Universidad de Cantabria

Facultad de Ciencias

Aprendizaje profundo y computación de reservorio para abordar problemas de regresión:

Aplicaciones en el sector primario

(Deep Learning and Reservoir Computing for Tackling Regression Problems: Applications in the Primary Sector)

Trabajo de Fin de Grado

para acceder al

GRADO EN MATEMÁTICAS

Autor: Iñigo Altuna
Director: Ángel Cobo Ortega
Fecha: Junio - 2025

Índice

1 Introducción

1.1 Resumen

En este Trabajo de Fin de Grado (TFG) se aborda un problema de gran interés práctico como es la predicción de la evolución de los precios de productos agropecuarios. Para ello se hace uso de diferentes modelos computacionales basados en redes neuronales y se aplicarán a la predicción de precios de productos agrícolas (aguacate) y del sector de acuicultura (dorada y salmón). En el trabajo se plantea el uso de técnicas neuronales de aprendizaje profundo y computación de reservorio para abordar los problemas propuestos. Se presentarán los fundamentos de estos enfoques, destacando su capacidad para modelar relaciones complejas en series temporales.

El estudio incluirá un análisis comparativo entre modelos de redes neuronales profundas, como redes recurrentes (RNN, LSTM, GRU), y la computación de reservorio mediante Redes de Estado de Eco (ESN), enfatizando la eficiencia de estas últimas en el procesamiento de datos secuenciales con un menor costo computacional.

Para dicho análisis, se emplearán series de datos temporales reales relacionadas con productos pesqueros y agrícolas. Finalmente, se discutirán las ventajas y desafios de la computación de reservorio frente a otros enfoques de aprendizaje profundo, destacando su eficiencia computacional, facilidad de implementación y potencial para abordar problemas de regresión en distintos ámbitos.

1.2 Summary (English version)

This Final Degree Project (TFG) addresses a problem of significant practical interest: predicting the evolution of prices for agricultural and aquaculture products. To this end, various computational models based on neural networks are employed and applied to price forecasting for agricultural products (avocado) and aquaculture products (gilt-head bream and salmon). The work proposes the use of deep learning techniques and reservoir computing to tackle the proposed problems. The foundations of these approaches are presented, highlighting their ability to model complex relationships in time series data.

The study includes a comparative analysis between deep neural network models, such as recurrent neural networks (RNN, LSTM, GRU), and reservoir computing through Echo State Networks (ESN), emphasizing the efficiency of the latter in processing sequential data with lower computational cost.

For this analysis, real-world time series data related to agricultural and aquaculture products are used. Finally, the advantages and challenges of reservoir computing are discussed in comparison to other deep learning approaches, emphasizing its computational efficiency, ease of implementation, and potential to address regression problems in various domains.

1.3 Motivación del trabajo

El presente Trabajo de Fin de Grado surge del interés por aplicar técnicas avanzadas de inteligencia artificial a problemas reales del mundo económico y productivo. En particular, el uso del aprendizaje profundo y la computación de reservorio en el ámbito de las series temporales representa una línea de investigación con un alto potencial de impacto, especialmente en sectores como el agrícola y pesquero, donde la toma de decisiones basada en datos es cada vez más importante.

Además del atractivo técnico y académico del tema, esta línea de trabajo representa también una proyección directa hacia mi desarrollo profesional. Considero que el análisis de datos y la modelización predictiva serán pilares fundamentales en los próximos años dentro de los sectores tecnológicos, logísticos y agroindustriales. Por ello, me resulta especialmente motivador desarrollar un proyecto en el que confluyen la innovación matemática, la inteligencia artificial y la aplicación práctica.

Este trabajo me permite profundizar en el uso de herramientas y modelos que forman parte del núcleo de lo que me gustaría hacer profesionalmente, combinando la investigación aplicada con el desarrollo de soluciones basadas en datos reales.

1.4 Descripción del problema

El sector primario, en particular los ámbitos agrícola y acuícola, depende cada vez más del análisis de datos para optimizar procesos y tomar decisiones informadas. Uno de los aspectos clave en esta transformación digital es la **predicción de precios**, una tarea fundamental para productores, distribuidores y entidades reguladoras. Sin embargo, la evolución de precios en productos agropecuarios presenta una alta complejidad debido a factores estacionales, logísticos, económicos y climáticos.

El objetivo principal de este trabajo es **abordar la predicción de precios en productos del sector primario**, tanto agrícolas como acuícolas, mediante el uso de técnicas de inteligencia artificial aplicadas a series temporales. En concreto, se analizarán series históricas de precios correspondientes a **aguacate** (producto agrícola), **dorada** y **salmón** (productos acuícolas), todos ellos representativos de mercados con dinámicas propias y desafíos predictivos distintos.

El enfoque metodológico se centra en comparar diferentes arquitecturas de redes neuronales, evaluando su capacidad predictiva y su eficiencia computacional. De forma particular, el objetivo central es probar el rendimiento de la **computación de reservorio**, y en especial de las *Echo State Networks* (ESN). Estas redes, a diferencia de las redes neuronales profundas tradicionales (como MLP, LSTM o GRU), no requieren el entrenamiento de todo su conjunto de parámetros, sino únicamente de la capa de salida, lo que reduce significativamente el coste computacional.

Esto plantea la siguiente pregunta de investigación:

¿Es posible obtener un rendimiento predictivo comparable al de las

redes neuronales profundas tradicionales utilizando computación de reservorio?

Para responder a esta cuestión, se han utilizado **series reales de precios de productos agrícolas y acuícolas**, proporcionadas por diversas fuentes nacionales e internacionales, y analizadas en colaboración con el grupo de investigación en Gestión Económica Sostenible del Sector Primario de la Universidad de Cantabria.

2 Marco teórico

2.1 Definición formal de serie temporal

En el contexto del aprendizaje automático para la predicción de dinámicas complejas, una **serie temporal** es una secuencia ordenada de observaciones tomadas a intervalos de tiempo discretos o continuos. Formalmente, se define como una función:

$$\mathcal{U} = \{u(t) \in \mathbb{R}^D \mid t \in \mathbb{T} \subset \mathbb{R}\}, \tag{1}$$

donde:

- $u(t)$ representa el estado del sistema observado en el instante t,
- D es la dimensión del espacio de observación (puede ser univariada $D = 1$ o multivariada $D > 1$),
- $\mathbb{T}$ es el conjunto de tiempos en el cual se registran las observaciones, generalmente uniforme (e.g., $\mathbb{T} = \{t_0, t_1, \ldots, t_N\}$ con $t_{i+1} - t_i = \Delta t$ constante).

Las series temporales pueden derivar de un sistema dinámico subyacente, cuya evolución sigue una ley determinista (aunque desconocida), como:

$$\dot{u}(t) = f(u(t)), \tag{2}$$

donde f representa la dinámica del sistema. El objetivo de los modelos predictivos es aproximar esta evolución para estimar futuros estados $u(t + \tau)$ a partir del historial conocido $\{u(t), u(t-1), \ldots\}$.

2.2 Redes Neuronales

Una red neuronal es un modelo computacional inspirado en el funcionamiento del cerebro humano, que está compuesto por un conjunto de nodos (neuronas artificiales) organizados en capas. Estos modelos son capaces de reconocer patrones y aprender de datos, de manera que son uno de los elementos esenciales del campo del aprendizaje automático (machine learning) y la base de muchas aplicaciones de inteligencia artificial (IA) modernas.

2.2.1 La Neurona

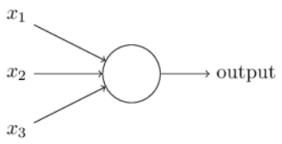

Figure 1: Esquema básico de una neurona artificial. Adaptado de [11].

Una *neurona* es una unidad de procesamiento que recibe valores de entrada $x_1, x_2, x_3, \ldots$ a los cuales les asocia unos pesos $w_1, w_2, w_3, \ldots$, y realiza una suma ponderada de sus valores de entrada para generar una salida a partir de una función de activación. La salida de la neurona se calcula como:

$$y = F(X) = F(w_1x_1 + w_2x_2 + \cdots + w_nx_n)$$

Donde $F(X)$ es la función de activación (en un caso sencillo, podría ser la función identidad, devolviendo en ese caso una suma lineal). Además, suele incluirse un *sesgo* b asociado a una variable de valor 1, que se añade para ajustar el resultado final:

$$y = F(X) = F(b + w_1x_1 + w_2x_2 + \cdots + w_nx_n)$$

En una neurona, las variables $x_1, x_2, \ldots, x_n$ pueden tomar valores reales en determinados intervalos o ser incluso valores binarios (0 o 1). La salida obtenida mediante la función de activación a partir de los valores de entrada y sus pesos puede ser también un valor real o binario dependiendo de si la salida cumple un cierto umbral, como $F(X) > 0$.

2.2.2 La Red Neuronal

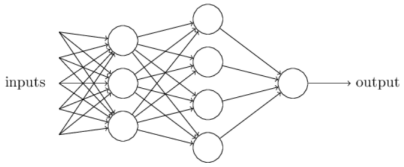

Figure 2: Estructura típica de una red neuronal multicapa. Fuente: [11].

Una *red neuronal* está formada por diferentes capas de neuronas: la capa de entrada, la capa de salida y las capas ocultas. Las capas intermedias son conocidas como *capas ocultas*. Las neuronas de una misma capa reciben los mismos datos de entrada, y cada una de ellas proporciona su salida a la siguiente capa.

Cada capa tiene su propia función de activación, que transforma la salida de las neuronas. Un modelo típico de red neuronal profunda puede ser representado como:

$$y^{(l)} = f^{(l)}\left(W^{(l)}x^{(l-1)} + b^{(l)}\right)$$

- $y^{(l)}$: salida de la capa l,
- $f^{(l)}$: función de activación en la capa l,
- $W^{(l)}$: matriz de pesos de la capa l,
- $x^{(l-1)}$: salida de la capa anterior (o la entrada para la primera capa),
- $b^{(l)}$: sesgo de la capa l.

Este modelo se utiliza para aprender a modelar relaciones complejas en los datos de entrada, como en el caso de la predicción de precios, donde las variables de entrada pueden ser precios pasados, características del mercado, entre otros factores.

2.3 Perceptrón Multicapa (MLP)

El **Perceptrón Multicapa** (MLP, *Multi-Layer Perceptron*) es una red feedforward compuesta por capas densamente conectadas. Para la predicción de series temporales se suele utilizar una *ventana deslizante* de tamaño S: dado el vector de entrada

$$\mathbf{x} = \left[x_{t-S+1},\, x_{t-S+2},\, \ldots,\, x_t\right]^\top \in \mathbb{R}^S,$$

el MLP estima la predicción $\hat{y}_{t+1} \in \mathbb{R}$ mediante varias transformaciones lineales y no lineales:[11]

$$\begin{aligned}
\mathbf{h}^{(1)} &= \phi\big(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\big),\\
\mathbf{h}^{(2)} &= \phi\big(W^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\big),\\
&\vdots\\
\hat{y}_{t+1} &= \phi\big(W^{(L)}\mathbf{h}^{(L-1)} + b^{(L)}\big),
\end{aligned}$$

donde:

- $W^{(\ell)}$ y $\mathbf{b}^{(\ell)}$ son los pesos y sesgos respectivamente de la capa ℓ,
- $\phi\colon \mathbb{R} \to \mathbb{R}$ es una función de activación (p. ej. $\mathrm{ReLU}(z) = \max\{0, z\}$ o $\tanh(z)$).
- L es el número total de capas (contando la capa de salida como la L-ésima),
- $\mathbf{h}^{(\ell)}$ denota la salida intermedia de la capa ℓ.

El entrenamiento del MLP, consistente en determinar los valores de los pesos, se realiza minimizando un *error cuadrático medio* regularizado, por ejemplo:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda \sum_{\ell=1}^{L}\|W^{(\ell)}\|_F^2,$$

donde $\lambda > 0$ es el coeficiente de *regularización de Tikhonov* y $\|\cdot\|_F$ la norma de Frobenius:

$$\|A\|_F \;=\; \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2}\,.$$

donde

$$A = [a_{ij}] \in \mathbb{R}^{m\times n}$$

Aunque el MLP puede aproximar funciones no lineales complejas, no incorpora explícitamente memoria de largo plazo, por lo que suele necesitar ventanas de entrada relativamente amplias y un número elevado de parámetros para modelar dependencia temporal.

2.4 Redes Neuronales Recurrentes (RNN)

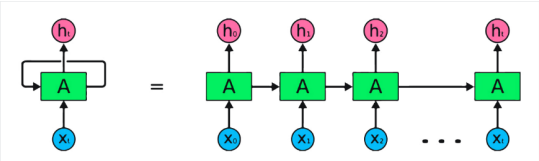

Figure 3: Esquema básico de una red neuronal recurrente (RNN). Fuente: [4].

Las Redes Neuronales Recurrentes (RNN, por sus siglas en inglés) son una clase de modelos diseñados para procesar datos secuenciales, como series temporales, texto o señales, donde el orden y la dependencia temporal entre los elementos son fundamentales. A diferencia de las redes neuronales feedforward, las RNN incorporan conexiones recurrentes que permiten mantener una "memoria" del estado anterior, facilitando el modelado de dependencias temporales.

2.4.1 Definición formal

Una RNN procesa una secuencia de entradas $\{x_1, x_2, \ldots, x_T\}$, donde cada $x_t \in \mathbb{R}^n$, generando una secuencia de salidas $\{y_1, y_2, \ldots, y_T\}$, con $y_t \in \mathbb{R}^m$. En cada paso temporal t, la red actualiza su estado oculto $h_t \in \mathbb{R}^d$ mediante las siguientes ecuaciones:[4]

$$h_t = \phi(W_{xh}x_t + W_{hh}h_{t-1} + b_h),$$
$$y_t = \psi(W_{hy}h_t + b_y),$$

donde:

- $W_{xh} \in \mathbb{R}^{d \times n}$ es la matriz de pesos que conecta la entrada con el estado oculto,
- $W_{hh} \in \mathbb{R}^{d \times d}$ es la matriz de pesos recurrentes que conecta el estado oculto anterior con el actual,
- $W_{hy} \in \mathbb{R}^{m \times d}$ es la matriz de pesos que conecta el estado oculto con la salida,
- $b_h \in \mathbb{R}^d$ y $b_y \in \mathbb{R}^m$ son los vectores de sesgo para el estado oculto y la salida, respectivamente,
- ϕ y ψ son funciones de activación, comúnmente no lineales, como la tangente hiperbólica o la función sinoidal.

Esta estructura permite que la red mantenga información sobre entradas anteriores a través del estado oculto h_t, lo que es esencial para tareas donde el contexto temporal es relevante.

2.4.2 Entrenamiento de RNN

El entrenamiento de una RNN se realiza mediante el algoritmo de retropropagación a través del tiempo (Backpropagation Through Time, BPTT), que consiste en desplegar la red a lo largo de los pasos temporales y aplicar el algoritmo de retropropagación estándar para ajustar los pesos y sesgos en función del error cometido en las predicciones.

2.4.3 Limitaciones y variantes

Las RNN tradicionales presentan dificultades para aprender dependencias a largo plazo debido al problema del *desvanecimiento* o *explosión* del gradiente durante el entrenamiento. Para mitigar estos problemas, se han desarrollado variantes como las redes de memoria a largo plazo (LSTM) y las unidades recurrentes con compuertas (GRU), que introducen mecanismos de control adicionales para preservar la información relevante a lo largo del tiempo.

2.4.4 Problemas del desvanecimiento y la explosión del gradiente

Durante el entrenamiento de redes neuronales recurrentes (RNN), se emplea el algoritmo de retropropagación a través del tiempo (*Backpropagation Through Time*, BPTT), que se ha mencionado previamente. En este proceso, los gradientes se calculan como productos sucesivos de derivadas a lo largo del tiempo, lo cual puede generar inestabilidades numéricas.

Este fenómeno se manifiesta de dos formas:

- **Desvanecimiento del gradiente:** cuando los valores propios de las matrices involucradas (por ejemplo, W_{hh}) tienen módulo menor que uno, los gradientes decrecen exponencialmente conforme se retrocede en el tiempo.

Como consecuencia, las actualizaciones de los parámetros asociados a pasos lejanos se vuelven insignificantes, lo que impide a la red aprender dependencias de largo plazo.

- **Explosión del gradiente:** si los valores propios son mayores que uno, los gradientes aumentan exponencialmente con cada paso hacia atrás, generando inestabilidad numérica y dificultando la convergencia del entrenamiento. Esto puede dar lugar a actualizaciones erráticas y divergencia del modelo.

Ambos problemas están estrechamente relacionados con la dinámica de los sistemas utilizados y con la sensibilidad del error a las condiciones iniciales. Estas limitaciones han motivado el desarrollo de arquitecturas recurrentes avanzadas, como las redes LSTM y GRU, que incorporan mecanismos de compuertas diseñados para regular el flujo de gradientes y preservar la información relevante a lo largo del tiempo.

2.5 Redes LSTM y GRU

Las redes neuronales recurrentes clásicas (RNN) presentan dificultades para capturar dependencias de largo plazo debido a los problemas de *desvanecimiento* y *explosión del gradiente*. Para superar esta limitación, se han desarrollado arquitecturas que incorporan mecanismos de memoria controlada. Entre las más representativas se encuentran las redes **LSTM** (Long Short-Term Memory) y las **GRU** (Gated Recurrent Unit).

Estudios recientes han demostrado la eficacia de modelos recurrentes como LSTM y GRU para la predicción de variables agrícolas, como la producción nacional de leche [2]. Estos enfoques, sin embargo, implican un coste computacional elevado y requieren conjuntos de datos extensos, lo cual motiva la exploración de alternativas más eficientes como las ESN.

2.5.1 LSTM: Long Short-Term Memory

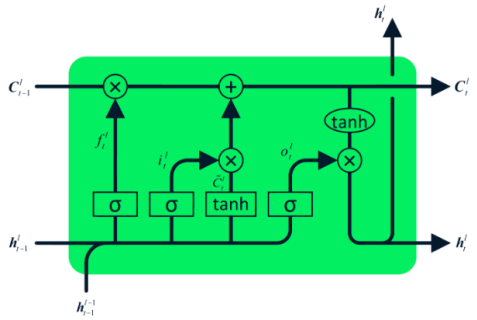

Figure 4: Esquema interno de una unidad LSTM. Fuente: [4].

Propuestas por Hochreiter y Schmidhuber (1997)[6], las redes LSTM incorporan un vector de *estado de celda* c_t que se actualiza mediante operaciones controladas por compuertas. Esto permite retener información durante intervalos largos sin que los gradientes se anulen.

La dinámica interna de una celda LSTM se define por el siguiente sistema:[14]

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{4}$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{5}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{6}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{7}$$
$$h_t = o_t \odot \tanh(c_t) \tag{8}$$

donde:

- f_t: compuerta de olvido,
- i_t: compuerta de entrada,
- $\tilde{c}_t$: contenido candidato para el nuevo estado de celda,
- c_t: estado interno actualizado de la celda,
- o_t: compuerta de salida,
- h_t: nuevo estado oculto,
- σ: función sigmoide logística, $\sigma(x) = \frac{1}{1+e^{-x}}$
- $\odot$: producto elemento a elemento (Hadamard).

2.5.2 GRU: Gated Recurrent Unit

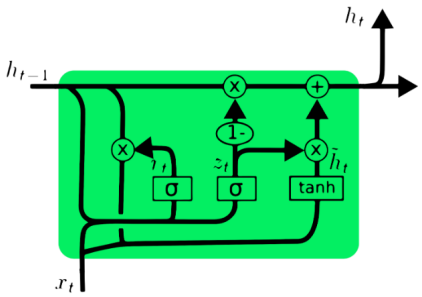

Figure 5: Esquema interno de una unidad GRU. Fuente: [4].

Las redes GRU fueron introducidas como una alternativa más simple a las LSTM, manteniendo capacidad para gestionar dependencias de largo plazo, pero con menos parámetros. Las GRU eliminan el estado de celda explícito y combinan las funciones de olvido y entrada en una única compuerta de actualización.

La GRU fue introducida por Cho et al. (2014) como una variante más simple al LSTM. [3]

La dinámica interna de una GRU está definida por:[13]

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{9}$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{10}$$
$$\tilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{11}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{12}$$

donde:

- z_t: compuerta de actualización.
- r_t: compuerta de reinicio.
- $\tilde{h}_t$: nuevo estado candidato.
- h_t: estado oculto actualizado.

Gracias a su menor complejidad estructural, las GRU suelen requerir menos tiempo de entrenamiento que las LSTM, y en muchos casos obtienen un rendimiento comparable.

2.6 Computación de reservorio: Echo State Networks (ESN)

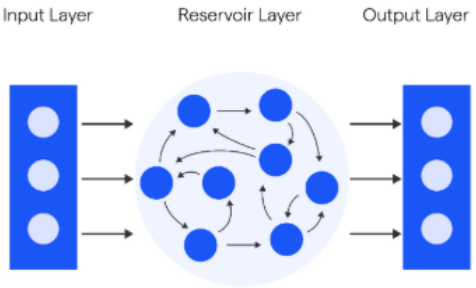

Figure 6: Esquema representativo de una red de tipo Echo State Network (ESN). Fuente: [7].

La **computación de reservorio** [12] es una familia de modelos recurrentes simplificados que permiten procesar datos secuenciales con un coste computacional significativamente menor que el de las redes RNN clásicas. Entre las arquitecturas más representativas se encuentran las **Redes de Estado de Eco** (*Echo State Networks*, ESN), introducidas por Herbert Jaeger en 2001.[8]

Diversos estudios recientes han resaltado la versatilidad de la computación de reservorio, extendiéndose a dominios tan variados como la robótica, la neurociencia o la computación física [5].

La idea principal de las ESN consiste en mantener fijo el núcleo recurrente de la red —denominado *reservorio*— sin necesidad de entrenar sus pesos internos. Solo se entrena la capa de salida, lo que convierte a las ESN en modelos lineales en los parámetros y, por tanto, mucho más eficientes de ajustar.

2.6.1 Estructura del modelo

El modelo ESN está compuesto por tres partes:

- Una capa de entrada de dimensión N_u,
- Un reservorio recurrente de dimensión N_r,
- Una capa de salida de dimensión N_y.

A cada instante t, el estado interno del reservorio $\mathbf{x}(t) \in \mathbb{R}^{N_r}$ evoluciona según:[7]

$$\mathbf{x}(t) = \tanh\left(W_{\text{in}}\mathbf{u}(t) + W\mathbf{x}(t-1) + \mathbf{b}\right), \tag{13}$$

donde:

- $\mathbf{u}(t) \in \mathbb{R}^{N_u}$ es el vector de entrada en el instante t,

- $W_{\text{in}} \in \mathbb{R}^{N_r \times N_u}$ es la matriz de pesos de entrada,
- $W \in \mathbb{R}^{N_r \times N_r}$ es la matriz de conexión interna del reservorio,
- $\mathbf{b} \in \mathbb{R}^{N_r}$ es un vector de sesgo,
- **tanh** es la función de activación.

La elección de funciones de activación no lineales en redes tipo ESN influye significativamente en su capacidad de predicción, especialmente en tareas complejas, como demuestra Binas et al. (2022) [1].
El vector de salida $\mathbf{y}(t) \in \mathbb{R}^{N_y}$ se calcula como:

$$\mathbf{y}(t) = W_{\text{out}}\mathbf{x}(t), \tag{14}$$

donde $W_{\text{out}} \in \mathbb{R}^{N_y \times N_r}$ es la única matriz de pesos que se entrena, generalmente mediante regresión lineal regularizada.

2.6.2 Condición de estado de eco

Para que el modelo funcione correctamente, es fundamental que el reservorio cumpla la **condición de estado de eco** (*Echo State Property*). Esta propiedad garantiza que el estado $\mathbf{x}(t)$ depende únicamente del historial de entradas y no del estado inicial. Una condición suficiente (aunque no necesaria) es que el **radio espectral** de la matriz W sea menor que uno:

$$\rho(W) < 1, \tag{15}$$

donde $\rho(W)$ denota el mayor valor absoluto de los autovalores de W.

2.6.3 Ventajas de las ESN

- El entrenamiento es rápido y escalable, al requerir solo una regresión lineal.
- Evita los problemas de desvanecimiento/explosión del gradiente.
- Ofrece una buena capacidad de modelado de series temporales no lineales.
- Ideal para entornos donde el coste computacional es un factor crítico.

En este trabajo se emplearán redes ESN para abordar la predicción de series temporales reales del sector primario, y se comparará su rendimiento con modelos más tradicionales del aprendizaje profundo.

2.7 Regresión lineal regularizada

La **regresión lineal regularizada**, también conocida como *Ridge Regression* o *regularización de Tikhonov*, es una técnica fundamental para evitar el sobreajuste en modelos lineales, especialmente útil cuando existe multicolinealidad entre las variables, o cuando el número de muestras disponibles es similar o menor al número de parámetros a ajustar —una situación común en modelos con gran dimensionalidad, como las redes de reservorio..

En el contexto de las redes de computación de reservorio, esta técnica se utiliza para entrenar únicamente la matriz de pesos de la capa de salida W_{out}, manteniendo fijos los pesos del reservorio. Esta elección permite formular el problema como una regresión lineal sobre un conjunto de características no lineales generadas por el reservorio.

2.7.1 Formulación matemática

Sea $\mathbf{X} \in \mathbb{R}^{N_r \times T}$ la matriz cuyas columnas son los estados del reservorio para cada instante de tiempo $t = 1, \ldots, T$, y $\mathbf{Y} \in \mathbb{R}^{N_y \times T}$ la matriz de salidas deseadas correspondientes. El objetivo es encontrar la matriz de pesos $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_y \times N_r}$ que minimice la función de coste regularizada:

$$\mathcal{L}(W_{\text{out}}) = \|\mathbf{Y} - W_{\text{out}}\mathbf{X}\|_F^2 + \lambda \|W_{\text{out}}\|_F^2 , \tag{16}$$

donde:

- $\|\cdot\|_F$ denota la norma de Frobenius,
- $\lambda > 0$ es el parámetro de regularización que controla el compromiso entre error de ajuste y complejidad del modelo.

2.7.2 Solución analítica

La minimización de esta función cuadrática admite una solución cerrada, dada por:

$$\mathbf{W}_{\text{out}} = \mathbf{Y}\mathbf{X}^\top \left(\mathbf{X}\mathbf{X}^\top + \lambda I\right)^{-1} , \tag{17}$$

donde I es la matriz identidad de dimensión $N_r \times N_r$.

2.7.3 Ventajas

Este enfoque presenta múltiples ventajas:

- La solución es explícita, no requiere métodos iterativos ni gradientes.
- Es computacionalmente eficiente.
- La regularización permite controlar la complejidad y mejora la generalización.

La regresión lineal regularizada constituye una parte esencial del entrenamiento en las redes ESN, donde el reservorio actúa como una transformación no lineal del espacio de entrada, y la salida se ajusta con esta técnica lineal eficiente.

2.8 Predicción de un solo paso y multipaso

En el ámbito de las series temporales, una distinción importante en los métodos de predicción es el horizonte temporal que se busca estimar. Existen principalmente dos enfoques:

2.8.1 Predicción de un solo paso

En este enfoque, el modelo predice únicamente el valor en el instante siguiente al último observado. Es decir, dados los valores $x_{t-S+1}, x_{t-S+2}, \ldots, x_t$, el objetivo es predecir x_{t+1}. Formalmente, el modelo usa una función:

$$f : \mathbb{R}^S \to \mathbb{R}, \quad \hat{x}_{t+1} = f(x_{t-S+1}, \ldots, x_t)$$

Este tipo de predicción es más estable, ya que el modelo siempre opera sobre datos reales observados. En el contexto de redes neuronales, se suele entrenar minimizando el error (de distintos tipos) entre la predicción y el valor real siguiente.

2.8.2 Predicción multipaso

En la predicción multipaso, se estima una secuencia de valores futuros a partir de una única entrada histórica. Este procedimiento puede realizarse de forma directa (entrenando el modelo para predecir varios pasos simultáneamente) o, más comúnmente, de forma iterativa, reutilizando las predicciones previas como entrada para las siguientes:

$$\begin{aligned}
\hat{x}_{t+1} &= f(x_{t-S+1}, \ldots, x_t) \\
\hat{x}_{t+2} &= f(x_{t-S+2}, \ldots, x_t, \hat{x}_{t+1}) \\
&\vdots \\
\hat{x}_{t+K} &= f(\hat{x}_{t+K-S}, \ldots, \hat{x}_{t+K-1})
\end{aligned}$$

Este enfoque permite proyectar la evolución de la serie a medio o largo plazo, pero introduce un fenómeno conocido como **acumulación de error**: pequeñas desviaciones en las primeras predicciones pueden amplificarse conforme se avanza, afectando la calidad de los resultados.

2.8.3 Aplicación en este trabajo

Ambos métodos han sido implementados en el presente trabajo. La predicción de un solo paso se utiliza como referencia de precisión, mientras que la predicción multipaso se emplea para analizar la capacidad de generalización de cada modelo

y evaluar su estabilidad en horizontes de predicción más largos, como semanas, trimestres o años.

3 Metodología

En esta sección se describe el proceso seguido para evaluar y comparar diferentes modelos de predicción aplicados a series temporales. El objetivo es analizar el rendimiento de redes neuronales profundas (Basic ANN, SimpleRNN, GRU) y de la computación de reservorio (ESN) en términos de precisión y eficiencia computacional.

El análisis experimental se ha estructurado en varias etapas:

- Preparación de los datos: tratamiento de series temporales, normalización, segmentación en ventanas y división en conjuntos de entrenamiento y prueba.
- Implementación de los modelos: descripción de la arquitectura y configuración de cada técnica, junto con los marcos de programación utilizados.
- Estrategia de predicción: definición del enfoque adoptado (predicción de un solo paso o multipaso) y los horizontes temporales considerados.
- Procedimiento de entrenamiento y validación: parámetros utilizados, funciones de pérdida, optimizadores y criterios de evaluación comunes a todos los modelos.

El objetivo final es establecer una base común sobre la que comparar objetivamente las distintas técnicas, permitiendo valorar no solo la calidad de las predicciones, sino también el coste computacional y la estabilidad de los modelos a lo largo del tiempo.

3.1 Preparación de datos

El análisis se ha llevado a cabo utilizando series reales de precios semanales de productos del sector primario. En concreto, se han empleado datos de precios de aguacates , dorada y salmón. El objetivo es evaluar la capacidad de generalización de los modelos en contextos controlados y en entornos reales sujetos a variabilidad externa.

3.1.1 Series reales

Las series reales utilizadas provienen de tres fuentes principales:

- El mercado de Corabastos (Bogotá, Colombia), con registros semanales de precios en pesos colombianos por kilogramo de aguacates *hass* y *papelillo*.
- El mercado español, con precios mensuales de dorada (*Sparus aurata*) registrados en euros por kilogramo.

- El mercado noruego, con precios mensuales de salmón en euros por kilogramo.

Para el procesamiento de estas series, se siguieron los siguientes pasos:

- **Conversión temporal:** los registros fueron reordenados como series temporales univariadas con frecuencia semanal o mensual, según el caso.
- **Normalización:** se aplicó un escalado lineal mínimo-máximo para acotar los datos al intervalo $[0, 1]$, utilizando la expresión:

$$x_t^{\text{norm}} = \frac{x_t - \min(x)}{\max(x) - \min(x)}$$

  Esta transformación mejora la estabilidad numérica y favorece la convergencia del entrenamiento en modelos neuronales.

3.1.2 Ventanas deslizantes

Con el fin de adaptar las series temporales al formato requerido por los modelos neuronales, se empleó una estrategia de segmentación en ventanas temporales de tamaño fijo S, lo que permitió construir pares entrada-salida de la forma:

$$\mathbf{x}_t = \left[x_{t-S+1}, \ldots, x_t\right]^\top, \qquad y_t = x_{t+1}$$

De esta forma, el problema de predicción se convierte en una tarea supervisada de regresión sobre vectores de entrada de dimensión $S \in \mathbb{N}$. En los experimentos, se utilizaron ventanas de un año $S = 52$.

3.1.3 División de datos

Cada serie temporal fue dividida en dos subconjuntos:

- **Entrenamiento:** el 80% de los datos, utilizados para el ajuste de los modelos.
- **Prueba:** el 20% restante, reservado para la evaluación objetiva del rendimiento.

Esta división permite estimar la capacidad de generalización de los modelos ante datos no vistos durante el entrenamiento.

3.2 Implementación de modelos

La implementación práctica de los modelos neuronales se ha realizado en el lenguaje de programación `Python`, empleando bibliotecas ampliamente utilizadas en aprendizaje automático, como `NumPy`, `scikit-learn` y `TensorFlow/Keras`. A continuación, se detallan las configuraciones empleadas en cada una de las arquitecturas, así como los criterios de entrenamiento comunes a todas ellas.

3.2.1 Entorno de trabajo

Los experimentos se desarrollaron en un entorno de notebooks interactivos con `Python 3.10`. Se emplearon las siguientes bibliotecas principales:

- `NumPy` para manejo de arrays y operaciones vectoriales.
- `pandas` para manipulación de datos tabulados.
- `Matplotlib` para visualización de resultados y gráficas.
- `scikit-learn` para escalado de datos y regresión lineal.
- `TensorFlow/Keras` para definición y entrenamiento de modelos neuronales.

3.2.2 Arquitectura de los modelos

Para hacer un análisis comparativo se implementaron cuatro modelos con configuraciones comparables:

- **Basic ANN**: arquitectura `feedforward` con una capa de entrada de tamaño $S = 52$, y una capa densa de salida con una neurona con función de activación identidad.
- **Simple RNN**: red neuronal recurrente básica, con una capa recurrente de tipo `SimpleRNN` (20 unidades), seguida de una capa `Dense` para la salida. En las unidades recurrentes se utiliza la función de activación **ReLu**.
- **GRU**: red recurrente avanzada con 100 unidades, sin capas adicionales intermedias y utilizando activaciones estándar y configuraciones recomendadas por Keras.
- **ESN (Echo State Network)**: red con reservorio de tamaño fijo (200 unidades), pesos internos aleatorios y normalizados para cumplir la condición de estado de eco. La capa de salida se ajustó mediante regresión lineal regularizada (Ridge Regression), entrenando únicamente la matriz W_{out}.

En el caso del modelo basado en ESN, un aspecto fundamental es que los pesos del reservorio (W) y de entrada (W_{in}) se generan aleatoriamente. Esta aleatoriedad, aunque permite una proyección rica de los datos, también introduce inestabilidad entre ejecuciones individuales. Para mitigar este efecto y mejorar la robustez de las predicciones, es habitual el uso de **modelos de ensamblado**.

La idea consiste en:

1. Ejecutar M instancias independientes de una ESN, cada una con diferentes inicializaciones aleatorias.
2. Obtener una predicción $\hat{y}_t^{(i)}$ de cada instancia $i = 1, \ldots, M$.

3. Calcular la predicción final como el promedio:

$$\hat{y}_t = \frac{1}{M} \sum_{i=1}^{M} \hat{y}_t^{(i)}$$

Este proceso estabiliza la salida, reduce la varianza y mejora la generalización. En problemas con alta sensibilidad a condiciones iniciales o ruido, el ensamblado puede reducir significativamente el RMSE sin aumentar excesivamente el coste computacional, ya que el entrenamiento de cada ESN es altamente eficiente.

En el caso del modelo ESN utilizado se optó por una estrategia de ensamble con 20 modelos.

3.2.3 Proceso de entrenamiento

Para los modelos basados en aprendizaje profundo se utilizaron los siguientes parámetros de entrenamiento:

- Función de pérdida: error cuadrático medio (`mean squared error`).

  La función de pérdida utilizada en el entrenamiento es el **error cuadrático medio** (MSE, por sus siglas en inglés).

- Optimizador: `Adam`[9] , con tasa de aprendizaje (`learning_rate` en el código) ajustada según el modelo, en el rango $\eta \in [0.001,\ 0.1]$.

- Número de épocas (epochs): es el número de veces que recorre los datos de entrenamiento.Entre 80 y 600, dependiendo de la arquitectura.

- Tamaño de lote: los grupos de muestras en los que se va a organizar el total de estas, típicamente 32.

- División de los datos: entrenamiento (80%) y validación (20%).

Para las ESN, el entrenamiento se limitó al ajuste de W_{out} mediante la solución cerrada de la regresión lineal regularizada, descrita en la Sección 2.7.

3.2.4 Predicción y horizonte temporal

En este trabajo se han implementado dos estrategias de predicción temporal: la predicción de un solo paso y la predicción multipaso. Cada una de ellas se adapta a un escenario distinto de uso y presenta ventajas y desafíos particulares.

Predicción de un solo paso En este enfoque, el modelo estima únicamente el valor x_{t+1} a partir de los últimos S valores observados:

$$\hat{x}_{t+1} = f(x_{t-S+1}, x_{t-S+2}, \ldots, x_t)$$

Esta estrategia se implementó directamente con el método `predict()` sobre las muestras del conjunto de prueba. El código utilizado fue:

```
y_pred = modelo.predict(X_test)
erroresPrediccion(y_test, y_pred)
```

La función `erroresPrediccion` calcula automáticamente varias métricas para comparar $\hat{x}_{t+1}$ con los valores reales x_{t+1}, tales como el error cuadrático medio (MSE), el error absoluto medio (MAE), la desviación estándar del error y el error máximo/mínimo.

Predicción multipaso La predicción multipaso permite estimar una secuencia futura de valores:

$$\hat{x}_{t+1}, \hat{x}_{t+2}, \ldots, \hat{x}_{t+K}$$

Esto se logra utilizando iterativamente las predicciones anteriores como parte de la entrada para los siguientes pasos. Por ejemplo:

$$\hat{x}_{t+2} = f(x_{t-S+2}, \ldots, x_t, \hat{x}_{t+1})$$

Este procedimiento se implementó en el código mediante la función personalizada `prediccionMedioPlazo(N)`, donde N indica el número de semanas que se desea predecir. Internamente, esta función utiliza un bucle de retroalimentación y aplica el modelo de forma recursiva:

```
prediccionMedioPlazo(Nsemanas)
```

También se empleó la función `ErroresMedioPlazo(X, y, N)` para obtener los errores numéricos asociados a cada predicción iterada, permitiendo construir gráficas de evolución del error en el horizonte futuro.

Resumen comparativo Ambos enfoques han sido evaluados en todas las arquitecturas implementadas (Basic ANN, Simple RNN, GRU y ESN). La predicción de un solo paso proporciona una medida estable de la precisión inmediata, mientras que la predicción multipaso permite analizar la degradación del rendimiento a lo largo del tiempo, un aspecto esencial en aplicaciones reales donde se requiere anticipar tendencias o ciclos completos.

Los modelos se evaluaron en las series reales de precios de aguacates, dorada y salmón, con horizontes de predicción de hasta 52 semanas.

3.3 Evaluación de modelos

Para comparar el rendimiento de los distintos modelos en las tareas de predicción temporal, se han empleado varias métricas estándar en problemas de regresión. Estas métricas cuantifican la diferencia entre las predicciones generadas por el modelo y los valores reales de la serie.

3.3.1 Error Cuadrático Medio (MSE)

La métrica principal utilizada durante el entrenamiento fue el **error cuadrático medio**:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

donde y_i representa el valor real, $\hat{y}_i$ la predicción, y N el número total de muestras. Esta métrica penaliza más fuertemente los errores grandes, favoreciendo modelos que eviten desviaciones severas.

3.3.2 Error Absoluto Medio (MAE)

El **error absoluto medio** mide la desviación promedio sin elevar al cuadrado, en valor absoluto:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

Es menos sensible a valores atípicos y resulta útil como métrica complementaria al MSE.

3.3.3 Raíz del Error Cuadrático Medio (RMSE)

También se ha calculado la **raíz del error cuadrático medio**, que se expresa en las mismas unidades que los datos originales:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}$$

Esta métrica es útil para interpretar la magnitud promedio del error y se utiliza frecuentemente en problemas de series temporales.

3.3.4 Otros indicadores

Además de las métricas estándar como el MSE, MAE y RMSE, se han calculado otras cantidades estadísticas que permiten caracterizar mejor el comportamiento del modelo, especialmente en tareas de predicción multipaso:

- **Error máximo:**

  $$\text{Error}_{\max} = \max_{1 \leq i \leq N} |y_i - \hat{y}_i|$$

  Representa la mayor desviación absoluta entre predicción y valor real, útil para detectar fallos puntuales graves.

- **Error mínimo:**

$$\text{Error}_{\min} = \min_{1 \leq i \leq N} |y_i - \hat{y}_i|$$

  Informa de qué tan cerca ha estado el modelo en su mejor predicción.

- **Desviación típica del error:**

$$\sigma_e = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(|y_i - \hat{y}_i| - \text{MAE} \right)^2}$$

  Esta medida cuantifica la dispersión del error absoluto respecto a su media. Cuanto más bajo sea su valor, más consistentes son las predicciones.

Todas estas métricas han sido implementadas en el código mediante la función `erroresPrediccion()`, que toma como entrada el vector de valores reales y el de predicciones y devuelve un resumen detallado de resultados. En el caso de predicción multipaso, se utiliza `ErroresMedioPlazo()` para evaluar la evolución del error en horizontes largos.

Estas métricas complementarias permiten detectar no solo si un modelo es preciso en promedio, sino también si es estable y fiable en distintos contextos temporales.

3.4 Coste computacional y eficiencia

Uno de los objetivos clave de este trabajo es analizar no solo la precisión de los distintos modelos, sino también su eficiencia computacional. En particular, se pretende evaluar en qué medida las redes de estado de Eco (ESN) ofrecen una ventaja en términos de velocidad de entrenamiento frente a modelos neuronales recurrentes tradicionales como LSTM o GRU.

3.4.1 Número de parámetros

El número total de parámetros entrenables en un modelo tiene un impacto directo en el coste computacional. Denotemos por:

- N_{in}: dimensión de la entrada,
- N_{hid}: número de neuronas en la capa oculta o reservorio,
- N_{out}: dimensión de la salida (en este caso, 1),
- L: número de capas ocultas (en MLP).

Modelos como MLP, RNN, LSTM, GRU El número total de parámetros entrenables se aproxima por:

$$P_{\text{total}} \approx \sum_{\ell=1}^{L} (N_\ell \cdot N_{\ell-1} + N_\ell) \quad + \quad N_{\text{hid}} \cdot N_{\text{out}} + N_{\text{out}}$$

La siguiente demostración justifica la expresión utilizada para estimar el número de parámetros entrenables en redes densamente conectadas:
Demostración:
Supongamos una red neuronal de tipo *feedforward* (como un MLP) con L capas, donde la entrada tiene dimensión N_0, y cada capa $\ell \in \{1, \ldots, L\}$ tiene N_ℓ neuronas.

Cada neurona de la capa ℓ recibe una conexión desde cada neurona de la capa anterior $\ell-1$, lo que implica $N_{\ell-1}$ pesos. Además, tiene asociado un sesgo. Por tanto, cada neurona en la capa ℓ introduce $N_{\ell-1} + 1$ parámetros.

Multiplicando por el número total de neuronas en la capa ℓ, el número de parámetros en dicha capa es:

$$P_\ell = N_\ell \cdot (N_{\ell-1} + 1) = N_\ell \cdot N_{\ell-1} + N_\ell$$

Sumando sobre todas las capas ocultas y la de salida, se obtiene la expresión general del número total de parámetros entrenables en la red:

$$P_{\text{total}} = \sum_{\ell=1}^{L} (N_\ell \cdot N_{\ell-1} + N_\ell)$$

□

En modelos recurrentes como LSTM o GRU, además se incluyen pesos internos adicionales para compuertas, lo que eleva aún más la cantidad total de parámetros.

El entrenamiento de estos modelos implica iteraciones del algoritmo de retropropagación (BPTT) durante múltiples épocas, con complejidad computacional:

$$\mathcal{O}(E \cdot N \cdot P_{\text{total}})$$

donde E es el número de épocas y N el número de muestras de entrenamiento (ambos definidos en la sección 3.2.3.).

Redes ESN En una red de estado de eco:

Los pesos internos W del reservorio y de entrada W_{in} son generados aleatoriamente y no se entrenan. Solo se ajusta la matriz de salida $W_{\text{out}} \in \mathbb{R}^{1 \times N_{\text{hid}}}$, mediante regresión lineal regularizada (ver Sección 2.7).

Por tanto, el número de parámetros ajustados es:

$$P_{\text{ESN}} = N_{\text{hid}}$$

Y el coste computacional de entrenamiento es el de resolver una ecuación de mínimos cuadrados:

$$\mathcal{O}(N_{\text{hid}}^2 \cdot T)$$

donde T es el número de pasos temporales. Esta operación tiene coste muy inferior al de modelos con entrenamiento por gradiente descendente.

3.4.2 Medición empírica

En el código, se registraron los tiempos de entrenamiento mediante la función `time()` de Python. Esto nos da otra manera de medir eficiencias de los distintos métodos, además del coste computacional previamente mencionado.

4 Bases de datos utilizadas

Para evaluar el rendimiento de las arquitecturas neuronales implementadas en este trabajo, se han empleado tres series de datos temporales procedentes del sector primario. Estas series permiten analizar distintas dinámicas de mercado, tanto en el ámbito agrícola como en el pesquero. En todos los casos, se trabaja con series univariadas $\{x_t\}_{t=1}^{T} \subset \mathbb{R}$, donde cada x_t representa un precio observado en un instante discreto $t \in \mathbb{N}$. Cabe destacar que la línea discontinua roja que se ve en todas las representaciones gráficas, separa datos de entrenamiento de datos de prueba.

4.1 Precios del salmón (`Forwardprices_full.csv`)

Esta base contiene precios *forward* del salmón expresados en euros (€), registrados con frecuencia mensual. Los datos han sido obtenidos a través del **Fish Pool ASA**, la bolsa de productos derivados con sede en Bergen, Noruega, especializada en la negociación de contratos de futuros de salmón y en la gestión de riesgos de precios dentro de la cadena de valor acuícola.

Las columnas principales incluyen:

- `Closing Date`: fecha de cierre del contrato (`YYYY-MM-DD`),
- `EUR value`: valor nominal en euros

La serie contiene observaciones mensuales desde **enero de 2006 hasta junio de 2024**, con una densidad temporal elevada y sin eventos extremos notables. La variable objetivo es $x_t = \texttt{EUR value}_t$, tratada como una secuencia mensual en tiempo discreto.

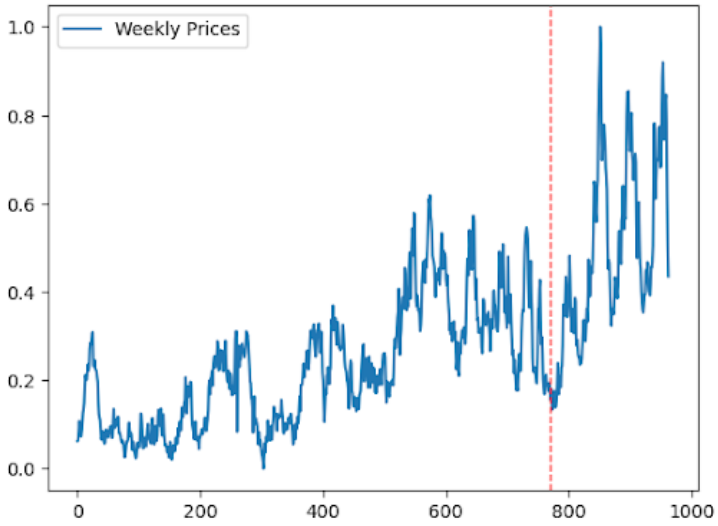

Figure 7: Precios del salmón.

Table 1: Resumen estadístico de la base de datos: Precios del salmón (Forward-prices_full.csv)

Atributo	**Valor**
Núm. observaciones	963
Periodicidad	Semanal
Inicio	Enero 2006
Fin	Junio 2024
Unidad	EUR/Kg
Media	5.3401
Desv. típica	1.9005
Mínimo	2.4700
Percentil 25	3.7650
Mediana (P50)	5.0300
Percentil 75	6.3750
Máximo	12.7700

4.2 Precios de la dorada [10] (`VC Dorada SP.xlsx`)

Este archivo contiene una serie temporal de precios de la dorada (*Sparus aurata*) en el mercado español, recogida en el contexto de la tarea 6.2 del proyecto europeo **MedAid (Mediterranean Aquaculture Integrated Development)** financiado por la Unión Europea (Horizon 2020, contrato n.º 727315).

Incluye precios en tres niveles de la cadena de valor: en granja, mayorista y

minorista. En este trabajo se ha utilizado específicamente la serie **minorista**, que cubre el periodo entre **enero de 2009 y octubre de 2018** con frecuencia mensual.

Las columnas relevantes son:

- `Fecha`: instante temporal de la observación (`YYYY-MM-DD`),
- `Precio EUR/kg`: precio de venta por kilogramo en euros.

La variable se modela como x_t = precio observado de dorada (EUR/kg), con un total de **118 observaciones**, un valor medio de **9.4891**, y una desviación estándar de **0.5302**. Se trata de una serie con menor estacionalidad y mayor dispersión relativa que las anteriores.

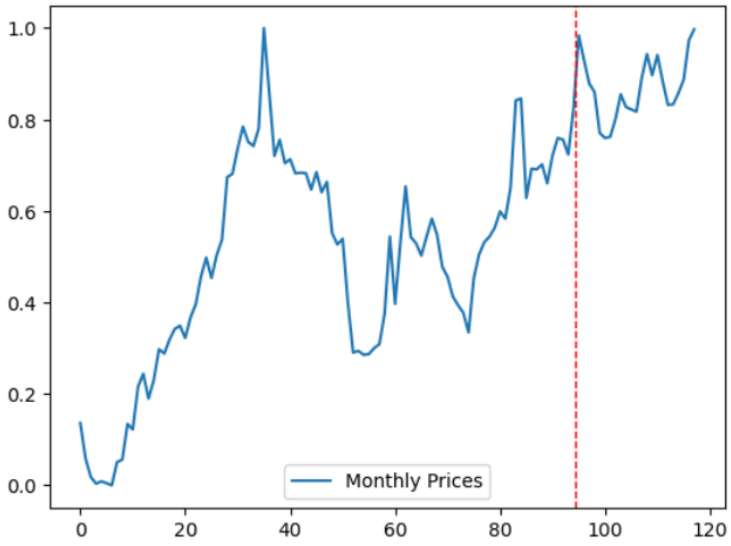

Figure 8: Precios de la dorada.

Table 2: Resumen estadístico de la base de datos: Precios de la dorada (VC Dorada SP.xlsx)

Atributo	**Valor**
Núm. observaciones	118
Periodicidad	Mensual
Inicio	Enero 2009
Fin	Octubre 2018
Unidad	EUR/Kg
Media	9.4891
Desv. típica	0.5302
Mínimo	8.34
Percentil 25	9.107
Percentil 50	9.53
Percentil 75	9.89
Máximo	10.38

4.3 Precios del aguacate (`datosCorabastos.csv`)

Este conjunto recoge precios del aguacate *hass* registrados semanalmente en el mercado **Corabastos** de Bogotá (Colombia), la central de abastos más importante del país y una de las mayores de América Latina. La serie se extiende desde **junio de 2012 hasta abril de 2024** y presenta una frecuencia semanal.

Las columnas del archivo son:

- `Mercado`: plaza de comercialización (ej. Bogotá – Corabastos),
- `Fecha`: fecha de la observación (formato libre),
- `Precio $ / KG`: precio en pesos colombianos (COP/kg).

La serie se ha modelado como una señal univariada:

$$x_t = \text{precio semanal (COP/kg)}, \quad t = 1, \ldots, T$$

y utilizada como base principal para la experimentación predictiva. Se han considerado tanto el conjunto completo como filtrados por plaza de mercado (principalmente Bogotá).

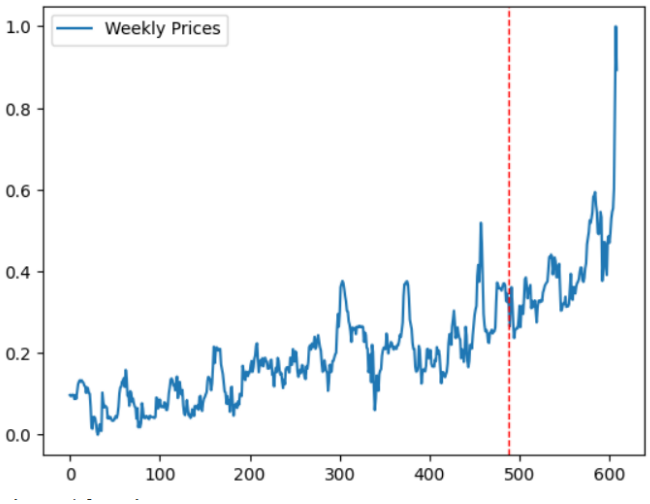

Figure 9: Precios del aguacate.

Table 3: Resumen estadístico de la base de datos: Precios del aguacate (datosCorabastos.csv)

Atributo	**Valor**
Núm. observaciones	609
Periodicidad	Semanal
Inicio	Junio 2012
Fin	Abril 2024
Unidad	COP/Kg
Valor medio	4529.35
Desv. típica	1208.05
Mínimo	2610.00
Percentil 25	3653.00
Mediana (P50)	4277.00
Percentil 75	5272.00
Máximo	11611.00

4.4 Normalización de datos

Antes de introducir las series temporales en los modelos predictivos, se ha aplicado una normalización a las variables de entrada con el fin de mejorar la estabilidad numérica del entrenamiento y acelerar la convergencia. Dado que las redes neuronales son sensibles a la escala de los datos, la normalización permite alinear las distintas magnitudes a un mismo rango de trabajo.

En este estudio se ha utilizado la técnica de **normalización min-max**, que

transforma una serie de valores reales $\{x_t\}_{t=1}^{T} \subset \mathbb{R}$ al intervalo unitario $[0, 1]$ mediante la fórmula:

$$x_t^{\text{norm}} = \frac{x_t - x_{\min}}{x_{\max} - x_{\min}}$$

donde:

- x_t es el valor original en el instante t,
- $x_{\min} = \min\{x_1, \ldots, x_T\}$,
- $x_{\max} = \max\{x_1, \ldots, x_T\}$.

Este procedimiento se ha aplicado de forma independiente a cada serie (aguacate, salmón, dorada), respetando su dominio temporal. La normalización preserva la forma de la señal, reescalando su rango y facilitando una comparación justa entre modelos.

5 Resultados

Consideraciones sobre el entorno de ejecución

Dado que en esta sección se analizan los tiempos de entrenamiento y predicción de los distintos modelos implementados, es relevante especificar las características del equipo sobre el que se han realizado todas las ejecuciones.

Las simulaciones se llevaron a cabo en un equipo con las siguientes especificaciones técnicas:

- **Procesador**: Intel® Core™ Ultra 7 155U a 1.70 GHz
- **Memoria RAM instalada**: 16 GB (15.5 GB utilizables)
- **Nombre del dispositivo**: PUC-5CG5092MH8

Estas condiciones de hardware deben tenerse en cuenta al interpretar los tiempos de entrenamiento y predicción.

5.1 Resultados para los precios del aguacate

En esta sección se presentan los resultados obtenidos por los distintos modelos aplicados a la predicción de precios del aguacate en el mercado de Corabastos. Se evaluaron cuatro arquitecturas: *Basic ANN*, *Simple RNN*, *GRU* y *ESN*, para horizontes de predicción de 4, 12, 26 y 52 semanas (equivalentes a 1 mes, 1 trimestre, 1 semestre y 1 año respectivamente).

Predicciones visuales

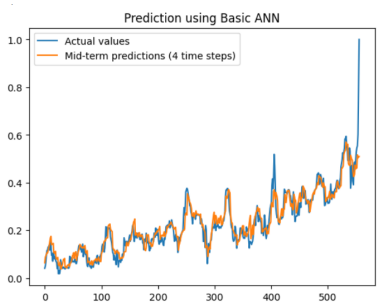

Figure 10: Predicción de Basic ANN a 4 semanas.

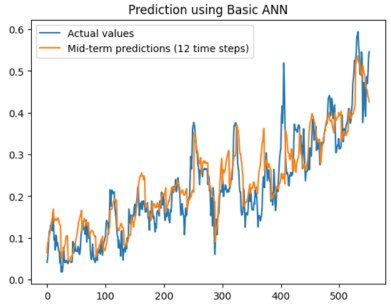

Figure 11: Predicción de Basic ANN a 12 semanas.

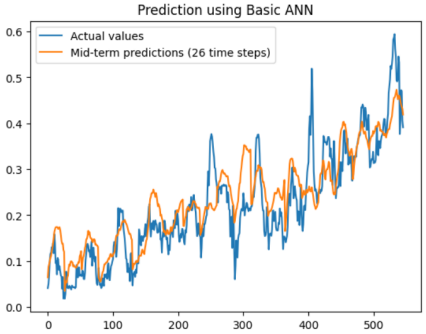

Figure 12: Predicción de Basic ANN a 26 semanas.

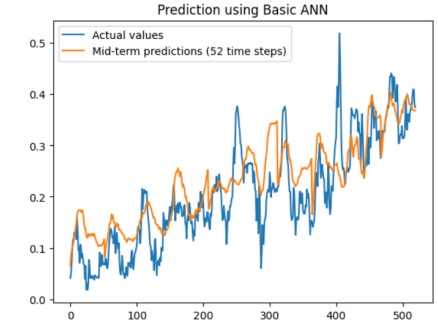

Figure 13: Predicción de Basic ANN a 52 semanas.

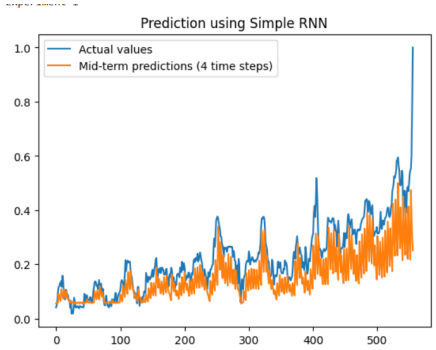

Figure 14: Predicción de Simple RNN a 4 semanas.

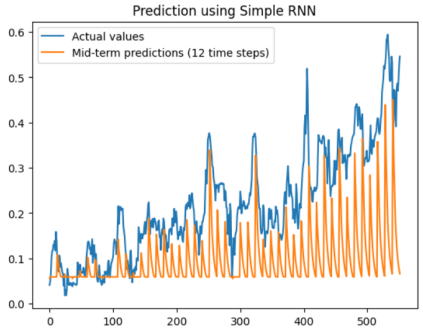

Figure 15: Predicción de Simple RNN a 12 semanas.

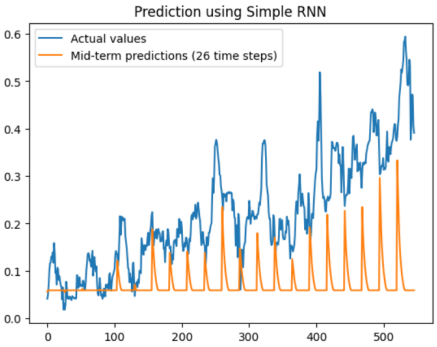

Figure 16: Predicción de Simple RNN a 26 semanas.

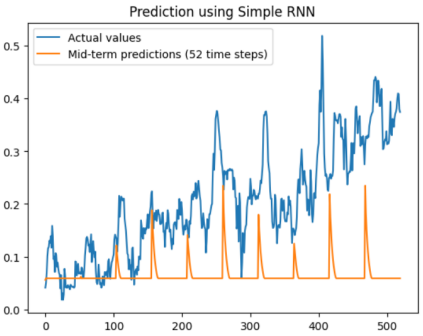

Figure 17: Predicción de Simple RNN a 52 semanas.

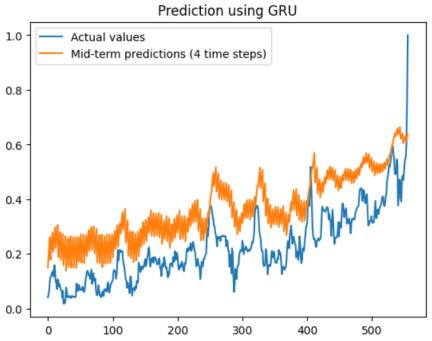

Figure 18: Predicción de GRU a 4 semanas.

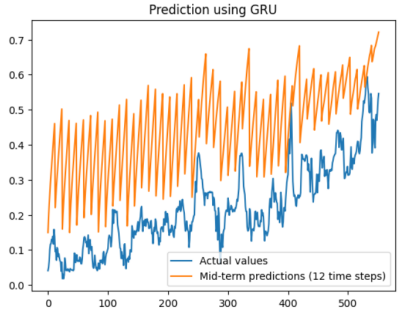

Figure 19: Predicción de GRU a 12 semanas.

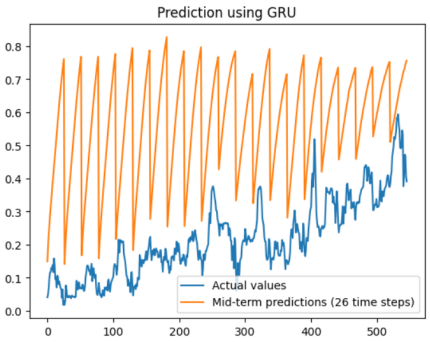

Figure 20: Predicción de GRU a 26 semanas.

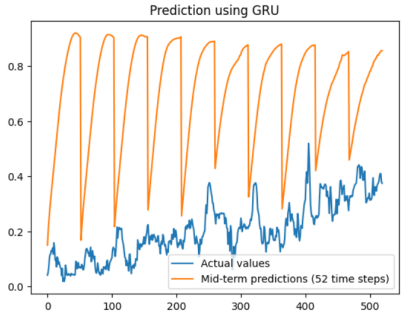

Figure 21: Predicción de GRU a 52 semanas.

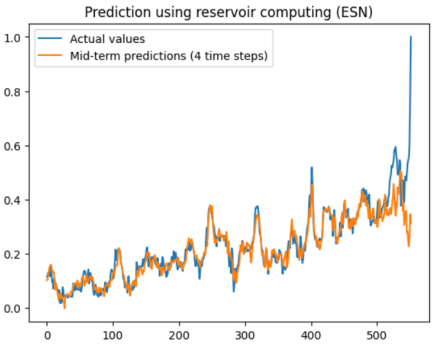

Figure 22: Predicción de ESN a 4 semanas.

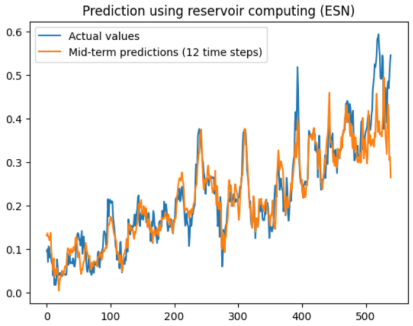

Figure 23: Predicción de ESN a 12 semanas.

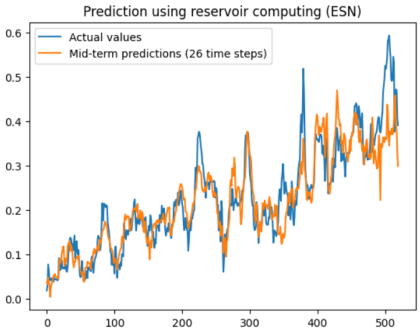

Figure 24: Predicción de ESN a 26 semanas.

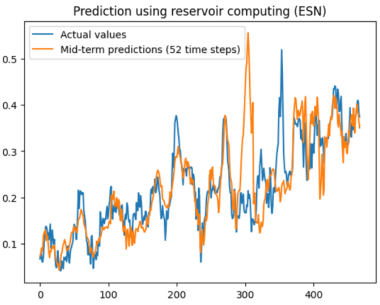

Figure 25: Predicción de ESN a 52 semanas.

Tabla de resultados:

Table 4: Resultados de la simulación para 4 semanas (aguacate)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	20.47	92.75	0.1500	0.1283	0.1974
Simple RNN	47.11	93.12	0.0609	0.0520	0.0801
GRU	147.81	111.18	0.0544	0.0480	0.0726
ESN	5.21	113.24	0.0553	0.0951	0.1100

Table 5: Resultados de la simulación para 12 semanas (aguacate)

Modelo	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	88.60	0.2728	0.2209	0.3510
Simple RNN	98.44	0.1186	0.0876	0.1474
GRU	107.92	0.0710	0.0598	0.0929
ESN	40.56	0.0659	0.1066	0.1254

Table 6: Resultados de la simulación para 26 semanas (aguacate)

Modelo	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	92.38	0.3324	0.2569	0.4201
Simple RNN	101.15	0.1692	0.1189	0.2068
GRU	112.98	0.0814	0.0671	0.1055
ESN	20.60	0.0884	0.1655	0.1877

Table 7: Resultados de la simulación para 52 semanas (aguacate)

Modelo	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	88.92	0.3169	0.2252	0.3888
Simple RNN	99.76	0.1846	0.1199	0.2201
GRU	110.32	0.0850	0.0680	0.1088
ESN	12.38	0.0943	0.1256	0.1570

5.2 Resultados para los precios de la dorada

En esta sección se presentan los resultados obtenidos por los distintos modelos aplicados a la predicción de precios de la dorada en el mercado español. Se evaluaron cuatro arquitecturas: *Basic ANN*, *Simple RNN*, *GRU* y *ESN*, para horizontes de predicción de 1, 3, 6, 12 meses (equivalentes a 1 mes, 1 trimestre, 1 semestre y 1 año respectivamente).

Predicciones visuales

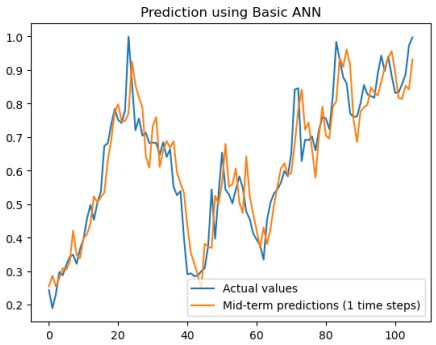

Figure 26: Predicción de Basic ANN a 4 semanas (dorada).

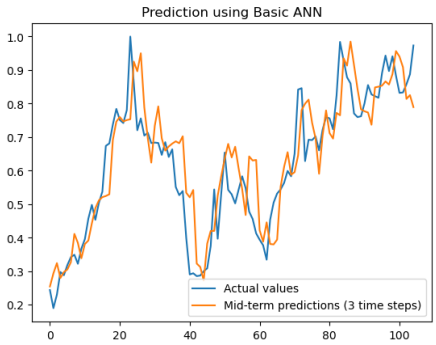

Figure 27: Predicción de Basic ANN a 12 semanas (dorada).

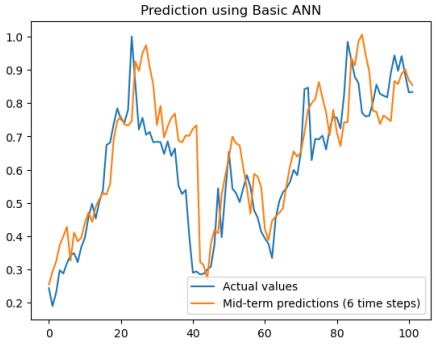

Figure 28: Predicción de Basic ANN a 26 semanas (dorada).

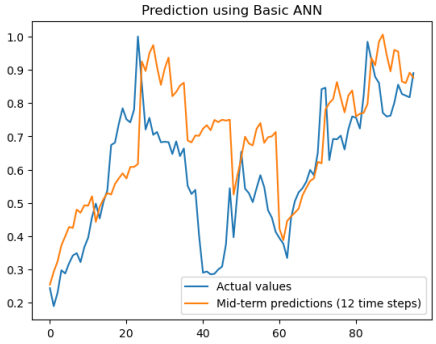

Figure 29: Predicción de Basic ANN a 52 semanas (dorada).

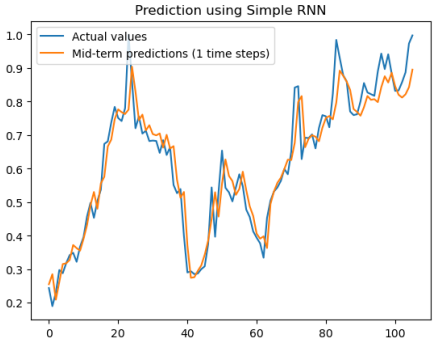

Figure 30: Predicción de Simple RNN a 4 semanas (dorada).

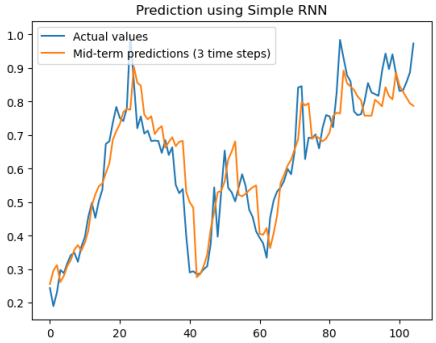

Figure 31: Predicción de Simple RNN a 12 semanas (dorada).

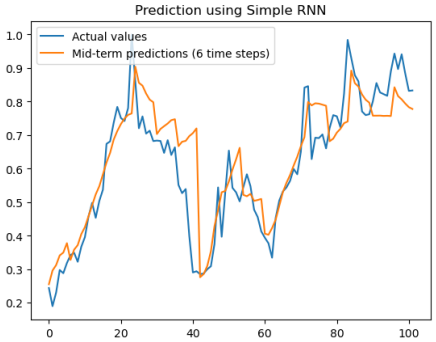

Figure 32: Predicción de Simple RNN a 26 semanas (dorada).

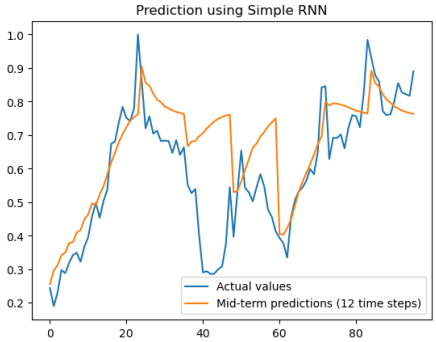

Figure 33: Predicción de Simple RNN a 52 semanas (dorada).

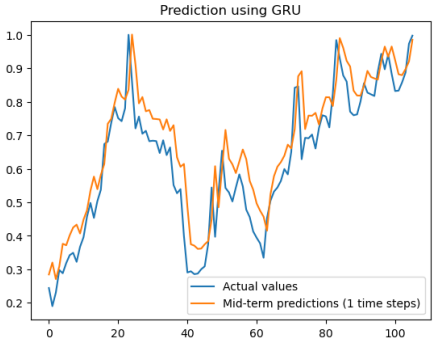

Figure 34: Predicción de GRU a 4 semanas (dorada).

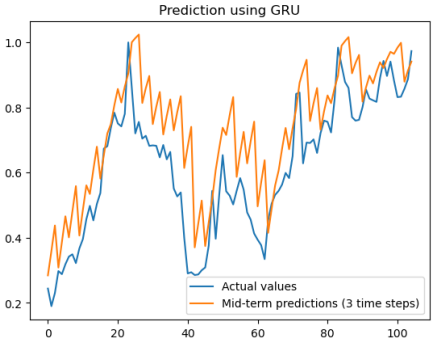

Figure 35: Predicción de GRU a 12 semanas (dorada).

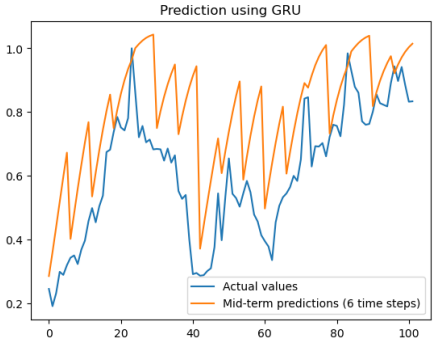

Figure 36: Predicción de GRU a 26 semanas (dorada).

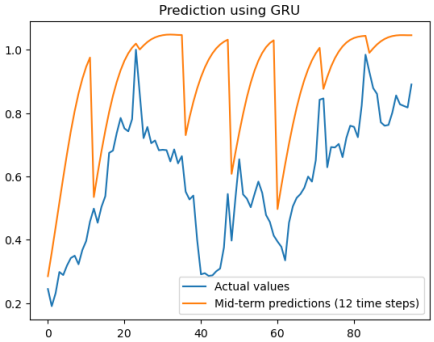

Figure 37: Predicción de GRU a 52 semanas (dorada).

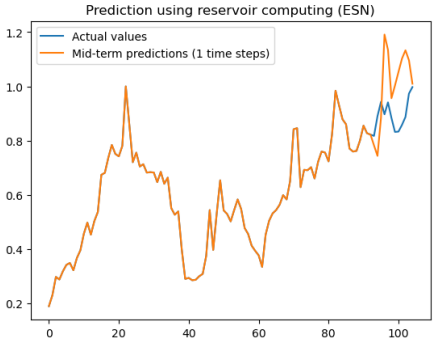

Figure 38: Predicción de ESN a 4 semanas (dorada).

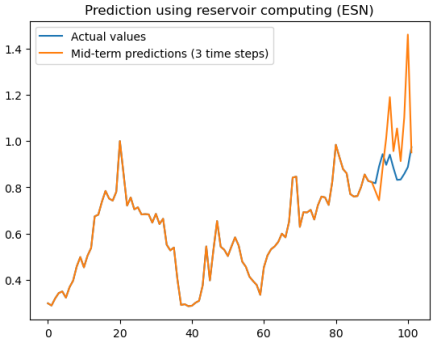

Figure 39: Predicción de ESN a 12 semanas (dorada).

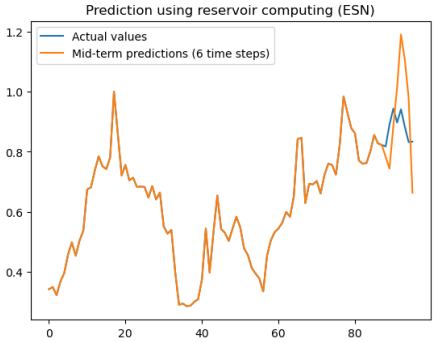

Figure 40: Predicción de ESN a 26 semanas (dorada).

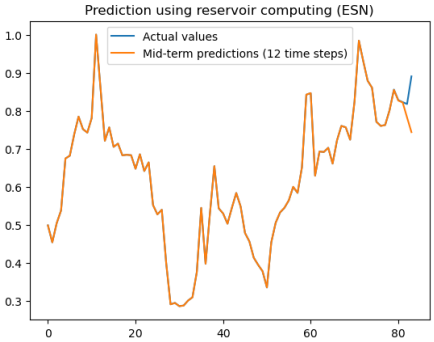

Figure 41: Predicción de ESN a 52 semanas (dorada).

Tabla de resultados:

Table 8: Resultados de la simulación para 1 mes (dorada)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	6.20344	5.04165	0.05884	0.04812	0.07601
Simple RNN	6.73562	5.03003	0.04517	0.04413	0.06315
GRU	8.56845	5.47031	0.07351	0.05027	0.08905
ESN	0.64617	11.50672	0.01732	0.05663	0.05922

Table 9: Resultados de la simulación para 3 meses (dorada)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	6.20344	4.96326	0.07671	0.06197	0.09861
Simple RNN	6.73562	5.09119	0.06245	0.05329	0.08210
GRU	8.56845	6.51541	0.12307	0.09183	0.15355
ESN	0.64617	2.90158	0.01789	0.07137	0.07358

Table 10: Resultados de la simulación para 6 meses (dorada)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	6.20344	4.78782	0.09235	0.08143	0.12312
Simple RNN	6.73562	6.56389	0.07266	0.07256	0.10268
GRU	8.56845	6.67296	0.19170	0.12604	0.22943
ESN	0.64617	1.32590	0.01200	0.04435	0.04595

Table 11: Resultados de la simulación para 12 meses (dorada)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	6.20344	4.55380	0.14432	0.11160	0.18243
Simple RNN	6.73562	6.12236	0.10926	0.11187	0.15637
GRU	8.56845	6.62181	0.30793	0.16253	0.34819
ESN	0.64617	0.67673	0.00228	0.01633	0.01649

5.3 Resultados para los precios del salmón

En esta sección se presentan los resultados obtenidos por los distintos modelos aplicados a la predicción de precios del salmón en el mercado noruego. Se evaluaron cuatro arquitecturas: *Basic ANN*, *Simple RNN*, *GRU* y *ESN*, para horizontes de predicción de 1, 3, 6 y 12 meses (equivalentes a 4, 12, 26 y 52 semanas respectivamente).

Predicciones visuales

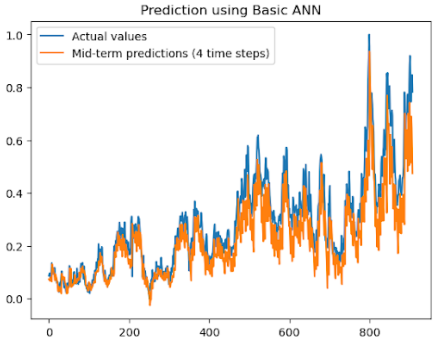

Figure 42: Predicción de Basic ANN a 4 semanas (salmón).

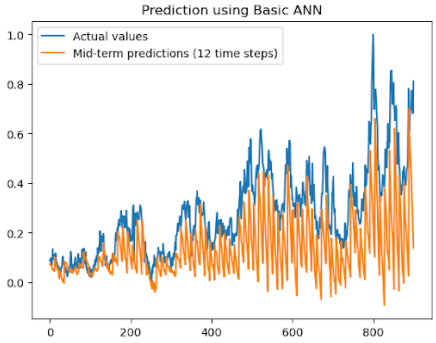

Figure 43: Predicción de Basic ANN a 12 semanas (salmón).

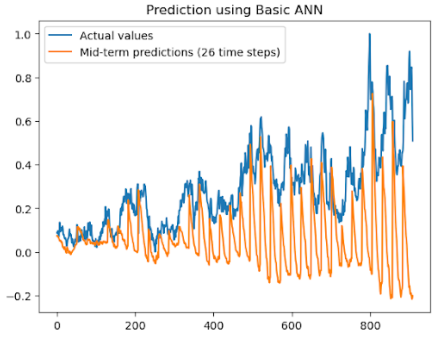

Figure 44: Predicción de Basic ANN a 26 semanas (salmón).

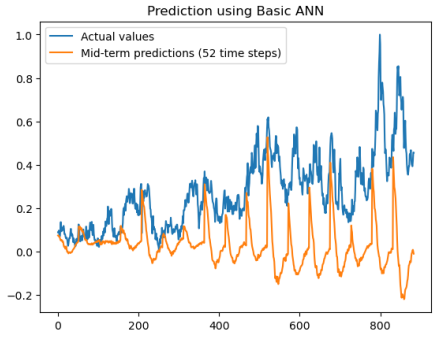

Figure 45: Predicción de Basic ANN a 52 semanas (salmón).

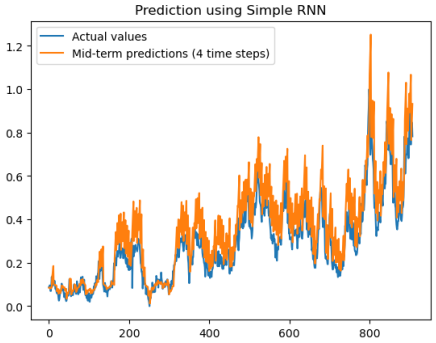

Figure 46: Predicción de Simple RNN a 4 semanas (salmón).

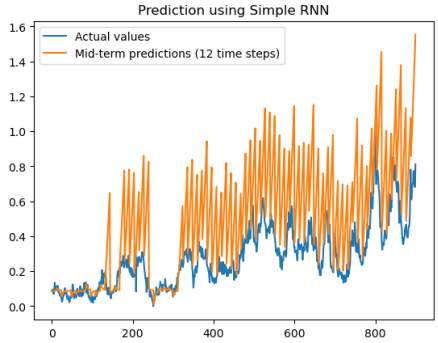

Figure 47: Predicción de Simple RNN a 12 semanas (salmón).

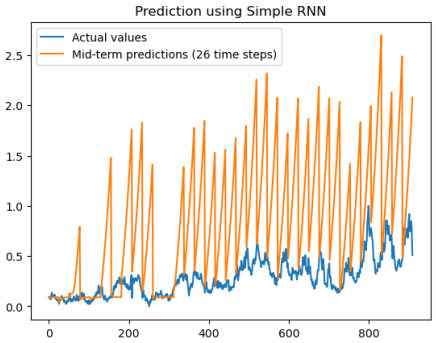

Figure 48: Predicción de Simple RNN a 26 semanas (salmón).

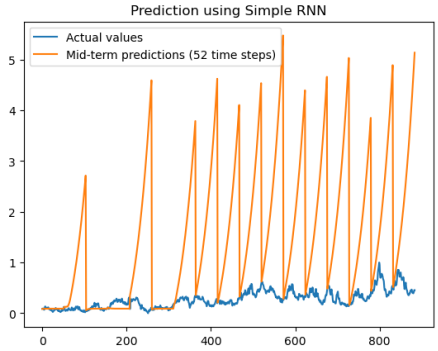

Figure 49: Predicción de Simple RNN a 52 semanas (salmón).

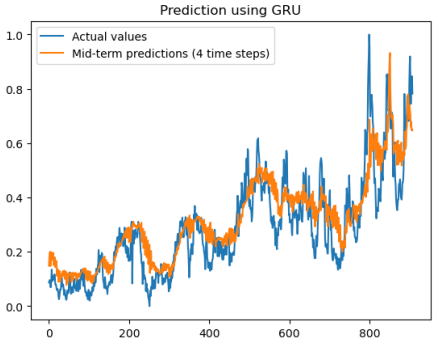

Figure 50: Predicción de GRU a 4 semanas (salmón).

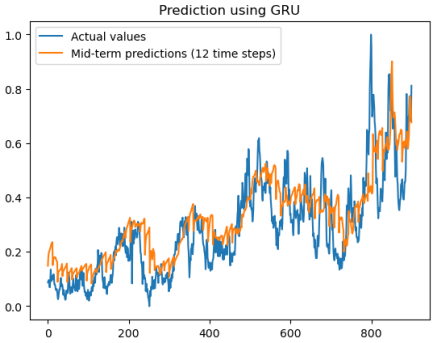

Figure 51: Predicción de GRU a 12 semanas (salmón).

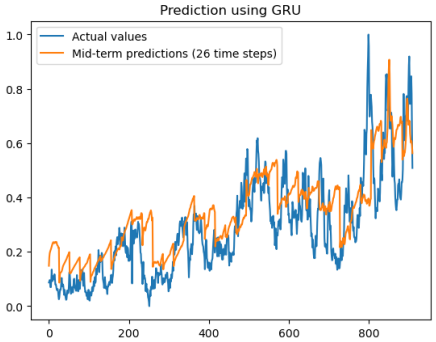

Figure 52: Predicción de GRU a 26 semanas (salmón).

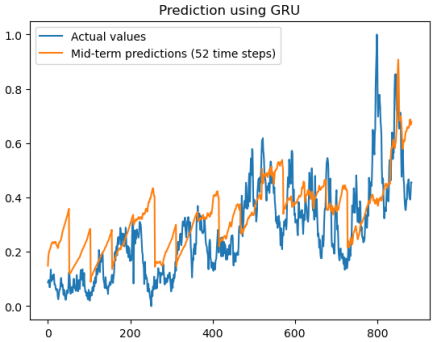

Figure 53: Predicción de GRU a 52 semanas (salmón).

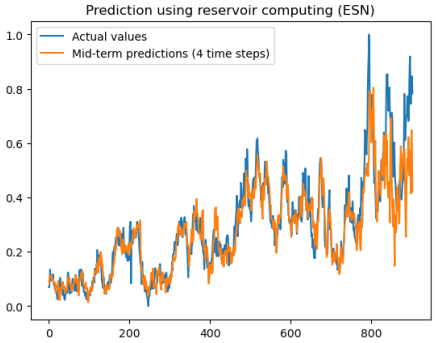

Figure 54: Predicción de ESN a 4 semanas (salmón).

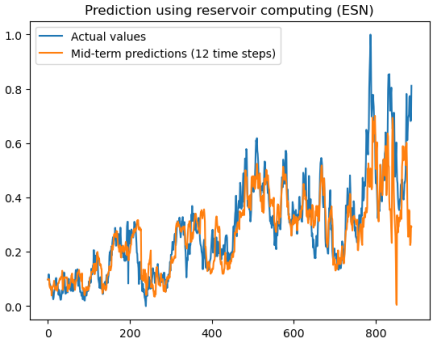

Figure 55: Predicción de ESN a 12 semanas (salmón).

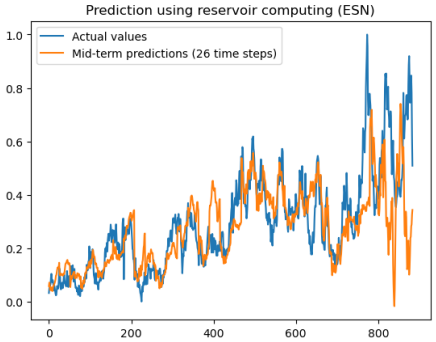

Figure 56: Predicción de ESN a 26 semanas (salmón).

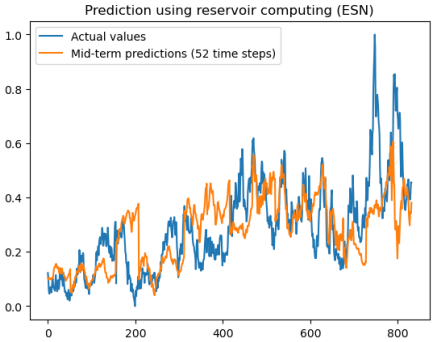

Figure 57: Predicción de ESN a 52 semanas (salmón).

Tabla de resultados:

Table 12: Resultados de la simulación para 4 semanas (salmón)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	7.92136	78.44680	0.07052	0.07009	0.09934
Simple RNN	17.36690	56.12030	0.08021	0.07659	0.11095
GRU	48.32250	44.70370	0.07319	0.06151	0.09561
ESN	1.38223	128.85400	0.06550	0.05052	0.08266

Table 13: Resultados de la simulación para 12 semanas (salmón)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	7.92136	96.94850	0.15201	0.13962	0.20644
Simple RNN	17.36690	61.92400	0.22021	0.19620	0.30032
GRU	48.32250	44.06760	0.08807	0.07167	0.11355
ESN	1.38223	46.58000	0.06333	0.05705	0.08667

Table 14: Resultados de la simulación para 26 semanas (salmón)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	7.92136	70.16130	0.23733	0.21859	0.32266
Simple RNN	17.36690	52.47650	0.58215	0.52267	0.78326
GRU	48.32250	44.22570	0.10629	0.08338	0.13059
ESN	1.38223	25.04040	0.08786	0.11437	0.14442

Table 15: Resultados de la simulación para 52 semanas (salmón)

Modelo	Entrenamiento (s)	Predicción (s)	Error medio	Desv. típica	RMSE
Basic ANN	7.92136	86.03340	0.25524	0.20186	0.32330
Simple RNN	17.36690	42.80560	1.29156	1.38219	1.85262
GRU	48.32250	43.55970	0.11430	0.09155	0.14644
ESN	1.38223	12.92590	0.09959	0.09559	0.13832

5.4 Resultados tras 10 ejecuciones:

En esta subsección se presentan los resultados agregados obtenidos tras realizar 10 ejecuciones independientes para cada uno de los modelos analizados. Las siguientes tablas comparativas muestran, para cada producto (dorada, aguacate y salmón) y para distintos horizontes de predicción (4, 12, 26 y 52 semanas), las métricas de evaluación más relevantes:

- Error medio,
- Desviación típica del error,
- Raíz del error cuadrático medio (RMSE),
- Tiempo medio de entrenamiento (en segundos),
- Tiempo medio de predicción (en segundos).

En cada tabla, se ha resaltado en color los mejores valores por columna (mínimos), lo cual permite identificar de forma visual el modelo más eficiente o preciso según el criterio evaluado. Esta representación facilita una comparación directa del comportamiento de las arquitecturas *Basic ANN*, *Simple RNN*, *GRU* y *ESN* frente a distintas series temporales y escalas temporales de predicción.

Table 16: Comparativa de errores y tiempos para 4 semanas (Dorada)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.05191	0.04460	0.06845	8.16227	6.87161
Simple RNN	0.05055	0.04700	0.06903	10.84246	8.24741
GRU	0.04780	0.03838	0.06142	8.44015	5.95791
ESN	0.00955	0.03114	0.03257	0.62169	8.99079

Table 17: Comparativa de errores y tiempos para 12 semanas (Dorada)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.07450	0.06270	0.09738	8.16227	7.18125
Simple RNN	0.07753	0.06672	0.10230	10.84246	7.90492
GRU	0.07182	0.05644	0.09139	8.44015	5.82843
ESN	0.00866	0.03362	0.03473	0.62169	2.92774

Table 18: Comparativa de errores y tiempos para 26 semanas (Dorada)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.09320	0.08528	0.12634	8.16227	6.43861
Simple RNN	0.09488	0.08454	0.12717	10.84246	6.92599
GRU	0.09740	0.08504	0.12962	8.44015	5.65156
ESN	0.01159	0.05294	0.05423	0.62169	1.40178

Table 19: Comparativa de errores y tiempos para 52 semanas (Dorada)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.14067	0.11180	0.17979	8.16227	6.27666
Simple RNN	0.13823	0.11043	0.17759	10.84246	6.12212
GRU	0.14417	0.11918	0.18803	8.44015	5.41962
ESN	0.00116	0.00867	0.00875	0.62169	0.69950

Table 20: Comparativa de errores y tiempos para 4 semanas (Aguacate)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.05889	0.04815	0.07643	7.92133	28.95859
Simple RNN	0.03820	0.03747	0.05362	12.93138	27.24229
GRU	0.07716	0.05731	0.09774	35.96335	32.69881
ESN	0.02428	0.04668	0.05262	1.12338	55.58690

Table 21: Comparativa de errores y tiempos para 12 semanas (Aguacate)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.11568	0.08542	0.14403	7.92133	26.64455
Simple RNN	0.07193	0.05798	0.09256	12.93138	26.70384
GRU	0.11832	0.07275	0.14070	35.96335	29.76177
ESN	0.02609	0.03308	0.04215	1.12338	18.09811

Table 22: Comparativa de errores y tiempos para 26 semanas (Aguacate)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.16238	0.11297	0.19820	7.92133	26.37118
Simple RNN	0.10978	0.07927	0.13577	12.93138	26.50829
GRU	0.14000	0.07840	0.16284	35.96335	29.06469
ESN	0.02835	0.03214	0.04289	1.12338	8.42009

Table 23: Comparativa de errores y tiempos para 52 semanas (Aguacate)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.24509	0.18751	0.30905	7.92133	27.45363
Simple RNN	0.14947	0.09307	0.17691	12.93138	25.45027
GRU	0.15043	0.07336	0.16990	35.96335	33.77007
ESN	0.03044	0.03227	0.04445	1.12338	4.40253

Table 24: Comparativa de errores y tiempos para 4 semanas (Salmón)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.14604	0.11454	0.18593	10.44956	56.70407
Simple RNN	0.06032	0.06177	0.08650	22.07616	81.94864
GRU	0.20552	0.10930	0.23380	51.74884	54.96437
ESN	0.04713	0.06576	0.08092	1.60455	150.34700

Table 25: Comparativa de errores y tiempos para 12 semanas (Salmón)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.33120	0.26713	0.42599	10.44956	60.03264
Simple RNN	0.11421	0.11110	0.15959	22.07616	80.42876
GRU	0.42740	0.26061	0.50821	51.74884	57.31983
ESN	0.06040	0.08801	0.10679	1.60455	50.97781

Table 26: Comparativa de errores y tiempos para 26 semanas (Salmón)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	0.67794	0.58986	0.90116	10.44956	61.95984
Simple RNN	0.18683	0.18081	0.26048	22.07616	78.48473
GRU	0.69281	0.40172	0.81246	51.74884	53.64663
ESN	0.08136	0.11432	0.14035	1.60455	24.56280

Table 27: Comparativa de errores y tiempos para 52 semanas (Salmón)

Modelo	Error medio	Desv. típica	RMSE	Entrenamiento (s)	Predicción (s)
Basic ANN	2.04674	2.46617	3.21790	10.44956	65.42675
Simple RNN	0.27690	0.25788	0.37906	22.07616	66.64768
GRU	0.97847	0.46300	1.08986	51.74884	52.73539
ESN	0.09684	0.10774	0.14497	1.60455	13.07896

6 Análisis de resultados

6.1 Comparación entre ESN y el resto de modelos

A continuación se presenta un análisis comparativo centrado en las redes *Echo State Networks* (ESN), en relación con las arquitecturas alternativas consideradas: *Basic ANN*, *Simple RNN* y *GRU*. Para ello se distinguen dos dimensiones principales: la calidad estadística de las predicciones y el coste computacional requerido por cada modelo.

6.1.1 Análisis estadístico

Los resultados obtenidos en los tres conjuntos de datos (aguacate, dorada y salmón) reflejan que las ESN proporcionan, de forma consistente, los errores más bajos en la mayoría de los horizontes de predicción. En particular:

- En las tablas de comparación tras 10 ejecuciones, las ESN presentan los valores más bajos de **RMSE** en casi todos los casos.

- En lo que respecta a la **desviación típica del error**, las ESN destacan por su estabilidad: las variaciones entre ejecuciones son mínimas en comparación con el resto de modelos, lo cual demuestra robustez frente a la aleatoriedad en la inicialización o el muestreo.
- Las redes *Simple RNN* y *Basic ANN* se sitúan sistemáticamente por detrás tanto en precisión como en estabilidad estadística debido a su sencillez.

En resumen, las ESN ofrecen un equilibrio óptimo entre baja magnitud de error y baja variabilidad.

6.1.2 Análisis de coste computacional

Desde el punto de vista del tiempo requerido para el entrenamiento y la predicción, las diferencias entre modelos son más significativas aún que en el apartado anterior:

- Las ESN requieren un **tiempo de entrenamiento muy reducido**, ya que únicamente ajustan la capa de salida mediante regresión lineal. El reservorio interno permanece fijo, lo que evita el uso de algoritmos de optimización iterativos.
- Los modelos GRU, al implicar entrenamiento recurrente con múltiples compuertas y dependencias temporales, presentan los **tiempos de entrenamiento más elevados**. Esta diferencia se acentúa en horizontes de predicción más largos.
- Las redes RNN y ANN presentan un coste de entrenamiento intermedio, aunque sin ofrecer mejoras competitivas en términos de error.
- En cuanto al **tiempo de predicción**, las ESN también muestran una ventaja o, al menos, un rendimiento comparable. Su arquitectura permite un cálculo eficiente al evitar ciclos de retroalimentación entrenados.

Por tanto, en términos de eficiencia, las ESN permiten acelerar significativamente el proceso de entrenamiento sin comprometer la calidad de la predicción, lo que las convierte en candidatas sólidas para constituirse como el modelo más adecuado en el contexto de predicción de series temporales.

7 Conclusiones

Los resultados muestran de forma consistente que las **Echo State Networks (ESN)** constituyen la opción más robusta y eficiente para este tipo de tareas. Han obtenido:

- Los errores más bajos en la mayoría de los horizontes y productos analizados.

- Una menor desviación típica entre ejecuciones, lo que indica alta estabilidad.
- El menor coste computacional de entrenamiento, debido a su entrenamiento cerrado por regresión lineal.

En vista de estos resultados, se recomienda el uso de redes ESN como modelo preferente para tareas de predicción univariada en series temporales. Su combinación de precisión, estabilidad estadística y bajo coste computacional las convierte en una herramienta especialmente adecuada para contextos donde se requiera eficiencia, interpretabilidad y reproducibilidad. Esta arquitectura demuestra ser una alternativa sólida y matemáticamente fundamentada frente a modelos más complejos y costosos como las GRU.

8 Notas

- **Nota 1:** el archivo PrediccionesProductosAgricolas.py (con el código python para realizar la comparación de modelos) se considera parte de este trabajo.
- **Nota 2:** Este trabajo[15] se ha tomado como punto de partida para realizar este trabajo.
- **Nota 3:** Los resultados y enfoques desarrollados en este Trabajo de Fin de Grado serán presentados en el **XII Congreso Internacional de Matemática Aplicada y Computacional (XII-CIMAC)**, organizado por la Sociedad Peruana de Matemática Aplicada y Computacional (SPMAC) y la Universidad Nacional del Altiplano de Puno (UNAP). El evento se celebrará del 11 al 15 de agosto de 2025 en Puno, Perú.

  El trabajo aceptado, titulado *Application of Deep Learning and Reservoir Computing to Time Series Analysis in the Primary Sector*, ha sido elaborado en colaboración con el profesor Ángel Cobo y será expuesto durante las sesiones del congreso como ejemplo de aplicación de técnicas avanzadas de aprendizaje automático en el sector primario.

References

[1] Jonathan Binas, Jürgen Schmidhuber, et al. On the role of strong nonlinearity in echo state networks. *Neural Networks*, 153:215–228, 2022.

[2] Lorenzo Cesarini, Rui Gonçalves, Mario Martina, Xavier Romão, Barbara Monteleone, Florentino L. Pereira, and Rui Figueiredo. Comparison of deep learning models for milk production forecasting at national scale. *Computers and Electronics in Agriculture*, 221:108933, 2024.

[3] Kyunghyun Cho, Bart Van Merriënboer, Căglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[4] DataCamp. Recurrent neural network (rnn) tutorial for beginners, 2020. https://www.datacamp.com/tutorial/tutorial-for-recurrent-neural-network.

[5] Keyan Ghazi-Zahedi, Ingo Fischer, et al. A survey on reservoir computing and its interdisciplinary applications beyond traditional machine learning. *Neural Networks*, 157:340–373, 2023.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[7] Lukas Illing and Niklas Shoefer. Reservoir-computing machine-learning algorithms as observers of dynamical systems. *Communications in Nonlinear Science and Numerical Simulation*, 85:105262, 2020.

[8] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical report, German National Research Center for Information Technology (GMD), Report 148, 2001. Erratum added in 2010.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[10] MedAID Consortium. Deliverable 6.2 – market dynamics and price analyses of the mediterranean aquaculture industry. `http://www.medaid-h2020.eu/index.php/deliverables/`, 2018. European Union Horizon 2020 project, Grant Agreement No. 727315.

[11] Michael Nielsen. *Neural Networks and Deep Learning*. 2015. Disponible en: http://neuralnetworksanddeeplearning.com/.

[12] Susan Stepney. Physical reservoir computing: A tutorial. *Natural Computing*, 23:665–685, 2024.

[13] Wikipedia contributors. Gated recurrent unit. Wikipedia, The Free Encyclopedia, 2025. https://en.wikipedia.org/wiki/Gated$_r ecurrent_u nit$.

[14] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge university press edition, 2024. Capítulo 10.1.1: Gated memory cell.

[15] Ángel Cobo, Rocío Rocha, and Lina Albor. Modelos predictivos de aprendizaje profundo en el sector agrícola. In *CLAIO XXII Congreso Latino Iberoamericano de Investigación Operativa*, pages 28 Oct–1 Nov, Guadalajara, México, oct 2024. CLAIO. Trabajo utilizado como guía general para el desarrollo de este TFG.