UNIVERSIDAD
TORCUATO DI TELLA

Típo de documento: Tesis de Maestría

Departamento de Economía. Maestría en Econometría

Forecasting en presencia de regime-switching

Análisis comparado de la precisión predictiva de redes neuronales LSTM frente a modelos ARIMA y ETS

Autoría: Pérez, Matías Damián
Año: 2024

¿Cómo citar este trabajo?
Pérez, M. (2024) "*Forecasting en presencia de regime-switching. Análisis comparado de la precisión predictiva de redes neuronales LSTM frente a modelos ARIMA y ETS*". [Tesis de Maestría. Universidad Torcuato Di Tella]. Repositorio Digital Universidad Torcuato Di Tella
https://repositorio.utdt.edu/handle/20.500.13098/13206

El presente documento se encuentra alojado en el Repositorio Digital de la Universidad Torcuato Di Tella bajo una licencia Creative Commons Atribución-No Comercial-Compartir Igual 4.0 Argentina (CC BY-NC-SA 4.0 AR)
Dirección: https://repositorio.utdt.edu

Biblioteca Di Tella

Forecasting en presencia de regime-switching

Análisis comparado de la precisión predictiva de redes neuronales LSTM frente a modelos ARIMA y ETS

Alumno: Matías D. Perez
Legajo: 19y2066

Tutor: Prof. Gabriel Martos Venturini

Maestría en Econometría

06/2024

Abstract

En este trabajo se realiza un estudio comparativo de la capacidad predictiva de las redes neuronales LSTM frente a los modelos econométricos ARIMA y ETS en contexto de *regime-switching*, es decir, en casos donde los parámetros que rigen el DGP cambian a lo largo del tiempo. Para ello se plantean tres experimentos controlados en donde se simula un cambio de régimen en la serie de tiempo y se comparan las predicciones de cada modelo antes y después de dicho cambio utilizando métricas de precisión como son el MSE o el MAE. Posteriormente, se repite este ejercicio pero esta vez para la serie de precios del Bitcoin, la cual se modela como un proceso del tipo Markov-switching. La conclusión general a la que se arriba es que los modelos de redes neuronales recurrentes no resultaron ser más precisos que las predicciones de los modelos ARIMA y ETS, más bien lo contrario, y esto puede atribuirse a la excesiva dependencia que tienen los primeros respecto a los datos de entrenamiento y la imposibilidad resultante de pronosticar por fuera del intervalo de valores aprendido durante su entrenamiento.

1. Introducción

Las últimas décadas han sido testigos del auge y la expansión de las técnicas de aprendizaje automático -tambien conocido como *Machine Learning*- más allá de un reducido número de aplicaciones en Inteligencia Artificial. La Economía en general, y la Econometría Financiera en particular, no se han quedados rezagadas en la carrera por la adopción de las mismas, principalmente en un contexto de producción de cada vez mayores volúmenes de datos.

Para comenzar, y antes de ahondar en los avances en la disciplina, es preciso definir que se entiende por Machine Learning o cuando un sistema puede decirse que "aprende". Según el Profesor Tom M. Mitchel de la Universidad Carnegie Mellon (1997) un sistema se dice que aprende de la experiencia E con respecto a algún tipo de tarea T y a una medida de desempeño P si su desempeño en tareas en T, medido por P, mejora con la experiencia E. Por regla general, los algoritmos de Machine Learning buscan reconocer y explotar patrones existentes en grandes volúmenes de datos (E) en pos de realizar tareas (T) de regresión, clasificación o clusterización, entre otras, buscando minimizar una función de pérdida (P) la cual guía el aprendizaje.

Del párrafo anterior se desprende que una diferencia entre dichos algoritmos y los modelos estadísticos tradicionalmente usados en Econometría es que los primeros aprenden directamente de los datos sin necesidad de hacer supuestos adicionales sobre el *data generating process* (DGP). Como se verá en una sección posterior, economistas influyentes como Susan Athey (2020) y Hal Varian (2014) volverán sobre este punto a la hora de evaluar el impacto real del Machine Learning en Economía.

Tradicionalmente, estos algoritmos se han dividido en tres grandes grupos de acuerdo al paradigma de aprendizaje que siguen (Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H, 2009):

- *Aprendizaje Supervisado*: el objetivo es predecir el valor de una variable $y \in \mathcal{Y}$, llamada output o target, en base a otro conjunto de variables $X \in \mathcal{X}$, conocidas como input o features. El aprendizaje se dice "supervisado" en tanto ejemplos de la variable y guían al algoritmo en el proceso de aprender una función $f : \mathcal{X} \to \mathcal{Y}$ que generalice mejor la relación entre y y X. Este aprendizaje surge de comparar, función de pérdida mediante, el verdadero valor y dado X con el valor generado por el algoritmo $f(X)$ y ajustar el mismo para que la diferencia entre ambos sea mínima.

- *Aprendizaje No Supervisado*: es un tipo de aprendizaje en donde no existe distinción entre variables output e input y cuyo objetivo es aprender asociaciones (clusters) o representaciones reducidas de los datos.

- *Reinforcement Learning*: es un paradigma de aprendizaje basado en el modelo de acción-respuesta y guarda similitudes con la programación dinámica. En él, y mediante la interacción con el medio, el algoritmo aprende cierta regla de acción (*policy*) que maximice la recompensa esperada.

Para la Econometría Financiera y de Series de Tiempo, y particularmente para la tarea de *forecasting*, resulta de especial interés el primer grupo de algoritmos en

tanto todo pronóstico en tiempo t sobre el valor futuro que adopte una variable $y_{t+h} \in \mathcal{Y}_{t+h}$ es equivalente a la tarea de predicir de y_{t+h} en base a la información disponible en momento t, $X_t \in \mathcal{X}_t$.

Sin embargo, de entre todos los algoritmos existentes, un subgrupo de ellos ha atraído particularmente la atención de la Econometría Financiera: el *Deep Learning* o aprendizaje profundo. Como subcampo del Machine Learning (Figura 1), el Deep Learning busca encontrar una transformación o representación de los datos usados como input para obtener una predicción cercana al verdadero valor de la variable target. La diferencia con otros modelos de aprendizaje automático, como podrían ser *Random Forests* o *Support Vector Machines*, es que los algoritmos de Deep Learning ponen énfasis en aprender capas sucesivas de representaciones cada vez más complejas y abstractas de los datos. Estas representaciones son aprendidas vía modelos llamados redes neuronales, estructuradas en capas una encima de la otra y conectadas entre sí. (Chollet, 2021, pp.5-7)

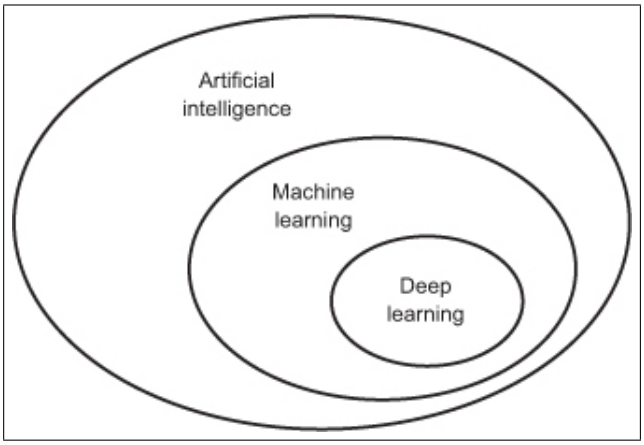

Figura 1: Relación entre AI, ML y DL

Pero, ¿cuál es el catalizador de dicho interés por parte de la Econometría Financiera? ¿Por qué, a juzgar por el vasto número de publicaciones científicas, el forecasting de variables financieras como acciones, bonos y criptomonedas parece ser terreno fértil para implementar redes neuronales? Entre las explicaciones posibles suelen citarse el tamaño de las series financieras, con observaciones diarias e incluso intradiarias frente a observaciones trimestrales o anuales de otras series macroeconómicas; la amplia disponibilidad de las mismas así como la habilidad de las redes neuronales de capturar relaciones no lineales entre los datos. De hecho, es esta última capacidad de identificar representaciones de alto nivel de las secuencias de datos lo que le daría una ventaja frente a modelos econométricos tradicionales -en su mayoría lineales- en la tarea de predecir variables que son por naturaleza volátiles y no estacionarias.

Como bien evidencian Nosratabadi et. al. (2020) en un meta-análisis conducido sobre la literatura reciente en la materia, el número de artículos *peer-reviewed* publicados en donde se aplican técnicas de Deep Learning para predecir la evolución de series financieras ha crecido consistentemente en la última década. En este paper también se menciona que la arquitectura de red neuronal más aplicada es la *Long Short-Term Memory* o LSTM. En este último punto coinciden con Ryll &

Seidens (2019), los cuales concluyen que el estado del arte en el aprendizaje profundo aplicado a series de tiempo financieras lo constituyen las redes LSTM dada su capacidad innata de capturar patrones temporales latentes en los datos gracias a su arquitectura recurrente.

Sin embargo, la literatura no explora como se desempeñan estos modelos en presencia de *regime switching*, es decir, cuando se levanta el supuesto simplificador de parámetros constantes y se admite la posibilidad que los mismos puedan evolucionar a lo largo del tiempo. De acuerdo a Elliot & Timmerman (2016a, p. 447) , la inestabilidad del modelo genera una caída súbita en la capacidad predictiva en tanto los parámetros del DGP han cambiado mientras que las estimaciones de los parámetros ponen demasiado énfasis en las observaciones que ocurrieron antes del cambio, produciendo pronósticos sesgados debido a una incorrecta especificación del modelo.

El objetivo de este trabajo es tomar las redes LSTM como caso de estudio e indagar sobre su capacidad predictiva en presencia de cambios de régimen en los parámetros que rigen el DGP de la variable a predecir y_t. Así, en un contexto donde las redes LSTM ganan popularidad, es importante preguntarse cómo sus predicciones se deterioran ante casos de regime switching y que medidas pueden tomarse para corregirlo.

El presente trabajo está estructurado del siguiente modo: en el apartado 2 se formaliza qué se entiende por pronóstico o forecasting en series de tiempo financieras y cómo se comparan los algoritmos de aprendizaje automático con modelos econométricos tradicionales, como pueden ser los modelos ARMA o de suavizamiento exponencial; en el apartado 3 se realiza una introducción a las redes neuronales recurrentes y se focaliza en las arquitecturas LSTM; la sección 4 se reserva para experimentar la capacidad predictiva de este tipo de algoritmos en casos de regime switching, simulando como cambian en los parámetros que rigen tres tipos de DGP: *random walk*, ARIMA y un proceso de tipo Markov-switching sobre un número finito de estados. En todos los casos, se evaluará la capacidad predictiva de la red neuronal utilizando métricas tradicionales de precisión como son el *Mean Square Error*(MSE) y el *Mean Absolute Error* (MAE). El punto 5 deja los datos sintéticos de lado y utiliza la serie de precios de cierre del par BTC-USD para comparar la capacidad predictiva de las redes LSTM en caso de cambios de régimen frente a modelos econométricos tradicionales. Finalmente el apartado 6 se reserva a las conclusiones del trabajo.

2. Forecasting en series de tiempo financieras

2.1 Definición y alcance

Adoptando la postura de Elliot & Timmermann (2016b), la tarea de predecir el valor futuro de una variable y en base a datos observados hasta el momento t debe analizarse dentro del marco de la teoría de la decisión en tanto la predicción siempre es input para una decisión ulterior. Por ejemplo, el forecasting del precio de un activo financiero no es un fin en sí mismo sino el input con el que se toman decisiones como serían reajustar un portafolio en base a los rendimientos pronosticados de dicho activo o fijar un *stop loss* en una estrategia de *trading* algorítmico. Como consecuencia de ello, el problema de predicción se transforma en un problema

de estimación: la predicción puntual en el momento t sobre el valor que adoptará la variable y en tiempo $t+h$ es equivalente al problema estadístico de estimar un parámetro de interés (usualmente, la media o la mediana) de la función de densidad de y_{t+h} condicional a la informacíon disponible al momento t.

Formalizando, se define y_{t+h} como el valor real (desconocido en t) que adopta la variable de interés h intervalos de tiempo en el futuro. La información disponible en t que informa al pronostico se condensa en $z_t = \{x_\tau, y_\tau\}_{\tau=1}^{t} \in Z_t^i$, donde x_τ es un vector en $\mathbf{R}^p$ de variables distintas a y pero con potencial capacidad para predecir su valor futuro, y_τ es un vector en $\mathbf{R}^k$ (usualmente, $k = 1$) que recoge el valor pasado observado de la variable y en tiempo τ y Z_t^i es el conjunto de información disponible en tiempo t para el usuario i.[1] Entonces, la información disponible en una muestra en t, $z_t = \{x_\tau, y_\tau\}_{\tau=1}^{t}$, se utilizará para construir una predicción sobre el verdadero valor y_{t+h} a tiempo t: $f_{t+h|t} = f(z_t)$, tal que $f : Z_t^i \to \mathbf{R}^k$.

El siguiente elemento en este *framework* es la función de pérdida $L\left(f(z_t), y_{t+h}\right)$, función que cuantifica el costo relativo de los errores de pronóstico $e_t = y_{t+h} - f_{t+h|t}$. La función de pérdida es una primitiva del *forecaster* (Elliot & Timmermann, 2016a, p.4) ya que refleja sus objetivos últimos y no surge del propio proceso de estimación, y puede entenderse como el instrumento para tratar con el *trade-off* existente entre errores de pronóstico de distinta magnitud y dirección. Formalmente:

$$L : \mathscr{F} \times \mathscr{Y} \to \mathbf{R}^+$$

donde $\mathscr{F}$ es un espacio funcional e $\mathscr{Y} \subseteq \mathbf{R}^k$. Entre los ejemplos de funciones de pérdida comúnmente usadas se encuentran el MSE o Error Cuadrático Medio, el cual le asigna un peso relativamente mayor (cuadrático) a errores de predicción "grandes" en valor absoluto en comparación a desviaciones "chicas", y el MAE o Error Absoluto Medio, el cual pondera linealmente los errores. Otra función de pérdida muy usada en Finanzas es la conocida como Linex o Linear-exponential (Elliot & Timmermann, 2016a, p.22): una función diferenciable en todo punto pero no simétrica que encapsula la premisa que los errores de predicción de distinto signo tienen distinto peso relativo (lineal versus exponencial). Esta última función podría ser primitiva de un *forecaster* para el cual las sobre-predicciones (esto es, $y_{t+h} < f\left(z_t\right)$) del precio de un activo son relativamente más costosas que las sub-predicciones ($y_{t+h} > f\left(z_t\right)$), y, con la aplicación de esta función de pérdida, se buscaría no subestimar la potenciales caídas en el precio.

Definida la función de pérdida $L(.)$ y seleccionado el conjunto de información Z_t^i, la siguiente tarea es la búsqueda del modelo $f^*(.)$ que minimice la pérdida esperada al momento t dado z_t. Formalmente:

$$\begin{aligned} f^*(z_t) &= argmin_{f(.)\in\mathscr{F}}\, \mathbb{E}_{y_{t+h}|z_t}[L] \\ &= argmin_{f(.)\in\mathscr{F}} \int L\left(f(z_t), y_{t+h}\right) p(y_{t+h}|z_t, \theta) dy \end{aligned} \tag{1}$$

donde la esperanza de la función de pérdida es tomada respecto a la función de densidad de y_{t+h} condicionada a la información disponible en el momento t, z_t, y al

[1] Este enfoque pone en evidencia el carácter subjetivo de la información disponible Z_t^i: dos personas puestas a predecir el mismo valor y_{t+h} pueden tener acceso a distinta información o escoger distintas variables de acuerdo a lo que consideren relevante.

vector de parámetros que rigen el DGP, $\theta \in \Theta$.[2]

En la práctica, la esperanza en la ecuación (1) es estimada utilizando el promedio muestral basado en una muestra de observaciones pasadas de tamaño T. Formalmente:

$$\hat{f}(z_t) = argmin_{f(.)\in\mathscr{F}}\, T^{-1} \sum_{\tau=1}^{T} L\left(f(z_\tau), y_{\tau+h}\right) \tag{2}$$

Se espera que $\hat{f}(z_t) \xrightarrow{p} f^*(z_t)$. Sin embargo, en muestra finita y en presencia de un DGP cuyos parámetros fluctúan a lo largo del tiempo, siempre existe un conjunto de modelos consistentes con los datos. En otras palabras, todos los modelos están de algún modo mal especificados y no suele existir una única estrategia dominante al enfrentarse a un problema de forecasting, lo cual justifica el enfoque de empírico y comparativo del presente trabajo.

Solo para un grupo reducido de funciones de pérdida existe una solución al problema de minimización planteado en (1). Dentro de este grupo se encuentra la pérdida cuadrática media o MSE, la cual se define cómo:

$$L\left(f(z_t), y_{t+h}\right) = \left(y_{t+h} - f(z_t)\right)^2 \tag{3}$$

Esto es, el MSE es una función de pérdida que solo depende del error de pronóstico e_{t+h} y no de los niveles de las variables. Puede demostrarse (Elliot & Timmermann, 2016a, p.42) que la función de forecast óptimo es la esperanza condicional de y_{t+h}, es decir, $f^*(z_t) = \mathbb{E}[y_{t+h}|z_t, \theta]$, la cual depende tanto de z_t como de θ. Si bien este resultado es óptimo en el sentido de ser único dado un DGP, la esperanza condicional adopta una forma funcional desconocida que depende de parámetros también desconocidos, los cuales deberán ser estimados en base a los datos.

Como se profundizará en el apartado siguiente, es en la búsqueda de f^* sobre el espacio funcional $\mathscr{F}$ donde radica una de las principales diferencias entre el abordaje de Machine Learning y el de la Econometría: mientras que la Econometría ha optado tradicionalmente por modelos parámetricos, definiendo un subconjunto de modelos $F \subset \mathscr{F}$ (por ejemplo, todos los modelos lineales) hasta un vector de parámetros β de dimensión finita y el cual será estimado a partir de los datos; el Machine Learning adopta un enfoque menos parametrizado, buscando f^* directamente sobre el espacio $\mathscr{F}$ y basándose en resultados teóricos -como el Teorema de Aproximación Universal[3] para redes neuronales (Hornik et. al., 1989)- que garantizan que los algoritmos puedan aproximar conjuntos de funciones arbitrariamente bien en la medida que se incorporen un número suficiente -potencialmente, infinito- de términos. Es este último punto uno de los aspectos más críticos de los algoritmos de Machine Learning: a pesar de ganar en flexibilidad, la estimación de modelos complejos con muestra finita puede llevar a problemas de sobreajuste (*overfitting*) que impidan que el modelo generalice bien ante observaciones nuevas.

[2]Se denominará DGP a la función de densidad conjunta $p_{y_{T+h}, z_T}(y, z_T|\theta)$, la cual por definición de función de densidad condicional puede descomponerse como $p_{y_{T+h}|z_T}(y|z_T, \theta)\, p_{z_T}(z|\theta)$.

[3]El teorema establece que para cualquier función continua $f : \mathbf{R}^m \to \mathbf{R}^n$, existe una red neuronal con solo una capa oculta, G, tal que puede aproximar f con un nivel arbitrario de precisión (i.e., $|f(\mathbf{x}) - G(\mathbf{x})| < \epsilon, \forall\epsilon$) .

2.2 Conciliando Machine Learning y Econometría

La diferencia en la forma de explorar el espacio funcional $\mathscr{F}$ en búsqueda de f^* revela una distinción fundamental en el objetivo de una y otra disciplina: la Econometría suele adoptar el supuesto simplificador de linealidad para enfocarse en el estudio de las relaciones causales en los datos; mientras que el Machine Learning busca maximizar la precisión de sus pronósticos, decatándose por métodos computacionales de búsqueda de patrones en los datos en donde el enfoque algorítmico prima por sobre el rigor estadístico. En palabras de Zheng et. al. (2017), encontrar un buen modelo predictivo con alta precisión es diferente a inferir la verdadera estructura subyacente que genera los datos: poder predictivo es distinto a capacidad explicativa. En la misma línea apunta Athey (2018) al decir que los modelos de aprendizaje automático poco aportan al problema de identficación, de suma importancia cuando el objeto de estudio es un efecto causal. Esto se explica en tanto la Econometría desarrolla sus modelos en base a teorías económicas que asumen una serie de supuestos causales sobre el proceso estocástico que genera los datos; mientras que el Machine Learning solo basa sus modelos en los datos, sin hacer supuestos sobre la dependencia o independencia de las variables antes de que puedan ser testeadas empíricamente.

Entonces, ¿por qué los modelos de Machine Learning han ganado terreno en tareas de forecasting frente a modelos econométricos tradicionales?¿Por qué a pesar de no identificar relaciones causa-consecuencia y ser tildados de "cajas negras" predicen mejor en muchos problemas concretos que modelos de base estadística? La literatura parece coincidir al atribuirle dicha dominancia a una serie de factores, a saber:

- *No-linealidad*: al no restringir el espacio de búsqueda, los algoritmos de Machine Learning son lo suficientemente flexibles para aproximar funciones no lineales, pudiendo descubrir patrones temporales latentes en datos ruidosos y no estacionarios como lo son las series de tiempo financieras. En este sentido cabe destacar el aporte de Goulet Coulombe et. al. (2022), quienes mediante un diseño experimental en el que aislaron como tratamiento las características distintivas de los algoritmos de aprendizaje automático en relación a los modelos econométricos, concluyeron que la no-linealidad fue el tratamiento individual más efectivo para mejorar la precisión de las predicciones, especialmente en el caso horizontes de tiempo muy lejanos.

- *Regularización*: técnicas como Lasso, Ridge o Elastic Net -desarrolladas en el dominio del aprendizaje automático pero rápidamente incorporadas en Econometría- garantizan mantener acotada la complejidad del modelo, previniendo el overfitting y asegurando la generalización de sus resultados.

- *Cross-validation y selección de modelos*: mientras que en Econometría es habitual que el investigador especifique un modelo en base a la teoría económica, lo estime utilizando el dataset completo de datos y utilice el herramental estadístico para construir intervalos de confianza sobre los parámetros estimados; en Machine Learning la selección de modelos suele basarse en los datos: el forecaster provee una lista de covariables o features y el algoritmo estima modelos alternativos[4], seleccionando aquel que maximiza cierto criterio (Athey,

[4]Los modelos puestos a competir suelen diferir en su complejidad y expresividad, cualidades

2018). La estrategia más empleada para llevarlo a cabo es *cross-validation* donde cada modelo es entrenado utilizando una partición del dataset llamado conjunto de entrenamiento y evaluado sobre el complemento de este, el conjunto de validación. El modelo seleccionado será aquel que minimice la pérdida promedio sobre el conjunto de validación. Esta técnica resulta efectiva en tanto la capacidad predictiva sobre este conjunto de datos, los cuales no han sido utilizados para estimar el modelo, es una buena aproximación a cómo performaría el modelo *out-of-sample*.

- *Funciones de pérdida personalizadas*: el desarrollo de algoritmos de optimización estocástica, como *Stochastic Gradient Descent* y sus extensiones y variantes, ha permitido ampliar el espectro de funciones de pérdida al alcance del analista para incluir casi cualquier función lo suficientemente "suave" (funciones diferenciables y sub-diferenciables).

Dado que el foco de este trabajo se ha puesto en los algoritmos de Deep Learning, en el próximo apartado se verá como las arquitecturas de redes neuronales explotan las características recién mencionadas -particularmente, la no linealidad- para generar predicciones potencialmente más precisas que los modelos econométricos tradicionales.

3. Deep Learning y Forecasting

3.1 Del perceptrón a las redes neuronales recurrentes

El bloque primigenio sobre el que se erige todo el edificio del Deep Learning lo constituye el *perceptrón* (Rosenblatt, 1958): arquitectura mononeuronal que permite mapear un vector $\mathbf{x} \in \mathbf{R}^n$ en $\mathbf{R}$ (problema de regresión) o en un intervalo acotado de $\mathbf{R}$ (problema de clasificación). Esquemáticamente, un perceptrón se representa de la siguiente manera:

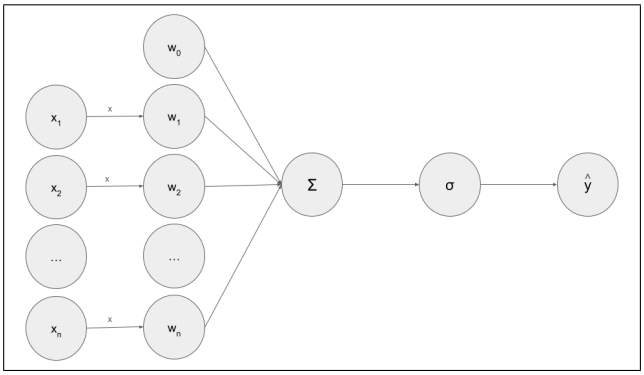

Figura 2: Arquitectura de perceptrón

Matemáticamente, un perceptrón es una función no lineal y (sub)diferenciable $f : \mathbf{R}^n \to \mathcal{Y} \subseteq \mathbf{R}$ definida de la siguiente forma:

$$\hat{y} = f(\mathbf{x}) = \sigma\left(w_0 + \Sigma_{i=1}^{n} x_i w_i\right) = \sigma(w_0 + \mathbf{x}^T\mathbf{w}) \tag{4}$$

Es decir, la función surge de aplicar una función no lineal σ (llamada función de activación) a una transformación afín del vector de inputs $\mathbf{x}$, siendo w_0 el sesgo

que son controladas por una serie de hiperparámetros propios de cada algoritmo.

y $\mathbf{w}$ los pesos que parametrizan la función. Así, cada componente del vector de input (el estímulo que recibe la neurona) es pesado por su importancia relativa en la estimación del output y la transformación lineal $\mathbf{x}^T\mathbf{w}$ es trasladada de acuerdo a la magnitud de w_0. La función σ determina, como su nombre lo indica, el nivel de activación que alcanza una neurona dado el estímulo recibido y regula la salida de la misma, propagando información hacia más adelante en la red. Es común elegir σ entre funciones no lineales y monótonas no decrecientes cuya primera (y segunda) derivada sean "fáciles" de calcular y adopten valores en un rango acotado (generalmente, $[0, 1]$). Entre las funciones más habituales se encuentran la función sigmoide, la función tangente hiperbólica y la función ReLU (*Rectified Linear Unit*). Definida la forma funcional $\sigma(.)$, solo resta estimar el vector de parámetros $\mathbf{w}$ y el sesgo w_0: como se explicará más adelante, a la estimación de los parámetros de la red neuronal se la conoce como "entrenamiento" de la red y consiste en minimizar iterativamente la función de pérdida $L(.)$ como función de los parámetros de la red $(w_0, \mathbf{w})$ hasta encontrar un mínimo global. Pero a diferencia de (2), una vez definida la arquitectura de la red, la búsqueda de una solución a dicho problema de optimización no se hará en el espacio funcional $\mathscr{F}$ sino en $\mathbf{R}^{n+1}$.

La extensión más directa del perceptrón surge de "apilar" neuronas de la siguiente forma:

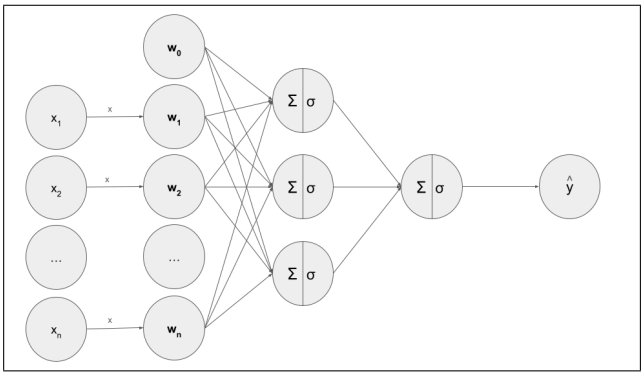

Figura 3: Arquitectura de perceptrón multicapa

En esta arquitectura llamada perceptrón multicapa o MLP, cada componente del vector de input $\mathbf{x}$ se conecta con todas y cada una de las neuronas de la capa intermedia (también conocida como capa oculta) del mismo modo en que lo hacía el perceptrón mononeuronal: al vector de input se le aplica una transformación afín y se lo reescala usando una función no lineal σ. Pero a diferencia del caso anterior, las salidas de cada una de las neuronas de la capa oculta se convierten en el nuevo input de la capa siguiente donde son combinadas y transformadas siguiendo la misma lógica. Llamando h_k a la salida k-esima de la capa intermedia, matemáticamente el perceptrón con una capa oculta de p neuronas puede definirse de la siguiente manera:

$$\begin{aligned} h_1 &= \sigma_1\left(w_{0,1} + \mathbf{x}^T\mathbf{w}_1\right) \\ h_2 &= \sigma_2\left(w_{0,2} + \mathbf{x}^T\mathbf{w}_2\right) \\ &\dots \\ h_p &= \sigma_p\left(w_{0,p} + \mathbf{x}^T\mathbf{w}_p\right) \\ \hat{y} &= \sigma_{out}\left(w_{0,out} + \mathbf{h}^T\mathbf{w}_{out}\right) \end{aligned} \tag{5}$$

O en términos matriciales:

$$\begin{aligned} \mathbf{h} &= \sigma_{hidden}\left(\mathbf{w}_0 + \mathbf{W}\mathbf{x}\right) \\ \hat{y} &= \sigma_{out}\left(w_{0,out} + \mathbf{h}^T\mathbf{w}_{out}\right) \end{aligned} \tag{6}$$

Donde los vectores $\mathbf{h}$ y $\mathbf{w}_0$ son vectores en $\mathbf{R}^p$, $\mathbf{W}$ es una matriz en $\mathbf{R}^{pxn}$ y σ_{hidden}, una función vectorial de $\mathbf{R}^p$ a $\mathbf{R}^p$. $\hat{y}$ es la estimación hecha por la red neuronal del verdadero valor y, y la discrepacia entre ambos términos se resume en la función de pérdida $L(y, \hat{y})$.

Es posible concatenar capas sucesivas de neuronas ocultas de distinta dimensión e incluso con distintas funciones de activación, dando lugar a lo que se conoce como red neuronal profunda. De esta manera, una red neuronal profunda con L capas ocultas intermedias se expresa como sigue:

$$\begin{aligned} \mathbf{h}^{(1)} &= \sigma_1\left(\mathbf{w}_0^{(1)} + \mathbf{W}^{(1)}\mathbf{x_t}\right) \\ \mathbf{h}^{(2)} &= \sigma_2\left(\mathbf{w}_0^{(2)} + \mathbf{W}^{(2)}\mathbf{h}^{(1)}\right) \\ &\dots \\ \mathbf{h}^{(L)} &= \sigma_L\left(\mathbf{w}_0^{(L)} + \mathbf{W}^{(L)}\mathbf{h}^{(L-1)}\right) \\ \hat{y} &= \sigma_{out}\left(w_{0,out} + \mathbf{w}_{out}^T\mathbf{h}^{(L)}\right) \end{aligned} \tag{7}$$

Así, la i-esima capa de la red cuenta con su propio vector de *biases* $\mathbf{w}_0^{(i)} \in \mathbf{R}^{p_i}$ y su matriz de pesos $\mathbf{W}^{(i)} \in \mathbf{R}^{p_i x p_{i-1}}$, donde p_i y p_{i-1} son el número de neuronas en la i-esima y en la (i-1)-esima capas de la red. Estos parámetros deberán ser estimados en fase de entrenamiento de la red.

Esta redes neuronales también se conocen como redes neuronales "densas" en tanto todas las neuronas de una capa reciben los estímulos provenientes de todas las neuronas de la capa anterior y están conectadas con todas las neuronas de la capa siguiente. De esta manera, en la medida que la red neuronal profunda aplica transformaciones sucesivas sobre el vector de inputs (algo que en la literatura de conoce como *forward-pass*), se van construyendo representaciones más complejas y abstractas de $\mathbf{x}$ que permiten aproximar y de manera cada vez más precisa. Esto puede observarse si se reescribe la última ecuación en (7) de manera recursiva como sigue:

$$\begin{aligned} \hat{y} = \sigma_{\text{out}}\big(&w_{0,\text{out}} + \mathbf{w}_{\text{out}}^T \\ &\Big(\sigma_L\Big(\mathbf{w}_0^{(L)} + \mathbf{W}^{(L)}\Big(\sigma_{L-1}\Big(\mathbf{w}_0^{(L-1)} \\ &+\mathbf{W}^{(L-1)}\Big(\dots\Big(\sigma_1\Big(\mathbf{w}_0^{(1)} + \mathbf{W}^{(1)}\mathbf{x}_t\Big)\dots\Big)\Big)\Big)\Big)\Big) \\ = f\,(&\mathbf{x}_t; \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(L)}, \mathbf{w}_{\text{out}}, \mathbf{w}_0^{(1)}, \mathbf{w}_0^{(2)}, \dots, \mathbf{w}_0^{(L)}, w_{0,\text{out}}\,) \end{aligned} \tag{8}$$

Definida una arquitectura de red $f(.)$; esto es, definido el número de capas ocultas, el número de neuronas en cada una de ellas y las funciones de activación, resta "entrenar" la red neuronal de manera tal que $\hat{y}$ se mapee contra el verdadero valor y minimizando el error de estimación. Este problema de optimización se realiza ajustando el conjunto de pesos $\mathbf{W} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \dots, \mathbf{W}^{(L)}, \mathbf{w}_{\text{out}}\}$ y sesgos $\mathbf{w} = \{\mathbf{w}_0^{(1)}, \mathbf{w}_0^{(2)}, \dots, \mathbf{w}_0^{(L)}, w_{0,\text{out}}\}$ iterativamente hasta que la función de pérdida alcanza un mínimo global. Formalmente:

$$f^*(\mathbf{x}) = argmin_{\substack{\mathbf{W}\in\Theta_1 \\ \mathbf{w}\in\Theta_2}} \mathbb{E}_{y|\mathbf{x}}[L\left(y, f\left(\mathbf{x}; \mathbf{W}, \mathbf{w}\right)\right)] \tag{9}$$

donde Θ_1 y Θ_2 son los espacios de parámetros donde residen $\mathbf{W}$ y $\mathbf{w}$ respectivamente. Como se mostró anteriormente, en la práctica se termina optimizando el contraparte muestral de esta expresión:

$$\hat{f}(\mathbf{x}) = argmin_{\substack{\mathbf{W}\in\Theta_1 \\ \mathbf{w}\in\Theta_2}} \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, f\left(\mathbf{x}_i; \mathbf{W}, \mathbf{w}\right)\right) \tag{10}$$

Como bien menciona Lehrbass (2021), por propiedades de los grandes números, $\hat{f}(\mathbf{x})$ es un estimador fuertemente consistente de $f^*(\mathbf{x})$. Pero, ¿cómo se minimiza efectivamente la función de pérdida respecto al conjunto de parámetros? La forma más común es aplicando un algoritmo iterativo conocido como "descenso por el gradiente" (*gradient descent*) el cual asume que la función $L\left(y,\ \hat{y}(\mathbf{W}, \mathbf{w}; \mathbf{x})\right)$[5] es diferenciable al menos una vez respecto a $(\mathbf{W}, \mathbf{w})$ y, por lo tanto, existe su gradiente $\nabla_{\mathbf{W},\mathbf{w}} L\left(y,\ \hat{y}(\mathbf{W}, \mathbf{w}; \mathbf{x})\right)$. Este algoritmo puede dividirse en dos etapas que se suceden hasta encontrar el mínimo (global) de la función de costo:

1. *Forward-pass*: dados $\mathbf{W}, \mathbf{w}$, se calcula el error de estimación entre el valor predicho por la red $\hat{y}$ y el verdadero valor de la variable a predecir y. Este dependerá no solo de la discrepancia entre ambos sino de la función de pérdida elegida.

2. *Backward-pass*: dado un input $\mathbf{x}$, se calcula el gradiente[6] de la función de pérdida respecto a $\mathbf{W}$ y $\mathbf{w}$. El gradiente negativo da la dirección de descenso más pronunciada sobre la superficie de la función de pérdida y la dirección en la cual ajustar $\mathbf{W}$ y $\mathbf{w}$ para reducir $L\left(y,\ \hat{y}(\mathbf{W}, \mathbf{w}; \mathbf{x})\right)$. El ajuste en la iteración k-esima se calcula de la siguiente manera:

$$\begin{aligned} \mathbf{W}_k &\leftarrow \mathbf{W}_{k-1} - \gamma \nabla_{\mathbf{W}_{k-1}} L\left(y,\ \hat{y}(\mathbf{W}, \mathbf{w}; \mathbf{x})\right) \\ \mathbf{w}_k &\leftarrow \mathbf{w}_{k-1} - \gamma \nabla_{\mathbf{w}_{k-1}} L\left(y,\ \hat{y}(\mathbf{W}, \mathbf{w}; \mathbf{x})\right) \end{aligned} \tag{11}$$

   donde γ es un hiperparámetro que controla el tamaño de ajuste y suele denominarse *learning rate*.

Estos dos pasos del algoritmo son repetidos, en la práctica, hasta que se cumpla alguna regla de parada o *stopping rule*. Sin embargo, como menciona Cook (2019), su implementación adolece de dos problemas: en primer lugar, el costo computacional de calcular $\nabla_{\mathbf{W},\mathbf{w}} L\left(y,\ \hat{y}(\mathbf{W}, \mathbf{w}; \mathbf{x})\right)$ crece en la medida que se incrementa la muestra de entrenamiento; y en segundo lugar, al ser el entrenamiento de la red neuronal un problema de optimización no convexo, se corre el riesgo de que el algoritmo no alcance el mínimo global y quede iterando en un mínimo local o en un punto silla de la función de pérdida. Respecto al primer punto, en la práctica se suelen aplicar modificaciones al algoritmo de descenso por el gradiente conocidas como *Stochastic Gradient Descent* y *Mini-batch Stochastic Gradient Descent*, en donde el gradiente se calcula para cada observación por separado ($n = 1$) o para un conjunto de observaciones o *batch* ($n = b$), en lugar de hacerlo para la totalidad de la muestra de

[5] Notar que en este punto se toma como fijo $\mathbf{x}$ y se considera $\hat{y}$ una función de sus parámetros.

[6] El gradiente respecto a los parámetros se computa aplicando otro algoritmo llamado *back-propagation* (Rumelhart, Hinton, & Williams, 1986), el cual es una generalización de la regla de la cadena del cálculo multivariable

entrenamiento. En lo que refiere al segundo punto, y para intentar evitar mínimos locales, se propusieron mejoras en (11) que emplean learning rates adaptativas y/o nociones de momentum: la idea general es que el algoritmo tome "pasos más largos" en las direcciones del gradiente más pronunciadas y relativamente estables, mientras toma "pasos más cortos" en aquellas direcciones en donde el gradiente comienza a aplanarse o es relativamente más volátil. Entre las implementaciones más utilizadas en la práctica se encuentran RMSprop (Root Mean Square Propagation), AdaGrad (Adaptative Gradient Algorithm) y Adam (Adaptive Moment Estimation).

No obstante, existe un problema inherente a las redes neuronales densas basadas en unidades de tipo perceptrón: es una arquitectura diseñada para tratar con inputs cuyas variables son independientes entre sí, algo que no ocurre cuando se trabaja con datos secuenciales como las series de tiempo. Estas redes carecen de mecanismos nativos que les permitan extraer las dependencias temporales que existen entre cada componente del vector $\mathbf{x}_t$ y que ayudan a predecir y_{t+h}. Si bien es cierto que es posible incorporar ciertas dependencias temporales en los datos reestructurando los inputs como en un modelo de rezagos distribuidos, este abordaje aumenta el tamaño del vector de inputs $\mathbf{x}_t$ requerido a la vez que incrementa en forma más que proporcional el tamaño del espacio de parámetros sobre el que buscar la solución del problema de optimización. Adicionalmente, exige que todos los inputs del modelo sean del mismo tamaño, impidiendo trabajar con secuencias de dimensión variable (Cook, 2019). Con el fin de resolver estos inconvenientes y tratar con datos secuenciales de la manera apropiada surgieron las redes neuronales recurrentes (RNN).

A diferencia de las redes neuronales densas, las redes recurrentes fueron diseñadas para procesar datos secuenciales gracias a los mecanismos de "memoria" que incorporan, los cuales permiten transmitir información pasada (el estado del sistema) a las observaciones futuras de la secuencia. La forma más simple de persistir información pasada relevante a través del tiempo y dar cuenta de las dependencias temporales en la data es incorporando a la red canales de retroalimentación o *feedback loops*. En consecuencia, la red ya no solo contará con conexiones en una sola dirección (de una capa a la siguiente) sino que las unidades recurrentes pueden transmitir información en la misma capa. En el siguiente diagrama se presenta la neurona recurrente más simple llamada unidad de Elman (Hewamalage, Bergmeir, & Bandara, 2021):

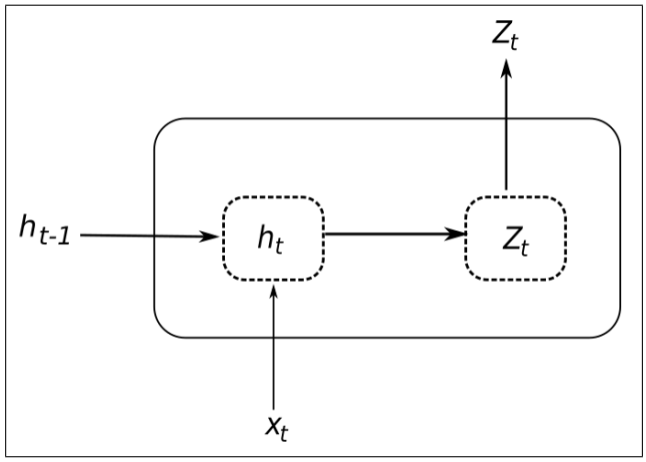

Figura 4: Unidad recurrente de Elman

A diferencia del perceptrón, las unidades de Elman cuentan con un estado oculto h_t que constituye la memoria de la red y permite propagar información relevante entre *timesteps* y así poder predecir el siguiente elemento en la secuencia. Las ecuaciones que definen la unidad de Elman son las siguientes:

$$h_t = \sigma_i \left(\mathbf{W_i}\, h_{t-1} + \mathbf{V_i}\, x_t + \mathbf{b_i} \right) \tag{12}$$

$$z_t = \sigma_o \left(\mathbf{W_o}\, h_t + \mathbf{b_o} \right) \tag{13}$$

donde $x_t \in \mathbf{R}^m$ es una componente del vector de input $\mathbf{x}_t$ (la observación t-esima de la secuencia), h_{t-1}, $h_t \in \mathbf{R}^d$ son vectores que representan el estado oculto de la red en los momentos $t-1$ y t y $z_t \in \mathbf{R}^d$, la salida de la neurona. Por su parte, W_i, W_o, V_i, b_i y b_o son los parámetros a estimar para la unidad y σ_i, σ_o son funciones de activación (usualmente, σ_i es una función tangente hiperbólica y σ_o, sigmoide). Cómo se observa en la ecuación (12), el estado de la red en el momento t depende del estado en el momento inmediato anterior $t-1$ así como el input en t.[7] La red neuronal debe aprender qué información pasada recordar y cuál olvidar (la matriz de parámetros W_i viene a cumplir esa función) y cómo combinarla con datos actuales (matriz V_i) para actualizar el estado del sistema. Y así, una vez actualizado el estado oculto de la red, este se usa para producir el output de la neurona en (13). Asumiendo que x_t es la última componente del vector de input, z_t o una transformación de este vector (por ejemplo, añadiendo una capa densa que transforme el vector en un escalar) se convierte en la predicción de la red $\hat{y}_{t+h}$.

Vale la pena mencionar que esta misma unidad se aplica de manera sucesiva sobre todos los componentes del vector de input, compartiendo los parámetros W_i, W_o, V_i, b_i y b_o (ver figura 5). Esto no solo reduce el número de parámetros a optimizar sino que perimite que la misma red pueda procesar secuencias de distinto largo sin tener que adaptar su arquitectura. Y, al igual que en las redes neuronales densas, es posibles concatenar varias unidades de Elman en la misma capa (cada una "mirará" aspectos distintos de la secuencia) y apilar varias capas de unidades recurrentes (en ese caso, z_t se convertirá en el input de la siguiente capa).

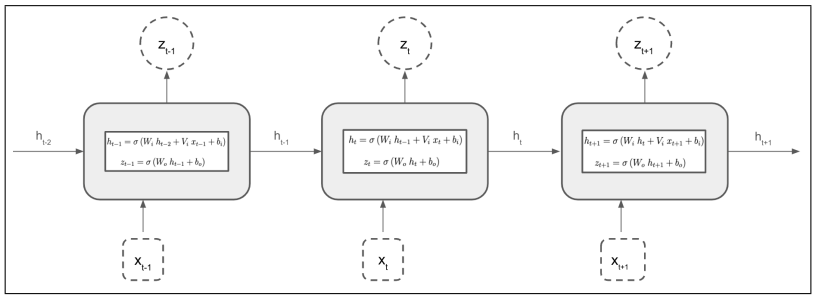

Figura 5: Versión *unrolled* de la RNN con una unidad de Elman

Al igual que las redes neuronales densas, las redes neuronales recurrentes se entrenan utilizando el gradiente de la función de pérdida respecto a los parámetros de la red pero, a diferencia de las primeras, se utiliza un algoritmo diseñado para

[7] Si se aplica la ecuación (12) de manera recursiva se obtiene $h_t = \sigma\left(W_i\, \sigma\left(W_i\, \sigma\left(W_i\ \left(\ldots\left(\sigma\left(W_i\, h_0 + V_i\, x_1 + b_i\right)\ldots\right) + V_i\, x_{t-2} + b_i\right) + V_i\, x_{t-1} + b_i\right) + V_i\, x_t + b_i\right)\right)$, es decir que el estado de la red al momento t depende de la respuesta del modelo a todos las observaciones anteriores a t y del valor inicial del estado h_0, hiperparámetro de la red.

datos secuenciales llamado *Backpropagation Trough Time* o BPTT: el mismo funciona "desenrrollando" la red neuronal en timesteps y, dado que los parámetros se comparten para toda la secuencia, sumando la contribución de cada timestep al gradiente total (Krishna, 2022). No obstante, y como puede observarse en el Apéndice A, las RNN más simples basadas en unidades de Elman sufren de un problema conocido como *vanishing gradient* o desvanecimiento del gradiente: en presencia de secuencias relativamente largas (en la práctica, más de diez timesteps), la red tiende a olvidar/desestimar la información más antigua de la secuencia lo cual dificulta que las primeras observaciones de la serie puedan ayudar a predecir observaciones futuras. Este fenómeno ocurre en la fase de backpropagation del entrenamiento de la red: en la medida que el gradiente es propagado hacia atrás por cada uno de los timesteps, este colapsa a cero en tanto es producto de sucesivos factores en el intervalo $(0, 1]$. Como resultado, los pesos y sesgos de la red se actualizan lentamente o no lo hacen en lo absoluto.

La solución al problema de vanishing gradient debe rastrearse en el artículo seminal de Hochreiter y Schmidhuber (1997): las unidades *long short-term memory* o LSTM, las cuales fueron diseñadas para mitigar los problemas inherentes al entrenamiento de RNN en presencia de secuencias largas como lo suelen ser las series de tiempo. Para lograrlo, estas unidades cuentan con módulos especiales llamados puertas o *gates* que permiten propagar el gradiente de manera continua aún a los primeros timesteps de la secuencia, posibilitando que los parámetros puedan ajustarse. Pero las unidades LSTM son más que eso: las mismas puertas que les permiten sortear el desvanecimiento del gradiente ayudan a distinguir entre eventos recientes (memoria de corto plazo) de eventos lejanos, muy atrás en la secuencia de datos (memoria de largo plazo); mientras la misma red olvida información irrelevante que no ayuda a predecir el siguiente output. De esta manera, las unidades LSTM son capaces de capturar dependencias de largo plazo en los inputs.

En el siguiente gráfico (Björnsjö, F., 2020) puede apreciarse la arquitectura de una unidad LSTM:

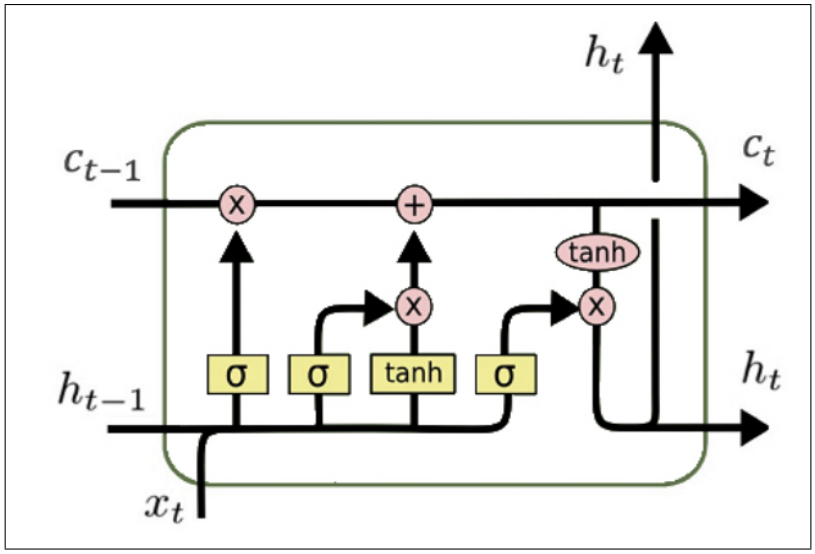

Figura 6: Unidad LSTM

A diferencia de las unidades de Elman, las unidades LSTM no solo consideran

el input $\mathbf{x}_t$ y el estado oculto $\mathbf{h}_{t-1}$ sino que incopora una memoria de largo plazo llamada *cell state* y simbolizada con $\mathbf{c}_{t-1}$. Además, la neurona cuenta con tres compuertas que regulan el flujo de información entre timesteps y permiten estimar el output: la *input gate* o puerta de ingreso, la *forget gate* o puerta de olvido y la *output gate* o puerta de salida. Así, la información de largo plazo, más inmanente al sistema, es almacenada en el cell state y actualizada en cada forward pass por la puerta de olvido y la puerta de ingreso. Estas puertas deciden que información olvidar y cual incorporar al nuevo cell state $\mathbf{c}_t$ respectivamente, propagando información de manera selectiva más adelante en la secuencia. Luego, el estado oculto $\mathbf{h}_t$ es actualizado mediante la puerta de salida, en combinación con $\mathbf{c}_t$, para producir el output de la neurona, output que puede utilizarse directamente para predecir y_{t+h} (concatenando un capa densa que lo transforme en un escalar) o servir de input para otra capa de neuronas LSTM (Björnsjö, F., 2020).

El flujo de información a través de una neurona LSTM puede describirse matemáticamente como sigue:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_{\mathbf{f},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{f},\mathbf{x}}\, x_t + \mathbf{b}_{\mathbf{f}}\right) \\
\mathbf{i}_t &= \sigma\left(\mathbf{W}_{\mathbf{i},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{i},\mathbf{x}}\, x_t + \mathbf{b}_{\mathbf{i}}\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_{\mathbf{o},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{o},\mathbf{x}}\, x_t + \mathbf{b}_{\mathbf{o}}\right) \\
\tilde{\mathbf{c}}_t &= \tanh\left(\mathbf{W}_{\mathbf{c},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{c},\mathbf{x}}\, x_t + \mathbf{b}_{\mathbf{c}}\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)
\end{aligned} \tag{14}$$

donde las primeras tres funciones definen las puertas de la neurona, mientras que las segundas tres, el estado oculto y el cell state. Al igual que en el caso anterior $x_t \in \mathbf{R}^m$ es una componente del vector de input $\mathbf{x}_t$ (la observación t-esima de la secuencia) y $h_{t-1} \in \mathbf{R}^d$ es el estado oculto de la red en el momento $t-1$. Las matrices de pesos $\mathbf{W}_{\mathbf{f},\mathbf{h}}$, $\mathbf{W}_{\mathbf{i},\mathbf{h}}$, $\mathbf{W}_{\mathbf{o},\mathbf{h}}$, $\mathbf{W}_{\mathbf{c},\mathbf{h}} \in \mathbf{R}^{d\,\mathrm{x}\,d}$ y $\mathbf{V}_{\mathbf{f},\mathbf{h}}$, $\mathbf{V}_{\mathbf{i},\mathbf{h}}$, $\mathbf{V}_{\mathbf{o},\mathbf{h}}$, $\mathbf{V}_{\mathbf{c},\mathbf{h}} \in \mathbf{R}^{d\,\mathrm{x}\,m}$, al igual que los biases $\mathbf{b}_{\mathbf{f}}$, $\mathbf{b}_{\mathbf{i}}$, $\mathbf{b}_{\mathbf{o}}$, $\mathbf{b}_{\mathbf{c}} \in \mathbf{R}^d$ son los parámetros del modelo a estimar y se aplican para todos los timesteps, sin alterarlos en la medida que se avanza en la secuencia. Serán estos parámetros estimados los que doten a la neurona de la capacidad para decidir que información preservar, olvidar o incorporar a la memoria de la misma. Por su parte, σ y tanh son las funciones de activación sigmoidea y tangente hiperbólicas aplicadas componente a componente sobre la transformación afín entre paréntesis, y $\odot$ es el producto de Hadamard.

Pero, ¿cómo opera esta unidad durante el forward-pass? El primer detalle a observar es que todas las puertas -así como $\tilde{c}_t$- reutilizan el mismo input (x_t y h_{t-1}), aunque mantienen matrices de pesos y biases propias para cada puerta. En cuanto a su funcionamiento, es conveniente presentarlo siguiendo una secuencia lógica:

1. Propuesta de un nuevo candidato a *cell state* $\tilde{\mathbf{c}}_\mathbf{t}$: se construye un nuevo candidato a representar la memoria de largo plazo en base a información reciente. Para ello se aplica sobre una combinación lineal de x_t y h_{t-1} una función tangente hiperbólica componente a componente, la cual tiene por codominio el intervalo $[-1, 1]$.[8]

[8]El uso de la función tanh se prefiere por sobre la sigmoide en tanto su gradiente adopta un rango más amplio de valores ($[0, 1]$ frente a $[0, 0.25]$), previniendo el desvanecimiento del gradiente.

2. Cómputo de la *forget gate* $\mathbf{f}_t$: esta puerta cumple la función de decidir que porción del *cell state* en $t-1$ mantener y cual olvidar en t: al aplicar una función sigmoidea con codominio entre 0 y 1, está puerta olvidará de la memoria de largo plazo toda componente que quede multiplicada por un valor cercano a 0, mientras perdurará hacia el futuro las componentes del cell state que sean multiplicadas por un valor cercano a 1. Así, por ejemplo, un valor del 0.25 en la primera posición de la puerta indica que solo se retendrá el 25 % de la información representada en la componente 0 del vector $\mathbf{c}_{t-1}$.

3. Cómputo de la *input gate* $\mathbf{i}_t$: la misma decide que porción del nuevo candidato a memoria de largo plazo $\tilde{\mathbf{c}}_\mathbf{t}$ retener y cual rápidamente olvidar usando nuevamente una función sigmoide para determinarlo.

4. Creación del nuevo cell state $\mathbf{c}_t$: el nuevo cell state, el cual fluirá hacia el futuro, surge de combinar la memoria de largo plazo existente $\mathbf{c}_{t-1}$ con un nuevo candidato $\tilde{\mathbf{c}}_\mathbf{t}$, el cual incorpora información reciente que merece reconfigurar la memoria existente y será de utilidad en algún momento más adelante en la secuencia.

5. Cómputo de la *output gate* $\mathbf{o}_t$: sobre la base de la información reciente (x_t y h_{t-1}), esta puerta regula qué porción de $\mathbf{c}_t$ se va a utilizar como output de la red usando nuevamente una transformación afín seguida de una función sigmoidea.

6. Generación del output de la neurona $\mathbf{h}_t$: en esta última instancia del flujo, se decide que porción del cell state actual usar para predecir el valor de la variable de interés. Cabe mencionar que el output de la unidad coincide con el estado oculto propagado hacia el siguiente elemento en la secuencia.

Por último, y por tratarse de un problema de *forecasting*, la red debe transformar el vector de estado oculto $\mathbf{h}_t \in \mathbf{R}^d$ -el cual codifica la mayor información posible sobre la secuencia- en la estimación escalar $\hat{y}_{t+h}$ y de ese modo poder entrenar la red bajo un enfoque supervisado. Para ello, y como puede verse en el siguiente gráfico, a la salida de la neurona se le concatena una capa densa con función de activación lineal de manera de convertir $\mathbf{h}_t$ en un escalar.

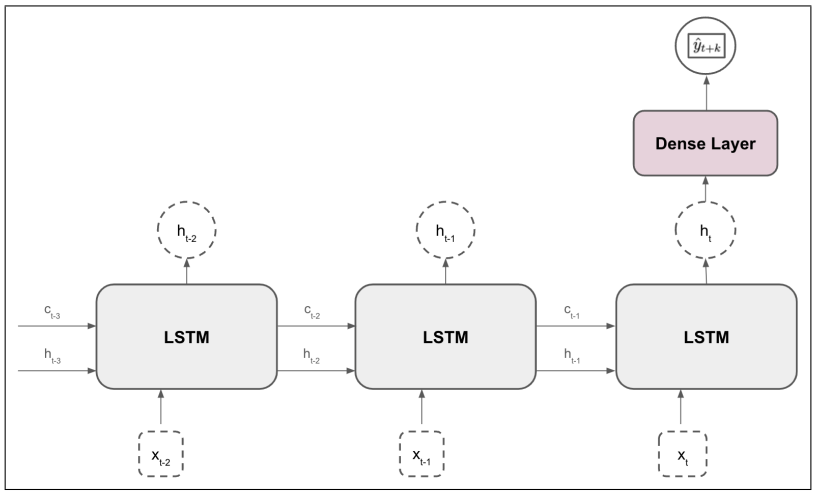

Figura 7: Versión *unrolled* de una RNN con una unidad LSTM

Al igual que las RNN basadas en unidades de Elman, las redes basadas en unidades LSTM son entrenadas utilizando *backpropagation-through-time*. Sin embargo, y gracias al uso del mecanismo de puertas, el problema del desvanecimiento del gradiente es mucho menos acuciante. Esto se debe a que el cell state no es modificado directamente por los parámetros del modelo (pesos y biases), sino indirectamente a través del mecanismo de puertas. Esto permite que la memoria de largo plazo de la red fluya libremente a través de la secuencia de timesteps sin causar que el gradiente colapse a cero.

Detallado el funcionamiento de las unidades LSTM y su superioridad respecto a las unidades de Elman[9], en el siguiente apartado se profundiza en las arquitecturas de redes neuronales que toman a las unidades LSTM como bloque fundamental. Serán en última instancia estas arquitecturas las que competirán con modelos econométricos tradicionales en la tarea de predecir una variable en contexto de *regime-switching*.

3.2 Arquitecturas basadas en unidades LSTM

Existen muchas redes neuronales basadas en unidades LSTM, la gran mayoría surgidas en el ámbito del procesamiento del lenguaje natural y luego adaptadas a otros problemas de tipo secuencial. En particular, en este apartado se profundizará en tres arquitecturas: *stacked*, *stacked* con *peepholes* y *encoder-decoder*.

Para entender la arquitectura *stacked* o multicapa, vale la pena volver un segundo a la descripción de las unidades LSTM: en la práctica, no suele usarse una única unidad LSTM sobre el vector de input $\mathbf{x}_t$ sino que un conjunto de unidades, cada una con sus respectivos parámetros pero compartiendo las dimensiones del estado oculto y el cell state, es aplicado de manera sucesiva sobre la secuencia. El output, en consecuencia, no será el vector $\mathbf{h}_t$ sino una matriz $\mathbf{H}_t \in \mathbf{R}^{d \times p}$, donde d es la dimensión del estado oculto/cell state y p, el número de neuronas de tipo LSTM concatenadas, siendo ambos hiperparámetros del modelo. Todo este bloque conforma una única capa de la red neuronal. Entonces, una red neuronal LSTM multicapa es una red neuronal multicapa en donde la salida de cada timestep $\mathbf{H}_t$ es usado como input de la capa inmediata siguiente y la predicción final es tomada de la última capa de la red. Esto puede observarse en la Figura 8 en donde se presenta una red neuronal LSTM apilada con dos capas ocultas, cada una de ellas con tres neuronas:

Al igual que en una RNN simple de una capa y una neurona, los parámetros de esta arquitectura multicapa se entrenan usando BPTT. Para ello, durante el forward pass, el error es calculado utilizando la salida del último elemento de la secuencia, descartándose las salidas de los timesteps anteriores.

[9]En este trabajo, se dejan de lado las unidades conocidas como *Gated Recurrent Unit* -o GRU- ya que, a pesar de ser más simples que las LSTM, en la práctica no gozan de la popularidad de estas últimas.

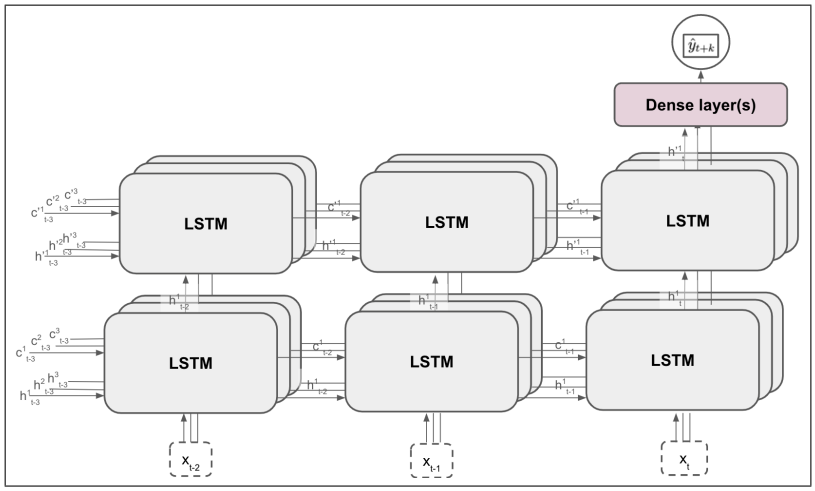

Figura 8: Red neuronal LSTM multicapa

La siguiente arquitectura por explorar es la de las redes neuronales LSTM con *peepholes connections*, introducidas por Gers y Schmidhuber (2000). Estas redes neuronales se caracterizan por modificar la estructura de puertas en las unidades LSTM: ahora las gates no solo se calculan en base a información reciente ($\mathbf{h}_{t-1}$ y $\mathbf{x}_t$) sino también a la información de largo plazo que fluye en la cell state. Matemáticamente:

$$\begin{aligned} \mathbf{f}_t &= \sigma\left(\mathbf{W}_{\mathbf{f},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{f},\mathbf{x}}\, x_t + \mathbf{P}_{\mathbf{f},\mathbf{c}}\, c_{t-1} + \mathbf{b}_{\mathbf{f}}\right) \\ \mathbf{i}_t &= \sigma\left(\mathbf{W}_{\mathbf{i},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{i},\mathbf{x}}\, x_t + \mathbf{P}_{\mathbf{i},\mathbf{c}}\, c_{t-1} + \mathbf{b}_{\mathbf{i}}\right) \\ \mathbf{o}_t &= \sigma\left(\mathbf{W}_{\mathbf{o},\mathbf{h}}\, h_{t-1} + \mathbf{V}_{\mathbf{o},\mathbf{x}}\, x_t + \mathbf{P}_{\mathbf{o},\mathbf{c}}\, c_t + \mathbf{b}_{\mathbf{o}}\right) \end{aligned} \tag{15}$$

Los *peepholes* funcionan como conexiones que permiten que la información de largo plazo puede combinarse con la de corto plazo antes de ser modulada por la output gate, algo que no ocurre en una una LSTM *vanilla*.

Finalmente, la última arquitectura basada en neuronas LSTM a mencionar en este apartado es la conocida como *encoder-decoder* o *autoencoder* (Sutskever et. al., 2014). Surgida cómo casi todas las RNN actuales para resolver problemas de NLP, las redes encoder-decoder son en realidad dos redes neuronales -en el presente trabajo, dos redes LSTM aunque no necesariamente deben compartir la misma arquitectura- trabajando en conjunto: un *encoder* que transforma la secuencia en un vector de dimensión fija -llamado vector de contexto- que representa la serie temporal en un espacio latente, y una red *decoder* que decodifica dicho vector devuelta en una secuencia. Es por ello que estos modelos son llamados habitualmente "secuencia a secuencia" o *seq2seq*: transforman una secuencia de entrada en una secuencia de salida, ambas de dimensiones variables.

El encoder de la red no es más que una red neuronal LSTM multicapa, en donde se procesa cada componente de la secuencia de input $\mathbf{x}_t$ uno a la vez. La salida de esta red, una vez procesado el último timestep, es un conjunto de vectores de estado oculto y cell states de dimensión igual al número de neuronas por capa por el número de capas que conforman el encoder. A este conjunto de vectores de dimensión fija se los conoce como vectores de contexto y representan a la secuencia como un todo en un espacio latente.

El decoder, por su parte, es otra red neuronal LSTM multicapa -con sus propios pesos y sesgos- en donde su estado oculto y cell state iniciales ($\mathbf{h}_0$ y $\mathbf{c}_0$) son inicializados usando los vectores de contexto del encoder. En cuanto a la secuencia que procesa el decoder, esta se compone de la última componente de $\mathbf{x}_t$ y del vector de outputs $\mathbf{y}_{t+h}$, donde h es el horizonte máximo de predicción. En otras palabras, y como puede verse en el gráfico 9, durante el entrenamiento de la red se utilizan dos secuencias: la secuencia de inputs y la secuencia de targets, siendo estos últimos los valores que la red debe aprender a predecir.[10]

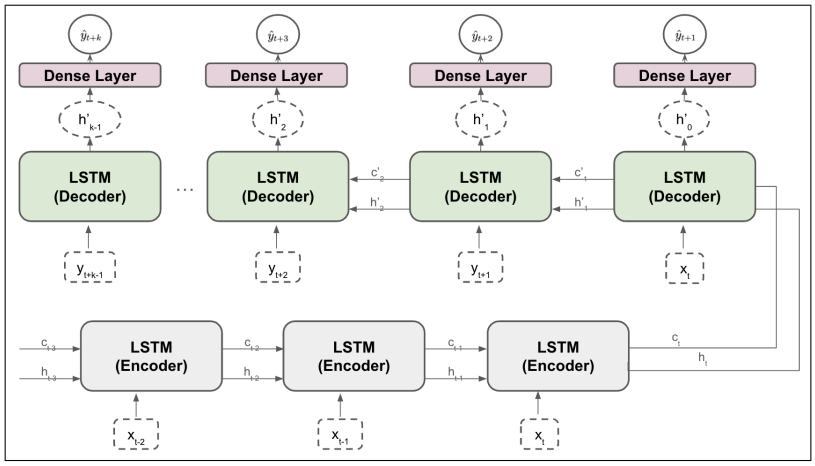

Figura 9: Autoencoder en fase de entrenamiento

El error computado durante el forward-pass se construye solo a partir de las predicciones del decoder -luego de pasar por una red neuronal densa que transforme el vector en un escalar- y es un error *multi-step* sobre todo el horizonte de predicción, a saber:

$$e = \sum_{i=1}^{h} y_{t+i} - \hat{y}_{t+i}$$

Este error, durante la fase de entrenamiento, es propagado hacia atrás en la red usando el mismo algoritmo de backpropagation-throught-time, ajustando los pesos y sesgos del decoder y del encoder conjuntamente.

La figura 10 muestra como tiene lugar la inferencia para el mismo modelo: en tanto se desconocen los verdaderos targets a predecir $\mathbf{y}_{t+h}$, se introducen conexiones autorregresivas en donde el output de un timestep es utilizado como input del siguiente.

[10]Este abordaje por el cuál los inputs del decoder lo constituyen los verdadero valores a predecir y_{t+i} en lugar de sus estimaciones $\hat{y}_{t+i}$ recibe el nombre de *teacher forcing*: se le busca enseñar al decoder cuán impreciso fue al predecir el output anterior y como debería corregirlo en el siguiente timestep (Hewamalage et. al., 2021 2019).

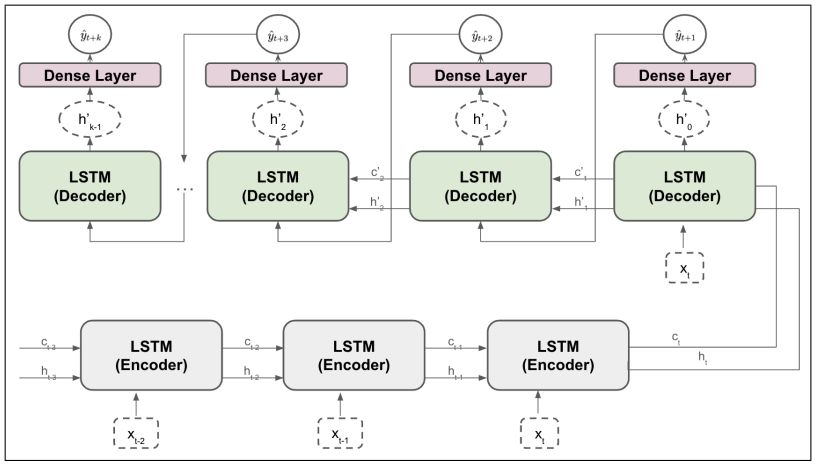

Figura 10: Autoencoder en fase de inferencia

4. Experimentos

4.1 Configuración general

En esta fase experimental del trabajo, como se mencionó anteriormente, se busca comparar la capacidad predictiva de las arquitecturas de redes neuronales recurrentes frente a modelos estadísticos tradicionales, particularmente los modelos ARIMA y ETS (o *Error-Trend-Seasonal models*). Estos experimentos controlados han sido implementados enteramente en Python y pueden encontrarse en este repositorio de GitHub.

El repositorio se estructura de la siguiente manera: en la carpeta `utils` podrá encontrarse el código fuente del proyecto, separado en el módulo de simuladores (`simulators.py`) y los dos módulos de estimadores (`statistical_models.py` y `NN_models.py`), correspondientes a los modelos econométricos y a las arquitecturas de redes neuronales respectivamente. Por su parte, en la carpeta `notebooks` se pueden encontrar las subcarpetas `simulaciones` y `bitcoin`: mientras que la primera contiene tres experimentos con datos sintéticos simulados, la segunda contiene un caso real de forecasting de los precios de cierre del par BTC-USD, el cual se retomará en la siguiente sección.

Antes de profundizar en cada experimento, vale la pena exponer como se construyó cada una de las clases de Python que constituyen los estimadores puestos a competir:

1. `ArimaModel`: esta clase funciona como *wrapper* de la clase `ARIMA` de la librería `pmdarima`, implementando adicionalmente una búsqueda automática (o cross-validation) de la mejor configuración de hiperparámetros del modelo ARIMA, esto es, del orden de la componente autorregresiva AR (p), del orden de la componente de medias móviles MA (q) y del orden de integración I (d). Para ello, y para un espacio de búsqueda de hiperparámetros construido como el producto cartesiano de los conjuntos $P = \{0, 1, 2, 3\}$, $D = \{0, 1, 2\}$ y $Q = \{0, 1, 2, 3\}$, se aplica una estrategia de validación cruzada de tipo *sliding window* en donde el conjunto de entrenamiento se construye a partir de una ventana móvil de longitud fija (un hiperparámetro adicional llamado `window_length`) y el conjunto de validación, como la observación que se encuentra h

(el horizonte de forecasting) timesteps por delante de la última observación de la ventana móvil. En el siguiente esquema se presenta como se construiría los *folds* para la validación cruzada bajo esta estrategia, con un window_length igual a 5 timesteps, un *stride* de 3 y un horizonte de forecasting h de 2:

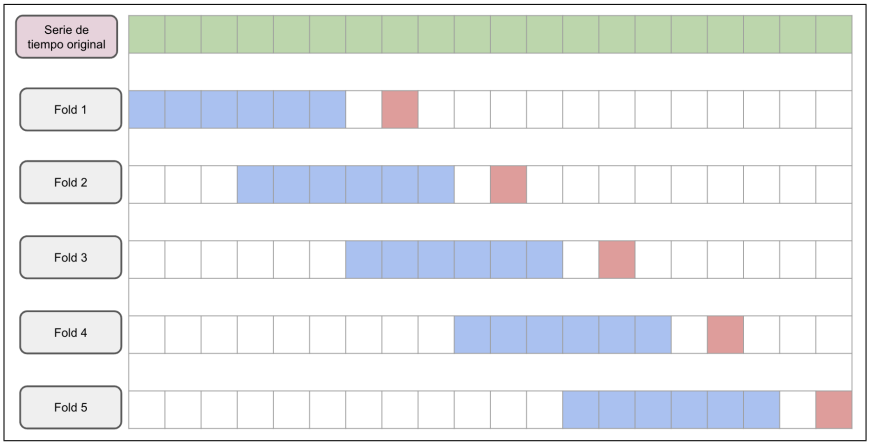

Figura 11: Estrategia de cross-validation de tipo sliding window

Para el caso todos los modelos ARIMA estimados en cada uno de los escenarios, el valor del stride se fija en 25 timesteps, mientras que se construye un conjunto de tres valores posibles -$w \in \{25, 50, 100\}$- que puede adoptar el window_length: la configuración óptima surgirá de la validación cruzada conjunta con los otros hiperparámetros del modelo. En todos los casos, se busca predecir dos horizontes de tiempo fijos: $h \in \{1, 7\}$.

Una vez construido los folds[11], se recorre el espacio de hiperparámetros entrenando el modelo con el conjunto de entrenamiento y evaluándolo con las observaciones reservadas para validación. La configuración ganadora será la tupla $(p^*, d^*, q^*, w^*, stride = 25)$ cuyo error cuadrático medio en el conjunto de validación sea el más pequeño. Una vez encontrado este conjunto de hiperparámetros, se reentrena el modelo ARIMA hasta su convergencia utilizando el algoritmo de Nelder-Mead.

2. `EtsModel`: esta clase es un wrapper de la clase ETS de la librería `statsmodels` e implementa una serie de modelos de tipo *state-space*, entre los que se encuentra el modelo de suavizamiento exponencial y el modelo aditivo de Holt-Winters. Todos estos modelos se caracterizan por descomponer a la serie temporal en sus partes componentes: tendencia (T), estacionalidad (S) y error (E).

   Al igual que para el modelo ARIMA, se deben cross-validar los hiperparámetros del modelo, aquellos que definen las forma funcional de las ecuaciones del sistema. Con dicho objetivo, se construye el espacio de hiperparámetros como el producto cartesiano de los siguientes conjuntos:

   - Tipo de error del modelo: {aditivo, multiplicativo}
   - Tipo de tendencia del modelo: {aditiva, multiplicativa, null}

[11] El número de folds se determina de manera automática en base a la longitud de la serie temporal, el window_length y el stride o gap entre folds.

- Tendencia amortiguada: {Verdado, Falso}

Se excluye la posibilidad que el modelo incorpore una componente estacional en tanto ninguno de los DGP simulados contiene fluctuaciones estacionales. Y, al igual que el modelo ARIMA, se cross-valida de manera simultánea el window_length junto con los demás hiperparámetros del modelo usando la estrategia de validación sliding window. Así, para cada horizonte de pronóstico, existirá un modelo ETS óptimo, el cual se reentrenará finalmente utilizando el algoritmo L-BFGS, algoritmo de la familia de métodos quasi-Newton.

3. `LSTMnetwork`: para la construcción de las redes neuronales recurrentes se utilizó la librería `Keras` de Python, la cual funciona como una API o *framing* de alto nivel montado sobre el backend de Tensorflow. Para el caso de las redes LSTM apiladas -o stacked LSTM- se utilizó la API secuencial de Keras, la cual permite construir la arquitectura final como la concatenación de capas de neuronas.

   El primer paso en el entrenamiento de la red neuronal lo constituye el preprocesamiento de los datos. Así, primero se estandarizó la variable restando la media y dividiendo por el desvío estándar de la muestra de entrenamiento[12], de manera que los datos queden centrados en 0 y de esa manera el entrenamiento de la red será más estable. A continuación, se construyó un conjunto de validación tomando el 33 % de los datos finales del conjunto de entrenamiento original. El siguiente paso fue transformar los datos a un formato que fuera aceptado por la red y que reflejara la naturaleza supervisada del problema. Así, el input de la red lo constituyen tensores 3-dimensionales cuyas dimensiones representan el *batch_size* (número de secuencias procesadas de manera conjunta por la red), el *seq_length* (la dimensión de la secuencia) y el número de features (en este caso 1 en tanto se trata de una serie univariada); siendo los dos primeros hiperparámetros a optimizar. En cuanto al output, se trata del escalar que se intenta predecir con un horizonte temporal igual a los casos anteriores, $h \in \{1, 7\}$.

   La arquitectura final de la red se determinó via cross-validation sobre un espacio de hiperparámetros dado por:

   - Inclusión de una segunda capa oculta de tipo LSTM: {Verdadero, Falso}
   - Función de activación: {relu, tanh, sigmoid, elu}
   - Número de neuronas en la capa LSTM: {16, 32, 64, 128}
   - Número de neuronas en la capa densa conectada a la salida de la última capa LSTM: {16, 32, 64, 128}
   - Learning rate: {1e-2, 1e-3, 1e-4}
   - *Dropout rate*: {0.0, 0.1, 0.2, 0.3, 0.4, 0.5}
   - *Weight decay rate* o penalización L2 sobre los pesos de la red: {0.0, 0.1, 0.2, 0.3, 0.4, 0.5}
   - Distribución con la que se inicializan los pesos de la red: {normal, glorot_uniform, he_normal}

[12] Si la serie de tiempo no es estacionaria, este abordaje puede no ser del todo correcto y convendría realizar una estandarización local, por ejemplo, por batches de datos.

- Optimizador: {Adam, RSMProp, SGD}
- Largo de la secuencia (equivalente a window_length): {2, 5, 25, 50, 100}
- *Batch size*: {32, 64, 128}

El módulo `KerasTuner` permitió cross-validar de manera aleatoria 10 combinaciones de dichos hiperparámetros durante 10 epochs. La mejor arquitectura se definió como aquella que minimiza el error cuadrático medio sobre el conjunto de validación y, una vez escogida, se pasó a entrenar dicha red durante 100 epochs. Adicionalmente, y para prevenir el overfitting, se aplicaron dos *callbacks*:

- `Reduce Learning Rate on Plateau`: el mismo reduce la learning rate por un factor de 0.1 una vez pasados 5 epochs sin que el MSE disminuya en el conjunto de validación.
- `Early Stopping`: este callback "corta" el entrenamiento de la red luego de 10 epochs en donde el MSE no se reduce sustancialmente en el conjunto de validación, evitando así el overfitting al conjunto de entrenamiento.

4. `PeepholedLSTMnetwork`: la construcción y entrenamiento de esta red es idéntica al caso anterior con la excepción que se usó la capa `PeepholeLSTMCell` del módulo `tensorflow_addons` en lugar de la capa LSTM nativa de Keras.

5. `Seq2SeqLSTM`: a diferencia de los dos modelos anteriores de redes neuronales, esta arquitectura es más compleja y requirió del uso de la API funcional de Keras. Esto permitió recrear la estructura de "embudo" típica de los Encoder-Decoder, pudiendo utilizar el vector de contexto del Encoder como valor inicial del Decoder.

   Dado que se trata de una red neuronal construída para resolver problemas de tipo *sequence-to-sequence* se hicieron adaptaciones en la forma en que se procesan y construyen los datasets de entrenamiento, validación y test: en lugar de que el output a predecir sea un escalar, este se configura como una secuencia de longitud h. Así, si $h = 7$, el Decoder deberá predecir todos los timesteps intermedios hasta llegar el horizonte temporal requerido. Así el output, al igual que el input, será un tensor de 3 dimensiones (batch_size, h, 1).

A la hora de realizar el forecasting, en todos los casos se optó por la estrategia dada a llamar "iterativa": en la medida que las observaciones de y_t se van realizando, estas se van incluyendo en el input utilizado para predecir observaciones ulteriores. Esta estrategia se la considera la más realista y un reflejo de lo que haría cualquier forecaster en la práctica: utilizar la información más reciente para mejorar la calidad de las predicciones futuras.[13]

En todos los escenarios planteados, se presentan una batería de métricas de la precisión predictiva de cada modelo para cada uno de los horizontes temporales. Se acompañan estas métricas con un test de Diebold-Mariano (1995) unilateral en

[13]En el caso de las redes neuronales, se adoptó la premisa muy extendida en la comunidad de Machine Learning que las mismas pueden predecir series temporales no estacionarias sin necesidad de ningún procesamiento adicional.

donde se contrasta la hipótesis nula que un pronóstico näive -utilizar solo la última observación realizada y_t para predecir y_{t+h}- es más o igual de preciso que aquel obtenido con uno de los modelos planteados, frente a la hipótesis alternativa que el modelo tiene mayor precisión que la estimación näive. En caso de ser de utilidad, se comparan entre sí -utilizando el mismo test- aquellos modelos que fueran estadísticamente más precisos que el pronóstico näive.

4.2 Random Walk

En este experimento se crean dos instancias de la clase `RandomWalk` del módulo de simuladores, y se simulan 1000 observaciones inicializadas en 0. Las especificaciones de cada DGP son las siguientes:

- Vanilla RW: $y_t = y_{t-1} + \epsilon_t \;/\; \epsilon_t \sim N(0, 1)$
- RW con drift y tendencia: $y_t = 0.01 + 0.0002\,t + y_{t-1} + \epsilon_t \;/\; \epsilon_t \sim N(0, 1)$

Para todos los modelos puestos a competir, los primeros 750 timesteps de la serie constituyen el conjunto de entrenamiento y los últimos 250, el conjunto de test. Esto puede verse en el siguiente gráfico:

Figura 12: Simulaciones de RW antes del cambio de régimen

Para simular un cambio de régimen, se procede a alterar discrecionalmente los parámetros de cada uno de los DGP de la siguiente manera:

- Vanilla RW: $y_t = y_{t-1} + \epsilon_t \;/\; \epsilon_t \sim N(0.25, 2)$
- RW con drift y tendencia: $y_t = 0.02 - 0.0002\,t + y_{t-1} + \epsilon_t \;/\; \epsilon_t \sim N(0, 1)$

Este cambio de régimen ocurre en el timestep $t = 750$, por lo tanto, las simulaciones lucen de la siguiente manerra:

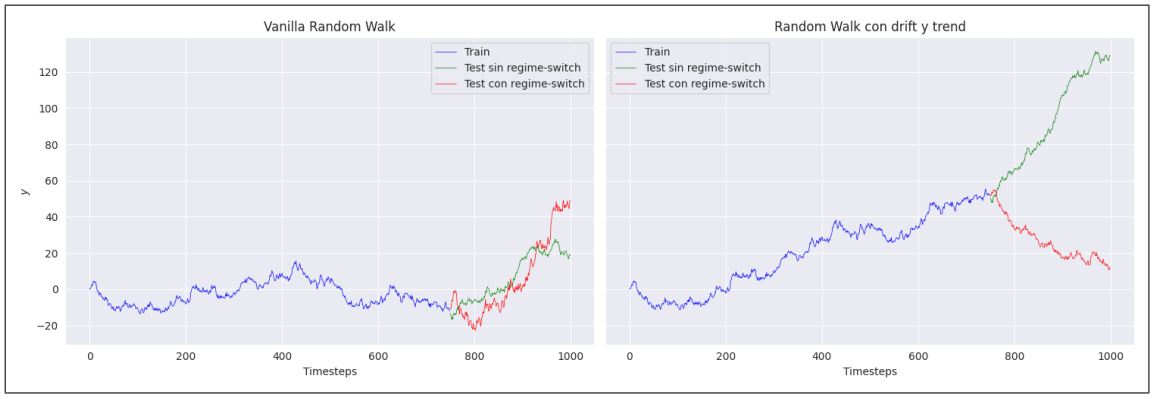

Figura 13: Simulaciones de RW luego del cambio de régimen

Una vez simuladas las series de tiempo original y el cambio de régimen, se procede a entrenar cada modelo con el conjunto de train y realizar el forecasting sobre el conjunto de test. En el Apéndice B podrán encontrarse las configuraciones de hiperparámetros resultantes de cross-validar cada uno de ellos.

Empezando por el caso original, en las figuras 15 y 16 pueden observarse las predicciones hechas por cada modelo para los dos horizontes de tiempo planteados, 1 y 7 timesteps.

Figura 14: Forecasting para el modelo *Vanilla RW*

Puede observarse a simple vista que los modelos econométricos -ARIMA y ETS- son superiores a las redes neuronales en su capacidad predictiva: los primeros parecerían incorporar más rápido la información contenida en y_t a medida que se va realizando y con ello actualizarían sus pronósticos de manera más precisa; las redes neuronales, en cambio, parecerían tener mayor inercia y depender en mayor medida

de las observaciones vistas durante su entrenamiento. Este último punto es evidente en el caso *Vanilla RW*, en donde el rango de valores que adoptan los forecasts pareciera confinado al de las observaciones en train, mientras que los verdaderos valores de y_t se encuentran por encima del máximo del conjunto de entrenamiento. Aún más notoria es la dificultad que muestran las redes neuronales (particularmente, las arquitecturas LSTM con peepholes y el Encoder-Decoder) para aprender la tendencia determinística en el caso del RW con drift y tendencia.

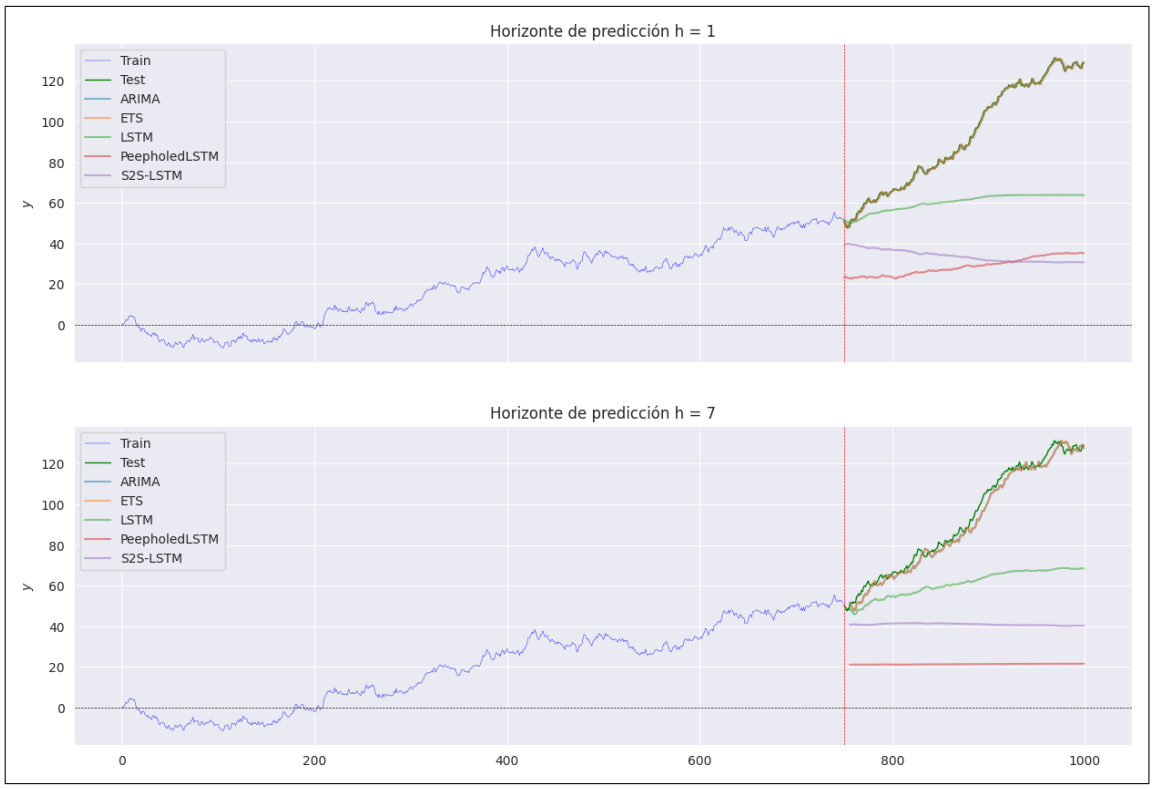

Figura 15: Forecasting para el modelo RW con drift y tendencia

Es interesante notar que para el caso vanilla, las predicciones de las redes neuronales no son "malas" para todo el conjunto de test: hasta algún momento entre los timesteps 850 y 900, las predicciones de los 5 modelos son muy similares. Casualmente este subconjunto de observaciones adoptan valores (y patrones) similares a los que se encuentran en el conjunto de train y, por lo tanto, similares a las secuencias utilizadas por las redes neuronales para su entrenamiento. Como se observa en el gráfico de los residuos (Figura 16), en algún momento entre los timesteps 850 y 900 las series de residuos divergen y esto ocurre luego de un salto abrupto, espúreo en la serie y_t; un movimiento inédito que no encuentra semejante en el conjunto de entrenamiento y que, por lo tanto, las redes neuronales no pudieron aprehenderlo con anterioridad.

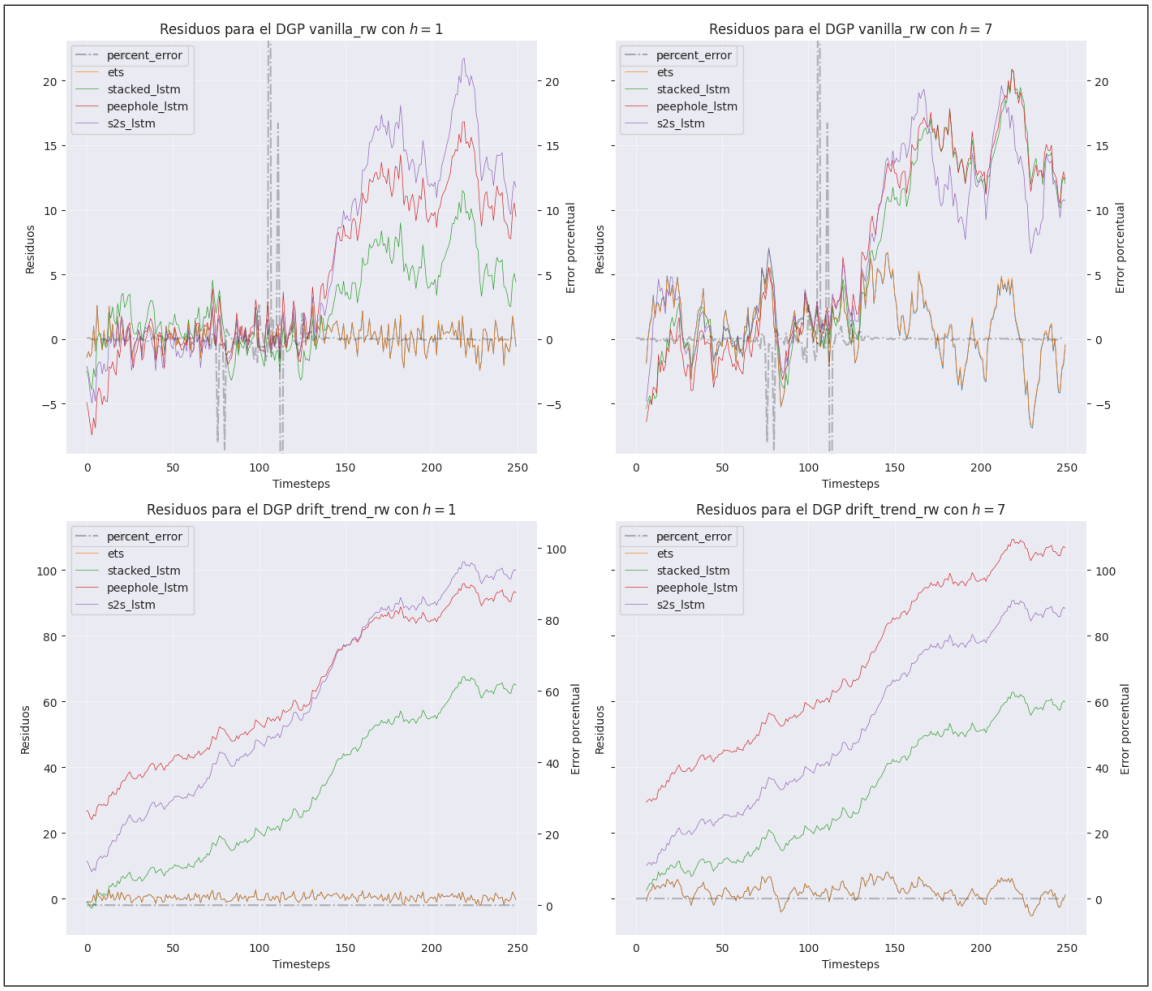

Figura 16: Serie de residuos y errores porcentuales.

A la hora de comparar métricas de precisión de la predicción (Cuadro 1) el resultado es claro: las performance predictiva de las redes neuronales en el caso base es muy inferior a la de los modelos ARIMA y ETS, e incluso, a la de la estimación näive.

DGP	horizon	metric model	MAE	MAPE	MSE	RMSE	SMAPE
vanilla_rw	h_1	naive	**0.77**	51.63	**0.94**	**0.97**	22.03
		arima	**0.77**	**50.12**	**0.94**	**0.97**	**22.01**
		ets	**0.77**	50.33	**0.94**	**0.97**	22.03
		stacked_lstm	3.45	720.80	19.98	4.47	45.43
		peephole_lstm	5.68	528.51	57.49	7.58	60.55
		s2s_lstm	6.65	453.75	90.14	9.49	66.38
	h_7	naive	**2.18**	**78.41**	**7.60**	**2.76**	**42.52**
		arima	2.19	79.02	7.65	2.77	42.54
		ets	2.19	79.11	7.61	**2.76**	42.79
		stacked_lstm	7.40	450.32	95.45	9.77	84.26
		peephole_lstm	7.86	403.26	103.16	10.16	94.48
		s2s_lstm	7.76	455.13	103.74	10.19	93.21

DGP	horizon	metric model	MAE	MAPE	MSE	RMSE	SMAPE
drift_trend_rw	h_1	naive	**0.81**	**0.95**	**1.02**	**1.01**	**0.95**
		arima	**0.81**	**0.95**	**1.02**	**1.01**	0.96
		ets	**0.81**	**0.95**	**1.02**	**1.01**	0.96
		stacked_lstm	32.75	32.15	1567.96	39.60	38.69
		peephole_lstm	63.73	65.98	4539.67	67.38	102.16
		s2s_lstm	60.26	61.90	4486.90	66.98	90.83
	h_7	naive	2.84	3.24	11.64	3.41	3.31
		arima	**2.82**	**3.22**	**11.52**	**3.39**	**3.30**
		ets	2.84	3.24	11.68	3.42	3.32
		stacked_lstm	32.07	31.72	1387.98	37.26	37.86
		peephole_lstm	72.00	74.91	5823.72	76.31	120.90
		s2s_lstm	50.61	50.19	3208.57	56.64	69.40

Cuadro 1: Comparación de métricas de precisión del forecasting antes del cambio de régimen.

En el Cuadro 2 se presenta el p-value del test de Diebold-Mariano en donde se compara cada modelo frente a la estimación näive, para cada simulación y horizonte temporal. No es sorprendente que (casi) ninguno de los modelos estimados sea significativamente más preciso que la estimación näive, en tanto es esta última la estimación óptima -aquella que minimiza el MSE- para una variable y_t la cual sigue un camino aleatorio. La excepción es el modelo ARIMA en caso de que el proceso estocástico subyacente siga un camino aleatorio con drift y tendencia y un horizonte temporal $h = 7$.

DGP	p_value h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
vanilla_rw	1	0.86	0.83	1.00	1.00	1.00
	7	0.91	0.98	1.00	1.00	1.00
drift_trend_rw	1	0.99	0.99	1.00	1.00	1.00
	7	**0.00**	1.00	1.00	1.00	1.00

Cuadro 2: p-values para diferentes modelos comparados con el pronóstico näive, antes del cambio de régimen.

La siguiente instancia en este experimento consiste en repetir el forecasting sobre el conjunto de test, pero esta vez post cambio de régimen. Se recuerda que, con el fin de analizar la robustez de las predicciones en caso de cambios en los parámetros del DGP, los modelos no son reentrenados.

En los gráficos 17 y 18 pueden observarse los nuevos pronósticos para cada uno de los modelos.

Figura 17: Forecasting para el modelo *vanilla RW* post cambio de régimen.

Una vez más, la conclusión es evidente: las predicciones de los modelos ARIMA y ETS son muy superiores a las producidas por las redes neuronales en ambos DGPs y horizontes de tiempo.

Figura 18: Forecasting para el modelo RW con drift y tendencia post cambio de régimen.

Cabe mencionar la escasa capacidad mostrada por las redes neuronales para adaptar sus pronósticos ante observaciones y patrones no observados durante el entrenamiento, lo cual es inherente a un serie temporal que ha sufrido cambios de régimen. Las redes parecen ser muy rígidas y, salvo excepciones, solo pueden retornar una predicción aproximadamente constante durante varios timesteps. Podría especularse que la red, al no identificar un patrón conocido en los datos, solo puede devolver un forecast por default. Esta conclusiones basadas solamente en los gráficos se convalidan en el Cuadro 3 en donde se analizan las métricas de precisión predictiva.

DGP	horizon	metric model	MAE	MAPE	MSE	RMSE	SMAPE
vanilla_rw	h_1	naive	**1.54**	94.47	3.78	**1.94**	31.75
		arima	**1.54**	**93.07**	**3.77**	**1.94**	31.39
		ets	**1.54**	93.15	**3.77**	**1.94**	**31.37**
		stacked_lstm	10.99	903.11	197.97	14.07	74.23
		peephole_lstm	10.06	570.63	212.16	14.57	71.41
		s2s_lstm	13.49	352.42	389.69	19.74	102.86
	h_7	naive	3.79	120.56	25.25	5.03	53.17
		arima	**3.77**	121.37	**25.03**	**5.00**	**53.06**
		ets	3.79	**119.18**	25.27	5.03	**53.06**
		stacked_lstm	13.39	383.95	373.42	19.32	99.92
		peephole_lstm	13.48	828.73	281.22	16.77	91.52
		s2s_lstm	31.69	1705.97	1846.94	42.98	128.76
drift_trend_rw	h_1	naive	**0.77**	**3.37**	**0.95**	**0.98**	**3.35**
		arima	**0.77**	**3.37**	**0.95**	**0.98**	**3.35**
		ets	**0.77**	**3.37**	**0.95**	**0.98**	**3.35**
		stacked_lstm	29.08	144.92	1032.48	32.13	74.62
		peephole_lstm	20.75	80.65	483.89	22.00	64.92
		s2s_lstm	10.20	42.77	143.16	11.96	37.86
	h_7	naive	2.11	8.97	6.90	2.63	8.73
		arima	**2.10**	**8.93**	**6.86**	**2.62**	**8.70**
		ets	2.11	8.97	6.90	2.63	8.73
		stacked_lstm	46.60	224.80	2314.08	48.10	97.52
		peephole_lstm	8.62	30.80	133.51	11.55	32.91
		s2s_lstm	11.22	38.07	164.98	12.84	42.81

Cuadro 3: Comparación de métricas de precisión predictiva entre modelos post cambio de régimen.

Como se desprende de lo expuesto hasta aquí, es lógico pensar que las redes neuronales tienen menor precisión incluso que un forecast näive para todos los DGP y horizontes temporales (Cuadro 4). De hecho, solo el modelo ARIMA para $h = 7$ lograr batir consistentemente a una predicción tan simple como forecastear $y_t + h$ con y_t.

4.3 ARIMA

En este segundo experimento se utiliza la clase `ARIMA` del módulo `simulators` y se simulan dos DGP con 1000 observaciones cada uno inicializadas en 0. Las especificaciones de los DGP son las que se muestran a continuación:

- ARIMA(2,0,2) con tendencia determinística:

$$y_t = 0.001\,t + 0.6\,y_{t-1} + 0.3\,y_{t-2} + \epsilon_t + 0.2\,\epsilon_{t-1} + 0.1\,\epsilon_{t-2} \;/\; \epsilon_i \sim N(0,1)$$

DGP	p_value h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
vanilla_rw	1	0.35	0.36	1.00	1.00	1.00
	7	**0.01**	0.78	1.00	1.00	1.00
drift_trend_rw	1	0.40	0.39	1.00	1.00	1.00
	7	**0.03**	0.88	1.00	1.00	1.00

Cuadro 4: p-values para diferentes modelos comparados con el pronóstico näive, post cambio de régimen.

- ARIMA(3,1,2):

$$\Delta y_t = 0.5\,\Delta y_{t-1} + 0.2\,\Delta y_{t-2} + 0.1\,\Delta y_{t-3} + \epsilon_t + 0.3\,\epsilon_{t-1} + 0.1\,\epsilon_{t-2} \;/\; \epsilon_i \sim N(0,1)$$

Como puede verse en la figura 14, los primeros 750 timesteps constituyen el conjunto de entrenamiento y los últimos 250, el conjunto de test.

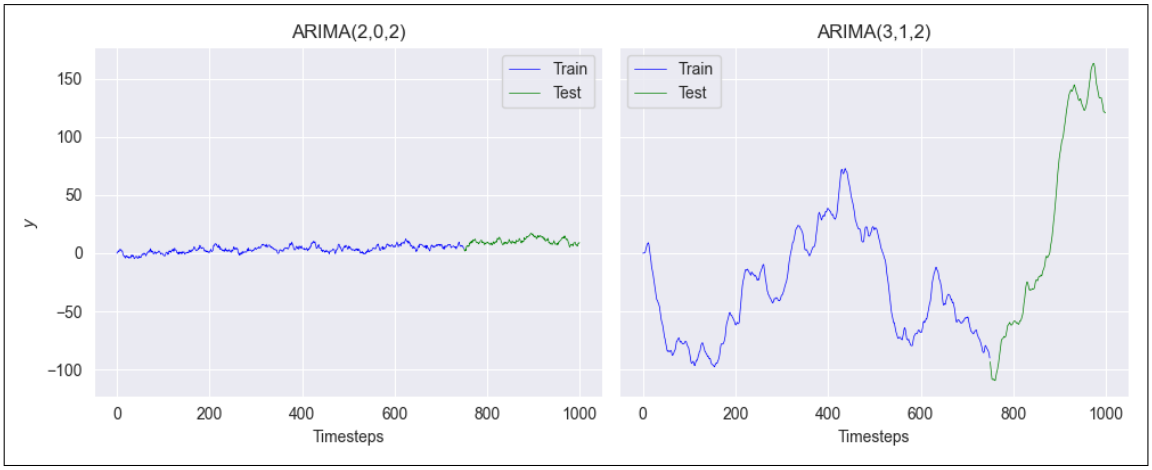

Figura 19: Simulaciones de ARIMA antes del cambio de régimen

El cambio de régimen tiene lugar en el timestep 750, alterando los parámetros de cada DGP de la siguiente manera:

- ARIMA(2,0,2) con tendencia determinística: se incrementan los parámetros tanto de la componente AR como MA en un 50 %, buscando simular un aumento de la dependencia de y_t sobre sus primeros rezagos (aka. mayor inercia):

$$y_t = 0.001\,t + 0.9\,y_{t-1} + 0.45\,y_{t-2} + \epsilon_t + 0.3\,\epsilon_{t-1} + 0.15\,\epsilon_{t-2} \;/\; \epsilon_i \sim N(0,1)$$

- ARIMA(3,1,2): se plantea el caso contrario, en donde los parámetros de cada componente se reducen en un 50 %:

$$\Delta y_t = 0.25\,\Delta y_{t-1} + 0.1\,\Delta y_{t-2} + 0.05\,\Delta y_{t-3} + \epsilon_t + 0.15\,\epsilon_{t-1} + 0.05\,\epsilon_{t-2} \;/\; \epsilon_i \sim N(0,1)$$

Las nuevas series temporales tras el cambio de régimen lucen de la siguiente manera:

Figura 20: Simulaciones de ARIMA luego del cambio de régimen

Al igual que en el experimento anterior, el primer paso consiste en entrenar cada modelo y producir el forecasting para ambos DGP y horizontes de tiempo antes del regime-switching. Así, y tal como se muestran en la figuras 21 y 22, la capacidad predictiva de las redes neuronales recurrentes sería nuevamente inferior a la de los modelos ARIMA y ETS.

Figura 21: Forecasting para el modelo ARIMA(2,0,2) antes del cambio de régimen

Si bien es cierto que para el caso ARIMA(2,0,2) con $h = 1$ la arquitectura stacked LSTM pareciera tener una performance aceptable durante los primeros timesteps, esta rápidamente se deteriora cuando y_t adopta valores (y forma patrones) distintos a los del conjunto de entrenamiento. Es notable como las arquitectura más complejas (LSTM con peepholes y Encoder-Decoder) vuelven a producir pronósticos *flat*, como si no incorporasen verdaderamente la información contenida en las realizaciones de y_t sino devolviesen más bien un promedio de las observaciones "vistas" durante entrenamiento.

Figura 22: Forecasting para el modelo ARIMA(3,1,2) antes del cambio de régimen

En el siguiente cuadro se presentan las métricas de precisión del forecasting, en donde vuelve a concluirse la superioridad de los modelos econométricos al menos para el caso base.

DGP	horizon	metric model	MAE	MAPE	MSE	RMSE	SMAPE
ARIMA(2,0,2)	h_1	naive	0.79	8.67	0.98	0.99	8.73
		arima	0.79	**8.57**	0.98	0.99	8.76
		ets	**0.78**	8.63	**0.96**	**0.98**	**8.65**
		stacked_lstm	1.70	28.71	4.45	2.11	17.40
		peephole_lstm	2.96	31.26	12.64	3.56	31.42
		s2s_lstm	3.79	37.06	21.20	4.60	41.99
	h_7	naive	**1.90**	19.93	**5.37**	**2.32**	19.74
		arima	2.63	28.27	10.73	3.28	30.17
		ets	**1.90**	**19.88**	5.38	**2.32**	**19.66**
		stacked_lstm	4.97	44.73	29.85	5.46	60.37
		peephole_lstm	6.04	55.70	42.06	6.49	79.38
		s2s_lstm	3.56	31.30	18.40	4.29	38.80

Cuadro 5: Métricas de precisión predictiva para ARIMA(2,0,2) antes del cambio de régimen

Adicionalmente, en el Cuadro 7 se presentan los p_values resultantes de comparar estadísticamente -usando un test de Diebold-Mariano- la precisión de cada modelo frente al pronóstico näive de utilizar la última observación realizada y_t para predecir el valor futuro y_{t+h}. Ningún modelo parece ser estadísticamente más preciso que la estrategia näive para el caso del ARIMA(2,0,2). Como puede verse en el Apéndice B, esto es así en parte porque la validación cruzada automática escogió para el modelo ARIMA configuraciones distintas al verdadero DGP en lugar de

DGP	horizon	metric model	MAE	MAPE	MSE	RMSE	SMAPE
ARIMA(3,1,2)	h_1	naive	1.68	5.73	4.58	2.14	5.75
		arima	1.17	4.37	2.10	1.45	4.31
		ets	**0.77**	**3.11**	**0.94**	**0.97**	**3.46**
		stacked_lstm	21.16	296.45	898.81	29.98	27.12
		peephole_lstm	35.00	240.76	2456.27	49.56	45.67
		s2s_lstm	39.02	228.81	3418.80	58.47	52.90
	h_7	naive	10.53	32.20	177.70	13.33	27.00
		arima	**6.98**	**26.38**	**75.12**	**8.67**	**21.44**
		ets	7.06	26.49	76.66	8.76	21.53
		stacked_lstm	48.40	211.22	4362.80	66.05	78.80
		peephole_lstm	28.52	282.56	1363.07	36.92	48.61
		s2s_lstm	28.91	281.16	1465.40	38.28	43.92

Cuadro 6: Métricas de precisión predictiva para ARIMA(3,1,2) antes del cambio de régimen

optar por un modelo ARIMA(2,0,2). Estas conclusiones se revierten para el caso ARIMA(3,1,2), donde existe evidencia suficiente para rechazar la hipótesis nula que la estimación näive es al menos tan precisa como la de los modelos ARIMA y ETS.

DGP	p_value h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
ARIMA(2,0,2)	1	0.51	0.15	1.00	1.00	1.00
	7	1.00	0.58	1.00	1.00	1.00
ARIMA(3,1,2)	1	**0.00**	**0.00**	1.00	1.00	1.00
	7	**0.00**	**0.00**	1.00	1.00	1.00

Cuadro 7: p-values para diferentes modelos comparados con el pronóstico näive, antes de regime-switching

Simulado el cambio de régimen en $t = 750$, se reitera el forecasting para ambos DGP y horizontes de tiempo, como puede apreciarse en las Figuras 23 y 24. Al igual que los casos anteriores, los modelos econométricos producen pronósticos puntuales mucho más cercanos a la realidad que los generados por las redes neuronales. Enfocándose en estas últimas, los pronósticos perecerían ser más precisos en el caso del modelo ARIMA(2,0,2) que para el modelo ARIMA(3,1,2), en donde las redes neuronales vuelven a producir un forecasting casi constante sin incorporar la información contenida en la observaciones realizadas. Pero, ¿a qué puede atribuirse esta diferencia entre escenarios? Tras el cambio de régimen en el modelo ARIMA(2,0,2), la serie adopta valores (y patrones) similares a los "vistos" por la red en fase de entrenamiento, sin generar valores excesivamente distintos a los del rango aprehendido durante el mismo. En cambio, para el caso del modelo ARIMA(3,1,2) la serie y_t adopta valores desconocidos por la red luego del cambio de régimen, lo cual posiblemente le impida reconocer patrones similares a aquellos con los cuales fue entrenada y, por lo tanto, solo pueda devolver un pronóstico promedio en base a las últimas observaciones.

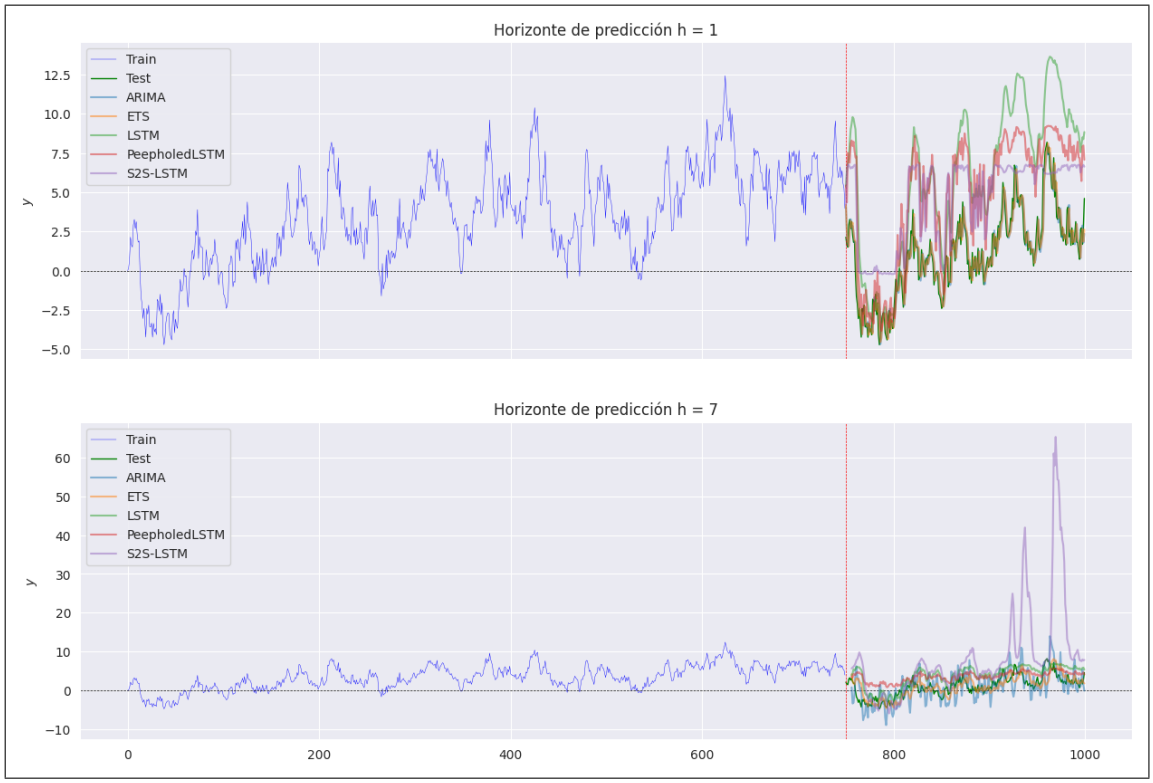

Figura 23: Forecasting para el modelo ARIMA(2,0,2) luego del cambio de régimen

Figura 24: Forecasting para el modelo ARIMA(3,1,2) luego del cambio de régimen

A similares conclusiones pueden arribarse a partir del siguiente cuadro donde se presentan nuevamente las métricas de precisión, ahora para el caso post regime switching: si bien son objetivamente menos precisas que los modelos ARIMA y ETS -incluso que la estimación näive-, las redes neuronales son relativamente mejores

pronosticando el modelo ARIMA(2,0,2) que el modelo ARIMA(3,1,2) al menos en términos de MSE.

DGP	horizon	metric model	MAE	MAPE	MSE	RMSE	SMAPE
ARIMA(2,0,2)	h_1	naive	0.80	136.75	1.04	1.02	56.88
		arima	**0.79**	137.85	**0.98**	**0.99**	56.60
		ets	**0.79**	**137.71**	1.00	1.00	**56.03**
		stacked_lstm	4.95	1183.81	31.29	5.59	117.88
		peephole_lstm	3.89	1003.21	19.25	4.39	109.19
		s2s_lstm	3.53	811.66	14.84	3.85	125.06
	h_7	naive	1.64	221.20	4.45	2.11	90.00
		arima	2.57	335.17	10.36	3.22	111.53
		ets	**1.63**	**213.35**	**4.40**	**2.10**	**88.38**
		stacked_lstm	2.94	763.68	11.95	3.46	101.27
		peephole_lstm	2.86	636.18	10.92	3.30	113.96
		s2s_lstm	7.84	1717.67	178.86	13.37	120.08
ARIMA(3,1,2)	h_1	naive	1.36	1.01	2.98	1.73	1.01
		arima	1.00	0.75	1.68	1.30	0.75
		ets	**0.79**	**0.58**	**1.07**	**1.03**	**0.58**
		stacked_lstm	44.97	27.16	2683.33	51.80	33.56
		peephole_lstm	70.75	45.12	5948.90	77.13	60.16
		s2s_lstm	85.44	55.61	8246.42	90.81	78.58
	h_7	naive	7.85	5.74	102.75	10.14	5.75
		arima	**5.71**	**4.16**	**57.54**	**7.59**	**4.16**
		ets	5.72	4.17	57.70	7.60	4.17
		stacked_lstm	95.28	62.16	9963.08	99.82	91.26
		peephole_lstm	45.68	37.53	2913.86	53.98	28.08
		s2s_lstm	61.71	38.42	4789.48	69.21	49.60

Cuadro 8: Métricas de precisión predictiva luego del cambio de régimen

Los resultados del test de Diebold-Mariano post cambio de régimen se muestran en el Cuadro 9: para un nivel de significancia del 5 %, los modelos ARIMA y ETS son más precisos que la estimación näive para $h = 1$ en caso del DGP ARIMA(2,0,2) y para ambos horizontes de tiempo para el DGP ARIMA(3,1,2). Por su parte, y coincidiendo con lo expuesto más arriba, no existe evidencia suficiente para rechazar la hipótesis nula que la estimación näive sea más precisa que aquella generada por todas las arquitecturas de redes neuronales.

DGP	p_value h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
ARIMA(2,0,2)	1	**0.02**	**0.03**	1.00	1.00	1.00
	7	1.00	0.15	1.00	1.00	1.00
ARIMA(3,1,2)	1	**0.00**	**0.00**	1.00	1.00	1.00
	7	**0.00**	**0.00**	1.00	1.00	1.00

Cuadro 9: p-values para diferentes modelos comparados con el pronóstico näive, luego del regime-switching

4.4 Markov-Switching Dynamic Regression Model

En este experimento se dejan de lado los cambios discrecionales de régimen para endogeneizarlos en el mismo DGP. Para ello se incorpora una nueva variable

S_t, la cual representa la serie de regímenes y adopta tantos valores discretos como regímenes pueda transitar el sistema. Los parámetros que rigen la dinámica de y_t son ahora función del estado en que se encuentra el sistema, S_t.

En cuanto a S_t, esta es una variable latente la cual sigue una cadena de Markov a tiempo discreto sobre los estados del sistema. Esto implica que S_t cumple con la propiedad de Markov, es decir, se cumple que la probabilidad que S_t transicione al estado j en tiempo t siendo que se encuentra en el estado k en $t-1$ solo depende de la probabilidad de transicionar del estado k al estado j y no de la trayectoria adoptada por S_t hasta ese momento. Matematicamente:

$$P(S_t = j \,|S_0 = s_0,\, S_1 = s_1, \ldots,\, S_{t-1} = k) = P(S_t = j \,|S_{t-1} = k)$$

Esta cadena de Markov a tiempo discreto es un proceso de Markov en el estado-espacio que puede representarse como un grafo dirigido y descripto mediante una matriz de transición P, la cual contiene en cada entrada la probabilidad de transicionar entre estados del sistema. Esta matriz determina la evolución del sistema en el tiempo en tanto la distribución de los estados en tiempo t, es decir, la probabilidad no condicionada de que el sistema se encuentre en cierto estado en tiempo t, no es más que la distribución de estados en tiempo $t-1$ multiplicada por la matriz de transición P.

El primer caso se trata de un proceso de tipo Markov-switching con un espacio-estado de dimensión igual a 2, en donde el régimen en que se encuentra S_t determina el valor de la constante y del parámetro autorregresivo de un modelo AR(1), mientras que no altera la distribución de los errores estocásticos la cual sigue siendo $N(0,1)$. Su grafo, incluidas las probabilidades de transición entre estados, es el siguiente:

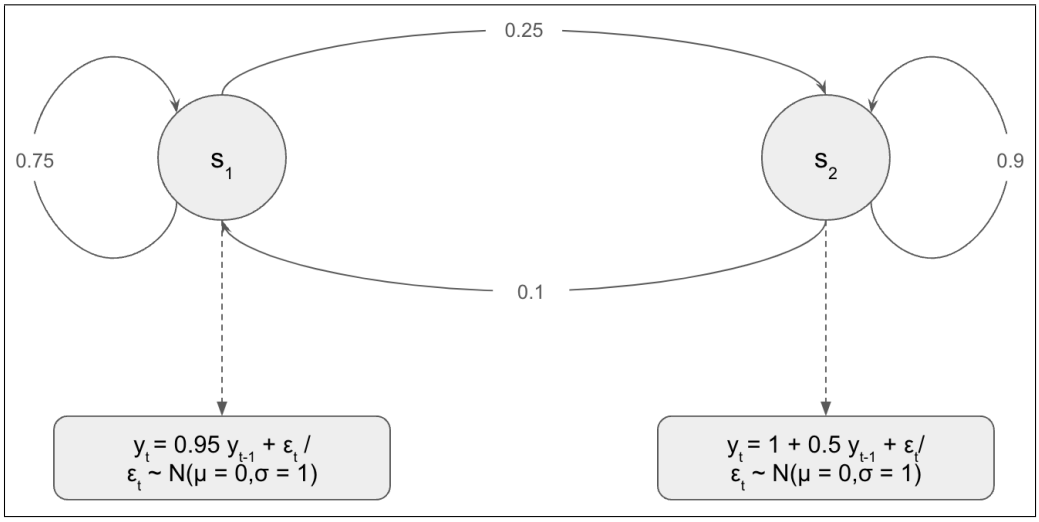

Figura 25: Representación gráfica del modelo

Su matriz de transición P correspondientes es:

$$\begin{bmatrix} 0.75 & 0.25 \\ 0.10 & 0.90 \end{bmatrix}$$

Para simular los datos, se eligió en este caso inicializar el estado/régimen *sampleándolo* directamente de la distribución de estado estacionario o largo plazo del

sistema. Puede probarse que esta existe en tanto la cadena de Markov es irreducible y aperiódica y se calculó como el autovector izquierdo correspondiente al máximo autovalor asociado. La simulación puede verse en la figura 26:

Figura 26: Modelo Markov-switching con 2 regímenes

Para el segundo caso, se simula otro modelo Markov-switching pero esta vez con un estado-espacio de dimensión 3, es decir, la variable S_t puede adoptar los valores en el conjunto $\{s_1, s_2, s_3\}$ y esto determinará los parámetros que regirán la dinámica de y_t en ese timestep. Su matriz de transición P correspondiente será:

$$\begin{bmatrix} 0.75 & 0.25 & 0 \\ 0.2 & 0.60 & 0.2 \\ 0 & 0.2 & 0.8 \end{bmatrix}$$

Como bien puede observarse en el siguiente grafo, el estado s_2 funciona de estado intermediario entre s_1 y s_3: S_t solo puede transicionar entre s_1 y s_3 pasando por s_2.

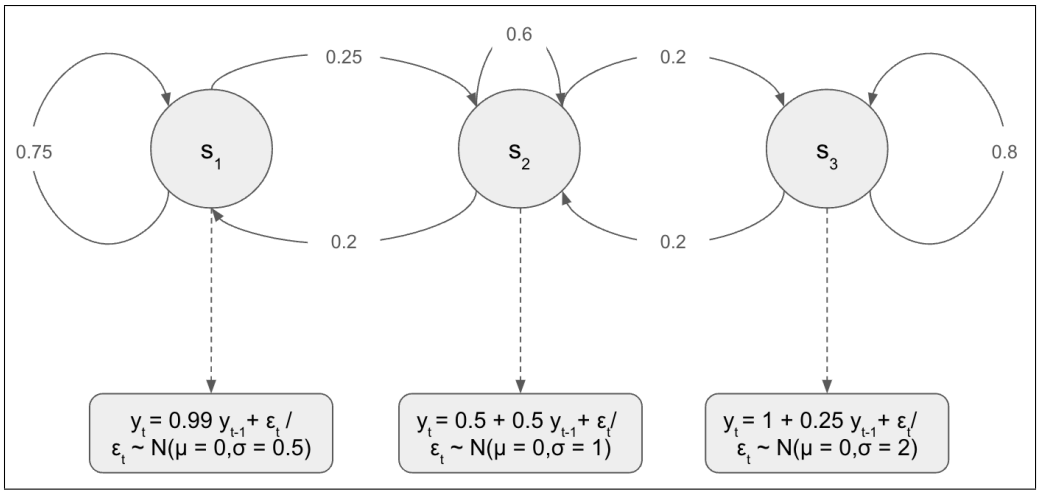

Figura 27: Representación gráfica del modelo

En la Figura 28 puede verse como quedaría la simulación. A diferencia del modelo con dos estados, en este caso la distribución de estado estacionario no es única y

por lo tanto se inicializa la secuencia de manera aleatoria sampleando el estado de una distribución uniforme.

Figura 28: Modelo Markov-switching con 3 regímenes

Como se hizo en los primeros dos experimentos, la siguiente tarea es generar los pronósticos para las 250 observaciones del conjunto de test. Una vez más, el mecanismo es iterativo: en la medida que se realiza y_t estas se incorporan como input al modelo para así poder generar los pronósticos subsecuentes. En las Figura 29 y 30 podrán verse las series de pronósticos para los modelos con dos y tres estados ocultos respectivamente:

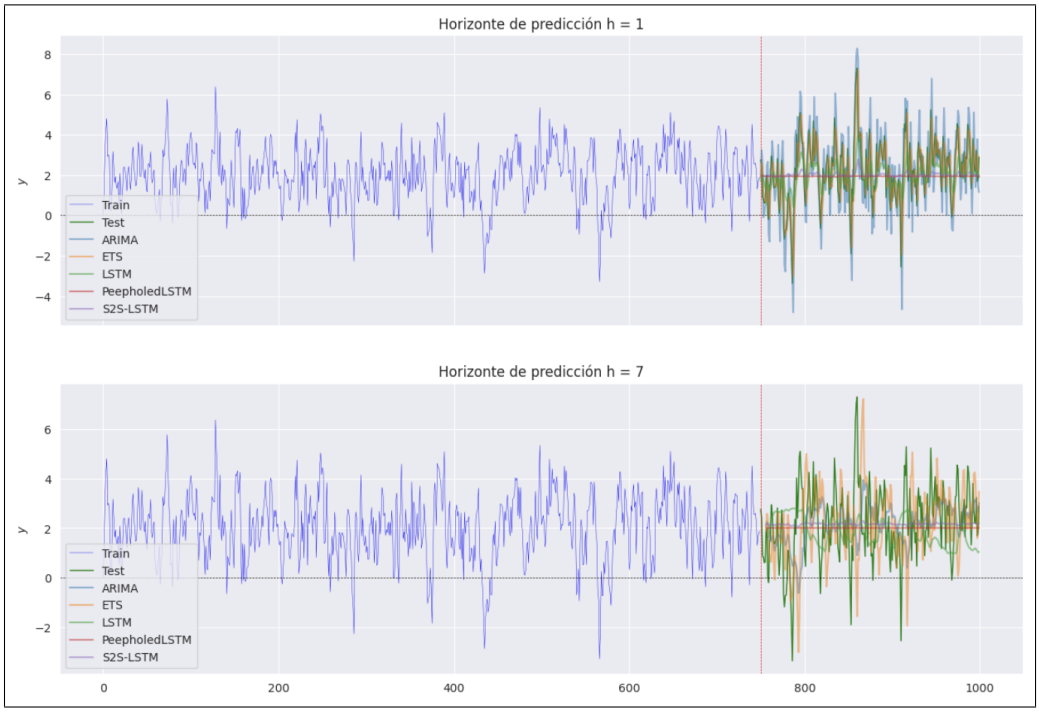

Figura 29: Forecasting para el modelo Markov-switching con 2 regímenes

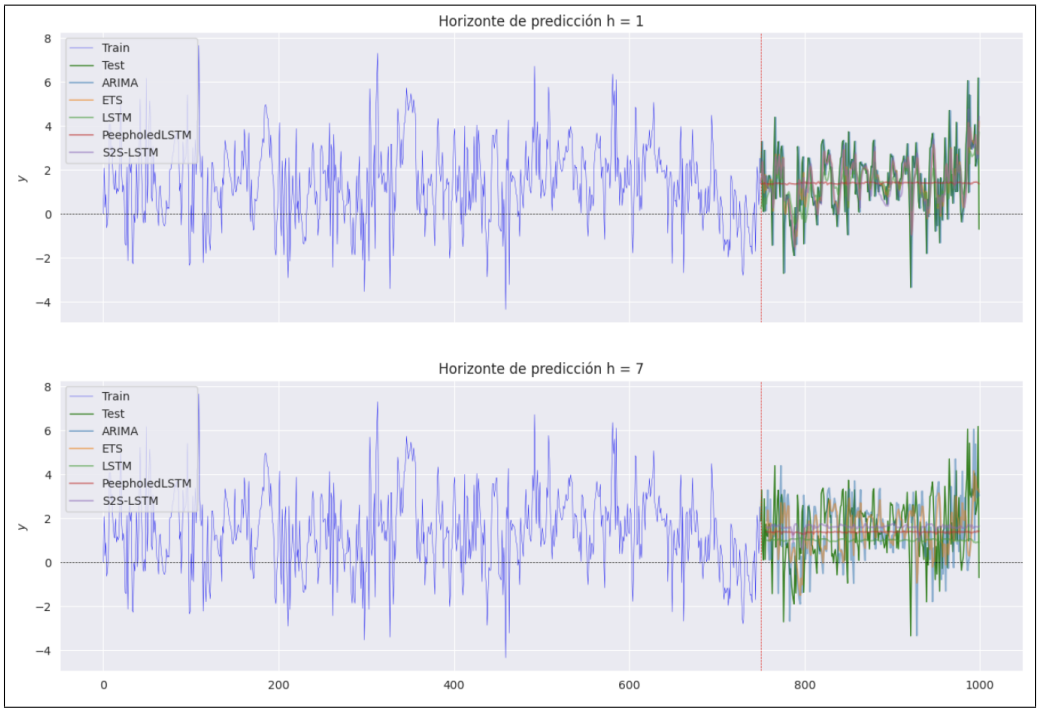

Figura 30: Forecasting para el modelo Markov-switching con 3 regímenes

A diferencia de los experimentos anteriores, y solo basándose en los gráficos presentados, los pronósticos de los modelos de redes neuronales parecieran ajustarse mucho mejor a los datos, particularmente la arquitectura stacked LSTM. En cuanto a las otras arquitecturas, tanto el Encoder-Decoder como LSTM con peepholes retornan predicciones que, si bien no son tan precisas como los otros modelos, se asemejan a un promedio de los datos efectivamente realizados.

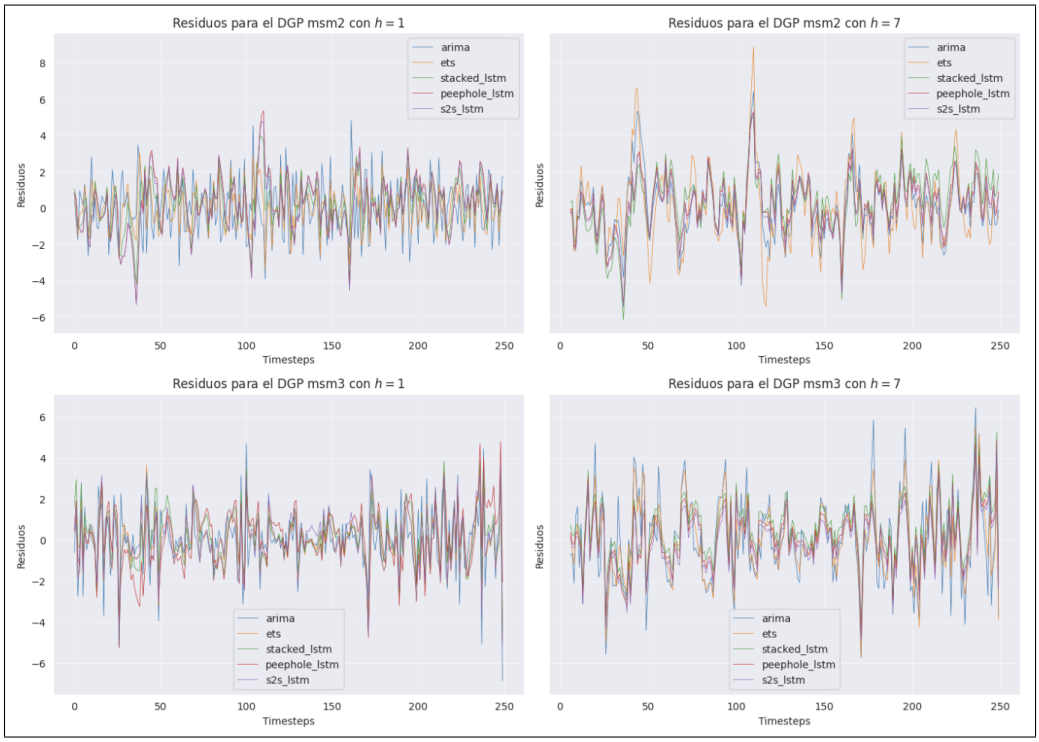

Figura 31: Errores de pronóstico para los modelos Markov-switching

Estas conclusiones se apoyan en la Figura 31, donde se exponen los errores de

predicción, así como en el Cuadro 10, donde se presentan las métricas de precisión del forecasting. En la primera se observa como, a diferencia de exprimentos anteriores, las series de errores no divergen entre modelos y en todos los casos parecieran centradas en 0.

DGP	horizon	metric / model	MAE	MAPE	MSE	RMSE	SMAPE
msm2	h_1	naive	**0.95**	**119.18**	**1.38**	**1.17**	55.16
		arima	1.21	146.32	2.29	1.51	65.95
		ets	0.96	120.57	1.39	1.18	55.29
		stacked_lstm	1.01	160.20	1.67	1.29	**53.88**
		peephole_lstm	1.17	152.71	2.32	1.52	57.83
		s2s_lstm	1.13	157.96	2.16	1.47	55.93
	h_7	naive	1.62	187.16	4.59	2.14	76.33
		arima	1.29	173.03	2.86	1.69	64.22
		ets	1.57	186.88	4.34	2.08	74.51
		stacked_lstm	1.32	**151.42**	2.88	1.70	64.69
		peephole_lstm	1.17	157.56	2.32	1.52	56.86
		s2s_lstm	**1.16**	168.33	**2.28**	**1.51**	**55.72**
msm3	h_1	naive	1.03	195.21	2.20	1.48	77.74
		arima	1.03	194.59	2.20	1.48	77.68
		ets	**0.99**	**183.69**	1.79	**1.34**	75.57
		stacked_lstm	1.02	244.63	**1.78**	**1.34**	78.72
		peephole_lstm	1.10	241.57	1.96	1.40	**75.41**
		s2s_lstm	1.02	277.78	1.85	1.36	77.09
	h_7	naive	1.62	412.77	4.19	2.05	103.07
		arima	1.61	411.69	4.18	2.04	103.07
		ets	1.36	327.25	3.00	1.73	93.24
		stacked_lstm	1.17	**182.20**	2.21	1.49	84.48
		peephole_lstm	1.10	232.62	**1.98**	1.41	75.37
		s2s_lstm	**1.09**	251.03	1.96	**1.40**	**73.70**

Cuadro 10: Métricas de precisión predictiva por modelo

Las métricas de precisión reconfirman lo expuesto: para el modelo con dos estados ocultos (`msm2`), las arquitecturas de redes neuronales tienen una performance comparable a la de los modelos econométricos y a la estrategia näive para el horizonte temporal $h = 1$, mientras que los superan para el horizonte temporal $h = 7$. Es destacable como la arquitectura stacked LSTM es más precisa que el modelo ARIMA para $h = 1$ si se toma como métrica a seguir el error cuadrático medio (MSE). Para el caso del modelo Markov-switching con tres estados ocultos (`msm3`), las redes neuronales parecieran ser aún más precisas: en términos de MSE, las tres arquitecturas superan a las estimaciones näive y ARIMA cuando $h = 1$, siendo muy similares a la performance del modelo ETS; mientras que para $h = 7$, las redes neuronales son más precisas que todas las demás estrategias. Sin embargo, cabe preguntarse si estas diferencias en precisión son estadísticamente significativas, pregunta que puede responderse con los resultados del test de Diebold-Mariano que se presentan en el siguiente cuadro:

DGP	p_value h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
msm2	1	1.00	0.61	1.00	1.00	0.95
	7	0.00	0.00	0.00	0.00	0.00
msm3	1	0.00	0.01	0.20	0.03	0.03
	7	0.00	0.00	0.00	0.00	0.00

Cuadro 11: p-values para diferentes modelos comparados con el pronóstico näive

Con un nivel de significancia del 5 %, y en base a los p-values mostrados, se rechaza la hipótesis nula que la estimación näive es más precisa que los modelos propuestos para el caso del modelo Markov-switching con 2 estados ocultos y $h = 7$ y para el DGP con 3 estados ocultos para todos los horizontes de tiempo (excepto para la arquitectura LSTM con peepholes para $h = 1$). El siguiente paso es repetir este mismo test pero tomando alternativamente el modelo ARIMA y ETS como baseline:

DGP	p_value h	ets	peephole_lstm	s2s_lstm	stacked_lstm
msm2	1	0.00	0.54	0.34	0.01
	7	1.00	0.00	0.00	0.52
msm3	1	0.01	0.21	0.03	0.03
	7	0.00	0.00	0.00	0.00

Cuadro 12: p-values para diferentes modelos comparados con el pronóstico ARIMA

DGP	p_value h	arima	peephole_lstm	s2s_lstm	stacked_lstm
msm2	1	1.00	1.00	1.00	0.96
	7	0.00	0.00	0.00	0.00
msm3	1	0.99	0.83	0.95	0.50
	7	1.00	0.00	0.00	0.00

Cuadro 13: p-values para diferentes modelos comparados con el pronóstico ETS

Comparando con las predicciones de los modelos ARIMA y ETS, existe evidencia suficiente para decir que estas son estadísticamente menos precisas que las producidas por las redes neuronales recurrentes cuando $h = 7$ (con excepción de la arquitectura stacked LSTM cuando el DGP tiene dos estados ocultos). En cambio, para el horizonte de tiempo $h = 1$, los resultados son mixtos: mientras el modelo ETS se muestra mucho más preciso que todas las redes neuronales cuando $h = 1$, las predicciones del modelo ARIMA son estadísticamente menos precisas que aquellas producidas por la arquitectura stacked LSTM.

Pero, ¿a qué puede atribuirse esta mejora relativa de los modelos de redes neuronales recurrentes? Una hipótesis plausible es que, a diferencia de los casos anteriores, el mecanismo de regime-switching está embebido en el propio DGP y las redes neuronales pueden aprenderlo en fase de entrenamiento. En el conjunto de

test no aparecen comportamientos espurios, movimientos demasiados innovadores que no hayan sido vistos por la red en el conjunto de entrenamiento y, a causa de ello, la red puede reconocer los patrones subyacentes en los datos y producir así predicciones más precisas basados en ellos.

5. Aplicación

En esta última sección se aplicará la misma metodología de análisis comparado pero esta vez sobre un problema concreto: la predicción del precio del Bitcoin. Para eso se seguirá a Figá-Talamanca et. al. (2021), quienes condujeron un estudio sobre cambios de regímenes y *commonalities* para una canasta de criptoactivos entre los que se encontraba el Bitcoin. Ellos demuestran que en caso de modelar cada serie de tiempo de manera individual, aquella alternativa que minimiza el RMSE sobre un conjunto de test asume que los precios del Bitcoin -especificamente, su primera diferencia- siguen un *Generalized White Noise* (GWN) con 3 estados ocultos. Matemáticamente:

$$\Delta y_t = \mu(s) + \sigma(s)\epsilon_t,\ t \geq 0$$

donde ambos parámetros $\mu(s)$ y $\sigma(s)$ dependen del estado s y $\epsilon_t \sim \mathrm{N}(0,1)$.

Estos autores sostienen que el origen de estos cambios de régimen se deben principalmente a cambios en la actitud de los inversores: el sector cripto pareciera transitar por períodos cíclicos de extremo entusiasmo por las tecnologías de la Web 3.0 seguidos por períodos de escepticismo, llevando a cambios estructurales y al surgimiento de burbujas. Para el caso del modelo con 3 estados ocultos, su interpretación es que existen dos regímenes de volatilidad (baja y alta) y, en el caso de alta volatilidad, puede estar asociado con retornos promedios positivos (euforia) o negativos (pánico).

Partiendo de la premisa que el precio del Bitcoin (su primera diferencia) puede modelarse como un GWN con 3 estados ocultos, la tarea en este apartado será comparar la precisión del forecasting para cada modelo utilizando datos reales y no simulados. Para eso se extraen 1000 observaciones directamente del sitio de Yahoo! Finance para el precio de cierre del par BTC-USD. La muestra abarca el período 2021-08-05 hasta 2024-04-30 y se toman los primeros 750 días como conjunto de entrenamiento y los últimos 250 como conjunto de test. La misma se acompaña con la serie de las primeras diferencias y la estimación del estado en que se encuentra el sistema para cada timestep. Para conseguirlo se instanció un objeto de la clase `MarkovAutoregression` del módulo `tsa` de la librería `Statsmodels` con las especificaciones dadas por Figá-Talamanca et. al.: tres estados ocultos, cada uno con su propia constante y varianza. Una vez estimado el modelo, se calcularon las probabilidades "suavizadas" de cada uno de los tres estados para cada timestep y se tomó la moda de dicha distribución como el estado que efectivamente se encontraba el sistema. Puede observarse claramente que el estado 2 se corresponde con períodos de alta volatilidad (serie de primeras diferencias con cambios abruptos), mientras que el estado 0 se corresponde con períodos de baja volatilidad; por su parte, el estado 1 parece ser marginal en esta muestra y pone en dudas la necesidad de modelar al problema con tres estados ocultos en lugar de dos.

Figura 32: Evolución del par BTC-USD para el período 2021-08-05:2024-04-30

A continuación se generan los forecasts para las 250 observaciones del conjunto de test utilizando los 5 modelos propuestos y para $h \in \{1, 7\}$.

Figura 33: Forecasting el precio de cierre del par BTC-USD

Al analizar las predicciones de cada modelo se repite un fenómeno que ya se evidenció en los experimentos controlados anteriores: las redes neuronales (con ex-

cepción de la arquitectura LSTM con peepholes) producen forecasts relativamente competitivos respecto a los modelos econométricos siempre que los valores realizados que se van incorporando como input se encuentren dentro del rango de valores que la red aprendió durante la fase de entrenamiento. Pero, ¿dónde se puede trazar este quiebre? La línea azul punteada destaca la fecha 2024-03-04, día en que la serie BTC-USD supera el máximo histórico (línea naranja punteada) alcanzado en el conjunto de entrenamiento: a partir de ese día (o en un entorno a ese día) las predicciones de las redes stacked LSTM y Encoder-Encoder divergen de las producidas por los modelos ARIMA y ETS y revierten a un valor constante tanto para $h = 1$ como $h = 7$. La serie de los residuos o errores de pronóstico muestran el mismo comportamiento.

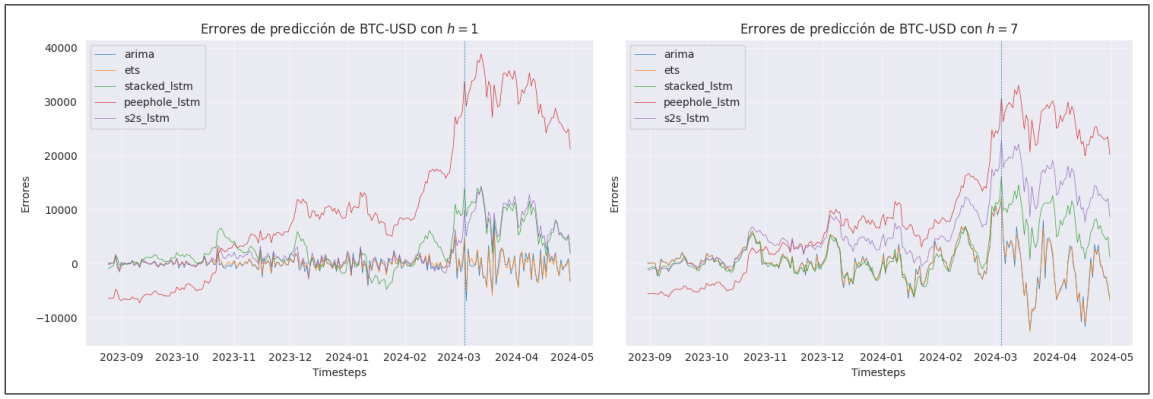

Figura 34: Errores de pronóstico para la serie BTC-USD

El Cuadro 14 muestra el set de métricas de precisión del forecasting para cada horizonte temporal y modelo; mientras que el cuadro 15, los resultados del test Diebold-Mariano.

horizon	metric / model	MAE	MAPE	MSE	RMSE	SMAPE
h_1	naive	896.39	**1.83**	**1966111.79**	**1402.18**	1.84
	arima	1026.36	2.09	2405580.94	1550.99	2.10
	ets	**897.18**	**1.83**	1974551.85	1405.19	**1.83**
	stacked_lstm	3374.67	36.48	23036376.99	4799.62	6.89
	peephole_lstm	13490.76	28.15	298674732.50	17282.21	30.89
	s2s_lstm	2678.27	36.33	19593188.98	4426.42	4.93
h_7	naive	2504.78	**5.13**	**12592215.58**	**3548.55**	5.27
	arima	2555.81	5.24	12988072.73	3603.90	5.29
	ets	**2501.78**	5.14	12669264.84	3559.39	**5.18**
	stacked_lstm	3347.97	35.57	24186007.66	4917.93	6.58
	peephole_lstm	11549.02	27.61	220955430.02	14864.57	25.52
	s2s_lstm	6708.04	31.82	80295807.95	8960.79	13.80

Cuadro 14: Métricas de precisión predictiva por modelo

Una conclusión que inmediatamente se desprende de estas métricas es lo dificultoso de predecir de manera precisa la serie BTC-USD, lo cual es la regla para la gran mayoría de las series financieras. De hecho, no existe evidencia empírica suficiente para descartar la hipótesis nula que la estrategia näive sea al menos tan precisa como las producidas por el resto de los modelos. En cuanto a las arquitecturas de redes neuronales, su precisión sobre el conjunto de test se encuentran muy por debajo del de los modelos econométricos.

h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
1	0.98	0.57	1.00	1.00	1.00
7	0.69	0.54	1.00	1.00	1.00

Cuadro 15: p-values para diferentes modelos comparados con el pronóstico näive

Pero, ¿que ocurre si se restringe el conjunto de test a las fechas anteriores al 2024-03-04, donde pareciera existir un quiebre en la serie de forecasts de las redes neuronales?

horizon	**metric** / **model**	**MAE**	**MAPE**	**MSE**	**RMSE**	**SMAPE**
h_1	näive	627.74	**1.57**	966558.21	983.14	1.58
	arima	708.77	1.78	1068710.28	1033.78	1.79
	ets	**624.71**	**1.57**	**937406.51**	**968.20**	**1.57**
	stacked_lstm	2022.82	27.27	8216773.93	2866.49	5.24
	peephole_lstm	8173.28	21.78	91864737.16	9584.61	22.00
	s2s_lstm	972.00	27.68	1933841.63	1390.63	2.45
h_7	näive	2019.42	4.84	9086692.34	3014.41	5.01
	arima	1974.54	4.79	8177410.61	2859.62	4.90
	ets	1946.22	**4.71**	**8075998.45**	**2841.83**	4.82
	stacked_lstm	**1937.87**	26.67	8510321.67	2917.25	**4.76**
	peephole_lstm	6933.21	21.35	70131507.45	8374.46	18.25
	s2s_lstm	4106.51	23.45	30419357.59	5515.37	10.24

Cuadro 16: Métricas de precisión predictiva por modelo antes de 2024-03-04

Para el subconjunto de datos de test con fecha anterior al 2024-03-04, en donde las observaciones adoptan valores (y patrones) similares al conjunto de entrenamiento, la performance de las redes neuronales mejora notoriamente al punto de superar en precisión, al menos la arquitectura stacked LSTM y para las métricas MAE y SMAPE, a los modelos tradicionales cuando $h = 7$. De hecho, si se relaja un poco el nivel de significancia del 5 %, existe cierta evidencia para rechazar la hipótesis nula que la estrategia näive es al menos tan precisa como la arquitectura stacked LSTM.

h	arima	ets	peephole_lstm	s2s_lstm	stacked_lstm
1	0.92	0.1	1.0	1.0	1.0
7	**0.014**	**0.00**	1.0	1.0	0.06

Cuadro 17: p-values para diferentes modelos comparados con el pronóstico näive antes de 2024-03-04

6. Conclusión

A lo largo del presente trabajo se ha intentado poner el foco en el uso de redes neuronales recurrentes -particularmente, aquellas arquitecturas basadas en unidades LSTM- en tareas de forecasting de series de tiempo. Para ello se hizo una presentación teórica sobre su funcionamiento y se esbozaron argumentos que justificarían su superioridad sobre modelos econométricos tradicionales, destacándose entre ellos su habilidad para capturar relaciones no lineales entre los datos y su capacidad para perdurar información útil en el tiempo gracias a sus mecanismos de memoria, a pesar del elevado costo computacional que conlleva su entrenamiento.

Sin embargo, ni los experimentos controlados ni la aplicación práctica para el caso del Bitcoin demostraron la superioridad de las redes neuronales recurrentes por sobre los modelos ARIMA y ETS, e incluso, por sobre una estrategia näive como predecir el futuro con la última realización de y_t. En todos los casos parece desprenderse la misma conclusión: las redes neuronales son muy dependientes de los datos con los que se han entrenado y producirán pronósticos relativamente precisos en tanto el DGP genere valores similares a aquellos que la red ha "visto" durante su fase de entrenamiento. Dado que las redes neuronales funcionan internamente reconociendo patrones y regularidades en los datos, en la medida que se produzcan variaciones abruptas que quiebren dinámicas previamente aprehendidas -por ejemplo, ante un cambio de régimen- la capacidad predictiva de la red caerá abruptamente. Este parecería ser el motivo principal de la pobre performance de las redes neuronales en los casos de las simulaciones RW y ARIMA, y para período posterior al 2024-03-04 para el caso del Bitcoin. Por el contrario, siempre que el DGP se mueva en un rango de valores conocidos, siguiendo patrones estables -como el caso de los modelos Markov-switching o la serie de precios del Bitcoin antes de sobrepasar su máximo histórico- las arquitecturas de redes neuronales llegan a ser incluso más precisas que los modelos econométricos. Entre ellas, se destaca el modelo stacked LSTM que, a pesar de ser la arquitectura más simple de las tres, fué la que generó predicciones más precisas.

Cabe mencionar que estas conclusiones son solo provisorias, basadas en una única muestra simulada por DGP. Futuros trabajos deberían continuar esta línea de investigación generalizando las conclusiones alcanzadas en cada DGP, por ejemplo, generando múltiples simulaciones de Monte Carlo sobre cada DGP y probando diferentes tipos de regime-switching. Otro camino a profundizar es expandir el espacio de búsqueda de hiperparámetros para el caso de las redes neuronales (en lugar de fijar el stride en 25 o sólo elegir 3 tamaños de batch para cross-validar) e intentar así mejorar su capacidad predictiva; o probar con otro tipo de arquitectura como son los Transformers y sus mecanismos de atención, los cuales han alcanzado gran popularidad gracias a los Large Language Models. Un posible tercer curso de acción a abordar es descartar la convención que las redes neuronales pueden trabajar con series de tiempos no estacionarias sin necesidad de transformar los datos y proceder a trabajar con la serie de las primeras diferencias, o proponer otra estrategia de procesamiento de los datos más inteligente que una simple estandarización (por ejemplo, eliminando la tendencia en cada batch). No obstante, en todos los casos existe el impedimento de la capacidad de cómputo: las redes neuronales recurrentes son costosas de entrenar y mucho más de cross-validar, y el diferencial de precisión respecto a los modelos econométricos tradiciones no pareciera -al menos en los

escenarios planteados- justificar su uso.

Bibliografía

[1] Susan Athey. The impact of machine learning on economics. In *The Economics of Artificial Intelligence: An Agenda*, pages 507–547. University of Chicago Press, 2018.

[2] Filip Björnsjö. Can deep learning beat traditional econometrics in forecasting of realized volatility?, 2020.

[3] Francois Chollet. *Deep Learning with Python*. Simon and Schuster, 2021.

[4] Thomas R. Cook. Neural networks. In Peter Fuleky, editor, *Macroeconomic forecasting in the era of big data: Theory and practice*, chapter 6, pages 161–189. Springer Nature, 2019.

[5] Francis X Diebold and Robert S Mariano. Comparing predictive accuracy. *Journal of Business & economic statistics*, 20(1):134–144, 2002.

[6] Graham Elliot and Allan Timmermann. *Economic Forecasting*. Princeton University Press, 2016a.

[7] Graham Elliott and Allan Timmermann. Forecasting in economics and finance. *Annual Review of Economics*, 8:81–110, 2016b.

[8] Gianna Figà-Talamanca, Sergio Focardi, and Marco Patacca. Regime switches and commonalities of the cryptocurrencies asset class. *The North American Journal of Economics and Finance*, 57:101425, 2021.

[9] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE, 2000.

[10] Philippe Goulet Coulombe, Maxime Leroux, Dalibor Stevanovic, and Stéphane Surprenant. How is machine learning useful for macroeconomic forecasting? *Journal of Applied Econometrics*, 37(5):920–964, 2022.

[11] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[12] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.

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

[14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[15] Neeraj Krishna. Backpropagation in rnn explained, April 9 2022. Medium.

[16] Frank Lehrbass. Deep learning diagnostics–how to avoid being fooled by tensorflow, pytorch, or mxnet with the help of modern econometrics. *Schriftenreihe des Instituts für Empirie & Statistik der FOM Hochschule*, 24, 2021.

[17] T Mitchel. Machine learning, mcgraw-hill education (ise editions). 1997.

[18] Saeed Nosratabadi, Amirhosein Mosavi, Puhong Duan, Pedram Ghamisi, Ferdinand Filip, Shahab S Band, Uwe Reuter, Joao Gama, and Amir H Gandomi. Data science in economics: a comprehensive review of advanced machine learning and deep learning methods. *Mathematics*, 8(10):1799, 2020.

[19] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[21] Lukas Ryll and Sebastian Seidens. Evaluating the performance of machine learning algorithms in financial market forecasting: A comprehensive survey. *arXiv preprint arXiv:1906.07786*, 2019.

[22] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[23] Hal R Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, 2014.

[24] Eric Zheng, Yong Tan, Paulo Goes, Ramnath Chellappa, DJ Wu, Michael Shaw, Olivia Sheng, and Alok Gupta. When econometrics meets machine learning. *Data and Information Management*, 1(2):75–83, 2017.

Apéndice A

Es este apéndice se muestran las derivadas de la función de pérdida L respecto a los parámetros de la red, los cuales son usados para entrenar la misma mediante el algoritmo BPTT. Las mismas son:

- Derivada de la función de pérdida respecto a W_i:

$$\frac{\partial L}{\partial W_i} = \sum_{k=0}^{t} \frac{\partial L}{\partial y_{t-k}} \frac{\partial y_{t-k}}{\partial h_{t-k}} \left(\prod_{j=t-k+1}^{t} \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-k-1}}{\partial W_i}$$

- Derivada de la función de pérdida respecto a V_i:

$$\frac{\partial L}{\partial V_i} = \sum_{k=0}^{t} \frac{\partial L}{\partial y_{t-k}} \frac{\partial y_{t-k}}{\partial h_{t-k}} \left(\prod_{j=t-k+1}^{t} \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-k-1}}{\partial V_i}$$

- Derivada de la función de pérdida respecto a W_o:

$$\frac{\partial L}{\partial W_o} = \sum_{k=0}^{t} \frac{\partial L}{\partial y_{t-k}} \frac{\partial y_{t-k}}{\partial h_{t-k}} \left(\prod_{j=t-k+1}^{t} \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-k-1}}{\partial W_o}$$

- Derivada de la función de pérdida respecto al sesgo b_i:

$$\frac{\partial L}{\partial b_i} = \sum_{k=0}^{t} \frac{\partial L}{\partial y_{t-k}} \frac{\partial y_{t-k}}{\partial h_{t-k}} \frac{\partial h_{t-k}}{\partial b_i}$$

- Derivada de la función de pérdida respecto al sesgo b_o:

$$\frac{\partial L}{\partial b_o} = \sum_{k=0}^{t} \frac{\partial L}{\partial y_{t-k}} \frac{\partial y_{t-k}}{\partial b_o}$$

Conviene detenerse en el factor $\prod_{j=t-k+1}^{t} \frac{\partial h_{t-j+1}}{\partial h_{t-j}}$ presente en las primeras tres derivadas: si se define h_t como:

$$h_t = \tanh\left(W_i \; h_{t-1} + V_i \; x_t + b_i\right)$$

entonces se cumple que su derivada con respecto a h_{t-1} es:

$$\frac{\partial h_t}{\partial h_{t-1}} = \cosh^{-2}\left(W_i \; h_{t-1} + V_i \; x_t + b_i\right) W_i$$

Dado que el coseno hiperbólico tiene por codominio $[1, +\infty)$, el primer factor de la derivada está acotado entre $(0, 1]$. Por lo tanto, en la medida que la secuencia sea más larga (a mayor t), el factor $\prod_{j=t-i+1}^{t} \frac{\partial h_{t-j+1}}{\partial h_{t-j}}$ tiende a 0, empujando a 0 todo el sumando. Este fenómeno, llamado desvanecimiento del gradiente o *vanishing gradient problem*, lleva a que la red no pueda propagar el gradiente a los primeros *timesteps* de la secuencia y, como consecuencia, tenga problemas para aprender dependencias de larga data en los datos.

Apéndice B

En este apéndice se presentan la configuración de hiperparámetros óptima por DGP, horizonte temporal y modelo.

ARIMA

DGP	h	stride	window_length	AR_order	I_order	MA_order
Vanilla RW	1	25	25	0	1	2
Vanilla RW	7	25	100	3	1	3
RW con drift y trend	1	25	25	3	1	1
RW con drift y trend	7	25	100	3	1	3
ARIMA(2,0,2)	1	25	50	0	1	1
ARIMA(2,0,2)	7	25	25	3	1	2
ARIMA(3,1,2)	1	25	100	0	1	1
ARIMA(3,1,2)	7	25	100	2	1	2
MSM_2	1	25	25	2	2	2
MSM_2	7	25	25	0	1	3
MSM_3	1	25	25	0	2	1
MSM_3	7	25	50	2	1	2
BTC-USD	1	25	100	2	1	2
BTC-USD	7	25	100	3	1	2

ETS

DGP	h	stride	window_length	Error	Trend	Damped Trend
Vanilla RW	1	25	100	additive	None	False
Vanilla RW	7	25	100	additive	additive	True
RW con drift y trend	1	25	100	additive	additive	False
RW con drift y trend	7	25	100	additive	additive	False
ARIMA(2,0,2)	1	25	25	additive	additive	True
ARIMA(2,0,2)	7	25	25	additive	None	False
ARIMA(3,1,2)	1	25	25	additive	additive	True
ARIMA(3,1,2)	7	25	50	additive	additive	True
MSM_2	1	25	100	additive	additive	False
MSM_2	7	25	25	additive	None	False
MSM_3	1	25	25	additive	None	False
MSM_3	7	25	100	additive	additive	True
BTC-USD	1	25	100	multiplicative	multiplicative	False
BTC-USD	7	25	100	multiplicative	None	False

Stacked LSTM

DGP	h	stride	window_length	batch_size	activation	recurrent_layer_1	recurrent_layer_2	dense_layer_1	learning_rate	optimizer	drop_out_rate	initializer	l2_coeff	second_hidden_layer
Vanilla RW	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
Vanilla RW	7	25	5	64	sigmoid	16	32	64	0.0100	rmsprop	0.0	glorot_uniform	0.0	False
RW con drift y trend	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
RW con drift y trend	7	25	2	32	elu	64	16	64	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
ARIMA(2,0,2)	1	25	50	128	elu	64	16	16	0.0010	rmsprop	0.4	glorot_uniform	0.0	False
ARIMA(2,0,2)	7	25	2	32	elu	64	16	64	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
ARIMA(3,1,2)	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
ARIMA(3,1,2)	7	25	5	64	sigmoid	16	32	64	0.0100	rmsprop	0.0	glorot_uniform	0.0	False
MSM_2	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
MSM_2	7	25	50	64	elu	128	16	32	0.0010	adam	0.2	he_normal	0.0	True
MSM_3	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
MSM_3	7	25	5	64	sigmoid	16	32	64	0.0100	rmsprop	0.0	glorot_uniform	0.0	False
BTC	1	25	50	64	sigmoid	128	64	128	0.0010	adam	0.1	glorot_uniform	0.0	True
BTC	7	25	2	32	elu	64	16	64	0.0100	rmsprop	0.4	glorot_uniform	0.0	True

LSTM con Peepholes

DGP	h	stride	window_length	batch_size	activation	recurrent_layer_1	recurrent_layer_2	dense_layer_1	learning_rate	optimizer	drop_out_rate	initializer	l2_coeff	second_hidden_layer
Vanilla RW	1	25	2	64	relu	16	128	128	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
Vanilla RW	7	25	2	32	elu	64	16	64	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
RW con drift y trend	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
RW con drift y trend	7	25	50	64	elu	128	16	32	0.0010	adam	0.2	he_normal	0.0	True
ARIMA(2,0,2)	1	25	2	64	relu	16	128	128	0.0010	rmsprop	0.4	glorot_uniform	0.0	True
ARIMA(2,0,2)	7	25	2	32	elu	64	16	64	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
ARIMA(3,1,2)	1	25	2	64	relu	16	128	128	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
ARIMA(3,1,2)	7	25	2	32	elu	64	16	64	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
MSM_2	1	25	50	64	tanh	128	64	128	0.0010	adam	0.1	glorot_uniform	0.0	True
MSM_2	7	25	50	128	relu	64	16	128	0.0001	rmsprop	0.3	random_normal	0.0	False
MSM_3	1	25	50	64	tanh	128	64	128	0.0010	adam	0.1	glorot_uniform	0.0	True
MSM_3	7	25	50	128	relu	64	16	128	0.0001	rmsprop	0.3	random_normal	0.0	False
BTC	1	25	50	128	elu	64	16	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	False
BTC	7	25	5	64	sigmoid	16	32	64	0.0100	rmsprop	0.0	glorot_uniform	0.0	False

Encoder-Decoder LSTM

DGP	h	stride	window_length	batch_size	activation	recurrent_layer_1	recurrent_layer_2	learning_rate	optimizer	drop_out_rate	initializer	l2_coeff	second_hidden_layer
Vanilla RW	1	25	2	32	tanh	64	32	0.0010	rmsprop	0.0	he_normal	0.0	False
Vanilla RW	7	25	5	128	elu	16	8	0.0010	adam	0.4	he_normal	0.0	True
RW con drift, y trend	1	25	2	32	tanh	64	32	0.0010	rmsprop	0.0	he_normal	0.0	False
RW con drift, y trend	7	25	50	128	tanh	32	8	0.0010	rmsprop	0.2	random_normal	0.0	False
ARIMA(2,0,2)	1	25	2	32	tanh	64	32	0.0010	rmsprop	0.0	he_normal	0.0	False
ARIMA(2,0,2)	7	25	5	64	tanh	128	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
ARIMA(3,1,2)	1	25	2	32	tanh	64	32	0.0010	rmsprop	0.0	he_normal	0.0	False
ARIMA(3,1,2)	7	25	50	128	tanh	32	8	0.0010	rmsprop	0.2	random_normal	0.0	False
MSM_2	1	25	5	64	sigmoid	32	4	0.0100	rmsprop	0.0	glorot_uniform	0.0	False
MSM_2	7	25	2	64	sigmoid	16	2	0.0100	sgd	0.0	random_normal	0.0	False
MSM_3	1	25	5	64	relu	32	4	0.0100	rmsprop	0.0	glorot_uniform	0.0	False
MSM_3	7	25	5	64	relu	128	16	0.0100	rmsprop	0.4	glorot_uniform	0.4	True
BTC	1	25	50	64	relu	128	16	0.0100	rmsprop	0.4	glorot_uniform	0.0	True
BTC	7	25	50	128	tanh	32	8	0.0010	rmsprop	0.2	random_normal	0.0	False