Universidad Zaragoza
1542

Trabajo Fin de Grado

Aprendizaje supervisado usando Redes Neuronales

Autor

Víctor Bouhaben Barrera

Directora

Piedad Garrido Picazo

Escuela Universitaria Politécnica de Teruel

2024

Tabla de Contenidos

Índice de figuras

Índice de tablas

Resumen

El propósito de este Trabajo de Final de Grado (TFG) consiste en el desarrollo una red neuronal para clasificar caracteres codificados mediante matrices 7x5 píxeles, donde un valor de 1 representa el color negro y un valor 0 representa el color blanco.

El software desarrollado tendrá la finalidad de ser usado como simulador para la resolución de problemas planteados en las sesiones de laboratorio de la asignatura de Inteligencia Artificial (IA), garantizando así que el aprendizaje de los alumnos, tanto del Grado en Ingeniería Informática (GII), como del Doble Grado ADE-GII, sea más fácil y provechoso.

Para el desarrollo del trabajo, se han creado distintos conjuntos de datos para realizar el entrenamiento de la red neuronal artificial creada, un Perceptrón Multicapa (MLP) creado con la librería de Python denominada PyTorch y comparado con otros modelos creados en la plataforma H2O.

Por último, se han realizado pruebas en ambos entornos, se han expuesto los resultados y se han obtenido interesantes conclusiones en base a ellos.

Palabras clave

Inteligencia Artificial, Aprendizaje Profundo, MLP, PyTorch, H2O.

Abstract

The purpose of this Final Degree Project is to develop a neural network to classify characters encoded using 7x5 pixel arrays, where a value of 1 performs as the color black and a value of 0 performs as the color white.

The developed software will be intended to be used as a simulator to resolve proposed practicing from laboratory sessions of Artificial Intelligence (AI) subject, an easier student learning will be secured with it, not only Computer Engineering Degree (GII), but also Double Grade ADE-GII.

For the project development, a few datasets have been created to allow neural network training, which is a Multilayer Perceptron developed with a Python's library called PyTorch and compared with some other models from H2O platform.

Finally, many tests have been done in both environments, result have been exposed and some interesting conclusions have been obtained.

Keywords

Artificial Intelligent, Deep Learning, MLP, PyTorch, H2O

1. Introducción

La Inteligencia Artificial (IA) es uno de los temas de mayor interés, en la actualidad, en cualquier ámbito. Este concepto se refiere a los sistemas informáticos capaces de realizar tareas que requieren de la inteligencia humana y sus características, como el razonamiento y el aprendizaje.

En la totalidad de lo que aborda la IA, está el aprendizaje automático o Machine Learning (ML), que se trata de una disciplina de la IA que, gracias al uso de diferentes algoritmos, consigue que los ordenadores sean capaces de identificar patrones en base a datos, relacionarlos y realizar predicciones en función de esto.

Dentro del aprendizaje automático existe el aprendizaje supervisado, donde la máquina recibe un entrenamiento controlado por humanos, es decir, el aprendizaje del ordenador requiere de la intervención de humanos para determinar qué datos, resultados o acciones son correctos. Esta retroalimentación permite que la máquina sea capaz de ajustarse al objetivo y mejorar su rendimiento de gran manera.

Una de las técnicas de aprendizaje automático más utilizada y extendida tanto para el desarrollo de modelos de aprendizaje supervisado, como para otras formas de aprendizaje automático, es el aprendizaje profundo o Deep Learning.

El aprendizaje profundo es un conjunto de algoritmos de aprendizaje automático basado en capas de redes neuronales que son modelos computacionales inspirados en la estructura y el funcionamiento del cerebro humano. Estas redes están compuestas por unidades de procesamiento llamadas neuronas que pertenecen a capas de la red y están conectadas entre sí.

Estas técnicas son idóneas para el desarrollo de este Trabajo de Fin de Grado (TFG), ya que han demostrado ser de gran utilidad para el procesamiento de lenguaje.

Para la implementación de este TFG se usarán las tecnologías actuales en este ámbito que hacen uso de las técnicas comentadas. En concreto, se utilizará PyTorch, un marco de aprendizaje profundo de código abierto orientado a la creación de redes neuronales.

PyTorch se compone de la combinación de la biblioteca de aprendizaje automático de Torch y una API de alto nivel basada en Python (1).

Por otro lado, también se utilizará el entorno H2O, una plataforma de código abierto líder en IA, especializada en aprendizaje profundo y AutoML (2).

Esta plataforma permitirá llevar a cabo la implementación de la red neuronal deseada y su entrenamiento de una forma muy intuitiva y sencilla. Además, proporcionará de manera clara los resultados obtenidos, detallándolos y empleando esquemas para visualizarlos de mejor forma.

Es en el ámbito del reconocimiento de texto donde se va a centrar en este Trabajo de Fin de Grado, tanto a nivel tecnológico como a nivel educativo, ya que se trata de la puesta a punto de una red neuronal que permite identificar caracteres descritos con píxeles. La red se desarrollará junto con un software de simulación que permite ver la creación de esta, junto con sus diferentes parámetros y características, además de los resultados obtenidos y las optimizaciones realizadas al aplicar estos parámetros.

Antes de todo esto y para su desarrollo, se ha llevado a cabo el estado del arte, en el que se han determinado e investigado los posibles softwares que puedan ofrecer una solución a lo que se plantea en este Trabajo de Fin de Grado.

Se han encontrado diferentes softwares simuladores de redes neuronales. Sin embargo, todos ellos disponen y hacen uso de tecnología ya obsoleta y antigua. Como bien puede ser el software Java Neural Network Simulator (JNNS), que se ha usado a modo orientativo para el desarrollo de este trabajo.

Por otro lado, también se ha tomado de referencia una práctica existente en 2017 dentro de la documentación y el plan de la asignatura de Inteligencia Artificial de la Escuela Politécnica de Teruel (EUPT).

Este Trabajo de Fin de Grado, pretende ser un simulador de redes neuronales desarrollado con la tecnología más actual, para que esta práctica pueda ser implementa con él.

Para ello, se parte del dataset que ya existía, encontrándose con que se trataba de un conjunto de datos muy limitado, por lo que se ha desarrollado una ampliación de éste de manera propia.

El documento concluye con un apartado en el que se indica la licencia software y documental, así como las conclusiones obtenidas del proyecto y unas líneas de trabajo futuro.

2. Objetivos

A la hora del desarrollo de un Trabajo de Fin de Grado, es de gran importancia definir previamente los objetivos que se quieren alcanzar con este.

En primer lugar, está la creación tanto de la red neuronal como del simulador, pero estos serían los objetivos finales con el TFG ya desarrollado plenamente, por lo que existen otros objetivos en los que se puede dividir el objetivo final, que serían:

- Creación de una red neuronal en el marco de trabajo de PyTorch.
- Desarrollo de una red neuronal en la interfaz H2O Flow.
- Creación de distintos conjuntos de datos o datasets para entrenar las redes neuronales de los dos puntos anteriores.
- Definir en ambos entornos de trabajo los diferentes parámetros y atributos de las redes neuronales.
- Realizar pruebas tanto con los datasets creados, como con los parámetros de las redes. Haciendo modificaciones en ellos para buscar la red más óptima.
- Crear un entorno en el que el usuario pueda realizar estas pruebas y visualizar los resultados y las optimizaciones.

3. Estado del Arte

Este estado del arte se ha centrado en la búsqueda de programas que permitan la creación, manipulación y entrenamiento de redes neuronales. Por un lado, se han realizado búsquedas de simuladores y, por otro lado, se han llevado a cabo búsquedas de entorno de desarrollo para poder implementar una red neuronal de creación propia.

Como punto de partida, se ha usado a modo orientativo el programa JNNS (Java Neural Network Simulator), explicado más adelante.

En concreto, se han encontrado 3 opciones de simuladores que parecen cumplir con los requisitos de lo buscado en este TFG y que se van a proceder a analizar en profundidad. Algunas de sus características principales se resumen en la siguiente tabla:

Nombre	Fuente	Año	Lenguaje
JNNS	https://github.com/mwri/javanns	1999	Java/C
Interactive Neural Network Simulator	https://interactive-neural-network-simulator.soft112.com/	2007	Java/Java3D
Bain	https://github.com/OliverColeman/bain	2012	Java
ANNSim	https://github.com/phaysaal/ANNSim	2015	Java

Tabla 1. Características principales del software encontrado

En primer lugar, se analizará el programa usado como guía para este Trabajo de Fin de Grado (TFG), “Java Neural Network Simulator” (JNNS).

Se trata de un simulador de redes neuronales que consiste en una interfaz de usuario escrita en Java, que hace uso del kernel “Stuttgart Neural Network Simulator” escrito en C. Fue desarrollado en torno al año 1999, por el “Wilhelm-Schickard-Institute for Computer Science (WSI)” en Tübingen, Alemania.

JNNS permite crear redes neuronales con diferentes capas de neuronas, incluyendo capas ocultas. Permite la elección de diferentes algoritmos de aprendizaje y archivos con datos de entrenamiento, así como diferentes parámetros a seguir durante el entrenamiento de la red.

El aspecto más interesante de este simulador es la facilidad para la obtención de resultados y su visualización en gráficas en función de los ciclos usados para el aprendizaje, siendo tremendamente didáctico.

El problema de JNNS y de los siguientes proyectos presentados, es la obsolescencia de las tecnologías usadas para crearlos, ya que en este caso hace uso de versiones muy antiguas de Java y, a su vez, deben ser ejecutados en sistemas operativos también antiguos.

El programa “Interactive Neural Network Simulator”, desarrollado por alumnos de Charles University de Praga, en el año 2007. Este software está disponible para Linux, Mac y Windows, y es de licencia libre.

Se trata de un simulador interactivo de redes neuronales escrito en los lenguajes Java y Java 3D. Este programa sería semejante a JNNS, ya que permite la creación de redes neuronales y su entrenamiento de manera interactiva y visual, también permite la carga de datos para ser usados en el entrenamiento de la red y la programación de algoritmos de aprendizaje. Sin embargo, falla al abrir y crear algunas redes neuronales y se cierra inesperadamente sin mensaje previo de error, presumiblemente por la obsolescencia de la tecnología usada.

Además de esto, el programa en ocasiones es poco intuitivo y difícil de usar, especialmente a la hora de visualizar resultados.

Otro de los softwares a analizar es ANNSim (Artificial Neural Network Simulator), un simulador visual de redes neuronales artificiales escrito en Java y desarrollado por Mahmudul FAISAL Al Ameen en el año 2015.

Este software tiene el problema de que no ofrece ningún tipo de documentación y aparenta ser un trabajo poco profesional. Por otro lado, tampoco ofrece muchas opciones de configuración en cuanto a algoritmos de aprendizaje y visualización de resultados.

El último software es Bain, un simulador de redes neuronales escrito en Java y desarrollado por Oliver Coleman sobre el año 2012 como parte de su doctorado. Este software fue diseñado para simular redes neuronales y ofrecer un framework para introducir y modificar diferentes parámetros relativos a la red y su entrenamiento, también permite el uso de GPUs.

El problema de este simulador es que usa una versión antigua de Java y existen problemas a la hora de construir el proyecto con los plugins de Gradle, que se trata de un sistema de automatización de construcción de código de software del que hace uso Bain.

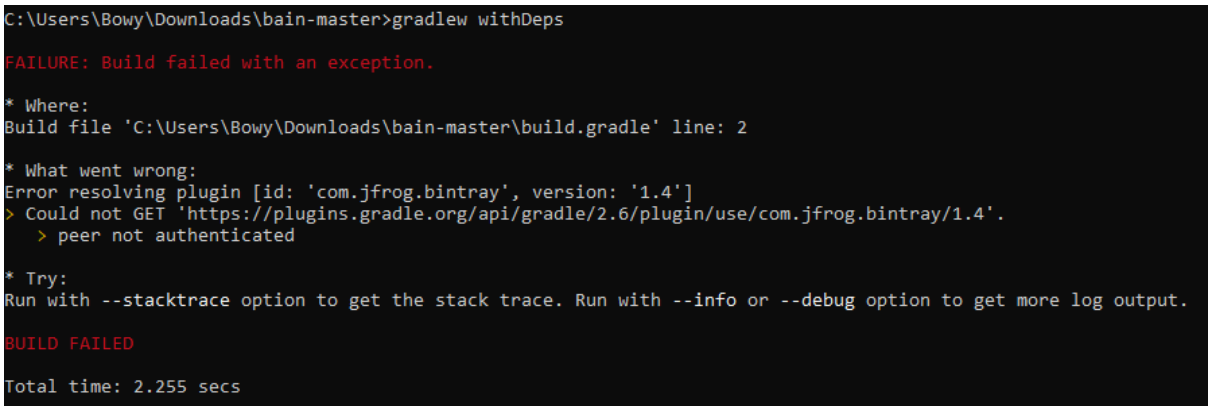


Figura 1. Problema al construir el proyecto de Bain

Por otro lado, se han analizado 4 entornos de desarrollo dentro del ámbito en el que se encuentra el problema a resolver.

Para empezar, TensorFlow, la gran alternativa a Pytorch. Se trata de una biblioteca de código abierto desarrollada por Google para ser utilizada en el aprendizaje automático con redes neuronales. Es implementado principalmente en el lenguaje de programación Python.

TensorFlow es multiplataforma, puede ser usado en Windows, Linux, macOS, Android e iOS, y permite trabajar tanto con CPUs, como con GPUs.

Por otro lado, está Keras, que también se trata de una biblioteca de redes neuronales de código abierto desarrollada en Python. Aunque esta puede funcionar de manera

independiente, también puede ejecutarse sobre TensorFlow Microsoft Cognitive Toolkit y Theano.

Fue lanzada en 2015, teniendo como autor principal a François Chollet, un ingeniero de Google. En 2017, TensorFlow de Google ofreció soporte a Keras, por lo que puede ejecutarse sobre TensorFlow, así como sobre Microsoft Cognitive Toolkit y Theano. Pese a esto, Keras también puede funcionar de manera independiente.

Esta tecnología está diseñada y enfocada a ser una interfaz de programación de aplicaciones (API), lo más amigable e intuitiva posible para el usuario.

Otra tecnología para el aprendizaje automático es Scikit-learn, también se trata de una biblioteca de software libre creada para su uso en Python. La primera distribución pública de Scikit-learn apareció en 2010 y su principal autor es David Cournapeau.

El proyecto utiliza diferentes algoritmos de clasificación, análisis de grupos y regresión y está implementado para interaccionar con las bibliotecas NumPy y SciPy.

Scikit-learn usa Cython en determinados algoritmos para optimizar su rendimiento, ya que se trata de un lenguaje que implementa C y C++ en Python.

Por último, JAX, que es quizá la opción menos conocida de las presentadas. Es también una biblioteca de Python para el aprendizaje automático. Su computación numérica se fundamenta en NumPy y dispone de un componente Just-In-Time que optimiza el código para el compilador, lo que deriva en una gran mejora. JAX es creado, utilizado y mantenido por Google.

Como conclusión de este estado del arte, que analiza las tecnologías de aprendizaje profundo más actuales y avanzadas, se observa gran predominancia de Python. Todas las tecnologías presentadas están ideadas para ser utilizadas en este lenguaje, esto se debe a la simplicidad y fácil comprensión de él, al gran auge que ha tenido y que sigue teniendo, y a la gran cantidad de librerías y herramientas de tratamiento de datos de las que ya se dispone en Python.

Por otro lado, destacar la presencia de Google en la mayoría de estas herramientas, por lo que hay que agradecer los aportes y el interés de esta compañía en el desarrollo de aprendizaje automático y la inteligencia artificial en general.

4. Propuesta

Este apartado supone el bloque principal de este TFG, en él, no solo se desarrollan explicaciones sobre el origen y la creación de los datos utilizados en el entrenamiento de las redes, sino que también se expone el trabajo y desarrollo llevado a cabo para la creación, el entrenamiento y la comparación de estas redes en los entornos de H2O y PyTorch.

4.1. Orígenes de los datos

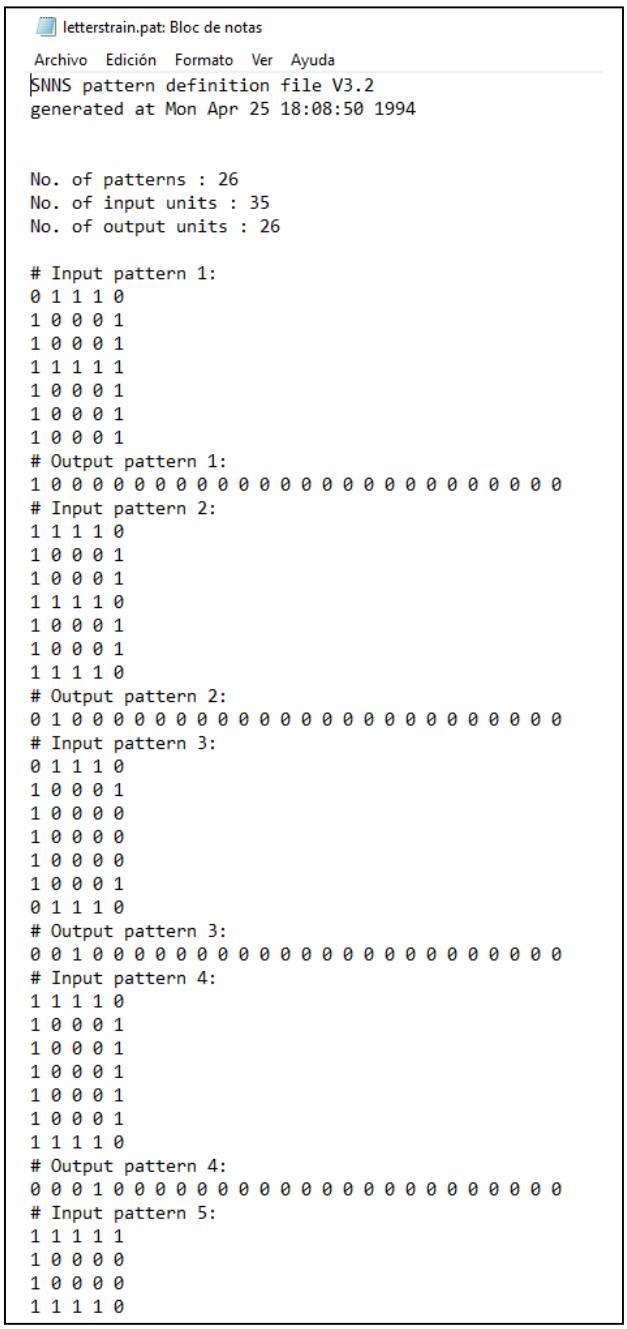
Para realizar el entrenamiento de las redes neuronales propuestas, se necesitan conjuntos de datos de los que las redes puedan aprender.

Para empezar, se ha partido un fichero perteneciente a una práctica académica realizada en la asignatura de Inteligencia Artificial en el Grado en Ingeniería Informática de la Escuela Universitaria Politécnica de Teruel de la Universidad de Zaragoza.

En este fichero se definen los datos de entrada que representan los caracteres a identificar con 1 y 0, es decir, un 1 haría semejanza con un píxel de color negro y un 0 con un píxel en blanco.

Además, en este fichero aparece también la salida que va ligada al patrón de datos. Dicha salida es una estructura de 25 ceros y un 1, en la que el 1 se sitúa en una posición entre la primera y la vigésima sexta, representado a su vez la posición en orden alfabético de la letra referenciada.

Las letras representadas en el fichero son las siguientes en el siguiente orden: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.



letterstrain.pat: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
SNNS pattern definition file V3.2
generated at Mon Apr 25 18:08:50 1994


No. of patterns : 26
No. of input units : 35
No. of output units : 26

# Input pattern 1:
0 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
# Output pattern 1:
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# Input pattern 2:
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
# Output pattern 2:
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# Input pattern 3:
0 1 1 1 0
1 0 0 0 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 1
0 1 1 1 0
# Output pattern 3:
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# Input pattern 4:
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
# Output pattern 4:
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# Input pattern 5:
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 1 1 1 0
```

Figura 2. Fichero con dataset original

El formato del fichero es “.pat”, por lo que se ha modificado y transformado a un formato más extendido y popular como es “.csv”, en el que se ha asignado cada dígito o píxel a una columna del fichero.

De esta manera se ha conseguido disponer de un conjunto de datos para realizar el entrenamiento de la red en un formato más actual y compatible con las tecnologías del momento. Por otro lado, también se requiere de un conjunto de test, que se utiliza para evaluar el rendimiento de la red, y un conjunto de validación, utilizado para ajustar el modelo y sus parámetros.

Debido a que el número de observaciones para el entrenamiento es limitado, se va a prescindir del conjunto de validación, por lo que la distribución de las particiones será del 80% para los datos de entrenamiento y el 20% para el conjunto de test. Para hacer el trabajo que haría el conjunto de validación, se va a utilizar la validación cruzada sobre el conjunto de entrenamiento durante la optimización de los hiperparámetros del modelo, esto se trata de una técnica usada para este ajuste del modelo anteriormente mencionado.

La validación cruzada requiere de un coste computacional muy alto, ya que se ajusta el modelo repetidas veces (3).

Para el uso del dataset en H2O, el patrón de salida ha sido modificado en el fichero ".csv", convirtiendo dicho patrón en la letra que representa. Por ejemplo, el patrón "10000000000000000000000000" se ha cambiado por la letra "a".

Este cambio es meramente aclarativo y estético a la hora de visualizar los resultados. Por otro lado, no ha sido posible aplicarlo en PyTorch, ya que este trabaja con tensores, que son objetos matemáticos o matrices que almacenan datos numéricos, y en este caso, el valor a almacenar es un caracter.

Viendo que inicialmente se disponía de un conjunto de observaciones muy limitado, se ha optado por ampliarlo artificialmente para conseguir 6 datasets distintos, datasets de unas 5000 y unas 10000 observaciones, con dos, tres y cinco posibles errores, explicado de manera más esquemática:

1. 5000 observaciones con hasta 2 posibles errores.
2. 5000 observaciones hasta 3 posibles errores.
3. 5000 observaciones con hasta 5 posibles errores.
4. 10000 observaciones con hasta 2 posibles errores.
5. 10000 observaciones hasta 3 posibles errores.
6. 10000 observaciones con hasta 5 posibles errores.

La creación de estos datasets se ha conseguido, primero, duplicando de forma manual el dataset original hasta conseguir el número de observaciones deseado.

Después, para crear variedad dentro del dataset, es decir, aplicar errores, se ha desarrollado un script que, de forma aleatoria, cambiaba uno, dos o ningún dígito de cada fila.

Ejemplo del script para 2 posibles errores:

```python
import csv
import random

# Función para cambiar un dígito aleatorio por su contrario
def cambiar_digito(numero):
    if numero == '0':
        return '1'
    elif numero == '1':
        return '0'
    else:
        return numero

# Ruta del archivo CSV
archivo_csv = 'dataLetra/letra5k2.csv'

# Leer el archivo CSV y almacenar los datos en una lista
with open(archivo_csv, 'r') as file:
    reader = csv.reader(file, delimiter=';')
    datos = list(reader)

# Iterar sobre cada fila y cambiar un dígito aleatorio
for fila in datos:
    c = 2
    probabilidad = random.randint(0,c)
    posicion_seleccionada = random.randint(0, 35-1)
    valor_original = fila[posicion_seleccionada]
    listaPosiciones = []

    if probabilidad == 0:
        pass
    elif probabilidad == 1:
        fila[posicion_seleccionada] = cambiar_digito(valor_original)
    else:
        fila[posicion_seleccionada] = cambiar_digito(valor_original)

        posicion_seleccionada1 = random.randint(0, 35-1)

        if posicion_seleccionada == posicion_seleccionada1:
            posicion_seleccionada1 = random.randint(0, 35-1)

        valor_original = fila[posicion_seleccionada1]
        fila[posicion_seleccionada1] = cambiar_digito(valor_original)
```

Figura 3. Script para aplicar errores al dataset

Para la realización de las pruebas se han aplicado dos divisiones del conjunto de datos de entrenamiento distintas, una con una proporción del 70% para el conjunto de entrenamiento y 30% para el conjunto de test, y otra división con una proporción del 80% para conjunto de entrenamiento y 20% para el conjunto de test.

Por lo que el dataset quedaría repartido de las siguientes maneras:

- Grupo de datasets con un número de observaciones entorno a las 5.000:

División	Dataset	Número de datos
100%	Total	4.940
70%	Entrenamiento	3.458
30%	Test	1.482

Tabla 2. División de los datos 70-30 en dataset de 5.000 observaciones

División	Dataset	Número de datos
100%	Total	4.940
80%	Entrenamiento	3.952
20%	Test	988

Tabla 3. División de los datos 80-20 en dataset de 5.000 observaciones

- Grupo de datasets con un número de observaciones entorno a las 10.000:

División	Dataset	Número de datos
100%	Total	10010
70%	Entrenamiento	7.007
30%	Test	3.003

Tabla 4. División de los datos 70-30 en dataset de 10.000 observaciones

División	Dataset	Número de datos
100%	Total	10010
80%	Entrenamiento	8.008
20%	Test	2.002

Tabla 5. División de los datos 80-20 en dataset de 10.000 observaciones

4.2. Implementación y entrenamiento

Para realizar el desarrollo de este TFG, se han instalado y configurado los entornos pertinentes y respectivas dependencias.

Una vez que se ha dispuesto tanto de H2O, como de PyTorch de forma totalmente funcional, se ha realizado el desarrollo de las redes neuronales, los entrenamientos y las pruebas realizadas para comprobar su comportamiento.

4.2.1. Instalación y configuración de H2O

El primer entorno en el que se va a trabajar es H2O. Se trata de una plataforma de código abierto de inteligencia artificial, especializada en aprendizaje automático.

H2O ofrece gran cantidad de herramientas, tanto a nivel de investigación, como a nivel empresarial. Por ejemplo, H2O-3, una herramienta de código abierto diseñada para construir y desplegar modelos de aprendizaje automático a gran escala, que integra el uso de lenguajes de programación como R, Python y Java.

Además, soporta gran variedad de algoritmos de aprendizaje profundo y es capaz de ajustar de manera automática los hiperparámetros.

Para este caso, se ha utilizado H2O Flow, una interfaz de usuario de código abierto para H2O, que consiste en un entorno web interactivo que permite al usuario combinar ejecución de código, texto, matemáticas, etc., en un solo documento.

H2O Flow ofrece una REST API y diferentes scripts en R que permiten a cualquier usuario implementar redes neuronales y seguir su desarrollo, incluso si este no tiene experiencia en la programación (4).

Para su instalación, se ha accedido a la web de descargas oficial https://h2o.ai/resources/download/ , y se ha hecho click en la última versión estable de H2O.

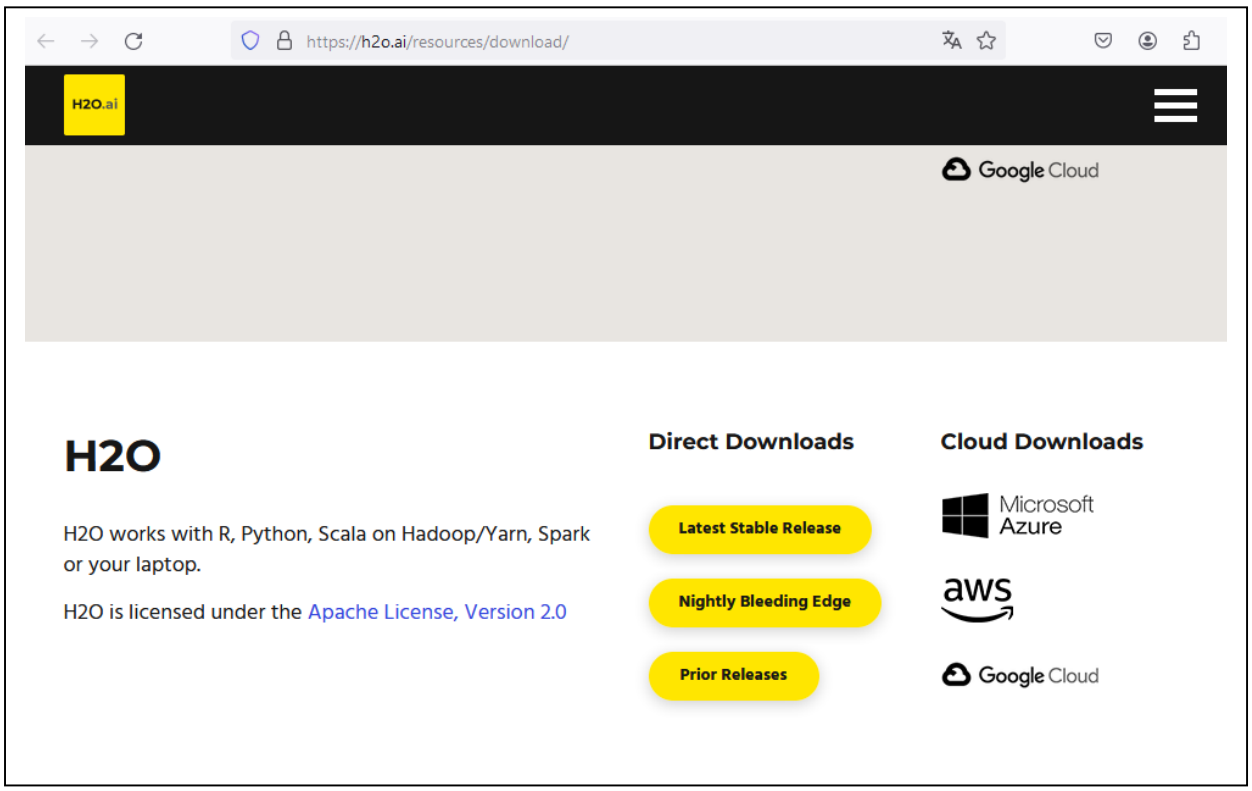


Figura 4. Web de descargas oficial de H2O

Después en la siguiente página se ha descargado dicha versión y se han seguido los pasos especificados para ejecutar H2O y acceder a su interfaz web. Que son los siguientes:

1. Descomprimir el fichero .zip descargado.
2. Acceder a la carpeta descomprimida.
3. Ejecutar el fichero .jar de H2O con la instrucción:

```
java –jar h2o.jar
```

4. Abrir el navegador y acceder a http://localhost:54321.

4.2.2. Instalación y configuración de PyTorch

El segundo entorno utilizado para el entrenamiento de la red neuronal es PyTorch. Como se comentaba en la introducción, este entorno es un marco de aprendizaje profundo de código abierto basado en software utilizado para la creación de redes neuronales.

PyTorch combina la biblioteca de aprendizaje automático de Torch con una API de alto nivel basada en Python. Se ha consagrado como tecnología líder para las comunidades académicas y de investigación gracias a su flexibilidad y su facilidad de uso, entre otras cosas.

Esta biblioteca es principalmente desarrollada por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR) (5).

PyTorch permite la creación de diferentes tipos de redes neuronales, desde las más simples, hasta las redes neuronales convolucionales más complejas. Además, ofrece procesamiento en GPU, ya que hace uso de tensores, que son estructuras de datos multidimensionales que tienen la capacidad de ser procesados en GPU, lo que permite acelerar sus cálculos.

Debido a que se trata de una librería de Python, lo primero que se debe hacer es descargar el lenguaje, para ello se ha accedido a las descargas ofrecidas por la web oficial de Python https://www.python.org/downloads/ , y se ha descargado la versión 3.11.8, relativa al 06/02/2024.

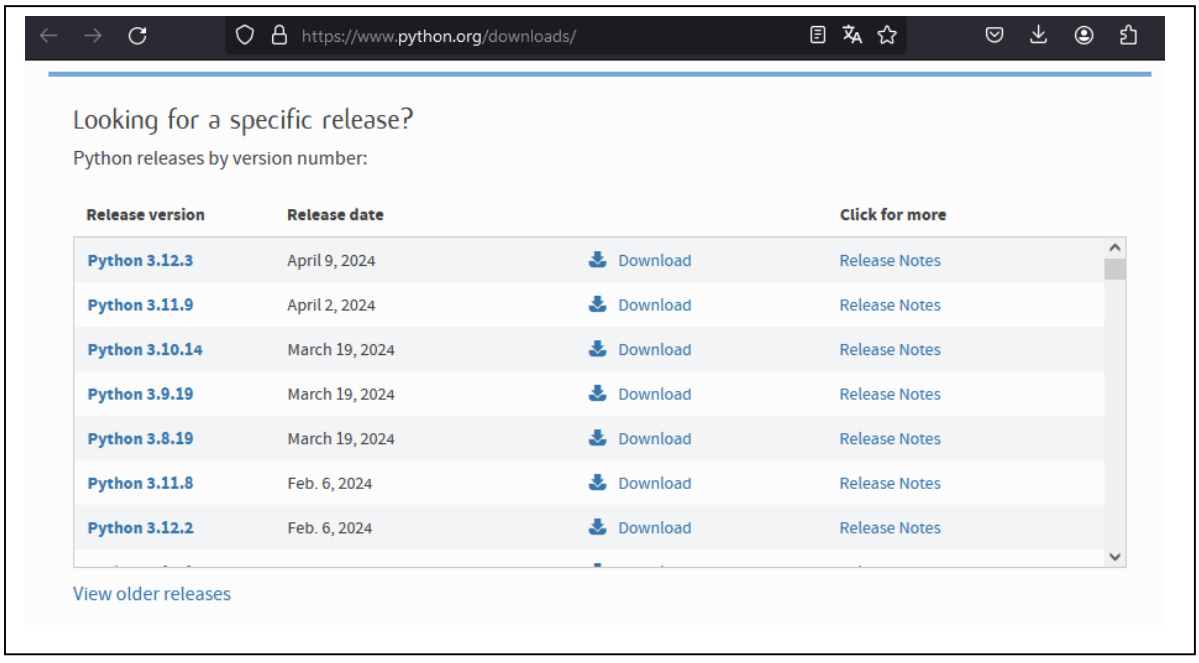
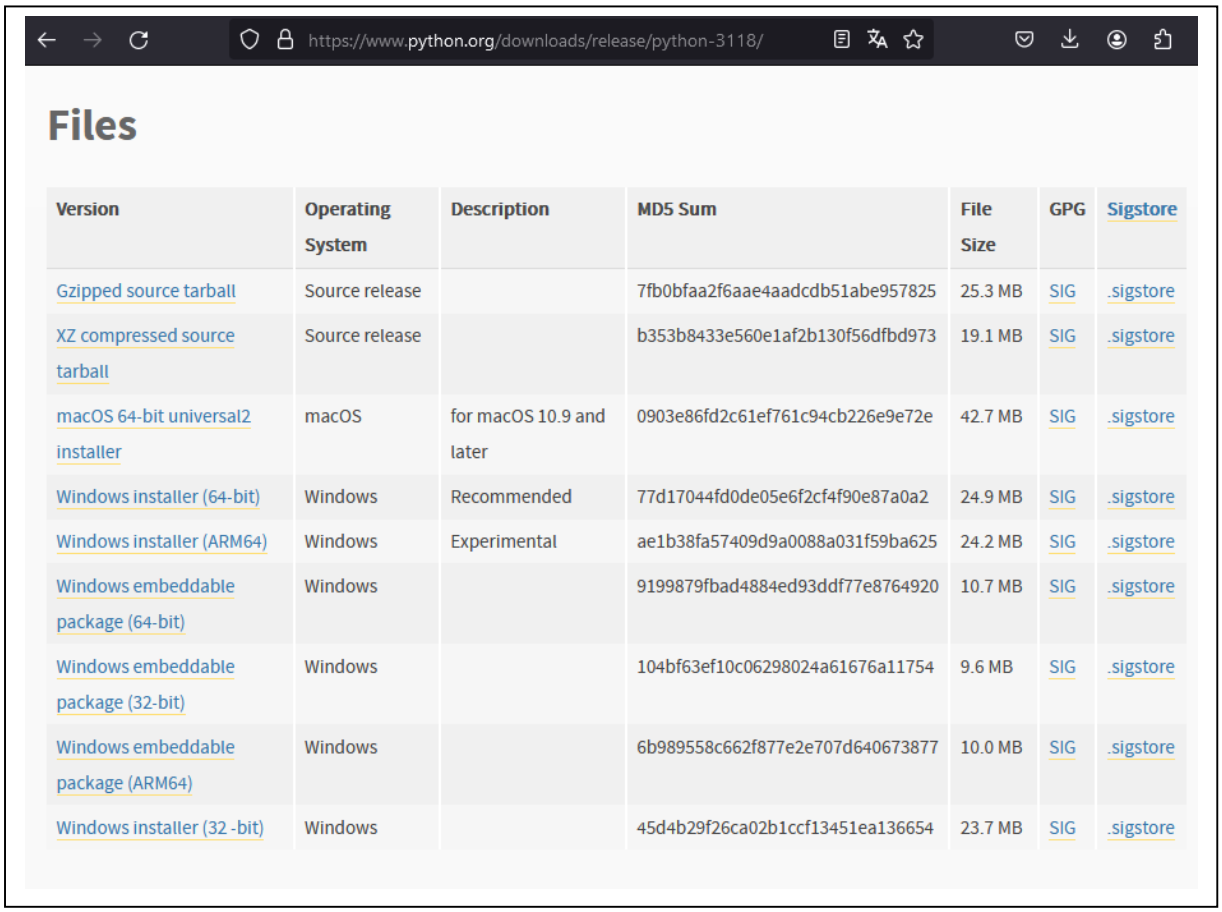


Figura 5. Web de descargas oficial de Python

Después, se ha seleccionado el archivo de descarga “Windows installer (64-bits)”, la descarga recomendada por la web.



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		7fb0bfaa2f6aae4aadcdb51abe957825	25.3 MB	SIG	.sigstore
XZ compressed source tarball	Source release		b353b8433e560e1af2b130f56dfbd973	19.1 MB	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	0903e86fd2c61ef761c94cb226e9e72e	42.7 MB	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	77d17044fd0de05e6f2cf4f90e87a0a2	24.9 MB	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	ae1b38fa57409d9a0088a031f59ba625	24.2 MB	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		9199879fbad4884ed93ddf77e8764920	10.7 MB	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		104bf63ef10c06298024a61676a11754	9.6 MB	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		6b989558c662f877e2e707d640673877	10.0 MB	SIG	.sigstore
Windows installer (32 -bit)	Windows		45d4b29f26ca02b1ccf13451ea136654	23.7 MB	SIG	.sigstore

Figura 6. Web de descargas disponibles de la versión Python 3.11.8

Se ha ejecutado el Setup descargado con las siguientes opciones y se ha elegido “Install Now”.

Figura 7. Herramienta de instalación de Python 3.11.8

Esta instalación realizada también incluye “pip”, un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.

Para la instalación de la librería PyTorch en concreto, se ha accedido también a la web oficial, la cual proporciona un esquema que en función de la versión que se quiere descargar y el entorno en el que se va a usar, muestra el comando requerido para la instalación.

Para el caso se ha descargado la versión estable, la 2.3.0, para Windows, usando el paquete “pip” comentado anteriormente para lenguaje Python.

Además, vemos que en la web se especifica que la última versión de PyToch requiere de la versión Python 3.8 o en adelante.

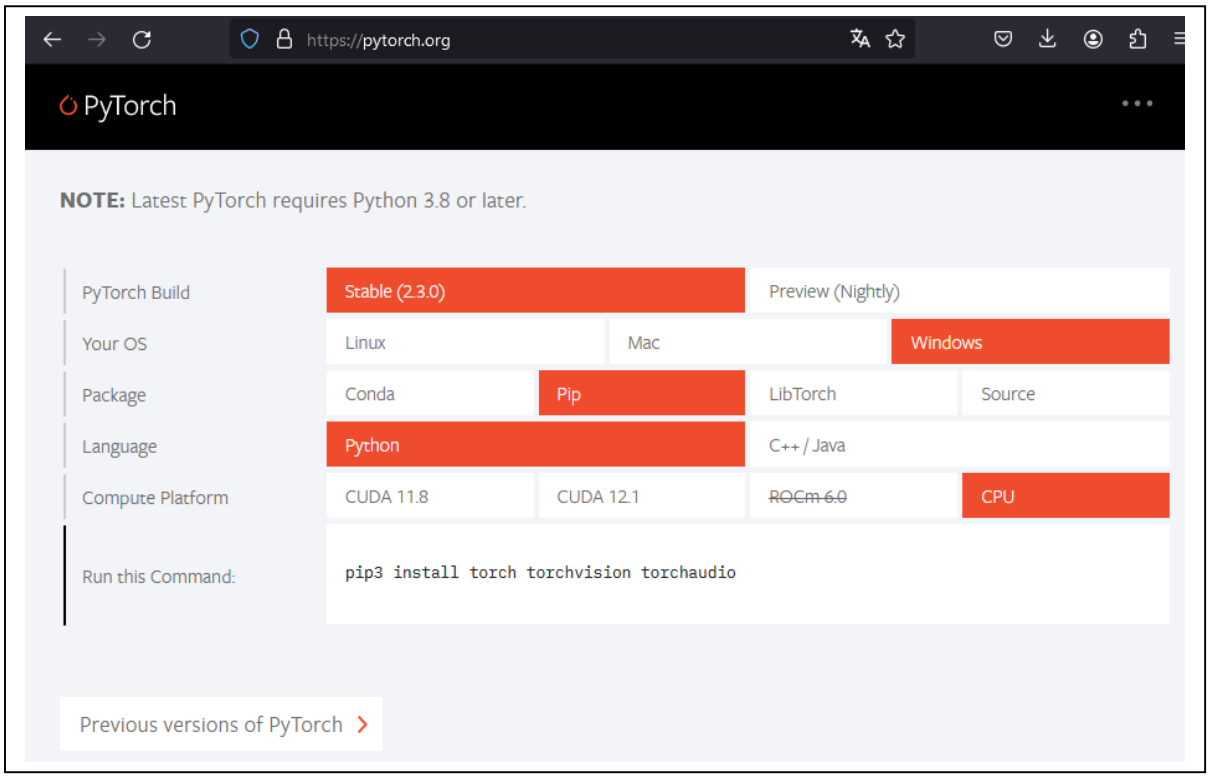


Figura 8. Web de descargas oficial de PyTorch

Instrucción a ejecutar:

```
pip3 install torch torchvision torchaudio
```

Después de realizar esto, para utilizar esta librería bastará con hacer una importación de ésta en el código a desarrollar:

```
import torch
```

Pip también se ha utilizado para la descarga de los siguientes paquetes usados en el desarrollo del software:

- Numpy
- Matplotlib
- Scikit-learn

Todos estos paquetes se han instalado con el siguiente comando:

```
pip install "nombre de paquete"
```

Numpy se ha utilizado para la importación del dataset y el manejo de los conjuntos de datos, ya que se trata de una biblioteca que da soporte para crear vectores y matrices de gran tamaño y varias dimensiones, también dispone de gran variedad de funciones matemáticas.

Por otro lado, está Matplotlib, una biblioteca que ha permitido generar gráficos para hacer más visibles y accesibles los resultados.

Y, por último, Scikit-learn, la biblioteca con la que se ha calculado la regresión lineal y el coeficiente de determinación.

4.2.3 H2O

El uso de H2O viene dado principalmente para realizar una comparación entre los resultados obtenidos en esta plataforma y en PyTorch.

Por otro lado, uno de los objetivos de su utilización es que aquellos estudiantes que no se vean preparados para implementar o desarrollar código relativo a las redes neuronales, puedan también hacer uso de una red para aprender modificando los parámetros de ésta observando los resultados que conlleva.

Para empezar a hacer uso de H2O Flow, lo primero es haber realizado su instalación y haber accedido a la interfaz web en http://localhost:54321.

La interfaz gráfica proporcionada por H2O Flow consta de un menú superior con el que se pueden realizar distintas acciones, como importar archivos o crear modelos, que son las que atañen en este caso.

Figura 9. Barra de menú de H2O Flow

Debajo de este menú, aparece un proyecto por defecto con una barra de tareas para realizar acciones sobre este y un asistente con las acciones más recomendadas e importantes.

Por último, a la derecha de la página web, aparece una sección de ayuda en la que se puede aprender sobre el uso de H2O.

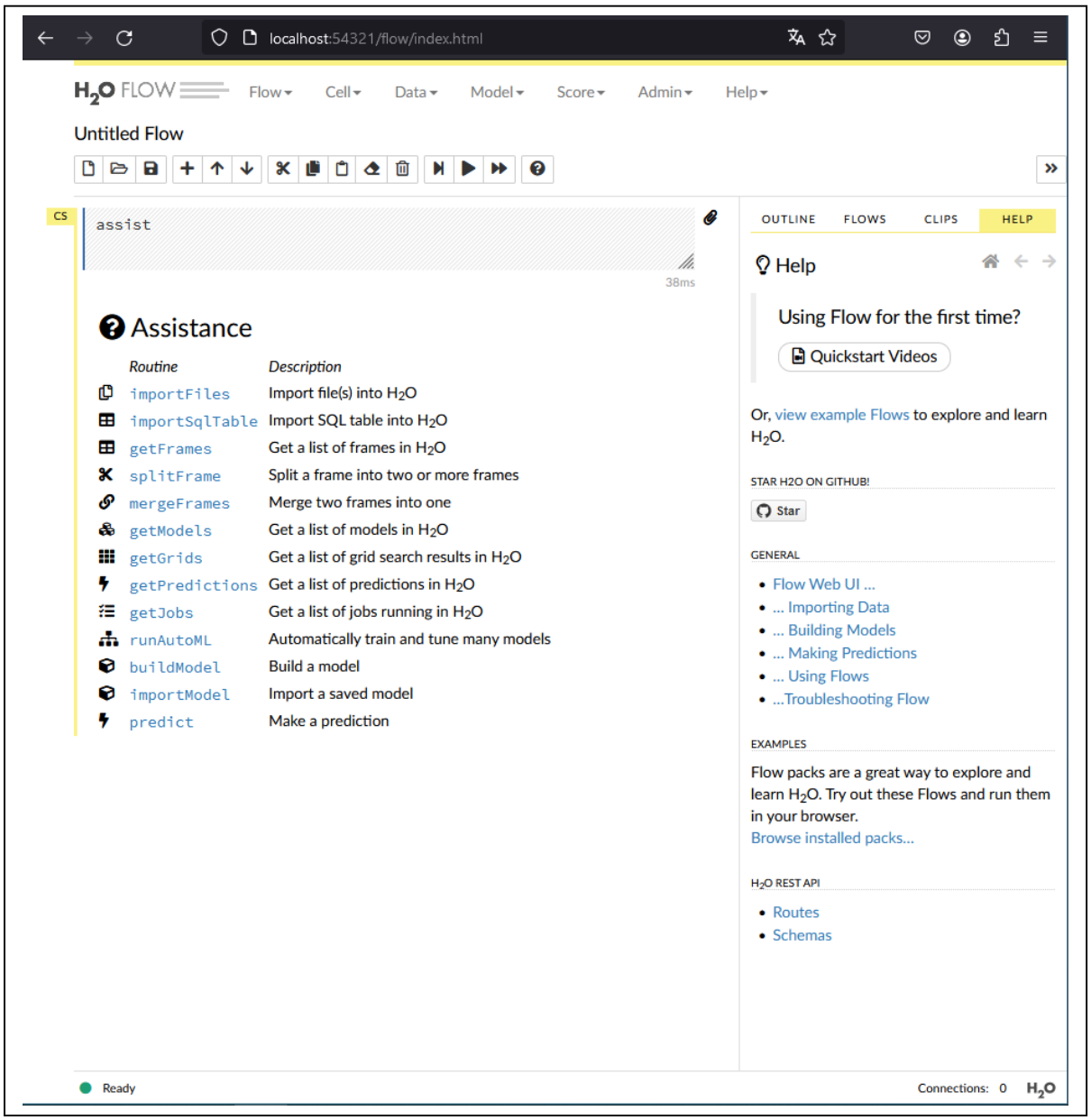


Figura 10. Interfaz gráfica H2O Flow

Para el desarrollo de la red que se busca, lo primero que se ha hecho ha sido importar los archivos con los conjuntos de datos, es decir, los diferentes dataset. Y después se ha hecho “Parse” de los ficheros, donde se ha indicado el tipo de fichero que es, en este caso “csv”, y el separador de columnas dentro del fichero, “;”.

También, se ha indicado el nombre de las columnas y el tipo de dato de éstas. Esto se ha hecho debido a que la estructura del fichero es la siguiente:

```
1row;2row;3row;4row;5row;6row;7row;8row;9row;10row;11row;12row;13row;14row;15row;
0;1;1;1;0;1;0;0;0;1;1;0;0;0;1;1;1;1;1;1;1;0;0;0;1;1;0;0;0;1;1;0;0;0;1;a
1;1;1;1;0;1;0;0;0;1;1;0;0;1;1;1;1;0;1;0;1;0;0;0;1;1;0;0;0;1;1;1;1;1;0;b
0;1;1;1;0;1;0;0;0;1;1;0;0;0;0;1;0;0;0;0;1;0;0;0;0;1;0;0;0;1;0;1;1;1;0;c
1;1;1;1;0;1;0;0;0;1;1;0;0;0;1;1;0;0;0;1;1;1;0;0;0;1;1;0;0;1;1;1;1;1;1;0;d
1;1;1;1;1;1;0;0;0;0;1;0;0;0;0;1;1;1;1;0;1;0;0;0;0;1;0;0;0;0;1;1;1;1;1;e
1;1;1;1;1;1;0;0;0;0;1;0;0;0;0;1;1;1;1;0;1;0;0;0;0;0;0;0;0;0;1;0;0;0;0;f
0;1;1;1;0;1;0;0;0;1;1;0;0;0;0;1;0;1;1;1;1;0;0;0;1;1;0;0;0;0;0;1;1;1;0;g
1;1;0;0;1;1;0;0;0;1;1;0;0;0;1;1;1;1;1;1;1;0;0;0;0;1;0;0;0;1;1;0;0;0;1;h
```

Figura 11. Estructura de los datasets utilizados en H2O

Además, se seleccionado la opción de que la primera línea del fichero contenga los nombres de las columnas.

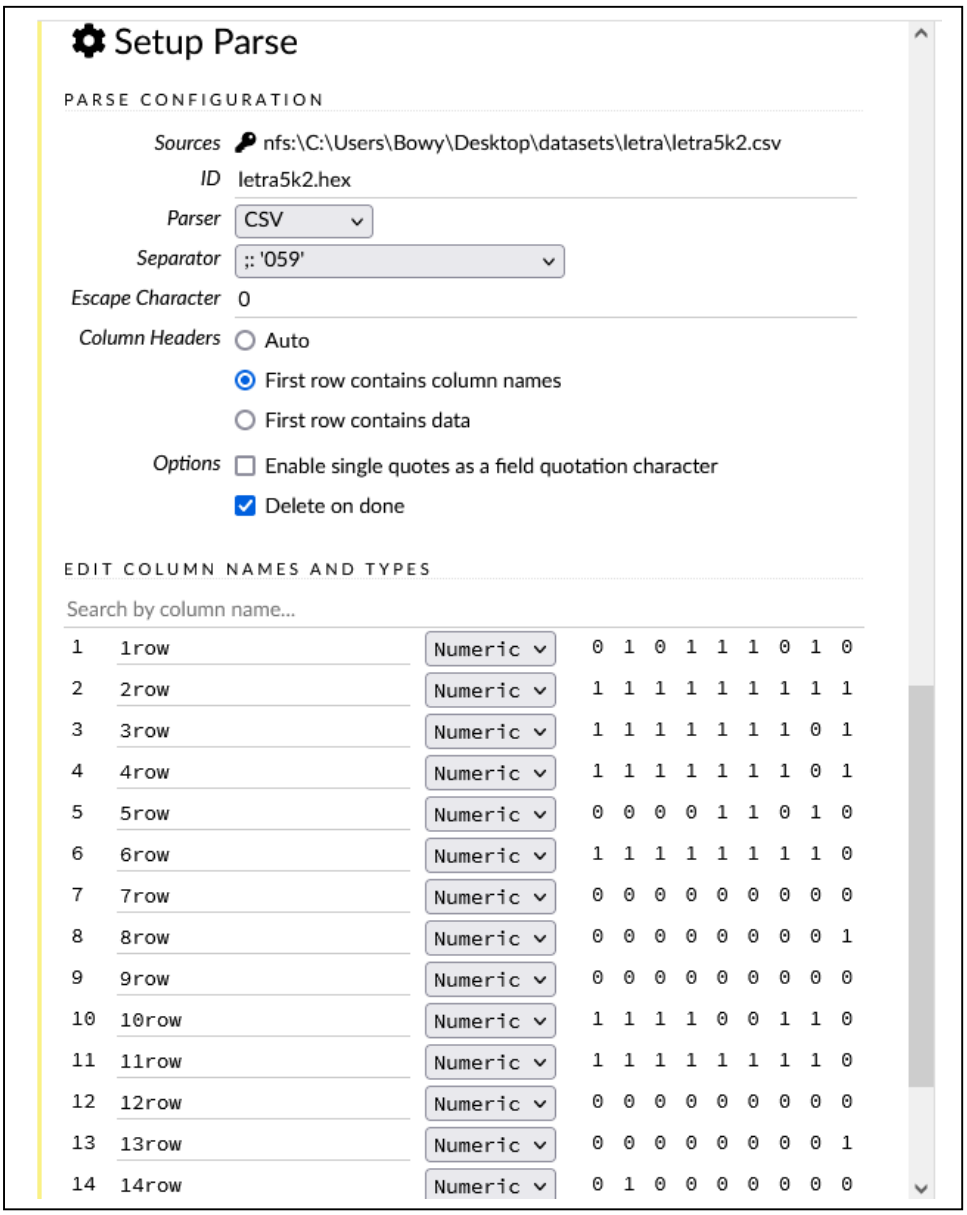


Figura 12. Opción "Parse" de H2O

Seguido de esto y de visualizar el fichero, se ha procedido a hacer la división del fichero en conjunto de entrenamiento y conjunto de validación.

H2O permite seleccionar el porcentaje que se quiere de división, fraccionando así el dataset en dos conjuntos de datos distintos: el conjunto de entrenamiento, que se encargará de proporcionar a la red neuronal los distintos ejemplos u observaciones de los que esta aprenderá patrones y características para realizar sus predicciones; y el conjunto de validación, que se ocupa de encontrar los mejores hiperparámetros posibles de la red neuronal para mejorar su rendimiento.

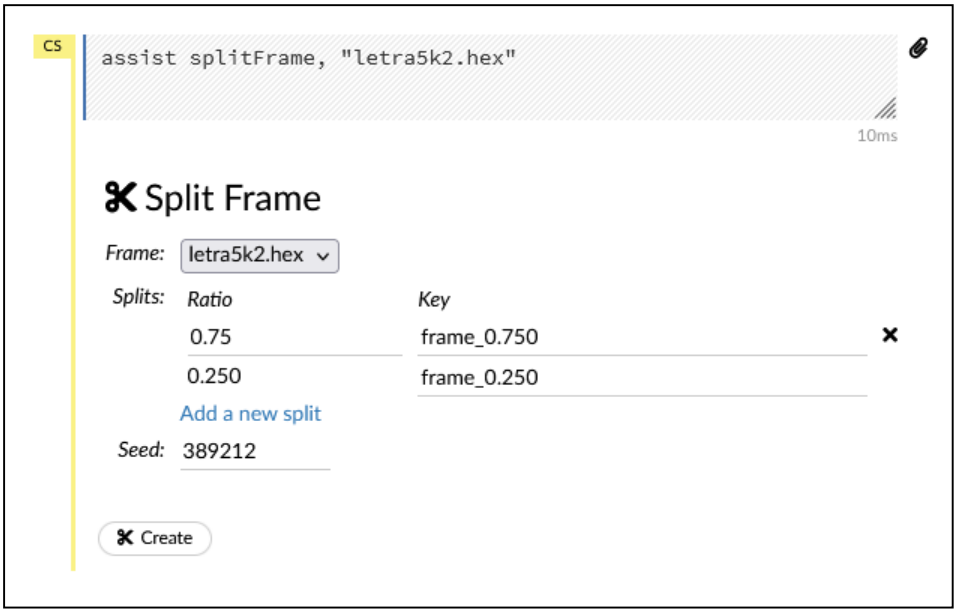


Figura 13. Opción "Split" de H2O

Una vez se han tenido los conjuntos de datos disponibles en la plataforma para que ésta los use, se ha creado el modelo.

H2O ofrece variedad en cuanto a los tipos de modelos de aprendizaje automático disponibles para implementar, como, por ejemplo, bosque de aislamiento (Isolation Forest) o modelos lineales generalizados (Generalized Linear Modeling).

En este caso, se ha seleccionado el modelo de aprendizaje profundo (Deep Learning), ya que este tipo de técnica es capaz de procesar grandes cantidades de datos y aprender de ellos para realizar diferentes tareas, como puede ser identificar patrones.

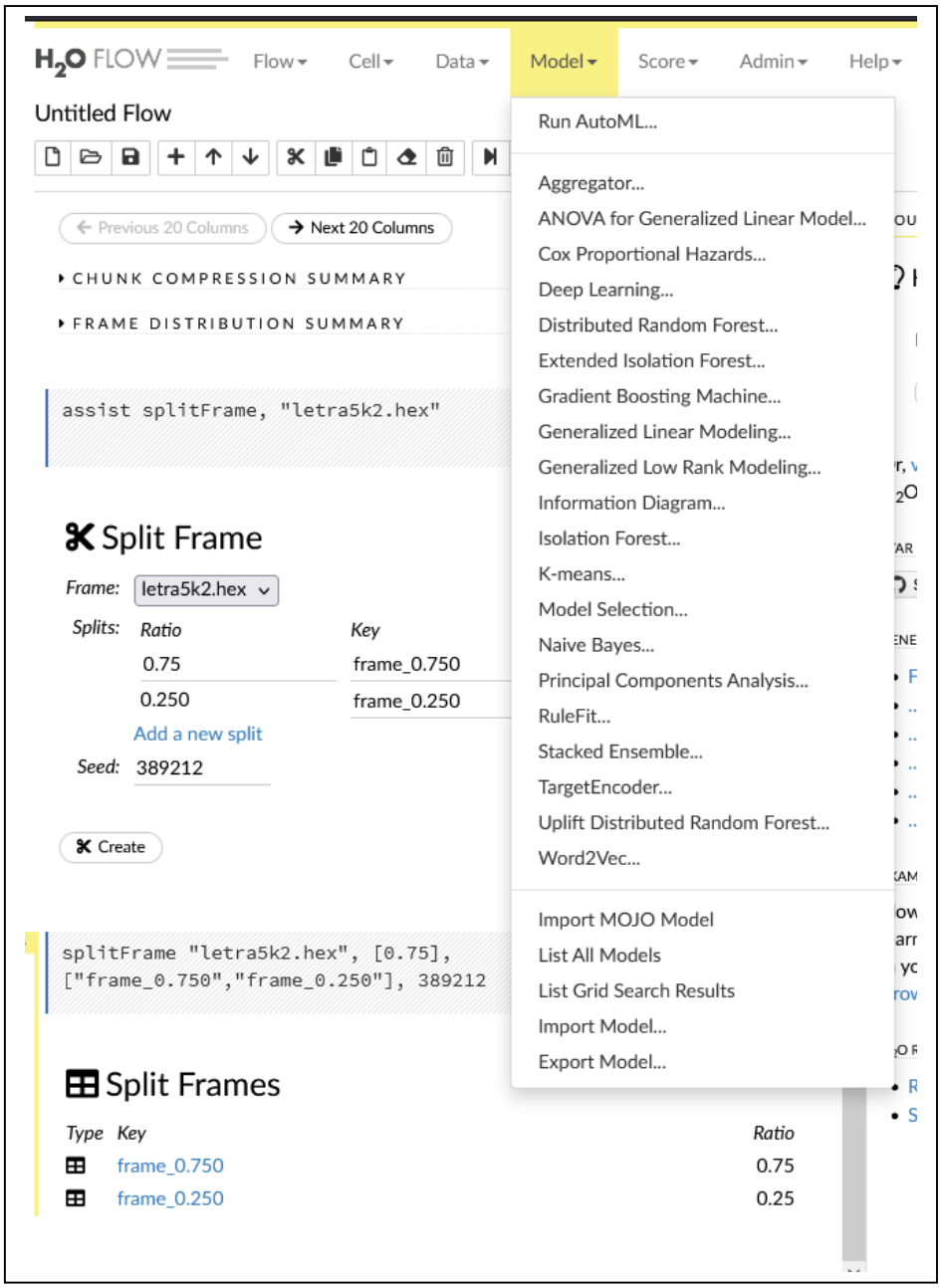


Figura 14. Menú de tipos de modelos disponibles en H2O

Para su creación, se han indicado los conjuntos de entrenamiento y validación anteriormente comentados, y en la opción “response_column”, se ha indicado la columna “output”, ya que es ésta la que contiene las etiquetas con los valores reales, es decir, los que el modelo intentará predecir.

Por otro lado, la plataforma permite seleccionar gran cantidad de parámetros, de los cuáles se ha hecho uso principalmente de los siguientes:

- Hidden: indica las capas ocultas y el número de neuronas que se están introduciendo. Por ejemplo, si se introduce “200, 100”, el modelo tendrá 2 capas ocultas, la primera de 200 neuronas y la segunda de 100 neuronas.
- Epochs: es el número de veces que el modelo va a ser iterado durante el entrenamiento.

- Loss: especifica la función de pérdida a utilizar por el modelo, manera de cuantificar las diferencias entre las predicciones hechas por la red y los valores reales.

Todos los demás parámetros se mantendrán como los proporciona la plataforma por defecto. De entre todos estos, cabe destacar que se va a usar la función de activación “Rectifier”, también conocida como “ReLU”.

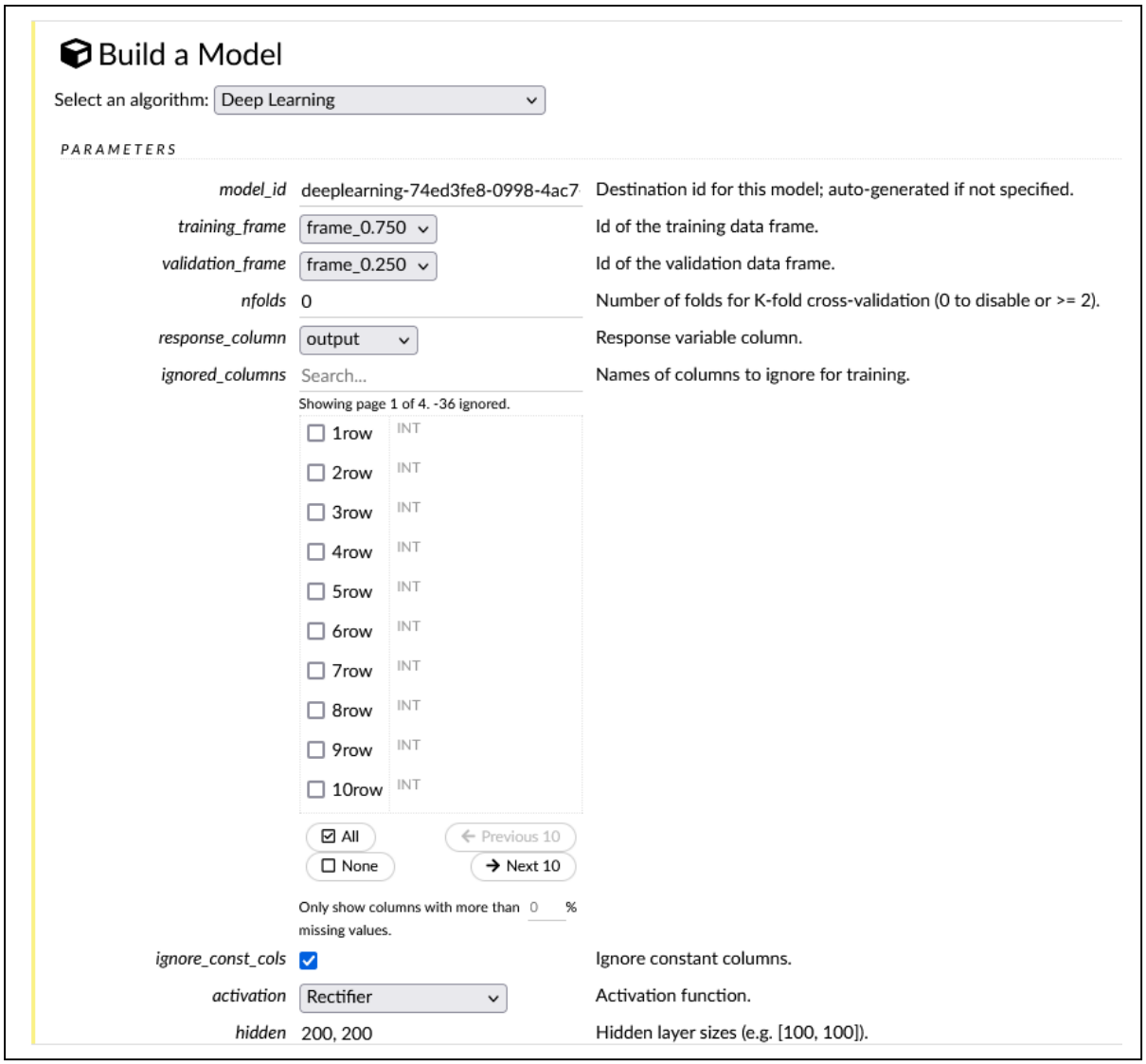


Figura 15. Construcción de un modelo en H2O

Las funciones de activación son funciones matemáticas que se aplican a los nodos o neuronas de la red, su propósito es transformar la señal de entrada que el nodo recibe, en una señal de salida.

Estas funciones permiten aplicar la no linealidad a las redes neuronales, lo que permite que estas aprendan patrones complejos.

En este caso se ha usado la función “ReLU”, que se define como:

$$f(x) = \max(0, x)$$

Donde “x” es la entrada de la neurona.

A la hora de construir el modelo, la plataforma muestra una barra de progreso, una vez que ésta se ha completado, es cuando se puede visualizar el entrenamiento del modelo.

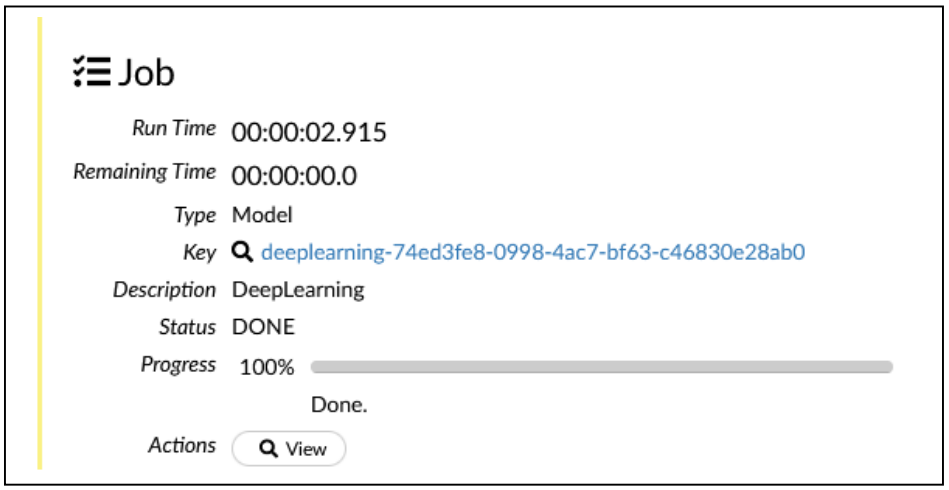


Figura 16. Entrenamiento de un modelo en H2O

A la hora de visualizar el entrenamiento del modelo, H2O ofrece distintas acciones a realizar sobre el modelo y gran cantidad de información, como, por ejemplo, gráficos con información sobre el “loss” e información sobre los parámetros y las variables.

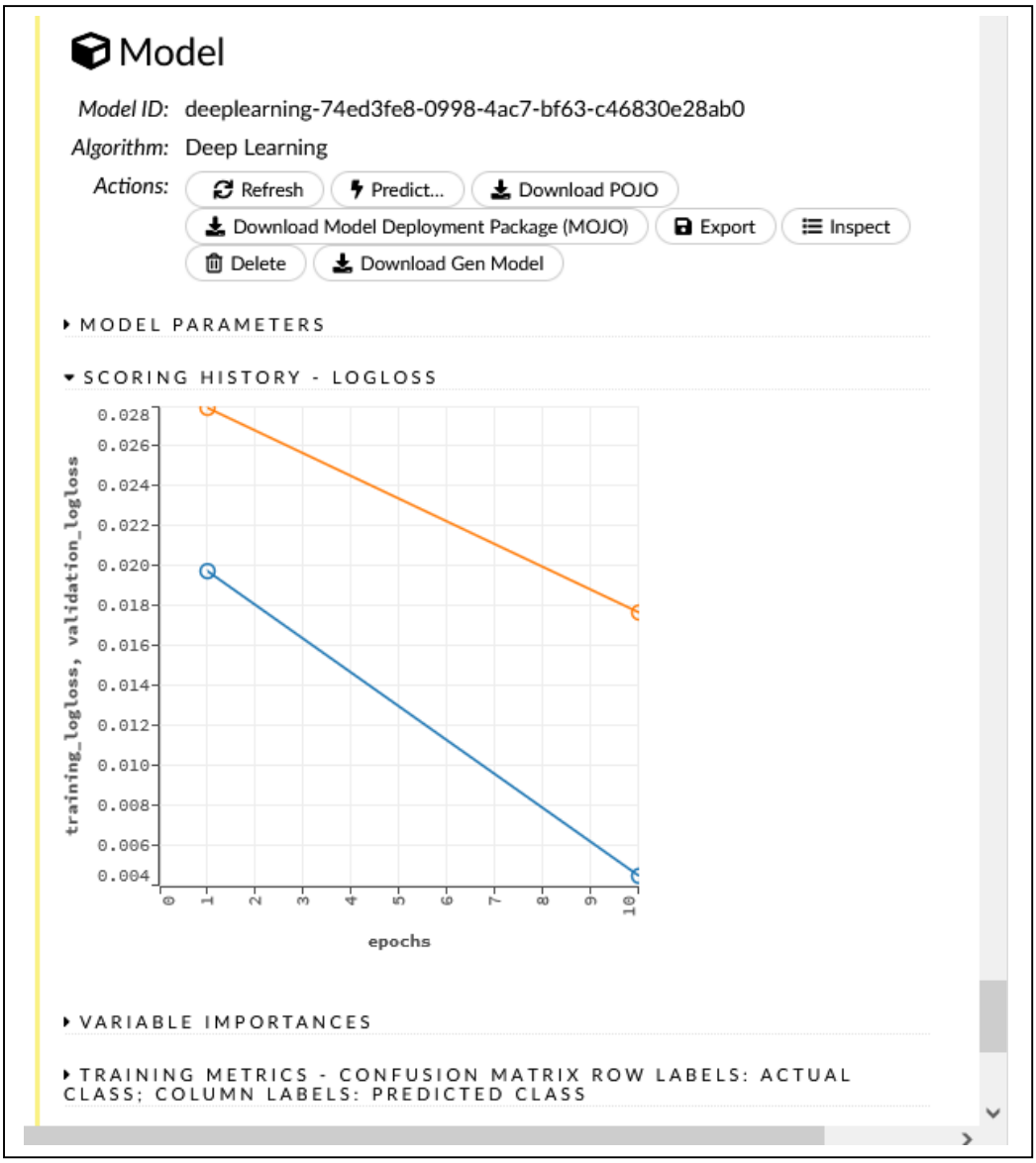


Figura 17. Resultados de un entrenamiento en H2O

Para efectuar las predicciones y comprobar que el modelo las realiza correctamente se ha utilizado la acción “Predict…”.
Además, se ha elegido el frame de validación sacado de la división de los datos comentada anteriormente, y se ha combinado con el frame principal.

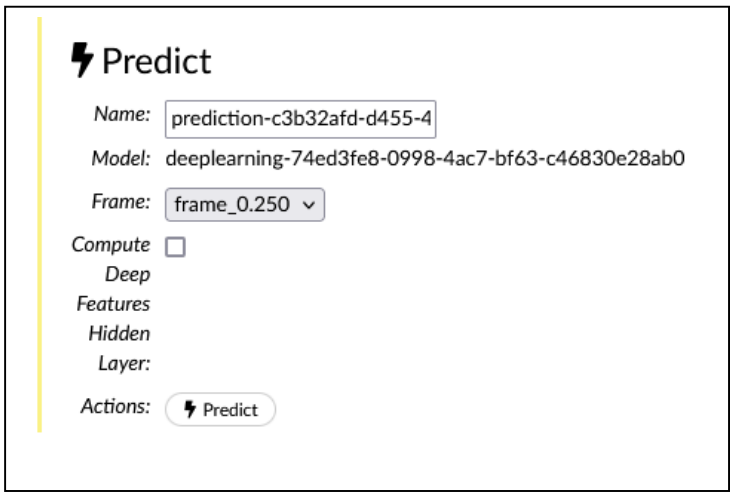


Figura 18. Opción de realizar una predicción en H2O

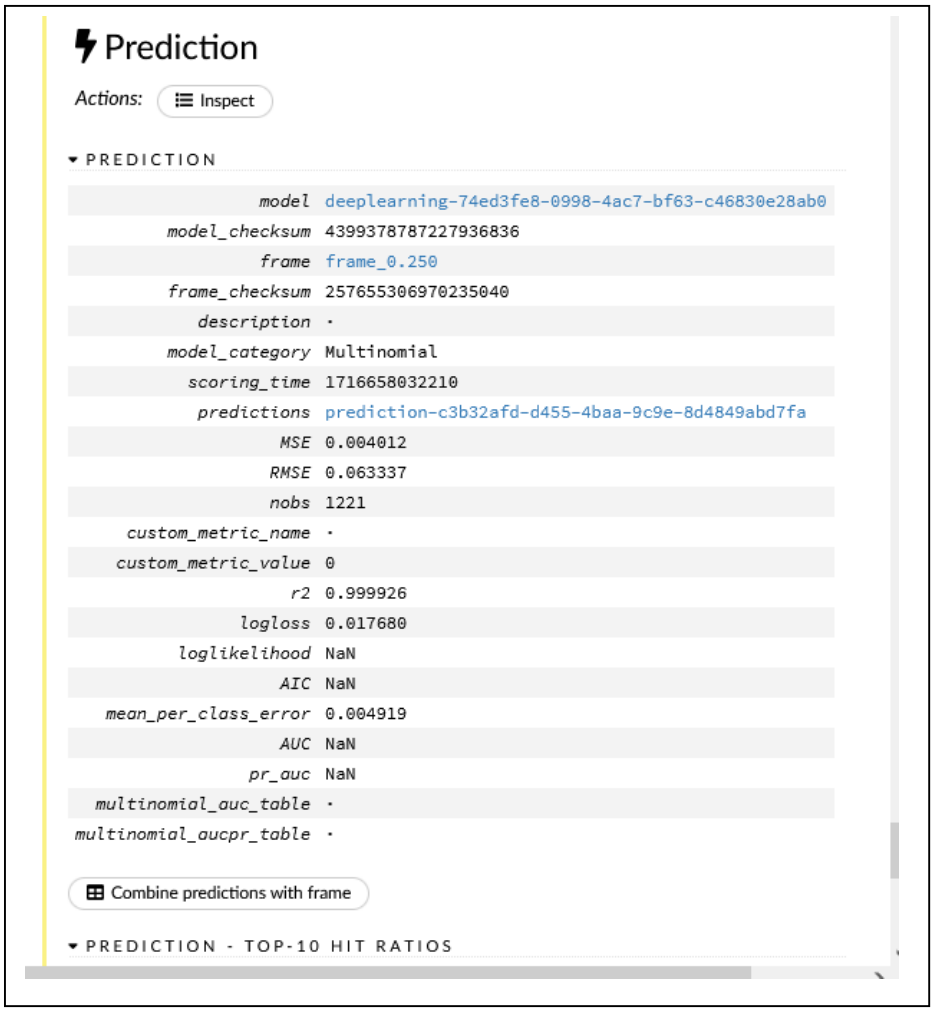


Figura 19. Predicción realizada en H2O

Después de combinarlo, H2O Flow nos da la opción de exportar los resultados de la predicción, con los que, gracias al desarrollo de un script en Python que compara los valores predichos con los valores reales, se ha calculado la precisión de la predicción hecha por el modelo.

Figura 20. Exportar frame en H2O

```python
#
#   Víctor Bouhaben
#   25/05/2024
#   Contexto: TFG Ingeniería Informática EUPT
#
#   Programa para calcular la precición de un modelo de H2O,
#   compara la columna con el valor predicho con la columna con
#   el valor real.
#

import pandas as pd

# Cargar el archivo CSV
df = pd.read_csv('C:\\Users\\Bowy\\Desktop\\resultados.csv')

columna1 = df['predict']
columna2 = df['output']

aciertos = 0

# Comparar las dos columnas
comparacion = columna1 == columna2
aciertos = comparacion.sum()

print("Precisión: " + str((aciertos / len(df) *100)) + " %")
```

Figura 21. Script para calcular la precisión de una predicción de H2O

4.2.4 PyTorch

El desarrollo del programa con PyTorch es el objetivo final de este Trabajo de Fin de Grado (TFG), ya que será el que utilicen principalmente los alumnos para aprender sobre las redes neuronales.

Para sus fines en el ámbito de la enseñanza, se ha buscado crear un simulador lo más entendible y legible para cualquier usuario.

La base de este software es el modelo de red neuronal artificial creado, que se trata de un Perceptrón Multicapa, o en inglés Multilayer Perceptron (MLP), un tipo de red neuronal artificial que supone una mejora en cuanto al Perceptrón, ya que el Perceptrón Multicapa es capaz de resolver problemas no lineales gracias a que está compuesto de capas.

Este modelo se ha implementado de dos formas distintas: sin capa oculta y con capa oculta.

La implementación sin capa oculta se basa en una capa de entrada de 35 neuronas, y una de salida con 26 neuronas, ambas capas totalmente conectadas. Estas neuronas hacen referencia a los dígitos de entrada y salida del dataset, respectivamente.

Las capas de este modelo disponen de una función de activación que se encarga de transformar las salidas a un valor entre 0 y 1, la función “Sigmoid”, que es especialmente útil cuando se desarrollan tareas de clasificación binaria, lo que es el caso, ya que a cada neurona de salida le corresponde el valor 0 o 1.

La función “Sigmoid” viene definida por la siguiente función:

$$P(t) = \frac{1}{1 + e^{-t}}$$

Por otro lado, el modelo con capa oculta, implementa tres capas distintas, la de entrada, la oculta y la de salida. La capa ocupa permite mejora en aprendizaje y la clasificación del modelo, y permite a este aprender patrones más complejos.

En este caso, el modelo hace uso de las funciones de activación explicadas anteriormente, tanto de la función “ReLU”, como de la función “Sigmoid”, de manera respectiva.

Ambos modelos trabajan con los mismos parámetros y características. Para empezar, la función de pérdida usada es la función de entropía cruzada binaria.

La entropía cruzada es una función que se encarga de comparar la distribución de probabilidad de los datos de salida con la distribución de probabilidad real, es decir, los datos reales. Para el caso, se ha utilizado un tipo de entropía, que ofrece PyTorch, destinada a la clasificación de datos binarios (6).

La función de entropía cruzada viene definida por la siguiente función:

$$\mathrm{H}(p, q) = \mathrm{E}_p[-\log q] = \mathrm{H}(p) + D_{\mathrm{KL}}(p \| q)$$

Siendo “q” la distribución de probabilidad predicha y “p” la real.

Por otro lado, el modelo hace uso del optimizador “Root Mean Square Propagation” (RMSProp), que es el encargado de minimizar los errores cometidos por el modelo, mediante el ajuste de parámetros y pesos (7).

Al optimizador RMSProp, se le pasa como parámetro la tasa de aprendizaje, en inglés, “learning rate”, que se trata del valor de actualización de los pesos durante el aprendizaje, este es un valor de complejo ajuste, debido a que si es excesivamente elevado puede acarrear un mal funcionamiento del modelo, y si es demasiado bajo, el aprendizaje puede ser demasiado largo en el tiempo (8).

El software se ha desarrollado de tal manera que ofrece opción de utilizar tanto la CPU, como la GPU para el aprendizaje de la red neuronal.

Finalmente, el simulador permite variar el número de “epochs” y el tamaño de “batch” para el entrenamiento del modelo.

El número de “epochs” es el número de veces que el modelo itera sobre el conjunto de datos, realizando durante un “epoch”, una iteración hacia delante (forward) y otra hacia detrás (backward), a través del conjunto de datos.

En cuanto al tamaño de “batch”, este se trata del número de observaciones o ejemplos de datos que se utilización en una única iteración del entrenamiento. De este valor puede depender la eficiencia a nivel computacional del entrenamiento del modelo.

En tema de resultados, el simulador ofrece retroalimentación en cuanto al dispositivo sobre el que se está desarrollando el entrenamiento. Dicho de otra manera, si el entrenamiento se está haciendo en la CPU o la GPU.

También, ofrece información sobre el “epoch” del entrenamiento en el que se encuentra y la pérdida que ha tenido en él.

Cuando el entrenamiento finaliza, muestra el tiempo en el que se ha desarrollado e inicia la fase de test, en la que el conjunto de datos de entrada de test es pasado al modelo, y la salida es comparada con los valores reales, lo que permite obtener y mostrar la precisión de predicción del modelo.

Para esto, el software divide el dataset en un conjunto de entrenamiento y otro de test, siendo el porcentaje de estos fácilmente modificable.

Una vez completada la fase de test, el simulador muestra dos gráficos:

- El primer gráfico muestra la evolución de la pérdida conforme avanzan los “epochs”.

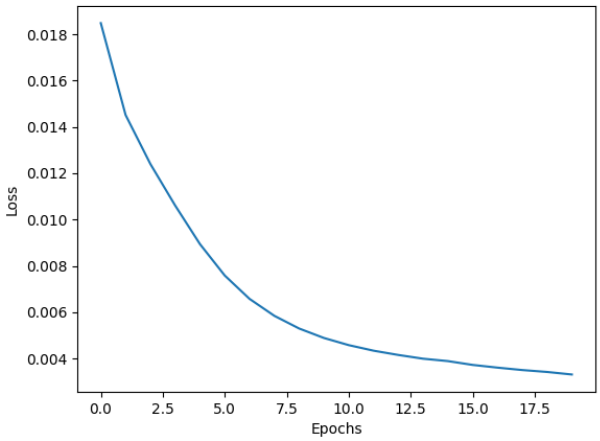


Figura 22. Ejemplo de gráfico de relación Loss-Epochs con PyTorch

- El segundo muestra la regresión lineal y el coeficiente de determinación.

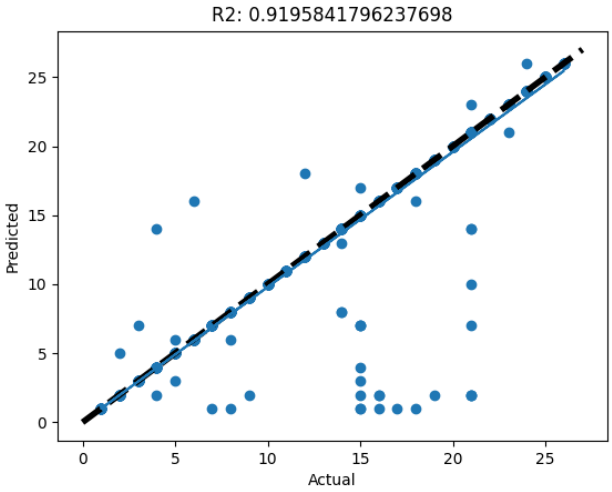


Figura 23. Ejemplo de gráfico de regresión lineal y coeficiente de determinación con PyTorch

5. Resultados

En este apartado se han recopilado todos los resultados obtenidos, tanto en el entrenamiento, como en las predicciones realizadas por las redes neuronales.

Todos estos resultados están diferenciados por los distintos parámetros y características que pueden ser aplicados a los modelos.

Además, todas las pruebas se han realizado con los distintos datasets ya explicados.

5.1 H2O

Los gráficos indicados en este apartado, muestran la cantidad de “epochs” en el eje x, y el valor del “loss” en el eje y. Dicho de otra manera, muestra la evolución del “loss” conforme avanzan los “epochs”.

Además, dichos gráficos muestran en color azul el “loss” durante el entrenamiento, y en naranja, el “loss” durante la validación.

La primera conclusión a sacar después de las pruebas realizadas con dos datasets, es que es totalmente ineficiente entrenar el modelo con una cantidad de epochs que no se encuentre entre 400 y 800, siendo por lo general, aproximadamente 600 el valor óptimo de epochs.

De todo el conjunto de pruebas se han seleccionado los siguientes resultados para ejemplificar lo anterior.

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	5000	Auto

Tabla 6. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,771	0,002371	0,048689	99,729 %

Tabla 7. Resultados modelo 4

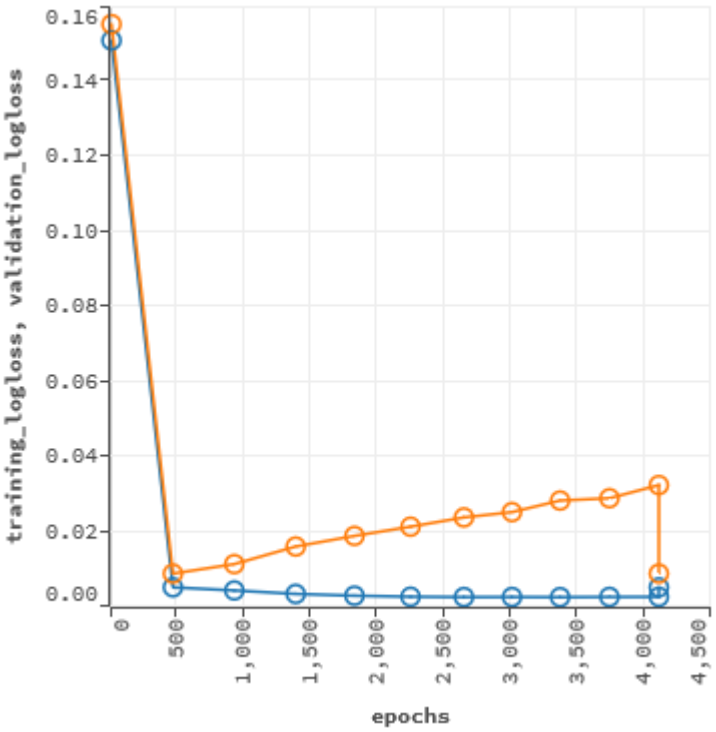


Figura 24. Gráfico modelo 4

Por lo que, de ahora en adelante se han entrenado las redes neuronales con este valor, excepto en aquellos entrenamientos en los que, viendo los resultados, se ha determinado que se necesita un número mayor de epochs para mejorarlos.

Después de analizar otros dos datasets, se ha observado que forzar a H2O a utilizar la función de pérdida “CrossEntropy”, es ligeramente mejor a nivel general que dejar que la plataforma decida que función de pérdida utilizar con la opción “Auto”.

Debido a esto, en las siguientes pruebas se utilizará únicamente la función de entropía cruzada.

Al analizar los dataset de 10.000 observaciones, se ha visualizado que el número de epochs óptimos para no realizar trabajo en vano, está entre 200 y 400.

Finalmente, después de haber realizado todas las pruebas en H2O, se ha concluido que, en este caso, utilizar una capa oculta con neuronas disminuye el tiempo de entrenamiento, pero reduce la precisión de manera mínima.

En cuanto a la división del conjunto de datos, no hay gran diferencia aparente entre en uso de las proporciones 80-20 y 70-30, sin embargo, la proporción 80-20 es ligeramente más precisa.

Todos los resultados del resto de pruebas realizadas de los que se han obtenido estas conclusiones, se han plasmado en el Anexo I.

5.2 PyTorch

Las pruebas realizadas en el simulador están directamente relacionadas con las realizadas en H2O, ya que en función de los resultados y conclusiones obtenidas en H2O aplicando los diferentes hiperparámetros, para PyTorch han sido aplicados únicamente aquellos que ofrecían un mejor rendimiento.

Por lo que, para las pruebas de PyTorch se ha aplicado una proporción de división del conjunto de datos de 80-20, un 80% para el conjunto de datos de entrenamiento y un 20% para el de test. Además, se ha aplicado un número de "epochs" de 600.

Por otro lado, se han implementado dos modelos distintos, uno sin capa oculta y otro con una capa oculta de 10 neuronas. A ambos modelos se les ha aplicado el optimizador con una tasa de aprendizaje (learning rate), con un valor de 0,01 y 0,1.

El entrenamiento se ha realizado inicialmente con un tamaño de "batch" de 35, ya que está ampliamente extendido el uso de la cantidad de neuronas de la capa de entrada como tamaño de "batch".

Después de realizar las pruebas con el primer dataset, la primera conclusión clara es que no se necesita de un número de "epochs" tan elevado como en H2O, es posible que esto se deba a PyTorch es una tecnología más puntera y que está siendo ejecutado directamente sobre el intérprete, sin ninguna API adicional, como pudiera ser la interfaz H2O Flow.

Viendo esto, se ha aplicado un número de "epochs" de 100. Aunque 600 "epochs" pueda vislumbrar una cantidad menor de pérdida, se ha considerado que esta disminución es demasiado baja en comparación con el coste computacional y temporal que la provoca.

Además, observando los resultados de la última prueba del primer dataset, se puede intuir un mal funcionamiento, pese a que se ha obtenido una alta precisión, del 96,916%, no llega a ser tan elevada como en las demás pruebas. Esta alteración en el entrenamiento se puede deber a un valor excesivamente alto de la tasa de aprendizaje, por lo que, en siguientes pruebas se ha utilizado únicamente el valor 0,01 como dicha tasa.

De todo el conjunto de pruebas se han seleccionado los siguientes resultados para ejemplificar lo anterior. Cabe recalcar como ejemplo del mal funcionamiento, el valor -0,089 del coeficiente de determinación

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	10	600	35	0,1

Tabla 8. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
47,225	-0,089	96,916 %

Tabla 9. Resultados modelo 4

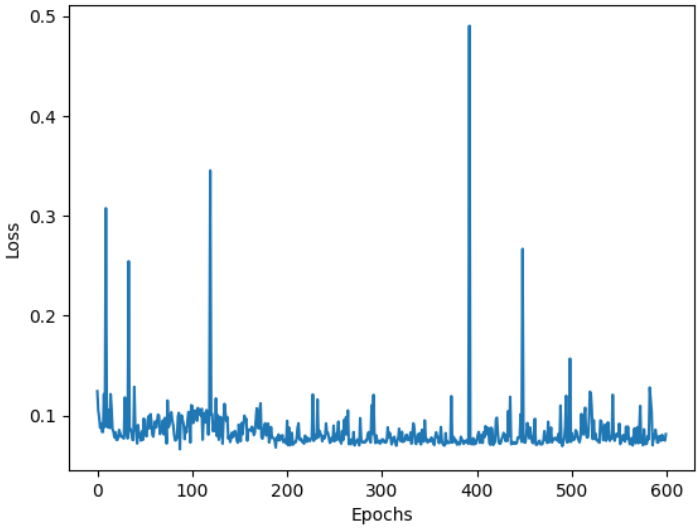


Figura 25. Gráfico de pérdida modelo 4

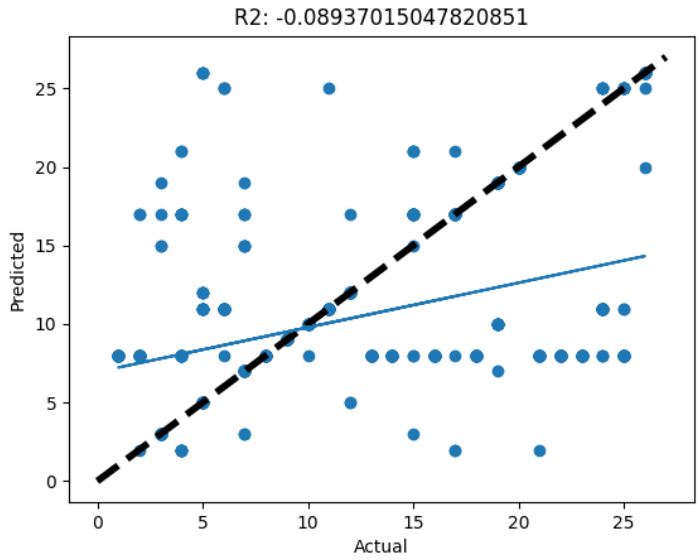


Figura 26. Gráfico de R2 modelo 4

Al concluir las pruebas con los datasets de 5.000 observaciones, se ha identificado que el número de “epochs” puede ser disminuido aún más, ya que en algunos casos no solo no es lo suficientemente eficiente, sino que genera un aumento en la pérdida. Así pues, se ha optado por bajar esta cifra a 10.

Por otro lado, se ha observado que no existe prácticamente cambio al usar o no capa oculta, por lo que se ha decidido aumentas las neuronas de la capa oculta a 128, para de esta manera, observar la diferencia en el uso o no de esta capa.

Después de las pruebas con el primer dataset de 10.000 observaciones, se ha decidido variar el tamaño de “batch”, alternando entre 35 y 300.

Tras haber realizado las pruebas con los datasets de 10.000, se ha apreciado que el tiempo se reduce sobre manera al aumentar el tamaño de “batch”, esto es algo que se podía presuponer en un principio, sin embargo, se ha de tener cuidado ya que puede reducir la eficiencia del entrenamiento.

Con respecto al aumento de las neuronas en la capa oculta, ha provocado una mejora en los resultados a nivel de predicción. También ha estabilizado dichos resultados, haciendo al modelo más resistente a cambios en otros hiperparámetros.

Se ha seleccionado la última prueba realizada para mostrar la evolución de los resultados.

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	128	10	300	0,01

Tabla 10. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
0,370	0,952	99,717 %

Tabla 11. Resultados modelo 4

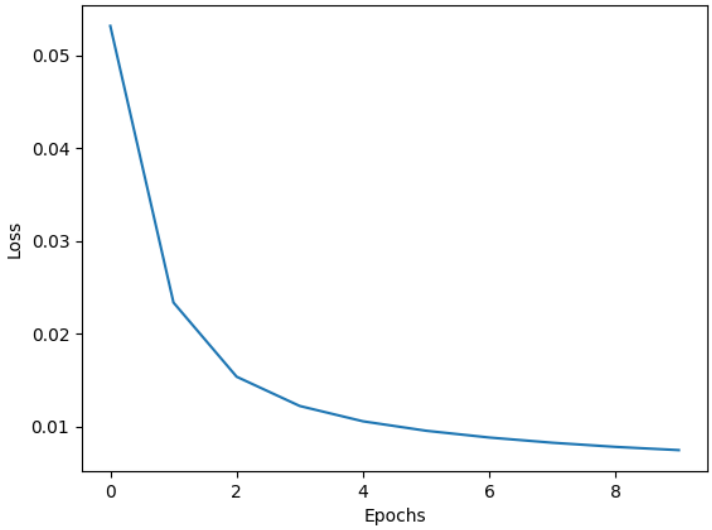


Figura 27. Gráfica de pérdida modelo 4

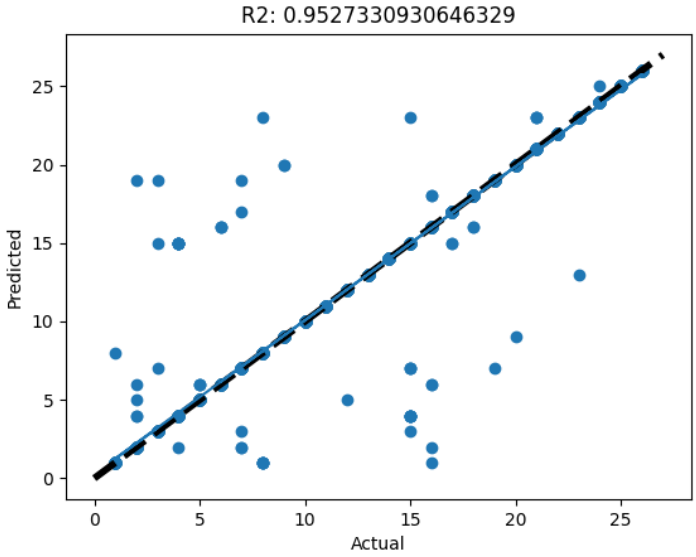


Figura 28. Gráfico de R2 modelo 4

Todos los resultados del resto de pruebas realizadas de los que se han obtenido estas conclusiones, se han plasmado en el Anexo II.

6 Licencia Software y Documental

Llegados a este apartado, se va a proceder a comentar tanto la licencia de software como la licencia documental.

En cuanto a la licencia de software se va a emplear Berkeley Software Distribution (BSD). Se trata de una licencia de software libre permisiva como puede ser OpenSSL o la MIT License. Existen diferentes tipos de licencias, en el caso de este TFG se ha utilizado la licencia “BSD modificada”, “BSD revisada”, “BSD-3” o “BSD de 3 cláusulas”.

Figura 29. Logotipo de la licencia BSD

Al igual que sucede en el mundo del software, se tienen que buscar formas de garantizar las libertades asociadas al trabajo elaborado y su inviolabilidad futura. Para garantizar que la libertad esté asociada al documento se buscan métodos, uno de ellos es la licencia GNU Free Documentation License GFDL).

Figura 30. Logotipo de la licencia GNU

El propósito de esta Licencia es hacer que en el caso de este TFG sea ’gratuito’ en el sentido de libertad: para asegurar a todos la libertad efectiva de copiarlo y redistribuirlo, con o sin modificarlo, ya sea comercial o no comercialmente. En segundo lugar, esta licencia preserva para el autor y el editor una forma de obtener crédito por su trabajo, sin ser considerado responsable de las modificaciones realizadas por otros. Es una especie de ’copyleft’, lo que significa que las obras derivadas del documento deben ser libres en el mismo sentido. Si por algún motivo se emplea este documento y se modifica, debe realizar una serie de acciones indicadas en el sitio web oficial de GNU.

Tampoco hay que olvidar que este documento, por defecto, está al amparo de la licencia 7.3, por su inclusión en el Repositorio Institucional de Documentos de la Universidad de Zaragoza: ZAGUAN

Figura 31. Licencia de ZAGUAN

7 Conclusiones y Trabajo Futuro

En este TFG se ha abordado el problema de la obsolescencia y antigüedad de los simuladores de redes neuronales existentes y la necesidad de estos para el ámbito docente.

Asimismo, se ha afrontado la creación, tanto de un conjunto de datos semejantes a la representación de las letras del abecedario a nivel de imagen con píxeles, como de una red neuronal artificial capaz de procesarlo e identificar de qué letra se trata, mostrando resultados lo más claros y concluyentes posibles para que puedan ser interpretados por los alumnos.

El desarrollo de este trabajo ha supuesto un aprendizaje continuo en un campo de las tecnologías de la información totalmente en auge y del que, a nivel personal, no se tenía gran conocimiento sobre él, más allá de lo estudiado durante el grado, lo cual abarcaba principalmente lo teórico sin hacer uso de las tecnologías y sus aplicaciones reales.

Durante estos meses de desarrollo, se ha aprendido a trabajar con herramientas punteras en el campo de la Inteligencia Artificial, como lo son PyTorch y H2O, lo que me ha permitido crecer hacia un sector más profesional en este ámbito y, enriquecer mi conocimiento sobre las redes neuronales, así como espero que este simulador pueda ser de ayuda y sirva de igual manera para los estudiantes venideros del grado.

En cuanto al trabajo futuro, se podrían implementar distintas mejoras sobre el simulador, cómo, por ejemplo, permitir la creación de múltiples tipos de redes neuronales, aplicando un conjunto de datos diferente a cada una y viendo las utilidades que éstas tienen dependiendo de los datos de observaciones.

Además, el desarrollo de una interfaz sería interesante, ya que se ha intentado crear una sencilla, sin embargo, al estar trabajando con un lenguaje interpretado como lo es Python, aparecían varios problemas e incapacidades.

Por otro lado, a nivel técnico, pruebas realizadas con PyTorch, y el acotamiento y la comparación realizada con H2O, han permitido observar la importancia de los diferentes hiperparámetros y que no existe una fórmula secreta, sino que esto de la Inteligencia Artificial es más bien un juego de prueba y error.

Bibliografía

1. **IBM.** What is PyTorch. [En línea] 2024. https://www.ibm.com/topics/pytorch.
2. **H2O.ai.** Why H2O.ai. [En línea] 2024. https://h2o.ai/platform/why-h2o/.
3. **Rodrigo, Joaquín Amat.** Machine Learning con H2O y R. [En línea] Abril de 2020. https://cienciadedatos.net/documentos/44_machine_learning_con_h2o_y_r#Carga_de_datos.
4. **H2o.ai.** Usin Flow - H2O's Web UI. [En línea] 2024. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/flow.html.
5. **Wikipedia.** PyTorch. [En línea] Marzo de 2024. https://es.wikipedia.org/wiki/PyTorch.
6. **Bischel, Sophie Helene.** El método de la entropía cruzada. Algunas aplicaciones. *Trabajo fin de máster.* [En línea] 2013. https://repositorio.ual.es/bitstream/handle/10835/3322/Trabajo.pdf.
7. **Repetur, Ariel E.** Redes Neuronales Artificiales. *Trabajo Final de Licenciatura.* [En línea] 2019. https://underpost.net/ir/pdf/redes/ciencias_matematicas_redes_neuronales.pdf.
8. **Fernández, Rquel Flórez López y José Miguel Fernández.** *Las Redes Neuronales Artificiales.* 2008.

Anexo I. Resultados de las pruebas en H2O

Las siguientes representaciones, tanto tabulares como gráficas, muestran los parámetros aplicados en el entrenamiento de los modelos, los resultados obtenidos de dicho entrenamiento, y como de aproximadas son las predicciones realizadas por el modelo en comparación con el conjunto de datos original. Todo esto realizado en el entorno de H2O.

Dataset 5.000 observaciones, hasta dos cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	2000	CrossEntropy

Tabla 12. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
21,936	0,005948	0,077122	99,285 %

Tabla 13. Resultados modelo 1

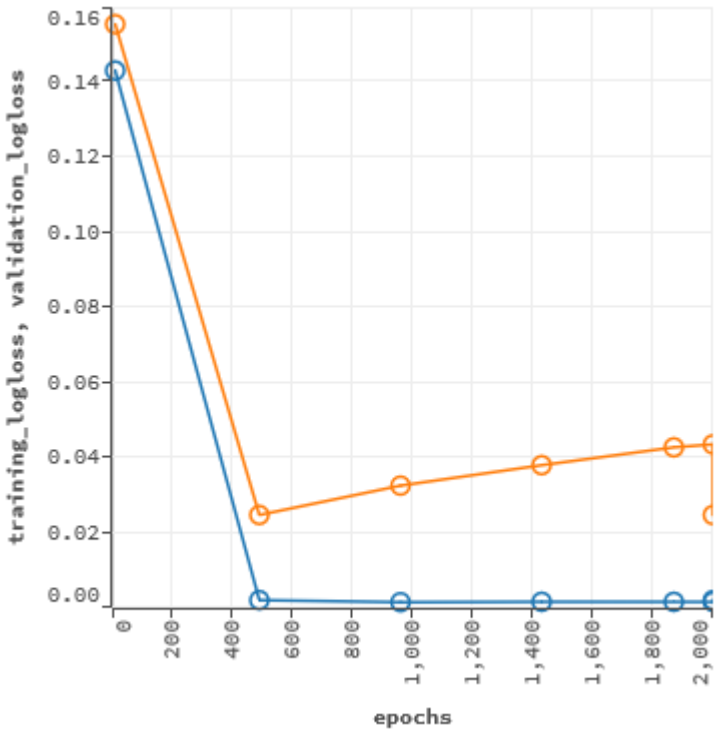


Figura 32. Gráfico modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	2000	Auto

Tabla 14. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
21,756	0,003589	0,059907	99,530 %

Tabla 15. Resultados modelo 2

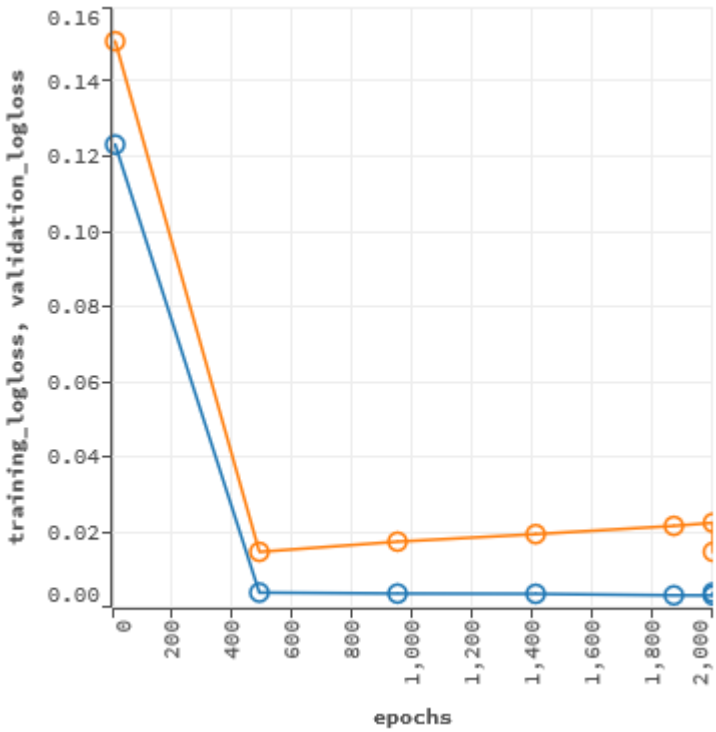


Figura 33. Gráfico modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function

70-30	No	5000	CrossEntropy

Tabla 16. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,547	0,002865	0,053526	99,615 %

Tabla 17. Resultados modelo 3

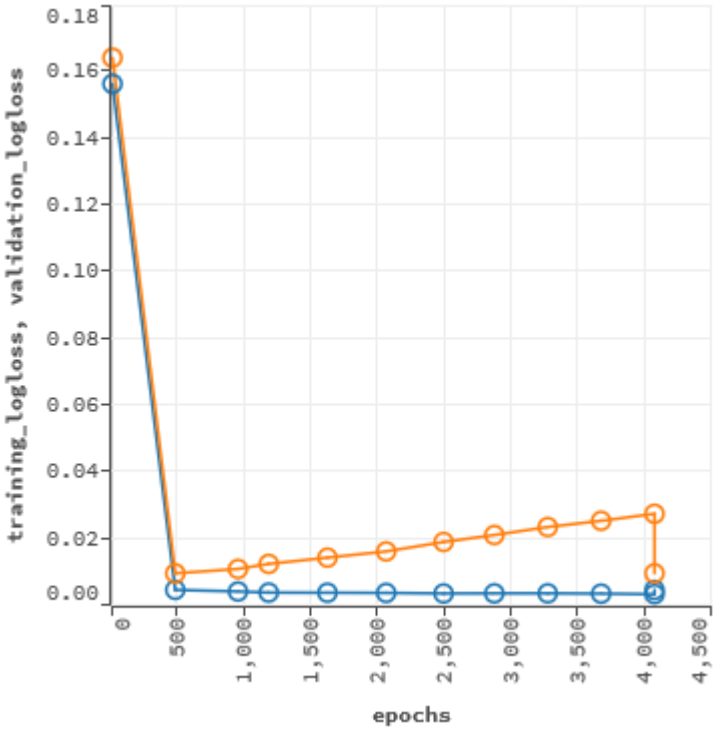


Figura 34. Gráfico modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	5000	Auto

Tabla 18. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión

50,771	0,002371	0,048689	99,729 %

Tabla 19. Resultados modelo 4

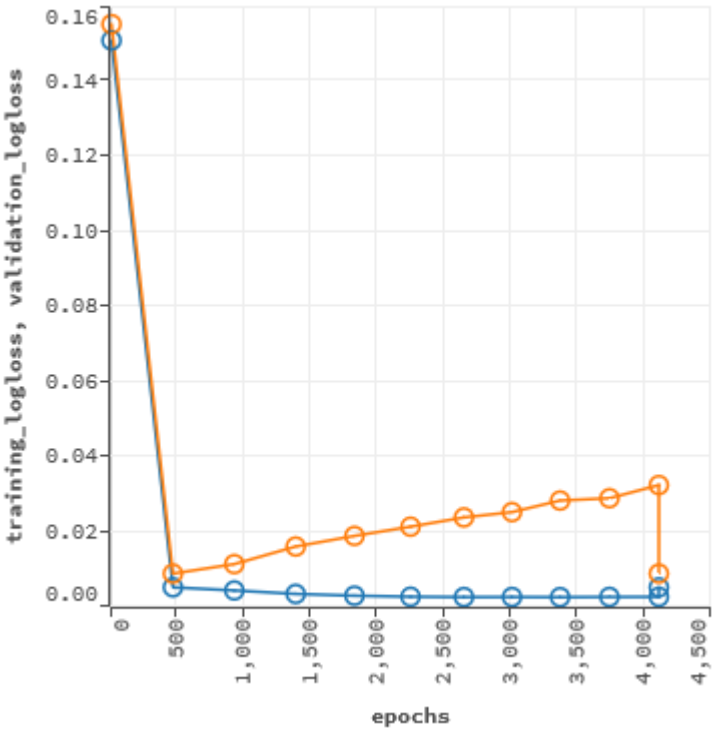


Figura 35. Gráfico modelo 4

Modelo 5

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	2000	CrossEntropy

Tabla 20. Parámetros modelo 5

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
13,666	0,006765	0,082250	99,260 %

Tabla 21. Resultados modelo 5

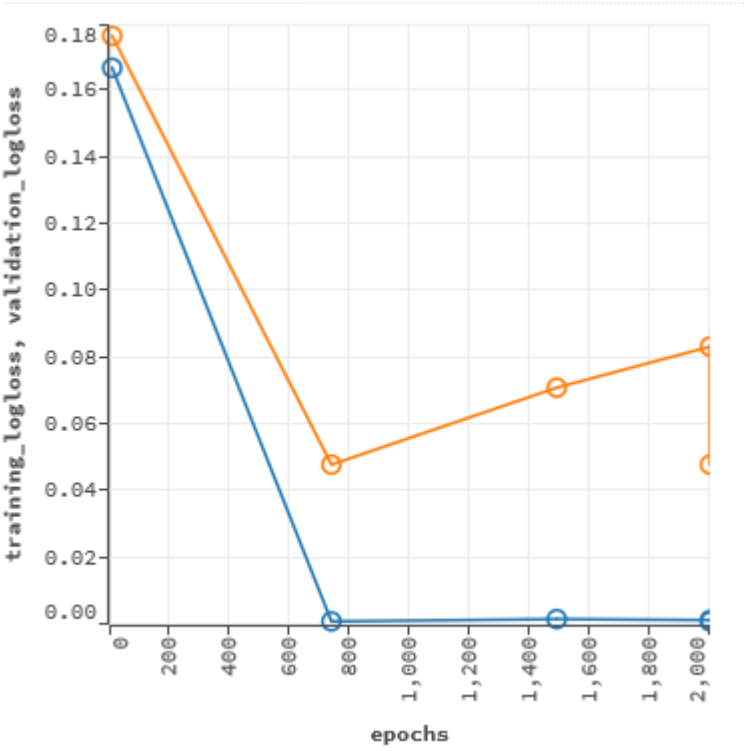


Figura 36. Gráfico modelo 5

Modelo 6

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	2000	Auto

Tabla 22. Parámetros modelo 6

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
12,938	0,007158	0,084603	99,208 %

Tabla 23. Resultados modelo 6

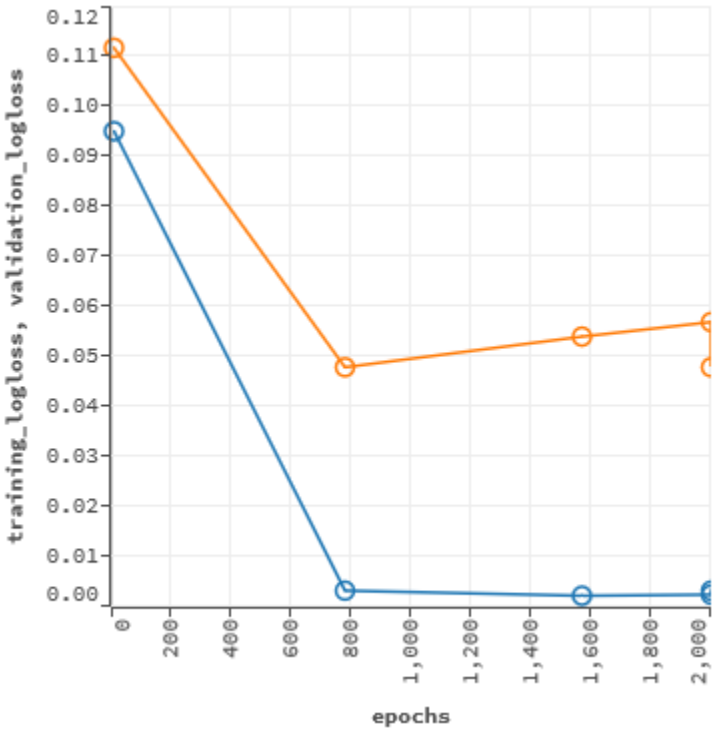


Figura 37. Gráfico modelo 6

Modelo 7

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	5000	CrossEntropy

Tabla 24. Parámetros modelo 7

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
34,370	0,005750	0,075829	99,334 %

Tabla 25. Resultados modelo 7

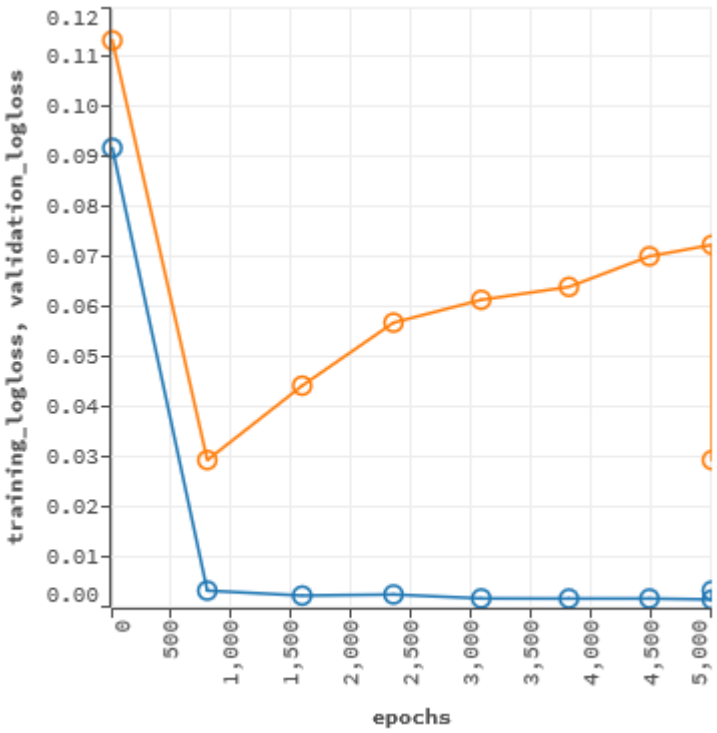


Figura 38. Gráfico modelo 7

Modelo 8

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	5000	Auto

Tabla 26. Parámetros modelo 8

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
35,498	0,007869	0,088705	99,196 %

Tabla 27. Resultados modelo 8

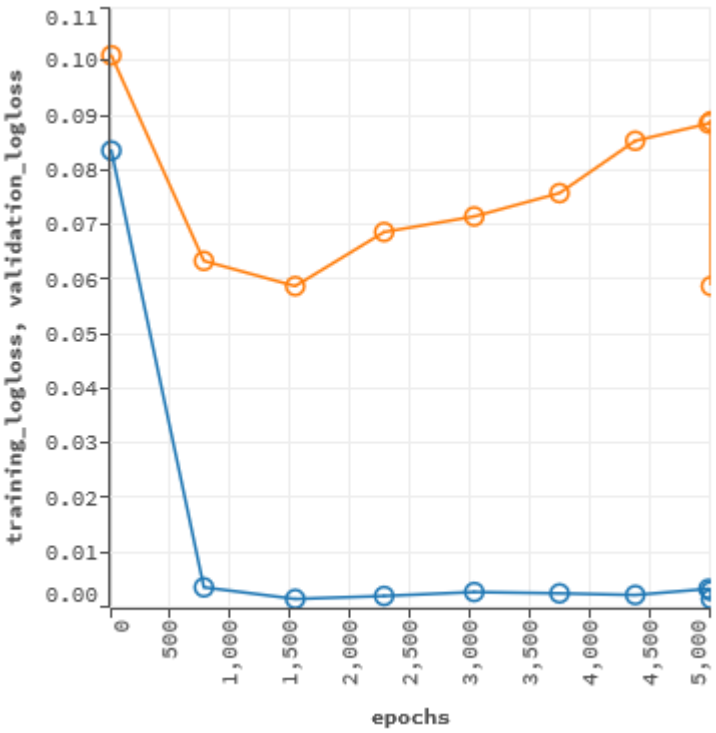


Figura 39. Gráfico modelo 8

Modelo 9

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	2000	CrossEntropy

Tabla 28. Parámetros modelo 9

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
25,354	0,005878	0,076668	99,290 %

Tabla 29. Resultados modelo 9

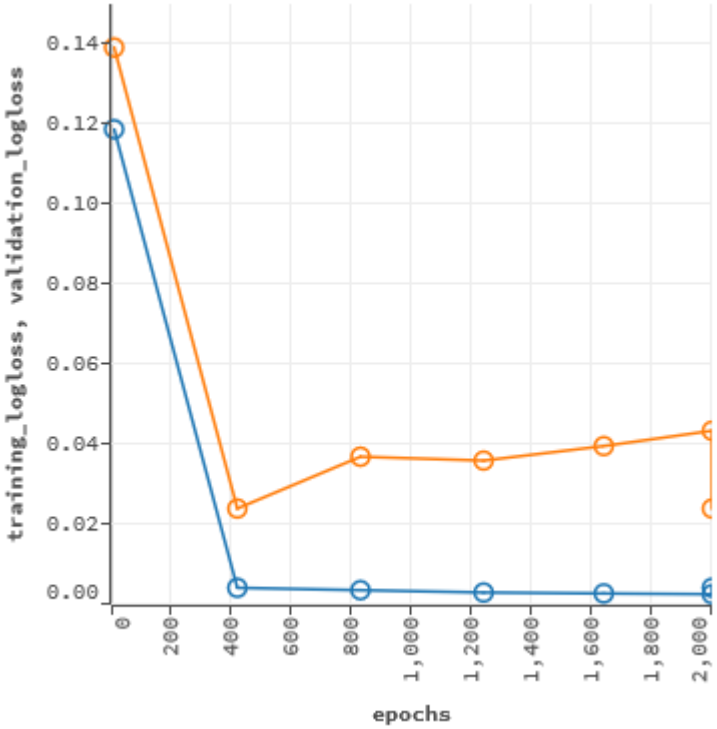


Figura 40. Gráfico modelo 9

Modelo 10

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	2000	Auto

Tabla 30. Parámetros modelo 10

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
25,118	0,004525	0,067266	99,196 %

Tabla 31. Resultados modelo 10

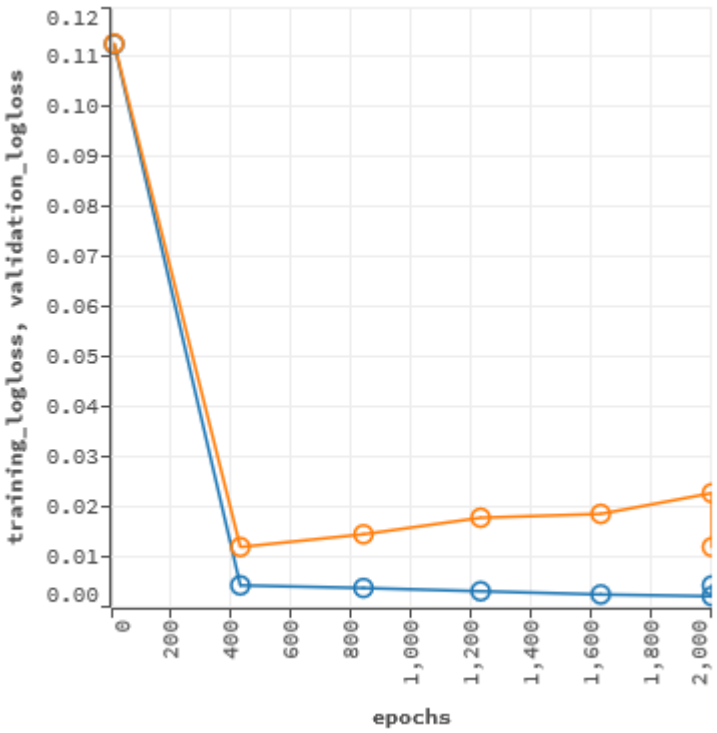


Figura 41. Gráfico modelo 10

Modelo 11

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	5000	CrossEntropy

Tabla 32. Parámetros modelo 11

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,923	0,003263	0,057122	99,611 %

Tabla 33. Resultados modelo 11

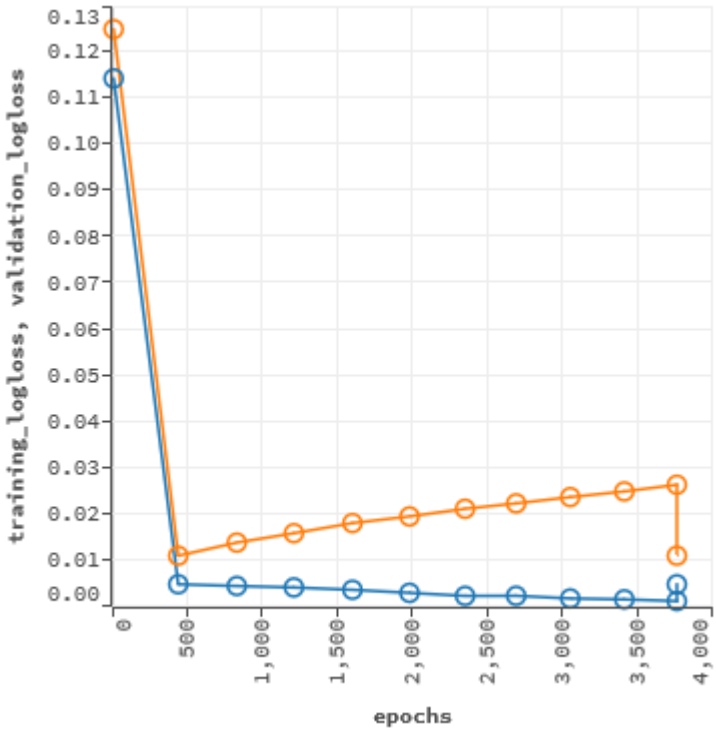


Figura 42. Gráfico modelo 11

Modelo 12

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	5000	Auto

Tabla 34. Parámetros modelo 12

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,746	0,003715	0,060950	99,413 %

Tabla 35. Resultados modelo 12

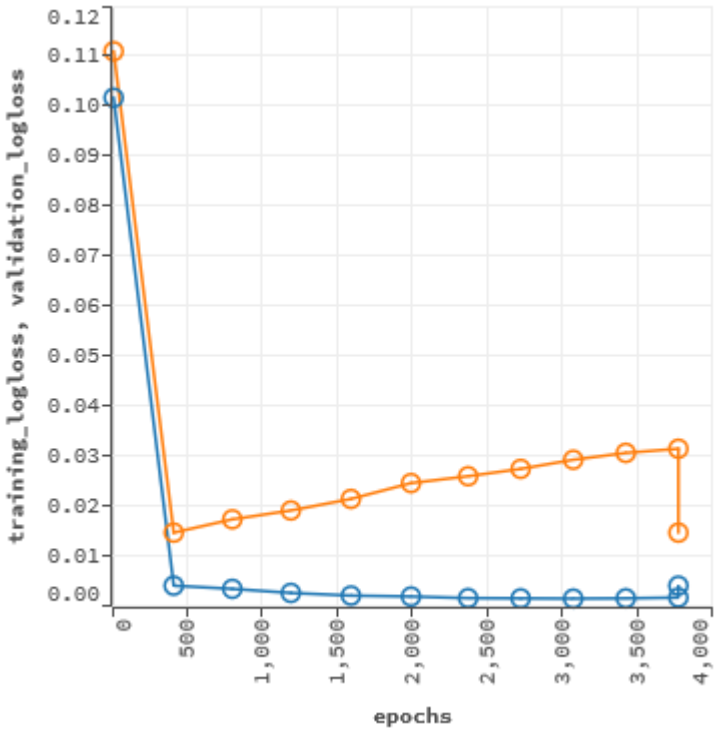


Figura 43. Gráfico modelo 12

Modelo 13

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	2000	CrossEntropy

Tabla 36. Parámetros modelo 13

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
15,122	0,006653	0,081568	99,200 %

Tabla 37. Resultados modelo 13

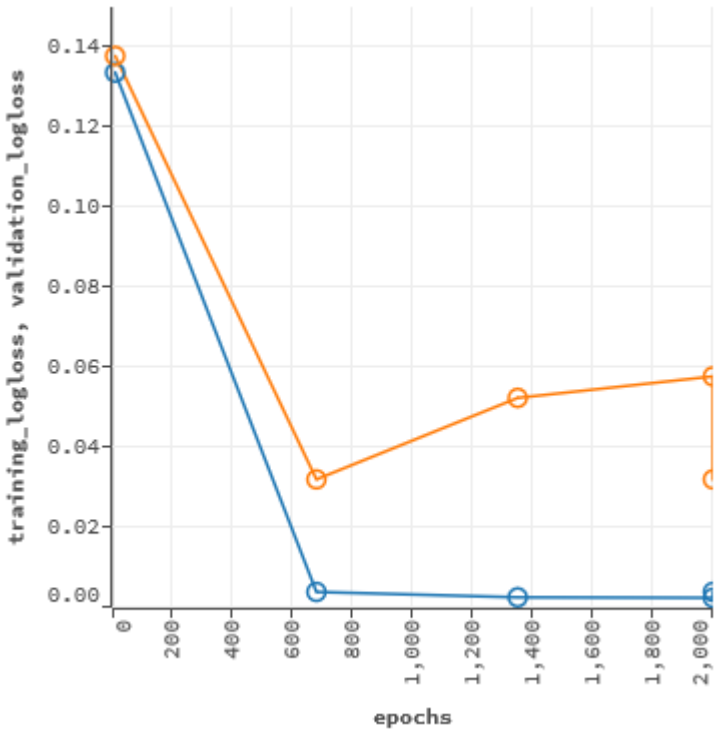


Figura 44. Gráfico modelo 13

Modelo 14

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	2000	Auto

Tabla 38. Parámetros modelo 14

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
15,225	0,004603	0,067849	99.510 %

Tabla 39. Resultados modelo 14

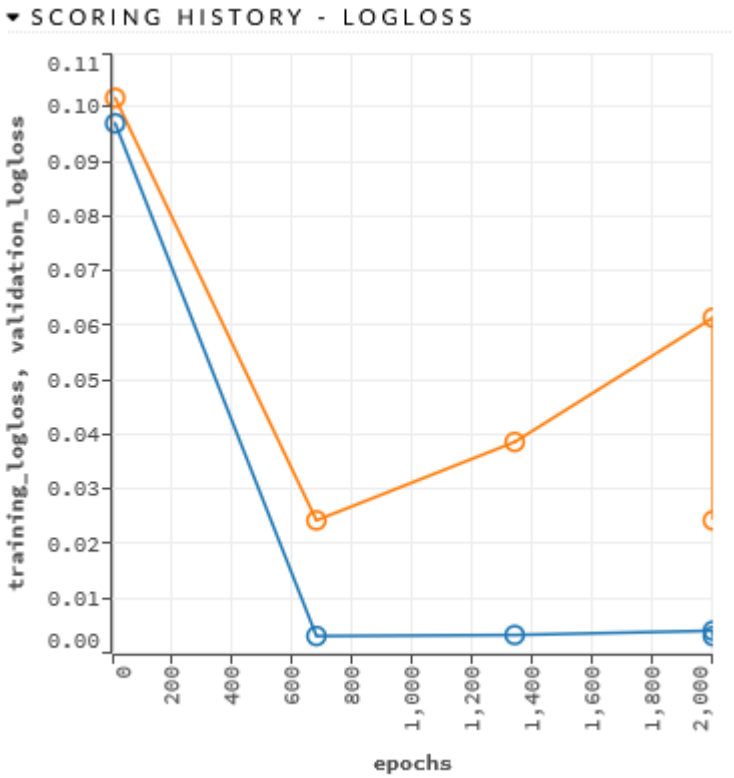


Figura 45. Gráfico modelo 14

Modelo 15

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	5000	CrossEntropy

Tabla 40. Parámetros modelo 15

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
39,168	0,010262	0,101301	98.839 %

Tabla 41. Resultados modelo 15

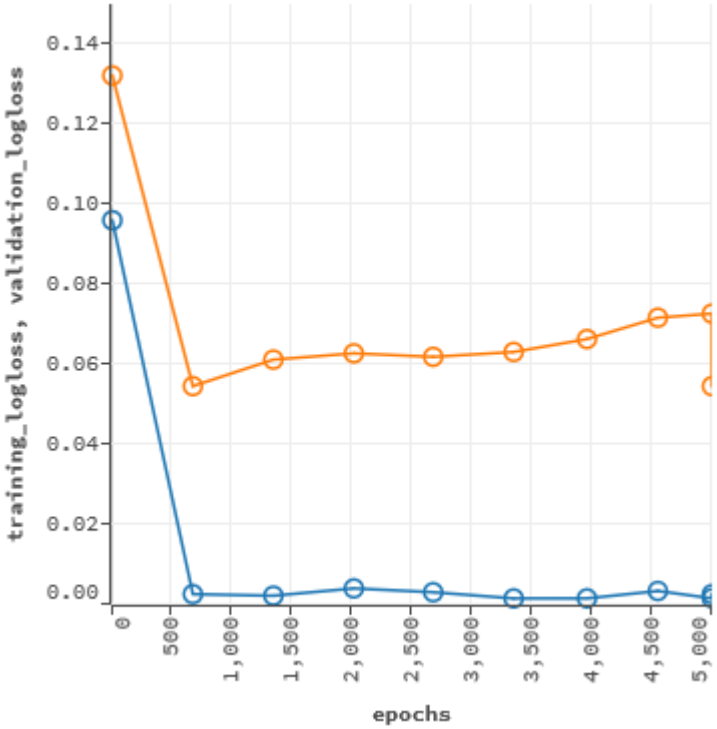


Figura 46. Gráfico modelo 15

Modelo 16

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	5000	Auto

Tabla 42. Parámetros modelo 16

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
41,584	0,013213	0,114949	98.454 %

Tabla 43. Resultados modelo 16

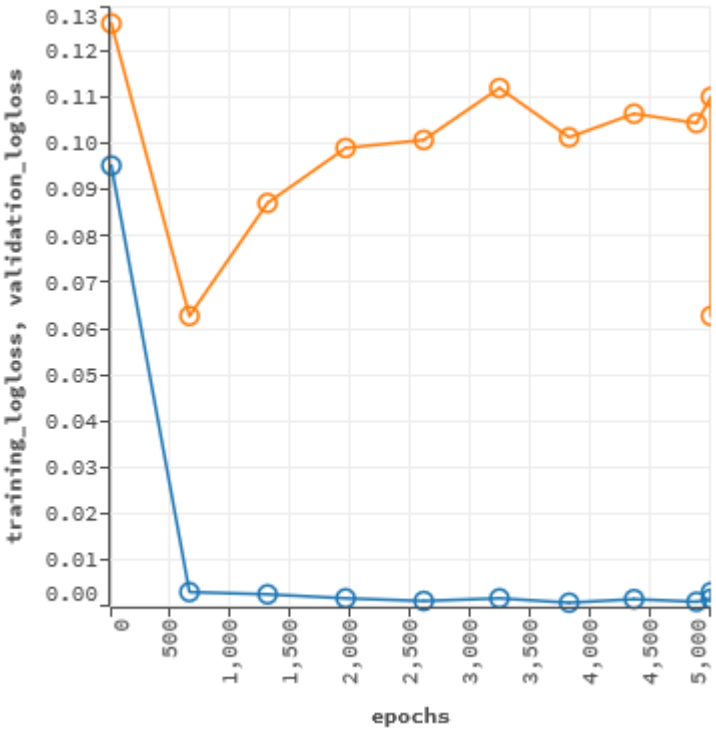


Figura 47. Gráfico modelo 16

Dataset 5.000 observaciones, hasta tres cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	2000	CrossEntropy

Tabla 44. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
22,771	0,003532	0,003532	99,465 %

Tabla 45. Resultados modelo 1

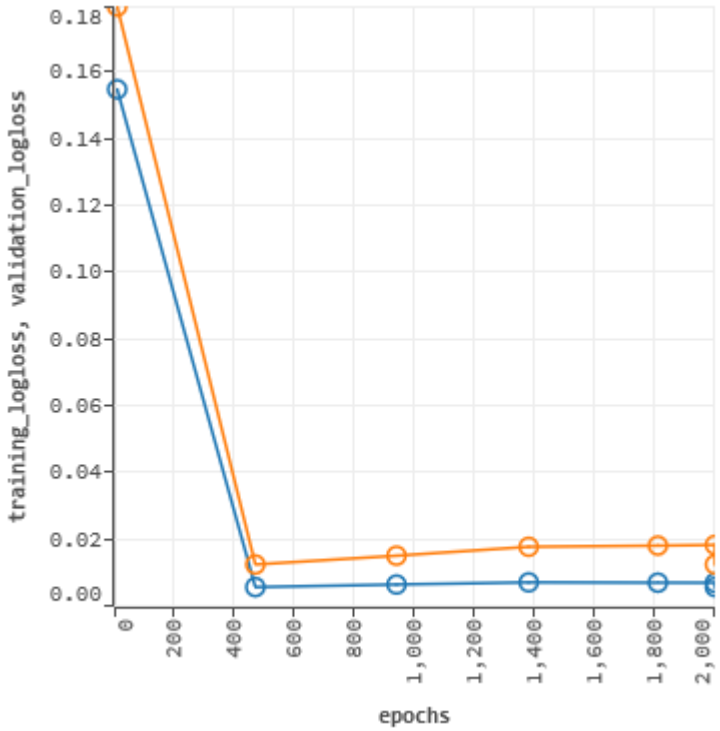


Figura 48. Gráfico modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function

70-30	No	2000	Auto

Tabla 46. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
21,633	0,003214	0,003214	99,523 %

Tabla 47. Resultados modelo 2

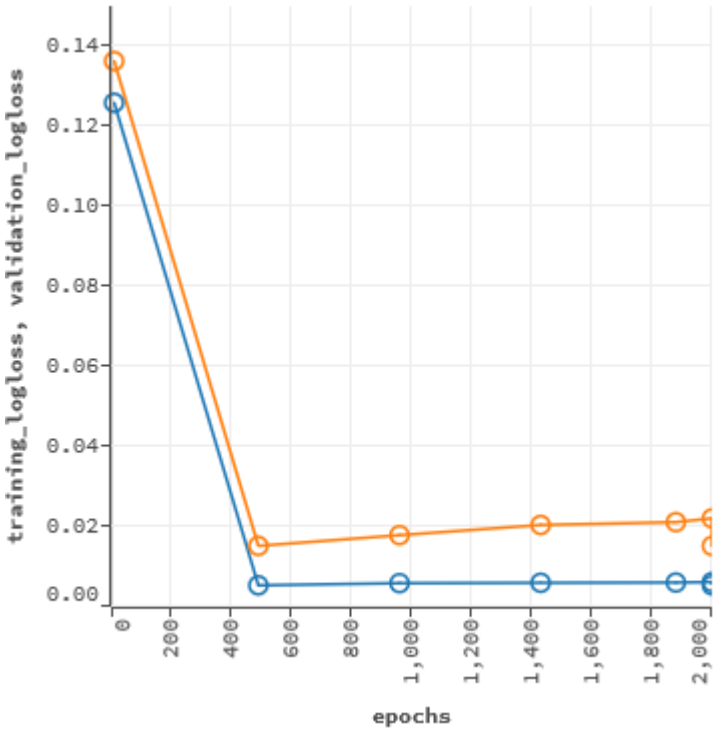


Figura 49. Gráfico modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	5000	CrossEntropy

Tabla 48. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,816	0,003874	0,062238	99,468 %

Tabla 49. Resultados modelo 3

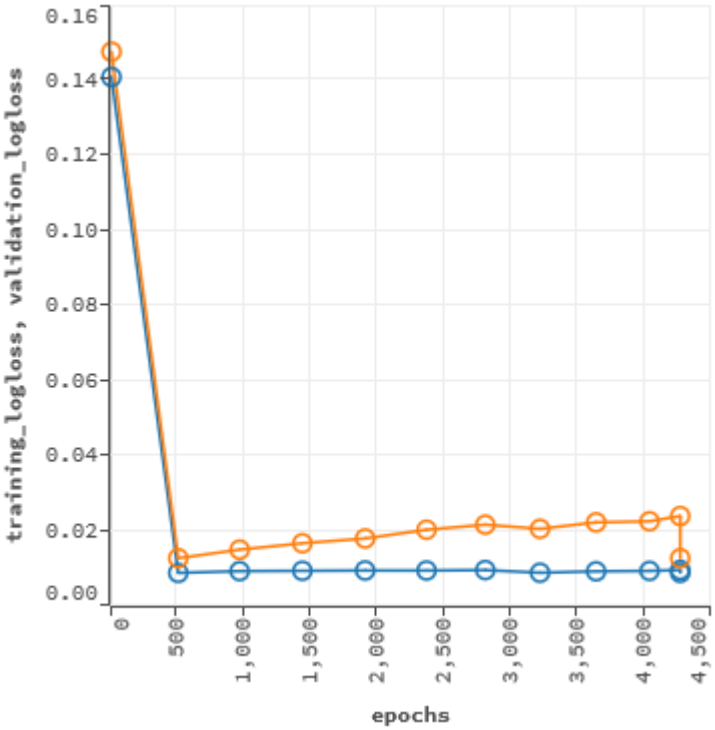


Figura 50. Gráfico modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	5000	Auto

Tabla 50. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,902	0,002798	0,052896	99,528 %

Tabla 51. Resultados modelo 4

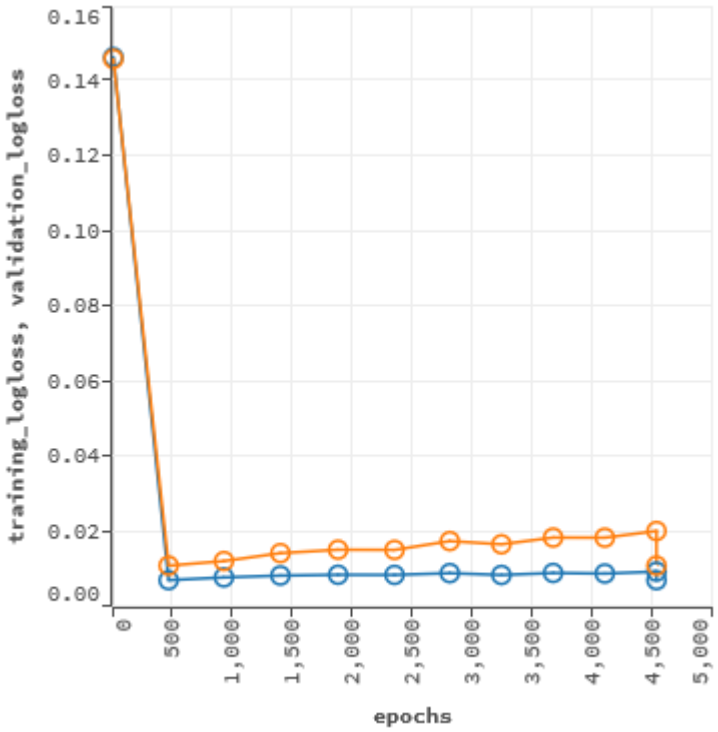


Figura 51. Gráfico modelo 4

Modelo 5

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	2000	CrossEntropy

Tabla 52. Parámetros modelo 5

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
13,805	0,012329	0,111035	98,495 %

Tabla 53. Resultados modelo 5

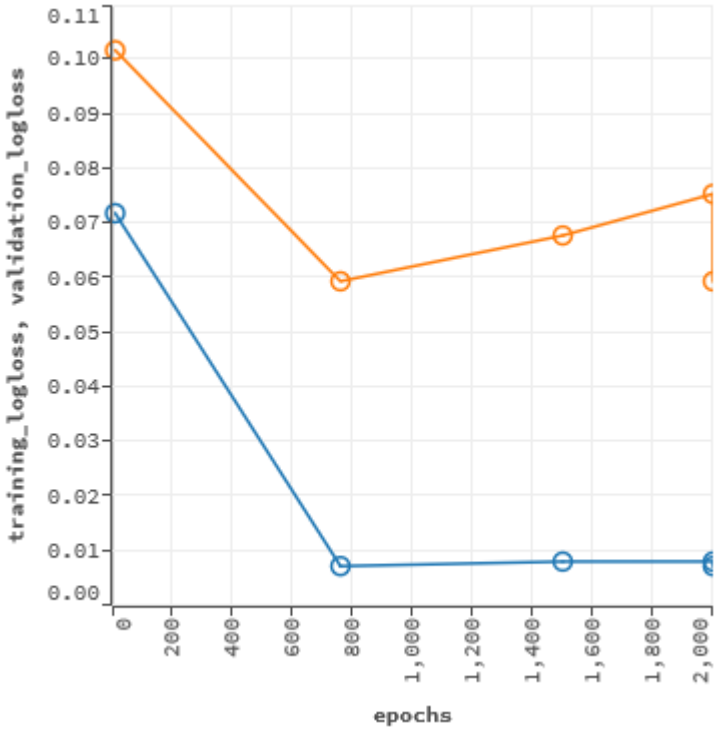


Figura 52. Gráfico modelo 5

Modelo 6

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	2000	Auto

Tabla 54. Parámetros modelo 6

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
13,543	0,008150	0,090277	99,078 %

Tabla 55. Resultado modelo 6

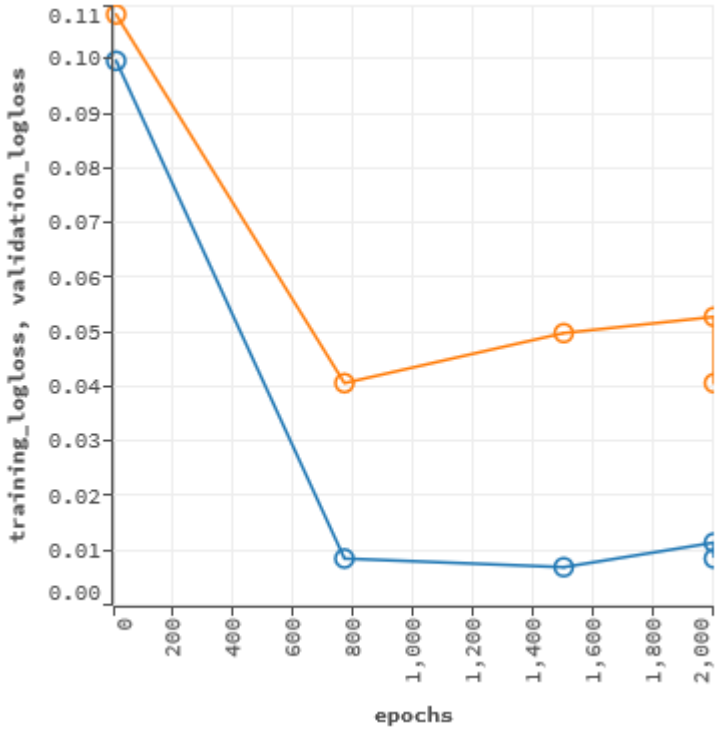


Figura 53. Gráfico modelo 6

Modelo 7

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	5000	CrossEntropy

Tabla 56. Parámetros modelo 7

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
35,633	0,005314	0,072895	99,406 %

Tabla 57. Resultados modelo 7

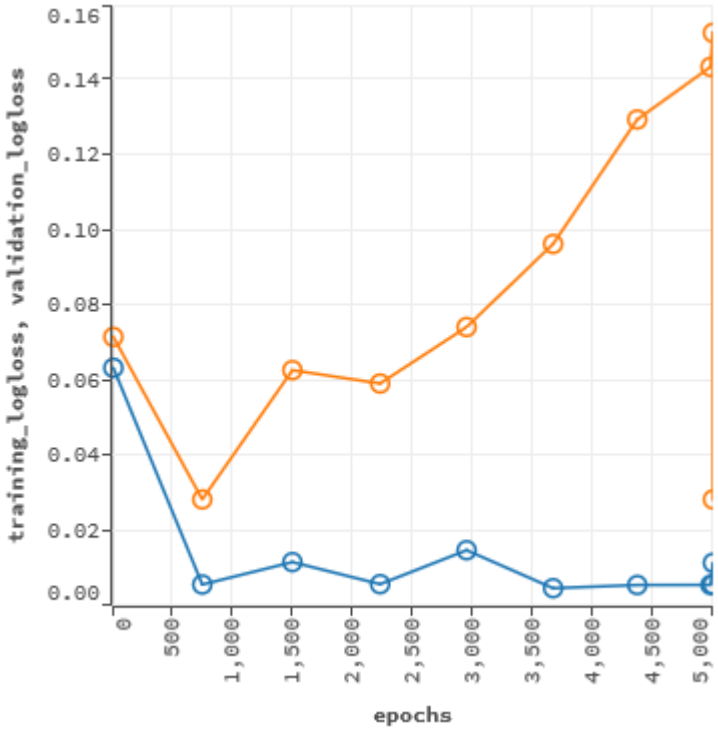


Figura 54. *Gráfico modelo 7*

Modelo 8

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	5000	Auto

Tabla 58. *Parámetros modelo 8*

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
35,891	0,005492	0,074111	99,244 %

Tabla 59. *Resultados modelo 8*

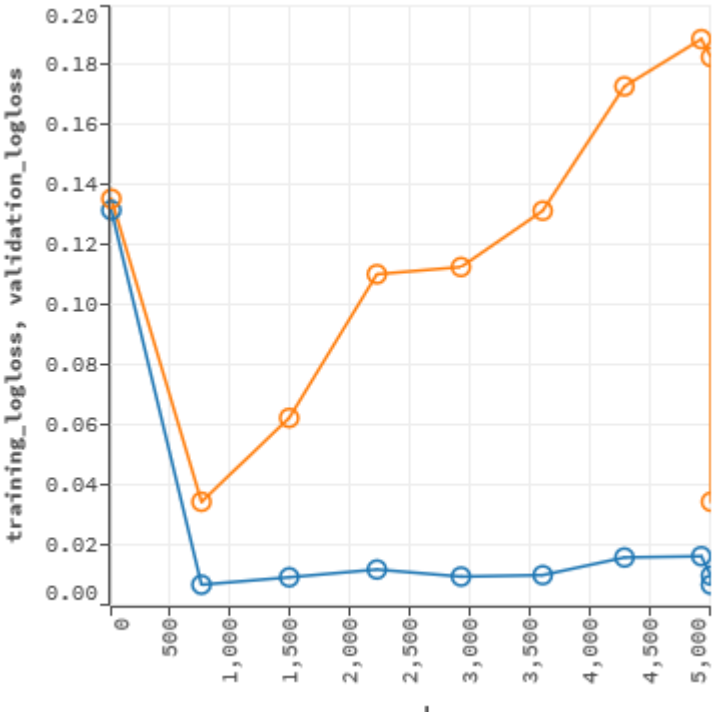


Figura 55. Gráfico modelo 8

Modelo 9

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	2000	CrossEntropy

Tabla 60. Parámetros modelo 9

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
26,727	0,002218	0,047099	99,705 %

Tabla 61. Resultados modelo 9

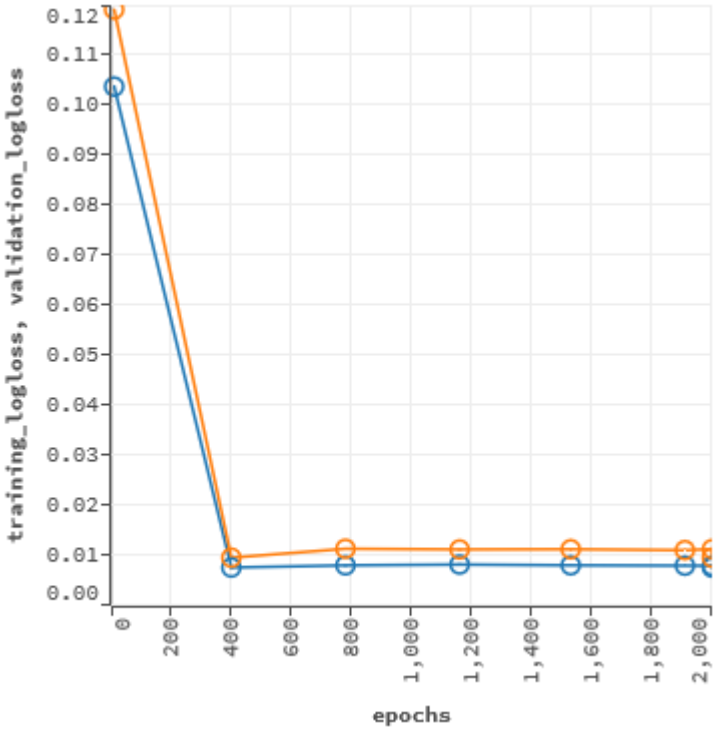


Figura 56. Gráfico modelo 9

Modelo 10

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	2000	Auto

Tabla 62. Parámetros modelo 10

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
26,929	0,004127	0,064240	99,494 %

Tabla 63. Resultados modelo 10

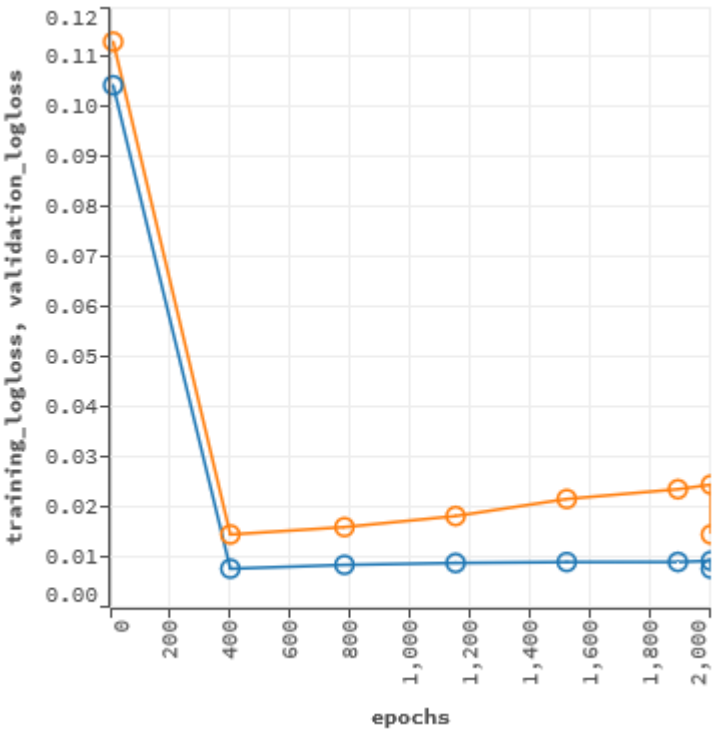


Figura 57. Gráfico modelo 10

Modelo 11

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	5000	CrossEntropy

Tabla 64. Parámetros modelo 11

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
51,112	0,002177	0,046661	99,788 %

Tabla 65. Resultados modelo 11

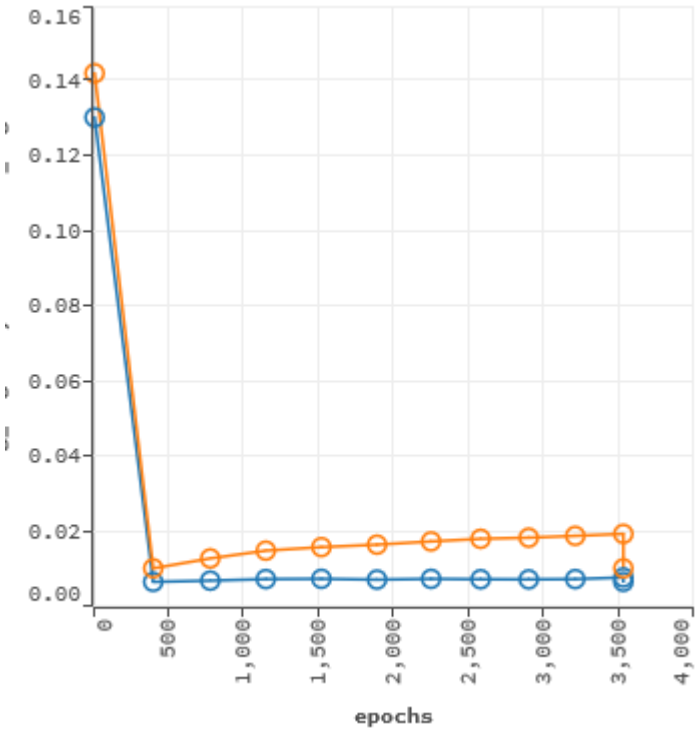


Figura 58. Gráfico modelo 11

Modelo 12

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	5000	Auto

Tabla 66. Parámetros modelo 12

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
50,734	0,003126	0,055911	99,598 %

Tabla 67. Resultados modelo 12

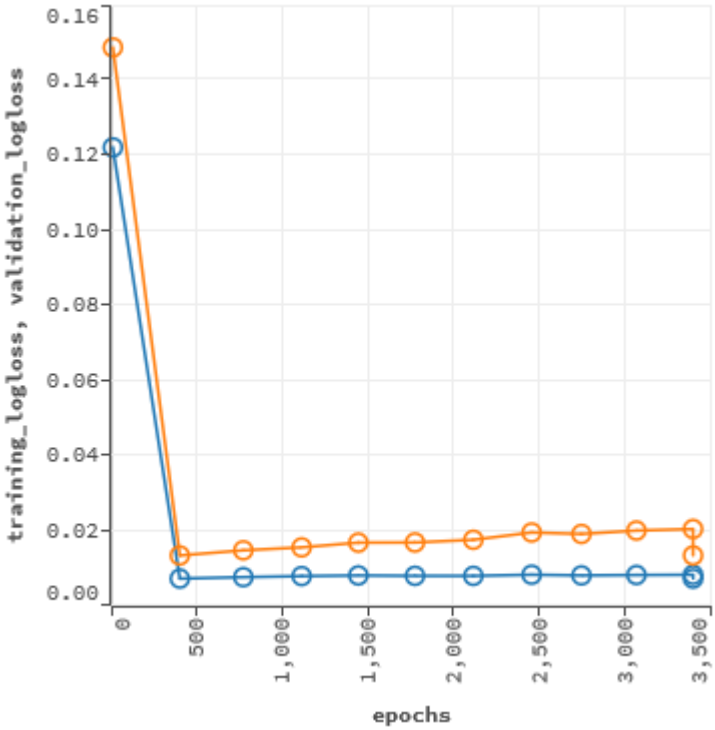


Figura 59. Gráfico modelo 12

Modelo 13

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	2000	CrossEntropy

Tabla 68. Parámetros modelo 13

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
16,593	0,009367	0,096781	98,803 %

Tabla 69. Resultados modelo 13

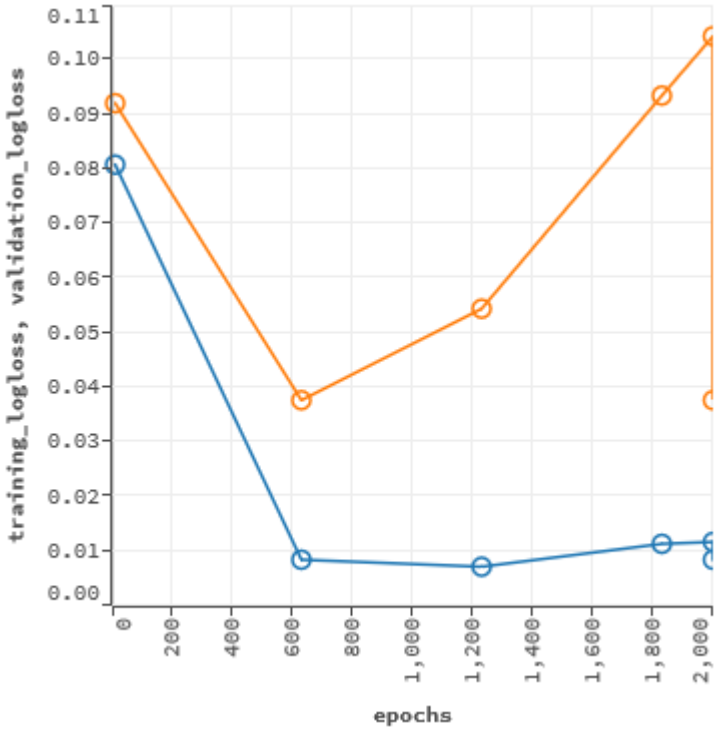


Figura 60. Gráfico modelo 13

Modelo 14

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	2000	Auto

Tabla 70. Parámetros modelo 14

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
16,418	0,007507	0,086645	99,008 %

Tabla 71. Resultados modelo 14

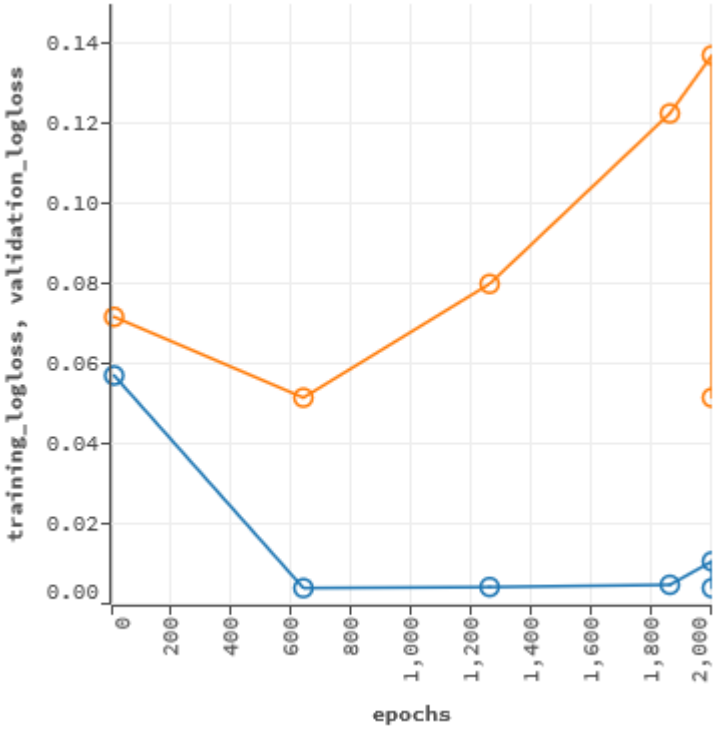


Figura 61. Gráfico modelo 14

Modelo 15

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	5000	CrossEntropy

Tabla 72. Parámetros modelo 15

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
41,583	0,005250	0,072454	99,430 %

Tabla 73. Resultados modelo 15

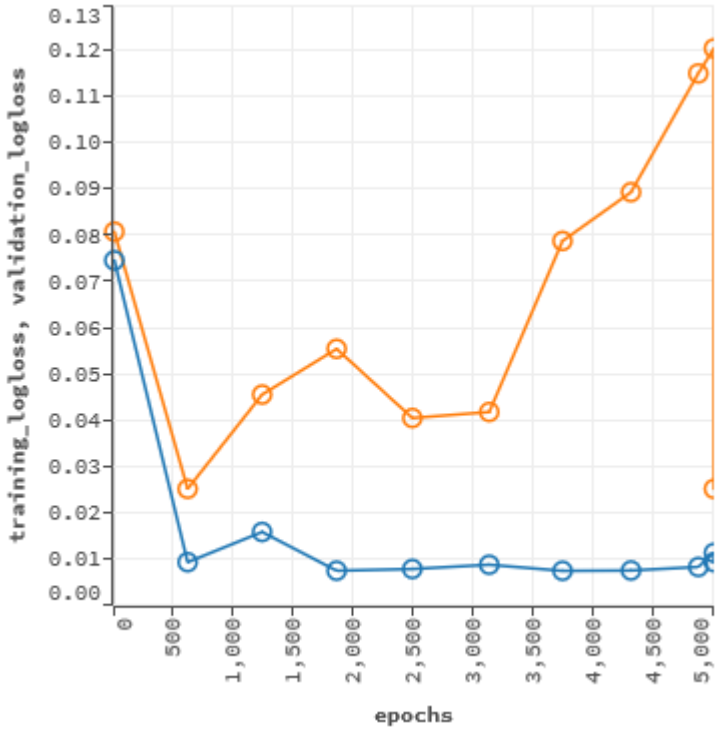


Figura 62. Gráfico modelo 15

Modelo 16

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	5000	Auto

Tabla 74. Parámetros modelo 16

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
42,459	0,004982	0,070584	99,430 %

Tabla 75. Resultados modelo 16

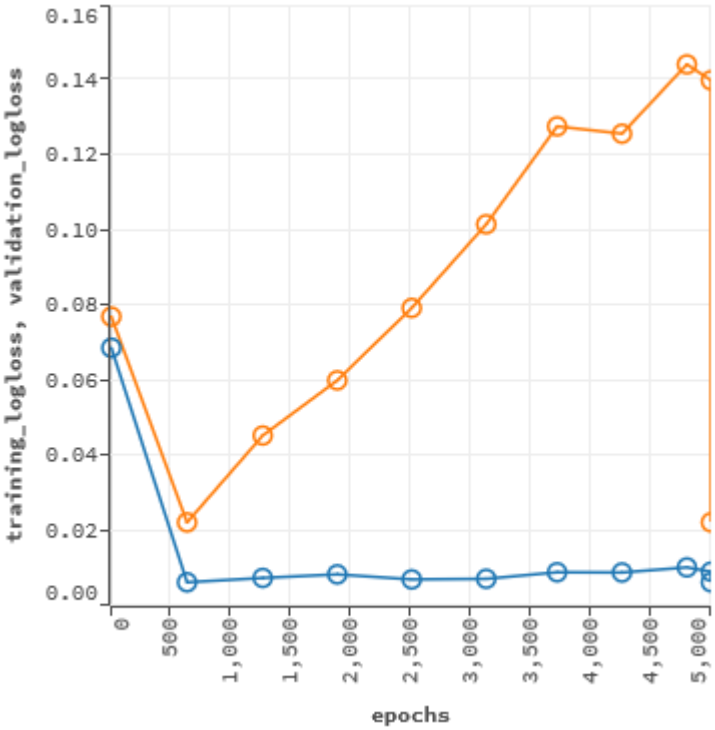


Figura 63. Gráfico modelo 16

Dataset 5.000 observaciones, hasta cinco cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	600	CrossEntropy

Tabla 76. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
6,538	0,022646	0,150486	97,375 %

Tabla 77. Resultados modelo 1

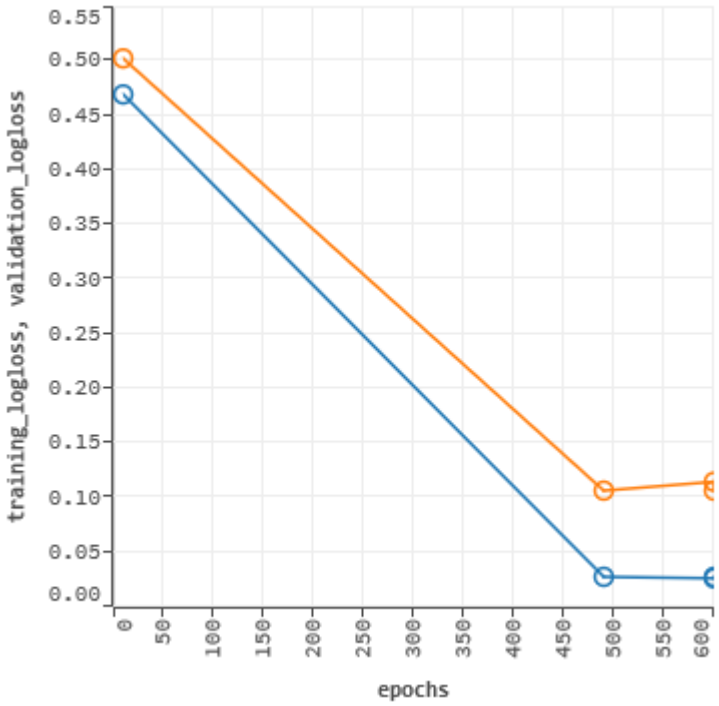


Figura 64. Gráfico modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function

70-30	No	600	Auto

Tabla 78. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
6,127	0,027384	0,165480	96,817 %

Tabla 79. Resultados modelo 2

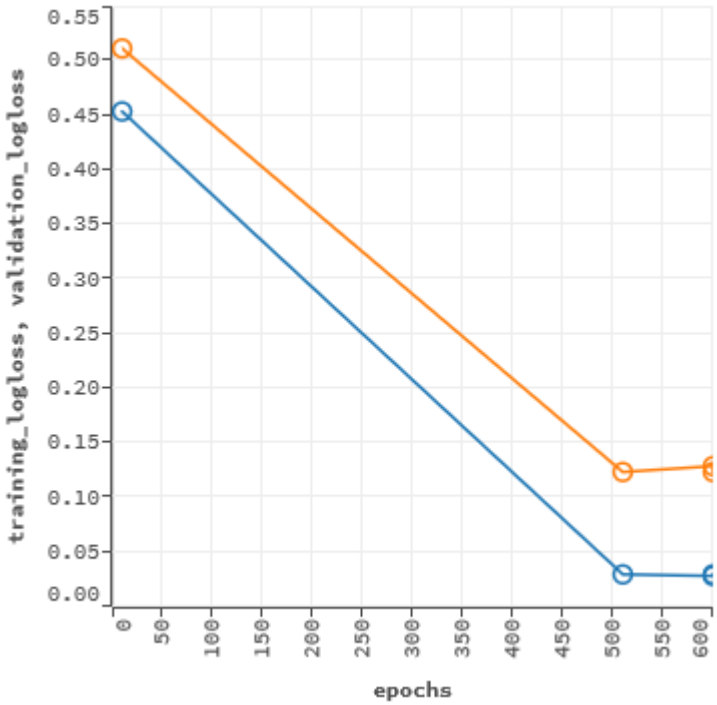


Figura 65. Gráfico modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	600	CrossEntropy

Tabla 80. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
3,894	0,052614	0,229376	96,839 %

Tabla 81. Resultados modelo 3

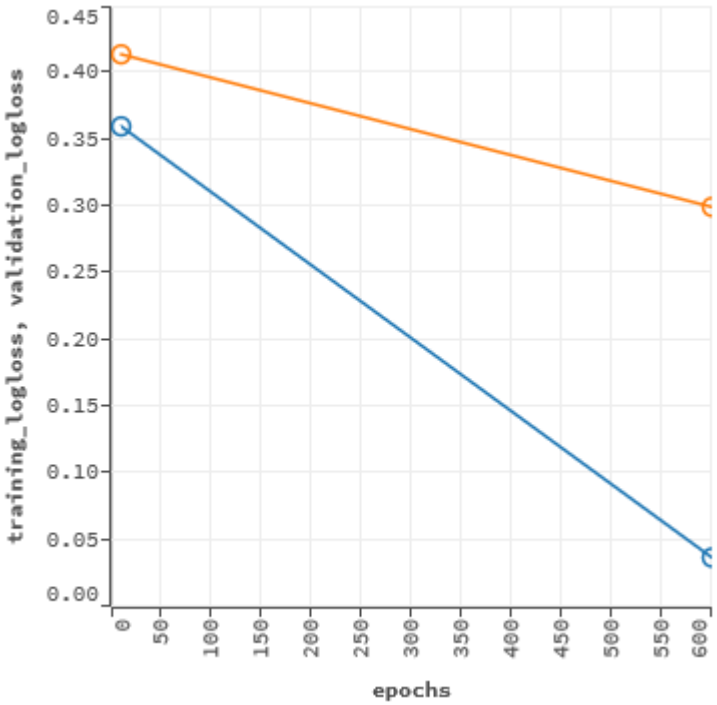


Figura 66. Gráfico modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	600	Auto

Tabla 82. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
3,764	0,061357	0,247703	92,866 %

Tabla 83. Resultados modelo 4

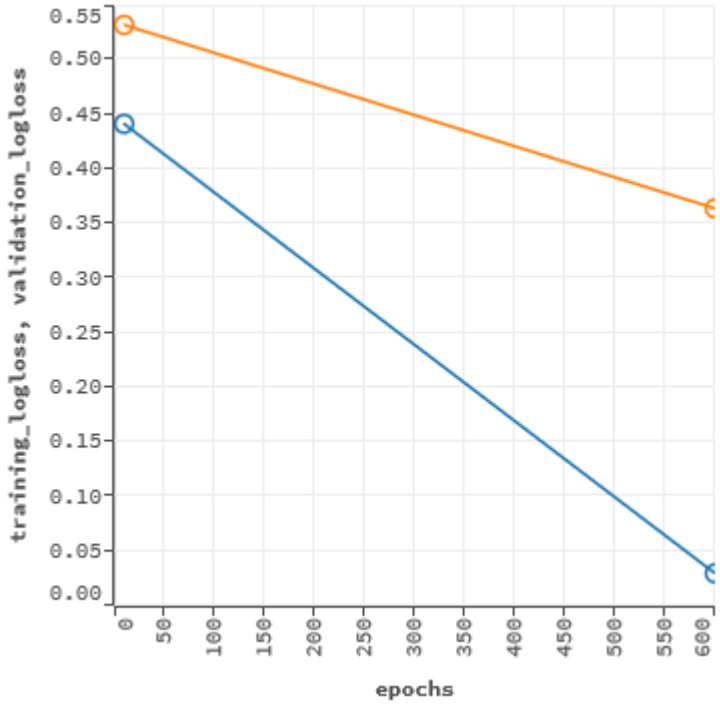


Figura 67. Gráfico modelo 4

Modelo 5

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	600	CrossEntropy

Tabla 84. Parámetros modelo 5

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
7,900	0,026184	0,161816	96,810 %

Tabla 85. Resultados modelo 5

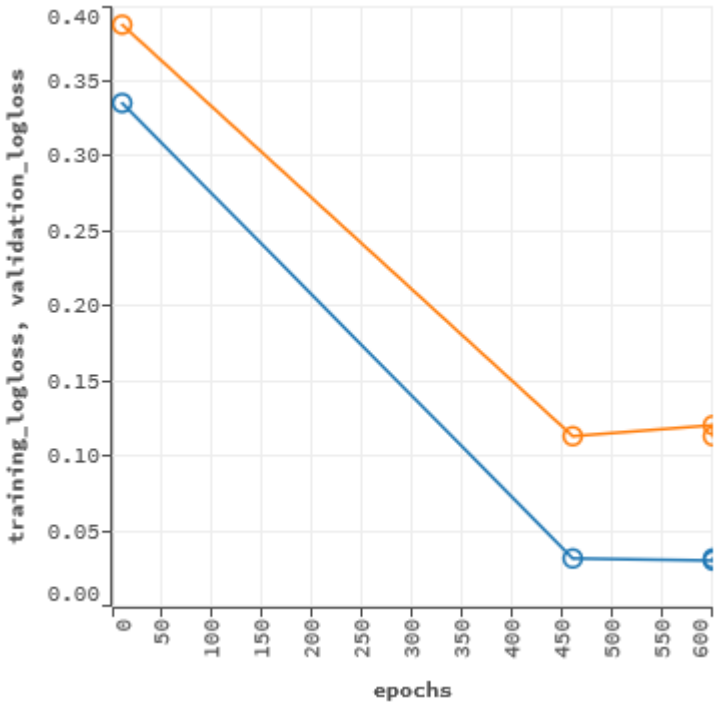


Figura 68. Gráfico modelo 5

Modelo 6

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	600	Auto

Tabla 86. Parámetros modelo 6

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
6,908	0,025179	0,158680	97,128 %

Tabla 87. Resultados modelo 6

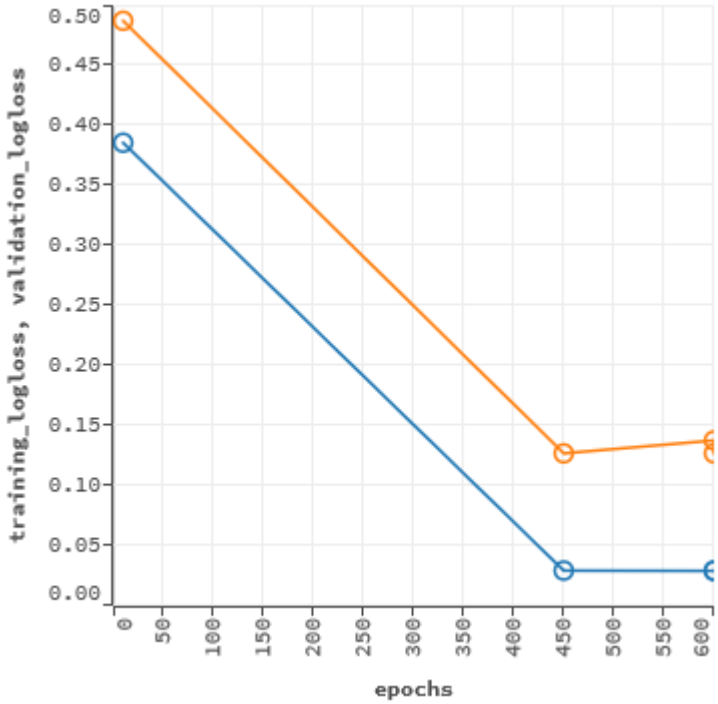


Figura 69. Gráfico modelo 6

Modelo 7

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	600	CrossEntropy

Tabla 88. Parámetros modelo 7

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
4,549	0,055693	0,235994	93,394 %

Tabla 89. Resultados modelo 7

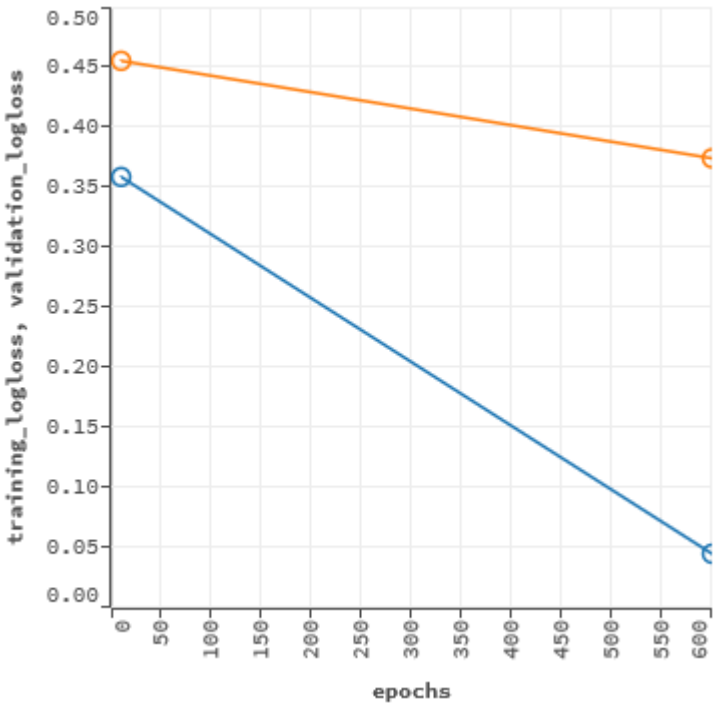


Figura 70. Gráfico modelo 7

Modelo 8

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	600	Auto

Tabla 90. Parámetros modelo 8

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
4,811	0,063923	0,252830	92,799 %

Tabla 91. Resultados modelo 8

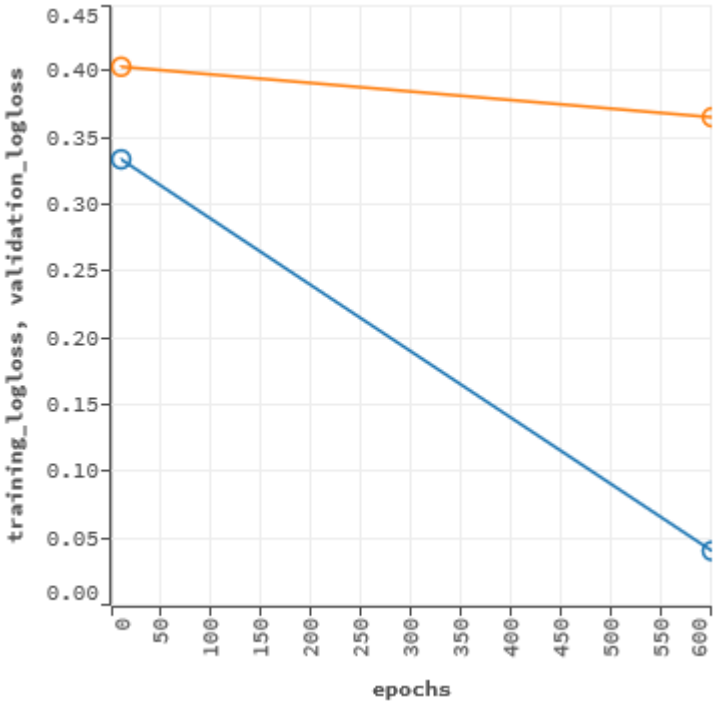


Figura 71. Gráfico modelo 8

Dataset 10.000 observaciones, hasta dos cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	600	CrossEntropy

Tabla 92. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
13,547	0,003400	0,058310	99,535 %

Tabla 93. Resultados modelo 1

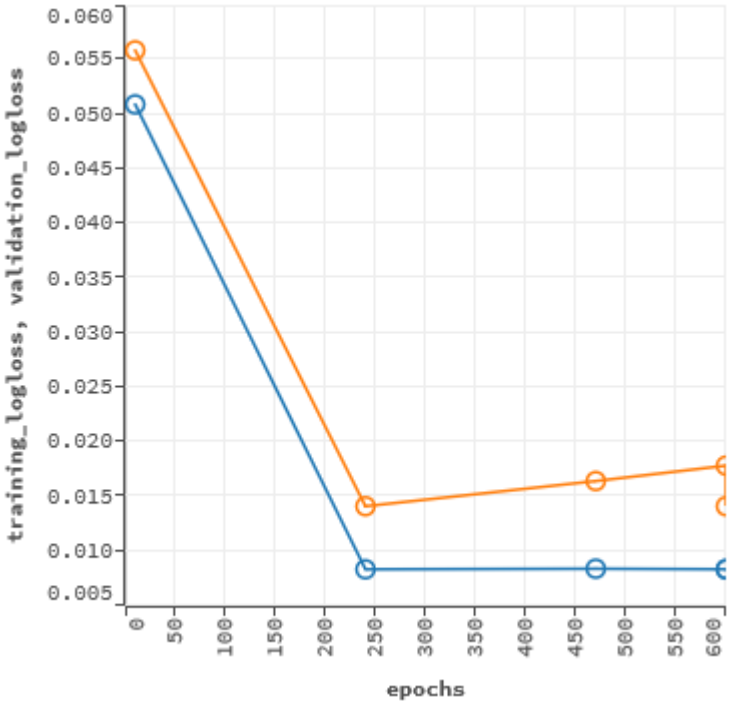


Figura 72. Gráfico modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	600	Auto

Tabla 94. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
13,244	0,002328	0,048253	99,633 %

Tabla 95. Resultados modelo 2

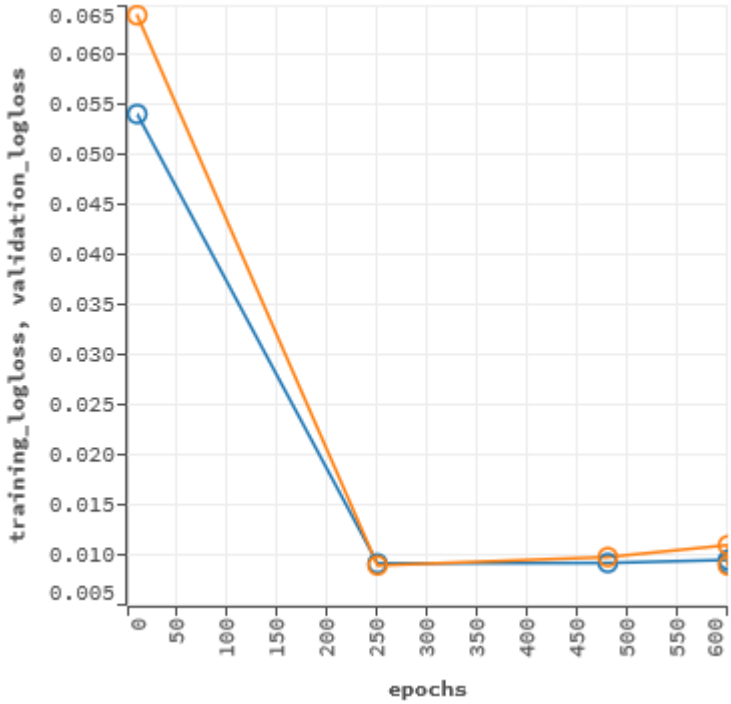


Figura 73. Gráfico modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	600	CrossEntropy

Tabla 96. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
8,499	0,004608	0,067885	99,432 %

Tabla 97. Resultados modelo 3

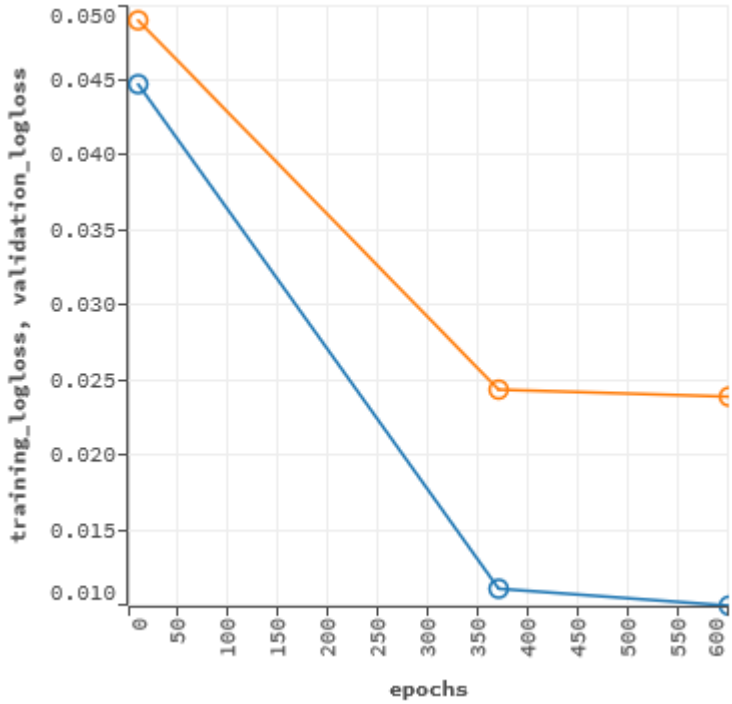


Figura 74. Gráfico modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	600	Auto

Tabla 98. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
8,768	0,004789	0,069203	99,516 %

Tabla 99. Resultados modelo 4

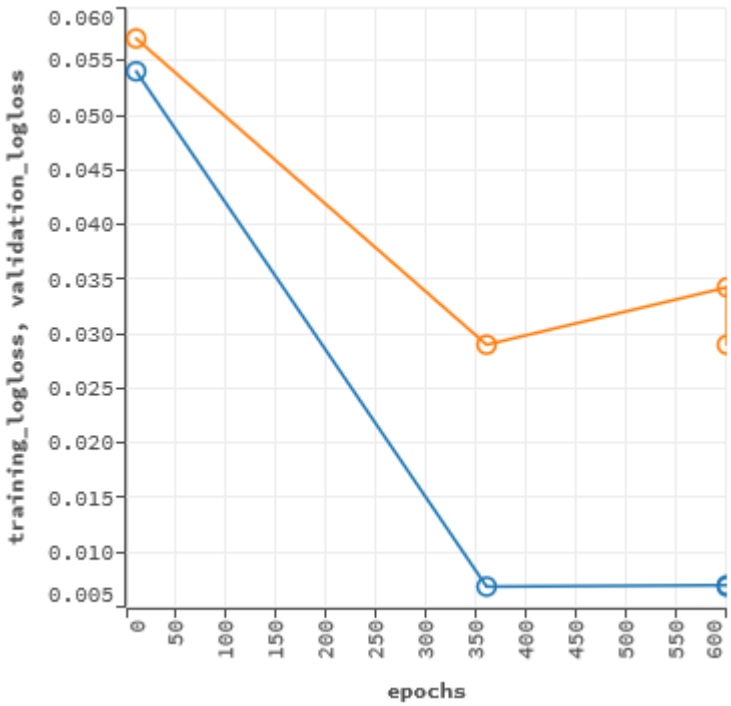


Figura 75. Gráfico modelo 4

Modelo 5

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	600	CrossEntropy

Tabla 100. Parámetros modelo 5

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
16,730	0,002150	0,046369	99,801 %

Tabla 101. Resultados modelo 5

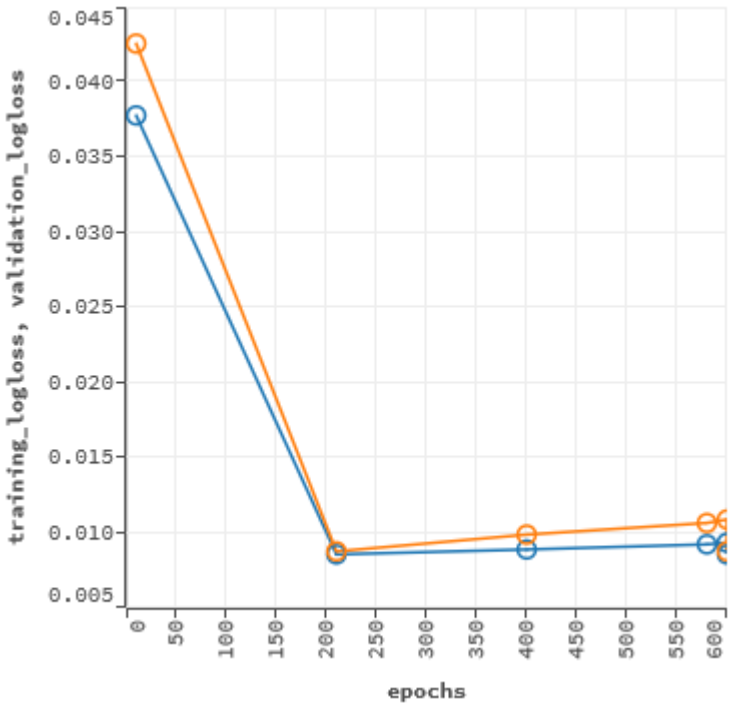


Figura 76. Gráfico modelo 5

Modelo 6

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	600	Auto

Tabla 102. Parámetros modelo 6

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
18,479	0,003506	0,059208	99,491 %

Tabla 103. Resultados modelo 6

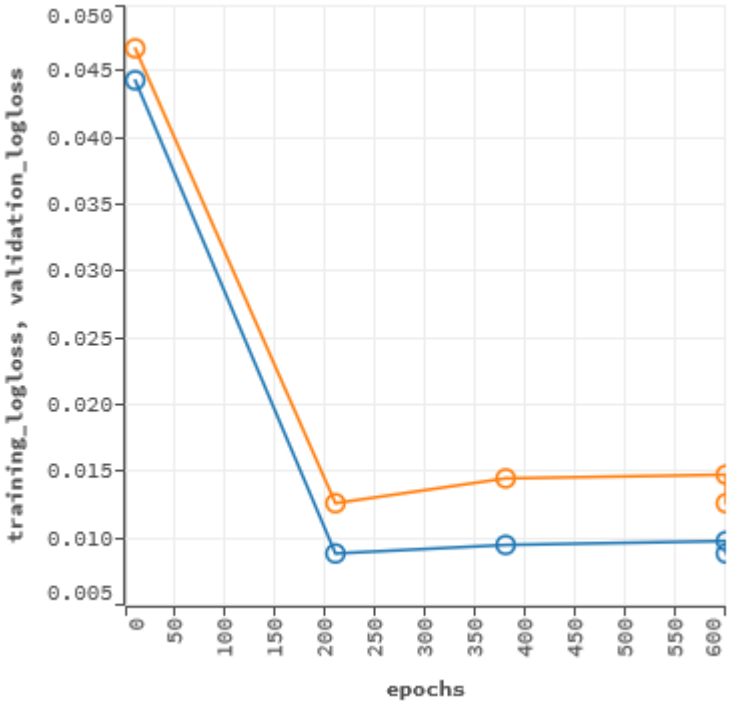


Figura 77. Gráfico modelo 6

Modelo 7

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	600	CrossEntropy

Tabla 104. Parámetros modelo 7

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
9,451	0,005071	0,071210	99,344 %

Tabla 105. Resultados modelo 7

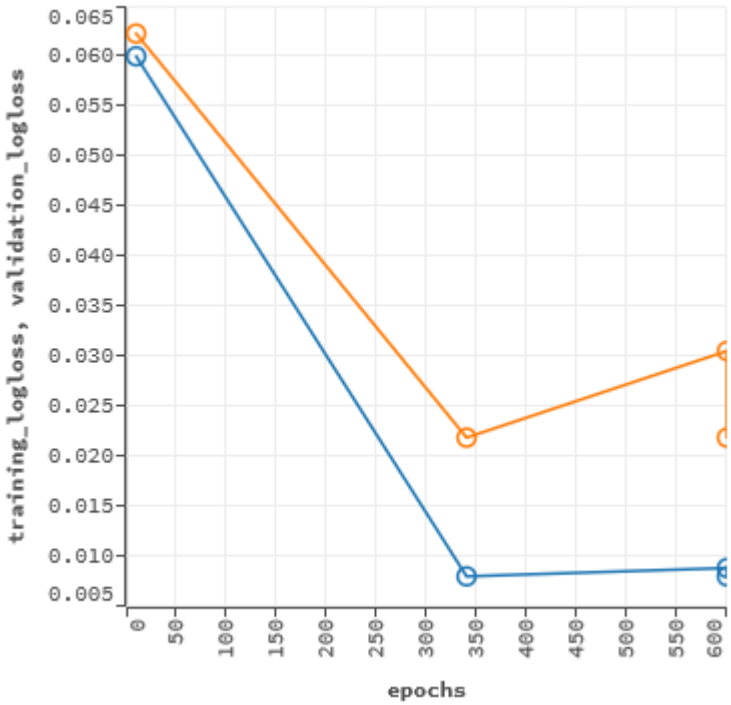


Figura 78. Gráfico modelo 7

Modelo 8

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	600	Auto

Tabla 106. Parámetros modelo 8

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
9,610	0,006048	0,077770	99,139 %

Tabla 107. Resultados modelo 8

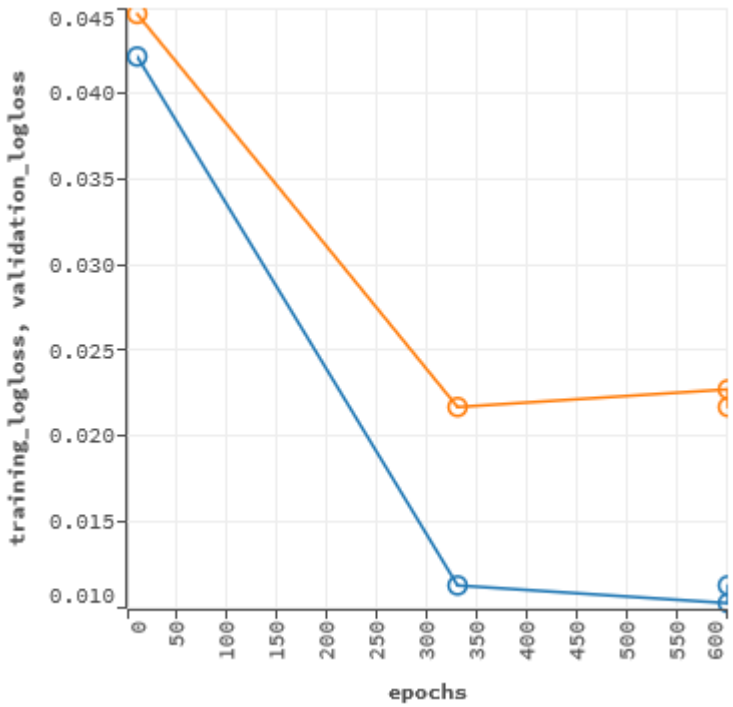


Figura 79. Gráfico modelo 8

Dataset 10.000 observaciones, hasta tres cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	600	CrossEntropy

Tabla 108. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
13,761	0,002712	0,052078	99,535 %

Tabla 109. Resultados modelo 1

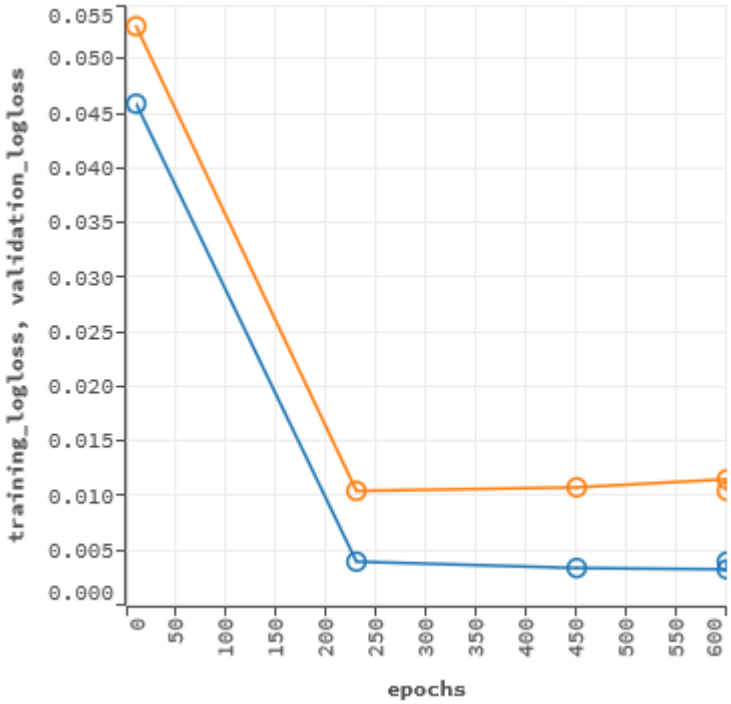


Figura 80. Gráfico modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function

70-30	10	600	CrossEntropy

Tabla 110. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
8,575	0,003266	0,057151	99,600 %

Tabla 111. Resultados modelo 2

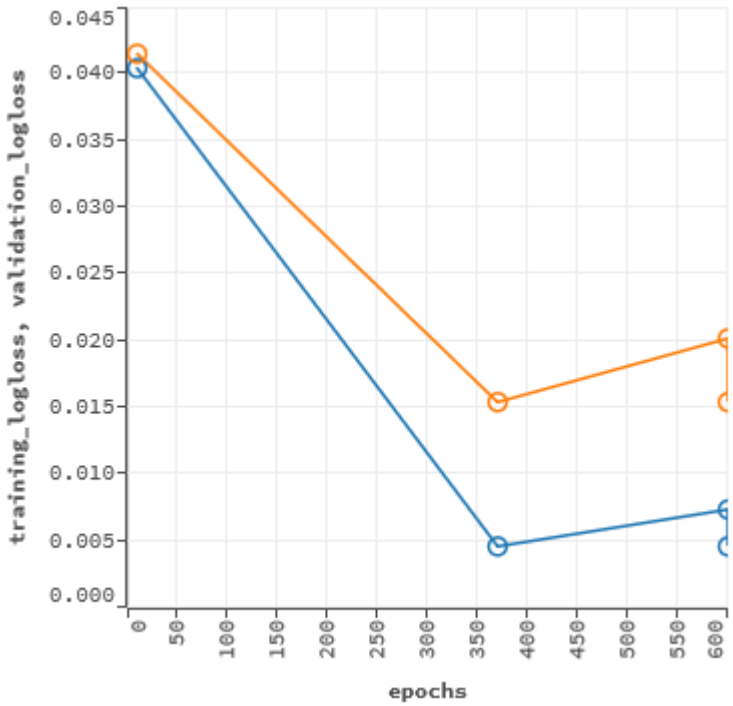


Figura 81. Gráfico modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	600	CrossEntropy

Tabla 112. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
15,587	0,001837	0,042856	99,705 %

Tabla 113. Resultados modelo 3

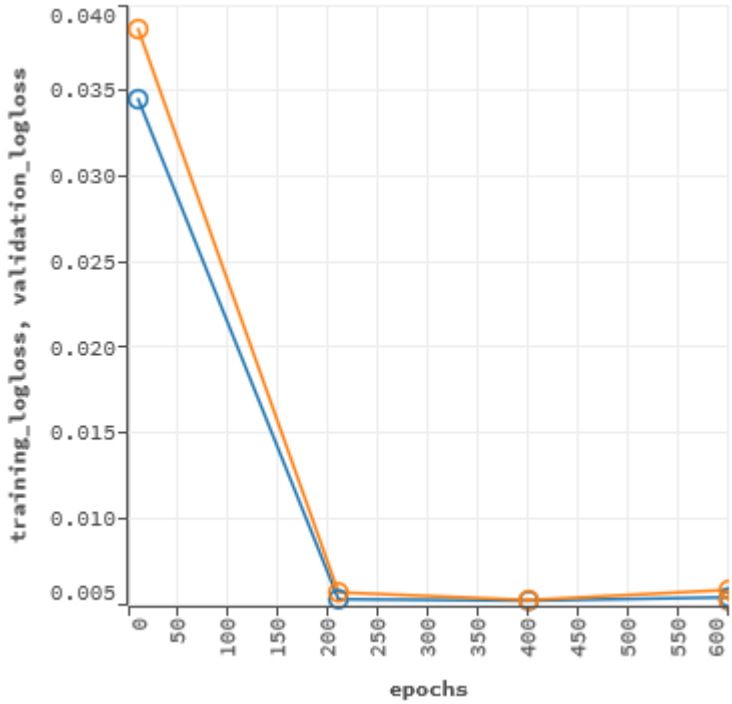


Figura 82. Gráfico modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	600	CrossEntropy

Tabla 114. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
9,915	0,005127	0,071605	99,345 %

Tabla 115. Resultados modelo 4

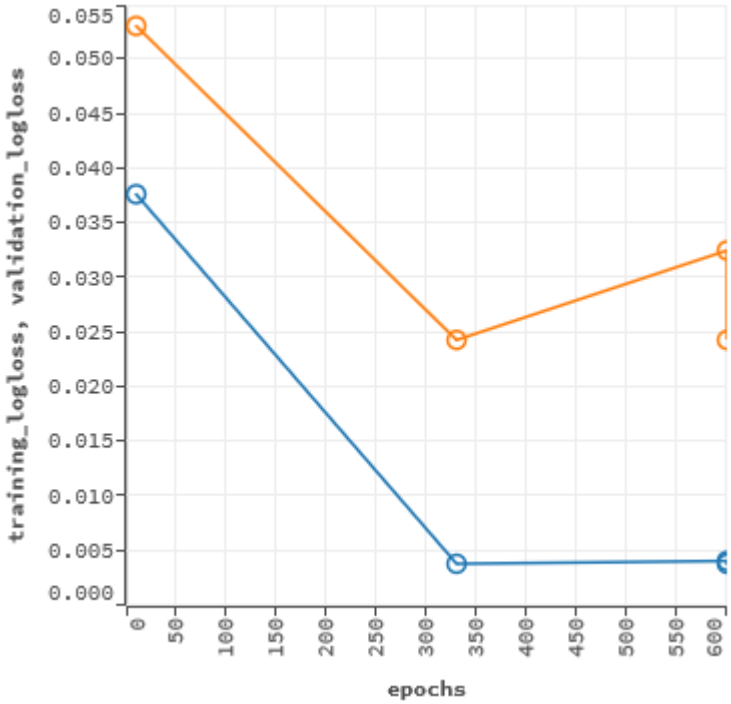


Figura 83. Gráfico modelo 4

Dataset 10.000 observaciones, hasta cinco cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	No	600	CrossEntropy

Tabla 116. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
14,398	0,020460	0,143039	97,536 %

Tabla 117. Resultados modelo 1

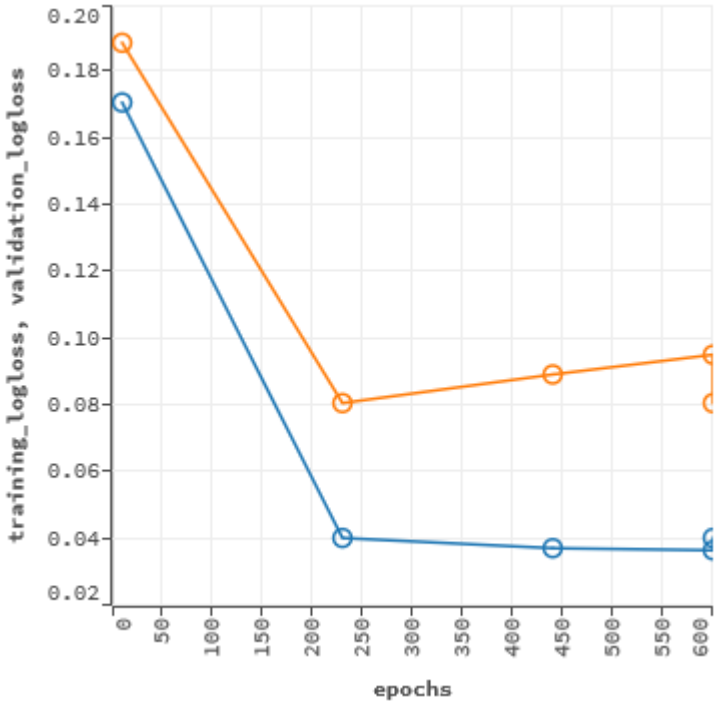


Figura 84. Gráfico modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
70-30	10	600	CrossEntropy

Tabla 118. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
11,887	0,038119	0,195242	95,591 %

Tabla 119. Resultados modelo 2

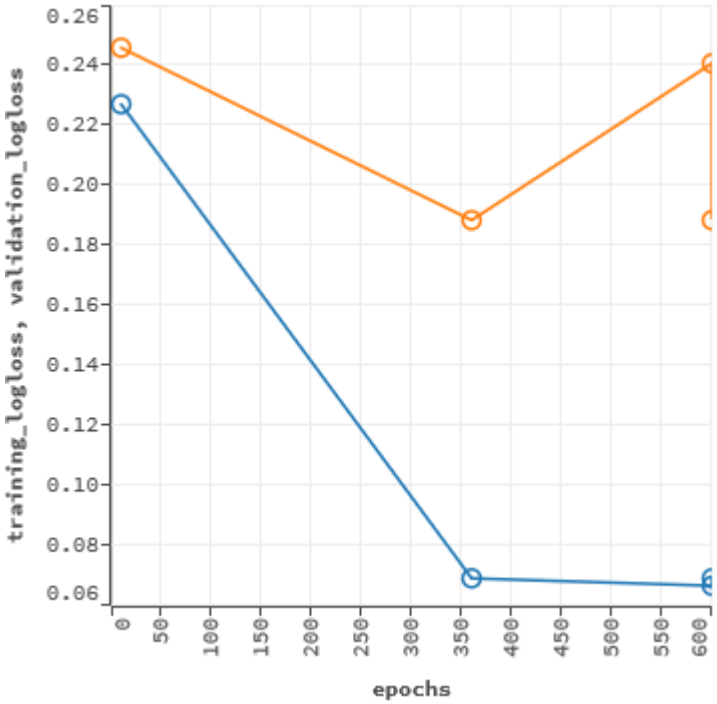


Figura 85. Gráfico modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	No	600	CrossEntropy

Tabla 120. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
16,728	0,024011	0,154954	97,041 %

Tabla 121. Resultados modelo 3

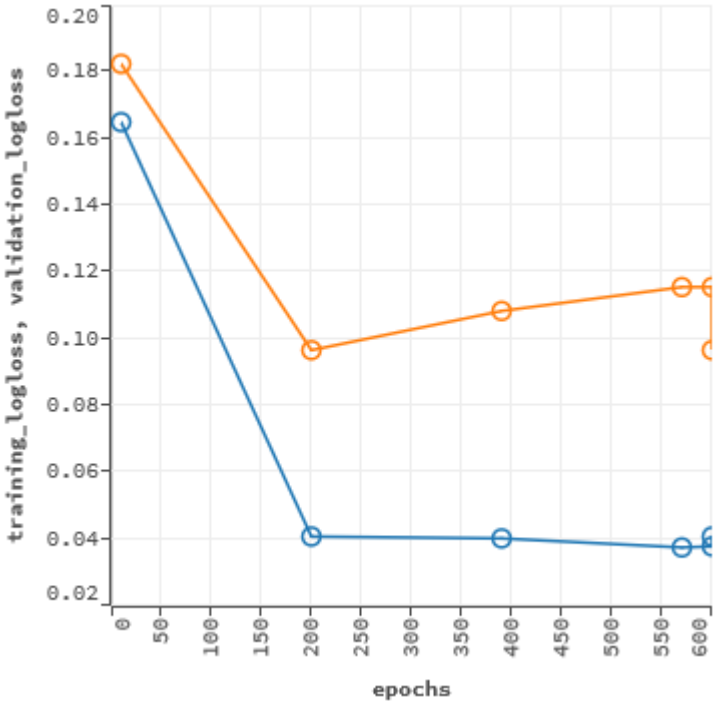


Figura 86. Gráfico modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Loss Function
80-20	10	600	CrossEntropy

Tabla 122. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	MSE	RMSE	Precisión
11,224	0,033228	0,182286	95,940 %

Tabla 123. Resultados modelo 4

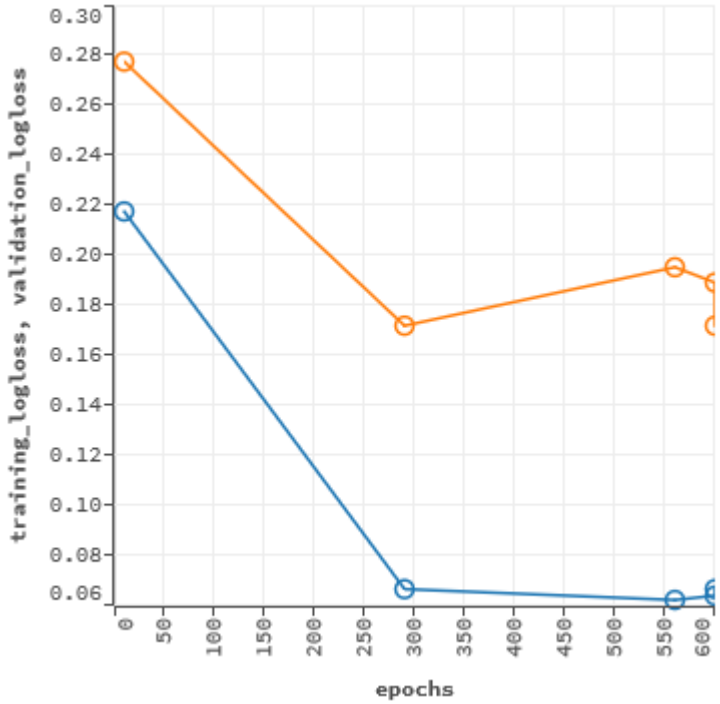


Figura 87. Gráfico modelo 4

Anexo II. Resultados de las pruebas en PyTorch

Las siguientes representaciones, tanto tabulares como gráficas, muestran los parámetros aplicados en el entrenamiento de los modelos, los resultados obtenidos de dicho entrenamiento, y como de aproximadas son las predicciones realizadas por el modelo en comparación con el conjunto de datos original. Todo esto realizado en el entorno de PyTorch.

Dataset 5.000 observaciones, hasta dos cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	600	35	0,01

Tabla 124. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
34,417	0,984	99,894 %

Tabla 125. Resultados modelo 1

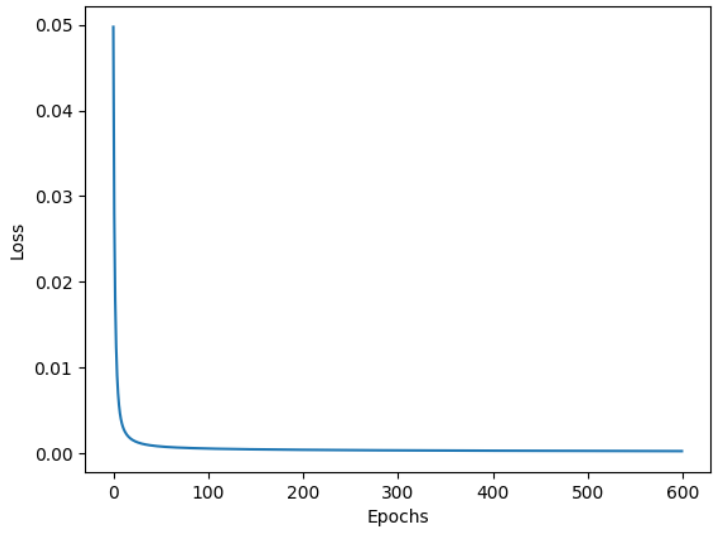


Figura 88. Gráfico de pérdida modelo 1

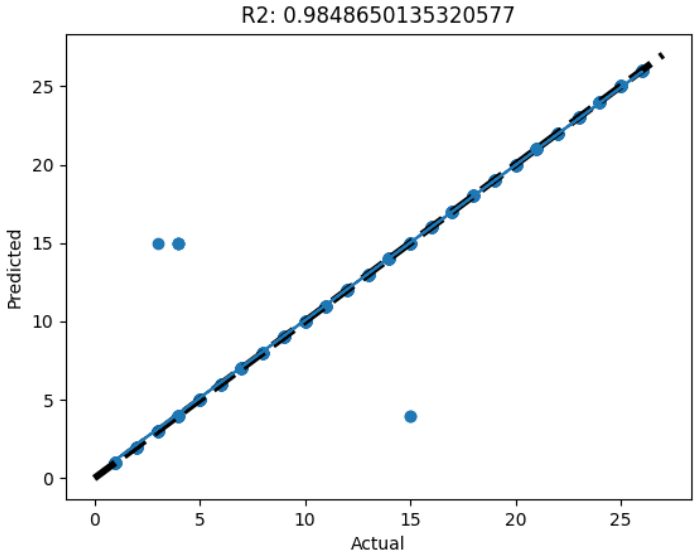


Figura 89. Gráfico de R2 modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	600	35	0,1

Tabla 126. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
37,258	0,989	99,914 %

Tabla 127. Resultado modelo 2

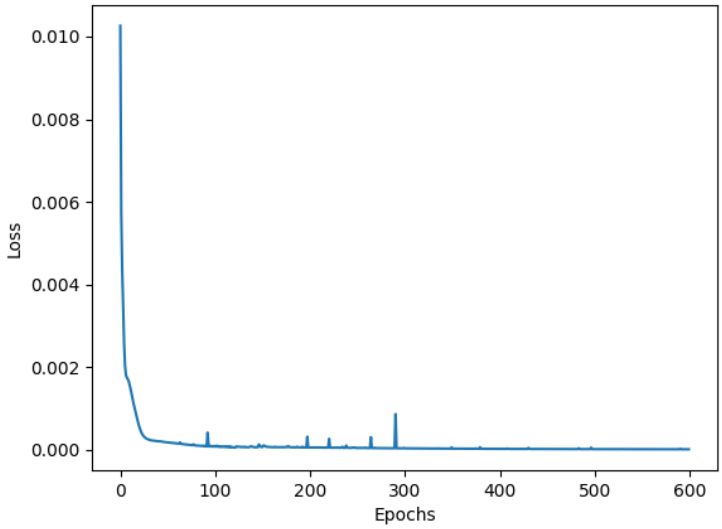


Figura 90. Gráfico de pérdida modelo 2

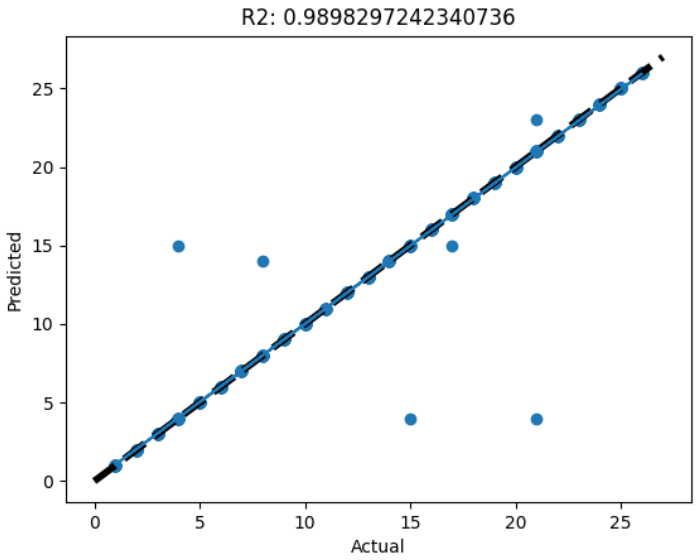


Figura 91. Gráfico de R2 modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	10	600	35	0,01

Tabla 128. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
48,391	0,967	99,708 %

Tabla 129. Resultados modelo 3

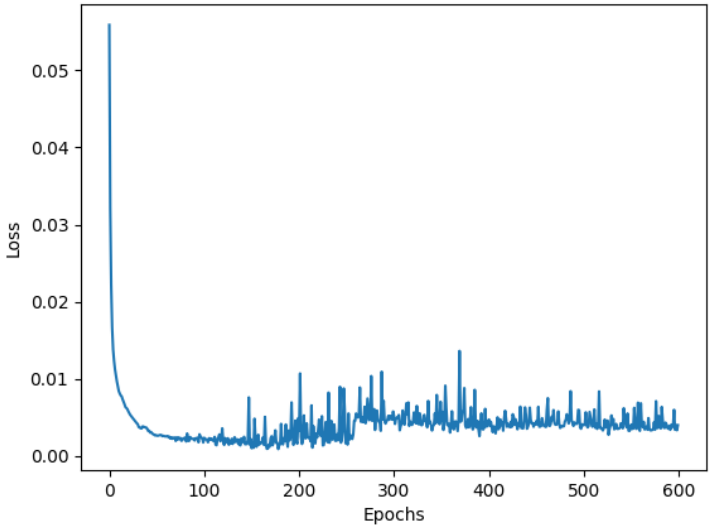


Figura 92. Gráfico de pérdida modelo 3

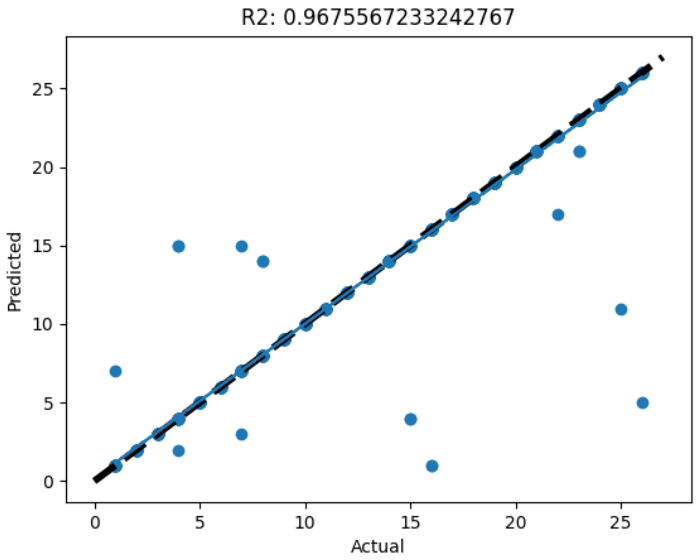


Figura 93. Gráfico de R2 modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate

80-20	10	600	35	0,1

Tabla 130. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
47,225	-0,089	96,916 %

Tabla 131. Resultados modelo 4

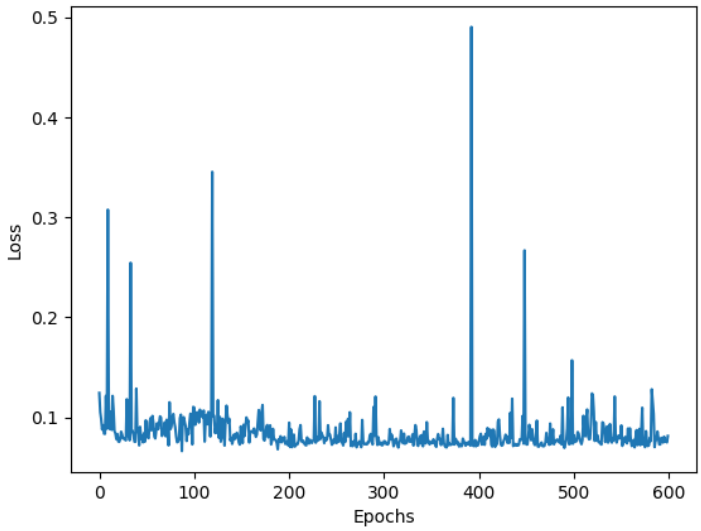


Figura 94. Gráfico de pérdida modelo 4

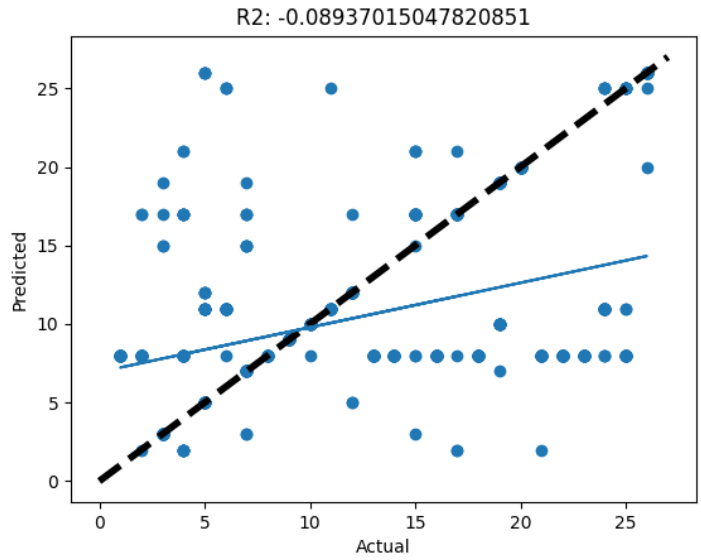


Figura 95. Gráfico de R2 modelo 4

Dataset 5.000 observaciones, hasta tres cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	100	35	0,01

Tabla 132. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
5,826	0,978	99,805 %

Tabla 133. Resultados modelo 1

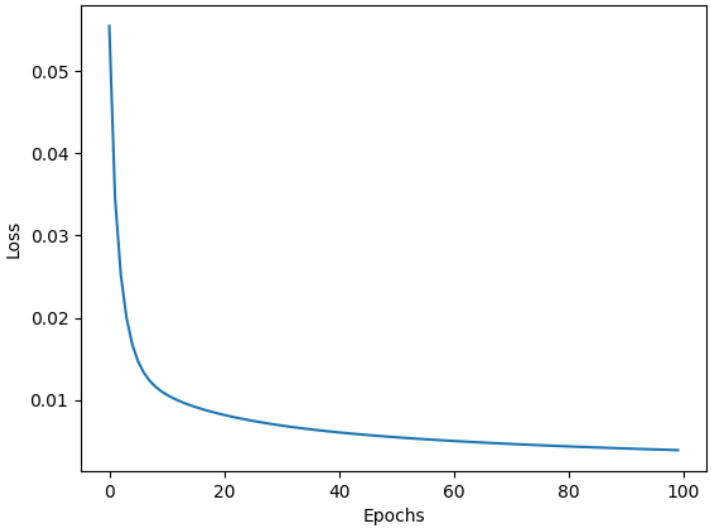


Figura 96. Gráfico de pérdida modelo 1

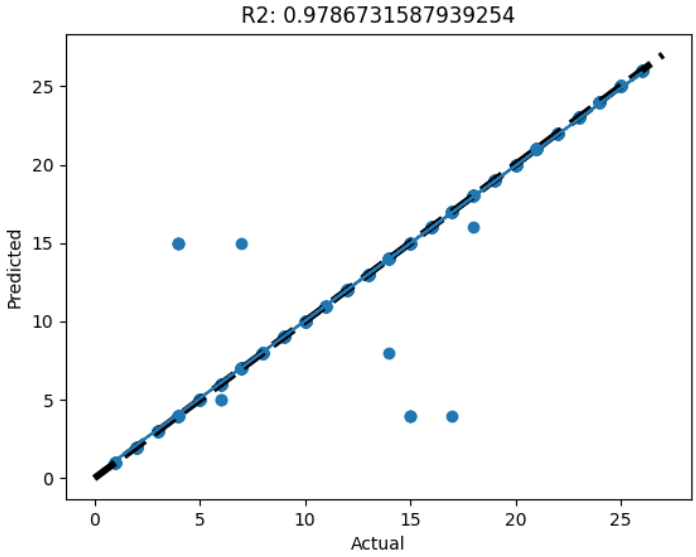


Figura 97. Gráfico de R2 modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	10	100	35	0,01

Tabla 134. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
7,674	0,946	99,419 %

Tabla 135. Resultados modelo 2

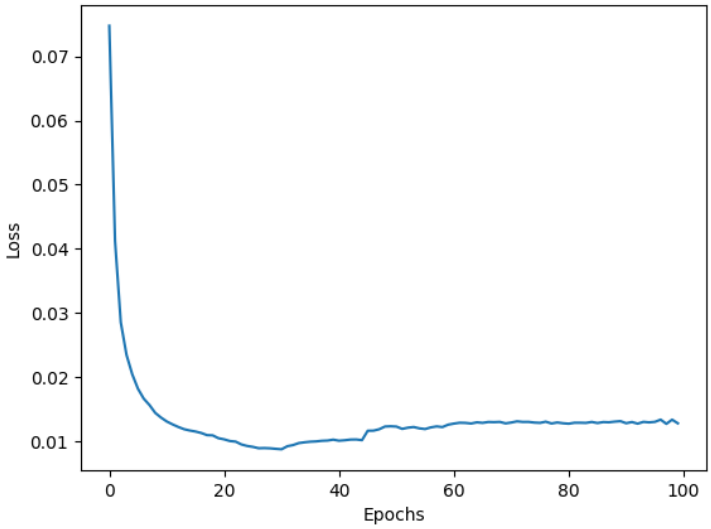


Figura 98. Gráfico de pérdida modelo 2

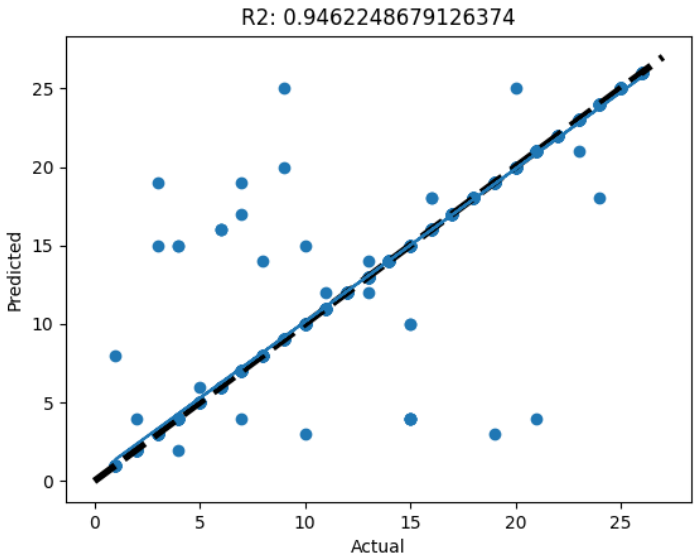


Figura 99. Gráfico de R2 modelo 2

Dataset 5.000 observaciones, hasta cinco cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	100	35	0,01

Tabla 136. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
5,84	0,931	99,575 %

Tabla 137. Resultados modelo 1

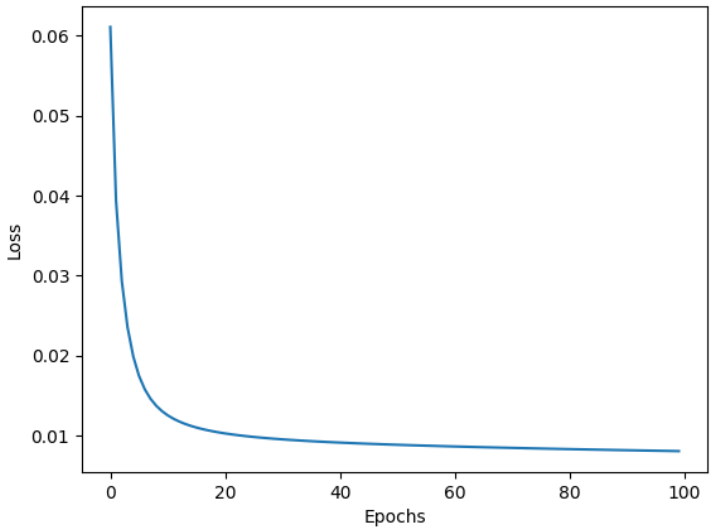


Figura 100. Gráfico de pérdida modelo 1

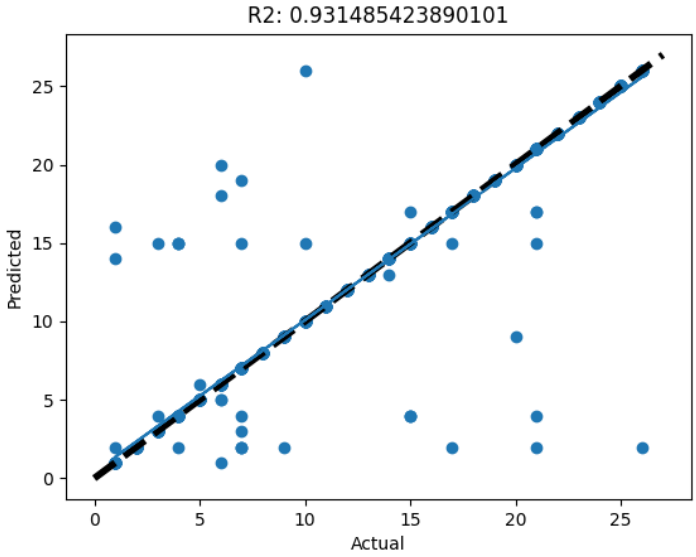


Figura 101. Gráfico de R2 modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	10	100	35	0,01

Tabla 138. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
10,918	0,953	99,634 %

Tabla 139. Resultados modelo 2

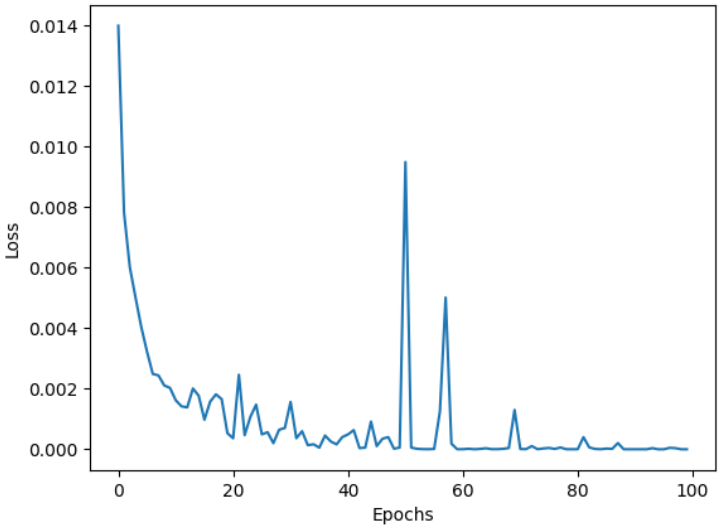


Figura 102. Gráfico de pérdida modelo 2

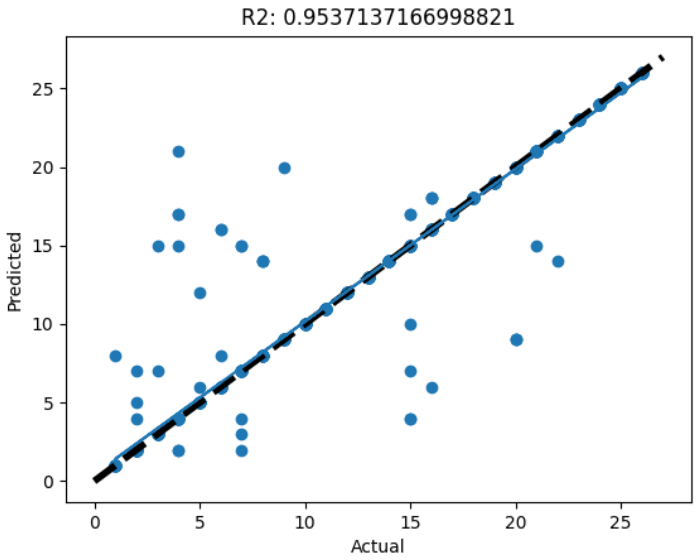


Figura 103. Gráfico de R2 modelo 2

Dataset 10.000 observaciones, hasta dos cambios
Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	10	35	0,01

Tabla 140. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
1,206	0,988	99,890 %

Tabla 141. Resultados modelo 1

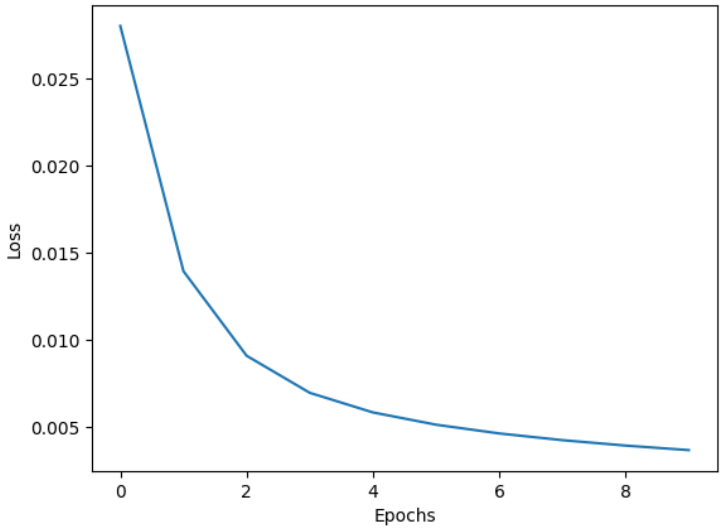


Figura 104. Gráfico de pérdida modelo 1

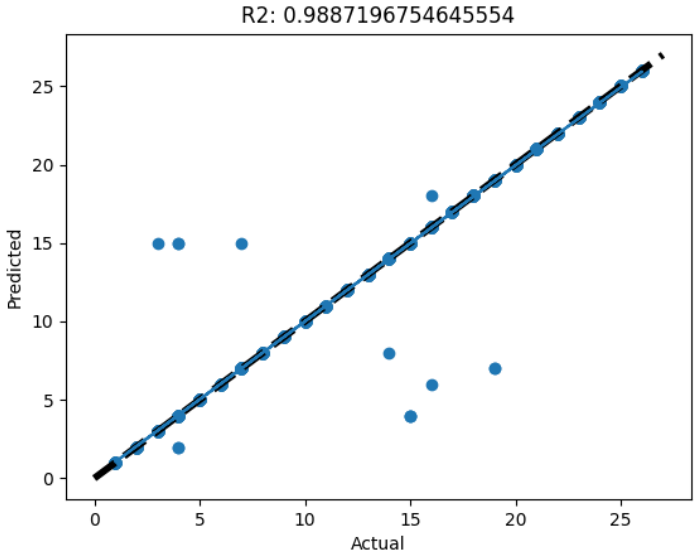


Figura 105. Gráfico de R2 modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	128	10	35	0,01

Tabla 142. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
2,015	0,975	99,809 %

Tabla 143. Resultados modelo 2

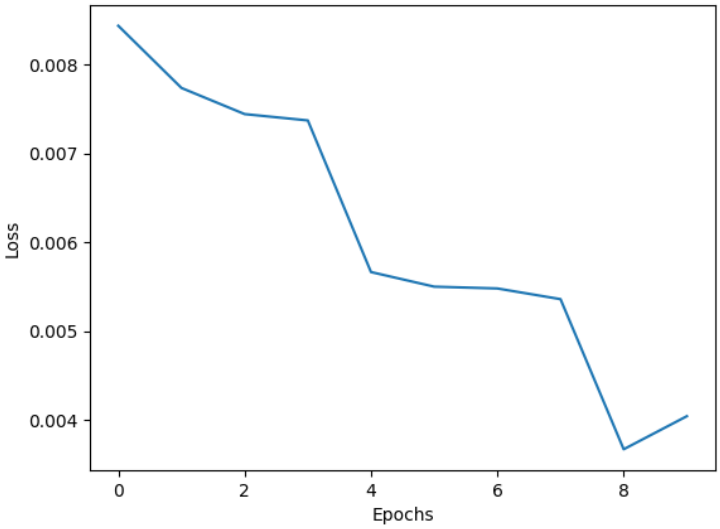


Figura 106. Gráfico de pérdida modelo 2

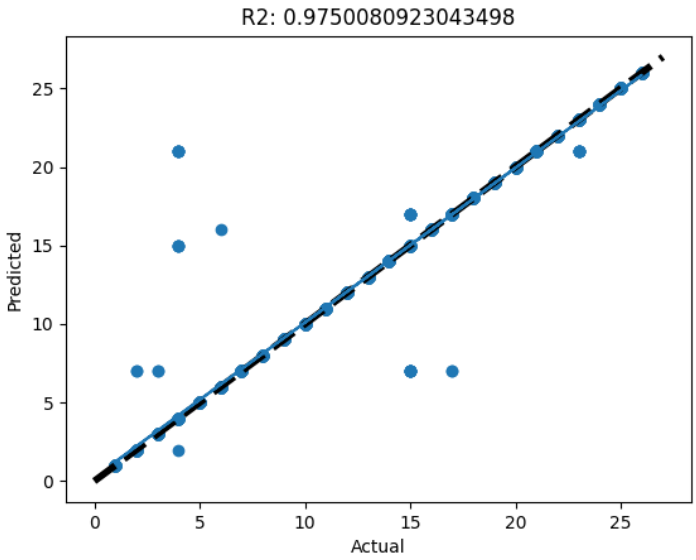


Figura 107. Gráfico de R2 modelo 2

Dataset 10.000 observaciones, hasta tres cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	10	35	0,01

Tabla 144. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
1,202	0,987	99,838 %

Tabla 145. Resultados modelo 1

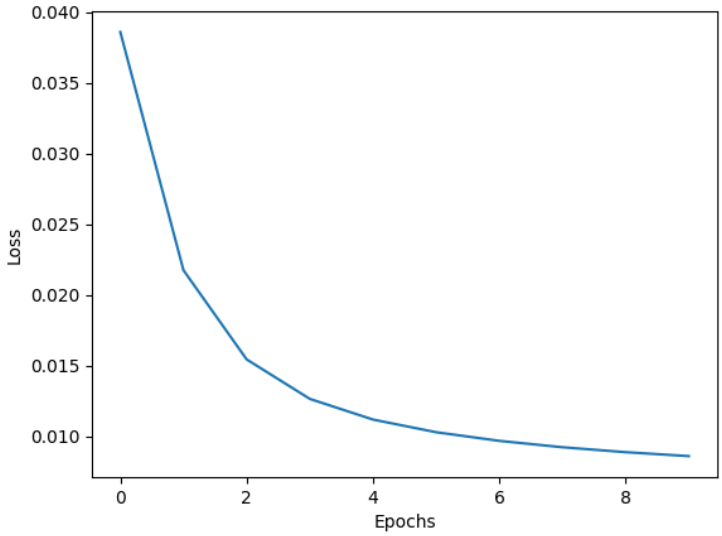


Figura 108. Gráfico de pérdida modelo 1

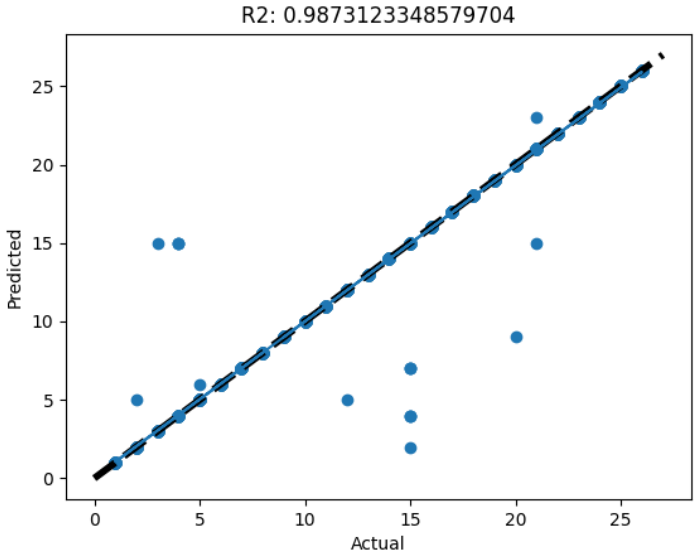


Figura 109. Gráfico de R2 modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	100	300	0,01

Tabla 146. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
0,262	0,983	99,490 %

Tabla 147. Resultados modelo 2

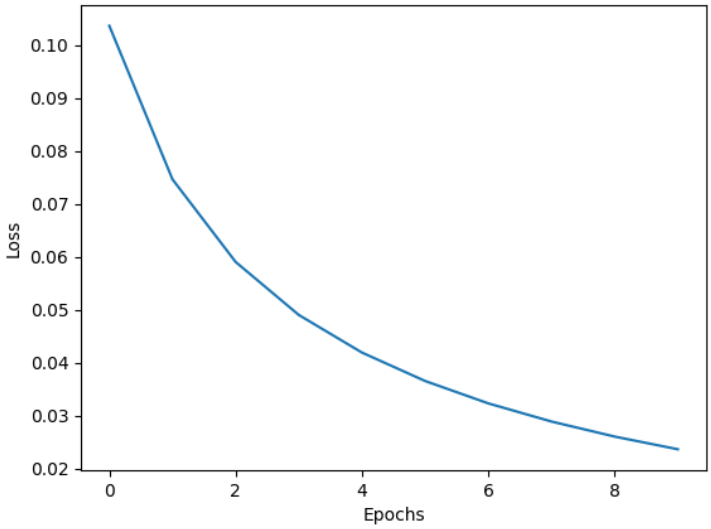


Figura 110. Gráfico de pérdida modelo 2

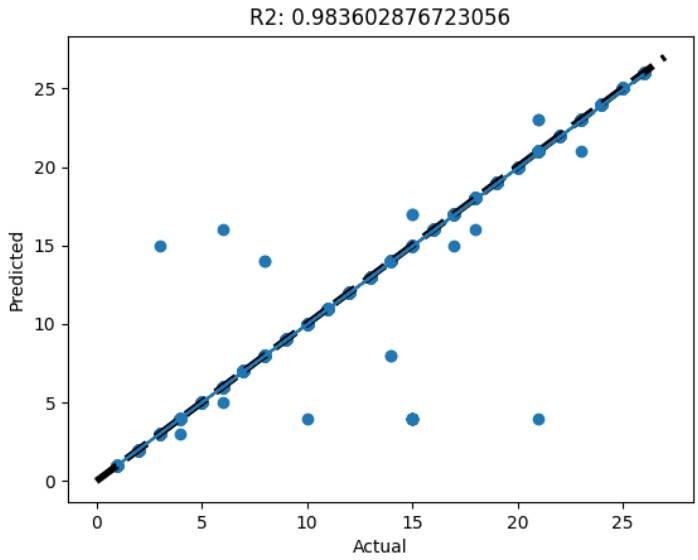


Figura 111. Gráfico de R2 modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	128	10	35	0,01

Tabla 148. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
1,926	0,986	99,902 %

Tabla 149. Resultados modelo 3

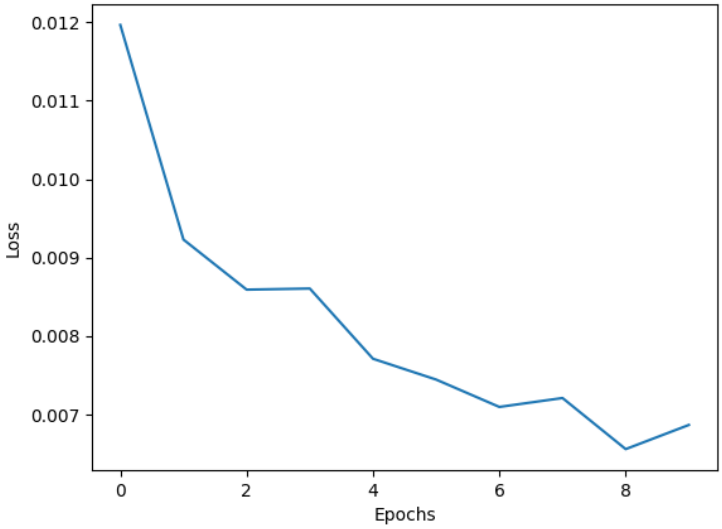


Figura 112. Gráfico de pérdida modelo 3

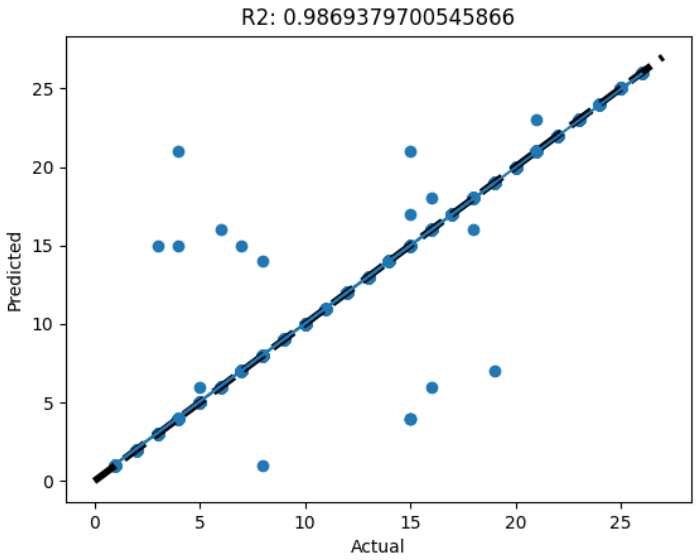


Figura 113. Gráfico de R2 modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	128	10	300	0,01

Tabla 150. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
0,367	0,987	99,888 %

Tabla 151. Resultados modelo 4

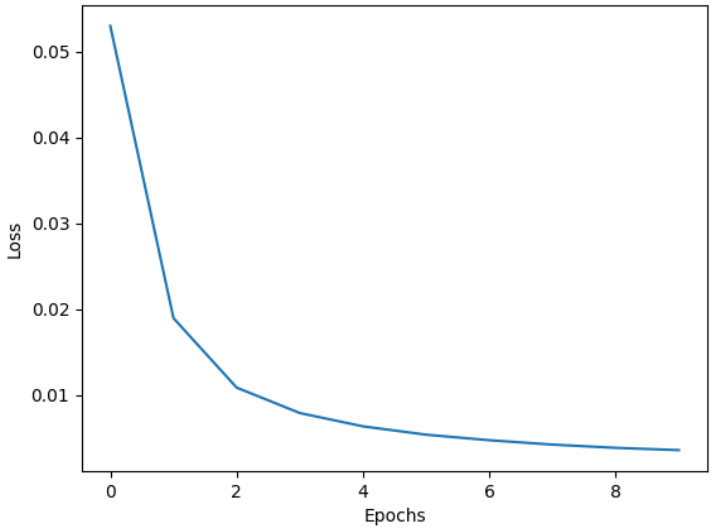


Figura 114. Gráfico de pérdida modelo 4

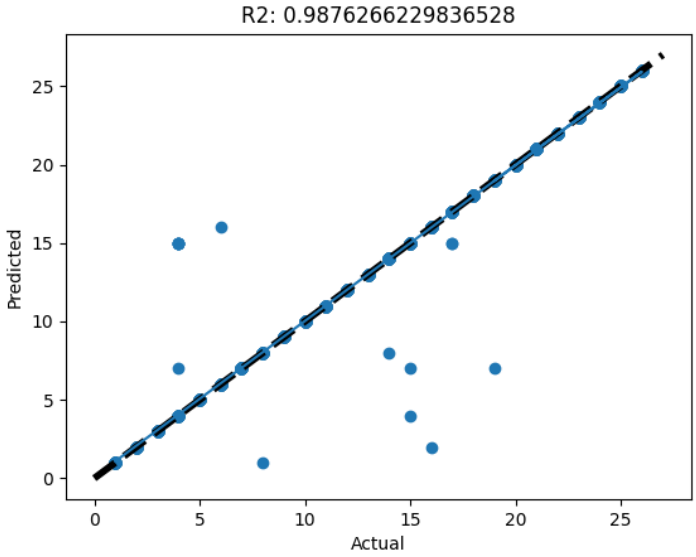


Figura 115. Gráfico de R2 modelo 4

Dataset 10.000 observaciones, hasta cinco cambios

Modelo 1

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	10	35	0,01

Tabla 152. Parámetros modelo 1

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
1,176	0,960	99,544 %

Tabla 153. Resultados modelo 1

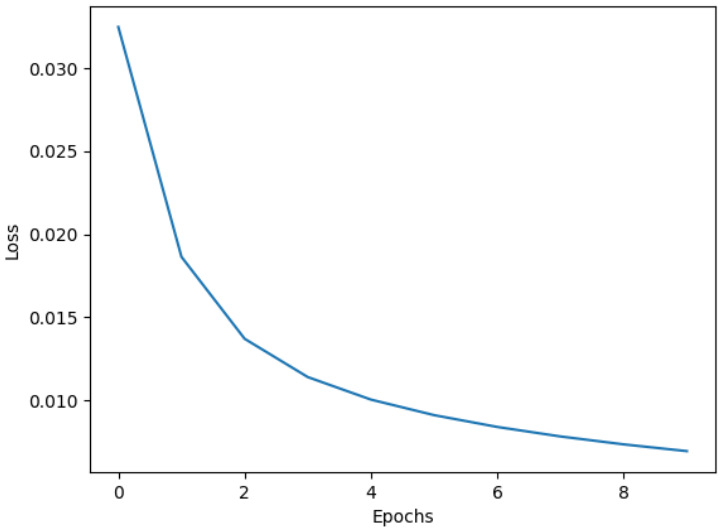


Figura 116. Gráfico de pérdida modelo 1

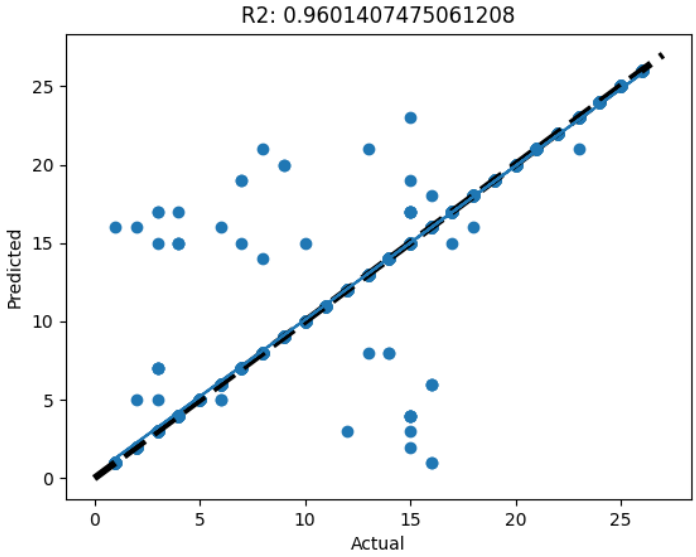


Figura 117. Gráfico de R2 modelo 1

Modelo 2

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	No	10	35	0,01

Tabla 154. Parámetros modelo 2

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
0,285	0,961	99,243 %

Tabla 155. Resultados modelo 2

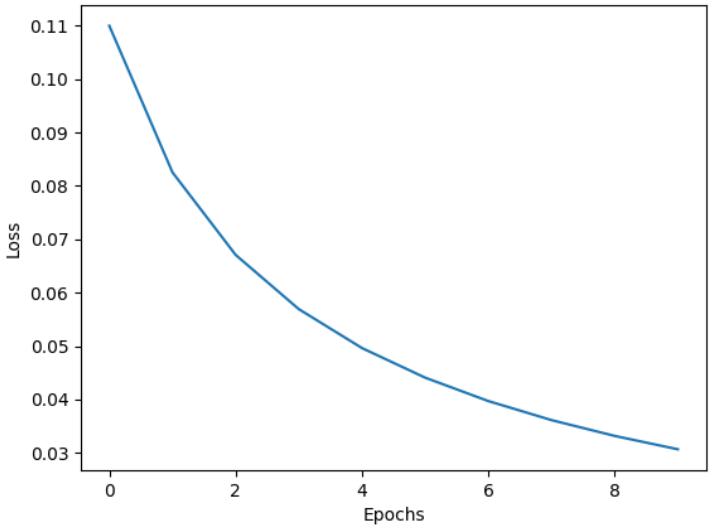


Figura 118. Gráfico de pérdida modelo 2

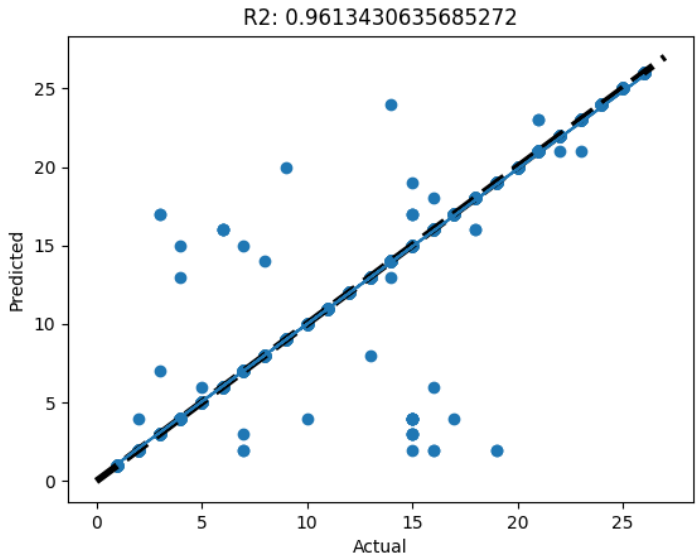


Figura 119. Gráfico de R2 modelo 2

Modelo 3

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	128	10	35	0,01

Tabla 156. Parámetros modelo 3

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
1,989	0,953	99,732 %

Tabla 157. Resultados modelo 3

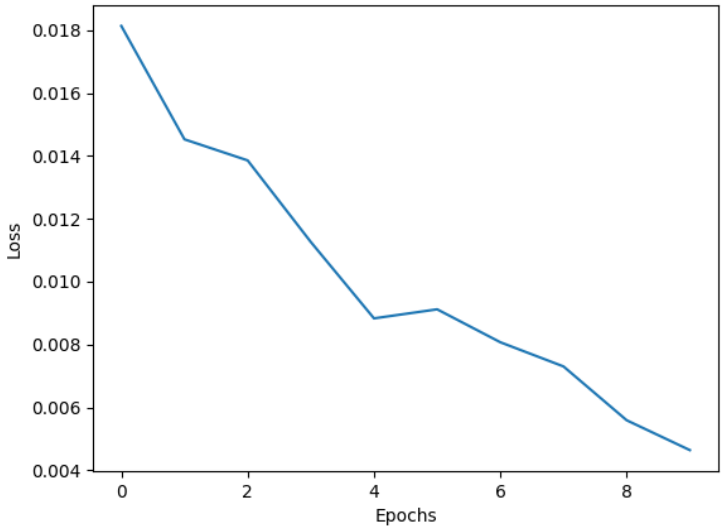


Figura 120. Gráfico de pérdida modelo 3

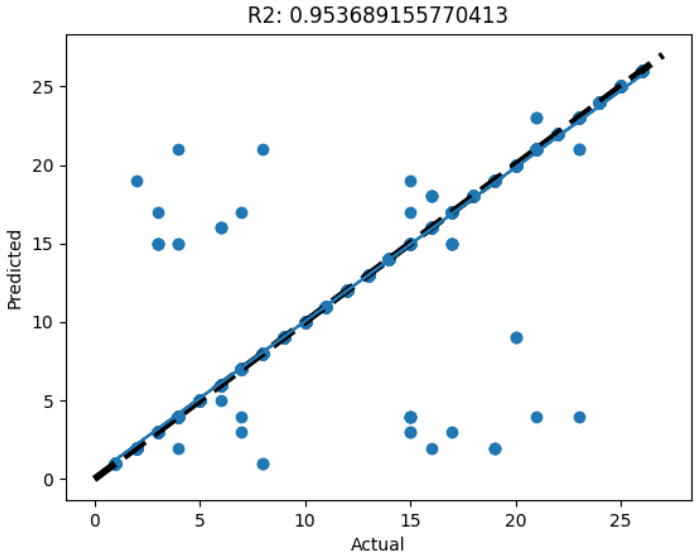


Figura 121. Gráfico de R2 modelo 3

Modelo 4

Parámetros:

División	Capa oculta (neuronas)	Epochs	Batch	Learning rate
80-20	128	10	300	0,01

Tabla 158. Parámetros modelo 4

Resultados:

Tiempo de entrenamiento (s)	Coeficiente de determinación	Precisión
0,370	0,952	99,717 %

Tabla 159. Resultados modelo 4

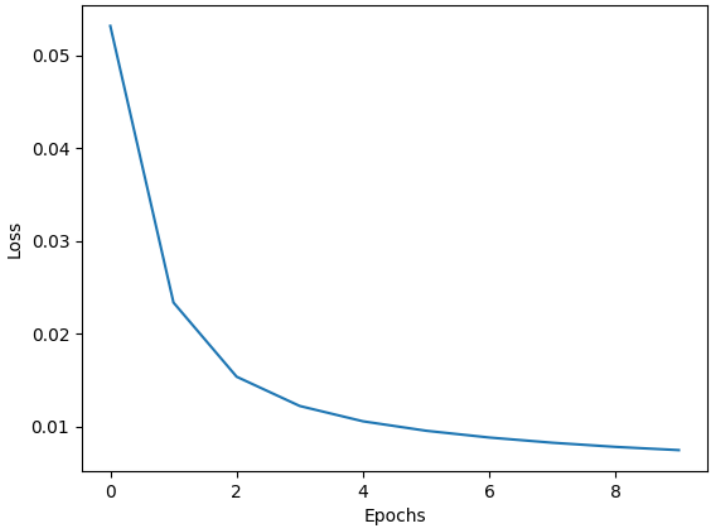


Figura 122. Gráfico de pérdida modelo 4

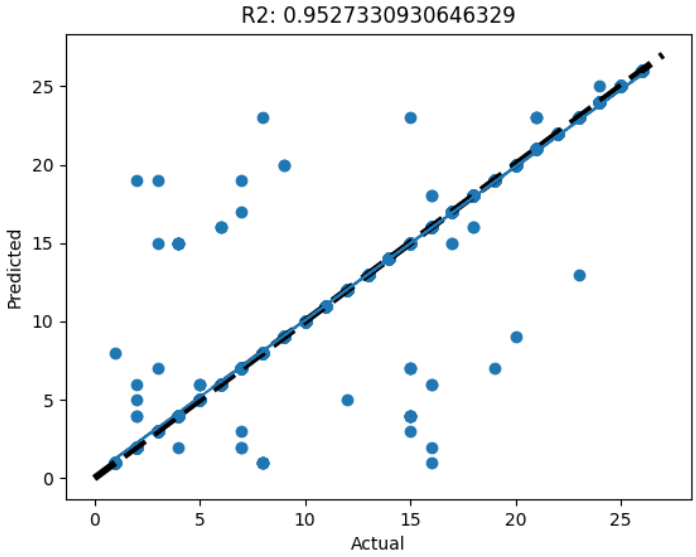


Figura 123. Gráfico de R2 modelo 4