Trabajo de Fin de Grado

MODELOS DE REDES NEURONALES PARA EL ESTUDIO DE LA PROPAGACIÓN DE ONDAS ELECTROMAGNÉTICAS

Trabajo realizado por:

ALEJANDRO BERNÉ PÉREZ

Trabajo dirigido por:

SERGIO GUTIÉRREZ RODRIGO

LUIS MARTÍN MORENO

Curso académico 2022-2023

Índice

1. Introducción

En los últimos años, la inteligencia artificial ha ido ganando una gran importancia hasta el punto de ser prácticamente imprescindible en muchos aspectos de nuestras vidas cotidianas, aunque no seamos conscientes de su presencia en muchos casos. Su origen se remonta a los años 30 con Alan Turing, quien es considerado el padre de la inteligencia artificial (IA) al diseñar una máquina con el objetivo de desentrañar los códigos secretos de los nazis, pero ha tenido un enorme desarrollo hasta la actualidad. Hoy en día, están presentes en una amplia gama de áreas, desde los asistentes de voz presentes en nuestros móviles hasta la automatización de la vida cotidiana con los dispositivos robóticos que podemos encontrar en nuestras propias casas, aunque el uso más importante probablemente radica en el análisis de datos, un campo presente en prácticamente todas las empresas tecnológicas actuales.

La revolución más reciente ha venido de mano de ChatGPT, una IA especializada en el diálogo creada por OpenAI que puede responder de una forma realmente detallada a prácticamente cualquier pregunta que se le realice y que además es capaz de aprender de la información que recibe para mejorar su futuro rendimiento y forma de responder. Algunas personas creen que estos avances son excesivos, que vulneran nuestra privacidad y nuestros derechos y que no estamos preparados para que existan todavía, hasta el punto de que algunos gobiernos han llegado a prohibirlos en sus países o han puesto limitaciones y restricciones para su uso. Sin embargo, la realidad es que el campo de la IA no deja de avanzar, aún tiene mucho recorrido y ha venido para quedarse en nuestras vidas.

Respecto a lo que nos concierne, la IA también ha tenido un gran impacto en el ámbito científico, sobre todo en la física, donde ya está desarrollada hasta el punto de que está siendo utilizada para resolver problemas complejos y a realizar nuevos descubrimientos que habrían sido imposibles sin su ayuda [1]. Algunas de las áreas en las que está teniendo una gran importancia son la física de partículas, donde se usa para analizar una gran cantidad de datos del LHC, por ejemplo, con el objetivo de detectar nuevas partículas y entender las fuerzas fundamentales de la naturaleza; la astrofísica, donde se analizan datos obtenidos de telescopios y simulaciones para intentar descubrir los misterios del universo y encontrar nuevos planetas fuera del sistema solar; y la meteorología, donde sirve para realizar modelos cada vez más precisos de la atmósfera y del clima histórico en la Tierra de modo que las predicciones que obtengamos sean cada vez más precisas.

Aparte de todos estos usos, también se han encontrado otras aplicaciones, como intentar encontrar una forma distinta de entender la física que llevamos estudiando cientos de años [2]. Otro experimento que se ha realizado con una IA es el de entender el funcionamiento de un péndulo doble y encontrar las variables del sistema simplemente analizando un vídeo del sistema en movimiento. Según la mecánica que conocemos, las variables de las que depende el sistema son el ángulo y la velocidad angular de cada uno de los brazos, sin embargo, lo más sorprendente del resultado es que de las cuatro variables que ha encontrado la IA, dos son equivalentes a los ángulos de los brazos, pero las otras dos son completamente distintas, aunque igual de válidas para describir el comportamiento del sistema. Esto lleva a pensar que pueden existir distintas formas de entender la física y que hay numerosos fenómenos físicos que se escapan al conocimiento de los humanos, como por ejemplo efectos cuánticos que intentamos

explicar con la teoría de cuerdas, y que quizás en un futuro sea una IA la que consiga dar con una explicación más acertada que nosotros.

Lo que está claro es que la IA en la actualidad tiene numerosos beneficios [3] y nos ayuda en gran medida en nuestros experimentos y cálculos. Para empezar, proporciona una mayor precisión y eficiencia al tratar grandes cantidades de datos y al identificar patrones y comportamientos que previamente podían ser pasados por alto, lo que nos permite avanzar más rápidamente con nuestras investigaciones. Además, también resulta realmente útil para realizar simulaciones más exactas que nos ayudan a comprender mejor algunos fenómenos y relaciones existentes en la naturaleza. Sin embargo, no todo es perfecto, ya que todavía hay algunas limitaciones que tenemos que intentar mejorar. Algunas de ellas son el comportamiento limitado que tienen, ya que pueden realizar predicciones según el entrenamiento previo que han realizado pero no comprenden los fenómenos físicos complejos que existen detrás de los datos, o la falta de transparencia, debido a que algunos algoritmos internos son difíciles de comprender y por tanto de modificar y mejorar para los físicos que no están acostumbrados a trabajar con ellos. Es decir, a pesar de que son muy útiles en algunos campos, aún queda mucho trabajo que realizar para mejorar su trabajo y para hacerlos más accesibles a las personas que no están especializadas en ellos.

En cuanto al área del electromagnetismo, también existe una gran variedad de aplicaciones de la IA. En concreto, una de las más importantes es el método FDTD (siglas en inglés de *Finite-Domain Time-Difference*), que es en el que vamos a centrarnos a lo largo de este trabajo. Este método numérico fue introducido por Yee en 1966 [4] con el objetivo de resolver las ecuaciones de Maxwell en el dominio temporal, creando por medio su propio algoritmo para ello, y ha sido estudiado por numerosos científicos desde su aparición con el fin de mejorar su funcionamiento y aumentar el número de sistemas a los que se puede aplicar. Estos estudios han comprendido las condiciones de estabilidad del algoritmo y la extrapolación a soluciones para ondas electromagnéticas en distintos materiales en dos y tres dimensiones, entre otros, así como la creación de múltiples variantes del método para ciertos propósitos más concretos.

El método FDTD ha resultado de gran utilidad para el campo de la modelización computacional de la electrodinámica, ya que implementado en una amplia variedad de estudios. Algunas de las aplicaciones más importantes de esta lista son la medicina, donde se ha utilizado para intentar diseñar un radar de banda ancha que detecte el cáncer antes que los rayos X; el ejército, donde se han estudiado los peligros que las microondas enemigas pueden tener en los sistemas de un caza; o la investigación de distintos campos, como la fotónica, donde se ha analizado si los láseres más pequeños del mundo pueden alcanzar su punto de operación a temperatura ambiente.

En lo que a este trabajo concierne, nosotros lo hemos aplicado al estudio de la propagación del pulso de una onda electromagnética en un material, donde comprobaremos si una red neuronal es capaz de aprender el funcionamiento del método FDTD partiendo de un instante inicial para devolvernos el comportamiento completo del sistema a lo largo del tiempo, entre otras cosas.

2. Objetivos y metodología

El objetivo principal de este trabajo es la búsqueda de redes neuronales que puedan aprender y replicar el comportamiento del método FDTD.

Lo primero que habría que comentar es que para realizar este trabajo se ha partido de otro que se llevó a cabo previamente [5]. En él, se implementó el algoritmo FDTD para replicar el comportamiento de una onda electromagnética mediante las ecuaciones de Maxwell y posteriormente se creó una red neuronal para intentar conseguir que aprendiese cuál era el funcionamiento del sistema, obteniendo los coeficientes de las ecuaciones que lo rigen.

A partir del trabajo previo, hemos estudiado distintos aspectos para profundizar en el estudio del sistema. Nuestra investigación ha comprendido la implementación de distintas redes para observar si el comportamiento del sistema mejoraba o si el programa podía optimizarse de una mejor manera, y el estudio de la predicción temporal de la propagación de la onda mediante la red entrenada para ver si era capaz de imitar el método FDTD.

En cuanto a la estructura del trabajo, en el apartado 3 se introducen algunos aspectos fundamentales de las redes neuronales de los que se van a hablar de forma recurrente. En el cuarto apartado, se describirán los aspectos más importantes del modelo FDTD, que es la base sobre la que se cimienta el trabajo. A continuación, en el quinto apartado se mostrarán los resultados previos de los que se han partido, así como los obtenidos mediante las simulaciones posteriores realizadas para nuestra investigación. Finalmente, en el último apartado se comentarán las conclusiones a las que hemos llegado con nuestro estudio.

Respecto a la metodología, los códigos se han escrito en Python, utilizando para programar las redes neuronales la API Keras de Tensorflow. Todos ellos se pueden encontrar en el enlace de Github que aparece en la bibliografía [6].

3. Conceptos básicos sobre redes neuronales

En este apartado, vamos a definir lo que son las redes neuronales y explicaremos sus aspectos más importantes, ya que son conceptos que van a aparecer a lo largo del trabajo y es conveniente tener una noción de lo que quieren decir.

3.1. Neuronas

Para entender el funcionamiento de una neurona artificial, vamos a explicar previamente la estructura y la forma de trabajar de su homólogo real, la neurona biológica. Estas se encuentran en nuestro sistema nervioso central, mayoritariamente en el cerebro, y están formadas principalmente por tres componentes: el cuerpo o soma, donde está el núcleo y el resto de elementos que necesita la neurona para mantenerse viva; las dendritas, que son unas prolongaciones que reciben los impulsos nerviosos que emiten otras neuronas; y el axón, que es el encargado de transmitir el impulso nervioso generado al resto de neuronas.

Su función principal es la de recibir y transmitir información a través de nuestro organismo mediante los impulsos nerviosos que recibe y envía al resto. La unión mediante las neuronas se denomina sinapsis, lo que significa que no están conectadas de forma directa. De esta forma, la

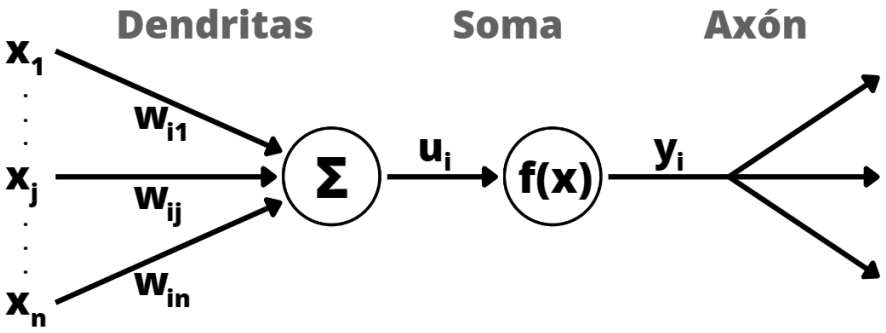


Figura 1: Esquema de una neurona artificial. En comparación con la unidad biológica, la entrada sería equivalente a las dendritas, el interior correspondería al soma y la salida sería el axón.

conexión se realizará mediante señales electroquímicas que recorrerán ese vacío, y la información siempre será transmitida en el mismo sentido. Además, debido a la gran ramificación que tienen, cada neurona puede estar conectada a otras miles.

Pasamos ahora a analizar cómo es una neurona artificial [7], representada en la Figura 1, que no difiere demasiado de lo que acabamos de ver para una biológica. En cuanto a su estructura, también podemos distinguir tres partes principales: la entrada, donde recibirá la información de las neuronas previas; el interior, donde se realizará alguna operación concreta; y la salida, en la que se transmitirá la información a las siguientes neuronas. En cada una de ellas habrá una función matemática que dependerá de los ciertos parámetros y que determinará el comportamiento de la neurona. En general y en nuestro caso concreto, consideraremos que la función de salida es la identidad, como muestra el dibujo.

Respecto a la función de entrada, dependerá de dos conjuntos de parámetros. El primero serán las señales procedentes de las neuronas de entrada, es decir, las neuronas cuyas salidas están conectadas a la entrada de esta, que vendrán representadas por x_j. El otro conjunto de parámetros serán los pesos w_{ij}, que indicarán la intensidad de la interacción entre nuestra neurona i y la de entrada j. Habitualmente, la función de entrada es la suma de los productos correspondientes de la forma $u_i(x) = \sum_j w_{ij} x_j$, que es la que hemos usado a lo largo del trabajo.

3.1.1. Función de activación

La última función que vamos a encontrar en nuestra neurona es la función de activación, que nos indicará si una neurona está activa o no, es decir, si está transmitiendo información. Esta función va a depender de la función de entrada y de un nuevo parámetro, el umbral o *bias* b. Este parámetro es una compensación (*offset*) que añadiremos a la función de entrada $u_i(x)$, de modo que la suma de ambas contribuciones será el argumento de la función de activación de la forma $f_i(x) = u_i(x) + b$. Durante la mayor parte del trabajo, estableceremos el umbral en cero.

En la mayoría de ocasiones, las funciones de activación que se implementan son monótonas crecientes y continuas. Además, si el umbral es cero, suelen ser nulas en su parte negativa (la neurona está inactiva) y positivas en el resto del dominio (la neurona transmite información). Teniendo en cuenta que prácticamente siempre la función de salida es la unidad, será la función de activación la que también veamos en el *output* de la neurona, $y_i(x) = f_i(x)$. Algunas de las funciones de activación más habituales son la ReLU (*Rectified Linear Unit*, que es nula antes del origen y lineal de pendiente unidad para valores positivos), la tangente hiperbólica o la sigmoide.

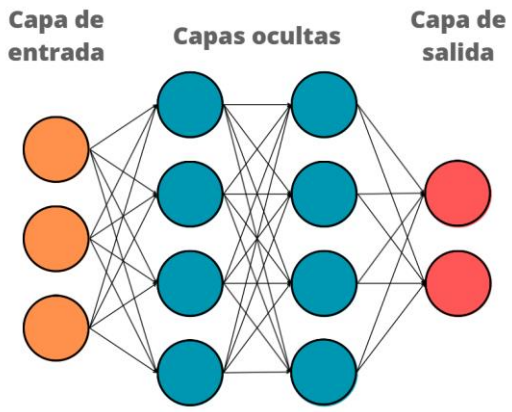


Figura 2: Ejemplo de red neuronal densa, compuesta por una capa de entrada con tres neuronas, dos capas ocultas de cuatro neuronas y una capa de salida con dos neuronas.

3.2. Redes neuronales

Una vez vista la estructura de una neurona artificial, las podremos juntar para formar redes neuronales que procesen el comportamiento de un sistema complejo inspiradas en el funcionamiento del cerebro humano. Las neuronas se suelen agrupar en unidades estructurales denominadas capas, dentro de las cuales todas ellas serán del mismo tipo. Podemos distinguir entre tres tipos de capas: la capa de entrada, donde se introduce la información directamente; las capas ocultas, que son las capas intermedias que procesan los datos, pudiendo haber tantas como sea necesario; y la capa de salida, de donde obtendremos los resultados tras haber sido analizados. No todas las capas deberán tener la misma cantidad de neuronas ni tampoco la misma función de activación, ya que tendremos que adaptar estos parámetros a nuestro problema concreto, así como el número de capas que creemos.

En cuanto a los tipos de redes neuronales, se han creado una gran variedad de ellas. La más simple es la red neuronal monocapa, que únicamente tiene una capa de entrada y una de salida, aunque la más habitual es la multicapa, donde sí se tienen varias capas, denominadas capas ocultas. A lo largo de este trabajo hemos utilizado redes neuronales densas, que son redes multicapa donde las neuronas de cada capa están conectadas con todas las de las capas colindantes, como la que aparece de ejemplo en la Figura 2. Otras redes importantes son las recurrentes, unas redes multicapa donde existen lazos de retroalimentación entre neuronas y que tienen un componente de memoria a corto plazo para recordar comportamientos que han reconocido anteriormente, o las convolucionales, que se inspira en el aprendizaje visual del cerebro y son muy útiles para el reconocimiento de imágenes.

3.3. Entrenamiento de la red

Tras haber explicado cómo es la estructura de una red neuronal, vamos a ver cómo se entrena y cuáles son las herramientas más importantes que aparecen en este proceso [3, 8-11].

3.3.1. Conjuntos de datos

Lo primero que se debe tener es un conjunto de datos, ya sean experimentales o creados de forma artificial, con los que podamos entrenar a la red para que aprenda cuál es el comportamiento del sistema que tenemos. Estos datos se dividen en distintos conjuntos, cada

uno de los cuales tendrá una función concreta. El primero de ellos es el conjunto de entrenamiento (*training set*), que es el que se usa para que la red neuronal aprenda los patrones que existen entre los datos y establezca cuáles son los pesos y los umbrales que existen. El segundo es el conjunto de validación (*validation set*), que debe tener datos distintos al primero y nos ayuda a ver cómo ha aprendido la red el comportamiento del sistema tras el entrenamiento. Este conjunto también nos sirve para encontrar los mejores hiperparámetros para nuestra red y evitar el sobreajuste, lo que explicaremos en profundidad en el tramo final del apartado. El último conjunto de datos es el de prueba (*test set*), que se utiliza al final del todo para comprobar lo buena que es la red tras haberla entrenado y haber encontrado los mejores hiperparámetros. Este tercer conjunto es prescindible y se puede entrenar la red únicamente con los dos primeros, como hemos hecho nosotros a lo largo de nuestra investigación.

3.3.2. Función de coste

La función de coste es la manera más sencilla que tenemos de cuantificar lo buena que es la red al predecir los resultados del sistema. En términos generales, va a medir la diferencia que hay entre los resultados reales y los obtenidos mediante el entrenamiento de la red, por lo que es necesario conocer cuáles son los valores reales que habría que obtener para nuestro sistema para poder calcularla. Esto se denomina método de aprendizaje supervisado, ya que estamos utilizando unos datos de entrenamiento para enseñar a la red a que produzca la salida deseada. Nuestro objetivo será minimizar la función de coste lo máximo posible, ya que eso significará que los resultados dados por la red son muy similares a los reales, y esto se hará mediante los datos de validación. Por tanto, será nuestro trabajo encontrar los hiperparámetros más adecuados para que este resultado sea lo mejor posible.

Hay muchas funciones de coste existentes que dan buenos resultados, aunque la más habitual y la que hemos utilizado nosotros se denomina error cuadrático medio (MSE, *Mean Squared Error*), que viene dada por la siguiente expresión:

$$J = \frac{1}{n}\sum_{i}(y_i - \hat{y}_i)^2 \tag{1}$$

donde n es el número de predicciones realizadas, $\hat{y}_i$ es el valor predicho por la red y y_i es el valor real al que nos queremos aproximar.

3.3.3. Descenso por gradiente y método de backpropagation

Existen distintos métodos para calcular el mínimo de la función de coste y aproximarnos a él, aunque el más conocido es el descenso por gradiente (*gradient descent*). Consiste en empezar con una elección arbitraria de parámetros que nos darán como resultado una función de coste, desde donde habrá que averiguar en qué dirección esta función decrece en mayor medida al cambiar sus parámetros (pesos y umbrales) para ir en esa dirección. La forma de hacerlo será calculando el gradiente de la función de coste respecto de todos sus parámetros, de modo que obtenemos la pendiente de la función en todos los ejes. Escribiendo los parámetros de los que depende la función de coste como v_i, la expresión del gradiente vendrá dada por:

$$\nabla J = \left(\frac{\partial J}{\partial v_1}, \frac{\partial J}{\partial v_2}, \dots, \frac{\partial J}{\partial v_n}\right) \tag{2}$$

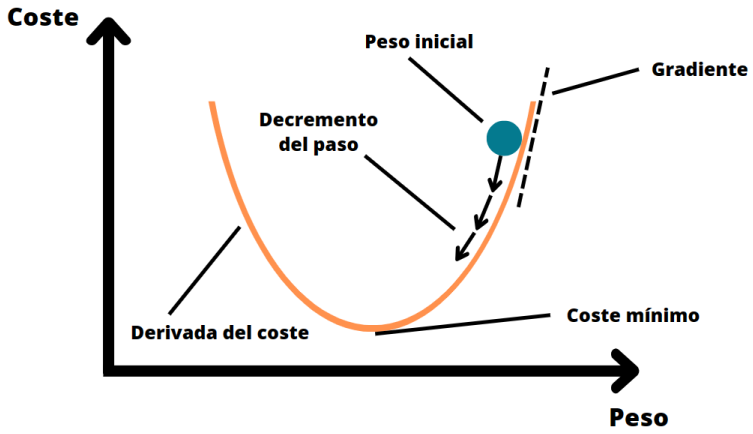


Figura 3: Descenso por gradiente con una función de coste de una sola variable, donde se busca su mínimo haciendo pequeños saltos desde un peso inicial. En el caso de una función multivariable, la curva pasaría a ser una hipersuperficie de varias dimensiones.

La siguiente pregunta que cabe hacerse es cuál tiene que ser el tamaño del paso que hagamos en cada iteración en la dirección que hemos encontrado apropiada. Esta consideración es importante, ya que un paso pequeño puede hacer que el tiempo necesario para llegar al mínimo sea excesivo, y un paso grande puede provocar que nos saltemos el mínimo o incluso nos desplacemos hacia zonas donde la función de coste tiene un valor mayor. Esto se va a controlar con un hiperparámetro denominado tasa de aprendizaje α (*learning rate*), que nosotros podemos establecer para que tome un valor adecuado a nuestro problema. Una elección apropiada es determinar que la variación que se va a hacer en los parámetros es:

$$\Delta v = -\alpha \cdot \nabla J \tag{3}$$

De esta forma, el cambio que tendremos en la función de coste siempre será negativo, por lo que será como si la estuviéramos dirigiendo hacia el mínimo que nos interesa encontrar. Iterando este proceso, esperamos alcanzar el mínimo de la función de coste, como se representa en la Figura 3. No obstante, deberemos tener cuidado de asegurarnos de encontrar el mínimo absoluto y no quedarnos en un mínimo local del que no estemos consiguiendo salir.

Antes de pasar al siguiente concepto, vamos a explicar (por encima, ya que es un proceso de bastante complejidad) qué es el método de backpropagation, que es esencial para el cálculo del gradiente de la función de coste. Este método consiste en calcular el error cometido en la capa de salida y propaga el error hacia atrás a través de todas las capas utilizando la regla de la cadena, por lo que guarda información de todos los parámetros de la red. De esta forma, es capaz de detectar cuáles son los nodos que producen un mayor error para solucionarlo en futuras iteraciones ajustando sus parámetros para reducir la función de coste. Este método es fundamental en el entrenamiento de las redes neuronales no solo por ayudar a minimizar el error, sino porque implementa una forma de calcular el gradiente que ahorra una gran cantidad de tiempo de simulación.

3.3.4. Optimizadores

A pesar de lo útil que puede parecer de primeras el método del descenso por gradiente, lo que encontramos sin embargo es un proceso demasiado lento, ya que una sola muestra de nuestros datos puede requerir el análisis de millones de operaciones. Esto se denomina

descenso por gradiente de lote (*batch gradient descent*). Se puede demostrar que con un subconjunto de puntos suficientemente grande se puede aproximar el gradiente por completo, de modo que no solo nos ahorramos una gran cantidad de tiempo, sino que además mejoramos el problema de quedarnos en un mínimo local.

Es aquí donde introducimos el concepto de optimizadores, que son algoritmos que va a intentar obtener los mejores parámetros para que el error cometido por la red sea lo más pequeño posible. El más habitual y el principal que soluciona el problema que acabamos de comentar es el denominado descenso por gradiente estocástico (*Stochastic Gradient Descent*, SGD). Este optimizador consiste en mezclar de forma aleatoria los datos que tenemos y dividirlos en pequeños grupos (*mini-batches*) de un tamaño apropiado como para que la aproximación del gradiente sea buena. Tras analizar todos los grupos en los que hemos dividido nuestros datos se dice que hemos completado una época de entrenamiento, y el resultado es mejor y más rápido que el que teníamos con el descenso por gradiente original.

Aparte de este optimizador, existen muchos otros, cada uno con sus propias características. Así como el SGD necesita que introduzcamos manualmente una tasa de aprendizaje que pueda parecer correcta, hay otros como el Adagrad, que intenta adaptarla a los parámetros de la red y al gradiente que hay en cada momento. Otros, en cambio, tienen memoria sobre los gradientes que había en el paso anterior y no modifica por completo los pesos basado en los datos de la iteración actual, como es el caso del Adam.

3.3.5. Sobreajuste e hiperparámetros

Hay veces que la red no termina de encontrar las relaciones correctas entre los datos con los que entrena, dando lugar a dos posibles situaciones. La primera de ella es el sobreajuste u *overfitting*, que describe la situación en la que el modelo prácticamente solo ha aprendido el funcionamiento de los datos de entrenamiento, pero no es capaz de generalizar el comportamiento a datos nuevos de forma correcta. El otro caso es el contrario, el *underfitting*, donde la red no ha conseguido aprender los patrones existentes en los datos y por tanto tampoco es capaz de predecir correctamente lo que ocurrirá con un conjunto nuevo de datos. Este último es más fácil de solucionar, ya que muchas veces solo se necesita aumentar el tiempo de simulación o la complejidad de la red.

Sin embargo, el *overfitting* tiene una mayor complicación, por lo que es en el que vamos a centrar nuestra atención. El mayor problema del sobreajuste viene porque la red no solo aprende el comportamiento de los datos, sino que también puede ocurrir que capture el ruido de fondo. Algunas veces este efecto se puede deber a que la complejidad de la red es demasiado grande, pero en la mayoría de casos se quiere luchar contra el sobreajuste manteniendo la potencia de nuestros modelos. Una de las soluciones que se implementan habitualmente es la que hemos comentado al comienzo: se separan los datos entre tres conjuntos distintos, de forma que con los dos primeros se entrena a la red y se obtienen los hiperparámetros más adecuados, y con el tercero, el de prueba, se comprueba que los resultados proporcionados por la red sean correctos.

Y de nuevo vuelve a aparecer el concepto de hiperparámetros, que ahora sí vamos a desarrollar. Su denominación viene para diferenciarlos de los parámetros propios de la red, los pesos y los umbrales, ya que estos los vamos a poder modificar nosotros con el fin de encontrar

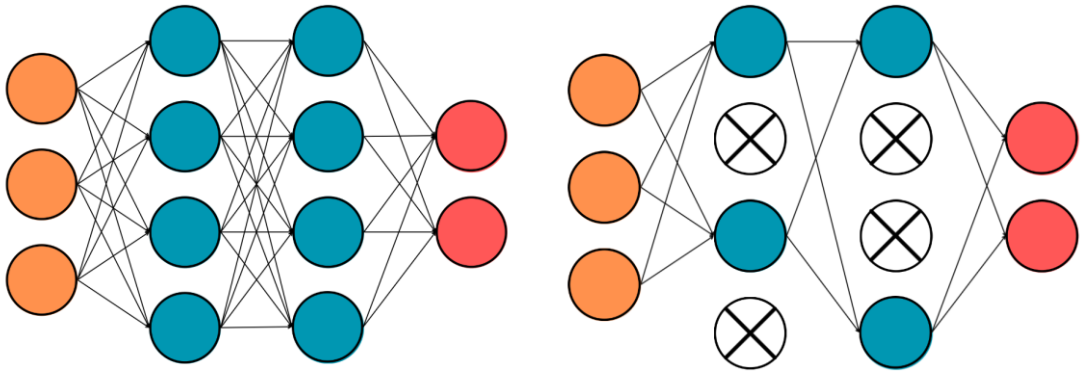


Figura 4: Efecto de la técnica de regularización de *Dropout*. En la segunda red, se ha aplicado un *Dropout* del 50% de las neuronas en las dos capas ocultas.

la mejor configuración para nuestra red. A lo largo de este apartado, hemos nombrado una gran cantidad de hiperparámetros que pueden cambiar en gran medida el funcionamiento de nuestro modelo. Algunos ejemplos son las funciones de activación de las neuronas, la cantidad y el tamaño de las capas que establecemos en la red, la tasa de aprendizaje, el tamaño de los *mini-batches* del optimizador o el propio optimizador. Una correcta elección de estos hiperparámetros puede ayudar a prevenir el sobreajuste, a ahorrarnos tiempo de simulación y a conseguir que la red nos proporcione el mejor resultado posible.

3.3.6. Regularizadores

Aparte de elegir los parámetros, para lo cual no hay una fórmula universal, la regularización es fundamental para evitar el sobreajuste. Esta técnica consiste en imponer algunas restricciones en nuestra red para que consiga generalizar mejor el comportamiento de nuevos datos, sin penalizar la potencia de nuestro modelo. Existen distintas técnicas de regularización, como el L1, el L2 o el aumento de datos, pero nosotros vamos a centrarnos únicamente en dos, las que vamos a utilizar en algunas ocasiones a lo largo de este trabajo.

La primera de ellas es la desactivación, comúnmente llamada *Dropout*. Esta técnica de regularización consiste en desactivar un porcentaje establecido (habitualmente entre el 20% y el 50%) de neuronas de forma aleatoria al comienzo de cada época de entrenamiento, como muestra la Figura 4. De este modo, se consigue que ninguna neurona memorice parte de la entrada y que la red no dependa siempre de las mismas neuronas, ya que pueden no estar siempre disponibles. Esto ayuda a que la red encuentre una forma más equilibrada de entrenar, lo que ayuda a combatir el sobreajuste.

La segunda es normalización por lotes o *Batch Normalization*. Esta técnica consiste en añadir un paso extra cuando tenemos una función de activación con el objetivo de normalizar la entrada de cada capa de la red. Para ello, se calcula la media y la varianza de cada *mini-batch* antes de cada capa, ya que cada neurona se habrá normalizado de forma independiente, y después se normalizan los valores entrantes mediante estos parámetros. De esta forma, en cada época tendremos unas magnitudes ligeramente distintas que dependerán del lote para normalizar la red, lo que hará que la red sea más robusta frente a variaciones en la entrada y ayudará a prevenir el sobreajuste.

4. El método FDTD

Una de las formas más sencillas que existen para resolver ecuaciones diferenciales consiste en utilizar métodos que sustituyan los operadores diferenciales por operadores en diferencias finitas, de forma que nos permiten sustituir un problema diferencial por otro algebraico más fácil de estudiar. Uno de estos métodos, que nos permite resolver las ecuaciones de Maxwell en forma diferencial es el método FDTD (*Finite-Domain Time-Difference* o Diferencias Finitas en el Dominio Temporal), que es el que vamos a utilizar a lo largo de este trabajo.

4.1. Las ecuaciones de Maxwell

Desde un punto de vista macroscópico, el campo electromagnético se rige por las ecuaciones de Maxwell, cuya expresión diferencial en el dominio temporal para un medio lineal, homogéneo e isótropo sin pérdidas viene dada por:

$$\frac{\partial \vec{H}(\vec{r},t)}{\partial t} = -\frac{1}{\mu_0 \mu_r} \nabla \times \vec{E}(\vec{r},t) \tag{4}$$

$$\frac{\partial \vec{E}(\vec{r},t)}{\partial t} = \frac{1}{\varepsilon_0 \varepsilon_r} \nabla \times \vec{H}(\vec{r},t) \tag{5}$$

donde $\vec{H}$ es la intensidad de campo magnético, $\vec{E}$ es la de campo eléctrico, ε_r es la permitividad relativa, ε_0 es la del vacío, μ_r es la permeabilidad relativa y μ_0 es la del vacío.

Como se puede ver a simple vista, la variación temporal del campo magnético depende de la variación espacial del campo magnético, y viceversa. Desarrollando los vectores en sus componentes cartesianas, obtendremos un sistema descrito mediante seis ecuaciones en derivadas parciales. Sin embargo, en este trabajo nos vamos a restringir a la propagación de una onda en una sola dimensión, por lo que reduciremos este número a dos únicas ecuaciones, como veremos más adelante.

4.2. Discretización de ecuaciones diferenciales en diferencias finitas

Antes de entrar en materia con el método FDTD, vamos a explicar en qué consiste el método de diferencias finitas [12], ya que es de gran importancia para el mismo. Este método es una técnica numérica en la que la ecuación diferencial original se resuelve de forma aproximada en un conjunto finito de puntos, de forma que estamos discretizando nuestro problema en nudos de una red. Es lógico razonar que cuanto menor sea el discretizado, más nos aproximaremos al caso continuo, es decir, la solución del problema algebraico tenderá a la del diferencial.

El método de diferencias finitas se fundamenta en el desarrollo en serie de potencias de Taylor, que nos permite predecir el comportamiento de una función con tan solo conocer su valor y el de sus derivadas en un punto. Utilizando el desarrollo de Taylor para una función continua de varias variables, podemos obtener que su derivada parcial en la dirección x se puede aproximar como:

$$\frac{\partial U(x,y,z,t)}{\partial x} = \frac{U\left(x+\frac{\Delta x}{2},y,z,t\right) - U\left(x-\frac{\Delta x}{2},y,z,t\right)}{\Delta x} + O(\Delta x^2) \tag{6}$$

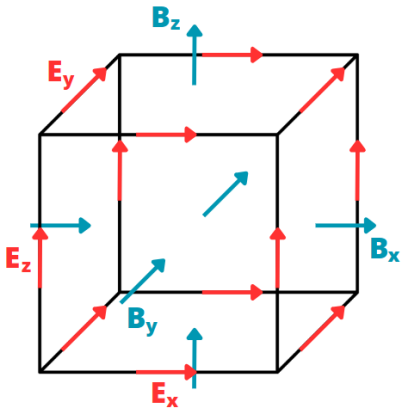


Figura 5: Representación de la celda de Yee.

4.3. Fundamentos del método FDTD

El método FDTD, introducido por K. S. Yee en 1966 [4], consiste en la sustitución de las derivadas parciales que aparecen en las ecuaciones de Maxwell por cocientes de diferencias finitas, de forma que el problema queda formulado en un sistema de ecuaciones lineales. Esto lo convierte en un método realmente útil en una gran variedad de tareas, como el electromagnetismo, ya que permite obtener mucha información de un sistema complejo mediante un proceso sencillo y rápido, comparado con otros métodos. Para llevar a cabo esta transformación diferencial en algebraica, se discretiza el espacio-tiempo mediante una malla, la más habitual de las cuales es la celda de Yee, cuya forma se puede ver en la Figura *5*.

Notemos que el campo eléctrico se encuentra en las aristas de la celda, mientras que el campo magnético está en las caras, por lo que existe un desfase entre ellas que nos ayuda a la hora de razonar la dependencia espacial del campo electromagnético. En cuanto al dominio temporal, Yee implementó una idea: situó el campo eléctrico en variaciones enteras del tiempo y el magnético en diferencias semienteras, como se puede observar en la Figura 6, de forma que se mantenía esta idea de la celda espacial también para el tiempo. Esto es lo que comúnmente se conoce como salto de rana (*leap-frog*) y nos permite calcular el campo eléctrico en un nodo de la malla para un tiempo t usando su valor para un tiempo anterior $t - \Delta t$ y las componentes del campo magnético en los nodos adyacentes en el instante $t - \Delta t/2$, y lo mismo sucedería extrapolado al campo magnético. De este modo, solo se necesitan unas condiciones iniciales para calcular la evolución temporal del campo electromagnético en todo el espacio.

Una vez comprendido como se ha discretizado el sistema, ya podemos escribir cuáles son las ecuaciones que van a regir su comportamiento. Para ello, vamos a aproximar las derivadas que aparecen en las ecuaciones de Maxwell mediante el método de discretización en diferencias finitas que se ha comentado previamente. En nuestro caso tendremos una onda que se propaga en una sola dirección, por lo que las ecuaciones para la evolución del campo eléctrico y magnético se pueden obtener de forma prácticamente directa:

$$E_y^{n+1}(x) = E_y^n(x) - \frac{\Delta t}{\mu_0 \varepsilon_0} \frac{B_z^{n+1/2}(x + \Delta x/2) - B_z^{n+1/2}(x - \Delta x/2)}{\Delta x} \tag{7}$$

$$B_z^{n+1/2}(x + \Delta x/2) = B_z^{n-1/2}(x + \Delta x/2) - \Delta t \frac{E_y^n(x + \Delta x) - E_y^n(x)}{\Delta x} \tag{8}$$

donde hemos denominado n al instante temporal cuya evolución buscamos conocer.

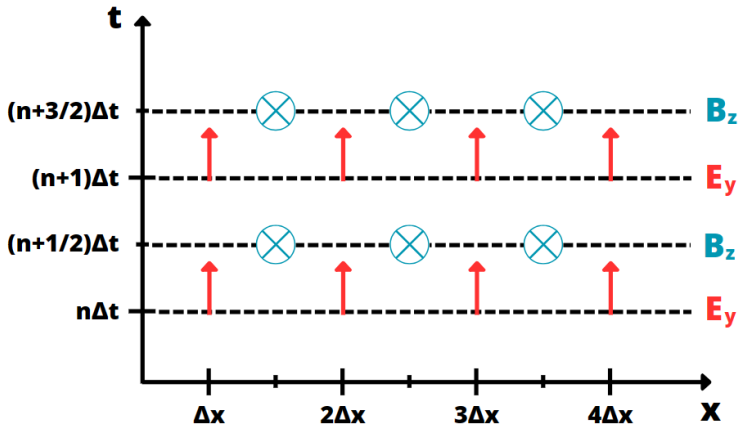


Figura 6: Esquema del método FDTD en 1D.

Cabe destacar que hemos sustituido el vector campo magnético por el vector inducción magnética porque en todo momento vamos a suponer que nuestra onda se está propagando por el vacío, de forma que siempre utilizaremos la permeabilidad magnética correspondiente.

4.4. Características del método FDTD

Una vez explicado el funcionamiento del método utilizado, vamos a comentar algunas de sus características fundamentales, tanto generales como relacionadas con nuestro trabajo concreto.

4.4.1. Exactitud y consistencia

El grado de exactitud de la aproximación que hemos realizado al convertir el sistema diferencial en algebraico suele expresarse en función del denominado error local de truncamiento, que nos indicará hasta qué punto las soluciones de la ecuación diferencial satisfacen la ecuación en diferencias finitas. En este caso, el error de truncamiento será de segundo orden, como ya hemos visto, y vendrá dado por:

$$O(\Delta x^2) = -\frac{1}{3!}\left(\frac{\Delta x}{2}\right)^2 \frac{d^3U(x)}{dx^3} - \frac{1}{5!}\left(\frac{\Delta x}{2}\right)^4 \frac{d^5U(x)}{dx^5} - \cdots \tag{9}$$

En cuanto a la consistencia, se dice que un esquema es consistente cuando el error local de truncamiento tiende a cero a medida que Δx también lo hace, lo que se puede ver que se cumple en la anterior ecuación. Esto nos garantiza que la ecuación en diferencias se aproxima correctamente a la ecuación diferencial deseada, es decir, es consistente con ella.

4.4.2. Estabilidad

La estabilidad numérica no tiene que ver con la ecuación diferencial de la que proviene, sino con la resolución numérica de la ecuación en diferencias finitas, e indica el comportamiento de los errores numéricos cometidos al resolverla. Aunque la ecuación tiene una solución exacta, en la práctica se cometen errores numéricos que pueden atenuarse o no, de forma que el esquema se considerará estable o inestable, respectivamente.

Para conocer la estabilidad de nuestro sistema, tendremos que encontrar la condición que tienen que cumplir los intervalos de discretización. En este caso, el tamaño de la celda deberá ser suficientemente pequeño como para que los campos no cambien drásticamente de un nodo

a otro. La condición que deberán satisfacer en general los parámetros de la celda para nuestro sistema en una dimensión es la denominada condición de Courant-Friedrichs-Lewy (también conocida como condición CFL):

$$S = \frac{c\Delta t}{\Delta x} \leq 1 \tag{10}$$

donde c es el valor de la velocidad de fase, y el parámetro S se conoce como factor de estabilidad y debe ser menor o igual que uno para garantizar la estabilidad del sistema. Como es lógico, nuestra elección de parámetros de discretización temporal y espacial serán siempre tales que nos encontremos en el caso estable.

4.4.3. Condiciones de contorno

Por último, vamos a hablar de las condiciones de contorno que hemos implementado en nuestro sistema. Las más habituales son las denominadas PML (*Perfectly Matched Layer*), mediante las cuales se simula que en los extremos existe un material absorbente que no refleja la onda recibida. Sin embargo, las que nosotros hemos usado son las PEC (*Perfect Electrical Conductor*), que consisten en anular las componentes del campo eléctrico (y por tanto del magnético) en los extremos del material, de forma que se simula el comportamiento que presenta un conductor perfecto.

5. Resultados

Lo primero que hemos hecho es reproducir los resultados que se obtuvieron en el trabajo anterior [5] para entender por completo el código del que hemos partido y comprobar antes de nada que el funcionamiento de todo el sistema es correcto.

Para empezar, se crearon numerosos ficheros que contenían la información de ondas electromagnéticas que se propagarían en el vacío. Estas ondas corresponderían a pulsos gaussianos y se crearían mediante el método FDTD comentado en el apartado anterior con el objetivo de servir de entrenamiento para nuestra red. La expresión general del campo eléctrico de estas ondas vendría dada por:

$$E_y(x) = E_0 e^{-\left(\frac{x-x_0}{D}\right)^2} e^{i\frac{\omega_0}{c}(x-x_0)} \tag{11}$$

donde D es la anchura del pulso, x_0 es la posición del máximo y ω_0 es la frecuencia en la que está centrado.

El cálculo del campo magnético se realizaría mediante la relación $B = E/c$, donde recordemos que $c = 1/\sqrt{\varepsilon_0\mu_0}$. Sin embargo, es necesario hacer una normalización para facilitar a la red el cálculo del error cuadrático medio. Esta normalización consiste en utilizar un campo magnético efectivo $\hat{B} = c \cdot B$, de forma que sea del mismo orden que el campo eléctrico.

Por tanto, vamos a iluminar nuestro material en el instante inicial con un paquete de ondas gaussiano de la forma descrita. La incidencia de esta onda es normal y parte del origen para propagarse hacia la derecha siguiendo las condiciones de contorno PEC, las cuales ya han sido descritas previamente. Un ejemplo de la propagación de un pulso se puede ver en la Figura *7*.

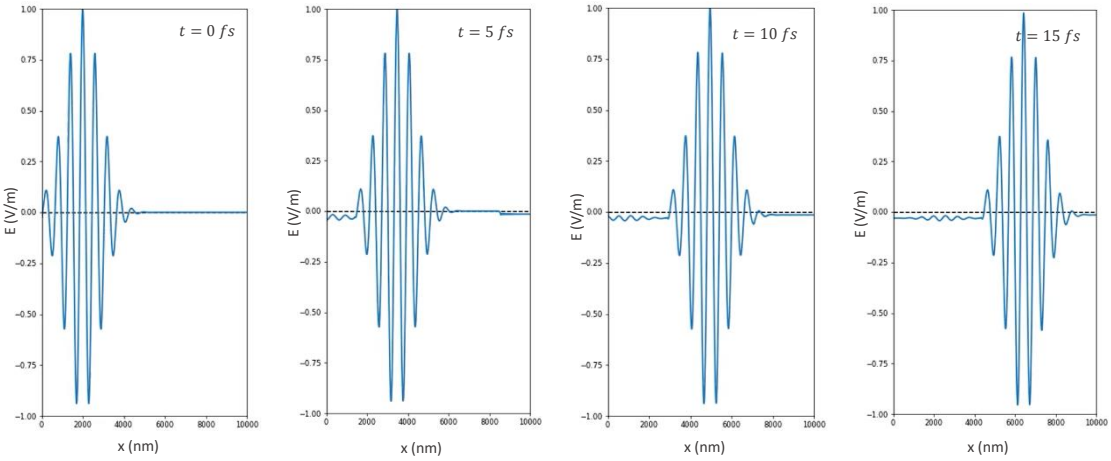


Figura 7: Propagación de una onda creada mediante el método FDTD. En la figura se ven cuatro capturas en instantes de tiempo distintos que muestran el avance de un pulso gaussiano con longitud de onda $\lambda = 780\ nm$ y anchura $D = 125\ nm$.

En cuanto a los parámetros que se han utilizado, las longitudes de onda que vamos a utilizar se encuentran dentro del rango del espectro visible, entre 380 y 780 nm, y las anchuras espaciales estarán entre 50 y 150 nm. La malla tiene una longitud de 10 µm, el discretizado espacial es de 10 nm y el temporal de unos 0.017 fs.

A continuación, se creó una red neuronal densa de tres capas que tenía 4 neuronas en la capa de entrada, 2 en la capa oculta y otras 2 en la de salida, y se la entrenó con los datos creados previamente. El número de neuronas elegido no fue al azar: se introduce en cada neurona inicial una componente del campo del instante de tiempo en el que nos encontramos y en las neuronas de salida obtenemos los campos en el instante posterior, como se muestra en la Figura 8. Posteriormente, se le pasó un fichero con datos nuevos para ver si era capaz de reproducir el comportamiento del pulso tras el aprendizaje con los datos de entrenamiento.

Tras haber comprobado que su funcionamiento era correcto, se calcularon los coeficientes de los campos de las ecuaciones (7) y (8). Antes de nada, vamos a escribir dichas ecuaciones en forma matricial para poder comparar de una forma más sencilla los resultados que vamos a obtener:

$$\begin{pmatrix} E_{y(i)}^{n+1} \\ B_{z(i)}^{n+3/2} \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 & -\frac{c\Delta t}{\Delta x} & \frac{c\Delta t}{\Delta x} \\ \frac{c\Delta t}{\Delta x} & -\frac{c\Delta t}{\Delta x} & 1.0 & 0.0 \end{pmatrix} \begin{pmatrix} E_{y(i)}^{n} \\ E_{y(i+1)}^{n} \\ B_{z(i)}^{n+1/2} \\ B_{z(i-1)}^{n+1/2} \end{pmatrix} \tag{12}$$

Notemos que los coeficientes cruzados han sido ligeramente modificados de cómo aparecían en las expresiones originales al haber introducido la velocidad de la luz c por simplicidad. En cuanto a las unidades, recordemos que se ha utilizado también esta velocidad para realizar una normalización del campo magnético, de ahí la forma de los coeficientes. Según nuestra elección de los parámetros de discretización espacial Δx y temporal Δt, estos coeficientes tomarán un valor de $c\Delta t/\Delta x = 0.519615\ldots$, con su correspondiente signo.

Para calcular estos coeficientes mediante el entrenamiento de la red, se sacaron los pesos que asignó la red a cada neurona y se multiplicaron, obteniendo los siguientes resultados:

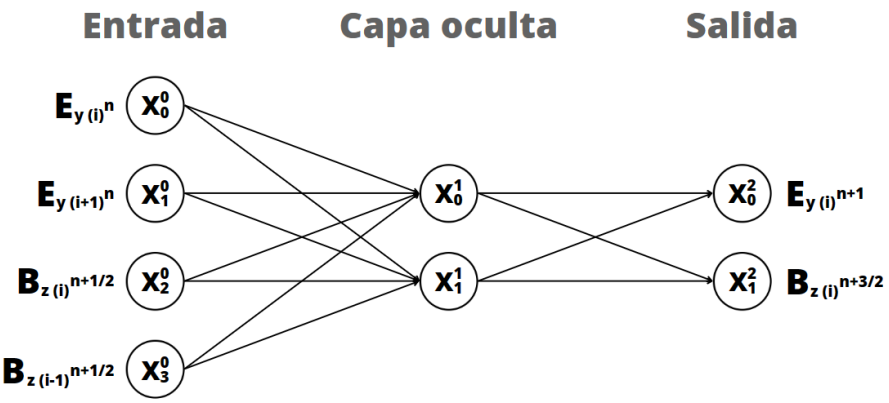


Figura 8: Primer modelo de red neuronal densa propuesto, donde se introducen las cuatro componentes de los campos para obtener la evolución de ambos.

$$\left[\begin{pmatrix} W_{0,0}^{0,1} & W_{0,1}^{0,1} \\ W_{1,0}^{0,1} & W_{1,1}^{0,1} \\ W_{2,0}^{0,1} & W_{2,1}^{0,1} \\ W_{3,0}^{0,1} & W_{3,1}^{0,1} \end{pmatrix} \begin{pmatrix} W_{0,0}^{1,2} & W_{0,1}^{1,2} \\ W_{1,0}^{1,2} & W_{1,1}^{1,2} \end{pmatrix}\right]^{T} = \begin{pmatrix} 0.9988 & 0.0011 & -0.5181 & 0.5181 \\ 0.5180 & -0.5184 & 0.9903 & 0.0053 \end{pmatrix} \quad (13)$$

El mejor resultado que se obtuvo de estas simulaciones proporcionó un error cuadrático medio de $MSE = 6.2 \cdot 10^{-7}$, lo cual ya se consideró suficientemente bueno como para poder decir que la red había aprendido a resolver correctamente las ecuaciones de Maxwell en una dimensión con el método FDTD.

Para finalizar, se sustituyeron los coeficientes obtenidos con la red por los reales en el método FDTD programado inicialmente para ver si el pulso creado era correcto o daba algún problema. Lo que se encontró no fue una réplica del pulso creado por los coeficientes reales, sino uno cuya amplitud decrecía conforme avanzaba en el tiempo.

Una vez hemos explicado los conceptos básicos que había que saber sobre redes neuronales, el funcionamiento del método FDTD y los resultados iniciales, podemos pasar a discutir las alternativas al primer modelo. Este proyecto se ha cimentado en tres bloques principales: la optimización de la red que se utilizó inicialmente, la evolución del sistema predicha directamente por la red y la búsqueda de otros tipos de redes más complejas que también aprendan el comportamiento del sistema.

5.1. Optimización de la red inicial

Lo primero en lo que vamos a centrar nuestra atención es la optimización de la red con la que se había trabajado en un principio para poder minimizar el error y mejorar sus predicciones.

Una de las ideas va a ser la adición de capas intermedias para ver si la red es capaz de representar mejor el comportamiento del sistema, al tener un mayor número de neuronas y al aumentar su complejidad. Sin embargo, el valor de la función de error permanece prácticamente igual, por lo que se deduce que la modificación de este hiperparámetro no ayuda a que la red aprenda mejor el comportamiento del sistema.

Otra de las ideas que se va a llevar a cabo, recuperando la red inicial por optimización de recursos tras no haber conseguido una mejoría, es la introducción de funciones de activación en

la capa oculta. Probaremos las tres funciones que se comentaron al comienzo del trabajo: la ReLU, la tangente hiperbólica y la sigmoide, ya que son tres funciones muy habituales con comportamientos distintos entre sí, y en todos los casos se va a añadir una capa de *Batch Normalization* para normalizar los datos de entrada. El resultado de la implementación de la ReLU resulta ser prácticamente imperceptible, probablemente debido a que la forma de la función no modifica apenas los datos que tenemos. Por otro lado, la introducción de la tangente hiperbólica y la sigmoide provoca un gran aumento del error cuadrático medio, ya que la red deja de captar correctamente el comportamiento del sistema y da como resultado predicciones totalmente distintas a lo que debería y hacía antes. Junto con la implementación de funciones de activación, también se va a probar en alguna ocasión a activar los umbrales para ver si hay una mejoría, pero tampoco se ha encontrado que esto sea así.

Al ver que haciendo ligeras modificaciones en la estructura de la red para aumentar su complejidad no logramos que esta dé mejores predicciones, vamos a probar otra opción diferente: cambiar el optimizador que estamos utilizando. Hasta ahora se había estado usando el SGD, que, como ya se ha comentado anteriormente, necesita que introduzcamos manualmente el valor de la tasa de aprendizaje que debe utilizar. Tras probar varios optimizadores, se ha llegado a la conclusión de que los mejores resultados son los proporcionados por el AdaGrad. Recordemos que la principal diferencia que este optimizador tiene con el SGD es que adapta la tasa de aprendizaje que hay en cada momento a los parámetros del sistema y al gradiente que hay en cada caso (de ahí su nombre, Algoritmo de Gradiente Adaptativo o *Adaptative Gradient Algorithm*). Por tanto, al proporcionarnos mejores resultados, a partir de ahora será este el optimizador que estemos utilizando. Sin embargo, la mejoría es bastante leve, considerablemente menor de lo que se buscaba conseguir.

5.1.1. Separación en dos redes

Tras haber realizado bastantes modificaciones en la red original que nos han dado unos resultados algo mejores, vamos a buscar otra alternativa para minimizar el error cometido por la red: separarla en dos redes independientes.

Lo primero que cabría pensar es si podemos hacer esto manteniendo el correcto funcionamiento del sistema, y la respuesta es que sí. Si nos fijamos en los coeficientes de la ecuación (12), podemos observar que hay unos coeficientes que tienen que ser nulos, uno para el cálculo de cada campo. Esto se debe a que, para calcular las componentes del campo eléctrico (o magnético), no encontramos una dependencia con el valor que tenía él mismo en la posición adyacente. Es decir, para cada una de las neuronas de salida de la Figura 8, hay una de entrada que no está introduciendo ningún tipo de información relevante, sino todo lo contrario, está introduciendo una componente de error.

Por tanto, para eliminar esta componente de error vamos a dividir la red en dos distintas, cada una con 3 neuronas de entrada, 2 neuronas en la capa oculta y una sola de salida, como se puede observar en la Figura 9. De esta forma, cada una de las redes estará prediciendo una componente del campo concreta, utilizando como datos de entrada únicamente los necesarios para calcularla. Para verlo de una forma más clara, vamos a extrapolar la ecuación (12) a este caso en forma de dos ecuaciones matriciales más sencillas:

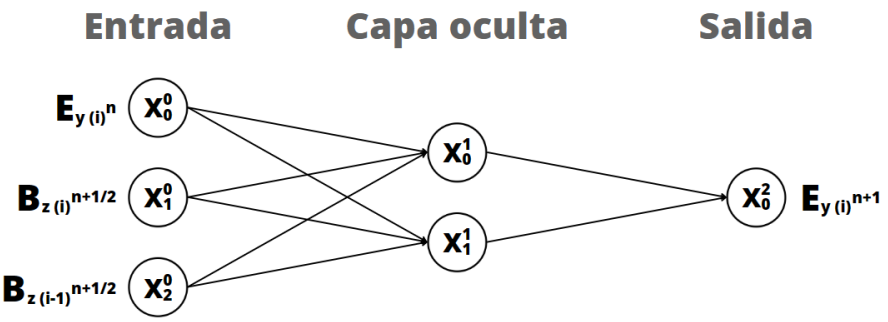


Figura 9: Red neuronal utilizada para calcular la evolución del campo eléctrico. La red correspondiente al cálculo del campo magnético sería equivalente, modificando únicamente las componentes de los campos que se introducen.

$$E_{y(i)}^{n+1} = \begin{pmatrix} 1.0 & -\frac{c\Delta t}{\Delta x} & \frac{c\Delta t}{\Delta x} \end{pmatrix} \begin{pmatrix} E_{y(i)}^{n} \\ B_{z(i)}^{n+1/2} \\ B_{z(i-1)}^{n+1/2} \end{pmatrix} \tag{14}$$

$$B_{z(i)}^{n+1/2} = \begin{pmatrix} \frac{c\Delta t}{\Delta x} & -\frac{c\Delta t}{\Delta x} & 1.0 \end{pmatrix} \begin{pmatrix} E_{y(i)}^{n} \\ E_{y(i+1)}^{n} \\ B_{z(i)}^{n-1/2} \end{pmatrix} \tag{15}$$

donde recordemos que el valor teórico de los elementos distintos de uno era $c\Delta t/\Delta x = 0.519615\ldots$ La gran ventaja que tiene esto es que así evitamos que la red necesite aprender el funcionamiento del algoritmo de salto de rana ya que, al haber separado las ecuaciones, ahora simplemente introducimos de forma directa la componente del campo que corresponda en cada caso.

Los coeficientes resultantes al multiplicar las matrices de los pesos obtenidos por las redes por separado son los siguientes:

$$\left[\begin{pmatrix} W_{0,0}^{0,1} & W_{0,1}^{0,1} \\ W_{1,0}^{0,1} & W_{1,1}^{0,1} \\ W_{2,0}^{0,1} & W_{2,1}^{0,1} \end{pmatrix} \begin{pmatrix} W_{0,0}^{1,2} \\ W_{1,0}^{1,2} \end{pmatrix} \right]_E^T = (0.9999 \quad -0.5191 \quad 0.5191) \tag{16}$$

$$\left[\begin{pmatrix} W_{0,0}^{0,1} & W_{0,1}^{0,1} \\ W_{1,0}^{0,1} & W_{1,1}^{0,1} \\ W_{2,0}^{0,1} & W_{2,1}^{0,1} \end{pmatrix} \begin{pmatrix} W_{0,0}^{1,2} \\ W_{1,0}^{1,2} \end{pmatrix} \right]_B^T = (0.5191 \quad -0.5181 \quad 0.9954) \tag{17}$$

Se puede observar que hay una ligera mejora en todos los coeficientes respecto al valor teórico comparados con el que tenían antes, probablemente debido a que la red ya no tiene que aprender el funcionamiento del salto de rana, lo cual podía introducir una importante componente de error. Además, sumando esto a la eliminación total del error en dos de los coeficientes, conseguimos obtener un error cuadrático medio mucho menor, con un valor de $MSE = 1.9 \cdot 10^{-11}$. Cabe destacar que, si bien esta modificación ha introducido un ligero aumento de tiempo de simulación, la gran mejora de los resultados lo compensa.

5.2. Evolución completa del sistema

Uno de los resultados previos que hemos reproducido al comienzo es la introducción de los coeficientes obtenidos por la red en el método FDTD para comprobar si el pulso que se creaba con ellos era correcto, obteniendo que existía una pérdida de amplitud que aumentaba en el tiempo conforme la onda se propagaba por el material. Este es uno de los motivos por los cuales se han intentado optimizar lo máximo posible los resultados proporcionados por la red, para ver si este decrecimiento desaparecía, pero tras hacer las simulaciones con los nuevos coeficientes se ha encontrado que no es así.

En este subapartado, lo que nosotros buscamos es ver si la red ha sido capaz de aprender correctamente el funcionamiento del método FDTD y si es capaz de replicarlo por su cuenta, para ver si también sigue ocurriendo lo mismo en esta ocasión. Para ello, vamos a utilizar unas condiciones iniciales a partir de las cuales la red neuronal tendrá que predecir el comportamiento temporal completo del sistema. Este proceso requiere una mayor complejidad y, por tanto, una mayor precisión en el entrenamiento previo de la red, lo cual es otra de las razones por las que se optimizaron los resultados obtenidos en el apartado anterior. La predicción obtenida mediante este código viene representada en la Figura 10.

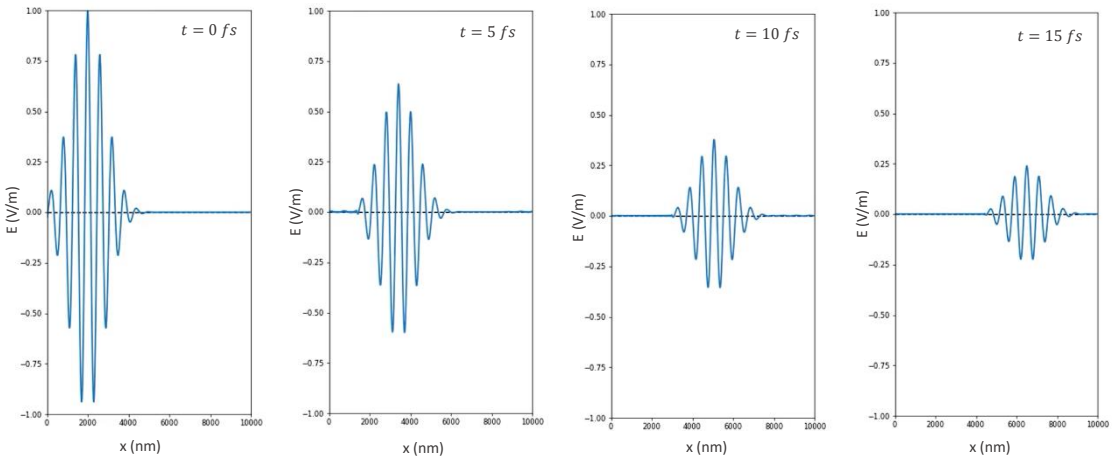


Figura 10: Predicción de la evolución temporal del pulso gaussiano de la Figura 7 realizada por la red, capturado en los mismos cuatro instantes de tiempo.

Como se puede observar, el pulso decrece perdiendo amplitud conforme se propaga, del mismo modo que lo hacía cuando simplemente se sustituían los coeficientes en las ecuaciones de Maxwell discretizadas. De nuevo, no tiene el comportamiento ideal que debería, como el pulso teórico que se ha mostrado en la Figura 7. Probablemente esto es debido a las pequeñas diferencias que existen entre los coeficientes obtenidos por la red y los reales, que puede acabar siendo bastante notable cuando hacemos la evolución completa mediante unas únicas condiciones iniciales.

5.2.1. Comprobación de la estabilidad del sistema

Al encontrarnos de nuevo con esta situación de pérdida de energía, vamos a disponernos a buscar una justificación para lo que sucede. La primera razón que podemos pensar es que estamos fuera del criterio de estabilidad del método FDTD (condición CFL) que hemos mostrado en la ecuación (10).

No vamos a entrar en detalle en los cálculos que hay detrás del criterio de estabilidad, ya que escapa del alcance de este trabajo. Sin embargo, por simplificar la explicación es importante saber que el hecho de estar fuera del criterio de estabilidad, es decir, tener un factor de estabilidad $S \geq 1$, hace que aparezca una frecuencia angular compleja en nuestra onda. Esta frecuencia se puede descomponer en la componente real que tenemos en el caso más estable más otro factor exponencial complejo $e^{\pm i\gamma t}$. Sería precisamente el factor γ del exponente el que podría provocar con su signo que la amplitud de la onda diverja creciendo hasta el infinito o disminuya hasta hacerse nulo, que es lo que podría estar sucediendo en este caso.

Es verdad que, como hemos comentado antes, hemos hecho la elección de los parámetros de discretización de la red de forma que siempre nos encontremos en el caso estable. Sin embargo, podría darse la situación de que los coeficientes que nos está proporcionando la red no cumplan el criterio de estabilidad. Para comprobarlo, podemos escribir los parámetros que aparecen en la ecuación (12) en función del factor de estabilidad, obteniendo que los coeficientes que relacionan los campos alternados son en esencia el propio factor S, independientemente de su signo.

Como ya habíamos visto previamente, los coeficientes obtenidos por la red son bastante similares a los reales, y en ningún caso superan la unidad, por lo que podemos confirmar que no nos encontramos en un situación de inestabilidad que esté produciendo este comportamiento decreciente.

5.2.2. Pérdida de energía por corrientes de absorción

Una vez nos hemos asegurado de que el decrecimiento del pulso no se debe a una situación de inestabilidad del algoritmo, podemos encontrar un sentido físico al comportamiento que está presentando el sistema. Esta explicación puede ser la pérdida de energía debido a corrientes de absorción eléctricas y magnéticas.

Las ecuaciones de Maxwell que hemos escrito en las ecuaciones (4) y (5) son las correspondientes a un material homogéneo, isótropo y sin pérdidas. Sin embargo, el comportamiento del campo electromagnético calculado con la red neuronal recuerda a la propagación de un paquete de ondas en un medio absorbente. La idea aquí es la de modelas la desviación que observamos mediante términos de absorción. Si escribimos las ecuaciones de Maxwell incluyendo términos de corriente, estas quedan de la siguiente forma:

$$\frac{\partial E_y}{\partial t} = \frac{1}{\varepsilon}\left(-\frac{\partial H_z}{\partial x} - J_E\right) = -\frac{1}{\varepsilon}\left(\frac{\partial H_z}{\partial x} + \sigma_E E_y\right) \tag{18}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left(-\frac{\partial E_y}{\partial x} - J_M\right) = -\frac{1}{\mu}\left(\frac{\partial E_y}{\partial x} + \sigma_M H_z\right) \tag{19}$$

donde σ_E y σ_M son las conductividades eléctrica y magnética que producen las corrientes de absorción J_E y J_M, respectivamente, y $\varepsilon = \varepsilon_r\varepsilon_0$ y $\mu = \mu_r\mu_0$ son la permitividad eléctrica y la permeabilidad magnética de este material, obtenidas al multiplicar los valores relativos (iguales o mayores que la unidad) por los del vacío.

Ahora, vamos a desarrollar estas expresiones para que queden de una forma similar a las ecuaciones (7) y (8), sustituyendo el vector campo magnético por el de inducción magnética y escribiendo los coeficientes de un modo más visual:

$$E_y^{n+1}(x) = \left(1 - \frac{\sigma_E}{\varepsilon}\right) E_y^n(x) - \frac{v\Delta t}{\Delta x}\left[B_z^{n+1/2}\left(x + \frac{\Delta x}{2}\right) - B_z^{n+1/2}\left(x - \frac{\Delta x}{2}\right)\right] \quad (20)$$

$$B_z^{n+1/2}\left(x + \frac{\Delta x}{2}\right) = \left[1 - \frac{\sigma_M}{\mu}\right] B_z^{n-1/2}\left(x + \frac{\Delta x}{2}\right) - \frac{v\Delta t}{\Delta x}\left[E_y^n(x + \Delta x) - E_y^n(x)\right] \quad (21)$$

donde v es la velocidad de la luz en este medio, ya que no podemos seguir considerando que nos encontramos en el vacío.

Viendo estas ecuaciones, podemos relacionar los coeficientes que acompañan a los campos con los que nos ha proporcionado la red para calcular las conductividades y las características del medio. Sin embargo, vemos que únicamente disponemos de tres coeficientes distintos, ya que aquellos que dependen de los campos cruzados son iguales en ambas ecuaciones, de forma que tenemos cuatro incógnitas para tres ecuaciones. Vamos a solucionar este problema asumiendo que nuestro material tiene $\mu_r = 1$, respuesta típica de la mayoría de medios en el régimen de energías en el que nos hemos centrado en este trabajo de modo que ahora sí que tenemos información suficiente para calcular el resto de las incógnitas.

Inicialmente, vamos a fijarnos en el coeficiente de los términos cruzados y en los valores que hemos obtenido para ellos. Como todos ellos deberían ser iguales, vamos a hacer el promedio de los cuatro resultados dados por la red para usarlo en nuestros cálculos. De esta forma, veamos cuál es el valor de la permeabilidad eléctrica del medio:

$$0.5189 = v\frac{\Delta t}{\Delta x} = \frac{1}{\sqrt{\varepsilon\mu}}\frac{\Delta t}{\Delta x} = \frac{c}{\sqrt{\varepsilon_r\mu_r}}\frac{\Delta t}{\Delta x} = \frac{0.5196}{\sqrt{\varepsilon_r}} \Rightarrow \varepsilon_r = 1.0027 \quad (22)$$

Como podemos ver, a pesar de tener una permitividad pequeña, nos encontramos en un medio dieléctrico, lo que justifica la existencia de corrientes de absorción que disminuyen progresivamente la cantidad de energía que tenemos en nuestro sistema.

Una vez conocidos los valores de la permitividad eléctrica y la permeabilidad magnética (que hemos asumido uno) del medio, podemos utilizar los coeficientes que dependen del propio campo que estamos calculando para calcular las conductividades que dan cuenta de las corrientes de absorción, cuyos valores vendrán dados por:

$$1 - \frac{\sigma_E}{\varepsilon} = 0.9999 \Rightarrow \sigma_E = 0.0001\ S/m \quad (23)$$

$$1 - \frac{\sigma_H}{\mu} = 0.9954 \Rightarrow \sigma_H = 0.0046\ H/m \quad (24)$$

Como vemos, el error asociado a la red neuronal, aunque es pequeño, implica serias desviaciones respecto de la respuesta típica del medio real (en nuestro caso, el vacío). Curiosamente, el error puede ser asumido como una pérdida, no dando lugar a inestabilidades numéricas, sino a un medio efectivo con propiedades absorbentes.

5.3. Implementación de redes más complejas

En la última parte de este trabajo, vamos a dejar de lado la red lineal que hemos estado utilizando hasta ahora. En su lugar, vamos a buscar redes más complejas que puedan igualar o incluso mejorar los resultados que hemos obtenido hasta ahora.

Los beneficios de encontrar redes más complejas capaces de hacer esta simulación serían numerosos. Para empezar, las redes con un mayor número de capas y neuronas y con funciones de activación san capaces de aprender patrones y encontrar relaciones más profundas en los datos. Además, son capaces de generalizar sus predicciones y adaptar sus características mejor ante nuevos datos.

El motivo principal de esta búsqueda es intentar ser capaces de sustituir el método FDTD original por una red que cumpla el mismo cometido y dé los mismos resultados. De esta forma, podríamos acelerar el proceso de simulación respecto al algoritmo original, ya que el método FDTD puede consumir mucho tiempo para problemas más complejos que los que estamos estudiando, y también reduciríamos los recursos necesarios para realizar las simulaciones.

Sin embargo, el trabajo de encontrar una red compleja que se ajuste correctamente a un sistema concreto no es nada sencillo, ya que hay una gran cantidad de hiperparámetros que hay que ajustar, más conforme mayor sea su complejidad. Estos hiperparámetros comprenden la cantidad de capas, el número de neuronas por capa, la introducción de funciones de activación correctas, la implementación de regularizadores… No existe una regla concreta que diga cómo hay que determinar estos hiperparámetros o cuáles de ellos son o no necesarios, por lo que es un trabajo meramente tentativo, prácticamente de prueba y error, en búsqueda de una red que se comporte como deseamos.

Por tanto, vamos a ir modificando hiperparámetros para probar redes de lo más variadas, y si encontramos alguna que nos dé una buena respuesta, podremos hacer pequeñas modificaciones en torno a su base principal para intentar encontrar una red con un comportamiento lo más similar posible al que nos proporciona la red sencilla que ya tenemos.

Lo primero que vamos a hacer es establecer el número de capas y las neuronas en cada una de ellas y jugar con la introducción de *dropout* y funciones de activación (con capas de *batch normalization*). Con la implementación de estos dos elementos, tenemos cuatro posibles combinaciones: no tener ninguna de ellos, tener solo uno o tener ambos. De esta forma, veremos cuál es la que funciona mejor y a partir de ahí crearemos las próximas redes, en las que modificaremos otros hiperparámetros. Lo único que mantendremos constante en todas las redes que probemos será que la capa de entrada tendrá siempre tres neuronas, ya que es donde introducimos las componentes del campo, y la de salida tendrá una, ya que corresponde con la salida del campo que calculamos.

Para empezar, vamos a establecer tres capas ocultas (de 256, 128 y 64 neuronas, respectivamente), de forma que estamos triplicando la cantidad que teníamos antes y aumentando de gran manera el número de neuronas que tiene la red. La función de activación que utilizaremos será la SELU (*Scaled Exponential Linear Unit*), una pequeña modificación de la ReLU cuya diferencia es que para los valores negativos la respuesta no es nula, sino un pequeño valor exponencial decreciente, ya que en el primer apartado vimos que la función ReLU probablemente estaba introduciendo pocas modificaciones en el tratamiento de los datos. En cuanto al *dropout*, introduciremos un 30% de neuronas desactivadas aleatoriamente en la primera capa, un 20% en la segunda y un 10% en la tercera, de forma que vaya en correspondencia al número de neuronas que tiene cada una. Veamos cuáles han sido las predicciones de los campos eléctricos realizadas por cada una de las cuatro redes que hemos entrenado de esta forma:

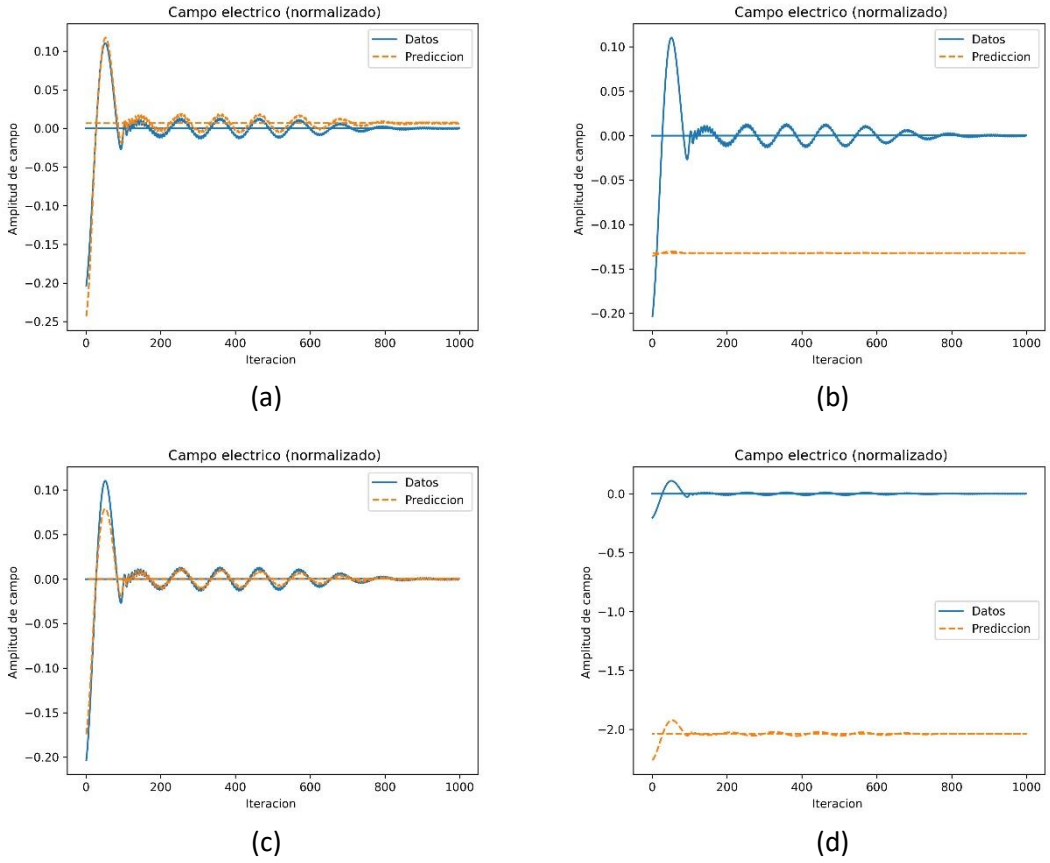
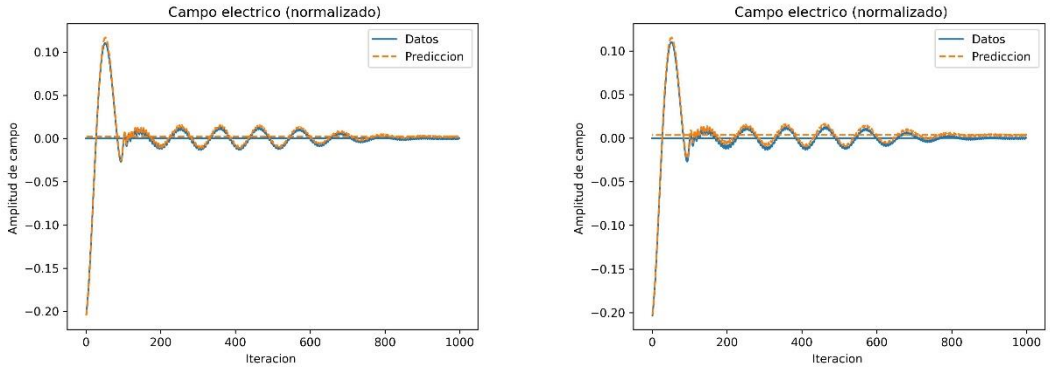


Figura 11: Predicción del campo eléctrico en $x = 50\ nm$ realizada por distintas redes con tres capas ocultas de 256, 128 y 64 neuronas, respectivamente. (a) Red con *Dropout* y función de activación SELU (con *Batch Normalization*). (b) Red solo con *Dropout*. (c) Red solo con función de activación SELU (con *Batch Normalization*). (d) Red sin *Dropout* ni función de activación.

En este caso, no tenemos la simulación de un pulso que se propaga a través de todo el material, sino que estamos representando el campo eléctrico que hay en un punto concreto del material conforme pasa el tiempo. Este punto se encuentra en la zona inicial del material, de ahí que el campo disminuya en él con el tiempo. El tiempo de simulación del sistema en este caso es tal que el pulso pasa a través de este punto una sola vez, es decir, que no rebota en los extremos del material.

De las cuatro redes que hemos probado en la Figura 11, hay dos que dan mejores resultados que otros. La que nos da una peor predicción es aquella que no tiene ni *dropout* ni función de activación, ya que realiza un *overfitting* que incrementa en gran medida el error. Es la que solo tiene función de activación la que predice mejor el comportamiento del campo, con un error de $MSE \sim 10^{-3}$ que todavía queda muy alejado del que teníamos con la red lineal.

Una vez hemos encontrado la combinación de hiperparámetros de activación y regularización que parece más adecuada, vamos a pasar a modificar el número de capas y el de neuronas que hay en cada una de ellas. Cuando intentamos aumentar la complejidad de la red incrementando el número de capas, los resultados obtenidos son nefastos, por lo que podemos deducir que lo que tenemos que hacer es reducir la cantidad de capas y neuronas. Veamos las predicciones obtenidas mediante otras dos redes con un número de capas y neuronas diferente:

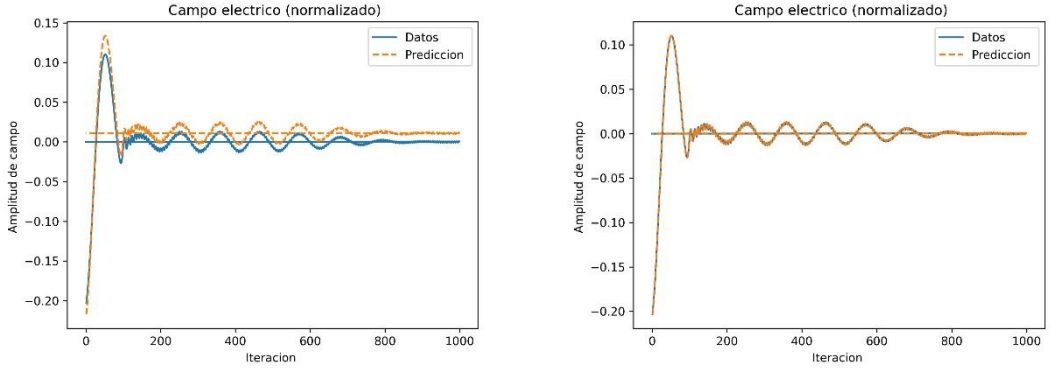


(a) (b)

Figura 12: Predicción del campo eléctrico en $x = 50\ nm$ realizada por distintas redes con una función de activación SELU y *Batch Normalization*. (a) Red con tres capas ocultas de 64, 32 y 16, respectivamente. (b) Red con dos capas ocultas de 128 y 32 neuronas, respectivamente.

La predicción de la primer red de la Figura 12 tiene un error similar al mejor de los casos anteriores, siendo superada por el error de $MSE \sim 10^{-4}$ de la segunda que, si bien ha supuesto una ligera mejoría, sigue distando mucho de lo que buscamos obtener.

Aun así, estos nuevos intentos nos han servido para descubrir que la predicción mejora conforme disminuimos el número de capas, por lo que vamos a hacer una última prueba con dos redes de una sola capa oculta con 32 neuronas, con y sin activación:



(a) (b)

Figura 13: Predicción del campo eléctrico en $x = 50\ nm$ realizada por distintas redes con una sola capa oculta de 32 neuronas. (a) Red con función de activación SELU (con *Batch Normalization*). (b) Red sin *Dropout* ni función de activación.

En este caso, la red con función de activación de la Figura 13 ha dado una predicción bastante peor que la lineal, cuyo error cuadrático medio ha sido de $MSE = 7 \cdot 10^{-10}$. Este error ya se podría considerar aceptable, dado que se acerca mucho al de la red sencilla que buscábamos igualar. Sin embargo, esta nueva red realiza un ligero *overfitting* que habría hecho que el error, en vez de disminuir, siguiera creciendo para un mayor tiempo de simulación.

6. Conclusiones

A lo largo de este trabajo, hemos estudiado principalmente distintas redes neuronales con el objetivo de ser capaces de implementarlas computacionalmente en sustitución del algoritmo FDTD. Lo primero que se ha hecho para ello es optimizar la red lineal sencilla de la que se partía inicialmente dividiéndola en dos distintas para que mejoraran los resultados que se predecían.

A continuación, se ha estudiado la predicción completa de la propagación de una onda electromagnética proporcionada por la red entrenada, encontrando de nuevo que el pulso decrecía con el tiempo. Tras asegurarnos de que se cumplía la condición de estabilidad del método, se ha concluido que una explicación física razonable podía ser la pérdida de energía mediante unas corrientes de absorción que no habían sido consideradas inicialmente en las ecuaciones de Maxwell.

Por último, se ha estudiado la posibilidad de que otras redes mucho más complejas que la nuestra consiguieran obtener predicciones iguales o mejores que las que teníamos, encontrándonos una respuesta negativa. El motivo de este resultado puede deberse al hecho de que nuestro problema es lineal y relativamente simple, por lo que redes mucho más complejas a las que además les introducimos una componente de no linealidad no son capaces de aprender correctamente el funcionamiento del sistema, ya que la complejidad de la red tiene que ir en concordancia con la del problema para funcionar bien. De hecho, esto se puede comprobar viendo que hemos partido de redes con tres capas ocultas y, tras hacer pruebas tentativas, hemos visto que los resultados eran mejores conforme reducíamos el número de capas de la red y el de neuronas por capa, lo que nos acercaba al caso simple y lineal que ya teníamos inicialmente. Por tanto, hemos comprobado que unos pequeños cambios en el sistema, como puede ser la introducción de una no linealidad, producen efectos devastadores en su funcionamiento. Esto se puede deber a que los campos dejan de tener un comportamiento lineal y, por tanto, dejamos de utilizar el primer término de la serie de Taylor en el que habíamos desarrollado sus ecuaciones. Con esto, podemos concluir que el método FDTD adaptado a nuestro problema solo funciona en el caso lineal.

Como trabajo futuro que se podría realizar a partir de este, encontramos distintas posibilidades. Un posible estudio de gran interés sería investigar qué ocurre cuando nos salimos del criterio de estabilidad, ya que podría darse la posibilidad de que el método FDTD diverja pero la red no, lo que proporcionaría un beneficio a la hora de utilizar la red como sustitución del algoritmo original. Otras cuestiones que se podrían estudiar son el comportamiento de dieléctricos u otros materiales, ya que aquí nos hemos centrado exclusivamente en estudiar ondas propagándose por el vacío. Por último, podría investigarse si mediante la implementación de otro tipo de redes, como las que tienen memoria temporal a corto plazo (LSTM), consiguen un mejor aprendizaje del comportamiento del sistema.

7. Bibliografía

[1] Azamat Abdoullaev, “How artificial intelligence is disrupting physics”, BBN Times, 2023. [En línea]. Disponible en: https://www.bbntimes.com/technology/how-artificial-intelligence-is-disrupting-physics. [Último acceso: 27 julio 2023].

[2] Azucena Martín, “Esta inteligencia artificial ha ‘reescrito’ la física tal y como la conocemos”, Hipertextual, 28 julio 2022. [En línea]. Disponible en: https://hipertextual.com/2022/07/inteligencia-artificial-otra-fisica. [Último Acceso: 28 julio 2023].

[3] Michael A. Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015.

[4] Allen Taflove, Susan C. Hagness, “Computational Electrodynamics: The Finite-Difference Time-Domain Method”, Artech House Inc., pp. 1-80, 2005.

[5] Blanca Azuara, Luis Martín, Sergio Gutiérrez, “Inteligencia artificial aplicada a la nanofotónica”, 2021.

[6] Código programado a lo largo del trabajo y utilizado para realizar las simulaciones: https://github.com/IrisFDTD/FDTD_on_Neural_Networks/tree/main.

[7] Vicente Rodríguez, “Conceptos básicos sobre redes neuronales”, Vincentblog, 30 octubre 2018. [En línea]. Disponible en: https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales. [Último acceso: 30 julio 2023].

[8] Andreas Refsgaard, Francis Tseng, “How neural networks are trained”, Github, 2020. [En línea]. Disponible en: https://ml4a.github.io/ml4a/how_neural_networks_are_trained/. [Último acceso: 3 agosto 2023].

[9] Ivan V. Meza, “Descenso por gradiente”, 21 noviembre 2016. [En línea]. Disponible en: https://turing.iimas.unam.mx/~ivanvladimir/posts/gradient_descent/. [Último acceso: 3 agosto 2023].

[10] Jesús Martínez, “Optimizadores en Deep Learning”, Datasmarts, 7 enero 2020. [En línea]. Disponible en: https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/. [Último acceso: 3 agosto 2023].

[11] Alfredo Canziani, “Overfitting and regularization”, NYU_DeepLearning, 2020. [En línea]. Disponible en: https://atcold.github.io/pytorch-Deep-Learning/es/week14/14-3/. [Último acceso: 3 agosto 2023].

[12] Néstor Aldea Ramos, Ana Grande, José Represa, “Modelado numérico de la propagación de ondas electromagnéticas mediante el método de las Diferencias Finitas en el Dominio del Tiempo (FDTD): aplicación a medios Tellegen”, pp. 13-27, 2015.