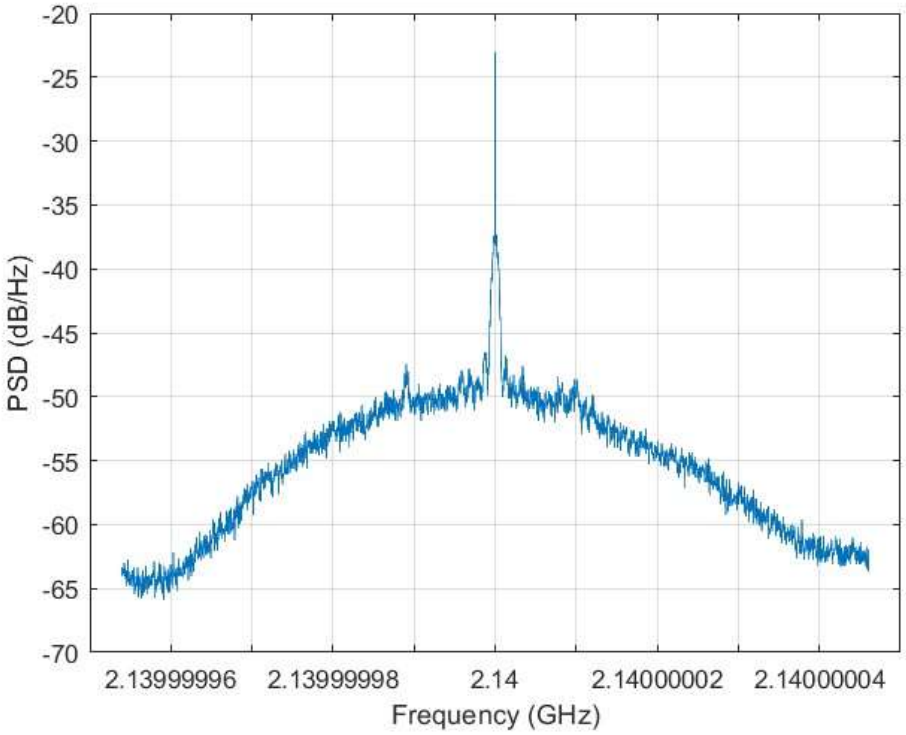


Figura 4.25. Señal error para el mejor caso

5 CONCLUSIONES Y LÍNEAS FUTURAS

La predistorsión digital (DPD) se ha consolidado como una herramienta crucial en la optimización de sistemas de amplificación de potencia, especialmente en aplicaciones donde la eficiencia energética y la integridad de la señal son primordiales. Este enfoque permite mitigar las distorsiones no lineales inherentes a los amplificadores de potencia, facilitando una operación más cercana al punto de saturación sin comprometer la calidad de la señal.

En este trabajo se ha observado cómo utilizar redes neuronales es una solución eficiente a las técnicas de DPD, aunque sea necesario un estudio previo con el que obtengamos la señal de salida del DPD necesaria para obtener una salida del PA lineal. Además, podemos concluir que para pocos recursos hardware obtenemos una solución decente, y si éstos se van mejorando, también mejora la predicción del modelo, llegando a niveles óptimos de NMSE. Concretamente, el número de muestras anteriores de la señal de entrada al modelo que se parametricen afecta en gran medida a esta predicción. Sin embargo, no se alcanzan valores de NMSE menores de -40 dB, que es el punto óptimo exigido en comunicaciones. Por ello, como mejora, se podría seguir modificando hiperparámetros que no sean la memoria de la señal, ya que al aumentarla a más de 100 muestras hemos empezado a notar overfitting en el entrenamiento de la red. Además, se podría haber probado con otras redes neuronales que aparecen en artículos de investigación y resultan, a nivel teórico, eficientes para este problema, como la PG Jannet.

APÉNDICE A:
DATASHEET DEL AMPLIFICADOR UTILIZADO

SKYWORKS®

DATA SHEET

SKY66394-11: 2000 to 2300 MHz Wide Instantaneous Bandwidth High-Efficiency Power Amplifier

Applications

- FDD and TDD 2G/3G/4G LTE systems
- 3GPP bands 1, 4, 10, and 23 small-cell base stations
- Driver amplifier for micro-base and macro-base stations
- Active antenna array and massive MIMO

Features

- High-efficiency: PAE = 35% @ +28 dBm
- High linearity: +28 dBm with < –50 dBc ACLR with pre-distortion (40 MHz LTE, 8.5 dB PAR signal)
- High gain: 38 dB
- Excellent input and output return loss: to 50 Ω system
- Integrated active bias: performance compensated over temp
- Integrated enable On/Off function: PAEN = 1.7 to 2.5 V
- Single supply voltage: 5.0 V
- Pin-to-pin compatible PA family supporting all 3GPP bands
- Compact (16-pin, 5 × 5 × 1.3 mm) package (MSL3, 260 °C per JEDEC J-STD-020)

Description

The SKY66394-11 is a high-efficiency fully input/output matched power amplifier (PA) with high gain and linearity. The compact 5 × 5 mm PA is designed for FDD and TDD 2G/3G/4G LTE small cell base stations operating from 2000 to 2300 MHz. The active biasing circuitry is integrated to compensate PA performance over temperature, voltage, and process variation.

The SKY66394-11 is part of a high-efficiency, pin-to-pin compatible PA family supporting all 3GPP bands. Table 1 lists the pin-to-pin compatible parts in the PA family.

A block diagram of the SKY66394-11 is shown in Figure 1. The device package and pinout for the device are shown in Figure 2. Signal pin assignments and functional pin descriptions are described in Table 2.

Table 1. Pin-to-Pin Compatible PA Family

Part Number	Frequency (MHz)	3GPP Band
SKY66391-12	1800 to 1900	Bands 3 and 9
SKY66399-11	1900 to 2000	Bands 2, 25, 33, 36, and 37
SKY66394-11	2000 to 2300	Bands 1, 4, 10, and 23
SKY66397-12	2300 to 2700	Bands 7, 38, and 41

Skyworks Green™ products are compliant with all applicable legislation and are halogen-free. For additional information, refer to *Skyworks Definition of Green™*, document number SQ04–0074.

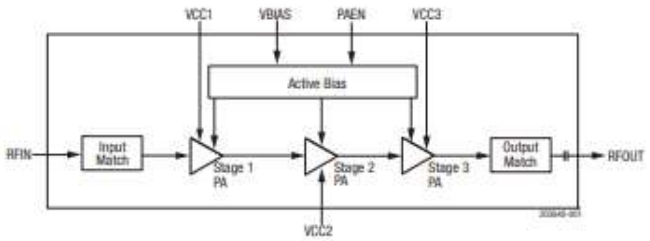


Figure 1. SKY66394-11 Block Diagram

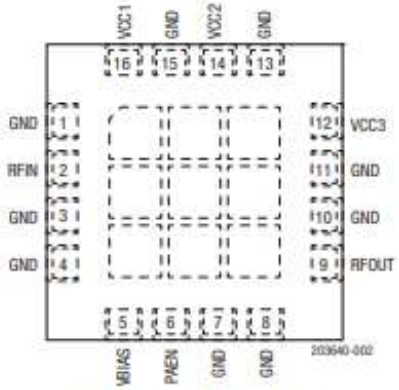


Figure 2. SKY66394-11 Pinout
(Top View)

Table 2. SKY66394-11 Signal Descriptions[1]

Pin	Name	Description	Pin	Name	Description
1	GND	Ground	9	RFOUT	RF output port
2	RFIN	RF input port[2]	10	GND	Ground
3	GND	Ground	11	GND	Ground
4	GND	Ground	12	VCC3	Stage 3 collector voltage
5	VBIAS	Bias voltage	13	GND	Ground
6	PAEN	PA enable	14	VCC2	Stage 2 collector voltage
7	GND	Ground	15	GND	Ground
8	GND	Ground	16	VCC1	Stage 1 collector voltage

[1] The center ground pad must have a low inductance and low thermal resistance connection to the application's printed circuit board ground plane.
[2] External DC block is required.

Technical Description

The matching circuits are contained within the device. An on-chip active bias circuit is included within the device for both input and output stages, which provides excellent gain tracking over temperature and voltage variations.

The SKY66394-11 is internally matched for maximum output power and efficiency. The input and output stages are independently supplied using the VCC1, VCC2, and VCC3 supply lines (pins 16, 14, and 12, respectively). The DC control voltage that sets the bias is supplied by the VCBIAS signal (pin 5).

Electrical and Mechanical Specifications

The absolute maximum ratings of the SKY66394-11 are provided in Table 3. Recommended operating conditions are specified in Table 4 and electrical specifications are provided in Table 5.

Table 3. SKY66394-11 Absolute Maximum Ratings[1]

Parameter	Symbol	Minimum	Maximum	Units
RF input power (CW)	P_{IN}		+8	dBm
Supply voltage (VCC1, VCC2, VCC3, VBIAS)	V_{CC}		5.5	V
PA enable	V_{EN}		3	V
Operating temperature	T_C	-40	+100	°C
Storage temperature	T_{ST}	-55	+125	°C
Junction temperature	T_J		+150	°C
Power dissipation	P_D		1.9	W
Device thermal resistance	θ_{JC}		15	°C/W
Electrostatic discharge:	ESD			
Charged Device Model (CDM)			500	V
Human Body Model (HBM)			1000	V

[1] Exposure to maximum rating conditions for extended periods may reduce device reliability. There is no damage to device with only one parameter set at the limit and all other parameters set at or below their nominal value. Exceeding any of the limits listed here may result in permanent damage to the device.

ESD HANDLING: *Although this device is designed to be as robust as possible, electrostatic discharge (ESD) can damage this device. This device must be protected at all times from ESD when handling or transporting. Static charges may easily produce potentials of several kilovolts on the human body or equipment, which can discharge without detection. Industry-standard ESD handling precautions should be used at all times.*

Table 4. SKY66394-11 Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Units
Supply voltage (VCC1, VCC2, VCC3, VBIAS)	V_{CC1}, V_{CC2}, V_{CC3}, V_{BIAS}	4.75	5	5.25	V
PA enable:	PAEN				
ON		1.7	2.0	2.5	V
OFF			0	0.5	V
PA enable current	I_{ENABLE}		1	12	µA
Operating frequency	f	2110		2170	MHz
Operating temperature	T_C	-40	+25	+85	°C

Table 5. SKY66394-11 Electrical Specifications[1]
(V_{CC1} = V_{CC2} = V_{CC3} = V_{BIAS} = 5 V, PAEN = 2.0 V, f = 2140 MHz, T_C = +25 °C, Input/Output Load = 50 Ω, Unless Otherwise Noted)

Parameter	Symbol	Test Condition	Min	Typ	Max	Units
Small signal gain	|S21|	P_{IN} = –30 dBm	36	38		dB
Gain @ +28 dBm	|S21|@28dBm	P_{OUT} =28dBm	36	38		dB
Input return loss	|S11|	P_{IN} = –20 dBm	12	16		dB
Output return loss	|S22|	P_{IN} = –20 dBm	10	14		dB
Reverse isolation[2]	|S12|	P_{IN} = –30 dBm		50		dB
ACLR @ +28 dBm	ACLR	P_{OUT} = +28 dBm (40 MHz LTE, 8.5 dB PAR signal)		-28.5	-27	dBc
Saturated output power	P_{SAT}	CW, P_{IN} = +5 dBm	+35.5	+36.3		dBm
Output power at 3dB gain compression[3]	P3dB	CW, reference to Gain @ +28 dBm	+34.5	+35.3		dBm
2nd harmonic	2fo	CW, P_{OUT} = +28 dBm		-29	-25	dBc
3rd harmonic	3fo	CW, P_{OUT} = +28 dBm		-63	-55	dBc
Power-added efficiency	PAE	CW, P_{OUT} = +28 dBm	32	35		%
Quiescent current	I_{CCQ}	No RF signal		105	135	mA

[1] Performance is guaranteed only under the conditions listed in this table.
[2] Not test in production. Verified by design.
[3] Refer to the performance plot in Figure 7.

REFERENCIAS

[1] M. Tanio, N. Ishii and N. Kamiya, «Efficient Digital Predistortion Using Sparse Neural Network„» *IEEE Access,* vol. 8, pp. 117841-117852, 2020.

[2] M. Tanio, N. Ishii and N. Kamiya, «A Sparse Neural Network-Based Power Adaptive DPD Design and Its Hardware Implementation,» *IEEE Access,* vol. 10, pp. 114673-114682, 2022.

[3] J. Chani-Cahuana, P. N. Landin, C. Fager and T. Eriksson, «Iterative Learning Control for RF Power Amplifier Linearization,» *IEEE Transactions on Microwave Theory and Techniques,* vol. 64, nº 9, pp. 2778-2789, 2016.

[4] R. W. Chang, «Synthesis of band-limited orthogonal signals for multichannel data transmission,» *The Bell System Technical Journal,* vol. 45, nº 10, p. 1775–1796, 1996.

[5] C. Crespo-Cadenas, M. J. Madero-Ayora, J. Reina-Tosina, and J. Becerra-González, «Formal deduction of a volterra series model for complex-valued systems,» *Signal Processing,* vol. 131, p. 245–248, 2017.

[6] J. K. a. K. Konstantinou, «Digital predistortion of wideband signals based on power amplifier model with memory,» *IEEE Transactions on Signal Processing,* vol. 37, nº 23, pp. 1417-1418, 2001.

[7] D. Morgan, Z. Ma, J. Kim, M. Zierdt, and J. Pastalan, «A generalized memory polynomial model for digital predistortion of RF power amplifiers,» *IEEE Transactions on Signal Processing,* vol. 54, nº 10, pp. 3852-3860, 2006.

[8] E. Marqués-Valderrama, Esquema de linealización con arquitectura de aprendizaje directo y regularización de Ridge. (Trabajo Fin de Grado Inédito), Sevilla, 2021.

[9] M. V. d. Górgolas, Predistorsión digital en sistemas MIMO 2x2 (Trabajo Fin de Grado)., Sevilla, 2022.

[10] I. G. Mayorga, Desarrollo de aplicación basada en redes neuronales para detectar imágenes médicas., Sevilla, 2022.

[11] S. Cruces, Apuntes de la asignatura Procesamiento Avanzado de Señal en Comunicaciones. Master Universitario en Ingeniería de Telecomunicación (ETSI). Departamento de Teoría de la Señal y Telecomunicaciones. Universidad de Sevilla, Sevilla, 2022.

[12] J. J. M. Fuentes, Apuntes de la asignatura IA en Imagen, Audio y Video. Master Universitario en Ingeniería de Telecomunicación (ETSI). Departamento de Teoría de la Señal y Telecomunicaciones. Universidad de Sevilla, Sevilla, 2022.

[13] E. M. Valderrama, Predistorsión de amplificadores de potencia con técnicas de aprendizaje distribuido, Sevilla, 2023.