Viento en popa: geometría eficiente en redes neuronales

TRABAJO FIN DE GRADO

UNIVERSIDAD COMPLUTENSE
MADRID

Facultad de Ciencias Matemáticas

GRADO EN MATEMÁTICAS

Enrique Ernesto de Alvear Doñate

Tutor: Robert Monjo Agut

Departamento de Álgebra Geometría y Topología

Curso 2022/2023

Resumen

Las redes neuronales artificiales son una herramienta matemática muy poderosa, con ella se pueden reconocer imágenes, diseñar inteligencias artificiales que pueden superar a los mejores jugadores de ajedrez o construir modelos de predicción. En este caso, el presente trabajo, se centra en el diseño de modelos de anticipación. En particular, el objetivo es analizar si las redes neuronales artificiales tienen la suficiente habilidad para diseñar un modelo de predicción de una variable con comportamiento altamente no lineal e incluso caótica, el viento. El trabajo se estructura de la siguiente manera:

Primero se explica qué es una red neuronal y cuál es su funcionamiento, por qué son tan versátiles y una herramienta con un potencial inmenso. Finalmente, hizo un estudio de cómo distintos modelos van generando la predicción del viento, para poder encontrar un modelo útil y fiable, utilizando datos reales de cuatro ciudades españolas.

Los modelos diseñados han mostrado resultados favorables, medidos utilizando el error medio cuadrático de la predicción, con valores desde $40m^2/s^2$ para el modelo más débil, hasta un error cercano a $3m^2/s^2$ para el modelo más complejo, utilizando un conjunto de datos de validación para esta medida.

Se ha podido concluir con todo este estudio que las redes neuronales artificiales son realmente una herramienta con mucho potencial que se puede aplicar a problemas tan complejos como puede ser una predicción de una variable caótica.

Abstract

Artificial neural networks are a very powerful mathematical tool, which can be used to image recognition, design artificial intelligences that can outperform the best chess players or build prediction models. In this case, the present work focuses on the design of anticipation models. In particular, the aim is to analyze whether artificial neural networks have enough networks have sufficient ability to design a prediction model of a variable with highly nonlinear and even chaotic behavior, the wind. The study is structured as follows:

First, it is explained what a neural network is and how it works, why they are so versatile and a tool with immense potential. Finally, a study of how different models are generating the wind prediction was made, in order to find a useful and reliable model, using real data from four Spanish cities.

The models that have been created have given favorable results, measured using the mean square error of the prediction, with values from $40m^2/s^2$ for the weakest model, to an error close to $3m^2/s^2$ for the most complex one, using a validation set for this measure.

It has been possible to conclude with all this study that artificial neural networks are really a tool with a lot of potential that can be applied to problems as complex as the prediction of a chaotic variable.

Índice

1. Introducción

1.1. Historia de las redes neuronales artificiales

El ser humano siempre ha mirado a la naturaleza para buscar inspiración y poder mejorar todo lo que necesita, así que, tan solo era cuestión de tiempo que se fijase en la estructura más compleja existente, el cerebro, más concretamente, las neuronas. Son una estructura altamente sofisticada capaz de reconocer patrones, razonar, procesar una gran cantidad de información al mismo tiempo y en general aprender. La estructura que se ha creado para intentar simular este mismo comportamiento son las Redes Neuronales Artificiales (RNA para abreviar).

Las RNA son un concepto que fue mencionado por primera vez por el neurofisiólogo Warren McCulloch y el matemático Walter Pitts en el paper [1], "A Logical Calculus of Ideas Immanent in Nervous Activity", en el cual explican el funcionamiento de un modelo computacional sobre cómo podrían funcionar las neuronas biológicas para hacer cómputos usando lógica de Umbral. Este es el primer concepto que se tiene de un modelo de red neuronal artificial. Abrió dos ramas de investigación, una enfocada en el estudio de los procesos biológicos del cerebro y otra dedicada a la aplicación de las redes neuronales a la inteligencia artificial.

Donald Hebb en 1949 explicó en "The organization of Behavior"[2] desde un punto de vista psicológico cómo funcionaba el aprendizaje, estableciendo así una base de cómo serían las funciones de aprendizaje hoy en día de una red neuronal. Se basaba en que el aprendizaje ocurría cuando ciertos receptores de estímulos en una neurona eran activados, es decir que se fortalecen las conexiones entre las neuronas cada vez que son activadas, argumentando que si dos neuronas se activan a la vez su conexión mejora.

No fue hasta los años 50, cuando los avances en los computadores permitieron simular una red neuronal. Este acto fue desarrollado por Nathanial Rochester, fue conocido como el "Perceptrón de Mark I". Desgraciadamente esta simulación falló no consiguiendo los resultados esperados debido a limitaciones tecnológicas y teóricas, ya que no había algoritmos de entrenamiento desa-

[1] "A Logical Calculus of Ideas Immanent in Nervous Activity," W. McCulloch y W. Pitts (1943).

[2] "The organization of Behavior", D.O. Hebb (1949)

rrollados aún. No fue hasta 1956 que Rochester junto a Minsky, McCarthy y Shanon, organizaron la primera conferencia de Inteligencia Artificial patrocinada por la Fundación Rochester. Fue la primera vez que se tomó en serio a las redes neuronales artificiales, estimulando la investigación en IA basada en redes neuronales.

En 1957, Frank Rosenblatt empezó a trabajar en el proyecto del Perceptrón, el cual es un clasificador de patrones, inspirado en los ojos de las moscas. Consistió en una rejilla de 400 células fotorreceptoras, las cuales simularían las neuronas receptoras de la retina, y según su activación mandarían impulsos eléctricos a un conjunto de unidades conectadas de forma aleatoria, ver Figura 1. Estas unidades simulan el trabajo de las neuronas, y según cuáles producirían una respuesta en una neurona de salida a la que están conectadas todas las unidades anteriores.

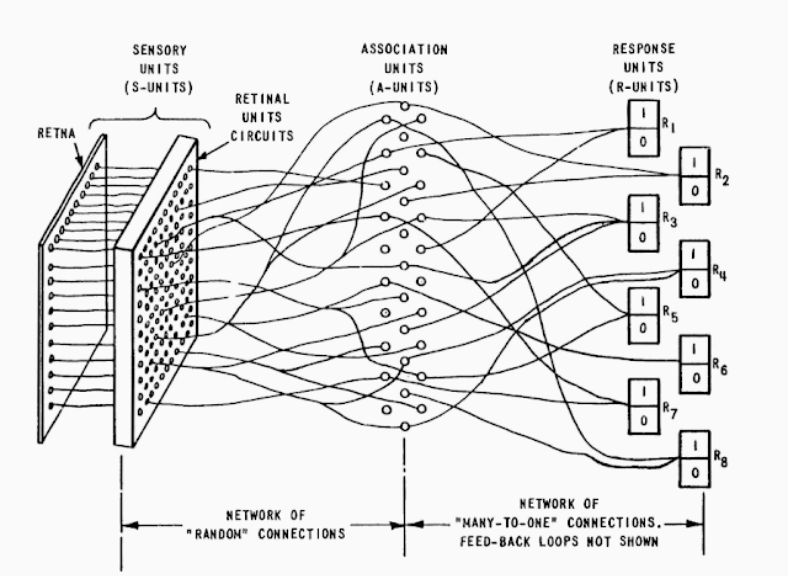


Figura 1: Perceptrón de Frank Roseblatt

El problema que hubo con esta máquina es que era muy limitada, por ejemplo era incapaz de resolver el problema de computar la función XOR (Or-exclusivo) y en general de clasificar clases no separables de forma lineal. En los años 60 se consiguió desarrollar los sistemas multicapa, que pueden resolver problemas más complejos.

En 1959 en Stanford, Bernard Widrow creó “Adaline” y “Madaline”, que son modelos neu-

ronales adaptativos, los cuales se llegaron a utilizar para tareas más complicadas como eliminar ecos de las líneas telefónicas.

En los años 60 Minsky y Papert demostraron matemáticamente que el Perceptrón no era capaz de resolver problemas sencillos no lineales y por lo tanto criticaron fuertemente en su libro "Perceptrons", lo que causó que disminuyeran las inversiones en investigación de la computación neuronal, casi llegando a conseguir que se dejase de investigar sobre el tema.

En 1974 Paul Werbos desarrolló la idea del algoritmo de retropropagación (backpropagation), el cual junto con la publicación del libro de John Hopfield "Computación neuronal de decisiones en problemas de optimización" en 1985, provocó que se reimpulsase el estudio de la computación neuronal.

En la actualidad las redes neuronales son uno de los campos de estudio más importantes en computación y existen infinidad de aplicaciones para las que se pueden utilizar.

1.2. Estructura de las redes neuronales

Las redes neuronales artificiales están basadas en el cerebro, no solamente en funcionamiento, sino también en la estructura.

Las neuronas como se puede ver en la Figura 2 tienen el cuerpo donde está el núcleo, unas dendritas, las cuales son los órganos receptores de la neurona, y el axón, que conduce la información originada en la neurona a la siguiente por los terminales donde se produce la sinapsis. Las neuronas se van conectando por los axones formando redes por las cuales se comparten la información. Una sola neurona funciona de forma muy simple, pero en el cerebro hay miles de millones de neuronas conectadas, se ha conseguido ver partes del cerebro y se ha observado, que en ocasiones las neuronas se organizan por capas consecutivas como se puede ver en el siguiente dibujo de Ramón y Cajal (Figura 3).

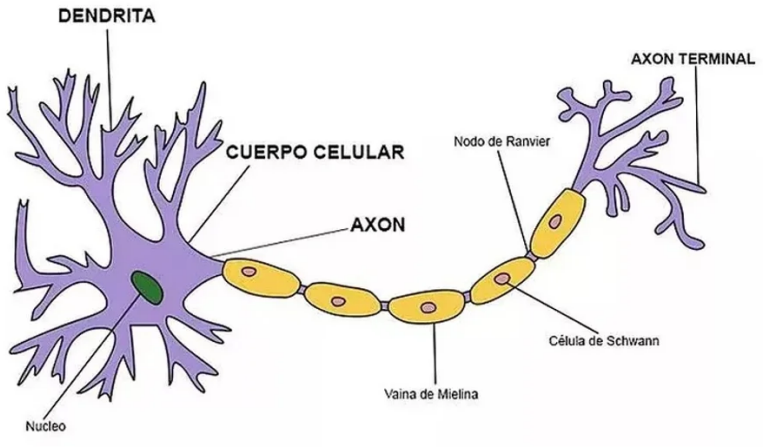


Figura 2: Estructura de una neurona

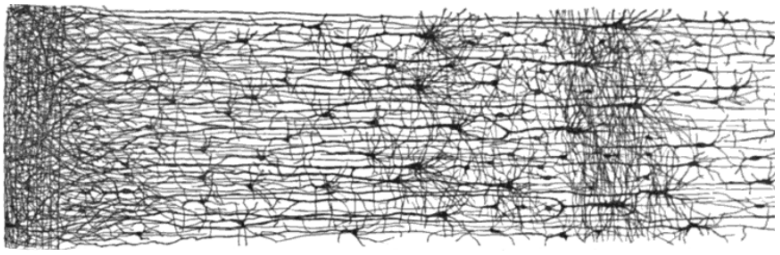


Figura 3: Red neuronal dibujada por Ramón y Cajal

La estructura de una neurona artificial se compone de:

1) Parámetros de entrada: las neuronas artificiales reciben un conjunto de datos de entrada $\{x_i\}_{i=1}^{n}$, donde muchas veces incluye un parámetro extra b llamado sesgo, para ayudar al aprendizaje de la red.

2) Pesos: con cada parámetro de entrada x_i viene asociado un peso w_i, no depende del valor x_i, determinan el grado de influencia de ese dato de entrada para que se active la neurona. Usualmente se agrupan todos los pesos de cada capa en una matriz $W^k = \{w_{i,j}^k\}_{i,j=1}^{n,m}$, donde n es el número de neuronas en la capa actual, m el número de neuronas en la capa anterior y k es la capa en la que estamos. Si la red tiene N capas, con $k \in \{2, N\}$, ya que la primera capa es la capa de input de los datos, a su vez podemos agrupar todas las matrices de pesos en $W = \{W^k\}_{k=2}^{N}$.

3) Función de agrupación: es la función que agrupa todos los datos de entrada con sus pesos para aplicar la función de activación, a esto también se le llama activación de la neurona. Normalmente se hace la suma de los valores de entrada multiplicado por sus pesos:

$$z = \sum_{i=1}^{n} x_i w_i + b$$

También se pueden usar otras funciones como el productorio de todos los datos de entrada y sus pesos, o el máximo de los datos de entrada multiplicado por sus pesos:

$$z = \prod_{i=1}^{n} x_i w_i$$

$$z = \max_{i}(x_i w_i)$$

4) Función de activación: las neuronas artificiales al igual que las biológicas pueden estar activas o inactivas. Para determinar esto usamos lo que se conoce como una función de activación, que determina el estado de activación de una neurona, normalmente se usa $\{0, 1\}$ para determinar si está activa o inactiva, pero también se puede utilizar un intervalo para ver la respuesta que tiene la neurona a un cierto estímulo, se suele usar el intervalo $(0, 1)$ o $(-1, 1)$. Se utiliza también un parámetro llamado umbral θ, el cual marcaría el mínimo valor por el cual la neurona se activa. Varios ejemplos de funciones de activación son:

- Función lineal:

$$f(x) = \begin{cases} -1 & x \leq -1/a \\ a * x & -1/a \leq x \leq 1/a \quad a > 0 \\ 1 & x > 1/a \end{cases}$$

- Función sigmoidea:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Función tangente hiperbólica:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Función ReLU:

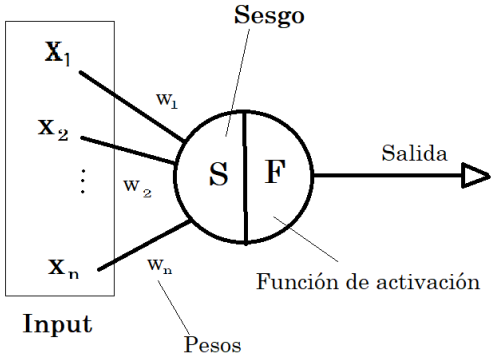
$$ReLU(x) = max(0, x)$$

5) Función de salida: normalmente se usa la función identidad, dejando el valor que salga en la función de activación como salida, o una función umbral, muchas veces se suele componer con la función de activación, usando $x = z - \theta$, y la salida es el valor de la función. Si no, se puede usar la función, para devolver un valor binario 0 o 1, según la activación a de la neurona, donde θ es el umbral.

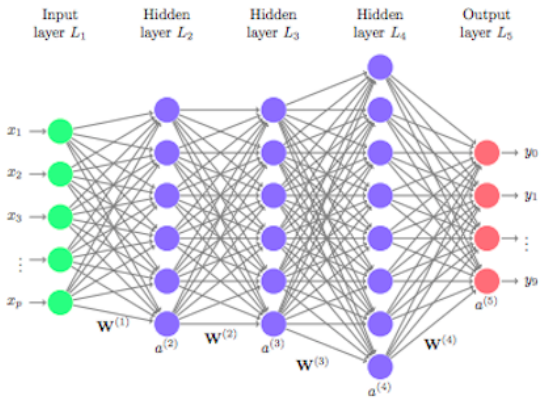
$$\begin{cases} 1 & a \geq \theta \\ 0 & a < \theta \end{cases}$$

Todo esto se puede ver expresado de forma esquemática en la Figura 4a.

Las neuronas se agrupan en capas, con la primera siendo la capa de entrada (input layer), la última la de salida (output layer) y todas las capas intermedias se llaman capas ocultas (hidden layers). Se van transmitiendo de capa en capa los resultados de las funciones de salida de las neuronas que estén interconectadas, si una capa tiene todas las neuronas conectadas con todas las de la siguiente se dice que es densa. Así, todo el conjunto forma una red de neuronas, a lo que se llama red neuronal (Figura 4b).



(a) Estructura de una neurona artificial



(b) Red neuronal artificial

1.3. Aprendizaje y entrenamiento

El objetivo del diseño de una red neuronal artificial, es conseguir que se comporte de una forma tal que con los datos de entrada adecuados nos devuelva un resultado de salida deseado. Por ejemplo, como se ha mencionado antes, las redes neuronales se pueden utilizar para reconocimiento de imágenes, es decir, si se introduce como dato de entrada una imagen, el objetivo

es que reconozca adecuadamente lo que hay en la imagen, o en el caso en el que se centra este trabajo, el objetivo deseado es que al proporcionar unos datos predictores del viento, como resultado que devuelva una predicción adecuada del mismo. Para ello, no hace falta solamente tener la red, habrá que ajustar todos los parámetros de los pesos y las funciones que se computan en la red neuronal para que haga esto correctamente, y de lo cual se encarga el entrenamiento o aprendizaje de la red neuronal.

Se utiliza lo que se llama un entrenamiento supervisado, es decir, se introduce un grupo de datos ya tratados con una salida ya conocida, para que la red compare con la salida que ha obtenido y si ha cometido algún error se modifique para que en la siguiente predicción lo haga correctamente. De otra forma, también se podrían usar métodos de entrenamiento no supervisado, un ejemplo de ello serían las redes de Hebb, las cuales aprenden reforzando el peso entre dos neuronas si estas se activan a la vez y los pesos se actualizan $w'_{ij} = w_{ij} + \alpha a_i a_j$, donde α es un parámetro real distinto de 0, a_k es el indicador de si la neurona k se han activado $a_k = 1$ o no $a_k = 0$.

En el aprendizaje supervisado de las redes neuronales el algoritmo más importante es el de retropropagación (backpropagation algorithm). Es un algoritmo aplicado para entrenar las redes neuronales que son del tipo "feed-forward", es decir, tienen una capa de entrada, una de salida y el flujo de la red neuronal nunca va hacia atrás. El algoritmo consiste en un descenso de gradiente de la función del error. Un ejemplo de función de error sería la función del error medio cuadrático $MSE(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$, para regresión lineal, donde x es el vector de variables de entrada, y es el valor esperado, θ es el vector de los parámetros de la regresión, y queremos ver cómo cambia la función del error por cada cambio de cada θ_i, calculamos el gradiente:

$$\frac{\partial}{\partial\theta_i}MSE(\theta) = \frac{2}{m}\sum_{j=1}^{m}(\theta^T x^{(j)} - y^{(j)})x_i^{(j)}$$

El vector gradiente $\bigtriangledown MSE(\theta)$ apunta hacia la dirección que maximiza la función, por lo tanto tomando el opuesto podremos encontrar los valores para los que se minimiza. Así, renombrando los θ de forma correcta, podremos encontrar el valor para el cual el error de predicción es mínimo.

El algoritmo de la retropropagación se basa en esto.

Algoritmo: Backpropagation

El algoritmo de retropropagación o "backpropagation" es un algoritmo de aprendizaje para redes prealimentadas o "feedforward", para las cuales va cambiando el conjunto de todos los pesos de la red $\{w_i^j\}$ según el error cometido con la predicción, haciendo mayor cambio en los pesos que hayan provocado mayor error en la predicción. Para ello, se toma la función del error cometido en la predicción $E = \sum_{i=1}^{N} \frac{1}{2}||a_i - y_i||^2$, para un conjunto de entrenamiento $\{x_i, y_i\}_{i=1}^{N}$, con x_i los parámetros de entrada, y los y_i, los valores observados (la salida deseada), los a_i son los valores predichos para la entrada x_i por la red neuronal. Por lo tanto, para cada capa se reajusta la matriz de pesos $W_j = W_j + \triangle W_j$. Lo que se desea averiguar es qué es $\triangle W$. Para ello se toma la red neuronal como una función que depende de los valores de entrada x y de las matrices de pesos W_j, $N(x, W_2, ..., W_m)$, entonces la función de error será $E(x, W_2, ..., W_m) = \sum_{i=1}^{N} \frac{1}{2}||N(x, W_2, ..., W_m) - y_i||^2$. Tomando las derivadas parciales respecto a cada una de las matrices de pesos W_j, se obtiene el vector de dirección máxima. Usando el opuesto de este se tendrá la forma de minimizar el error, cambiando los pesos, esto es tomando $\triangle W_j = -\gamma \frac{\partial E(x, W_1, ..., W_m)}{\partial W_j}$, siendo $\gamma \in (0, 1)$ un parámetro conocido llamado tasa de aprendizaje. El proceso para computar todo esto sería:

1) Inputs:

1. Una tasa de aprendizaje $0 < \lambda < 1$

2. Un valor crítico $\epsilon > 0$ por el cual se determina que la red está convergiendo y un número de iteraciones máximas $MaxIt$

1. Un sistema $\{x_i, y_i\}_{i=1}^{N}$ de N elementos

2. Una Red Neuronal $N(x, W_2, ..., W_m)$

3. Una función de activación f para las neuronas

2) Outputs: un conjunto de matrices de pesos $\{W_j\}_{j=2}^{m}$

3) Objetivo: obtener una matriz de pesos que minimice todo lo posible el error

4) Proceso:

a) Inicializar un contador de iteraciones cont = 1

b) Calcular las salidas de los datos de entrenamiento $a_i^m = N(x_i, W_2, ..., W_m)$ tomando $a_i^j = f(W_j a_i^{j-1})$, es decir el vector con la activación de la capa j, sabiendo que $a_i^1 = x_i$

c) Hacer la propagación hacia atrás del error de todas las neuronas de salida y de las capas ocultas:

$$E_i^m = a_i^m - y_i$$

$$E_i^{m-k} = W_{m-k} E_i^{m-k+1} f'(a_i^{m-k})$$

d) Reajustar el valor de los pesos $W_j = W_j - \lambda E^i a_i^j$, para cada uno de los datos de entrada $i \in 1, ..N$

e) Si $cont = 1$, guardar el error $E = \sum_{i=1}^{N} E_i^m$ y aumentar el contador $cont = cont + 1$ y volver al segundo paso

f) Parar cuando $\sum_{i=1}^{N} E_i^m - E < \epsilon$ o si se alcanza el máximo número de iteraciones. Si no se cumplen ninguna de estas condiciones $E = \sum_{i=1}^{N} E_i^m$ y $cont = cont + 1$ y volver al paso 2

Lo que se está haciendo es calcular el $\triangle W_j = -\gamma \frac{\partial E^j(x, W_j)}{\partial W_j} \simeq -\gamma E^j a_i^j$ y aplicar un descenso de gradiente en la matriz de pesos.

1.4. El problema de predecir sistemas dinámicos no lineales

Un sistema dinámico es un conjunto de elementos que están relacionados, los cuales tienen unos estados y van evolucionando con el tiempo. Existe un gran número de ejemplos de sistemas dinámicos en prácticamente todas las ramas del conocimiento, por ejemplo, modelos económicos, modelos poblacionales para especies, una reacción química, un sistema físico de partículas... Los sistemas dinámicos se pueden modelizar matemáticamente de muchas formas. En física, por ejemplo, un sistema dinámico suele describirse con ecuaciones diferenciales para poder modelizarlos, ya que se puede tomar como valor inicial un estado de los elementos del sistema y ver su evolución como una ecuación diferencial ordinaria, cuyo parámetro es el tiempo y los elementos dependen del estado inicial y del tiempo.

No siempre se pueden hacer predicciones con los sistemas dinámicos de forma precisa, ya que son muy sensibles a las condiciones iniciales, por lo tanto, hay que recurrir a modelos estocásticos para poder conseguir una aproximación. En muchos casos, incluso esto, es insuficiente para poder hacer predicciones a largo plazo, ya que muchos de los sistemas dinámicos, sobre todo en los que no son lineales, pueden presentar un comportamiento caótico, es decir que sean muy sensibles a los estados iniciales y ello provoque que el valor predicho se aleje mucho del valor real.

Este es el caso que concierne a este trabajo, el viento, se rige por las ecuaciones de la dinámica de fluidos, que se puede describir de forma simplificada utilizando la ecuación de Navier-Stokes, que es la ecuación principal de la dinámica de fluidos. A día de hoy, no se ha podido resolver de forma analítica esta ecuación, siendo de hecho uno de los famosos problemas del milenio.

Para poder hacer predicciones, hay que recurrir a métodos numéricos que se ajusten lo máximo posible a una solución de este sistema, a lo que se añade la complejidad de que además es un sistema caótico, por lo tanto no se pueden hacer predicciones seguras a largo plazo, ya que lo más probable es que la realidad no se ajuste a la predicción. Existen predicciones que fallan por un cambio repentino, por lo tanto, normalmente se usan varios tipos de modelos, para generar con ellos predicciones lo más precisas posibles.

Por lo tanto, el objetivo de este trabajo ha sido crear un modelo de predicción del viento utilizando el gran poder computacional de las Redes Neuronales Artificiales, para ello se investigan varios tipos de estructuras distintas de la red, viendo cuál es el más preciso.

2. Datos y metodología

Para toda esta parte se utiliza el lenguaje de programación Python, para modelizar redes neuronales, usando la API Keras, la cual se ejecuta sobre la plataforma TensorFlow. También se hace uso de otras librerías como son NumPy, sklearn, Pandas y Matplotlib, para poder operar, tratar con datos y visualizar los resultados.

2.1. Datos y control de calidad

Los datos con los que se ha trabajado están basados en un modelo meteorológico, llamado Action de Recherche Petite Echelle Grande Echelle (ARPEGE), un modelo desarrollado en conjunto por Météo-France y el Centro Europeo para Predicción Meteorológica de Medio-alcance (ECMWF). Consisten en 15 variables predictoras (predU10m, predV10m, predM10m, predU150m, predV150m, predM150m, predMes, predHora, predMesU, predMesV, predHoraU, predHoraV, predDia, predDiaU y predDiaV) y 3 variables observadas (obsU, obsV y obsM). El trabajo está hecho con datos de Madrid, Tarifa, Barcelona y A Coruña, los datos están organizados por fecha y hora, datando desde 01/04/2017 hasta el 30/03/2022. Primero se calcula la correlación que tienen los datos con cada uno de los predictores. Se observa en la Figura 5, que hay algunos predictores que influyen muy poco para predecir obsU, obsV y obsM, pero aún así, si no se añaden, se pierde poder de predicción con la red, la correlación entre las variables a predecir es alta y se puede usar esto para ayudar a construir de forma inteligente los modelos.

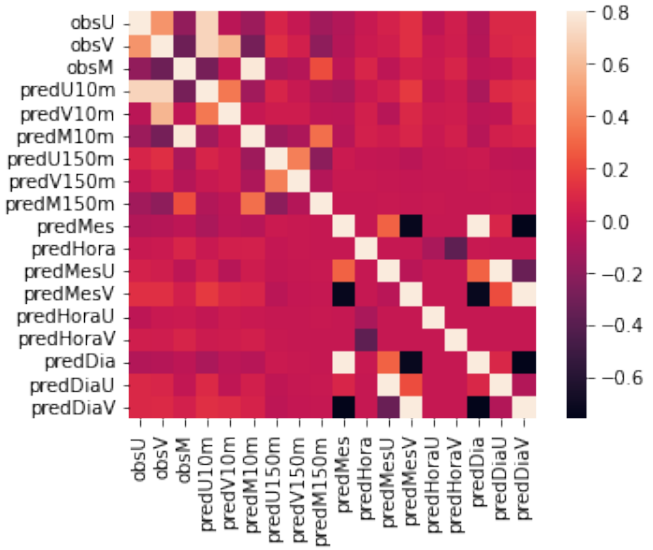


Figura 5: Gráfico correlaciones de los datos

Para depurar y obtener los datos se usa la librería Pandas, eliminando todas las filas que tengan NA (not available), ya que la falta de ese predictor puede provocar problemas en el modelo. Luego se quitan las columnas que indican la fecha y la hora, y se separan las variables predicto-

ras de las variables observadas en dos dataframes distintos, X e Y respectivamente. Ver Anexo 5.2

Un paso muy importante que se hace para las redes neuronales es hacer un tratamiento de los datos de entrada, es decir, transformarlos para que le sea más fácil a la red tratar con ellos. La forma más común de transformarlos es normalizándolos, es decir, tranformándolos de tal forma que la media sea 0 y tenga desviación típica 1. Esto se puede hacer utilizando métodos de la librería sklearn.Ver Anexo 5.3

2.2. Metodología

En este proyecto, se ha utilizado una función auxiliar para poder visualizar gráficamente la capacidad predictiva de los modelos. Para ello, se crea una nube de puntos, comparando el valor predicho, para un subconjunto del conjunto de validación, con el valor real, y se calcula la recta de regresión de toda esa nube de puntos, la cual si la predicción es buena se debería ajustar a la recta $y = x$. Para esto, se usa la librería gráfica matplotlib, además de numpy para algunos cálculos y un módulo de sklearn para hacer la recta de regresión. Consultar Anexo 5.4.

Ya con los datos depurados y las funciones auxiliares necesarias, se empieza a crear los modelos, para ello se utiliza la clase keras.layers.Input() y keras.layers.Dense() de keras. Para entrenar el modelo se hace uso de una validación cruzada (kfold cross validation), que es un método estadístico para valorar modelos de predicción, el cual divide el conjunto de datos en k grupos, usando un conjunto de esos grupos para entrenar el modelo y el resto para evaluarlo. Esto permite comparar el rendimiento del modelo, para ello se usa la función KFold de sklearn para separar los datos. Finalmente se entrena el modelo utilizando las funciones de Keras compile() y fit(). Haciendo uso de la librería matplotlib para ver cómo evoluciona el error en la predicción, para observar gráficamente cómo va aprendiendo. Ver Anexo 5.5.

En este trabajo se usa, sobre todo, las funciones de activación para las capas ocultas es la función $ReLu$ y para la capa de salida la función lineal. Como medida del error, se hace uso de el error medio cuadrático (MSE). Con toda esta información se puede ver cómo son los distintos modelos de redes neuronales.

Respecto al entrenamiento, se aplica un "early stop", es decir, en el aprendizaje, se va monitorizando la curva de minimización del error, y cuando se empieza a ralentizar el aprendizaje en los datos de validación, el early stop detiene el proceso de aprendizaje , para evitar que haya *overfitting* del modelo con respecto a los datos de entrada. El *overfitting* es un problema que ocurre en los modelos de predicción cuando el modelo se ajusta tanto a los datos de entrenamiento que deja de dar valores significativos para valores predichos, es decir, el error de predicción es mayor que en iteraciones anteriores. Queremos encontrar ese punto en el que el error en la predicción ha descendido lo suficiente, pero no ha llegado a ajustarse el modelo a los datos como para provocar *overfitting*. Para ello, se emplea la función EarlyStopping() de keras, que se ocupa de detener el entrenamiento cuando el error en la validación no ha disminuido, como mínimo, un cierto valor en las últimas iteraciones, se puede especificar el número de iteraciones para la comprobación. Para este estudio se ha utilizado 5, y como valor mínimo se usa 0,01. En la Figura 6 se puede ver una repesentación del problema del overfitting.

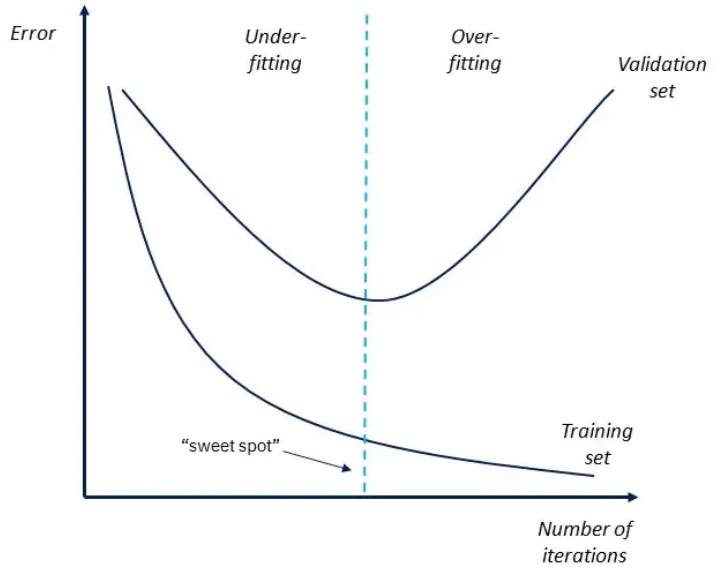


Figura 6: Representación del overfitting

Por lo tanto, cada uno de los experimentos que se ha llevado a cabo ha consistido en:

1) Crear la estructura del modelo que se quería estudiar. Primero se construye una capa de entrada (keras.layers.Input()), se conecta esa capa con las sucesivas, las cuales son del tipo keras.layers.Dense() y se crea el modelo con keras.models.Model([capainput, capaoutput]).

2) Se compila el modelo asignando la función de pérdida, que en este caso era el error medio cuadrático "MSE" y el optimizador.

3) Se crea la función callback, la cual se ocupa de que no haya overfitting. Y se entrena el modelo con un subconjunto de los datos para hacer la validación cruzada.

3) Ya con el modelo entrenado, se estudia la curva de aprendizaje y se evaúuan los resultados. El modelo final era el que obtuviese menor error medio cuadrático.

4) Se hace una evaluación del modelo usando la función auxiliar anteriormente mencionada, y se analizan todos los resultados obtenidos.

3. Resultados y discusión

3.1. Primera aproximación: Perceptrón

Para un primer intento de modelo, se utilizó la estructura del perceptrón mencionada anteriormente. Consiste en la capa de inputs y la capa de salida, véase es un modelo con 18 neuronas, 15 de ellas son de entrada, están conectadas de forma densa con la capa de salida, la cual tiene 3 neuronas, la función de activación es lineal. Se ha entrenado para el conjunto de datos de Madrid.

Como se puede observar en las siguientes figuras, para el modelo de perceptrón, cuyo código se encuentra en el anexo 5.1, y usando los datos de Madrid, a pesar de que se le ha establecido que haga 5000 iteraciones del algoritmo de aprendizaje, solo ha completado 134 como se puede ver en la Figura 8. Esto se debe a que en las últimas 5 iteraciones no estaba disminuyendo la tasa de aprendizaje lo suficiente, cuyo decrecimiento se ve representado en la Figura 7.

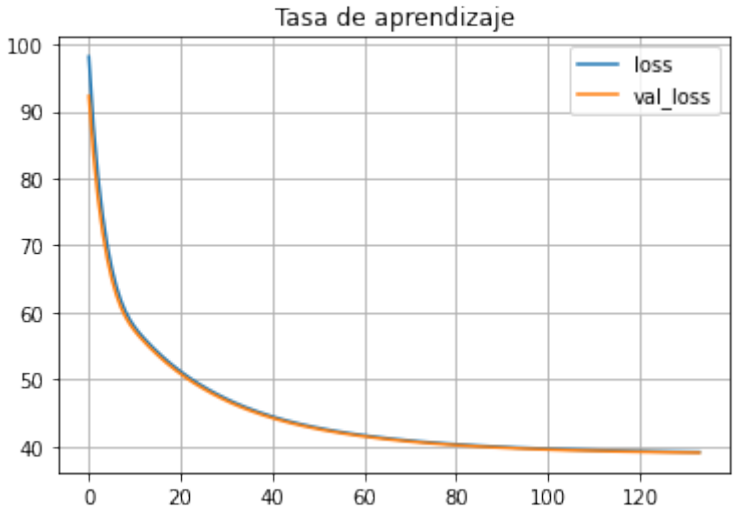
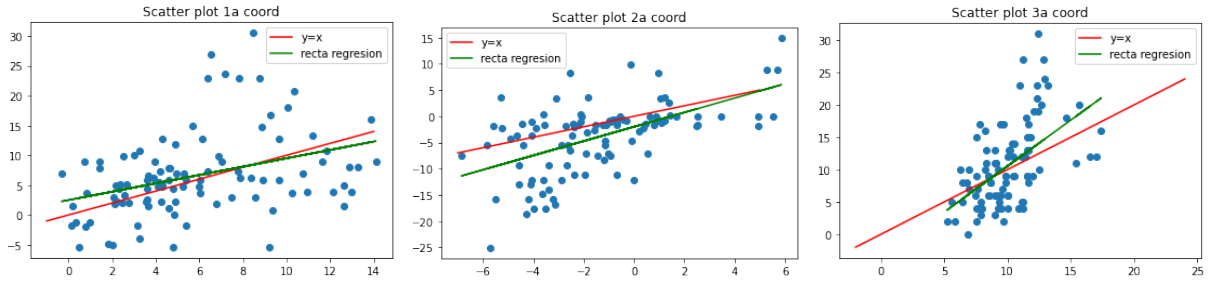


Figura 7: Tasa aprendizaje Perceptrón, donde los ejes representan "número de epochs" frente el error medio cuadrático medido en m^2/s^2.

```
Epoch 1/2000
480/480 [==============================] - 2s 3ms/step - loss: 98.1568 - val_loss: 92.2620
Epoch 2/2000
480/480 [==============================] - 1s 2ms/step - loss: 87.8701 - val_loss: 83.5122
Epoch 3/2000
480/480 [==============================] - 1s 2ms/step - loss: 80.2751 - val_loss: 76.9655
Epoch 4/2000
480/480 [==============================] - 1s 2ms/step - loss: 74.5167 - val_loss: 71.9277
Epoch 5/2000
480/480 [==============================] - 1s 2ms/step - loss: 70.0711 - val_loss: 68.0234
Epoch 6/2000
480/480 [==============================] - 1s 2ms/step - loss: 66.5890 - val_loss: 64.9572
Epoch 7/2000
480/480 [==============================] - 1s 3ms/step - loss: 63.8615 - val_loss: 62.5578
Epoch 8/2000
480/480 [==============================] - 1s 2ms/step - loss: 61.7473 - val_loss: 60.7176
Epoch 9/2000
480/480 [==============================] - 1s 2ms/step - loss: 60.1101 - val_loss: 59.2761
Epoch 10/2000
480/480 [==============================] - 1s 2ms/step - loss: 58.8283 - val_loss: 58.1425
Epoch 11/2000
480/480 [==============================] - 1s 2ms/step - loss: 57.8003 - val_loss: 57.2105
Epoch 12/2000
480/480 [==============================] - 1s 2ms/step - loss: 56.9254 - val_loss: 56.3893
Epoch 13/2000
...
Epoch 133/2000
480/480 [==============================] - 1s 2ms/step - loss: 39.1671 - val_loss: 39.0653
Epoch 134/2000
480/480 [==============================] - 1s 2ms/step - loss: 39.1584 - val_loss: 39.0578
```

Figura 8: Proceso de aprendizaje del Perceptrón

Con esto queda entrenado el modelo, en las siguientes imágenes se puede observar el análisis de resultados de la predicción, usando la función auxiliar definida anteriormente para ver cómo se ajusta la predicción a la realidad.



(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 9: Nube de puntos de la predicción del modelo Perceptrón. Ambos ejes de cada gráfica se miden en m/s.

Como se puede observar en la Figura 9, aunque las rectas de regresión en el intervalo de valores que obtenidos se parece a la recta $y = x$, la cual, es el objetivo deseado, pero las nubes de puntos están muy dispersas, lo cual no es un resultado aceptable. En la Figura 10 se puede ver el scoring del modelo, lo que se muestra es el MSE del conjunto de entrenamiento y del conjunto de validación respectivamente. Ambos valores están alrededor de $39m^2/s^2$, lo que refleja un modelo bastante débil.

El perceptrón, cómo se ha observado, no da buenos resultados, esto tiene sentido ya que si no fuese así entonces significa que con una simple función lineal se podría hacer una predicción buena de un problema que ya hemos visto que viene dado por un sistema no lineal.

```
Scoring del modelo con respecto al MSE: [39.19, 39.10]
```

Figura 10: Scoring del modelo perceptrón

3.2. Segunda aproximación: Red Multicapa simple

El siguiente modelo consisitió en una red multicapa, la cual está conformada por 5 capas densamente conectadas con 250 neuronas en la primera capa y reduciendo el número en 50 cada vez en las siguientes. Las capas tienen como función de activación ReLu y la capa final, tiene activación lineal. Los resultados obtenidos para el conjunto de datos de Madrid que se pueden

ver en las Figuras 11 y 12, son los siguientes:

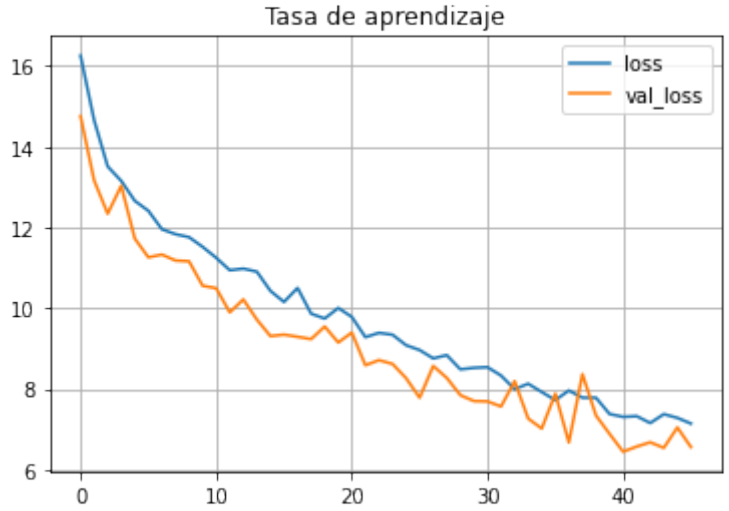


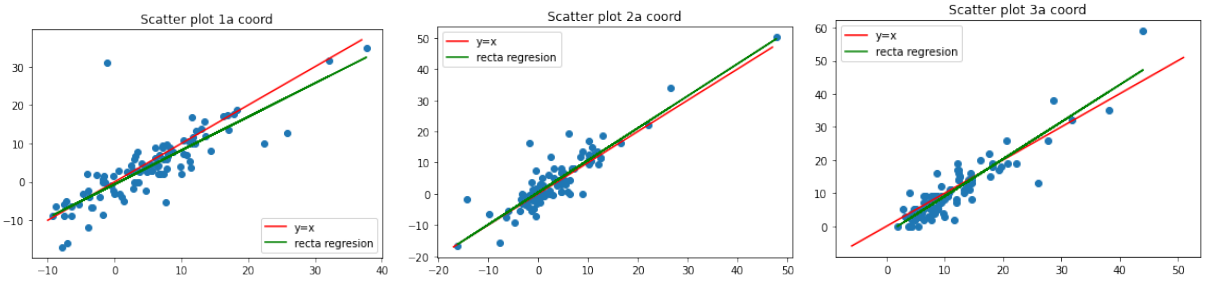
Figura 11: Tasa aprendizaje modelo multicapa versión 1, donde los ejes representan "número de epochs" frente el error medio cuadrático medido en m^2/s^2.

```
Epoch 1/2000
408/408 [==============================] - 3s 5ms/step - loss: 45.3336 - val_loss: 37.0637
Epoch 2/2000
408/408 [==============================] - 2s 4ms/step - loss: 34.3988 - val_loss: 32.8180
Epoch 3/2000
408/408 [==============================] - 2s 4ms/step - loss: 32.6058 - val_loss: 31.6875
Epoch 4/2000
408/408 [==============================] - 2s 4ms/step - loss: 32.0377 - val_loss: 30.8472
Epoch 5/2000
408/408 [==============================] - 2s 4ms/step - loss: 31.7308 - val_loss: 30.6130
Epoch 6/2000
408/408 [==============================] - 2s 5ms/step - loss: 31.4718 - val_loss: 30.4849
Epoch 7/2000
408/408 [==============================] - 3s 6ms/step - loss: 31.1535 - val_loss: 30.4164
Epoch 8/2000
408/408 [==============================] - 3s 7ms/step - loss: 30.9508 - val_loss: 30.1853
Epoch 9/2000
408/408 [==============================] - 2s 6ms/step - loss: 30.5905 - val_loss: 29.4956
Epoch 10/2000
408/408 [==============================] - 2s 5ms/step - loss: 30.4085 - val_loss: 30.6820
Epoch 11/2000
408/408 [==============================] - 2s 5ms/step - loss: 30.3142 - val_loss: 31.3660
Epoch 12/2000
...
Epoch 45/2000
408/408 [==============================] - 2s 4ms/step - loss: 7.2821 - val_loss: 7.0449
Epoch 46/2000
408/408 [==============================] - 2s 4ms/step - loss: 7.1419 - val_loss: 6.5601
```

Figura 12: Proceso de aprendizaje del modelo multicapa versión 1

De la misma forma que antes, las siguientes nubes de puntos (Figura 13) representan la

capacidad predictiva de la red, donde podemos destacar una ligera mejoría de los resultados del modelo.



(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 13: Nube de puntos de la predicción del modelo multicapa versión 1. Ambos ejes de cada gráfica se miden en m/s.

Finalmente con el resultado del scoring del modelo que se tiene a continuación (Figura 14), se puede ver realmente, la mejoría del modelo respecto al MSE, el cual ha bajado de estar alrededor de $39m^2/s^2$ a estar cerca de $6m^2/s^2$. Esto se ha conseguido con solamente añadir unas cuantas capas más al modelo, por lo tanto el siguiente paso lógico es ver cómo afecta a los resultados si se añaden más capas.

```
Scoring del modelo con respecto al MSE: [6.44 , 6.44]
```

Figura 14: Scoring del modelo multicapa versión 1

3.3. Tercera aproximación: Red Multicapa compleja

Al igual que en la sección anterior esta es una red multicapa, pero un poco más compleja. En vez de 5 capas intermedias tiene 8, y empieza con 1000 neuronas, reduciendo el número en 50 con cada una de las siguientes capas. Del mismo modo que en las anteriores, esta ha sido entrenada para los datos de Madrid, obteniendo los siguientes resultados:

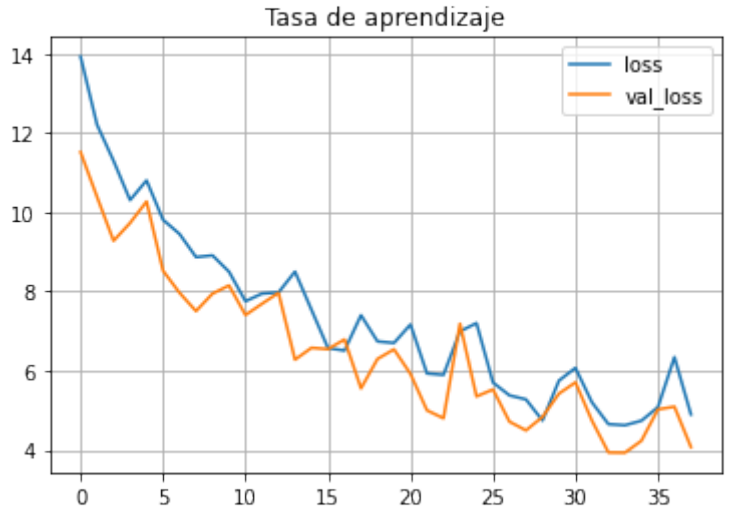


Figura 15: Tasa aprendizaje modelo multicapa versión 2, donde los ejes representan "número de epochs" frente el error medio cuadrático medido en m^2/s^2.

```
Epoch 1/2000
408/408 [==============================] - 39s 93ms/step - loss: 49.1079 - val_loss: 37.9457
Epoch 2/2000
408/408 [==============================] - 40s 98ms/step - loss: 36.8551 - val_loss: 35.7244
Epoch 3/2000
408/408 [==============================] - 33s 81ms/step - loss: 35.1101 - val_loss: 35.0395
Epoch 4/2000
408/408 [==============================] - 28s 69ms/step - loss: 33.4977 - val_loss: 32.5529
Epoch 5/2000
408/408 [==============================] - 28s 69ms/step - loss: 33.2100 - val_loss: 32.2927
Epoch 6/2000
408/408 [==============================] - 27s 66ms/step - loss: 32.2026 - val_loss: 31.1336
Epoch 7/2000
408/408 [==============================] - 27s 66ms/step - loss: 31.9314 - val_loss: 33.2588
Epoch 8/2000
408/408 [==============================] - 27s 66ms/step - loss: 31.7183 - val_loss: 31.5639
Epoch 9/2000
408/408 [==============================] - 27s 67ms/step - loss: 31.0094 - val_loss: 29.7275
Epoch 10/2000
408/408 [==============================] - 27s 66ms/step - loss: 30.7025 - val_loss: 29.0956
Epoch 11/2000
408/408 [==============================] - 27s 66ms/step - loss: 30.0266 - val_loss: 29.1862
Epoch 12/2000
408/408 [==============================] - 27s 67ms/step - loss: 30.0555 - val_loss: 29.0053
Epoch 13/2000
...
Epoch 37/2000
408/408 [==============================] - 35s 85ms/step - loss: 6.3449 - val_loss: 5.0962
Epoch 38/2000
408/408 [==============================] - 26s 63ms/step - loss: 4.9011 - val_loss: 4.0785
```

Figura 16: Proceso de aprendizaje del modelo multicapa versión 2

Con esto ya se tendría entrenada la red, se puede ver una representación del poder predictivo

con la nube de puntos en la Figura 17.

(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 17: Nube de puntos de la predicción del modelo multicapa versión 2. Ambos ejes de cada gráfica se miden en m/s.

Se puede observar en la Figura 18, que obtenemos aún mejores resultados, bajando el scoring de 6 a un poco menor que 4. Estos ya son unos resultados muy buenos, como se puede comprobar también por cómo está ajustada la nube de puntos.

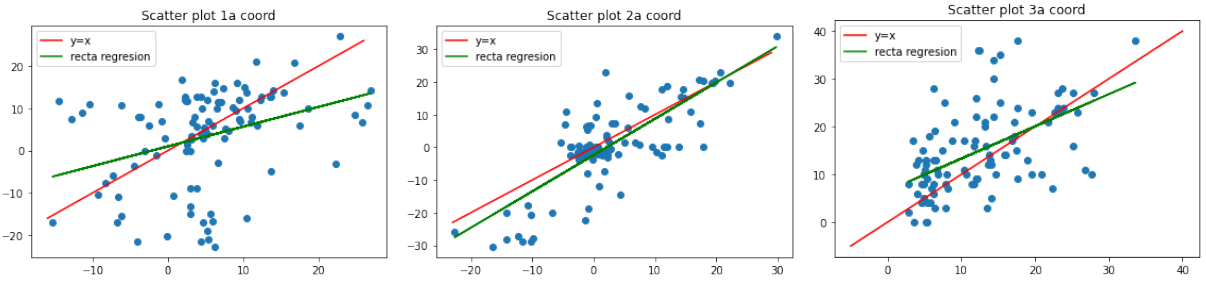
```
Scoring del modelo con respecto al MSE: [3.93, 3.94]
```

Figura 18: Scoring del modelo multicapa versión 2

Ahora la duda es si se pueden obtener resultados similares utilizando esta red para hacer las predicciones de las otras ciudades, ya que solo la hemos entrenado para Madrid. Para probar estos resultados, se ha aplicado el mismo test de scoring que se ha usado anteriormente para medir el MSE y, de forma representativa, las gráficas de nube de puntos de la predicción. Primero, se trata los datos de la misma forma que se ha hecho con los de Madrid, aplicando la misma transformación que a estos. A continuación, se procede con los test mencionados.

```
Scoring del modelo para todos los datos respecto al MSE: 80.94
```

Figura 19: Scoring del modelo multicapa versión 2, para los datos de todas las ciudades



(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 20: Nube de puntos de la predicción del modelo multicapa versión 2, para todos los datos. Ambos ejes de cada gráfica se miden en m/s.

Los resultados, resumidos en las Figuras 19 y 20 son muy distintos a los obtenidos solamente para los de Madrid. Esto se debe a que las condiciones meteorológicas en las diferentes ciudades son muy distintas a las de Madrid. Influye la geografía del terreno y también la variabilidad de los predictores en cada una de las ciudades, que en Madrid suele ser más calmado todo, y en el resto es más volátil.

Podemos concluir que, para esta región, los resultados son favorables, ya que la red está acostumbrada a las condiciones de Madrid y cuando se aplica a otras, empieza a devolver peores resultados.

3.4. Cuarta aproximación: Red Multicapa compleja general

Como se ha visto en el punto anterior, la red entrenada no es suficiente para poder hacer la predicción general de todos los datos que hay disponibles. Para ello, se ha seguido con la misma estructura de red y en vez de entrenar únicamente con los datos de Madrid, vamos a incluir en el entrenamiento datos de todas las geografías disponibles, para que así la red pueda tener esa información. Primero se tratan los datos, normalizando todas las variables, al igual que se ha hecho anteriormente con los datos de Madrid, y usando la misma estructura que la red anterior, vamos a probar el poder predictivo que se conseguiría si se entrena con un subconjunto de todos los datos, dejando el resto de datos para la validación del modelo.

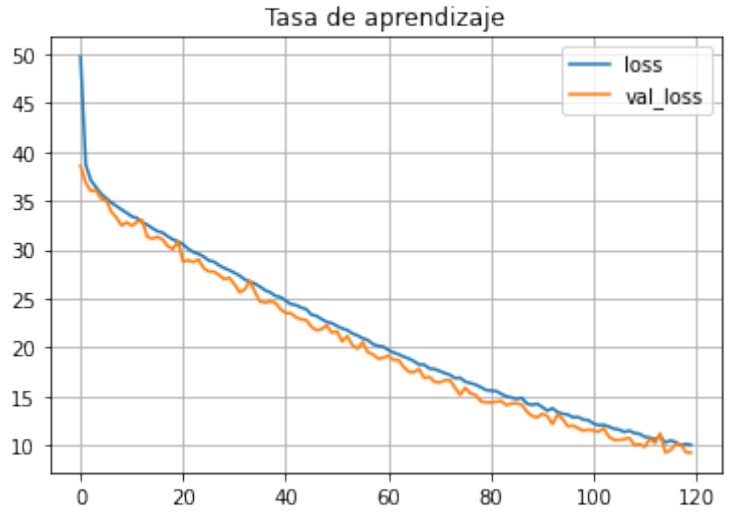
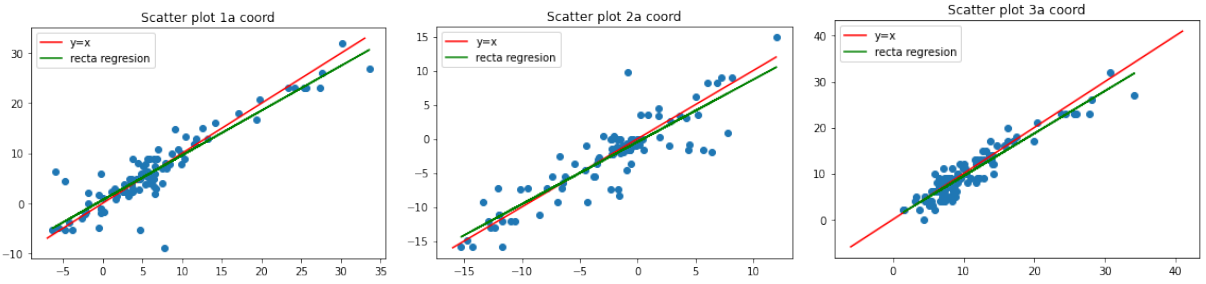


Figura 21: Tasa aprendizaje modelo multicapa versión 3, donde los ejes representan "número de epochs" frente el error medio cuadrático medido en m^2/s^2.

```
Epoch 1/2000
1889/1889 [==============================] - 135s 70ms/step - loss: 49.7328 - val_loss: 38.5626
Epoch 2/2000
1889/1889 [==============================] - 112s 59ms/step - loss: 38.7199 - val_loss: 36.8209
Epoch 3/2000
1889/1889 [==============================] - 111s 59ms/step - loss: 37.0801 - val_loss: 35.9793
Epoch 4/2000
1889/1889 [==============================] - 116s 61ms/step - loss: 36.3964 - val_loss: 36.0678
Epoch 5/2000
1889/1889 [==============================] - 111s 59ms/step - loss: 35.7044 - val_loss: 35.2141
Epoch 6/2000
1889/1889 [==============================] - 111s 59ms/step - loss: 35.2441 - val_loss: 35.0391
Epoch 7/2000
1889/1889 [==============================] - 111s 59ms/step - loss: 34.7977 - val_loss: 33.8579
Epoch 8/2000
1889/1889 [==============================] - 112s 59ms/step - loss: 34.4422 - val_loss: 33.2690
Epoch 9/2000
1889/1889 [==============================] - 113s 60ms/step - loss: 34.0812 - val_loss: 32.4648
Epoch 10/2000
1889/1889 [==============================] - 108s 57ms/step - loss: 33.7477 - val_loss: 32.7695
Epoch 11/2000
1889/1889 [==============================] - 110s 58ms/step - loss: 33.3720 - val_loss: 32.4243
Epoch 12/2000
1889/1889 [==============================] - 108s 57ms/step - loss: 33.2288 - val_loss: 32.8162
Epoch 13/2000
...
Epoch 119/2000
1889/1889 [==============================] - 108s 57ms/step - loss: 10.1012 - val_loss: 9.2703
Epoch 120/2000
1889/1889 [==============================] - 111s 59ms/step - loss: 10.0129 - val_loss: 9.2127
```

Figura 22: Proceso de aprendizaje del modelo multicapa versión 3

Se observa en la Figura 22, que el proceso de aprendizaje se ha ralentizado, mirando el tiempo que tarda cada uno de los "epochs", ya que se tienen muchos más datos de los que la red tiene que aprender. A continuación, se muestra cómo ha sido el rendimiento de esta al terminar su aprendizaje, con las nubes de puntos y el scoring del modelo (Figuras 23 y 24).



(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 23: Nube de puntos de la predicción del modelo multicapa versión 3. Ambos ejes de cada gráfica se miden en m/s.

```
Scoring del modelo con respecto al MSE: [9.23, 9.23]
```

Figura 24: Scoring del modelo multicapa versión 3

La red ha perdido rendimiento comparando con las anteriores aproximaciones únicamente tenía que aprender de los datos de Madrid, ya que el scoring ha subido de $4m^2/s^2$ a $9m^2/s^2$, obteniendo resultados incluso peores que los obtenidos en la segunda aproximación. Esto indica que el problema que hay a resolver es mucho más complicado cuando se añaden todos los datos, por lo tanto, necesitaremos una red más compleja.

3.5. Quinta aproximación: Red multicapa especializada

Para la siguiente aproximación se ha intentado una estructura un tanto diferente. Hasta ahora se ha hecho la predicción de $ObsU$, $ObsV$ y $ObsM$ con una única red, sin embargo, en esta aproximación se ha optado por seguir la filosofía de "Divide y vencerás". Se ha fraccionado el problema en 3 partes, creando una red para cada uno de los valores que se quiere predecir.

Por lo tanto, ahora se ha construido una red que a su vez está dividida en 3 partes distin-

tas, para cada una de las partes se ha utilizado una red con un menor número de neuronas y de capas que en la cuarta aproximación. Para hacer la predicción de $ObsU$ y $ObsV$, se usa una red (para cada una), que consiste en 5 capas con 1000 neuronas en la primera y reduciendo el número de neuronas en 50 por cada capa consecutiva; y para $ObsM$ se ha observado que se pueden conseguir buenos resultados con únicamente 4 capas. Finalmente, se juntan los 3 resultados que serán las predicciones en una capa de concatenación.

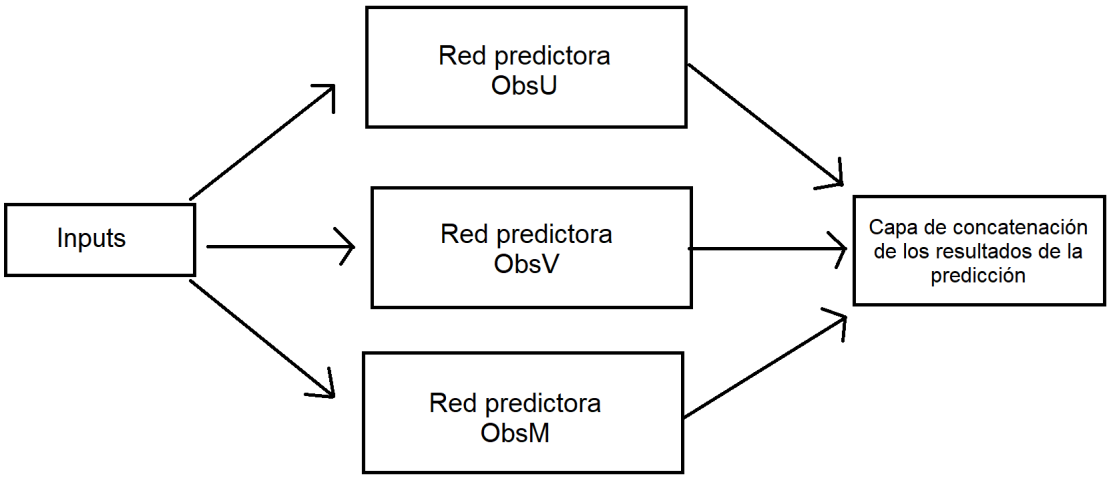


Figura 25: Representación de la estructura de la red propuesta en la quinta aproximación

Ahora que se entiende cómo es la estructura de la nueva propuesta de red, se procede al entrenamiento de la misma.

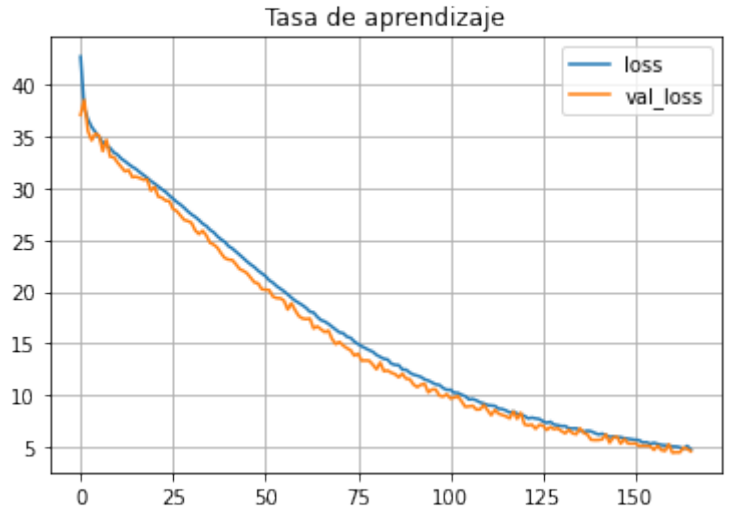
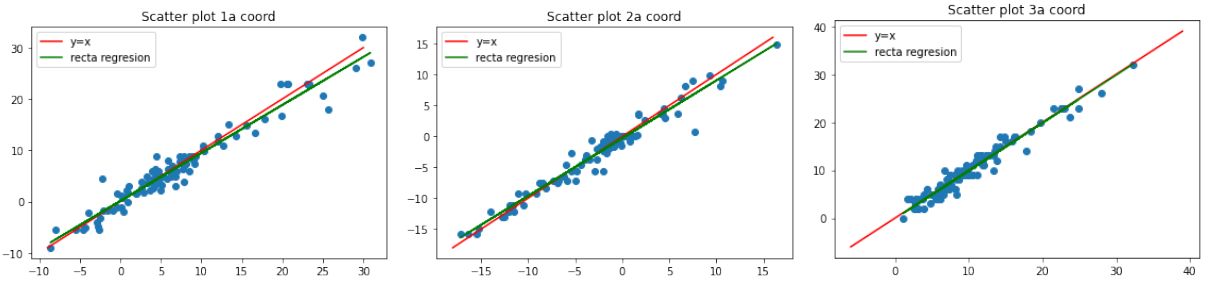


Figura 26: Tasa aprendizaje modelo multicapa versión 4, donde los ejes representan "número de epochs" frente el error medio cuadrático medido en m^2/s^2.

```
Epoch 1/2000
1889/1889 [==============================] - 255s 134ms/step - loss: 42.6888 - val_loss: 37.0576
Epoch 2/2000
1889/1889 [==============================] - 198s 105ms/step - loss: 37.7187 - val_loss: 38.4979
Epoch 3/2000
1889/1889 [==============================] - 167s 88ms/step - loss: 36.6395 - val_loss: 35.5026
Epoch 4/2000
1889/1889 [==============================] - 165s 88ms/step - loss: 35.8977 - val_loss: 34.6103
Epoch 5/2000
1889/1889 [==============================] - 161s 85ms/step - loss: 35.4169 - val_loss: 35.2562
Epoch 6/2000
1889/1889 [==============================] - 1252s 663ms/step - loss: 34.9239 - val_loss: 35.0593
Epoch 7/2000
1889/1889 [==============================] - 182s 96ms/step - loss: 34.4185 - val_loss: 33.5684
Epoch 8/2000
1889/1889 [==============================] - 186s 98ms/step - loss: 34.1137 - val_loss: 34.6500
Epoch 9/2000
1889/1889 [==============================] - 182s 97ms/step - loss: 33.8650 - val_loss: 33.0335
Epoch 10/2000
1889/1889 [==============================] - 180s 95ms/step - loss: 33.4450 - val_loss: 32.9782
Epoch 11/2000
1889/1889 [==============================] - 181s 96ms/step - loss: 33.2190 - val_loss: 32.4670
Epoch 12/2000
1889/1889 [==============================] - 177s 94ms/step - loss: 32.8425 - val_loss: 32.0251
Epoch 13/2000
...
Epoch 165/2000
1889/1889 [==============================] - 177s 94ms/step - loss: 5.1208 - val_loss: 4.8202
Epoch 166/2000
1889/1889 [==============================] - 177s 94ms/step - loss: 4.7627 - val_loss: 4.6091
```

Figura 27: Proceso de aprendizaje del modelo multicapa versión 4

El proceso de entrenamiento es largo, porque, aunque se ha reducido la dimensión de cada una de las redes, hay 3 para entrenar en vez de una sola, siendo hasta ahora la red que más tiempo ha tardado en entrenar de las que se han construido hasta este punto. Pero, a cambio, se obtienen los siguientes resultados, los cuales se pueden analizar viendo las gráficas de las nubes de puntos y el scoring del modelo para evaluar el poder predictivo que obtenido.



(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 28: Nube de puntos de la predicción del modelo multicapa versión 4. Ambos ejes de cada gráfica se miden en m/s.

```
Scoring del modelo con respecto al MSE: [4.46 , 4.46 ]
```

Figura 29: Scoring del modelo multicapa versión 4

Se puede observar en las Figuras 28 y 29, que el modelo obtenido consigue un poder predictivo comparable con el del modelo 2, el cual únicamente estaba entrenado para los datos de Madrid. Aún con el coste computacional que tiene entrenarlo, es un modelo con suficiente potencial como para ser considerado un buen modelo como solución final para el problema.

3.6. Sexta aproximación: Red implícita

Una práctica muy común en redes neuronales es utilizar una red preentrenada para una tarea similar a la que se quiere resolver y adaptarla. Sabiendo esto, para el siguiente modelo se intenta utilizar alguna de las redes utilizadas anteriormente para poder crear un modelo que proporciona mejor solución al propio problema que se está abordando.

Para poder utilizarlos, se usa la idea de los métodos numéricos llamados de predicción-corrección. Muchos métodos numéricos consisten en una solución con una ecuación implícita, la cual es de la forma $x = f(x, y, z, ...)$, es decir, que se necesita la solución para poder encontrar la solución. Para poder resolver esto, lo que se suele hacer es utilizar un método explícito, el cual no requiere la solución, más débil, con el que encontrar una aproximación de la solución $x' = g(u, v, ...)$, y se utiliza esta aproximación x' como dato de entrada de la ecuación implícita, consiguiendo así un valor aproximado de la solución real $x \simeq f(x', y, z, ...) = f(g(u, v, ...), y, z, ...)$.

Con esta idea en mente, se pueden utilizar los modelos de red anteriores como método explícito, el cual hace una predicción inicial de la solución, que luego es uno de los datos de entrada de la red "implícita", de la cual se puede ver una representación de su estructura en la Figura 30.

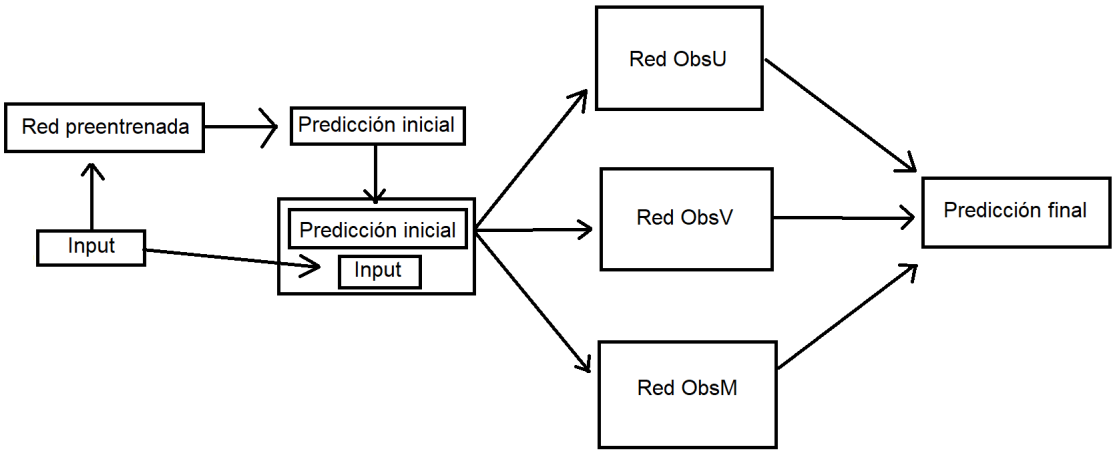


Figura 30: Representación de la estructura de la red propuesta en la sexta aproximación

En general, se podría hacer con cualquiera de los modelos que se han entrenado para el conjunto de datos que se va a utilizar, ya que usar alguno de los modelos que solo han sido entrenados para Madrid podría ser contraproducente porque no están especializadas para el mismo problema exactamente.

Para este caso, se ha utilizado para la predicción inicial la red de la cuarta aproximación, aunque

podría ser de cualquiera. Se ha optado por esta porque tarda menos en dar una predicción que la quinta aproximación, pero los resultados que se obtienen de ella son aceptables. Se crea una copia de esta y configurándola de tal manera que no se puedan modificar los valores de esta red, ya que esta red ya está entrenada. Con todo esto en mente, se procede a entrenar la red.

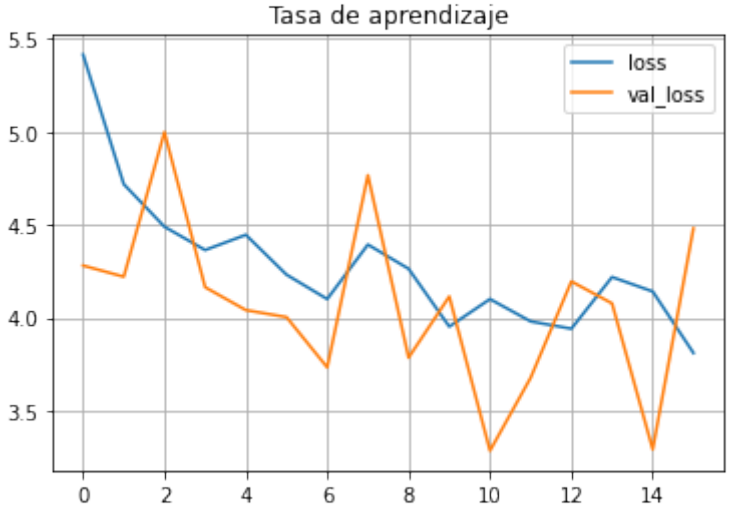
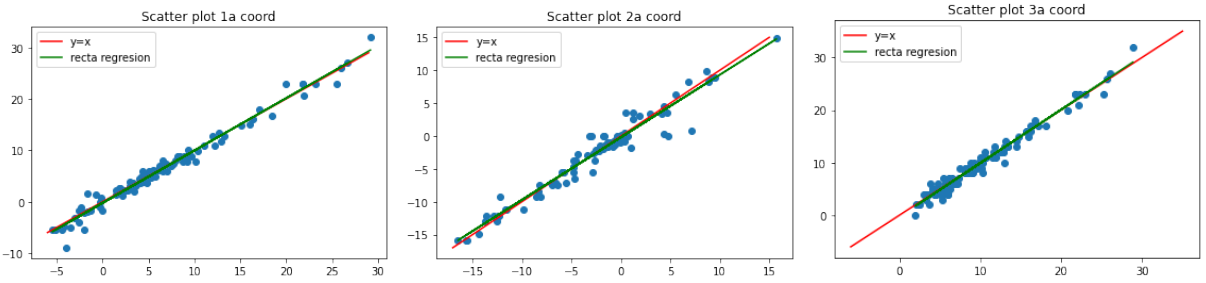


Figura 31: Tasa aprendizaje modelo multicapa implícito, donde los ejes representan "número de epochs" frente el error medio cuadrático medido en m^2/s^2.

```
Epoch 1/2000
1890/1890 [==============================] - 283s 149ms/step - loss: 5.4148 - val_loss: 4.2805
Epoch 2/2000
1890/1890 [==============================] - 284s 150ms/step - loss: 4.7181 - val_loss: 4.2205
Epoch 3/2000
1890/1890 [==============================] - 287s 152ms/step - loss: 4.4894 - val_loss: 4.9980
Epoch 4/2000
1890/1890 [==============================] - 287s 152ms/step - loss: 4.3639 - val_loss: 4.1640
Epoch 5/2000
1890/1890 [==============================] - 285s 151ms/step - loss: 4.4457 - val_loss: 4.0409
Epoch 6/2000
1890/1890 [==============================] - 288s 152ms/step - loss: 4.2315 - val_loss: 4.0031
Epoch 7/2000
1890/1890 [==============================] - 288s 153ms/step - loss: 4.1003 - val_loss: 3.7330
Epoch 8/2000
1890/1890 [==============================] - 288s 152ms/step - loss: 4.3933 - val_loss: 4.7662
Epoch 9/2000
1890/1890 [==============================] - 289s 153ms/step - loss: 4.2631 - val_loss: 3.7854
Epoch 10/2000
1890/1890 [==============================] - 287s 152ms/step - loss: 3.9521 - val_loss: 4.1143
Epoch 11/2000
1890/1890 [==============================] - 283s 150ms/step - loss: 4.1001 - val_loss: 3.2849
Epoch 12/2000
...
Epoch 15/2000
1890/1890 [==============================] - 283s 150ms/step - loss: 4.1411 - val_loss: 3.2922
Epoch 16/2000
1890/1890 [==============================] - 283s 150ms/step - loss: 3.8106 - val_loss: 4.4802
```

Figura 32: Proceso de aprendizaje del modelo multicapa implícito

Se puede ver en la figura 32, que el número de iteraciones que hace en el aprendizaje (epochs) es bastante menor que en el resto de modelos, esto se debe a que como se ha introducido un modelo ya preentrenado, solamente tiene que hacer una corrección de la predicción. A continuación se puede ver cómo son los resultados.



(a) Análisis predicción "ObsU" (b) Análisis predicción "ObsV" (c) Análisis predicción "ObsM"

Figura 33: Nube de puntos de la predicción del modelo multicapa implícito. Ambos ejes de cada gráfica se miden en m/s.

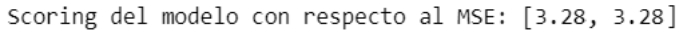


Figura 34: Scoring del modelo multicapa implícito

Aun necesitando mucho menos entrenamiento, los resultados son ligeramente mejores que los obtenidos en los modelos anteriores, obteniendo con esta red un error medio cuadrático alrededor de $3m^2/s^2$, como se ve en la Figura 34, siendo $4m^2/s^2$ el mejor resultado obtenido hasta ahora. Puede que esto no sea una diferencia tan grande como la que había entre el error del perceptrón y el error de la primera red multicapa, pero sigue siendo un muy buen avance, ya que cada vez es más costoso ir reduciendo el error. Hay que tener en cuenta que este es un modelo para hacer una predicción de un suceso caótico, que no tiene una solución explícita y con esto conseguimos una aproximación de la solución real.

4. Conclusiones

En este trabajo se ha explicado el origen de las redes neuronales, su funcionamiento y varios ejemplos de uso en un caso real para distintas geometrías de redes neuronales secuenciales, aplicados al problema de la predicción del viento.

Con el modelo del perceptrón aplicado al conjunto de datos de Madrid se ha podido confirmar que el problema realmente no se podría resolver con una simple función lineal, corroborando nuestra necesidad de crear modelos más complejos. A continuación, con los siguientes modelos multicapa se ha visto que con un número reducido de capas 8-5 ya se puede hacer una predicción realmente buena, pero como solo se ha aplicado al conjunto de datos de Madrid, al intentar aplicarlo a otras zonas realmente se ve que no es tan útil. Esto se puede razonar ya que, en general, en Madrid las condiciones del viento son más calmadas en comparación con otros sitios como puede ser Tarifa o Barcelona, incluso pudiendo influir la geografía de los distintos sitios aunque no conste como parámetro de entrada. Por lo tanto, surge la necesidad de tener que entrenar con una muestra de todos los datos disponibles.

Al comenzar a entrenar con los datos para un modelo que ya producía buenos resultados para

Madrid, se ha visto que se pierde poder predictivo, por lo tanto generando así la necesidad de crear modelos más potentes. En vez de continuar con modelos puramente secuenciales, se ha optado por dividir el problema, para así poder reducir el coste de entrenamiento, obteniendo una mejora en los resultados, pero no llegando al mismo nivel que con el mejor modelo que se ha entrenado para únicamente el conjunto de datos de Madrid. Finalmente, inpirándose en los métodos numéricos se ha creado una red, que utilizaba la predicción hecha ya por un modelo preentrenado, para poder corregir con los datos de entrada y la predicción mencionada anteriormente. Esta última red ha proporcionado resultados favorables.

Ya con todo esto podemos concluir que las redes neuronales son una poderosa herramienta, muy versátil y moldeable, a la que se le puede aplicar estrategias muy distintas para conseguir resultados precisos para problemas tan complejos como puede ser un sistema dinámico no lineal, que es el caso de este trabajo.

5. Anexos

5.1. Anexo 1: Código completo utilizado

A continuación, se adjunta el código utilizado en un archivo .py por si se quieren replicar los resultados, probablemente no salgan exactamente los mismos resultados numéricos, ya que depende de la máquina utilizada y que las inicializaciones de los modelos se hacen de forma aleatoria. Para ejecutarlo se ha utilizado Visual Studio Code, en el cual se puede ejecutar el código como si fuese un jupyter notebook, en el cual los bloques están separados por "#%%".

```
#%%
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split, KFold
from tensorflow import keras
import numpy as np
import pandas as pd
from sklearn import linear_model
import matplotlib.pyplot as plt

#%%
######Preparación de los datos#########
data = pd.read_table(r"C:\Users\Administrador\Documents\cosas_universidad\TFG\
    Viento\Datos_viento_Madrid.txt" , header=0 , sep = "\t")
data = data.dropna() #Quitamos los Nan

scaler=Normalizer() #Vamos a noramlizar los datos de los predictores
X = data.drop(['fecha','obsU','obsV','obsM'], axis=1)
Xscaler = scaler.fit_transform(X)
Xnorm = pd.DataFrame(Xscaler,  columns=['predU10m','predV10m','predM10m','
    predU150m','predV150m','predM150m','predMes','predHora','predMesU','predMesV
    ','predHoraU','predHoraV','predDia','predDiaU','predDiaV'])

Y=data.drop(['fecha','predU10m','predV10m','predM10m','predU150m','predV150m','
    predM150m','predMes','predHora','predMesU','predMesV','predHoraU','predHoraV
    ','predDia','predDiaU','predDiaV'], axis=1)

#%%Funcion para visualizar resultados
def graficas_result(modelo, x, y, N= 100):
```

```
    a = modelo.predict(x.iloc[0:N], verbose =0)#Veamos con 100 predicciones, cambiar por el modelo que se quiera ver
    plt.scatter(a[:,0],y.iloc[0:100,0])
    x0=[i for i in range(int(np.floor(min(a[:,0]))),int(np.ceil(max(a[:,0]))))]
    plt.plot(x0,x0,"r-", label="y=x")
    regr=linear_model.LinearRegression()
    regr.fit(a[:,0].reshape(-1,1), y.iloc[0:100,0])
    plt.plot(a[:,0],regr.predict(a[:,0].reshape(-1,1)), "g-",label="recta regresion")
    plt.title("Scatter plot 1a coord")
    plt.legend()
    plt.show()

    plt.scatter(a[:,1],y.iloc[0:100,1])
    x0=[i for i in range(int(np.floor(min(a[:,1]))),int(np.ceil(max(a[:,1]))))]
    plt.plot(x0,x0,"r-", label="y=x")
    regr=linear_model.LinearRegression()
    regr.fit(a[:,1].reshape(-1,1), y.iloc[0:100,1])
    plt.plot(a[:,1],regr.predict(a[:,1].reshape(-1,1)),"g-",label="recta regresion")
    plt.title("Scatter plot 2a coord")
    plt.legend()
    plt.show()

    plt.scatter(a[:,2],y.iloc[0:100,2])
    x0=[i for i in range(int(np.floor(min(a[:,2])))-7,int(np.ceil(max(a[:,2])))+7)]
    plt.plot(x0,x0,"r-", label="y=x")
    regr=linear_model.LinearRegression()
    regr.fit(a[:,2].reshape(-1,1), y.iloc[0:100,2])
    plt.plot(a[:,2],regr.predict(a[:,2].reshape(-1,1)),"g-",label="recta regresion")
    plt.title("Scatter plot 3a coord")
    plt.legend()
    plt.show()

#%%
#Vamos a ver primero el modelo del perceptrón
```

```
fold_no=1
compare_scores= []
mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro

kf= KFold(n_splits =5, shuffle= True ,random_state=8)
for train, valid in kf.split(Xnorm, Y):

    inputdataperceptron = keras.layers.Input(shape=(15,))
    outputdataperceptron = keras.layers.Dense(3, activation = "linear")(
    inputdataperceptron)


    modelopred = keras.models.Model(inputs=[inputdataperceptron], outputs=[
    outputdataperceptron])

    modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
    Adam(learning_rate=.001))
    callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
    mode="min" ,patience=5, restore_best_weights=True)

    history= modelopred.fit(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(train
    [-1]+1)], epochs=2000,
    validation_data=(Xnorm[valid[0]:(valid[-1]+1)], Y[valid[0]:(valid[-1]+1)]),
    callbacks=[callback], verbose=1)

    pd.DataFrame(history.history).plot()
    plt.grid(True)
    plt.gca() # set the vertical range to [0-1]
    plt.title("Tasa de aprendizaje")
    plt.show()
    scores = [modelopred.evaluate(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(
    train[-1]+1)], verbose=0), modelopred.evaluate(Xnorm[valid[0]:(valid[-1]+1)
    ], Y[valid[0]:(valid[-1]+1)])]
    if scores[1] < mejor[1]:
        modeloperceptron = modelopred
        mejor = scores
        validf = valid
    print(f'Score for fold {fold_no}: loss of {scores}')
```

```
87      fold_no+=1
88
89  graficas_result(modeloperceptron, Xnorm[validf[0]:(validf[-1]+1)], Y[validf[0]:(
        validf[-1]+1)])
90  print(f"Scoring del modelo con respecto al MSE: {mejor}")
91  modeloperceptron.save(r'C:\Users\Administrador\Documents\cosas_universidad\TFG\
        modelos\modelo_2a_coord_general.h5')
92
93  # %%
94  #Veamos un modelo multicapa
95
96  fold_no=1
97  compare_scores= []
98  mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro
99
100 kf= KFold(n_splits =5, shuffle= True ,random_state=8)
101 for train, valid in kf.split(Xnorm, Y):
102
103     inputdataMLv1 = keras.layers.Input(shape=(15,))
104     capa1MLv1 = keras.layers.Dense(250, activation = "relu")(inputdataMLv1)
105     capa2MLv1 = keras.layers.Dense(200, activation = "relu")(capa1MLv1)
106     capa3MLv1 = keras.layers.Dense(150, activation = "relu")(capa2MLv1)
107     capa4MLv1 = keras.layers.Dense(100, activation = "relu")(capa3MLv1)
108     capa5MLv1 = keras.layers.Dense(50, activation = "relu")(capa4MLv1)
109     outputdataMLv1 = keras.layers.Dense(3, activation = "linear")(capa5MLv1)
110
111
112     modelopred = keras.models.Model(inputs=[inputdataMLv1], outputs=[
        outputdataMLv1])
113
114     modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
        Adam(learning_rate=.001))
115     callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
        mode="min" ,patience=5, restore_best_weights=True)
116
117     history= modelopred.fit(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(train
        [-1]+1)], epochs=2000,
```

```
118     validation_data=(Xnorm[valid[0]:(valid[-1]+1)], Y[valid[0]:(valid[-1]+1)]),
        callbacks=[callback], verbose=1)
119 
120     history= modelopred.fit(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(train
        [-1]+1)], epochs=2000,
121     validation_data=(Xnorm[valid[0]:(valid[-1]+1)], Y[valid[0]:(valid[-1]+1)]),
        callbacks=[callback], verbose=1)
122 
123 
124     pd.DataFrame(history.history).plot()
125     plt.grid(True)
126     plt.gca() # set the vertical range to [0-1]
127     plt.title("Tasa de aprendizaje")
128     plt.show()
129     scores = [modelopred.evaluate(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(
        train[-1]+1)], verbose=0), modelopred.evaluate(Xnorm[valid[0]:(valid[-1]+1)
        ], Y[valid[0]:(valid[-1]+1)])]
130     if scores[1] < mejor[1]:
131         modeloMLV1 = modelopred
132         mejor = scores
133         validf = valid
134     print(f'Score for fold {fold_no}: loss of {scores}')
135     fold_no+=1
136 graficas_result(modeloMLV1, Xnorm[validf[0]:(validf[-1]+1)], Y[validf[0]:(validf
        [-1]+1)])
137 print(f"Scoring del modelo con respecto al MSE: {mejor}")
138 
139 modeloMLV1.save(r'C:\Users\Administrador\Documents\cosas_universidad\TFG\modelos
        \modeloMLv1.h5')
140 
141 # %%
142 #Veamos un segundo modelo multicapa con más capas y neuronas
143 
144 fold_no=1
145 compare_scores= []
146 mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro
147 
148 kf= KFold(n_splits =5, shuffle= True ,random_state=8)
```

```
for train, valid in kf.split(Xnorm, Y):

    inputdataMLv2 = keras.layers.Input(shape=(15,))
    capa1MLv1 = keras.layers.Dense(1000, activation = "relu")(inputdataMLv2)
    capa2MLv1 = keras.layers.Dense(950, activation = "relu")(capa1MLv1)
    capa3MLv1 = keras.layers.Dense(900, activation = "relu")(capa2MLv1)
    capa4MLv1 = keras.layers.Dense(850, activation = "relu")(capa3MLv1)
    capa5MLv1 = keras.layers.Dense(800, activation = "relu")(capa4MLv1)
    capa6MLv1 = keras.layers.Dense(750, activation = "relu")(capa5MLv1)
    capa7MLv1 = keras.layers.Dense(700, activation = "relu")(capa6MLv1)
    capa8MLv1 = keras.layers.Dense(650, activation = "relu")(capa7MLv1)
    outputdataMLv2 = keras.layers.Dense(3, activation = "linear")(capa8MLv1)


    modelopred = keras.models.Model(inputs=[inputdataMLv2], outputs=[
    outputdataMLv2])

    modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
    Adam(learning_rate=.001))
    callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
    mode="min" ,patience=5, restore_best_weights=True)

    history= modelopred.fit(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(train
    [-1]+1)], epochs=2000,
    validation_data=(Xnorm[valid[0]:(valid[-1]+1)], Y[valid[0]:(valid[-1]+1)]),
    callbacks=[callback], verbose=1)


    pd.DataFrame(history.history).plot()
    plt.grid(True)
    plt.gca() # set the vertical range to [0-1]
    plt.title("Tasa de aprendizaje")
    plt.show()
    scores = [modelopred.evaluate(Xnorm[train[0]:(train[-1]+1)], Y[train[0]:(
    train[-1]+1)], verbose=0), modelopred.evaluate(Xnorm[valid[0]:(valid[-1]+1)
    ], Y[valid[0]:(valid[-1]+1)])]
    if scores[1] < mejor[1]:
        modeloMLv2 = modelopred
```

```
180         mejor = scores
181         validf = valid
182     print(f'Score for fold {fold_no}: loss of {scores}')
183     fold_no+=1
184 
185 graficas_result(modeloMLv2, Xnorm[validf[0]:(validf[-1]+1)], Y[validf[0]:(validf
    [-1]+1)])
186 print(f"Scoring del modelo con respecto al MSE: {mejor}")
187 
188 modeloMLv2.save(r'C:\Users\Administrador\Documents\cosas_universidad\TFG\modelos
    \modeloMLv2.h5')
189 
190 # %%
191 #Este ya es un modelo bastante bueno, pero solo lo hemos probado para los datos
    de Madrid veamos su rendimiento para los datos de todas las ciudades
192 data1 = pd.read_table(r"C:\Users\Administrador\Documents\cosas_universidad\TFG\
    Viento\Datos_viento_Madrid.txt" , header=0 , sep = "\t")
193 data1 = data1.dropna() #Quitamos los Nan
194 data2 = pd.read_table(r"C:\Users\Administrador\Documents\cosas_universidad\TFG\
    Viento\Datos_viento_Coruña.txt" , header=0 , sep = "\t")
195 data2 = data2.dropna()
196 data3 = pd.read_table(r"C:\Users\Administrador\Documents\cosas_universidad\TFG\
    Viento\Datos_viento_Tarifa.txt" , header=0 , sep = "\t")
197 data3 = data3.dropna()
198 data4 = pd.read_table(r"C:\Users\Administrador\Documents\cosas_universidad\TFG\
    Viento\Datos_viento_Barcelona.txt" , header=0 , sep = "\t")
199 data4 = data4.dropna()
200 
201 datag=pd.concat([data1,data2,data3,data4], ignore_index=True)
202 scaler=Normalizer() #Vamos a noramlizar los datos de los predictores
203 scaler.fit(data1.drop(['fecha','obsU','obsV','obsM'], axis=1))
204 Xg = datag.drop(['fecha','obsU','obsV','obsM'], axis=1)
205 Xscaler = scaler.transform(Xg)
206 Xnormg = pd.DataFrame(Xscaler,  columns=['predU10m','predV10m','predM10m','
    predU150m','predV150m','predM150m','predMes','predHora','predMesU','predMesV
    ','predHoraU','predHoraV','predDia','predDiaU','predDiaV'])
207 Yg=datag.drop(['fecha','predU10m','predV10m','predM10m','predU150m','predV150m',
    'predM150m','predMes','predHora','predMesU','predMesV','predHoraU','
```

```
    predHoraV','predDia','predDiaU','predDiaV'], axis=1)
208
209 xtestg, xtraing, ytestg, ytraing = train_test_split(Xnormg, Yg, test_size= .15,
    random_state = 100)
210 #%%
211 print(f"Scoring del modelo para todos los datos respecto al MSE: {modeloMLv2.
    evaluate(xtraing.iloc[0:100],ytraing.iloc[0:100] , verbose=0)}")
212 graficas_result(modeloMLv2, xtestg, ytestg)
213 #%%
214 #Modelo multicapa para todos los datos disponibles
215 #Primero tratemos los datos
216 scaler=Normalizer() #Vamos a noramlizar los datos de los predictores
217 Xg = datag.drop(['fecha','obsU','obsV','obsM'], axis=1)
218 Xscaler = scaler.fit_transform(Xg)
219 Xnormg = pd.DataFrame(Xscaler,  columns=['predU10m','predV10m','predM10m','
    predU150m','predV150m','predM150m','predMes','predHora','predMesU','predMesV
    ','predHoraU','predHoraV','predDia','predDiaU','predDiaV'])
220 Yg=datag.drop(['fecha','predU10m','predV10m','predM10m','predU150m','predV150m',
    'predM150m','predMes','predHora','predMesU','predMesV','predHoraU','
    predHoraV','predDia','predDiaU','predDiaV'], axis=1)
221
222 #%%
223 fold_no=1
224 compare_scores= []
225 mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro
226
227 kf= KFold(n_splits =5, shuffle= True ,random_state=8)
228 for train, valid in kf.split(Xnormg, Yg):
229
230     inputdataMLv3 = keras.layers.Input(shape=(15,))
231     capa1MLv3 = keras.layers.Dense(1000, activation = "relu")(inputdataMLv3)
232     capa2MLv3 = keras.layers.Dense(950, activation = "relu")(capa1MLv3)
233     capa3MLv3 = keras.layers.Dense(900, activation = "relu")(capa2MLv3)
234     capa4MLv3 = keras.layers.Dense(850, activation = "relu")(capa3MLv3)
235     capa5MLv3 = keras.layers.Dense(800, activation = "relu")(capa4MLv3)
236     capa6MLv3 = keras.layers.Dense(750, activation = "relu")(capa5MLv3)
237     capa7MLv3 = keras.layers.Dense(700, activation = "relu")(capa6MLv3)
238     capa8MLv3 = keras.layers.Dense(650, activation = "relu")(capa7MLv3)
```

```
    outputdataMLv3 = keras.layers.Dense(3, activation = "linear")(capa8MLv3)


    modelopred = keras.models.Model(inputs=[inputdataMLv3], outputs=[
    outputdataMLv3])

    modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
    Adam(learning_rate=.001))
    callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
    mode="min" ,patience=5, restore_best_weights=True)

    history= modelopred.fit(Xnormg[train[0]:(train[-1]+1)], Yg[train[0]:(train
    [-1]+1)], epochs=2000,
    validation_data=(Xnormg[valid[0]:(valid[-1]+1)], Yg[valid[0]:(valid[-1]+1)])
    , callbacks=[callback], verbose=1)


    pd.DataFrame(history.history).plot()
    plt.grid(True)
    plt.gca() # set the vertical range to [0-1]
    plt.title("Tasa de aprendizaje")
    plt.show()
    scores = [modelopred.evaluate(Xnormg[train[0]:(train[-1]+1)], Yg[train[0]:(
    train[-1]+1)], verbose=0), modelopred.evaluate(Xnormg[valid[0]:(valid[-1]+1)
    ], Yg[valid[0]:(valid[-1]+1)])]
    if scores[1] < mejor[1]:
        modeloMLv3 = modelopred
        mejor = scores
        validf= valid
    print(f'Score for fold {fold_no}: loss of {scores}')
    fold_no+=1

graficas_result(modeloMLv3, Xnormg[validf[0]:(validf[-1]+1)], Yg[validf[0]:(
    validf[-1]+1)])
print(f"Scoring del modelo con respecto al MSE: {mejor}")

modeloMLv3.save(r'C:\Users\Administrador\Documents\cosas_universidad\TFG\modelos
    \modeloMLv3.h5')
```

```
268
269 # %%
270 #Como podemos ver no conseguimos los mismos resultados que para el mismo modelo
    pero de solamente Madrid, por lo que podemos deducir que es una tarea más
    compleja
271 #Vamos a hacer ahora un modelo más especializado, ya que nuestra solución son 3
    coordenadas vamos a dividir el problema en 3. Creando una red para cada
    coordenada de la solución
272 #Ya que dividimos la solución vamos a intentar reducir la dimensión de cada una
    de las redes para que el coste computacional no sea tan grande
273 fold_no=1
274 compare_scores= []
275 mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro
276
277 kf= KFold(n_splits =5, shuffle= True ,random_state=8)
278 for train, valid in kf.split(Xnormg, Yg):
279
280     inputdataMLv4 = keras.layers.Input(shape=(15,))
281
282
283     capa1MLv41 = keras.layers.Dense(1000, activation = "relu")(inputdataMLv4)
284     capa2MLv41 = keras.layers.Dense(950, activation = "relu")(capa1MLv41)
285     capa3MLv41 = keras.layers.Dense(900, activation = "relu")(capa2MLv41)
286     capa4MLv41 = keras.layers.Dense(850, activation = "relu")(capa3MLv41)
287     capa5MLv41 = keras.layers.Dense(800, activation = "relu")(capa4MLv41)
288     outMLv41 = keras.layers.Dense(1, activation = "linear")(capa5MLv41)
289
290     capa1MLv42 = keras.layers.Dense(1000, activation = "relu")(inputdataMLv4)
291     capa2MLv42 = keras.layers.Dense(950, activation = "relu")(capa1MLv42)
292     capa3MLv42 = keras.layers.Dense(900, activation = "relu")(capa2MLv42)
293     capa4MLv42 = keras.layers.Dense(850, activation = "relu")(capa3MLv42)
294     capa5MLv42 = keras.layers.Dense(800, activation = "relu")(capa4MLv42)
295     outMLv42 = keras.layers.Dense(1, activation = "linear")(capa5MLv42)
296
297     capa1MLv43 = keras.layers.Dense(1000, activation = "relu")(inputdataMLv4)
298     capa2MLv43 = keras.layers.Dense(950, activation = "relu")(capa1MLv43)
299     capa3MLv43 = keras.layers.Dense(900, activation = "relu")(capa2MLv43)
300     capa4MLv43 = keras.layers.Dense(850, activation = "relu")(capa3MLv43)
```

```
    outMLv43 = keras.layers.Dense(1, activation = "linear")(capa4MLv43)

    outputdataMLv4 = keras.layers.concatenate([outMLv41, outMLv42, outMLv43])


    modelopred = keras.models.Model(inputs=[inputdataMLv4], outputs=[
    outputdataMLv4])

    modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
    Adam(learning_rate=.001))
    callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
    mode="min" ,patience=5, restore_best_weights=True)

    history= modelopred.fit(Xnormg[train[0]:(train[-1]+1)], Yg[train[0]:(train
    [-1]+1)], epochs=2000,
    validation_data=(Xnormg[valid[0]:(valid[-1]+1)], Yg[valid[0]:(valid[-1]+1)])
    , callbacks=[callback], verbose=1)


    pd.DataFrame(history.history).plot()
    plt.grid(True)
    plt.gca() # set the vertical range to [0-1]
    plt.title("Tasa de aprendizaje")
    plt.show()
    scores = [modelopred.evaluate(Xnormg[train[0]:(train[-1]+1)], Yg[train[0]:(
    train[-1]+1)], verbose=0), modelopred.evaluate(Xnormg[valid[0]:(valid[-1]+1)
    ], Yg[valid[0]:(valid[-1]+1)])]
    if scores[1] < mejor[1]:
        modeloMLv4 = modelopred
        mejor = scores
        validf= valid
    print(f'Score for fold {fold_no}: loss of {scores}')
    fold_no+=1

graficas_result(modeloMLv4, Xnormg[validf[0]:(validf[-1]+1)], Yg[validf[0]:(
    validf[-1]+1)])
print(f"Scoring del modelo con respecto al MSE: {mejor}")

```

```
331 modeloMLv4.save(r'C:\Users\Administrador\Documents\cosas_universidad\TFG\modelos
        \modeloMLv4.h5')
332
333 # %%
334 #Con esto hemos conseguido un Modelo bastante bueno, pero vamos a intentar hacer
         otro intento
335 #cambiando un poco la geometría del modelo, para ello vamos a juntar las ideas
        de los modelos anteriores
336 #creando un modelo que haga una predicción y usaremos los datos de esa predicció
        n para hacer una predicción
337 #real, como si fuese un método implícito
338
339 #Para ello usaré como modelo predictivo inicial uno de los modelos creados
        anteriormente, por ejemplo el modelo 3
340
341 modelo3 = keras.models.load_model(r'C:\Users\Administrador\Documents\
        cosas_universidad\TFG\modelos\modeloMLv4.h5')
342 modelo_pred_inic=keras.models.clone_model(modelo3)
343 modelo_pred_inic.set_weights(modelo3.get_weights())
344 modelo_pred_inic.trainable=False
345 #He puesto que no sea entrenable, ya que el modelo ya está entrenado para este
        conjunto de datos, y solo
346 #Queremos una aproximación a la predicción real
347
348 #ahora nos haremos el modelo de verdad, usando la estructura del modelo 4,
        reduciendo un poco el modelo
349 fold_no=1
350 compare_scores= []
351 mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro
352
353 kf= KFold(n_splits =5, shuffle= True ,random_state=8)
354 for train, valid in kf.split(Xnormg, Yg):
355
356     inputdataMLv5 = keras.layers.Input(shape=(15,))
357
358     capa_pred_inic = modelo3(inputdataMLv5)
359
360     input_MLv5 = keras.layers.concatenate([inputdataMLv5, capa_pred_inic])
```

```
361
362     capa1MLv51 = keras.layers.Dense(1000, activation = "relu")(input_MLv5)
363     capa2MLv51 = keras.layers.Dense(950, activation = "relu")(capa1MLv51)
364     capa3MLv51 = keras.layers.Dense(900, activation = "relu")(capa2MLv51)
365     capa4MLv51 = keras.layers.Dense(850, activation = "relu")(capa3MLv51)
366     outMLv51 = keras.layers.Dense(1, activation = "linear")(capa4MLv51)
367
368     capa1MLv52 = keras.layers.Dense(1000, activation = "relu")(input_MLv5)
369     capa2MLv52 = keras.layers.Dense(950, activation = "relu")(capa1MLv52)
370     capa3MLv52 = keras.layers.Dense(900, activation = "relu")(capa2MLv52)
371     capa4MLv52 = keras.layers.Dense(850, activation = "relu")(capa3MLv52)
372     outMLv52 = keras.layers.Dense(1, activation = "linear")(capa4MLv52)
373
374     capa1MLv53 = keras.layers.Dense(1000, activation = "relu")(input_MLv5)
375     capa2MLv53 = keras.layers.Dense(950, activation = "relu")(capa1MLv53)
376     capa3MLv53 = keras.layers.Dense(900, activation = "relu")(capa2MLv53)
377     capa4MLv53 = keras.layers.Dense(850, activation = "relu")(capa3MLv53)
378     outMLv53 = keras.layers.Dense(1, activation = "linear")(capa4MLv53)
379
380     outputdataMLv5 = keras.layers.concatenate([outMLv51, outMLv52, outMLv53])
381
382
383     modelopred = keras.models.Model(inputs=[inputdataMLv5], outputs=[
    outputdataMLv5])
384
385     modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
    Adam(learning_rate=.001))
386     callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
    mode="min" ,patience=5, restore_best_weights=True)
387
388     history= modelopred.fit(Xnormg[train[0]:(train[-1]+1)], Yg[train[0]:(train
    [-1]+1)], epochs=2000,
389     validation_data=(Xnormg[valid[0]:(valid[-1]+1)], Yg[valid[0]:(valid[-1]+1)])
    , callbacks=[callback], verbose=1)
390
391
392     pd.DataFrame(history.history).plot()
393     plt.grid(True)
```

```
    plt.gca() # set the vertical range to [0-1]
    plt.title("Tasa de aprendizaje")
    plt.show()
    scores = [modelopred.evaluate(Xnormg[train[0]:(train[-1]+1)], Yg[train[0]:(train[-1]+1)], verbose=0), modelopred.evaluate(Xnormg[valid[0]:(valid[-1]+1)], Yg[valid[0]:(valid[-1]+1)])]
    if scores[1] < mejor[1]:
        modeloMLv5 = modelopred
        mejor = scores
        validf= valid
    print(f'Score for fold {fold_no}: loss of {scores}')
    fold_no+=1

graficas_result(modeloMLv5, Xnormg[validf[0]:(validf[-1]+1)], Yg[validf[0]:(validf[-1]+1)])
print(f"Scoring del modelo con respecto al MSE: {mejor}")

modeloMLv5.save(r'C:\Users\Administrador\Documents\cosas_universidad\TFG\modelos\modeloMLv4.h5')
```

5.2. Anexo 2: Depuración de los datos

```
import pandas as pd
data = pd.read_table("Datos_viento_Madrid.txt" , header=0 , sep = "\t")#Ejemplo con los datos de Madrid
data = data.dropna() #Quitamos los Nan

X = data.drop(['fecha','obsU','obsV','obsM'], axis=1)
Y=data.drop(['fecha','predU10m','predV10m','predM10m','predU150m','predV150m','predM150m','predMes','predHora','predMesU','predMesV','predHoraU','predHoraV','predDia','predDiaU','predDiaV'], axis=1)
```

5.3. Anexo 3: Tratamiento de los datos

```
from sklearn.preprocessing import Normalizer
scaler=Normalizer()
X = data.drop(['fecha','obsU','obsV','obsM'], axis=1)
```

```
Xscaler = scaler.fit_transform(X)
Xnorm = pd.DataFrame(Xscaler,  columns=['predU10m','predV10m','predM10m','predU150m','predV150m','predM150m','predMes','predHora','predMesU','predMesV','predHoraU','predHoraV','predDia','predDiaU','predDiaV'])
```

5.4. Anexo 4: Función auxiliar para visualización de resultados

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model

def graficas_result(modelo, x, y, N= 100):
    a = modelo.predict(x.iloc[0:N], verbose =0)#Veamos con 100 predicciones, cambiar por el modelo que se quiera ver
    plt.scatter(a[:,0],y.iloc[0:100,0])
    x0=[i for i in range(int(np.floor(min(a[:,0]))),int(np.ceil(max(a[:,0]))))]
    plt.plot(x0,x0,"r-", label="y=x")
    regr=linear_model.LinearRegression()
    regr.fit(a[:,0].reshape(-1,1), y.iloc[0:100,0])
    plt.plot(a[:,0],regr.predict(a[:,0].reshape(-1,1)), "g-",label="recta regresion")
    plt.title("Scatter plot 1a coord")
    plt.legend()
    plt.show()

    plt.scatter(a[:,1],y.iloc[0:100,1])
    x0=[i for i in range(int(np.floor(min(a[:,1]))),int(np.ceil(max(a[:,1]))))]
    plt.plot(x0,x0,"r-", label="y=x")
    regr=linear_model.LinearRegression()
    regr.fit(a[:,1].reshape(-1,1), y.iloc[0:100,1])
    plt.plot(a[:,1],regr.predict(a[:,1].reshape(-1,1)),"g-",label="recta regresion")
    plt.title("Scatter plot 2a coord")
    plt.legend()
    plt.show()

    plt.scatter(a[:,2],y.iloc[0:100,2])
```

```
    x0=[i for i in range(int(np.floor(min(a[:,2])))-7,int(np.ceil(max(a[:,2])))
    +7)]
    plt.plot(x0,x0,"r-", label="y=x")
    regr=linear_model.LinearRegression()
    regr.fit(a[:,2].reshape(-1,1), y.iloc[0:100,2])
    plt.plot(a[:,2],regr.predict(a[:,2].reshape(-1,1)),"g-",label="recta
    regresion")
    plt.title("Scatter plot 3a coord")
    plt.legend()
    plt.show()
```

5.5. Anexo 5: Codigo para creación de los modelos

```
from sklearn.model_selection import KFold
from tensorflow import keras

fold_no=1
compare_scores= []
mejor=[0,999999999] #Voy poner un valor arbitrario que va a superar seguro

kf= KFold(n_splits =5, shuffle= True ,random_state=8)
for train, valid in kf.split(Xnorm, Y):

    inputdataperceptron = keras.layers.Input(shape=(15,))
    capaperceptron = keras.layers.Dense(50, activation = "relu")(
    inputdataperceptron)
    outputdataperceptron = keras.layers.Dense(3, activation = "linear")(
    capaperceptron)


    modelopred = keras.models.Model(inputs=[inputdataperceptron], outputs=[
    outputdataperceptron])

    modelopred.compile(loss="mean_squared_error", optimizer=keras.optimizers.
    Adam(learning_rate=.001))
    callback = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=.05,
    mode="min" ,patience=5, restore_best_weights=True)

```

```
    history= modelopred.fit(xtrain_valid[train[0]:(train[-1]+1)], ytrain_valid[
    train[0]:(train[-1]+1)], epochs=2000,
    validation_data=(xtrain_valid[valid[0]:(valid[-1]+1)], ytrain_valid[valid
    [0]:(valid[-1]+1)]), callbacks=[callback], verbose=1)

    pd.DataFrame(history.history).plot()
    plt.grid(True)
    plt.gca() # set the vertical range to [0-1]
    plt.title("Tasa de aprendizaje")
    plt.show()
    scores = [modelopred.evaluate(xtrain_valid[train[0]:(train[-1]+1)],
    ytrain_valid[train[0]:(train[-1]+1)], verbose=0),
              modelopred.evaluate(xtest, ytest)]
    if scores[1] < mejor[1]:
        modeloperceptron = modelopred
        mejor = scores
    print(f'Score for fold {fold_no}: loss of {scores}')
    fold_no+=1

graficas_result(modeloperceptron, xtest, ytest)
print(f"Scoring del modelo con respecto al MSE: {mejor}")
modeloperceptron.save('modeloperceptron.h5')
```

6. Bibliografía y fuentes

6.1. Fuentes de las figuras

1. Figura 1: https://www.researchgate.net/figure/Frank-Rosenblatt-with-his-Mark-I-perceptron fig2_345813508
2. Figura 2: https://www.significados.com/neurona/
3. Figura 3: https://en.wikipedia.org/wiki/Cerebral_cortex
4. Figura 4a: Elaboración propia
5. Figura 4b: https://uc-r.github.io/feedforward_DNN
6. Figura 5: Elaboración propia

7. Figura 6: https://www.quora.com/How-do-you-reduce-model-overfitting

8. Figuras 7-34: Elaboración propia

6.2. Bibliografía

1. Géron A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc..

2. Chollet F. (2021). Deep Learning with Python, 2nd ed. Manning.

3. Nielsen M. (2019) How the backpropagation algorithm works. Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/chap2.html

4. Basogain Olabe X. (s.f.). REDES NEURONALES ARTIFICIALES Y SUS APLICACIONES. OpenCourseWare. https://ocw.ehu.eus/pluginfile.php/40137/mod_resource/content/1/redes_neuro/contenidos/pdf/libro-del-curso.pdf

5. Jorge Matich D. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Facultad Regional Rosario. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

6. Centre National de Recherches Météorologiques. (2014). ARPEGE. Centre National de Recherches Météorologiques. http://www.umr-cnrm.fr/spip.php?article121&lang=en

7. iars.geo (2018). Breve Historia de las Redes Neuronales Artificiales (Articulo 1). Steemit. https://steemit.com/spanish/@iars.geo/breve-historias-de-las-redes-neuronales-artificia