Universidad de Zaragoza

Grado en Física

Trabajo Fin de Grado

INTELIGENCIA ARTIFICIAL APLICADA A NANOFOTÓNICA

Elisa Lupón Monge

Director: Luis Martín Moreno

Curso 2019/2020

Índice

1. Introducción

En las últimas décadas se han producido grandes avances en el campo de la Inteligencia Artificial (IA). Este término fue introducido por John McCarthy en 1956[1] a partir del test de Turing [1] como referencia a la capacidad de las máquinas de imitar las funciones cognitivas asociadas a los seres humanos para aprender, resolver problemas y adaptarse al entorno.

En la actualidad, existen varias ramas en desarrollo dentro de la IA siendo una de las más importantes el denominado *Machine Learning* o aprendizaje automático. Esta técnica se basa en el diseño de algoritmos capaces de aprender automáticamente a partir de datos. Debido al reciente aumento de la cantidad de bases de datos disponibles (*big data*), son numerosas las posibles aplicaciones del *Machine Learning* a nuevos campos de investigación tales como la física de partículas, astronomía, biofísica, materia condensada o computación cuántica [2].

En concreto, el objetivo de este trabajo es aplicar los conceptos característicos de las redes neuronales y *deep learning* a un problema básico encontrado en múltiples ámbitos de la física y desarrollar posibles aplicaciones específicas dentro de la nanofotónica.

2. Redes neuronales y Deep Learning

El aprendizaje automático es el subcampo de la ciencia computacional que trata de desarrollar técnicas que permitan obtener predicciones fiables a partir del análisis de series de datos. Al contrario de los algoritmos de computación tradicionales en los que un problema complejo se divide en tareas sencillas realizables por el ordenador, el objetivo es que sea el propio algoritmo el que busque la solución que mejor se adapte al problema dado.

Para llevar a cabo esta tarea existen tres tipos de aprendizaje automático: aprendizaje supervisado, no supervisado y reforzado. Las dos primeras categorías hacen referencia a la información que se ofrece sobre los datos al programa, mientras que la tercera depende de su interacción con el entorno. A lo largo de este trabajo nos centraremos en el aprendizaje supervisado, utilizado habitualmente en problemas de clasificación y regresión. Por su parte, el aprendizaje no supervisado se emplea en la obtención de patrones y estructuras de datos no etiquetados como, por ejemplo, *clustering*.

A continuación, describiremos los conceptos básicos sobre los que se construyen los programas que vamos a utilizar, así como su origen, relevancia y funcionamiento en líneas generales. Entre ellos destacaremos las redes neuronales, el método de descenso por gradiente, *backpropagation*, *deep learning* y redes convolucionales.

2.1. Estructura de redes

La estructura de los algoritmos de aprendizaje automático, y en especial de aprendizaje profundo (*deep learning*), está compuesta por una serie de elementos conocidos como *neuronas artificiales* que se relacionan entre sí formando *redes neuronales artificiales* (RNA). El origen de este nombre se encuentra en la aparente similitud que presentan con las neuronas del cerebro humano, si bien su funcionamiento en la práctica difiere en ciertos aspectos [3].

[1] Conferencia de Dartmouth organizada por John McCarthy, Marvin L. Minsky, Nathaniel Rochester y Claude E. Shannon.

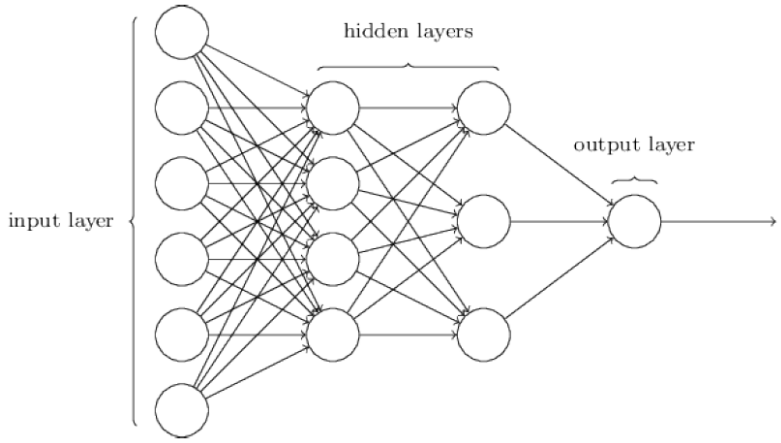


Figura 1: Esquema red neuronal artificial (RNA). Fuente: *Neural Networks and Deep Learning.*

Como se muestra en la Figura 1, las redes neuronales se organizan en capas: una capa en entrada o *input* donde se recibe la información, una o varias capas ocultas (*hidden layers*) y una capa de salida o *output* que devuelve el resultado. A excepción de las capas de entrada y salida que solo presentan conexiones hacia la capa siguiente y precedente respectivamente, cada neurona está unida a las de las capas anterior y posterior mediante pesos o *weights*. Estos indican la contribución relativa de cada neurona al valor de las siguientes. La suma de cada una de estas contribuciones en relación con el valor umbral o *bias* propio de cada neurona determinará su grado de activación o *output*. Atendiendo a la dirección en la que fluye la información, se diferencian las redes *feedforward* en las que el *output* de una capa es el *input* de la siguiente y las redes recurrentes (RNN) donde pueden realizarse bucles de retroalimentación. En lo respectivo a este trabajo, nos limitaremos al uso de redes *feedforward*.

2.2. Tipos de neuronas

El primer modelo de neurona artificial que se diseñó para formar redes neuronales fue el *perceptron* [4]. En una neurona de este tipo, dada una serie de inputs x_1, x_2, x_3... junto con sus respectivos pesos w_1, w_2, w_3..., el *output* puede adoptar los valores 0 o 1 en función del valor umbral b de forma que:

$$output = \begin{cases} 0 & \text{si } \sum_i x_i w_i + b \leq 0 \\ 1 & \text{si } \sum_i x_i w_i + b > 0 \end{cases} \tag{1}$$

Este modelo supone un problema al limitar la respuesta a dos únicos posibles estados. Para evitar que pequeños cambios afecten significativamente al comportamiento de la red conviene introducir otro tipo de neurona: la *sigmoid*. Como su nombre indica su función de activación viene dada por la función $\sigma(z)$:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

donde $z = x \cdot w + b$.[2] Como resultado, el *output* variará entre todos los posibles valores intermedios entre 0 ($e^{-z} \to \infty$) y 1 ($e^{-z} \approx 0$).

[2] $\sum_i x_i w_i = x \cdot w$. Se ha simplificado el sumatorio como producto escalar.

Para nuestro primer problema utilizaremos este tipo de neuronas, sin embargo, veremos más adelante que existen otras funciones de activación más útiles dependiendo de las características de la red que se quiera implementar.

2.3. Algoritmos de aprendizaje

Dado un conjunto de datos etiquetados la red neuronal se entrena en sucesivos ciclos hasta conseguir que el *output*, a, sea lo más cercano posible al esperado $y(x)$. En esta etapa entran en juego diversos factores, siendo el primero de ellos a tener en cuenta la serie de datos ofrecidos. Para obtener un resultado óptimo conviene disponer de un número amplio de datos, los cuáles suelen dividirse en tres sets: *training data*, *validation data* y *test data*.

El *training data* es el conjunto de n datos sobre los que la red neuronal aprende. Este proceso puede llevarse a cabo por distintos métodos pero el objetivo general es siempre minimizar una función coste $C(w, b)$ modificando los valores de los parámetros de la red (*weights*, *bias*). Las funciones más comunes son el error cuadrático medio:

$$C(w, b) = \frac{1}{2n} \sum_x ||y(x) - a||^2 \tag{3}$$

con $a = \sigma(z)$ y la función *cross-entropy*[3]:

$$C(w, b) = -\frac{1}{n} \sum_x [y \; ln\, a + (1 - y) ln(1 - a)] \tag{4}$$

El método de minimización más habitual es el descenso por gradiente estocástico o *stochastic gradient descent* (SGD). La idea básica de los descensos por gradiente es ajustar de forma iterativa el valor de los parámetros $\nu = (w, b)$ en la dirección de máxima pendiente negativa de la función coste. El cambio en cada paso será:

$$\Delta C \approx \nabla C \cdot \Delta \nu \tag{5}$$

$$\nu \to \nu' = \nu - \eta \nabla C \tag{6}$$

donde η es un hiperparámetro denominado *learning rate*. La sucesiva actualización de los parámetros siguiendo esta regla provoca que el sistema evolucione hacia un mínimo local en el cual podría estancarse, por lo que se incluye un nuevo concepto: el *minibatch*. Un *minibatch* es un subgrupo de m elementos del *training data* sobre los que se aplica el método de descenso por gradiente. Esta división confiere al sistema de una componente estocástica que le permite salir de mínimos locales, además de reducir significativamente el tiempo de cálculo. Cuando se realiza una interacción sobre todos los subconjuntos del *training data* se dice que se ha completado una época (*epoch*). El algoritmo que lleva a cabo esta tarea se conoce como *backpropagation* o propagación hacia atrás. Su funcionamiento se resume en los siguientes pasos:

1. ***Input***: introducción de los datos de entrada en la capa *input* ($l = 1$).

[3]La función *cross-entropy* se utiliza principalmente en problemas de clasificación. Para problemas de regresión es mejor el error cuadrático medio o *mse* (*mean square error*).

2. ***Feedforward***: cálculo en las capas sucesivas $l = 2, 3, ..., L$ la salida $a^l = \sigma(z^l)$ de cada neurona.

3. ***Output error*** δ^L: estimación del error $\delta^L = \dfrac{\partial C}{\partial z^L}$ en la última capa.[4]

4. ***Backpropagate error***: propagación hacia atrás del error $\delta^{l-1} = \delta^l \dfrac{\partial z^l}{\partial a^{l-1}} \dfrac{\partial a^{l-1}}{\partial z^{l-1}}$.

5. ***Output***: cómputo del gradiente de la función coste dado por $\dfrac{\partial C}{\partial w^l} = a^{l-1}\delta^l$ y $\dfrac{\partial C}{\partial b^l} = \delta^l$.

Una vez tenemos nuestra red entrenada podemos comprobar directamente su eficacia dándole como datos de entrada elementos del *test data* que no han sido utilizados para el aprendizaje. No obstante, a menudo resulta conveniente validar el comportamiento de la red mediante el *validation data* para evitar problemas de *overfitting*. Este concepto hace referencia al sobreentrenamiento del sistema, el cual provoca que se ajuste de manera excelente al *training data* perdiendo capacidad para predecir correctamente datos nuevos. Deteniendo el entrenamiento cuando la exactitud de los resultados del *validation data* satura (*early stopping*) es una buena manera de asegurarnos de que no tenemos *overfitting*.

El *validation data* también es útil para seleccionar el valor de los hiperparámetros. Dentro de esta categoría se encuentran parámetros ya nombrados como el *learning rate* η, el tamaño del *minibatch* m, el número de neuronas por capa, etc. Además de estos, es preciso introducir un nuevo hiperparámetro λ conocido como *regulation term*. Su función es penalizar valores grandes en los pesos de las neuronas (o una gran cantidad de estos). Se aplica como un término extra en la función coste del tipo:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{7}$$

donde C_0 es la función coste original sin regularizar. La finalidad de este hiperparámetro es reducir la probabilidad de *overfitting* anteriormente descrita.

2.4. Modelos más complejos

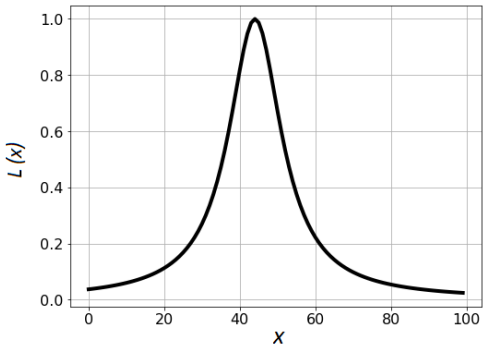
Por último, presentamos el concepto de las *redes neuronales convolucionales* (CNN) como modelo más complejo de la estructura de redes vista hasta ahora. Su funcionamiento se inspira en el mecanismo de las neuronas biológicas de la corteza visual primaria, siendo especialmente efectivas para problemas de visión artificial tales como la clasificación de imágenes. Su estructura consiste en una serie de capas ocultas jerarquizadas cada vez más especializadas sobre las que se aplican filtros (*kernel*) de una o más dimensiones en forma de función de mapeo no lineal. Más adelante veremos con mayor detalle sus particularidades en la implementación de un algoritmo para la caracterización de espectros.

[4] Se sigue la regla de la cadena: $\dfrac{\partial C}{\partial z^L} = \dfrac{\partial C}{\partial a^L} \dfrac{\partial a^L}{\partial z^L}$.

3. Primer acercamiento: la función Lorentziana

El primer problema al que nos enfrentamos es un problema de regresión básico. Nuestro objetivo será diseñar una red neuronal que *aprenda* la función Lorentziana $L(x)$, es decir, dados los cuatro parámetros característicos devuelva la forma de la función esperada:

$$L(x) = \frac{A}{(x - x_0)^2 + \sigma^2} + B \tag{8}$$



La elección de esta función, también conocida generalmente como distribución de Cauchy, se debe a su presencia en múltiples problemas dentro la física, tanto en el análisis de espectros de emisión y absorción como en el descubrimiento de nuevas partículas en resonancias nucleares con su versión relativista (Breit-Wigner). Este hecho, añadido a su relativa facilidad de cálculo, la convierte en una candidata ideal para realizar un primer acercamiento práctico a las redes neuronales.

3.1. Planteamiento del algoritmo

Para construir el modelo que cumpla con nuestro propósito vamos a tomar como referencia el código desarrollado por Michael Nielsen en su libro *Neural Networks and Deep Learning* accesible en GitHub. Originalmente el código está diseñado para la clasificación de dígitos escritos a mano procedentes del MNIST data set, por lo que será necesario realizar algunas modificaciones.
En primer lugar, dadas las características del problema necesitamos generar un conjunto de datos sobre los que trabajar. Estos estarán formados por cuatro parámetros de entrada (A, x_0, σ, B) y una serie de puntos correspondientes a la función evaluada $y = L(x)$. Cabe mencionar que esto implica discretizar una función continua, por lo que el resultado dependerá de la cantidad de puntos elegida ($n = 100$).

$$input = \begin{pmatrix} A \\ x_0 \\ \sigma \\ B \end{pmatrix} \qquad\qquad y = \begin{pmatrix} L(x_1) \\ L(x_2) \\ \vdots \\ L(x_n) \end{pmatrix} \tag{9}$$

De esta forma, creamos una subrutina que genere un conjunto de datos de dichas características. Para procurar un correcto aprendizaje, todos los parámetros estarán normalizados. Cada ejemplo se guardará en listas formadas por dos arrays de dimensión (4, 1) y (100, 1) de manera que se adapte a la estructura del código diseñado por Michael Nielsen. A su vez, los datos serán divididos en los sets: *training data*, *validation data* y *test data*. A cada uno le corresponderán 25000, 5000 y 100 ejemplos, respectivamente. Esta elección, en principio arbitraria, se debe a una búsqueda de compromiso entre coste computacional y mejora real del modelo.

A continuación, diseñamos la estructura de la red neuronal. Dada la relativa sencillez del problema vamos a utilizar una red *feedforward* de tres capas con neuronas tipo *sigmoid*. La primera capa se compondrá de cuatro neuronas, la segunda será la capa oculta de dimensión variable y la tercera la capa de salida. La red adoptará la siguiente forma:

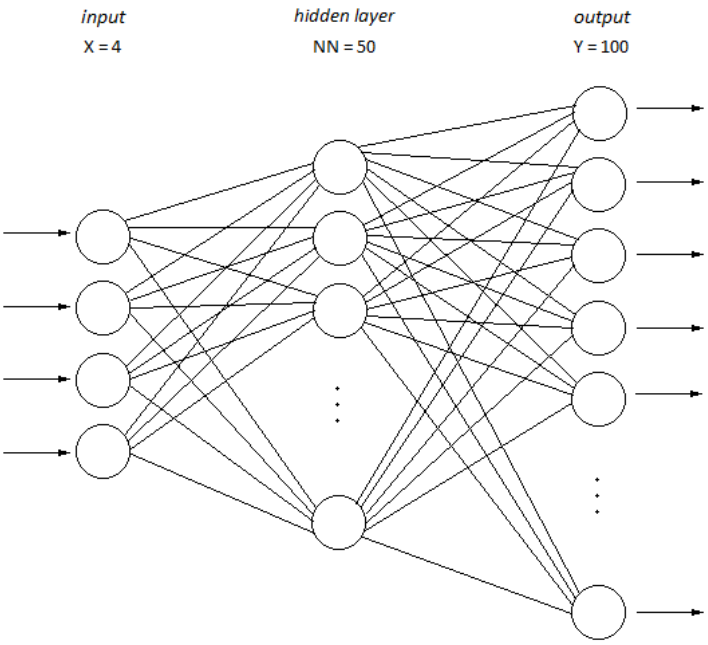


Figura 2: Estructura de la red neuronal. La primera capa o *input* se corresponde con los valores normalizados de los parámetros A, x_0, σ y B. Utilizamos una única capa oculta compuesta por 50 neuronas. La capa de salida tendrá el mismo número de neuronas que puntos seleccionados para discretizar la función. La red será de dimensión [4, 50, 100].

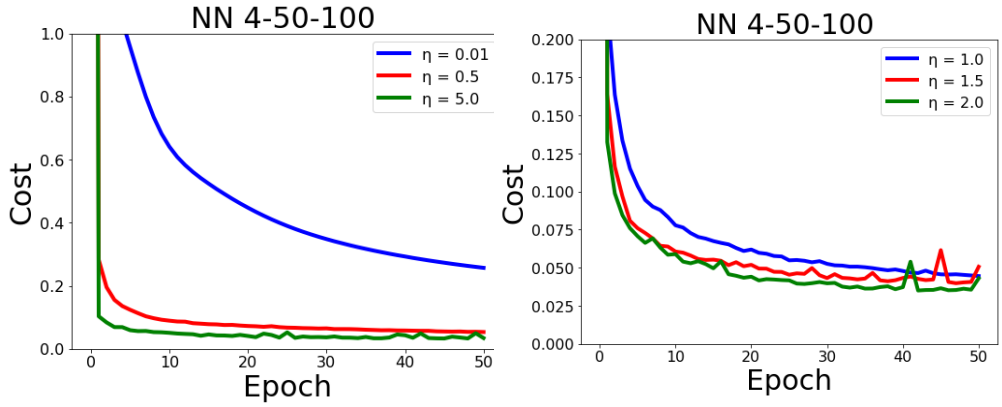
Inicializamos cada *weight* y *bias* mediante la función $default\_weight\_initializer$, la cuál devuelve un valor aleatorio con una distribución Gaussiana de media 0 y desviación estándar 1. En el caso de los pesos, el valor se divide por la raíz cuadrada del número de pesos conectados a una misma neurona. El algoritmo lee los datos de entrada en la primera capa y evalúa a la salida la diferencia entre el *output* obtenido y el valor esperado $y = L(x)$. Utilizaremos como función coste el error cuadrático (ec. 3), sobre el que añadiremos posteriormente el término de regulación (ec. 7). El aprendizaje se realizará por el método de *backpropagation* aplicando sucesivamente el descenso por gradiente estocástico (SGD) en *minibatch* de dimensión $m = 10$.

En cuanto a la elección de los hiperparámetros η y λ, realizaremos varias pruebas para determinar su valor óptimo. Los resultados obtenidos en función de estos, junto con el análisis correspondiente, se encuentran en el siguiente apartado.

3.2. Análisis de hiperparámetros

Una vez establecida la estructura de nuestra red neuronal conviene estudiar su comportamiento para determinados valores de los hiperparámetros. Este proceso nos ayudará a estimar cuáles son los valores óptimos para su funcionamiento, además de servir como ejemplo de las propiedades deseadas o a evitar en este tipo de modelos.

El primer hiperparámetro que vamos a optimizar es la tasa de aprendizaje o *learning rate*, η. Este parámetro determina la magnitud del cambio producido al modificar los *weights* y *bias* por el SGD (ec. 6). Nos interesa encontrar un valor que permita al sistema alcanzar un mínimo global estable en un tiempo razonable. Los resultados de las pruebas realizadas pueden verse en las siguientes gráficas (Figura 3). En todos los casos se ha utilizado el mismo modelo a excepción del valor de η evaluado sobre el *training data*.



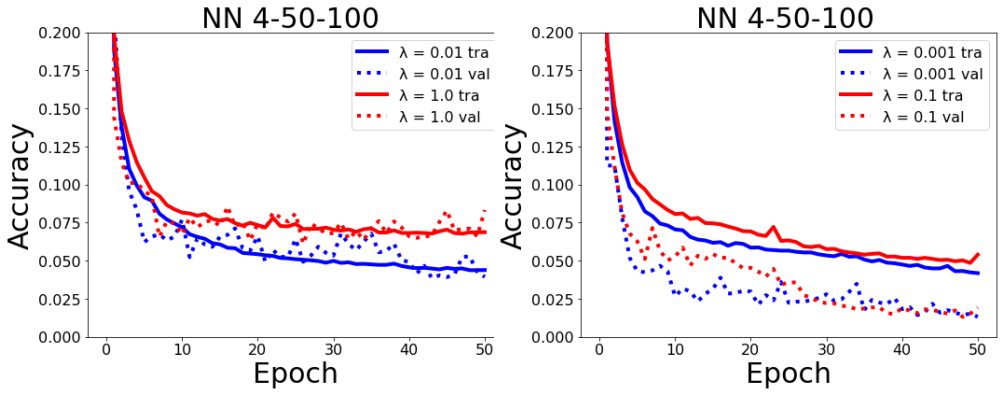
(a) (b)

Figura 3: Evolución de la función coste para diferentes valores de η. Resultados para una red neuronal [4, 50, 100], $m = 10$, *epoch* $= 50$, $\lambda = 0$.

La Figura 3.a corresponde a la primera prueba realizada para determinar el orden de magnitud adecuado del hiperparámetro η. Se observa como para $\eta = 0{,}01$ el proceso de aprendizaje es significativamente más lento, por lo que necesitaría más épocas para obtener resultados satisfactorios. En el caso de $\eta = 0{,}5$, la función desciende suavemente hasta alcanzar un mínimo ligeramente superior al de $\eta = 5{,}0$. Por otro lado, $\eta = 5{,}0$ disminuye rápidamente pero muestra ligeras oscilaciones aleatorias entorno al mínimo. Dichas oscilaciones indican que para valores de este orden de magnitud el sistema puede *sobrepasar* el mínimo. Esto resulta más evidente en la Figura 3.b, donde se ha ampliado la zona inferior del eje de ordenadas para $\eta = 1{,}0$, $\eta = 1{,}5$ y $\eta = 2{,}0$. De estos tres valores, seleccionaremos $\eta = 1{,}0$ como mejor estimación para el *learning rate* de nuestro modelo al ser el que menos oscilaciones presenta.

Una vez fijado como valor óptimo estimado $\eta = 1{,}0$ procedemos a estudiar el comportamiento de la red neuronal en función el término de regulación λ. Para ello vamos a utilizar los datos guardados como *validation data*. Conocemos que el propósito de λ es evitar un sobreentrenamiento del modelo mediante la penalización en la función coste de valores elevados de w. Dicho sobreentrenamiento se produce cuando la red neuronal *memoriza* características peculiares del *training data* perdiendo la capacidad de predecir correctamente nuevos datos. En la práctica, podemos identificar este problema representando la *accuracy*[5] evaluada sobre el *training* y *validation data*: si los resultados del *validation data* empeoran mientras que los del *training* siguen mejorando habrá *overfitting*.

En la Figura 4 se muestra el resultado de las pruebas realizadas para diversos valores de λ. Se observa como en todos los casos la *accuracy* del *validation data* se mantiene en niveles próximos a los del *training data* por lo que podemos descartar, en principio, un sobreentrenamiento de la red. Fijándonos en la gráfica de la derecha, el modelo parece mostrar una preferencia por los valores $\lambda = 0{,}001$ y $\lambda = 0{,}1$. Comparando ambas curvas se observa cierta saturación en las últimas épocas de $\lambda = 0{,}001$ por lo que seleccionamos como valor óptimo $\lambda = 0{,}1$.



(a) (b)

Figura 4: Evolución de la *accuracy* para diferentes valores de λ. Los datos en línea continua pertenecen al *training data*; en línea discontinua al *validation data*. Resultados para una red neuronal [4, 50, 100], $m = 10$, *epoch* $= 50$, $\eta = 1{,}0$.

Llegados a este punto es preciso remarcar que el proceso de optimización puede extenderse indefinidamente dado que a menudo los hiperparámetros son dependientes entre sí. Existen numerosos artículos que establecen posibles técnicas o recomendaciones para facilitar la búsqueda de hiperparámetros, sin embargo, la particularidad de cada problema hace inviable la creación de una única regla general a seguir. [6]

[5] En el código de Michael Nielsen diseñado para la clasificación de dígitos, la *accuracy* está definida como el porcentaje de respuestas correctas. En nuestro caso, trabajaremos con la *distancia* entre el valor esperado y el calculado por la red.

[6] Michael Nielsen. *Neural Networks and Deep Learning*. Pag. 116 - 117.

3.3. Discusión de resultados

A continuación, presentamos una selección de los resultados predichos por la red neuronal compuesta por ejemplos pertenecientes al *test data* (Figura 5). Comprobamos como la red diseñada, entrenada con los hiperparámetros escogidos ($\eta = 1{,}0$, $\lambda = 0{,}1$), es capaz de predecir correctamente la forma característica de la función Lorentziana.

Los ejemplos seleccionados muestran como para la mayoría de los casos el ajuste resulta excelente, si bien se aprecia una tendencia a presentar errores cuanto menor es la anchura de la curva. Este fenómeno se observa especialmente en el último ejemplo (derecha abajo), donde el pico es particularmente estrecho. La causa del fallo puede deberse a una dificultad de cálculo cuando $\sigma \approx 0$, lo cual afectaría también al ajuste de la altura. Además, debemos tener en cuenta el efecto de la discretización, ya que cuantos menos puntos abarque el pico más difícil será ajustarlo.

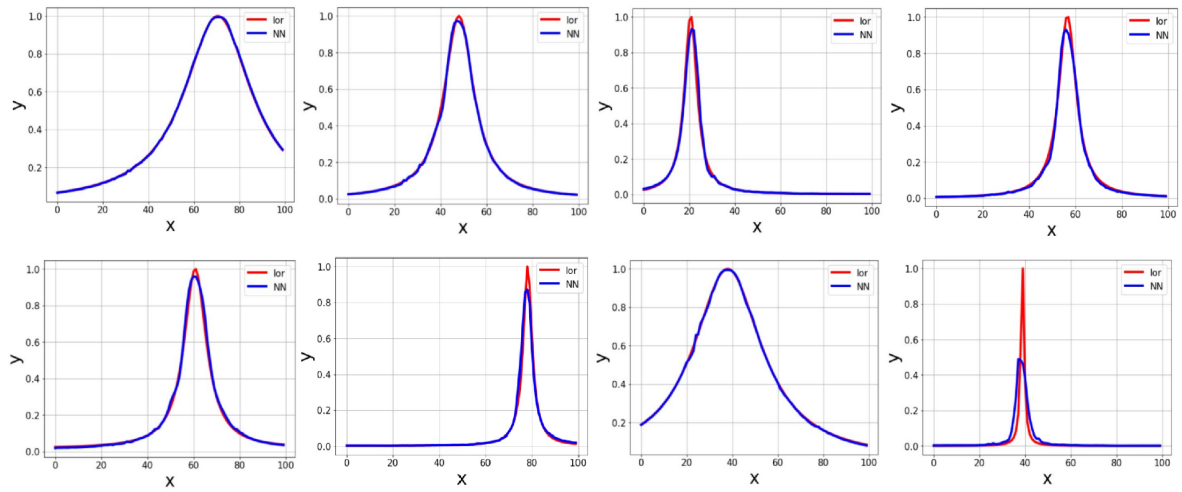


Figura 5: Selección representativa de los resultados pertenecientes al *test data* obtenidos con la red neuronal [4, 50, 100], $m = 10$, *epoch* $= 50$, $\eta = 1{,}0$, $\lambda = 0{,}1$. En rojo se muestra la función Lorentziana $L(x)$ teórica; en azul la predicción de la red.

Con el problema de la discretización en mente, podríamos intuir que una posible solución sería aumentar el número de puntos a la salida. Sin embargo, esto tendría un efecto negativo en el modelo al incrementar considerablemente el número de parámetros a optimizar y podría dar lugar a *overfitting*. En consecuencia, consideramos que la red neuronal diseñada aporta unos buenos resultados a pesar de sus limitaciones, ajustando correctamente la forma, la posición del pico y, salvo casos extremos, la altura y la anchura de la Lorentziana.

Finalmente, resulta interesante preguntarnos si el modelo ha *aprendido* realmente la fórmula de la función $L(x)$ (ec. 8) o si simplemente ha desarrollado un mecanismo a partir del ajuste de pesos y *bias* que le permite simular *algo que se comporta aproximadamente como la Lorentziana*. La respuesta no es sencilla, puesto que la propia definición de aprendizaje es algo abierto y actualmente todavía no es posible interpretar a la perfección lo que ocurre en el interior de una red neuronal artificial, especialmente en estructuras más complejas como las que veremos en los siguientes apartados.

4. Nanofotónica y machine learning

Una vez puestos en práctica los conceptos básicos del aprendizaje automático supervisado basado en redes neuronales, procedemos a estudiar sus posibles aplicaciones en problemas de mayor complejidad presentes en el campo de la *nanofotónica*. La nanofotónica es una ciencia emergente y multidisciplinaria que se ocupa del estudio de la luz y sus interacciones con la materia a escala nanométrica [6]. Su interés en las últimas décadas se ha visto fomentado por las innovaciones tecnológicas que suscita, siendo a menudo necesarias grandes capacidades de cálculo inaccesibles o ineficientes por los métodos computacionales tradicionales. Es aquí donde la nanofotónica encuentra un beneficioso nexo de unión con las técnicas más sofisticadas de *machine learning* [7].

Recientemente, se han publicado numerosos artículos que abordan diversas aplicaciones de las redes neuronales y el *deep learning* para la resolución de problemas de optimización e *inverse desing* [8], predicción de campos en nanoestructuras 3D [9] o modelización de circuitos cuánticos [10]. En lo que respecta a lo sucesivo de este trabajo, nos centraremos en analizar el fenómeno de la *transmisión óptica extraordinaria* (EOT) desde tres enfoques distintos: optimización, optimización inversa y análisis de grados de libertad.

4.1. Modelo teórico: Transmisión óptica extraordinaria

La EOT es un fenómeno descubierto por Ebbesen *et al* en 1998 [11] por el cuál se produce una transmisión óptica varios órdenes de magnitud superior a la esperada para determinadas longitudes de onda al propagarse una onda electromagnética a través de un array de huecos en películas de metal. Según estudios posteriores, su origen se encuentra en el acoplamiento resonante de los modos de superficie electromagnéticos mediante los campos evanescentes dentro de los huecos [12].

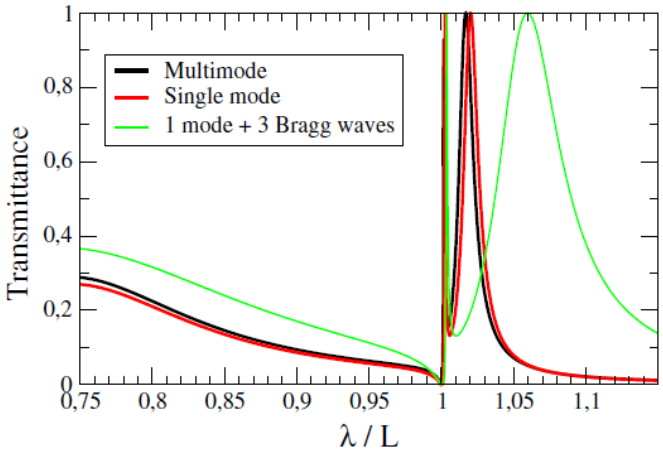


Figura 6: Transmitancia a través de un array de huecos en un conductor eléctrico perfecto. Las curvas de interés son: en negro, la curva exacta resultado de la convergencia de la expansión a multimodos; y, en rojo, la curva calculada por el modelo simplificado por aproximación al modo fundamental. La curva verde corresponde al modelo descrito en [12].
Fuente: L Martín-Moreno and F J García-Vidal 2008 J. Phys.: Condens. Matter 20 304214

Dentro de las investigaciones realizadas para caracterizar la EOT nos es de especial interés el modelo mínimo de transmisión óptica a través de películas de metal agujereadas propuesto por L. Martín-Moreno y F. J. García-Vidal [13]. En su artículo, se desarrolla bajo la aproximación al modo de onda fundamental y el tratamiento de la película de metal como conductor eléctrico perfecto[7] un conjunto de ecuaciones simplificadas que permiten calcular de manera efectiva el espectro de transmitancia óptica (Figura 6).

Como sistema bajo estudio vamos a utilizar el modelo de la Figura 7: un array infinito de huecos situados periódicamente en una película de metal de espesor h sobre la que incide una onda plana con polarización σ_0 y vector de onda k_0. Los huecos en el metal forman una red cuadrada de parámetro de red L y lado a. Se diferencian tres regiones caracterizadas por su constante dieléctrica: I, II y III. Tomando como referencia $\epsilon_I = \epsilon_{III} = 1$, el modelo permite caracterizar el espectro de transmisión extraordinaria con cuatro parámetros: h, L, a y ϵ_2[8].

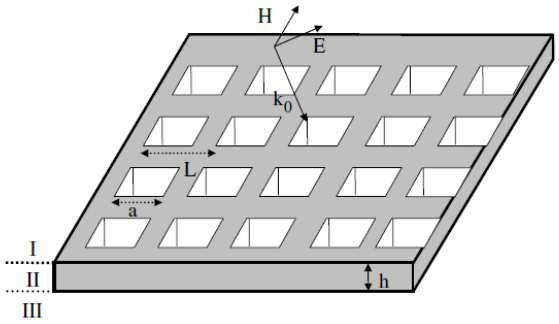


Figura 7: Esquema del modelo: una onda plana p-polarizada de vector de onda k_0 incide desde el medio I sobre un array de huecos en forma de red cuadrada de parámetro L y lado a situado en una película de metal de espesor h (II). La onda se propaga por los huecos transmitiéndose al medio III.
Fuente: L Martín-Moreno and F J García-Vidal 2008 J. Phys.: Condens. Matter 20 304214

[7] En un conductor eléctrico ideal (PEC) la constante dieléctrica $\epsilon_M = -\infty$, de forma que el campo no penetra en su interior.

[8] La constante dieléctrica ϵ_2 se corresponde con la del interior de los huecos.

4.2. Optimización de espectros EOT

Con el modelo mínimo para la EOT introducido en la anterior sección se pretende diseñar una red neuronal que dada una serie de parámetros calcule el espectro de transmisión extraordinaria correspondiente. En concreto, los parámetros de entrada serán los mínimos necesarios para caracterizar el sistema: el espesor de la lámina h, el parámetro de red L, el lado de los cuadrados a y la constante dieléctrica de los huecos ϵ_2.

El principio de funcionamiento del algoritmo será similar al desarrollado para la función Lorentziana, siendo en esta ocasión el *output* deseado el espectro de transmisión óptica computado según las ecuaciones del modelo mínimo [13]. Dado el considerable aumento de complejidad, será necesario construir una red con un mayor número de neuronas y varias capas ocultas. Esto supondrá un notable incremento de la cantidad de parámetros a optimizar por lo que, para mejorar la eficiencia del algoritmo, utilizaremos algunas de las herramientas especializadas que ofrece el entorno *TensorFlow*[9].

4.2.1. Planteamiento del algoritmo: DNNs

El primer paso a la hora de construir un modelo que resuelva correctamente el problema planteado será elegir la estructura de la red neuronal. Anteriormente se ha trabajado con una red simple formada únicamente por una capa oculta y neuronas de tipo *sigmoid*. Ahora vamos a introducir dos conceptos nuevos: redes neuronales profundas y neuronas *ReLU*.[10]

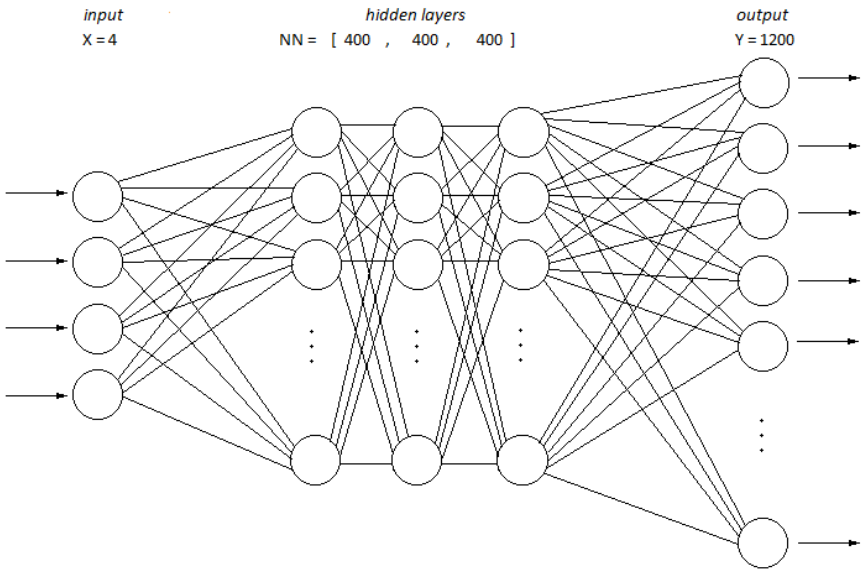


Figura 8: Esquema de una red neuronal profunda de dimensión [4, 400, 400, 400, 1200].

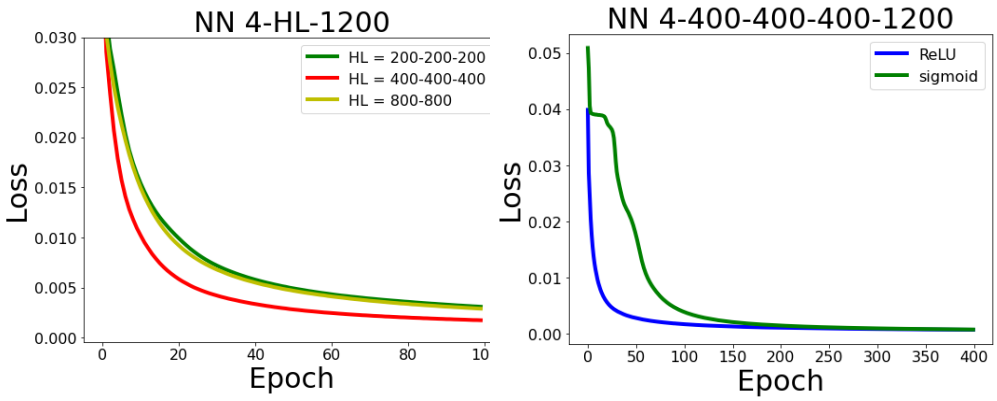
[9]Plataforma de código abierto desarrollada por Google Brain para hacer frente a problemas complejos de aprendizaje automático.

[10]Las redes neuronales profundas o *deep neural networks* (DNNs) se caracterizan por tener múltiples capas ocultas entre las capas de entrada y salida. Esto permite dividir el problema en sucesivos niveles de abstracción, si bien presenta un aspecto negativo al dificultar el entrenamiento en el llamado *desvanecimiento del gradiente*. Para solucionar este problema, se opta por el uso de neuronas *ReLU* (*Rectified Linear Unit*) que utilizan como función de activación el rectificador $f(x) = max(0, x)$. Ver [5, *Chapter 5: Why are deep neural networks hard to train?*]

Nuevamente generaremos los datos sobre los que se entrenará el modelo en una subrutina aparte. Cada ejemplo estará formado por dos arrays: uno de dimensión 4 (h, L, a, ϵ_2) y otro de 1200 puntos correspondientes al espectro discretizado. Los parámetros serán generados aleatoriamente entre los rangos $h = [0.2, 0.5]$, $L = [0.9, 1.1]$, $a = [0.3, 0.5]$, $\epsilon_2 = [1, 3]$ y posteriormente normalizados. Análogamente al primer problema, dividiremos los ejemplos en los sets: *training data*, *validation data* y *test data*. Debido al coste computacional que requerirá entrenar el nuevo modelo, limitaremos la cantidad de ejemplos a 15000, 3000 y 100.

Como función de coste utilizaremos el error cuadrático medio (*loss_function* = '*mse*'). Los *weights* y *bias* serán inicializados con las funciones *tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.005, seed=None)* y *tf.keras.initializers.Constant(0.5)*, respectivamente. El proceso de optimización se realizará mediante el algoritmo *Adam*[11] con un *learning rate* $\eta = 0{,}0001$. Tomaremos como *minibatch* $m = 10$ y excluiremos del análisis el hiperparámetro λ.

Para determinar la estructura de red más adecuada realizamos pruebas con tres modelos: [4, 200, 200, 200, 1200], [4, 400, 400, 400, 1200] y [4, 800, 800, 1200].[12] Los resultados mostrados en la Figura 9.a indican una preferencia por el modelo [4, 400, 400, 400, 1200]. Asimismo, analizamos el comportamiento de dicha estructura según la función de activación de las capas ocultas (Figura 9.b). La evolución de la función coste (*loss*) muestra como durante las primeras épocas las neuronas tipo *ReLU* ofrecen un mejor resultado frente a las *sigmoid*, minimizando el coste rápidamente de forma regular. Este hecho concuerda con el esperado problema de desvanecimiento del gradiente, según el cual las neuronas *sigmoid* en capas profundas son más difíciles de entrenar al presentar sus derivadas valores muy pequeños fuera de su zona lineal. No obstante, al continuar entrenando ambos modelos encontramos que se alcanza una situación similar en los dos casos.



(a) (b)

Figura 9: (a) Evolución de la función coste según diferentes estructuras de capas ocultas (HL) con función de activación *ReLU*. (b) Representación del coste para una red neuronal [4, 400, 400, 400, 1200] con neuronas tipo *ReLU* (azul) o *sigmoid* (verde) en las capas ocultas.

[11] Variante del SGD más eficiente para redes neuronales profundas. Permite adaptar individualmente el *learning rate* asociado a cada parámetro mediante la estimación de primeros y segundos momentos del gradiente.

[12] En todos los modelos la última capa estará formada por neuronas *sigmoid* para facilitar la evaluación del *output* a la salida.

4.2.2. Visualización de resultados

A continuación, se muestra una selección de los resultados predichos por los modelos [4, 400, 400, 400, 1200] *ReLU* y *sigmoid* para los mismos ejemplos del *test data* tras 400 épocas de entrenamiento. En cada gráfica se ha representado el espectro de transmisión EOT (calculado por el modelo mínimo) junto con las predicciones correspondientes. Adicionalmente, se indican los parámetros de entrada h, L, a y ϵ_2 normalizados que han dado lugar a cada espectro.

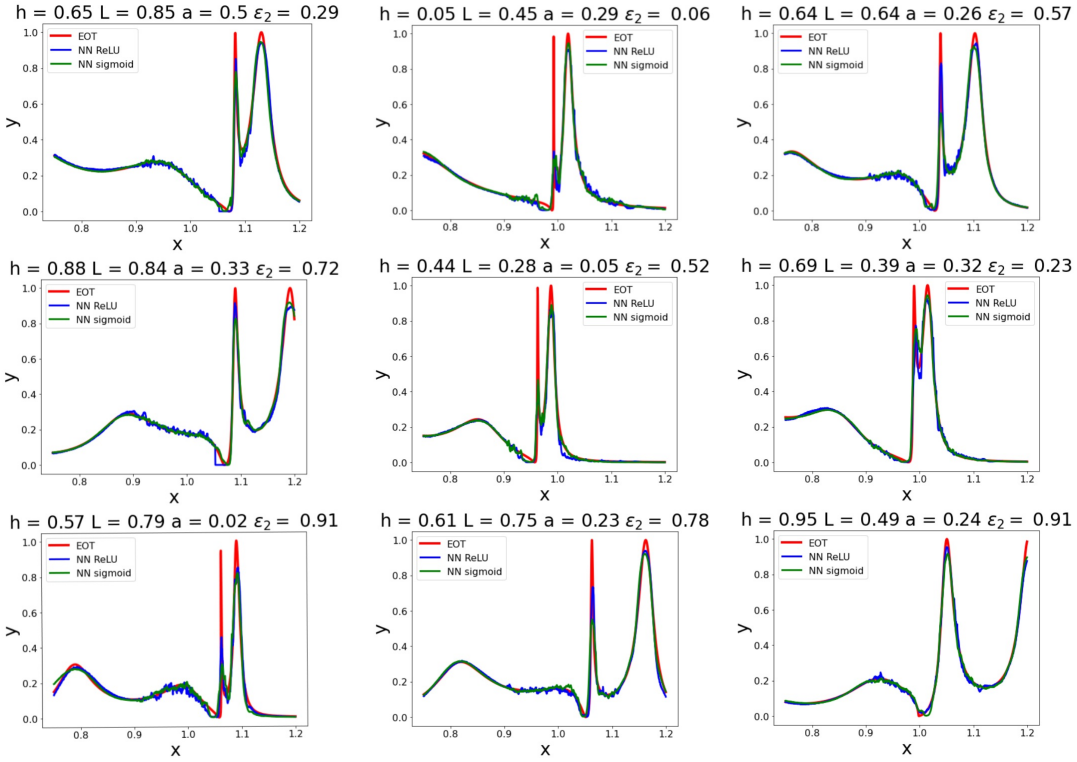


Figura 10: Selección de resultados pertenecientes al *test data*: espectro teórico EOT (rojo), predicciones con el modelo NN = [4, 400, 400, 400, 1200] *ReLU* (azul) y *sigmoid* (verde).

A simple vista resulta notable la gran similitud que las predicciones de ambos modelos guardan con el espectro real, reproduciendo de forma certera la posición de los picos, su anchura y la silueta de la curva inicial. Fijándonos en detalle, destaca la habilidad de identificar correctamente la presencia de un primer pico más agudo en aquellos casos en los que la separación entre ellos es mínima. En cuanto a la altura, ambos modelos encuentran dificultades para representar los picos más estrechos, si bien cabe destacar que el ajuste con *ReLU* es ligeramente mejor al *sigmoid*. Por otro lado, se observa un error persistente antes del mínimo en espectros con el pico inicial entorno a $x = 1{,}1$. Este fallo es más acentuado en el modelo *ReLU*, tal y como puede verse en la Figura 10.d. La razón detrás de este comportamiento es desconocida y podría dar lugar a futuros estudios.

4.3. Optimización inversa

En esta sección se va a abordar el problema de optimización de manera inversa, es decir, se pretende que dado un espectro de transmisión concreto la red neuronal devuelva los valores de los parámetros que lo generaron. Dado que se trata de un problema espejo al anterior, utilizaremos los mismos conjuntos de datos ordenados de forma inversa: el *input* serán los 1200 puntos del espectro y los datos de comparación a la salida se corresponderán con h, L, a, y ϵ_2.

4.3.1. Modificaciones del algoritmo: CNNs

Inicialmente seleccionamos como base una red neuronal profunda similar a la anterior de dimensión [1200, 400, 400, 400, 4]. De igual modo, analizaremos el comportamiento del modelo para las funciones de activación tipo *ReLU* y *sigmoid* en las capas ocultas. En adición a esto, vamos a introducir un nuevo tipo de estructura: las redes neuronales convolucionales (CNNs). Las CNNs se utilizan comúnmente en problemas de detección, reconocimiento y clasificación de imágenes, siendo claves actualmente en el desarrollo de la visión artificial. Gracias a su estructura interna formada por capas convolucionales alternadas con capas de reducción, son capaces de identificar características simples (curvas, bordes, etc.) y componer sucesivamente elementos más complejos.

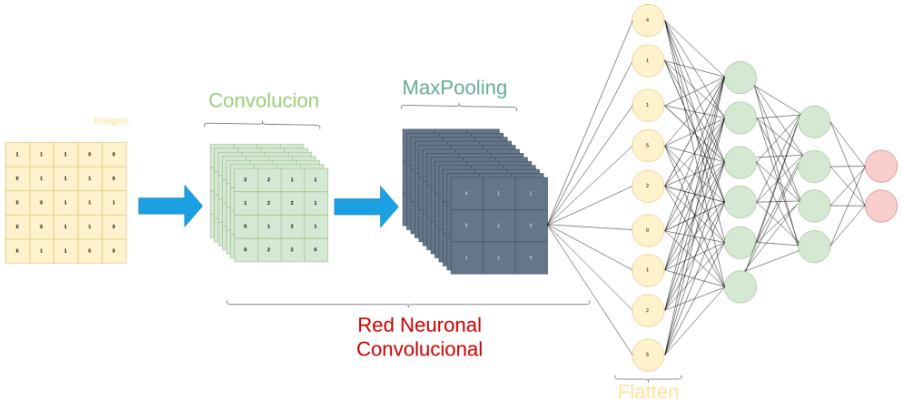


Figura 11: Esquema de la estructura una red neuronal convolucional (CNN). En líneas generales, el proceso de convolución consiste en recorrer de forma iterativa con un detector de propiedades (llamado *kernel* o filtro) un conjunto de pixeles de una imagen, guardando el resultado de la operación en un mapa de propiedades. Tras la serie de filtros, se aplica la función rectificadora ($ReLU$) para romper la linealidad de la imagen resultante por convolución. A continuación, se realiza el proceso de *Max Pooling* para dotar a la red de cierta flexibilidad ante pequeñas variaciones de objetos. Finalmente, se conecta la salida a una red neuronal mediante una capa de *flattening*. Fuente: *Introducción al deep learning parte 2: Redes Neuronales Convolucionales.*

La Figura 11 muestra un ejemplo de convolución 2D diseñado para el tratamiento de imágenes. En nuestro caso, la CNN que vamos a implementar estará formada por capas convolucionales de tipo Conv1D debido a que trabajamos con secuencias de datos unidimensionales[13]. Basándo-

[13]Existen también las capas Conv3D especializadas en imágenes 3D con aplicaciones en resonancia magnética (MRI) y tomografía computarizada (CT).

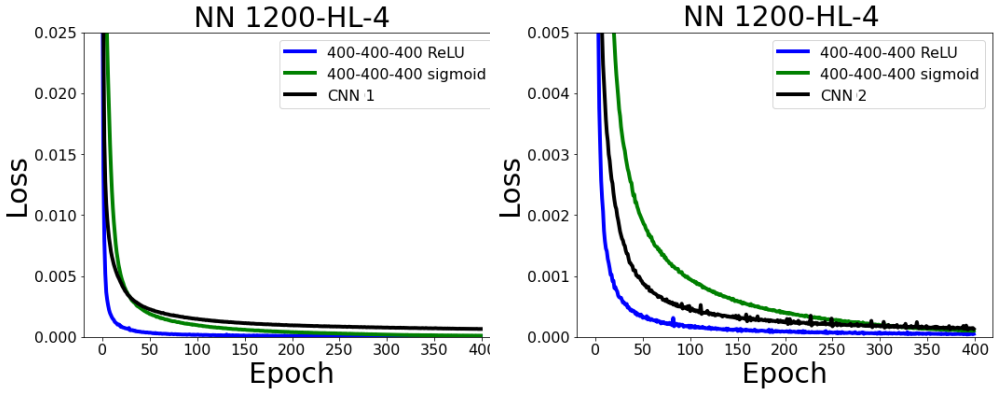
nos en la capacidad de detección de propiedades que presentan este tipo de estructuras, se busca que los filtros sean capaces de identificar características del espectro tales como la posición del mínimo y la separación entre picos, empleando un número de parámetros menor al de modelos anteriores.

De esta forma, construimos dos CNNs diferentes: la primera estará formada por tres capas convolucionales con número de filtros [16, 32, 64] (CNN 1) y la segunda añadirá una cuarta capa con 128 filtros (CNN 2). Ambas utilizaran *kernel* de dimensión 7, función de activación *ReLU* y estarán conectadas tras el proceso de *flattering* a una red formada por una capa de neuronas tipo *ReLU* de dimensión 15 y, por último, 4 neuronas *sigmoid* como *output*. En la Tabla 1 se muestra la comparativa entre todos los modelos, indicando el número total de parámetros y el tiempo promedio requerido de entrenamiento[14].

	n^o *param*	*s/epoch*	$\mu s/sample$
ReLU	802804	10	675
sigmoid	802804	10	672
CNN 1	38383	13	897
CNN 2	644015	88	6000

Tabla 1

Cada modelo se ha entrenado siguiendo el método de *backpropagation* con optimizador *Adam*, *learning rate* $\eta = 0{,}0001$, *loss* = *mse*, *minibatch* = 10 y *epoch* = 400. Los resultados de la función coste indican como el modelo que más rápido minimiza es el *ReLU* 400-400-400, seguido por el CNN 2. Nuevamente el modelo *sigmoid* presenta un peor comportamiento en las épocas iniciales, aunque consigue alcanzar entorno a la época 30 al CNN 1 y en la 300 al CNN 2.



(a) (b)

Figura 12: Evolución de la función coste según diferentes estructuras de capas ocultas (HL).

En vista de estas gráficas, si tenemos en cuenta que el tiempo de cálculo necesario para entrenar las redes convolucionales ha sido similar (CNN 1) o significativamente mayor (CNN 2) al de los modelos de capas profundas, no podemos concluir que la reducción del número de parámetros suponga un beneficio práctico en el problema que nos ocupa.

[14]El tiempo de cálculo puede variar según las capacidades del ordenador utilizado.

4.3.2. Interpretación de resultados

Para visualizar los resultados obtenidos en el anterior apartado, optamos por representar los valores estimados por las redes neuronales (h_{eff}, L_{eff}, a_{eff} y ϵ_{2eff}) frente a los parámetros teóricos. De esta forma, las parejas de datos se distribuirán idealmente como puntos entorno a una recta de pendiente unidad. Cuanto menor sea la desviación de la recta, mejor será el ajuste de dicho parámetro.

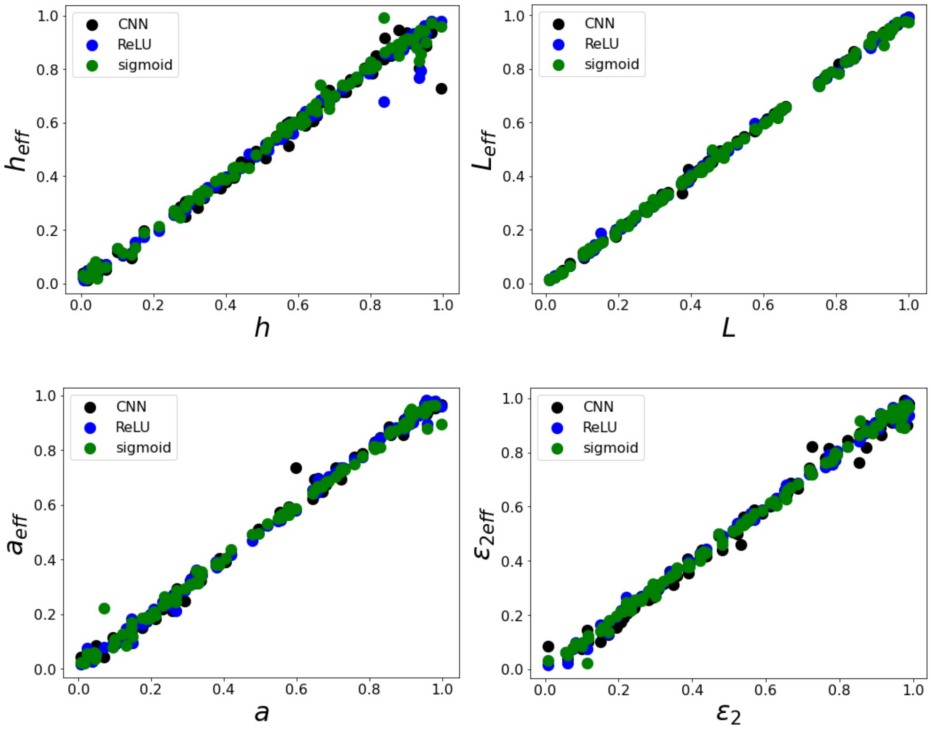


Figura 13: Resultados obtenidos con los modelos CNN 2 (negro), *ReLU* (azul) y *sigmoid* (verde).

En la Figura 13 se muestran los diagramas obtenidos para cada parámetro con los modelos CNN 2[15], *ReLU* y *sigmoid*. Comparando cada gráfica se aprecia claramente que el parámetro que presenta menos desviación es L, lo cual indica que las redes detectan correctamente el mínimo del espectro. En cuanto al resto de parámetros, se advierte una mayor desviación en h y ϵ_2, especialmente en los valores extremos. Esto podría tener una explicación física ya que dichos parámetros están relacionados entre sí en la propagación de la luz en el interior de los huecos. Adicionalmente, la separación entre picos depende de a, h y ϵ_2, por lo que la desviación encontrada podría ser un indicador de que los modelos son capaces de detectar una relación entre los diferentes parámetros.

[15]Dado que en esta ocasión se van a mostrar conjuntamente todos los ejemplos pertenecientes al *test data*, decidimos excluir de la representación al modelo CNN 1 ya que no ofrece diferencias apreciables en los resultados respecto a su homólogo.

4.4. Análisis de grados de libertad: autoencoder

En esta última sección, se pretende realizar un estudio cuantitativo de los grados de libertad presentes en el problema de la transmisión óptica extraordinaria. Para ello nos valdremos de un tipo particular de red neuronal denominada *autoencoder*. La filosofía detrás de dicha estructura se basa en la auto-codificación, es decir, la capacidad de reducir al mínimo una cantidad de información (codificación) haciendo posible su reconstrucción (descodificación). Son utilizados habitualmente para el aprendizaje de propiedades o *feature learning*[16] en problemas de reducción dimensional, detección de anomalías, procesamiento de imágenes y traducción automática.

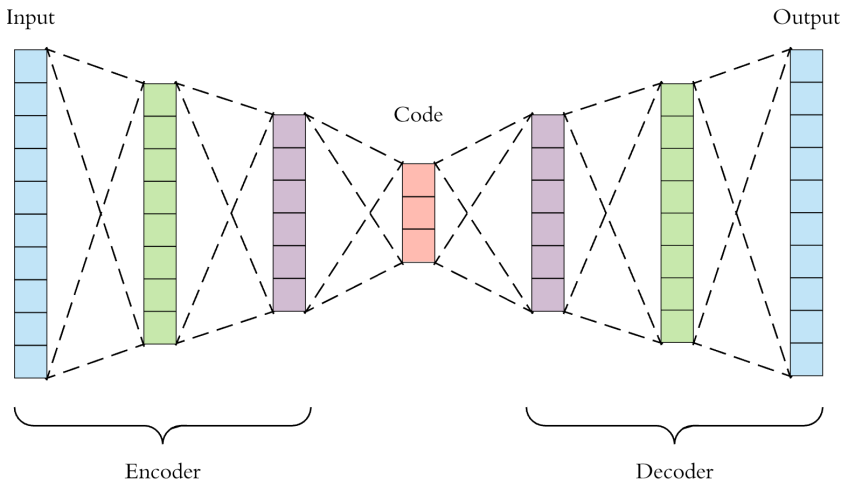


Figura 14: Esquema de un *autoencoder*. Se diferencian dos subestructuras: *encoder* y *decoder*. El *encoder* lleva a cabo el proceso de codificación de los datos de entrada hasta la capa central o *code*. El *decoder* se encarga del proceso de descodificación, reconstruyendo la información en la salida. Fuente: *Deep Learning Básico Parte Final: Autoencoders.*

Nuestro objetivo será construir un *autoencoder* que emplee como datos de entrada y salida los espectros de EOT generados anteriormente, de forma que las capas de codificación y descodificación serán análogas a las de los modelos de optimización (inversa - codificación, directa - descodificación). El punto de interés será la capa central en la que se comprimirá al máximo la información. Intuitivamente, podemos interpretar cada nodo como un grado de libertad. Así, si la dimensión de la capa central es menor al número necesario de parámetros independientes que definen el problema, la red no será capaz de reconstruir correctamente el espectro a la salida.

A priori esperamos que el número de nodos necesarios sea 4 al ser esta la cantidad de parámetros con los que hemos generado los espectros según el modelo mínimo de la EOT: h, L, a y ϵ_2. Para comprobar si esto es correcto en la práctica, vamos a entrenar con las mismas series de datos y parámetros habituales[17] un conjunto de *autoencoders* de estructura general [1200, 400, 400, X, 400, 400, 1200] con dimensión central variable.

[16] El *feature learning*, también conocido como *representation learning*, es un conjunto de técnicas por las cuales un sistema aprende automáticamente las representaciones necesarias para la detección o clasificación de características a partir de datos sin procesar.

[17] Los utilizados anteriormente: $loss = mse,\ opt = Adam,\ \eta = 0{,}0001,\ minibatch = 10,\ epoch = 400$.

Examinando los valores finales de las respectivas funciones de coste representados en la Figura 15 se aprecia una mejora gradual conforme aumenta el número de nodos en la capa central. El coste disminuye hasta alcanzar un mínimo entorno a X = 4, valor a partir del cual no se producen cambios significativos. De nuevo se ha realizado la prueba con neuronas tipo *ReLU* y *sigmoid*, obteniendo en ambos casos un comportamiento similar alrededor del mínimo. Cabe destacar la diferencia para el extremo X = 1, donde en el caso *sigmoid* la función coste apenas se aleja de los valores iniciales.

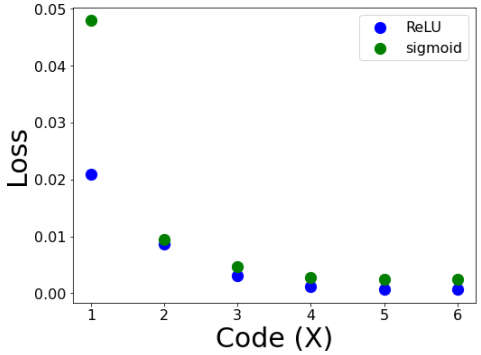
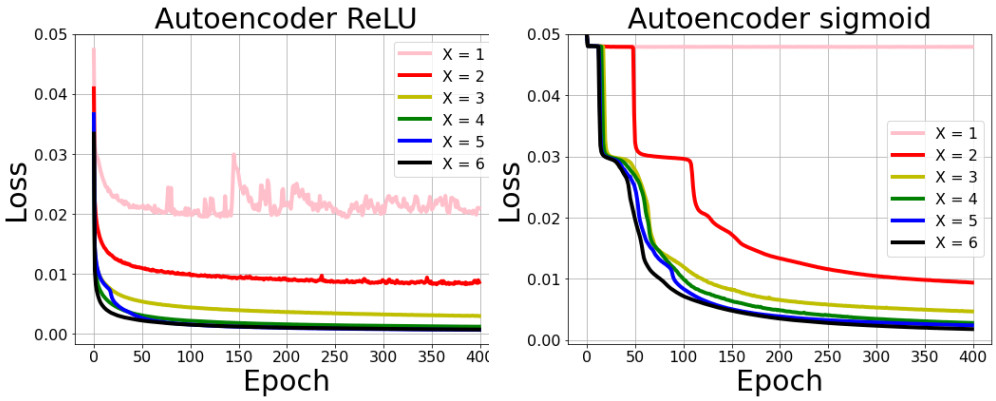


Figura 15: Valores finales de la función coste para cada *autoencoder* tras 400 épocas.

En vista de la gráfica anterior, podríamos preguntarnos si X = 3 es un valor suficientemente cercano al mínimo como para considerar la posibilidad de reducir el problema a solo tres grados de libertad. En primer lugar, podemos ver analizando las gráficas de la Figura 16 que mientras X = 1 y X = 2 muestran serias dificultades a la hora de minimizar la función, X = 3 consigue aproximarse al comportamiento de los siguientes modelos manteniendo siempre una cierta distancia. Esta magnitud no es fácil de traducir en algo cualitativo, ya que los valores de la función coste no tienen una equivalencia directa a escala con el problema. Sin embargo, conociendo teóricamente el modelo de la EOT podemos interpretar la razón de estos resultados.



(a) (b)

Figura 16: Evolución de la función coste según la dimensión de la capa central X. Resultados para red [1200, 400, 400, X, 400, 400, 1200] con función de activación (a) *ReLU* y (b) *sigmoid*.

Como ya se mencionó en la discusión de la sección 4.3.2. existen características del espectro que dependen únicamente de uno o varios parámetros. Por un lado, el mínimo viene dado por L y es independiente al resto de valores, por lo que necesariamente ocupará uno de los nodos de la constricción. Por su parte, la admitancia en el agujero viene dada por $Y = k_z/g$,[18] donde

$$k_z = \sqrt{\epsilon_2 \; g^2 - (\frac{\pi}{a})^2} \tag{10}$$

que se relaciona a su vez con el espesor h en la exponencial e_h empleada para calcular los coeficientes de Fresnel $r_{k\sigma}$ y $t_{k\sigma}$:

$$e_h = \exp\left(ik_z h/2\right) \tag{11}$$

Atendiendo a estas ecuaciones, no resultaría desencaminado pensar que la red neuronal haya podido dar con una combinación de valores que le permita reconstruir gran parte del espectro a partir de solo tres parámetros. Esto puede observarse en los ejemplos mostrados en la Figura 17, donde se aprecia como a pesar de aproximarse al espectro teórico en ciertos aspectos, X = 3 falla al representar la posición del primer pico e incluso en ocasiones directamente no lo incluye. La razón física por la que intuimos que sucede esto es porque mientras la propagación puede definirse a través de tres parámetros, se necesita un cuarto para incluir el efecto de la reflexión que da lugar al primer pico.

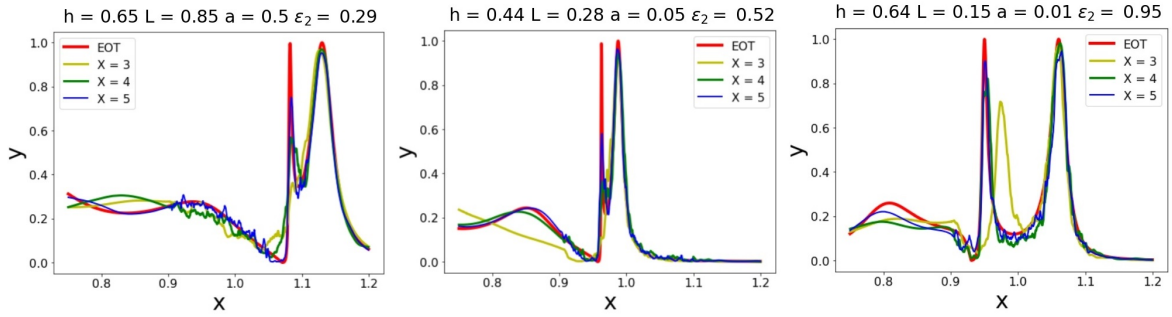


Figura 17: Selección de ejemplos del *test data*: espectro teórico EOT (rojo), reconstrucción con *autoencoder* X = 3 (amarillo), X = 4 (verde) y X = 5 (azul).

En la anterior figura podemos ver también que X = 5 no supone una mejora significativa respecto a X = 4. Si nuestro modelo fuera exacto, podríamos comprobar como para valores iguales o mayores de cuatro en la constricción la función de coste tendería a valer cero. No obstante, la realidad es que el modelo diseñado no es perfecto y la función coste solo nos da una información orientativa cuya interpretación es subjetiva.

[18] Se define $g = \omega/c$ con ω la frecuencia del campo y c la velocidad de la luz.

5. Conclusiones

En la era del *big data*, son indiscutibles los beneficios que el uso de la Inteligencia Artificial puede aportar en la resolución de los problemas cada más complejos que presenta el desarrollo de nuevas tecnologías físicas. En este trabajo se ha intentado introducir de manera sencilla a los conceptos básicos sobre los que se sustentan los algoritmos de aprendizaje automático actuales, poniendo especial énfasis en las estructuras de redes neuronales y su aplicación en problemas de nanofotónica.

En un primer acercamiento, se ha entrenado una red neuronal simple que ajusta la función Lorentziana, sirviendo como ejemplo práctico de elección de hiperparámetros. Posteriormente, hemos explorado algunas de las posibles aplicaciones del *Machine Learning* en nanofotónica tomando como problema base el modelo mínimo de la EOT. Las pruebas realizadas muestran como las DNNs pueden ser de gran utilidad tanto en la predicción de espectros como en problemas de optimización inversa. Es en esta última categoría donde pensamos que las características de las redes neuronales pueden ser especialmente beneficiosas a la hora de determinar el valor de los parámetros necesarios para obtener una respuesta deseada. Finalmente, se ha visto como los *autoencoders* pueden emplearse para corroborar o incidir nuevas propiedades de modelos físicos desde un punto de vista cuantitativo complementario a los desarrollos teóricos.

En conclusión, consideramos que hoy en día la IA constituye una herramienta clave que ofrece interesantes oportunidades de encontrar nuevas soluciones a problemas conocidos y posee la capacidad de plantear potencialmente cuestiones no formuladas hasta ahora.

6. Bibliografía

1. Alan Turing. Computing machinery and intelligence. Mind, 1950.

2. Pankaj Mehta, Ching-Hao Wang, Alexandre G. R. Day, and Clint Richardson. A high-bias, low-variance introduction to Machine Learning for physicists. 2019.

3. Richard Nagyfi. The differences between Artificial and Biological Neural Networks. Towards Data Science, 2018.

4. F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. Psychological Review, 1958.

5. Michael A. Nielsen. Neural Networks and Deep Learning.

6. Novotny L, Hecht B. Principles of nano-optics. Cambridge, Cambridge University Press, 2012.

7. Kan Yao, Rohit Unni and Yuebing Zheng. Intelligent Nanophotonics: Merging Photonics and Artificial Intelligence at the Nanoscale. 2018.

8. J. Peurifoy, Y. Shen, L. Jing, Y. Yang, F. Cano-Renteria, B. G. DeLacy, J. D. Joannopoulos, M. Tegmark, M. Soljačić. Nanophotonic particle simulation and inverse design using artificial neural networks. Sci. Adv. 4, eaar4206 (2018).

9. Peter R. Wiecha and Otto L. Muskens. Deep learning meets nanophotonics: A generalized accurate predictor for near fields and far fields of arbitrary 3D nanostructures.

10. Alec Michael Hammond. Machine Learning Methods for Nanophotonic Design, Simulation, and Operation.

11. Ebbesen, T., Lezec, H., Ghaemi, H. et al. Extraordinary optical transmission through sub-wavelength hole arrays. Nature 391, 667–669 (1998).

12. L. Martín-Moreno, F. J. García-Vidal, H. J. Lezec, K. M. Pellerin, T. Thio, J. B. Pendry, and T. W. Ebbesen. Phys. Rev. Lett. 86, 1114.

13. L Martín-Moreno and F J García-Vidal 2008 J. Phys.: Condens. Matter 20 304214.

14. Introducción al deep learning parte 2: Redes Neuronales Convolucionales.

15. Deep Learning Básico Parte Final: Autoencoders. https://medium.com/@jcrispis56/deep-learning-básico-parte-final-autoencoders-3d5c6fff2966