
Universidad Zaragoza

# ‘Machine learning’ para simulación hidráulica: Predicción de inundaciones usando redes neuronales con información física

Pablo Baquedano Coarasa

Trabajo Fin de Grado
Facultad de Ciencias
Grado en Física

Directores:
Pilar García Navarro
Sergio Martínez Aranda

2024

# Índice

# 1. Introducción

## 1.1. Flujo transitorio 1D en lámina libre

Las ecuaciones que gobiernan la Mecánica de Fluidos son las ecuaciones de Navier-Stokes. Estas ecuaciones no tienen solución analítica, por lo que usualmente se ha recurrido a métodos de cálculo numérico para resolverlas. Estos métodos se han desarrollado mucho en las últimas décadas y son capaces de resolver numéricamente las ecuaciones que gobiernan el flujo en lámina libre ([1], [2], [3]), sin embargo, esto tiene un coste computacional muy elevado. Para solventar este problema, estos métodos usan aproximaciones como el promedio de las ecuaciones en la vertical (aproximación 2D) y el promedio en la sección transversal (aproximación 1D). El modelo de aguas poco profundas es una aproximación usada para simular flujos en la superficie. Este trabajo se apoyará en la herramienta PEKA1D, desarrollada por el Departamento de Ciencia y Tecnología de Materiales y Fluidos, que resuelve las ecuaciones del modelo de aguas poco profundas en aproximación 1D ([1]). La resolución numérica de este sistema requiere de una discretización espacial y otra temporal, así como de condiciones iniciales y de contorno.

$$\frac{\partial \mathbf{U}(x,t)}{\partial t} + \frac{\partial \mathbf{F}(x,\mathbf{U})}{\partial x} = \mathbf{H}(x,\mathbf{U}) \tag{1}$$

$$\mathbf{U} = \begin{pmatrix} A \\ Q \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} Q \\ \frac{Q^2}{A} + gI_1 \end{pmatrix}, \quad \mathbf{H} = \begin{pmatrix} 0 \\ g[I_2 + A(S_0 - S_f)] \end{pmatrix} \tag{2}$$

donde A es el área transversal mojada, $S_0$ es la pendiente del fondo en la dirección longitudinal del cauce, $S_f$ la pendiente de fricción e $I_1$ es la fuerza de presión hidrostática.

La principal ventaja que ofrecen las redes neuronales frente a estos métodos de cálculo numérico es la habilidad para extraer las relaciones entre variables de entrada y variables de salida, sin necesidad de considerar las leyes físicas en el proceso intermedio. Las redes neuronales son una herramienta muy útil para aplicar a diversos campos, como el reconocimiento de patrones, la clasificación de objetos, el procesamiento de lenguaje natural, la conducción autónoma, las finanzas y el *marketing*, los videojuegos o la robótica. Entre las posibles aplicaciones también se encuentra la Mecánica de Fluidos.

Su aplicación a este campo se ha debido al interés por encontrar modelos que estimen los caudales hidrográficos de manera precisa. Esto puede ser de gran utilidad para poder planificar y gestionar los recursos hídricos [4]. Además la correcta implementación de estos métodos puede suponer tener la capacidad de predicción de fenómenos como las inundaciones, que tan dañinas son en algunas áreas ([5]), con sistemas de alerta temprana.

El comienzo del estudio del campo de las redes neuronales artificiales se sitúa hace unos 80 años [6]. El interés en esta forma de computación creció especialmente desde que en 1986

se desarrollase un marco teórico riguroso para el funcionamiento de las redes neuronales: el algortimo de *backpropagation* [7]. Tras esto, evolucionó enormemente el campo de la computación y de la aplicación práctica de esta idea.

Las redes neuronales están inspiradas en el funcionamiento del cerebro humano [8]. Cuando a una de nuestras neuronas le llega una señal, las dendritas recogen la señal y la pesan según su importancia, si esta supera un cierto valor de umbral, la neurona se activa y envía una señal a otras neuronas.

## 1.2. La neurona matemática

Análogamente, se definen las neuronas matemáticas. En este caso, la señal es un vector de entrada, $\vec{x} = (x_1, x_2, ..., x_n)$ de dimensión $n$, llamado *input*. Cada neurona tiene $n$ pesos, $\omega_i$, que multiplican a cada valor del vector de entrada. Esto hace que la red se pueda ajustar a la importancia de cada variable, pues podemos suponer que no todas tendrán la misma. Cada neurona también tiene un sesgo, $b$, que determina el umbral de activación de la neurona mencionado anteriormente. A la neurona habrá que asignarle una función de activación $a(z)$, donde $z = \sum_{i=1}^{n} \omega_i x_i - b$, siguiendo la notación de [9]. Es decir, cada neurona recibe el vector de *input* de dimensión $n$ y, tras pesar los valores y restar el sesgo, dará un resultado dependiendo de la función de activación $a(z)$ elegida.

Hay multitud de opciones para elegir esta función, así que habrá que hacer una elección inteligente teniendo en cuenta las características del problema que queremos abordar con la red neuronal. La inmensa mayoría de problemas que se pueden intentar resolver con una red neuronal son de característica no lineal, por lo tanto, a(z) deberá ser una función no lineal, capaz de recoger la complejidad del problema. Si nuestro problema tuviera una característica lineal, lo apropiado sería elegir una función de activación lineal. Las ecuaciones del modelo de aguas poco profundas claramente tienen una característica no lineal, así que sólo tiene sentido elegir una función de activación que pueda representar esto.

Algunas funciones de activación no lineales son:

$$relu(z) = max(0, z) \tag{3}$$

$$selu(z) = \begin{cases} sz & \text{si } z > 0 \\ s\alpha(e^z - 1) & \text{si } z < 0 \end{cases} \tag{4}$$

donde $s$ y $\alpha$ son constantes.

$$leaky\_relu(z) = \begin{cases} -0{,}2z & \text{si } z < 0 \\ z & \text{si } z > 0 \end{cases} \tag{5}$$

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

$$silu(z) = \frac{z}{1 + e^{-z}} \tag{7}$$

Algunas son apropiadas para problemas de clasificación y otras son apropiadas para problemas de regresión.

Un problema de clasificación es aquel en el que se busca catalogar los datos de entrada en una serie de categorías, mientras que en un problema de regresión buscamos encontrar los valores de los pesos y los sesgos que hagan que la red neuronal pueda predecir los valores de salida de cualquier dato de entrada. En los problemas de clasificación queremos asignar los datos de entrada a categorías discretas en la salida y en los problemas de regresión queremos predecir el valor de salida de una (o varias) variables continuas. Por tanto, nuestro problema es un problema de regresión.

## 1.3. Estructura de una red neuronal

Una vez definido el funcionamiento de una neurona, podemos hablar de la estructura de una red neuronal. La estructura que sigue una red neuronal consiste en varias capas de un número $m$ de neuronas, este número será uno de los hiperparámetros con los que podremos ajustar la configuración de la red neuronal.

En una red neuronal tenemos una capa de entrada, una capa de salida y una o varias capas ocultas, el número de capas ocultas también es un hiperparámetro que variaremos hasta elegir el óptimo. En principio, todas las neuronas de una capa están conectadas con todas las de la siguiente capa, aunque también se pueden definir configuraciones de redes neuronales en las que se eliminen algunas conexiones.

La capa de entrada tiene la función de únicamente transmitir la información de los datos de entrada a todas las neuronas de la siguiente capa, por lo que las neuronas de esta capa no deben tener ninguna función de activación. El número de neuronas en la capa de activación debe ser igual al número de variables de entrada, es decir, la dimensión del vector de *input*.

La capa de salida es en la que se recogen los valores que da la red neuronal como resultado para cada variable de salida y, como estamos en un problema de regresión, la función de activación de estas neuronas debe ser la función lineal, también llamada identidad. El número de neuronas en esta capa debe ser igual al número de variables de salida, es decir, la dimensión del vector de *output*.

En las capas ocultas es donde tiene que realizarse el ajuste de los pesos y los sesgos que recoja la complejidad no lineal del problema, de manera que la red neuronal nos dé predicciones apropiadas, por lo que aquí es donde tendremos la función de activación no lineal. Si ponemos menos capas y neuronas de las necesarias es posible que no tengamos la capacidad de reproducir la complejidad del problema, si ponemos demasiadas neuronas y/o capas, probablemente tengamos problemas de *overfitting*, esto significa que nuestra red será capaz de hacer predicciones muy buenas para los datos de entrada que le proporcionamos en el entrenamiento, pero tendrá problemas para generalizar esos resultados cuando le pidamos que haga una predicción para unos valores distintos a los del entrenamiento.

En concreto, la estructura descrita pertenece a una red perceptrón multicapa totalmente conectada, un tipo de red neuronal artificial en el que cada salida de una neurona de la capa i es entrada de todas las neuronas de la capa i+1. Además, la información sólo se transmite hacia adelante (*feedforward*). Es decir, todas las neuronas están conectadas con todas las de la capa siguiente y no hay conexiones entre neuronas de una misma capa. Este tipo de red neuronal es muy usada en problemas de hidrología [10], [11].

Este es el esquema de una red neuronal con un *input* de dimensión 2, un *output* de dimensión 2 y 3 capas ocultas de 5 neuronas cada una:

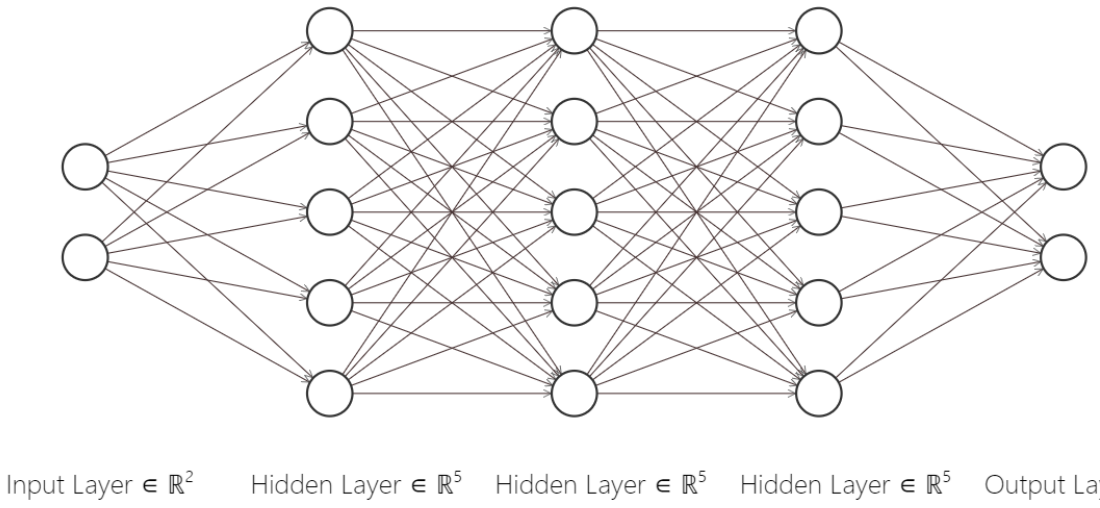



Figura 1: Esquema de un red neuronal con un *input* de dimensión 2, un *output* de dimensión 2 y 3 capas ocultas (*hidden*) de 5 neuronas cada una.

## 1.4. Aprendizaje automático

Una vez hemos definido la configuración de nuestra red neuronal, tenemos que pasar a hablar de cómo va a aprender la relación entre las variables de entrada y las de salida y, por tanto, de cómo va a saber qué valores debe asignar a los pesos y los sesgos. Así pues, tenemos que hablar de la función de pérdida.

Se define *época* como un ciclo completo de evaluación de todos los datos de entrenamiento. En este contexto, la *función de pérdida*, es la encargada de medir la diferencia entre el valor de una variable de salida calculado por la red neuronal en una cierta época del entrenamiento y el valor real de esa variable de salida, dado por los datos de entrenamiento que proporcionamos a la red neuronal. Esta diferencia se puede calcular, por ejemplo, definiendo la función de pérdida, $\mathcal{L}$, como el error cuadrado medio (MSE):

$$\mathcal{L} = \frac{1}{N}\sum_i (\vec{y}_{real}(\vec{x}_i) - \vec{y}_{NN}(\vec{x}_i))^2 \tag{8}$$

donde N es el número de datos de entrenamiento usados.

También se va a usar como métrica para evaluar el entrenamiento la función de pérdida de validación (*val_loss*), que es la función de pérdida evaluada con una fracción de los datos de entrenamiento dedicada únicamente a este propósito, este mecanismo evita el *overfitting*.

El método que va a usar la red neuronal se llama *método del gradiente descendente*. En cada paso de entrenamiento, los pesos y los sesgos se actualizan según las fórmulas:

$$\omega_i^{n+1} = \omega_i^n - \eta\frac{\partial \mathcal{L}}{\partial \omega_i} \tag{9}$$

$$b_i^{n+1} = b_i^n - \eta\frac{\partial \mathcal{L}}{\partial b_i} \tag{10}$$

donde $\eta$ es el *learning rate*, un hiperparámetro. Este método se basa en que, como el gradiente siempre va hacia el máximo de una función, para buscar el mínimo nos debemos mover en la dirección opuesta. A mayor $\eta$ más avanzaremos hacia el mínimo en cada paso. Si tenemos un $\eta$ muy pequeño, corremos el riesgo de quedarnos atrapados en un mínimo local de $\mathcal{L}$, pero si tenemos un $\eta$ muy grande podemos no encontrar el mínimo absoluto por estar siempre oscilando alrededor de este. La red neuronal calcula las derivadas parciales con el algoritmo de *backpropagation*.

## 1.5. Aproximadores universales

Como se muestra en el trabajo *Teorema de Aproximación Universal: prueba constructiva basada en conjuntos semisimpliciales* [12], existe un teorema matemático que dice que toda función real entre conjuntos compactos puede ser aproximada tanto como se desee por un capa oculta de una red neuronal. Por esto se dice que las redes neuronales son aproximadores universales.

Como una red neuronal artificial verifica las hipótesis del teorema de Stone-Weierstrass, se demuestra la existencia de una red neuronal que puede aproximar una función continua en un conjunto compacto de $\mathbb{R}^n$.

# 2. Objetivos y metodología

El objetivo general de este trabajo es estudiar la aplicación de las redes neuronales a problemas hidráulicos como alternativa a los métodos de cálculo numérico tradicionales, que son muy laboriosos y con un coste computacional elevado.

Por ello, nos planteamos otros objetivos específicos que servirán para el fin general:

- Estudiar cómo afectan todos los hiperparámetros a la capacidad de predicción de la red neuronal: el número de capas ocultas, el número de neuronas por capa oculta, el *learning rate*, los parámetros de ponderación, el *batch size*, el optimizador y las funciones de activación de las neuronas de las capas ocultas.
- Construir una red neuronal en la que vamos a tener que ajustar dichos parámetros con la intención de optimizar los resultados predictivos.
- Realizar evaluaciones mediante métricas de la calidad del entrenamiento de la red neuronal y de su fiabilidad al predecir variables hidráulicas.

Antes de exponer la metodología que usaremos para alcanzar el objetivo propuesto, se procede a hacer una breve explicación de los hiperparámetros mencionados:

- Número de capas ocultas: este hiperparámetro es uno de los que determina la profundidad de la red neuronal. Un número muy bajo puede implicar que la red neuronal no va a ser lo suficientemente compleja como para capturar las dependencias entre hiperparámetros, que como hemos dicho, son dependencias no lineales con un cierto grado de complejidad. Por otro lado, un número muy alto puede provocar que tengamos un exceso de parámetros entre todos los pesos y los sesgos para ajustar esas dependencias (*overfitting*).
- Número de neuronas por capa oculta: al igual que el anterior, debe ser suficientemente alto como para capturar bien las dependencias del problema, pero sin excedernos innecesariamente.
- *Learning rate*: es la constante $\eta$ de las ecuaciones (9) y (10). Como se mencionó debajo de esas ecuaciones, determina la "velocidad" con la que nos movemos por el gradiente descendente. Las redes neuronales son muy sensibles a variaciones en este parámetro, ya que el aprendizaje automático es exitoso cuando la función de pérdida llega a su mínimo absoluto.
- Optimizador: Es un algoritmo dedicado a llegar lo antes posible al mínimo absoluto además de evitar mínimos locales. Durante todo este trabajo se ha usado el optimizador *Adam*, el más popular en problemas de regresión.

- *Batch size*: Indica el tamaño de los lotes de datos que se le pasan juntos al método de gradiente descendente. Si este tiene un valor pequeño (comparado con el número de datos de entrenamiento), es esperable tener más oscilaciones en la búsqueda del mínimo.
- Parámetros de ponderación: son los $\lambda_1$ y $\lambda_2$ que aparecerán más adelante en la ecuación (11), nos dan la capacidad de hacer que los errores en una variable pesen más en la función de pérdida que los errores en la otra variable.
- Funciones de activación: determinan el valor que devuelve una neurona.

Por todas estas cuestiones, las redes neuronales requieren de un análisis metódico y preciso de ajuste de estos parámetros para funcionar de manera óptima.

Plantearemos ahora algunas consideraciones metodológicas del proceso. Vamos a abordar un problema en el que tenemos un río con una geometría simple: un fondo plano con un obstáculo en forma de trapecio (figura 2):

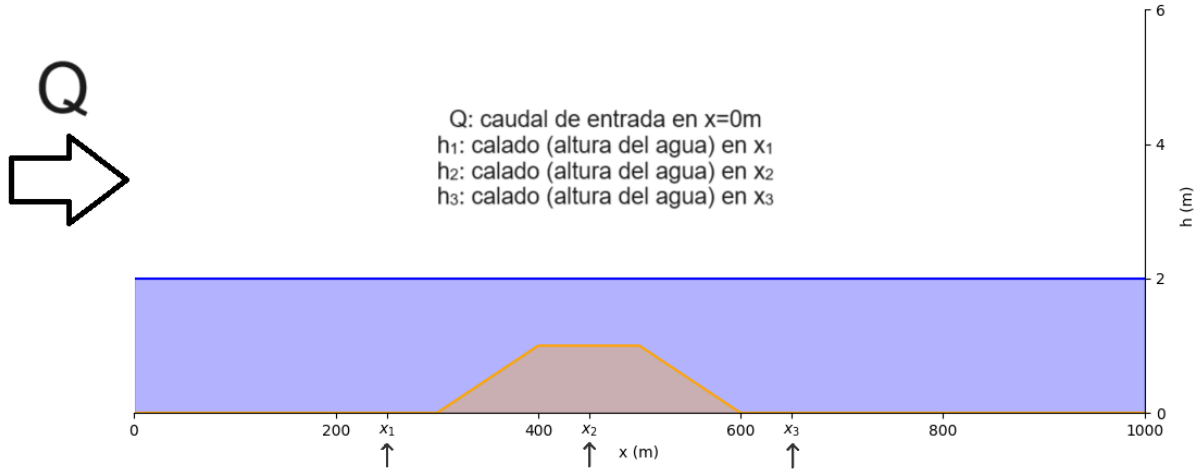



Figura 2: Esquema y geometría del problema.

Este es el esquema del problema. Distinguiremos dos casos según el tipo del caudal de entrada. Definiremos las variables de entrada y salida de cada caso en los subapartados expuestos a continuación.

Dependiendo del caudal de entrada podemos encontrarnos cambios de régimen entre régimen subcrítico ($Fr < 1$) y régimen supercrítico ($Fr > 1$), donde el número de Froude es $Fr = u/c$, con $c = \sqrt{gh}$ ; $u$ la velocidad en el eje $x$.

Los datos de entrenamiento los simularemos con la herramienta de cálculo numérico PEKA1D, explicada en la introducción, y para construir la red neuronal usaremos la librería TensorFlow de Python. Pasamos ahora a explicar la metodología específica seguida para ambos casos:

## 2.1. Caso estacionario: caudal de entrada constante

En este primer caso, vamos a considerar un caudal de entrada constante durante todo el tiempo de simulación, por lo que $Q$ será la única variable del vector de *input*.

Como el caudal de entrada es constante durante toda la simulación, al final de la misma se alcanzará un estado estacionario en el cual el calado (altura del agua) se mantendrá estable en un valor. Las tres variables de salida serán esos calados en el estado estacionario al final del tiempo de simulación en tres puntos distintos de la posición x: un punto aguas arriba del obstáculo ($x_1 = 250m$), un punto aguas abajo del obstáculo ($x_3 = 650m$) y un punto en el medio del obstáculo ($x_2 = 450m$), como se observa en la figura 2. Estos calados se denominarán $h_1$, $h_2$ y $h_3$ y formarán el vector de *output*.

Proporcionaremos datos de entrenamiento para 28 caudales de entrada aleatorios entre 0 y $100m^3/s$ a la red neuronal y posteriormente pediremos que haga una predicción para unos valores de entrada distintos a los de entrenamiento.

## 2.2. Caso transitorio: caudal de entrada dependiente del tiempo

Considerando la misma geometría (figura 2), ahora vamos a tratar un caudal de entrada cambiante con el tiempo. Esto va a resultar en que no se alcanza un estado estacionario: el calado en todos los puntos va cambiando a lo largo del tiempo. En concreto, vamos a trabajar con un hidrograma de entrada triangular, como se ve en la figura 3:

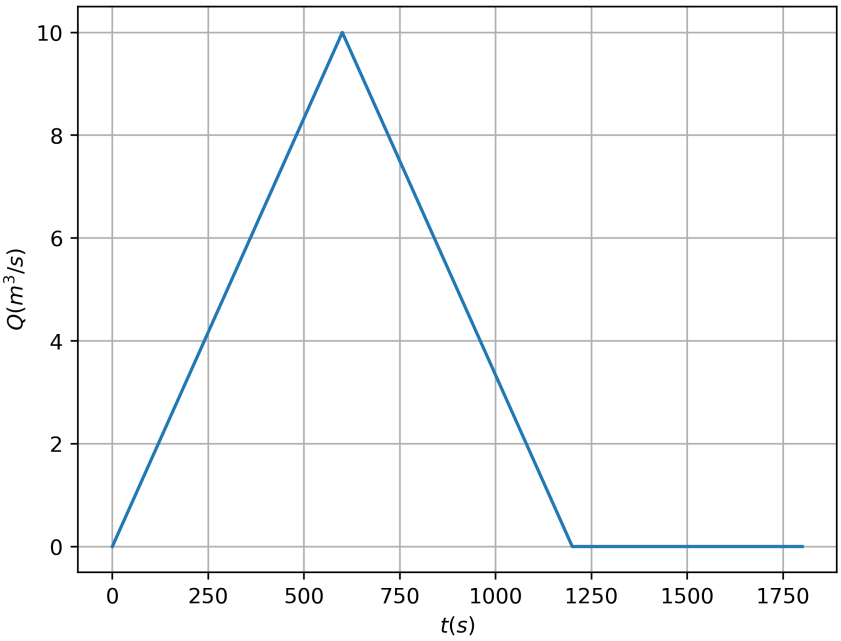



Figura 3: Hidrograma triangular de entrada para un caudal pico de $10m^3/s$ y un tiempo de pico de $600s$.

En este hidrograma, el caudal de entrada de la figura 2 queda definido por dos variables: el caudal de pico, $Q_p$, y el tiempo de pico, $t_p$. Estas serán las dos variables del vector de *input*.

En este caso, nos fijaremos en la posición de medida $x_1$ (sonda 1). Como estamos en el caso transitorio (ya no se llega a un calado estable al final de la simulación sino que va cambiando a lo largo de toda esta), queremos estudiar el calado máximo alcanzado en esa posición de medida durante la simulación, $h_{max}$. También queremos estudiar el instante del tiempo en el cual se ha alcanzado este máximo, $t_{max}$. Estas serán las dos variables del vector de *output*. Repetiremos este mismo proceso para estudiar por separado las posiciones de medida $x_2$ (sonda 2) y $x_3$ (sonda 3).

Se ha definido una función de pérdida para este caso, personalizada, que me permita normalizar las diferencias y añadir unos factores de ponderación $\lambda_1$ y $\lambda_2$ que se puedan ajustar dando mayor o menor importancia a cada término de la función de pérdida, $\mathcal{L}$:

$$\mathcal{L} = \frac{1}{N}\left(\sum_{i=1}^{n} \lambda_1 \frac{|h_{i,real} - h_{i,NN}|}{h_{max}} + \lambda_2 \frac{|t_{i,real} - t_{i,NN}|}{t_{max}}\right) \tag{11}$$

donde el sumatorio recorre los N datos de entrenamiento proporcionados.

Hemos realizado varias simulaciones con PEKA1D para combinaciones aleatorias de parejas $(Q_p, t_p)$ dentro de los intervalos $0 < Q_p < 100 m^3/s$ y $0 < t_p < 1200 s$. Con estos datos de entrada entrenaremos a la red neuronal, con el objetivo de que luego le preguntaremos una serie de parejas de datos de entrada $(Q_p, t_p)$ distintos de los que ha entrenado y hará una predicción para las variables de salida, $(h_{max}, t_{max})$. Nosotros habremos simulado también con la herramienta PEKA1D estos casos, por lo que tendremos los valores que debería darnos la red neuronal. Así podremos calcular el error normalizado tanto en la predicción de $h_{max}$ como de $t_{max}$.

En este apartado, el caso más complejo, es donde nos vamos a detener a hacer el análisis sistemático de todos los hiperparámetros de la red neuronal. El análisis sistemático consiste en ir estudiando una a una las consecuencias de las variaciones en los hiperparámetros de la red neuronal. Para ello, dejaremos fijados el resto de hiperparámetros y haremos que únicamente uno de ellos tome distintos valores dentro de un rango razonable. Analizaremos la calidad del entrenamiento y la calidad de la predicción ofrecida con las métricas correspondientes para así elegir un valor para el parámetro en estudio. Una vez ya seleccionado el valor de este parámetro, lo dejaremos fijado para el resto de simulaciones y pasaremos a estudiar otro parámetro.

Si las predicciones para variables como el calado son correctas, podría entrenarse una red neuronal con datos reales medidos en un río con el fin de predecir inundaciones.

# 3. Resultados y discusión

## 3.1. Caso estacionario: caudal constante

Vamos a usar este caso como una primera aproximación a la aplicación de las redes neuronales para problemas hidráulicos. Propocionamos a la red neuronal 28 datos de entrenamiento, usamos 5 neuronas por capa oculta y 2 capas ocultas.

La evolución de la función de pérdida durante el entrenamiento se muestra en la figura 4:

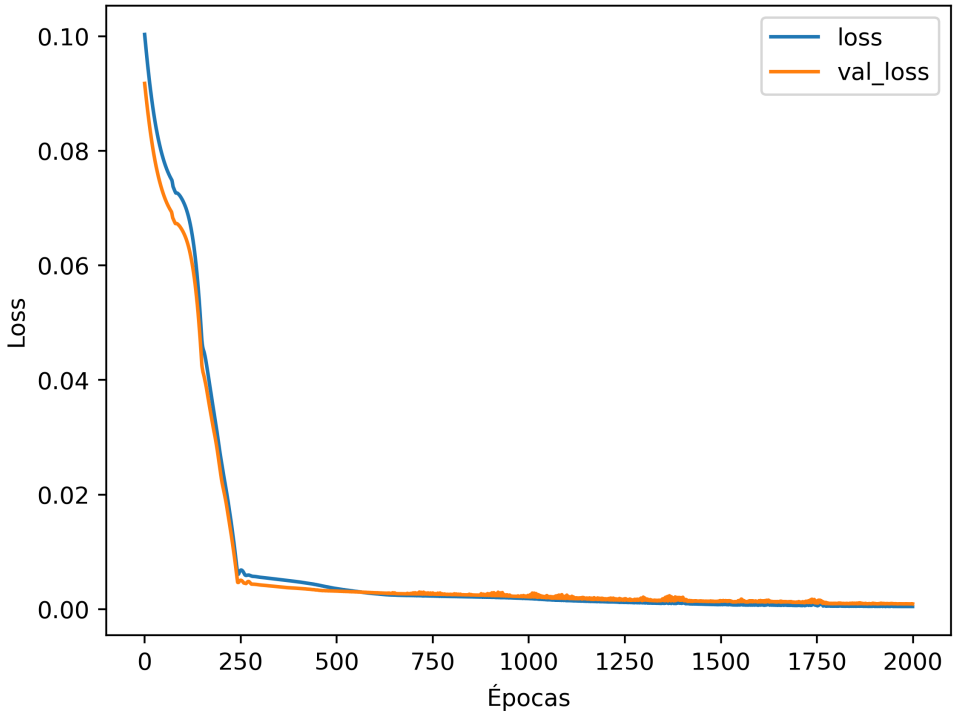



Figura 4: Evolución de la función de pérdida a lo largo del entrenamiento.

Los resultados para la predicción del calado en función del caudal de entrada en los 3 puntos de medida se muestran en la figura 5.

Los errores normalizados se recogen en la tabla 1:

| Variable | $h_1$ | $h_2$ | $h_3$ |
|---|---|---|---|
| error | 0.0023 | 0.0100 | 0.0024 |

Tabla 1: Errores de $h_1$, $h_2$ y $h_3$ en la predicción.

Como este caso es tan simple, desde la primera prueba los errores obtenidos son extraordinariamente bajos, por lo tanto, haremos la optimización de parámetros en el segundo

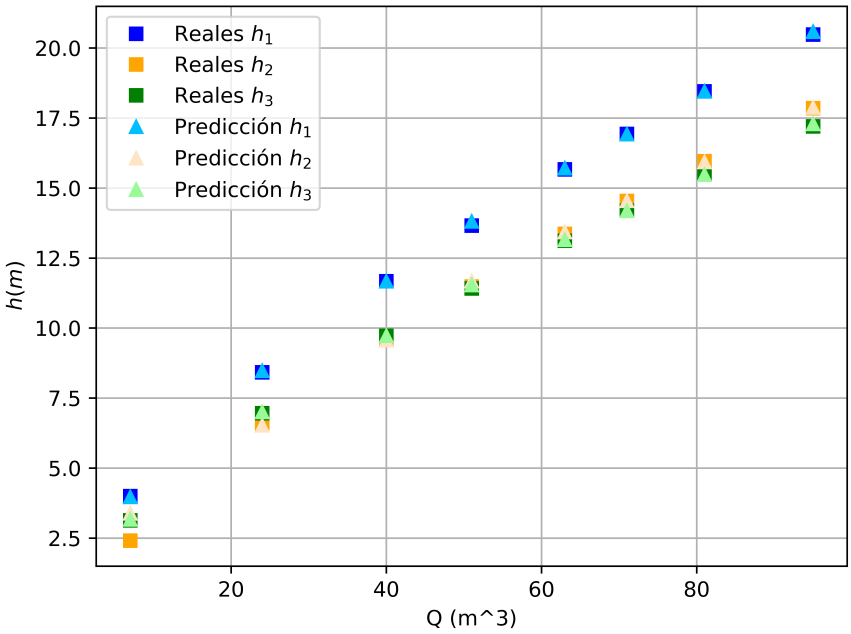



Figura 5: Predicción de $h_1$, $h_2$ y $h_3$ en función de $Q$, con los colores indicados en la leyenda.

caso, más complejo e interesante. El error en $h_2$ es más alto que los otros dos porque, como se aprecia en la figura 5, la predicción de $Q = 7m^3/s$ para esta variable se encuentra notablemente más alejada del valor real.

## 3.2. Caso transitorio: caudal de entrada dependiente del tiempo

En este caso, como ya no tenemos simplemente un caudal constante, sino un caudal dependiente del tiempo con forma triangular, vamos a dedicarle a este caso el esfuerzo a intentar afinar los valores de los parámetros que minimizan la función de pérdida y los errores en la predicción.

Para comenzar, en la ecuación (11) de la función de pérdida fijaremos $\lambda_1 = 1$ y $\lambda_2 = 1$, dando un peso por igual a los errores en ambas variables.

### 3.2.1. Estudio del número de capas

Comenzamos encontrando el número de capas ocultas más apropiado, para ello se han hecho varias pruebas con distinto número de ellas y el resto de hiperparámetros fijados. En la tabla 2 aparecen los valores de la función de pérdida (*loss*), la función de pérdida

para los datos de validación (*val_loss*) y el error en la predicción tanto para $h_{max}$ como para $t_{max}$:

| capas | loss | val_loss | error $h_{max}$ | error $t_{max}$ |
|---|---|---|---|---|
| 1 | 0.0013 | 0.0015 | 0.0506 | 0.0142 |
| 2 | 0.0013 | 0.0011 | 0.0394 | 0.0291 |
| 3 | 0.0009 | 0.0010 | 0.0259 | 0.0145 |
| 4 | 0.0009 | 0.0011 | 0.0248 | 0.0140 |
| 5 | 0.0019 | 0.0020 | 0.1232 | 0.0156 |

Tabla 2: Errores obtenidos para los distintos números de capas.

Como se puede ver los mejores valores son 3 y 4 capas ocultas. Elegiremos el valor de 3 capas ocultas de aquí en adelante porque, aunque los errores son muy ligeramente menores con 4 capas, la función de pérdida para los datos de validación es mejor para 3 capas. Además, trabajando con redes neuronales, a igualdad de resultados obtenidos es conveniente quedarse con la configuración de red más sencilla.

La figura 6 muestra la evolución de la función de pérdida para la red con 3 capas:

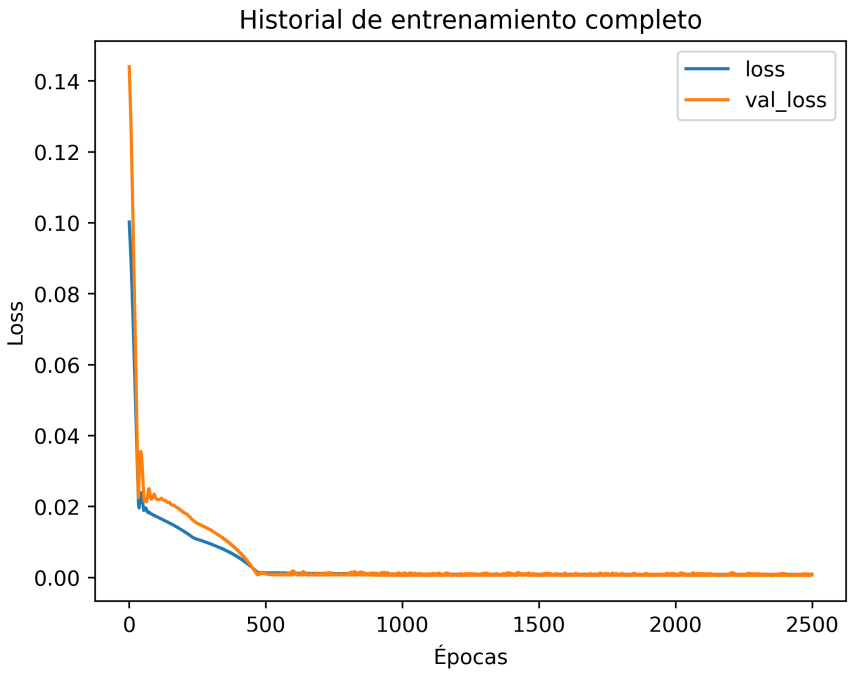



Figura 6: Evolución de la función de pérdida a lo largo del entrenamiento para la configuración de las 3 capas.

La función de pérdida tiene un descenso rápido al comienzo seguido de un período más lento hasta la convergencia final, donde ya se queda estable pese a que continúen pasando las épocas de entrenamiento, es decir, la función de pérdida ya ha alcanzado el

mínimo absoluto. Esta forma con dos períodos bien distinguidos de decrecimiento es típica y esperable.

### 3.2.2. Estudio del número de neuronas

Análogamente, ahora en la tabla 3 se reflejan los resultados obtenidos para distintos números de neuronas por capa oculta, dejando el resto de parámetros fijos:

| neuronas | loss | val_loss | error $h_{max}$ | error $t_{max}$ |
|---|---|---|---|---|
| 2 | 0.0300 | 0.0259 | 0.5619 | 0.5521 |
| 3 | 0.0013 | 0.0013 | 0.0603 | 0.0153 |
| 4 | 0.0012 | 0.0014 | 0.0555 | 0.0148 |
| 5 | 0.0014 | 0.0013 | 0.0550 | 0.0148 |
| 7 | 0.0017 | 0.0015 | 0.0502 | 0.0155 |
| 10 | 0.0013 | 0.0015 | 0.0506 | 0.0142 |

Tabla 3: Errores obtenidos para los distintos números de neuronas.

Representamos en la figura 7 únicamente los errores en las variables de salida respecto del número de neuronas para centrarnos en la calidad de la predicción:

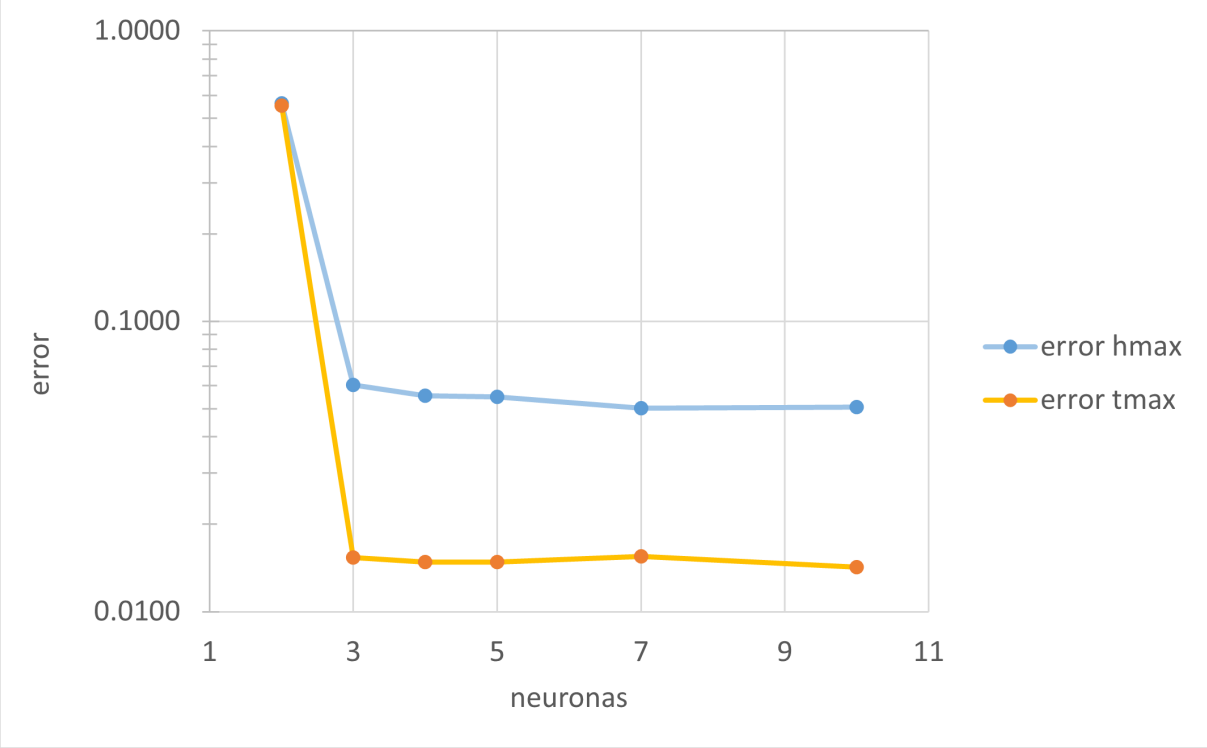



Figura 7: Errores de $h_{max}$ y $t_{max}$ en función del número de neuronas.

En la gráfica (figura 7) se observa que el error en $t_{max}$ no es muy sensible al cambio

en el número de neuronas, igual que tampoco era sensible al cambio en el número de capas, es una variable que se predice bien con facilidad. En cuanto al error en $h_{max}$, vemos cómo decrece al aumentar el número de neuronas hasta tener 7 y al añadir hasta 10 ya no mejora. Esto me hace elegir 7 como el valor óptimo del parámetro del número de neuronas y seguiremos con él para el resto de simulaciones. Al pasar de 7 neuronas por capa a 10, el error ya no mejora, probablemente porque se esté comenzando a dar el fenómeno mencionado anteriormente del *overfitting*.

### 3.2.3. Estudio del learning rate

Ahora nos disponemos a estudiar cómo afecta la variación del parámetro *learning rate*, para ello, dejando el resto de parámetros fijos, hemos obtenido los siguientes resultados (tabla 4) para la función de pérdida y los errores:

| *learning_rate* | loss | val_loss | error $h_{max}$ | error $t_{max}$ |
|---|---|---|---|---|
| 0.0005 | 0.0008 | 0.0007 | 0.0233 | 0.0147 |
| 0.0010 | 0.0007 | 0.0008 | 0.0171 | 0.0143 |
| 0.0015 | 0.0008 | 0.0007 | 0.0291 | 0.0138 |
| 0.0020 | 0.0027 | 0.0039 | 0.1138 | 0.0242 |

Tabla 4: Errores obtenidos para los distintos learning rate

Para estudiar la calidad de la predicción en función del *learning rate*, representamos en la figura 8 los errores en las variables de salida en función de este hiperparámetro:

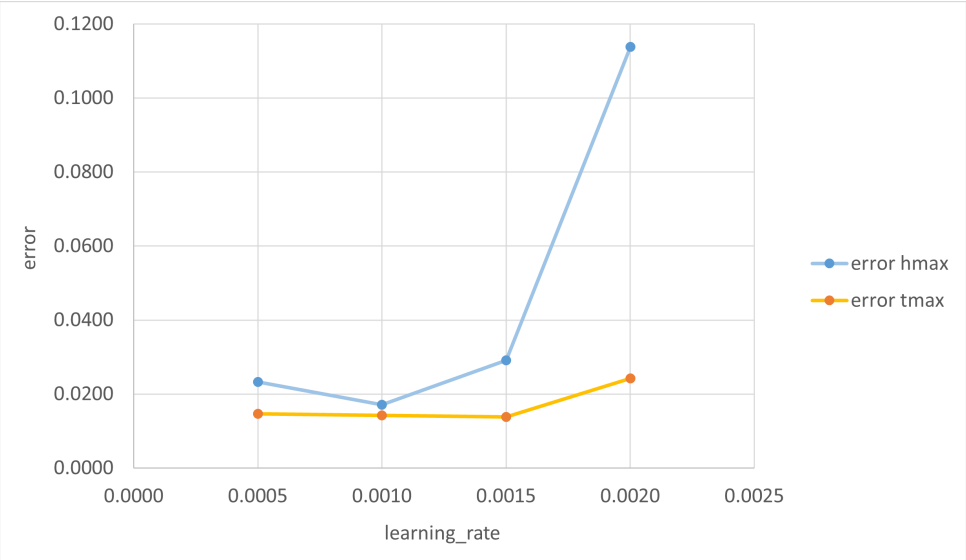



Figura 8: Errores de $h_{max}$ y $t_{max}$ en función del valor del hiperparámetro *learning rate*.

La evolución del valor final del entrenamiento de la función de pérdida en función del parámetro *learning rate* es:

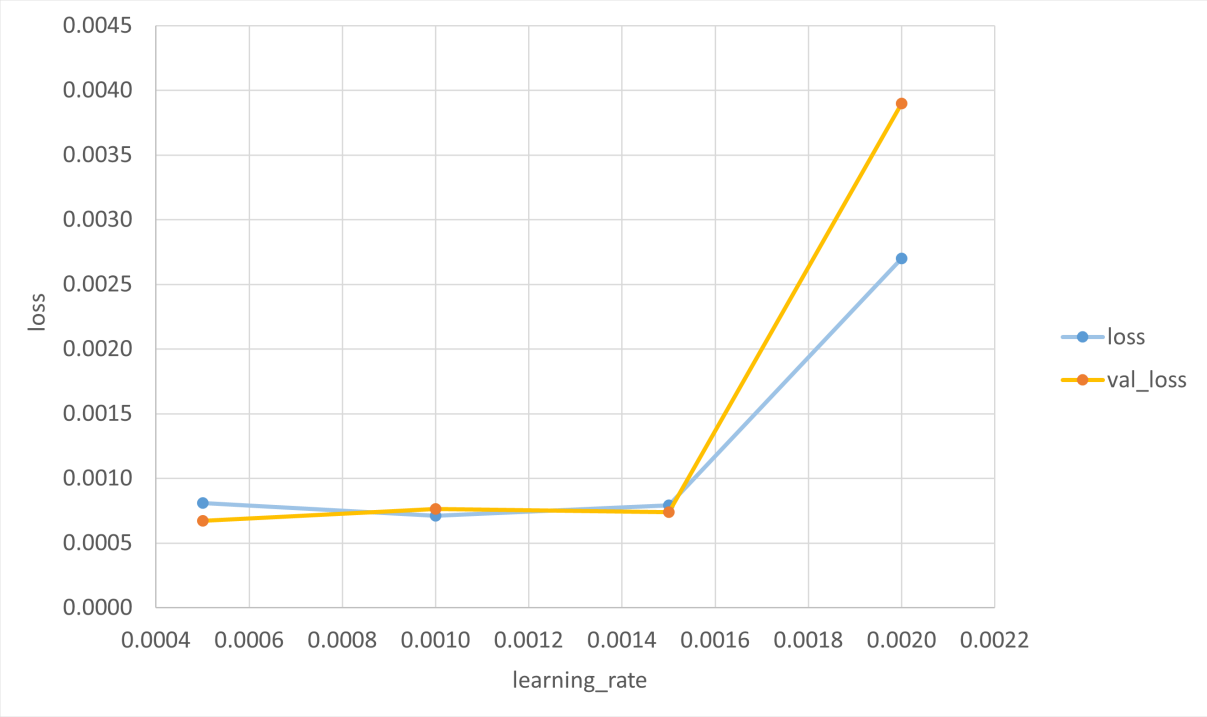



Figura 9: Valor final del entrenamiento de la función de pérdida para los distintos valores de *learning rate*.

A la vista de los resultados, me parece el más óptimo para continuar el valor de *learning rate* de 0.001. Este valor nos ha dado los errores más bajos y además es lo suficientemente pequeño como para que no se pueda salir de un "salto" del mínimo absoluto a un mínimo local.

### 3.2.4. Estudio de $\lambda_1$

Como estamos viendo, $h_{max}$ tiene mayor error que $t_{max}$, entonces, cambiando el parámetro $\lambda_1$ de la función de pérdida definida a mano en la ecuación (11), podemos dar más peso a la variable $h_{max}$, de manera que un error en esta variable hace aumentar más la función de pérdida que un error en $t_{max}$. Así, al tener que minimizar la función de pérdida, estamos creando un incentivo para reducir los errores en $h_{max}$ ligeramente más que los errores en $t_{max}$.

Al haber cambiado la función de pérdida, los valores de *loss* y *val_loss* de este apartado no son comparables a los de los apartados en los que estábamos usando $\lambda_1 = 1$.

Los resultados obtenidos para los errores en las variables de salida en función de $\lambda_1$ (figura 10) son:

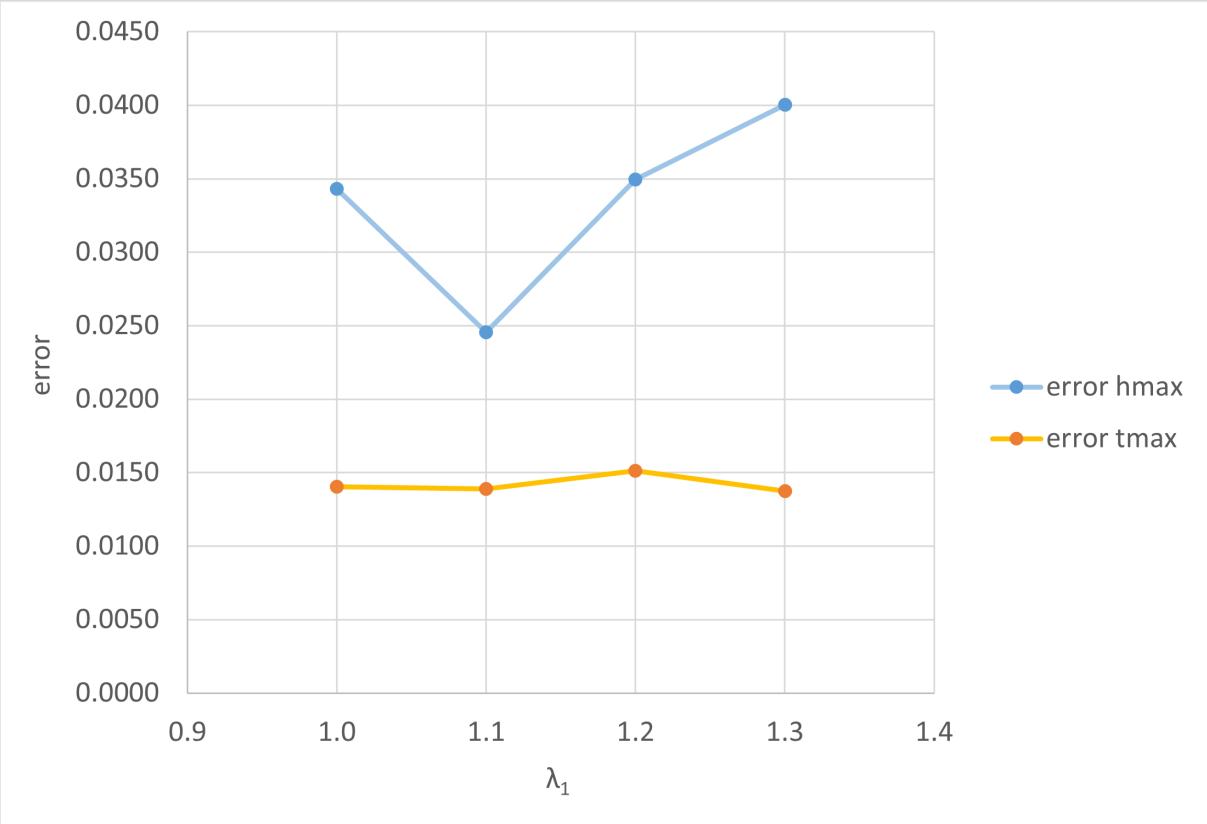



Figura 10: Evolución de los errores al variar $\lambda_1$.

En base a la tendencia observada en la gráfica anterior, el valor para el cual se alcanzan los mejores errores es para $\lambda_1 = 1{,}1$. Aumentar más este valor resulta contraproducente en el entrenamiento de la red neuronal, pero pasar de $\lambda_1 = 1$ a $\lambda_1 = 1{,}1$ sí consigue el efecto esperado de mejorar el error en $h_{max}$. Además, esto se consigue sin perjudicar el error en $t_{max}$ que se mantiene estable.

### 3.2.5. Estudio del batch size

Este es uno de los parámetros más importantes durante el entrenamiento. Una mala elección podría empeorar significativamente los resultados obtenidos. Hasta ahora, habíamos usado por defecto un valor de *batch size* del 100 % de los datos de entrenamiento.

Se han probado cinco valores de la fracción de datos de entrenamiento que elegimos como *batch size*, descendiendo desde un *batch size* del 100 % de los datos de entrenamiento empleados hasta un 20 %, a estos porcentajes los voy a nombrar como *bs_percent*.

Los resultados obtenidos para los errores en las variables de salida en función del *batch size* escogido (figura 11) son:

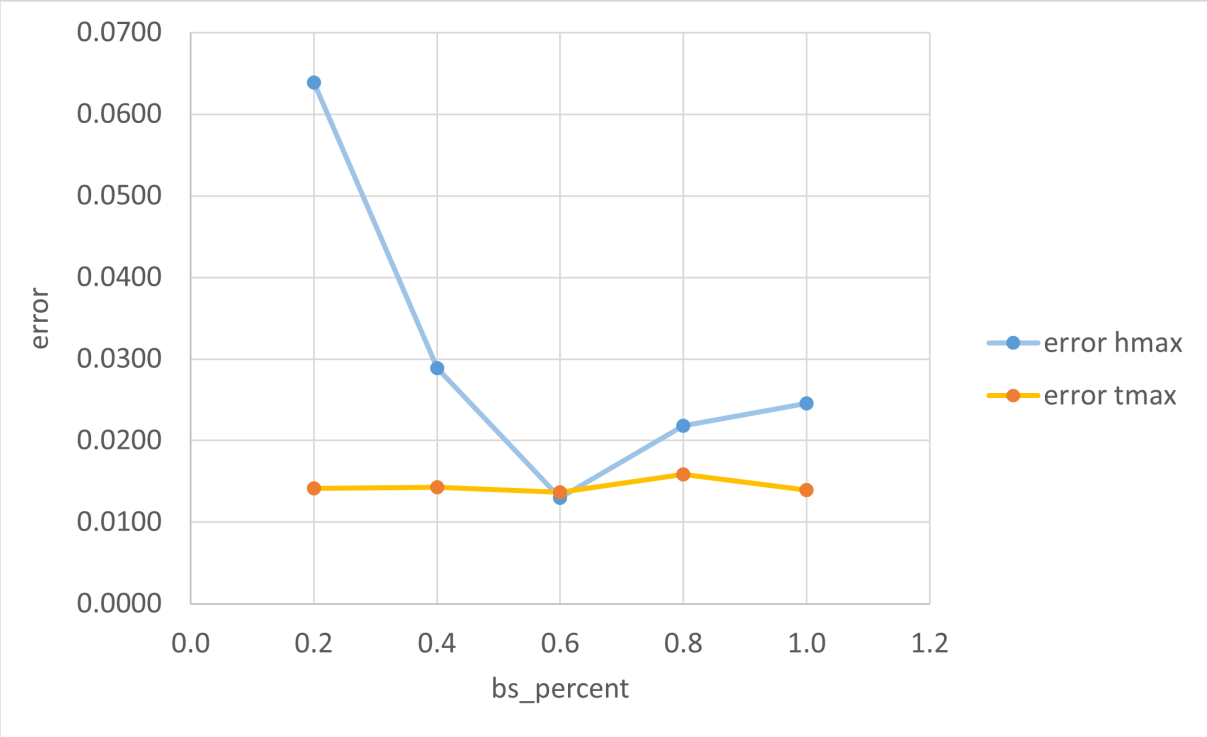


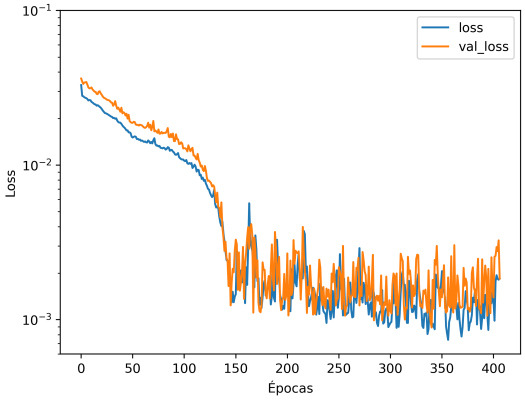

Figura 11: Errores obtenidos para $h_{max}$ y $t_{max}$ para los distintos *batch size*.

El error en $t_{max}$ es bastante indiferente a este parámetro, sin embargo, claramente los mejores resultados para $h_{max}$ se obtienen usando un *batch size* del 60 % de los datos de entrenamiento.

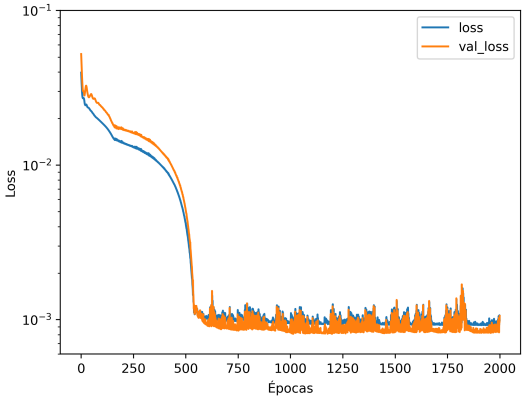
Este parámetro condiciona de manera notable el entrenamiento de la red neuronal, cambiando la función de pérdida (*loss*) y la función de pérdida de los datos de validación (*val_loss*), tanto la forma de la convergencia como el valor en que se estabiliza y la forma en que lo hace. Esto lo podemos ver en la figura 12. En ella apreciamos la diferencia en la forma de converger y el valor y la forma de la estabilización, con el eje vertical en escala logarítmica, entre el caso (a) y el caso (b).

Para el valor más pequeño de *batch size* observamos (figura 12) que se crean más oscilaciones en la función de pérdida, como se podía preveer. Esto significa que, al coger los datos de entrenamiento en mayor cantidad de lotes más pequeños para hacer la *backpropagation*, se está generando "ruido" o "confusión" y no es capaz de encontrar el mínimo absoluto de la función de pérdida.

La comparación de la evolución de la función de pérdida en el entrenamiento usando *bs_percent* = 0,2 y en el entrenamiento usando *bs_percent* = 1 la podemos ver en la figura 12:




(a) Evolución de la función de pérdida en escala logarítmica a lo largo del entrenamiento para un batch_size del 20 % de los datos.




(b) Evolución de la función de pérdida en escala logarítmica a lo largo del entrenamiento para un batch_size del 100 % de los datos.

Figura 12: Evolución de la función de pérdida (y la función de pérdida de validación) en escala logarítmica a lo largo del entrenamiento. Nota: La escala del eje $x$ no puede ser la misma en ambas gráficas porque para *bs_percent* = 1 la función de pérdida tarda más en converger.

### 3.2.6. Estudio de las funciones de activación

Hasta ahora hemos usado la función de activación *relu*, de las más recomendadas para los problemas de regresión, sin embargo, vamos a probar algunas otras, como se resume en la tabla 5:

| Función | loss | val_loss | error $h_{max}$ | error $t_{max}$ |
|---|---|---|---|---|
| relu | 0.0006 | 0.0008 | 0.0130 | 0.0137 |
| selu | 0.0009 | 0.0014 | 0.0414 | 0.0121 |
| leaky_relu | 0.0013 | 0.0026 | 0.0643 | 0.0136 |
| silu | 0.0010 | 0.0014 | 0.0419 | 0.0144 |

Tabla 5: Funciones de pérdida y errores obtenidos para las distintas funciones de activación.

Basado en esto, claramente elegimos como función de activación *relu*, ecuación (3).

### 3.2.7. Predicciones finales

Una vez optimizados todos los parámetros, presentamos la precisión de la predicción para las 4 gráficas de dependencias entre variables de entrada y variables de salida:

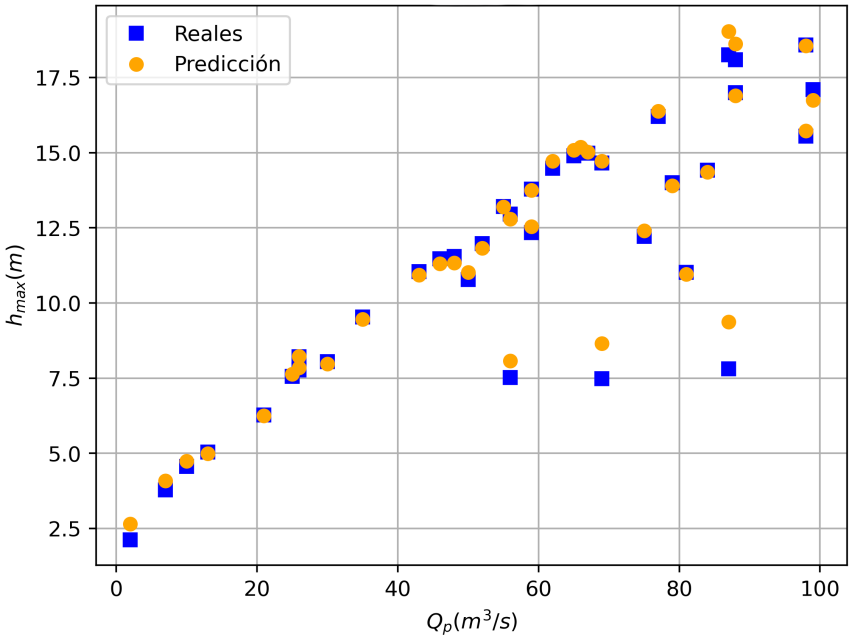



Figura 13: Predicción de $h_{max}$ en función de $Q_p$.

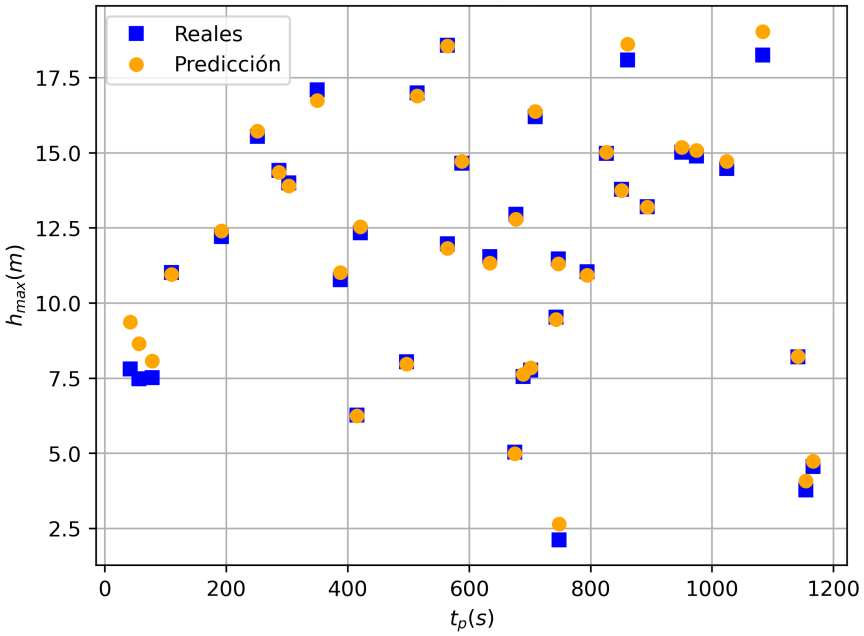



Figura 14: Predicción de $h_{max}$ en función de $t_p$.

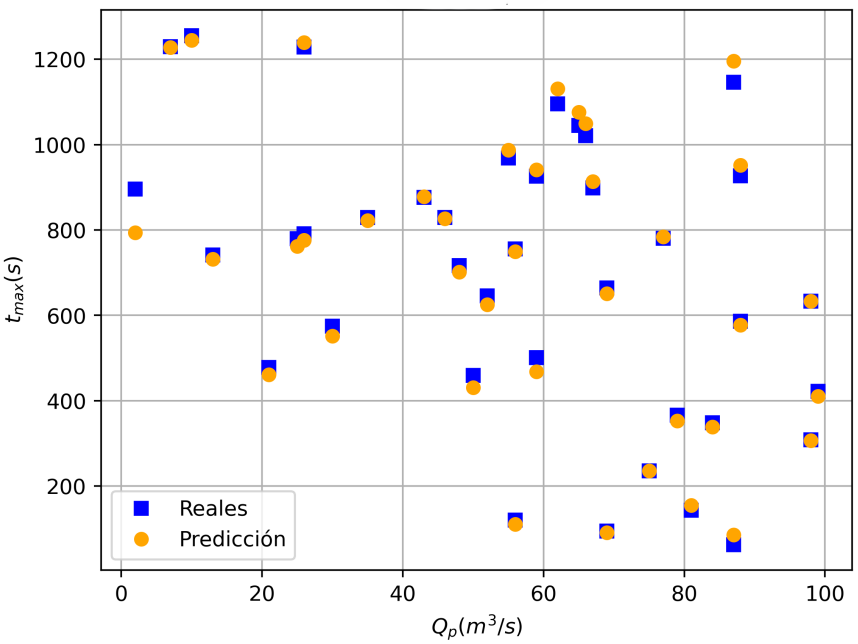



Figura 15: Predicción de $t_{max}$ en función de $Q_p$.

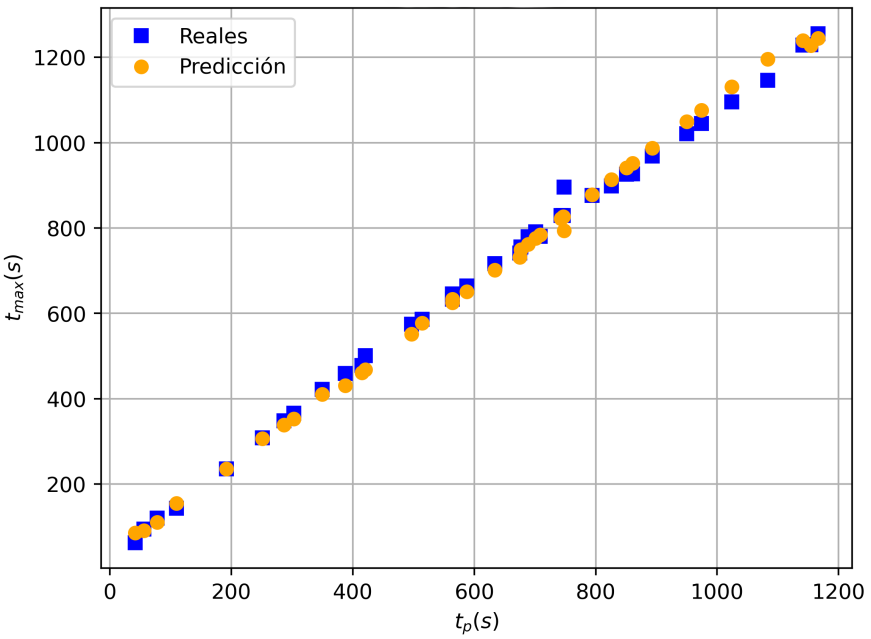



Figura 16: Predicción de $t_{max}$ en función de $t_p$.

En estas 4 gráficas (figuras 13, 14, 15 y 16) observamos la diferencia entre la predicción de la red neuronal (círculos naranjas) y los datos reales (cuadrados azules). Estas son $h_{max}$ en función de $Q_p$, (13); $h_{max}$ en función de $t_p$, (14); $t_{max}$ en función de $Q_p$, (15); $t_{max}$ en función de $t_p$, (16). Los errores correspondientes a las variables de salida $h_{max}$ y $t_{max}$ son los presentados en la tabla 5 para la función de activación *relu*.

### 3.2.8. Correlación entre la función de pérdida y el error en la predicción

Estudiamos si podemos usar la función de pérdida y la función de pérdida de validación, que son métricas que miden la calidad del entrenamiento de la red neuronal, como métricas para estimar el error que comete la red neuronal en una predicción.

Para una selección de predicciones con errores menores del 5 % se obtienen las correlaciones mostradas en las figuras 17 y 18:

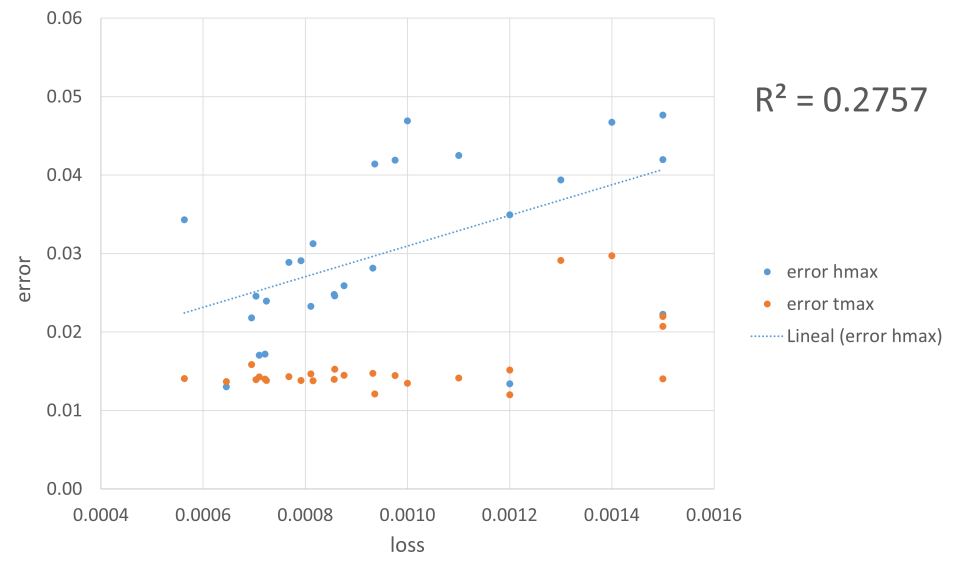



Figura 17: Representación del error de ambas variables de salida frente al valor de la función de pérdida de la red al final de ese entrenamiento.

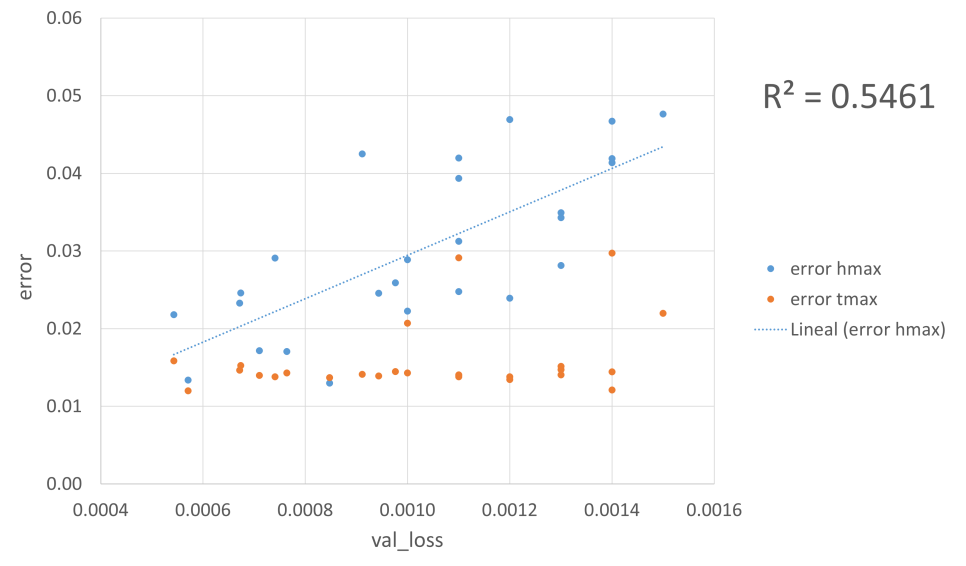



Figura 18: Representación del error de ambas variables de salida frente al valor de la función de pérdida de validación de la red al final de ese entrenamiento.

Por una parte, vemos que los errores de $t_{max}$ son bajos independientemente de la función de pérdida, sin embargo, $h_{max}$ sí es más sensible a estos. Por otra parte, observamos que la correlación de *val_loss* con los errores en la predicción es aproximadamente del doble que la correlación de *loss* con los errores en la predicción, es decir, *val_loss* es una métrica más eficaz que *loss* para medir la calidad de un entrenamiento. Esto tiene mucho sentido si recordamos que la función de pérdida de validación usa una fracción de los datos de entrenamiento únicamente para calcular la métrica *val_loss* de cada época. Esto significa que los datos de validación son desconocidos a la red neuronal, es decir, que no se entrena con ellos. Por lo tanto, ver la calidad con la que es capaz de ajustar estos datos es un problema más exigente que ver la calidad con la que se ajustan los datos con los que se ha entrenado. La red neuronal puede ajustar bien los datos de entrenamiento teniendo *overfitting*, sin embargo, es ajustar unos datos desconocidos lo que exige capacidad para generalizar y, por lo tanto, supone un reto mayor.

También tiene sentido que la correlación no sea de la unidad, pues hacer el cálculo de *val_loss* con el conjunto de datos de validación no garantiza que la red neuronal vaya a ser capaz de generalizar con la misma precisión para cualquier dato que se le pida predecir, ya que los datos de validación son limitados, en nuestro caso un 20 % de los datos de entrenamiento totales. Es esperable que si se le porporcionasen más datos de entrenamiento (y, por tanto, más datos de validación) la correlación entre *val_loss* y los errores en la predicción aumentase.

### 3.2.9. Otras pruebas

Hasta ahora, habíamos usado las mediciones de la sonda 1 para hacer el ajuste de los parámetros, así que, para terminar el trabajo, vamos a hacer la obtención de resultados para los datos de entrenamiento, recogidos de las otras sondas de medición, para la configuración de red con los parámetros ajustados (tabla 6):

| Sonda | loss | val_loss | error $h_{max}$ | error $t_{max}$ |
|---|---|---|---|---|
| 1 | 0.0006 | 0.0008 | 0.0130 | 0.0137 |
| 2 | 0.0014 | 0.0011 | 0.0206 | 0.0198 |
| 3 | 0.0014 | 0.0013 | 0.0182 | 0.0184 |

Tabla 6: Comparación de errores en $h_{max}$ y $t_{max}$ y de la función de pérdida en las 3 sondas.

Comprobamos que el error para los otros datos de medición es del mismo orden que para la sonda 1. Tiene sentido que los errores para la sonda 2 y 3 salgan ligeramente más altos porque el calado máximo (y, por tanto, el tiempo máximo) tienen variaciones más complejas sobre el obstáculo y después del obstáculo.

# 4. Conclusiones

Una vez presentados los resultados, nos disponemos a analizar las conclusiones y aprendizajes que podemos sacar de este proceso:

- En el caso estacionario, sorprende la facilidad con la que se ha obtenido un resultado aceptable. Es un caso en el que los parámetros no estaban optimizados. Uno de los problemas que se podían esperar era que las redes neuronales necesitasen de cantidades muy grandes de datos para poder entrenarse correctamente, sin embargo, al obtener estos resultados se demostró que con pocos datos ya pueden empezar a funcionar relativamente bien.
- En el caso transitorio:
    1. El problema es más complejo, pero la red sigue una dinámica parecida. Desde las primeras pruebas, se ha obtenido un error del 5.06 %. Esto se puede considerar un gran resultado teniendo en cuenta que todavía no se había hecho el ajuste fino de parámetros y que la cantidad de datos usada para el entrenamiento de esta parte era de 38, es decir, bastante pequeño comparado con las cantidades que se suelen usar en redes neuronales (idealmente del orden de millares).
    2. Efectivamente, hemos observado el problema del *overfitting*, explicado en la introducción, al probar los mayores valores de número de capas ocultas y de número de neuronas por capa oculta. Esto demuestra la importancia de no excederse en estos parámetros, puesto que poner valores mayores de los necesarios empeora los resultados obtenidos.
    3. Las convergencias de las funciones de pérdida con el paso de las épocas han sido las esperadas. Aunque la forma exacta de esta convergencia puede variar de un entrenamiento a otro, es habitual que tengan 2 partes: un rápido decrecimiento inicial seguido de uno más lento que termina por estabilizarse cuando llega a la convergencia.
    4. En cuanto al tiempo que tarda esta convergencia no he podido dar datos muy precisos, ya que el entorno de programación usado, en este caso Google Colaboratory, a veces se quedaba parado repentinamente falseando todas las medidas. Puedo dar un intervalo de tiempo usado para el entrenamiento de entre 45 y 90 segundos.
    5. El error en las predicciones finales es aproximadamente 3.9 veces menor que en la prueba inicial, sin ningún parámetro optimizado y 1 capa oculta. Esto demuestra que la optimización de hiperparámetros es fundamental para sacar el máximo potencial de las redes neuronales.

- Se han encontrado los siguientes valores óptimos de los hiperparámetros:

  1. Número de capas ocultas: 3
  2. Número de neuronas por capa oculta: 7
  3. Optimizador: Adam
  4. *Learning rate*: 0.001
  5. $\lambda_1$: 1.1
  6. $\lambda_2$: 1
  7. *Batch size*: 60 % de los datos de entrenamiento
  8. Función de activación de las neuronas de las capas ocultas: *relu*

  El número de épocas de entrenamiento se ha elegido según era necesario para la convergencia de la función de pérdida. Además, se ha implementado una función de *Keras* llamada *Early Stopping*, la cual finaliza el entrenamiento después de un cierto número de épocas sin que se haya obtenido un valor de la función de pérdida menor que el mínimo absoluto encontrado previamente.

- El valor de la función de pérdida de validación (*val_loss*) es una mejor métrica que el valor de la función de pérdida (*loss*) para evaluar la calidad de un entrenamiento.

- Los resultados indican que las redes neuronales pueden ser una herramienta eficaz para la predicción de inundaciones y el desarrollo de sistemas de alerta temprana, incluso con escasez de datos de entrenamiento.

# 5. Referencias


[1] J. Burguete P. Brufau M. Morales-Hernández, P. García-Navarro. A conservative strategy to couple 1D and 2D models for shallow water flow simulation. *Computers Fluids, nº 81, p. 26-44*, 2013.

[2] A. Lacasta P. Brufau y P. García-Navarro I. Echeverribar, M. Morales-Hernández. Simulación numérica con RiverFlow2D de posibles soluciones de mitigación de avenidas en el tramo medio del río Ebro. *Ingeniería del Agua, 21.1*, 2017.

[3] P.D. Bates M.S. Horritt. Evaluation of 1D and 2D numerical models for predicting river flood inundation. *Journal of hydrology, 268, 89–99*, 2002.

[4] P. Jimeno Sáez. Simulación de procesos hidrológicos utilizando técnicas de machine learning y modelos hidrológicos. *Universidad Católica de Murcia*, 2018.

[5] P. Vallés Oliván. Estudio de soluciones para mitigar inundaciones en el río Ebro mediante simulación numérica. *Universidad de Zaragoza*, 2021.

[6] W. S. McCulloch W. Pitts. A logical calculus of the ideas immanent in neurons activity. *Bulletin of Mathematical Biophysics*, 1943.

[7] D. E. Rumelhart R. J. Williams, G. E. Hinton. Learning representations by back-propagating errors. *Nature Vol. 323*, 1986.

[8] S. Haykin. Neural networks: A Comprehensive Foundation. *Pearson Education*, 1998.

[9] S. G. Rodrigo L. Medrano Navarro, L. Martin Moreno. Solving differential equations with Deep Learning: a beginner's guide. *Universidad de Zaragoza*, 2023.

[10] E. Olyaie et al. A comparison of various artificial intelligence approaches performance for estimating suspended sediment load of river systems: a case study in United States. *Springer Link*, 2015.

[11] H. T. Hsu J. T. Shiau. Suitability of ANN-Based daily streamflow extension models: a case study of Gaoping River Basin, Taiwan. *Springer Link*, 2016.

[12] M. Villota Miranda. Teorema de aproximación universal: prueba constructiva basada en conjuntos semisimpliciales. *Universidad de la Rioja*, 2020.