**REAL ACADEMIA DE CIENCIAS EXACTAS, FÍSICAS,
QUÍMICAS Y NATURALES DE ZARAGOZA**

INTELIGENCIA ARTIFICIAL APLICADA A LAS CIENCIAS: CONCEPTOS BÁSICOS Y EJEMPLOS

DISCURSO DE INGRESO LEÍDO POR EL ACADÉMICO ELECTO

Ilmo. Sr. D. LUIS MARTÍN MORENO

*EN EL ACTO DE SU RECEPCIÓN SOLEMNE
CELEBRADO EL DÍA 25 DE MAYO DEL AÑO 2022*

Y

DISCURSO DE CONTESTACIÓN POR EL

Ilmo. Sr. D. MANUEL ASOREY CARBALLEIRA

ACADÉMICO NUMERARIO

ZARAGOZA

2022

Depósito legal: Z 769-2022

Imprime:
Servicio de Publicaciones. Universidad de Zaragoza

INTELIGENCIA ARTIFICIAL APLICADA A LAS CIENCIAS: CONCEPTOS BÁSICOS Y EJEMPLOS

POR EL

Ilmo. Sr. D. LUIS MARTÍN MORENO

Excmo. Sr. Presidente

Ilmos. Sras. y Srs. Académicos

Señoras y Señores

Es para mí un honor aceptar la invitación de ingreso en la Real Academia de Ciencias de Zaragoza. Agradezco de corazón a todos los académicos esta invitación, y en especial a los de la sección de Físicas, que han promovido mi elección. Lamento profundamente el fallecimiento de Rafael Navarro, que me consta que apoyó esta invitación y que, tristemente, no podrá leer estas líneas de agradecimiento.

También quiero recordar a mi antecesor en la medalla académica, el profesor Rafael Núñez-Lagos. Rafael Núñez-Lagos tras licenciarse en la Universidad Complutense de Madrid en 1958, y completar su doctorado en la misma universidad en 1961, pasó a trabajar como investigador en la Junta de Energía Nuclear hasta 1967. Su extensa carrera ha estado ligada a la universidad, habiendo sido Catedrático de Física Nuclear en Granada, Sevilla y, desde 1970, en la Universidad de Zaragoza, además de haber realizado estancias temporales en Caltech (USA), ICTP (Trieste, Italia) y en el CERN. Ha dirigido seis tesis doctorales y ha publicado más de un centenar de artículos, en especial sobre neutrinos (en concreto sobre la doble desintegración beta) y sobre la búsqueda de materia oscura. Es de destacar la labor de gestión realizada por Núñez-Lagos, que ha sido Vicedecano y Decano en Funciones en la Facultad de Ciencias de la Universidad de Zaragoza, Vicepresidente y socio de Honor de la Real Sociedad Española de Física (donde recibió la Medalla en 2004), Vicepresidente de la Comisión Española del Consejo Internacional de la Ciencia, y Académico numerario de esta Academia, habiendo sido Presidente de la Sección de Física. Su labor impulsora de la Ciencia queda claramente reflejada en haber sido Co-fundador del Laboratorio Subterráneo de Canfranc, Presidente del Laboratorio del Amplificador de Energía S.A. (LAESA), y el Fundador del Laboratorio de Bajas Actividades (LABAC) de la Universidad de Zaragoza, acreditado para medir la radiactividad alfa total y beta total del agua de consumo humano, ríos y lagos.

Quiero agradecer a mi familia, tanto a mis padres, como a mi pareja Carmen y a mi hijo Yago, el apoyo constante para poder dedicarme a la investigación, y el respetar mis periodos de concentración y distracción.

Mi formación universitaria se realizó en la Universidad Autónoma de Madrid, donde cursé la licenciatura y realicé mi tesis. Quiero agradecer la confianza, el ejemplo y la formación que recibí durante la misma de mi director, Jose Antonio Vergés. Posteriormente,

tras estancias post-doctorales en la Universidad de Cambridge (Reino Unido), Imperial College de Londres y en el Instituto de Ciencia de Materiales de Madrid, llegué a Zaragoza en 1995 al conseguir una plaza de Profesor Titular en el Departamento de Física de la Materia Condensada de la Universidad de Zaragoza. En 2008 entré en el Consejo Superior de Investigaciones Científicas, manteniendo mi despacho en el antiguo Instituto de Ciencia de Materiales de Aragón (ahora, Instituto de Nanociencia y Materiales de Aragón). Me gustaría aprovechar la ocasión para agradecer las enseñanzas recibidas y el aprendizaje conjunto con diversos científicos con los que ha sido un placer colaborar a lo largo de mi carrera después de mi tesis, como Carlos Tejedor, John Pendry, Thomas Ebbesen, Francisco José García-Vidal, Luis Brey y Paco Guinea. Mención especial merecen todos las personas que en algún momento han formado parte del grupo de Nanofotónica del Instituto de Nanociencia y Materiales de Aragón, gracias a las cuales he ido aprendiendo y descubriendo aspectos apasionantes sobre la interacción entre la luz y la materia, que ha sido mi principal área de investigación en los últimos 25 años. En el tema de Inteligencia Artificial, quiero agradecer a Sergio G. Rodrigo y a David Zueco el aprendizaje conjunto y el haber dedicado tiempo y esfuerzo a preparar el curso en Inteligencia Artificial, que hemos empezado a impartir en la Universidad de Zaragoza.

En mi disertación haré una brevísima introducción a algunos conceptos básicos de la Inteligencia Artificial. En el discurso de entrada a esta Academia, nos recordaba el Prof. Manuel Asorey el aforismo por el cual en la antigua URSS los físicos más veteranos recomendaban a los más jóvenes no dedicarse a la Gravitación antes de haber cumplido 60 años. Bien poco me queda para poder dedicarme a ese tema tan interesante. Mientras tanto, he encontrado en la Inteligencia Artificial un campo apasionante en el que, como pretenden ilustrar los ejemplos que mostraré, existen aplicaciones directas a diversos ámbitos en Ciencia, además de hacernos reflexionar sobre qué es, y cómo funciona, la inteligencia.

1. Introducción

El extraordinario desarrollo de las capacidades de cómputo que se inició a finales de los años 40 del siglo pasado ha motivado la consideración de cuáles son los límites de la computación y, en concreto, de si es posible crear una inteligencia artificial similar (o incluso superior) a la humana. Las primeras previsiones sobreestimaron la velocidad y el alcance de lo que se podría conseguir en un corto periodo de tiempo y preveían un futuro inmediato distópico (ampliamente representado en la literatura y el cine) donde los humanos eran superados por las máquinas [1]. Esas expectativas no se cumplieron y la investigación en Inteligencia Artificial (IA) sufrió uno sus primeros "inviernos", en los que la comunidad científica (y los agentes que financian la investigación) perdieron gran parte del interés. Tras varios periodos de "veranos" e "inviernos", actualmente nos encontramos en un momento de vertiginoso crecimiento en la investigación en IA, apoyada por otra de las revoluciones en computación: el acceso a una enorme cantidad de datos, propiciada por las mejoras en las tecnologías de comunicación y de almacenamiento de datos. Claramente nos encontramos aún muy lejos de disponer de una IA general, y aún más de la revolucionaria "singularidad" (como se conoce al hipotético momento en el que una IA es capaz de mejorar de forma autónoma el diseño de IAs, lo que podría marcar el inicio de desarrollo exponencial de la inteligencia). Aún así, incluso al nivel actual de la IA, sus distintas aplicaciones a problemas parciales están permeando en la sociedad y en la toma de decisiones a velocidad vertiginosa, con profundas implicaciones éticas que están muy lejos de ser resueltas [2].

Una de las novedades de la última ola de desarrollo de la IA ha sido su penetración en prácticamente todas las áreas de la ciencia, como se refleja en la explosión en el número de artículos científicos publicados, no solo sobre desarrollo de IA sino también sobre la aplicación de técnicas de IA en nuevos ámbitos.

Hacer siquiera una revisión de estos trabajos es, incluso ya en estos momentos, una tarea imposible. Aquí se abordará la tarea mucho más modesta de *ilustrar* algunas de las técnicas que pueden ser utilizadas en un amplio espectro de problemas científicos. Las ideas expuestas no son originales, pero los ejemplos sí lo son.

2. Tipos de IA

En la introducción se ha omitido intencionadamente una definición precisa de qué se considera IA. De forma general se considera que IA es todo análisis de datos que parece sorprendente que haya sido hecho con algoritmos matemáticos[1]. Dejar la definición de IA tan difusa es una necesidad originada por nuestro desconocimiento de qué es realmente la inteligencia.

En estos momentos, y sin ser exhaustivos, podemos distinguir algunos tipos de IA [3]:

- Aprendizaje Supervisado. En este tipo de problemas se dispone de un conjunto de N datos x_i, donde $i = 1, \ldots, N$. Cada x_i es un vector con N_r dimensiones, cada una de ellas representando un *rasgo* de los datos (de manera que podemos representar x como una matriz $N \times N_r$. Además, para cada dato se dispone de una *etiqueta* y_i (que en general es un vector de N_e componentes). La idea es encontrar un procedimiento matemático que, tras un proceso de aprendizaje, sea capaz de predecir la etiqueta que cabe esperar para un dato nuevo.

  El aprendizaje supervisado se divide a su vez en dos grandes categorías, según las etiquetas puedan tomar valores dentro de un continuo (en cuyo caso hablamos de *regresión*, y el problema se convierte en ajustar una función de N_e componentes, cada una de ellas función de N_r argumentos) o un discreto (problemas de clasificación).

- Aprendizaje No-Supervisado. Como el anterior, pero sin disponer de las etiquetas. El problema en este caso es encontrar correlaciones entre los datos que permitan agruparlos de forma automática.

- Aprendizaje Semi-Supervisado. Es como el aprendizaje no-supervisado, pero donde tras una agrupación se incorporan validaciones externas de las asignaciones (como ocurre, por ejemplo, en la identificación de personas en fotos realizada hoy en día por los teléfonos móviles).

- Aprendizaje Reforzado. En este tipo de aprendizaje un programa genera posibilidades acorde a una reglas de juego y una estrategia, y "aprende" modificando la estrategia según las valoraciones que va obteniendo. Los ejemplos paradigmáticos

[1] He de confesar que cuando empecé a interesarme por estos temas me sorprendió que el ajuste por mínimos cuadrados, que me enseñaron allá por los años 80 en una clase de prácticas de laboratorio en el primer año de carrera, se considera en estos momentos como IA (posteriormente he apreciado la gran utilidad didáctica de esta sencilla técnica para ilustrar qué debe ser evitado en problemas de IA más complejos).

son máquinas que consiguen niveles de juego sobrehumanos en videojuegos, *go* o ajedrez [4], sin que se les haya programado ninguna estrategia, es decir, sin ninguna aportación humana más allá de programar las reglas del juego (y en el caso de los videojuegos, ni siquiera eso fue necesario).

En esta exposición consideraremos solo ejemplos de Regresión. El principal problema de este tipo de aprendizaje es la necesidad de disponer de las etiquetas, lo que muy a menudo requiere una costosa supervisión humana. Pero, en muchos casos, los datos se pueden generar de forma sintética con un ordenador, de manera que las etiquetas son conocidas. Esta será la estrategia que se seguirá en los ejemplos que se presentarán posteriormente. Además, la exposición se restringirá a técnicas basadas en redes neuronales, que se describen brevemente a continuación.

3. Redes Neuronales

Las neuronas biológicas trasmiten la señal eléctrica a partir de la integración de las señales que les llegan de otras neuronas a través de conexiones llamadas axones. Cuando la señal integrada que llega a una neurona sobrepasa un cierto umbral, esta neurona emite un pulso eléctrico hacia la siguiente.

Esta forma de transferencia de información motivó la siguiente representación matemática: en el contexto de IA, una "neurona" es una función, $y(\vec{x})$ que se construye mediante la aplicación consecutiva de (i) una transformación lineal del vector $\vec{x}$ de N_{input} dimensiones a un número real z y (ii) una función no-lineal $g(z)$, conocida como ***función de activación***. Esto es,

$$y(x_1, x_2, \cdots, x_{N_{input}}) = g(z) \tag{3.1}$$

donde $z = \sum_{i=1}^{N_{input}} w_i x_i + b$. Esquemáticamente, podemos representar estas operaciones matemáticas como se refleja en la Fig. 1.

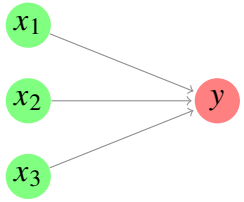

Figura 1: Representación esquemática de una neurona en IA. En este caso la figura representa una función $\mathbb{R}^3 \to \mathbb{R}$, que se construye como $y = g\left(\sum_{i=1}^{3} w_i x_i + b\right)$, donde $\{w_i, b\}$ son los parámetros que definen la transformación lineal y $g(x)$ es una función no-lineal.

Una Red Neuronal (RN) es una función que se construye recursivamente mediante la aplicación sucesiva de capas de operaciones, cada una de ellas compuesta de una transformación lineal seguida de una no-lineal. Dentro de esta construcción general existen aún muchos tipos de RN. En lo que sigue, se considerarán lo que se conoce como *Redes Neuronales Densas* (RNDs). La Fig. 2 muestra una representación esquemática de una RND.

Las RNDs constan de una capa de entrada (en este caso con 3 neuronas), una capa de salida (con dos neuronas en la figura) y un número arbitrario N_{HL} de capas ocultas, cada una de ellas con un número arbitrario de neuronas. Cuando la RND tiene más de una capa oculta hablamos de una RND *profunda*. Caracterizaremos las RND por el número de neuronas de las distintas capas, empezando por la de la capa de entrada y acabando con la de salida, y separándolas por un guion (como ejemplo, la RND representada en la Fig. 2 es del tipo 3-3-4-2).

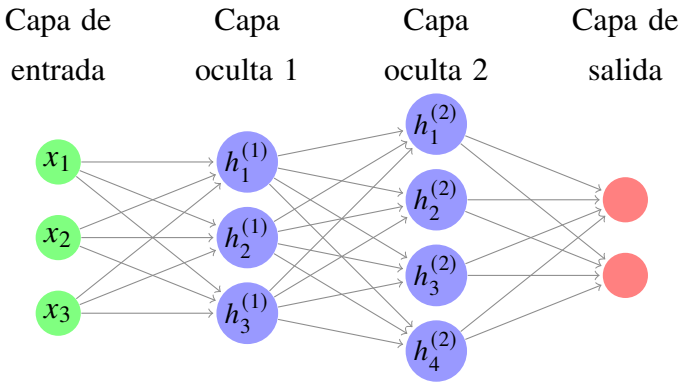

Figura 2: Representación esquemática de una Red Neuronal Densa. En este caso la figura representa una función $\mathbb{R}^3 \to \mathbb{R}^2$ con dos capas ocultas, la primera con 4 neuronas y la segunda con 3. Decimos en este caso que la red es del tipo 3-3-4-2. En esta arquitectura cada neurona está conectada con todas las de la capa anterior.

Utilizando la convención de suma sobre índices repetidos de Einstein, las reglas recur-

sivas de construcción de la función son:

$$z_i^{(1)} = W_{ij}^{1\leftarrow 0} x_j + b_i^{(1)} \tag{3.2}$$

$$h_i^{(1)} = g^{(1)}(z_i^{(1)}) \tag{3.3}$$

$$\dots$$

$$z_i^{(n)} = W_{ij}^{n\leftarrow n-1} h_j^{(n-1)} + b_i^{(n)} \tag{3.4}$$

$$h_i^{(n)} = g^{(n)}(z_i^{(n)}) \tag{3.5}$$

$$\dots$$

$$z_i^{N_h+1} = W_{ij}^{N_h+1\leftarrow N_h} h_j^{(N)} + b_i^{(N_h+1)} \tag{3.6}$$

$$y_i^{RN} = g^{N_h+1}(z_i^{N_h+1}) \tag{3.7}$$

La función definida por la RND depende, además de la arquitectura y de las de las funciones de activación elegidas ($\{g^{(n)}(z)\}$, una para cada capa), de unos parámetros numéricos: los "pesos" (el conjunto de matrices $\{W^{n\leftarrow n-1}\}$) y los "sesgos" (los vectores $\{b^{(n)}\}$, cada uno de ellos con una dimensión igual al número de neuronas en la n-ésima capa).

La aplicación de las RNDs en el aprendizaje supervisado se basa en presentar un conjunto de N datos de entrenamiento, con rasgos $\{\vec{x}_i\}_{i=1,\dots,N}$ y etiquetas $\{\vec{y}_i\}_{i=1,\dots,N}$, a una RN. Para un valor determinado de los parámetros, ésta produce un conjunto de valores de la función $\{\vec{y}_i^{RN}\}_{i=1,\dots,N}$. El proceso de aprendizaje consiste en (i) elegir una función "coste" que nos indique cuán lejos está la predicción proporcionada por la RN de las etiquetas proporcionadas y (ii) una minimización de esta función mediante la variación de los parámetros numéricos que definen la RN.

Las RNDs tienen una gran capacidad para ajustar datos de entrenamiento si se les proporciona un número de parámetros suficientemente grande (bien por tener un número elevado de capas ocultas, un número elevado de neuronas por cada capa oculta, o ambos). Es importante notar que el que la función coste sea pequeña, o incluso cero, *para todos los datos* no es necesariamente indicativo de un buen ajuste si los datos contienen ruido. En este caso, la RN puede estar aprendiendo a reproducir más el ruido específico de ese conjunto de datos que las correlaciones reales entre los mismos. Esto conllevaría a predicciones de la RND erróneas sobre datos del mismo sistema físico que no se usaron para entrenar la red. La manera de asegurarse de que este problema no ocurra es dividir los datos disponibles para entrenamiento en tres categorías: entrenamiento, validación y comprobación. Los datos de validación se utilizan para comprobar la bondad del aprendizaje *para una*

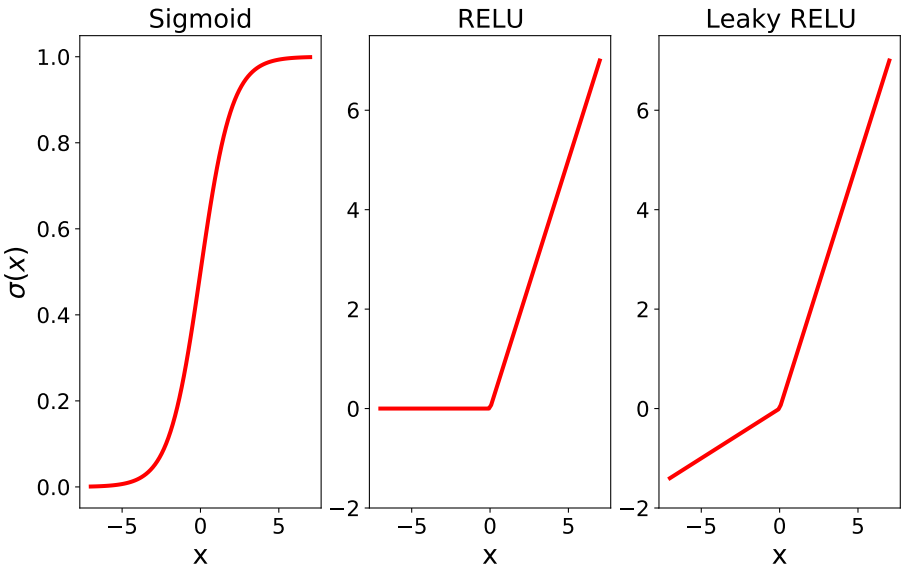

Figura 3: Funciones de activación ampliamente utilizadas en RNDs. La función "*Leaky* RELU"(x) se define como αx para $x < 0$, y x para $x \geq 0$. En la figura se ha tomado $\alpha = 0,2$.

realización de los hiperparámetros (que son todos aquellos que definen la arquitectura de la red, la función coste o el proceso de aprendizaje). La bondad del aprendizaje (estimado por el valor final de la función coste) con los datos de validación se utiliza como guía para elegir estos hiperparámetros. La bondad ***final*** del aprendizaje se evalúa utilizando los datos de comprobación, en una red con parámetros e hiperparámetros óptimos.

Adicionalmente, se pueden utilizar técnicas de ***regularización*** [5], que permiten eludir el sobre-entrenamiento incluso en RNDs con un gran número de parámetros, penalizando la utilización de todos ellos, o incluso eliminando un subconjunto de ellos aleatoriamente durante el proceso de optimización (lo que se conoce como técnica "*Dropout*").

A continuación se proporciona información adicional sobre las RN y el proceso de aprendizaje.

3.1. Funciones de activación usadas en RND.

Aunque virtualmente cualquier función no-lineal se puede usar como función de activación en una RND, hay unas cuantas que se utilizan con más frecuencia, y que se representan en la Fig. 3. La sigmoide $\sigma(x) = 1/(1 + \exp(-x))$ es ampliamente utilizada, sobre todo en la última capa y para problemas de clasificación, por sus asíntotas horizontales y por lo sencilla que es su derivada $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ (lo que facilita el cálculo

numérico y simplifica algún cálculo analítico). Sin embargo, para capas intermedias en RN *profundas* la función de activación de elección es la $\mathbf{RELU}(x) = \mathbf{máx}(0, x)$ (del inglés "*REctified Lineal Unit*"), o alguna función similar, como la "*Leaky* RELU". La razón es que el valor máximo de la derivada de la RELU es 1 mientras que el de la sigmoide es $1/4$ y que, como se verá más tarde, utilizar funciones de activación con derivada pequeña entorpece fuertemente el aprendizaje de RN profundas de muchas capas. De hecho, el sencillo paso de empezar a utilizar la función RELU permitió el uso de RN profundas, proporcionando un fuerte impulso a la IA.

3.2. Propiedades generales de la RND.

Las RND con funciones de activación continuas y derivables dan lugar a funciones continuas y derivables. A efectos prácticos, podemos incluir en esta categoría también a la función de activación RELU, que presenta una derivada discontinua en un solo punto.

Se puede demostrar un teorema de completitud para las RNDs: cualquier función continua $\mathbb{R}^n \to \mathbb{R}^m$ se puede representar fielmente con una RN, incluso de una sola capa oculta. El número de neuronas necesario depende, por supuesto, de la función a representar, de la función de activación utilizada, del nivel de fidelidad requerido, etc., y puede ser astronómicamente grande pero, aún así, este teorema demuestra la gran versatilidad de las RND. Nótese que en el caso en el que la función de activación de la última capa está acotada (como ocurre con la sigmoide), se entiende que la función a representar ha sido escalada de manera que su rango de variación está dentro del de la función de activación (lo que se puede considerar como una elección del sistema de unidades y del origen de ordenadas apropiados).

3.3. Función coste.

La elección de la función coste ideal para guiar el aprendizaje es un problema abierto. La regla heurística actual es la utilización del error cuadrático medio

$$C_{RMS} = \frac{1}{2N} \sum_{i=1}^{N} |y_i - y_i^{RN}|^2$$

para problemas de regresión y la función de "entropía cruzada"

$$C_{CE} = \frac{1}{2N} \sum_{i=1}^{N} \left[y_i \log(y_i^{RN}) + (1 - y_i) \log(1 - y_i^{RN}) \right]$$

para problemas de clasificación. Estas funciones coste tienen la ventaja de presentar un solo mínimo en ciertos problemas sencillos, pero en general pueden presentar una estructura compleja de máximos y mínimos locales en el espacio de parámetros que definen la RN, lo que complica enormemente el proceso de minimización.

3.4. Proceso de Minimización.

El proceso de minimización de la función coste más ampliamente utilizado es el llamado Descenso por Gradiente. Este es un proceso iterativo que, empezando por unos valores semilla de los parametros que definen la RND (los pesos y los sesgos), sigue los siguientes pasos: (i) se calcula el gradiente de la función coste con respecto a estos parámetros y (ii) hasta que la función coste alcanza un mínimo, los parámetros se actualizan ***restándoles*** el gradiente, multiplicado por un cierto número α, conocido como ***tasa de aprendizaje***. En general, el acercamiento al mínimo será muy lento para valores muy pequeños de α, mientras que valores demasiado altos pueden producir que el mínimo se sobrepase, dando lugar a lentos procesos de acercamiento oscilatorios, e incluso a procesos de alejamiento del mínimo. La tasa de aprendizaje es pues un hiperparámetro adicional de cálculo, susceptible de optimización.

La técnica de Descenso por Gradiente tiene muchas variantes, destinadas a remediar los problemas que ocurren cuando el gradiente se anula en algún paso del proceso, pero no por encontrar un mínimo sino un máximo o un punto de silla. Típicamente, estas variantes consideran no solo el último valor del gradiente, sino también valores obtenidos en iteraciones anteriores.

Nótese que, aunque sencillo conceptualmente, en general el Descenso por Gradiente es costosísimo computacionalmente. Una sola llamada a la RN implica la evaluación de la cadena de ecuaciones (3.2)-(3.7). El cómputo de la función coste implica tantas llamadas a la RN como datos se disponga (lo que típicamente está en el rango $10^3 - 10^6$), y cada cálculo del gradiente implica la derivada con respecto a todos los pesos y rasgos (típicamente $10^2 - 10^8$). Afortunadamente, se han desarrollado dos técnicas que reducen fuertemente el coste computacional de la función coste y del cálculo del gradiente. Estas son:

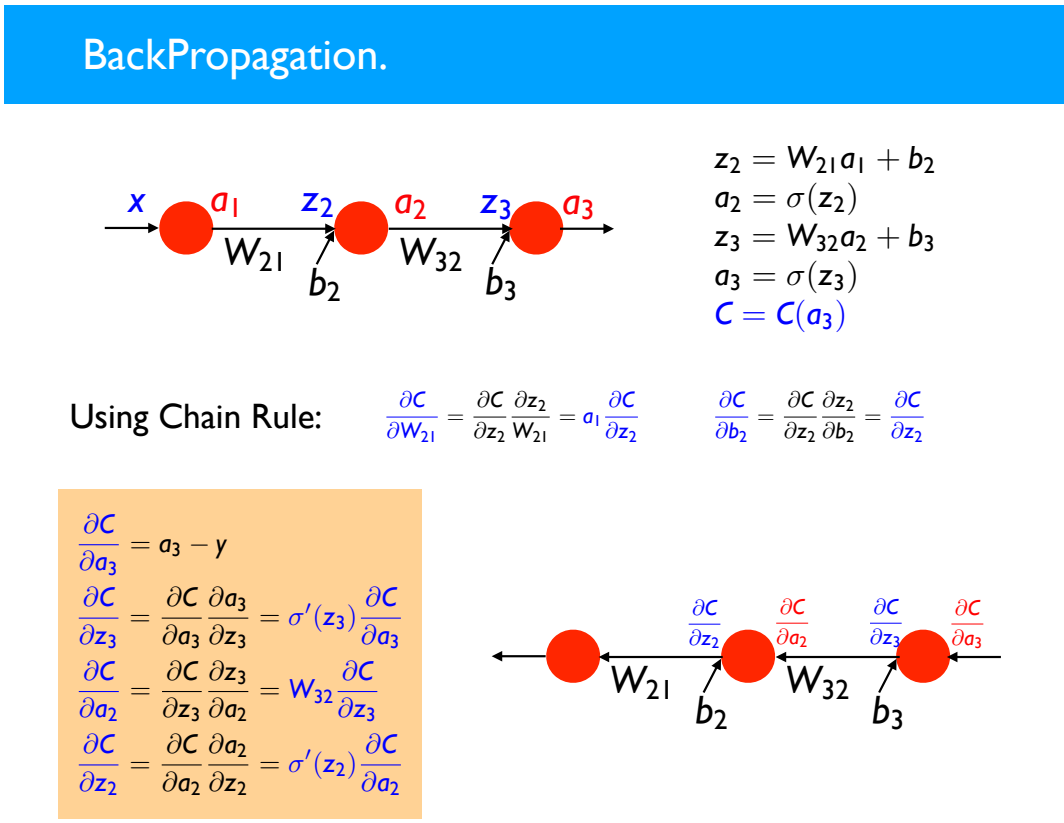

Figura 4: Derivación de las ecuaciones que definen técnica de "*Back Propagation*" para el cálculo del gradiente de la función coste en RNDs, para el caso de 1 neurona por capa, y considerando la contribución a la función coste de un solo dato. La notación de las variables se ha modificado con respecto a las de las ecuaciones (3.2)-(3.7) para destacar la forma recurrente de las ecuaciones. La función coste elegida es $C = (a_3 - y)^2/2$.

- "*Back Propagation*". Esta técnica utiliza la estructura recurrente de las RND, donde la "información" fluye unidireccionalmente desde la capa de entrada a la de salida (de izquierda a derecha en la representación utilizada en la Fig. 2). Usando la regla de la cadena, la derivada con respecto a un parámetro siempre se puede expresar en función de la derivada para parámetros que se encuentran más a la "derecha". Esto permite empezar calculando la derivada de la función coste con respecto a los parámetros más a la derecha y desde ahí calcular recursivamente las derivadas con respecto a todos los parámetros, yendo progresivamente hacia la izquierda. Este proceso se ilustra en la Fig. 4 en el caso simplificado de una sola neurona por capa. La extensión al caso general es sencilla pero algo más farragosa, al involucrar expresiones matriciales [3, 5]. Esta técnica permite la evaluación del gradiente mediante *una sola* evaluación recurrente de la RN (pero en sentido contrario al de la evaluación de la función, de ahí su nombre), en lugar de un número de llamadas del orden del

número de parámetros, como sería necesario si el gradiente se calculara mediante una discretización de la derivada. Aún más, mientras que la discretización de la derivada produciría una *aproximación* al valor de ésta, el cálculo del gradiente mediante la técnica de "*Back Propagation*" es exacta.

- Descenso por Gradiente Estocástico.

  Este método aproxima el gradiente exacto de la función coste. Para ello, los datos de entrenamiento se dividen en N/N_p paquetes ("mini-batches"), siendo un paquete un conjunto de N_p datos (de manera que N_p se convierte en un hiperparámetro, que típicamente se escoge en el rango 1–100). El gradiente total se aproxima secuencialmente por el obtenido para cada paquete. Cuando se han considerado todos los paquetes, cada uno proporcionando un paso de Descenso de Gradiente, se dice que se ha completado una "época". Para no favorecer a unos datos frente a otros el proceso de entrenamiento conlleva ejecutar un cierto número de épocas completas.

  Esta aproximación tiene la ventaja evidente de reducir el tiempo de computación del gradiente. Pero además, el hecho de que el gradiente sea aproximado (pero no sesgado, ya que el promedio del gradiente sobre paquetes es igual al gradiente exacto) es útil también para salir de mínimos locales, máximos o puntos de silla.

3.5. Otras arquitecturas de RN.

Las Redes Neuronales Densas son sólo uno de los muchos tipos de RN. Citaremos aquí tan solo dos muy ampliamente utilizadas en estos momentos:

- Redes Neuronales de Convolución. En una RND, los datos se introducen como un vector unidimensional en la capa de entrada. Si los rasgos iniciales "residen" en un espacio multidimensional se deben ordenar de manera que se colocan en un vector uni-dimensional. Además, al estar conectadas todas las neuronas de una capa con todas las de la siguiente, se pierde la estructura espacial que pueden tener los datos (como ilustración, la arquitectura de la red no tiene información de que algunos puntos pueden estar más próximos espacialmente entre sí que otros). Por contra, las RN convolucionales mantienen la estructura espacial original (de manera que, por ejemplo, una fotografía vendría representada por una red bidimensional de intensidades en los píxeles) y, en esencia, los pesos son filtros que se aplican a toda la red mediante convoluciones con los datos en cada capa. Igual que las fotografías pueden tener varios canales de datos de entrada (que para cada pixel pueden ser

las intensidades en distintos colores), cada capa puede tener varios canales (los distintos filtros). Este tipo de redes es también muy ventajosa cuando los datos tienen simetría translacional (como en las fotos, donde los objetos se reconocen aunque no estén centrados), lo que está automáticamente tenido en cuenta por el proceso de convolución.

- Redes Neuronales Recurrentes. Diferentes variantes de RN van más allá del flujo de información lineal (de entrada a salida) de la RND y tienen en cuenta procesos de realimentación. Estas redes son relevantes en sistemas donde la historia de la información es importante. Un ejemplo paradigmático es el reconocimiento de lenguaje, donde el sentido de una frase o de un párrafo puede estar distribuido en varios puntos.

4. Ejemplos de aplicaciones de Redes Neuronales Densas

Pasamos ahora a analizar ejemplos sencillos que muestran diferentes usos de las RNDs en ciencia.

4.1. Capacidad de representación de una Red Neuronal Densa.

El primer ejemplo está elegido para ilustrar la complejidad que es posible conseguir incluso con una elección sencilla de RND.

Para ello representamos una función de dos variables $f_{RN}(x, y)$ como una RND con 2 neuronas de entrada, 1 de salida y N_{HL} capas ocultas. Para ver el efecto de la arquitectura en la complejidad de la función, tomamos la misma función de activación en todas las capas y fijamos el número total de neuronas en las capas ocultas a 2000, de manera que el número de neuronas por capa es $N_n = 2000/N_{NL}$. Los pesos y sesgos de la RND se extraen aleatoriamente conforme a una distribución normal centrada en 0, con una desviación típica de 3 para los pesos y 1 para los sesgos (valores que han sido elegidos de forma arbitraria).

En la Fig. 5 se representan algunos ejemplos de funciones definidas de esta manera, utilizando la sigmoide como función de activación, para realizaciones elegidas al azar de pesos y sesgos. La figura muestra una progresión desde una RND con una sola capa oculta y $N_n = 2000$ neuronas hasta una RND con $N_{HL} = 250$ y 8 neuronas en cada capa (redes aún más "profundas" prácticamente dan como resultado 0, y no se representan en la figura).

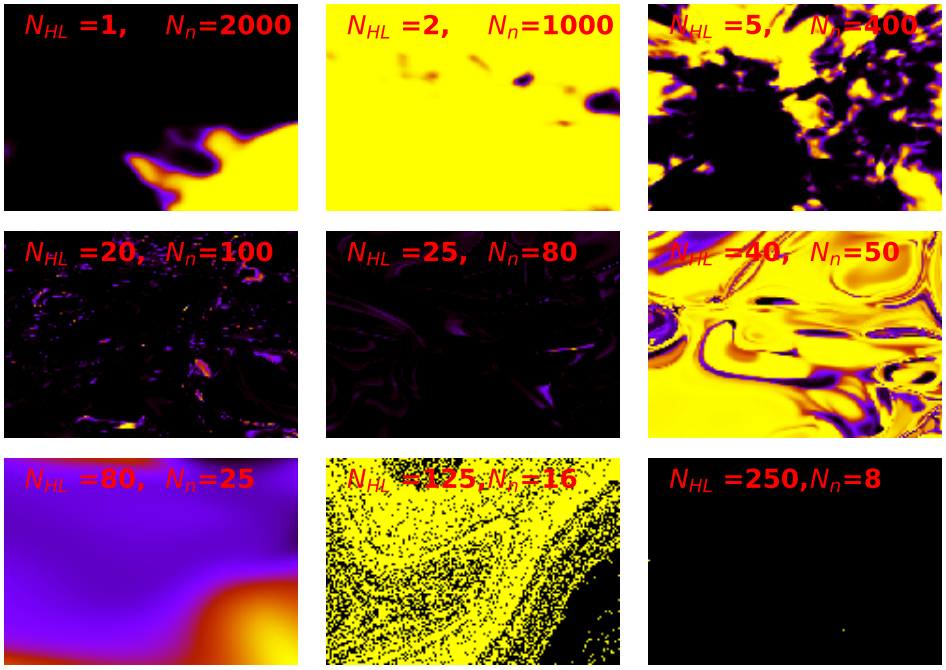

Figura 5: Ejemplos de funciones $f_{RN}(x, y)$ creadas a partir de una RND del tipo 2-X-1, donde X representa un conjunto de N_{HL} capas ocultas, cada una de ellas con N_n neuronas. El número total de neuronas en las capas ocultas es $N_{HL} \times N_n = 2000$. Tanto los valores de x como los de y varían entre 0 y 1, y se ha discretizado la función con 250 puntos en cada dirección. La escala de colores de $f_{RN}(x, y)$ es distinta para cada caso, yendo del negro para el mínimo de la función al amarillo para el máximo.

Fijándonos en la figura, y estimando visualmente la complejidad de las funciones, vemos que la complejidad aumenta con el número de capas ocultas hasta un máximo, que ocurre cuando $N_{NL}/N_n \approx 0.2 - 1$. Para valores mayores de N_{NL}/N_n la complejidad disminuye (véase por el caso $N_{NL}/N_n = 89/25$) pero, sorprendentemente, a veces aparecen patrones extremadamente complejos que recuerdan estructuras fractales (como para $N_{HL} = 125$ y $N_n = 16$).

Estos son tan solo ejemplos, pero demuestran que las RND son capaces de generar funciones de tipo fractal o que recuerdan a turbulencias por lo que, dándole la vuelta al argumento, pueden proporcionar una buena parametrizan de funciones de este tipo, que son notoriamente difíciles de representar por otros medios.

La discusión anterior sobre la complejidad está basada en criterios visuales subjetivos,

pero se puede justificar con criterios matemáticos. Para ello, tomamos los valores de la función $f_{RN}(x, y)$ en un retículo de $N_p \times N_p$ en 2 dimensiones. Esto define una matriz M de "pixeles", sobre la que podemos hacer una Descomposición de Valores Singulares (SVD, de sus siglas en inglés), y expresarla como $M = U\Sigma V^T$, donde Σ es una matriz diagonal, que se puede elegir de manera que sus valores sean reales positivos (que es práctica usual ordenar de mayor a menor).

Esto es, en general para una matriz $n \times m$ tenemos:

$$M = [\vec{u}_1\, \vec{u}_2 \ldots \vec{u}_n] \begin{bmatrix} \sigma_1 & & & 0 \\ & \sigma_2 & & \\ & & \sigma_r & \\ & & & \ddots \\ 0 & & & 0 \end{bmatrix} [\vec{v}_1\, \vec{v}_2 \ldots \vec{v}_m]^T \tag{4.1}$$

donde $\{\vec{u}\}$ y $\{\vec{v}\}$ forman una base ortonormal de $\mathbb{R}^n$ y $\mathbb{R}^m$, respectivamente.

La SVD puede utilizarse para comprimir la información contenida en una matriz, reemplazando la matriz Σ por una aproximada donde se se han anulado los elementos más pequeños. Para ello, se define la varianza como $Var = 1/(N-1)\sum_i \sigma_{ii}^2$, donde la suma es a *todos* los elementos de Σ. La Varianza Explicada se define como la varianza, pero truncando la suma eligiendo un número dado de los valores más altos (por tanto la Varianza Explicada depende de este número). En lo que sigue, definimos n como el número autovalores que hay que incluir para que la Varianza Explicada sea una fracción determinada de la Varianza, fracción que vamos a tomar como 0.99. En otras palabras, aunque cada "imagen" discretizada de la función requiere N_p autovalores y autovectores, una representación muy fidedigna se puede conseguir con tan solo n de ellos.

A cada realización de los pesos y rasgos de una RND le corresponde un valor de n. Representamos por n_{av} y σ el valor medio y la desviación estándar de la distribución de n, a arquitectura de red dada. Otra medida relevante se obtiene a partir de cada realización de $f_{RN}(x, y)$ restándole su valor medio (lo que equivale a desplazar el origen de la función). De esta manera podemos caracterizar funciones que presentan variaciones complicadas pero de pequeña amplitud con respecto a un valor medio más alto. Tras aplicar la SVD a la función transformada, obtenemos el n mínimo para que la Varianza Explicada sea 0.99 veces la varianza, y calculamos la media y la desviación estándar de su distribución (considerando diversas realizaciones de pesos y rasgos), que denotamos por $n_{av}^{f_{av}=0}$ y $\sigma^{f_{av}=0}$. Estas magnitudes se representan en la Fig. 6, en función de N_{HL} (con

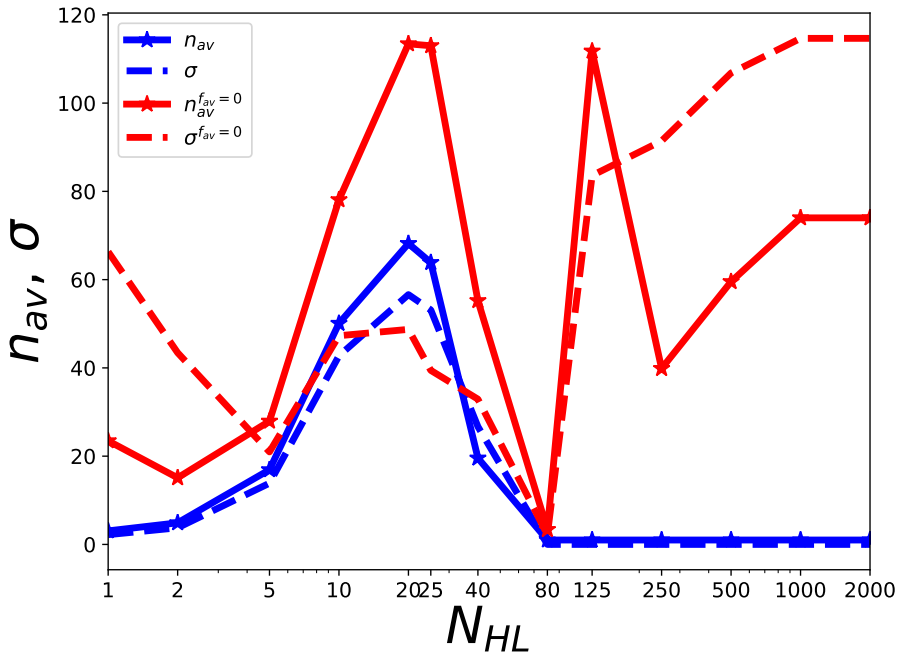

Figura 6: Dependencia con el número de capas ocultas del valor medio n_{av} y de la desviación estándar σ de la distribución del número de autovectores de una SVD necesarios para reproducir el 99 % de la varianza de imágenes de funciones $f_{RN}(x, y)$ (definidas con pesos y sesgos aleatorios, como se describe en el texto). El número de neuronas por capa es $2000/N_{HL}$. Las imágenes corresponden a una discretización de $f_{RN}(x, y)$ con 250×250 valores de (x, y) en el intervalo $[0, 1] \times [0, 1]$. Los valores con superíndice $f_{av} = 0$ se han calculado restando a cada función f_{RN} su valor medio. Para cada valor de N_{HL} los promedios han sido realizados sobre 1000 realizaciones aleatorias de las funciones $f_{RN}(x, y)$.

$N_n = 2000/N_{HL}$). La evolución de n_{av} con N_{HL} presenta un máximo alrededor de 25 capas, de acuerdo con la estimación visual de la complejidad de las $f_{RN}(x, y)$ presentadas en la Fig. 5. Su comparación con la dependencia de $n_{av}^{f_{av}=0}$ muestra como para $N_{HL} \geq 80$ la función $f_{RN}(x, y)$ es prácticamente constante, pero con complejas variaciones espaciales (que requieren de muchos autovectores de la SVD para representarlas). Nótese que para $N_{HL} \geq 80$ la desviación estándar $\sigma^{f_{av}=0}$ es muy alta. Un análisis de los datos muestra que para $N_{HL} \geq 80$ la distribución de valores de n es prácticamente bi-modal, siendo n en con alta probabilidad bien 0 o el máximo posible (250 en el caso considerado en la figura). Claramente, este punto, así como la sorprendente disminución del número de vectores SVD que se necesitan para expresar las funciones $f_{RN}(x, y)$ para $N_{HL} = 80$, requieren un estudio más exhaustivo que el aquí presentado.

4.2. Ajustes de funciones y diseño inverso.

En lo que sigue se van a ilustrar algunas de las posibles aplicaciones de las RND en ciencia.

4.2.1. Datos utilizados y parámetros elegidos.

Los siguientes ejemplos van utilizar datos sintéticos (creados a partir de un modelo matemático). En concreto se va a considerar la transmisión cuántica de un electrón moviéndose en una dimensión, en presencia de un potencial compuesto por dos "deltas de Dirac", situadas en las posiciones $x = \pm 1$ [2].

La función de onda del electrón, $\Psi(x)$, cumple la Ecuación de Schrodinger:

$$-\frac{\hbar^2}{2m}\frac{\partial^2\Psi(x)}{\partial x^2} + V(x)\Psi(x) = E\Psi(x), \tag{4.2}$$

con $V(x) = \delta_1\,\delta(x+1) + \delta_2\,\delta(x-1)$, donde δ_1 y δ_2 son dos parámetros a elegir, y donde $\hbar$ y m son la constante de Planck y la masa de electrón, respectivamente (en el cálculo elegiremos un sistema de unidades donde $\hbar = m = 1$).

Suponemos que un electrón se propaga desde $-\infty$ hacia la derecha con energía E (esto es, con número de ondas k tal que $E = \hbar^2k^2/2m$), y es parcialmente reflejado y transmitido por el potencial. En este caso, las condiciones de contorno son: $\Psi(x) = e^{\imath kx} + r(k)\,e^{-\imath kx}$ para $x < -1$, y $\Psi(x) = t(k)\,e^{\imath kx}$ para $x > 1$. Los coeficientes de reflexión y transmisión ($r(k)$ y $t(k)$, respectivamente) se pueden calcular aplicando métodos estándar en mecánica cuántica [6]. A partir de estos coeficientes se puede extraer la reflectancia $R(k) = |r(k)|^2$ y la transmitancia $T(k) = |t(k)|^2$

Los datos sintéticos se han preparado eligiendo 10000 valores aleatorios para el par (δ_1, δ_2), donde cada valor de $\delta_{1,2}$ puede variar entre 0 y 5, y calculando para cada par $T(\delta_1, \delta_2, k)$ para 300 valores de k equiespaciados entre 0 y 10.

A continuación se exponen algunos detalles técnicos de los cálculos desarrollados. Como función coste se ha elegido el error cuadrático medio (MSE, por sus siglas en inglés). El método de optimización utilizado es el denominado "Adam" [7], que es una variante del descenso estocástico por gradiente que varía la tasa de aprendizaje de forma adaptativa utilizando estimaciones extraídas del primer y segundo momento de los gradientes. En

[2] Estos datos han sido utilizados en el Trabajo Fin de Grado de Pablo A. Calvo Barlés "Inteligencia Artificial aplicada al problema de scattering en una dimensión", que he dirigido en el curso 2020-21.

todas las redes se ha elegido como función de activación la RELU, excepto para las neuronas de la última capa, para las que se ha elegido la sigmoide. El tamaño de los mini-batches se ha tomado en todos los casos igual a 10. Para los cálculos se ha utilizado la implementación "`TensorFlow 2.0`", que es una plataforma de código en abierto para IA con redes neuronales, basada en el código `Keras` [8]. Todos los cálculos que se presentan a continuación se han realizado en un ordenador de sobremesa, con tiempos de ejecución del orden de minutos.

Es importante destacar que en los ejemplos que siguen no se ha realizado una búsqueda sistemática en el espacio de hiperparámetros. Por consiguiente, solo se han usado conjuntos de entrenamiento y validación, y no de comprobación. Tampoco se ha utilizado ninguna técnica de regularización. En el mejor de los casos, se han probado 3–4 arquitecturas y/o inicializaciones de los parámetros, y se ha detenido la búsqueda cuando se ha estimado que el resultado era suficientemente bueno como para ilustrar el potencial de la técnica.

Por tanto, en cada caso se entrena la red con un número de datos N_{train}, con los que se encuentran los pesos y sesgos que minimizan la función coste. Previamente se ha reservado otro conjunto con $N_{validation}$ datos, que nunca se utilizan para entrenar la red. Los valores al final del entrenamiento de la función coste para los datos de entrenamiento y validación proporcionan información muy valiosa sobre la bondad de la aproximación. La RND ha capturado la ley matemática que rige la función, y por tanto tiene valor predictivo, si el valor de la función coste es pequeña para ambos conjuntos de datos. Si, por el contrario, la función coste es mucho menor para los datos de entrenamiento que para los de validación, la red ha aprendido a reproducir fidedignamente solo los valores que se usaron para entrenarla, actuando más como una memoria de la colección de datos de entrenamiento que como una representación de la función que los generó. Finalmente, si la función coste es grande, tanto para los datos de entrenamiento como de validación, la RND considerada no tiene una capacidad variacional de representación suficiente para el problema que se está tratando, y es conveniente considerar redes con más neuronas y/o más capas.

4.2.2. Creación de una RND capaz de predecir $T(\delta_1, \delta_2, k)$

La transmitancia así representada es una función real de tres variables (es decir, es una función de $\mathbb{R}^3 \to \mathbb{R}$), y por tanto es susceptible de ser representada por una RND con 3 neuronas de entrada (donde se introducen los valores de las variables) y una de salida (que proporciona el valor de la función).

Se han elegido dos arquitecturas de RND, ambas con dos capas ocultas. La RND más pequeña considerada tiene 20 neuronas en la primera capa oculta y 10 en la segunda (por tanto esta RND es del tipo 3-20-10-1). La RND más grande tiene 10 veces más neuronas en cada capa oculta, siendo por tanto del tipo 3-200-100-1. La red pequeña tiene 301 parámetros a entrenar (pesos y sesgos), mientras que la mayor tiene 21001.

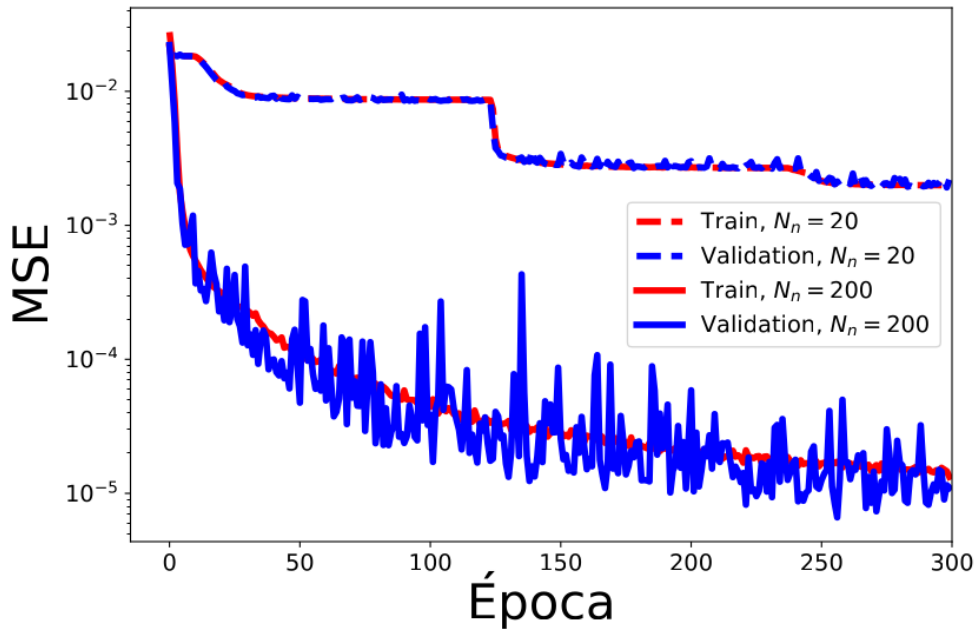

Figura 7: Curvas de entrenamiento RNDs del tipo 3-X-X/2-1 para ajustar la función $T(\delta_1, \delta_2, k)$, para $X = 20, 200$. La función de coste utilizada es el Error Cuadrático Medio (MSE).

Para que el cálculo fuera más rápido, de los 10000×300 datos generados (10000 δs y 300 ks), se eligió un subconjunto más pequeño de 8000×30 datos para entrenamiento y 2000×30 para validación. Es importante destacar que, antes de crear los subconjuntos de datos, todos los datos han sido desordenados aleatoriamente. Con esto se evita el error que sería, por ejemplo, elegir los 30 primeros valores de k de cada par (δ_1, δ_2), lo que entrenaría la red solo para valores de la función con $k \in (0, 1)$, y no en el rango $k \in (0, 10)$, como se pretende.

La curva de entrenamiento (evolución con la época de entrenamiento de las funciones coste para datos de entrenamiento y validación) se muestra, para ambas redes, en la Fig. 7. Para cada red, los MSE de entrenamiento y validación son muy similares, lo que demuestra que incluso la red más grande no sufre de sobre-entrenamiento. Tomando el valor de la MSE de validación en la Época=300 como ya convergido, podemos estimar

que el error esperado al evaluar la transmitancia con la RND es $\epsilon = \sqrt{\text{MSE}}$. Para la red más pequeña $\epsilon \approx 0.05$, mientras que para la mayor es $\epsilon \approx 0.007$.

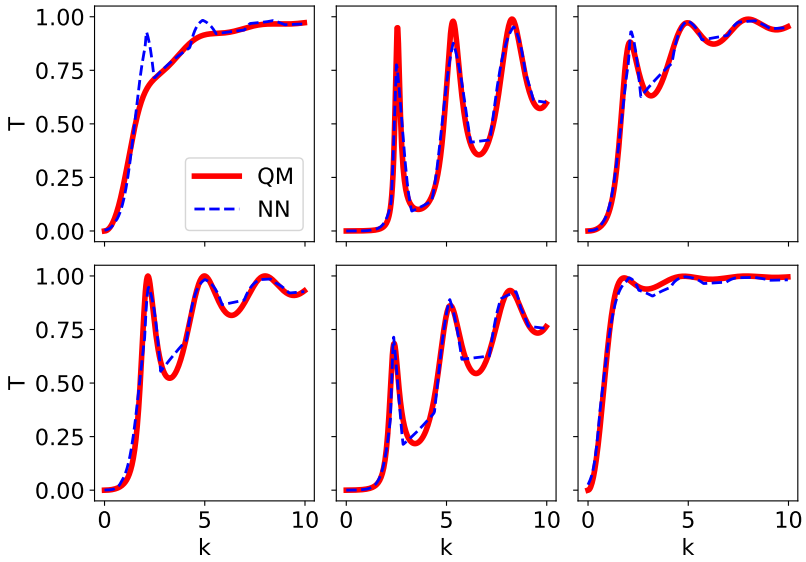

Figura 8: Comparación entre espectros de transmitancia calculados con la Ecuación de Schrödinger (QM) y los predichos por RNDs del tipo 3-20-10-1 (NN). La transmitancia es para un electrón moviéndose en una dimensión en presencia de un potencial $V(x) = \delta_1\delta(x+1) + \delta_2\delta(x-1)$. Distintos paneles son para distintos valores de $\delta_{1,2}$.

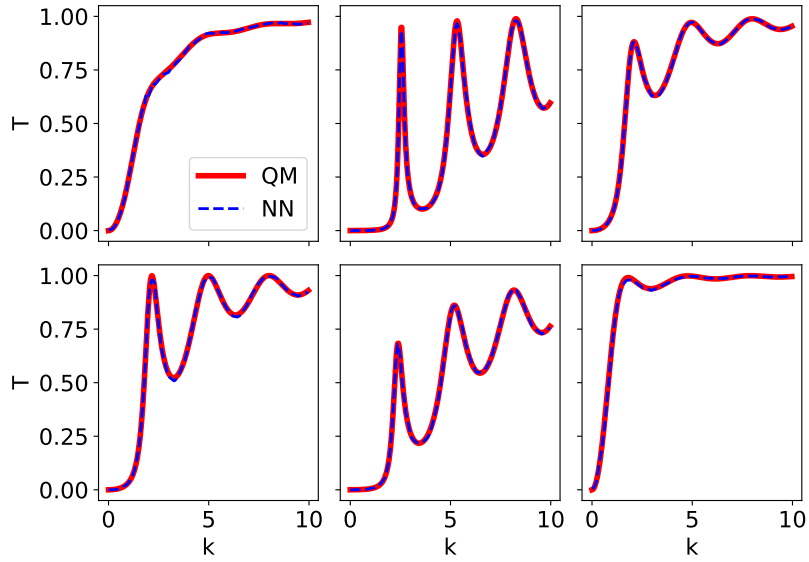

Figura 9: Lo mismo que en la Fig. 8, pero para una RND del tipo 3-200-100-1

La comparación entre espectros de transmitancia obtenidos al resolver el problema mecanocuántico con los que proporciona la RND entrenada se encuentran en la Fig. 8

para la red 3-20-10-1, y Fig. 9 para la red 3-200-100-1, en ambos casos para valores de (δ_1, δ_2) elegidos al azar.

Como demuestran las figuras, la RND más pequeña ya es capaz de representar las principales tendencias cualitativas de la transmitancia. La RND proporciona incluso predicciones cuantitativamente muy buenas en varias regiones del espectro pero, al hacer estimaciones incorrectas en otras regiones, es poco fiable en este aspecto. Sin embargo, la red 3-200-100-1 proporciona una representación muy fidedigna de $T(\delta_1, \delta_2, k)$.

Otra forma interesante de entrenar una RND para calcular el espectro de transmitancia consiste en, para cada par (δ_1, δ_2), agrupar los valores de la transmitancia para los distintos k en un vector, en este caso de dimensión 300: $(T(k_0), T(k_1), \cdots, T(k_{300}))$. Es decir, la RND ha de ajustar una función $\mathbb{R}^2 \rightarrow \mathbb{R}^{300}$. Obsérvese que ahora no es necesario proporcionar los valores de k, porque la transmitancia está evaluada siempre para los mismos k. Dentro de esta forma de representar el espectro de transmitancia, se dispone de

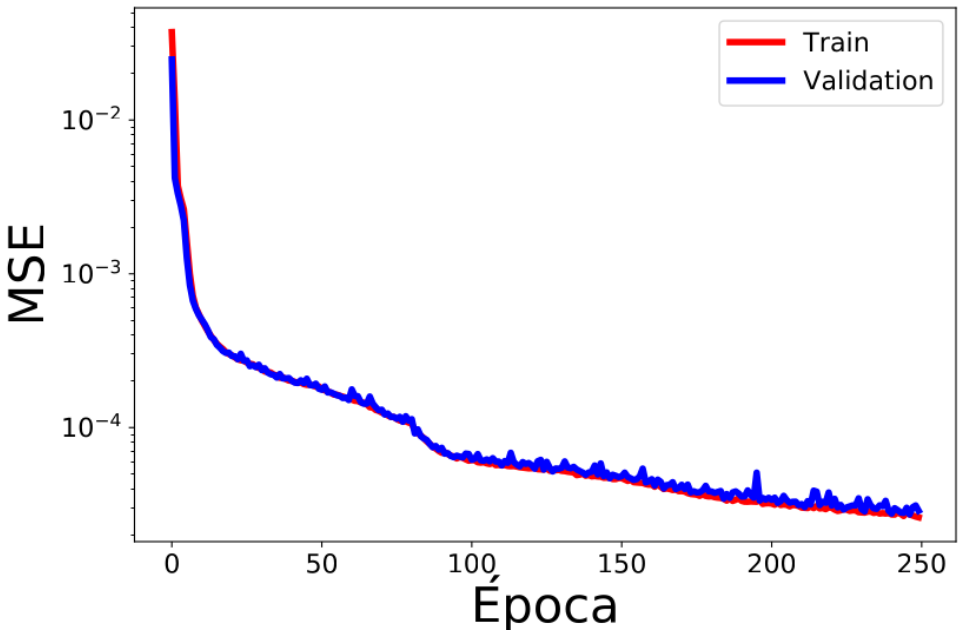

Figura 10: Curva de entrenamiento de una RND del tipo 2-100-50-300 para ajustar $(T(k_0), T(k_1), \cdots, T(k_{300}))$ en función de (δ_1, δ_2). La función de coste utilizada es el Error Cuadrático Medio (MSE).

los 10000 datos previamente generados, que se han separado en 8000 para entrenamiento y 2000 para validación. Para representar la transmitancia se ha elegido una RND del tipo 2-100-50-300. En la Fig. 10 se representa la MSE a lo largo del aprendizaje, mostrando que (i) la red no está sobre-entrenada y (ii) que con este método se pueden obtener repre-

sentaciones del espectro tan fidedignas como con el método anterior. La Fig. 11 presenta la comparativa entre los espectros de transmitancia y los obtenidos con la RND, para valores de (δ_1, δ_2) elegidos al azar de entre los datos validación, demostrando que la RND proporciona una representación excelente.

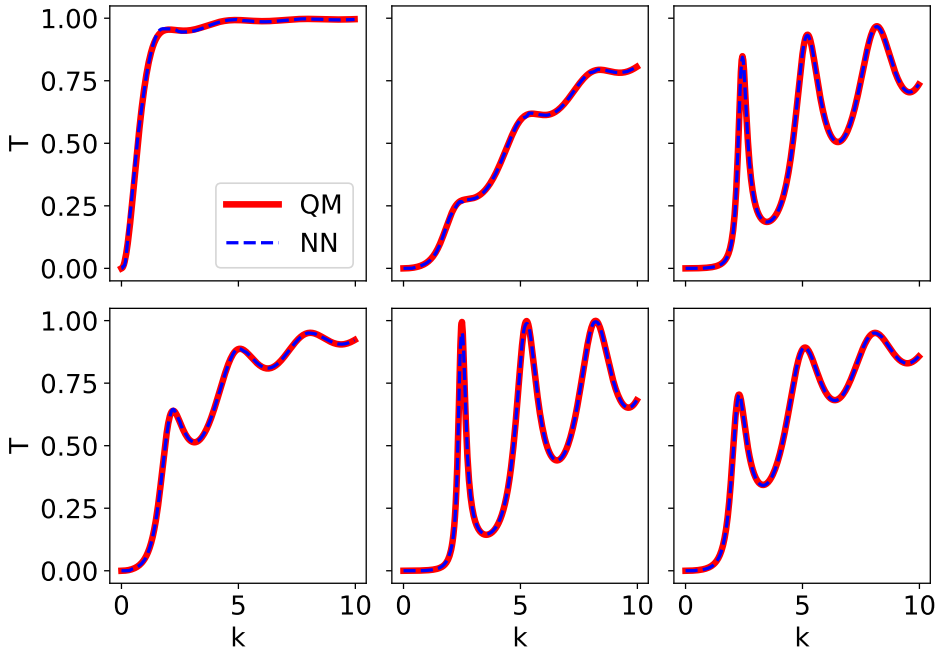

Figura 11: Comparación entre espectros de transmitancia calculados con la Ecuación de Schrödinger (QM) y los predichos por RNDs del tipo 2-100-50-300 (NN). Distintos paneles son para distintos valores de $\delta_{1,2}$ de entre los usados en los datos de validación.

4.2.3. Diseño inverso.

Aunque los datos de transmitancia se generaron a partir de los valores de $\delta_{1,2}$, no estamos obligados a presentárselos así a la RND. Las redes buscan dependencias entre datos de entrada y de salida, sin tener información de cómo fueron creados. Podemos, por tanto y con igual razón, probar si una RND es capar de encontrar los $\delta_{1,2}$ que dieron lugar al espectro $(T(k_0), T(k_1), \cdots, T(k_{300}))$. Para ello solo tenemos que considerar una función en dos dimensiones (δ_1, δ_2) que depende de 300 parámetros. Es decir, tenemos una función $\mathbb{R}^{300} \rightarrow \mathbb{R}^2$, donde ahora, para entrenar la RND, la "x" son los 300 valores de cada espectro de transmitancia y la "y" son las dos amplitudes $\delta_{1,2}$. Para ilustrar esta transformación inversa, se ha elegido una RND 300-100-50-2, y los mismos conjuntos

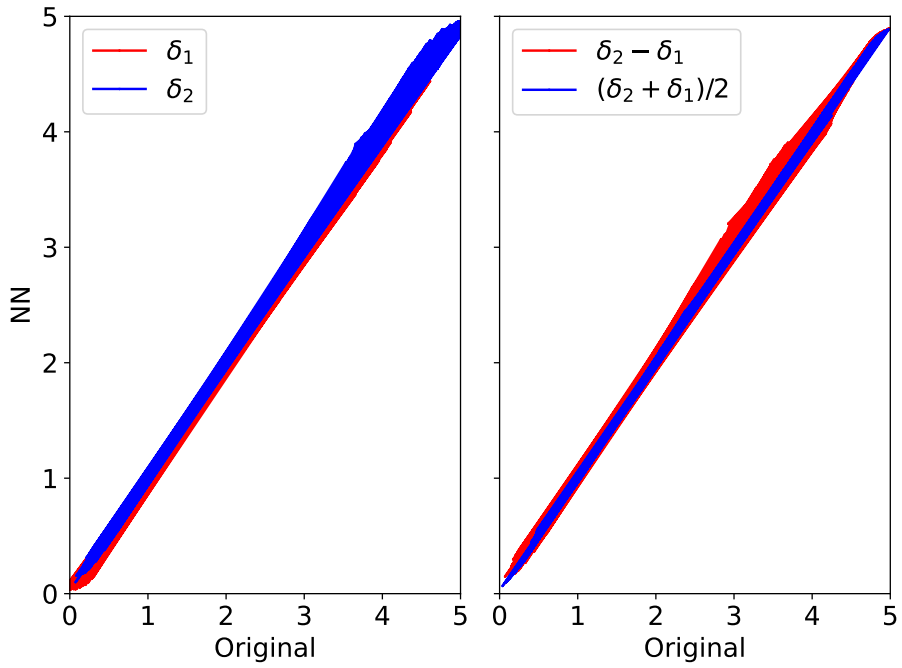

Figura 12: Comparación entre la predicción de una RND y los valores (δ_1, δ_2) utilizados para generar los espectros $(T(k_0), T(k_1), \cdots, T(k_{300}))$. El panel de la izquierda muestra la comparación para δ_1 y δ_2, mientras que el panel de la derecha lo hace para $\delta_1 - \delta_2$ y $(\delta_1 + \delta_2)/2$. La RND utilizada es del tipo 300-100-50-2.

de datos de entrenamiento y validación del caso directo anterior[3]. Al final del proceso de entrenamiento la RMS obtenida fue de 8.0×10^{-5} para datos de entrenamiento y 7.5×10^{-5} para los de validación (de manera que el error esperado en la estimación de cada valor de δ a partir de los espectros es $\epsilon = 8.6 \times 10^{-3}$).

La Fig. 12 representa los valores predichos por la RND para $\delta_{1,2}$ en función de los valores reales. Es de destacar que en esta figura sólo se han representado datos de validación, es decir que no han sido utilizados en el proceso de aprendizaje. La figura demuestra que la RND es capaz de predecir con bastante precisión el potencial que generó cada espectro. El panel de la derecha muestra que esto es más cierto para el promedio del potencial que para la diferencia entre las amplitudes de las dos deltas.

Este tipo de transformaciones inversas son de gran utilidad potencial en problemas de diseño inverso, donde tras entrenar una red podemos obtener los parámetros que pro-

[3]Como detalle técnico, para entrenar la RN las amplitudes de las deltas se han de re-escalar (dividiéndolas por 5) al hacer el cálculo de manera que varíen entre 0 y 1, ya que la función de activación de la última capa es una sigmoide. Una vez entrenada la red, los valores proporcionados por la RND para $\delta_{1,2}$ se expresan en las unidades originales multiplicándolos por 5.

porcionan una respuesta deseada (en el caso considerado, qué potencial definido por dos deltas proporciona un determinado espectro).

4.3. Autoencoders *aplicados a la reducción de dimensionalidad*

Una RND muy interesante se obtiene cuando los datos de salida coinciden con los de entrada. En este caso, la RND es una representación de la identidad, pero no una representación trivial sino una que se consigue cuando alguna capa oculta de la RND tiene un número de neuronas $N_n^{\text{mín}}$ menor que los de entrada y salida (es decir la red tiene una capa que forma una constricción). Como la información en un RND fluye secuencialmente de entrada a salida, toda la información sobre la función se encuentra en la constricción, y por tanto se puede expresar con un número reducido de dimensiones.

Como en otros casos de RND, la función coste se define de manera que sea mínima cuando la $\vec{y}^{RN}$ coincide con la $\vec{y}$ de los datos, que en el caso del *Autoencoder* es $\vec{y} = \vec{x}$. Nótese que las etiquetas originales no se utilizan en el entrenamiento del *Autoencoder*.

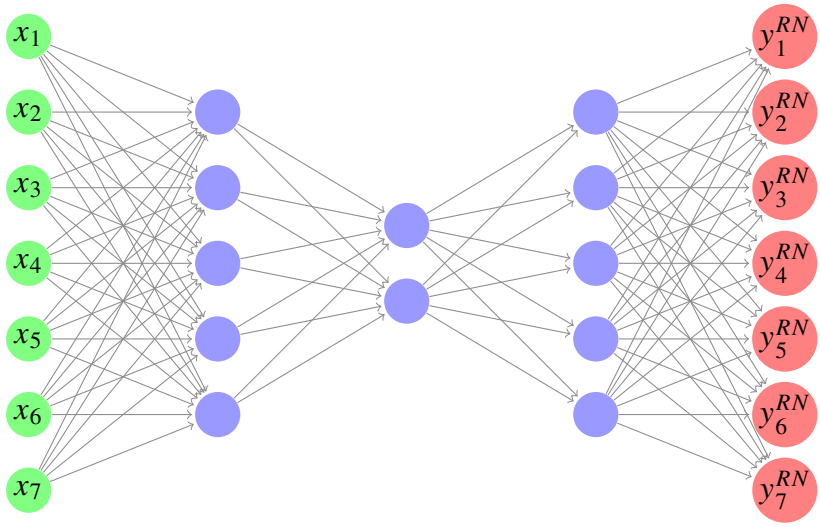

Figura 13: Representación esquemática de un *Autoencoder*, en este caso una RND del tipo 7-5-2-5-7. El número de neuronas en la constricción es $N_n^{\text{mín}} = 2$. Este *Autoencoder* es simétrico con respecto a la capa oculta central pero esta característica, aunque común, no es imprescindible. En el *Autoencoder* la función coste se define de manera que sea mínima cuando $\vec{y}^{RN} = \vec{x}$

Un ejemplo de *Autoencoder* se muestra en la Fig. 13. En el caso representado, la capa de entrada recibe datos de una función en 7 dimensiones. Si, tras el proceso de entrenamiento, el *Autoencoder* fuera capaz de reproducir los datos de forma fidedigna esto indicaría que la función se puede representar como una función en dos dimensiones (el número de neuronas de la construcción), ya que dando valores de entrada a esas dos

neuronas (que forman lo que se denomina *espacio latente*) se reproducirían valores de la función original en 7 dimensiones. Dicho de otra manera, la función inicial estaría definida en $\mathbb{R}^7$, pero se habría encontrado una representación en $\mathbb{R}^2$, es decir, se habría reducido la dimensionalidad de los datos.

Es de destacar que hay varias técnicas para la reducción de dimensionalidad. Una de ellas es la SVD antes considerada (que cuando se aplica con este propósito se denomina también Análisis de Componentes Principales). Pero, como se ha descrito anteriormente, la SVD solo emplea transformaciones lineales de los datos mientras que los *Autoencoders* pueden usar complicadas transformaciones no-lineales. Para entender la diferencia entre la reducción de dimensionalidad proporcionada por la SVD y por un *Autoencoder*, consideremos el caso sencillo de una función en $\mathbb{R}^2$. La SVD puede identificar si los puntos de la función están sobre una recta (en cuyo caso la función 'vive' en $\mathbb{R}$), mientras que un *Autoencoder* puede detectar esto, pero también el caso en el que los puntos estén en, por ejemplo, una espiral (lo que también reduciría la dimensionalidad, ya que para representar un punto solo necesitaríamos saber la distancia *a lo largo* de la espiral con respecto a un origen arbitrario).

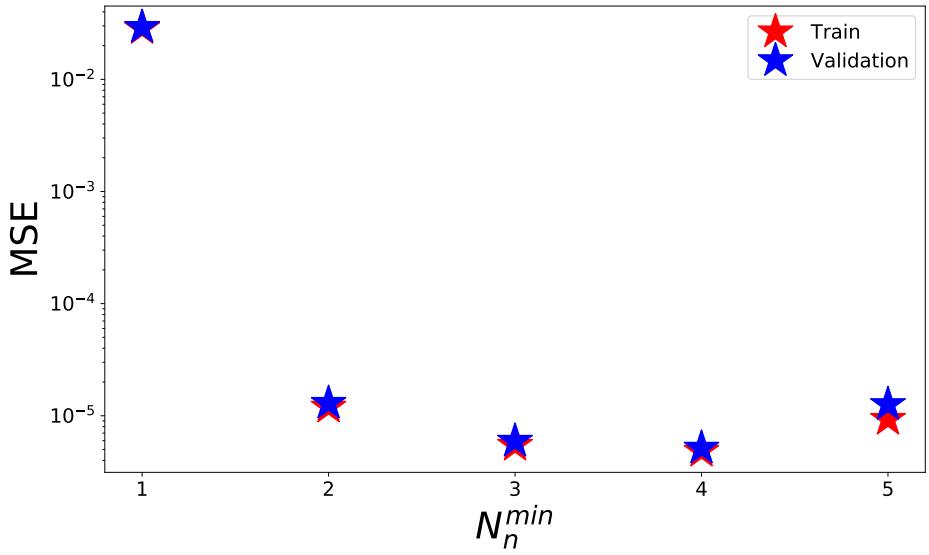

Figura 14: Valores de la función coste MSE al final del periodo de entrenamiento, en función del número de neuronas en la constricción de un *Autoencoder* del tipo $300-200-100-N_n^{\text{mín}}-100-300$. Los datos de entrada son los espectros de transmitancia $\{(T(k_0), T(k_1), \cdots, T(k_{300}))\}$.

Una aplicación de la Reducción de Dimensionalidad es encontrar el número mínimo de argumentos que son necesarios para representar una función, lo que puede ser muy útil en el proceso de crear modelos que expliquen datos experimentales.

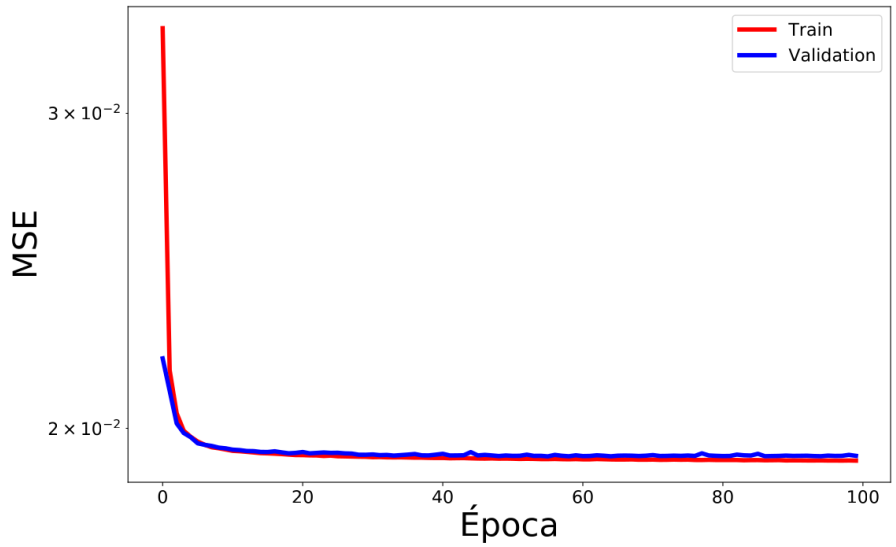

Figura 15: Curva de entrenamiento de un *Autoencoder* del tipo $300-200-100-2-100-300$ para reproducir los datos $(T(k_0), T(k_1), \cdots, T(k_{300}))$ a los que se ha añadido ruido, como se describe en el texto. La función de coste utilizada es el Error Cuadrático Medio (MSE).

Como ejemplo, consideremos los espectros de transmitancia detallados en la sección anterior $\{(T(k_0), T(k_1), \cdots, T(k_{300}))\}$. Sabemos que corresponden a la dispersión de un electrón por un potencial definido por dos funciones delta. Pero olvidemos esto por un momento y tomémoslos sencillamente como un conjunto de datos que han sido obtenidos de alguna manera (es decir, pueden ser espectros de otra magnitud física para la que quizás no tengamos aún un modelo, precios de 300 objetos en distintos momentos, o cualquier otro conjunto de datos). Consideremos ahora distintos *Autoencoders*, cada uno con un valor de $N_n^{\text{mín}}$. Cuando la MSE de un *Autoencoder* es baja éste proporciona una representación fidedigna de los datos, mientras que cuando la MSE es alta el *Autoencoder* no es capaz de representar bien la función con ese $N_n^{\text{mín}}$. Por tanto el $N_n^{\text{mín}}$ más pequeño para el que la MSE del *Autoencoder* es baja indica el número mínimo de parámetros que es necesario para modelizar los datos. La Fig. 14 muestra la MSE calculada para *Autoencoders* del tipo $300-200-100-N_n^{\text{mín}}-100-300$. La MSE es del mismo orden de magnitud para $N_n^{\text{mín}} = 2-5$, mientras que para $N_n^{\text{mín}} = 1$ la MSE es unas 3000 veces mayor. Esto indica que los datos $\{(T(k_0), T(k_1), \cdots, T(k_{300}))\}$ pertenecen a una distribución que ha sido generada por dos parámetros. Por supuesto, en este caso sabemos que esto es así y que los parámetros son $\delta_{1,2}$, pero el resultado parece mágico cuando se tiene en cuenta que esta información nunca ha sido proporcionada al realizar el cálculo descrito anteriormente (¡ni tampoco qué representa cada uno de los vectores $(T(k_0), T(k_1), \cdots, T(k_{300}))$!).

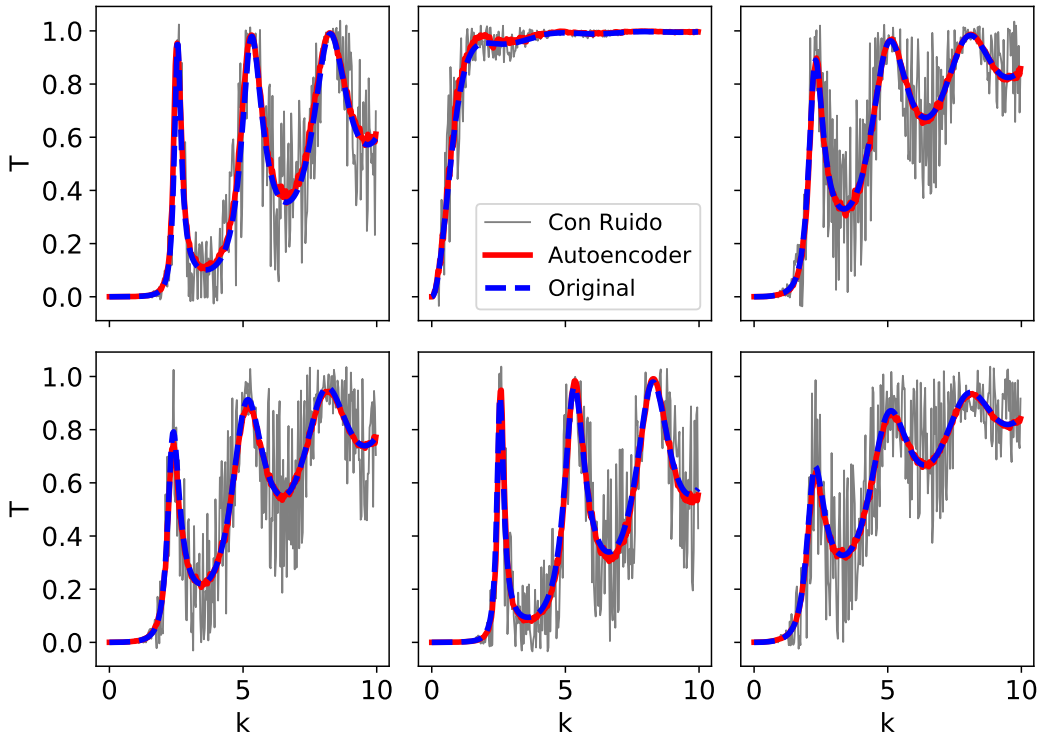

Figura 16: Comparación entre espectros de la transmitancia de un electrón moviéndose en una dimensión en presencia de un potencial caracterizado por dos funciones delta: (i) espectros con ruido [en gris] (ii) los filtrados utilizando un *Autoencoder* del tipo $300 - 200 - 100 - 2 - 100 - 300$, entrenado con los espectros ruidosos [en rojo] y (iii) los datos originales antes de añadirle ruido [en azul]. Distintos paneles son para distintos valores de $\delta_{1,2}$, extraídos de entre los usados en los datos de validación.

4.4. Filtrado de ruido.

Otra de las aplicaciones de los *Autoencoders* es la eliminación de ruido de un conjunto de datos. La idea es que al representar los datos con un *Autoencoder* con un $N_n^{\text{mín}}$ mucho menor que el número de neuronas de entrada se obliga a la RND a representar características que son comunes a todos los datos, eliminando por tanto el ruido (que es particular para cada dato).

Como ejemplo, se ha añadido un ruido aleatorio[4] a cada transmitancia. El *Autoencoder* considerado es del tipo $300 - 200 - 100 - 2 - 100 - 300$. La Fig. 15 muestra la curva

[4]Para generar el ruido r, para cada dato T se extrae un número aleatorio ξ de una distribución normal centrada en 0 y con varianza 1, y se calcula $r = 3\,\xi\,T\,(1-T)$. Los factores $T\,(1-T)$ no son esenciales para el cálculo, pero se introducen por simplicidad para que la transmitancia ruidosa siga estando, en su mayor parte, definida entre 0 y 1.

de aprendizaje, ilustrando que la red aprende tras unas pocas épocas pero que la MSE que se alcanza es mayor que en los casos considerados anteriormente. Esto indica que el *Autoencoder* no está reproduciendo los datos fidedignamente (lo que en este caso es una característica positiva, porque si lo hiciera ¡estaría reproduciendo el ruido!).

La comparativa entre los datos sin ruido, los datos con ruido y los predichos por el *Autoencoder* (cuando como entrada se le proporcionan los datos ruidosos) se ilustra en la Fig. 16, para varios espectros extraídos al azar de entre los datos de validación. Aunque se ha añadido mucho ruido a los espectros, la predicción del *Autoencoder* prácticamente coincide con los datos originales sin el ruido añadido, ilustrando claramente la gran capacidad que tienen los *Autoencoders* para el filtrado de ruido.

Nótese además que la aplicación del *Autoencoder* se puede combinar con otras técnicas tradicionales para reducir ruido, como por ejemplo hacer un filtrado preliminar de los datos usando métodos basados en la transformada de Fourier y aplicando el *Autoencoder* posteriormente.

5. Conclusiones

Tras una serie de altibajos en las expectativas de la Inteligencia Artificial, varios métodos han alcanzado una madurez que les permite ser aplicados a muy diversos problemas, alcanzando niveles en la identificación de correlaciones entre datos que son, en muchos casos, superiores a las que podemos conseguir los humanos. Las aplicaciones en el ámbito de las ciencias son numerosas y, de hecho, en estos momentos estamos viviendo una revolución tanto en el número de técnicas que aparecen como en el ámbito de su aplicación. En esta exposición se ha ilustrado cómo las Redes Neuronales Densas (un tipo de función definida por la aplicación sucesiva de transformaciones lineales y no-lineales) pueden ser utilizadas para representar complejas funciones multi-dimensionales, para el diseño inverso (encontrar los valores de los parámetros que proporcionan un cierto valor de una función), para el filtrado de ruido, e incluso como orientación al número mínimo de parámetros que debe tener una teoría que intente explicar un conjunto de datos.

Bibliografía

[1] D. Crevier, *AI: The Tumultuous History of the Search for Artificial Intelligence*. 01 1993.

[2] J. I. Latorre, *Etica para máquinas*. 2019.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. (MIT Press, 2016).

[4] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, Mastering atari, go, chess and shogi by planning with a learned model, *Nature*. **588**(7839), 604–609 (Dec, 2020). ISSN 1476-4687.

[5] M. A. Nielsen. Neural networks and deep learning, (2018). URL `http://neuralnetworksanddeeplearning.com/`.

[6] C. Cohen-Tannoudji, B. Diu, and F. Laloe, *Quantum mechanics; 1st ed.* (Wiley, New York, NY, 1977). URL `https://cds.cern.ch/record/101367`. Traducción de: Mecanique quantique. Paris : Hermann, 1973.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, (2014). URL `http://arxiv.org/abs/1412.6980`. Comentario: Publicado como artículo en la 3rd International Conference for Learning Representations, San Diego, 2015.

[8] F. Chollet et al. Keras. `https://keras.io`, (2015).