

Creación de una Red Neuronal

Modelo de predicción del Beneficio

Profesor:

Juan José Ramos González

Autor:

Roger Villalba Fuentes

ÍNDICE

Preámbulo

- 0.1. Abreviaciones
- 0.2. Resumen

1. Objetivos

- 1.1. Tóricos
- 1.2. Prácticos

2. Cambio de Paradigma

3. Redes Neuronales

- 3.1. Introducción
- 3.2. Fundamentos
 - 3.2.1. Funciones Activación
 - 3.2.2. Función de Coste
 - 3.2.3. Descenso Gradiente
 - 3.2.4. Backpropagation

4. Estudio preliminar de los datos

5. Presentación *dataset*

- 5.1. Limpieza datos
- 5.2. Preparación de los datos
- 5.3. Separación datos *entrenamiento/test*

6. Creación modelo

- 6.1. Entrenamiento
- 6.2. Evaluación modelo
 - 6.2.1. Resultados en $f(\text{parámetros})$
 - 6.2.2. Resumen de las configuraciones

7. Presentación *Airline Profit Predictor*

8. Conclusiones

9. Bibliografía

Preámbulo

0.1. Abreviaciones

ANNs	Redes Neuronales Artificiales (<i>Artificial Neural Networks</i>)
NN	Red Neuronal (<i>Neural Network</i>)
RN	Red Neuronal
ML	Aprendizaje Automático (<i>Machine Learning</i>)
CV	Visión por Computador (<i>Computer Vision</i>)
IA	Inteligencia Artificial
DF	Data Frame (<i>Marco de datos</i>)

0.2. Resumen

Este trabajo pretende aplicar nuevas técnicas de optimización en el sector aeronáutico. En el desarrollo de este, aprenderemos a aplicar nuevas técnicas que permitan a través de los datos, prever de forma precisa el beneficio esperado para una compañía X, establecer una nueva ruta, con origen Barcelona (*BCN*) y un destino Y. Permitiendo de esta forma disponer de una herramienta basada en el uso del *Big Data* para la toma de decisiones a nivel táctico y estratégico.

Esta predicción se articulará en base a un *dataset* que presentaremos y comentaremos a lo largo de este trabajo. Una de las razones por las cuales hemos decidido crear una red neuronal es investigar y descubrir el funcionamiento de esta herramienta.

A fecha de hoy, está siendo aplicada en una gran variedad de sectores. Las aplicaciones de inteligencia artificial permiten obtener soluciones de distinta índole, según la técnica de IA que se decida emplear y en el objetivo que se quiera alcanzar (*clustering, regresión, reconocimiento de patrones,...*).

Estas técnicas permiten a los desarrolladores, *Data Scientist* y *Data Engineers*, arrojar luz sobre la incertidumbre que supone el futuro, pudiendo llegar a predecir determinados factores con un cierto nivel de confianza.

Para poder entender la finalidad de lo que queremos conseguir en este trabajo, hemos confeccionado un primer punto, donde explicamos de forma resumida, los objetivos planteados en las distintas áreas que abarca este trabajo.

En definitiva, se pretende crear un modelo predictivo que permita a una aerolínea en concreto, determinar si le resultará beneficioso y cuánto (*cuantifiable*), abrir una nueva ruta desde Barcelona a otro destino. Para lograrlo entrenaremos a una red neuronal con un conjunto de valores de forma que la predicción sea lo más fiel a la realidad posible

1. Objetivos

Los objetivos que se persiguen con la confección de este trabajo, se pueden clasificar según el ámbito. Por un lado disponemos de los objetivos teóricos, basados en comprender el alcance y los principios de las redes neuronales.

Por otro lado, disponemos de los objetivos prácticos, basados en aplicar los aspectos estudiados para confeccionar la herramienta y analizar de forma crítica y objetiva los resultados obtenidos.

1.1. Objetivos Teóricos

- Entender que es una Red Neuronal.
- Analizar los beneficios de una RN.
- Identificar los elementos que constituyen una RN.
- Comprender el algoritmo de *backpropagation*.
- Interpretar los datos obtenidos, *output* de la RN.
- Crear una guía para la creación del modelo.
- Desarrollar un pensamiento crítico para evaluar una RN.

1.2. Objetivos Prácticos

- Purgar y limpiar los datos de nuestro *dataset*.
- Mezclar datos y dividir nuestro *dataset* (*entrenamiento/test*)
- Entrenar una RN con los datos de entrenamiento.
- Desarrollar un modelo predictivo a través de *Python*.
- Evaluar los resultados obtenidos (*precisión*).
- Reevaluar el modelo y mejoras.

2. Cambio de Paradigma

Para poder entender el funcionamiento de una red neuronal, primero debemos comprender en qué se basan y cómo varían el paradigma de estudio. Hasta la fecha la programación tal y como la conocemos se basa en el cumplimiento de ciertas condiciones. Por ello la conocemos como programación imperativa, dónde en el código se infieren una serie de normas o instrucciones y la “máquina” simplemente las cumple de forma estricta.

Cómo podemos ver esto supone una gran cantidad de limitaciones. Esto es evidenciable si quisiéramos crear un programa, ya sea aplicación o página web, con la que queramos jugar al piedra, papel o tijera.

Este juego parece simple, pues sólo existen 3 posibles situaciones para el usuario, pero cuando entramos a programar este juego, como desarrolladores nos damos cuenta de que las posibles manos que debe reconocer el código, no son infinitas pero para cualquier programador cómo si lo fueran, existen gran variedad de tonalidades, gran variedad de posibles tijeras, papeles y piedras. Podríamos inducir al algoritmo a reconocer una piedra, un papel o una tijera de forma imperativa (*a través de instrucciones*) pero esto supone una ardua tarea.

Es aquí dónde el *ML (Machine Learning)*, a través de la visión por computador (*CV*), permite a la aplicación aprender a identificar una piedra, un papel o una tijera. Esto que *a priori* parece obra de magia, no es así, pues radica en el cambio de planteamiento a la hora de abordar el problema o más bien en el planteamiento de la programación del problema.

En la siguiente figura podemos ver representado de forma visual este cambio de paradigma.

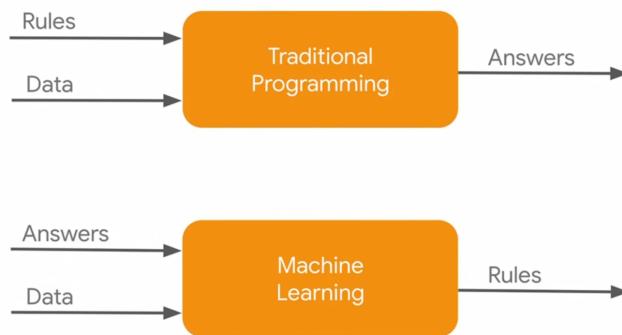


Figura 1. Representación de la diferencia entre la programación tradicional y *ML*

En primer lugar (*diagrama 1*) vemos como el programa reaccionaba en base a unos datos de entrada (*Data*) y a unas normas (*Rules*) establecidas por el desarrollador. De esta forma se podía emitir un *output (answer)* como resultado del esfuerzo computacional de pasar el *Data* a través de las normas.

En cambio en el segundo diagrama podemos observar como el *ML (campo en el que se encuentran las RN)*, el problema pasa de ser interpretado por el humano, a ser la *máquina/algoritmo/aplicación* la que debe extraer las normas a través de las cuales regir su funcionamiento a través de los resultados (*answers*) y de la información (*data*).

Volviendo al ejemplo anterior, en vez de decirle a la máquina que es una tijera, una piedra o un papel y todas las posibles variaciones que se podría encontrar, tan sólo debemos proveer de una cierta cantidad de imágenes de tijeras, papeles y piedras, cada una con su *label (etiqueta)* de forma que la máquina pueda aprender a través de un proceso de optimización (*forward propagation* y *backward propagation*) iterativo, cuales son las normas que necesita para identificar el objeto.

En este proceso iterativo, lo que realiza la máquina es asignar coeficientes a cada uno de los arcos que conectan los nodos que conforman la red, con la finalidad de aumentar su tasa de acierto, o en otras palabras, reducir su tasa de fallo.

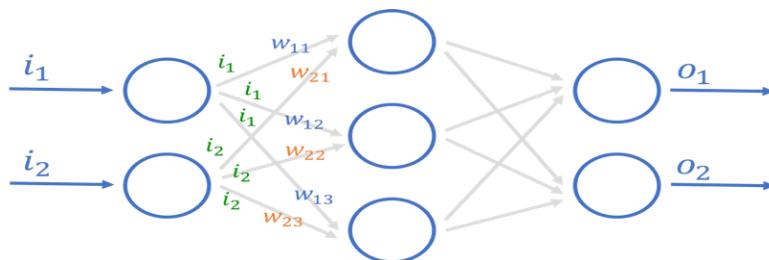
Nota: Esta forma de optimizar el algoritmo, puede variarse, pudiendo ser adaptada a la necesidad restrictiva del problema que estemos afrontando en cada caso. A continuación mostramos una imagen que permite ver representado en código esto que acabamos de comentar.

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Figura 2. Ejemplo de codificación de los parámetros para compilar el modelo

En la figura anterior podemos observar como para realizar la compilación de un modelo, la métrica que emplea este modelo es la precisión.

Esto hace referencia a cómo le inducimos al modelo el criterio de mejora, en otras palabras en base a qué parámetro debe reconfigurar los coeficientes de arco que se muestran en la siguiente figura para poder corregir los pesos en cada uno de los arcos que conforman la RN. Esto lo veremos en mayor detalle en el apartado de *backpropagation*.



$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

Figura 3. Representación visual red neuronal y coeficientes de los arcos

Ahora que ya tenemos una visión global del cambio de paradigma que supone el uso de la IA para resolver problemas, a través del aprendizaje automatizado, ya estamos listos para ubicar las redes neuronales dentro del gran campo que supone la inteligencia artificial y el ML. La siguiente figura permite esta clasificación de forma visual.

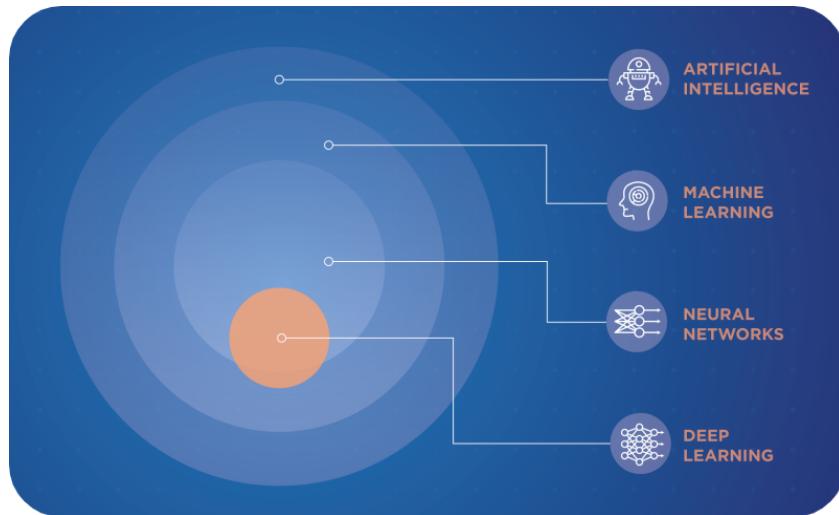


Figura 4. Clasificación de las RN en el mundo de la IA

Como se puede observar en la figura anterior, las redes neuronales se encuentran dentro de la Inteligencia Artificial (IA) y a su vez son un campo dentro del aprendizaje automático (ML).

Si buscamos una definición más teórica de que es una Red Neuronal, podríamos decir que una RN es un modelo computacional, formado por un conjunto de unidades (*nodos*) también denominados neuronas artificiales, conectadas entre sí a través de arcos, cada uno de estos arcos dispone de un coeficiente numérico, que permiten que la información atraviese la red de neuronas artificiales.

Cuando este sujeto atraviesa la RN es ponderado por estos pesos de arco, resultando en otro valor que la RN es capaz de clasificar en una etiqueta (*clustering*) o en un valor numérico (*regresión*). Este proceso nos permite además obtener un porcentaje de confianza con respecto al valor predicho.

Estos coeficiente que conforman la red neuronal, se establecen en la fase de entrenamiento, en la cual el modelo establece una serie de coeficientes (*initialmente, random*) y a medida que el modelo va iterando con todos los datos disponibles para el entrenamiento, va reconfigurando y ajustando para mejorar su capacidad predictiva.

Ahora ya disponemos de una idea general sobre las RNs y estamos listos para entender bien que son, en qué se basan y cuáles son los principios matemáticos, entorno a los cuales se articularán las redes neuronales. Así como también ver y analizar los fundamentos, ideas y características principales.

3. Redes Neuronales

3.1. Introducción

Para comprender el funcionamiento de una red neuronal es imprescindible analizar las unidades básicas que conforman una RN. Es por ello que comenzaremos con la explicación de que es una neurona. Principalmente hay que entender su funcionamiento y cómo esta interpreta la información en su interior. Una vez entendida la unidad mínima podremos proceder a entender el funcionamiento de la red. Finalmente explicaremos como una RN es capaz de aprender.

Primero debemos comprender que la neurona es la unidad básica de procesamiento de una red neuronal. Al igual que una neurona biológica, estas tienen conexiones de entrada (*inputs*) a través de las cuales la neurona realiza un cálculo interno para generar un valor de salida, *output*. En otras palabras, una neurona es una función matemática que permite a la RN generar un resultado.

Esta función matemática se basa en una suma ponderada de cada uno de los valores de entrada. Aunque no tan sólo se tiene en cuenta el valor del *input* sino también el peso de los arcos/conexiones de entrada. Este peso de cada conexión sirve para definir qué intensidad supone cada capa para el resultado final. A continuación podemos ver resumida la formulación matemática simplificada de cada uno de los “agentes” o elementos que conforman una neurona.

<i>Entrada</i>	<i>Arcos</i>	<i>Neurona</i>	<i>Salida</i>
$x_1, x_2, x_3, \dots, x_n$ $y(x_n, w_n)$	$w_1, w_2, w_3, \dots, w_n$	$w_1 * x_1 + w_2 + x_2 + w_3 * x_3 + \dots + w_n * x_n$	

Podemos observar como esta suma ponderada, que se produce en el interior de cada neurona, es comparable a un modelo matemático de regresión lineal. Podríamos resumir el funcionamiento de una neurona como una regresión lineal, ya que las variables de entrada definen una recta o un plano. Asimismo, la neurona en su interior tiene un parámetro que se denomina sesgo (*BIAS*) también conocido como término independiente.

Este valor se representa como otra conexión a la neurona. Más adelante, veremos cómo este valor se puede manipular, de hecho se manipulará en el proceso de aprendizaje (*backpropagation*). Más adelante veremos que existe un inconveniente que detallaremos en el punto (3.2.1).

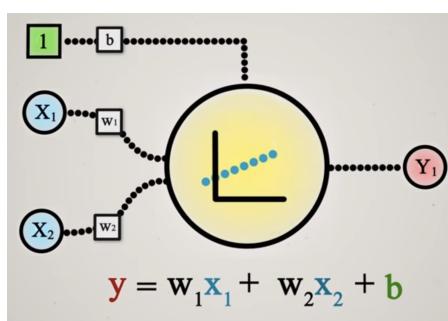


Figura 5. Simplificación cálculo de una neurona

3.2. Fundamentos

Una vez entendida la unidad básica de procesamiento estamos listos para entender que es una red neuronal.

Una RN se puede definir como un conjunto de neuronas, conectadas entre sí, distribuidas por capas, la cual permite en función de unos datos de entrada y unas etiquetas (*labels*), ser entrenada de forma que pueda proporcionarnos un *output* predictivo en función de unos datos de entrada. Según la distribución y la cantidad de las neuronas dentro de la RN, podemos crear redes más o menos complejas, según la dificultad o necesidad de cómputo del modelo. Si disponemos una o más neuronas en una misma columna, estaremos confeccionando una capa. Según la ubicación de la capa dentro de la red, encontraremos las siguientes tres capas, que se muestran de forma simplificada y visual en la siguiente figura.

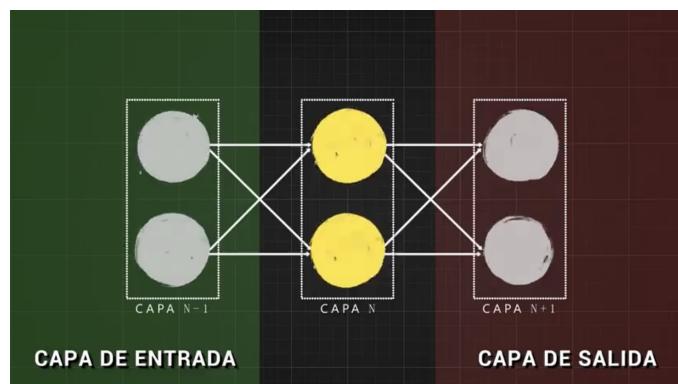


Figura 6. Representación esquemática arquitectura RN

Cada neurona de una capa, recibe los mismos valores de entrada, a estos les aplicará el cálculo de regresión que la neurona haya asignado en dicho caso. En este caso, la red dispone de 3 (*entrada, salida y capa/s oculta/s*). Para la generalización de este esquema debemos comentar que la única capa que puede variar de dimensión es la capa oculta. Pudiendo aumentar hasta n capas. De forma análoga, el número de neuronas por capa también puede variar entre $[1, n]$.

La disposición de neuronas en diferentes capas supone una gran ventaja, pues permite a la RN aprender conocimiento jerarquizado. Es decir, ordenar el conocimiento de mayor a menor importancia. Para entenderlo de forma fácil, permite a la red focalizar la atención del modelo primero en unos aspectos y luego en otros.

Aunque parezca que contra más neuronas tenga la red, mayor será su capacidad de cómputo y por tanto su predicción será más precisa, esto es falso. Debemos considerar que un modelo, es la representación lo más simplificada posible de las dinámicas de interés de un sistema, por tanto cuanto menos capas y menos neuronas, mejor será el modelo. De esta forma conseguiremos un modelo ligero, potente, escalable y generalizable.

También es cierto que cuando el conocimiento que queremos modelar es complejo, cuantas más neuronas tengas y más valores de entrada dispongas más elaborada será tu red

neuronal y podrás generar un conocimiento más complejo. Cuando generamos un modelo complejo con una gran cantidad de capas ocultas, es lo que se denomina aprendizaje profundo (*Deep Learning*).

Un aspecto que aún no hemos comentado es que si todas las capas aplican regresiones lineales, al final es como aplicar una única regresión lineal, de forma que no existiría una ventaja a la hora de emplear una RN frente a un modelo de regresión lineal.

Para evitar este fenómeno, y ayudar a la RN a representar distribuciones no lineales, o que no ajustan de forma fácil a una recta o plano uniforme, las redes neuronales disponen de un elemento que permite una manipulación no lineal y de esta forma distorsionar la regresión lineal de cada neurona.

Dicha distorsión que se produce en el interior de cada neurona recibe el nombre de función de activación. A continuación podemos observar dos figuras que representan de forma visual este fenómeno que debemos evitar cuando confeccionamos un RN.

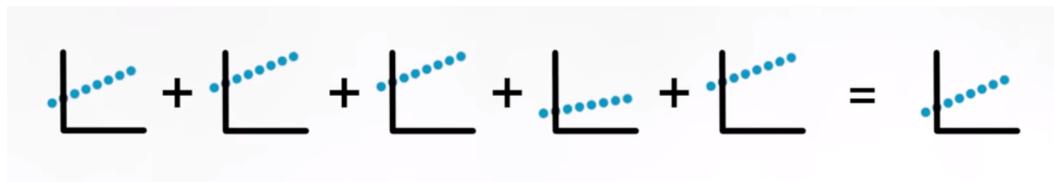


Figura XX. Clasificación de las RN en el mundo de la IA

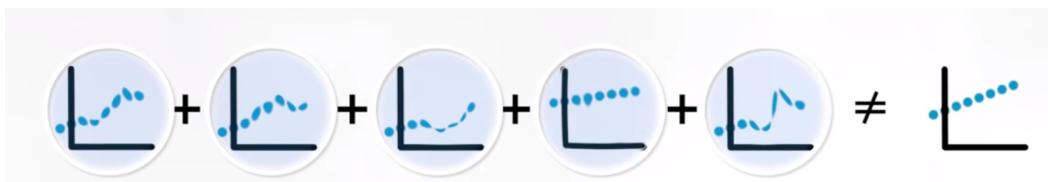


Figura 7. Clasificación de las RN en el mundo de la IA

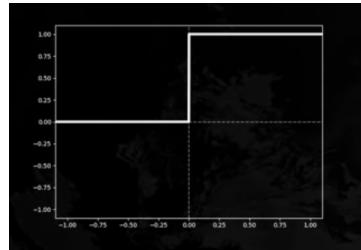
En la figura **XX** podemos observar cómo se han aplicado deformaciones no lineales sobre las rectas que generaban cada una de las neuronas. De esta forma el resultado final del cálculo de la red neuronal no tiene carácter lineal, sino que permite ir más allá y caracterizar conjuntos de datos con distribuciones más complejas.

3.2.1. Funciones Activación

El objetivo de las funciones de activación es añadir deformaciones no lineales a las rectas de regresión de las neuronas para evitar, si es necesario, la limitación lineal en los resultados de salida.

Existen cuatro tipos diferentes de deformaciones no lineales:

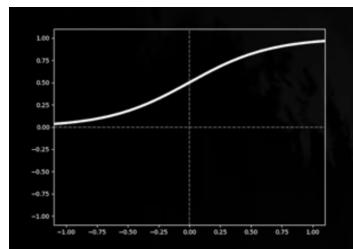
- **Función Escalonada:** Esta función recibe el nombre de escalonada debido a que el cambio no se produce de forma gradual. Dicha función fuerza a la neurona a devolver un valor entre [0, 1], siendo estos dos los únicos valores posibles.



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Un punto a tener en cuenta es que esta función no favorece a un aprendizaje profundo. El porque es fácil de ver, los comportamientos de datos simples pueden representados mediante este tipo de función, pero cuando entramos en *Deep Learning*, no podemos aplicar este tipo de función de activación, pues existen gran cantidad de valores intermedios que la función no está contemplando, limitando de forma drástica el resultado del *output* de la red.

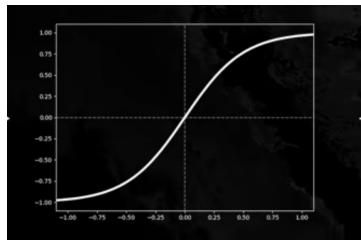
- **Función Sigmoidal:** Dicha función consiste en que los valores mayores se saturan en 1 y los valores pequeños en 0, contemplando también los valores intermedios entre el rango [0, 1]. Esta función no solo permite distorsionar el comportamiento lineal sino que es muy útil para representar probabilidades.



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

La principal característica de esta función es que el *output* viene dado en formato probabilístico, de esta forma podemos representar a través de cada neurona la probabilidad de pertenecer o no y en qué grado a un *cluster* o conjunto. Es por este motivo por lo que suele aparecer como función de activación para las últimas capas de la red. Obteniendo de esta manera la probabilidad de pertenecer y además en qué grado de confianza.

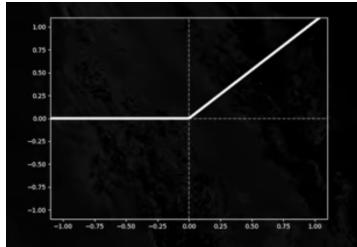
- **Función Tanh:** Esta función es similar a la anterior aunque en este caso el rango varía de -1 a 1 y está centrada en el valor 0, tal y como se puede observar a continuación.



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

Una ventaja significativa de la función Tanh respecto de la sigmoide, es que el intervalo es mayor, con lo que la distribución de los valores permite un cálculo más eficiente, pudiendo de esta forma manipular un mayor intervalo probabilístico. Este tipo de funciones, son muy útiles a la hora de distorsionar valores en las capas ocultas intermedias.

- **Función Relu:** Estas siglas significan unidad de rectificado lineal. Esta función tiene dos comportamiento distintos, cuando el valor es 0 o negativo se dibuja una recta constante, en cambio cuando su valor es mayor de 0 se convierte en una función lineal.



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

Podríamos reformular esta función como $\max(0, x)$, de esta forma nos asegurarnos de trabajar con un valor significativo por su propio valor y no limitando en un intervalo, tal y como hacen las funciones sigmoide y tanh.

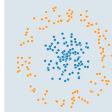
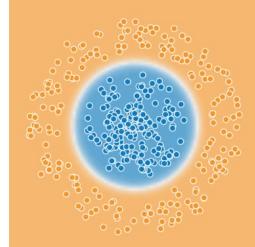
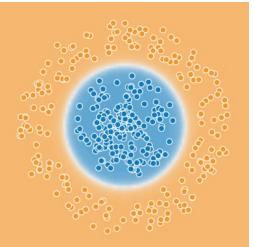
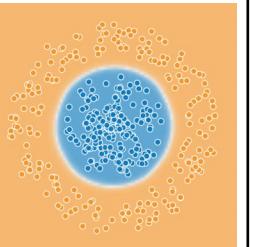
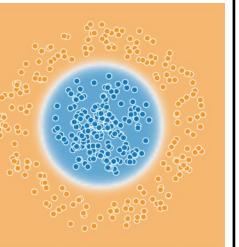
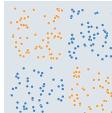
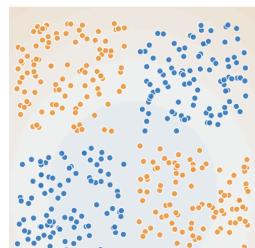
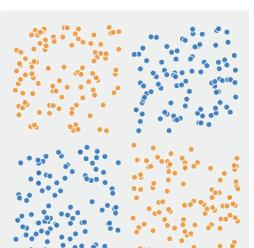
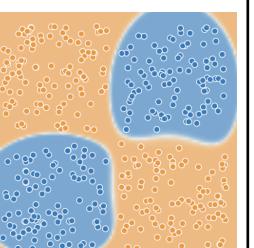
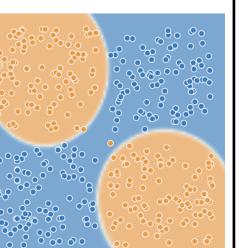
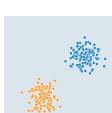
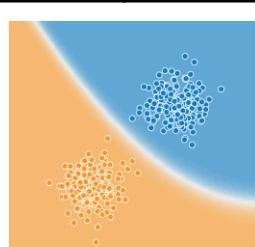
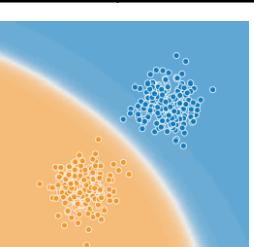
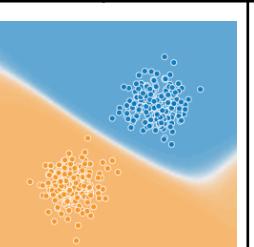
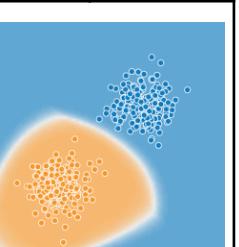
Para poder corroborar toda esta teoría y ver cual de estas funciones es más útil para la confección del modelo que queremos desarrollar. Para ello partiremos del siguiente supuesto. Vamos a modelar 4 conjuntos de datos que mostramos a continuación.

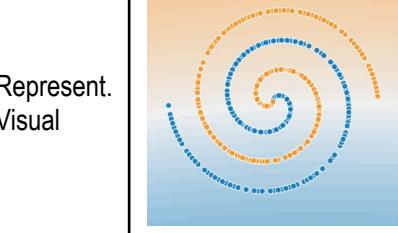
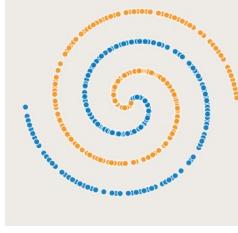
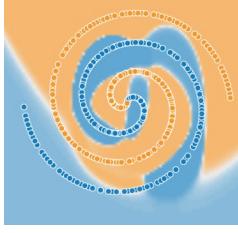
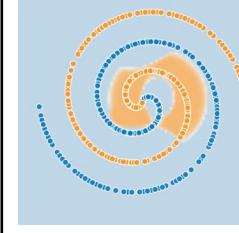
Conjunto de Datos a modelar				
-----------------------------	--	--	--	--

	Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 4
--	------------	------------	------------	------------

Ahora que sabemos que datos queremos modelar, a través de [Playground TensorFlow](#) vamos a aplicar las funciones de activación que hemos analizado y extraer de ellas los valores de *test loss* y *training loss*. Para poder estandarizar este ejercicio supondremos:

- Learning Rate: 0,03
- Epochs: 500
- Features: x_1, x_2, x_1^2, x_2^2
- Ratio train/test 80% – 20%

Función Activación	Lineal		Sigmoid		Tanh		ReLU	
	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss
	0 %	0 %	0,1 %	0,1 %	0 %	0 %	0 %	0 %
Represent. Visual								
	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss
	50 %	49 %	50 %	50 %	0 %	0 %	0,2 %	0 %
Represent. Visual								
	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss
	0 %	0 %	0,2 %	0,2 %	0 %	0 %	0 %	0 %
Represent. Visual								

	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss
	49,1 %	46,7 %	50,4 %	50 %	2,4 %	5,3 %	34,1%	33,4 %
Represent. Visual								

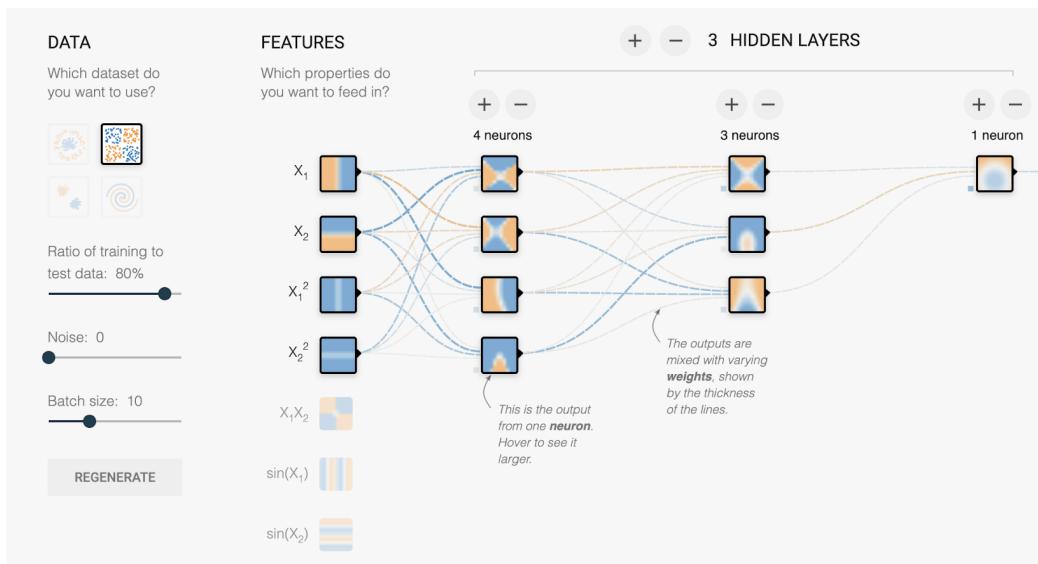
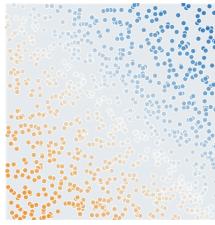
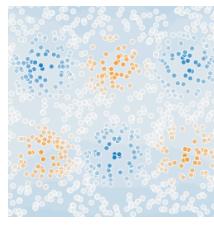


Figura 8. Estructura RN para experimentar con problemas de clasificación

Tras analizar las distintas funciones a través de *Playground TensorFlow* para problemas de clasificación podemos observar como, dependiendo de los datos (*muestras*) las funciones de activación son capaces o no de clasificar correctamente los datos. Sin embargo, en según qué situaciones la red es incapaz de clasificar los datos. Esto es debido a la estructura de la RN y a la función de activación seleccionada.

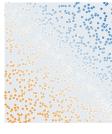
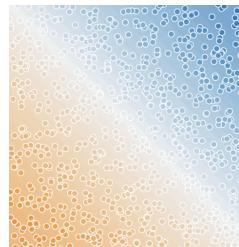
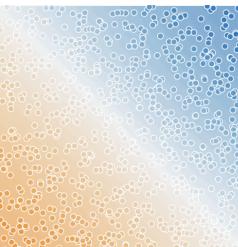
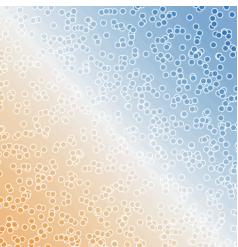
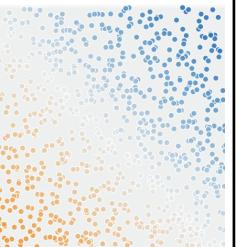
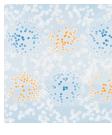
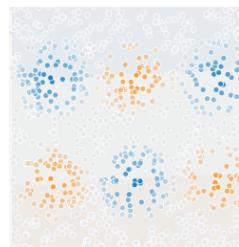
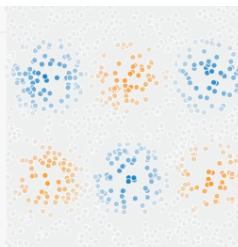
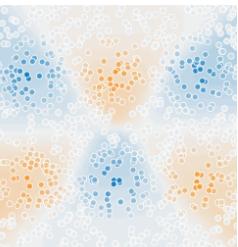
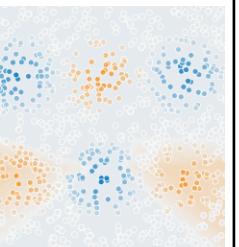
En la mayor parte de los casos de estudio las funciones de activación son capaces de clasificar de forma correcta los datos, cabe destacar que el número de capas y de neuronas se puede modificar para adecuarse al caso de estudio en concreto.

Con el fin de simplificar la comparativa y el análisis hemos empleado la misma plantilla para todos los casos y únicamente se modifica la función de activación.

Conjunto de Datos a modelar		
	Conjunto 1	Conjunto 2

Para la confección de este estudio emplearemos los siguientes parámetros para estandarizar el caso y poder así extraer conclusiones.

- Learning Rate: 0,03
- Epochs: 500
- Features: x_1, x_2
- Ratio train/test 80% – 20%

Función Activación	Lineal		Sigmoid		Tanh		ReLU	
	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss
	0 %	0 %	0 %	0 %	0,1 %	0,1 %	11,2 %	11,7 %
Represent. Visual								
	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss	Test Loss	Train Loss
	45 %	43 %	47 %	44 %	19 %	18 %	35 %	34 %
Represent. Visual								

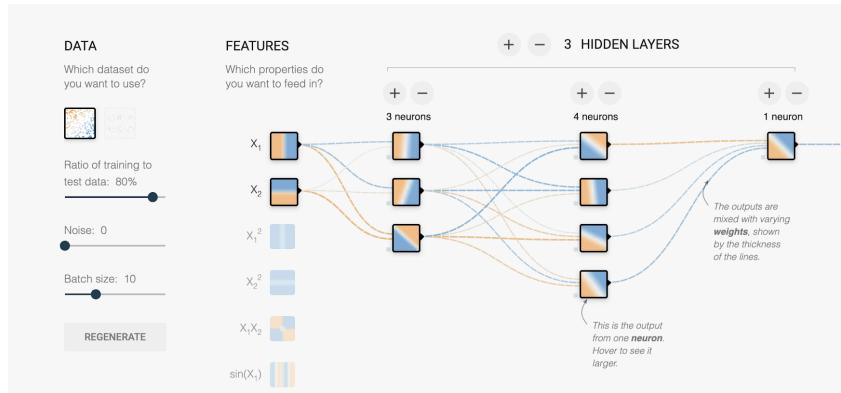


Figura 9. Estructura RN para experimentar con problemas de regresión

En este segundo caso hemos modificando la RN, variando el número de neuronas por capa.

Tras observar la tabla podemos realizar las siguientes observaciones:

1. Cada función de activación tiene un resultado distinto.
2. Dependiendo del número de capas y de neuronas por capa, la función de activación interactúa de manera distinta con cada muestra.
3. Existen muestras que requieren de una red neuronal más compleja.
4. A mayor número de *Epochs* mayor ajuste en la clasificación de los datos de la muestra. Aunque esto no significa que siempre sea así.

3.2.2. Función de Coste

Tiene como objetivo determinar un valor situado entre el estimado y el real. Asimismo, la finalidad de esta función es la de optimizar los parámetros de la red neuronal. No obstante, dicha función mide el rendimiento de la red neuronal actual, es decir la el estado de la red dentro de la *epoch* en la que se encuentra, esto se realiza a través de los valores w (*pesos de cada arco*). Para optimizar esta función existen distintos métodos, el más frecuente es el método del descenso de gradiente, *sgd* (*Stochastic Gradient Descendent*).

3.2.3. Descenso Gradiente

Pretende analizar el error cometido por cada neurona en cada iteración. Sabemos que en una red neuronal, el error se acumula y se transmite capa a capa, es decir si en una capa una neurona comete un error, este se transmite a la siguiente y así hasta llegar al *output*. Utilizando la retropropagación de errores se puede ajustar los parámetros de la red de tal forma que se minimice su desviación en la salida. Esto es posible gracias al *backpropagation*.

Para calcular el descenso de gradiente se debe realizar la derivada multivariable de la función de coste con respecto a todos los parámetros de entrada de la red. A partir de estas derivadas parciales lo que se persigue es alcanzar el mínimo global.

Si propagamos el error de una capa a la anterior, acabaremos llegando a la primera (*capa de entrada*) de esta forma se obtiene el error (*y su peso*) para cada neurona de la red. Esta estructura de retropropagación es muy eficiente ya que solamente se debe ejecutar una vez.

Una vez entendidos los fundamentos del descenso de gradiente ya estamos listos para introducir el algoritmo *backpropagation*.

3.2.4. Backpropagation

El algoritmo de *backpropagation* es el encargado de distribuir el error cometido en la capa de salida, a través de la red, concretamente entre todas las neuronas que la conforman.

Para comprender mejor el funcionamiento de este algoritmo debemos analizar los cálculos de la derivada que aplica a cada capa.

Derivada	Descripción	Fórmula Matemática
Función de activación respecto a la f (coste).	Cómputo del error de la última capa	$\delta^l = (\partial C / \partial a^l) \cdot (\partial a^l / \partial z^l)$
Función de activación con respecto a z .	Retropropagación del error a la capa anterior	$\delta^{l-1} = W^l \delta^l \cdot (\partial a^{l-1} / \partial z^l)$
Función de activación con respecto a los parámetros de la neurona.	Derivadas de la capa usando el error respecto el parámetro sesgo.	$(\partial C / \partial b^{l-1}) = \delta^{l-1}$
Función de activación con respecto a la capa previa.	Derivadas de la capa usando el error respecto el parámetro w .	$(\partial C / \partial w^{l-1}) = \delta^{l-1} a^{l-2}$

Tal y como hemos visto el algoritmo de *backpropagation* únicamente necesita 4 cálculos por capa para propagar el error desde la última capa a la primera. Este algoritmo es el que permite a la red neuronal aprender. Pues es capaz de realizar un cálculo en base a los parámetros establecidos (*coeficientes*) de cada una de las ecuaciones de regresión que conforman la RN y una vez realizado el cálculo retro propagarlo de forma que se pueda realizar una trazabilidad del error cometido y en qué nodo/neurona se ha producido e imputar el error en mayor o menor medida, según el grado de implicación de cada una las neuronas en el cálculo de dicho valor.

Esto supone un gran avance en el mundo del *ML* y de las redes neuronales, pues antes este proceso de imputar el error se hacía a través de un algoritmo de fuerza bruta, buscando todas las combinaciones posibles y finalmente, tras un largo cálculo era posible imputar el error correspondiente.

4. Estudio preliminar de los datos

Para poder desarrollar el modelo a través de una RN, debemos analizar nuestro *dataset*. Hemos creado este punto para analizar de forma previa a la creación del modelo, cuáles son las mejores variables predictoras candidatas para entrenar nuestra RN.

Como ya hemos comentado, en el interior de una neurona se realiza una regresión lineal. Para obtener una aproximación de cómo ajustan las variables que disponemos, hemos empleado la herramienta de *Minitab*. Pues nos permite crear de forma rápida rectas de regresión. Pudiendo así evaluar cada una de las variables que disponemos.

Minitab es un programa diseñado para ejecutar funciones estadísticas. Gracias a esta herramienta podemos encontrar tendencias, predecir patrones de comportamiento y descubrir relaciones entre las variables. También se pueden visualizar de forma gráfica los datos y el error cometido. La metodología que hemos empleado para poder determinar cuáles serán las variables candidatas para entrenar nuestro modelo son:

Metodología del estudio

- Seleccionar variables predictoras, iniciar el análisis con todas las variables predictoras disponibles.
- Insertarlas en *Minitab*, obtener la recta de regresión y más importante aún, las estadísticas correspondientes.
- Analizarlas y proponer nuevas variables predictoras.
- Repetir este proceso, eliminando una o dos variables predictoras ($p\text{-valor} > 0,05$).

Para poder mostrar los resultados de este estudio y entender la elección de las variables predictoras finales, empleadas para el entrenamiento, hemos confeccionado una tabla que pretende resumir los siguientes datos de *Minitab*.

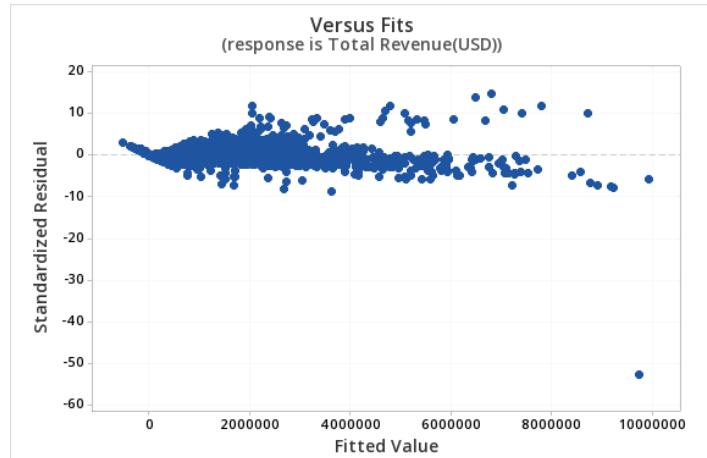
A continuación se muestra un ejemplo visual de la información suministrada por la herramienta y a través de las cuales hemos creado la tabla.

Regression Equation

```
Total Revenue(USD) = -14403 + 6,053 Distance (km) + 648,1 Airline Share + 304,4 Avg. Total Fare(USD) + 6,29 Passengers + 1184 PPDEW
+ 1,07241 Base Revenue(USD) - 371,6 Avg. Base Fare(USD)
```

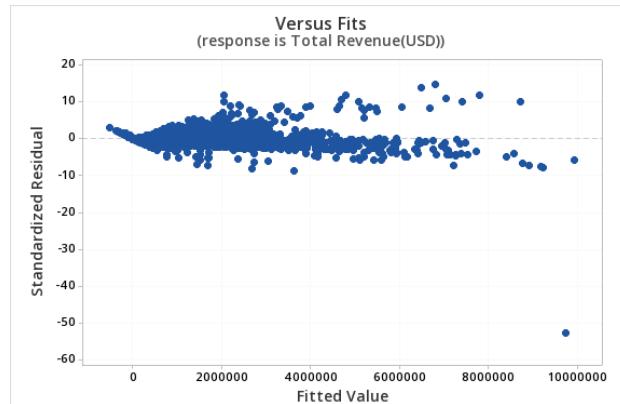
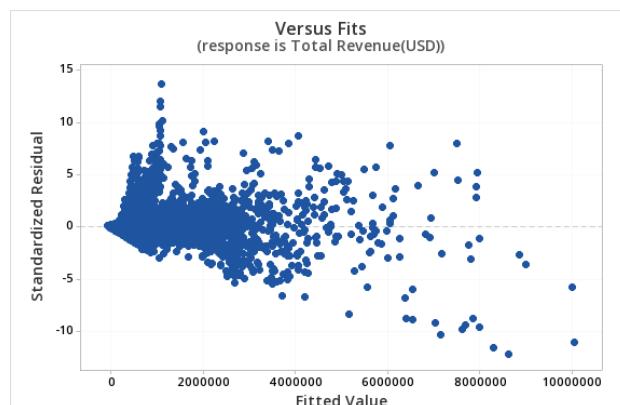
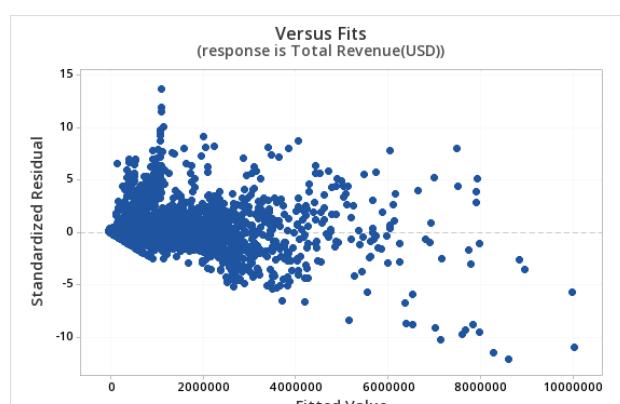
Model Summary

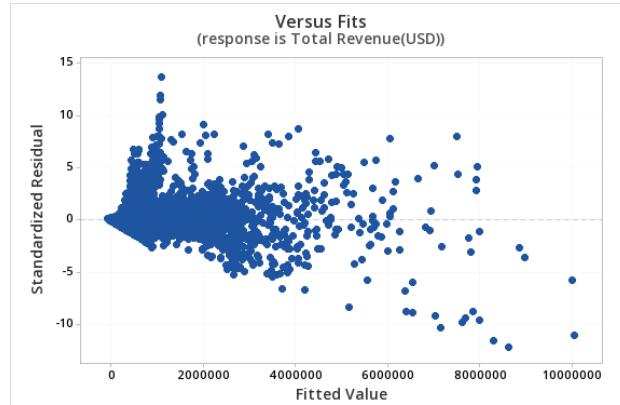
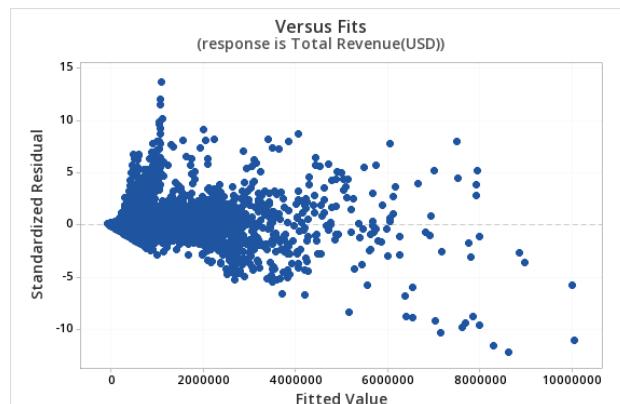
S	R-sq	R-sq(adj)	R-sq(pred)
175220	96,05%	96,05%	95,95%



En la siguiente tabla se recogen los datos recopilados de la herramienta *Minitab*. Para reducir la extensión de la tabla hemos codificado las variables predictoras:

- Distance (*D*)
- Airline Share (*AS*)
- Avg. Total Fare (*ATF*)
- Avg. Base Fare (*ABF*)
- Passengers (*P*)
- PPDEW (*PW*)
- Base Revenue (*BR*)

Iter.	Variables Predictoras	S	R-sq	R-sq (pred)	Gráfica del ajuste
1	<i>D, AS, ATF, ABF, P, PW, BR</i>	175.220	96,05 %	95,95 %	
2	<i>D, AS, ATF, ABF, PW</i>	372.126	82,18 %	82,10 %	
3	<i>D, AS, PW</i>	372.879	82,10 %	82,04 %	

4	<i>D, AS, ATF, PW</i>	372.167	82,17 %	82,10 %	
5	<i>D, AS, ABF, PW</i>	372.117	82,18 %	82,11 %	

Hemos considerado que aquellas variables con p-valor > 0,05 debían ser eliminadas.

Iteración	Var. pred	p-valor	Iteración	Var. pred	p-valor
1	<i>D</i>	0	2	<i>D</i>	0
	<i>AS</i>	0		<i>AS</i>	0
	<i>ATF</i>	0		<i>ATF</i>	0,529
	<i>ABF</i>	0		<i>ABF</i>	0,053
	<i>P</i>	0,429		<i>PW</i>	0
	<i>PW</i>	0,015			
	<i>BR</i>	0			
3	<i>D</i>	0	4	<i>D</i>	0
	<i>AS</i>	0		<i>AS</i>	0
	<i>PW</i>	0		<i>ATF</i>	0
				<i>PW</i>	0
5	<i>D</i>	0	<i>ABF</i>		0
	<i>AS</i>	0	<i>PW</i>		0

Para poder estudiar las variables predictoras del modelo hemos procurado evitar problemas de multicolinealidad con la finalidad de obtener un modelo óptimo, que incluya únicamente aquellas variables imprescindibles.

Para ello hemos confeccionado diferentes modelos, partiendo del conjunto entero de variables predictoras. En el primer modelo se observa un sobreajuste debido a la cantidad de variables, además la variable *Passengers* obtiene un p-valor de 0,429. Esto implica que esta variable no es significativa.

En el segundo modelo de regresión lineal la cantidad de variables se ha reducido con el fin de obtener un menor valor de error. A pesar de ello, observamos que en este nuevo caso existen dos variables que no son significativas para el modelo. Estas variables predictoras son *Avg. Total Fare* y *Avg. Base Fare*, destacamos que ambas son dependientes ya que el *ATF* depende del *ABF*.

Por último los modelos 3, 4 y 5 tienen unas pequeñas diferencias en lo que respecta a las variables predictoras utilizadas. Aunque, estas combinaciones no suponen que la R-Sq aumente su valor y por ende afecte al ajuste del modelo.

Ahora que ya disponemos de un mayor conocimiento de los datos, podríamos estar tentados de entrenar el modelo con aquellas variables predictoras que disponen de un *R-sq* más elevado, aunque antes de realizar la elección de qué variables debemos suministrar a nuestro modelo, debemos considerar el siguiente problema detectado.

Problema: Cada vez que se agrega un predictor a nuestro modelo este tiende a incrementar el *R-sq* (*reducir el error*), aunque esto puede deberse a un sobreajuste tal y como hemos observado en el análisis anterior sobre las variables predictoras.

Este problema afectará a la creación de nuestra RN. Pues queremos que nuestra red tenga la capacidad de generalizar. Por lo que un elevado *R-sq*, como hemos visto disminuye esta capacidad. Y queremos evitarlo a toda costa, por ello evitaremos elegir aquellas variables que ofrecen *R-sq* muy tentadoras.

***Un factor clave que no tiene en cuenta el modelo de regresión son los destinos.
En otras palabras, nuestra ecuación de regresión es global para todos los destinos
y no sería válida para la herramienta que queremos confeccionar .***

5. Presentación de los datos

5.1. Limpieza datos

Para poder crear un modelo basado en RNs, ha sido necesario procesar los datos, de forma que las librerías que empleemos puedan realizar el trabajo de forma coherente y así ahorrarnos errores al ejecutar el modelo.

Los pasos que hemos realizado para poder purgar y limpiar los datos han sido:

1. Ejecución de un *Script* en *Python* que nos permitió reemplazar todas las “/” por “-”.

```
text = open("bcn17.csv", "r")
text = ''.join([i for i in text]) \
.replace("/", "-")
x = open("processedBcn17.csv", "w")
x.writelines(text)
x.close()
```

Figura 10. Código empleado para el reemplazo

2. Uno de los primeros inconvenientes que encontramos fue que nuestro conjunto de datos disponía de una excesiva cantidad de campos (*Columns*) informativos. Algunos de los cuales no eran indispensables para nuestra RN. Procedimos a eliminar todas las muestras que tenían una o más escalas (*saltos*), de forma que nuestro modelo solo sea capaz de predecir el beneficio de una ruta directa.

Con este paso de limpieza de datos pasamos de tener 216.102 muestras a 12.524 muestras.

3. Una vez eliminados todos los vuelos con escala solo disponemos de vuelos NON-STOP por lo cual podemos eliminar la columna *Itinerary*.
4. Otros campos innecesarios fueron:
 - a. Aeropuerto de Origen (*Origin Airport*)
 - b. Nombre ciudad de Origen (*Origin City Name*)
 - c. Nombre país Origen (*Origin Country Name*)
 - d. Nombre región de Origen (*Origin Region Name*)

Todos estos campos no son de interés para nuestro modelo, pues son variables categóricas y no aportan valor al modelo.

5. Para simplificar el funcionamiento de nuestro modelo, también hemos eliminado la información de:
 - a. Nombre aeropuerto de Destino (*Destination Airport Name*)
 - b. Nombre ciudad de Destino (*Destination City Name*)
 - c. Nombre país de Destino (*Destination Country Name*)
 - d. Nombre región Destino (*Destination Region Name*)
 - e. Nombre aerolínea comercial (*Marketing Airline Name*)

Todos estos campos (*columnas*) son variables categóricas, cuya codificación ensancha de forma considerable el tamaño de nuestro modelo. Por ello y por su falta de relevancia en el cómputo, han sido eliminados.

6. Al centrarnos en vuelos directos tampoco necesitábamos los campos de las conexiones, eliminamos las 12 columnas (*4 por escala*).
7. Hemos eliminado aquellas muestras (*casos de estudio*) que no contenían información sobre alguno de los campos en los que queríamos basar nuestro estudio. Valores como `#`, `Null`, `NaN` o simplemente campos en blanco, de forma que al no disponer de todos los campos de dicha muestra se eliminó la muestra del *dataset*.
 - a. Año (*YEAR_STR*)
 - b. Mes (*Month*)
8. Para corroborar que el paso anterior fue aplicado de forma satisfactoria y que no obviamos ningún dato problemático, ejecutamos el siguiente comando para asegurarnos.

```
df = pd.read_csv('finalBCN17clean.csv',sep=';',low_memory=False)

df.isna().sum()

Destination Airport      0
Marketing Airline        0
Airline Share             0
Passengers                0
Avg. Base Fare(USD)      0
Avg. Total Fare(USD)      0
Base Revenue(USD)         0
Total Revenue(USD)         0
PPDEW                      0
Distance (km)              0
dtype: int64
```

Figura 11. Comprobación de coherencia en los datos

5.2. Preparación de los datos

A continuación presentaremos los datos a través de los cuales entrenaremos a nuestro modelo para que sea capaz de predecir el *output* fruto de este trabajo.

Para poder realizar la lectura de los datos, hemos ejecutado el siguiente código.

```
df = pd.read_csv('finalBCN17clean.csv',sep=';',low_memory=False)
```

Esta línea de código nos permite almacenar en la variable `df` (*data frame*) toda la información de nuestro *dataset*.

Para poder visualizar toda la información que contiene `df` deberíamos ejecutar el comando `df`. Sin embargo, usaremos el `df.head()` solo para tener un ejemplo y visualizar los 5 primeros *Rows* de nuestro *dataset*. Como se puede apreciar en la siguiente figura.

```
dataset.dtypes
#dataset['Destination Airport']
#dataset['Destination Airport'].unique()
df.head()
```

	Destination Airport	Marketing Airline	Airline Share	Passengers	Avg. Base Fare(USD)	Avg. Total Fare(USD)	Base Revenue(USD)	Total Revenue(USD)	PPDEW	Distance (km)
0	MAD	0B	0	1,36	52,26	74,66	71	102	0	504
1	MAD	0Z	0	1,57	46,99	69,58	74	109	0	504
2	MAD	0Z	0	6,47	62,07	84,5	402	547	0,1	504
3	MAD	2C	0	2,64	44,89	73,22	119	193	0	504
4	MAD	2C	0	4,22	43,99	67,95	186	287	0	504

Figura 12. Visualización de las 5 primeras filas

1. Para poder procesar los datos, estos no pueden ser categóricos, deben ser valores numéricos. Para salvar esta instancia hemos empleado la siguiente estrategia:

Para poder codificar la información de forma que la RN pueda interpretar de dónde procede el vuelo.

```
dest_type = CategoricalDtype(['OTP', 'IAS', 'BVA', ..., 'STR'])
```

A través de `CategoricalDtype` estamos asignando un orden concreto en el que aparecen nuestras muestras. Esto nos será de gran ayuda para los próximos pasos.

Ahora que ya disponemos de un orden concreto podremos ejecutar el siguiente comando.

```
dataset['Destination Airport'] = dataset['Destination Airport'].astype(dest_type).cat.codes
```

Gracias a esta línea de código, le estamos indicando que debe sustituir los valores de la columna *Destination Airport* ('MAD', 'BCN', etc) por valores numéricos. El orden de estos valores vienen indicado por `dest_type`.

Ahora ya disponemos de los datos de la siguiente forma:

```
dataset['Destination Airport'] = dataset['Destination Airport']+1
dataset.tail()
```

Destination Airport	Marketing Airline	Airline Share	Passengers	Avg. Base Fare(USD)	Avg. Total Fare(USD)	Base Revenue(USD)	Total Revenue(USD)	PPDEW	Distance (km)	
12519	25	ZZ	100	11,85	89,96	137,05	1066	1624	0,2	2289
12520	149	ZZ	100	1,5	68,17	125,91	102	189	0	1496
12521	25	ZZ	100	12,4	59,03	105,68	732	1311	0,2	2289
12522	450	ZZ	100	1,47	369,5	574,99	543	845	0	8810
12523	34	ZZ	100	2,66	41,05	79,48	109	212	0	842

Figura 13. Visualización del dataset

Como se puede ver en la figura anterior, ahora *Destination Airport* ya no contiene variables categóricas, sino numéricas.

Para que la red entienda qué relación existe entre esos números, pues no son cuantitativos, en el sentido, MAD (*Madrid*) por ser el 4, no es peor o mejor que STR (*Stuttgart*) siendo este el 453. Y tampoco debe la RN interpretar que la relación entre MAD-STR es de 4:453.

Por ello crearemos tantas columnas como aeropuertos de destino tengamos, en este caso 453. El siguiente comando nos permite evitar esta problemática, de forma que en cada *Row*, tan sólo habrá una variable destino con valor 1.0.

```
dataset['OTP'] = (destination==1)*1.0
dataset['IAS'] = (destination==2)*1.0
dataset['BVA'] = (destination==3)*1.0
dataset['MAD'] = (destination==4)*1.0
...
... (x 453 Líneas 1 x Destino)
```

Lo mismo sucede con la columna *Marketing Airline* (compañía aérea). Para la cual aplicamos el mismo procedimiento.

```
airline_type = CategoricalDtype(['0B','0Z','2B',...,'ZZ'])
```

Ahora procedemos a asignar un valor numérico a cada uno de los códigos referentes a cada una de las aerolíneas a través de:

```
dataset['Marketing Airline'] = dataset['Marketing Airline'].astype(airline_type).cat.codes
dataset['Marketing Airline'] = dataset['Marketing Airline']+1
dataset.head()
```

Gracias a esta línea de código, le estamos indicando que debe sustituir los valores de la columna *Marketing Airline* ('VY', 'UX', etc) por valores numéricos. El orden de estos valores vienen indicado por *airline_type*.

Ahora nuestro *dataset* contiene la siguiente información:

Marketing Airline	Airline Share	Passengers	Avg. Base Fare(USD)	Avg. Total Fare(USD)	Base Revenue(USD)	Total Revenue(USD)	PPDEW	Distance (km)	OTP	IAS	BVA	MAD	PNA	TNG	NNE	SXF
0	1	0	1,36	52,26	74,66	71	102	0	504	0,0	0,0	0,0	1,0	0,0	0,0	0,0
1	2	0	1,57	46,99	69,58	74	109	0	504	0,0	0,0	0,0	1,0	0,0	0,0	0,0
2	2	0	6,47	62,07	84,5	402	547	0,1	504	0,0	0,0	0,0	1,0	0,0	0,0	0,0
3	4	0	2,64	44,89	73,22	119	193	0	504	0,0	0,0	0,0	1,0	0,0	0,0	0,0
4	4	0	4,22	43,99	67,95	186	287	0	504	0,0	0,0	0,0	1,0	0,0	0,0	0,0

Figura 14. Visualización del dataset

Ahora debemos proceder a repetir el proceso anterior para evitar que la RN genere relaciones numéricas entre las aerolíneas. Para ello crearemos 216 columnas, una para cada aerolínea. Esto se ha realizado a través del comando.

```
dataset['0B'] = (airline==1)*1.0
dataset['0Z'] = (airline==2)*1.0
dataset['2B'] = (airline==3)*1.0
dataset['2C'] = (airline==4)*1.0
...
... (x 216 Líneas 1 x Compañía aérea)
```

Tras haber finalizado el paso 1, disponemos de una matriz de *200 rows x 674 columns*.

2. Para poder crear el modelo también fue necesario atender al tipo de variables que contenían las columnas. Y comprobar que los valores eran de tipo *float*.

A continuación podemos ver a través del comando mostrado visualizar los últimos valores de nuestro dataset. Debemos considerar que hemos omitido las columnas referentes a destino y compañía aérea, pues acabamos de asignarles valores *float*.

```
dataset.tail()
```

Airline Share	Passengers	Avg. Base Fare(USD)	Avg. Total Fare(USD)	Base Revenue(USD)	Total Revenue(USD)	PPDEW	Distance (km)
12519	100	11,85	89,96	137,05	1066	1624	0,2
12520	100	1,5	68,17	125,91	102	189	0
12521	100	12,4	59,03	105,68	732	1311	0,2
12522	100	1,47	369,5	574,99	543	845	0
12523	100	2,66	41,05	79,48	109	212	0

Figura 15. Visualización del tipo de variable que contiene el dataset

Si analizamos el tipo de dato que contienen estas variables a través de `dataset.dtype` podemos observar:

Airline Share	object
Passengers	object
Avg. Base Fare(USD)	object
Avg. Total Fare(USD)	object
Total Revenue(USD)	int64
PPDEW	object
Distance (km)	int64
OTP	float64
IAS	float64
BVA	float64
MAD	float64
PNA	float64
TNG	float64
NNE	float64
SXF	float64
YJV	float64

Figura 16. Tipo de valores de cada column

En la figura anterior se puede corroborar que no todos los datos son del tipo *float (decimal)*, de forma que esto puede ocasionar problemas en la creación de la RN.

Pues como ya hemos visto la RN se basa en aplicar cálculos matemáticos sobre cada uno de los arcos que componen la red.

Para poder solventarlo hemos aplicado el siguiente código para poder transformar estos objetos (*la mayor parte de tipo String*) a variables de tipo *float*.

```
dataset = dataset.replace('.',' ', regex=True).replace(',','.',
regex=True).astype(float)
```

Si analizamos ahora el contenido de nuestro dataset veremos:

```
dataset.tail()
```

	Airline Share	Passengers	Avg. Base Fare(USD)	Avg. Total Fare(USD)	Base Revenue(USD)	Total Revenue(USD)	PPDEW	Distance (km)
12519	100.0	11.85	89.96	137.05	1066.0	1624.0	0.2	2289.0
12520	100.0	1.50	68.17	125.91	102.0	189.0	0.0	1496.0
12521	100.0	12.40	59.03	105.68	732.0	1311.0	0.2	2289.0
12522	100.0	1.47	369.50	574.99	543.0	845.0	0.0	8810.0
12523	100.0	2.66	41.05	79.48	109.0	212.0	0.0	842.0

Figura 17. Visualización de todos los valores formato *float*

Ahora ya disponemos de las variables en el formato deseado, para poder comprobarlo volveremos a ejecutar el comando `.dataset.dtype`.

Airline Share	<code>float64</code>
Passengers	<code>float64</code>
Avg. Base Fare(USD)	<code>float64</code>
Avg. Total Fare(USD)	<code>float64</code>
Total Revenue(USD)	<code>float64</code>
PPDEW	<code>float64</code>
Distance (km)	<code>float64</code>
OTP	<code>float64</code>
IAS	<code>float64</code>
BVA	<code>float64</code>

Figura 18. Comprobación general del contenido de cada *Column*

5.3. Separación datos *entrenamiento/test*

El siguiente paso para crear el modelo consiste en separar los datos. Una parte de ellos (80%) los emplearemos para el entrenamiento y el 20% restante se usarán para evaluar la capacidad de nuestro modelo para generalizar. De esta forma estaremos por un lado validando el modelo y a la vez identificando si existen problemas de *Overfitting*.

Para poder separar los datos hemos utilizado:

```
train_dataset = dataset.sample(frac=0.8, random_state=0)

test_dataset = dataset.drop(train_dataset.index)
```

Como podemos observar `frac=0.8` representa la selección del 80% de las muestras para el *dataset* de entrenamiento.

Una vez hemos separados los datos, podemos ejecutar algunas líneas que nos permitan visualizar la distribución de los datos y ver como estos se comportan y se relacionan entre ellos.

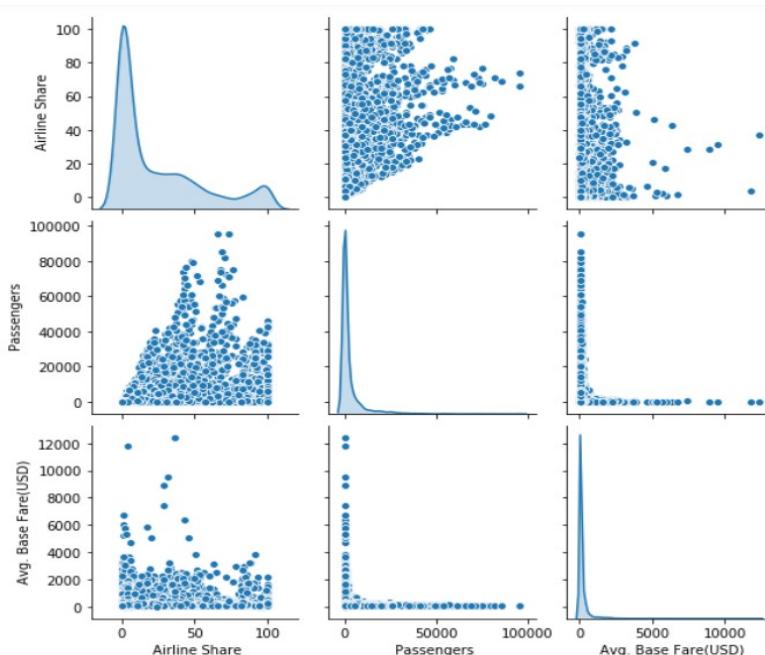


Figura 19. *Airline Share, Passengers. Avg. Base Fare*

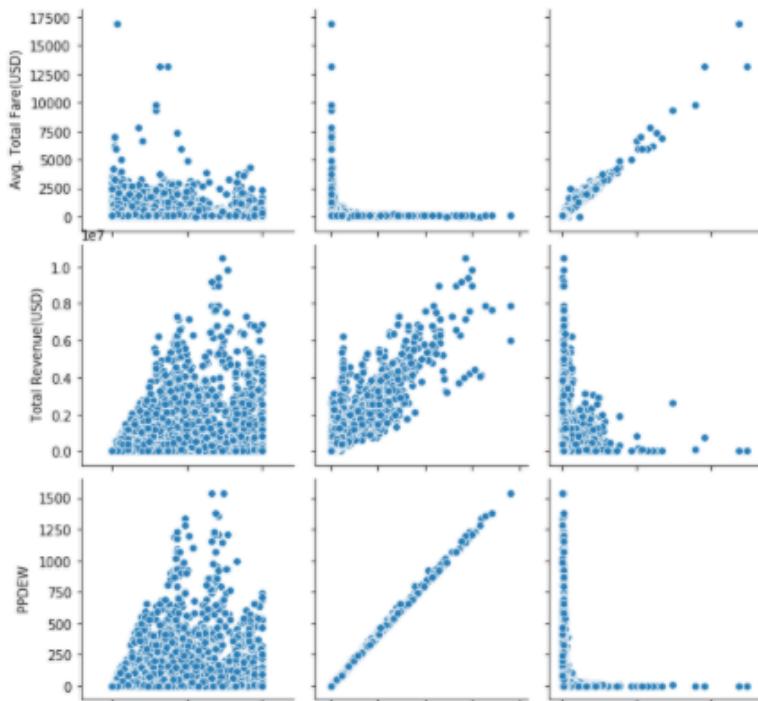


Figura 20. Avg. Total Fare, Total Revenue, PPDEW

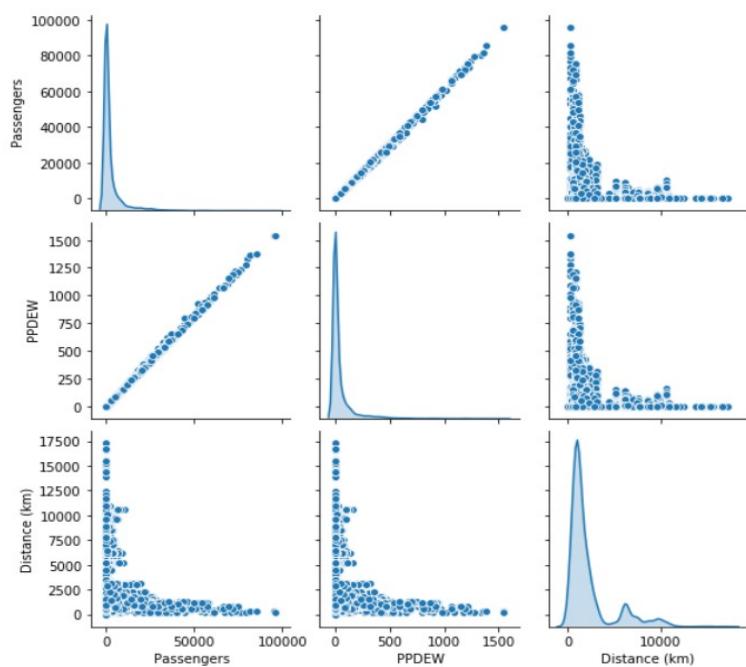


Figura 21. Passengers, PPDEW, Distance

Estas gráficas se han podido generar a través del comando:

```
sns.pairplot(train_dataset[[' Airline Share', ' Passengers', ' Avg. Base
    Fare(USD)', ' Avg. Total Fare(USD)', ' Base Revenue(USD)', ' Total
    Revenue(USD)', ' PPDEW', 'Distance (km)']],diag_kind="kde")
```

Ahora procederemos a visualizar datos estadísticos sobre cada una de nuestras columnas, datos como: *media (mean)*, *desviación estándar (std)*, *valor mínimo (min)*, *valor máximo (max)*.

	count	mean	std	min	25%	50%	75%	max
Airline Share	10019.0	24.435612	30.271543	0.00	0.300	8.82	40.450	100.00
Passengers	10019.0	2998.437941	7502.725539	0.15	7.285	145.09	2386.335	95646.26
Avg. Base Fare(USD)	10019.0	208.650433	434.827075	4.90	66.990	96.13	167.720	12380.23
Avg. Total Fare(USD)	10019.0	283.745092	530.458192	9.45	102.680	137.04	238.805	16942.07
Base Revenue(USD)	10019.0	284686.979239	650702.688212	12.00	910.000	22973.00	261556.000	8597890.00
PPDEW	10019.0	49.203952	123.001208	0.00	0.100	2.40	38.900	1542.70
Distance (km)	10019.0	2225.600958	2432.845769	0.00	852.000	1245.00	2258.000	17278.00

Figura 22. Datos estadísticos de cada Column

Como podemos observar en la figura anterior, los datos tienen una gran disparidad, para poder facilitar al modelo la comprensión de los datos debemos realizar un proceso de normalización de los datos. Para ello creamos una función que reciba nuestro *dataset* de entrenamiento y de test y los convierta en *normed data (información normalizada)*.

En nuestro caso, hemos estudiado el comportamiento de las RN cuando los datos están normalizados y cuando no, pudiendo de esta forma comprender cómo varía el *performance (rendimiento)* del modelo.

```
ASMean = dataset[' Airline Share'].mean()
ASStd = dataset[' Airline Share'].std()
dataset[' Airline Share'] = (dataset[' Airline Share'] - ASMean)/ASStd
```

De forma análoga hemos realizado lo mismo con todas las variables con las que finalmente realizaremos el entrenamiento de la RN. A continuación mostramos el *dataset* tras este proceso de normalización.

Airline Share	Passengers	Avg. Base Fare(USD)	Avg. Total Fare(USD)	Total Revenue(USD)	PPDEW	Distance (km)
12519	2.492156	-0.398665	-0.271475	-0.277370	1624.0	-0.398933
12520	2.492156	-0.400063	-0.321795	-0.298665	189.0	-0.400580
12521	2.492156	-0.398591	-0.342902	-0.337337	1311.0	-0.398933
12522	2.492156	-0.400067	0.374069	0.559794	845.0	-0.400580
12523	2.492156	-0.399906	-0.384423	-0.387421	212.0	-0.400580

Figura 23. Datos normalizados

Ahora que disponemos de los datos preparados ya estamos listos para construir el modelo, entrenarlo y finalmente evaluarlo.

Nota: Para poder normalizar los datos, tuvimos que realizar este paso antes de insertar las variables de destino y compañía aérea

6. Creación Modelo

Ahora que ya hemos realizado un estudio de las variables disponibles, sabemos cuales son las mejores candidatas para el modelo y hemos manipulado y dispuesto los datos de forma óptima, estamos listos para crear nuestra RN, entrenarla y evaluarla.

El primer paso para crear la RN es definir una función en python que lo construya. Para ello hemos empleado la librería *Keras* para facilitarnos el proceso. A continuación mostramos la función *build_model()*:

```
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation=tf.nn.relu, input_shape=[len(train_dataset.keys())]),
        layers.Dense(64, activation=tf.nn.relu),
        layers.Dense(1)
    ])

    optimizer = tf.keras.optimizers.RMSprop(0.001)

    model.compile(loss='mse',
                  optimizer=optimizer,
                  metrics=['mae', 'mse'])
    return model
```

Como podemos observar la declaración del modelo consiste en definir el número de capas de la RN, en nuestro caso 3 (*layers*), las dos primeras capas con 64 neuronas (*nodos*) y funciones de activación del tipo *relu*. Finalmente añadimos una capa para que la red pueda unificar los datos antes de mostrarnos el *output*.

El optimizador empleado es *RMSprop (Root Mean Square Propagation)* y la compilación del modelo se realizará a través de una función de pérdida (*función de coste*) basada en '*mse*' (*mean square error*) y las métricas empleadas serán el '*mse*' y '*mae*' (*mean absolute error*).

Luego debemos almacenar el modelo en una variable que en nuestro caso denominaremos *model*. Si ejecutamos el comando *model.summary()* podremos observar un resumen del modelo que vamos a emplear.

Layer (type)	Output Shape	Param #
<hr/>		
dense (Dense)	(None, 64)	43136
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 1)	65
<hr/>		
Total params:	47,361	
Trainable params:	47,361	
Non-trainable params:	0	

Llegados a este punto estamos a punto de iniciar el entrenamiento pero un paso extra que podemos realizar para comprobar que hemos creado bien el modelo, es suministrarle unos

datos y ver si es capaz de mostrar una predicción, claro está que la predicción será errónea, pues aún no hemos entrenado a la RN pero de esta forma nos aseguramos que el modelo es capaz de aceptar un *input* y generar un *output*. A continuación mostramos una captura que permite evidenciar cómo nuestro modelo es capaz de generar *outputs*.

```
example_batch = train_dataset[:10]
example_result = model.predict(example_batch)
example_result

array([[-34.752117],
       [-18.22887],
       [-245.76122],
       [-85.411835],
       [-319.4567],
       [-66.639496],
       [-17.099953],
       [-80.331985],
       [-13.435909],
       [-192.06342]], dtype=float32)
```

Ahora ya estamos listos para iniciar el entrenamiento de nuestro modelo a través de los datos que hemos preparado previamente.

En concreto:

- *train_dataset* (*datos entren.*)
- *train_layers* (*etiquetas entren.*)
- *test_dataset* (*datos test*)
- *test_layers* (*etiquetas test*)

6.1. Entrenamiento

En esta fase le suministraremos al modelo que hemos creado los datos de entrenamiento (*train_dataset*) para que pueda de esta forma ajustar los parámetros de cada una de las neuronas que conforman la red. Esta fase es la más larga, en cuanto a tiempo de ejecución, debido a la forma en que las RNs aprenden (*forward y backpropagation*).

A veces este entrenamiento puede durar días, incluso semanas. Aunque este no es nuestro caso. Una técnica o más bien práctica utilizada es ayudarse de una función que nos permita saber si el modelo está entrenando o por algún motivo la compilación se ha detenido.

Para ello hemos creado una función simple de forma que cada 100 iteraciones (*epochs*) printeé un punto (.) por pantalla. A continuación podemos ver una imagen que muestra cómo la red neuronal está realizando el entrenamiento.

```
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch%100==0: print('')
        print('.', end='')

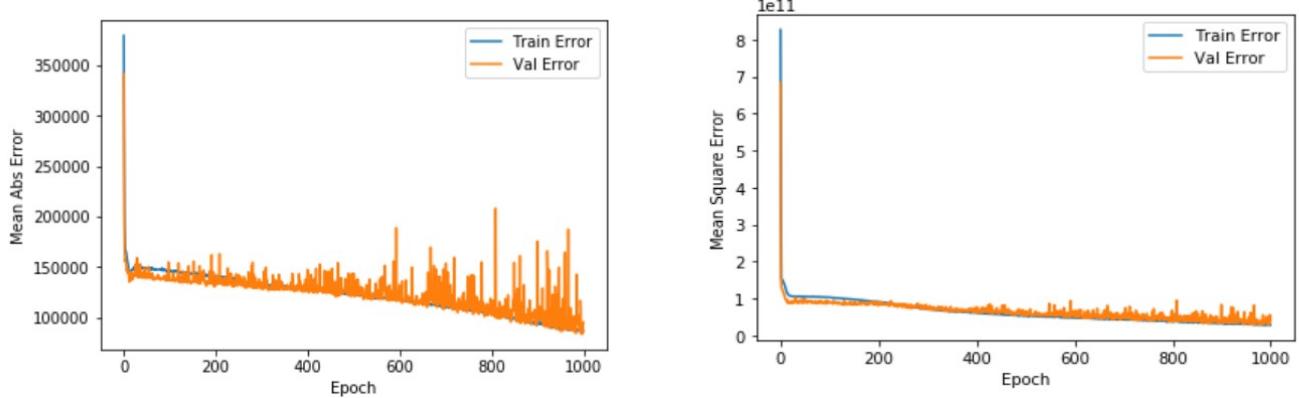
EPOCHS = 1000

history = model.fit(
    train_dataset, train_labels,
    epochs=EPOCHS, validation_split=0.2, verbose=0,
    callbacks=[PrintDot()])
```

.....
.....
.....
.....
.....

Una vez disponemos del modelo entrenado hemos creado una función llamada `plot_history(history)`, a través de la cual podemos generar unas gráficas para visualizar la evolución (*rendimiento*) a lo largo de las iteraciones (*epochs*) que ha ido realizando el modelo.

En nuestro caso, tras 1.000 *epochs* hemos obtenido:



Como se puede observar existe un error considerable en la evolución a través de la métrica *mae* en comparación con la de *mse*.

La razón por la cual aparecen tantos picos en las gráficas es porque no hemos normalizado la información con la que hemos entrenado el modelo por lo que el error que estamos contemplando no es un valor que oscila entre valores pequeños sino que como podemos ver en el eje Y del primer gráfico, los valores oscilan entre (100.000, 350.000).

Llegados a este punto ya podemos afirmar que hemos confeccionado una red neuronal capaz de predecir el beneficio esperado por una compañía que desea abrir una ruta de BCN a un destino X.

Para poder dotar de mayor rigor al informe y poder extraer conclusiones relevantes sobre la aproximación práctica de la confección de una red neuronal dedicaremos el siguiente apartado (6.2. *Evaluación del Modelo*) a manipular los parámetros de la red neuronal, a través de variar:

- Datos sin normalizar
 - Separación de los datos de validación (*dentro del dataset de entrenamiento*).
 - Indicar a la red neuronal que finalice su entrenamiento de forma anticipada.
- Datos normalizados
 - Separación de los datos de validación (*dentro del dataset de entrenamiento*).
 - Indicar a la red neuronal que finalice su entrenamiento de forma anticipada.
 - Modificar el número de capas del modelo, número de neuronas por capa y aplicar diferentes funciones de activación.

6.2. Evaluación Modelo

6.2.1. Resultados en f (parámetros)

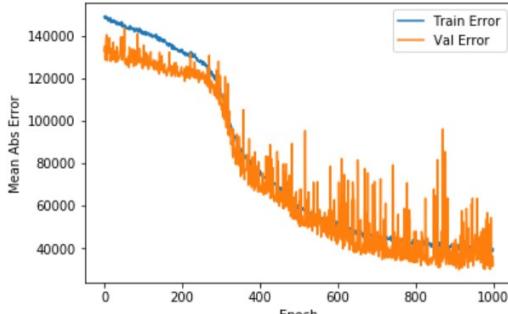
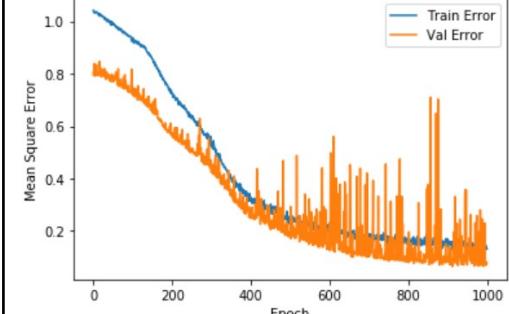
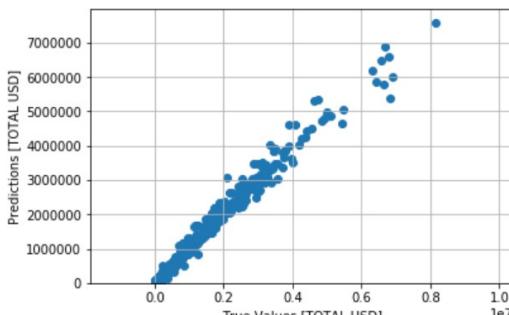
A continuación presentaremos diferentes tablas en las que se tunean los parámetros del modelo y veremos cómo varía la predictibilidad del modelo, pudiendo identificar cuando nuestra RN es capaz de realizar buenas predicciones y cuando no. Permitiéndonos de esta forma encontrar el mejor modelo para nuestro caso de estudio en concreto.

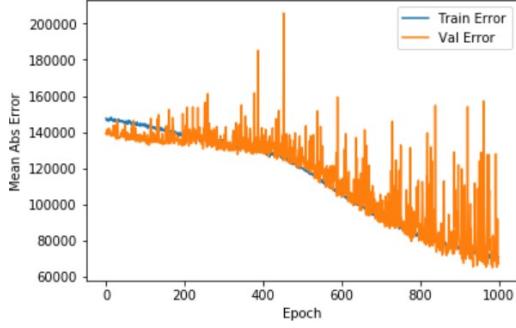
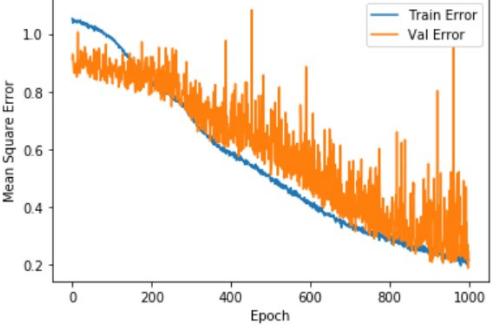
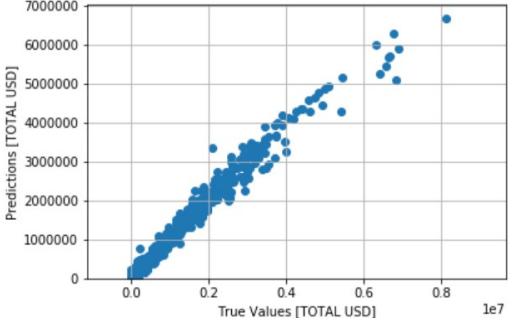
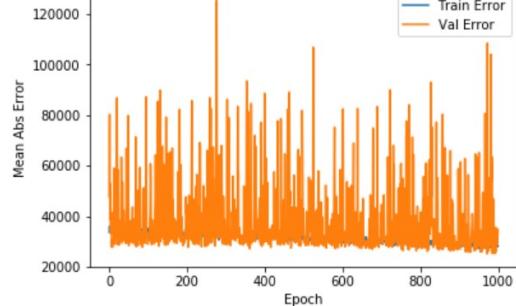
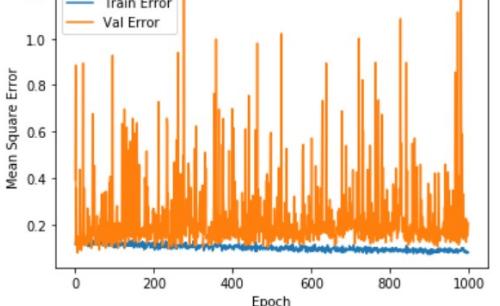
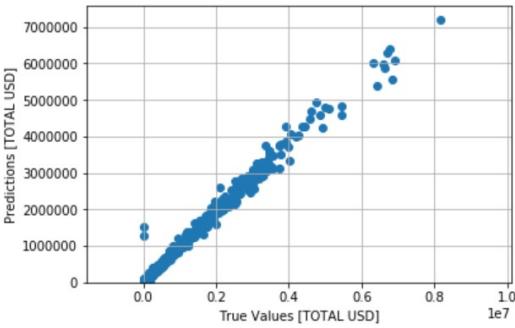
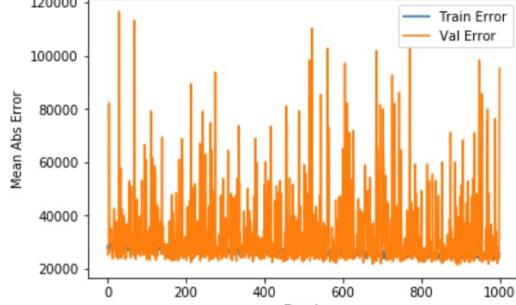
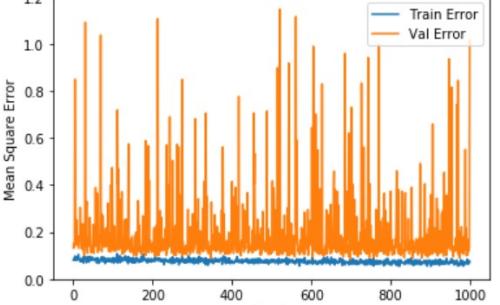
En la siguiente tabla presentamos el comportamiento de la red en su entrenamiento acorde a la modificación del tamaño del *validation split* (*partición dentro de los datos de entrenamiento*) y cómo se comporta la variable que queremos predecir, es decir cual es el error asociado de la red neuronal en base a los parámetros especificados.

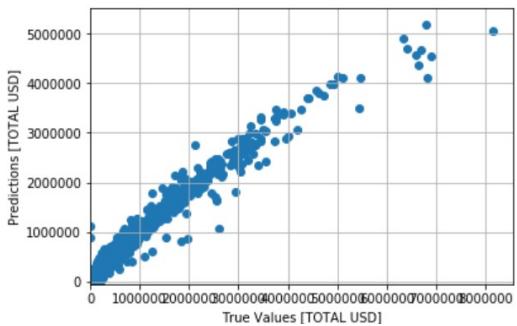
Para reducir las dimensiones de la tabla, en ella hemos considerado los siguientes parámetros:

(Objetivo: Separación de los datos de validación)

- **Estado de la información (datos entrenamiento):** No normalizados
- **Nº capas:** 3
- **Configuración (neuronas/capa):** 64 (*relu*), 64 (*relu*), 1

Pred.	Valid. split	Total Revenue	Mean Absolute Error	Mean Squared Error
AS, P, ATF, ABF, PW	5 %	33.357,55	 <p>Mean Abs Error</p> <p>Epoch</p>	 <p>Mean Square Error</p> <p>Epoch</p>
Prediction Plot		 <p>Predictions [TOTAL USD]</p> <p>True Values [TOTAL USD]</p>		

AS, P, ATF, ABF, PW	20 %	62.377,24	 
Prediction Plot			
AS, P, ATF, ABF, PW	30%	30.343,55	 
Prediction Plot			
AS, P, ATF, ABF, PW	45%	88.649,29	 

<i>Prediction Plot</i>			
Units	%	USD	<i>plot</i>

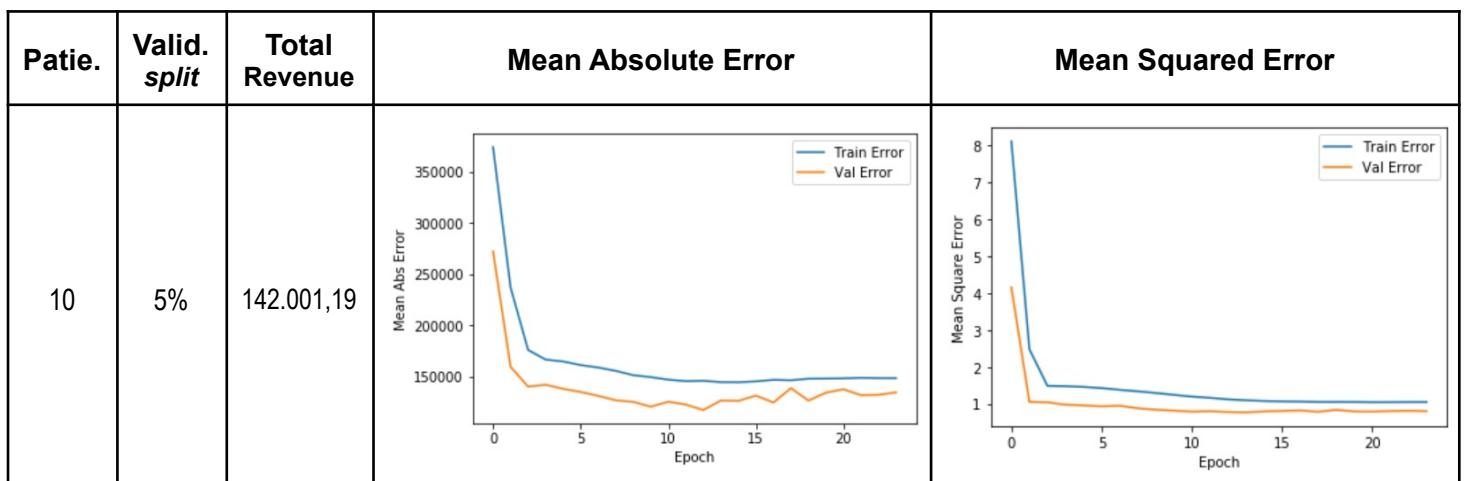
Ahora que conocemos el comportamiento y el error al que está sujeto la red si modificamos los parámetros del conjunto de datos *train-val* es decir en la separación del conjunto de datos incluido en *train_dataset*, vemos como las gráficas anteriores empiezan a despuntar en reiteradas ocasiones pero que al inicio del entrenamiento parecen, en general ajustar bien.

Para evitar esta problemática hemos empleado una función de keras que permite al modelo identificar a partir de qué punto estamos sobre ajustando el modelo. Si aplicamos *keras.callbacks.EarlyStopping()* sobre la variable *model*, podemos indicarle a la red a través del parámetro *patience*, cuantas *epochs* debe esperar hasta finalizar el entrenamiento.

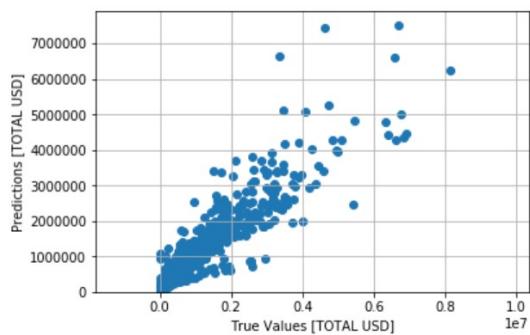
En este caso la información considerada:

(Objetivo: finalizar el entrenamiento de forma anticipada)

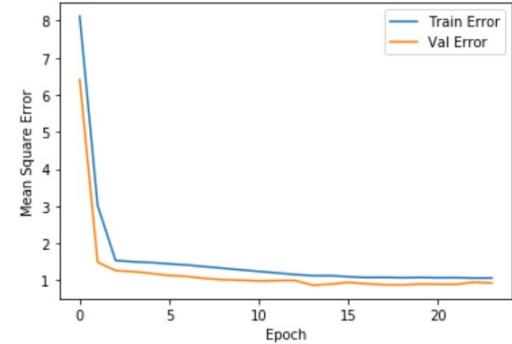
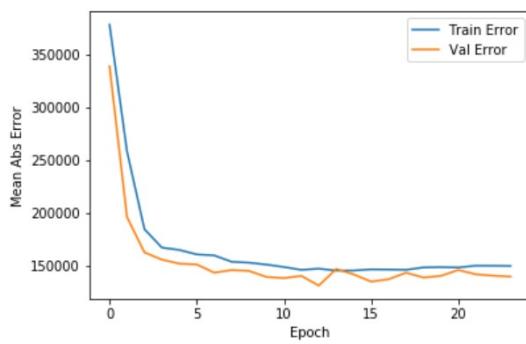
- **Estado de la información (datos entrenamiento):** No normalizados
- **Nº capas:** 3
- **Configuración (neuronas/capa):** 64 (*relu*), 64 (*relu*), 1



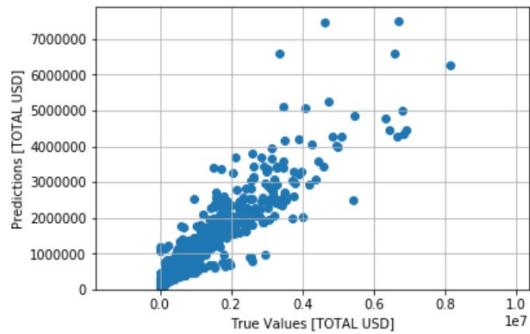
Prediction Plot



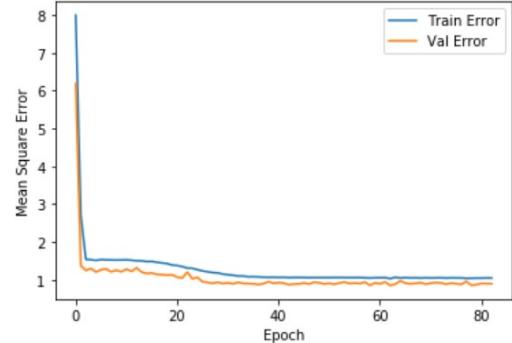
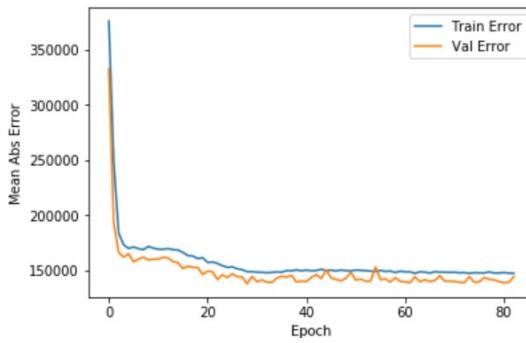
10 20% 137.687,86



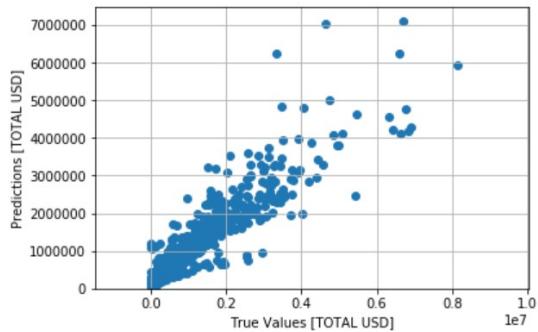
Prediction Plot



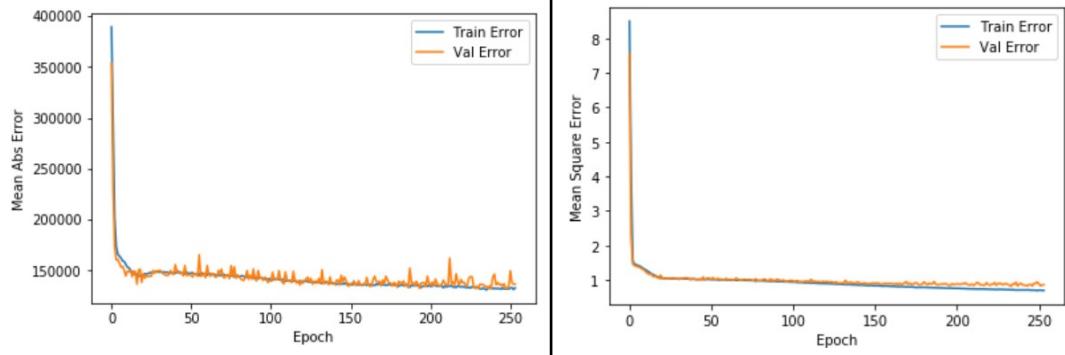
20 20% 138.604,21



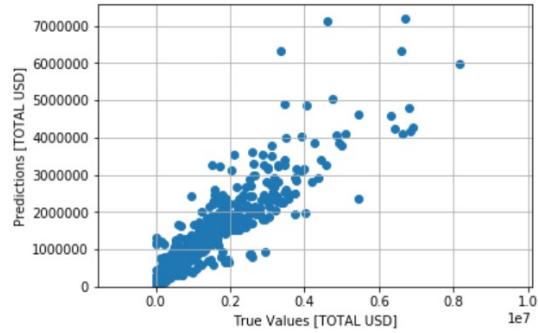
Prediction Plot



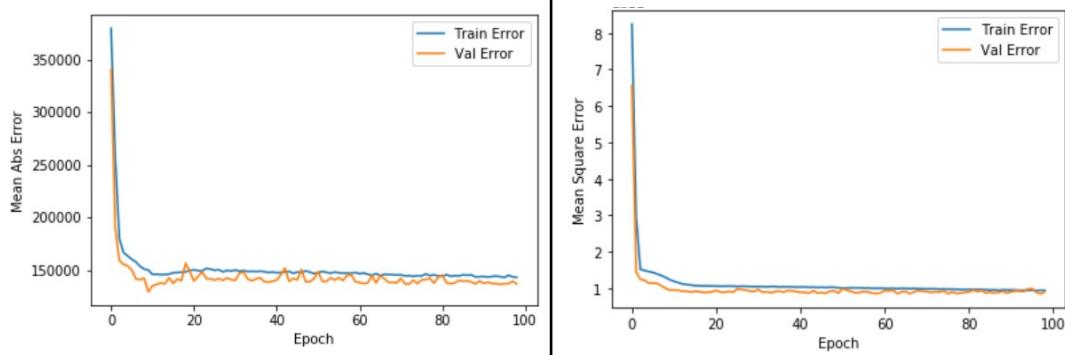
20 35% 125.626,35



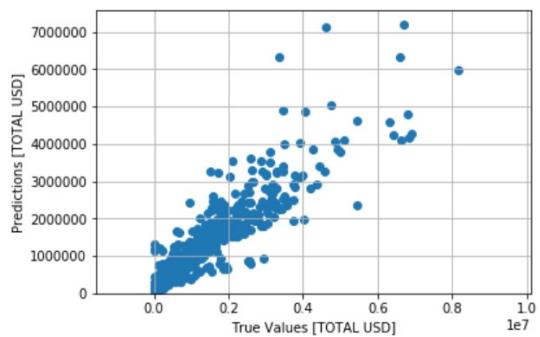
Prediction Plot



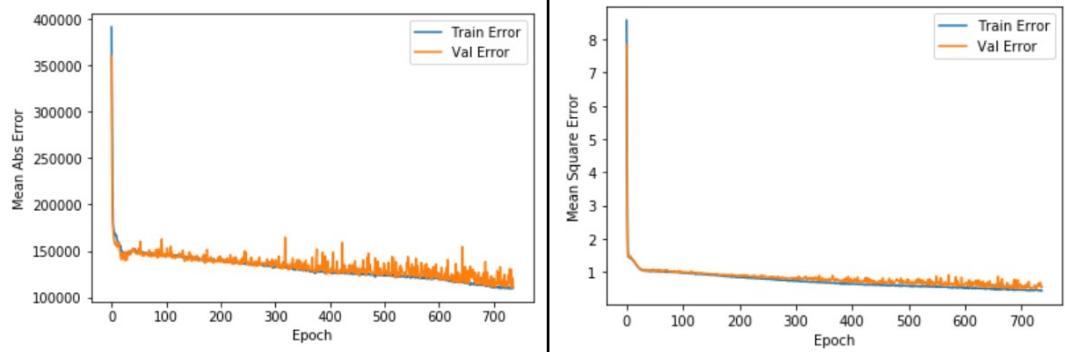
35 20% 131.721,73



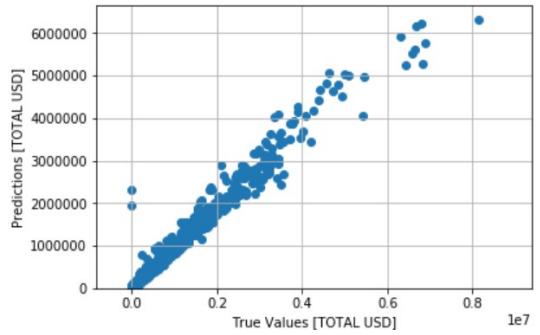
Prediction Plot



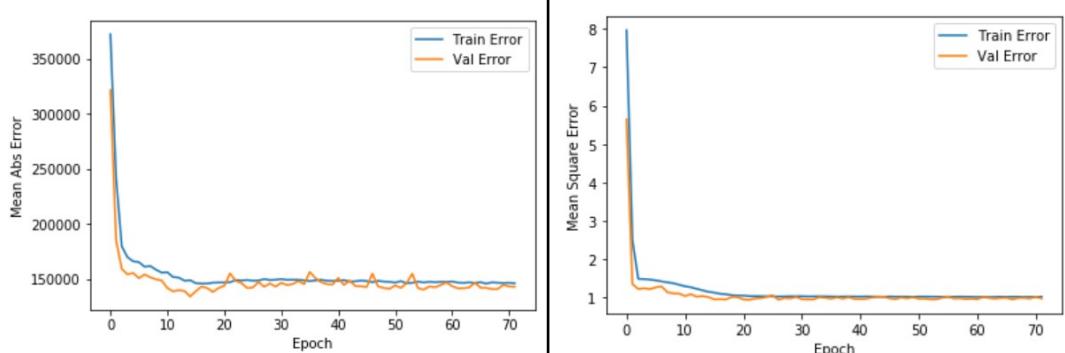
35 45% 48.932,63

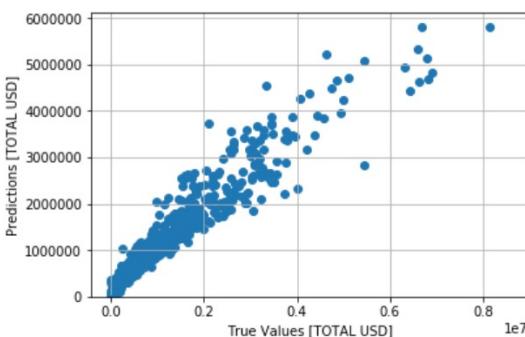
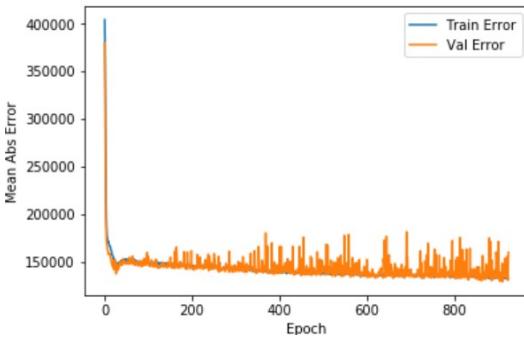
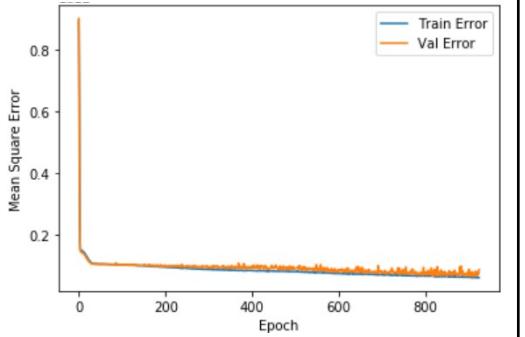
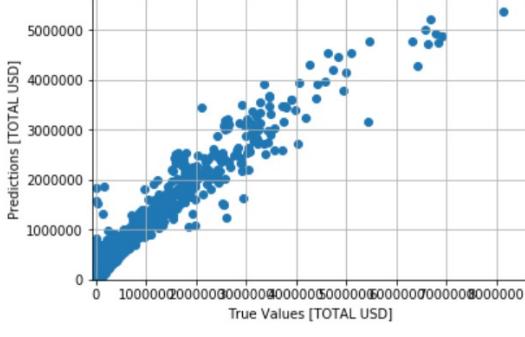


Prediction Plot



50 20% 120.963,49



<i>Prediction Plot</i>			
50	50%	150.683,23	 
<i>Prediction Plot</i>			
Units	%	USD	<i>plot</i>

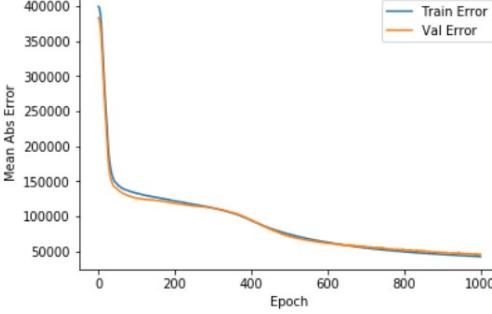
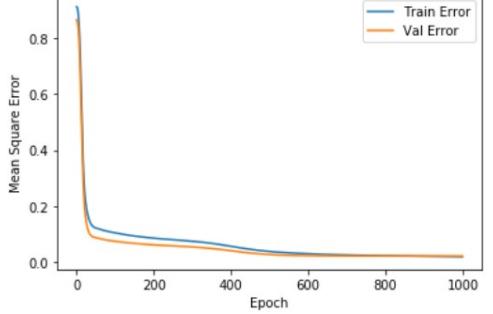
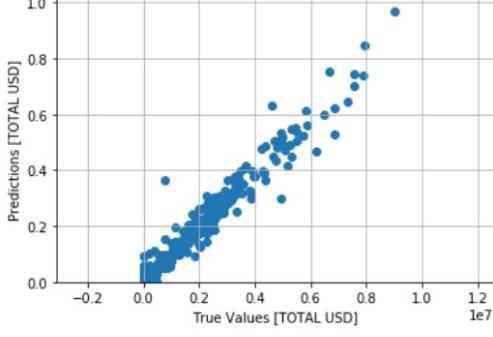
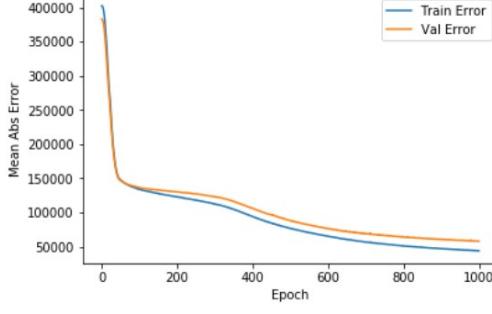
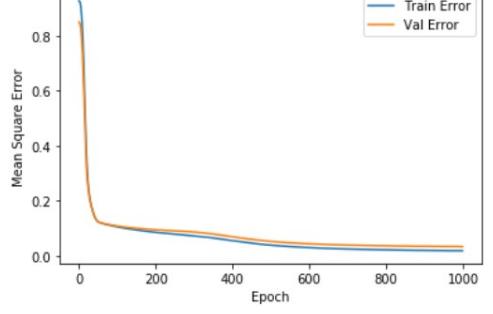
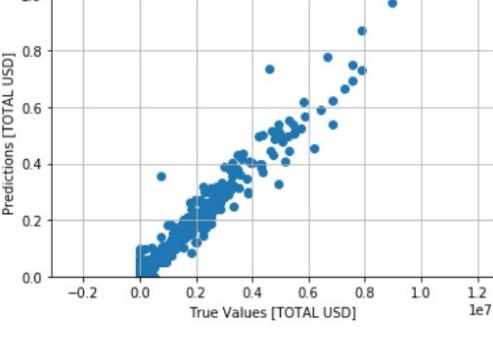
Tras experimentar con los valores no normalizados cuando aplicamos la función de Keras [EarlyStopping\(\)](#) el error aumenta de forma significativa en los siguientes casos:

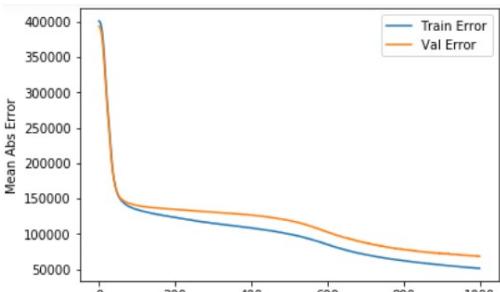
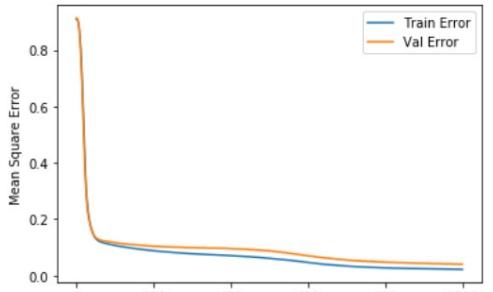
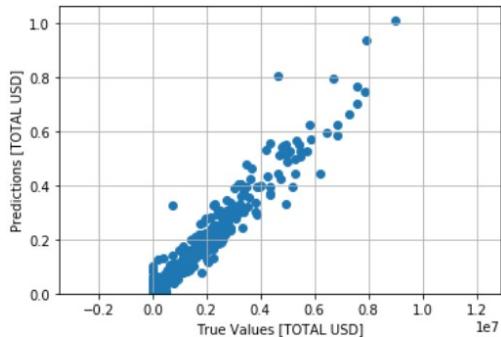
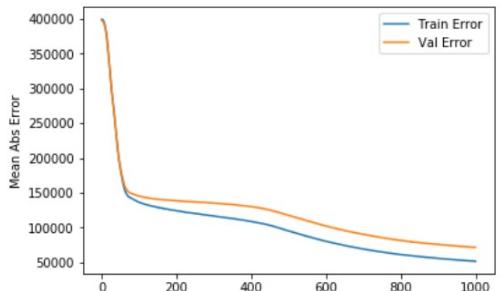
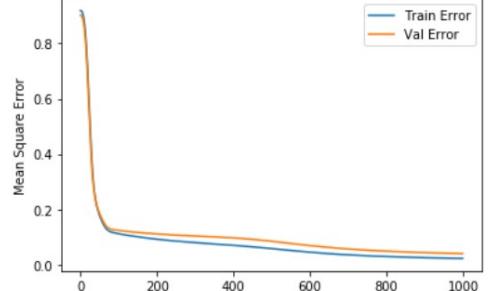
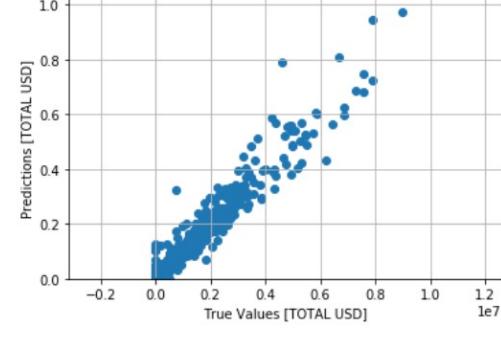
- Cuando el valor *patience* se encuentra dentro del intervalo (10, 30) U (40,50), pues o bien le indicamos que debe esperar muy poco para parar las iteraciones o por contra es demasiado elevado, en cuyo caso cuando entrenamos la red esta se entrena el mismo número de *epochs* que si no le aplicaramos esta función.
- En lo referente al *validation split* surge un efecto similar, si dejamos un pequeño porcentaje de muestras dentro del entrenamiento para efectuar la validación (20-35%) el modelo dispone de pocos elementos para la comprobación y de forma análoga si le ofrecemos demasiados, acabamos con problemas de sobreajuste (*overfitting*), imposibilitando así la capacidad de generalizar.

Para concluir con esta parte podemos determinar que los valores más óptimos serían para las muestras no normalizadas, emplear un *patience* de 35 y un *validation split* de 45%.

(Objetivo: Separación de los datos de validación)

- Estado de la información (datos entrenamiento): Normalizados
- Nº capas: 3
- Configuración (neuronas/capa): 64 (*relu*), 64 (*relu*), 1

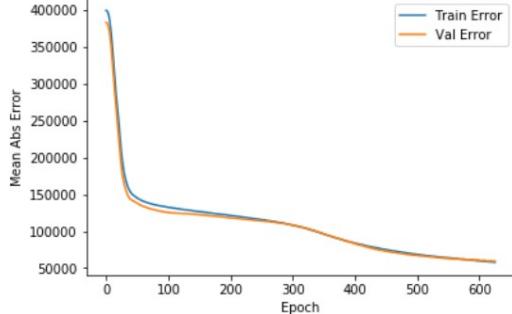
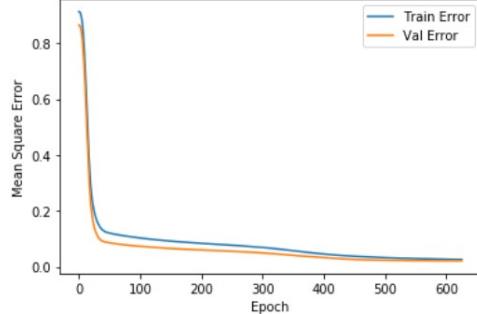
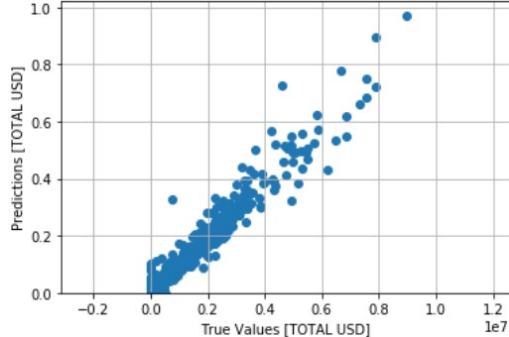
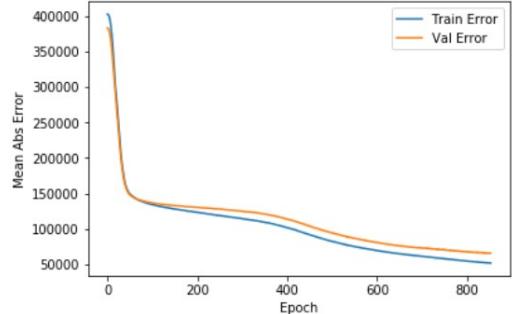
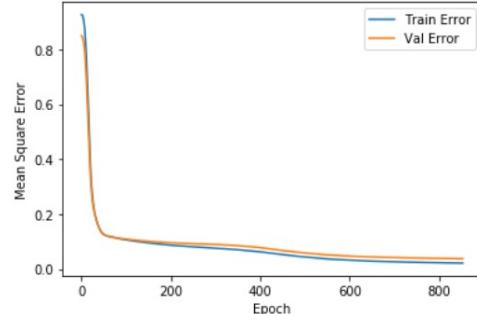
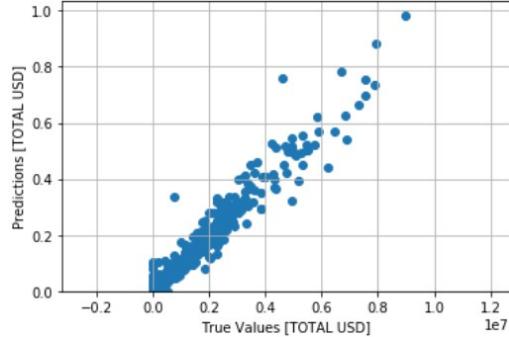
Pred.	Valid. split	Total Revenue	Mean Absolute Error	Mean Squared Error
AS, P, ATF, ABF, PW	5%	50.060,50		
Prediction Plot				
AS, P, ATF, ABF, PW	20%	56.032,53		
Prediction Plot				

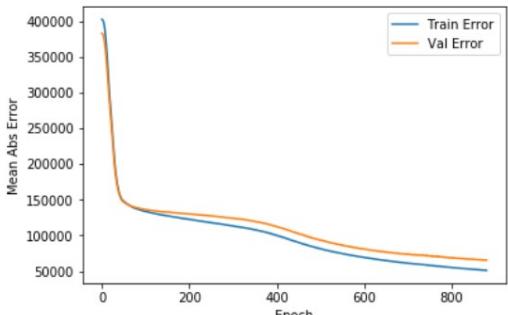
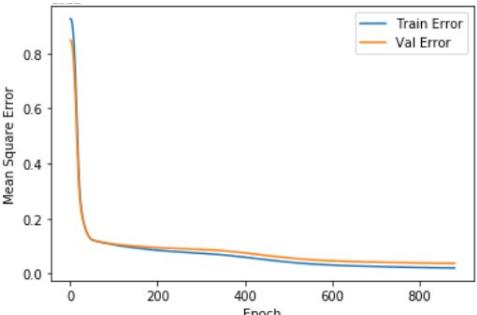
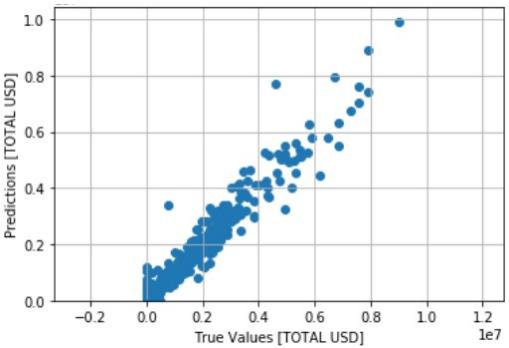
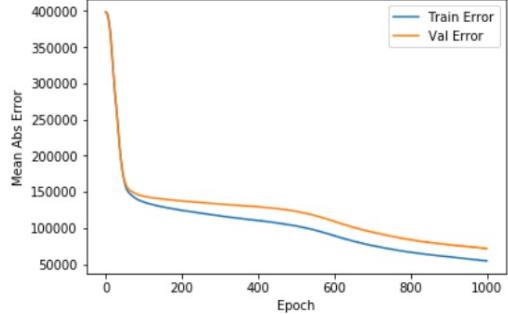
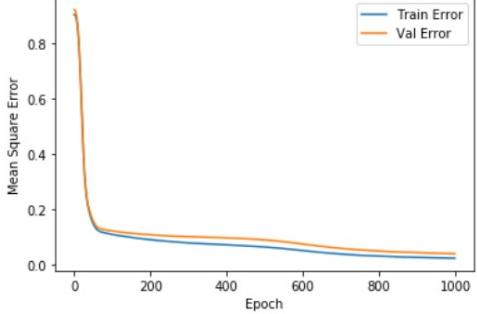
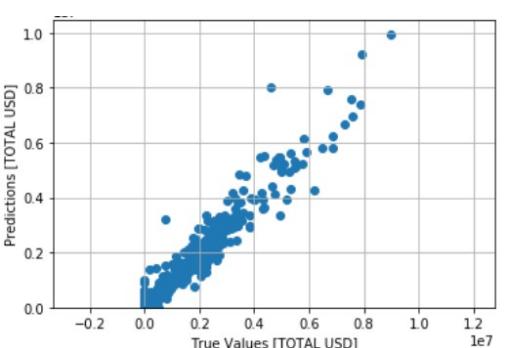
AS, P, ATF, ABF, PW	30%	65.404,30		
Prediction Plot				
AS, P, ATF, ABF, PW	45%	74.888,19		
Prediction Plot				
Units	%	USD	plot	

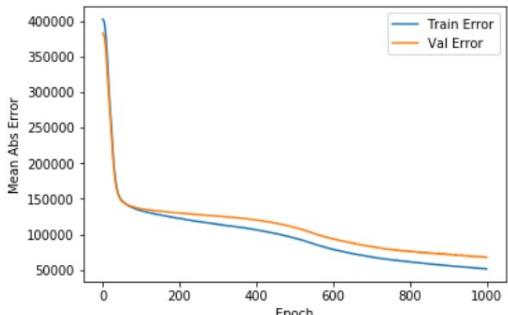
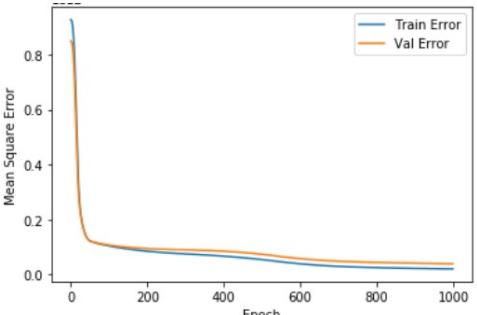
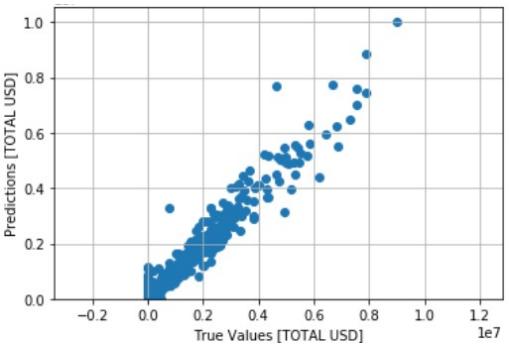
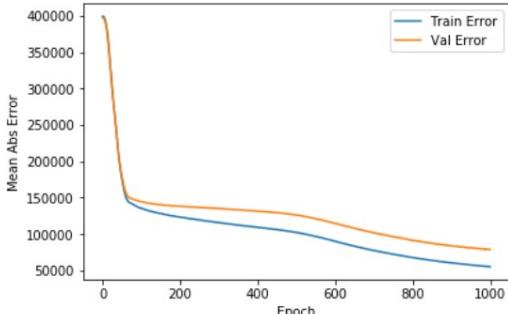
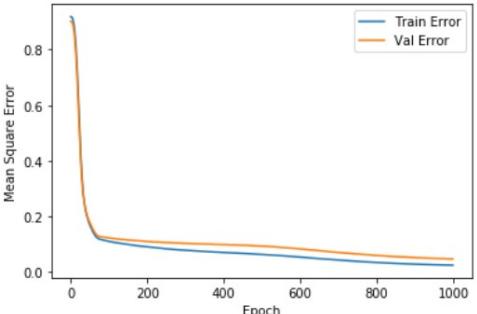
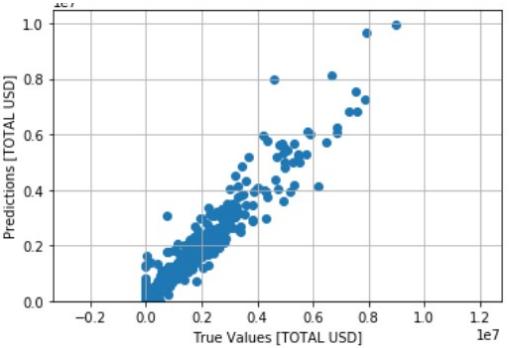
Ahora que disponemos de los datos normalizados, recordemos que para normalizarlos hemos ajustado cada uno de los valores de nuestro *dataset* a través de restarle el valor medio de la columna y dividirlo por la desviación estándar de la misma. Las gráficas ahora ya no presentan tanto ruido y podemos ver cómo los datos pueden ser presentados de forma más clara.

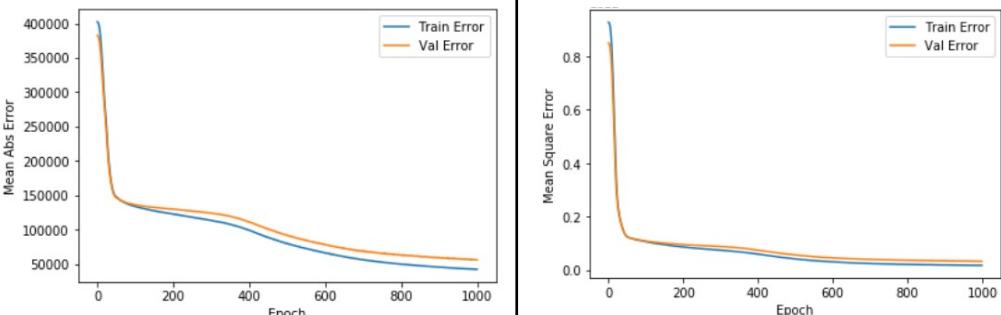
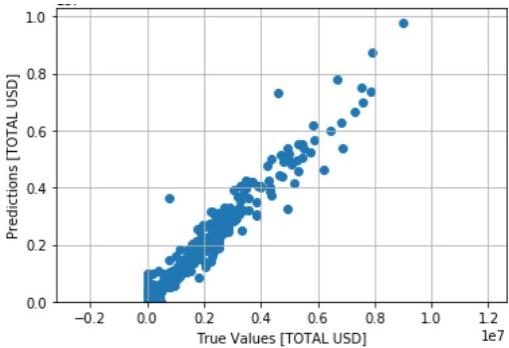
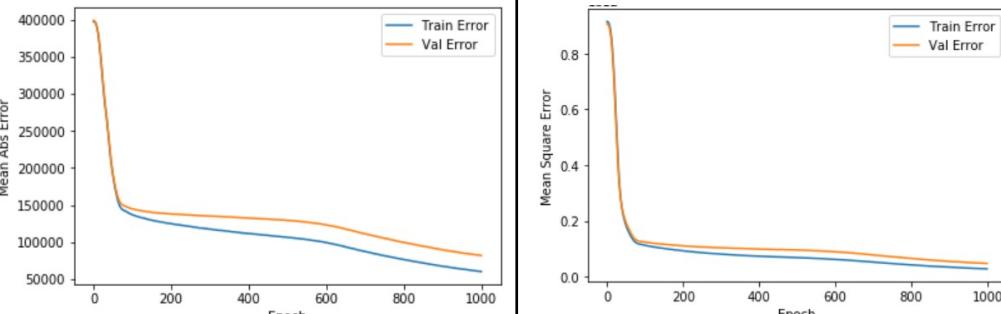
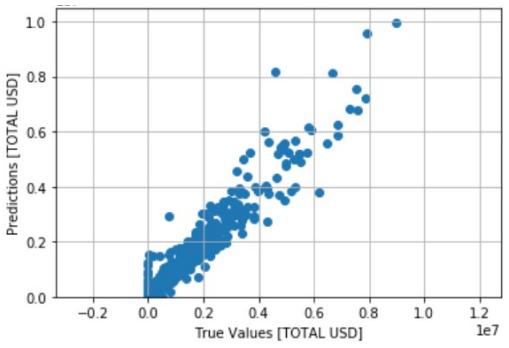
(Objetivo: finalizar el entrenamiento de forma anticipada)

- Estado de la información (datos entrenamiento): Normalizados
- Nº capas: 3
- Configuración (neuronas/capa): 64 (*relu*), 64 (*relu*), 1

Patié	Valid. split	Total Revenue	Mean Absolute Error	Mean Squared Error
10	5%	65.968,25		
<i>Prediction Plot</i>				
10	20%	63.356,83		
<i>Prediction Plot</i>				

20	20%	63.239,18	 
<i>Prediction Plot</i>			
20	35%	70.327,68	 
<i>Prediction Plot</i>			

35	20%	65.097,49	 	<p>Prediction Plot</p> 
35	45%	76.510,40	 	<p>Prediction Plot</p> 

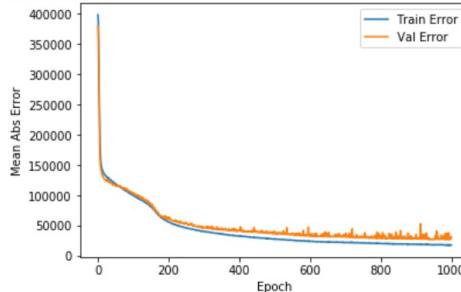
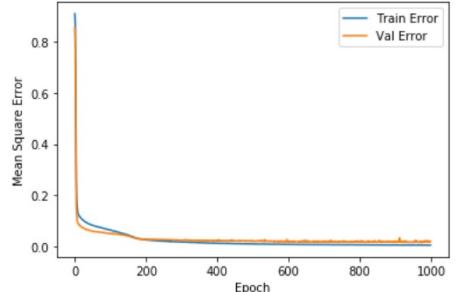
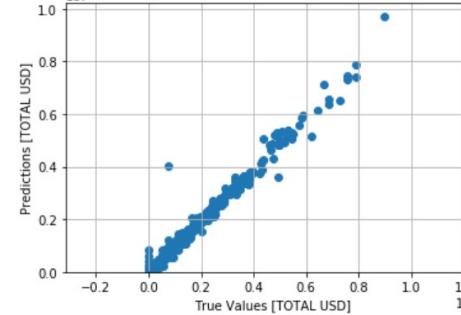
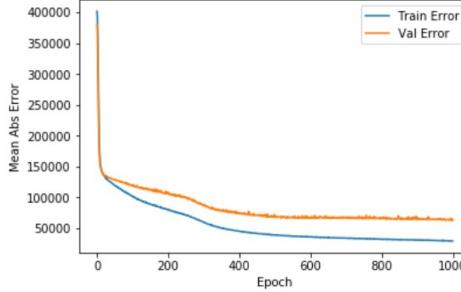
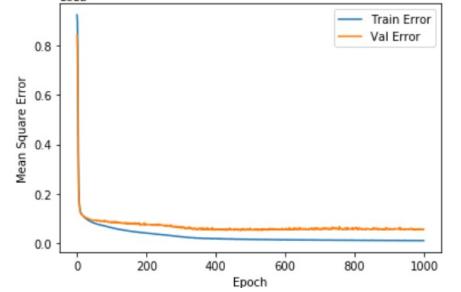
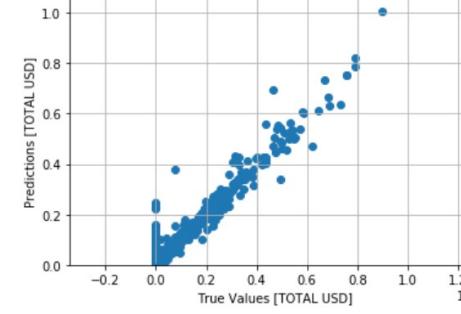
50	20%	54.109,86	
Prediction Plot			
50	50%	81.062,48	
Prediction Plot			
Units	%	USD	plot

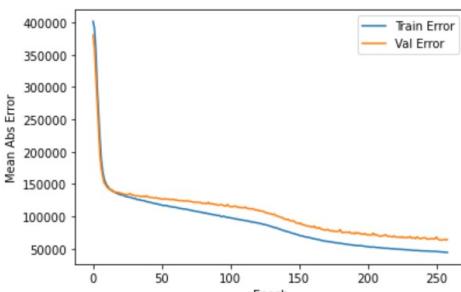
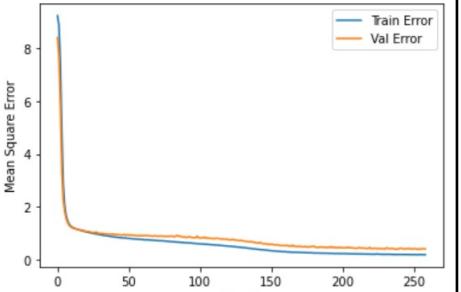
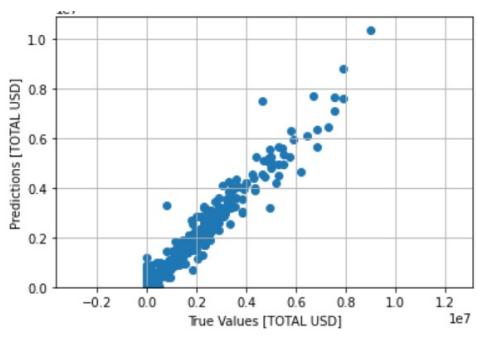
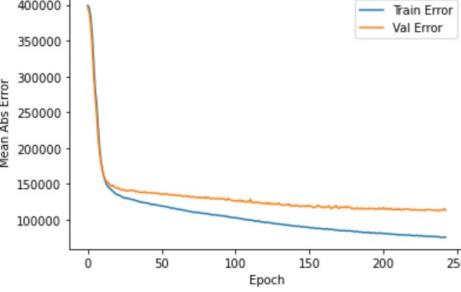
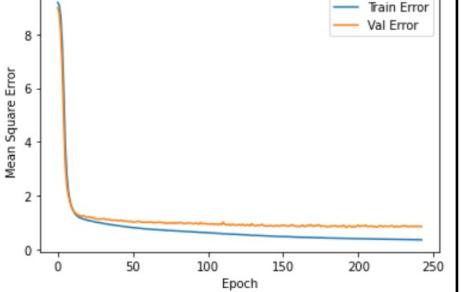
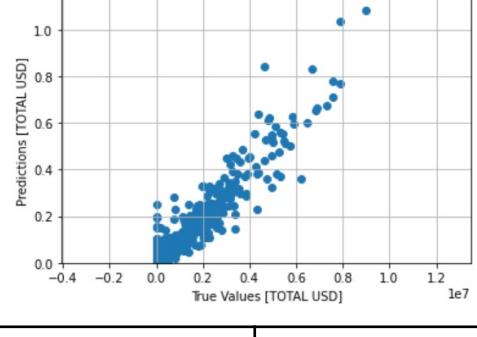
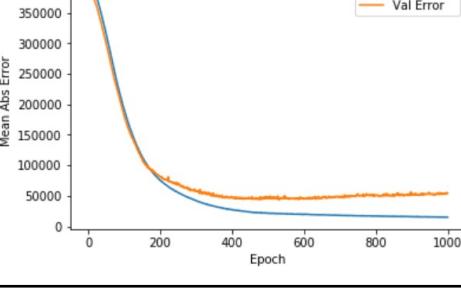
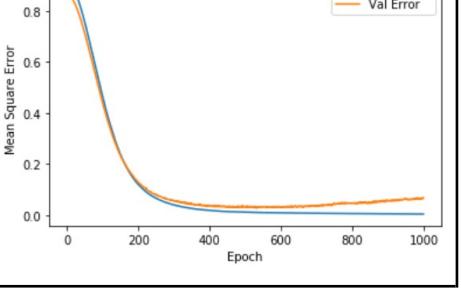
Ahora que ya hemos podido contemplar el funcionamiento, comportamiento y rendimiento de los parámetros *validation split* y *patience*, en los próximos experimentos tan sólo consideraremos aquellos valores que hasta ahora hemos podido constatar que proporcionan un menor error con respecto a la variable que queremos predecir.

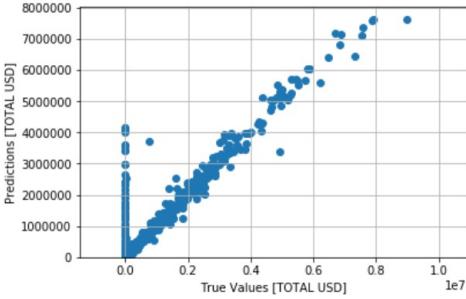
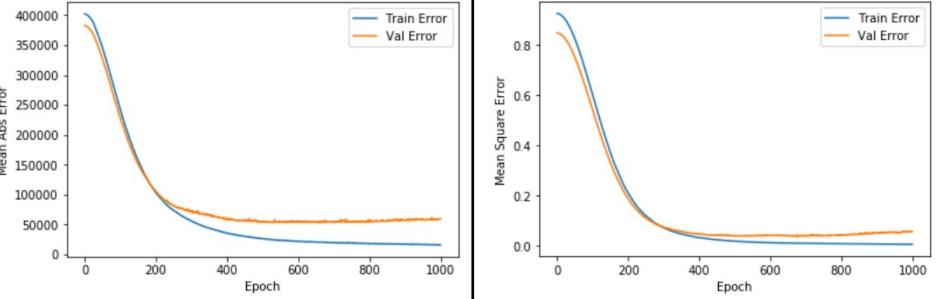
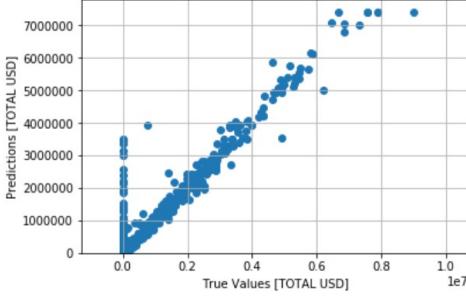
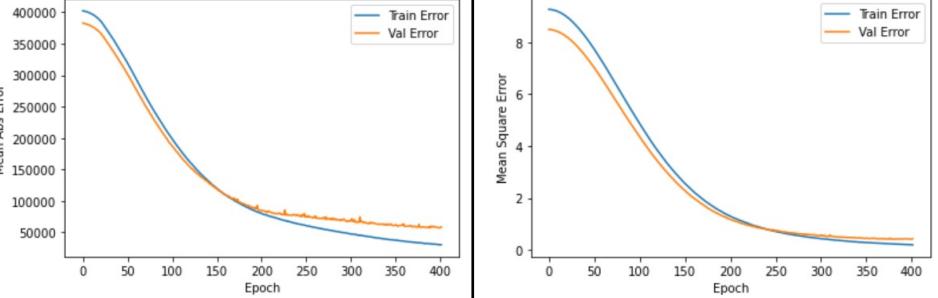
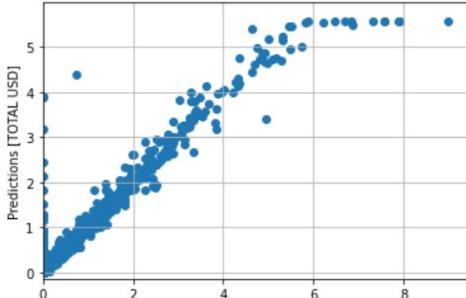
A continuación modificaremos la configuración de la RN para ver cómo varía el resultado.

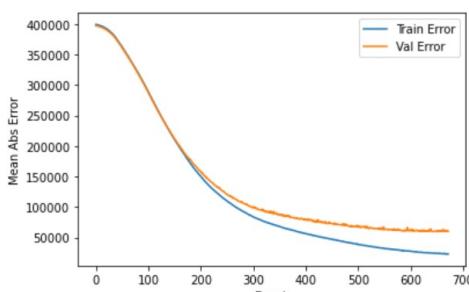
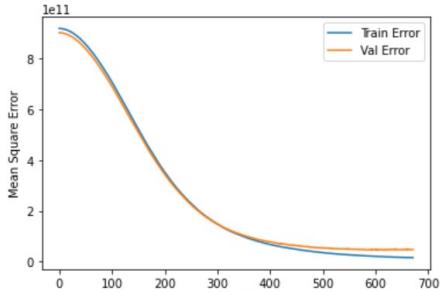
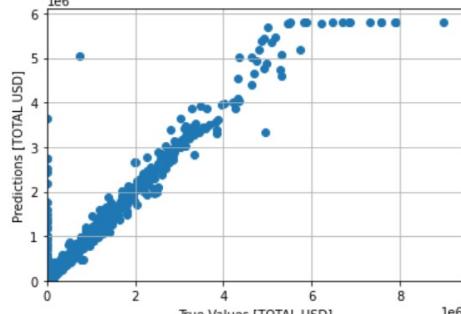
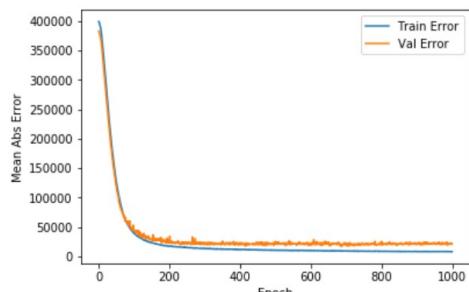
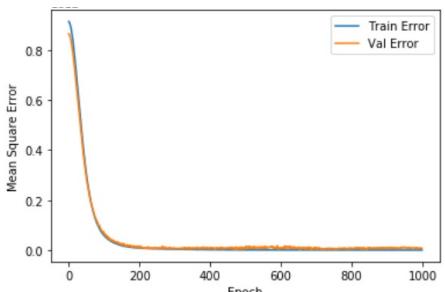
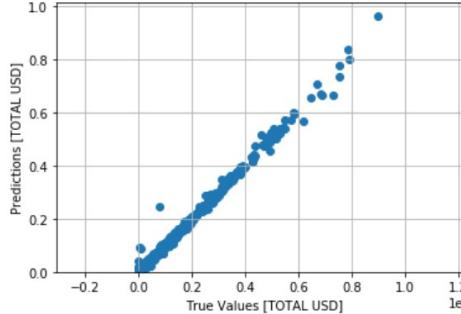
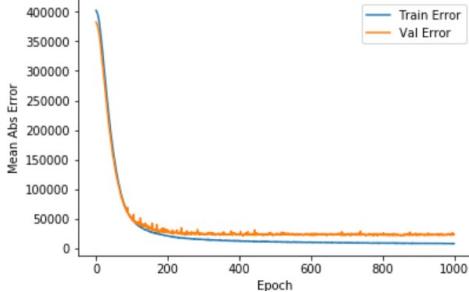
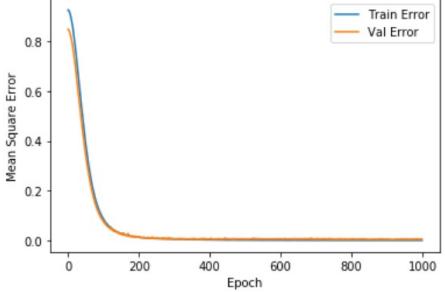
(Objetivo: Separación de los datos de validación)

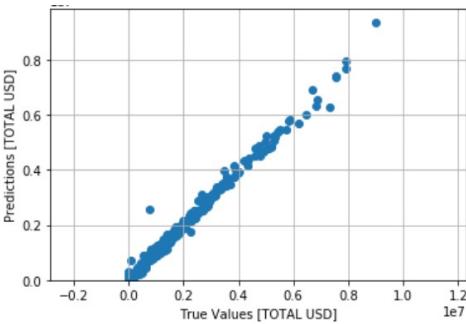
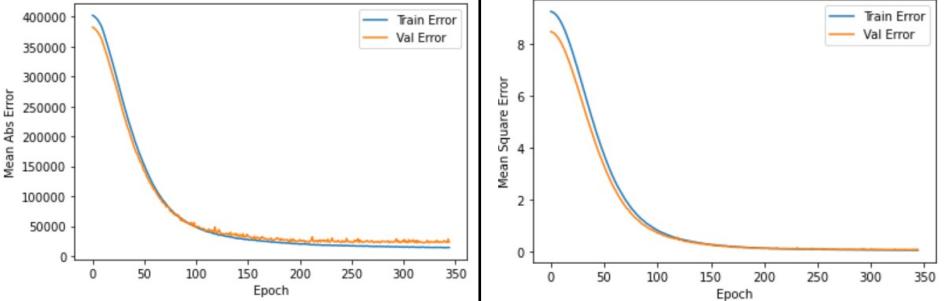
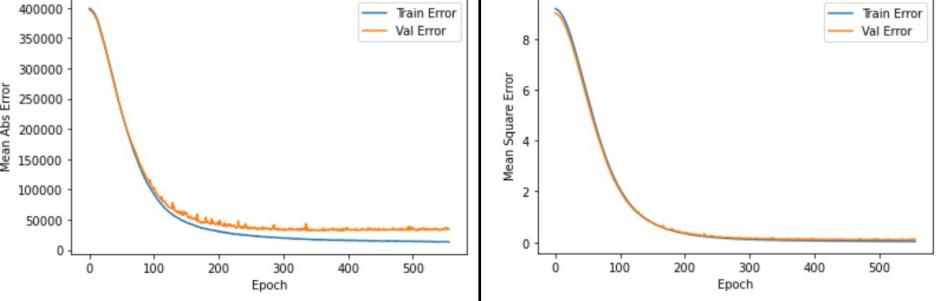
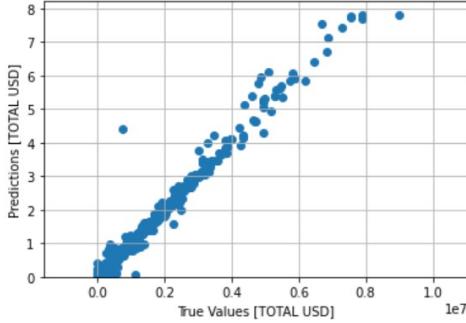
- Estado de la información (datos entrenamiento): Normalizados
- Nº capas: Variable
- Configuración (neuronas/capa): Variable

Nº Capas	Config.	Val. split	Mean Absolute Error	Mean Squared Error
4	128 relu, 128 relu, 32 relu, 1 //No EarlyStop	5%		
Total Rev.	32.000,06			
4	128 relu, 128 relu, 32 relu, 1 //No EarlyStop	20%		
Total Rev.	60.962,90			

4	128 relu, 128 relu, 32 relu, 1 //EarlyStop (pat): 20	20%	 
Total Rev.	61.823,59		
4	128 relu, 128 relu, 32 relu, 1 //EarlyStop (pat): 50	45%	 
Total Rev.	106.249,59		
4	64 relu, 64 sigmoide, 64 relu, 1 //No EarlyStop	5 %	 

Total Rev.	65.235,16		
Prediction Plot			
4	64 relu, 64 sigmoide, 64 relu, 1 //No EarlyStop	20 %	
Total Rev.	60.031,33		
Prediction Plot			
4	64 relu, 64 sigmoide, 64 relu, 1 //EarlyStop (pat): 20	20 %	
Total Rev.	66.350,76		
Prediction Plot			

4	64 relu, 64 sigmoide, 64 relu, 1 //EarlyStop (pat): 50	45 %	 
Total Rev.	62.750,59		
4	128 relu, 128 tanh, 128 relu, 1 //No EarlyStop	5 %	 
Total Rev.	20.307,07		
4	128 relu, 128 tanh, 128 relu, 1 //No EarlyStop	20 %	 

Total Rev.	23.086,98	
Prediction Plot		
4	128 relu, 128 tanh, 128 relu, 1 //EarlyStop (pat): 20	20 % 
Prediction Plot		
4	128 relu, 128 tanh, 128 relu, 1 //EarlyStop (pat): 50	45 % 
Total Rev.	33.229,33	
Prediction Plot		

Podemos ver cómo en esta última tabla los valores de error en la variable que queremos predecir se han reducido de forma considerable, pues en este caso gracias al *EarlyStopping* podemos evitar problemas de sobreajuste.

La forma más fácil de visualizar cuando este fenómeno sucede es cuando la red se encuentra alrededor de las 500 epochs pues podemos ver como la distancia entre ambas curvas se acentúa. Este problema sucede debido a que la red ha sido sobreentrenada y se ajusta muy bien a ciertos casos, aquellos que ya ha visto anteriormente, en el entrenamiento pero después a la hora de generalizar la red no responde de la forma deseada. Por ello el uso del *EarlyStopping* nos permite evitar este fenómeno.

6.2.2. Resumen de las configuraciones

Ahora que disponemos de una gran cantidad de datos resultantes de la modificación de parámetros del modelo, debemos de alguna forma recopilarlos todos para poder lograr una comprensión clara de cuáles han sido las configuraciones y parámetros que han permitido una mayor predicción. Para ello hemos confeccionado una tabla resumen de los 35 experimentos realizados, tan sólo para este apartado.

En ella hemos destacado en verde las configuraciones más óptimas y en amarillo aquellas que pese a no ser las más óptimas también ofrecen un buen rendimiento. De esta forma podemos, tras todo el cómputo realizado, extraer conclusiones que permitan extrapolar la lógica para reducir el tiempo de decisión en futuras confecciones de RNs.

Conclusiones para desarrollar *Airline Profit Predictor*

Como podemos ver en la siguiente tabla las configuraciones más óptimas son:

Nº Capas	Configuración de la RN				Norm.	Early Stop		Val. split	Media Error Total Revenue	
	Desglose por capa					X	Patience			
	1 ^a	2 ^a	3 ^a	4 ^a						
4	128, <i>relu</i>		128, <i>tanh</i>		X			5	20.307,07	
								20	23.086,98	
						X	20	20	22.767,21	

La configuración que nos permite obtener un menor error en la variable *Total Revenue* es combinar las funciones de activación *tangente hiperbólica* (*tanH*) y *rectified linear unit* (*Relu*). La diferencia de aplicar esta combinación junto con el aumento del doble de neuronas por capa, permite reducir el error de 30.000 (aprox), mínimo anterior, a 20.000, reduciendo el error cometido por la red en un 33,3%. Permitendonos ofrecer una mejora sustancial para el modelo final. A continuación se muestra la tabla con todos los experimentos contemplados para poder ofrecer la tabla anterior.

Configuración de la RN					Norm.	Early Stop		Val. split	Media Error Total Revenue	Error %				
Nº Capas	Desglose por capa					X	Patience							
	1 ^a	2 ^a	3 ^a	4 ^a										
3	64, relu	64, relu	1					5 %	33.357,55	1,28%				
								20 %	62.377,24	2,39%				
								30 %	30.343,55	1,16%				
								45 %	88.649,29	3,39%				
						X	10	5 %	142.001,19	5,44%				
							10	20 %	137.687,86	5,27%				
							20	20 %	138.604,21	5,31%				
							20	35 %	125.626,35	4,81%				
	128, relu	128, relu	32, relu	1	X			35	20 %	131.721,73	5,04%			
								35	45 %	48.932,63	1,87%			
								50	20 %	120.963,49	4,63%			
								50	50 %	150.683,23	5,77%			
						X		5 %	50.060,50	1,92%				
								20 %	56.032,53	2,15%				
								30 %	65.404,30	2,50%				
								45 %	74.888,19	2,87%				
4	128, relu	128, relu	32, relu	1	X			10	5 %	65.968,25	2,53%			
								10	20 %	63.356,83	2,43%			
								20	20 %	63.239,18	2,42%			
								20	35 %	70.327,68	2,69%			
						X		35	20 %	65.097,49	2,49%			
4	64, relu	64, sigmoide	64, relu	1	X			35	45 %	76.510,40	2,93%			
								50	20 %	54.109,86	2,07%			
								50	50 %	81.062,48	3,10%			
							5 %	32.000,06	1,23%					
							20 %	60.962,90	2,33%					
4	128, relu	128, tanh	128, relu	1	X	X	20	20 %	61.823,59	2,37%				
								50	45 %	106.249,59	4,07%			
								5 %	65.235,16	2,50%				
								20 %	60.031,33	2,30%				
								20	20 %	66.350,76	2,54%			
4	128, relu	128, tanh	128, relu	1	X	X	50	45 %	62.750,59	2,40%				
								5 %	20.307,07	0,78%				
								20 %	23.086,98	0,88%				
						X	20	20 %	22.767,21	0,87%				
								50	45 %	33.229,33	1,27%			

TOTAL ERROR (<i>Total Revenue</i>)	2.611.800,58	100,00%
--------------------------------------	---------------------	----------------

7. Presentación *Airline Profit Predictor*

Para finalizar el desarrollo de este proyecto, ya estamos listos para introducir la herramienta confeccionada en este trabajo.

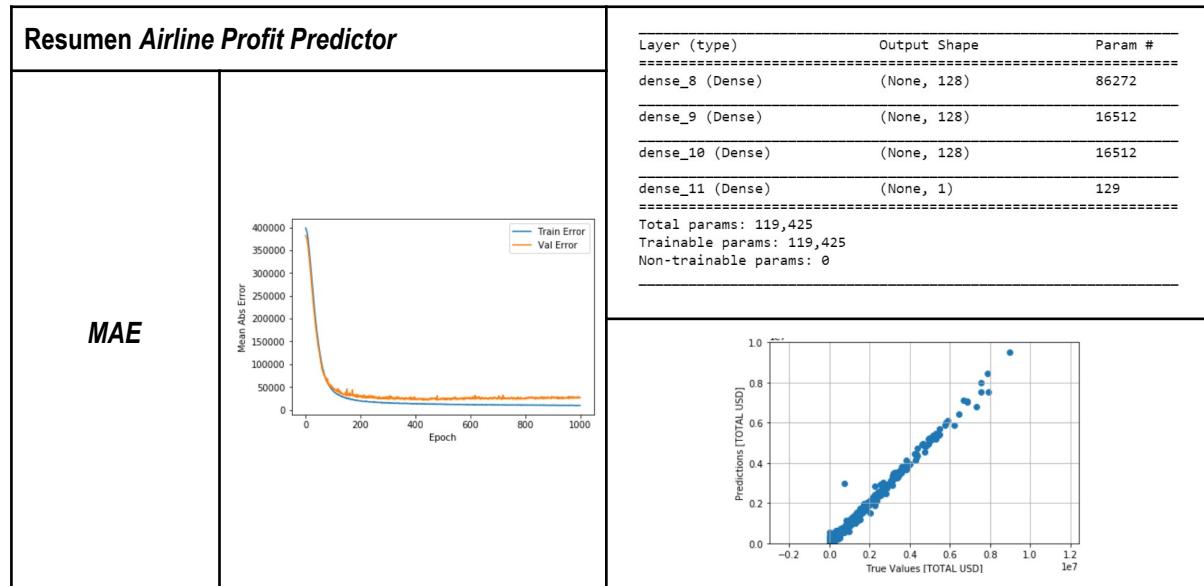
Gracias al estudio y procesamiento de nuestro *dataset* hemos logrado crear una herramienta predictiva para aerolíneas que quieran reducir el riesgo e incertidumbre al abrir una ruta de BCN a cualquiera de los 452 destinos contemplados por nuestra red. Los modelos de redes neuronales suelen ejecutarse de forma local, pues la mayor parte no son creadas para producción sino para investigación (*Data Science*), en nuestro caso creemos que la forma más fácil y rápida de distribuir y comercializar nuestra herramienta es a través de una API (*Application Programming Interface*).

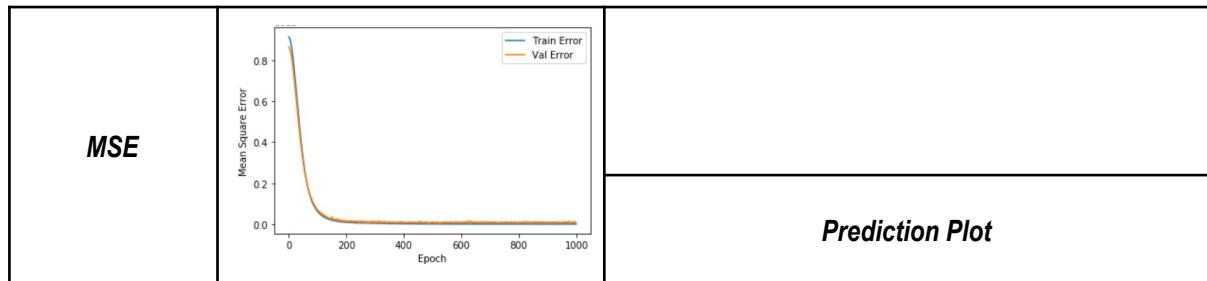
De esta forma cualquier aerolínea o desarrollador que quiera emplear nuestra tecnología tan sólo debe realizar una *request* con los siguientes parámetros:

- Cuota de Mercado (*Airline Share*)
- Pasajeros (*Passengers*)
- Media de la Tasa (*Avg. Total Fare*)
- Media de la Base Imponible (*Avg. Base Fare*)
- Pasajeros por día por ruta (*PPDEW*)

A través de estos parámetros podemos ofrecer a nuestros clientes un servicio capaz de predecir con un error medio absoluto (*mae*) de 20.000\$, el beneficio esperado. Comentar que error no es muy significativo cuando hablamos de millones o miles de millones de dólares.

El modelo que se ha creado para el desarrollo de esta herramienta se anexa a este trabajo bajo el nombre de ***Airline Profit Predictor.pdf***.





Con la finalidad de ofrecer una visión rápida de la rentabilidad económica del producto que hemos confeccionado y ver la viabilidad presentamos un resumen de los resultados antes de impuestos del primer año.

Resultados 1r Año				100.972,04
	Facturación			101.319,04
Concepto	Precio/ud	Unidades	Meses	Total
Request unitaria	1,00	100.000,00	1,00	100.000,00
Subscripción Básica	9,99	5,00	12,00	599,40
Subscripción Premium	19,99	3,00	12,00	719,64
	Costes			347
Concepto	Precio/ud	Unidades	Meses	Total
Requests	0,0001	1.070.000,00	1,00	107
Mantenimiento plataforma	20,00	1,00	12,00	240

Como se puede observar los costes asociados son muy pequeños comparados con la expectativa de beneficio, por lo que no es de extrañar que este estilo de negocio esté cada vez más y más presente.

8. Conclusiones

Tras el desarrollo de todo el trabajo de investigación y experimentación podemos, en base a los objetivos planteados inicialmente afirmar:

Objetivos Teóricos

- Hemos entendido que es y en que se basan las Redes Neuronales.
- Las RN de regresión nos permiten confeccionar modelos basados en los propios datos y no en heurísticas o ideas preconcebidas.
- Una RN se articula en base a neuronas, que estas conforman capas, funciones de activación que deforman el comportamiento lineal y la retropropagación del error cometido, todo esto de forma iterada.
- El algoritmo *backpropagation* nos permite imputar el error la influencia que cada neurona ha impuesto en la iteración para el cálculo del *output*.
- Las tablas y gráficas nos han permitido comprender la evolución y rendimiento de las RNs confeccionadas.
- Para la creación de un modelo se deben: analizar los datos, prepararlos y entrenar la red acorde a ellos. Testear y modificarla para mejorar la precisión y reducir el error cometido por la RN.
- Para evaluar una RN se debe analizar la coherencia del *output* y sobretodo mezclar los datos de forma que podamos evitar problemas de sobreajuste (*overfitting*).

Objetivos Prácticos

- Hemos sido capaces de limpiar y preparar los datos para crear la herramienta *Airline Profit Predictor*.
- Hemos mezclado y suministrado los datos a la RN.
- Se han entrenado varias RNs con los datos de entrenamiento.
- Hemos desarrollado un modelo predictivo capaz de predecir el beneficio neto de una aerolínea.

- Se han realizado múltiples análisis sobre los *outputs* en función de la combinación de *nº neuronas, nº capas, normalización y partición de datos*.
- Hemos reevaluado el modelo e implementado mejoras en cada iteración.

9. Bibliografía

https://es.wikipedia.org/wiki/Red_neuronal_artificial

https://www.youtube.com/watch?v=VwVg9jCtqaU&t=747s&ab_channel=TensorFlow

https://www.shanelynn.ie/python-pandas-read_csv-load-data-from-csv-files/

<https://www.freecodecamp.org/news/how-to-build-your-first-neural-network-to-predict-house-prices-with-keras-f8db83049159/>

<http://www.redes-neuronales.com.es/tutorial-redes-neuronales/clasificacion-de-las-redes-neuronales-artificiales.htm>

<https://torres.ai/redes-neuronales-recurrentes/>

<https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

<http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm>

<https://playground.tensorflow.org>

<https://www.google.com/url?sa=t&source=web&rct=j&url=https://dialnet.unirioja.es/descarga/articulo/7025156.pdf&ved=2ahUKEwjVoaHJ893tAhUGjhQKHWpxAw0QFjABegQIDBAbusq=AOvVaw0zLzPbjfdh9JELEcEqWFk1https://stackoverflow.com/questions/40790031/pandas-to-numeric-find-out-which-string-it-was-unable-to-parse>

<https://stackoverflow.com/questions/40083266/replace-comma-with-dot-pandas>

<https://stackoverflow.com/questions/49088443/search-and-replace-dots-and-commas-in-pandas-dataframe>

<https://stackoverflow.com/questions/16729483/converting-strings-to-floats-in-a-dataframe>

<https://living-sun.com/es/python/730923-pandas-dataframe-encode-categorical-variable-with-thousands-of-unique-values-python-pandas-categorical-data.html>

<https://www.geeksforgeeks.org/python-pandas-categoricaldtype/>

<https://stackoverflow.com/questions/51047676/how-to-get-accuracy-of-model-using-keras>

https://www.w3schools.com/python/ref_func_max.asp

<https://www.youtube.com/watch?v=kft1AJ9WVDk&t=92s>

https://www.youtube.com/watch?v=Wo5dMEP_Bbl

<https://www.youtube.com/watch?v=9UBqkJVP4g>

