UNIVERSIDAD DE VALLADOLID

GRADO EN INGENIERÍA INFORMÁTICA: MENCIÓN COMPUTACIÓN

TRABAJO DE FIN DE GRADO

ExoplanetIA: Machine Learning para la detección de exoplanetas

Alejandro Del Hierro Diez

Julio 2020

Tutores

Benjamín Sahelices, Director de la Escuela de Ingeniería Informática

Manuel Hermán Capitán, Observatorio Tecnológico HP, HP | SCDS

Alejandro Viloria Lanero, Observatorio Tecnológico HP, HP | SCDS

Prefacio

Agradecimientos

Antes que nada quiero agradecer al Observatorio Tecnológico HP por darme la oportunidad de realizar mi trabajo de fin de grado en un proyecto tan interesante como este.

En detalle quiero agradecer su tiempo, dedicación y hospitalidad a Manuel Hermán Capitán y a Alejandro Viloria Lanero, quienes han estado ayudándome y me recibieron en sus oficinas como uno más.

Agradezco a la Universidad de Valladolid y en concreto a los profesores del Grado en Ingeniería Informática el trabajo que realizan para que los alumnos nos formemos en las bases y los principios de la informática y por motivarnos a continuar con una vida profesional dedicada a la misma.

En particular, doy las gracias a Benjamín Sahelices, director de la escuela, por darme a conocer la existencia del proyecto, por supervisar el desarrollo de este trabajo desde la perspectiva académica y por su tiempo e interés.

Finalmente, agradezco a mi familia y a mis compañeros por su apoyo incondicional para sacar adelante el grado y todas sus partes, incluyendo este trabajo.

Código y conjunto de datos

Todos los cómputos de este trabajo fueron implementados en lenguaje *Python*. El código esta disponible en el repositorio Gitlab: *https://gitlab.com/HP-SCDS/Observatorio/2019-2020/uva-exoplanetia.git*.

Los datos están accesibles en la plataforma *Kaggle* en el siguiente enlace: *https://www.kaggle.com /keplersmachines/kepler-labelled-time-series-data*

Índice general

Capítulo 1

Introducción

En las últimas décadas la cantidad de datos generados se ha incrementado exponencialmente. La facilidad de acceso a la red y de almacenamiento de datos ha facilitado que cualquier mínima acción pueda ser registrada con bajo coste. Los datos no sirven de mucho por sí solos, sin embargo, con las herramientas adecuadas pueden, literalmente, convertirse en oro.

En este contexto de generación masiva de datos como diamantes en bruto entran dos de las palabras posiblemente más escuchadas en los últimos años: *machine learning*. El *machine learning* constituye esa herramienta capaz de convertir los datos en información, y por consiguiente, en dinero. *Machine learning* puede traducirse directamente al castellano como "Aprendizaje máquina", y es que, sin entrar en aspectos filosóficos sobre el significado de aprender, esta herramienta engloba métodos que pueden hacer que una máquina u ordenador aprenda o extraiga información de los datos.

Dentro del concepto de *machine learning* se encuentra un grupo de algoritmos llamados **redes neuronales** que basan su funcionalidad en imitar el funcionamiento de las neuronas en el cerebro humano. Surgieron en 1943 por obra de McCulloch y Pitts, pero no fue hasta 1986 que se pudieron utilizar a gran escala por el aparecimiento de un algoritmo capaz de entrenar o hacer aprender a las redes, la retro-propagación.

Desde entonces el avance de las redes neuronales se ha visto principalmente condicionado por la potencia de cómputo de los ordenadores, hasta llegar a la década de 2010 en la que los ordenadores consiguen una capacidades suficiente para que este concepto se expandiese a casi todos los aspectos

del día a día: asistentes virtuales, coches autónomos, clasificación automática de imágenes...

HP | SCDS es un departamento de investigación y desarrollo de HP principalmente dedicado a la impresión de gran formato, y el interés de las redes neuronales también ha llegado a este campo. Entre las utilidades en el campo de la impresión se encuentran el mejorado de la calidad de imágenes, suavizado de contornos, reducción de ruido en escaneado o rellenado de zonas.

Desde su Observatorio Tecnológico, HP | SCDS ha puesto en marcha este proyecto con el objetivo de aplicar algunas de las tecnologías en el estado del arte de las redes neuronales para la búsqueda de exoplanetas basándose en los datos reales ofrecidos por el satélite espacial Kepler, con respecto a la intensidad de luz (flux) registrada durante más de 9 años.

Capítulo 2

Estado del arte y contexto tecnológico

2.1. La misión Kepler

La misión Kepler surgió como un intento de dar un paso adelante en la búsqueda de la respuesta a una de las más sonadas e intrigantes preguntas: ¿Cuál es nuestro lugar en el universo?

Se trataba de una de las primeras misiones capaces de encontrar planetas similares a la Tierra que orbitasen estrellas en la conocida como "zona habitable", ese espacio en la órbita de una estrella en el cual el agua estaría en estado líquido. Por la clara relación entre la vida y el agua en nuestro planeta, un planeta sería más adecuado para albergar vida siempre que el agua que contuviese estuviera en estado líquido.

2.1.1. Método del tránsito

Fue utilizado por primera vez por el astrónomo Edmond Halley para registrar el tránsito de Mercurio alrededor del Sol, y en las últimas décadas ha cobrado significativa importancia por su uso en algunas misiones, entre ellas, Kepler.

Cuando un cuerpo celeste pasa entre otro más grande (como una estrella) al cual orbita, y

el observador (en este caso, la sonda Kepler), se observa una disminución en la intensidad de luz procedente del cuerpo más alejado que es captada por el observador. Esta disminución de intensidad puede utilizarse para estimar algunas propiedades del cuerpo, como el tamaño o la densidad (figura 2.1).

Para poder registrar el tránsito, es necesario que el plano de la órbita del planeta con la estrella este alineado con el observador. Se puede calcular la probabilidad de observar el tránsito mediante la siguiente formula [1]:

$$P_t = (R_s + R_p)/\alpha,$$

donde P_t es la probabilidad deseada, R_s y R_p los radios de la estrella y del planeta, respectivamente, y α el semieje mayor de la elipse formada por la órbita. Aproximadamente, esto significa que si se poblase aleatoriamente de estrellas el universo, podrían observarse tránsitos de, a lo sumo, el diez por ciento de ellas.

2.1.2. El satélite

Kepler era diferente a la mayoría de satélites destinados a observar el universo. En lugar de realizar una fotografía de un punto concreto del universo, Kepler tomaba imágenes de espacios muy amplios incluyendo muchas estrellas.

En marzo de 2009 fue puesto en órbita heliocéntrica y con un rango de observación entre las constelaciones de Lyra y Cygnus. Este rango comprende en torno a seis millones y medio de estrellas, que con los cálculos explicados sobre el método del tránsito, hacen unas 500.000 estrellas que Kepler observaba simultáneamente, de las cuales la NASA seleccionó las 150.000 más interesantes [2]. Equipado con una cámara de 95 megapíxeles (la más grande enviada al espacio hasta el momento de su lanzamiento) y con un espejo de alrededor de un metro y medio, Kepler era capaz de detectar disminuciones de intensidad de luz de veinte partes por millón en esas 150.000 estrellas al mismo tiempo [3].

No solo era capaz de detectar objetos celestes orbitando estrellas. Gracias al registro de los tránsitos ocurridos en una estrella (habitualmente se ha considerado que una secuencia de tres tránsitos equiespaciados en el tiempo confirmaría la órbita del planeta alrededor de la estrella [4]), y al tiempo transcurrido entre ellos se podía estimar el tamaño de la órbita: cuanto menor periodo, mayor cercanía a la estrella. De esta manera se podría saber si el planeta se encuentra en la zona

habitable de la estrella. Además. en función de la luz sustraída de la estrella, era posible estimar el tamaño de la misma.

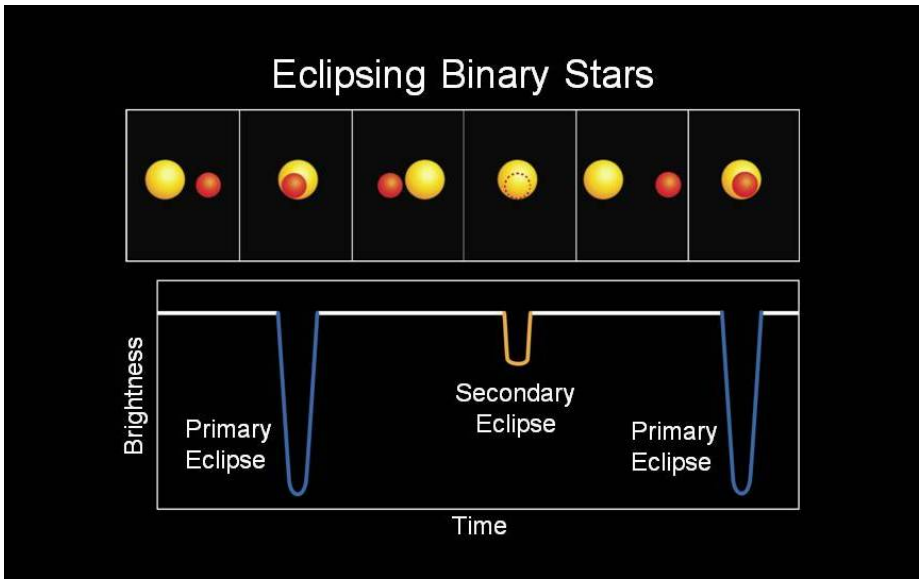

Figura 2.1: Curva de intensidad producida por el transito entre dos estrellas binarias. [5]

2.1.3. Resultados de la misión

La misión tenía una duración prevista de tres años y medio. No obstante, tras varias prolongaciones, su duración se extendió hasta los nueve años, para finalmente desactivar el satélite en noviembre de 2018 tras agotar todo su combustible.

Kepler ha arrojado resultados fascinantes sobre la cantidad y el tipo de los exoplanetas. Ha detectado unos 2600 exoplanetas confirmados y otros 2500 candidatos, lo cual hace alrededor del 70 % de los exoplanetas descubiertos hasta la fecha [6].

Pero su legado no se trata de una gran base de datos llena de exoplanetas. Kepler ha demostrado que los sistemas planetarios son más comunes de lo que se creía y que existen más planetas que estrellas. Gracias a sus resultados y estimaciones científicas, se sabe que de media cada estrella tiene al menos un planeta orbitándolo, y que entre el 20 y el 50 por ciento tiene un planeta rocoso del tamaño de la Tierra y en su zona habitable [7]. También se sabe que el tipo de planeta más común tiene un tamaño mayor que la Tierra pero menor que Neptuno, un tipo de planeta que no existe en

nuestro sistema solar [2].

Por desgracia, no todos los descubrimientos son esperanzadores. La mayoría de sistemas planetarios descubiertos son sistemas muy compactos, con los planetas orbitando alrededor de la estrella en un rango menor al de la órbita de Mercurio y con órbitas mucho más rápidas que el año terrestre [8].

2.2. Fundamentos teóricos

2.2.1. Perceptrón multicapa

El perceptrón es una de las primeras redes neuronales que se inventaron, por obra de Frank Rosenblatt en 1957 [9]. Consta de un vector de entrada, un vector de pesos y una salida función del producto escalar de ambos vectores, lo cual, como se verá en la siguiente sección, lo hacía muy similar a una regresión logística.

En la práctica, este clasificador resultaba muy limitado, motivo por el cual surgió el perceptrón multicapa, que incorporaba varios perceptrones simples plenamente conectados de forma secuencial, siendo la entrada de cada uno la salida del anterior.

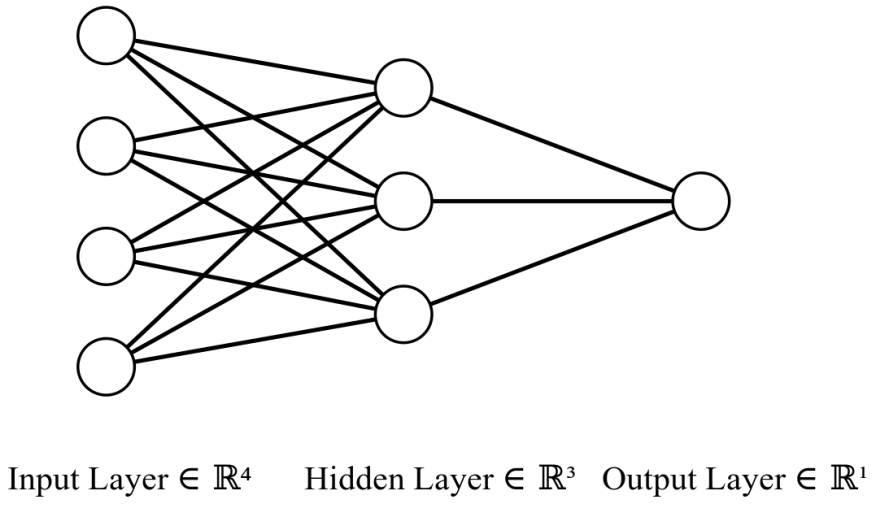

Figura 2.2: Perceptrón multicapa con una capa oculta.

Según el teorema de aproximación universal [10], un perceptrón del tipo de la figura 2.2 es suficiente para representar cualquier función, aunque en la práctica estas arquitecturas son bastante limitadas.

Para poder abordar problemas de una complejidad mayor, solo habría que añadir más capas al perceptrón, pero para entender qué implica esto, antes hay que comprender el funcionamiento de una neurona y la manera en que se entrena un perceptrón.

2.2.1.1. Neurona

El modelo matemático de la neurona se debe a McCulloch y Pitts, quienes lo desarrollaron en 1943 [11]. Las neuronas procesan la información a escala más básica, recibiendo una entrada, x, y produciendo una salida única y. Cada relación entre neuronas (representadas con círculos en la figura 2.2) tiene un peso asociado w_i, aunque para simplificar la notación se tratarán los pesos como asociados a las neuronas en lugar de a las relaciones. La salida de la neurona es una función del producto $X * W$, o lo que es lo mismo: $\sum x_i w_i$.

Es notable que con n entradas, una neurona computa una fórmula del siguiente tipo: $y = x_1 w_1 + x_2 w_2 + ... + x_n w_n$, lo cual puede recordar a una regresión múltiple, y es que, con una arquitectura como esta, una red neuronal solo será capaz de realizar regresiones más o menos complejas, o clasificadores que separen linealmente (o mediante hiperplanos) el espacio de las observaciones. El conjunto de observaciones podría no ser linealmente separable, por lo que añadir más capas de neuronas a la red haría que se utilizasen múltiples hiperplanos para separar las observaciones y obtener clasificadores.

McCulloch y Pitts definieron una función de activación para evitar la linealidad intrínseca de ese modelo. Dicha función era de tipo escalón, definida a trozos: si $\vec{x}\vec{w} > 0$, entonces $f(\vec{x}\vec{w}) = 1$, si no $f(\vec{x}\vec{w}) = 0$.

2.2.1.2. Función de activación

Rosenblatt utilizó para su perceptrón la función de activación definida por McCulloch y Pitts. Sin embargo, existen otros tipos de funciones de activación que fueron surgiendo según las necesidades:

- **Identidad**: $f(y) = y$, no introduce ninguna transformación ni ningún componente no lineal a la regresión efectuada por la neurona.

- **Función logística o sigmoidal**: $f(y) = 1/(1+\exp(-y))$, realiza una regresión logística. Toma valores entre 0 y 1, siendo los extremos los más probables. Esto lo convierte en una medida drástica y que, como veremos en posteriores apartados, podría dificultar el entrenamiento por obtener salidas con valor cero, favoreciendo el problema de *gradient vanishing*, que se verá en el apartado 2.2.1.6.

- **Tangente hiperbólica**: $f(y) = (1-\exp(-2y))/(1+\exp(-2y))$. Toma valores entre -1 y 1. Soluciona el problema de la función logística, pero sigue acotando los valores de la salida. Esta función, junto con la logística, son funciones derivables, lo cual facilita el ajuste de los pesos mediante descenso de gradiente, que basa su funcionamiento en cadenas de derivadas y que será explicado en el apartado 2.2.1.3.

- **ReLU** (Rectified Linear Unit): $f(y) = \{ \begin{matrix} y\ ;\ y \geq 0 \\ 0\ ;\ y < 0 \end{matrix}$. Daban más libertad a la salida, pero se volvía a tener el problema de convertir salidas en cero.

- **LReLU** (Leaky ReLU): $f(y) = \{ \begin{matrix} y\ ;\ y \geq 0 \\ \alpha y\ ;\ y < 0;\ \alpha \ll \end{matrix}$. Es una modificación de la anterior función para evitar convertir la salida en cero y mitigar el problema de *gradient vanishing*.

Como se verá posteriormente, la elección de la función de activación condicionará el funcionamiento de la red no solo en lo explicado en cada función, sino en el cálculo de sus gradientes.

2.2.1.3. Entrenamiento

Ya se ha explicado el paso de alimentación hacia delante (*feedforward*), que consiste en dar la salida de las neuronas de una capa a las neuronas de la siguiente. Las redes neuronales aproximan funciones y proporcionan salidas estimadas, por lo que el siguiente paso es medir cuánto difiere la salida de una red de la observación real. Si se tratase de una regresión, se podría utilizar una variante del error cuadrático medio (MSE), como por ejemplo: $\sum_i^n (y_i - t_i)^2/n$. El objetivo es por tanto minimizar este valor.

En esta línea y al igual que el MSE, dicha función tendrá un mínimo global. Para encontrar dicho mínimo es necesario saber si el valor de la función aumenta o disminuye cuando se cambian los pesos, lo cual es "fácilmente" calculable con la derivada parcial de dicha función en función de los pesos. Llamando E a una función cualquiera que mida el error:

$$\vec{d} = \delta E/\delta w_j \tag{2.1}$$

Moviéndose en dirección contraria a la indicada por $\vec{d}$, se avanzará hacia el mínimo absoluto. Los pesos se ajustarán entonces dependiendo de este valor en base a un factor de aprendizaje ν, el cual determinará en qué medida se actualizan los pesos:

$$w_j \leftarrow w_j - \nu \, \delta E/\delta w_j \tag{2.2}$$

Este proceso es ampliamente conocido como **descenso de gradiente**, y es extrapolable a cualquier problema con otras funciones objetivo a minimizar. En el caso de las redes neuronales, existen múltiples funciones diferentes que pueden ser minimizadas, en este apartado solo se ha dado un ejemplo.

2.2.1.4. Optimizadores

Es posible que la función de pérdida o función objetivo no sea cuadrática. Esto implica que no necesariamente el descenso de gradiente alcanzará el mínimo absoluto de la función. Si la función a minimizar tiene, por ejemplo, la forma de la figura 2.3, entonces, si se comienza desde el punto rojo avanzando hacia la izquierda tal y como dicta la pendiente en el punto, según lo explicado el descenso de gradiente se estancará en el mínimo local.

El optimizador clásico es el llamado momento (μ), que se añade a la fórmula de actualización de pesos anterior para dar una "inercia" al incremento o decremento que ha sufrido el peso en la actualización anterior de manera que:

$$\begin{aligned} \Delta_p w_j &\leftarrow \mu \Delta_{p-1} w_j - \nu \, \delta E/\delta w_j \\ w_j &\leftarrow w_j + \Delta_p w_j, \end{aligned} \tag{2.3}$$

siendo $\Delta_{p-1} w_j$ el incremento anterior. Así se consigue determinar la medida en la que el nuevo valor depende de los anteriores, es decir, cuanto más haya cambiado, más cambiará en la siguiente iteración. Esto ayuda al algoritmo del descenso del gradiente a no siempre seguir la dirección de minimización indicada por la pendiente.

En los últimos años se han desarrollado algoritmos que no requieran de otro parámetro a decidir, como es el momento. Es por eso que surgen los optimizadores adaptativos: separan el factor de aprendizaje de cada parámetro y lo van adaptando en el curso del aprendizaje. Uno de los más utilizados recientemente es el conocido como RMSProp. Este algoritmo adapta individualmente el

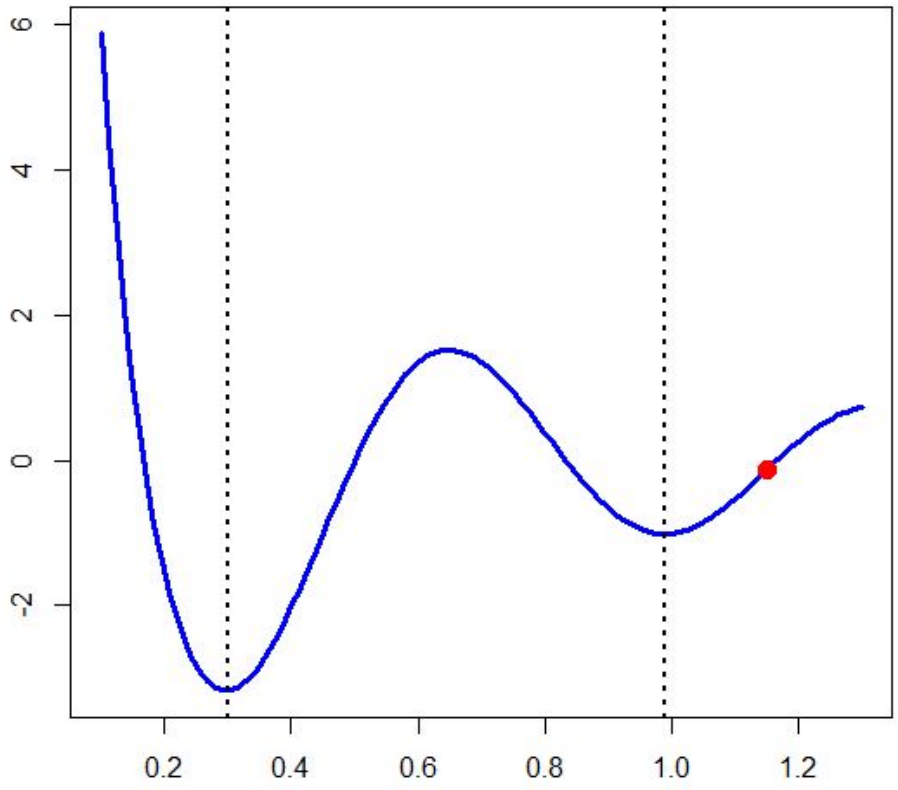

Figura 2.3: Función con un mínimo local y uno absoluto.

gradiente de todos los parámetros del modelo escalándolos mediante una media móvil del histórico de los gradientes con pesos exponenciales. De esta manera, factores que fueron usados mucho tiempo atrás en el entrenamiento serán menos influyentes.

Otro conocido algoritmo de optimización es el de los **momentos adaptativos** (AdaM). Es una combinación de RMSProp y el método del momento. AdaM estima el momento en función del gradiente y se lo aplica a los gradientes escalados de RMSProp.

2.2.1.5. Retro-propagación

El ejemplo de la regresión funciona para redes de una sola capa, o en caso de tener varias, solo contemplaría el error ocurrido en la última capa. Si una red tiene varias capas, sería útil conocer cuál es el error cometido por cada capa para ajustarlas de forma independiente.

Esto se hará mediante una retro-propagación del error, (en inglés, *backpropagation*), que consiste en propagar este error de la última capa a todas las anteriores. El proceso es algo tedioso pero se tratará de explicar lo más sencillamente posible [12].

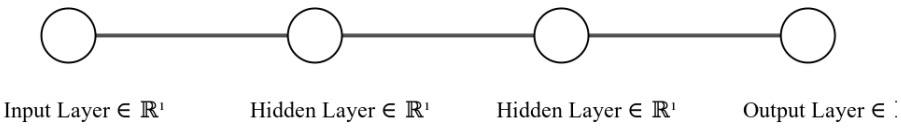

Figura 2.4:

Sea una red como la de la figura 2.4. Resulta interesante conocer cómo pequeños cambios en los pesos afectan al coste (MSE en la regresión) o función a minimizar. Sea $a^{(l)}$ el valor de $z^{(l-1)}w^{(l)}$, es decir, el producto de la entrada a la capa (o salida de la anterior) por los pesos. Si se tiene una función de activación σ, $z^{(l)}$ es la función $\sigma(a^{(l)})$. Aplicando la regla de la cadena:

$$\frac{\delta E}{\delta w^{(l)}} = \frac{\delta a^{(l)}}{\delta w^{(l)}} \frac{\delta z^{(l)}}{\delta a^{(l)}} \frac{\delta E}{\delta z^{(l)}} \tag{2.4}$$

En resumen, esto quiere decir que se calculan los cambios en el error viendo qué cambios produce en la salida de la capa una variación en los pesos, qué cambios produce esto en la activación y finalmente qué cambios produce la activación en el error.

Como la salida de una capa se obtiene de $a^{(l)} = z^{(l-1)}w^{(l)}$, se sabe que $\frac{\delta a^{(l)}}{\delta w^{(l)}} = z^{(l-1)}$. Por tanto, la actualización de los pesos se realiza mediante la siguiente fórmula:

$$w^{(l)} \leftarrow w^{(l)} - \nu \frac{\delta z^{(l)}}{\delta a^{(l)}} \frac{\delta E}{\delta z^{(l)}} z^{(l-1)} \tag{2.5}$$

Lo cual evidencia la **gran dependencia de la actualización de los pesos con la función de activación**, y el por qué de que las funciones de activación derivables faciliten el entrenamiento. Cabe recordar que el factor ν es el conocido como factor de aprendizaje, un parámetro que servirá para controlar cuánto se cambian los pesos de manera independiente al gradiente.

2.2.1.6. *Gradient vanishing*

Como se ha visto, retro-propagar el error implica encadenar productos de derivadas, donde los factores referencian errores cometidos (valores cada vez más pequeños). Cuantas más capas tenga una red, mayor será el número de multiplicaciones involucradas, lo que implicará que, si no se tiene cuidado, la magnitud final del gradiente puede ser muy poco significativa e incluso cero, provocando una actualización pobre o nula de los pesos. Además, como ya se introdujo en el apartado 2.2.1.2, si una función de activación convierte la salida en cero, este problema se verá incrementado.

En [13] se muestra cómo las primeras capas de una red neuronal son cruciales en su aprendizaje por ser capaces de encontrar las características más simples de los datos, mientras que las capas más lejanas son importantes para los detalles. Si se incurre en el problema de *gradient vanishing*, las primeras capas dejarán de aprender ya que sus pesos no serán modificados. A continuación se explica una conocida aproximación para mitigar el efecto del problema.

2.2.2. Bloques residuales

Los bloques residuales surgieron para solucionar varios problemas en el entrenamiento de las redes neuronas, y uno de ellos es el *gradient vanishing*. Su funcionamiento es relativamente sencillo de entender: consiste en poder "saltar" entrenamiento de capas con conexiones llamadas conexiones salto o conexiones residuales. Esto es lo que se observa en la figura 2.5.

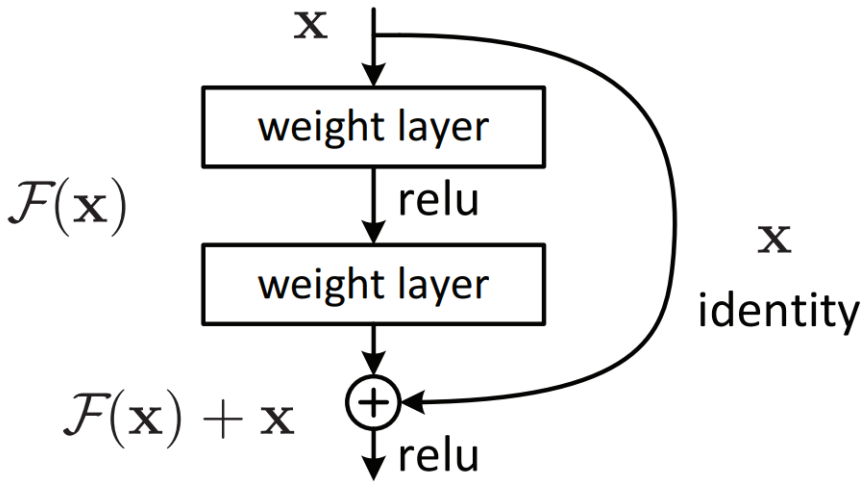

Figura 2.5: Diagrama de una conexión residual entre capas [14]

Un bloque esta formado por varias capas de neuronas conectadas secuencialmente. En la figura se observa un bloque de dos capas. Considerando este bloque, sea $a^{(2)}$ la salida de la segunda capa, se puede afirmar que la diferencia (o residuo) entre la entrada x y la salida del bloque es:

$$R(x) = a^{(2)} - x, \tag{2.6}$$

por lo que despejando

$$a^{(2)} = x + R(x) \tag{2.7}$$

El funcionamiento de la red es el que se explica en la figura 2.5. Al final del bloque se alimenta la salida de éste con la entrada, de manera que en la retro-propagación del error habrá parte de éste que no pase por todas las capas y se "salte" unos cuantos factores en la regla de cadena, puesto que la salida viene definida por capas que están antes en la secuencia. De esta manera se consigue un decrecimiento menos pronunciado del gradiente.

2.2.3. Redes recurrentes y LSTM

Las redes recurrentes son una interesante opción a desarrollar para este problema, puesto que sirven para tratar variables de secuencias que tienen una relación recurrente. Se verá en la sección 4.1 que el conjunto de datos del que se dispone tiene una serie de variables con relación recurrente entre ellas, por tratarse de series temporales.

Una red recurrente aplica la misma función sobre una secuencia de forma reiterada. Sea s_t el estado en el momento de tiempo t y x_t la entrada en el mismo instante (es decir, el valor que toma en el instante de tiempo t la secuencia de datos).

$$s_t = f(s_{t-1}, x_t) \tag{2.8}$$

Por lo tanto, según la función de recurrencia, se puede ver el estado como un resumen de todos los estados y entradas anteriores. Pero al igual que en los perceptrones, en las redes recurrentes también existen pesos que sirven para ajustar la red. En este caso, hay varios tipos:

- W: Aplica una transformación lineal al estado anterior de la red.

- U: Aplica una transformación lineal a la entrada.

- V: Aplica una transformación lineal al estado actual para producir la salida.

Por lo que la función anterior de cambio de estado se puede redefinir como sigue:

$$\begin{aligned} s_t &= f(s_{t-1} * W + x_t * U) \\ y_t &= s_t * V \end{aligned} \tag{2.9}$$

Del mismo modo que ocurría con los perceptrones, para evitar esta linealidad presente en las transformaciones con los pesos hay que aplicar una función no lineal, que será una de las funciones de activación presentadas en la sección 2.2.1.2. Esta función es la representada por f en la formula 2.9. En la figura 2.6 se aprecia el funcionamiento de una red recurrente y los parámetros que utiliza.

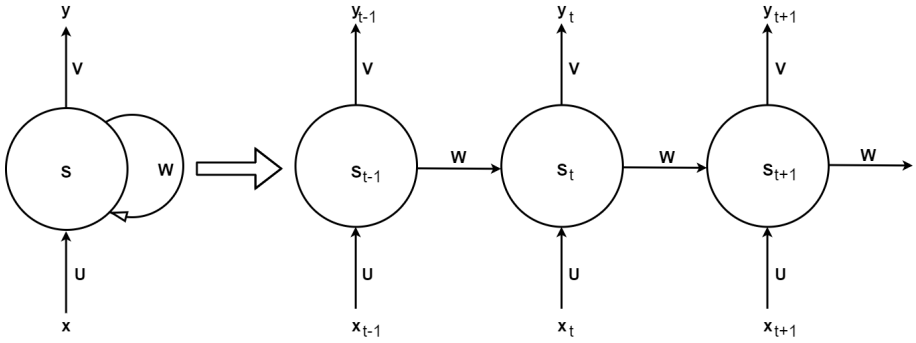

Figura 2.6: Red neuronal recurrente y su desdoblamiento. [5]

Múltiples variantes de este tipo de redes neuronales sirven para muchas tareas como la de clasificación que se usará en este problema (red recurrente *muchos-a-uno*), pero también para predecir valores de series temporales, para traducción de texto (redes *muchos-a-muchos*)...

Antes de explicar el tipo concreto de red recurrente que se va a utilizar (*Long-Short Term Memories*) es preciso entender cómo funciona el entrenamiento en este tipo de redes para explicar por qué estas redes funcionan mejor que las redes recurrentes clásicas.

2.2.3.1. Retro-propagación en el tiempo

En la figura 2.6 se observa la arquitectura de una red recurrente, con varias capas recurrentes apiladas. El entrenamiento de la red se realiza de una forma similar al perceptrón, pero por la naturaleza de esta arquitectura es necesario modificar la retro-propagación, puesto que en este tipo de redes se tienen dos entradas en cada instante de tiempo: x_t y s_{t-1}. A diferencia de las redes neuronales plenamente conectadas (perceptrones), en las redes recurrentes todos los pesos son compartidos a lo largo de la secuencia de tiempo.

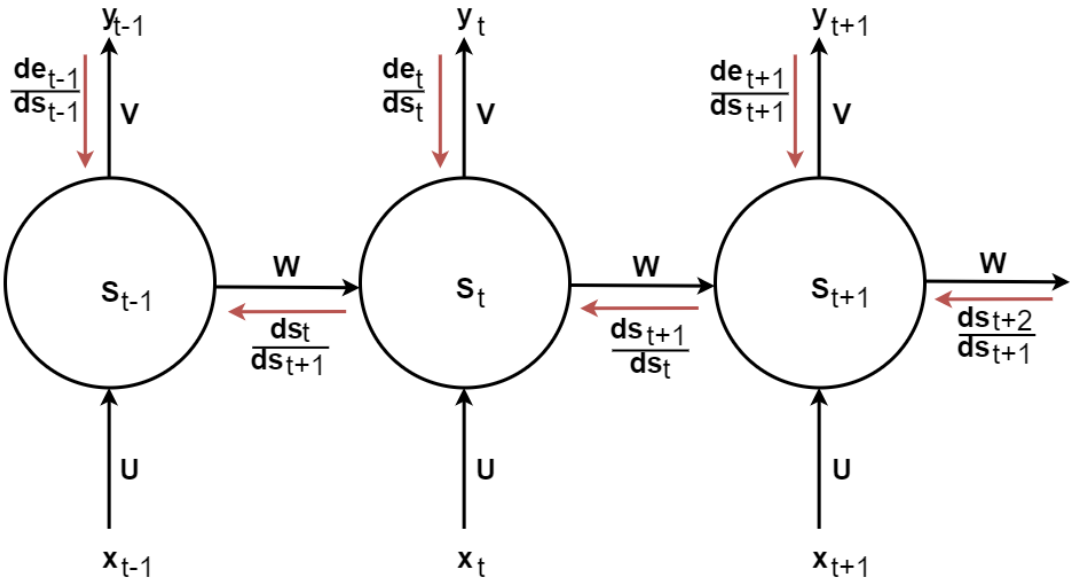

Figura 2.7: Retro-propagación en el tiempo [15]

Para evitar extensiones en la explicación, se contó con que las matrices V y U son 1. Cada capa de la figura 2.7 genera una salida y cada una de esas salidas tiene un error asociado con respecto al valor real en el momento de tiempo, e_t, por lo que el error total será la suma de todos estos:

$$\frac{\delta E}{\delta W} = \sum_{1 \leq t \leq T} \frac{\delta e_t}{\delta W} \tag{2.10}$$

Entonces, como las variaciones en un estado dependen de todos los estados anteriores:

$$\frac{\delta e_t}{\delta W} = \sum_{1 \leq k \leq t} \frac{\delta e_t}{\delta s_t} \frac{\delta s_t}{\delta s_k} \frac{\delta s_k}{\delta W} \tag{2.11}$$

$$\frac{\delta s_t}{\delta s_k} = \prod_{t \geq i > k} \frac{\delta s_i}{\delta s_{i-1}} \tag{2.12}$$

Todo esto se puede traducir en que los pesos se ajustan en base al cambio que producen en el coste en el paso k para los pasos $t > k$.

2.2.3.2. *Exploding* y *vanishing gradients*

Este tipo de redes también sufren de este problema. Se sabe que $\frac{\delta s_i}{\delta s_{i-1}} = W * s_{i-1}$, por lo que el cálculo del gradiente acaba por provocar una multiplicación de los pesos W tantas veces como larga sea la secuencia de datos. Esto provoca un crecimiento exponencial del gradiente en caso de que (si W fuera un escalar) $|W| > 1$, lo que sería el suceso de "*exploding gradient*", y un decrecimiento exponencial hacia cero previamente explicado si $|W| < 1$. Puesto que W es una matriz, estos fenómenos ocurrirán si el valor absoluto del mayor autovalor es mayor o menor que uno, respectivamente [15].

2.2.3.3. *Long short-term memories*

Se trata de un tipo de red recurrente capaz de solucionar el problema de los gradientes "seleccionando" solamente la información más útil. La idea básica es que añade al estado de las anteriores redes un estado de celda en el cual se escribirá o quitará información si no hay nuevas aportaciones relevantes.

Se utilizará la arquitectura mostrada en la figura 2.8 para introducir los conceptos relevantes. Por no saturar la imagen, se ha decidido no incluir los pesos que afectan a cada operación.

El primer concepto a comprender es la puerta de olvido, señalada con F en la figura (del inglés *Forget gate*). Puesto que se hace pasar el estado anterior s_{t-1} y que la función de activación asociada (sigmoide) solo puede tomar valores 0 o 1, esta puerta "decide" si mantener o borrar el estado de

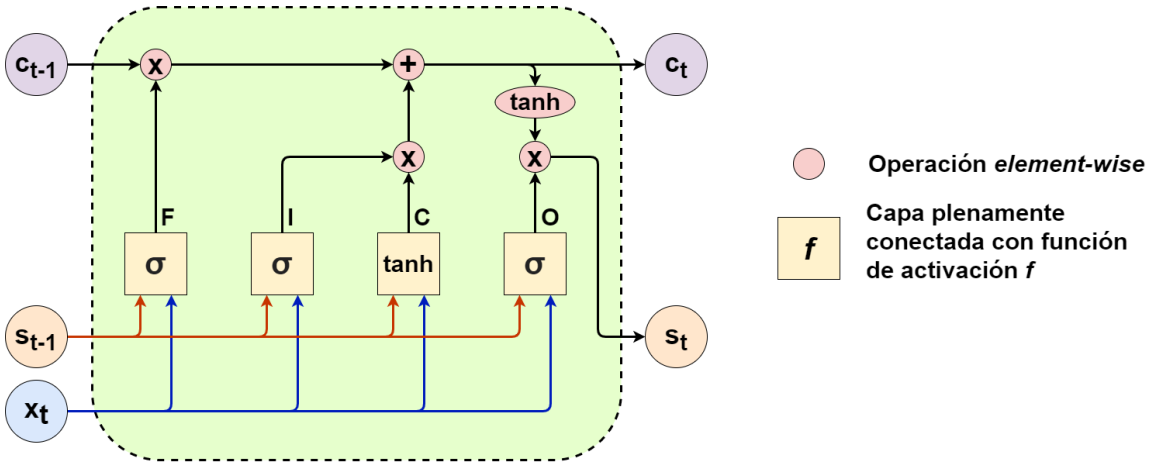

Figura 2.8: Módulo LSTM [16]

la celda. Para ilustrar su funcionamiento con pesos:

$$f_t = \sigma(W_f x_t + U_f s_{t-1}) \tag{2.13}$$

Por tener una activación sigmoidal la salida de esta puerta será 0 o 1, lo que provoca que al aplicar la multiplicación elemento a elemento sobre el estado de celda (c_{t-1} en la figura), se puede olvidar o hacer pasar dicho estado. Este es el paso en el que la LSTM puede eliminar información no útil.

El siguiente concepto es la puerta de entrada, nombrada I en la figura (del inglés *Input gate*). Su funcionamiento es similar a la puerta anterior y va muy ligado a la siguiente puerta, la puerta de estado (C en el diagrama). Esta última da un estado llamado candidato:

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i s_{t-1}) \\ c'_t &= tanh(W_c x_t + U_C s_{t-1}) \end{aligned} \tag{2.14}$$

En la figura se ve cómo estas dos funciones se combinan multiplicándose elemento a elemento. De este modo se decide que no se añade el nuevo estado de celda si la salida de la puerta de entrada

es 0, y se añade en caso contrario. Por lo que el nuevo estado de celda será:

$$c_t = f_t * c_{t-1} + i_t * c'_t \tag{2.15}$$

Finalmente la puerta de salida O (del inglés *Output*) decide la salida global de la celda. Como se observa, si el valor es 0 el estado anterior s_t se borrará y no habrá salida, y en caso de ser 1 se modificará el estado transformando elemento a elemento el estado de celda.

$$\begin{aligned} o_t &= \sigma(W_o x_t + U_o s_{t-1}) \\ s_t &= o_t * tanh(c_t) \end{aligned} \tag{2.16}$$

Todas las funciones enunciadas son diferenciables, por lo que se puede aplicar el descenso de gradiente. Los gradientes de este tipo de redes se formulan [17]:

$$\begin{aligned} \frac{\delta c_t}{\delta c_{t-1}} = \frac{\delta c'_t}{\delta c_{t-1}} = \frac{\delta f_t c_{t-1}}{\delta c_{t-1}} = c_{t-1}\frac{\delta f_t}{\delta c_{t-1}} + \frac{\delta c_{t-1}}{\delta c_{t-1}} f_t = f_t \\ \frac{\delta s_t}{\delta s_k} = \prod_{t \geq i > k} f_{t+k} \end{aligned} \tag{2.17}$$

Como se puede comprobar, ya no hay un productorio que implique ningún tipo de peso, por lo que se ha solucionado el problema de los gradientes que sí había en las redes recurrentes básicas. Visualmente se puede ver en la figura 2.8 que retro-propagar de c_t a c_{t-1} solo pasa por una función de multiplicación elemento a elemento, lo cual es mucho menor en términos absolutos que una multiplicación matricial. Además, cada paso se multiplica por una diferente función de puerta F, por lo que se evita este crecimiento exponencial.

2.2.4. Mapas auto-organizados

Propuestos por Kohonen, los SOM (del inglés *self-organizing maps*) son un tipo de red neuronal artificial típicamente utilizadas para problemas de aprendizaje no supervisado. Se diferencian de las redes neuronales tradicionales o *feed forward* en que en estas redes las relaciones de las neuronas no son secuenciales, sino que son por vecindad, además de que el entrenamiento no se realiza mediante retro-propagación.

2.2.4.1. Mapa y neurona

En la figura 2.9 se puede observar de forma gráfica cómo se distribuyen las neuronas en un SOM bidimensional. Cada una de las neuronas mostradas en la figura tiene un vector de pesos $w_i = [w_{i1}, w_{i2}, ..., w_{in}]$. El funcionamiento del mapa consiste en encontrar la neurona más cercana al vector de entrada $x = [x_1, x_2, ..., x_n]$. Hay muchos métodos para encontrar la neurona más cercana, pero generalmente el más utilizado es la distancia euclídea entre x_i y w_i. La neurona que satisfaga esta condición se llamará de ahora en adelante la neurona "ganadora".

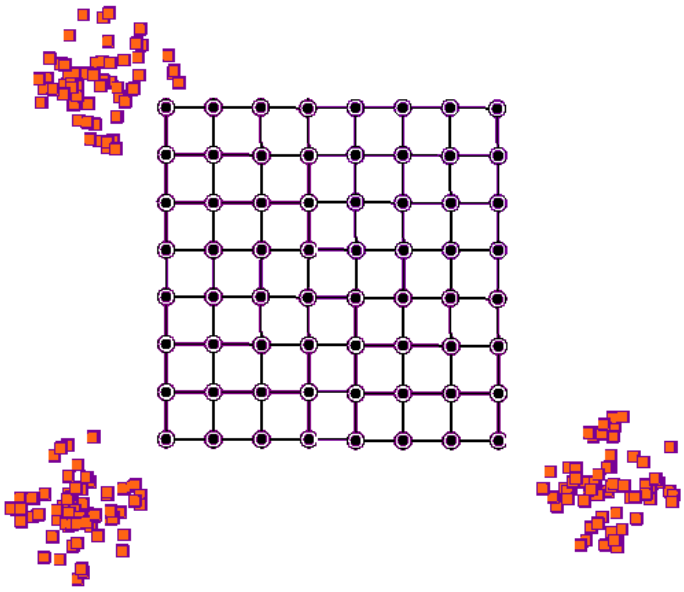

Figura 2.9: Neuronas antes del entrenamiento (rejilla) [18]

2.2.4.2. Actualización de pesos

Para cada vector de entrada, una vez encontrada la neurona ganadora, se actializan los pesos de las neuronas "vecinas". La función de vecindad esta muy relacionada con la estructura del mapa. Existen múltiples formas de medir la vecindad, como funciones de vecindad circular o mediante factores de vecindad, es decir, un factor que permita realizar un ajuste más o menos severo en función de la lejanía con la neurona ganadora. Sea cual sea la medida adoptada, está ampliamente

probado que el aprendizaje del mapa resulta más eficiente si el radio se va decrementando con el avance de las iteraciones.

Para el subconjunto de neuronas vecinas y la ganadora, Kohonen propone la siguiente actualización de pesos:

$$w_i(t+1) \leftarrow w_i(t) + \alpha(t)\left[\boldsymbol{x}(t) - w_i(t)\right] \tag{2.18}$$

siendo $\alpha(t)$ un factor de aprendizaje cuyo funcionamiento es el mismo que en la ecuación 2.2, que, al igual que el radio de vecindad, debe disminuir con las iteraciones.

2.2.4.3. Clustering y clasificación

El algoritmo original tiene como fin extraer una serie de características de un conjunto de datos. Las neuronas se irían agrupando (en términos de la distancia que separa los vectores de pesos de cada una) hasta formar un número de clústers. Pensando en un mapa bi-dimensional con un conjunto de datos con dos clases, las neuronas se agruparían idealmente en dos zonas definidas, de manera que entre ambas zonas las neuronas fueran lejanas y dentro de cada zona las neuronas fueran cercanas. Se puede ver como acabaría el mapa de la figura 2.9 en la figura 2.10 en un problema con tres clases.

No obstante, hay múltiples modificaciones del SOM que permiten utilizar la red para clasificación. A continuación se explican dos conocidas pero no las únicas maneras de abordarlo:

- Uso de la moda: para cada neurona, se anota la clase y_i de la cada observación x_i que la hizo ganadora. Al final, se asignará a cada neurona la moda de las clases y_i registradas. Para la clasificación de una nueva entrada, bastará con buscar la clase de la neurona más cercana.
- Etiquetado de neuronas: para cada observación x_i, se etiqueta la neurona ganadora con la clase y_i. A partir de aquí, la actualización de pesos y búsqueda de neuronas más cercanas solo se hará para aquellas neuronas sin etiqueta o etiquetadas con la misma clase. Finalmente se clasifica una observación asignando la etiqueta de la neurona más cercana.

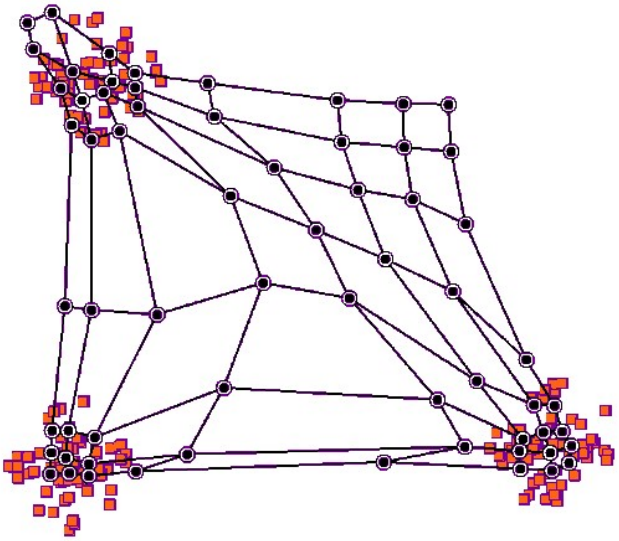

Figura 2.10: Neuronas tras el entrenamiento [18].

Capítulo 3

Proyecto y planificación

3.1. Objetivos

3.1.1. Conjunto de datos

Se pretendía realizar una correcta preparación de los datos disponibles. Entre otras medidas, se requería la correcta división de los datos en conjuntos de entrenamiento, validación y prueba teniendo en cuenta que los datos no están balanceados (más casos negativos que positivos), para que en cada conjunto las estrellas con exoplaneta tuvieran la misma representatividad.

Además, se quería probar con diferentes métodos de tratamiento de los datos, ya fuera para corregir el desequilibrio en los datos, o para ayudar a los modelos a la tarea de clasificación.

3.1.2. Redes neuronales

El segundo objetivo del trabajo consistía en estudiar la adecuación de tres modelos diferentes de red neuronal:

- Perceptrón multicapa.
- *Long-Short Term Memories*.
- Mapas auto-organizados.

Tras esto, se implementarían los modelos y diferentes ajustes de estos con variaciones de sus hiperparámetros y métodos.

Finalmente, se buscaba realizar una comparación de los resultados de estos modelos con los obtenidos por otras aproximaciones disponibles en la plataforma *Kaggle* [19].

3.1.3. Métricas

Se requería de un estudio de diferentes métricas posibles para la medición de la calidad de los modelos ajustados, que no solo pudiesen servir para compararlos, sino que también contasen con que los datos no están balanceados para obtener medidas de rendimiento fiables y acordes con el conjunto del que se dispone.

3.2. Metodología de trabajo

El proyecto se ha llevado a cabo mediante una metodología en **cascada**. Dicha metodología es una de las más clásicas en el desarrollo de proyectos software, y cuyos detalles pueden verse desarrollados en [20]. Mediante esta metodología se fue desarrollando cada uno de los tres modelos del segundo objetivo de forma teóricamente paralela con cada uno de los modelos, pero que en práctica fue secuencial con cada una de las fases puesto que sólo se disponía de un desarrollador.

Esto quiere decir que primero se analizaron las tres redes y sus posibilidades, luego se diseñaron fijando los parámetros, posteriormente se implementaron y finalmente se probaron. En la figura 3.1 se puede ver este orden de las actividades para cualquiera de los modelos.

El desarrollo de la metodología en cascada es fundamentalmente secuencial hacia abajo si se tiene como referencia el diagrama, aunque por la naturaleza de todo proyecto, y en concreto el de este, en ocasiones surge la necesidad de retroceder en los pasos para realizar ajustes en fases previas. En este caso fue muy necesario volver reiteradamente de la fase de pruebas a la fase de diseño por la necesidad de cambio de hiperparámetros en las redes neuronales.

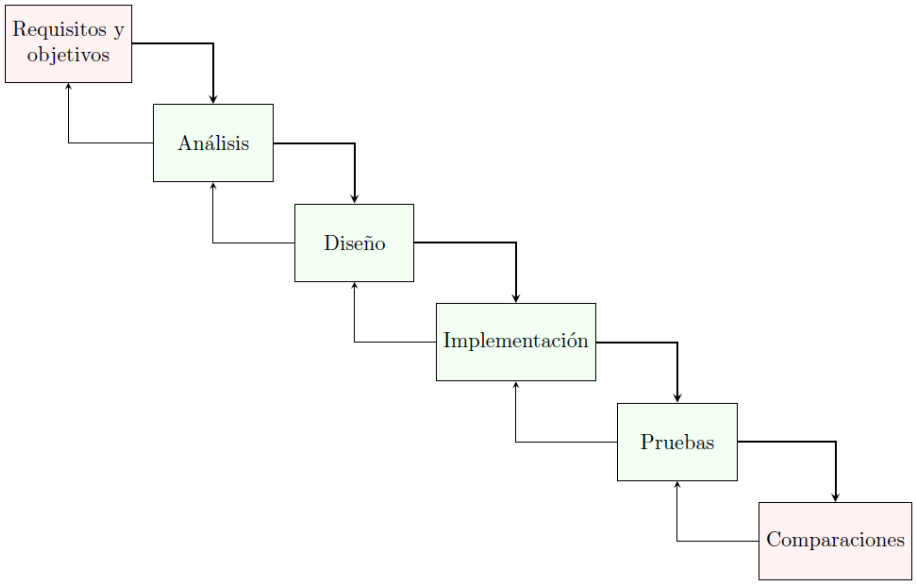

Figura 3.1: Rojo: Actividades comunes para todos los modelos. Verde: Actividades que se realizaron para cada modelo paralelamente.

3.2.1. Identificación de actividades

Se hizo una identificación de las actividades principales necesarias para la consecución de los objetivos y se plasmaron en una **estructura de descomposición del trabajo** visible en la figura 3.2. Cabe destacar que todos los niveles se realizaron para los modelos comentados en los objetivos.

En este diagrama se pueden ver las actividades llevadas a cabo, aunque, por no sobrecargar la imagen, no desglosadas hasta el más bajo nivel.

De la misma manera, en la figura 3.3 se puede observar un diagrama de Gantt con la planificación del desarrollo de las tareas. Flechas dirigidas del final de una tarea al principio de otra indican que la tarea posterior depende de la finalización de la tarea anterior.

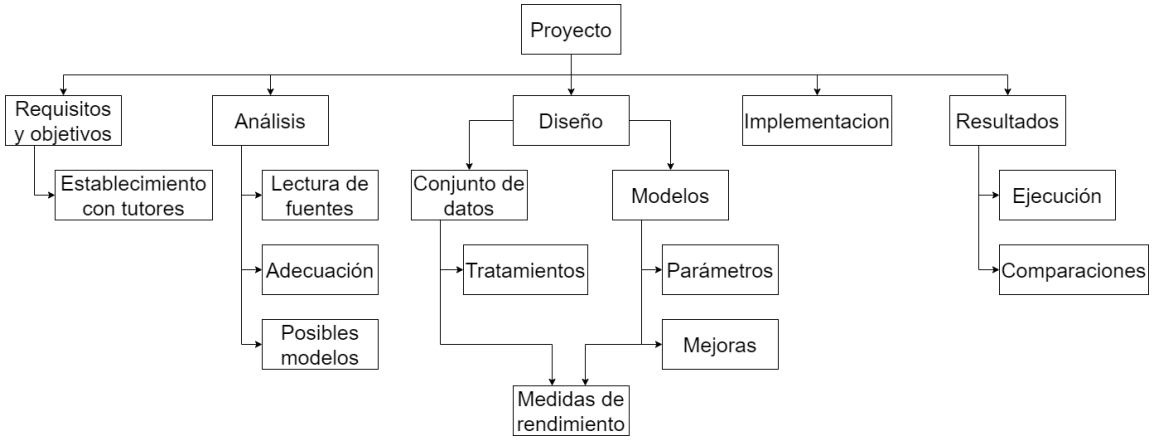

Figura 3.2: Estructura de descomposición del trabajo.

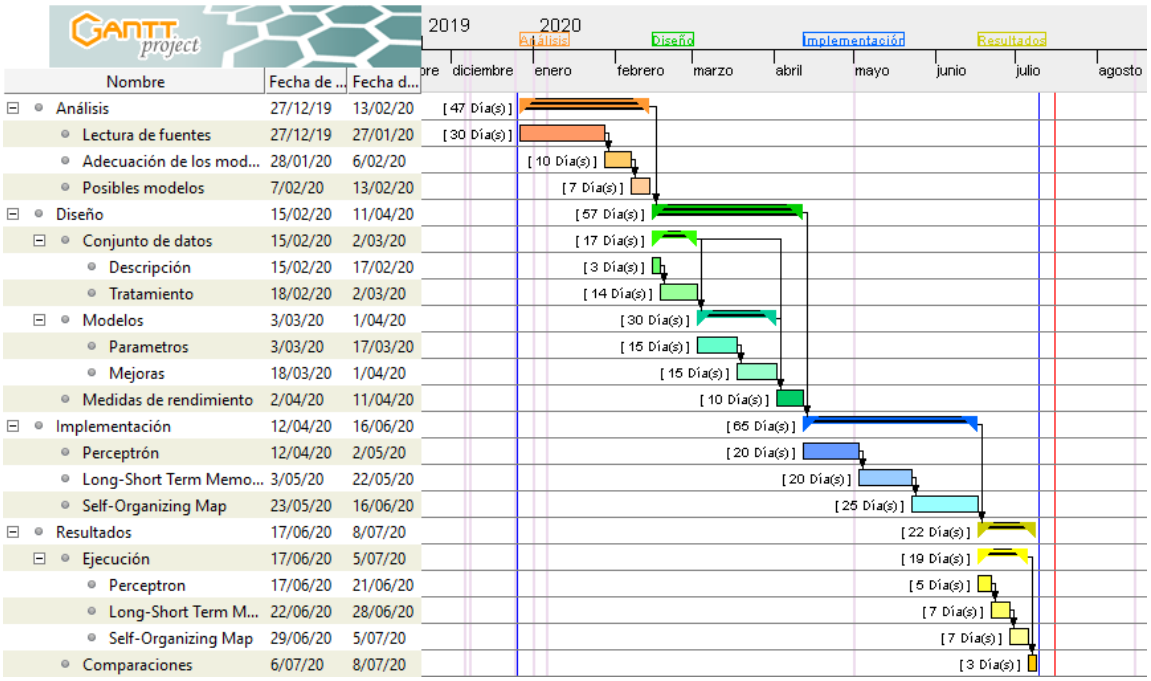

Figura 3.3: Diagrama de Gantt.

Capítulo 4

Materiales y métodos

4.1. Conjunto de datos

Se dispone de un conjunto de datos que posee series temporales de intensidad de luz (Flux) recogida de 5087 estrellas, en 3198 momentos de tiempo. Cada una de las observaciones o estrellas esta etiquetada con "2" o "1" si dicha estrella poseía un exoplaneta orbitándola o no, respectivamente. Este conjunto de datos contiene todas las observaciones de la tercera campaña de Kepler, además de otras estrellas con exoplanetas confirmados provenientes de otras campañas. El conjunto, por tanto, ha sido preparado previamente para su uso directo [21].

En la figura 4.1 se muestra una comparativa del Flux para dos estrellas con diferente clase, observándose mayor variación en aquella con al menos un exoplaneta confirmado, posiblemente debido a los tránsitos de dichos exoplanetas.

Se puede observar que el valor oscila en torno al cero. La unidad de medida del Flux es el lúmen, y, contrariamente a lo que se observa, no se espera que haya valores negativos. Esto se debe a que la NASA realizaba un fuerte preprocesado de los datos que le llegaban desde el satélite [22]. Entre otras medidas estaban:

1. Calibración: conversión de los datos en crudo a píxeles calibrados. Se realizaban correcciones a nivel de píxel, como reducción del sesgo, asociación entre fotoelectrones captados y unidades digitales, o la corrección de campo plano, que maperará el porcentaje de cambio en el brillo

de cada píxel en relación con la media.

2. Análisis fotométrico: construcción de series temporales (curvas de luz) en función de los píxeles del anterior paso. Entre otras tareas de este paso estaban la corrección de tiempo baricéntrico, de ruido causado por rayos cósmicos o correcciones debidas al movimiento elipsoidal propio del satélite.

3. Acondicionamiento de datos: correcciones a las curvas de luz. Las tareas pasaban por corregir discontinuidades en los datos, eliminación de excesos de flux (provocado por fuentes cercanas pero diferentes al objetivo), e identificación de datos atípicos o *outliers*.

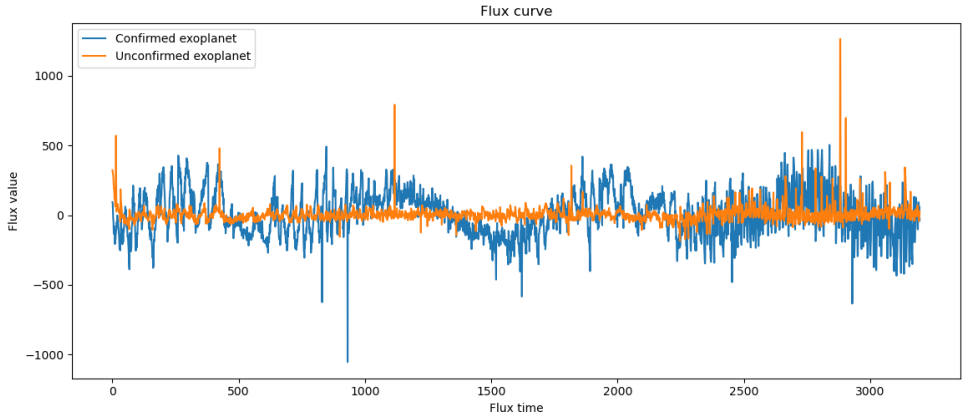

Figura 4.1: Flux para dos estrellas de diferente clase.

4.2. Framework

Son varias las opciones que hay hoy en día si se quiere trabajar con redes neuronales. Se realizará un breve análisis y una pequeña comparativa sobre tres diferentes *frameworks* para tratar de tomar una decisión sobre cual se utilizará para el desarrollo del proyecto. La mayoría de ellas, o al menos las que se comparan en esta sección (PyTorch, TensorFlow y Keras), tienen como unidad de básica de almacenamiento de datos el tensor. A niveles prácticos, un tensor es similar a un vector multidimensional.

Típicamente, los tensores pueden almacenar valores enteros o dobles, tienen un número de ejes (cuyo equivalente al álgebra tradicional sería la dimensión del vector) y una forma o tamaño de

cada eje.

Todas ellas representan las redes como un grafo de operaciones. La manera de trabajar es definiendo la arquitectura de la red, el número de capas y las funciones de activación, y las librerías automáticamente realizaran el entrenamiento mediante retro-propagación del error.

4.2.1. Tensorflow

Se trata de una biblioteca matemática para el desarrollo de redes neuronales creada por Google. Su sintaxis es de bajo nivel, aunque las ultimas versiones permiten utilizar Keras sobre Tensorflow, lo cual facilita su uso. Permite utilizar la CPU o la GPU para acelerar el procesamiento, algo que lo convierte en una de las opciones más rápidas junto con PyTorch.

4.2.2. Keras

Biblioteca para desarrollo de redes neuronales escrita en Python, con sintaxis de alto nivel para facilitar el prototipado de redes y la experimentación rápida. Es capaz de ejecutar sobre Tensorflow, así como sobre otras librerías como Theano o CNTK. Debido a ello, su uso disminuye la eficiencia.

Si bien nació de forma independiente, su crecimiento en los últimos años ha ido de la mano de Tensorflow, hasta el punto de que ellos mismos recomiendan utilizar el módulo Keras que implementa Tensorflow.

4.2.3. PyTorch

Desarrollada por Facebook, es una biblioteca utilizada para aplicaciones de *machine learning* de menor nivel que Keras y ligeramente más alto nivel que TensorFLow. También permite utilizar la GPU.

Tras realizar algunas redes de prueba, se puede concluir que PyTorch y Tensorflow ofrecen características muy potentes. PyTorch ha sido más tedioso de utilizar por no tener módulos como los que sí que tiene Tensorflow, obligando a codificar manualmente algunos bucles de entrenamiento y alimentación de la red, siempre teniendo cuidado de que los datos sean tratados como tensores.

Tensorflow, por su parte, ofrece algunos módulos con funciones predefinidas que facilitan la

creación de modelos y su comprensibilidad. Es cierto que la mayoría de dichas funciones pertenecen al módulo Keras, pero incluso en los tutoriales de la página oficial de Tensorflow hacen uso de ellas. Aún así, permite crear bucles y capas personalizados al igual que PyTorch.

Por estos motivos, se desarrolló el proyecto con **Tensorflow**, principalmente para hacer uso de **Keras** a través de este, pero manteniendo la potencia que ofrece por sí mismo en caso de ser necesaria.

4.3. Metodología

4.3.1. Tratamiento de los datos

4.3.1.1. División en conjuntos de entrenamiento, validación y prueba

En un problema de aprendizaje automático es necesario dividir el conjunto de entrenamiento en otros subconjuntos:

- Entrenamiento, con el que se construye el modelo.
- Validación, para ajustar los parámetros.
- Prueba, con el que se calculan las medidas de rendimiento reales del modelo.

Idealmente, estos conjuntos serían independientes, obtenidos aleatoriamente y suficientemente grandes. El conjunto del que se dispone es grande, pero limitado. Esto repercute en que cualquier obtención de subconjuntos los hará dependientes, pero no existen mejores aproximaciones. El conjunto de prueba, previamente definido, se reservará y usará únicamente para la estimación final de la tasa de error y otras medidas del rendimiento de las redes, consiguiendo así que el clasificador obtenido y este conjunto sean independientes.

La división del resto de datos en entrenamiento y validación tiene inconvenientes en la estimación de tasas de error debido a que ambos conjuntos deberían ser suficientemente grandes, y hacer grande el entrenamiento compromete la estimación (se obtendrán mejores clasificadores pero estimaciones de tasa de error con más varianza), así como hacer grande el de validación compromete el entrenamiento.

Para solucionar esto existen dos técnicas ampliamente conocidas de reserva de datos: ***Holdout*** y validación cruzada. La primera consiste en dejar fuera para la validación un porcentaje de los datos (típicamente el 33 %) lo cual sería el caso previamente hablado. Puede realizarse repetidamente para reducir la varianza de la estimación, no obstante, por la selección aleatoria haría que los conjuntos de validación se solapasen y no fueran plenamente independientes.

Por otra parte, la validación cruzada consiste en dividir el conjunto en varias particiones, y utilizar cada una de ellas como validación y el resto como entrenamiento. Es trivial pensar que de esta manera los conjuntos de prueba no se solapan, sin embargo, los cómputos llevados a cabo son pesados y complejos y utilizar este método supondría un notable incremento del tiempo de cómputo.

Para la división en conjunto de entrenamiento y validación se realizarán **muestreos estratificando por clase** para que todos los subconjuntos tengan la misma representatividad en términos de clase. De otra manera podría elaborarse un clasificador con observaciones de un solo tipo, algo probable en este conjunto tan desbalanceado. En [23] se encuentra una profundización en los conceptos tratados en esta sección.

4.3.1.2. Datos desbalanceados

Un conjunto de datos esta desbalanceado cuando se tienen proporciones muy diferentes de datos para cada clase del conjunto. En el conjunto de datos del que se dispone, tan solo 37 de las 5087 estrellas están etiquetadas como poseedoras de al menos un exoplaneta, lo que supone en torno al 0,7 % de ellas. Es preciso utilizar alguna técnica para tratar con este problema y evitar que las redes neuronales aprendan a predecir la clase mayoritaria.

Existen métodos que permiten realizar un aprendizaje sensible al coste teniendo en cuenta el coste de clasificar mal una instancia. Sin embargo, se desconoce cual es el coste de obtener falsos positivos o falsos negativos en el caso de estudio, por lo que se ha decidido modificar el conjunto de datos para obtener un entrenamiento sensible al coste.

Son muchas las técnicas existentes para este propósito. De las utilizadas en este estudio, la primera pasa por sub-muestrear la clase mayoritaria de forma aleatoria hasta alcanzar el número de instancias de la clase minoritaria, técnica que proporciona buenos resultados como se ha probado en [24], [25].

Entre las técnicas de muestreo, también se puede re-muestrear la clase minoritaria, ya sea con

las mismas instancias o con instancias sintéticas. En este trabajo se optó por la segunda opción utilizando el método *Synthetic Minority Oversampling Technique* o SMOTE, basado en la obtención de individuos sintéticos situados entre los "k" vecinos más próximos a cada observación de la clase minoritaria [26]. Con esta técnica se incrementaron las instancias de la clase minoritaria hasta formar el 25 % de las instancias de la clase mayoritaria.

Otra medida tomada es la asignación de pesos a las instancias. Idealmente y conociendo el coste de provocar falsos positivos o falsos negativos, los pesos asignados a las instancias corresponderían al coste relativo de provocar cada error. Desafortunadamente se desconoce el coste de los errores, por lo que tal y como se recomienda en [27], se ha optado por asignar pesos a las clases en función de su representatividad en el conjunto de datos:

$$\begin{aligned} w_0 &= \frac{1}{\sum_{y_i=0} 1} * total/2 \\ w_1 &= \frac{1}{\sum_{y_i=1} 1} * total/2, \end{aligned} \tag{4.1}$$

siendo w_j el peso asignado a una instancia con clase j y $total$ el número total de observaciones. La división por dos sirve para mantener el total de la función de pérdida o coste (la función de error de la ecuación 2.1).

4.3.1.3. Transformada de Fourier

Como se ha explicado anteriormente, se trata de un conjunto que dispone de la variación de intensidad de una señal (luz) a lo largo de varios periodos de tiempo. Por ello podría ser interesante saber si la frecuencia del tránsito en una estrella con exoplaneta confirmado se compone de frecuencias distintas que las de aquellas sin exoplanetas, algo que a priori sí ocurre si se piensa que una estrella sin exoplaneta no tiene variaciones. Para esto es útil la transformada de Fourier, la cual consigue transformar una escala de tiempo como la que se dispone en escala de frecuencias mediante la siguiente fórmula:

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-2\pi i f} \delta t, \tag{4.2}$$

siendo $g(t)$ la señal de una estrella en función del tiempo y $G(f)$ su transformada de Fourier en función de la frecuencia.

La transformada de la señal de una estrella sin exoplaneta, que a priori es constante o cercana a constante, tendría una energía concentrada en frecuencias bajas, mientras que una estrella con exoplaneta lo tendría en frecuencias más altas. Por esto, una red neuronal podría aprender a diferenciar entre señales con frecuencias más altas o más bajas para realizar la clasificación.

En la figura 4.2 se observan las transformadas de los ejemplos anteriores. Es interesante ver como una transformada de Fourier es capaz de encontrar la frecuencia principal en una señal. La línea azul para la estrella con exoplaneta es un claro ejemplo de una transformada de Fourier de una señal sinusoidal, tal y como se apreció en la figura 4.1, mientras que la naranja para la estrella sin exoplaneta presenta un patrón similar pero con la energía más distribuida por todo el rango de frecuencias.

4.3.2. Medidas del rendimiento

Se emplearon tres métodos diferentes para medir la calidad de los modelos. El primero de ellos es la precisión del clasificador, o lo que es lo mismo, la tasa de acierto. Se calcula como $N^o aciertos/Total\ observaciones$. Si bien es cierto que se trata de una medida muy directa del desempeño de un clasificador, no siempre es adecuada. Un clasificador (o red neuronal) que clasifica a todas las estrellas como negativas solo cometería 37 fallos, por lo que su precisión sería de $5050/5087 = 0{,}993$, un valor muy elevado para una red que se sabe que no sería útil por el desbalance de las clases.

Por ello es de vital importancia en este problema el uso de otras medidas del rendimiento. Una de las utilizadas en este trabajo es la definida en [25], que calcula un *Score* en función de la especificidad y la sensibilidad del modelo:

$$\text{Score} = \text{Succ} * (\alpha * \text{Sens} + \beta * \text{Spec}) \tag{4.3}$$

Succ hace referencia a la tasa de aciertos. La sensibilidad y la especificidad pueden obtenerse del número de verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN):

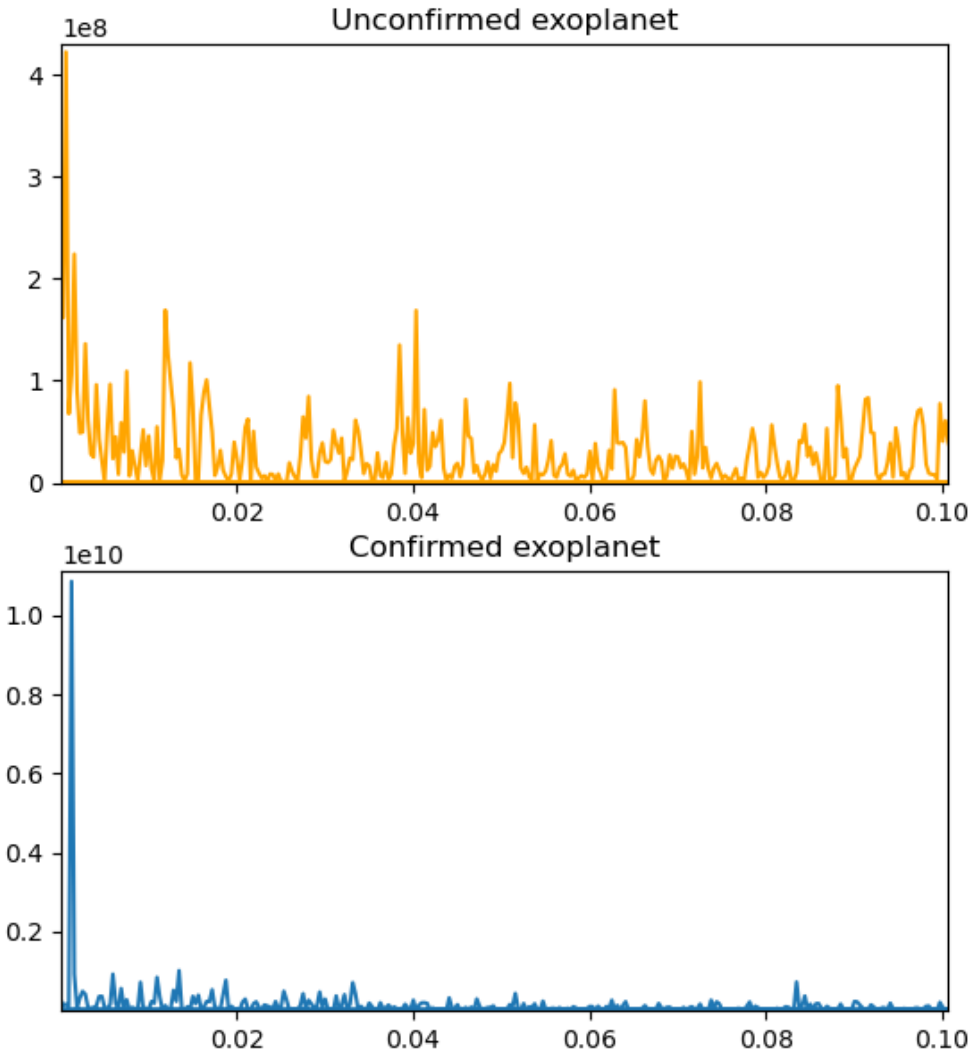

Figura 4.2: Transformadas de Fourier para las estrellas de la figura 4.1.

$$Sens = \frac{TP}{TP + FN} \tag{4.4}$$

$$Spec = \frac{TN}{TN + FP} \tag{4.5}$$

En resumen, este *Score* corrige la precisión con una media ponderada de la sensibilidad y la especificidad. Los pesos α y β se mantuvieron en 0.5 tal y como se hizo en [25].

4.3.2.1. Curvas ROC

La última medida utilizada es el área bajo la curva ROC (AUC). Una curva ROC dibuja en el espacio la tasa de verdaderos positivos frente a la tasa de verdaderos negativos, es decir, la curva se define por puntos de pares (FP/N, TP/N).

Las redes neuronales con una función de activación sigmoidal se pueden interpretar como clasificadores que asignan una probabilidad a las observaciones de ser positivas. Es decir, si una red da una salida de 0.8 a una entrada, podría interpretarse como que la probabilidad de que esa entrada sea positiva es 0.8. Para clasificadores de este tipo las curvas ROC tienen mucho interés por el modo en el que son construidas: dadas las probabilidades de una predicción, para cada probabilidad se obtiene un punto (FP/N, TP/N) de la curva considerando que se clasifican como positivos todas aquellas observaciones cuya probabilidad sea superior a la actual. De esta manera una curva ROC permite decidir en qué umbral situar la cota para realizar la predicción. Un buen umbral será el dado por la probabilidad que se obtuvo para una observación que aumentó la tasa de verdaderos positivos manteniendo baja la de falsos positivos.

Visto el funcionamiento, la mejor curva ROC posible es aquella que consigue un punto en el (0,1), obteniendo un área bajo la curva de 1. Por ello, el área bajo la curva es un valor numérico entre 0 y 1 que facilita la comparación de dos clasificadores, siendo mejor clasificador aquel que mayor área bajo la curva tiene, motivo por el que ha sido utilizada en este estudio. Si el AUC es 0.5, o la curva es diagonal, es indicativo de que la clasificación se ha hecho de manera aleatoria.

Una vez evaluados los modelos con técnicas generales como son el *Score* y el AUC, es interesante saber qué valor de sensibilidad (usualmente llamado *Recall* en inglés) tiene un modelo concreto, ya que es una medida de cuántos de los verdaderamente positivos se predijeron como positivos. El interés de esta medida para este problema parte de que podría no ser tan interesante obtener una buena tasa de verdaderos negativos como obtener una buena tasa de verdaderos positivos, indepen-

dientemente de que el clasificador provoque falsos positivos (dentro de unos márgenes razonables) para poder realizar un descartado rápido de observaciones.

4.3.3. Perceptrón multicapa

Se ha implementado tanto un perceptrón multicapa clásico con diferentes arquitecturas. En el caso de estudio se requieren predicciones binarias, por lo que una salida con función logística es adecuada. Esto se debe a que, en términos probabilísticos, la variable a predecir sigue una distribución binomial y la red necesitaría predecir $P(y = 1|x)$, es decir, la probabilidad de que la clase sea 1 dada la entrada. Tal y como se demuestra en [28], con activación lineal se tendrían gradientes de valor 0. La salida deseada en este caso es la sigmoidal o función logística, para poder transformar la salida lineal en una probabilidad.

La elección de la función de coste a minimizar esta muy relacionada con la elección de la salida de la red. En el caso de activación logística, el principio de máxima verosimilitud (o entropía cruzada [29]) asegura que el gradiente no se reduzca desmesuradamente con predicciones incorrectas [28].

En las capas ocultas del perceptrón, se probaron varios tipos de activación. Cada función de activación proporciona unas características al entrenamiento de la red. Como se vio en el apartado 2.2.1.5, la función de activación tiene mucho que ver con el entrenamiento.

La función de activación sigmoidal (logística), se satura con facilidad, pues esta definida de forma que, salvo en la fase de crecimiento, sus valores sean 0 o 1. Analizando la ecuación 2.5 se puede ver que una activación logística muy probablemente provocará o ningún cambio en el peso (valor 0), o el cambio máximo posible. Esto repercute en una menor conservación de la información, ya que muchos gradientes pasan a ser cero. Puede ser interesante para una red neuronal aprender a hacer cero pesos de neuronas que aportan poca información o información poco representativa, como picos inusuales en la señal, aunque puede ser bastante perjudicial para el aprendizaje mediante el descenso de gradiente.

La función de activación tangente hiperbólica es similar a la sigmoidal, con la diferencia de que satura en valores -1 o 1. Esto implica que en la actualización un peso se actualice lo máximo posible con diferentes signos, es decir, permite que la actualización no solo pueda decrementar el valor del peso sino que pueda aumentarlo. En este sentido y en el de que su valor en $x = 0$ es 0, es una función que se asemeja más a la identidad, mejorando la velocidad de entrenamiento por asemejarse a la función de regresión. Por estos motivos, también resulta interesante para este problema dado que se trata de una señal que tiene valores positivos y negativos.

Otro tipo de activaciones con uso creciente en los perceptrones multicapa son las lineales y sus generalizaciones. Ya se explicó en el apartado 2.2.1.2 que una lineal haría una simple regresión. Sin embargo, sus modificaciones ReLU y LReLU favorecen el entrenamiento de la red porque siempre que una neurona esté activa ayudan a mantener el gradiente en valores grandes, sin saturarlo como hacían la sigmoidal o la tangente hiperbólica. Es decir, con estas activaciones se gana algo de libertad en la actualización de pesos. Si bien es cierto que la activación ReLU perjudica al aprendizaje mediante gradiente ya que en la mitad de su dominio su valor es cero, la modificación de esta con LReLU ayudaría a solucionar este problema.

Se probaron estos cuatro tipos de funciones de activación, usando la misma para todas las capas ocultas. En el caso de la LReLU, se usó un $\alpha = 0.01$.

El número de capas ocultas y de neuronas por capa se varió con potencias de dos, lo cual es una aproximación extendida que es cierto que no tiene un soporte teórico establecido, pero podría ayudar a la complejidad de los cómputos por facilitar el trabajo a la memoria. Se probaron perceptrones de 2 y 4 capas porque más podrían provocar el ya comentado *gradient vanishing*. Se usaron entre 64 y 128 neuronas por capa, en orden ascendente, descendente y constante.

Una arquitectura en orden descendente (primera capa 128 neuronas, segunda capa 64, etc.) puede interpretarse como una compresión de la información que se tiene, mientras que una arquitectura en orden ascendente podría interpretarse como una extracción de información de los datos.

4.3.4. Bloques residuales

Se construyeron redes neuronales plenamente conectadas con bloques residuales cada dos capas. En este caso, se varió la función de activación con las mismas que en los perceptrones. También se probaron redes con 8 y 16 capas, todas con 64 neuronas por capa.

4.3.5. LSTM

Por el mismo motivo que en el perceptrón multicapa, la salida de la red se implementó mediante función logística y la función de coste a minimizar será la entropía cruzada. Ya se ha explicado cual es el funcionamiento de una celda LSTM, sobre la cual no se han realizado modificaciones. Se ha implementado una sóla capa LSTM con diferentes números de celdas LSTM.

El número de celdas de una capa LSTM no es sencillo de decidir. Un valor adecuado estaría

relacionado con la periodicidad de la señal, si ésta fuera única. Sin embargo, se tienen señales con periodos muy diferentes, por lo que no se tiene un número de celdas adecuado para todas. Se probaron números de celda potencias de dos entre 2 y 32.

Para esta red no se usó la transformada de Fourier ya que no es una serie temporal, sino una función de la frecuencia.

Este tipo de redes son mucho más costosas computacionalmente que los perceptrones, ya que requieren procesar cada momento de tiempo de cada una de las entradas. Esto significa que evalúa cada una de los 3198 momentos temporales para las 5087 estrellas, lo cual provoca un tiempo de cómputo desmesurado. Para solucionar este problema se usaron diferentes tamaños de *batch* (potencias de dos entre 32 y 128).

El *batch* es el número de observaciones o instancias que van a pasarse en cada iteración a la red. Esto provoca estimadores del gradiente más precisos ya que se calcula a partir de varias muestras y no individualmente, algo que también ayuda a corregir el *gradient vanishing* por hacer que se encadenen menos derivadas. Sin embargo, es importante comprender que por el hecho de calcular el gradiente para varias muestras, valores muy grandes de *batch* provocarán "saltos" muy grandes en el recorrido de la función a minimizar en el proceso del descenso de gradiente (ver secciones 2.2.1.3 y 2.2.1.4) en lugar de realizarlo progresivamente, por lo que puede hacer que el descenso no alcance mínimos aceptables [30].

Debido a que en este caso se tienen series temporales, en un *batch* irán los momentos de tiempo de varias series. Para realizar un entrenamiento correcto y estimar siempre los gradientes en base a las mismas muestras, hubo que garantizar que en cada *batch* siempre hubiera los momentos de tiempo de las mismas series temporales.

Se utilizó un tipo específico de red LSTM, las *many-to-one* para la clasificación. Estas redes toman como entrada un vector de una secuencia y dan como salida un sólo valor, típicamente una clasificación de la entrada [31].

4.3.6. Mapas auto-organizados

Se han implementado dos variantes de un mapa auto-organizado con diferentes funciones de vecindad, en concreto, las presentadas en [32] . En lugar de utilizar un mapa bidimensional plano, se ha utilizado uno de forma toroidal, lo cual amplia mucho la potencia del SOM ya que se permite que las neuronas de los bordes de la rejilla también tengan vecinos. Se utilizó el etiquetado de

neuronas explicado en la sección 2.2.4.

La primera variante del SOM se implementó haciendo uso de una vecindad rectangular delimitada por un radio. De este modo solo se actualizarán las neuronas dentro de un radio rectangular alrededor de la neurona ganadora, que es decreciente con las épocas (ver ecuación 4.6). Se utilizó un decrecimiento exponencial del radio.

$$r(t) = r_0 \exp\left(-\frac{t}{T}\right), \tag{4.6}$$

siendo $r(t)$ el radio en cada iteración, r_0 el radio inicial fijado al tamaño del mapa, t la iteración en la que se encuentra el algoritmo y T el número total de épocas. En esta implementación se ha utilizado un factor de aprendizaje ($\alpha(t)$ en la ecuación 2.18) que decrece con las iteraciones:

$$\alpha(t) = \frac{\alpha_I}{1 + \frac{t}{T}}, \tag{4.7}$$

siendo α_I un valor fijo que determina el factor de aprendizaje inicial, que se fijó en 1.

En la segunda variante implementada se otro modo de determinar la vecindad. En este caso se actualizaban los pesos de todas las neuronas, pero en función a un factor de vecindad: cuanto más alejada una neurona de la ganadora, menor era el impacto en la actualización de sus pesos. Para ello, además de un factor de aprendizaje fijado en 0.1 (lr), se utilizó una función gaussiana de la distancia entre las neuronas. Sea n_w la neurona ganadora y n_c una neurona cuyo peso se va a actualizar:

$$\alpha(t) = lr * exp\left(\frac{d(n_w, n_c)}{\theta(t)}\right) \tag{4.8}$$

$$\theta(t) = \Theta + \Theta\frac{t}{T}, \tag{4.9}$$

siendo Θ un factor de vecindad inicial, que se fijó en 1. La distancia entre dos neuronas $d(n_w, n_c)$ se calculó en función de las neuronas que ambas para para un mapa toroidal:

$$d(n_w, n_c) = 2\left(\frac{diff_i}{dim_i} + \frac{diff_j}{dim_j}\right), \tag{4.10}$$

con $diff_i$ y dim_i las posiciones de diferencia y la dimensión del mapa en filas, y $diff_j$ y dim_j sus respectivos valores para las columnas.

Este etiquetado de neuronas tiene un inconveniente, y es que tras el entrenamiento, puede ocurrir que algunas neuronas no tengan ninguna etiqueta asignada, por lo que al realizar la predicción de una observación podría no haber ninguna clase que predecir. Las observaciones en las que ocurrió esto no se consideraron para el cálculo de medidas de rendimiento.

Finalmente, se probaron ambas aproximaciones con tamaños de mapa de 8*8 neuronas y de 16*16 neuronas.

Capítulo 5

Resultados

5.1. Perceptrón multicapa

5.1.1. Dos capas ocultas

En la tabla 5.1 se encuentran los resultados obtenidos para cada clasificador y cada tratamiento de los datos en un perceptrón con dos capas ocultas y 128 y 64 neuronas por capa, respectivamente. Es notable que el sub-muestreo produjo resultados no lejanos de 0.5 para todas las medidas, lo cual indica que las señales de estrellas con exoplaneta no difieren de las que no lo tienen. Sin embargo, sí parecen diferenciarse en las frecuencias de las oscilaciones, ya que los mejores valores para todas las medidas del rendimiento se obtuvieron para las transformadas de Fourier.

Los resultados para el perceptrón con dos capas ocultas y 64 neuronas en cada capa se pueden ver en la tabla 5.2. De nuevo, la transformada de Fourier proporcionó los mejores resultados salvo en el AUC sobre el conjunto de prueba, en el que los mejores resultados se obtuvieron con el tratamiento SMOTE. En este caso se tiene un AUC de 0.77, que no es muy distinto de su valor para la transformada de Fourier, siendo 0.74. Ambos valores se obtuvieron para activación tangente hiperbólica, la cual también produjo el mejor valor de *Score*, con 0.6.

Para terminar con los MLP de dos capas ocultas, en la tabla 5.3 se muestran los resultados para las neuronas en orden creciente (64-128). De nuevo sucede que la transformada de Fourier ayudó a los clasificadores para todas las medidas de rendimiento.

Neuronas capa	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
128-64	Sigmoidal	-	0.99	0.99	0.84	0.5	0.9	0.54
		Pesos	0.99	0.98	0.84	0.49	0.91	0.7
		SMOTE	0.99	0.98	0.83	0.49	0.89	0.69
		FT	0.99	0.99	0.84	0.49	0.9	**0.88**
		Sub-muestreo	0.57	0.61	0.41	0.43	0.83	0.78
	TanH	-	0.99	0.99	0.81	0.5	0.87	0.87
		Pesos	0.99	0.98	0.83	0.48	0.91	0.79
		SMOTE	0.99	0.98	0.83	0.49	0.89	0.67
		FT	0.99	0.99	0.54	0.5	**0.93**	0.76
		Sub-muestreo	0.57	0.56	0.42	0.38	0.86	0.64
	ReLU	-	0.99	0.99	0.69	0.49	0.83	0.59
		Pesos	0.99	0.97	0.83	0.48	0.85	0.49
		SMOTE	0.99	0.99	0.85	0.5	0.85	0.5
		FT	0.99	0.98	0.87	**0.59**	0.88	0.59
		Sub-muestreo	0.59	0.62	0.44	0.38	0.76	0.66
	LReLU	-	0.99	0.99	0.81	0.5	0.82	0.5
		Pesos	0.99	0.98	0.83	0.49	0.84	0.49
		SMOTE	0.99	0.98	0.83	0.49	0.83	0.49
		FT	0.99	0.98	**0.91**	0.49	0.9	0.5
		Sub-muestreo	0.42	0.44	0.27	0.27	0.68	0.61
Mejor resultado			0.99	0.99	0.91	0.59	0.93	0.88

Cuadro 5.1: Resultados para el perceptrón con 2 capas ocultas con 128 y 64 neuronas, respectivamente.

Neuronas capa	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
64-64	Sigmoidal	-	0.99	0.99	0.81	0.6	0.84	0.6
		Pesos	0.99	0.98	0.84	0.48	0.85	0.49
		SMOTE	0.99	0.99	0.85	0.49	0.85	0.5
		FT	0.99	0.98	**0.88**	0.59	0.89	0.59
		Sub-muestreo	0.65	0.61	0.5	0.37	0.8	0.65
	TanH	-	0.99	0.99	0.84	0.49	0.86	0.88
		Pesos	0.98	0.98	0.84	0.49	0.91	0.69
		SMOTE	0.99	0.99	0.84	**0.6**	0.85	**0.77**
		FT	0.99	0.99	0.85	0.49	**0.94**	0.74
		Sub-muestreo	0.59	0.61	0.44	0.43	0.81	0.74
	ReLU	-	0.99	0.98	0.81	0.49	0.84	0.5
		Pesos	0.98	0.96	0.85	0.5	0.86	0.49
		SMOTE	0.99	0.99	0.84	0.5	0.84	0.5
		FT	0.99	0.99	0.87	0.5	0.88	0.5
		Sub-muestreo	0.61	0.67	0.45	0.42	0.77	0.57
	LReLU	-	0.99	0.98	0.82	0.49	0.82	0.5
		Pesos	0.98	0.97	0.82	0.47	0.83	0.49
		SMOTE	0.99	0.98	0.83	0.49	0.84	0.5
		FT	0.99	0.98	**0.88**	**0.59**	0.89	0.59
		Sub-muestreo	0.57	0.59	0.41	0.41	0.75	0.69
Mejor resultado			0.99	0.99	0.88	0.59	0.94	0.77

Cuadro 5.2: Resultados para el perceptrón con 2 capas ocultas con 64 y 64 neuronas, respectivamente.

En general, se puede decir que la mayoría de *Scores* ganadores fueron para la activación *Leaky-ReLU* con los datos de la transformada de Fourier, sin aparentes diferencias entre arquitecturas de red (número de capas y neuronas). Sin embargo, ninguno de estos fue superior a 0.59, valor que se consiguió superar con la tangente hiperbólica y SMOTE en una ocasión. Obtener en todos los casos *Scores* inferiores o iguales que 0.6 puede indica que los clasificadores distan mucho de tener un buen balanceo entre tasas de verdaderos positivos y negativos, o sensibilidad y especificidad.

A modo de ejemplo, en la figura 5.1 se observan las curvas ROC para el clasificador con activación tangente hiperbólica, SMOTE y dos capas de 64 neuronas, para el conjunto de entrenamiento y el de prueba. Se muestra este por ser uno de los que mejores resultados obtuvieron, con *Scores* de 0.84 y 0.6, y AUCs de 0.85 y 0.77, respectivamente. Se ve que ambas curvas son lejanas de la diagonal en todos sus tramos, eso sumado a los valores de AUC dejan ver que para este modelo se tuvo una buena clasificación, lejos de ser aleatoria. Para este modelo se obtuvo una sensibilidad de 0.7 para el conjunto de entrenamiento, algo bastante esperanzador, pero una de sólo 0.2 sobre el conjunto de prueba.

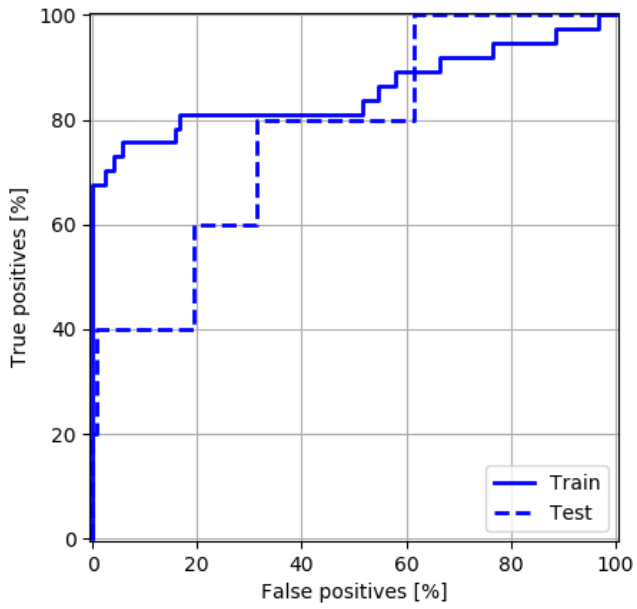

Figura 5.1: Curvas ROC para el clasificador con activación tangente hiperbólica, SMOTE y dos capas de 64 neuronas.

Neuronas capa	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
64-128	Sigmoidal	-	0.99	0.98	0.79	0.49	0.9	0.93
		Pesos	0.98	0.98	0.79	0.48	0.82	**0.79**
		SMOTE	0.99	0.98	0.83	0.49	0.89	0.84
		FT	0.99	0.98	0.84	0.49	0.91	0.46
		Sub-muestreo	0.55	0.61	0.39	0.43	0.76	0.73
	TanH	-	0.99	0.99	0.82	0.49	0.94	0.62
		Pesos	0.99	0.98	0.83	0.49	0.9	0.74
		SMOTE	0.99	0.98	0.83	0.49	0.86	0.67
		FT	0.99	0.99	0.84	0.5	**0.93**	0.75
		Sub-muestreo	0.62	0.68	0.46	0.44	0.79	0.51
	ReLU	-	0.99	0.99	0.81	0.49	0.84	0.5
		Pesos	0.99	0.97	0.84	0.48	0.85	0.49
		SMOTE	0.99	0.99	0.85	0.49	0.85	0.5
		FT	0.99	0.99	0.86	0.5	0.86	0.5
		Sub-muestreo	0.52	0.59	0.37	0.41	0.73	0.69
	LReLU	-	0.99	0.97	0.8	0.48	0.81	0.49
		Pesos	0.99	0.97	0.83	0.48	0.83	0.49
		SMOTE	0.99	0.99	0.83	0.49	0.84	0.5
		FT	0.99	0.98	**0.89**	**0.59**	0.89	0.59
		Sub-muestreo	0.69	0.77	0.52	0.45	0.77	0.53
Mejor resultado			0.99	0.99	0.89	0.59	0.93	0.73

Cuadro 5.3: Resultados para el perceptrón con 2 capas ocultas con 64 y 128 neuronas, respectivamente.

Neuronas capa	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
128-64-32-16	Sigmoidal	-	0.99	0.99	0.79	0.49	0.93	0.48
		Pesos	0.99	0.99	0.84	0.5	0.57	0.7
		SMOTE	0.99	0.98	0.83	0.49	**0.91**	0.57
		FT	0.99	0.98	0.85	0.49	0.85	0.57
		Sub-muestreo	0.44	0.46	0.27	0.28	0.77	0.78
	TanH	-	0.99	0.99	0.81	0.5	0.89	0.52
		Pesos	0.98	0.97	0.83	0.59	0.88	0.63
		SMOTE	0.99	0.98	0.83	**0.67**	0.88	**0.81**
		FT	0.99	0.99	0.83	0.5	0.88	0.71
		Sub-muestreo	0.58	0.59	0.4	0.48	0.82	0.81
	ReLU	-	0.99	0.99	0.5	0.5	0.5	0.5
		Pesos	0.99	0.99	0.49	0.49	0.5	0.5
		SMOTE	0.92	0.91	0.79	0.6	0.87	0.66
		FT	0.99	0.99	0.51	0.5	0.5	0.5
		Sub-muestreo	0.36	0.43	0.22	0.27	0.65	0.62
	LReLU	-	0.99	0.99	0.83	0.6	0.85	0.6
		Pesos	0.96	0.95	0.81	0.45	0.85	0.48
		SMOTE	0.99	0.99	0.83	0.49	0.83	0.48
		FT	0.99	0.97	**0.88**	0.57	0.89	0.59
		Sub-muestreo	0.62	0.74	0.46	0.49	0.46	0.65
Mejor resultado			0.99	0.99	0.88	0.59	0.91	0.81

Cuadro 5.4: Resultados para el perceptrón con 4 capas ocultas con 128, 64, 32 y 16 neuronas.

5.1.2. Cuatro capas ocultas

En la tabla 5.4 se muestran los resultados para un orden decreciente de neuronas. En esta ocasión se han obtenido mejores valores para el tratamiento con SMOTE, aunque sus puntuaciones no son muy lejanas de las obtenidas para las transformadas de Fourier. Destaca un valor de *Score* sobre el conjunto de prueba superior a cualquier otro modelo hasta el momento, de 0.67, obtenido con SMOTE para activación tangente hiperbólica, que además en vista de que el AUC sobre el mismo conjunto tuvo un valor de 0.81, se pudo concluir que se trataba de uno de los mejores modelos.

Para el perceptrón con capas de tamaño constante (64 neuronas) se obtuvieron los resultados de la tabla 5.5. En esta ocasión los ganadores están mucho más dispersos, habiéndose obtenido el mejor *Score* para el conjunto tras aplicar SMOTE con neuronas activadas por la función sigmoidal. Destaca un AUC de 0.94 para la misma activación sobre el conjunto sin tratamientos, aunque no

Neuronas capa	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
64-64-64-64	Sigmoidal	-	0.99	0.98	0.8	0.58	**0.94**	0.9
		Pesos	0.98	0.97	0.76	0.48	0.78	0.31
		SMOTE	0.99	0.99	0.84	**0.59**	0.91	**0.92**
		FT	0.99	0.99	0.84	0.5	0.88	0.45
		Sub-muestreo	0.56	0.59	0.39	0.41	0.74	0.74
	TanH	-	0.99	0.99	0.78	0.5	0.86	0.71
		Pesos	0.9	0.88	0.75	0.56	0.88	0.6
		SMOTE	0.99	0.99	0.83	0.49	0.82	0.78
		FT	0.99	0.98	0.84	0.49	0.86	0.62
		Sub-muestreo	0.53	0.59	0.35	0.41	0.72	0.71
	ReLU	-	0.99	0.99	0.51	0.49	0.84	0.67
		Pesos	0.99	0.98	0.83	**0.59**	0.83	0.6
		SMOTE	0.99	0.98	0.83	0.49	0.84	0.5
		FT	0.99	0.99	**0.86**	0.49	0.86	0.5
		Sub-muestreo	0.58	0.56	0.42	0.44	0.78	0.86
	LReLU	-	0.99	0.99	0.78	**0.59**	0.79	0.6
		Pesos	0.98	0.97	0.61	0.48	0.63	0.49
		SMOTE	0.99	0.98	0.83	0.49	0.83	0.49
		FT	0.99	0.98	0.85	0.49	0.85	0.5
		Sub-muestreo	0.26	0.29	0.16	0.16	0.72	0.66
Mejor resultado			0.99	0.99	0.86	0.59	0.94	0.92

Cuadro 5.5: Resultados para el perceptrón con 4 capas ocultas con 64 neuronas cada una.

es lejana del 0.91 obtenido con SMOTE con la misma activación.

En general, se han obtenido buenos resultados para los perceptrones con cuatro capas ocultas. Transformar los datos con la transformada de Fourier ofreció buenos resultados en general, aunque SMOTE fue ganador en más ocasiones. No se vió que alguna activación diese mejores resultados que otra, habiendose obtenido buenas medidas para la sigmoidal, la tangente hiperbólica y la *Leaky ReLU*. En concreto, parece que de nuevo la tangente hiperbólica junto con SMOTE obtuvo buenas marcas en todas medidas, siendo el mejor AUC el obtenido para arquitectura en orden decreciente.

Por último, el perceptrón con capas en tamaño creciente proveyó los resultados presentes en la tabla 5.6. En ella se puede observar que el mejor *Score* y AUC sobre el conjunto de entrenamiento se obtuvieron, de nuevo, para las transformadas de Fourier y activaciones *LeakyReLU*. En el conjunto de prueba funcionaron mejor otros métodos, pero con diferencias no muy notables.

En la figura 5.2 se muestran las curvas ROC obtenidas con la arquitectura de tamaño de capa descendente para la activación tangente hiperbólica de dos tipos de tratamiento de datos. Se observa como son levemente mejores las curvas para el tratamiento con SMOTE, pero no difieren significativamente de la transformada de Fourier. Para el modelo con SMOTE se obtuvo una sensibilidad de 0.7 y de 0.2 sobre los conjuntos de entrenamiento y prueba.

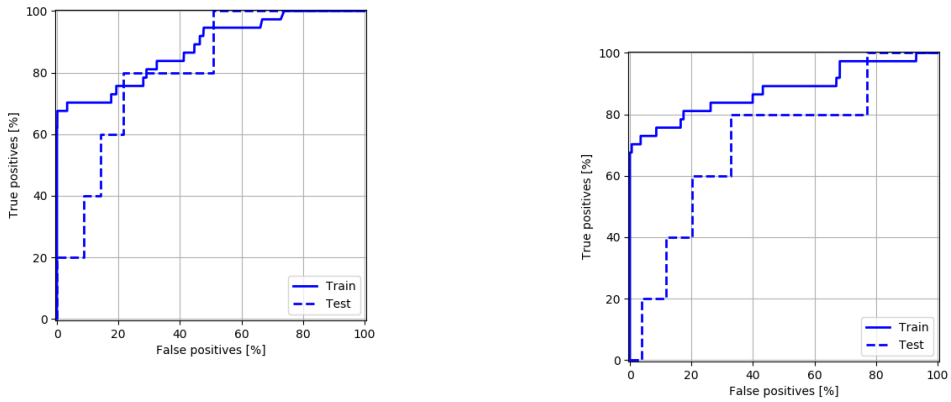

Figura 5.2: Izquierda: tratamiento con transformada de Fourier. Derecha: tratamiento con SMOTE.

5.2. Bloques residuales

La primera red con bloques residuales proporcionó los resultados de la tabla 5.7. De manera similar a los perceptrones multicapa clásicos, el mejor comportamiento sobre el conjunto de entrenamiento se obtuvo para los datos de la transformada de Fourier, aunque también para el conjunto de prueba estuvieron cerca de ser los ganadores. SMOTE parece tener buen comportamiento con la activación tangente hiperbólica. Sin embargo, con notable diferencia, el mejor valor de *Score* sobre el conjunto de test se obtuvo con ponderación de clases y activación sigmoidal, combinación que también obtuvo buenos valores de *Score* y AUC para el entrenamiento.

En la tabla 5.8 se muestran los resultados de la red con bloques residuales de 16 capas. La transformada de Fourier con activación *LeakyReLU* proporcionó el mejor *Score* sobre el conjunto de entrenamiento, además de uno aceptable de 0.57 sobre el conjunto de prueba, aunque con un AUC de 0.59 sobre el de prueba. Esa misma transformación con activación sigmoidal proporcionó buenos resultados salvo para el *Score* sobre el conjunto de prueba, aunque con un valor de AUC de 0.87 puede no darse demasiada importancia a dicho valor.

Neuronas capa	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
16-32-64-128	Sigmoidal	-	0.99	0.98	0.51	0.49	0.52	0.43
		Pesos	0.55	0.54	0.38	0.25	0.76	0.46
		SMOTE	0.98	0.98	0.82	0.49	0.87	0.68
		FT	0.99	0.98	0.83	0.49	0.85	0.56
		Sub-muestreo	0.43	0.47	0.28	0.25	0.66	0.6
	TanH	-	0.99	0.99	0.63	0.49	0.89	0.41
		Pesos	0.95	0.95	0.76	0.46	0.89	0.7
		SMOTE	0.99	0.98	0.83	**0.59**	0.86	0.66
		FT	0.99	0.98	0.81	0.49	0.53	0.5
		Sub-muestreo	0.56	0.61	0.4	0.37	0.74	**0.77**
	ReLU	-	0.99	0.99	0.5	0.5	0.5	0.5
		Pesos	0.98	0.96	0.81	0.46	0.8	0.4
		SMOTE	0.99	0.99	0.83	0.5	0.84	0.5
		FT	0.99	0.99	0.86	0.5	0.88	0.5
		Sub-muestreo	0.71	0.73	0.55	0.48	0.8	0.59
	LReLU	-	0.98	0.98	0.73	0.58	0.84	0.67
		Pesos	0.98	0.97	0.82	0.47	0.84	0.49
		SMOTE	0.99	0.98	0.83	0.49	0.87	0.5
		FT	0.99	0.98	**0.88**	0.58	**0.89**	0.59
		Sub-muestreo	0.67	0.68	0.53	0.51	0.81	0.7
Mejor resultado			0.99	0.99	0.88	0.59	0.89	0.77

Cuadro 5.6: Resultados para el perceptrón con 4 capas ocultas con 16, 32, 64, y 128 neuronas.

Capas x Neuronas	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
8x64	Sigmoidal	-	0.99	0.99	0.71	0.5	0.89	0.66
		Pesos	0.98	0.97	0.81	0.57	0.87	**0.82**
		SMOTE	0.99	0.99	0.83	0.5	**0.92**	0.74
		FT	0.99	0.98	0.84	0.49	0.91	0.6
		Sub-muestreo	0.58	0.59	0.42	0.41	0.76	0.81
	TanH	-	0.99	0.98	0.81	0.49	0.88	0.45
		Pesos	0.99	0.98	0.83	0.58	0.87	0.64
		SMOTE	0.99	0.99	0.84	**0.59**	0.84	0.68
		FT	0.99	0.99	0.84	0.5	0.84	0.61
		Sub-muestreo	0.58	0.56	0.41	0.27	0.82	0.56
	ReLU	-	0.99	0.99	0.67	0.5	0.68	0.5
		Pesos	0.96	0.95	0.84	0.46	0.88	0.48
		SMOTE	0.98	0.98	0.81	0.49	0.83	0.49
		FT	0.99	0.99	0.87	0.5	0.88	0.5
		Sub-muestreo	0.47	0.39	0.29	0.27	0.66	0.74
	LReLU	-	0.99	0.98	0.85	0.49	0.85	0.5
		Pesos	0.99	0.97	0.8	0.48	0.81	0.49
		SMOTE	0.99	0.99	0.83	0.49	0.84	0.5
		FT	0.99	0.98	**0.88**	0.58	0.89	0.59
		Sub-muestreo	0.37	0.35	0.23	0.24	0.67	0.73
Mejor resultado			0.99	0.99	0.88	0.59	0.89	0.82

Cuadro 5.7: Resultados para la red con bloques residuales con 8 capas de 64 neuronas.

Capas x Neuronas	**Activación ocultas**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
16x64	Sigmoidal	-	0.99	0.99	0.78	0.49	0.91	0.69
		Pesos	0.76	0.76	0.62	0.44	**0.92**	0.65
		SMOTE	0.99	0.98	0.83	0.49	0.89	0.8
		FT	0.99	0.99	0.83	0.5	0.89	**0.87**
		Sub-muestreo	0.51	0.62	0.34	0.44	0.7	0.81
	TanH	-	0.99	0.99	0.63	0.5	0.9	0.63
		Pesos	0.87	0.86	0.72	0.45	0.85	0.55
		SMOTE	0.99	0.98	0.81	0.49	0.9	0.58
		FT	0.99	0.98	0.85	0.49	0.86	0.43
		Sub-muestreo	0.59	0.6	0.4	0.42	0.78	0.73
	ReLU	-	0.99	0.99	0.49	0.49	0.5	0.5
		Pesos	0.99	0.99	0.62	0.5	0.62	0.5
		SMOTE	0.99	0.99	0.85	0.49	0.85	0.5
		FT	0.99	0.99	0.85	**0.59**	0.85	0.72
		Sub-muestreo	0.73	0.76	0.56	0.52	0.77	0.62
	LReLU	-	0.99	0.98	0.88	0.49	0.88	0.5
		Pesos	0.98	0.96	0.79	0.46	0.8	0.48
		SMOTE	0.99	0.98	0.83	0.49	0.83	0.49
		FT	0.99	0.97	**0.89**	0.57	0.9	0.59
		Sub-muestreo	0.56	0.52	0.41	0.37	0.74	0.68
Mejor resultado			0.99	0.99	0.89	0.59	0.92	0.87

Cuadro 5.8: Resultados para la red con bloques residuales con 16 capas de 64 neuronas.

En la figura 5.3 se puede ver las curvas ROC para la red de 8 capas con residuales, activación sigmoidal y ponderación de clases, que no fue el mejor modelo en AUC, pero que sí tuvo uno de los mejores *Scores* para estas redes. Para este modelo, se obtuvo también una sensibilidad de 0.2 en el conjunto de prueba, y de 0.68 en el de entrenamiento.

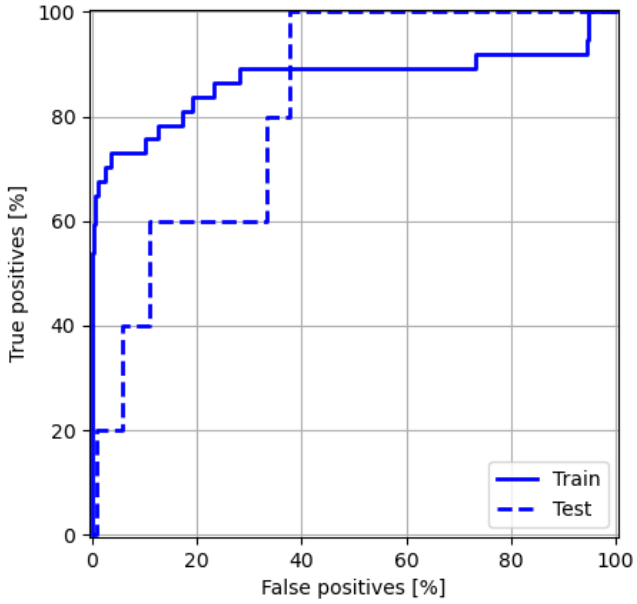

Figura 5.3: Curvas ROC para el red de ocho capas de 64 neuronas con residuales, activación sigmoidal y ponderación de clases.

5.3. LSTM

La primera implementación de la LSTM se hizo con sólo dos celdas. Los resultados obtenidos se muestran en la tabla 5.9. En general funcionó mejor la técnica SMOTE que las demás, y no hay diferencias notables entre tamaños de *batch*. En ninguna ocasión se logró un *Score* superior a 0.5 en el conjunto de prueba, y el mejor valor de AUC sobre dicho conjunto fue de 0.76, resultados que indican que la clasificación fue cercana a aleatoria.

Aumentar a cuatro celdas no produjo mejoras muy grandes pero sí fueron notables. En el caso de SMOTE con *batch* de tamaño 64 se obtuvo un *Score* de 0.58 para el conjunto de prueba. En

Nº celdas	**Tamaño Batch**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
2	32	-	0.99	0.99	0.5	0.5	0.61	0.57
		Pesos	0.75	0.74	0.46	0.38	0.72	0.6
		SMOTE	0.99	0.99	0.5	0.49	0.66	0.44
		Sub-muestreo	0.44	0.35	0.22	0.16	0.53	0.61
	64	-	0.99	0.99	0.5	0.5	0.6	0.58
		Pesos	0.67	0.73	0.41	0.42	0.66	0.48
		SMOTE	0.96	0.97	**0.54**	0.47	**0.73**	0.57
		Sub-muestreo	0.56	0.28	0.15	0.15	0.52	0.56
	128	-	0.99	0.99	0.5	0.5	0.7	0.59
		Pesos	0.37	0.4	0.2	0.2	0.63	0.49
		SMOTE	0.8	0.99	0.4	0.5	0.68	**0.76**
		Sub-muestreo	0.64	0.66	0.41	0.29	0.62	0.43
Mejor resultado			0.99	0.99	0.54	0.5	0.73	0.76

Cuadro 5.9: Resultados para la red LSTM con dos celdas.

general, se han obtenido mejores valores de AUC que con sólo dos celdas LSTM. Los resultados están en la tabla 5.10.

La LSTM de 8 celdas tampoco produjo resultados muy esperanzadores (ver tabla 5.11). Ningún *Score* sobre el conjunto de prueba fue superior a 0.5, aunque se mejoró sobre el conjunto de entrenamiento en dicha medida al igual que en el AUC, llegándose por primera vez en las LSTM a 0.9. En resumen, no se puede establecer diferencias muy claras entre tamaños de *batch* o tratamientos de datos.

Incrementar a 16 celdas sí que proporciono mejores resultados. Se alcanzó un *Score* de 0.79 para el conjunto de entrenamiento y de 0.52 para el de prueba, que, aunque fue para diferentes tamaños de *batch*, ambas fueron para un tratamiento de datos con pesos de clase. Los mejores valores de AUC también se obtuvieron para tratamiento con ponderación de clases. En dos casos el *batch* de tamaño 64 tuvo las mejores marcas, pero con el batch de tamaño 128 se obtuvo el mejor valor de *Score* sobre el conjunto de prueba. Se pueden ver estos datos en la tabla 5.12.

Por último y como se ve en la tabla 5.13, la red LSTM con 32 celdas proporcionó en general buenas puntuaciones. El tratamiento de ponderar las clases y *batch* de 64 obtuvo un buen rendimiento con *Score* de 0.89 y AUC de 0.99 sobre el conjunto de entrenamiento, manteniendo un buen valor de AUC para el de prueba, con 0.75. No obstante, no existió ningún modelo que consiguiese

Nº celdas	**Tamaño Batch**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
4	32	-	0.99	0.99	0.5	0.49	0.7	0.65
		Pesos	0.52	0.55	0.37	0.26	**0.8**	0.6
		SMOTE	0.8	0.99	0.42	0.49	0.77	0.73
		Sub-muestreo	0.46	0.59	0.21	0.35	0.53	0.7
	64	-	0.99	0.99	0.5	0.49	0.69	0.76
		Pesos	0.59	0.66	0.39	0.41	0.79	0.68
		SMOTE	0.82	0.97	0.49	**0.58**	0.76	0.68
		Sub-muestreo	0.56	0.26	0.31	0.14	0.69	0.51
	128	-	0.99	0.99	0.5	0.49	0.68	**0.79**
		Pesos	0.74	0.77	**0.53**	0.46	0.79	0.71
		SMOTE	0.8	0.99	0.4	0.5	0.75	0.7
		Sub-muestreo	0.54	0.22	0.3	0.15	0.68	0.4
Mejor resultado			0.99	0.99	0.53	0.58	0.8	0.79

Cuadro 5.10: Resultados para la red LSTM con cuatro celdas.

Nº celdas	**Tamaño Batch**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
8	32	-	0.99	0.99	0.5	0.49	0.78	0.71
		Pesos	0.71	0.73	0.56	0.34	0.86	0.62
		SMOTE	0.83	0.96	0.52	0.47	0.84	0.71
		Sub-muestreo	0.56	0.63	0.31	0.32	0.63	0.59
	64	-	0.99	0.99	0.5	0.49	0.75	**0.76**
		Pesos	0.63	0.7	**0.59**	0.46	0.83	0.73
		SMOTE	0.84	0.97	0.52	0.47	0.86	0.57
		Sub-muestreo	0.54	0.47	0.3	0.25	0.55	0.53
	128	-	0.99	0.99	0.5	0.49	0.69	0.72
		Pesos	0.73	0.78	0.57	0.38	0.86	0.65
		SMOTE	0.84	0.97	0.54	0.48	**0.9**	0.67
		Sub-muestreo	0.7	0.56	0.49	0.31	0.73	0.39
Mejor resultado			0.99	0.99	0.59	0.49	0.9	0.76

Cuadro 5.11: Resultados para la red LSTM con ocho celdas.

Nº celdas	**Tamaño Batch**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
16	32	-	0.99	0.99	0.5	0.49	0.89	0.72
		Pesos	0.78	0.83	0.66	0.51	0.91	**0.79**
		SMOTE	0.9	0.95	0.75	0.45	0.95	0.68
		Sub-muestreo	0.76	0.83	0.58	0.43	0.86	0.55
	64	-	0.99	0.99	0.5	0.49	0.87	0.72
		Pesos	0.86	0.86	**0.79**	0.47	**0.97**	0.58
		SMOTE	0.87	0.98	0.63	0.49	0.92	0.65
		Sub-muestreo	0.64	0.51	0.4	0.29	0.7	0.61
	128	-	0.99	0.99	0.5	0.49	0.74	0.77
		Pesos	0.82	0.84	0.75	**0.52**	0.96	0.73
		SMOTE	0.86	0.97	0.6	0.48	0.89	0.69
		Sub-muestreo	0.66	0.47	0.44	0.35	0.74	0.67
Mejor resultado			0.99	0.99	0.79	0.52	0.97	0.79

Cuadro 5.12: Resultados para la red LSTM con 16 celdas.

buenos resultados en todas las medidas.

En general, los resultados para las redes LSTM fueron mejorando conforme se aumentaba el número de celdas. En casi todas las ocasiones, el *batch* de tamaño 64 obtuvo al menos dos marcas ganadoras, pero el de tamaño 32 fue mejor porque consiguió buenos resultados en todas las marcas. La utilización de pesos para las instancias en función de su clase resultó en mejor rendimiento que otras aproximaciones. En la figura 5.4 se muestran las curvas ROC para el modelo con 16 celdas, *batch* de tamaño 32 y ponderación de instancias, que pese a no haber conseguido los mejores valores de precisión, sus valores de AUC y *Score* indicaron que fue un buen clasificador (mejor que los demás) sobre el conjunto de prueba. Este modelo consiguió una sensibilidad de 0.73 y de 0.2 sobre los conjuntos de entrenamiento y prueba.

5.4. Mapas auto-organizados

Para la primera variante del SOM con vecindad rectangular se obtuvieron los resultados de la tabla 5.14. Como se ve en esta, la clasificación no es muy adecuada. Rara vez se obtuvo una curva ROC con AUC superior a 0.5, lo que indica que las clasificaciones fueron prácticamente aleatorias, algo que se confirma con los bajos valores de *Score* y con la precisión baja para el submuestreo.

Nº celdas	**Tamaño Batch**	**Tratamiento Datos**	**Precisión Train**	**Precisión Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC Test**
32	32	-	0.99	0.99	0.68	0.5	0.98	0.73
		Pesos	0.93	0.91	0.88	0.41	0.99	0.73
		SMOTE	0.84	0.96	0.55	**0.56**	0.88	0.54
		Sub-muestreo	0.73	0.45	0.53	0.194	0.75	0.47
	64	-	0.99	0.99	0.54	0.5	0.96	0.65
		Pesos	0.92	0.9	**0.89**	0.41	**0.99**	0.75
		SMOTE	0.93	0.95	0.79	0.46	0.97	0.7
		Sub-muestreo	0.69	0.45	0.48	0.28	0.81	0.68
	128	-	0.99	0.99	0.5	0.49	0.79	0.7
		Pesos	0.9	0.88	0.73	0.4	0.85	0.71
		SMOTE	0.94	0.96	0.83	0.47	0.98	0.74
		Sub-muestreo	0.95	0.98	0.86	0.48	0.98	**0.81**
Mejor resultado			0.99	0.99	0.89	0.56	0.99	0.81

Cuadro 5.13: Resultados para la red LSTM con 32 celdas.

Tamaño mapa	**Tratamiento Datos**	**Precisión Train**	**Precision Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC test**
8x8	-	0.98	0.98	0.48	0.49	0.49	0.49
	SMOTE	0.79	0.98	0.39	0.49	0.49	0.49
	FT	0.99	0.98	0.49	0.49	0.5	0.5
	Sub-muestreo	0.83	0.97	0.28	0.49	**0.53**	0.5
16x16	-	0.99	0.99	**0.5**	0.49	0.5	0.5
	SMOTE	0.8	0.99	0.4	**0.5**	0.5	0.5
	FT	0.99	0.99	**0.5**	0.49	0.5	0.5
	Sub-muestreo	0.51	0.99	0.26	0.49	0.51	0.5
Mejor resultado		0.99	0.99	0.52	0.57	0.55	0.59

Cuadro 5.14: Resultados para el SOM de vecindad rectangular.

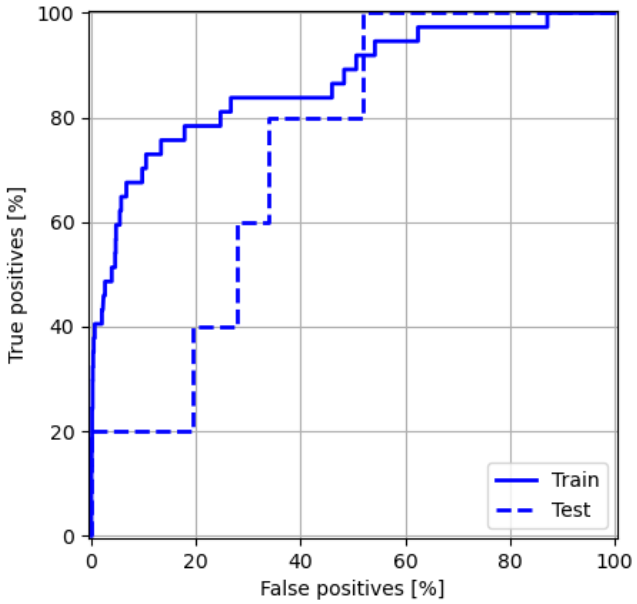

Figura 5.4: Curvas ROC para la red LSTM de 16 celdas, *batch* tamaño 32 y ponderación de instancias.

La segunda aproximación, con vecindad gaussiana, pareció conseguir mejores clasificaciones aunque no con mucha diferencia. Se obtuvieron buenos resultados para *Score* en el conjunto de prueba y los mejores valores de AUC para el tratamiento de datos con sub-muestreo. Sin embargo, la precisión y el *Score* sobre el conjunto de entrenamiento fueron de 0.55 y 0.31, respectivamente, lo cual nos indica que nada asegura que esos resultados no fueran fruto de clasificaciones aleatorias. Estos resultados están visibles en la tabla 5.15. En la figura 5.5 se puede ver cómo las curvas ROC en este caso son muy cercanas a la diagonal.

5.5. Comparación con trabajos previos

Previo a este trabajo ya existían varias aproximaciones al problema de predecir las estrellas con exoplanetas con el mismo conjunto de datos, todas ellas accesibles en la página web de *Kaggle* [19]. En este trabajo, se compararon los resultados obtenidos con aquellos proporcionados mediante algunas de dichas aproximaciones.

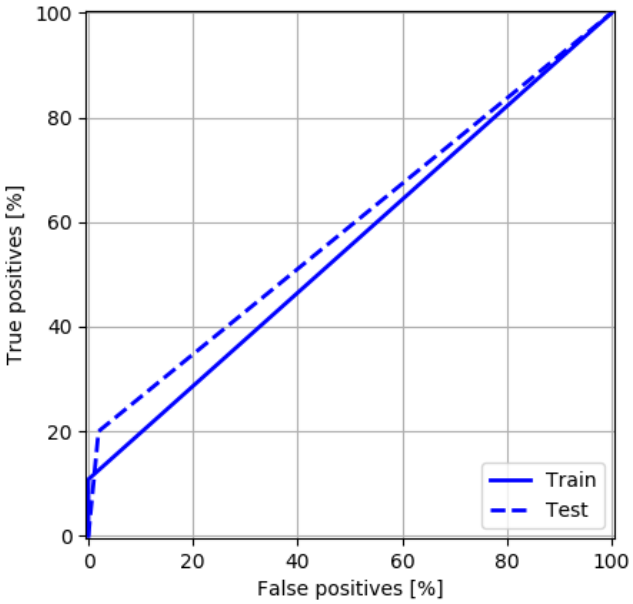

Figura 5.5: Curvas ROC para el SOM de tamaño 8x8, vecindad gaussiana y sub-muestreo.

De soluciones disponibles sólo se contrastaron aquellas que realizaban alguna medida con respecto al desbalance en los datos o que contaban con esto a la hora de obtener resultados, ya que, de otra manera, resultaba muy complejo (en términos de precisión) comparar dos modelos. Muchas de las soluciones presentes en [19] no tomaban medidas necesarias con respecto a este problema.

La primera solución propuesta es la presente en [33]. Pese a que esta solución no contempló el desbalance de los datos y sólo utiliza como medida la precisión de los modelos, proporciona las matrices de confusión con las que se pudieron calcular medidas como la sensibilidad. El primer clasificador es un árbol de decisión, con el que obtuvo una precisión de 0.98 y una sensibilidad de 0.2, indicando que la clasificación no era para nada buena. El segundo modelo, un *random forest* (método de ensamblado de árboles de decisión), obtuvo precisión de 0.99, con sensibilidad de 0.4, mejor que el anterior pero aún bastante inadecuada. El último modelo, *XGBoost* (otro tipo de ensamblado de árboles) predijo todas las observaciones como negativas, dando una sensibilidad de cero.

Otra solución comparada es la presente en [34]. En esta, se prueban máquinas de vectores soporte (SVM) con diferentes kernels, obteniéndose para todas sensibilidad de cero. Posteriormente

Tamaño mapa	**Tratamiento Datos**	**Precisión Train**	**Precision Test**	**Score Train**	**Score Test**	**AUC Train**	**AUC test**
8x8	-	0.98	0.99	**0.52**	0.49	0.52	0.5
	SMOTE	0.79	0.98	0.39	0.48	0.5	0.49
	FT	0.99	0.99	0.49	0.5	0.5	0.5
	Sub-muestreo	0.55	0.97	0.31	**0.57**	**0.55**	**0.59**
16x16	-	0.98	0.99	0.5	0.49	0.51	0.5
	SMOTE	0.8	0.99	0.4	0.49	0.5	0.5
	FT	0.99	0.99	0.5	0.5	0.5	0.5
	Sub-muestreo	0.51	0.99	0.25	0.49	0.5	0.5
Mejor resultado		0.99	0.99	0.52	0.57	0.55	0.59

Cuadro 5.15: Resultados para el SOM de vecindad gaussiana.

se obtienen mejores valores de sensibilidad, pero a consta de predecir la mayoría como positivos.

Una tercera aproximación es la tomada en [35]. Aquí se normalizan los datos, se elimina la tendencia y se crean instancias con SMOTE. Finalmente se construye un modelo SVM de kernel lineal. Tras esto, se obtiene una sensibilidad de 0.61, pero con un poder predictivo positivo (PPV) de 0.1. El poder predictivo positivo es el porcentaje de positivos reales frente a todos los positivos predichos, por lo que un valor de 0.1 indica que se predijeron muchos más positivos de los reales.

Por último, se estudió la solución propuesta en [36]. En ella se usan redes neuronales con capas convolucionales 1D junto con capas recurrentes. Sin tratamiento de datos, obtiene una sensibilidad de 1 y un PPV de 0.83. Con diferencia, el mejor modelo de los existentes en *Kaggle* .

Que los modelos desarrollados en este trabajo no hayan alcanzado grandes valores de sensibilidad es algo desalentador pero no desesperanzador. Para muchos modelos probados en este trabajo se obtuvieron áreas bajo la curva ROC superiores a 0.8. Haber obtenido estos valores indica que existe algún umbral para clasificar que obtiene buenos valores de tasa de verdaderos positivos manteniendo baja la de falsos positivos, como se explicó en el apartado 4.3.2.1. Por ejemplo, retomando las curvas ROC para la red con residuales (figura 5.3), se observa cómo en el conjunto de prueba existe un umbral que produce una tasa de aciertos positivos del 100 %, manteniendo en menos de un 40 % la de falsos positivos. En la figura 5.6 se puede ver en rojo la observación que produjo este valor de umbral, desconocido, pero fácilmente recuperable observando las predicciones.

Desafortunadamente, los trabajos previos disponibles en [19] no daban información sobre el AUC

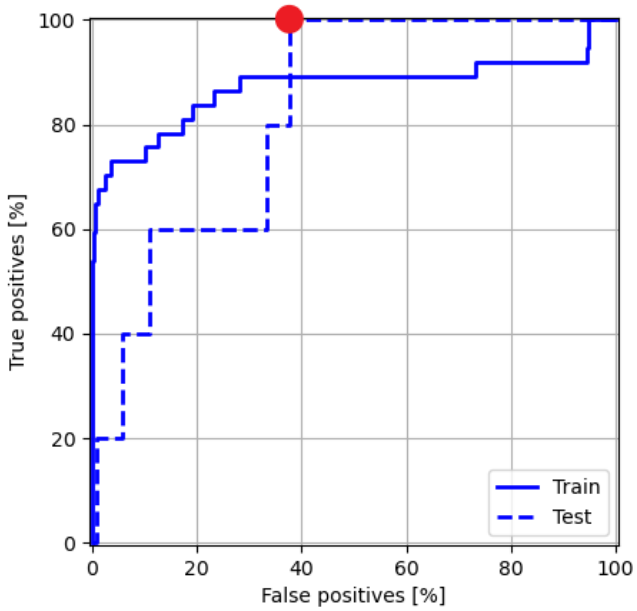

Figura 5.6: Curvas ROC para la red con residuales. En rojo se ve un posible candidato para fijar el umbral de la clasificación.

de los modelos y la comparación sólo se ha podido realizar mediante la sensibilidad. El problema de la sensibilidad es que en este estudio se ha utilizado el valor por defecto para una clasificación positiva: 0.5. Todas aquellas observaciones cuya probabilidad de ser positiva fuera superior a 0.5 fueron consideradas positivas. Sabemos gracias a las curvas ROC que utilizando otros valores de umbral, los clasificadores habrían sido más precisos.

Capítulo 6

Conclusiones

Para los modelos de perceptrón multicapa, la activación LReLU proporcionó en general mejores resultados que la ReLU. Lo mismo ocurrió entre la activación tangente hiperbólica y la sigmoidal, confirmando en ambos casos que estas funciones de activación ayudan a mitigar el efecto del *vanishing gradient*.

El mejor modelo se obtuvo con una arquitectura de 4 capas con 128, 64, 32 y 16 neuronas en cada una, activación tangente hiperbólica y generación de instancias con SMOTE. Dio AUC, *Score* y sensibilidad de 0.81, 0.59 y 0.2 sobre el conjunto de prueba. Pese a haberse usado SMOTE en el considerado como mejor modelo, en general, la transformada de Fourier proveyó de resultados similares y competentes con SMOTE.

Para las redes con residuales no hubo diferencias claras entre arquitecturas, pero se puede considerar mejor a la red de 8 capas, activación sigmoidal y tratamiento con ponderación de instancias. Esta combinación produjo AUC, *Score* y sensibilidad de 0.82, 0.57 y 0.2 sobre el conjunto de prueba.

Con las redes LSTM se obtuvieron los mejores resultados con 16 celdas. Ningún modelo consiguió destacar frente a los anteriores. El considerado como mejor, obtuvo un AUC, *Score* y sensibilidad de 0.79, 0.51 y 0.2 sobre el conjunto de prueba.

Los resultados de ambos SOM no fueron nada esperanzadores, en ningún caso se tuvieron resultados que no se asemejasen a una clasificación aleatoria.

Pese a no haberse conseguido mejorar la sensibilidad de los modelos propuestos con anterioridad

a este trabajo, haber obtenido estas sensibilidades bajas de 0.2 al mismo tiempo que AUCs de valores cercanos a 1, indica que la cota de clasificación como positivo o poseedor de exoplaneta no se encuentra en el 0.5, sino en valores superiores. Queda para trabajos futuros la identificación de las predicciones que provocan crecimiento en el AUC para su utilización en la definición de una cota adecuada para la clasificación como positivos.

Los datos de los que se dispone son difíciles de diferenciar. Es posible que se contenga estrellas que presentan tránsitos por motivos distintos a poseer un exoplaneta. Un ejemplo es el caso de las estrellas binarias: los registros de luz en la estrella de mayor luminosidad presentarán tránsitos, mientras que nunca dicha estrella será un positivo. Además, tampoco se sabe si estos datos han sido filtrados con el criterio de tener mínimo tres tránsitos para confirmar que existe un cuerpo orbitando. De no ser así, la clasificación se complicaría a un más, por lo que hay que tomar los resultados con cautela.

En vista de que SMOTE y la transformada de Fourier fueron los mejores tratamientos, se propone para trabajos futuros la prueba de modelos aplicando ambas medidas al mismo tiempo.

Bibliografía

[1] Wikipedia. (2020). Transit Method, dirección: https://en.wikipedia.org/wiki/Transit_(astronomy) (visitado 24-06-2020).

[2] NASA. (2018). Legacy of NASA's Kepler Space Telescope: More Planets Than Stars, dirección: https://www.youtube.com/watch?v=_V7J05fK5e0 (visitado 24-06-2020).

[3] R. Gilliland, W. Chaplin, E. Dunham, V. Argabright, W. Borucki, G. Basri, S. Bryson, D. Buzasi, D. Caldwell, Y. Elsworth, J. Jenkins, D. Koch, J. Kolodziejczak, A. Miglio, J. Cleve, L. Walkowicz y a. Welsh, "Kepler Mission Stellar and Instrument Noise Properties", *The Astrophysical Journal Supplement Series*, 2011. DOI: 10.1088/0067-0049/197/1/6.

[4] J. Lissauer, G. Marcy, S. Bryson, J. Rowe, D. Jontof-Hutter, E. Agol, W. Borucki, J. Carter, E. Ford, R. Gilliland, R. Kolbl, K. Star, J. Steffen y G. Torres, "Validation of Kepler's Multiple Planet Candidates. II: Refined Statistical Framework and Descriptions of Systems of Special Interest", *The Astrophysical Journal*, 2014. DOI: 10.1088/0004-637X/784/1/44.

[5] NASA. (2018). Brightness variation on binary stars, dirección: https://www.nasa.gov/mission_pages/kepler/kepler-news (visitado 24-06-2020).

[6] ——, (2018). NASA Exoplanet Archive, dirección: https://exoplanetarchive.ipac.caltech.edu/docs/counts_detail.html (visitado 12-07-2020).

[7] F. Chou, A. Hawkes y C. Cofield. (2020). NASA Retires Kepler Space Telescope, Passes Planet-Hunting Torch, dirección: https://www.nasa.gov/press-release/nasa-retires-kepler-space-telescope-passes-planet-hunting-torch (visitado 12-07-2020).

[8] S. Institute. (2017). Latest Exoplanet Results from NASA's Kepler/K2 Mission - Ian Crossfield (SETI Talks 2017), dirección: https://www.youtube.com/watch?v=qPL9cFg8e38 (visitado 12-07-2020).

[9] Wikipedia. (2020). Perceptron, dirección: https://en.wikipedia.org/wiki/Perceptron#History (visitado 11-07-2020).

[10] ——, (2020). Universal approximation theorem, dirección: https://en.wikipedia.org/wiki/Universal_approximation_theorem (visitado 24-06-2020).

[11] ——, (2020). Neurona de McCulloch-Pitts, dirección: https://es.wikipedia.org/wiki/Neurona_de_McCulloch-Pitts (visitado 11-07-2020).

[12] 3. (https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw). (2017). Retropropagación cálculo | Apéndice para el aprendizaje profundo capítulo 3, dirección: https://www.youtube.com/watch?v=tIeHLnjs5U8 (visitado 24-06-2020).

[13] H. Lee, R. Grosse, R. Ranganath y A. Y. Ng, "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations", *Proceedings of the 26th International Conference on Machine Learning*, 2009.

[14] K. He, X. Zhang, S. Ren y J. Sun, "Deep Residual Learning for Image Recognition", *arXiv.org*, 2015.

[15] R. Pascanu, T. Mikolov e Y. Bengio, "On the dificulty of training recurrent neural networks", *International Conference on Machine Learning*, 2013.

[16] C. B. (https://colah.github.io/). (2015). Understanding LSTM Networks, dirección: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visitado 24-06-2020).

[17] J. Bayer, "Learning Sequence Representations", *Technical University of Munich*, 2015.

[18] Wikipedia. (2020). Self-organizing map, dirección: https://en.wikipedia.org/wiki/Self-organizing_map (visitado 26-06-2020).

[19] K. (https://www.kaggle.com/keplersmachines). (2017). Exoplanet Hunting in Deep Space - Kernels, dirección: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data/kernels (visitado 14-07-2020).

[20] B. Hughes y M. Cotterell, *Software project management*. Mc Graw Hill Education, 2009, ISBN: 3 0-07-712279-8.

[21] K. (https://www.kaggle.com/keplersmachines). (2017). Exoplanet Hunting in Deep Space, dirección: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data (visitado 24-06-2020).

[22] NASA. (2020). Kepler and K2 data processing pipeline, dirección: https://keplerscience.arc.nasa.gov/pipeline.html (visitado 24-06-2020).

[23] I. H. Witten y E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, ép. The Morgan Kaufmann Series in Data Management Systems. Elsevier, 2005, ISBN: 0-12-088407-0.

[24] W. Klement, S. Wilk, W. Michaowski y S. Matwin, "Dealing with Severely Imbalanced Data", *Proc. of the PAKDD Conference*, 2009.

[25] A. Viloria-Lanero y V. Cardeñoso-Payo, "Integration of Skin Segmentation Methods using ANNs", *Computational Modelling of Objects Represented in Images Conference*, 2009.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall y W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research*, 2002.

[27] Google-Tensorflow. (2020). Classification on imbalanced data, dirección: `https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#class_weights` (visitado 04-06-2020).

[28] I. Goodfellow, Y. Bengio y A. Courville, "Deep Learning", en. MIT Press, 2016, cap. 6.2.2.2, `http://www.deeplearningbook.org`.

[29] Wikipedia. (2020). Cross entropy - Relation to log-likelihood, dirección: `https://en.wikipedia.org/wiki/Cross_entropy#Relation_to_log-likelihood` (visitado 11-06-2020).

[30] itdxer (https://stats.stackexchange.com/users/64943/itdxer). (2019). What is batch size in neural network?, dirección: `https://stats.stackexchange.com/q/153535` (visitado 05-07-2020).

[31] A. K. blog (http://karpathy.github.io/). (2015). The Unreasonable Effectiveness of Recurrent Neural Networks, dirección: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/` (visitado 16-07-2020).

[32] W. Natita, W. Wiboonsak y S. Dusadee, "Appropriate Learning Rate and Neighborhood Function of Self-organizing Map (SOM) for Specific Humidity Pattern Classification over Southern Thailand", *International Journal of Modeling and Optimization*, 2016.

[33] A. (https://www.kaggle.com/iabhishekmaurya). (2020). Well the Hunt Ends with 99.12 %, dirección: `https://www.kaggle.com/iabhishekmaurya/well-the-hunt-ends-with-99-12-without-nn` (visitado 14-07-2020).

[34] N. B. (https://www.kaggle.com/nishantbhadauria). (2020). Hunting Exoplanets using SMOTE/SCALE_POS_WEIGHTS, dirección: `https://www.kaggle.com/nishantbhadauria/hunting-exoplanets-using-smote-scale-pos-weights` (visitado 14-07-2020).

[35] A. D. (https://www.kaggle.com/aleksod). (2017). 0.75 Precision/0.60 Recall Linear SVC, dirección: `https://www.kaggle.com/aleksod/0-75-precision-0-60-recall-linear-svc` (visitado 14-07-2020).

[36] U. W. (https://www.kaggle.com/uzairwali). (2019). Exoplanet Classifier (CNN + RNN), dirección: `https://www.kaggle.com/uzairwali/exoplanet-classifier-cnn-rnn/comments` (visitado 14-07-2020).