E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

# Predicción de temperatura de zonas de Navarra mediante Redes Neuronales Recurrentes y modificaciones de las mismas.

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alexandru Iulian Gavrila

Director: Mikel Ferrero Jaurrieta

Pamplona, 08 / Septiembre / 2022

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

# Resumen

El objetivo de este trabajo es la predicción de la temperatura a lo largo del tiempo tomada en diferentes puntos de la geografía navarra.

Para ello, hemos utilizado diferentes modelos basados en Redes Neuronales Recurrentes e incluso modificaciones de alguna de ellas. Lo primero que hemos hecho es descargar los datos que necesitamos para trabajar, posteriormente creamos los modelos que utilizaremos y nos quedamos con los que obtengamos mejores resultados. Con estos, hemos ido aumentando el periodo de tiempo a predecir hasta encontrar un periodo razonable que no falle demasiado en sus predicciones. Por último, hemos probado a modificar un tipo de red para comprobar si se pueden mejorar los resultados obtenidos anteriormente.

A lo largo del proyecto se muestran tablas de resultados y gráficas de las series temporales con el fin de poder comparar las diferentes redes y obtener las que mejores predicciones realicen.

# Palabras Clave

- Series temporales
- Redes Neuronales Recurrentes
- LSTM
- Predicción
- Temperatura

# Índice

# 1. Introducción

La temperatura, científicamente hablando, es la propiedad física que mide la cantidad de calor que tiene la materia *[1]*. Para el ser humano en general, la temperatura simplemente es un número que le dice si hay que abrigarse más o menos. Además, el valor de este dato se mide en diferentes escalas dependiendo del ámbito y del país en el que nos encontremos.

En España, la escala que usamos es la escala Celsius. En esta escala el agua se congela a los 0°C y entra en ebullición a los 100°C. Esto resulta importante para tener en cuenta ya que las mediciones son muy diferentes en una escala u otra (30°C no son lo mismo que 30°F).

La predicción de la temperatura forma parte de nuestro día a día. Todas las personas miran la predicción del tiempo antes de salir de casa o de realizar un viaje. Realizar estas predicciones es posible gracias a que, entre otras cosas, teniendo en cuenta la tendencia de las temperaturas de todos los días hasta hoy, mediante una serie temporal, podemos predecir la de mañana.

Una serie temporal es una sucesión de datos obtenidos en momentos concretos que se guardan ordenados cronológicamente. Cuando los datos de una serie varían de manera cíclica se puede decir que esta es una serie estacionaria (Figura 1) *[2]*.

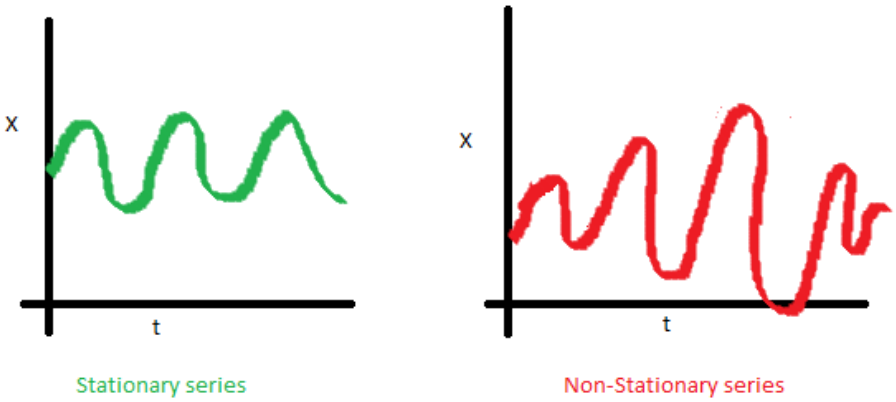


*Figura 1: Diferencia entre las series estacionarias y las no-estacionarias.*

La temperatura es un dato que no produce series completamente estacionarias, ya que varía cíclicamente a lo largo de un día completo, pero se producen alteraciones. Esto quiere decir que no todos los días va a hacer la misma temperatura a la misma hora, ya que existen agentes externos que pueden afectar a la temperatura. Las series temporales, si se crean mediante datos aleatorios, están compuestas por varias componentes distintas que hacen que los ciclos no sean exactamente iguales. Estas características de la temperatura es la que hace que pueda ser predecible.

La temperatura además no es igual en dos sitios, si estos sitios están cerca pueden estar en la misma zona climática y por tanto tener temperaturas parecidas o si están en zonas distintas o a altitudes distintas (Figura 2). En este proyecto trabajaremos con series de temperaturas de distintos puntos de la geografía navarra.

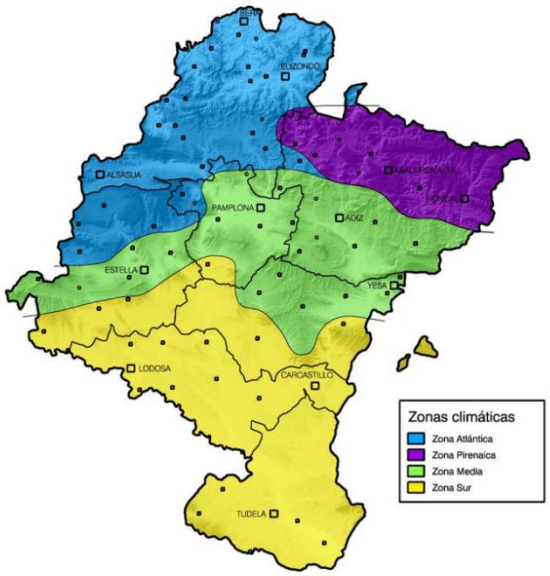


*Figura 2: Mapa de climas de navarra.*

En Navarra hay muchas estaciones meteorológicas distribuidas por toda la comunidad, estas toman otros datos a parte de la temperatura como por ejemplo la velocidad del viento, la presión atmosférica, etc (Figura 3).

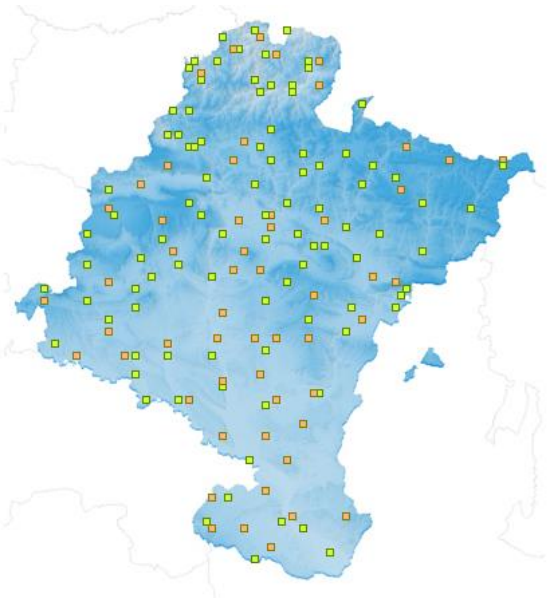

*Figura 3: Mapa de estaciones meteorológicas de Navarra.*

Para realizar estas predicciones, utilizaremos herramientas del aprendizaje automático y utilizaremos técnicas de clasificación basadas en redes neuronales.

Las redes neuronales son un modelo de aprendizaje automático creado en 1943 que trabajan de forma similar a como lo hacen las neuronas de un cerebro humano. Estas se basan en crear una combinación de parámetros que mediante unos datos de entrada realizan predicciones. En nuestro caso, trabajaremos con unas redes neuronales especiales, las redes neuronales recurrentes, puesto que son necesarias para trabajar con modelos de datos secuenciales ya que estas, a diferencia de las redes neuronales clásicas, aprovechan el paso del tiempo como una herramienta más para predecir datos.

# 2. Aprendizaje automático y Deep learning: Redes Neuronales

En esta sección vamos a explicar la base teórica fundamentar para poder realizar el trabajo de predicción de temperaturas. Se explican que son el aprendizaje automático y el aprendizaje profundo y tras ello las redes neuronales desde el perceptrón hasta las redes más profundas.

El aprendizaje automático es una rama de la inteligencia artificial que se basa en desarrollar técnicas que permitan que la máquina aprenda, es decir, que su desempeño mejore con la experiencia y mediante el uso de datos *[3]*.

A partir de unos datos, se crea un modelo que resuelve una tarea con la menor interacción de un ser humano. Hay dos tipos de aprendizaje en función de los datos que usan para entrenar el modelo:

- Aprendizaje supervisado: utiliza datos de entrada y salida para entrenar el modelo
- Aprendizaje no supervisado: utiliza únicamente datos de entrada.

El Deep learning o aprendizaje profundo, es un tipo de aprendizaje automático que trata de modelar abstracciones de alto nivel de datos usando modelos que admiten transformaciones múltiples, no lineales e iterativas de datos expresados en forma de matrices *[4]*. Los datos no se procesan de una manera predefinida, sino que se utilizan múltiples capas de procesamiento no lineal. Principalmente se utilizan redes neuronales profundas.

Las redes neuronales son un modelo de aprendizaje automático que se inspiran en el funcionamiento del cerebro humano, reproduciendo su comportamiento de manera informática *[5]*. Para ello se crea el Perceptrón, que imita el funcionamiento de una neurona.

## 2.1 El perceptrón

La neurona artificial o perceptrón se modela imitando el funcionamiento de una neurona del cerebro. Estas tienen sus ramificaciones que pueden ser de entrada o de salida y un núcleo al que comúnmente se le llama nodo que procesa la información que le llega por las ramificaciones de entrada y las trasmite por las ramificaciones de salida.

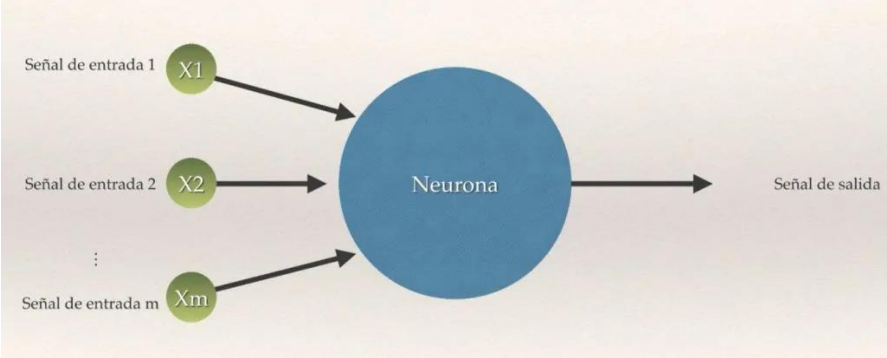


*Figura 4: Perceptrón simple.*

Como podemos observar en la ilustración 3, a nuestra neurona le llegan unos datos de entrada, los cuales los procesa para posteriormente obtener una salida. Para poder llevar esto a cabo, toda ramificación de la neurona lleva asignado un peso. En el nodo se procesan los datos de entrada y los pesos, por ejemplo, mediante una combinación lineal de estos.

$$w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m$$

Al resultado obtenido anteriormente se le puede añadir un valor de sesgo, comúnmente conocido como bias. Para obtener la salida, se le aplica una función de activación:

$$y = \phi(w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m + b)$$

Sean a, b dos números reales a < b, una función de activación es una función $\phi : \mathbb{R} \to [a, b]$ que comúnmente cumple las siguientes propiedades:

1. $\phi$ es continua.
2. Es una función no lineal.

3. Esta acotada, es decir:
   1) $\lim_{x \to -\infty} \phi(x) = a$
   2) $\lim_{x \to +\infty} \phi(x) = b$

Una función de activación podría ser, por ejemplo, una función sigmoide:

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

## 2.2 Perceptrón Multicapa

El perceptrón imita la función de una neurona, pero como sabemos, el cerebro está compuesto por millones de estas, por ello, se crea el perceptrón multicapa. El perceptrón multicapa es una red neuronal, que como su nombre indica, está formado por varias capas. De esta manera consigue la capacidad para resolver problemas que no son linealmente separables. Estos se componen de tres capas principales, la capa de entrada, cuyas neuronas no tienen ramificaciones en la entrada puesto que cada una de ellas coge directamente el dato que le proporcionamos, la o las capas ocultas, que se encargan de transmitir la información, u por ultimo la capa de salida, cuyas neuronas no tienen ramificaciones de salida si no que dan la información de salida de la red neuronal.

En caso de que solo tengamos una capa oculta, la llamaremos red neuronal simple y en caso de que tenga varias capas ocultas la llamaremos red neuronal profunda.

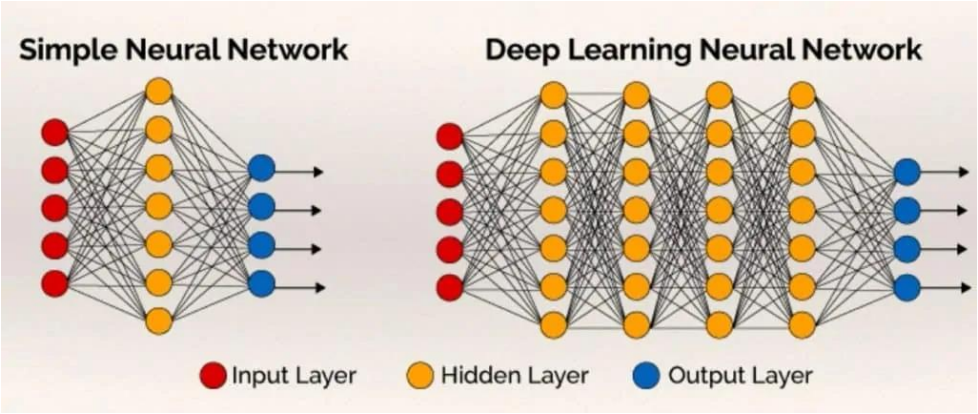


*Figura 5: Perceptrón Multicapa.*

## 2.3 Entrenamiento de la Red neuronal

Para entrenar una red neuronal, comenzamos con pesos aleatorios entre cada capa e iremos realizando las operaciones de cada perceptrón. Todo esto se puede realizar mediante operaciones matriciales, metiendo todos los datos de entrada en una matriz de entrada y lo mismo con los pesos y el bias. Nos quedaría esta operación en cada capa:

$$y_i = \phi(W_i X_{i-1} + b_i)$$

Una vez obtenida la salida, veremos que generalmente no tienen nada que ver con los valores reales de salida que deberíamos obtener. Para acercarnos más utilizamos primero una función de coste para calcular el error obtenido, esta puede ser, por ejemplo, el error cuadrático medio. Con este error obtenido, se aplica un descenso por gradiente, que es un algoritmo que sirve para estimar numéricamente donde una función genera sus valores más bajos. Esto se realiza mediante una propagación hacia atrás de los errores obtenidos de las salidas para modificar los pesos y los bias de todas las capas ocultas. A estos se les añade una parte del error que se calcula multiplicándolo por un valor al que se le conoce como learning rate.

Realizando lo anterior de manera iterativa, cada vez obtendremos un error menor, hasta obtener una salida lo suficientemente válida.

## 2.4 Predicción de datos

Para obtener predicciones, solo tenemos que darle a la red neuronal entrenada los datos de entrada sobre los cuales queremos obtener una salida. Las predicciones pueden obtenerse de muchas maneras, con un solo valor de entrada podemos obtener uno o varios valores de salida, y de la misma manera, mediante varios valores de entrada podemos obtener uno o varios valores de salida.

# 3. Redes Neuronales Recurrentes

Una vez explicadas las redes neuronales en la sección anterior, aquí se definen las redes neuronales recurrentes y sus diferentes versiones, LSTM y GRU.

Las series temporales son datos secuenciales, es decir, tienen una característica especial en comparación con otros tipos de datos, cada dato depende de los anteriores. Las Redes Neuronales Recurrentes (RNN, por sus siglas en inglés) utilizan esta dependencia para obtener mejores resultados en la predicción de este tipo de datos. Ejemplos de datos secuenciales podrían ser los vídeos, que tienen un fotograma en cada instante de tiempo y cada fotograma nuevo tiene relación con el anterior, o, los valores de la temperatura en un mismo punto a lo largo del tiempo.

Las RNN se basan en las Redes neuronales clásicas, con la diferencia de que la red no solo tiene los estados de entrada y de salida, si no que añade uno nuevo, el estado recurrente que calcula todos los estados entre la entrada y la salida *[6]*.

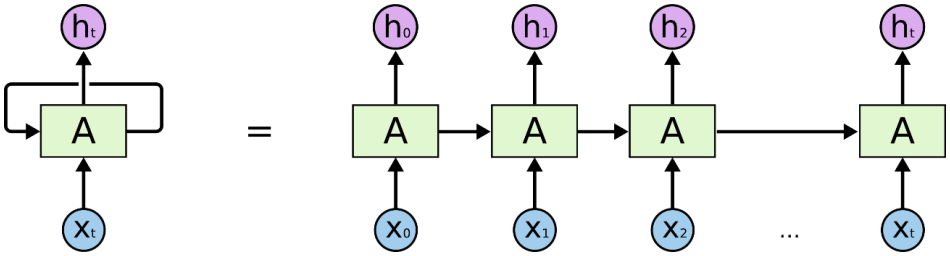


*Figura 6: Representación de una RNN y su vista desplegada.*

Como podemos observar en la figura 6, en cada instante de tiempo entran unos datos y obtenemos las salidas. En el momento del entrenamiento, el estado interno de las capas ocultas se basa no solo en su entrada si no también en las capas ocultas del instante de tiempo anterior. Al realizar las predicciones, los valores obtenidos son los de alguna de las últimas salidas dependiendo de lo que queramos obtener (Figura 7).

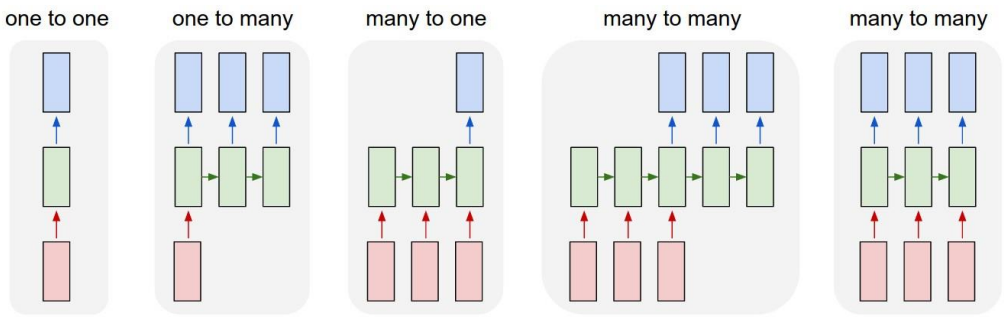


*Figura 7: Posibles situaciones de entrada/salida de las RNN.*

En cada instante de tiempo, a diferencia de las redes neuronales clásicas, los resultados no se obtienen únicamente de los datos, los pesos y el bias de cada capa, si no que se utilizan también los datos en las capas ocultas del instante anterior y unos pesos entre las capas ocultas del instante anterior y las actuales. Por tanto, los datos pasarían a calcularse de la siguiente manera:

$$y_t = W_y h_t = W_y f(W_x x_t + W_h h_{t-1} + b_t)$$

Siendo $y_t$ la salida en el instante t, $W_y$ la matriz de pesos entre la capa oculta y la salida, $h_t$ los valores de la capa oculta (su forma de calcularse es la que cambia entre las redes clásicas y las RNN), $W_x$ los pesos entre la entrada y la capa oculta, $x_t$ los datos de entrada en el instante t, $W_h$ los pesos entre la capa oculta del instante anterior y el actual y $h_{t-1}$ los datos de la capa oculta del instante anterior.

Este tipo de redes tienen un problema, la dependencia de datos a largo plazo, ya que por la forma en la que se calculan los nuevos valores, al aumentar el tamaño de las series, se produce un gradiente desvaneciente y los elementos antiguos tendrán cada vez valores más pequeños y por lo tanto insignificantes (Figura 7). Para evitar este problema, se crearon otros tipos de RNN, como las LSTM, que son más avanzadas y tienen en cuenta que valores son importantes y cuales no de guardar en el tiempo.

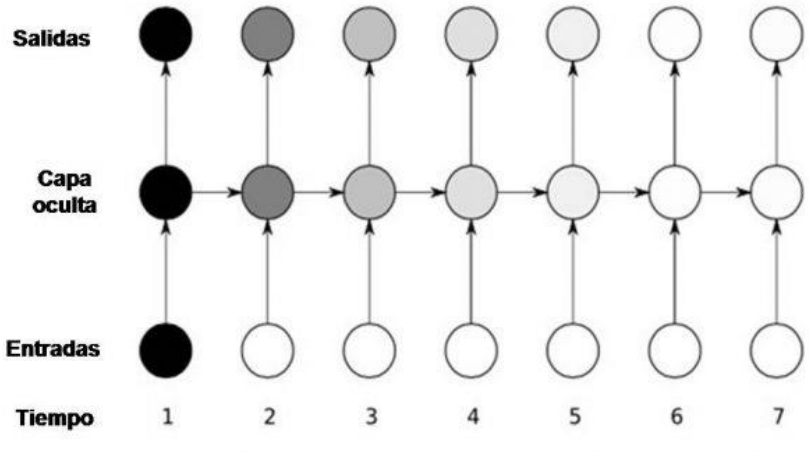


*Figura 8: Funcionamiento del gradiente desvaneciente.*

## 3.1 LSTM

Para poder trabajar con entradas secuenciales largas, que con las RNN tenían problemas para entrenar, en 1997 se introducen las LSTM (Long-Short Term Memory) *[7]*. Estas, a diferencia de las RNN clásicas, guardan los datos tanto a corto como a largo plazo que mas importantes resulten. Para ello, internamente son mucho más complejas que las otras ya que están formadas por varias operaciones internas. Los elementos de los que se componen sus celdas son los siguientes:

- Memoria a largo plazo
- Memoria a corto plazo
- Forgate Gate
- Input Gate
- Output Gate

Esto sirve para evitar el problema de gradiente desvaneciente que tenían las RNN.

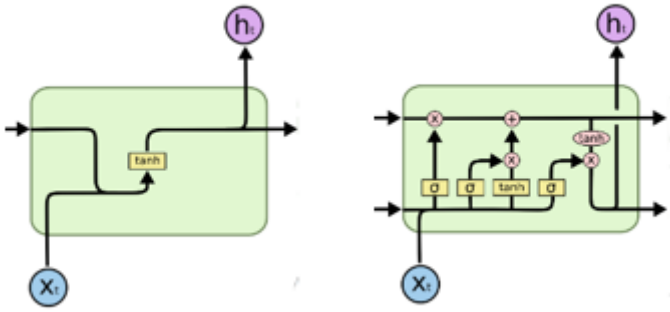


*Figura 9: Diferencias internas entre las neuronas de una RNN clásica y una LSTM.*

Como se puede observar en la figura 9, a diferencia de las RNN, las celdas de las LSTM están compuestas por dos salidas recurrentes. La que hasta ahora llamábamos $h_t$ es la que lleva la memoria a corto plazo, es decir, los datos del instante anterior, mientras que la nueva, que llamamos $c_t$, lleva consigo memoria a largo plazo, es decir, datos más antiguos *[8]*.

Esta celda realiza tres operaciones más que las RNN. Llamaremos puertas al momento en el que se realiza cada operación en función de lo que se realice en cada una de ellas.

La primera puerta de la LSTM es la Forgate Gate (Figura 10), la cual decide qué información no queremos que se tenga en consideración en nuestro modelo. La puerta aplica una función sigmoide:

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

El resultado de esta función siempre es un valor entre 0 y 1, significando 0 que de ese dato no entra nada y 1 que ese dato entra en su totalidad al modelo. El resultado que calcula esta puerta se consigue de esta manera:

$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f)$$

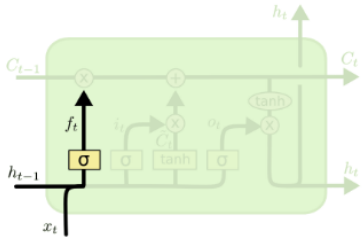


*Figura 10: Forget Gate de la LSTM.*

La segunda puerta es la input gate, la cual decide qué nueva información si queremos que entre al modelo. El resultado de esta puerta se calcula de manera similar al de la puerta anterior:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

Posteriormente, a los nuevos datos antes de utilizarlos se les aplica la tangente hiperbólica:

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

Esto sirve para acotar los valores en (-1, 1), manteniendo los valores negativos y así evitamos también el gradiente explosivo.

$$\widetilde{C_t} = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_C)$$

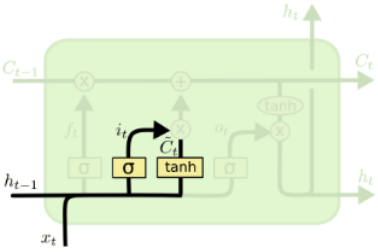


*Figura 11: Input Gate de la LSTM.*

Para obtener la nueva memoria a largo plazo, utilizamos la siguiente función:

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t}$$

Se decide que parte de los datos a largo plazo olvidar (mediante la forgate gate) y los datos nuevos que guardamos en la memoria (mediante la input gate). Una vez obtenidos se unifican para obtener la nueva memoria a largo plazo.

En este caso usamos el símbolo $*$ ya que no es una multiplicación matricial si no el producto de Hadamard, es decir, la multiplicación elemento a elemento de las matrices.

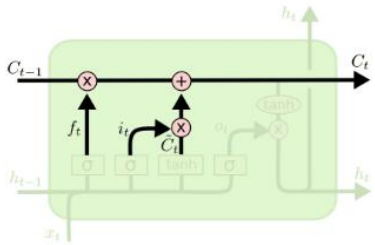


*Figura 12: Salida $C_t$ de la LSTM.*

Por último, tenemos que obtener la salida a través de la última puerta, la output gate, para ello aplicamos otra sigmoide para sacar solo los datos a corto plazo que la red decide que sigan en el modelo. Podemos observar que los datos de salida en este instante y los datos que vamos a usar como memoria a corto plazo son los mismos.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$y_t = h_t = o_t * \tanh(C_t)$$

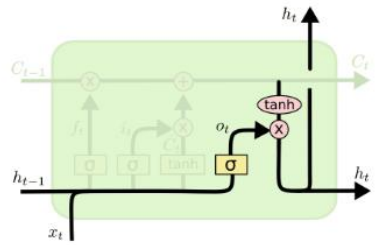


*Figura 13: Salida y memoria a corto plazo de las LSTM.*

## 3.2 GRU

Las GRU (Gated Recurrent Unit) son una modificación de las LSTM introducidas en 2014 *[9]*.

Este nuevo tipo de RNN son similares a las LSTM pero realizan menos operaciones y por tanto son más simples y más rápidas. Las GRU en lugar de tener tres puertas, tienen dos. Por tanto los elementos de cada celda son los siguientes:

- Memoria del estado oculto
- Update Gate
- Reset Gate

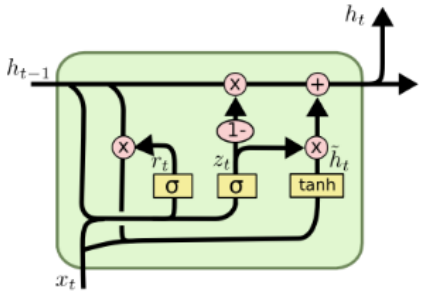


*Figura 14: Estructura interna de la celda de una GRU.*

Por una parte, se vuelven a unificar la memoria a largo plazo con la memoria a corto plazo y ahora tenemos solo dos puertas que actúan de modo similar a las que teníamos en la LSTM.

La primera puerta (Figura 14, elemento $z_t$), se llama la update gate. Esta puerta actúa de manera similar a las forget e input gate de la LSTM, es decir, decide que información mantener y cual olvidar.

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

La segunda puerta (Figura 14, elemento $r_t$), se llama reset gate. Esta puerta decide cuanta información antigua descartar.

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

Posteriormente, al igual que el la LSTM, a los nuevos valores se les aplica una tangente hiperbólica para evitar problemas de gradiente explosivo provocados por la función sigmoide. Esto se realiza después de eliminar los datos de la memoria que la reset gate ha decidido eliminar.

$$\widetilde{h_t} = tanh(Wx_t + W\, r_t h_{t-1} + b_h)$$

Por ultimo, obtenemos la salida final aplicando el producto de Hadamard de la siguiente manera:

$$y_t = h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t}$$

## 3.3 Fundamentos matemáticos

En esta subsección vamos a definir algunos fundamentos matemáticos que vamos a utilizar posteriormente para realizar nuestras modificaciones sobre las redes neuronales. Los concetos snecesarios son los siguientes:

- Función de agregación: Esta función sirve para realizar la fusión de varios datos en uno solo y viene especificada de la siguiente manera *[10]*:

  Sea $m$ un entero positivo, una función $M : [0,1]^m \to [0,1]$ es una función de agregación m-aria si satisface lo siguiente:

  - $M(0, \dots, 0) = 0 \; y \; M(1, \dots, 1) = 1$
  - Es no decreciente en cada variable, es decir, para todo $(x_1, \dots, x_m), (y_1, \dots, y_m) \in [0,1]^m$, $M(x_1, \dots, x_m) \leq M(y_1, \dots, y_m)$ si $x_1 \leq y_1, \dots, x_m \leq y_m$ .

- Medida difusa: una medida difusa definida sobre $\{1, \dots, m\}$ es una función $v : 2^{\{1,\dots,m\}} \to [0,1]$ tal que:

  - $v(\emptyset) = 0 \; y \; v(\{1, \dots, m\}) = 1$
  - $v(A) \leq v(B) \; para \; todo \; A \subseteq B \subseteq \{1, \dots, m\}$

  Una medida difusa se dice que es aditiva si $v(A \cup B) = v(A) + v(B)$ para todo $A, B \subseteq \{1, \dots, m\}$ tales que $A \cap B = \emptyset$.

- Integral de Choquet discreta: Esta integral, junto con la medida difusa, sirven para tener en cuenta la posible coalición entre datos. Se define de la siguiente forma *[11]*:

  La integral de Choquet discreta en $[0,1]$, con respecto a la medida difusa $v$ se define como una aplicación $Ch_v : [0,1]^m \to [0,1]$

  $$Ch_v(x) = \sum_{i=1}^{m} (x_{\sigma(i)} + x_{\sigma(i-1)}) v(A_{\sigma(i)})$$

  Donde $X = (x_1, \dots, x_m) \in [0,1]^m, v : 2^{\{1,\dots,m\}} \to [0,1]$ es una medida difusa en el conjunto $\{1, \dots, m\}$, $\sigma : \{1, \dots, m\} \to \{1, \dots, m\}$ es una permutación, con $x_{\sigma(1)} \leq \cdots \leq x_{\sigma(m)}$ con la convención $x_{\sigma(0)} = 0$ y $A_{\sigma(i)} := \{\sigma(i), \dots, \sigma(m)\}$ es el subconjunto de los índices correspondientes a los m-i+1 mayores elementos de x para todo $i \in \{1, \dots, n\}$.

- Función de Disimilitud Restringida (RDF): Esta función se utiliza para medir diferencias entre dos datos. Se define así *[12]*:

  Una función $\delta : [0,1]^2 \to [0,1]$ es considerada función de disimilitud restringida en $[0,1]$ si satisface para todo $x, y, z \in [0,1]$ lo siguiente:

  - Es simétrica: $\delta(x,y) = \delta(y,x)$
  - $\delta(x,y) = 1$ sí y sólo sí $\{x,y\} = \{0,1\}$
  - $\delta(x,y) = 0$ sí y sólo sí $x = y$
  - Si $x \leq y \leq z, entonces\ \delta(x,y) \leq \delta(x,z)$ y $\delta(y,z) \leq \delta(x,z)$

- d-integral: Todas las funciones anteriores se unifican de manera que se crea la d-integral. Esta se define de la siguiente manera *[13]*:

  Sea n un número entero y $m : 2^{\{1,\dots,n\}} \to [0,1]$ una medida difusa sobre el conjunto $\{1, \dots, n\}$. Sea $\delta : [0,1]^2 \to [0,1]$ una función de disimiltud restringida. Una d-integral de Choquet discreta n-aria en $[0,1]$ con respecto de $m$ y $\delta$ es definida como una función $C_m^{\delta} : [0,1]^n \to [0,n]$ tal que:

$$C_m^{\delta}(x_1, \dots, x_n) = \sum_{i=1}^{n} \delta(x_{\pi(i)}, x_{\pi(i-1)}) m(A_{\pi(i)})$$

  Donde $\pi$ es una permutación en $\{1, \dots, n\}$ tal que $x_{\pi(1)} \leq x_{\pi(2)} \leq \cdots \leq x_{\pi(n)}$ con la convención $x_{\pi(0)} = 0$ y $A_{\pi(i)} = \{\pi(i), \dots, \pi(n)\}$ para todo $i \in \{1, \dots, n\}$.

# 4. Planteamiento del problema

En esta sección se plantean los objetivos a conseguir en este proyecto.

El objetivo principal de este proyecto es conseguir la predicción de datos de temperatura más larga posible en el tiempo, intentando obtener el mínimo error. Para ello utilizaremos tanto RNN, como LSTM, como GRU. De todas ellas probaremos con diferente número de capas ocultas, diferente número de neuronas por capa y diferentes learning rates para obtener los mejores modelos para nuestros datos. La cantidad de valores a predecir se irá aumentando progresivamente, intentando conseguir predecir la temperatura que va a hacer durante el mayor número de horas posible. Como función de coste, para ver el error obtenido en las predicciones, usaremos el error cuadrático medio (MSE).

$$MSE(y, pred) = \frac{1}{n}\sum_{i=1}^{n}(y_i - pred_i)^2$$

Siendo $y_i$ los valores de la salida reales, $pred_i$ los valores de salida predichos y $n$ la cantidad de valores predichos, para $i \in \{1, \dots, n\}$.

También nos centraremos en encontrar mejoras a estas redes, principalmente a las LSTM modificando la forma de calcular las puertas. Para ello vamos a utilizar una función llamada función de disimilitud restringida (RDF, por sus siglas en inglés), y la vamos a aplicar de la siguiente manera:

Partimos de la función $f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$ de la forgate gate (para el resto de las puertas se aplica de la misma manera).

Hay que realizar un cambio en la función anterior por varios motivos:

- Hay que adaptar los datos para poder utilizar la d-integral. Primero teníamos una función $\sigma: (\mathbb{R}^n)^3 \to \mathbb{R}^n$ y nosotros necesitamos los tres elementos separados para aplicar la d-integral.
- Los elementos deben estar en el rango $[0, 1]$ para evitar la explosión del gradiente y por ello se aplica una función $g$ donde $g: \mathbb{R} \to [0, 1]$ elemento a elemento.

Tras aplicar los cambios la función nos quedaría así:

$$f_t = g(W_{xf}x_t) + g(W_{hf}h_{t-1}) + g(b_f)$$

En este punto, en vez de sumar las tres matrices, vamos a aplicar la d.integral elemento a elemento:

$$C_m^{\delta}(x_1, \dots, x_n) = \sum_{i=1}^{n} \delta(x_{\pi(i)}, x_{\pi(i-1)}) m(A_{\pi(i)})$$

Puesto que en nuestro caso solo tenemos tres elementos podemos darnos cuenta de que uno de ellos será el mínimo, otro la mediana y otro el máximo.

$$x_{\pi(1)} = x_{min}$$

$$x_{\pi(2)} = x_{med}$$

$$x_{\pi(3)} = x_{max}$$

En nuestro caso, la medida difusa que vamos a utilizar es la de la cardinalidad:

$$m(A) = \frac{|A|}{n}$$

donde $|A|$ es la cardinalidad, o número de elementos de A.

Si realizamos los cambios de nomenclatura y aplicamos el sumatorio utilizando la medida difusa de la cardinalidad, nos quedaría así:

$$C_m^{\delta}(x_{min}, x_{med}, x_{max}) = x_{min} + \frac{2}{3}\delta(x_{min}, x_{med}) + \frac{1}{3}\delta(x_{med}, x_{max})$$

Esto lo aplicamos a cada grupo de tres elementos en la misma posición de los vectores y así obtenemos los valores de la forgate gate, cuya formula queda así: $f_t = C_m^{\delta}(g(W_{xf}x_t), g(W_{hf}h_{t-1}), g(b_f))$. En este caso hemos utilizado $g$ como la función sigmoidea.

La distancia entre elementos que le aplicamos a la formula anterior se puede calcular de distintas maneras, como, por ejemplo:

- $\delta_1(x, y) = |x - y|$
- $\delta_2(x, y) = \sqrt{|x^2 - y^2|}$

# 5. Datos

En esta sección se muestran los datos con los que vamos a trabajar. Junto con los datos, os mostramos la manera de obtenerlos y todo el preprocesamiento realizado previo a la realización del proyecto.

En este proyecto hemos decidido usar datos de 4 estaciones meteorológicas localizadas en distintas localidades de la Comunidad Foral de Navarra. Estas estaciones son: El Perdón, Tafalla, Ujué y Doneztebe.

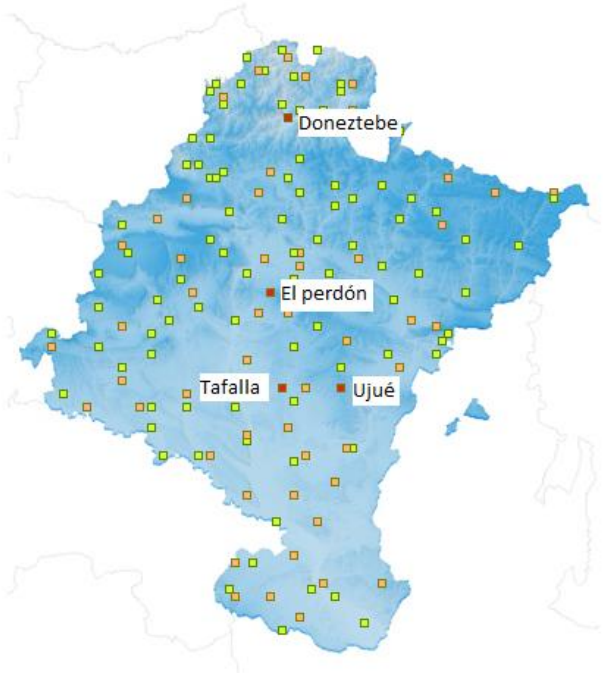


*Figura 15: Mapa de estaciones. En rojo las estaciones que usamos.*

Hemos elegido estas estaciones ya que están todas en diferentes zonas climáticas excepto El Perdón y Ujué, que están en la misma zona climática, pero están a diferentes altitudes.

## 5.1 Descarga de datos

Para descargar los datos hemos usado las herramientas BeautifulSoup y Requests para obtener los datos de la web http://meteo.navarra.es/. La herramienta beautifulSoup nos sirve para obtener los IDEstacion de cada estación a partir de la web. Este dato posteriormente lo utilizamos junto con la herramienta Requests.

La herramienta Requests sirve para descargar el contenido de una URL. Hemos observado que los datos de temperatura se pueden encontrar perfectamente ordenados en un enlace de este estilo:

http://meteo.navarra.es/download/estacion_datos.cfm?IDEstacion=28&p_10=1&fecha_desde=01%2F01%2F2001&fecha_hasta=31%2F12%2F2001&dl=csv

Si se observa adecuadamente se ven el IDEstacion, una fecha de inicio y otra de final. La web solo nos permite descargar datos de un lapso de un año, con lo cual tenemos que realizar una iteración por estaciones y otra por años, modificando la URL hasta conseguir todos los años deseados.

Hemos descargado los datos año por año desde 2001 hasta 2020 incluidos ya que casi todas las estaciones no disponen de datos ni más antiguos ni más nuevos.

## 5.2 Integración de los datos

Lo primero que hemos hecho ha sido integrar todos los datos de cada estación en un único fichero y hemos eliminado los datos que estaban vacíos.

Como resultado hemos obtenido por cada ciudad a predecir una serie de un millón de datos. De la web hemos obtenido el valor de la temperatura medido cada 10 minutos, por lo tanto, tenemos 6 mediciones de temperatura cada hora. Esto es importante a tener en cuenta más adelante ya que si queremos predecir 2 horas tenemos que saber que tenemos que predecir los 12 siguientes valores y en el caso de 6 horas serían los 36 siguientes.

En el resto de las estaciones nos hemos dado cuenta de que en la mayoría faltaban muchos datos o las mediciones estaban realizadas en periodos de tiempo distintos por lo tanto hemos decidido predecir en las cuatro mencionadas anteriormente y el resto se han intentado usar para entrenar el modelo.

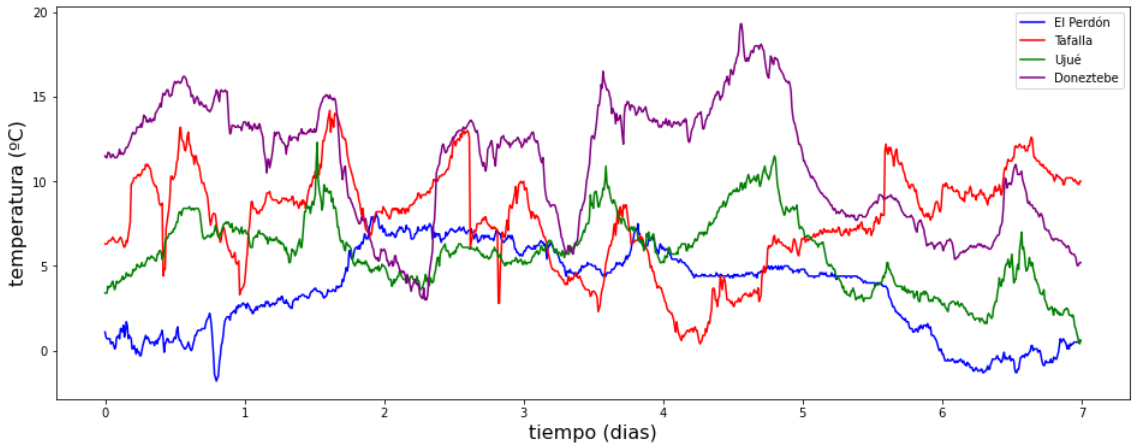


*Figura 16: Comparativa de graficas de las distintas estaciones.*

## 5.3 Limpieza de los datos

En este caso, el único tratamiento que le hemos realizado a los datos ha sido un suavizado mediante la herramienta hpfilter de statsmodel. Esta herramienta aplica el filtro de Hodrick-Prescott que sirve para extraer la tendencia de una serie temporal. Este filtro divide la serie en dos componentes, un componente cíclico y otro tendencial (Figura 17) *[14]*.

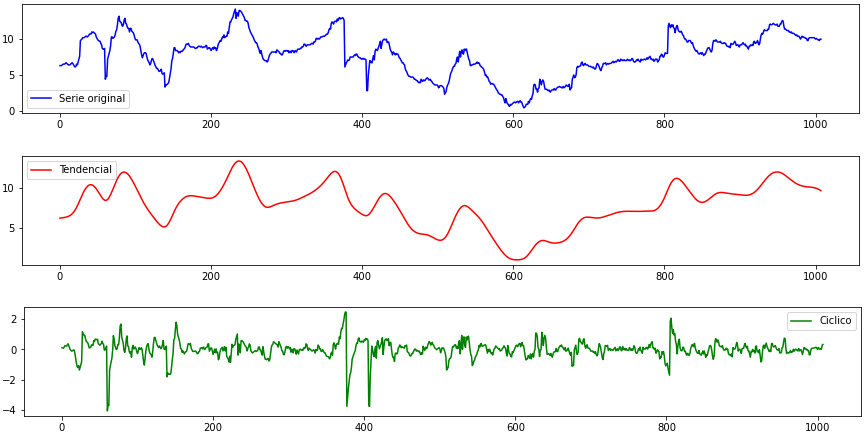


*Figura 17: Funcionamiento del filtro de Hodrick-Prescott.*

$$y_t = c_t + \tau_t \text{ para } t = 1, 2, 3, \dots, T$$

Siendo $y_t$ la serie temporal, $c_t$ el componente cíclico y $\tau_t$ el componente tendencial. Dado un valor $\lambda$ positivo escogido, se calcula el componente tendencial mediante la siguiente fórmula:

$$\min \sum_{t=1}^{T} (y_t - \tau_t)^2 + \lambda \sum_{t=2}^{T-1} [(\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1})]^2$$

El tendencial calculado siempre pasa por el centro de la serie cumpliéndose:

$$\sum_{t=1}^{T} (y_t - \tau_t) = 0$$

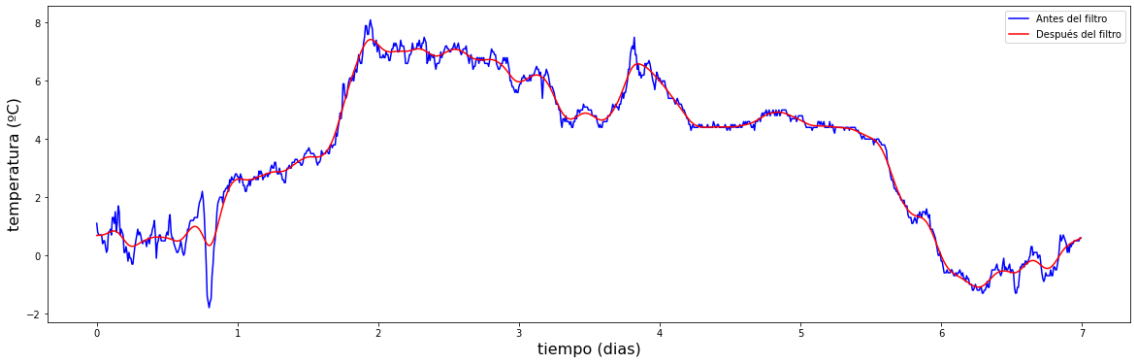


*Figura 18: Comparativa de graficas antes (azul) y después (rojo) de aplicar el filtro.*

## 5.4 Transformación de los datos

Finalmente, después de aplicar todo lo anterior, para que los modelos funcionen correctamente hemos tenido que normalizar los datos, haciendo que todos tengan valores entre -1 y 1 ya que los modelos no funcionan correctamente con datos fuera de ese rango. Posteriormente, para mostrar los resultados todos los datos han sido desnormalizados.

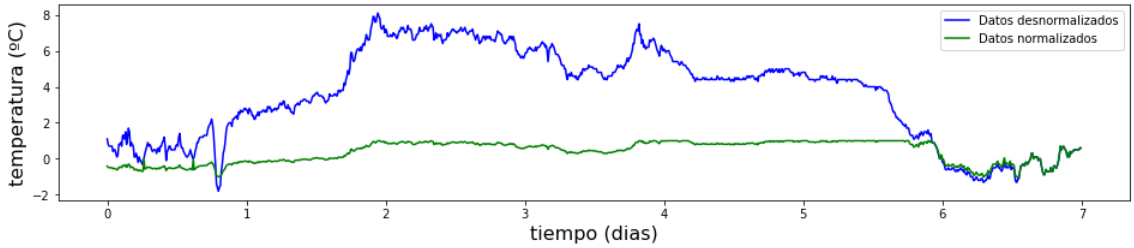


*Figura 19: Normalización de los datos.*

## 5.5 Creación de los Datasets

Para la creación de los datasets hemos dividido los datos en series de un tamaño preseleccionado, eligiendo en todo momento el número de datos de entrada y el número de datos de salida de cada una de esas series. Al modelo no le metemos todas las series a la vez, en su lugar, las metemos por bloques. Al tamaño de esos bloques lo llamamos batch size A su vez, el dataset lo dividimos en conjuntos de entrenamiento, validación y test con un 70%, 15% y 15% de los datos respectivamente.

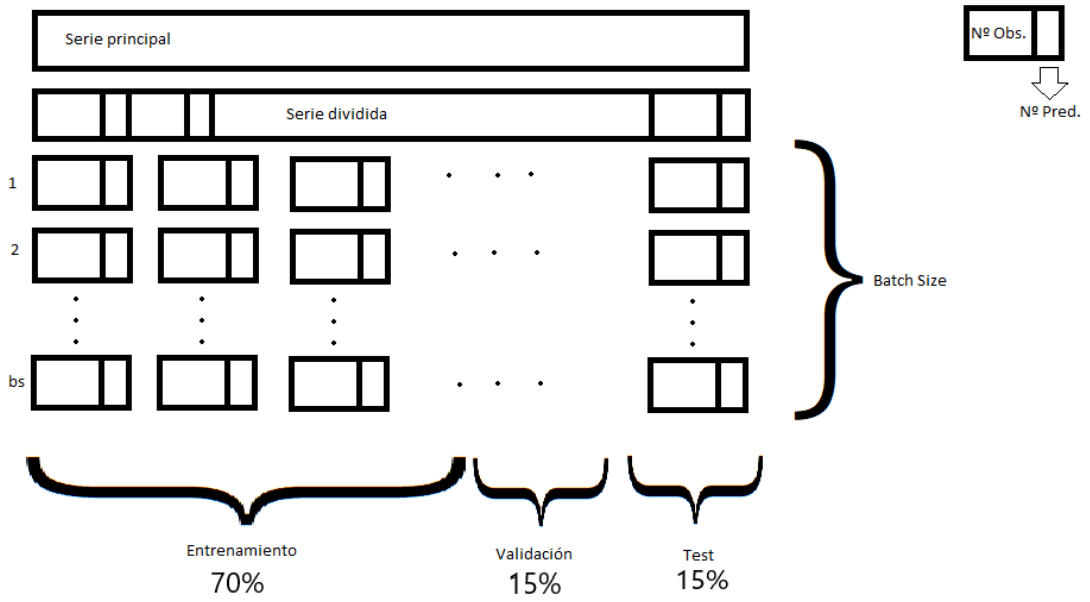


*Figura 20: División de la serie principal.*

# 6. Pruebas realizadas

En esta sección se muestran todas las pruebas realizadas. Para ello primero hemos elegido los modelos con los que queremos trabajar. Después hemos ido realizando distintas pruebas, en función de los resultados de cada prueba y de las conclusiones que sacamos de cada una de ellas hemos realizado las siguientes.

Una vez que ya teniamos los datos preprocesados correctamente, hemos realizado una prueba para evaluar distintas arquitecturas, tanto RNN, como LSTM y GRU. Para ello hemos utilizado una de las estaciones, el Perdón, el MSE como función de coste y los parámetros que hemos combinado para hacer las pruebas son los siguientes:

- Cantidad de capas ocultas: 1, 2, 3 y 4
- Número de celdas en cada capa oculta: 10, 25, 50 y 100
- Learning rate: 0.01, 0.005 y 0.001

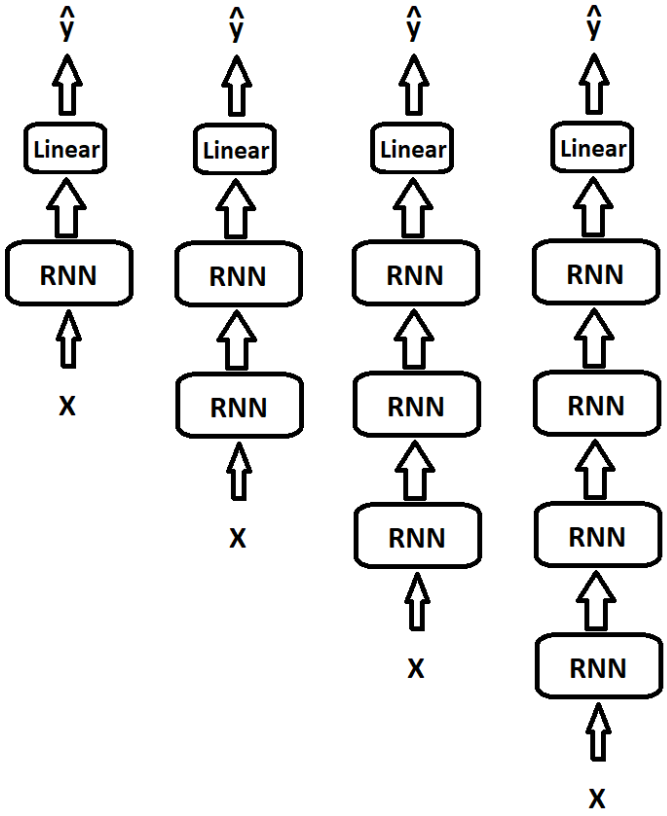


*Figura 21: Estructuras de las RNN en función del número de capas ocultas.*

Al utilizar diferente número de capas y numero de celdas dentro de cada capa hacemos que los parámetros que aprende la red sean diferentes y con ello las predicciones. Al utilizar un menor learning rate, lo que conseguimos es cambiar esos parámetros en menor medida consiguiendo ajustarnos más en las predicciones.

Hemos utilizado 100 valores de entrada, 6 valores de salida y un batch size de 100. Hemos utilizado de nuevo el MSE para calcular el error obtenido y así poder comparar los distintos modelos.

| Layers | Hidden Size | Learning rate | MSE |
|---|---|---|---|
| **4** | **25** | **0.01** | **0.00050** |
| 3 | 25 | 0.01 | 0.00068 |
| 3 | 100 | 0.001 | 0.00076 |
| 4 | 25 | 0.005 | 0.00078 |
| 2 | 100 | 0.001 | 0.00082 |
| 4 | 50 | 0.001 | 0.00084 |
| 3 | 50 | 0.005 | 0.00089 |
| 3 | 10 | 0.01 | 0.00122 |
| 2 | 50 | 0.001 | 0.00123 |
| 3 | 25 | 0.005 | 0.00147 |

*Figura 22: Resultados obtenidos de las pruebas con RNN.*

| Layers | Hidden Size | Learning rate | MSE |
|---|---|---|---|
| **4** | **50** | **0.01** | **0.00025** |
| **3** | **50** | **0.01** | **0.00033** |
| 3 | 100 | 0.01 | 0.00037 |
| 4 | 25 | 0.01 | 0.00054 |
| 1 | 100 | 0.01 | 0.00054 |
| 3 | 100 | 0.005 | 0.00055 |
| 4 | 100 | 0.005 | 0.00056 |
| 2 | 25 | 0.01 | 0.00057 |
| 4 | 25 | 0.005 | 0.00062 |
| 2 | 10 | 0.01 | 0.00063 |

*Figura 23: Resultados obtenidos de las pruebas con LSTM.*

| Layers | Hidden Size | Learning rate | MSE |
|---|---|---|---|
| 3 | 100 | 0.01 | 0.00014 |
| **3** | **50** | **0.005** | **0.00025** |
| 4 | 50 | 0.005 | 0.00025 |
| 4 | 50 | 0.01 | 0.00027 |
| **1** | **50** | **0.01** | **0.00038** |
| 3 | 100 | 0.005 | 0.00041 |
| 3 | 25 | 0.005 | 0.00048 |
| 3 | 25 | 0.01 | 0.00052 |
| 4 | 10 | 0.01 | 0.00052 |
| 1 | 100 | 0.005 | 0.00061 |

*Figura 24: Resultados obtenidos de las pruebas con GRU.*

En las tablas de las figuras 22,23 y 24 podemos visualizar los 10 mejores resultados de cada uno de los modelos. Subrayados están los casos considerados mejores, en el caso de GRU podemos observar que hay resultados mejores, pero, teniendo en cuenta el número de celdas que tendría cada modelo hemos decido quedarnos con los que tienen menos ya que los resultados no son muy diferentes pero los tiempos de ejecución aumentarían demasiado.

Como podemos comprobar, las LSTM y las GRU dan resultados mucho mejores que las RNN clásicas. Esto se debe gracias a las mejoras incluidas ya que utilizan de mucha mejor manera los datos más antiguos.

A partir de aquí hemos trabajado únicamente con los modelos y parámetros elegidos (figuras 21, 22 y 23). Llamaremos RNN al modelo RNN subrayado en amarillo en la figura 21, LSTM1 y LSTM2 a los modelos de LSTM subrayados en amarillo en la figura 22 correspondientes y GRU1 y GRU2 a los modelos de GRU subrayados en amarillo en la figura 23 correspondientes. Las pruebas que se han realizado son:

- Entrenamiento con todas las estaciones de Navarra excepto las 4 sobre las que vamos a predecir.
- Entrenamiento con las estaciones más cercanas a cada estación sobre la que queremos predecir.
- Entrenamiento con datos antiguos de las propias estaciones sobre las que queremos predecir.
- Entrenamiento con modificaciones de la LSTM1.

## 6.1 Primera prueba

Lo primero que hemos realizado ha sido unas pruebas utilizando todas las estaciones excepto las del Perdon, Tafalla, Ujue y Doneztebe para entrenamiento y validación, y estas 4 únicamente para test. Lo hemos realizado de esta manera ya que hay muchas estaciones con el fin de tener el mayor número de series de entrenamiento posibles. Estas pruebas han sido muy duraderas ya que teníamos datos de todas las estaciones de Navarra.

Los errores obtenidos los podemos observar en las figuras 25 a 28.

| DONEZTEBE | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.103408 | 0.103052 | 0.021398 | 0.105844 | 0.034761 | 0.032418 | 0.037126 | 0.100551 | 0.102889 |
| **LSTM1** | **0.000003** | **0.000056** | 0.002143 | **0.001178** | **0.003310** | 0.031189 | 0.009587 | 0.035592 | **0.044256** |
| **LSTM2** | 0.000020 | 0.000062 | 0.001876 | 0.002236 | 0.004562 | **0.009445** | 0.017561 | 0.036249 | 0.101841 |
| **GRU1** | 0.000030 | 0.000496 | **0.001509** | 0.003407 | 0.004482 | 0.016887 | **0.008801** | 0.036541 | 0.051290 |
| **GRU2** | 0.000096 | 0.000405 | 0.022201 | 0.023914 | 0.032301 | 0.027062 | 0.032556 | **0.035248** | 0.048128 |

*Figura 25: Errores MSE obtenidos en Doneztebe usando todas las estaciones de Navarra.*

| PERDÓN | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.114438 | 0.115248 | 0.019374 | 0.114670 | 0.031624 | 0.030259 | 0.035396 | 0.109959 | 0.108292 |
| **LSTM1** | **0.000003** | **0.000053** | 0.001791 | **0.001309** | **0.003418** | 0.029491 | 0.009201 | 0.036021 | **0.045119** |
| **LSTM2** | 0.000019 | 0.000066 | 0.001672 | 0.002104 | 0.004420 | **0.007905** | 0.017580 | 0.035356 | 0.107698 |
| **GRU1** | 0.000032 | 0.000436 | **0.001333** | 0.003128 | 0.004356 | 0.014690 | **0.008390** | 0.036374 | 0.050604 |
| **GRU2** | 0.000080 | 0.000394 | 0.020029 | 0.023902 | 0.031142 | 0.026222 | 0.031781 | **0.032078** | 0.051021 |

*Figura 26: Errores MSE obtenidos en El Perdón usando todas las estaciones de Navarra.*

| TAFALLA | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.104401 | 0.104162 | 0.016603 | 0.106384 | 0.029452 | 0.027478 | 0.030860 | 0.108039 | 0.104616 |
| **LSTM1** | **0.000002** | **0.000050** | 0.001785 | **0.001346** | **0.003871** | 0.026787 | 0.010744 | 0.036822 | **0.043948** |
| **LSTM2** | 0.000018 | 0.000062 | 0.001839 | 0.002536 | 0.004959 | **0.009690** | 0.019142 | 0.037315 | 0.103958 |
| **GRU1** | 0.000029 | 0.000462 | **0.001365** | 0.003280 | 0.004861 | 0.015025 | **0.010317** | 0.037999 | 0.051248 |
| **GRU2** | 0.000081 | 0.000386 | 0.017282 | 0.014615 | 0.028538 | 0.024800 | 0.027868 | **0.033876** | 0.044664 |

*Figura 27: Errores MSE obtenidos en Tafalla usando todas las estaciones de Navarra.*

| UJUÉ | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.096506 | 0.094931 | 0.004095 | 0.095145 | 0.013246 | 0.022007 | 0.024643 | 0.091631 | 0.089737 |
| **LSTM1** | **0.000001** | **0.000017** | 0.000609 | **0.000273** | **0.001023** | 0.012443 | 0.005392 | **0.009370** | 0.011491 |
| **LSTM2** | 0.000007 | 0.000020 | 0.000621 | 0.000502 | 0.001396 | **0.004244** | 0.007721 | 0.010365 | 0.088634 |
| **GRU1** | 0.000017 | 0.000101 | **0.000338** | 0.000839 | 0.001619 | 0.007295 | **0.004872** | 0.010606 | **0.008935** |
| **GRU2** | 0.000018 | 0.000181 | 0.004518 | 0.003541 | 0.009460 | 0.012388 | 0.024393 | 0.024898 | 0.009893 |

*Figura 28: Errores MSE obtenidos en Ujué usando todas las estaciones de Navarra.*

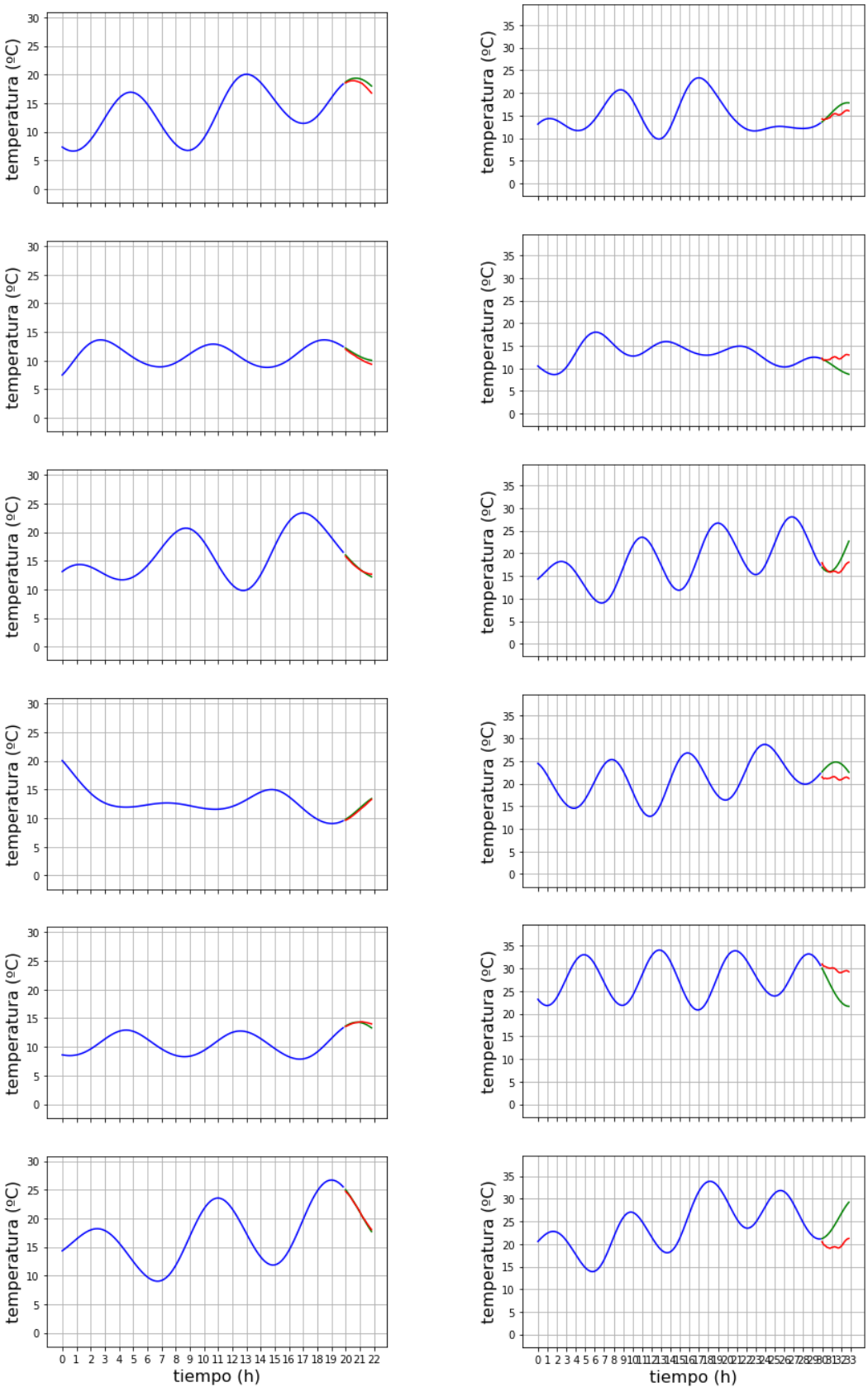


*Figura 29: Algunas de las series obtenidas entrenando con todas las estaciones de navarra. Predicciones de 1 hora (Izquierda) y de 3 horas (derecha).*

Hemos observado que las ejecuciones eran muy largas en el tiempo puesto que se usaba una gran cantidad de datos para realizar el entrenamiento y los resultados no eran lo suficientemente correctos, puesto que, a partir de predecir 3 horas, podemos observar en las gráficas de la figura 29 que las predicciones están muy alejadas de los valores reales. Consideramos que esto se debe a que, al tener muchas estaciones en puntos distintos de la geografía navarra, incluidas estaciones en la zona de La Ribera, las series que se forman en estas estaciones son muy distintas a las de las cuatro estaciones de test.

## 6.2 Segunda Prueba

Para evitar los problemas tanto de velocidad como los de las estaciones muy lejanas hemos decidido cambiar el dataset de entrenamiento. Para ellos hemos entrenado, para cada una de las cuatro estaciones, únicamente con sus estaciones vecinas. Con esto conseguimos que nuestros modelos entrenen de manera más rápida y más eficiente ya que así también eliminamos las estaciones que producen series muy diferentes a las que queremos predecir.

Para predecir temperaturas en cada estación hemos usado como entrenamiento y validación las siguientes estaciones:

- Donetzebe: Iñarbegi, Gorramendi, Goizueta y Bera.
- Perdón: Adiós, Carrascal, Pamplona y Pamplona UPNA.
- Tafalla: Olite, San Martín de Unx, Miranda de Arga y Artajona.
- Ujué: San Martín de Unx, Aibar, Murillo el Fruto y Getadar.

En la figura 30 se muestran, con cruces rojas las estaciones vecinas utilizadas.

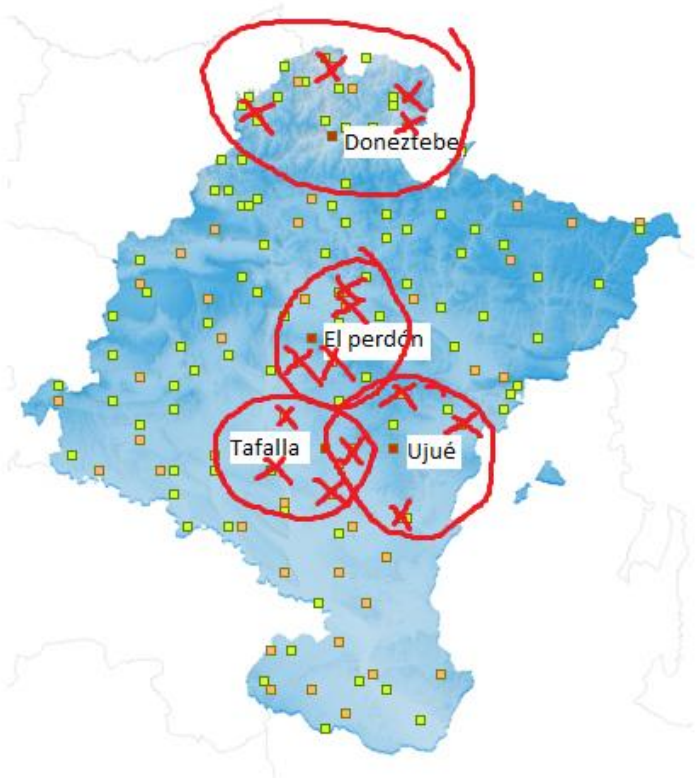


*Figura 30: Mapa que muestra los vecinos utilizados en la segunda prueba.*

Al entrenar los modelos, esta vez eran bastante más rápidos que en la anterior prueba y los resultados algo mejores.

| DONEZTEBE | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.000656 | 0.000415 | 0.000673 | 0.003622 | 0.007749 | 0.025943 | 0.033876 | 0.030925 | 0.032962 |
| **LSTM1** | **0.000065** | **0.000131** | **0.000532** | 0.004484 | 0.009053 | 0.015952 | 0.020377 | 0.034025 | 0.034019 |
| **LSTM2** | 0.000071 | 0.000197 | 0.000833 | 0.001516 | 0.002898 | 0.014826 | 0.029019 | 0.030161 | 0.033212 |
| **GRU1** | 0.000080 | 0.000188 | 0.000698 | **0.001243** | **0.002359** | **0.008707** | 0.014554 | 0.020043 | **0.020661** |
| **GRU2** | 0.000087 | 0.000288 | 0.000715 | 0.002813 | 0.002816 | 0.011169 | **0.012408** | **0.018906** | 0.026081 |

*Figura 31: Errores MSE obtenidos en Doneztebe entrenando con sus estaciones vecinas.*

| PERDÓN | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.000120 | 0.001261 | 0.002180 | 0.027685 | 0.032008 | 0.028967 | 0.038098 | 0.036178 | 0.115134 |
| **LSTM1** | 0.000529 | 0.000852 | 0.001726 | 0.014326 | 0.030813 | 0.030239 | 0.015689 | 0.039488 | 0.043910 |
| **LSTM2** | 0.000118 | 0.000715 | 0.013760 | 0.003216 | 0.032733 | 0.044294 | 0.032815 | 0.038402 | 0.037014 |
| **GRU1** | 0.000133 | **0.000313** | **0.001378** | **0.002957** | **0.006551** | **0.011438** | **0.012494** | **0.031557** | **0.034053** |
| **GRU2** | **0.000101** | 0.007712 | 0.009226 | 0.038302 | 0.010092 | 0.014403 | 0.016199 | 0.032742 | 0.034432 |

*Figura 32: Errores MSE obtenidos en El Perdón entrenando con sus estaciones vecinas.*

| TAFALLA | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.000136 | 0.000544 | 0.012485 | 0.003844 | 0.017825 | 0.081310 | 0.018559 | 0.025466 | 0.034411 |
| **LSTM1** | **0.000051** | 0.000361 | 0.001138 | **0.002625** | 0.008285 | 0.013629 | 0.015584 | 0.024755 | 0.032720 |
| **LSTM2** | 0.000050 | **0.000292** | **0.001007** | 0.005273 | 0.008228 | 0.013469 | 0.018129 | 0.027827 | 0.045426 |
| **GRU1** | 0.000188 | 0.000387 | 0.001052 | 0.003480 | **0.007455** | 0.011869 | **0.013542** | 0.020776 | **0.021943** |
| **GRU2** | 0.000069 | 0.000472 | 0.007942 | 0.005317 | 0.008536 | **0.011684** | 0.014798 | **0.020651** | 0.022564 |

*Figura 33: Errores MSE obtenidos en Tafalla entrenando con sus estaciones vecinas.*

| UJUÉ | 1 | 2 | 3 | 4 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|---|---|
| **RNN** | 0.000054 | 0.000569 | 0.002757 | 0.019111 | 0.024957 | 0.024429 | 0.113572 | 0.035012 | 0.037403 |
| **LSTM1** | 0.000068 | 0.000238 | **0.000860** | 0.002249 | 0.006556 | 0.010944 | 0.026742 | 0.030800 | 0.032311 |
| **LSTM2** | 0.000054 | 0.000372 | 0.001018 | 0.003218 | 0.020605 | 0.025738 | 0.018758 | 0.032432 | 0.034595 |
| **GRU1** | **0.000042** | **0.000225** | 0.000981 | **0.002205** | **0.005203** | **0.009486** | **0.011795** | **0.019454** | **0.027830** |
| **GRU2** | 0.000061 | 0.000448 | 0.005703 | 0.002663 | 0.025787 | 0.013117 | 0.011917 | 0.024675 | 0.030162 |

*Figura 34: Errores MSE obtenidos en Ujué entrenando con sus estaciones vecinas.*

Pese a haber mejorado en algunos casos, sigue dando errores bastante altos, en las tablas podemos observar que las predicciones son un 23'96% mejores cuando predecimos 24 horas, pero los resultados siguen siendo muy poco precisos (Figura 35). Si nos fijamos en un mapa de Navarra (Figura 15), podemos ver que las estaciones con las que entrenamos están a diferentes alturas y altitudes y por tanto, aunque estén cerca, las series que se crean son diferentes y, aunque en menor medida, tenemos el mismo problema que antes.

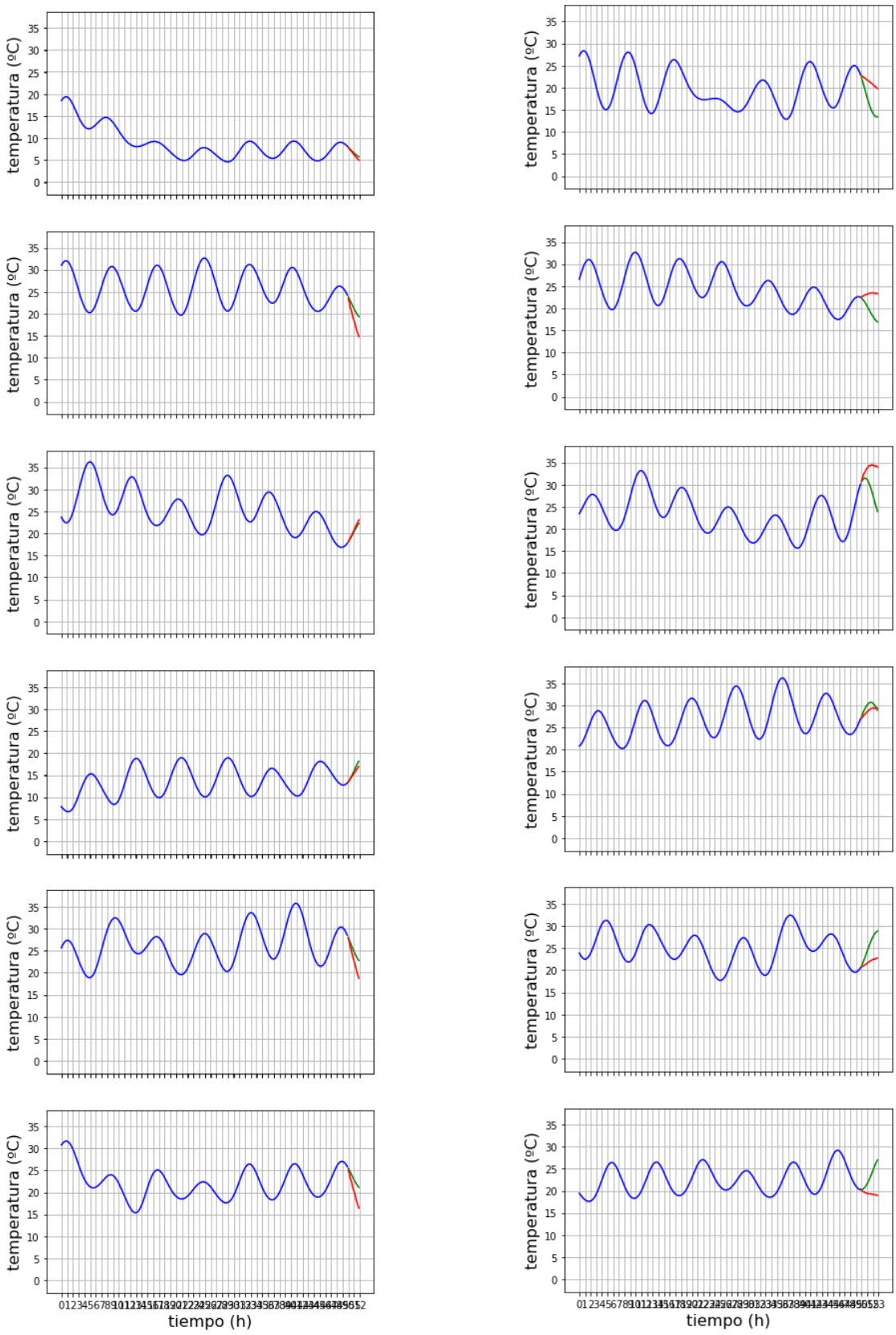


*Figura 35: Algunas de las series obtenidas entrenando con las estaciones vecinas. Predicciones de 2 horas (Izquierda) y de 3 horas (derecha).*

## 6.3 Tercera prueba

Esta vez hemos decidido entrenar con las mismas estaciones sobre las cuales queremos predecir datos. Esto va a evitar los problemas que nos han surgido anteriormente. Para ello lo que hacemos es usar el 70% de los datos de cada estación, unirlos y guardarlos como datos de entrenamiento. Con el resto de los datos, la mitad los usaremos para validación y la otra mitad como test. Las ejecuciones de esta prueba son mucho más rápidas que las anteriores con lo cual nos permiten ejecutar nuestros modelos varias veces, así que, esta vez, en lugar de realizar una única ejecución, hemos creado y entrenado nuestros modelos 10 veces y luego hemos obtenido una media de los resultados. Con esta media hemos creado nuestras gráficas y obtenido los errores.

| DONEZTEBE | 1 | 2 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|
| **RNN** | 0.007636 | 0.018895 | 0.028379 | 0.047602 | 0.031828 | 0.044568 | 0.040476 |
| **LSTM1** | 0.011906 | 0.010941 | 0.012448 | 0.024540 | 0.014884 | 0.033355 | 0.039120 |
| **LSTM2** | 0.000027 | 0.000075 | 0.017004 | 0.017795 | 0.016130 | 0.013534 | 0.013863 |
| **GRU1** | 0.000023 | **0.000024** | **0.004226** | **0.006182** | **0.006988** | **0.012579** | **0.012415** |
| **GRU2** | **0.000015** | 0.000948 | 0.009371 | 0.009884 | 0.009099 | 0.012916 | 0.012680 |

*Figura 36: Errores MSE obtenidos en Doneztebe entrenando con datos suyos antiguos.*

| PERDÓN | 1 | 2 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|
| **RNN** | 0.008942 | 0.020123 | 0.022880 | 0.043251 | 0.023554 | 0.032692 | 0.029805 |
| **LSTM1** | **0.000006** | 0.000034 | 0.016267 | 0.013512 | 0.010146 | 0.029367 | 0.014443 |
| **LSTM2** | 0.000024 | 0.000070 | 0.012845 | 0.009264 | 0.010320 | 0.009845 | 0.013059 |
| **GRU1** | 0.000019 | **0.000019** | **0.002712** | **0.004303** | **0.005805** | **0.009245** | **0.012108** |
| **GRU2** | 0.000010 | 0.001083 | 0.006030 | 0.006306 | 0.007209 | 0.009407 | 0.012424 |

*Figura 37: Errores MSE obtenidos en El Perdón entrenando con datos suyos antiguos.*

| TAFALLA | 1 | 2 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|
| **RNN** | 0.009507 | 0.023137 | 0.025532 | 0.049971 | 0.037514 | 0.042402 | 0.042532 |
| **LSTM1** | **0.000008** | 0.000037 | 0.018476 | 0.015790 | 0.016528 | 0.028310 | 0.013807 |
| **LSTM2** | 0.000029 | 0.000072 | 0.015741 | 0.013976 | 0.015664 | 0.007237 | 0.011392 |
| **GRU1** | 0.000022 | **0.000020** | **0.003069** | **0.004018** | **0.005270** | **0.006458** | **0.010337** |
| **GRU2** | 0.000012 | 0.001227 | 0.007067 | 0.007021 | 0.007311 | 0.006774 | 0.010500 |

*Figura 38: Errores MSE obtenidos en Tafalla entrenando con datos suyos antiguos.*

| UJUÉ | 1 | 2 | 6 | 9 | 12 | 18 | 24 |
|---|---|---|---|---|---|---|---|
| **RNN** | 0.009508 | 0.021726 | 0.024402 | 0.047167 | 0.027199 | 0.035238 | 0.033185 |
| **LSTM1** | **0.000006** | 0.000033 | 0.017872 | 0.013997 | 0.011263 | 0.029377 | 0.013436 |
| **LSTM2** | 0.000026 | 0.000067 | 0.014640 | 0.009907 | 0.010853 | 0.008859 | 0.011800 |
| **GRU1** | 0.000020 | **0.000017** | **0.002674** | **0.003538** | **0.004579** | **0.008343** | **0.010553** |
| **GRU2** | 0.000010 | 0.001099 | 0.006315 | 0.005736 | 0.006318 | 0.008620 | 0.011070 |

*Figura 39: Errores MSE obtenidos en Ujué entrenando con datos suyos antiguos.*

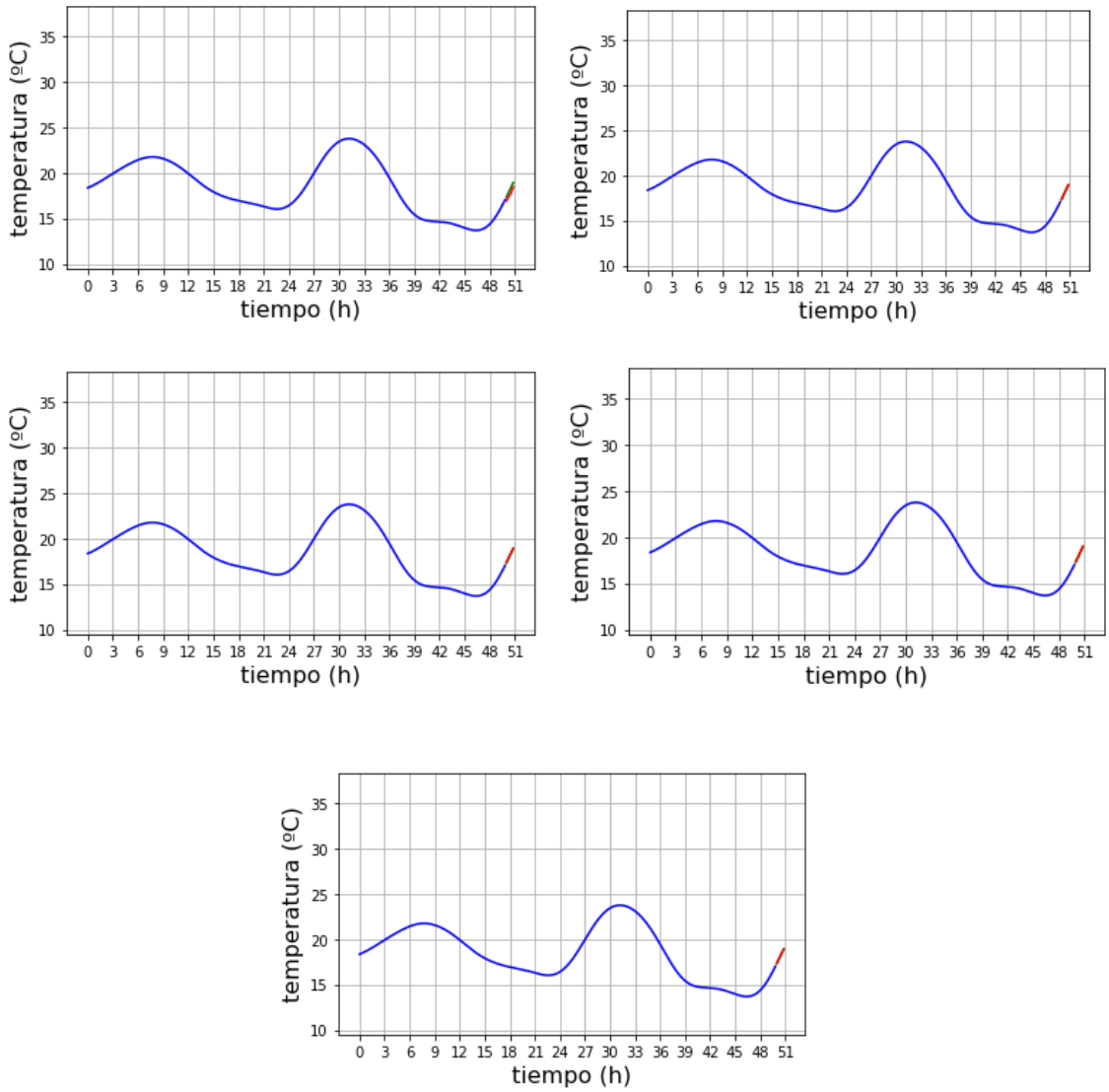


*Figura 40: Predicción de 1 hora para Doneztebe con los distintos modelos.*

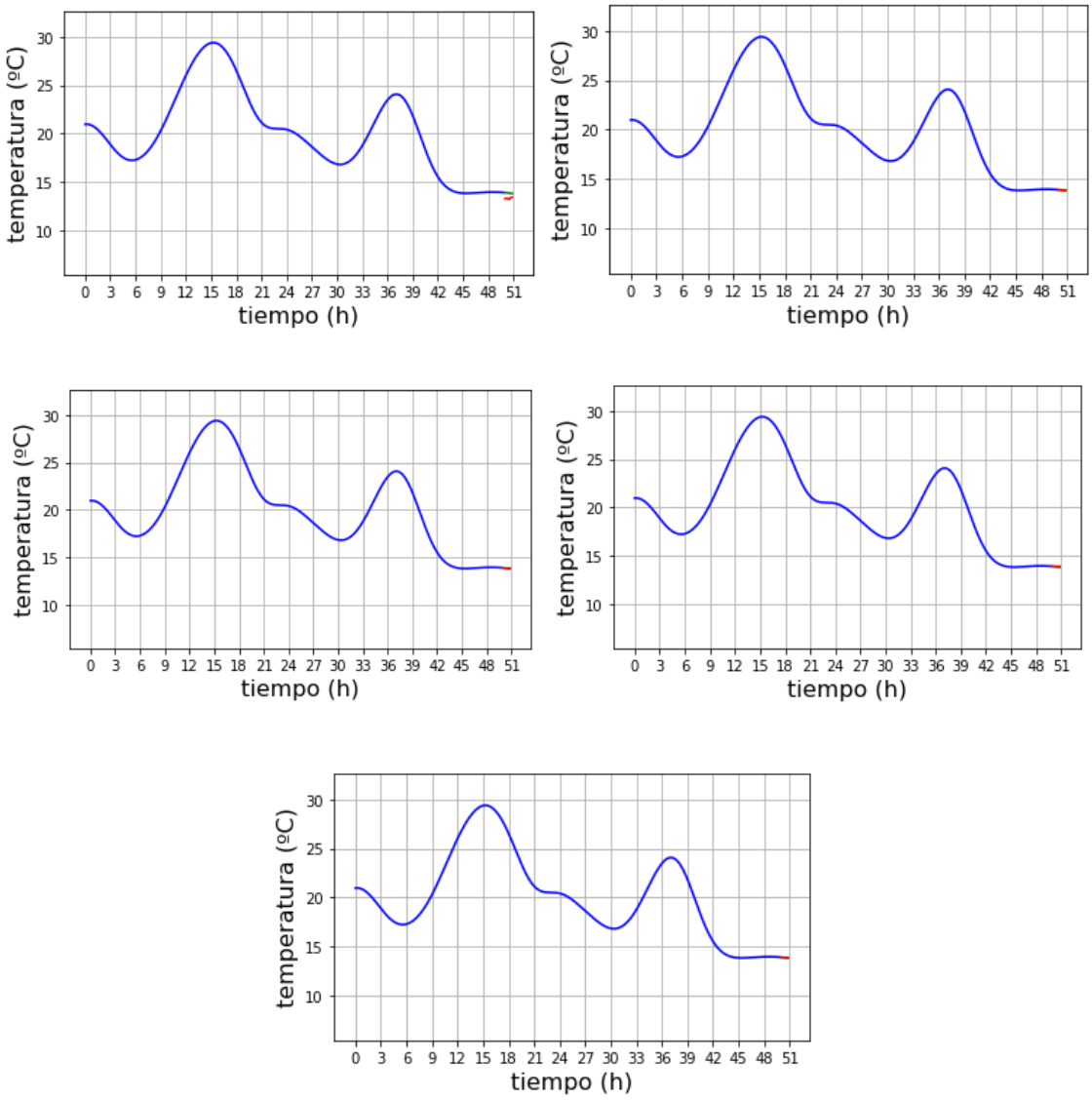


*Figura 41: Predicción de 1 hora para El Perdón con los distintos modelos.*

En las figuras 40 y 41, como podemos observar, las predicciones son muy buenas ya que solo estamos prediciendo 6 valores. Se puede observar muy bien el problema del gradiente desvaneciente de las RNN (grafica de arriba a la derecha en ambas figuras) ya que como usamos bastantes elementos como entrada al modelo, la predicción es bastante peor que con el resto de los modelos, especialmente para el Perdón puesto que veníamos de muchas horas con temperaturas bastante cambiantes y pasamos a que las últimas tres horas son bastante estables. Los errores MSE de las RNN son 100 veces más altos que las del resto prediciendo 1 hora.

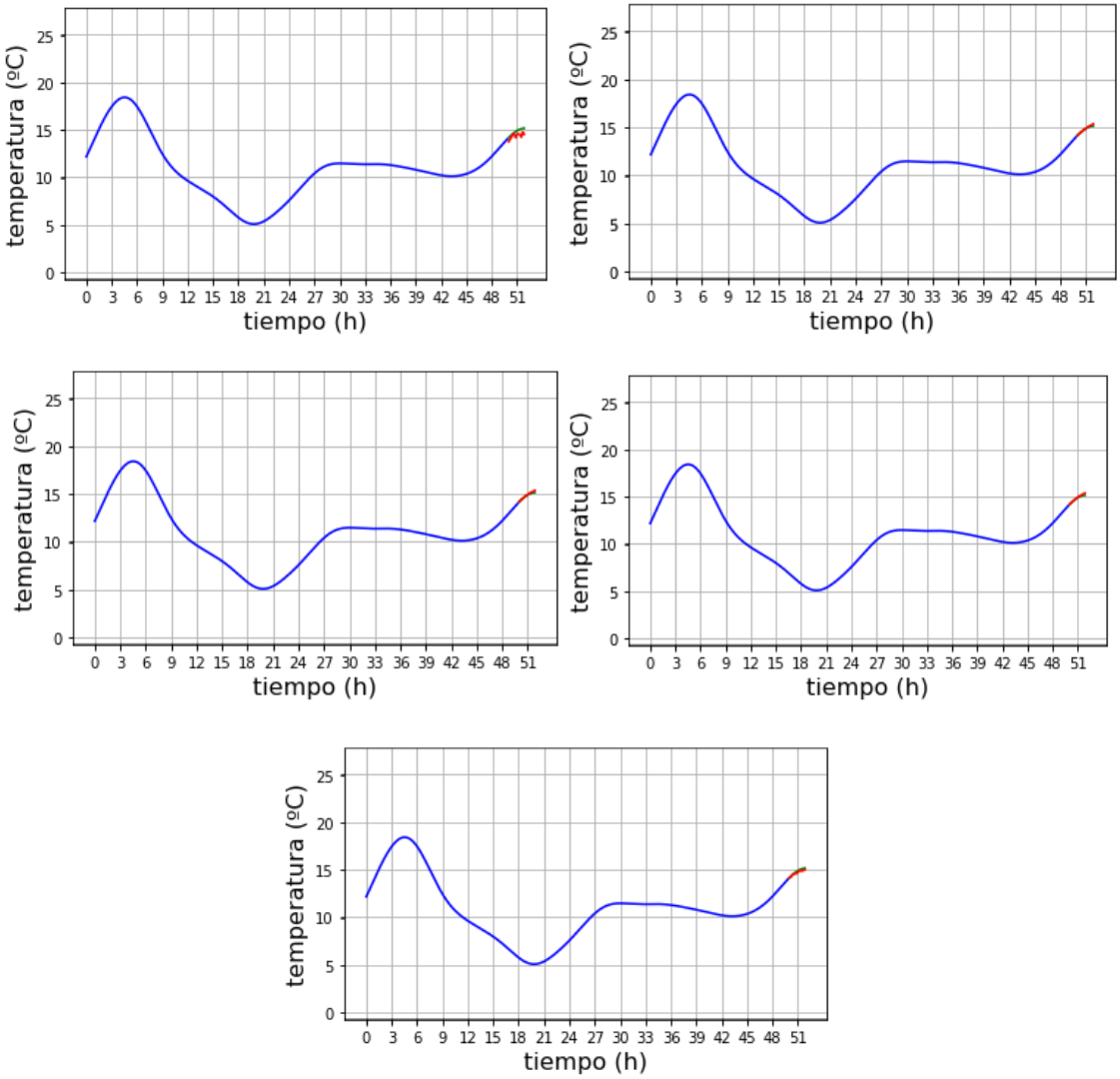


*Figura 41: Predicción de 2 horas para Tafalla con los distintos modelos.*

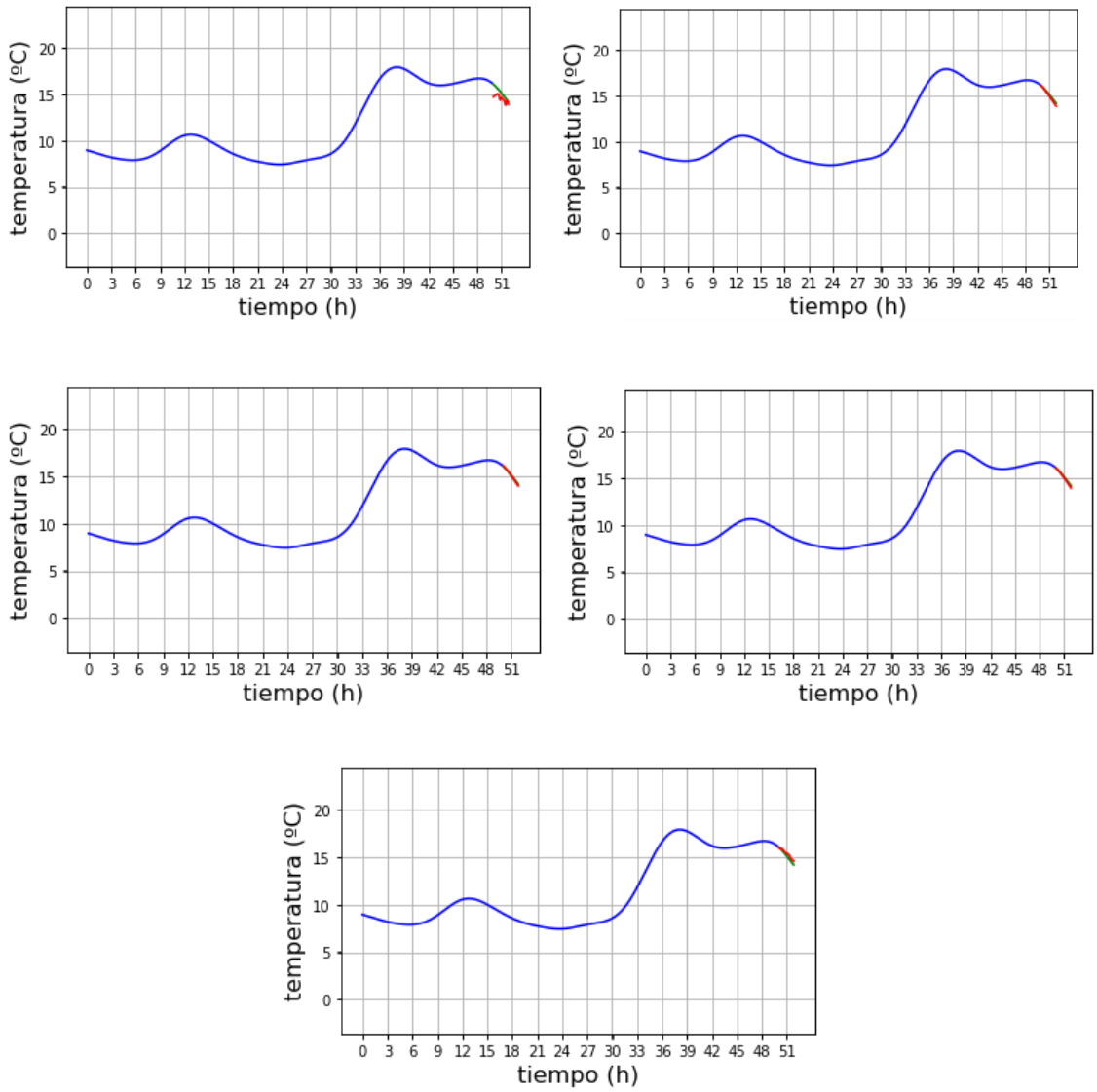


*Figura 42: Predicción de 2 horas para Ujué con los distintos modelos.*

Como podemos ver en las figuras 41 y 42, las predicciones de la RNN (arriba a la derecha) están muy alejadas de los valores reales en comparación con las demás modelos, para predicciones de 24 horas, el error en la predicción es 3,3 veces más alto.

A partir de aquí dejaremos de utilizar las RNN, se puede observar que genera errores que consideramos demasiado elevados. Como se observa en la figura 42, al predecir 12 valores ya empieza a funcionar peor que el resto y en la figura 43 se muestra como al predecir 6, 12 o 18 horas no nos sirve para nada.

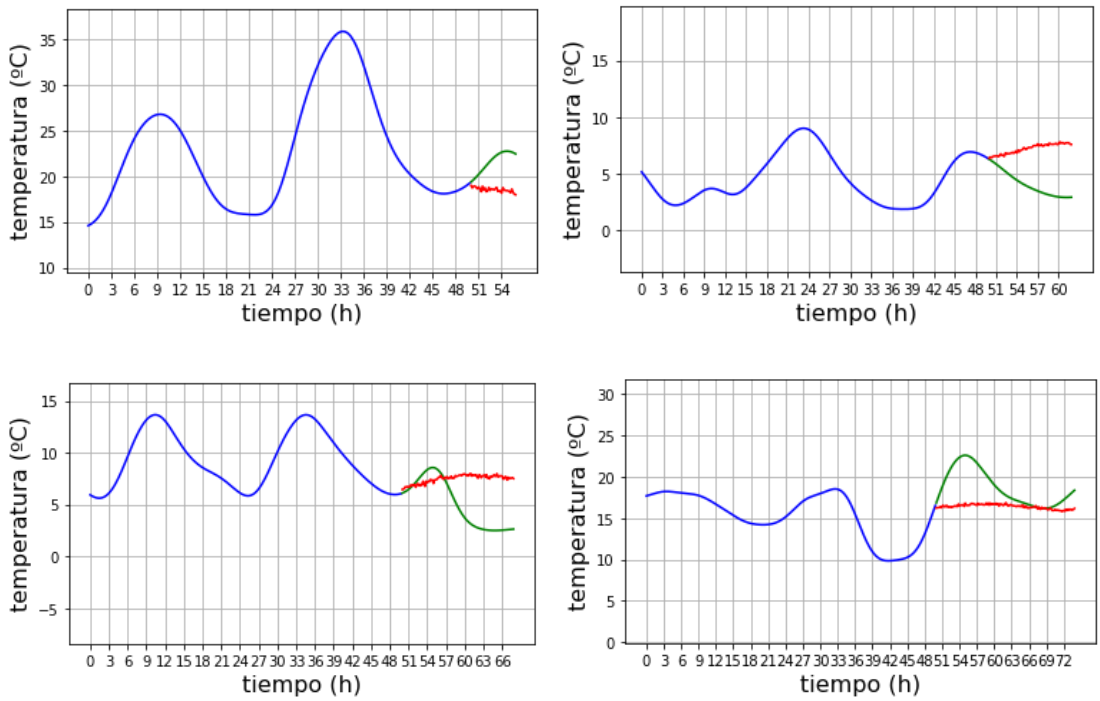


*Figura 43: Predicción de 6, 12, 18 y 24 horas mediante RNN clásicas.*

Como podemos observar en la figura 43, las RNN no predicen nada parecido a la serie real. El MSE no es demasiado alto tampoco porque en casos como las dos graficas de abajo, las predicciones siguen estando cercanas a los datos reales aunque ni si quiera siguen la tendencia de la serie.

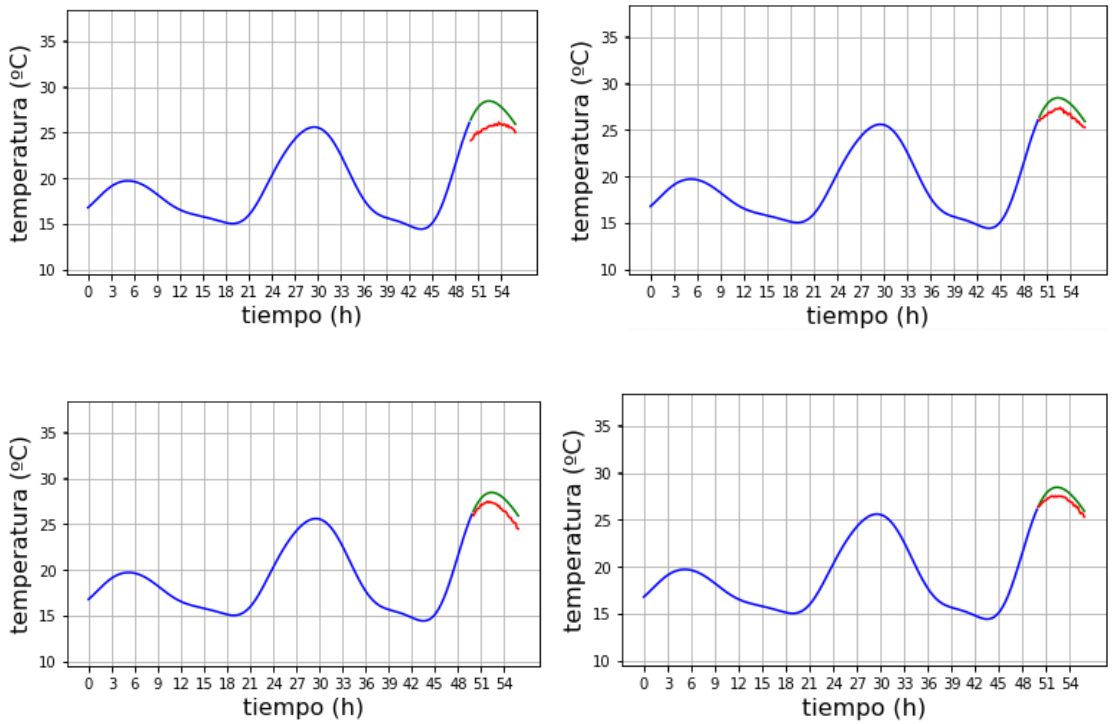


*Figura 44: Predicción de 6 horas en Donezteb mediante todos los modelos excepto RNN.*

En la figura 44 podemos observar que los modelos de LSTM y de GRU funcionan bastante mejor. El modelo LSTM1 da peores resultados que el resto (el error es 5,13 veces mayor que GRU1), pero sigue manteniendo la tendencia que realiza la temperatura. Al estar aumentando el número de valores a predecir es normal que cada vez las predicciones estén un poco mas alejadas de los valores reales.

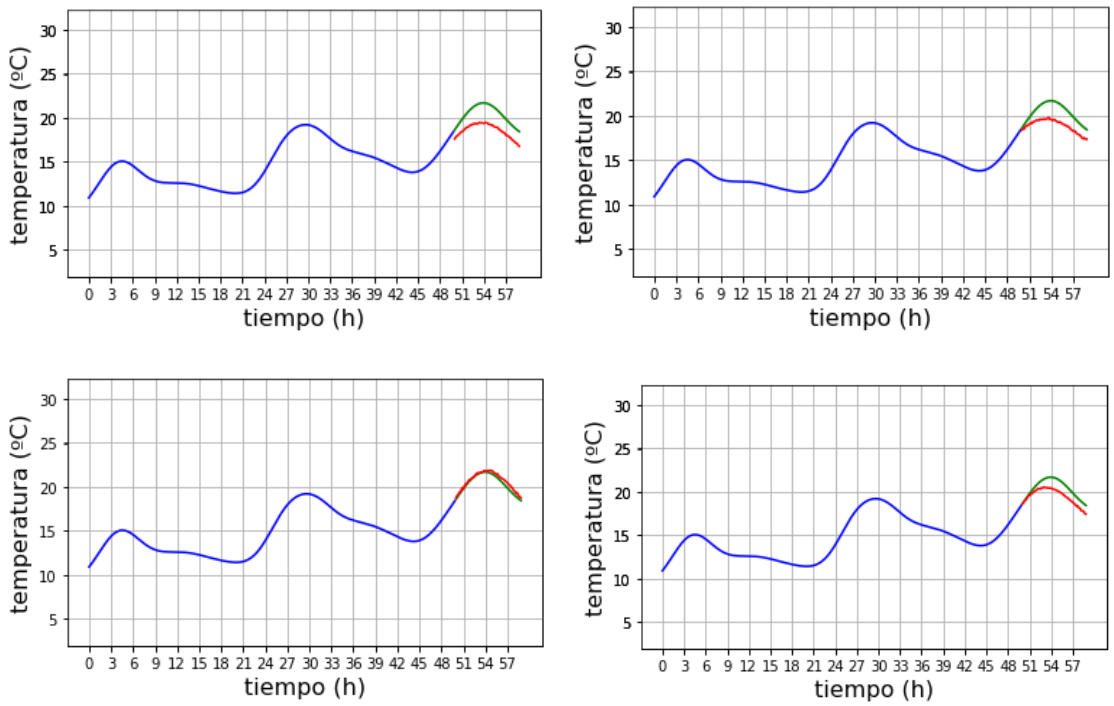


*Figura 45: Predicción de 9 horas en el Perdón mediante todos los modelos excepto RNN.*

En las gráficas de la figura 45 se observa claramente lo que hemos comentado anteriormente, GRU1 (abajo a la izquierda), que tiene un error medio de 0.004512, funciona mejor que el resto, concretamente un 70% mejor que LSTM1, un 54% mejor que LSTM2 y un 32% mejor que GRU2.

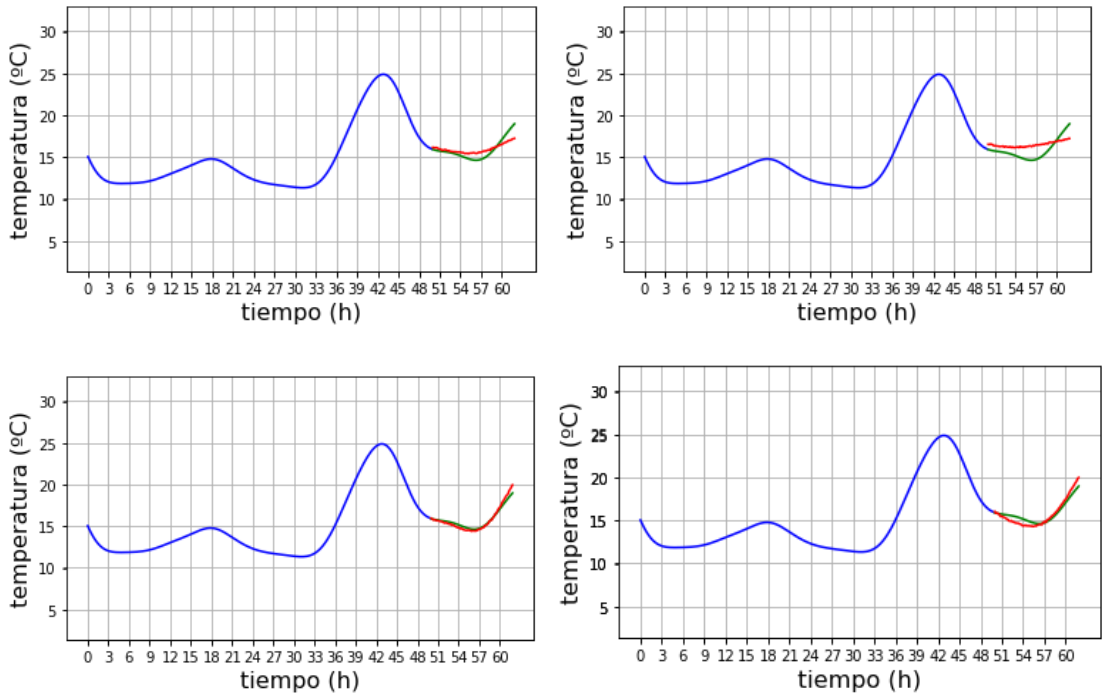


*Figura 46: Predicción de 12 horas en el Perdón mediante todos los modelos excepto RNN.*

En la figura 46 podemos observar que ambos modelos de GRU (GRU1 abajo a la izquierda y GRU2 abajo a la derecha) realizan predicciones similares, siendo GRU1 solo un 24,4 % mejor que GRU2. En comparación de los modelos de LSTM, ambos modelos tienen 2,33 veces más error que el GRU1. En las gráficas se observa que tanto LSTM1 como LSTM2 siguen manteniendo la tendencia de la temperatura, pero de manera mucho menos precisa que hasta ahora.

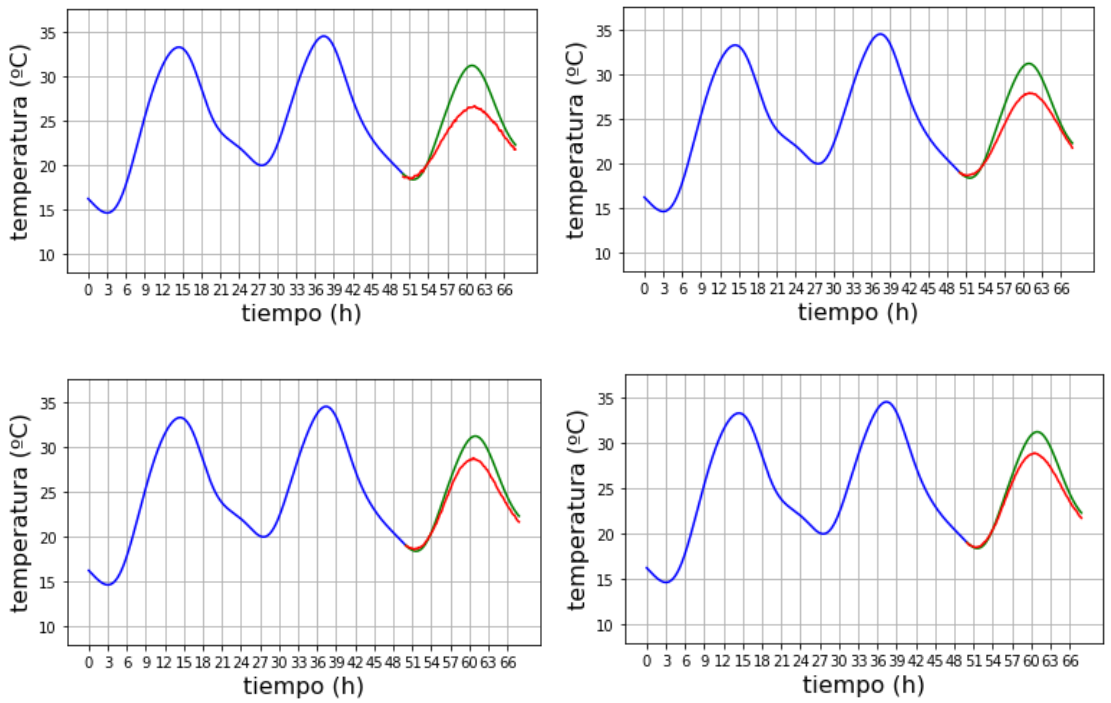


*Figura 47: Predicción de 18 horas en Tafalla mediante todos los modelos excepto RNN.*

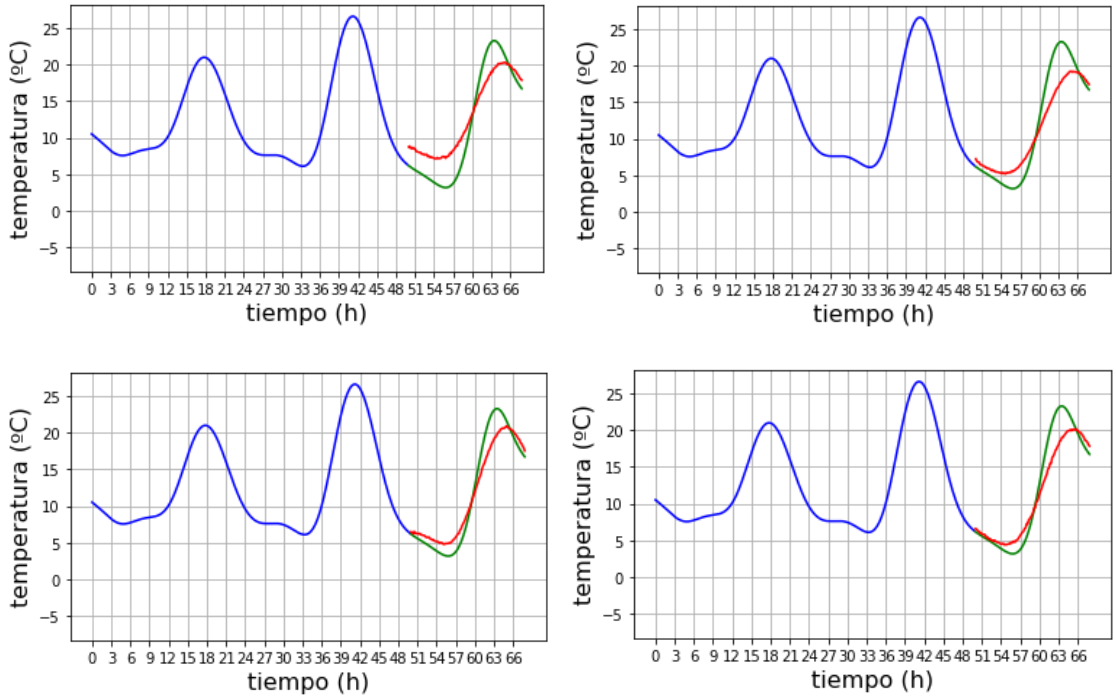


*Figura 48: Predicción de 18 horas en Doneztebe mediante todos los modelos excepto RNN.*

Como se puede observar en las figuras 47 y 48, las series temporales de las temperaturas de Tafalla tienen menos variaciones cíclicas que las de Doneztebe. Esto hace que las predicciones de las LSTM1 no estén tan alejadas de los datos reales en Tafalla, pero las predicciones en Doneztebe son peores. Igualmente, podemos observar que los modelos GRU1 y GRU2, producen resultados mucho mejores. Los errores que produce GRU1 son un 69,6% menores que los que produce LSTM1. Las predicciones de GRU1 siguen siendo las mejores, pero solo un 3% mejores que las de GRU2.

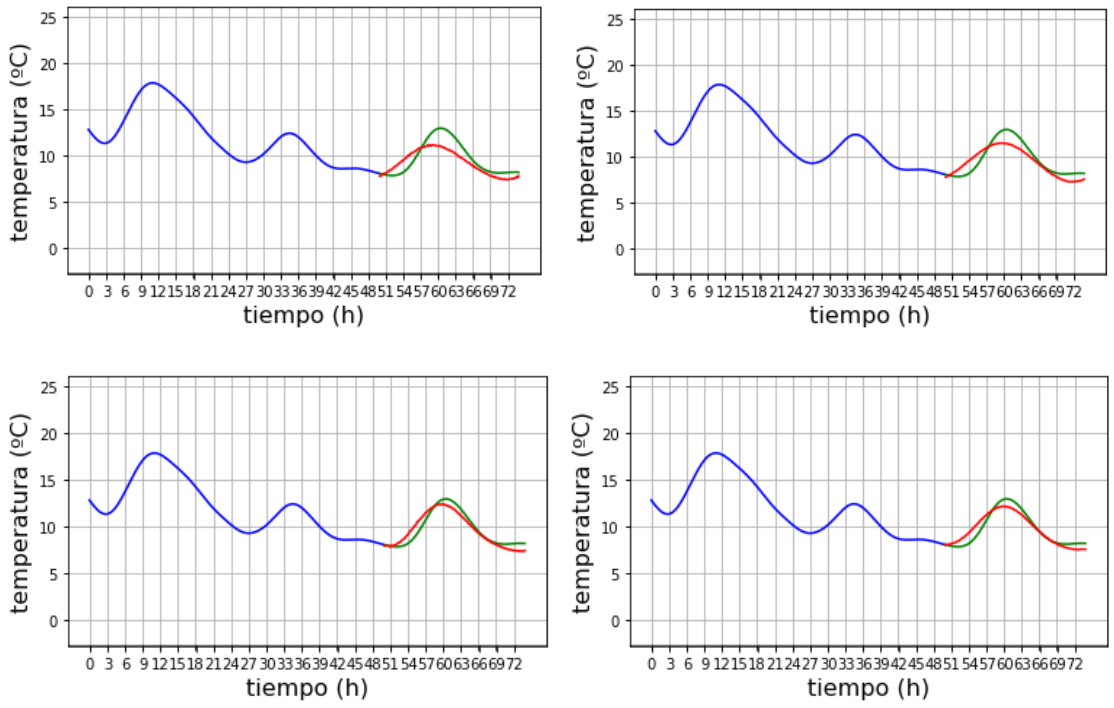


*Figura 49: Predicción de 24 horas en el Perdón mediante todos los modelos excepto RNN.*

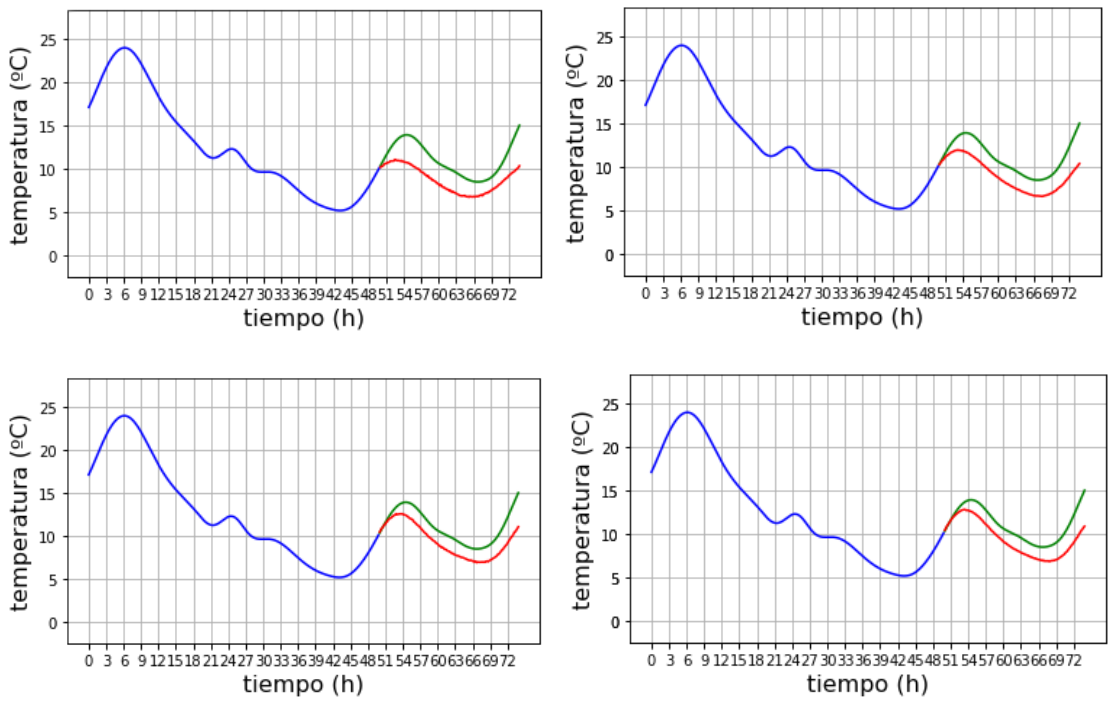


*Figura 50: Predicción de 24 horas en el Ujué mediante todos los modelos excepto RNN.*

En las figuras 49 y 50 se observan las predicciones de un día completo para el Perdón y para Ujué. Podemos observar que tanto LSTM (gráficas de arriba en ambas figuras) como GRU (gráficas de abajo en ambas figuras) producen resultados buenos. GRU sigue produciendo resultados mejores, concretamente, GRU1 tiene un error 43.8% menor que LSTM1. En este caso, GRU1 y GRU2 nos dan resultados muy similares.

## 6.4 Cuarta prueba

Esta es la cuarta y ultima prueba que hemos realizado. Aquí hemos decidido modificar la LSTM1 de la manera que se ha explicado en el apartado 4 para intentar obtener una mejora de las predicciones en comparación con los resultados obtenidos de la tercera prueba. Para ello hemos utilizado las siguientes medidas de disimilitud:

- $\delta_1(x, y) = |x - y|$
- $\delta_2(x, y) = \sqrt{|x^2 - y^2|}$

La primera de ellas es la RDF estándar y correspondea la integral de Choquet usual.

A partir de ahora, a los resultados obtenidos con la LSTM1 modificada con la primera RDF le llamaremos CUS1 y a la modificada con la segunda RDF CUS2.

Los resultados obtenidos los vamos a mostrar junto a los obtenidos por LSTM1 en la tercera prueba y los obtenidos por GRU1, puesto que LSTM1 es la que intentamos mejorar y GRU1 es la que mejores predicciones ha dado hasta el momento.

| DONETZEBE | 1 | 12 | 18 | 24 |
|---|---|---|---|---|
| **CUS1** | **0.000015** | 0.019411 | 0.025445 | 0.025378 |
| **CUS2** | 0.000025 | **0.005369** | **0.009752** | **0.010277** |
| **LSTM1** | 0.011906 | 0.014884 | 0.033355 | 0.039120 |
| **GRU1** | 0.000023 | 0.006988 | 0.012579 | 0.012415 |

*Figura 51: Errores MSE obtenidos para Donetzebe en la cuarta prueba.*

| PERDÓN | 1 | 12 | 18 | 24 |
|---|---|---|---|---|
| **CUS1** | 0.000013 | 0.014904 | 0.017940 | 0.021715 |
| **CUS2** | 0.000028 | **0.005199** | **0.008636** | 0.013118 |
| **LSTM1** | **0.000006** | 0.010146 | 0.029367 | 0.014443 |
| **GRU1** | 0.000019 | 0.005805 | 0.009245 | **0.012108** |

*Figura 52: Errores MSE obtenidos para el Perdón en la cuarta prueba.*

| TAFALLA | 1 | 12 | 18 | 24 |
|---|---|---|---|---|
| **CUS1** | 0.000015 | 0.018430 | 0.018984 | 0.023391 |
| **CUS2** | 0.000032 | **0.004795** | **0.006189** | 0.012695 |
| **LSTM1** | **0.000008** | 0.016528 | 0.028310 | 0.013807 |
| **GRU1** | 0.000022 | 0.005270 | 0.006458 | **0.010337** |

*Figura 53: Errores MSE obtenidos para Tafalla en la cuarta prueba.*

| UJUÉ | 1 | 12 | 18 | 24 |
|---|---|---|---|---|
| **CUS1** | 0.000013 | 0.015594 | 0.017882 | 0.021311 |
| **CUS2** | 0.000030 | **0.004031** | **0.007979** | 0.012072 |
| **LSTM1** | **0.000006** | 0.011263 | 0.029377 | 0.013436 |
| **GRU1** | 0.000020 | 0.004579 | 0.008343 | **0.010553** |

*Figura 54: Errores MSE obtenidos para Ujué en la cuarta prueba.*

Como podemos observar en las tablas, CUS1 da resultados parecidos a LSTM1 y CUS2 mejora los resultados de LSTM1 en predicciones largas, e incluso CUS2 mejora las de GRU1 excepto cuando predecimos 24 horas en el Perdón, Tafalla y Ujué. En todos los casos, las predicciones de GRU1 y CUS2 son muy similares, veámoslo en las gráficas.

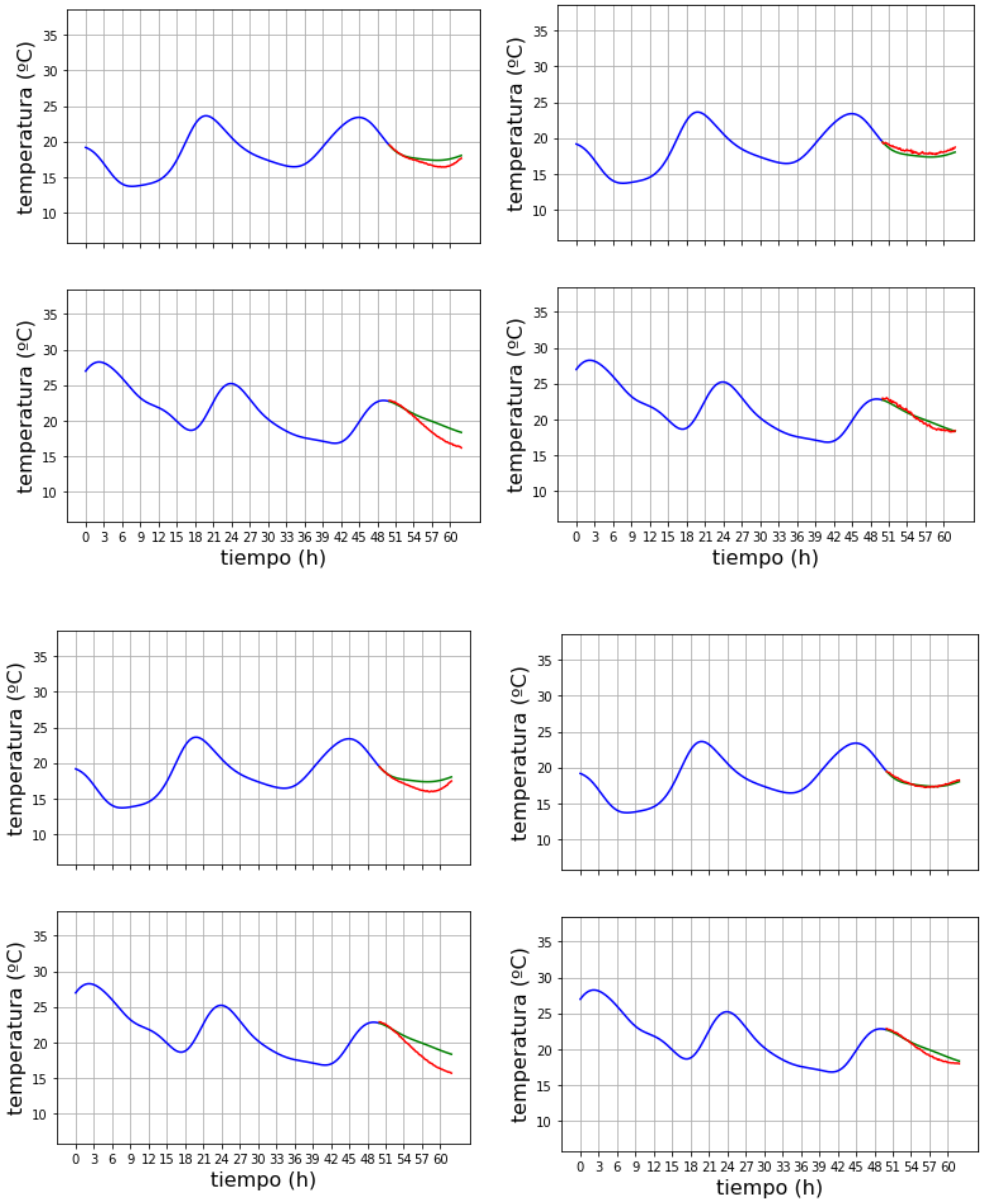


*Figura 55: Predicción de 12 horas en Doneztebe mediante los cuatro modelos.*

En la figura 55, tenemos arriba a la izquierda CUS1, a la derecha CUS2, abajo a la izquierda LSTM1 y abajo a la derecha GRU1. CUS2 es la que mejores resultados obtiene, especialmente mejorando LSTM1, dando un error un 64% menor. En este caso, las predicciones de CUS1 son un 14% mejores que las de GRU1.

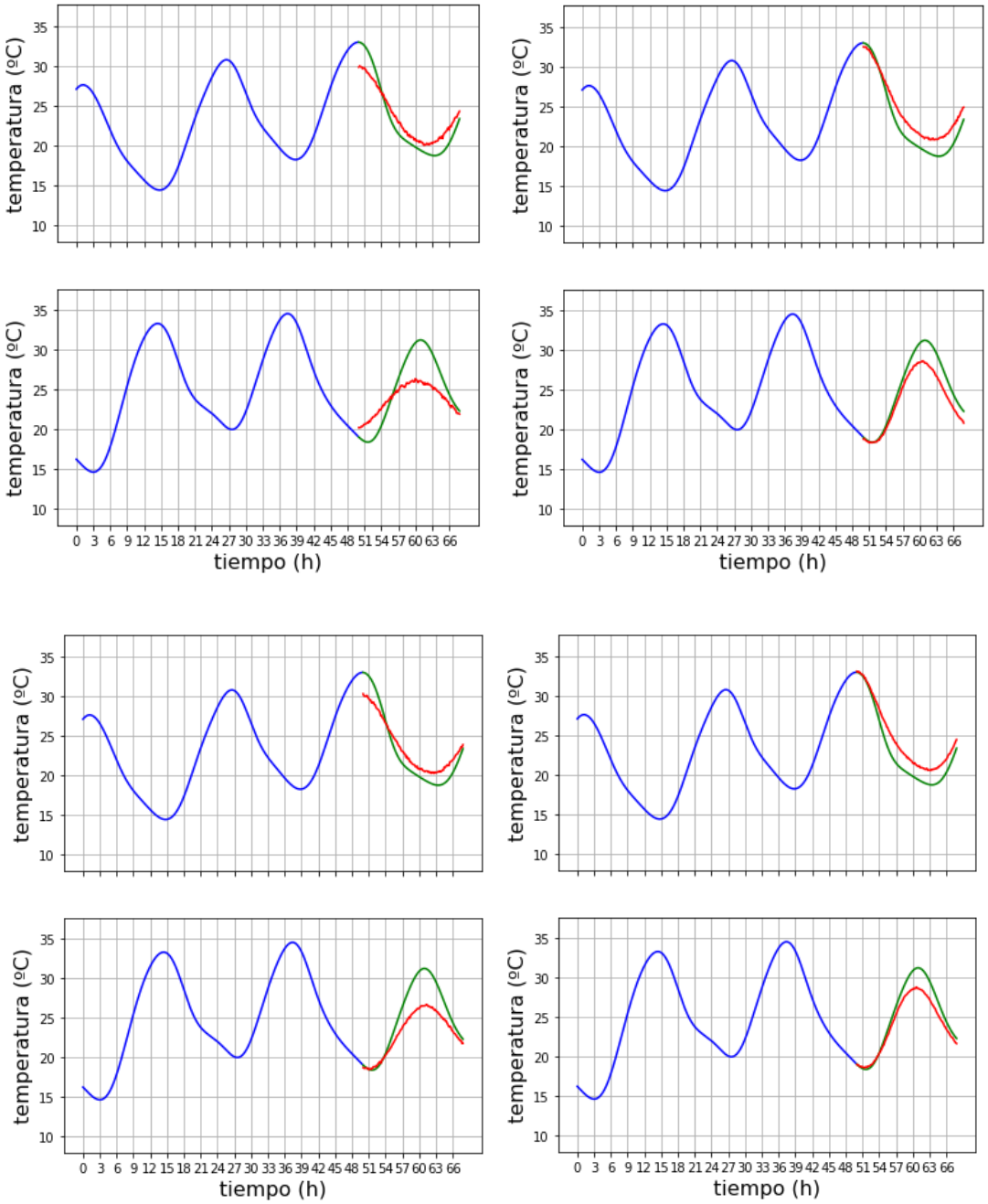


*Figura 56: Predicción de 18 horas en Tafalla mediante los cuatro modelos.*

Como se puede observar en la figura 56, los modelos CUS2 y GRU1 (gráficas de la derecha) dan resultados muy parecidos, mientras que CUS1 y LSTM1 dan resultados peores, concretamente,el error de LSTM1 es 3,7 veces mayor que el de CUS2 y el de CUS1 es 2,46 veces mayor que el de CUS2.

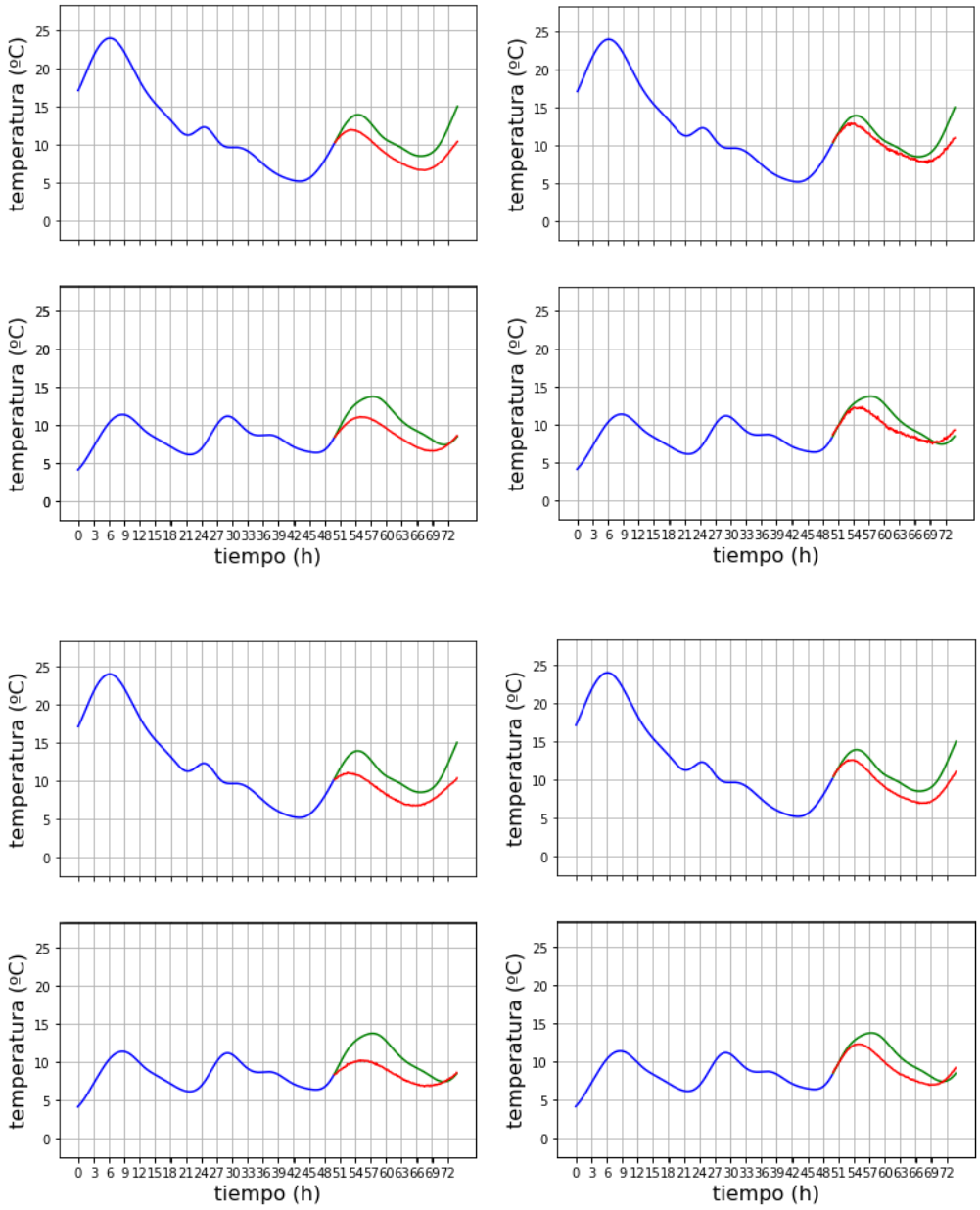


*Figura 57: Predicción de 24 horas en Ujué mediante los cuatro modelos.*

En la figura 57 podemos observar las predicciones para Ujué de 24 horas. En este caso, el modelo que mejores resultados obtiene es GRU1, pero son muy parecidos a CUS2, solo un 5,8% mejores. Igualmente, CUS2 sigue siendo mucho mejor que LSTM1 y CUS1, concretamente, el error obtenido con LSTM1 es un 67% mayor que CUS2 y el error obtenido con CUS1 es casi el doble que el obtenido con CUS2, un 90% mayor.

# 7. Conclusiones

En este apartado se van a exponer las conclusiones obtenidas a raíz del trabajo realizado.

Con las pruebas realizadas anteriormente hemos podido observar el problema del gradiete explosivo que tienen las RNN clásicas y que no nos resultan útiles para realizar predicciones cuando aumentan el numero de valores a predecir.

También obsevamos que, tanto con LSTM como con GRU, cuanto más aumentamos el numero de valores a predecir, más se desvían las predicciones de los valores reales, aun así podemos observar que la tendencia de la serie predicha es bastante correcta.

Con los resultados obtenidos en la tercera prueba realizada, podemos decir que las predicciones obtenidas son lo suficientemente correctas puesto que nos servirían en la vida cotidiana para saber que tendencia va a llevar la temperatura a lo largo del día teniendo en cuenta un error de un par de grados en todo momento.

Con la cuarta prueba nos damos cuenta de que con modificaciones en la LSTM original podemos obtener datos igual de precisos que con GRU, aun así, GRU funciona mejor que LSTM en predicciones a largo plazo. También un problema que han tenido las LSTM modificadas es que la ejecución era mucho mas larga en el tiempo que las de GRU o las de la LSTM sin modificar, esto es debido al aumento de operaciones que tiene que realizar.

# 8. Bibliografía


[1] Yunus A, Çengel (2009). *Temodinámica, 6ta edición*. Mc Graw Hill.

[2] Pena, D., Tiao, G. C., & Tsay, R. S. (2011). *A course in time series analysis*. John Wiley & Sons.

[3] Mitchell, T. (1997). *Machine Learning*, McGraw Hill.

[4] Trask, A. W. (2019). *Grokking deep learning*. Simon and Schuster.

[5] Villanueva García, J. D. (2020). *Redes neuronales desde cero (I) – Introducción*. https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/

[6] Karpathy, A. (2015). *The unreasonable effectiveness of recurrent neural networks*. Andrej Karpathy blog, 21, 23.

[7] Olah, C. (2015). *Understanding lstm networks*.

[8] Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. *Neural computation*, 9(8), 1735-1780.

[9] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning phrase representations using RNN encoder-decoder for statistical machine translation*.

[10] Beliakov, G., Bustince, H., & Calvo, T. (2016). *A practical guide to averaging functions*. Switzerland: Springer International Publishing.

[11] Choquet, G. (1954). *Theory of capacities. Annales de l'institut Fourier* (Vol. 5, pp. 131-295).

[12] Bustince, H., Barrenechea, E., & Pagola, M. (2008). Relationship between restricted dissimilarity functions, restricted equivalence functions and normal EN-functions: Image thresholding invariant. *Pattern Recognition Letters*, 29(4), 525-536.

[13] Bustince, H., Mesiar, R., Fernández, J., Galar, M., Paternain, D., Altalhi, A., ... & Takáč, Z. (2021). d-Choquet integrals: Choquet integrals based on dissimilarities. *Fuzzy Sets and Systems*, *414*, 1-27.

[14] Del Rio, A. (1999). *Agregación temporal y filtro Hodrick-Prescott*. Centro de Estudios Monetarios y Financieros.