Análisis comparativo de metodologías de pronóstico para múltiples series de tiempo de conteos.

Daniel Betancur Rodríguez

Universidad Nacional de Colombia
Facultad de Ciencias, Escuela de Estadística
Medellín, Colombia
2024

Análisis comparativo de metodologías de pronóstico para múltiples series de tiempo de conteos.

Daniel Betancur Rodríguez

Tesis de grado presentada como requisito parcial para optar al título de:
Magíster en Estadística

Directora:
Nelfi Gertrudis Gonzáles Alvarez
Doctora en Ciencias Estadísticas

Co-director:
Daniel Cabarcas Jaramillo
Doctor en Ciencias Matemáticas

Líneas de investigación:
Analítica
Procesos estocásticos

Universidad Nacional de Colombia
Facultad de Ciencias, Escuela de Estadística
Medellín, Colombia
2024

Lo importante en la ciencia no es tanto obtener nuevos datos, sino descubrir nuevas formas de pensar sobre ellos.

William Lawrence Bragg

A Kyra.

Resumen

El pronóstico de series de tiempo de conteos, con soporte en los enteros no negativos, es un caso particular de interés para la asignación óptima de capacidades e inventarios acorde a la demanda esperada, entre otras aplicaciones. Para abordar el pronóstico de series de tiempo de conteos se han propuesto modelos estadísticos como los modelos autorregresivos para series de conteo o los modelos dinámicos generalizados. Por otro lado, se han aplicado metodologías basadas en algoritmos de machine learning apalancándose en la creciente potencia computacional, como las redes neuronales recurrentes, las arquitecturas de redes LSTM y las arquitecturas basadas en algoritmos de atención, llamadas Transformers. El presente trabajo explora el problema del pronóstico paralelo de múltiples series de conteo, aplicando metodologías propias de la estadística y el machine learning en diversos escenarios de simulación en los cuales se compara la calidad de pronóstico, el tiempo computacional demandado y el esfuerzo para adaptar cada metodología a casos reales.

Palabras clave: Modelos lineales generalizados, predicción, datos de conteos, regresión Poisson, modelos espacio-estado, series de tiempo, redes neuronales, redes neuronales recurrentes, transformers.

Abstract

Comparative analysis of forecasting methodologies for multiple time series of counts

Forecasting time series of counts, with support on non-negative integers, is a particular case of interest for optimal job assigment and inventory allocation according to expected demand, among other applications. To address the problem of forecasting time series of counts, statiscal models such as autorregresive models for count data or dynamic generalized models have been proposed. On the other side, methodologies based on machine learning algorithms have been applied, leveraging on the increasing computational power, such as recurrent neuronal netwroks, LSTM networks architecures and architectures based in attention algorithms called Transformers. This study explores the problem of parallel forecasting multiple time series of counts, applying statistical and machine learning methodologies to various simulation scenarios in which the forecasting performance, demanded computational time, and the effort to adapt each methodology to real cases are compared.

Keywords: Generalized lineal models, prediction, count data, Poisson regression, state-space models, time series, neuronal networks, recurrent neuronal networks, transformers

Contenido

Notación

Esta sección presenta la notación utilizada en este trabajo tanto en el desarrollo desde una perspectiva estadística clásica, como la notación matemática en los desarrollos desde la perspectiva del deep learning.

Notación estadística

Notación	Significado
Y	Variable de interés a ser pronosticada
$Y^{(t)}$	Variable de interés en el momento t
$y^{(t)}$	Valor observado de la variable de interés en el momento t
X_j	j-ésima variable regresora
x_{ij}	i-ésima observación de la j-ésima variable regresora
$\boldsymbol{X}$	Matriz de diseño
n	Número de observaciones en la matriz de diseño
$\boldsymbol{\theta}$	Vector de parámetros
$L(\boldsymbol{\theta})$	Función de verosimilitud

Notación deep learning

Notación	Significado
Superíndice $[l]$	Denota la l-ésima capa
Superíndice (t)	Denota él t-ésimo elemento en una secuencia
Subíndice i	Denota la i-ésima observación o ejemplo
Subíndice j	Denota la j-ésima variable
m	Número de ejemplos en las entradas del modelo. Puede ser el tamaño total del conjunto de datos o el tamaño del lote, en cuyo caso $m = n$
n	Número de ejemplos en el conjunto de datos. Para el caso de conjuntos de datos
n_l	Número de nodos ocultos de la l-ésima capa

Notación	Significado
n_y	Número de variables en la respuesta para el caso multivariado
p	Número de variables en el conjunto de datos
L	Número de capas en la red neuronal
T_x	Longitud de las secuencias de entrada para datos de secuenciales
T_y	Longitud de las secuencias de salida para datos de secuenciales
L	Número de capas en la red neuronal
$\boldsymbol{A} \in \mathbb{R}^{m \times p}$	Matriz de datos de entrada
$\boldsymbol{x_j} \in \mathbb{R}^{m \times 1}$	Vector de ejemplos muestrales de la j-ésima variable predictora
$x_{ij} \in \mathbb{R}$	i-ésimo ejemplo muestral u observación de la j-ésima variable predictora
$\boldsymbol{y} \in \mathbb{R}^{m \times 1}$	Variable respuesta o etiqueta del modelo para casos de respuesta univariada
$\boldsymbol{Y} \in \mathbb{R}^{m \times n_y}$	Matriz de respuesta o etiquetas del modelo para el caso de respuesta multivariada
$y_i \in \mathbb{R}$	i-ésimo ejemplo de la variable respuesta en el caso univariado
$\boldsymbol{y_i} \in \mathbb{R}^{1 \times n_y}$	vector del i-ésimo ejemplo de la variable respuesta en el caso multivariado
$\boldsymbol{W^{[l]}} \in \mathbb{R}^{n_{l-1} \times n_l}$	Matriz de pesos de las activaciones de la $(l-1)$-ésima capa sobre los nodos de la l-ésima capa
$\boldsymbol{w_j^{[l]}} \in \mathbb{R}^{n_{l-1} \times 1}$	Vector de pesos de las activaciones de la $(l-1)$-ésima capa sobre él j-ésimo nodo de la l-ésima capa
$\boldsymbol{b^{[l]}} \in \mathbb{R}^{1 \times n_l}$	Vector de sesgos de la l-ésima capa
$b_j^{[l]} \in \mathbb{R}$	Sesgo de la j-ésima neurona de la l-ésima capa
$\hat{\boldsymbol{Y}} \in \mathbb{R}^{m \times n_y}$	Matriz de salida o valores estimados por el modelo en el caso multivariado
$\hat{\boldsymbol{y}} \in \mathbb{R}^{m \times 1}$	Vector de salida o valores eestimadps por el modelo en el caso univariado
$\boldsymbol{A^{[l]}} \in \mathbb{R}^{m \times n_l}$	Matriz con las activaciones de la l-ésima capa. Note que cuando $l = 0$ es igual a la matriz de datos de entrada $\boldsymbol{A}$ y cuando $l = L$ es igual a la matriz de salida o valores estimados $\boldsymbol{Y}$
$\boldsymbol{a_j^{[l]}} \in \mathbb{R}^{m \times 1}$	Vector con las activaciones de la j-ésima neurona o nodo de la l-ésima capa
$J(\hat{\boldsymbol{Y}}, \boldsymbol{Y})$ ó $J(\hat{\boldsymbol{y}}, \boldsymbol{y})$	Función de costo

Abreviaturas

Abreviatura	Término
NN	Neural Network (red neuronal)
RNN	Recurrent Neural Network (red neuronal recurrente)

Abreviatura	**Término**
LSTM	Long Short-Term Memory (Memoria larga de corto plazo)

1. Introducción

Pronosticar ha sido una tarea de gran interés y sumamente ligada a la actividad humana desde hace mucho tiempo, tratándose de una actividad necesaria para muchas situaciones, por ejemplo en la proyección de la demanda eléctrica futura para determinar si es necesaria la creación de plantas eléctricas, o en la distribución de capacidades de acuerdo al volumen de llegada de clientes a un punto de servicio (Hyndman & Athanasopoulos, 2021). Cualesquiera que sean las circunstancias u horizonte de tiempo, pronosticar resulta importante para la planeación efectiva y eficiente.

Un caso particular de interés en el pronóstico para la asignación de capacidades, es el pronóstico de las llegadas de clientes a un sistema (donde por cliente puede entenderse una llamada, solicitud de gestión, etc.) en el cual se realizan múltiples tipos de atenciones, de modo que las llegadas pueden quedar clasificadas de acuerdo al proceso de atención, lo que implica tiempos de gestión diferentes. Bajo tal escenario, el problema del pronóstico involucra múltiples series de tiempo sobre diferentes variables de conteos, donde la sucesión de los conteos observados para cada tipología de llegada, resulta en una serie de tiempo particular cuyo pronóstico brinda información importante para la programación de la fuerza de trabajo, lo cual impacta en el balance entre la eficiencia operativa y la calidad de atención (Excoffier et al., 2016).

De acuerdo a Christou y Fokianos (2015) una serie de tiempo es una secuencia de valores, comúnmente medidos como una sucesión de valores a intervalos regulares de tiempo, y para el caso de series de tiempo sobre variables de conteo, dicha sucesión toma valores en el rango de los números enteros positivos, lo que puede causar que sea inapropiada la aplicación de los modelos tradicionales en el pronóstico de series de tiempo que trabajan sobre variables reales y que asumen distribuciones normales. Si bien es cierto que en los casos donde los conteos se muevan en valores grandes (muy superiores a cero) este incumplimiento de supuestos puede no afectar fuertemente el desempeño de los modelos clásicos de series de tiempo (Hyndman & Athanasopoulos, 2021), en esta tesis se va a trabajar series de conteos que pueden tomar valores pequeños o incluso con la presencia de ceros, lo que refuerza la idea de que deben explorarse otras alternativas.

Sobre series de conteos se han formulado, entre otros, modelos autorregresivos que asumen distribuciones adecuadas para el rango de las variables, como la distribución Poisson o la Binomial negativa, como se puede observar en Christou y Fokianos (2015), así como modelos

que se basan en metodologías Bayesianas para el análisis de series de tiempo con variables de conteo, por ejemplo el pronóstico a partir del algoritmo MCMC para distribuciones no normales, con una distribución Poisson como base, como se implementa en Nariswari y Pudjihastuti (2019).

Por otro lado, los métodos más recientes de pronóstico apalancados en las capacidades computacionales y el machine learning, proponen aproximaciones no paramétricas y robustas al incumplimiento de supuestos (Montero-Manso & Hyndman, 2021). Un atractivo de estas metodologías es la posibilidad de la automatización del proceso del ajuste de los modelos y de los pronósticos, lo que puede permitir la implementación de ciclos de modelamiento automático en sistemas de ejecución de tareas para muchos de los retos del mundo empresarial.

Apalancándose en estas capacidades computacionales, se ha observado un desempeño sobresaliente en el pronóstico de múltiples series de tiempo al utilizarlas como un conjunto para el entrenamiento de algoritmos diseñados para pronosticar las series de manera global (Montero-Manso & Hyndman, 2021). En el pronóstico global se utiliza la información de todas las series de tiempo del conjunto, en lugar del análisis individual para cada una, lo que permite disponer de los datos de todo el conjunto como insumo para superar el problema de la insuficiencia de datos para el entrenamiento, el cual había sido uno de los mayores desafíos para implementar metodologías de machine learning en el pronóstico de series de tiempo univariadas.

Una de las herramientas más importantes en muchas de las recientes aproximaciones de machine learning son las redes neuronales para el pronóstico de series de tiempo, cuya mayor ventaja es el buen desempeño que logran en muchos casos, así como una gran tolerancia a los datos con ruido o relaciones muy complejas entre variables (Shmueli et al., 2018), además, su funcionamiento requiere muy poca intervención humana, haciéndoles buenas candidatas para tareas de pronósticos automáticos (Shmueli & Lichtebdahl, 2018). Sin embargo, debe tenerse en cuenta algunas consideraciones en el uso de éstas: estos algoritmos son vistos comúnmente como “cajas negras” que capturan relaciones demasiado complejas para entenderlas (Shmueli & Lichtebdahl, 2018), aunque estos algoritmos logren capturar las relaciones de los datos observados con gran precisión, existe el riesgo de cometer extrapolaciones incorrectas en los pronósticos no supervisados por humanos, además su flexibilidad y funcionamiento depende enormemente de la disponibilidad de datos suficientes para el entrenamiento y al tener tiempos computacionales relativamente elevados es posible que generen demoras inaceptables en procesos de tomas de decisiones que requieran actuar en tiempo real (Shmueli et al., 2018).

Esta aproximación al problema del pronóstico por medio de redes neuronales tiene también la ventaja de no tener restricciones asociadas a supuestos paramétricos sobre la relación de las variables de entrada con el pronóstico (Allende et al., 2002), por lo que no presentan pro-

blemas relativos al incumplimiento de supuestos al implementarse sobre conjuntos discretos no negativos en lugar de datos sobre variables continuas.

Vaswani et al. (2017) mencionan que las redes neuronales recurrentes, redes neuronales de memoria larga de corto plazo, más conocidas como LSTM (*long short-term memory neuronal networks*) y las unidades recurrentes de compuerta, más conocidas como GRU (*Gated recurrent units*), se han establecido como parte del estado del arte para el modelamiento secuencial, sin embargo, las arquitecturas basadas en algoritmos de atención, llamadas Transformers, presentan mejores tiempos computacionales y de acuerdo con Zeng et al. (2022) son, posiblemente, la arquitectura más exitosa en el modelamiento secuencial y han tenido un auge reciente en su exploración para la aplicación al pronóstico de series de tiempo.

De lo anterior se evidencia que si bien existen diversas herramientas y enfoques que pudieran aplicarse en el problema del pronóstico simultáneo de múltiples series de tiempo de conteos, no hay consenso en la conveniencia de aplicar un determinado método por encima de otros. Este trabajo pretende explorar diversas metodologías para el pronóstico de múltiples series de tiempo con variables de conteo para determinar cuáles metodologías presentan los mejores resultados bajo diferentes escenarios, presentando también hallazgos respecto al uso práctico de las metodologías en entornos de grandes volúmenes de datos y cantidades de series de tiempo que se puedan escalar.

Para esta comparación, este trabajo explora metodologías estadísticas univariadas para series de tiempo de conteos y metodologías de machine learning para datos secuenciales, sus respectivas aplicaciones en el pronóstico simultáneo de múltiples series de tiempo obtenidas mediante simulación y la comparación de los pronósticos obtenidos, implementando en **Python** todas las herramientas algorítmicas necesarias de manera que se pudieran establecer comparaciones entre las metodologías evitando que cualquier diferencia entre éstas fuera dependiente de variaciones en el lenguaje de programación. Esto lleva a los siguientes aportes:

- Se formula una estructura matricial para el modelo de autorregresión Poisson en su forma lineal y log-lineal y las respectivas funciones de verosimilitud condicionales, con una estructura análoga a la representación de matriz de diseño y vector de parámetros del modelo de regresión lineal. A partir de ésta se obtiene el gradiente de la función de verosimilitud condicional respecto al vector de parámetros y se plantea la obtención de los parámetros del modelo mediante el algoritmo de descenso gradiente. Esta estructura se utiliza para la creación de un módulo en **Python** para el modelo de autorregresión Poisson, en el cual se incluye un algoritmo de selección automática de los órdenes autorregresivos del modelo basado en validación cruzada para optimizar una métrica de libre elección.

- Se parte del modelo de regresión Poisson para presentar el modelo de Red Neuronal y mostrar su equivalencia a un modelo de regresión generalizado no lineal. Se derivan las ecuaciones para el cálculo del valor ajustado y para la obtención de valores de los parámetros de una red neuronal equivalente a un modelo de regresión Poisson y se muestra su funcionamiento mediante implementación en Python. A partir de lo anterior se muestra el símil entre los modelos lineales espacio-estado y las redes neuronales recurrentes y se presentan casos de aplicación de estas, utilizando celdas LSTM, para el pronóstico de series de tiempo de conteos.

- Se propone una arquitectura basada en el modelo Transformer para el pronóstico de series de tiempo, se implementa en **Python** y se ilustra su funcionamiento en el pronóstico de series de tiempo de conteos.

- Tomando como referencia los trabajos y resultados previos hallados en la literatura revisada sobre el pronóstico de múltiples series de tiempo de variables continuas, en esta tesis se desarrolla un estudio de simulación con características controladas en la generación de los datos temporales, con el fin de establecer comparaciones entre las capacidades predictivas en las diferentes metodologías abordadas y a partir de los hallazgos obtenidos, dar luz sobre el uso de éstas y sobre el trabajo a seguir en la investigación del pronóstico de múltiples series de tiempo de conteos

Este trabajo sigue la siguiente estructura: en el Capítulo 2 revisan las bases teóricas del problema del pronóstico de series de tiempo de conteos, las metodologías estadísticas encontradas en la bibliografía para abordar este tipo de series, las redes neuronales y su uso en el pronóstico de series de tiempo y el pronóstico de múltiples series de conteos. El Capítulo 3 presenta las bases teóricas del modelo de autorregresión Poisson y los aportes mencionados. El Capítulo 4 se centra en las redes neuronales, desde sus bases conceptuales hasta el uso en el pronóstico de las RNN con celdas LSTM. El Capítulo 5 estudia el modelo Transformer y la propuesta adaptada a series de tiempo. Capítulo 6 presenta el estudio de simulación para la comparación de metodologías de pronóstico. Finalmente, el Capítulo 7 presenta las conclusiones alcanzadas en este trabajo y algunas notas sobre el trabajo futuro en la investigación del pronóstico de múltiples series de tiempo de conteos.

2. Estado del arte

2.1. Series de Tiempo

Una serie de tiempo es una sucesión de variables aleatorias indexadas por el tiempo, comúnmente observadas a intervalos regulares. Su análisis tiene múltiples áreas de aplicación, como en economía, análisis de mercados financieros, análisis deportivos, entre otros (Jia, 2018). De allí, que en las últimas décadas la modelación de las series de tiempo ha atraído la atención de un gran número de investigadores (Nariswari & Pudjihastuti, 2019) con el objetivo principal de ajustar modelos que describan lo mejor posible los patrones en las series de tiempo y provean interpretaciones aplicables a la realidad.

De las aplicaciones relacionadas con las series de tiempo, uno de los aspectos más cruciales es el problema del pronóstico (Christou & Fokianos, 2015), el cual puede entenderse como una predicción del futuro al estudiar el comportamiento previo de un fenómeno (Nariswari & Pudjihastuti, 2019). A nivel aplicado, en muchas actividades económicas, generar pronósticos precisos juega un rol crucial en la toma de decisiones del negocio. Por ejemplo, para grandes empresas de ventas al por menor como Amazon y Walmart, los pronósticos de ventas resultan fundamentales en la optimización del manejo de inventarios. En el sector energético se determinan precios de oferta, locación de combustible, entre otros. Sectores como el turismo y el cuidado de la salud también requieren pronósticos acertados de la demanda de servicios (Hewamalage et al., 2022).

Más fundamentalmente, podemos definir una serie de tiempo de longitud n como una sucesión de variables aleatorias $Y^{(1)}, Y^{(2)}, ..., Y^{(n)}$ denotada por $\{Y^{(t)}\}$ con $Y^{(t)} \in \mathbb{R}$, donde $t = 1, 2, ..., n$ representa el tiempo. En este trabajo se ha optado por colocar el índice t como superíndice en lugar de subíndice (como usualmente se usa en la literatura) con el fin de facilitar y simplificar la notación en los modelos del deep learning que más adelante son desarrollados, en los cuales son necesarios múltiples subíndices y superíndices en la notación de sus elementos y así mismo garantizar homogeneidad en la indexación temporal a lo largo de todos los capítulos. sea $\mu^{(t)} = E[Y^{(t)}]$ la media de la serie en el momento t y la función de auto covarianzas, es decir, de covarianzas entre variables del proceso separadas h periodos de tiempo, la función $\gamma(t, t+h)$ definida como,

$$\gamma(t, t+h) \coloneqq Cov(Y^{(t)}, Y^{(t+h)}), h \in \mathbb{Z}^+. \tag{2-1}$$

2.2. Series de tiempo de conteos

Las series de tiempo de conteos corresponden al caso en el cual el rango de las variables $Y^{(1)}, Y^{(2)}, ..., Y^{(n)}$ en la sucesión temporal corresponde a números enteros no negativos. El modelamiento del conteo de eventos puede encontrarse en todas las áreas de la estadística, economía y las ciencias (Fokianos, 2012). Los ejemplos incluyen desde el número de pacientes admitidos diariamente en un hospital, al número de transacciones por minuto de una determinada acción, o el total de accidentes mensuales en una región. A menudo se trata de procesos sobre dispersos (varianza superior a la media), con autocorrelaciones no negativas y con más conteos iguales a cero de lo que pueden explicar las distribuciones clásicas para conteos (Davis et al., 2021).

Para el caso de las series de conteo, los modelos Gaussianos tradicionales, como los modelos ARIMA, con una teoría para series estacionarias fuertemente desarrollada (Jia, 2018) tienen un desempeño pobre en muchos casos, además de no ajustarse a los supuestos de no negatividad y de datos que toman valores en el conjunto de los números enteros (Davis et al., 2021). Si bien Hyndman y Athanasopoulos (2021) mencionan que para series de conteos con valores suficientemente grandes, el efecto del incumplimiento de estos supuestos no tiene una gran importancia práctica, para conteos con valores pequeños (cercanos a cero) necesitamos utilizar métodos más apropiados para enteros no negativos. Los autores mencionan un método simple, llamado el método de Croston que, si bien no trata la naturaleza de conteo de las series, se utiliza a menudo para estos casos. El método construye dos series a partir de la original: Una con los valores diferentes a cero en la serie y otra con el número de valores hasta observar uno diferente a cero. Sin embargo, Shenstone y Hyndman (2005) concluyen que el método resulta arbitrario y sin una formulación apropiada del modelo estocástico subyacente.

Mientras que en el caso de las series con distribuciones marginales Gaussianas el modelo tradicional ARMA/ARIMA funciona bien, no existe una clase de modelos que domine la literatura en series de tiempo con datos de conteos (Jia, 2018), de modo que a continuación se exploran algunas metodologías que se han propuesto para la aproximación al problema del pronóstico de series de tiempo de conteos, que sí tratan con la naturaleza discreta y de valor entero de los datos, o no tienen supuestos paramétricos distribucionales que se vean afectados por ésta.

2.2.1. Modelos DARMA

Algunos intentos iniciales de modelar las series de conteos fue el uso de modelos de valores discretos autorregresivos de media móvil (DARMA por su sigla en inglés) introducidos en 1970's por Jacobs y Lewis (Jia, 2018). Según Jia (2018), el modelo utiliza mezclas para construir series con distribuciones marginales predeterminadas. Por ejemplo, una serie discreta autorregresiva $\{Y^{(t)}\}$ de primer orden (DAR(1)) se construye de series de variables de conteo $\{X^{(t)}\}$ independientes e idénticamente distribuidas, con una función de probabilidad acumulada marginal $F(x)$, y una sucesión IID de variables Bernoulli (llamada sucesión de afinación) $V^{(t)}$ con $P(V^{(t)} = 1) = p \in [0, 1]$. La serie de tiempo se analiza de manera recursiva, inicializando $Y_0 = X_0$ y haciendo recursivamente:

$$Y^{(t)} = V^{(t)}Y^{(t-1)} + (1 - V^{(t)})X^{(t)}, t = 1, 2... \tag{2-2}$$

Hay algunas propiedades indeseables para estos modelos (Jia, 2018). Para comenzar, puede haber largos periodos ajustados como constantes, problema que se hace mayor a mayor sea el valor de p. Por otro lado, estos modelos no pueden aproximarse a series correlacionadas de forma negativa con los valores del proceso en períodos anteriores, por lo que no puede esperarse que su estructura de covarianzas describa todo tipo de series de conteo estacionarias, razón por lo cual estos modelos perdieron popularidad en los años 1980's (Jia, 2018).

2.2.2. Modelos INARMA

En la investigación temprana de series de conteos, el objetivo, a menudo, era construir series estrictamente estacionarias soportadas en los enteros no negativos, con una variedad de funciones de autocorrelación y con funciones de distribución marginal como la Poisson y la binomial negativa (Davis et al., 2021). Según Davis et al. (2021) una línea de investigación actual descansa en los modelos enteros autorregresivos de media móvil (INARMA por su sigla en inglés), construidos al sumar operadores de afinación aplicados a las p observaciones anteriores con una innovación que se escoge de una familia paramétrica como la Poisson o la binomial negativa. Un ejemplo ilustrativo es el caso del modelo entero autorregresivo de orden uno (INAR(1)) para $\{Y^{(t)}\}$ el cual obedece a:

$$Y^{(t)} = \alpha \circ Y^{(t-1)} + \epsilon^{(t)},$$

donde $\circ$ denota el operador de afinación, que corresponde a una operación probabilística que se puede aplicar sobre conteos aleatorios para reducir su valor aleatoriamente (Aghababaei Jazi & Alamatsaz, 2012), y que actúa sobre la variable aleatoria de conteos $Y^{(t)}$ manteniendo sus valores enteros (Davis et al., 2021). Específicamente, el operador $\circ$ actúa sobre una

variable aleatoria de conteos Y vía $\alpha \circ Y = \sum_{i=1}^{Y} B_i$, donde $\{B_i\}_{i=1}^{\infty}$ es una sucesión de ensayos Bernoulli tal que $P(B_i = 1) = \alpha \in [0, 1]$ y $\{\epsilon^{(t)}\}$ es una sucesión IID de variables aleatorias de conteo con media $\mu_\epsilon > 0$ y varianza finita y constante σ_ϵ^2, independientes de la secuencia $\{B_i\}_{i=1}^{\infty}$ utilizada en el operador de afinación

Los procesos INARMA no tienden a permanecer constantes para corridas largas, pero también carecen de la capacidad de manejar correlaciones negativas, de modo que si bien pueden construirse órdenes de autorregresión de órdenes superiores y componentes de media móvil, no es posible obtener ninguna correlación negativa, Jia (2018).

2.2.3. Modelos autorregresivos para series de tiempo de conteos

En Davis et al. (2021) se define el concepto de modelos generalizados de espacio-estado como aquel definido por la especificación de la observación de la serie de tiempo $\{Y^{(t)}\}$ en el momento t dado un estado del proceso $\alpha^{(t)}$ y la especificación de la evolución del estado $\{\alpha^{(t)}\}$. Uno de dichos modelos consiste en la generalización simple de los modelos lineal y log-lineal autorregresivos. Fokianos (2012) presenta este modelo para el caso de la distribución Poisson como el modelo de regresión Poisson para series de tiempo y en Christou y Fokianos (2015) se habla de los modelos autorregresivos para series de tiempo de conteos, generalizando a otras distribuciones para enteros no negativos.

El modelo lineal autorregresivo Poisson de orden p corresponde a:

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = d + \sum_{j=1}^{p} b_j Y^{(t-j)}, \tag{2-3}$$

donde $F^{(t)} = \sigma(Y^{(t)}, Y^{(t-1)}, ...)$ es la historia del proceso, $d > 0$, $b_j > 0$ con $j = 1, 2, ..., p$ parámetros del modelo y $\lambda^{(t)} = E[Y^{(t)}|F^{(t-1)}]$ denota la media del proceso $Y^{(t)}$ dada su historia y $\lambda^{(t)} > 0$, ya que $Y^{(t)}$ es un entero no negativo. Este es un modelo lineal generalizado (Davis et al., 2021).

Sí consideramos el modelo de primer orden, es decir $p = 1$, dado por

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = d + b_1 Y^{(t-1)}, \tag{2-4}$$

tenemos la representación:

$$Y^{(t)} = d + b_1 Y^{(t-1)} + \epsilon^{(t)}, \text{ con } \epsilon^{(t)} = (Y^{(t)} - \lambda^{(t)}), \tag{2-5}$$

esta ecuación muestra que los valores del proceso en el tiempo t dependen de los valores en el momento $t-1$ más un proceso $\{\epsilon^{(t)}\}$ que corresponde al proceso de ruido, el cual es una sucesión de variables aleatorias incorrelacionadas de media cero y varianza constante, como se muestra en Fokianos (2012). De la ecuación (2-5) se puede observar que el modelo presenta una estructura similar que el caso usual $AR(1)$.

Por otro lado, en Davis et al. (2021) se presenta el modelo general de orden p, dado por

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = d + \sum_{j=1}^{p} b_j Y^{(t-j)}, \tag{2-6}$$

y mencionan que la función de autocovarianza de $\{Y^{(t)}\}$ tiene la misma estructura que aquella de los modelos $AR(p)$ clásicos.

Una variación es el modelo general de orden (p, q) propuesto por Ferland et al. (2006) dado por:

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = d + \sum_{i=1}^{p} a_i \lambda^{(t-i)} + \sum_{j=1}^{q} b_j Y^{(t-j)}. \tag{2-7}$$

El cual es un proceso estacionario de segundo orden sí $0 < \sum_{i=1}^{p} a_i + \sum_{j=1}^{q} b_j < 1$. En Fokianos (2012) se considera el modelamiento del proceso para el caso de primer orden dado por $\lambda^{(t)} = d + a_1\lambda^{(t-1)} + b_1 Y^{(t-1)}$, con $\epsilon^{(t)} = Y^{(t)} - \lambda^{(t)}$, de modo que el modelo para $Y^{(t)}$ queda de la forma:

$$Y^{(t)} = d + (a_1 + b_1)Y^{(t-1)} + \epsilon^{(t)} - a_1\epsilon^{(t-1)},$$

que puede escribirse como

$$\left(Y^{(t)} - \frac{d}{1-(a_1+b_1)}\right) = (a_1+b_1)\left(Y^{(t-1)} - \frac{d}{1-(a_1+b_1)}\right) + \epsilon^{(t)} - a_1\epsilon^{(t-1)}, \tag{2-8}$$

lo que muestra que el modelo tiene propiedades de segundo orden (es decir, aquellas que dependen de los dos primeros momentos, como por ejemplo la estacionariedad) idénticas a las del modelo $ARMA(1,1)$, lo que significa que, para $0 < a_1 + b_1 < 1$, existe una solución estacionaria $\{Y^{(t)}\}$ para el modelo, con media $E[Y^{(t)}] = \mu^{(t)} = d/(1-a_1-b_1)$ y función de autocovarianza (Fokianos, 2012).

$$\gamma(t,t+h) = \begin{cases} \frac{(1-(a_1+b_1)^2+b_1^2)\mu^{(t)}}{1-(a_1+b_1)^2}, h=0, \\ \\ \frac{b_1(1-a_1(a_1+b_1))(a_1+b_1)^{h-1}\mu^{(t)}}{1-(a_1+b_1)^2}, h \geq 1, \end{cases} \tag{2-9}$$

Para el caso general tenemos que $E[Y^{(t)}|F^{(t-1)}] = Var[Y^{(t)}|F^{(t-1)}]$ bajo la distribución Poisson, por lo que el modelo es denominado INGARCH(p,q), o modelo Entero GARCH, por tener una estructura análoga a la del modelo GARCH clásico, donde la volatilidad es una función de regresión en los valores pasados. De manera análoga, Davis et al. (2021) presenta una generalización para el modelo en forma log-lineal dada por:

$$v^{(t)} = d + \sum_{i=1}^{p} a_i v^{(t-i)} + \sum_{j=1}^{q} b_j \log(Y^{(t-j)} + 1), \tag{2-10}$$

cuyas propiedades se analizan en Fokianos y Tjøstheim (2011), donde $v^{(t)} = \log(\lambda^{(t)})$ y el término $\log(Y^{(t-j)} + 1)$ corresponde a una transformación uno a uno común en el manejo de valores nulos para evitar la indeterminación. Davis et al. (2021) menciona que las propiedades del modelo son similares a las del modelo lineal dado en (2-6).

Fokianos (2012) presenta otros modelos de tipo no autorregresivos y no lineales; por su parte, Christou y Fokianos (2015) consideran variaciones con distribuciones binomiales negativas. También otros modelos para series de conteos son revisados en Davis et al. (2021).

2.2.4. Modelos Bayesianos Dinámicos Generalizados

La aproximación Bayesiana al modelamiento de las series de conteos también ha sido un objeto activo de estudio (Davis et al., 2021). Gamerman et al. (2016) presentan un marco conceptual fundamentado en los modelos dinámicos lineales generalizados (DGLM), los cuales son un caso especial de los modelos de espacio-estado. Si se considera una serie de tiempo de valores de conteos $Y^{(t)}$ y denotamos con $\text{EF}(\mu, \phi)$ una distribución de la familia exponencial, de media μ y varianza $\phi c(\mu)$, con $c()$ una función cualquiera de la media, el modelo corresponde a:

Ecuación del valor observado: $Y^{(t)}|\boldsymbol{x^{(t)}}, \boldsymbol{\theta} \sim \text{EF}(\mu, \phi)$, (2-11)

Función de enlace: $g(\mu^{(t)}) = (\boldsymbol{z^{(t)}})^T \boldsymbol{x^{(t)}}$, (2-12)

Ecuación del sistema: $\boldsymbol{x^{(t)}} = \boldsymbol{G^{(t)}} \boldsymbol{x^{(t-1)}} + \boldsymbol{w^{(t)}}$, donde $\boldsymbol{w^{(t)}}|\boldsymbol{\theta} \sim N(\boldsymbol{0}, \boldsymbol{W})$, (2-13)

donde $\boldsymbol{z^{(t)}}$ es un vector conocido (posiblemente de covariables) en el momento t, $\boldsymbol{x^{(t)}}$ es un estado latente dependiente del tiempo en el momento t y $\boldsymbol{\theta}$ es un vector de hiperparámetros, incluyendo ϕ, con $\boldsymbol{G^{(t)}}$ y $\boldsymbol{W}$ componentes desconocidas. El modelo se completa especificando de forma a priori el estado latente $\boldsymbol{x^{(0)}}$, correspondiente al estado inicial cuando $t = 0$.

Gamerman et al. (2016) mencionan que el modelo más popular para series de tiempo de conteos es el modelo log-lineal dinámico especificado con distribución Poisson, dado por:

$$Y^{(t)}|\boldsymbol{x^{(t)}} \sim Poisson(\lambda^{(t)}), \quad t = 1, 2, \ldots, n \tag{2-14}$$

$$\log(\lambda^{(t)}) = (\boldsymbol{z^{(t)}})^T \boldsymbol{x^{(t)}}, \quad t = 1, 2, \ldots, n \tag{2-15}$$

$$\tag{2-16}$$

Las estimaciones de los parámetros y el estado latente pueden obtenerse de dos formas: secuencialmente o en un único bloque, que desde la perspectiva Bayesiana, implica la obtención de la sucesión de distribuciones de $(\boldsymbol{x^{(t)}}, \Theta)|F^t$ para $t = 1, 2, ...$ o para $[(\boldsymbol{x^{(1)}}, ..., \boldsymbol{x^{(t)}}, \Theta)|F^n]$ donde $F^t = F^{(0)}, F^{(1)}, ..., F^{(t)}$ es la sucesión de historias hasta el momento t, con $F^{(0)}$ que representa la información inicial.

Las distribuciones posteriores de los parámetros y el estado latente no pueden obtenerse analíticamente, por lo que deben utilizarse aproximaciones. Gamerman et al. (2016) presentan algunas técnicas que pueden utilizarse para dicha aproximación. De estas técnicas se destaca la aproximación INLA (Integrated Nested Laplace Approximation), una metodología que toma ventaja de la estructura latente de los modelos Gaussianos y combina una serie de aproximaciones determinísticas para obtener distribuciones marginales posteriores y los parámetros desconocidos del modelo, metodología implementada en **R** en el paquete INLA (Rue et al., 2009). Davis et al. (2021) mencionan también otros modelos no tan comúnmente aplicados que se derivan de la estructura de los modelos dinámicos generalizados, tales como: los modelos de nivel correlacionado, los modelos dinámicos jerárquicos multivariados y el modelo condicional Poisson basado en procesos log-gamma latentes multivariados.

2.2.5. Otros modelos

Si bien en el presente trabajo se abordan más en detalle las metodologías afines a los modelos lineales generalizados aplicados, en la literatura se han observado otras aproximaciones a las series de tiempo con datos de conteo, algunas de las cuales se mencionan a continuación.

- Davis et al. (2021) describen, tres aproximaciones para vectores de conteos bajo el paradigma dinámico bayesiano, en el cual los parámetros del modelo se asumen como variables aleatorias con distribuciones posteriores de interés.

- En Dunsmuir (2016) se presentan los Modelos Lineales Generalizados Autorregresivos y de Medias Moviles (GLARMA) como una clase de modelos estado-espaciales no Gaussianos y no lineales basados en observaciones en donde el estado, que no es observado, depende linealmente de covariables y no linealmente de valores pasados del proceso observado. Una aproximación similar es presentada en Sathish et al. (2020) donde se propone una clase de modelo de series de tiempo de conteos basado en observaciones para modelos cero inflados y sobre dispersos.
- En Lund y Livsey (2016) se presenta la aproximación al modelamiento con el paradigma de renovación, también presentado en Davis et al. (2021) entre las nuevas metodologías con objetivo de construir series de tiempo a partir de la distribución marginal acumulada, específicamente a partir de series binarias y la construcción de distribuciones marginales para procesos estacionarios a partir de transformaciones sobre procesos Gaussianos estacionarios.

2.3. Redes Neuronales y pronósticos de series de tiempo

Allende et al. (2002) mencionan que las redes neuronales artificiales (ANN por su sigla en inglés) han sido usadas en áreas de predicción y clasificación, donde metodologías como la regresión y otras técnicas estadísticas relacionadas habían sido usadas tradicionalmente.

Por otro lado, las redes neuronales se han estudiado como una alternativa a los modelos no lineales orientados por los datos, cuyo análisis depende de los datos disponibles, con muy poca información previa establecida sobre la naturaleza de la relación de las variables y el modelo. Según Allende et al. (2002) el proceso de construir relaciones entre las entradas y salidas del modelo se realiza mediante algoritmos de aprendizaje. Aún hoy existen algunos inconvenientes en la aplicación de las redes neuronales, relacionados con la gran cantidad de datos sobre los que se demanda su uso.

Los tres elementos principales de las redes neuronales son las unidades básicas de procesamiento llamadas neuronas, la arquitectura que describe las conexiones entre ellas y el algoritmo de entrenamiento con el que se encuentran los valores de los parámetros (Allende et al., 2002). Las neuronas se organizan en capas, donde la capa de las neuronas receptoras de los datos se conoce como capa de entrada, la capa de las neuronas que calculan la salida se llama capa de salida y aquellas entre la capa de entrada y la de salida se llaman “capas ocultas”. El número de capas se determina según la aplicación. Estos elementos pueden verse representados en la Figura **2-1** que ilustra de forma general el modelo básico de las redes neuronales, denominado el perceptrón multicapa.

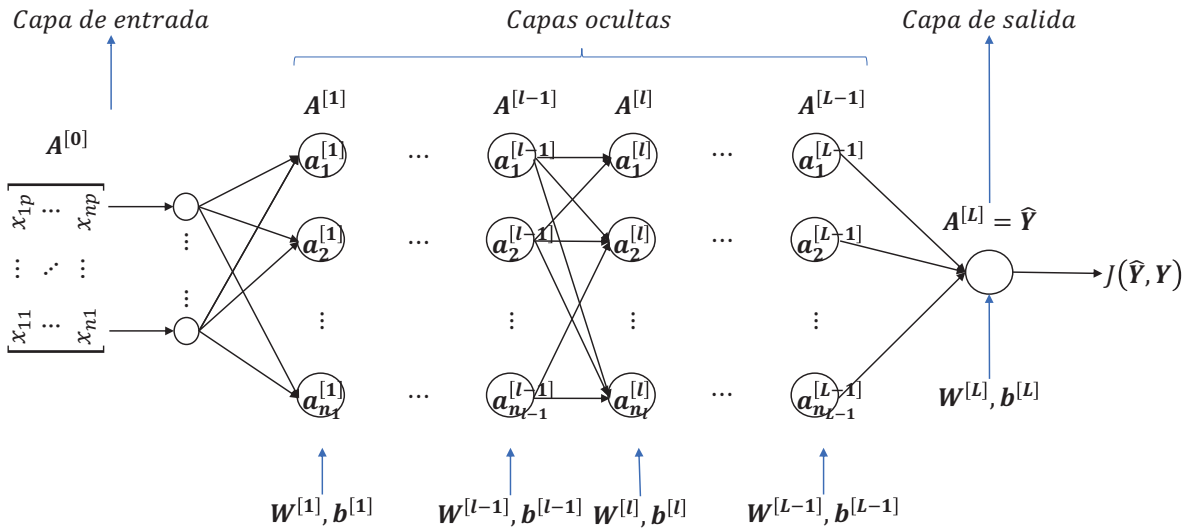

Figura 2-1.: Grafo computacional del perceptrón multicapa. Fuente: Elaboración propia

Según Allende et al. (2002) la clase que representa una neurona en el modelo de una red neuronal puede especificarse como:

$$S = \{g(\boldsymbol{x}, \boldsymbol{w}, b), \boldsymbol{x} \in \mathbb{R}^m, \boldsymbol{w} \in \mathbb{R}^p, b \in \mathbb{R}\}, \tag{2-17}$$

lo que representa el modelo de las neuronas que comparten la misma arquitectura parametrizada por el vector $\boldsymbol{w} = (w_1, w_2, ..., w_p)^T$, usualmente llamados pesos y el sesgo b, y considerando como datos de entrada $\boldsymbol{x} = (x_1, x_2, ..., x_p)^T$ para una única observación (la generalización matricial como se presenta en la Figura **2-1** será vista en el Capítulo 4). Sea $a_j^{[l]}$ el valor o activación de una neurona j en la capa l, tenemos que esta puede escribirse como función de los valores de las neuronas en la capa $(l-1)$ y de los parámetros de pesos $\boldsymbol{w}$ adicionando el parámetro de sesgo b, de la forma:

$$a_j^{[l]} = g\left(\sum_{i=1}^{n_{l-1}} w_i^{[l]} a_i^{[l-1]} + b^{[l]}\right),$$

donde n_{l-1} es el número de neuronas de la capa $(l-1)$ y la función $g(\cdot)$ se conoce como función de activación. La estimación de los parámetros $\boldsymbol{w}$ y b, se realiza minimizando de manera iterativa una función de costo $J(\boldsymbol{w}, b)$, utilizando en el proceso de optimización el algoritmo conocido como "back propagation" para el cálculo de las derivadas parciales de esta función con respecto a los parámetros de las diferentes capas. A este proceso de optimización se le conoce como el aprendizaje de la red. En el Capítulo 4 de este trabajo se presenta una descripción más detallada de estos modelos a partir de su relación con los modelos no lineales

generalizados y su representación como grafo computacional para facilitar la comprensión del proceso de cálculo de las salidas del modelo y de la estimación de los parámetros.

De acuerdo a Allende et al. (2002) las redes neuronales son esencialmente estrategias para una inferencia estadística no paramétrica, pues desde del punto de vista estadístico, tienen una interpretación simple: dada una muestra $D_n = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ generada por una función desconocida $f(\boldsymbol{x})$ con la adición de una componente estocástica ϵ, se tiene:

$$y_i = f(\boldsymbol{x_i}) + \epsilon_i, \tag{2-18}$$

la tarea del "aprendizaje" de la red neuronal es la construcción de un estimador $g(\boldsymbol{x}, \boldsymbol{w}, b) = \hat{f}(\boldsymbol{x_i})$ de $f(\boldsymbol{x_i})$, y desde que no se realiza ningún supuesto a priori sobre la forma funcional de $f(\boldsymbol{x_i})$, el modelo de la red neuronal $g(\boldsymbol{x}, \boldsymbol{w}, b)$ es un estimador no paramétrico de la esperanza condicional $E[y|\boldsymbol{x}]$, a diferencia del estimador paramétrico que asume a priori una forma funcional para $f(\boldsymbol{x_i})$. Esto permite que restricciones como el rango de la variable respuesta no sean inconvenientes para la aplicación del método, por lo que no hay problemas de supuestos en su aplicación sobre las series de tiempo de conteos, a diferencia de lo que sucede con muchos modelos estadísticos de series de tiempo clásicos.
Ahora bien, con relación al pronóstico de series de tiempo mediante redes neuronales, Vaswani et al. (2017) afirman las redes neuronales recurrentes (RNN), particularmente aquellas con compuertas de memoria larga de corto plazo (LSTM) y las de compuertas recurrentes (GRNN), se han establecido firmemente como el estado del arte para aproximarse al modelamiento secuencial.

De acuerdo a Vaswani et al. (2017), los modelos recurrentes suelen factorizar el cálculo según las posiciones de los símbolos de las secuencias de entrada y salida. Al alinear las posiciones con los pasos en el tiempo en que se calcula, generan una secuencia de estados ocultos $h^{(t)}$, en función del estado oculto anterior $h^{(t-1)}$ y la entrada para la posición t. Esta naturaleza inherentemente secuencial impide la paralelización dentro de los ejemplos de entrenamiento, lo que se vuelve crítico en secuencias de gran longitud, puesto que las limitaciones de memoria limitan el procesamiento por lotes entre ejemplos. Trabajos recientes han logrado mejoras significativas en la eficiencia computacional a través de trucos de factorización y cálculo condicional, al tiempo que mejoran el rendimiento del modelo en el caso de este último. Sin embargo, persiste la limitación fundamental del cálculo secuencial. Como solución a estos problemas Vaswani et al. (2017) proponen una arquitectura que depende completamente de algoritmos de atención con dependencias globales (no en secuencia) entre entradas y salidas, llamada Transformers, lo que permite mayor paralelización y menores tiempos computacionales.

Según Zeng et al. (2022) durante las últimas décadas, el pronóstico de series de tiempo ha evolucionado desde los métodos estadísticos tradicionales hacia el uso de estrategias de

machine learning y luego a soluciones basadas en aprendizaje profundo y particularmente los Transformers son posiblemente la arquitectura más exitosa para el modelamiento secuencial, demostrando un rendimiento sin paralelos en múltiples aplicaciones, como el lenguaje natural y el reconocimiento de voz. Recientemente, también ha habido un auge en la exploración del problema del pronóstico a largo plazo de series de tiempo con estas arquitecturas, con propuestas como la presentada por Farsani et al. (2021).

2.4. Pronóstico de múltiples series de tiempo

De acuerdo con Montero-Manso y Hyndman (2021) en muchas actividades económicas, como las ventas al por menor, energética y turismo, generar pronósticos precisos juega un rol crucial en la toma de decisiones. Un ejemplo, para compañías de ventas al por menor por comercio electrónico como Amazon, es el pronóstico de la demanda para posibilitar una mejor planeación de niveles de inventario, estrategias de precio competitivas, estrategias de promoción oportunas, entre otros (Bandara et al., 2019). En muchas ocasiones en la industria, este problema se presenta como el pronóstico de muchas series de tiempo como un grupo, tal como mencionan en Montero-Manso y Hyndman (2021), siendo algunos ejemplos, el pronóstico de llegadas de turistas a los resorts en la siguiente temporada, la demanda de productos ofrecidos en la venta minorista, la carga en los servidores de un centro de datos o el número de viajes compartidos completados en todas las zonas de una ciudad.

Según Hewamalage et al. (2022) anteriormente, un paradigma para el pronóstico de series de tiempo ha sido el tratar cada serie de tiempo como un conjunto de datos independiente. Como resultado, las técnicas tradicionales de pronóstico son univariadas, considerando cada serie de tiempo y su pronóstico de forma aislada y asumiendo que cada serie en el conjunto a pronosticar proviene de un proceso generador de datos diferente, aproximación que Montero-Manso y Hyndman (2021) denominan pronóstico *local*. De acuerdo con Hewamalage et al. (2022) las metodologías más prominentes para la aproximación local son los modelos espacio-estado de suavizado exponencial y los modelos autorregresivos integrados de media móvil (ARIMA).

Sin embargo, muchas compañías hoy en día están recolectando de forma cotidiana grandes cantidades de series de tiempo de fuentes similares, como pueden ser las ventas de miles de productos, el número de llamadas a líneas de atención por diferentes tipos de servicios o las llegadas de clientes a cientos de diferentes sucursales. Si bien las técnicas univariadas tradicionales pueden ser utilizadas para el pronóstico en estas circunstancias, hay un gran potencial para el aprendizaje de patrones a través de las series de tiempo provenientes de procesos similares que no son aprovechadas con dichas metodologías (Hewamalage et al., 2022).

Por otro lado, Montero-Manso y Hyndman (2021) resaltan que en la aproximación de pronóstico *local*, las dependencias temporales y la corta longitud de las series de tiempo hacen que el modelamiento no pueda realizarse de una forma automática basada en los datos, ya que fácilmente se puede caer en el problema del sobre ajuste y evitarlo requiere de la inclusión de información a priori sobre el fenómeno y la intervención humana, dificultando la escalabilidad del proceso por restricciones de tiempo.

Por los motivos anteriores, recientemente se ha dado un cambio en el paradigma para el pronóstico, donde ahora un conjunto de series de tiempo, en lugar de una sola, es visto como el conjunto de datos, entrenando los modelos a través de todas las series del conjunto, utilizando los mismos parámetros (Hewamalage et al., 2022). Montero-Manso y Hyndman (2021) llaman a esta aproximación la alternativa del pronóstico *global* o *aprendizaje cruzado*, la cual ha sido introducida para explotar el escenario en el cual todas las series del conjunto son similares, análogas o relacionadas. La idea tras esta aproximación es introducir el supuesto de que todas las series en el conjunto provienen de un mismo proceso, lo que, incluso sin ser cierto, tiene beneficios en términos de la precisión del pronóstico.

Según Montero-Manso y Hyndman (2021), el enfoque global supera la principal limitación del enfoque local: evita el sobre ajuste porque utiliza un tamaño de muestra mayor para el aprendizaje en comparación con su contra parte local. Por otro lado, se debe usar la misma función para pronosticar todas las series del conjunto, incluso cuando éstas sean diferentes, aunque esto parezca restrictivo y menos general que el enfoque local. La adopción más amplia de métodos globales ha sido impedida por la creencia de que solo ofrecen beneficios cuando las series de tiempo en el conjunto provienen de procesos similares o relacionados.

Algoritmos de aprendizaje local versus global

Sea $\mathbb{S}$ el conjunto de p de series de tiempo univariadas $\boldsymbol{y_i} = [y_i^{(1)}, y_i^{(2)}, \dots y_i^{(n)}]$ de longitud n, denotado $\mathbb{S} = \{\boldsymbol{y_i} \in \mathbb{R}^n, i = 1, \dots, p\}$, y permitimos que $\mathbb{S}$ contenga series repetidas (Montero-Manso & Hyndman, 2021). Si el interés es el pronóstico de los siguientes h pasos de tiempo de cada serie de tiempo, para cada una nos interesa un vector de valores futuros en $\mathbb{R}^h$. Las funciones de pronóstico son funciones que van de las series de tiempo observadas al futuro de interés, tales que:

$$f : \mathbb{R}^n \to \mathbb{R}^h$$

Sea A_L un algoritmo de aprendizaje local, es decir, una función que toma la serie $\boldsymbol{y_i}$ y devuelve una función de pronóstico f_i, tenemos

$$A_L : \mathbb{R}^n \to (\mathbb{R}^n \to \mathbb{R}^h)$$

El mismo algoritmo local es aplicado a cada serie de tiempo en $\mathbb{S}$, produciendo una función de pronóstico diferente para cada $\boldsymbol{y_i}$.

Por otro lado, un método global A_G es también un algoritmo de aprendizaje, pero únicamente produce una función de pronóstico para todo el conjunto $\mathbb{S}$.

$$A_G : \mathbb{R}^{p \times n} \to (\mathbb{R}^{p \times n} \to \mathbb{R}^{p \times h}).$$

En Montero-Manso y Hyndman (2021) se presenta la siguiente Proposición sobre la equivalencia de los algoritmos locales y globales para el pronóstico de series de tiempo:

Sean

- $\mathbb{A}_L$ = Conjunto de todos los algoritmos de aprendizaje local;
- $\mathbb{A}_G$ = Conjunto de todos los algoritmos deaprendizaje glocal;

Entonces

1. Para todo $A_G \in \mathbb{A}_G$, existe un $A_L \in \mathbb{A}_L$ tal que $(A_G(\mathbb{S}))(\boldsymbol{y_i}) = A_L(\boldsymbol{y_i})$
2. Para todo $A_L \in \mathbb{A}_L$, existe un $A_G \in \mathbb{A}_G$ tal que $A_L(\boldsymbol{y_i}) = (A_G(\mathbb{S}))(\boldsymbol{y_i})$

2.5. Comparación de métodos de pronóstico

Si bien los supuestos tras la aproximación global pueden resultar de difícil cumplimiento, de acuerdo con Montero-Manso y Hyndman (2021) tienen grandes beneficios en términos de la precisión del pronóstico y trabajos empíricos recientes muestran resultados desconcertantemente buenos en su aplicación sobre series de tiempo que no se consideraban como relacionadas. Un ejemplo es que en la competencia M4 (Makridakis et al., 2020), los mejores métodos utilizan alguna forma de globalidad, la mayoría en combinación con localidad. Sin embargo, se encontró también buenos resultados con métodos únicamente basados en la aproximación global.

El problema de entender cuándo y por qué los pronósticos globales fallan, es uno de los mayores interrogantes en los pronósticos de series de tiempo actuales (Hewamalage et al., 2022). Si bien Montero-Manso y Hyndman (2021) estudian este problema mediante la aplicación de metodologías sobre múltiples datos reales, encontrando beneficios de los modelos globales para aproximarse a relaciones más complejas, manteniendo además una generalidad del modelo que facilita detectar fallos algorítmicos con facilidad, Hewamalage et al. (2022) mencionan que aún quedan preguntas sin responder respecto a las circunstancias en las cuales los modelos funcionan y cuál es la complejidad ideal del modelo para conjuntos de datos de

ciertos tamaños y particularidades. Para lograr el control total sobre la dependencia entre las series del conjunto y sus características resulta ideal la realización de estudios de simulación.

Otro factor a tener en cuenta es el tipo de metodología que se utiliza para el pronóstico. En los resultados de la competencia M4, la cual hace parte de una serie de competencias cuyo objetivo es lograr avanzar en la precisión con las formas de pronóstico, se encontró que los mejores resultados fueron mediante modelos que utilizaban una combinación de metodologías, siendo más precisos los pronósticos puntuales y por intervalos obtenidos mediante un enfoque mixto entre modelos estadísticos y características de machine learning (Makridakis et al., 2020). En la competencia se encontró también que en general los enfoques individualmente estadísticos o de machine learning tienen una precisión de pronóstico baja, mostrando que las aproximaciones híbridas son el camino a seguir. Los autores resaltan que si bien los métodos de machine learning puros mostraron un desempeño pobre con relación a los estadísticos, tal como se encontró también en Makridakis et al. (2018), dichos modelos utilizaban lineamientos genéricos y algoritmos estándares sujetos a ser optimizados substancialmente.

Por otro lado, los estudios como los realizados por Montero-Manso y Hyndman (2021), Hewamalage et al. (2022) y la competencia M4, asi como otros estudios encontrados en una extensa revisión bibliográfica, se centran en el pronóstico de series de tiempo con valores en el conjunto de los reales. Cabe resaltar que en el caso de la competencia M4, donde el conjunto de las $100,000$ series utilizadas en la competencia provienen de datos reales, los organizadores re-escalaron las observaciones para lograr que las medidas estuvieran en conjuntos mayores o iguales a 10, alejándolas del rango de las ocurrencias de cero comunes en las series de tiempo de conteos. Esto deja un campo abierto a la exploración del comportamiento de las metodologías de pronóstico en este tipo de datos específicamente.

Métricas de desempeño de pronóstico

De acuerdo con Hyndman y Koehler (2006) si bien muchas medidas de la precisión de pronóstico han sido propuestas, muchas de estas no son aplicables de forma general, pueden dar valores de infinito o indefinidos y pueden llevar a resultados confusos. Dado que no existe un consenso en la literatura sobre cuál de las medidas disponibles debe usarse de forma general, Makridakis et al. (2018) proponen una medida a partir del promedio entre el sMAPE (Makridakis, 1993) y el MASE (Hyndman & Koehler, 2006) las cuales son dos de las medidas más populares para la precisión del pronóstico. La primera utiliza errores porcentuales, los cuales son independientes de la escala y es de fácil interpretación, y la segunda apunta a corregir problemas potenciales de la anterior medida, como el sesgo (Hyndman & Koehler, 2006), siendo una alternativa con mejores propiedades matemáticas. Estas medidas también fueron encontradas en otros trabajos revisados sobre el pronóstico de múltiples series de

tiempo, por ejemplo, en Montero-Manso y Hyndman (2021) se utiliza el MASE y Hewamalage et al. (2022) presentan tanto el sMAPE como el MASE.

Las fórmulas para el cálculo de las métricas (de forma univariada) son:

$$\text{sMAPE} = \frac{2}{h} \sum_{t=n+1}^{n+h} \frac{|Y^{(t)} - \widehat{Y}_n(t)|}{|Y^{(t)}| + |\widehat{Y}_n(t)|} * 100\,\% \tag{2-19}$$

$$\text{MASE} = \frac{1}{h} \frac{\sum_{t=n+1}^{n+h} |Y^{(t)} - \widehat{Y}_n(t)|}{\frac{1}{n-1} \sum_{t=2}^{n} |Y^{(t)} - Y^{(t-1)}|} \tag{2-20}$$

donde $Y^{(t)}$ es el valor de la serie de tiempo en el momento t y $\widehat{Y}_n(t)$ el pronóstico del modelo en dicho momento, h el horizonte de pronóstico y n el número de observaciones en el conjunto utilizado para el ajuste.

3. Modelos autorregresivos para series de tiempo de conteos

Davis et al. (2021) definen los modelos generalizados de espacio-estado como aquellos establecidos mediante la especificación de la observación de la serie de tiempo $\{Y^{(t)}\}$ en el momento t dado un estado del proceso $\alpha^{(t)}$ y la especificación de la evolución del estado $\{\alpha^{(t)}\}$. Uno de dichos modelos consiste en la generalización simple de los modelos lineal y log-lineal autorregresivos. Fokianos (2012) presenta este modelo para el caso de la distribución Poisson como el modelo de regresión Poisson para series de tiempo y en Christou y Fokianos (2015) se habla de los modelos autorregresivos para series de tiempo de conteos, generalizando a otras distribuciones para enteros no negativos.

El presente Capítulo retoma el modelo presentado en Fokianos (2012) que corresponde a un modelo autorregresivo para series de tiempo de conteos basado en la distribución Poisson, revisa las bases conceptuales del modelo y explica su ajuste y optimización de parámetros desarrollando un enfoque matricial y aplicando la metodología del descenso gradiente en el proceso de optimización.

En este trabajo de tesis se utiliza **Python** como el software de base debido a las ventajas que ofrece en la implementación de las aplicaciones de machine learning que se explorarán más adelante sobre series de tiempo de conteo. Sin embargo, el modelo autorregresivo Poisson no está implementado en este lenguaje, por esta razón, y con el fin de comparar los diferentes modelos bajo una misma herramienta computacional, se propone un módulo en **Python** para el modelo autorregresivo Poisson. Con un ejemplo de aplicación se ilustran los resultados del módulo creado y para evaluar su desempeño, se comparan con los resultados obtenidos mediante la función **tsglm** de la librería **tscount** Liboschik et al. (2017) en **R**.

3.1. Regresión Poisson e implementación en Python

Para facilitar la comprensión de la estructura del modelo autorregresivo Poisson y de su proceso de estimación, a continuación se presenta una breve introducción al modelo de regresión Poisson. De acuerdo con Fokianos (2012) la distribución Poisson es comúnmente usada para modelar la tasa de ocurrencia (o llegada) de eventos aleatorios en un intervalo de tiempo

fijo. Sea λ la media la variable aleatoria Y que denota el número de llegadas en un periodo de tiempo fijo. Si Y sigue una distribución Poisson su función de masa de probabilidad está dada por

$$P(Y = y) = \frac{\exp(-\lambda)\lambda^y}{y!}, \quad y = 0, 1, 2, ... \tag{3-1}$$

con $E(Y) = Var(Y) = \lambda$. Es importante resaltar que la distribución Poisson pertenece a la familia exponencial, definida por las distribuciones de densidad dadas por $f(x; \boldsymbol{\theta})$, con $\boldsymbol{\theta}$ un vector de parámetros, las cuales pueden ser expresadas como (Fokianos, 2012),

$$f(x; \boldsymbol{\theta}) = h(x) \exp(\boldsymbol{\theta} x - b(\boldsymbol{\theta})), \quad x \in \mathcal{A}, \tag{3-2}$$

donde $h(\cdot)$ y $b(\cdot)$ son funciones conocidas y $\mathcal{A}$ es un subconjunto de los reales.
El modelo general de regresión asume p variables regresoras $X_1, X_2, ..., X_p$ observadas junto a una variable respuesta Y que sigue la distribución Poisson (Fokianos, 2012). De forma simple, el modelo puede plantearse como

$$P(Y|X_1, X_2, ..., X_p) = \frac{\exp(-\lambda)\lambda^Y}{Y!}, \quad \text{es decir} \quad Y|X_1, X_2, ..., X_p \sim Poisson(\lambda), \tag{3-3}$$

con

$$\lambda = \beta_0 + \sum_{j=1}^{p} \beta_j X_j = \boldsymbol{x^T \beta}, \text{ con } \boldsymbol{x} = (1, X_1, X_2, \ldots, X_p)^T, \text{ y } \boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \ldots, \beta_p)^T. \tag{3-4}$$

Sin embargo, el ajuste de este modelo puede conducir a soluciones no apropiadas, dado que el parámetro λ debe ser positivo y la anterior ecuación no lo garantiza. Una elección más natural es el modelo log-lineal, dado por

$$\log(\lambda) = \beta_0 + \sum_{j=1}^{p} \beta_j X_j, \text{ con } \boldsymbol{x} = (1, X_1, X_2, \ldots, X_p)^T, \text{ y } \boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \ldots, \beta_p)^T. \tag{3-5}$$

Tanto el modelo lineal como el log-lineal pertenecen a la clase de los modelos lineales generalizados (GLM por su sigla en inglés) los cuales tienen tres componentes (Nelder & Wedderburn, 1972):

- Una componente aleatoria Y que pertenece a la familia exponencial y es tal que $E(Y) = \mu$.
- Una componente sistemática η.

- Una función de enlace, $g(\cdot)$, la cual debe ser dos veces diferenciable y asocia las dos componentes anteriores así: $g(\mu) = \eta$.

En el caso de los modelos de regresión Poisson, estos elementos corresponden a:

- Componente aleatoria: La variable respuesta Y con $Y|X_1, X_2, ..., X_p \sim Poisson(\lambda)$ y $E(Y|X_1, X_2, ..., X_p) = \lambda(X_1, X_2, ..., X_p)$.
- Componente sistemática: $\eta = \beta_0 + \sum_{i=1}^{p} \beta_i X_i$.
- Función de enlace: $\eta = g(\lambda) = \lambda$ para el modelo lineal y $\eta = g(\lambda) = \log(\lambda)$ para el log-lineal.

El vector de parámetros $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$ puede estimarse por el método de mínimos cuadrados generalizados o utilizando un descenso gradiente para estimar los valores que minimicen la función convexa dada por el negativo del logaritmo de la función de verosimilitud.

Si bien en **Python** existen funciones para la estimación de los modelos de regresión Poisson en librerías como *scikit-learn* (Pedregosa et al., 2011) o *statsmodels* (Seabold & Perktold, 2010), el presente trabajo propone para su estimación un algoritmo programado también en **Python**, basado en la estrategia de optimización por el método de descenso gradiente, la cual sirve posteriormente como base para el procedimiento computacional propuesto en la implementación de los modelos autorregresivos Poisson. Esta implementación preliminar en la estimación del modelo de regresión se toma como un referente inicial del funcionamiento del algoritmo del descenso gradiente programado, utilizando una función de costo cuya optimización es equivalente a maximizar la verosimilitud, y cuyos resultados pueden compararse con los obtenidos a través de las librerías de **Python** previamente mencionadas, con el fin de comprobar que la rutina de optimización implementada funciona adecuadamente antes de incorporarla en el procedimiento de estimación de los modelos autorregresivos que son de interés en esta tesis y para los cuales no existe alguna función en **Python**.

Implementación en Python

Considere el modelo general de regresión Poisson dado en (3-3) con vector de parámetros $\boldsymbol{\beta}$ y p predictores X_j, $j = 1, 2, \ldots, p$. Sea un conjunto de n observaciones independientes $(y_i, x_{i1}, x_{i2}, \ldots, x_{ip})$, entonces la función de verosimilitud para $\boldsymbol{\beta}$ es dada por

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{y_i!}, \text{ con } \lambda_i = E[y_i|x_{i1}, x_{i2}, \ldots, x_{ip}], \tag{3-6}$$

luego, el logaritmo de la verosimilitud corresponde a:

$$\log L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \times \log \lambda_i - \lambda_i - \log\left(y_i!\right) \right]. \tag{3-7}$$

Particularmente, bajo el modelo log-lineal en (3-5), la anterior ecuación se convierte en

$$\log L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[y_i \times (\boldsymbol{x}_i^T \boldsymbol{\beta}) - \exp(\boldsymbol{x}_i^T \boldsymbol{\beta}) - \log(y_i!) \right]. \tag{3-8}$$

Sea el vector de respuesta $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$ y la matriz de diseño asociada al modelo de regresión,

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_{11} & \ldots & X_{1p} \\ 1 & X_{21} & \ldots & X_{2p} \\ \vdots & \vdots & & \vdots \\ 1 & X_{n1} & \ldots & X_{np} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix}, \text{ con } \boldsymbol{x_i}^T = (1, x_{i1}, x_{i2}, \ldots, x_{ip}).$$

Podemos escribir $\log L(\boldsymbol{\beta})$ de la siguiente forma,

$$\log L(\boldsymbol{\beta}) = -C(\boldsymbol{\beta}) - \sum_{i=1}^{n} \log(y_i!), \tag{3-9}$$

donde

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[\exp(\boldsymbol{x}_i^T \boldsymbol{\beta}) - y_i \times (\boldsymbol{x}_i^T \boldsymbol{\beta}) \right]. \tag{3-10}$$

Si $\widehat{\boldsymbol{\beta}}$ es el estimador de máxima verosimilitud del vector $\boldsymbol{\beta}$, esto es, el vector $\boldsymbol{\beta}$ que maximiza a $\log L(\boldsymbol{\beta})$, entonces una vez hallado $\widehat{\boldsymbol{\beta}}$, las respuestas estimadas son calculadas como la estimación de la respuesta media condicional, es decir, en cada i,

$$\widehat{y}_i = \exp(\boldsymbol{x}_i^T \boldsymbol{\beta}), \tag{3-11}$$

de modo que el vector de respuestas estimadas corresponde a

$$\widehat{\boldsymbol{y}} = \begin{bmatrix} \exp(\boldsymbol{x_1}^T \widehat{\boldsymbol{\beta}}) \\ \exp(\boldsymbol{x_2}^T \widehat{\boldsymbol{\beta}}) \\ \vdots \\ \exp(\boldsymbol{x_n}^T \widehat{\boldsymbol{\beta}}) \end{bmatrix}, \text{ con } \boldsymbol{x_i}^T = (1, x_{i1}, x_{i2}, \ldots, x_{ip}), i = 1, 2, \ldots, n.$$

Ahora bien, observe que $\boldsymbol{\beta}$ que maximiza a $\log L(\boldsymbol{\beta})$ es también el $\log L(\boldsymbol{\beta})$ que minimiza $C(\boldsymbol{\beta})$ dado en (3-10). Esta función tiene como gradiente respecto a $\boldsymbol{\beta}$ a

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{n} \left[\mathbf{exp}(\boldsymbol{x_i^T\beta})\boldsymbol{x}_i - y_i\boldsymbol{x}_i\right] = \boldsymbol{X}^T(\mathbf{exp}(\boldsymbol{X\beta}) - \boldsymbol{y}), \tag{3-12}$$

donde $\mathbf{exp}(\boldsymbol{X\beta})$ es definido como:

$$\mathbf{exp}(\boldsymbol{X\beta}) = \begin{bmatrix} \exp(\boldsymbol{x_1^T\beta}) \\ \exp(\boldsymbol{x_2^T\beta}) \\ \vdots \\ \exp(\boldsymbol{x_n^T\beta}) \end{bmatrix}$$

A partir de la solución del sistema de ecuaciones presentado en (3-12) es posible obtener los estimadores de máxima verosimilitud, $\widehat{\boldsymbol{\beta}}$, hallando el valor del vector de parámetros que maximice a $C(\boldsymbol{\beta})$. En el repositorio que contiene las implementaciones de código de este trabajo, el cual es accesible en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast, puede encontrarse una implementación de esta metodología que se basa en la optimización de la función de verosimilitud mediante un descenso gradiente con el algoritmo Adam (ver Anexo A) dentro del módulo "PoissonRegression". A continuación, se presenta un ejemplo de simulación y la comparación de los resultados de estimación obtenidos mediante la implementación computacional propuesta versus los obtenidos mediante las funciones disponibles en las librerías de referencia anteriormente mencionadas, cuyo código puede revisarse en el notebook "poisson_regression"del repositorio mencionado.

Aplicación

Se define arbitrariamente el modelo dado por

$$y_i \sim P(\lambda_i),$$

y denotamos

$$\log \boldsymbol{\lambda} = \boldsymbol{X\beta}, \text{ o equivalentemente } \boldsymbol{\lambda} = \mathbf{exp}(\boldsymbol{X\beta}) = \begin{bmatrix} \exp(\boldsymbol{x_1^T\beta}) \\ \exp(\boldsymbol{x_2^T\beta}) \\ \vdots \\ \exp(\boldsymbol{x_n^T\beta}) \end{bmatrix},$$

donde $\boldsymbol{\beta} = [1, 2, 3]^T$, $\boldsymbol{x_i^T} = (1, x_{i1}, x_{i2}, \ldots, x_{ip}), i = 1, 2, \ldots, n$ y $\boldsymbol{X}$ es la matriz de diseño dada por

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_{11} & X_{12} \\ 1 & X_{21} & X_{22} \\ \vdots & \vdots & \vdots \\ 1 & X_{n1} & X_{n2} \end{bmatrix}. \tag{3-13}$$

Los valores de las observaciones del predictor X_1, denotados x_{i1}, son generados aleatoriamente de una distribución $U(0,1)$ en tanto que los valores de las observaciones de la segunda variable, denotados x_{i2}, provienen de una $N(0,1)$. A partir de los valores de los predictores se generan los valores de las observaciones de la variable respuesta, denotados y_i, utilizando el modelo previamente definido.

Se generan 1000 muestras de tamaño $n = 1000$ y se obtienen los valores ajustados con las librerías de referencia y la implementación propia. En la Tabla **3-1** se presentan resultados para los valores promedios de las estimaciones con los tres algoritmos:

Tabla 3-1.: Valores promedio de las estimaciones de los parámetros con los diferentes algoritmos

Parámetro	**Scikit learn**	**Stats models**	**Módulo propio**
β_0	1.0840	1.0001	0.9968
β_1	1.9293	1.9999	2.0007
β_2	2.9830	2.9999	3.0014

Lo que permite observar que en efecto se obtiene una convergencia a los valores reales de los parámetros utilizando la aproximación propuesta en el módulo creado, con un desempeño similar a los módulos de referencia.

Por otro lado, la Tabla **3-2** contiene los errores promedio de estimación obtenidos para los tres algoritmos, donde para cada parámetro β_j el error promedio se calcula como:

$$\bar{E_j} = \frac{1}{1000}\sum_{l=1}^{1000} E_{j,l}, \text{ con } E_{j,l} = \beta_{j,l} - \widehat{\beta}_{j,l} \tag{3-14}$$

Tabla 3-2.: Valores promedio de los errores de estimación

Parámetro	**Scikit learn**	**Stats models**	**Módulo propio**
β_0	0.8396	0.0001	-0.0032
β_1	-0.0707	0.0001	0.0007
β_2	-0.0170	0.0001	0.0014

y las desviaciones estándar de los errores, que para cada parámetro β_j se calculan como

$$S_{E_j} = \sqrt{\frac{1}{999}\sum_{l=1}^{1000}\left(E_{j,l} - \bar{E}_j\right)^2}, \tag{3-15}$$

se pueden observar en la Tabla **3-3**

Tabla 3-3.: Desviaciones estándar de los errores de estimación

Parámetro	**Scikit learn**	**Stats models**	**Módulo propio**
β_0	0.0374	0.0069	0.0113
β_1	0.0289	0.0059	0.0079
β_2	0.0129	0.0024	0.0042

De los resultados obtenidos podemos observar que el algoritmo con mejor desempeño, al presentar tanto un menor error medio como menor desviación estándar, es el de la librería statsmodels, sin embargo el algoritmo del módulo propuesto presenta un desempeño cercano, superando ambos el rendimiento de la librería sklearn. Si observamos los histogramas de los errores en la Figura **3-1** se confirma este comportamiento.

De lo anterior, se concluye que el algoritmo propuesto tiene un rendimiento competitivo frente a los algoritmos ya existentes en **Python**, y se pasará a utilizarlo en la implementación del modelo Autorregresivo Poisson.

3.2. Autoregresión Poisson

Si bien esta sección toma su nombre de la metodología presentada por Fokianos et al. (2009), el modelo es también introducido como Procesos GARCH para valores enteros (INGARCH por la sigla en ingles de Integer-values GARCH process) por Ferland et al. (2006), pero también es trabajado como modelo de regresión Poisson para series de tiempos de conteos en Fokianos (2012), entre otros trabajos en la literatura. De acuerdo a Gamerman et al. (2016) estos modelos pertenecen a la categoría del análisis de series de tiempo de conteos desde una perspectiva de modelos lineales generalizados y en Davis et al. (2021) se les menciona como una subclase de los modelos generalizados espacio-estado. En la literatura también se encuentran modelos de la misma naturaleza basados en otras distribuciones para datos de conteos (siendo la distribución binomial negativa la más común, además de la Poisson) y

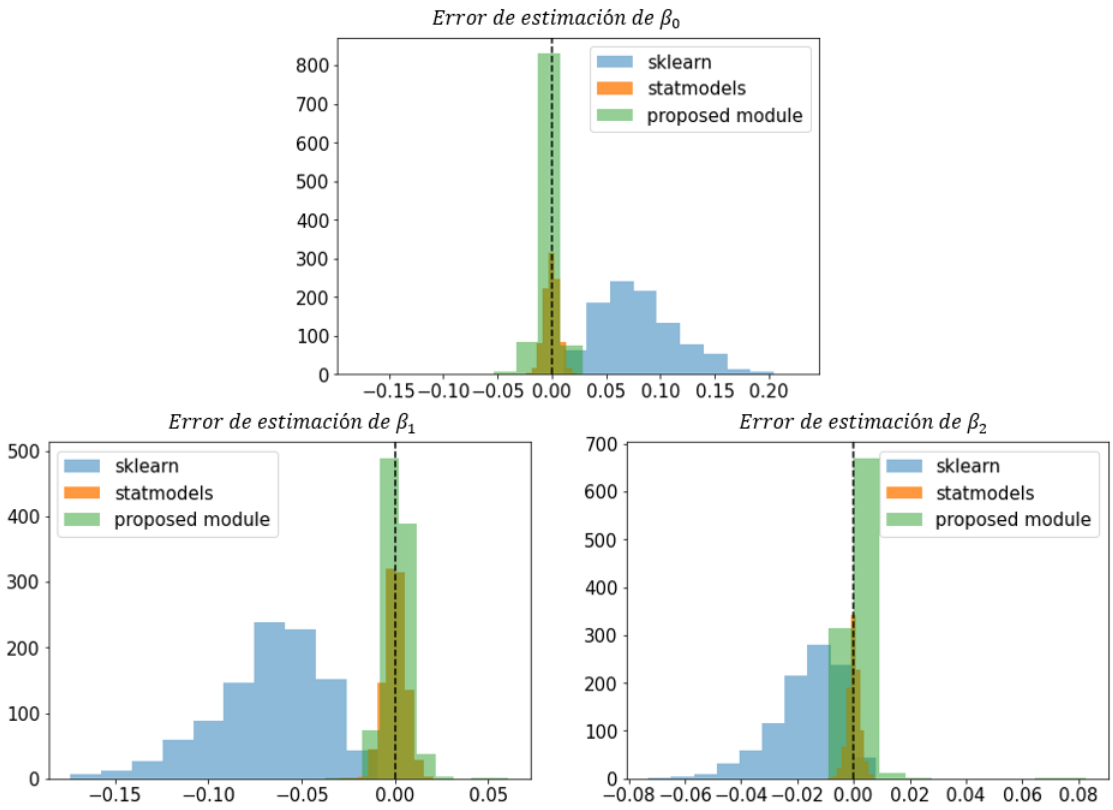

Figura 3-1.: Histogramas de los errores de estimación en las simulaciones para los parámetros β_0, β_1 y β_2

además modelos de tipo no lineales (ver Fokianos (2012), Gamerman et al. (2016), entro otros), sin embargo, en el presente trabajo nos centramos en el modelo lineal y log-lineal bajo el supuesto de distribución Poisson.

3.2.1. Modelo lineal

Si asumimos que $\{Y^{(t)}\}$ denota una serie de tiempo de conteo, llamada "respuesta", el modelo está dado por (Ferland et al., 2006):

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = d + \sum_{i=1}^{p} a_i \lambda^{(t-i)} + \sum_{j=1}^{q} b_j Y^{(t-j)}, \quad t \geq \text{máx}(p, q) \qquad (3\text{-}16)$$

con $0 < \sum_{i=1}^{p} a_i + \sum_{j=1}^{q} b_j < 1$ para garantizar estacionariedad de segundo orden (Ferland et al. (2006)).

Si consideremos el vector de parámetros $\boldsymbol{\theta} = (d, a_1, \ldots, a_p, b_1, \ldots, b_q)^T$ la función de verosimilitud condicional (Fokianos (2012)) está dada por:

$$L(\boldsymbol{\theta}) = \prod_{t=1}^{n} \frac{e^{-\lambda^{(t)}(\boldsymbol{\theta})} \left(\lambda^{(t)}(\boldsymbol{\theta})\right)^{Y^{(t)}}}{Y^{(t)}!} \tag{3-17}$$

y se tiene que el logaritmo de la verosimilitud $l(\boldsymbol{\theta})$ es tal que:

$$l(\boldsymbol{\theta}) \propto \sum_{t=1}^{n} \left[Y^{(t)} \log(\lambda^{(t)}(\boldsymbol{\theta})) - \lambda^{(t)}(\boldsymbol{\theta}) \right],$$

en tanto que el gradiente de $l(\boldsymbol{\theta})$ respecto a $\boldsymbol{\theta}$ es un vector de dimensión $p+q+1$ dado por:

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{n} \left(\frac{Y^{(t)}}{\lambda^{(t)}(\boldsymbol{\theta})} - 1 \right) \frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{3-18}$$

Si existe una solución para $\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0}$ tal que $l(\boldsymbol{\theta})$ es máximo, ésta corresponde al estimador de máxima verosimilitud de $\boldsymbol{\theta}$ que se denota $\widehat{\boldsymbol{\theta}}$.

Matricialmente, el modelo puede expresarse como:

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}) \quad con \quad \lambda^{(t)} = \left(\boldsymbol{x}^{(t)}\right)^T \boldsymbol{\theta}, \tag{3-19}$$

donde el vector temporal $\boldsymbol{x^{(t)}}$ y el vector de parámetros $\boldsymbol{\theta}$, corresponden a

$$\boldsymbol{x}^{(t)} = \begin{bmatrix} 1 \\ \lambda^{(t-1)} \\ \vdots \\ \lambda^{(t-p)} \\ Y^{(t-1)} \\ \vdots \\ Y^{(t-q)} \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} d \\ a_1 \\ \vdots \\ a_p \\ b_1 \\ \vdots \\ b_q \end{bmatrix},$$

sujeto a $0 < \sum_{i=1}^{p} a_i + \sum_{j=1}^{q} b_j < 1$.

Teniendo en cuenta la verosimilitud condicional dada en (3-17) y partiendo de la ecuación del gradiente respecto al vector de parámetros $\boldsymbol{\theta}$ de dimensión $p+q+1$ presentada en (3-18)

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{n} \left(\frac{Y^{(t)}}{\lambda^{(t)}(\boldsymbol{\theta})} - 1 \right) \frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}},$$

por la regla de la derivada total (Parr & Howard, 2018) tenemos que la derivada de $\lambda^{(t)}$ con respecto a $\boldsymbol{\theta}$ es un vector gradiente $\frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{1\times(p+q+1)}$ dado por

$$\frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial \left(\left(\boldsymbol{x}^{(t)}\right)^T \boldsymbol{\theta} \right)}{\partial \boldsymbol{\theta}} = \left(\boldsymbol{x}^{(t)}\right)^T + \boldsymbol{\theta}^T \frac{\partial \boldsymbol{x}^{(t)}}{\partial \boldsymbol{\theta}}, \tag{3-20}$$

Teniendo en cuenta que la derivada parcial de $\boldsymbol{x^{(t)}}$ con respecto a $\boldsymbol{\theta}$ ésta dada por el jacobiano de dimensiones $(p+q+1)\times(p+q+1)$ dado por:

$$\frac{\partial \boldsymbol{x}^{(t)}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \mathbf{0^T} \\ \frac{\partial \lambda^{(t-1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \lambda^{(t-p)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \mathbf{0^T} \\ \vdots \\ \mathbf{0^T} \end{bmatrix},$$

donde $\mathbf{0}$ es un vector columna de ceros de dimensión $p+q+1$, se tiene que

$$\frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = (\boldsymbol{x}^{(t)})^T + \begin{bmatrix} d & a_1 & \dots & a_p & b_1 & \dots & b_q \end{bmatrix} \begin{bmatrix} \mathbf{0^T} \\ \frac{\partial \lambda^{(t-1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \lambda^{(t-p)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \mathbf{0^T} \\ \vdots \\ \mathbf{0^T} \end{bmatrix},$$

con

$$\begin{bmatrix} \mathbf{0^T} \\ \frac{\partial \lambda^{(t-1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \lambda^{(t-p)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \mathbf{0^T} \\ \vdots \\ \mathbf{0^T} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \frac{\partial \lambda^{(t-1)}}{\partial d} & \frac{\partial \lambda^{(t-1)}}{\partial a_1} & \dots & \frac{\partial \lambda^{(t-1)}}{\partial a_p} & \frac{\partial \lambda^{(t-1)}}{\partial b_1} & \dots & \frac{\partial \lambda^{(t-1)}}{\partial b_q} \\ \vdots & & & & & & \\ \frac{\partial \lambda^{(t-p)}}{\partial d} & \frac{\partial \lambda^{(t-p)}}{\partial a_1} & \dots & \frac{\partial \lambda^{(t-p)}}{\partial a_p} & \frac{\partial \lambda^{(t-p)}}{\partial b_1} & \dots & \frac{\partial \lambda^{(t-p)}}{\partial b_q} \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix},$$

luego,

$$\frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} 1 & \lambda^{(t-1)} & \dots & \lambda^{(t-p)} & Y^{(t-1)} & \dots & Y^{(t-q)} \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^{p} a_i \frac{\partial \lambda^{(t-i)}}{\partial d} & \sum_{i=1}^{p} a_i \frac{\partial \lambda^{(t-i)}}{\partial a_1} & \dots & \sum_{i=1}^{p} a_i \frac{\partial \lambda^{(t-i)}}{\partial a_p} & \sum_{i=1}^{p} a_i \frac{\partial \lambda^{(t-i)}}{\partial b_1} & \dots & \sum_{i=1}^{p} a_i \frac{\partial \lambda^{(t-i)}}{\partial b_q} \end{bmatrix}.$$

Teniendo en cuenta que $\lambda^{(t)}(\boldsymbol{\theta})$ depende de los anteriores p valores, $\lambda^{(t-i)}(\boldsymbol{\theta}) \quad i = 1, 2, \dots, p$ y de los primeros $Y^{(t-j)} \quad j = 1, 2, \dots, q$ valores, los cuales son desconocidos, el modelo es definido para $t \geq \text{máx}(p, q)$, por tanto, también debe definirse del mismo modo la función de verosimilitud condicional. En el caso del modelo Arima (Dufour, 2008) se toman los primeros p valores como valores iniciales (reduciendo el número de observaciones). En este caso se toma como valores iniciales las $k \geq \text{máx}(p, q)$ observaciones y se reemplazan los primeros $\lambda^{(t-i)}(\boldsymbol{\theta}) \quad i = 1, 2, \dots, p$ valores de las tasas por los valores de $Y^{(t-j)} \quad j = 1, 2, \dots, p$ de modo que el proceso de obtener el estimador de máxima verosimilitud para el vector de parámetros $\boldsymbol{\theta}$ es equivalente al problema de minimizar a

$$C(\boldsymbol{\theta}) = -l(\boldsymbol{\theta}) = \sum_{t=k}^{n} (\lambda^{(t)}(\boldsymbol{\theta}) - Y^{(t)} \log(\lambda^{(t)}(\boldsymbol{\theta}))), \text{ con } k = \text{máx}(p, q), \tag{3-21}$$

con gradiente respecto al vector de parámetros,

$$\frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{t=k}^{n} \left(1 - \frac{Y^{(t)}}{\lambda^{(t)}(\boldsymbol{\theta})}\right) \frac{\partial \lambda^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{3-22}$$

que es equivalente al producto entre la matriz de dimensiones $(p+q+1) \times (n-k)$ dada por el producto entre $\left[\right] \left(\frac{\partial \lambda_k(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T, \dots, \left(\frac{\partial \lambda_n(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right)^T \Bigg]$ y el vector de dimensión $(n-k)$ dado por $\left[\left(1 - \frac{Y_k}{\lambda_k}\right), \dots, \left(1 - \frac{Y_n}{\lambda_n}\right)\right]^T$

$$\frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial \lambda_k}{\partial d} & \dots & \frac{\partial \lambda_n}{\partial d} \\ \frac{\partial \lambda_k}{\partial a_1} & \dots & \frac{\partial \lambda_n}{\partial a_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial \lambda_k}{\partial a_p} & \dots & \frac{\partial \lambda_n}{\partial a_p} \\ \frac{\partial \lambda_k}{\partial b_1} & \dots & \frac{\partial \lambda_n}{\partial b_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial \lambda_k}{\partial b_q} & \dots & \frac{\partial \lambda_n}{\partial b_q} \end{bmatrix} \begin{bmatrix} 1 - \frac{Y_k}{\lambda_k} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 1 - \frac{Y_n}{\lambda_n} \end{bmatrix}$$

Las ecuaciones (3-21) y (3-22) se utilizan para la estimación de los parámetros del modelo en el módulo creado en **Python**, que puede hallarse en el repositorio que contiene las implementaciones de código de este trabajo, el cual es accesible en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast.

3.2.2. Modelo log-lineal

Para este modelo Davis et al. (2021) presentan una generalización obtenida al considerar la función del enlace canónica $\nu^{(t)} = \log(\lambda^{(t)})$, de modo que si $Y^{(t)}$ denota la serie de tiempo de conteos, el modelo considerado es

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \nu^{(t)} = d + \sum_{i=1}^{p} a_i \nu^{(t-i)} + \sum_{j=1}^{q} b_j \log(Y^{(t-j)} + 1), \quad t \geq \text{máx}(p, q),$$

donde $F^{(t)}$ es la historia de proceso $\{Y^{(0)}, \ldots, Y^{(t)}, \nu_0\}$, es decir, la $\sigma - algebra$ $F^{(t)} = \sigma(Y^{(s)}, \nu_0, s \leq t)$. Las propiedades del modelo se estudian en Fokianos y Tjøstheim (2011), entre ellas la condición necesaria para la ergodicidad, que en el caso $p = q = 1$, siendo el vector de parámetros del modelo el vector tridimensional $\boldsymbol{\theta} = (d, a, b)^T$, está dada por $|a + b| < 1$. Sin embargo, en la revisión bibliográfica no se encuentran condiciones de ergodicidad para el caso de $p + q + 1$ parámetros con $p \geq 1$, $q \geq 1$, donde el vector de parámetros está dado por $\boldsymbol{\theta} = (d, a_1, \ldots, a_p, b_1, \ldots, b_q)^T$. Si bien en Liboschik et al. (2017) se introducen como condiciones $|a_1|, \ldots, |a_p|, |b_1|, \ldots, |b_q| < 1$ y $\left| \sum_{i=1}^{p} a_i + \sum_{j=1}^{q} b_j \right| < 1$ para el espacio paramétrico del modelo, las cuales son planteadas como una extensión razonable de las restricciones demostradas para el caso $p = q = 1$ pero no presentan una prueba formal para éstas, por lo que no se incluirán dichas restricciones en este caso.

Matricialmente el modelo puede escribirse,

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = \exp(\nu^{(t)}), \nu^{(t)} = \left(\boldsymbol{x}^{(t)}\right)^T \boldsymbol{\theta} \tag{3-23}$$

$$\boldsymbol{x}^{(t)} = \begin{bmatrix} 1 \\ \nu^{(t-1)} \\ \vdots \\ \nu^{(t-p)} \\ \log(Y^{(t-1)} + 1) \\ \vdots \\ \log(Y^{(t-q)} + 1) \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} d \\ a_1 \\ \vdots \\ a_p \\ b_1 \\ \vdots \\ b_q \end{bmatrix}.$$

La función de verosimilitud condicional para $\boldsymbol{\theta}$ dado un valor inicial $\lambda^{(0)} = e^{\nu^{(0)}}$, está dada por

$$L(\boldsymbol{\theta}) = \prod_{t=1}^{n} \frac{\exp(-\lambda^{(t)}(\boldsymbol{\theta}))(\lambda^{(t)}(\boldsymbol{\theta}))^{Y^{(t)}}}{Y^{(t)}!}, \tag{3-24}$$

luego, el logaritmo de la verosimilitud $l(\boldsymbol{\theta})$ es tal que:

$$l(\boldsymbol{\theta}) \propto \sum_{t=1}^{n}(Y^{(t)} \log(\lambda^{(t)}(\boldsymbol{\theta})) - \lambda^{(t)}(\boldsymbol{\theta})) = \sum_{t=1}^{n} \left[Y^{(t)}\nu^{(t)}(\boldsymbol{\theta}) - \exp\left(\nu^{(t)}(\boldsymbol{\theta})\right)\right] \tag{3-25}$$

El gradiente de $l(\boldsymbol{\theta})$ respecto a $\boldsymbol{\theta}$ es el vector $\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \in \mathbb{R}^{1\times(p+q+1)}$ dado por

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{t=1}^{n} \left[Y^{(t)} - \exp\left(\nu^{(t)}(\boldsymbol{\theta})\right)\right] \frac{\partial \nu^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}. \tag{3-26}$$

Utilizando la notación presentada en (3-23) se puede obtener que $\frac{\partial \nu^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ es igual a

$$\frac{\partial \nu^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial\left(\left(\boldsymbol{x^{(t)}}\right)^T \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}} = \left(\boldsymbol{x^{(t)}}\right)^T + \boldsymbol{\theta}^T \frac{\partial \boldsymbol{x^{(t)}}}{\partial \boldsymbol{\theta}}. \tag{3-27}$$

Teniendo en cuenta que la derivada parcial de $\boldsymbol{x^{(t)}}$ con respecto a $\boldsymbol{\theta}$ corresponde al jacobiano de dimensiones $(p+q+1)\times(p+q+1)$ dado por:

$$\frac{\partial \boldsymbol{x}^{(t)}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \mathbf{0^T} \\ \frac{\partial \nu^{(t-1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \nu^{(t-p)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \mathbf{0^T} \\ \vdots \\ \mathbf{0^T} \end{bmatrix},$$

donde $\mathbf{0}$ es un vector columna de zeros de dimensión $p+q+1$. Luego,

$$\frac{\partial \nu^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \left(\boldsymbol{x}^{(t)}\right)^T + \begin{bmatrix} d & a_1 & \dots & a_p & b_1 & \dots & b_q \end{bmatrix} \begin{bmatrix} \mathbf{0^T} \\ \frac{\partial \nu^{(t-1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \nu^{(t-p)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \mathbf{0^T} \\ \vdots \\ \mathbf{0^T} \end{bmatrix},$$

con

$$\begin{bmatrix} \mathbf{0^T} \\ \frac{\partial \nu^{(t-1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \vdots \\ \frac{\partial \nu^{(t-p)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ \mathbf{0^T} \\ \vdots \\ \mathbf{0^T} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \frac{\partial \nu^{(t-1)}}{\partial d} & \frac{\partial \nu^{(t-1)}}{\partial a_1} & \dots & \frac{\partial \nu^{(t-1)}}{\partial a_p} & \frac{\partial \nu^{(t-1)}}{\partial b_1} & \dots & \frac{\partial \nu^{(t-1)}}{\partial b_q} \\ \vdots & & & & & & \\ \frac{\partial \nu^{(t-p)}}{\partial d} & \frac{\partial \nu^{(t-p)}}{\partial a_1} & \dots & \frac{\partial \nu^{(t-p)}}{\partial a_p} & \frac{\partial \nu^{(t-p)}}{\partial b_1} & \dots & \frac{\partial \nu^{(t-p)}}{\partial b_q} \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix},$$

entonces, se tiene que,

$$\frac{\partial \nu^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \begin{bmatrix} 1 & \nu^{(t-1)} & \dots & \nu^{(t-p)} & log(Y^{(t-1)}+1) & \dots & log(Y^{(t-q)}+1) \end{bmatrix} + \\ \begin{bmatrix} \sum_{i=1}^{p} a_i \frac{\partial \nu^{(t-i)}}{\partial d} & \sum_{i=1}^{p} a_i \frac{\partial \nu^{(t-i)}}{\partial a_1} & \dots & \sum_{i=1}^{p} a_i \frac{\partial \nu^{(t-i)}}{\partial a_p} & \sum_{i=1}^{p} a_i \frac{\partial \nu^{(t-i)}}{\partial b_1} & \dots & \sum_{i=1}^{p} a_i \frac{\partial \nu^{(t-i)}}{\partial b_q} \end{bmatrix}.$$

Similarmente al caso lineal, podemos expresar el problema de hallar el estimador de máxima verosimilitud para el vector de parámetros como el problema de minimizar:

$$C(\boldsymbol{\theta}) = -l(\boldsymbol{\theta}) = \sum_{t=k}^{n} (\exp(\nu^{(t)}(\boldsymbol{\theta})) - Y^{(t)}\nu^{(t)}(\boldsymbol{\theta})) \quad k = \text{máx}(p,q), \tag{3-28}$$

con gradiente

$$\frac{\partial C(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{t=k}^{n} \left(\exp(\nu^{(t)}(\boldsymbol{\theta})) - Y^{(t)}\right) \frac{\partial \nu^{(t)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{3-29}$$

$$= \begin{bmatrix} \frac{\partial \nu_k}{\partial d} & \cdots & \frac{\partial \nu_n}{\partial d} \\ \frac{\partial \nu_k}{\partial a_1} & \cdots & \frac{\partial \nu_n}{\partial a_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial \nu_k}{\partial a_p} & \cdots & \frac{\partial \nu_n}{\partial a_p} \\ \frac{\partial \nu_k}{\partial b_1} & \cdots & \frac{\partial \nu_n}{\partial b_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial \nu_k}{\partial b_q} & \cdots & \frac{\partial \nu_n}{\partial b_q} \end{bmatrix} \begin{bmatrix} \exp(\nu_k(\boldsymbol{\theta})) - Y_k \\ \\ \\ \vdots \\ \\ \\ \exp(\nu_n(\boldsymbol{\theta})) - Y_n \end{bmatrix}.$$

Las ecuaciones (3-28) y (3-29) se utilizan para la estimación de los parámetros del modelo en el módulo creado en **Python** que puede hallarse en el repositorio que contiene las implementaciones de código de este trabajo, el cual es accesible en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast.

3.2.3. Valores ajustados y pronósticos

Partiendo del hecho de que $\lambda^{(t)} = E[Y^{(t)}|F^{(t-1)}]$ (Fokianos, 2012), la estimación de la serie de tiempo en t debe ser una estimación de esta media condicional, es decir,

$$\widehat{Y}^{(t)} = \widehat{\lambda}^{(t)} = \widehat{E}[Y^{(t)}|F^{(t-1)}], \text{ para } t = 1, 2, \ldots, n, \tag{3-30}$$

calculada de acuerdo a la estimación del modelo autorregresivo considerado. De forma similar, se puede definir la función de los pronósticos para h periodos después de $t = n$, es decir, el pronóstico de la observación futura $Y^{(n+h)}$, como la estimación de una media condicional, así,

$$\widehat{Y}_n(h) = \widehat{\lambda}_n(h) = \widehat{\lambda}^{(n+h)} = \widehat{E}[Y^{(n+h)}|F^{(n)}], \quad h \geq 1, \tag{3-31}$$

es decir, el pronóstico de $Y^{(n+h)}$ corresponde a una proyección de la respuesta media dada la historia hasta $t = n$.

Ahora bien, según el modelo autorregresivo, las ecuaciones de ajuste y pronóstico toman las siguientes formas:

- **Modelo lineal:**

  Ecuación de ajuste:

$$\widehat{Y}^{(t)} = \widehat{d} + \sum_{i=1}^{p} \widehat{a}_i \widehat{\lambda}^{(t-i)} + \sum_{j=1}^{q} \widehat{b}_j Y^{(t-j)}, \quad t \geq \max(p, q). \tag{3-32}$$

Ecuación de pronóstico:

$$\widehat{Y}_n(h) = \widehat{d} + \sum_{i=1}^{p} \widehat{a}_i \widehat{\lambda}_n(h-i) + \sum_{j=1}^{q} \widehat{b}_j \widehat{Z}_n(h-j). \tag{3-33}$$

donde

$$\widehat{\lambda}_n(h-i) = \begin{cases} \widehat{\lambda}^{(n+h-i)}, & si \quad h \leq i \\ \widehat{Y}_n(h-i), & si \quad h > i \end{cases}, \tag{3-34}$$

y

$$\widehat{Z}_n(h-j) = \begin{cases} Y^{(n+h-j)}, & si \quad h \leq j \\ \widehat{Y}_n(h-j), & si \quad h > j \end{cases}, \tag{3-35}$$

donde $\widehat{Y}_n(h-j)$ es el pronóstico para $Y^{(t+h-j)}$.

sea $\widehat{\boldsymbol{x}}_n(h)$ tal que

$$\widehat{\boldsymbol{x}}_n(h) = \begin{bmatrix} 1 \\ \widehat{\lambda}_n(h-1) \\ \vdots \\ \widehat{\lambda}_n(h-p) \\ \widehat{Z}_n(h-1) \\ \vdots \\ \widehat{Z}_n(h-q) \end{bmatrix}, \tag{3-36}$$

entonces, la ecuación de pronóstico puede escribirse como

$$\widehat{Y}_n(h) = [\widehat{\boldsymbol{x}}_{\boldsymbol{n}}(h)]^T \widehat{\boldsymbol{\theta}}.$$

- **Modelo log-lineal:**

  Ecuación de ajuste:

  $\widehat{Y}^{(t)} = \exp(\widehat{\nu}^{(t)})$, con

  $$\widehat{\nu^{(t)}} = \widehat{d} + \sum_{i=1}^{p} \widehat{a}_i \widehat{\nu}^{(t-i)} + \sum_{j=1}^{q} \widehat{b}_j \log(Y^{(t-j)} + 1), \quad t \geq \text{máx}(p, q) \tag{3-37}$$

  Ecuación de pronóstico

  $$\widehat{Y}_n(h) = \exp(\widehat{\nu}_n(h)), \quad \widehat{\nu}_n(h) = \widehat{d} + \sum_{i=1}^{p} \widehat{a}_i \widehat{W}_n(h-i) + \sum_{j=1}^{q} \widehat{b}_j \widehat{Z}_n(h-j) \tag{3-38}$$

  con

  $$\widehat{W}_n(h-i) = \begin{cases} \widehat{\nu}^{(n+h-i)}, & if \quad h \leq i \\ \widehat{\nu}_n(h-j), & if \quad h > i \end{cases}, \tag{3-39}$$

  y

  $$\widehat{Z}_n(h-j) = \begin{cases} \log(Y^{(n+h-j)} + 1), & if \quad h \leq j \\ \log(\widehat{Y}_n(h-j) + 1), & if \quad h > j \end{cases}, \tag{3-40}$$

  donde $\widehat{Y}_n(h-j)$ es el pronóstico para $Y^{(t+h-j)}$.

  sea $\widehat{\boldsymbol{x}}_n(h)$ tal que

  $$\widehat{\boldsymbol{x}}_n(h) = \begin{bmatrix} 1 \\ \widehat{W}_n(h-1) \\ \vdots \\ \widehat{W}_n(h-p) \\ \widehat{Z}_n(h-1) \\ \vdots \\ \widehat{Z}_n(h-q) \end{bmatrix} \tag{3-41}$$

entonces, la ecuación de pronóstico se puede escribir como

$$\widehat{\lambda}_n(h) = \exp(\widehat{\nu}_n(h)) = \exp\left([\widehat{\boldsymbol{x}}_n(h)]^T\widehat{\boldsymbol{\theta}}\right). \tag{3-42}$$

3.2.4. Selección de parámetros p, q

La implementación del algoritmo de autoregresión Poisson desarrollada en este trabajo incluye la selección automática del orden de autoregresión del modelo. Para el planteamiento del algoritmo de selección tengamos en cuenta que el modelo INGARCH(p,q) (acorde al nombre planteado en Ferland et al. (2006)) está dado por

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \lambda^{(t)} = d + \sum_{i=1}^{p} a_i\lambda^{(t-i)} + \sum_{j=1}^{q} b_j Y^{(t-j)}, \quad t \geq \text{máx}(p,q) \tag{3-43}$$

$$\boldsymbol{x}^{(t)} = \begin{bmatrix} 1 \\ \lambda^{(t-1)} \\ \vdots \\ \lambda^{(t-p)} \\ Y^{(t-1)} \\ \vdots \\ Y^{(t-q)} \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} d \\ a_1 \\ \vdots \\ a_p \\ b_1 \\ \vdots \\ b_q \end{bmatrix}, \tag{3-44}$$

con $0 < \sum_{i=1}^{p} a_i + \sum_{j=1}^{q} b_j < 1$ en su forma lineal y por

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}),$$

$$\nu^{(t)} = d + \sum_{i=1}^{p} a_i\nu^{(t-i)} + \sum_{j=1}^{q} b_j \log(Y^{(t-j)} + 1), \quad t \geq \text{máx}(p,q), \tag{3-45}$$

en su forma log-lineal, con

$$\nu^{(t)} = \left(\boldsymbol{x}^{(t)}\right)^T\boldsymbol{\theta},$$

donde

$$\boldsymbol{x}^{(t)} = \begin{bmatrix} 1 \\ \nu^{(t-1)} \\ \vdots \\ \nu^{(t-p)} \\ \log(Y^{(t-1)}+1) \\ \vdots \\ \log(Y^{(t-q)}+1) \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} d \\ a_1 \\ \vdots \\ a_p \\ b_1 \\ \vdots \\ b_q \end{bmatrix}, \tag{3-46}$$

Para la selección automática de los órdenes p, q se propone encontrarlos minimizando una métrica para el error del pronóstico sobre un conjunto de observaciones no usadas en la estimación del modelo. Particularmente, se utilizará el error cuadrático medio de predicción, definido en la estrategia de la validación cruzada sobre modelos de series de tiempo basada en el algoritmo k-fold presentado en Hyndman y Athanasopoulos (2021). En este caso, el error cuadrático medio o MSE (por su sigla en inglés), para el modelo con órdenes p, q, sobre h pronósticos después de $t = n$, es definido por

$$\text{MSE}(p,q) = \frac{1}{h}\sum_{i=1}^{h}(y^{(n+i)} - \widehat{y}_n(i))^2, \tag{3-47}$$

donde $y^{(n+i)}$ es la i-ésima observación de prueba que a su vez corresponde a la observación en $t = n + i$, en tanto que $\widehat{y}_n(i)$ es su pronóstico calculado con el modelo ajustado hasta $t = n$.

Esta métrica del MSE se calcula k veces al dividir la serie de tiempo en k conjuntos de entrenamiento y prueba y obtener la métrica en cada caso, como se visualiza en la Figura **3-2**, donde se aprecia que la longitud de la muestra usada para el ajuste es la franja azul (comenzando en $t = 1$) y la longitud de la franja roja es el tamaño de la muestra usada para la validación cruzada con las h observaciones siguientes a $t = n$.

De este modo, se inicia entrenando el modelo con un segmento pequeño de la serie de tiempo que va desde el inicio hasta un momento t, se pronostica para los siguientes h valores y se calcula el MSE acorde a (3-47). Luego, se expande la muestra de entrenamiento hasta $t+h$ y se pronostica desde $t + h + 1$ hasta $t + 2h$ y se continúa moviendo el segmento de prueba hasta llegar a la última observación disponible. Como resultado, se obtienen tantas particiones como segmentos de h valores puedan acomodarse entre el primer conjunto de entrenamiento y la última observación. El algoritmo creado en el módulo en **Python** permite la selección del número de particiones que se desea utilizar y estima el valor de h automáticamente a partir de este. En este trabajo se utilizaron 5 particiones para los ajustes.

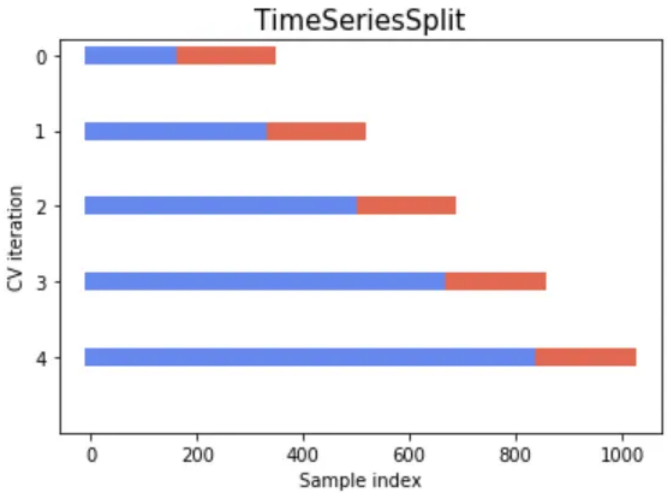

Figura 3-2.: División de una serie de tiempo para validación cruzada. Fuente: Shrivastava (2020).

Luego, el MSE medio obtenido por validación cruzada (que denotaremos CVMSE) se calcula al promediar todos los MSE para los diferentes conjuntos de prueba,

$$\text{CVMSE}(p,q) = \frac{1}{k}\sum_{j=1}^{k}\text{MSE}(p,q)_j,$$

donde $\text{MSE}(p,q)_j$ es el MSE obtenido para el j-ésimo conjunto de entrenamiento.

Finalmente, para encontrar p, q óptimos, el algoritmo de selección los busca como el par $p, q \geq 0$ tales que $\text{CVMSE}(p,q)$ es mínimo.

3.2.5. Implementación en Python

El código con la implementación de los modelos autorregresivos Poisson está disponible en la URL: https://github.com/dbeta95/On_time_series_of_counts_forecast, en el módulo denominado "PoissonAutoregression". En éste, se han aplicado dos lógicas diferentes de optimización, de acuerdo a si el modelo es lineal o el log-lineal, como se explica a continuación.

Para el caso lineal, el procedimiento de optimización de su verosimilitud condicional debe garantizar el cumplimiento de la condición necesaria para la estacionariedad del modelo, previamente enunciada como $0 < \sum_{i=1}^{p} a_i + \sum_{j=1}^{q} b_j < 1$. Por ello, aunque el procedimiento utiliza por defecto un descenso gradiente según el algoritmo Adam (ver Anexo A), incluye mensajes de advertencia cuando la solución a la cual converge no cumple con la condición

anterior. En este caso, se sugiere el uso de uno de los siguientes algoritmos de optimización restringida, que se encuentran disponibles en **Python** en la librería *SciPy* (Virtanen et al., 2020), y que pueden invocarse desde el módulo creado en este trabajo. Los algoritmos son los siguientes:

- Algoritmo de la región de confianza con restricciones (Trust-Region Constrained Algorithm) el cual está diseñado para problemas de minimización de la forma (Byrd et al., 1987):

  $$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \text{ sujeto a : } c^l \leq c(\boldsymbol{x}) \leq c^u, \boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u,$$

  donde las desigualdades entre vectores $\boldsymbol{x}^l \leq \boldsymbol{x} \leq \boldsymbol{x}^u$ denotan desigualdad elemento a elemento, y el algoritmo de optimización se basa en la solución sucesiva de subproblemas de programación cuadrática con la adición de una restricción de frontera de confianza.

- Algoritmo de programación de mínimos cuadrados secuenciales (Sequential Least Squares Programming Algorithm) el cual es un algoritmo de la familia de métodos de programación cuadrática secuencial (Sequential Quadratic Programming), uno de los métodos más exitosos para la solución numérica de problemas de optimización no lineal con restricciones de la forma (Hoppe, 2006):

  $$\min_{\boldsymbol{x}}(\boldsymbol{x}) \text{ sujeto a : } h(\boldsymbol{x}) = 0, g(\boldsymbol{x}) \leq 0,$$

  lo cual es alcanzado realizando un proceso iterativo de solución de problemas de programación no lineal.

Por otro lado, la estimación del modelo log-lineal se plantea como un problema de optimización sin restricciones y por ende no se garantiza su estacionariedad dado que en la revisión bibliográfica no se encontraron otras condiciones para ello que las propuestas en Liboschik et al. (2017) como una generalización razonable, pero sin proveer una prueba de éstas. Por ello, el único algoritmo que se incluye para este modelo en el módulo creado en **Python** es el descenso gradiente mediante el algoritmo Adam.

Como se mencionó previamente, hasta la fecha no se encuentra en **Python** implementaciones de los modelos autorregresivos Poisson, pero sí en **R**, es por esta razón que a continuación se tomará como marco de referencia la función **tsglm** de la librería de **R tscount** (Liboschik et al., 2017), con el fin de evaluar el desempeño del módulo en **Python** propuesto en esta tesis. Para ello, se presentan los resultados obtenidos al estimar los modelos autorregresivos

Poisson lineal y log-lineal sobre dos conjuntos de datos disponibles en **R**, el primero en la librería **tscount** y el segundo en el paquete base. Cabe resaltar que si bien se evalúa la calidad de los ajustes y se realizan análisis de residuos, con el fin de evaluar validez de algunos supuestos sobre los errores de los modelos aplicados, el objetivo perseguido por el momento no es determinar cuál es el modelo más apropiado para cada conjunto de datos, sino simplemente comparar los resultados del módulo propuesto en **Python** con los de la función **tsglm**, aun cuando los modelos probados no son necesariamente los más adecuados. Por ello, en primer lugar se prueba con los modelos autorregresivos más simples, es decir, los que solo contemplan rezagos de primer orden. Posteriormente, el modelamiento será refinado aplicando el algoritmo de selección automática según el procedimiento de validación cruzada previamente descrito y con el modelo resultante se evalúa también la calidad de ajuste y la validez de supuestos sobre los errores de ajustes.

Cabe resaltar que actualmente el módulo implementado en **Python** solo permite obtener estimaciones puntuales de los parámetros, así como estimaciones puntuales para los valores ajustados y pronósticos de la variable respuesta, pues no se aborda la programación de la estimación de la matriz de varianzas y covarianzas para el vector de parámetros estimados, dado que los modelos de machine learning, como se mencionó anteriormente, al no realizar ningún supuesto sobre la forma de la función de pronóstico ni supuestos estadísticos sobre los errores de ajuste, no tienen estructuras de varianzas y covarianzas definidas, por lo que la comparación que se realiza en este trabajo de tesis solo contempla evaluar la precisión del pronóstico puntual con las medidas de calidad de pronóstico presentadas en (2-19) y (2-20).

Caso 1: Infecciones de Campylobacter en Canada

En primer lugar, se utilizará el conjunto de datos correspondiente a casos de infecciones de campylobacter en el norte de la provincia de Quebec (Canada) en intervalos de cuatro semanas (28 días), entre enero de 1990 y octubre de 2000, con un total de 140 observaciones, incluido en el paquete **tscount** (Liboschik et al., 2017) en el objeto "campy". Para este caso, podemos denotar:

$Y^{(t)}$: Número de casos de infecciones por campylobacter en el norte de la provincia de Quebec en un momento t.

A continuación, en la Figura **3-3** se presenta el gráfico de la serie temporal en el panel superior izquierdo, la tendencia estimada mediante media móvil en el panel superior derecho y la distribución de los valores de acuerdo al periodo estacional (es decir, el número de cada uno de los 13 intervalos de 4 semanas en el año), panel inferior.

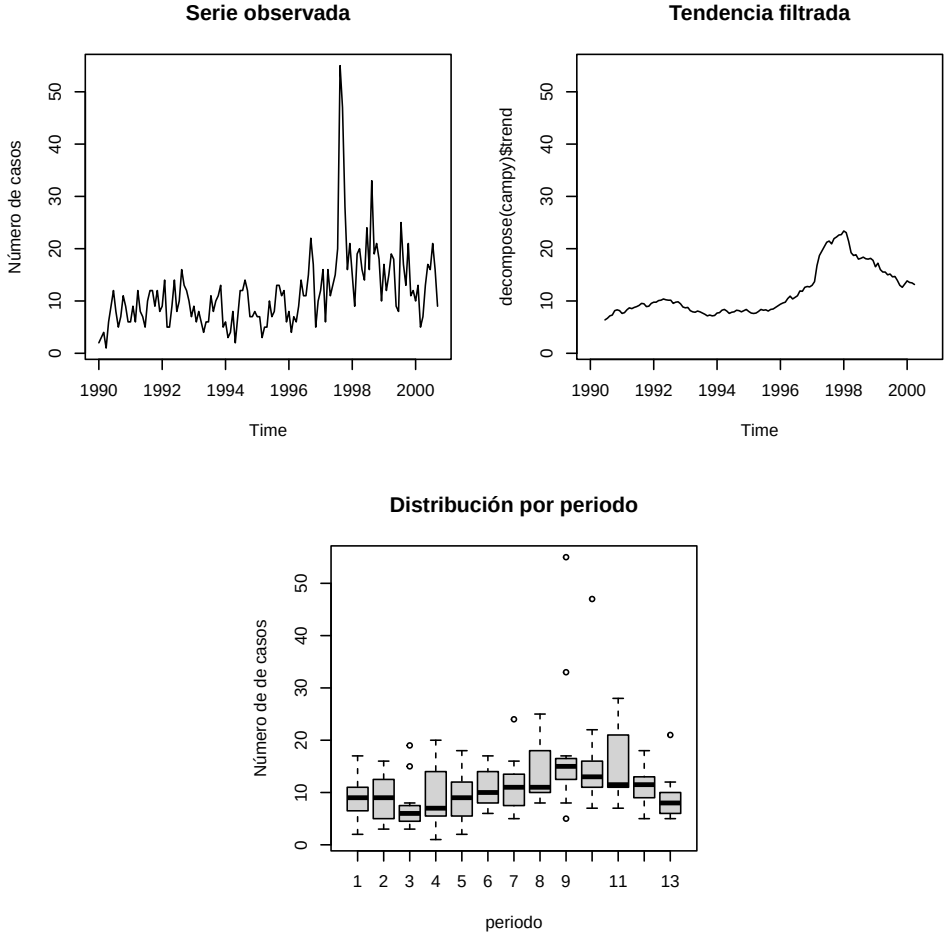

Figura 3-3.: Gráficos de la serie temporal, la tendencia filtrada, y la distribución por periodo estacional de los datos de infecciones por campylobacter.

La Figura **3-3** permite apreciar que la serie presenta un cambio de nivel. En cuanto al comportamiento estacional, parece existir la repetición de un patrón anual donde los promedios de la serie tienden a aumentar entre el octavo y onceavo período y ser particularmente bajos en el tercero. También se puede apreciar que la serie parece presentar heteroscedasticidad, con su varianza aumentando a la vez que lo hace la media.

En cuanto a la modelación, al aproximarse al ajuste de la serie con los modelos simples que solo consideran rezagos de primer orden, para el caso del modelo lineal, asumimos

$$Y^{(t)}|F^{(t-1)} \sim \text{Poisson}(\lambda^{(t)}), \quad \lambda^{(t)} = d + a_1\lambda^{(t-1)} + b_1 Y^{(t-1)}, \text{ con } d, a_1, b_1 > 0, 0 < a_1 + b_1 < 1.$$

Ajustando con la función **tsglm** en el paquete **tscount** se obtiene:

$$\widehat{Y}^{(t)} = 2{,}389 + 0{,}2693\widehat{\lambda}^{(t-1)} + 0{,}5183Y^{(t-1)},$$

y utilizando el módulo implementado en **Python** "PoissonAutorregression", se obtiene:

$$\widehat{Y}^{(t)} = 2{,}293 + 0{,}2881\widehat{\lambda}^{(t-1)} + 0{,}5191Y^{(t-1)}.$$

Apreciándose que en ambos casos se llegan a valores cercanos en la estimación de los parámetros.

Por otro lado, si se toma el caso del modelo log-lineal, se tiene

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}),\ \text{con}\ \lambda^{(t)} = \exp\left(\nu^{(t)}\right)\ \text{y}$$

$$\nu^{(t)} = d + a_1\nu^{(t-1)} + b_1 log(Y^{(t-1)} + 1), |a + b| < 1,$$

y en este caso al usar la función **tsglm** en el paquete **tscount** se obtiene:

$$\widehat{Y}^{(t)} = \exp\left(\widehat{\nu}^{(t)}\right),\ \text{con}\ \widehat{\nu}^{(t)} = 0{,}2917 + 0{,}2276\widehat{\nu}^{(t-1)} + 0{,}637\log(Y^{(t-1)} + 1),$$

mientras que utilizando el módulo implementado en **Python**:

$$\widehat{Y}^{(t)} = \exp\left(\widehat{\nu}^{(t)}\right),\ \text{con}\ \widehat{\nu}^{(t)} = 0{,}4043 + 0{,}2425\widehat{\nu}^{(t-1)} + 0{,}5858\log(Y^{(t-1)} + 1).$$

En las Figuras **3-4** y **3-5** se puede observar el ajuste de los modelos lineal y log-lineal de la serie de tiempo, mediante la rutina de estimación en la librería **tscount** y mediante la rutina propuesta en **Python**, respectivamente. En estas figuras se observa que los valores ajustados por los modelos lineal y log-lineal son muy similares entre sí, tanto con la librería **tscount** como con el módulo creado en **Python**, además, exhiben un patrón de desplazamiento a la derecha respecto a los valores de la serie observada, lo cual indica carencia de ajuste. Sin embargo, recuerde que el objetivo no es evaluar en sí la calidad de ajuste de estos modelos, sino determinar si hay grandes diferencias entre los resultados de estimación de las rutinas **R** y del módulo programado en **Python**. Vemos a través de las figuras **3-4** y **3-5** que los ajustes logrados por ambas rutinas son similares. Por otro lado, comparando las ecuaciones estimadas, vemos que aun cuando existen diferencias en los valores de los parámetros estimados en cada tipo de modelo, éstas no son muy grandes.

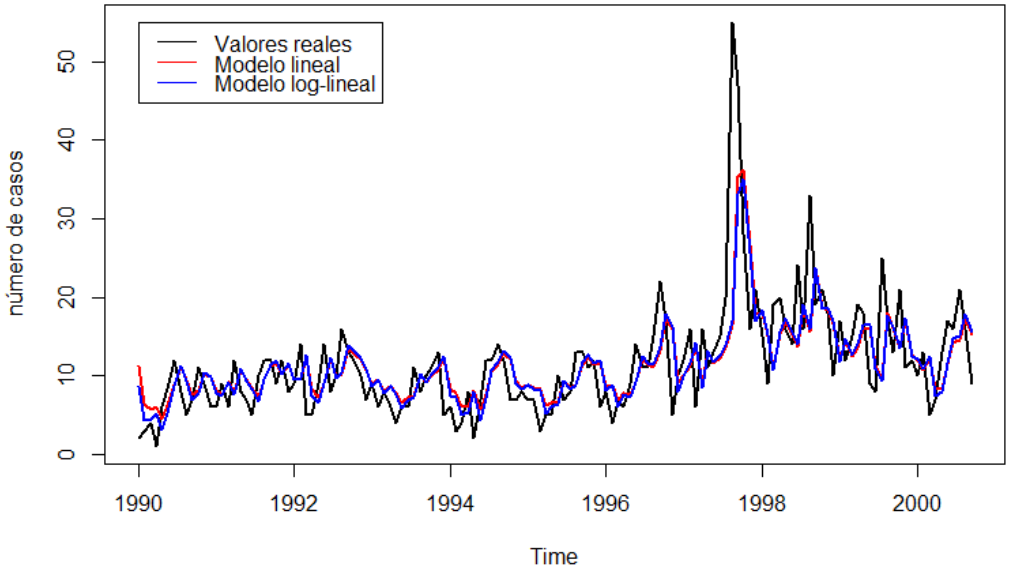

Figura 3-4.: Número de casos de infecciones por campylobacter en el norte de la provincia de Quebec contra valores ajustados con el paquete **tscount**

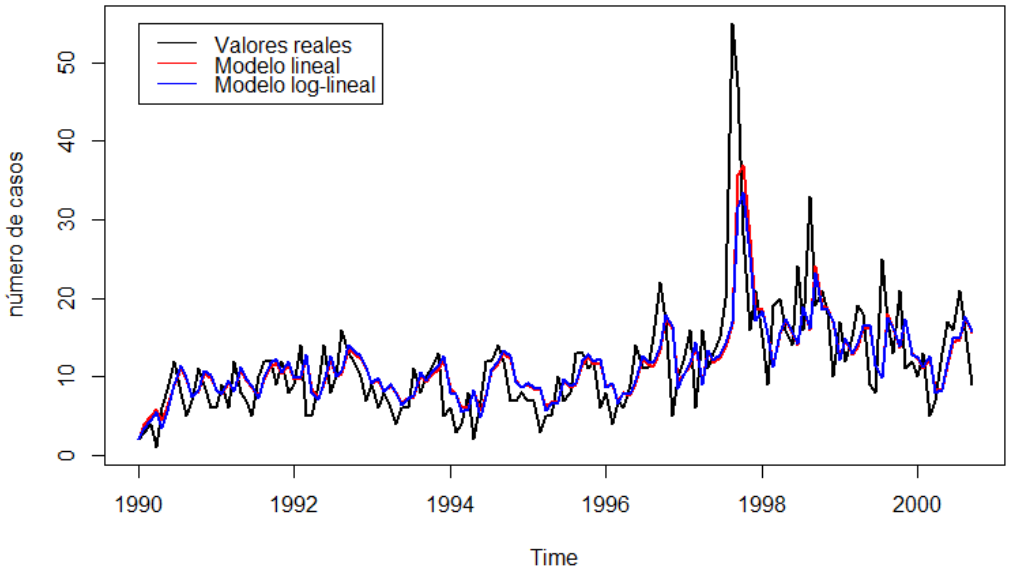

Figura 3-5.: Número de casos de infecciones por campylobacter en el norte de la provincia de Quebec contra valores ajustados con módulo implementado

Para el análisis del cumplimiento de los supuestos sobre los residuos del ajuste se calculan como:

$$\widehat{\epsilon}^{(t)} = y^{(t)} - \widehat{y}^{(t)}. \tag{3-48}$$

Es importante señalar que, si bien en la literatura sobre modelos de regresión para datos de conteo, han sido formulados varios tipos de residuos escalados, como los tipo Pearson (Feng et al., 2020), y particularmente Fokianos (2012) deduce la ecuación para este tipo de residuos teniendo en cuenta el modelo autorregresivo Poisson, de forma general su cálculo requiere la aproximación a la varianza de la variable respuesta estimada, la cual no se tiene para los modelos de machine learning, por lo cual, a fin de para mantener la homogeneidad en el análisis de los residuos de ajuste en los distintos modelos abordados en este trabajo de tesis, se opta por utilizar los residuos simples presentados en (3-48).

En los gráficos de residuos del ajuste a lo largo de este trabajo se calculan las bandas de variabilidad a dos desviaciones estándar muestral que permiten analizar más fácilmente el supuesto de varianza constante y verificar que solo algunas observaciones atípicas superan dichos límites (del orden del 5 % bajo una aproximación a la normalidad).

Para analizar el cumplimiento de supuestos de los errores del modelo lineal ajustado con el paquete **tscount**, en primer lugar, en la Figura **3-6** se presentan los gráficos de los residuos del ajuste versus el tiempo y versus los valores ajustados, así como la ACF obtenida a partir de estos, con el fin de evaluar el cumplimiento de los supuestos de que los errores de ajuste son una sucesión de variables incorrelacionadas de media cero y varianza constante. En esta puede observarse que, si bien en el gráfico de autocorrelaciones se muestra una caída rápida en las correlaciones, se observan cortes en los rezagos 12 y 13 que presentan evidencia contra del supuesto de independencia de los errores de ajuste. Por otro lado, el gráfico de la serie de residuales contra el tiempo parece presentar un patrón de tipo estacional que habla de problemas de ajuste y parecen existir problemas en el cumplimiento del supuesto de varianza constante. Además, en el gráfico de los residuos versus el tiempo es claro que existe carencia de ajuste debido al corrimiento por debajo de cero en la media de los residuos, hasta $t < 80$, lo cual es consecuencia de la carencia de ajuste que se detecta al observar la gráfica de ajuste y presenta evidencia contra el supuesto de media cero.

Por otro lado, en cuanto al modelo estimado con el módulo propuesto en **Python** se presentan los gráficos de los residuos del ajuste versus el tiempo y versus los valores ajustados y la ACF obtenida con los residuos del ajuste en la Figura **3-7**, donde se puede observar que los resultados son muy similares a los obtenidos con el modelo ajustado con la librería **tscount** en **R** y las conclusiones sobre estas gráficas son las mismas que las establecidas con la Figura **3-6**

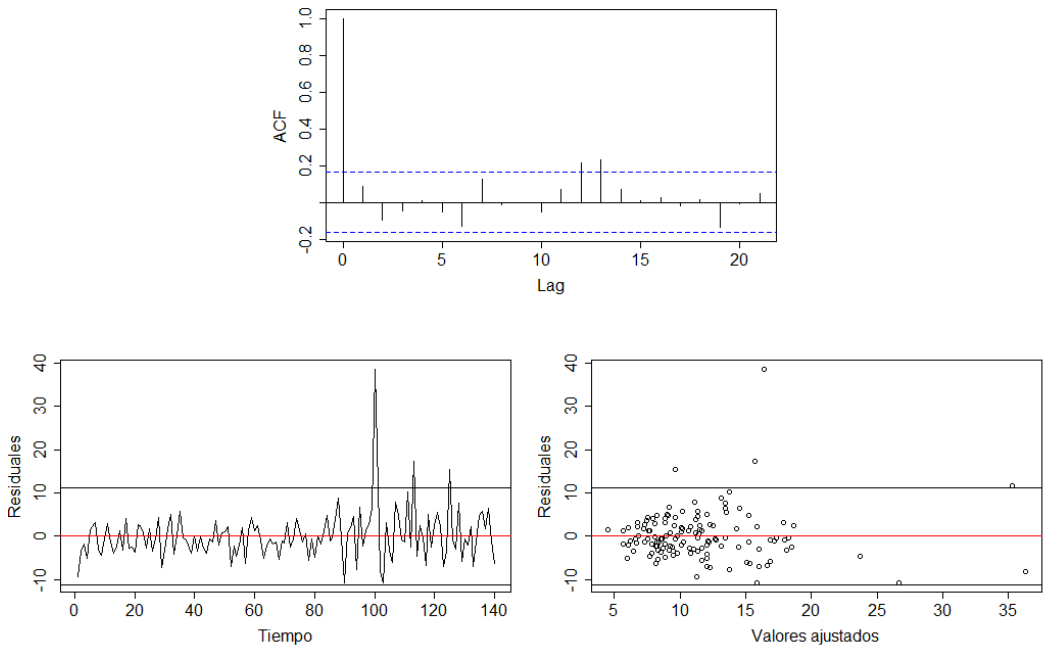

Figura 3-6.: Gráficos para el análisis de supuestos de los errores para el modelo lineal ajustado en **R**

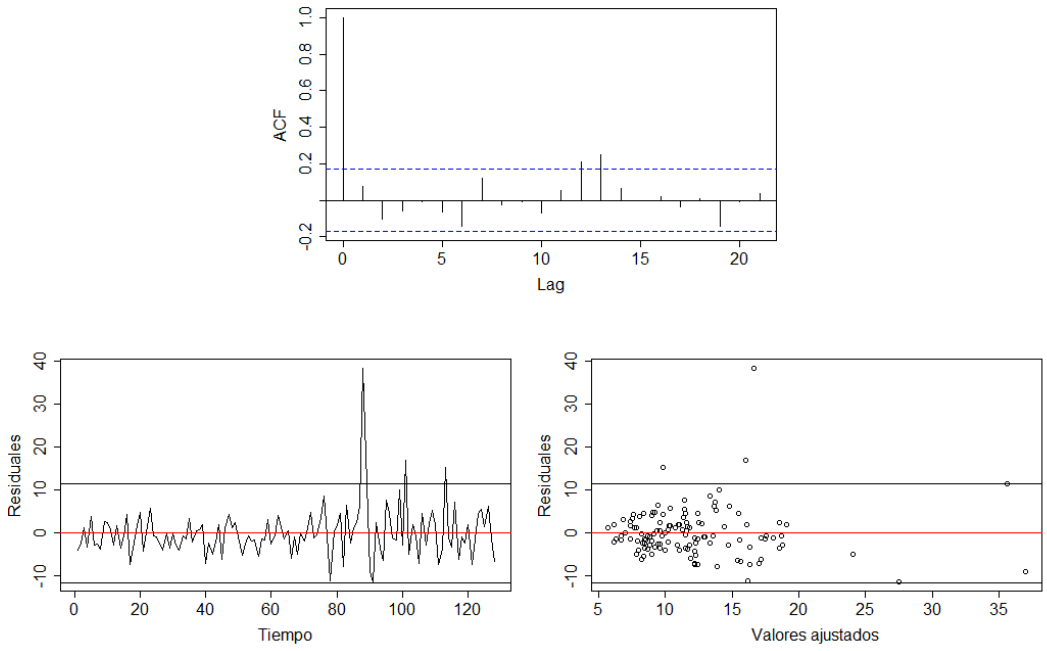

Figura 3-7.: Gráficos para el análisis de supuestos de los errores para el modelo lineal ajustado en **Python**

Para el análisis del cumplimiento de supuestos de los errores del modelo log-lineal obtenido con el paquete **tscount** se presentan los gráficos, ya enunciados en los modelos anteriores, en la Figura **3-8**, y para el modelo log-lineal obtenido con el módulo propuesto en **Python** se presenta la Figura **3-9**.

En las Figuras **3-8** y **3-9** se puede observar que nuevamente el comportamiento es muy similar tanto en el modelo implementado en R como en **Python**, además, las conclusiones alcanzadas también son muy similares con respecto al modelo lineal.

Como métrica para comparar el ajuste de los modelos se utiliza el error cuadrático medio. Si bien su ecuación es similar a la presentada en (3-47) para el error de pronóstico, el MSE de ajuste no penalizado por grados de libertad se calcula dentro de la muestra de ajuste como

$$\text{MSE}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \frac{1}{n}\sum_{t=1}^{n}(y^{(t)} - \widehat{y}^{(t)})^2, \tag{3-49}$$

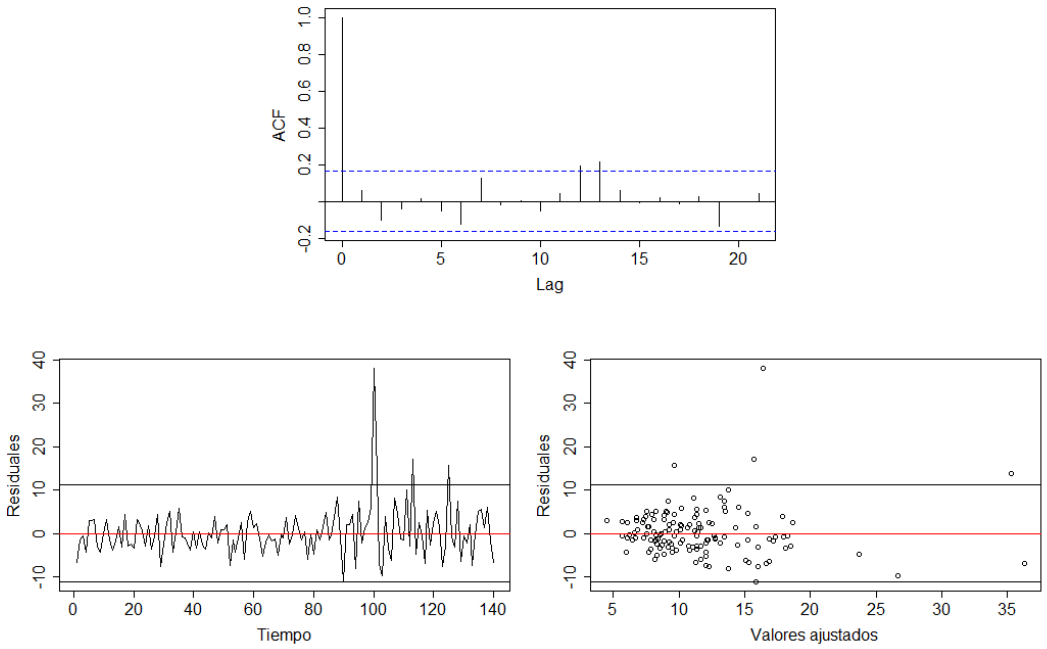

Figura 3-8.: Gráficos para el análisis de supuestos de los errores para el modelo log-lineal ajustado en **R**

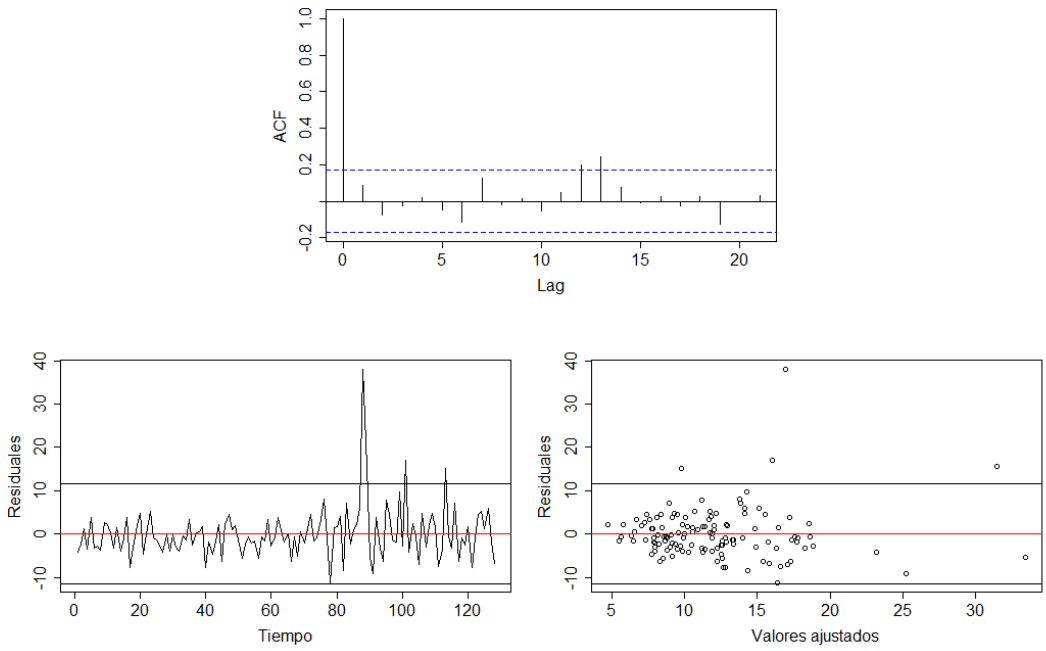

Figura 3-9.: Gráficos para el análisis de supuestos de los errores para el modelo log-lineal ajustado en **Python**

Aplicando la ecuación (3-49) se obtiene el MSE de ajuste para los diferentes modelos, presentados en la Tabla **3-4**

Tabla 3-4.: MSE de ajuste para los modelos trabajados.

	Modelo lineal	**Modelo log-lineal**
Implementación tscount	31.32525	31.23778
Implementación propia	30.66945	30.90294

Resultados que apuntan a que comparado con la función **R tsglm** de la librería **tscount** el módulo propuesto presenta un desempeño similar y que el ajuste parece estar bien implementado en el módulo propuesto en **Python**.

Para modelos más complejos, donde p, q toman valores superiores a 1, se puede aprovechar el algoritmo de selección automática previamente descrito en la sección 3.2.4 para la selección de los valores de p y q. Particularmente, sobre el conjunto de datos del número de casos de infecciones por campylobacter se obtienen $p = 8$ y $q = 13$, es decir, el modelo

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \text{con } \lambda^{(t)} = \exp\left(\nu^{(t)}\right), \text{ donde,}$$

$$\nu^{(t)} = d + \sum_{i=1}^{8} a_i \nu^{(t-i)} + \sum_{j=1}^{13} b_j \log(Y^{(t-j)} + 1),$$

y al ajustarlo con la función **tsglm** en el paquete **tscount** se obtiene un MSE de 23,7204, mientras que al ajustar utilizando la implementación propuesta en **Python** se obtiene un MSE de 16,1408. En la Figura **3-10** se presentan los gráficos del ajuste del modelo propuesto, donde se observa una mejor aproximación a la serie, aunque existen ciertas diferencias entre los ajustes obtenidos con la función **R tsglm** versus el módulo propuesto en **Python**.

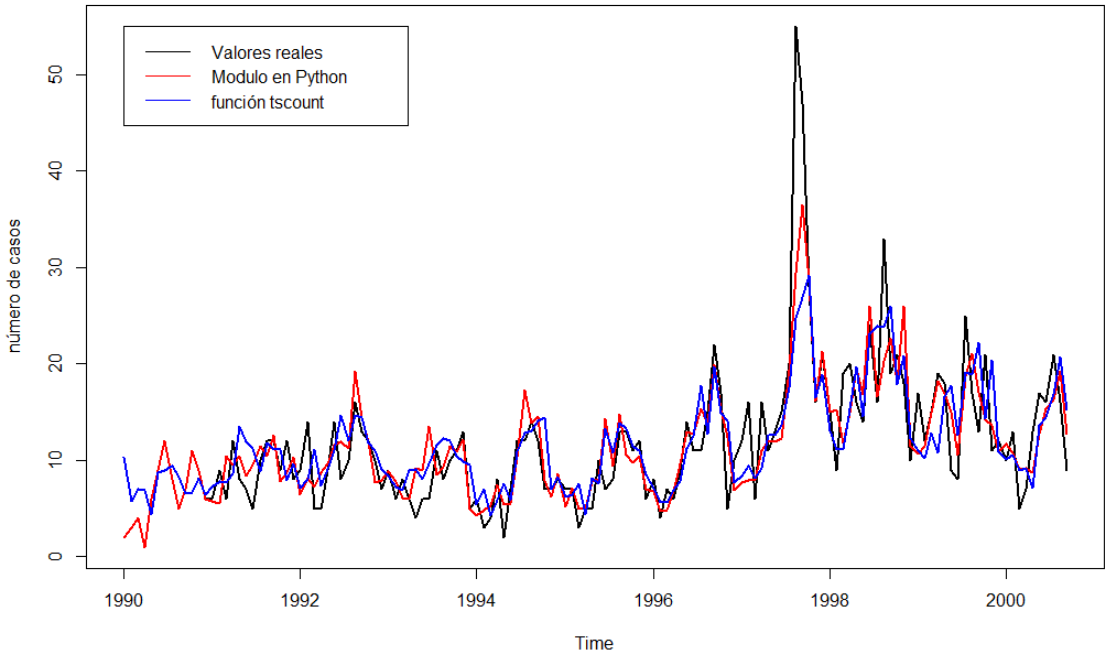

Figura 3-10.: Gráficos del ajuste del modelo estimado por validación cruzada usando el módulo creado en **Python** y la función del paquete **tscount**.

Para el modelo ajustado en **R**, en la Figura **3-11** se presentan los gráficos para el análisis de los supuestos de que los errores de ajuste son secuencias de variables aleatorias incorrelacionadas de media cero y varianza constante. En el gráfico de autocorrelación muestral, se observa que para ningún orden de autocorrelación, se encuentra evidencia de significancia estadística. Por otro lado, no existe patrones temporales en el gráfico de residuales contra el tiempo, sin embargo, se observa un corrimiento de la media de los residuos por debajo de cero para $t < 80$, es decir, todavía hay carencia de ajuste y mayor varianza después dé $t = 80$.

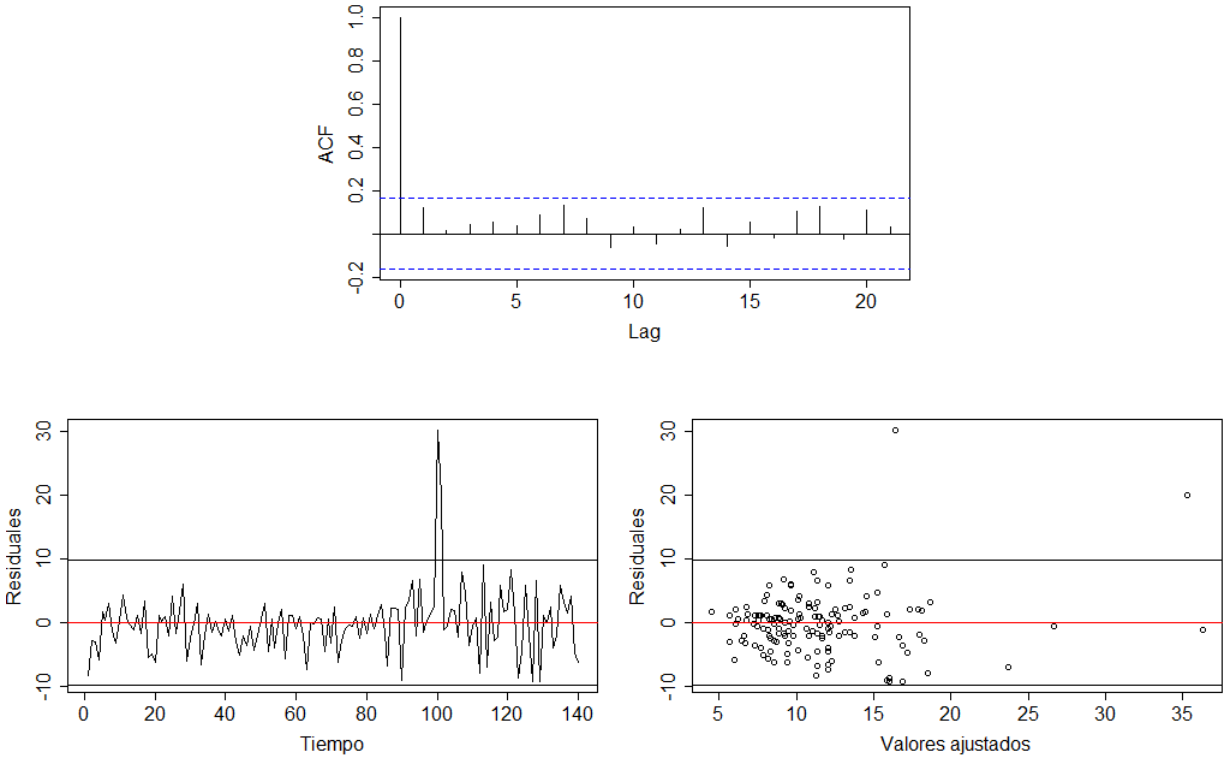

Figura 3-11.: Gráficos para el análisis de supuestos de los errores para el modelo obtenido por validación cruzada ajustado en **R**

Por otro lado, para el modelo ajustado en **Python** con el módulo programado, se presentan los gráficos observados en la Figura **3-12** para la validación de supuestos de que los errores de ajuste donde se observa en el gráfico de autocorrelación muestral que las autocorrelaciones según el rezago varían sin un patrón particular alrededor de cero y son estadísticamente no significativas, exceptuando en los rezagos de orden 7 y 13, sin embargo, en estos dos casos es posible que esté ocurriendo error tipo I a causa de la influencia de los residuos extremos que claramente son observados en las gráficas. Por otro lado, las conclusiones sobre los gráficos de residuos son similares a las ya anotadas respecto a los residuos del ajuste con la función **tsglm**. Cabe resaltar que los problemas de ajuste y la varianza no constante eran de esperar, dado el cambio de nivel y heterocedasticidad de la serie de tiempo.

Lo anterior apunta, nuevamente, a un desempeño similar en el ajuste entre el módulo propuesto en **Python** y la función **R tsglm** de la librería **tscount**.

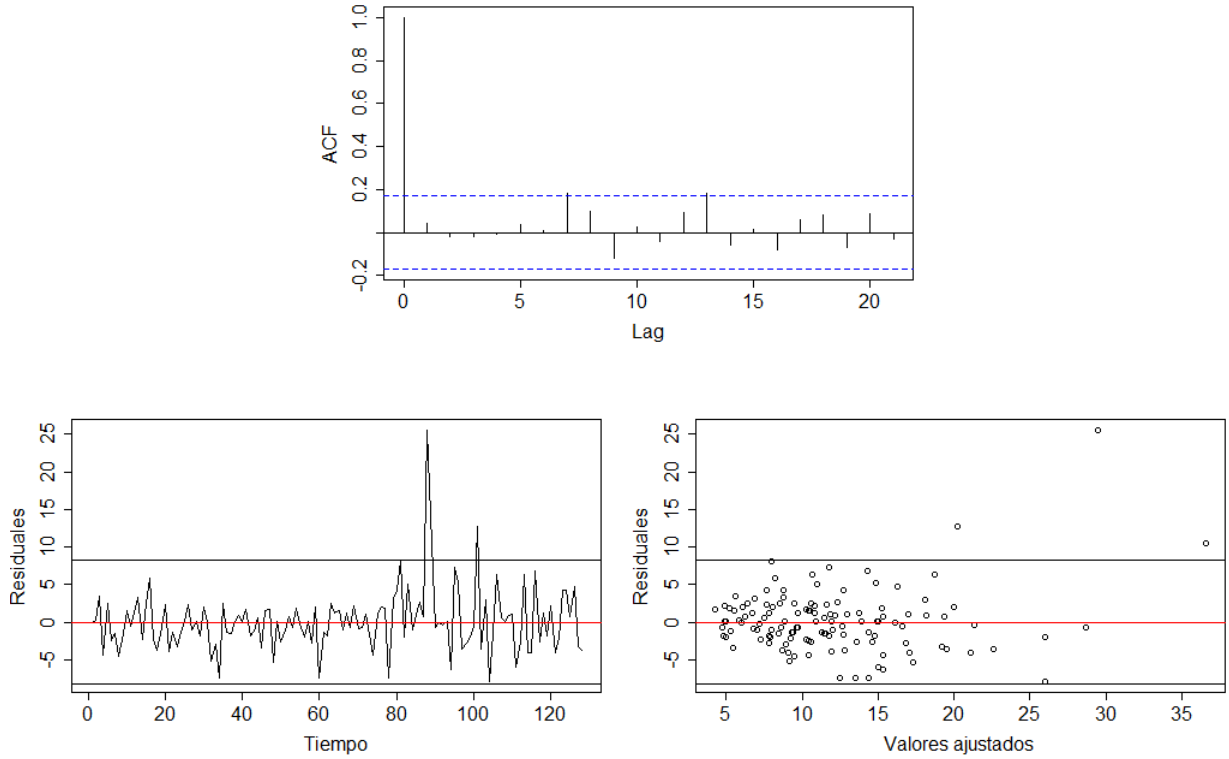

Figura 3-12.: Gráficos para el análisis de supuestos de los errores para el modelo obtenido por validación cruzada ajustado en **Python**

Caso 2: Muertes de conductores en Gran Bretaña

Este conjunto de datos correspondiente a la serie mensual del total de conductores muertos en Gran Bretaña entre enero de 1969 y diciembre de 1984 está incluido en la instalación básica del software estadístico **R** (R Core Team, 2023) como la variable "DriversKilled" en el conjunto de datos "Seatbelts". En este caso, denotamos:

$Y^{(t)}$: Total de conductores muertos en Gran Bretaña en un momento t.

En la Figura **3-13** se presenta la serie en el panel superior izquierdo, una estimación de su tendencia en el panel superior derecho, la estimación de la función periodograma sobre la serie diferenciada en el panel inferior izquierdo y la distribución de los valores de acuerdo al mes del año calendario, panel inferior derecho. De estas gráficas es claro que la serie observada presenta tendencia por su cambio de nivel en el intervalo de tiempo observado, y un comportamiento estacional por la repetición de un patrón anual en el cual la serie tiende en promedio a aumentar de septiembre a diciembre respeto a los niveles registrados de enero

a agosto. Además, el periodograma construido sobre la diferencia regular de orden 1 (para estabilizar el nivel), muestra que por lo menos en dos de las frecuencias Fourier de la forma $F_j = j/12$, con $j = 1, 2, 3, 4, 5, 6$, resaltadas con las líneas verticales de color rojo, existen componentes periódicas con las cuales la serie presenta asociación que puede ser significativa, más específicamente, en las frecuencias 1/12 (fenómeno estacional de periodo 12 meses) y 1/6 (fenómeno estacional de periodo 6 meses).

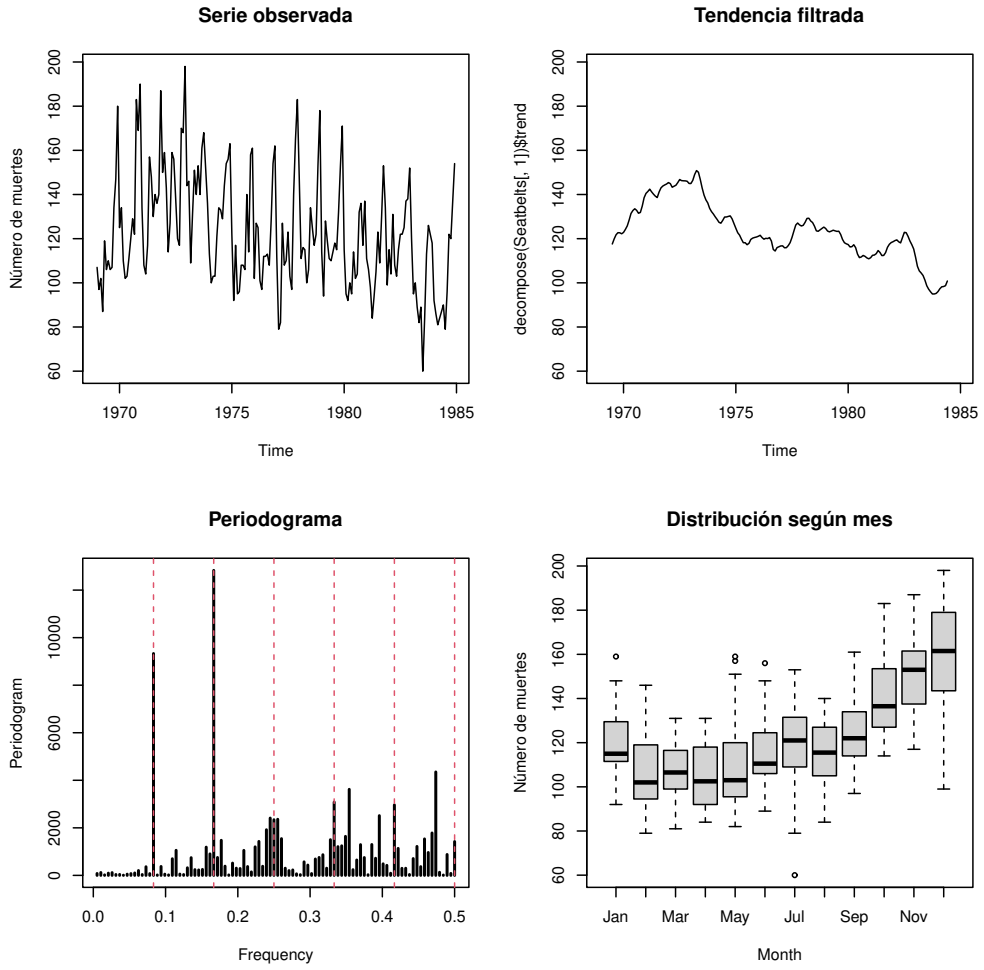

Figura 3-13.: Gráficos de la serie temporal y la estimación de la tendencia, el periodograma y la distribución de los valores de acuerdo al mes, para los datos del número conductores muertos.

Al seguir una estructura de modelación análoga al ejemplo anterior, inicialmente se considera el modelo lineal con rezagos de primer orden. Al ajustar con la función **tsglm** en el paquete **tscount** se obiene:

$$\widehat{Y}^{(t)} = 4{,}6289 + 5{,}5734x10^{-9}\widehat{\lambda}^{(t-1)} + 0{,}6246Y^{(t-1)},$$

y utilizando el módulo implementado en **Python**, la ecuación ajustada corresponde a

$$\widehat{Y}^{(t)} = 48{,}4732 - 0{,}1108\widehat{\lambda}^{(t-1)} + 0{,}7166Y^{(t-1)}.$$

En cuanto al modelo log-lineal, al estimar mediante la función **tsglm** en el paquete **tscount** se obtiene:

$$\widehat{Y}^{(t)} = \exp\left(\widehat{\nu}^{(t)}\right), \text{ con } \widehat{\nu}^{(t)} = 2{,}184 - 0{,}1641\widehat{\nu}^{(t-1)} + 0{,}7099\log(Y^{(t-1)} + 1),$$

y utilizando el módulo implementado en **Python**:

$$\widehat{Y}^{(t)} = \exp\left(\widehat{\nu}^{(t)}\right), \text{ con } \widehat{\nu}^{(t)} = 2{,}08 - 0{,}1431\widehat{\nu}^{(t-1)} + 0{,}7107\log(Y^{(t-1)} + 1)$$

En las Figuras **3-14** y **3-15** se puede observar la serie y sus diferentes ajustes mediante los modelos lineal y log-lineal con las rutinas de estimación de la función **tsglm** y el algoritmo desarrollado en **Python**. Al igual que se observó sobre los datos del ejemplo anterior, los ajustes con ambos modelos lineal y log lineal son muy cercanos uno de otro, pero con carencia de ajuste al presentar valores con cierto retardo respecto a la serie observada, además, no logran seguir los valores más altos y más bajos en cada año debido a que estos modelos no están captando los cambios de nivel que presenta la serie. Sin embargo, en cuanto al desempeño del módulo propuesto, nuevamente se observa que logra un desempeño muy similar en el ajuste al de la función de **tsglm**.

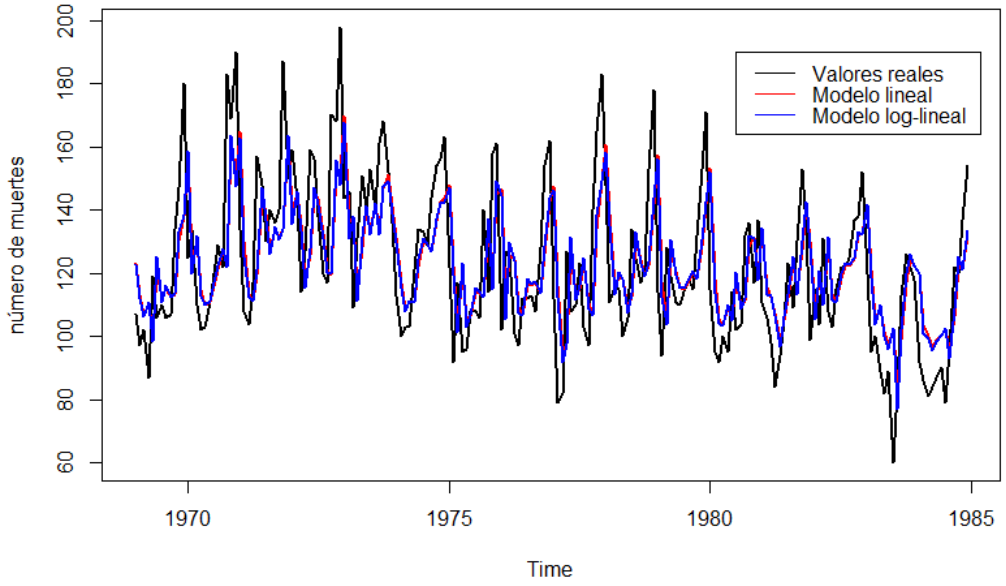

Figura 3-14.: Total de conductores muertos en Gran Bretaña y sus ajustes mediante modelos lineal y log-lineal con el paquete **tscount**.

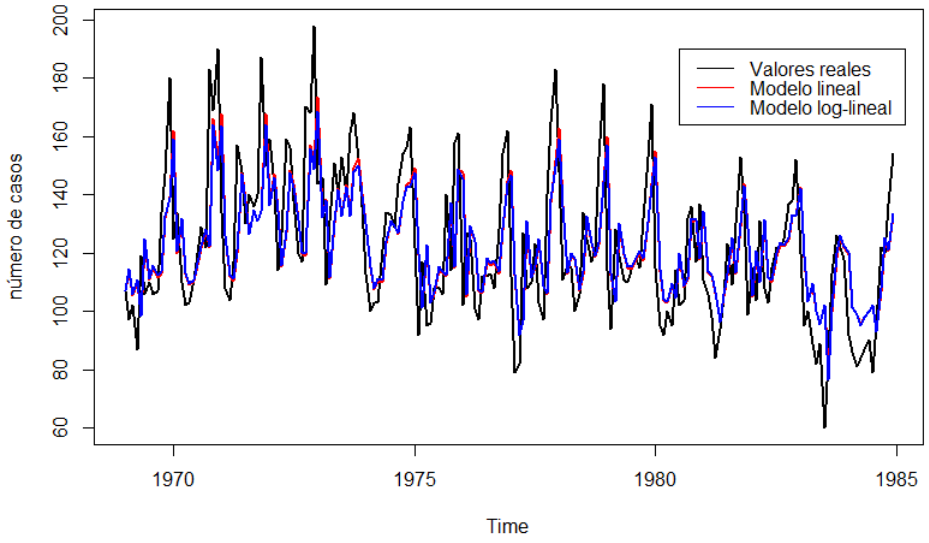

Figura 3-15.: Total de conductores muertos en Gran Bretaña y sus ajustes mediante modelos lineal y log-lineal con módulo implementado en **Python**.

En cuanto al análisis de supuestos sobre los errores de ajuste mediante los residuos, calculados según (3-48), para el ajuste del modelo lineal obtenido con el paquete **tscount** se presentan en la Figura **3-16** los gráficos de los residuos del ajuste versus el tiempo y versus los valores ajustados, así como la ACF obtenida a partir de estos. En estas gráficas se puede observar que existen autocorrelaciones de orden 11,12 y 13 significativas y a además en la serie de residuos se percibe patrón de tipo estacional y que parece existir una leve tendencia descendente en los residuos, consecuencia del cambio de nivel que el modelo no logra seguir. Por otro lado, no parece haber evidencia contra el supuesto de varianza constante

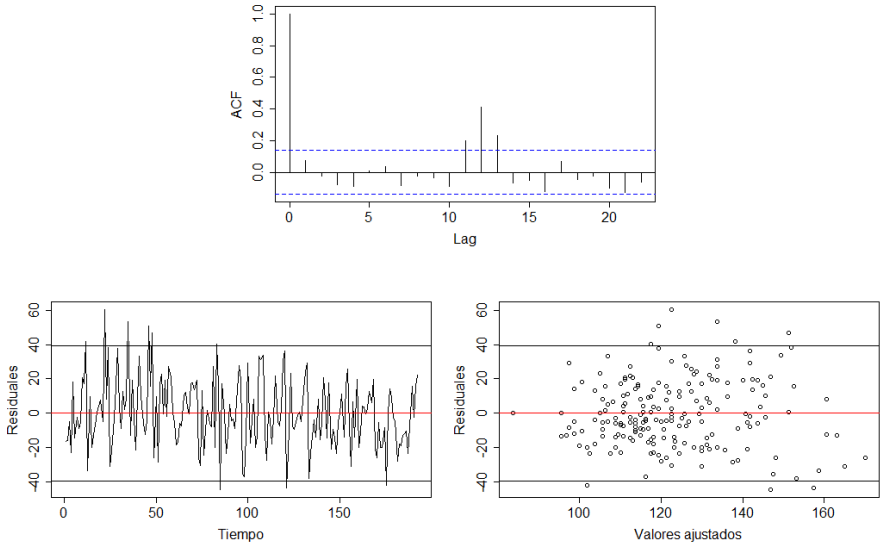

Figura 3-16.: Gráficos para el análisis de supuestos de los errores para el modelo lineal ajustado en **R**

Para el ajuste del modelo lineal obtenido con el módulo propuesto en **Python**, los gráficos para la validación de los supuestos de los errores se presentan en la Figura **3-17**, y, al igual que en el ajuste anterior, los residuos presentan un patrón estacional y correlaciones significativas de orden 11, 12 y 13, una leve tendencia descendente por el cambio de nivel de la serie y varianza constante.

Pasando al análisis del cumplimiento de supuestos de los errores del modelo log-lineal, se presentan los gráficos para la validación de los supuestos de los errores del modelo obtenido al ajustar con el paquete **tscount** en la Figura **3-18** y los del modelo obtenido al ajustar con el módulo propuesto en **Python** en la Figura **3-19**. Como puede observarse, los patrones en estas figuras son similares a los observados en los ajustes presentados con el modelo lineal, presentándose nuevamente correlaciones significativas de orden 11, 12 y 13 y un patrón estacional en la serie de los residuales, así como una leve tendencia descendente producto del cambio de nivel de la serie que no logran capturar los modelos ajustados, pero sin existir evidencia contra el supuesto de varianza constante.

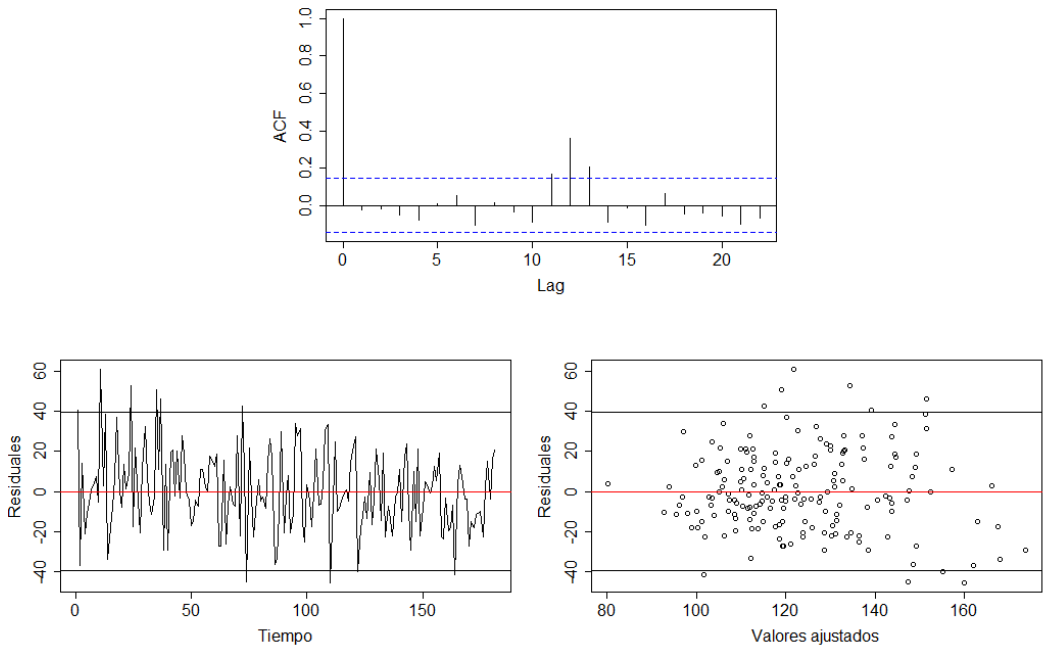

Figura 3-17.: Gráficos para el análisis de supuestos de los errores para el modelo lineal ajustado en **Python**

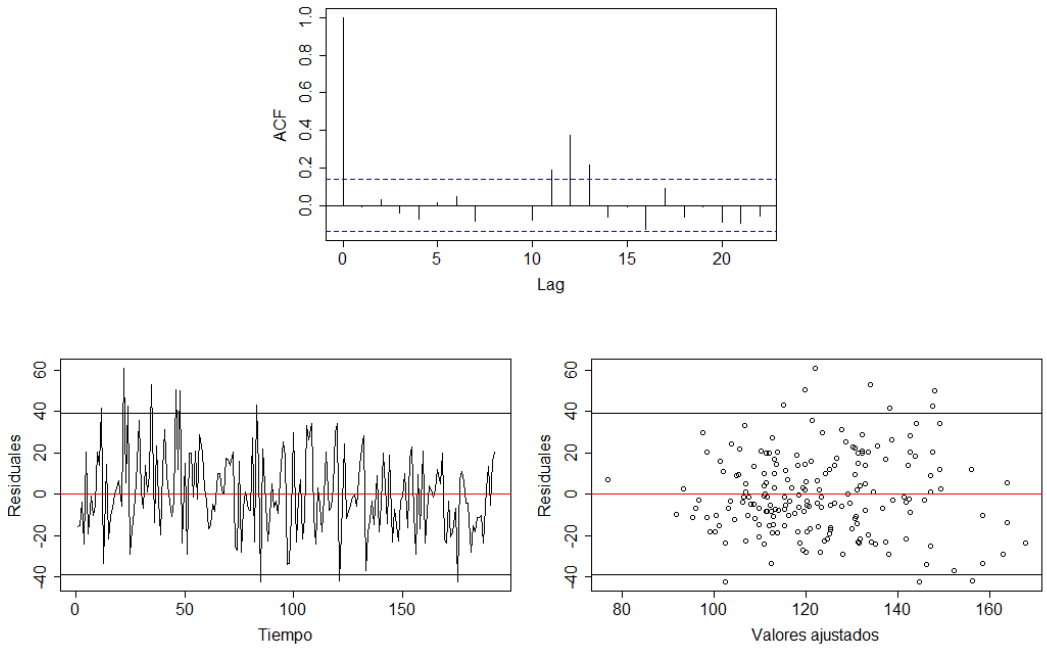

Figura 3-18.: Gráficos para el análisis de supuestos de los errores para el modelo log-lineal ajustado en **R**

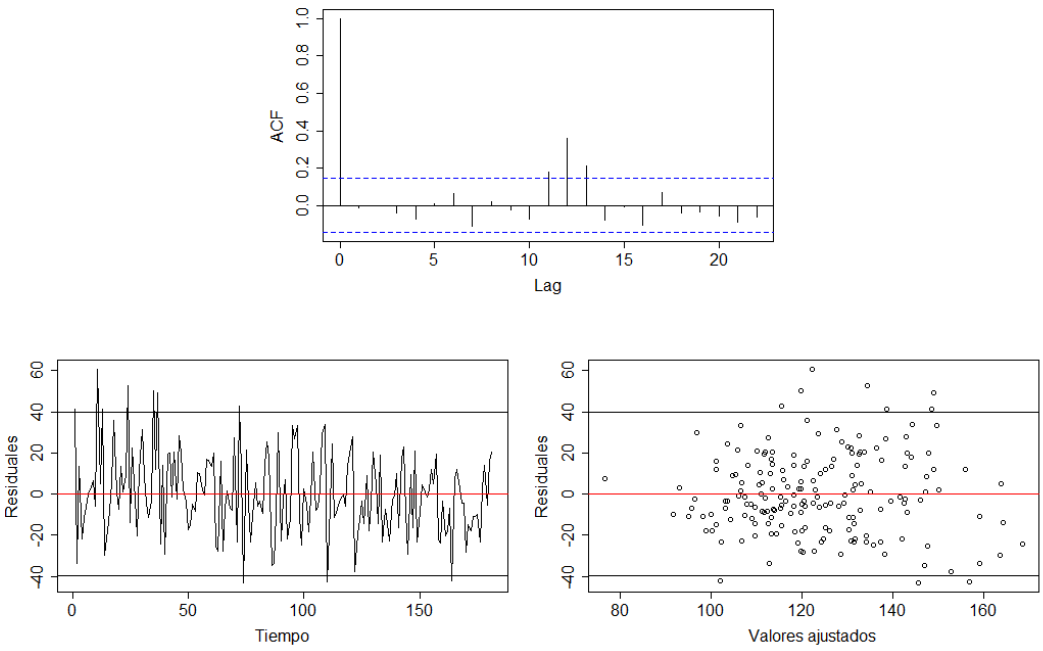

Figura 3-19.: Gráficos para el análisis de supuestos de los errores para el modelo log-lineal ajustado en **Python**

Una medida global de ajuste es calculada usando el error cuadrático medio de ajuste, presentado en (3-49), y se obtienen los resultados presentados en la Tabla **3-5**.

Tabla 3-5.: MSE de ajuste para los modelos trabajados.

	Modelo lineal	**Modelo log-lineal**
Implementación tscount	388.3806	380.3249
Implementación propia	384.4242	379.3494

Como se observa en la Tabla **3-5**, la calidad del ajuste de los modelos usando el algoritmo propuesto en **Python** presenta cifras de error un poco menores, particularmente en el ajuste del modelo lineal, que el ajuste obtenido con la función **tsglm** de la librería **tscount** de **R**. Teniendo en cuenta que en términos de los análisis sobre residuos, los resultados fueron los mismos para el módulo propuesto respecto a los resultados obtenidos con el paquete de referencia en **R**, se puede concluir que el desempeño del módulo propuesto es competitivo para la estimación de los modelos de primer orden, lo que en primera instancia apunta a que puede utilizarse esta rutina de ajuste en la posterior comparación de los modelos autorregresivos con los modelos que se implementarán más adelante en machine learning (ver Capítulo 6).

Usando el método de selección automática de órdenes de p y q para el modelo log-lineal, utilizando el módulo propuesto en **Python**, se obtuvieron $p = 1$ y $q = 12$ como los mejores valores para minimizar el CVMSE obtenido en la validación cruzada, definida previamente en la sección 3.2.4, de forma que la ecuación del modelo es:

$$Y^{(t)}|F^{(t-1)} \sim Poisson(\lambda^{(t)}), \quad \text{con } \lambda^{(t)} = \exp\left(\nu^{(t)}\right), \text{ donde,}$$

$$\nu^{(t)} = d + a_1\nu^{(t-1)} + \sum_{j=1}^{12} b_j \log(Y^{(t-j)} + 1).$$

Al ajustar el modelo con los órdenes seleccionados con la función **tsglm** en el paquete **tscount** se obtiene un MSE de ajuste de 277,5208, mientras que al ajustar utilizando la implementación propuesta en **Python** se obtiene un MSE de ajuste de 250,79. En la Figura **3-20** se presentan los gráficos del ajuste de los modelos ajustados, donde se puede apreciar que si bien los ajustes resultan mejores que con los modelos previamente ajustados, existen problemas en el ajuste dado el cambio de nivel con caída de la serie, que los ajustes no logran acompañar. Además, los valores más altos y más bajos observados no logran ser ajustados en general, precisamente porque la serie tiene una tendencia que los modelos no están considerando.

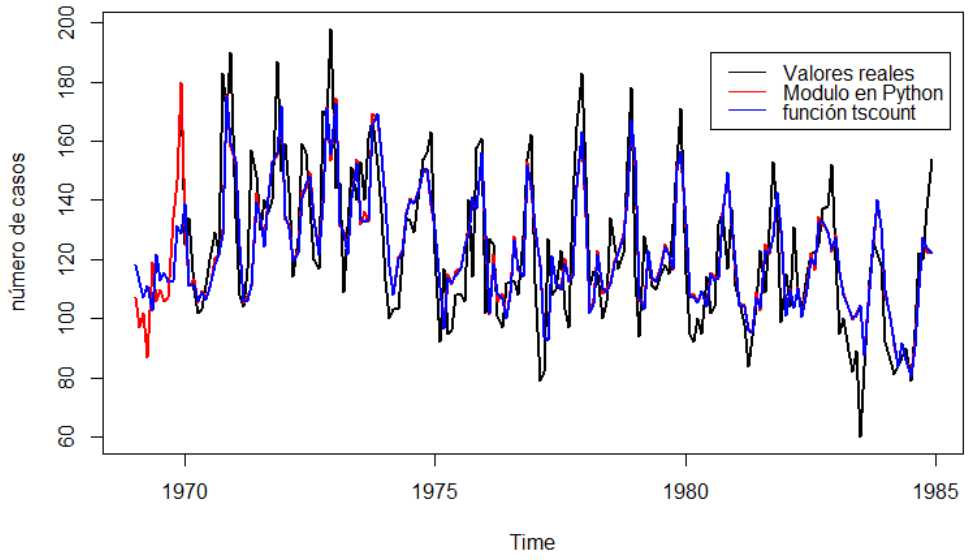

Figura 3-20.: Gráficos del ajuste del modelo estimado por validación cruzada usando el módulo creado en **Python** y la función del paquete **tscount**.

La Figura **3-21** exhibe los residuos del ajuste y la ACF estimada, para diagnosticar supuestos sobre los errores del modelo, cuando es ajustado con la función **tsglm** de la librería **tscount** y en la Figura **3-22** se presentan los mismos para el modelo ajustado usando el módulo propuesto en **Python**. Puede observarse que para ambos casos no se detectan autocorrelaciones significativas y, junto con las gráficas de residuales, se concluye que no hay evidencia fuerte en contra de los supuestos de independencia y varianza constante sobre los errores de ajuste del modelo resultante del algoritmo de selección automática implementando en el módulo propuesto en **Python**. Sin embargo, se observa una leve tendencia en los residuos versus el tiempo a causa del no ajuste en el cambio de nivel de la serie no explicados por el modelo propuesto.

Estos ejemplos ilustran el desempeño del módulo propuesto en **Python** para este trabajo para realizar el ajuste del modelo Autorregresivo Poisson, resultando incluso mejores las métricas de ajuste respecto al paquete de referencia **tscount** en **R**, particularmente cuando es mayor el número de parámetros en los modelos. El uso de módulo propuesto tiene, además, la ventaja de permitir seleccionar de forma automática los órdenes de autorregresión. Si bien, una comparación rigurosa que evalúe criterios de información u otras medidas de ajuste puede arrojar más información en torno al desempeño de la función **tsglm** en relación con el módulo propuesto, esto escapa al objetivo de este trabajo y con la revisión realizada se logró validar el funcionamiento del algoritmo propuesto para su uso en la comparación con otros modelos que serán explorados en el Capítulo 6 de este trabajo.

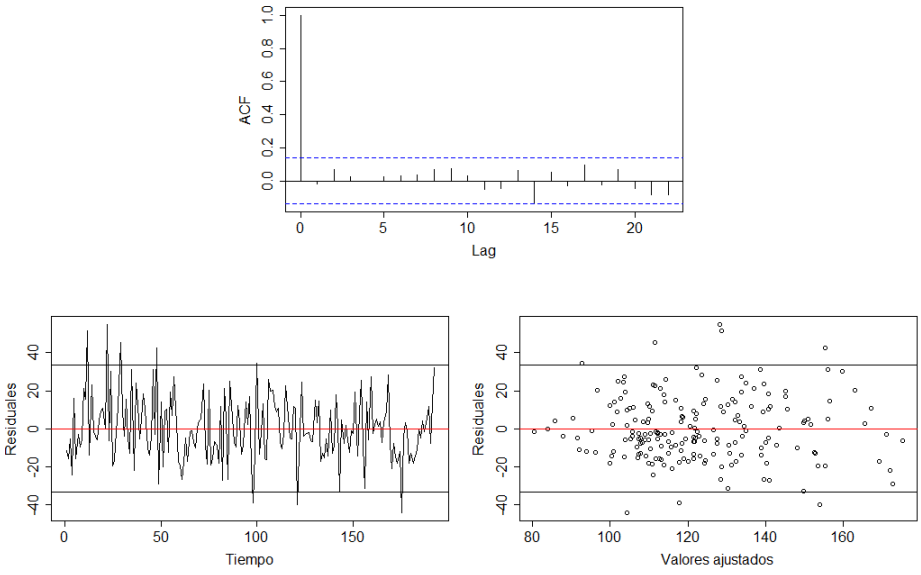

Figura 3-21.: Gráficos para el análisis de supuestos de los errores para el modelo obtenido por validación cruzada ajustado en **R**

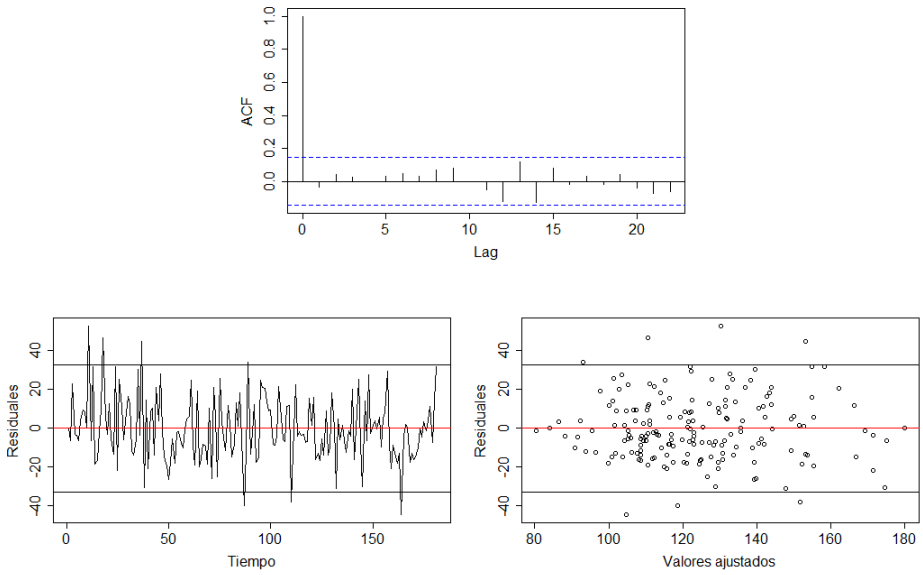

Figura 3-22.: Gráficos para el análisis de supuestos de los errores para el modelo obtenido por validación cruzada ajustado en **Python**

4. Pronóstico de series de tiempo con redes neuronales

De acuerdo con Zhang et al. (2021) el Machine Learning (o aprendizaje automático, como ha sido traducido al español en la literatura) es el estudio de los algoritmos que pueden aprender de la experiencia. A medida que los algoritmos acumulan experiencia, típicamente en la forma de datos observacionales o interacciones con el ambiente, su desempeño mejora. Particularmente el deep learning (DL), o aprendizaje profundo, es un campo particular del Machine Learning que está generando innovaciones en áreas tan diversas como la visión computacional, el procesamiento del lenguaje natural y los servicios de salud y está basado en el concepto de las redes neuronales, o neural networks (NN) por su nombre en inglés.

En su curso sobre redes neuronales Ng et al. (2023) describen una neurona como un operador simple que realiza un ajuste matemático y una red neuronal como múltiples neuronas apiladas. Los conceptos que preceden a estos modelos se remontan a los años de 1940s cuando la arquitectura temprana de la neurona computacional fue introducida por McCulloch y Pitts (1943). Sin embargo, el algoritmo de propagación hacia atrás (back propagation), que es clave en la forma actual del cómputo de las NN hoy en día, solo fue descrito hasta 1970 en una tesis de maestría y se utilizó en aplicaciones específicas de NN en 1981 (Schmidhuber, 2015). Para la entrada del nuevo milenio, las NN profundas finalmente atrajeron la atención al lograr rendimientos superiores a métodos alternativos de machine learning en múltiples aplicaciones importantes. De hecho, desde 2009, las NN supervisadas profundas han ganado múltiples competencias oficiales de reconocimiento de patrones (Schmidhuber, 2015).

Con el rápido crecimiento de los modelos de DL, el número de trabajos de investigación en el pronóstico de series de tiempo utilizando este tipo de modelos ha aumentado significativamente. Los modelos profundos han mostrado un excelente desempeño no solo en tareas de pronóstico, sino también en tareas de aprendizaje, donde representaciones abstractas pueden extraerse y transferirse a otras tareas, como lo son la clasificación y la detección de anomalías (Wen et al., 2023). Para el año 2017 las redes neuronales recurrentes (RNN), particularmente aquellas con nodos de memoria larga de corto plazo (LSTM por la sigla de su nombre en inglés long short-term memory) y nodos de unidad de compuerta recurrentes (GRU por la sigla de su nombre en inglés Gated Recurrent Unit) se habían establecido como el estado del arte para el modelamiento secuencial (Vaswani et al., 2017), sin embargo en Vaswani et al. (2017) se introduce la arquitectura de los Transformers, modelo de DL que ha alcanzado

gran éxito en múltiples aplicaciones como el procesamiento de lenguaje natural, la visión computacional, el reconocimiento del habla y más recientemente las series de tiempo, al beneficiarse de los mecanismos de atención que pueden aprender de manera automática las conexiones entre elementos en una secuencia (Nie et al., 2023).

Este trabajo parte desde los modelos de regresión, específicamente la regresión Poisson, como un caso particular del modelo de red neuronal. El modelo se presenta con los elementos clásicos de los modelos del deep learning, para facilitar la comprensión de los grafos computacionales y los algoritmos de propagación hacia adelante y hacia atrás. Se muestra también cómo los supuestos estadísticos sobre la variable respuesta pueden capturarse con la función de pérdida del modelo y que las variaciones del modelo (como la forma log-lineal), se capturan en la función de activación. A partir de lo anterior, se generaliza el modelo para llegar a la estructura matemática de las redes neuronales clásicas (Perceptrón multicapa). Cabe resaltar que la estructura de presentación del modelo de las redes neuronales presentada en este capítulo está inspirada en la presentada en Zhang et al. (2021) y Ng et al. (2023), sin embargo, en este trabajo se han adaptado los supuestos distribucionales para variables aleatorias discretas con distribución Poisson y se ha desarrollado nuevamente la presentación de conceptos a partir de este cambio.

Posteriormente, se pasa a los modelos enfocados en datos secuenciales y se toma como base los modelos estado-espaciales para pasar a la generalidad de las RNN con nodos LSTM. Luego, se presentan ejemplos de uso de estos últimos modelos, los cuales se compararán con el modelo autorregresivo presentado en el Capítulo 3 y el modelo basado en el Transformer que se presenta en el Capítulo 5. Si bien el modelo Transformer es una red neuronal, se explora por separado dada la complejidad de esta arquitectura.

4.1. Regresión Poisson con un enfoque de redes neuronales

Si bien los modelos de redes neuronales pueden contener estructuras computacionales complejas con un número de parámetros implicados en la modelación del orden de millones, la idea de su funcionamiento siempre es utilizar datos de entrada para ajustar un resultado. Una forma de facilitar el entendimiento de estos modelos es partir de la estructura más simple de la relación entre un conjunto de covariables o datos de entrada y una variable respuesta, esto es, el modelo de regresión. Particularmente, esta sección presenta el equivalente del modelo de regresión Poisson como una red neuronal, evidenciando que al ser las redes neuronales modelos del tipo no lineal generalizado, el caso lineal de la regresión es un caso particular.

Considere de nuevo el modelo de regresión Poisson en su forma log-lineal, presentado en el capítulo anterior en las ecuaciones (3-1) y (3-3). A continuación, se introducen algunos cambios en la notación para aproximar el problema desde la perspectiva de redes neuronales, utilizando la nomenclatura típica en deep learning. Como mencionan Zhang et al. (2021), en el caso de los modelos de regresión asumimos una relación entre unas variables independientes, dadas por el vector $\boldsymbol{x} \in \mathbb{R}^p = [X_1, X_2, \ldots, X_p]^T$, y un objetivo y, que permite expresar la esperanza condicional $E[Y|\boldsymbol{x}]$ como una suma ponderada de las variables en $\boldsymbol{x}$, siendo el objetivo del modelo el describir cómo transformar las variables independientes en una estimación de la variable respuesta, que denotamos $\hat{y}$. Si además utilizamos la notación de w (por la inicial de peso en inglés "weight") para los coeficientes de las variables independientes en la ecuación y la notación b (por la inicial de sesgo en inglés "biass") para el intercepto de la función de regresión, el valor ajustado por el modelo, dados un conjunto de variables independientes y de parámetros, está dado por:

$$\hat{y} = \hat{\lambda} = \exp\left(\sum_{j=1}^{p} w_j X_j + b\right). \tag{4-1}$$

Cabe resaltar que en este caso no se utiliza la notación de parámetros ajustados, $\widehat{w}_j, \widehat{b}$, ya que la ecuación de la estimación $\hat{y}$, de la variable respuesta, está definida para cualquier conjunto de parámetros, no solo para valores que se ajusten a unos supuestos paramétricos.

Si igualamos la componente sistémica a una variable z (que llamaremos función afín) y recolectando todas las variables independientes en el vector $\boldsymbol{x} \in \mathbb{R}^p = [X_1, X_2, \ldots, X_p]^T$ y todos los pesos en el vector $\boldsymbol{w} \in \mathbb{R}^p = [w_1, w_2, \ldots, w_p]^T$ tenemos:

$$z = \boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{w} + b. \tag{4-2}$$

Para expresar la información de entrada correspondiente al conjunto de n observaciones muestrales (o ejemplos como es común llamarlas en la bibliografía del deep learning) se utiliza la notación $\boldsymbol{A} \in \mathbb{R}^{n \times p}$ que hace alusión a "activación", concepto que se profundizará más adelante. De este modo, para una colección de variables independientes $\boldsymbol{A}$, el vector $\boldsymbol{z} \in \mathbb{R}^n = [z_1, z_2, \ldots z_n]^T$ que contiene el valor de la función afín para cada ejemplo muestral puede expresarse como:

$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{w} + b \tag{4-3}$$

Esta diferencia en la nomenclatura se introduce, ya que en la matriz de diseño $\boldsymbol{X} \in \mathbb{R}^{n \times (p+1)}$ introducida en el capítulo anterior, se incluye un vector de unos como primera columna,

asociada al intercepto, mientras que la matriz de datos de entrada $\boldsymbol{A}$ en los modelos del presente capítulo, solo contiene las observaciones de cada variable. De este modo:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1p} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{bmatrix} \neq \begin{bmatrix} x_{11} & \dots & x_{1p} \\ \vdots & \dots & \vdots \\ x_{n1} & \dots & x_{np} \end{bmatrix} = \boldsymbol{A}, \quad \boldsymbol{X} = [\mathbf{1}_n \ \boldsymbol{A}],$$

donde $\mathbf{1}_n \in \mathbb{R}^{n\times 1}$ es el vector de unos $[1, 1, \dots, 1]^T$.

En la ecuación (4-3) debe entenderse que la adición del escalar b corresponde a una notación simplificada de la operación elemento a elemento entre vectores (Parr & Howard, 2018) al sumar a cada elemento del vector $\boldsymbol{Aw}$ el sesgo b, de modo que con $\boldsymbol{b} = \mathbf{1_n}b$, por tanto (4-3) es equivalente a escribir $\boldsymbol{Aw} + \boldsymbol{b}$ (ver anexo B). Finalmente, se llega a la función log de verosimilitud especificada en (3-8).

Si hacemos $g(x) = \exp(x)$, que corresponde a la función que llamaremos "función de activación", el vector de las estimaciones de la variable respuesta puede escribirse así:

$$\hat{\boldsymbol{y}} = \boldsymbol{g}(\boldsymbol{z}) = \begin{bmatrix} g(z_1) \\ g(z_2) \\ \vdots \\ g(z_n) \end{bmatrix}. \tag{4-4}$$

Este proceso de hallar la estimación $\hat{\boldsymbol{y}}$, a partir del conjunto de datos de entrada y los valores de los parámetros, lo llamaremos "forward pass"(paso hacia adelante).

Por otro lado, la función de pérdida involucrada en el aprendizaje del modelo corresponde a aquella previamente definida en (3-10), pero en este caso, la notación que usaremos la expresa como una función del valor ajustado por el modelo y de la variable respuesta observada en cada ejemplo muestral, ya que dicha función mide el error entre estos dos valores,

$$c(\hat{y}_i, y_i) = \hat{y}_i - \log \hat{y}_i \times y_i. \tag{4-5}$$

Para medir la calidad del modelo sobre el conjunto de datos completo de n ejemplos muestrales, simplemente se promedian las pérdidas en el conjunto de entrenamiento en la que denominaremos la función de costo:

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n} c(\hat{y}_i, y_i) = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{y^{(i)}} - y^{(i)} \times \log(\hat{y}^{(i)})\right]. \tag{4-6}$$

Teniendo en cuenta que los valores de $\hat{\boldsymbol{y}}$ dependen de los parámetros $(\boldsymbol{w}, b)$ y los de $\boldsymbol{y}$ son observados, podemos escribir la función de costo en términos de los parámetros, es decir $J(\boldsymbol{w}, b)$, y el "entrenamiento" del modelo, entonces, se refiere a encontrar los valores $(\hat{\boldsymbol{w}}, \hat{b})$, como los valores de los parámetros $(\boldsymbol{w}, b)$ que minimicen la pérdida total en todos los ejemplos del conjunto de entrenamiento.

$$(\hat{\boldsymbol{w}}, \hat{b}) = \underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad J(\boldsymbol{w}, b).$$

Según Zeng et al. (2022), el descenso gradiente es la técnica utilizada para optimizar casi todos los modelos de deep learning y consiste en reducir de forma iterativa una métrica de error, dada por la función de pérdida, al actualizar los parámetros en la dirección en que disminuye dicha función de forma incremental (Ver anexo A). La aplicación utilizada en el capítulo anterior del descenso gradiente calcula el promedio de las pérdidas en todos los ejemplos del conjunto de datos y es la forma más simple del método. Sin embargo, en la práctica, esto puede resultar sumamente lento, pues implica pasar sobre todo el conjunto de datos antes de realizar cada actualización.

Otra alternativa a considerar es utilizar un único ejemplo muestral a la vez para realizar la actualización de los parámetros con el gradiente. En este caso es usado el algoritmo del descenso del gradiente estocástico (SGD por su sigla en inglés). Sin embargo, este método presenta algunos inconvenientes computacionales y estadísticos (Zhang et al., 2021).

Una solución a los problemas hallados en las dos alternativas anteriores, es seleccionar una estrategia intermedia: tomar mini-lotes (minibatch) de observaciones. Una buena estrategia empírica es la selección de un tamaño de lote en potencias de 2, entre 32 y 256 (Zhang et al., 2021). Esto conduce al descenso gradiente estocástico de minilotes (minibatch stochastic gradient descent o MSGD). En su forma más básica, en cada iteración t se selecciona aleatoriamente un lote K_t con un número fijo $|K|$ de ejemplos de entrenamiento. Luego, se calcula el gradiente de la pérdida promedio en el lote respecto a los parámetros del modelo. Estos valores de los gradientes se utilizan para actualizar los valores de los parámetros y completar un paso del descenso gradiente según la metodología que se esté utilizando (ver Anexo A).

Cómo se mencionó anteriormente, llamaremos paso hacia adelante (forward pass) al proceso de obtener las salidas del modelo y calcular la función de costo. Por otro lado, llamaremos paso hacia atrás (backward pass) al proceso de partir de la función de pérdida y calcular los gradientes respecto a los parámetros del modelo. Para ilustrar este proceso y la nomenclatura, en la Figura **4-1** se presenta el grafo computacional del modelo, que se complementará a medida que se incorporen elementos para el cálculo del barckward pass:

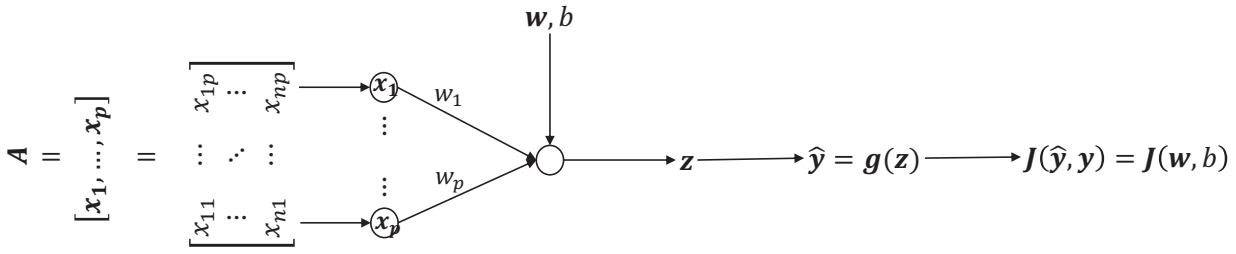

Figura 4-1.: Grafo computacional del modelo de regresión log-lineal Poisson. Fuente: Elaboración propia

Sean $c_{\hat{y}}(\hat{y}, y) = \frac{\partial c(\hat{y},y)}{\partial \hat{y}}$ y $\boldsymbol{c_{\hat{y}}(\hat{y}, y)}$ el operador elemento a elemento (ver Anexo B) dado por,

$$\mathbf{c}_{\hat{\mathbf{y}}}(\hat{\mathbf{y}}, \mathbf{y}) = \begin{bmatrix} c_{\hat{y}}(\hat{y_1}, y_1) & c_{\hat{y}}(\hat{y_2}, y_2) & \dots & c_{\hat{y}}(\hat{y_n}, y_n) \end{bmatrix}^T \tag{4-7}$$

y $\mathbf{g}'(\mathbf{z})$ a la operación elemento a elemento (ver Anexo B):

$$\mathbf{g}'(\mathbf{z}) = \begin{bmatrix} g'(z_1) & g'(z_2) & \dots & g'(z_n) \end{bmatrix}^T, \tag{4-8}$$

entonces las derivadas parciales para obtener el gradiente de la función de costo con respecto a los parámetros, cuyo cálculo se presenta en el Anexo C, están dadas por las ecuaciones (4-9) y (4-10) donde $\odot$ denota el producto uno a uno entre vectores, también llamado producto Hadamard (ver Anexo B):

$$\frac{\partial J}{\partial \boldsymbol{w}} = \boldsymbol{A^T} \frac{1}{n} \boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)} \tag{4-9}$$

$$\frac{\partial J}{\partial b} = sum\left(\frac{1}{n} \boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)} \right), \tag{4-10}$$

donde $sum(\boldsymbol{u})$ representa la suma de los valores componentes de su argumento vectorial $\boldsymbol{u}$.

Particularmente, en el caso de la regresión Poisson log-lineal, tenemos que:

$g(z) = \exp(z) \longrightarrow g'(z) = \exp(z)$,

luego,

$$\boldsymbol{g'(z)} = \begin{bmatrix}\exp(z_1) & \exp(z_2) & \dots & \exp(z_n)\end{bmatrix}^T$$

y con

$$c(\hat{y}, y) = \hat{y} - y\log(\hat{y}) \longrightarrow c_{\hat{y}}(\hat{y}, y) = 1 - \frac{y}{\hat{y}}$$

se sigue que,

$$\boldsymbol{c_{\hat{y}}(\hat{y}, y)} = \begin{bmatrix}\left(1 - \frac{y_1}{\widehat{y}_1}\right) & \left(1 - \frac{y_2}{\widehat{y}_2}\right) & \dots & \left(1 - \frac{y_n}{\widehat{y}_n}\right)\end{bmatrix}^T,$$

y reemplazando en (4-9) y (4-10), se obtiene,

$$\frac{\partial J}{\partial \boldsymbol{w}} = \boldsymbol{A^T}\left[\frac{1}{n}\boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)}\right] = \frac{1}{n}\begin{bmatrix} x_{11} & x_{21} & \dots & x_{n1} \\ x_{12} & x_{22} & \dots & x_{n2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1p} & x_{2p} & \dots & x_{np}\end{bmatrix}\begin{bmatrix}\exp(z_1)(1 - \frac{y_1}{\widehat{y}_1}) \\ \exp(z_2)(1 - \frac{y_2}{\widehat{y}_2}) \\ \vdots \\ \exp(z_n)(1 - \frac{y_n}{\widehat{y}_n})\end{bmatrix},$$

$$\frac{\partial J}{\partial b} = sum\left(\frac{1}{n}\boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)}\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\exp(z_i)(1 - \frac{y_i}{\widehat{y}_i})\right).$$

Puede evidenciarse en el grafo computacional de la Figura **4-2** que este proceso de cálculo de las derivadas con respecto a los parámetros corresponde a "propagar" hacia atrás los valores desde el cálculo de la función de pérdida hasta los gradientes de los parámetros.

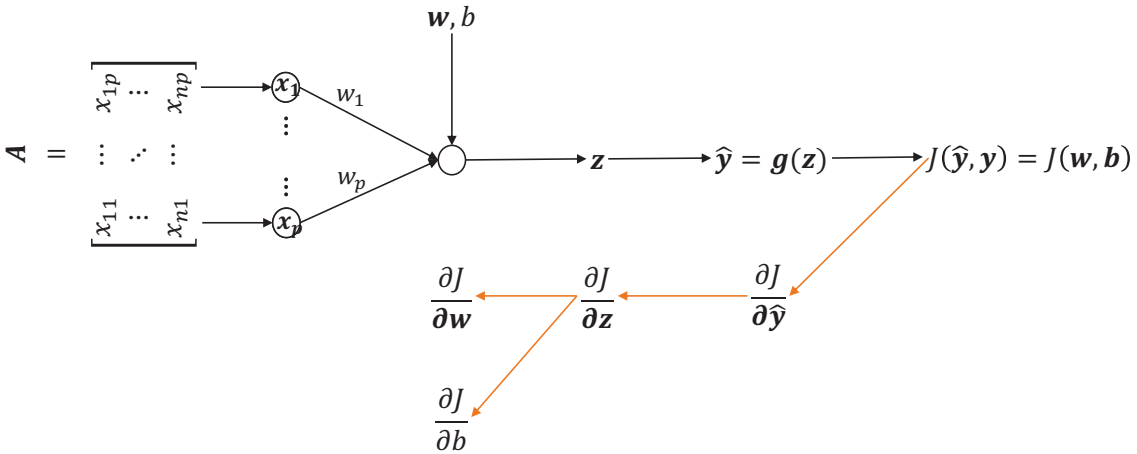

Figura 4-2.: Grafo computacional del modelo de regresión log-lineal Poisson con propagación hacia atrás. Fuente: Elaboración propia

Finalmente, el ajuste del modelo de regresión Poisson como red neuronal requiere aplicar estos pasos para el cálculo de los gradientes y la actualización de los parámetros para cada subconjunto de datos $\boldsymbol{A}^{\{k\}}$, correspondiente al lote, en lugar de todo el conjunto de datos $\boldsymbol{A}$, para la aplicación del MSGD previamente descrito. Así, si se toman lotes de m ejemplos muestrales seleccionados aleatoriamente del conjunto $\boldsymbol{A}$, se tendrán en total $K = \lceil \frac{n}{m} \rceil$ lotes. La completación un paso por todo el conjunto de datos, es decir, los K lotes, realizando actualizaciones de los parámetros mediante descenso del gradiente, se le conoce como una época.

El algoritmo para el ajuste del modelo con enfoque de redes neuronales es un proceso con dos ciclos. En primer lugar, se repite la actualización de los parámetros para un determinado número de épocas, e internamente, se repite para cada uno de los lotes en el conjunto de datos. Así, el algoritmo con E épocas es:

Inicializar $\boldsymbol{w}, b$

Repetir E veces:

para k de 1 a K:

$\boldsymbol{z}^{\{k\}} = \boldsymbol{A}^{\{k\}}\boldsymbol{w} + b$

$\hat{\boldsymbol{y}}^{\{k\}} = \boldsymbol{g}(\boldsymbol{z}^{\{k\}})$

$$\frac{\partial J}{\partial \boldsymbol{z}^{\{k\}}} = \tfrac{1}{n}\boldsymbol{c}_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \odot \boldsymbol{g}'(\boldsymbol{z}) = \tfrac{1}{n}\begin{bmatrix} \exp(z_1)(1 - \frac{y_1}{\hat{y}_1}) \\ \exp(z_2)(1 - \frac{y_2}{\hat{y}_2}) \\ \vdots \\ \exp(z_n)(1 - \frac{y_n}{\hat{y}_n}) \end{bmatrix}$$

$$\frac{\partial J}{\partial \boldsymbol{w}} = (\boldsymbol{A}^{\{k\}})^T \frac{\partial J}{\partial \boldsymbol{z}^{\{k\}}}$$

$$\frac{\partial J}{\partial b} = \mathbf{1}_n^T \frac{\partial J}{\partial \boldsymbol{z}^{\{k\}}}$$

actualizar w, b usando $\dfrac{\partial J}{\partial \boldsymbol{w}}, \dfrac{\partial J}{\partial b}$ según algoritmo de descenso gradiente usado.

Como referente computacional, en el repositorio que contiene las implementaciones de código de este trabajo, el cual es accesible en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast, se encuentra el código de la implementación del modelo de regresión Poisson como red neuronal utilizando la librería Tensorflow (Martín Abadi et al., 2015). Esta implementación está consignada en el notebook "poisson_regression_as_n" y muestra los resultados de la estimación de parámetros de la regresión cuando se aplica como red neuronal, los cuales son comparados con las estimaciones obtenidas en el Capítulo previo.

Los datos simulados corresponden al modelo de regresión Poisson log-lineal presentado en la Sección 3.1, ecuación (3-5), con $\boldsymbol{\beta} = [1, 2, 3]^T$ y $\boldsymbol{x} = [1, X_1, X_2]$ y la matriz de diseño es tal como se definió en (3-13). Los valores de X_1 se generan de una distribución uniforme en el intervalo $[0, 1)$ y los valores de X_2 de una normal estándar.

Tal y como se presentó en el experimento del capítulo anterior, se generan 1000 simulaciones de muestras de tamaño $n = 1000$ y se obtienen los valores ajustados con las librerías **scikit-learn** (Pedregosa et al., 2011) y **statsmodels** (Seabold & Perktold, 2010) en **Python**, y con la implementación propia para la regresión Poisson. Luego, se obtienen los valores de los parámetros estimados utilizando una red neuronal. La Tabla **4-1** permite comparar los valores promedios de las estimaciones de los parámetros obtenidos con los cuatro algoritmos:

Tabla 4-1.: Valores promedio de las estimaciones de los parámetros con los diferentes algoritmos

Parámetro	**Scikit learn**	**Stats models**	**Módulo propio**	**Red neuronal**
β_0	1.0833	1.0002	0.9970	1.0148
β_1	1.9303	1.9999	2.0004	1.9896
β_2	2.9830	2.9999	3.0014	2.9970

Estos resultados evidencian que los valores medios, estimados por la red neuronal, en efecto convergen a los valores reales de los parámetros y que el uso de este algoritmo se desempeña tan bien como los paquetes de referencia en **Python** para la regresión Poisson.

Por otro lado, al calcular los errores promedio de estimación para los cuatro algoritmos utilizando (3-14), se obtienen los resultados presentados en la Tabla **4-2**, y al calcular las desviaciones de los errores de estimación utilizando (3-15) se obtienen los resultados presentados en la Tabla **4-3**

Tabla 4-2.: Valores promedio de los errores de estimación

Parámetro	**Scikit learn**	**Stats models**	**Módulo propio**	**Red neuronal**
β_0	0.0833	0.0002	-0.0030	0.0148
β_1	-0.0697	-0.0001	0.0004	-0.0104
β_2	-0.0170	-0.0001	0.0014	-0.0029

Tabla 4-3.: Desviaciones estándar de los errores de estimación

Parámetro	Scikit learn	Stats models	Módulo propio	Red neuronal
β_0	0.0365	0.0070	0.0106	0.0788
β_1	0.0285	0.0057	0.0098	0.0698
β_2	0.0124	0.0023	0.0047	0.0149

Se observa que las estimaciones de los parámetros con la red neuronal tuvieron un error promedio mayor y más variabilidad que el módulo propio implementado para la regresión Poisson y Stats models para el número de épocas con las que se ajustó (300). Si observamos los histogramas de los errores de estimación en la Figura **4-3** se confirma este comportamiento.

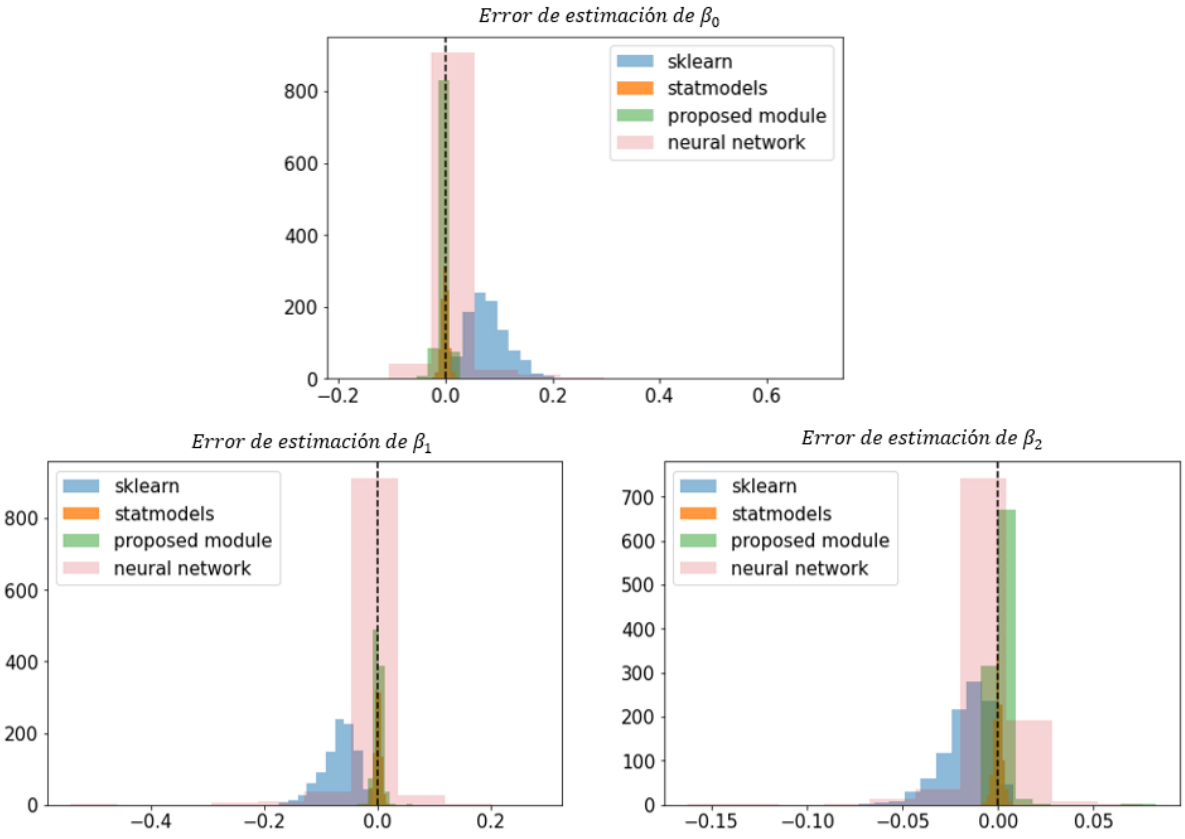

Figura 4-3.: Histogramas de los errores de estimación en las simulaciones para los parámetros β_0, β_1 y β_2

De lo anterior se concluye que, si bien la red neuronal tiene un sesgo mayor en la estimación de los parámetros y una mayor dispersión que la presentada por los ajustes con la librería **statsmodels** y el módulo propuesto en este trabajo en **Python** para la regresión Poisson, el sesgo obtenido no es muy grande, como tampoco parece serlo la variabilidad de las estimaciones, y el ajuste tiene una calidad aceptable. Sin embargo, la complejidad computacional resulta muy elevada y el desempeño, aunque similar, es menor al de otras librerías, por lo que no resulta recomendable como alternativa a la regresión. Sin embargo, el modelo de redes neuronales permite generalizaciones sobre la relación entre las covariables y la variable respuesta lo cual justifica su uso en casos más complejos y para los cuales se introduce la estructura de la siguiente sección.

4.2. El perceptrón multicapa

El perceptrón multicapa es el nombre que recibe la red neuronal profunda con la arquitectura más simple (Zhang et al., 2021) y consiste de múltiples capas de neuronas o nodos completamente conectados con aquellos de la capa anterior y la capa siguiente. Para ilustrar este concepto sé inicia retomando el grafo computacional presentado para el modelo de regresión Poisson con algunos ajustes notacionales para la generalización del modelo, tal como se observa en la Figura **4-4**.

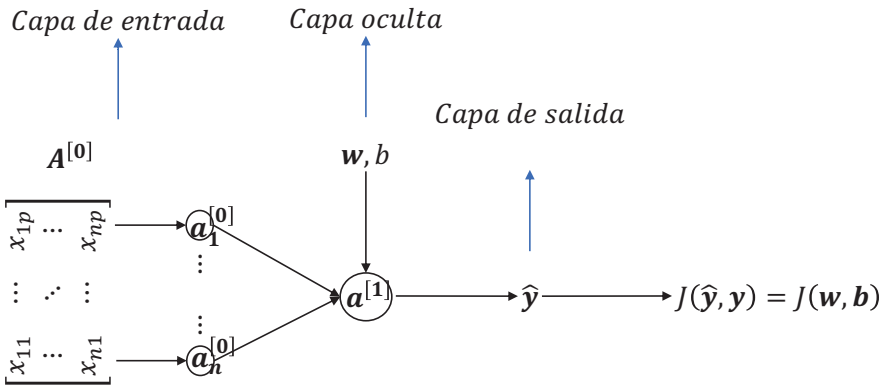

Figura 4-4.: Grafo computacional del modelo de regresión log-lineal Poisson. Fuente: Elaboración propia

En este caso comenzamos a utilizar la notación de capas, en donde la capa de entrada se refiere al ingreso de los valores observados en la red y la capa de salida a la estimación del modelo. Las capas ocultas corresponden a apilar múltiples neuronas o nodos, donde un nodo es la unidad computacional que recibe como insumo las salidas de la capa anterior y calcula un valor de salida que llamaremos activación (Zhang et al., 2021).

En el caso del grafo en la Figura **4-4**, la capa oculta tiene una única neurona que realiza el cálculo,

$$\boldsymbol{a}^{[1]} = \boldsymbol{g}^{[1]}\left(\boldsymbol{A}^{[\boldsymbol{0}]}\boldsymbol{w} + b\right), \tag{4-11}$$

donde, se tiene que $\boldsymbol{g}^{[\boldsymbol{l}]}(\boldsymbol{z})$ es la operación elemento a elemento

$$\boldsymbol{g}^{[\boldsymbol{l}]}(\boldsymbol{z}) = \begin{bmatrix} g^{[l]}(z_1) \\ g^{[l]}(z_2) \\ \vdots \\ g^{[l]}(z_n) \end{bmatrix}, \tag{4-12}$$

teniendo en cuenta que en este caso $g^{[1]}$ hace referencia a la función de activación de la primera capa oculta, que para el caso del modelo de regresión Poisson log-lineal es $g^{[1]}(x) = \exp(x)$. El concepto de funciones de activación se refiere a operadores diferenciables (o cuya derivada pueda hallarse por tramos) que transforman entradas en salidas, añadiendo no-linealidad, de modo que se evite que el resultado de la red sea equivalente a una única función lineal (Zhang et al., 2021), ya que aplicar transformaciones lineales consecutivamente es equivalente a una única transformación lineal. Algunas funciones de activación comúnmente utilizadas en deep learning son:

- ReLU: Su nombre proviene de *rectified linear unit* y provee una transformación no lineal simple dada por:

$$f(x) = \begin{cases} x & si \quad \geq 0 \\ 0 & si \quad x < 0 \end{cases} = \text{máx}(x, 0). \tag{4-13}$$

- Función sigmoide: Transforma entradas en el dominio de los reales a salidas en el intervalo $(0, 1)$ y está dada por:

$$f(x) = \frac{1}{1 - e^{-x}}. \tag{4-14}$$

- Función Tanh: la función tangente hiperbólica, transforma las entradas de los reales al intervalo $(-1, 1)$ y está dada por:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{4-15}$$

De forma general, podemos definir el cálculo de una activación para una neurona en la l-ésima capa oculta para todos los ejemplos muestrales (con tamaño de lote $m = n$ para el caso en que se utiliza como entrada el conjunto muestral completo, tal como se vio en el modelo de regresión como red neuronal) como:

$$\boldsymbol{a}^{[\boldsymbol{l}]} = \boldsymbol{g}^{[l]}(\boldsymbol{A}^{[\boldsymbol{l-1}]}\boldsymbol{w}^{[l]} + b^{[l]}), \tag{4-16}$$

que por facilidad computacional, suele introducirse una variable intermedia $\boldsymbol{z}^{[l]}$, llamada función afín, de modo que el cómputo se realiza como:

$$\boldsymbol{z}^{[\boldsymbol{l}]} = \boldsymbol{A}^{[l-1]}\boldsymbol{w}^{[l]} + b^{[l]}, \quad \text{luego } \boldsymbol{a}^{[\boldsymbol{l}]} = \boldsymbol{g}^{[l]}(\boldsymbol{z}^{[\boldsymbol{l}]}), \tag{4-17}$$

y $\boldsymbol{A}^{[l-1]}$ corresponde a la matriz de activaciones de la capa anterior. La definición de la matriz de activaciones para una determinada capa está dada por:

$$\boldsymbol{A}^{[\boldsymbol{l}]} = \begin{bmatrix} \boldsymbol{a}_{\boldsymbol{1}}^{[\boldsymbol{l}]} & \boldsymbol{a}_{\boldsymbol{2}}^{[\boldsymbol{l}]} & \dots & \boldsymbol{a}_{\boldsymbol{n}}^{[\boldsymbol{l}]} \end{bmatrix} = \begin{bmatrix} a_{11}^{[l]} & a_{12}^{[l]} & \dots & a_{1n_l}^{[l]} \\ a_{21}^{[l]} & a_{22}^{[l]} & \dots & a_{2n_l}^{[l]} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1}^{[l]} & a_{n2}^{[l]} & \dots & a_{nn_l}^{[l]} \end{bmatrix}, \tag{4-18}$$

donde $\boldsymbol{a}_{\boldsymbol{j}}^{[\boldsymbol{l}]}$ corresponde al vector de las n observaciones muestrales de la activación de la j-ésima neurona en la l-ésima capa. Para el cálculo de cada activación se utilizan como parámetros el vector de pesos $\boldsymbol{w}_{\boldsymbol{j}}^{[l]}$ y el sesgo $b^{[l]}$, donde cada vector de pesos contiene el peso que se asigna en la j-ésima neurona de la l-ésima capa, a cada una de las $n-1$ activaciones de la $(l-1)$-ésima capa, luego:

$$\boldsymbol{w}_{\boldsymbol{j}}^{[l]} = \begin{bmatrix} w_{11} \\ w_{21} \\ \vdots \\ w_{n_{l-1}1} \end{bmatrix}; \tag{4-19}$$

y definiendo la matriz $\boldsymbol{W}^{[\boldsymbol{l}]}$ como la matriz de todos los vectores de pesos de las n_l neuronas de la l-ésima capa y $\boldsymbol{b}^{[\boldsymbol{l}]}$ como el vector fila de los sesgos, tenemos:

$$\boldsymbol{W}^{[\boldsymbol{l}]} = \begin{bmatrix} \boldsymbol{w}_{\boldsymbol{1}}^{[l]} & \boldsymbol{w}_{\boldsymbol{2}}^{[l]} & \dots & \boldsymbol{w}_{\boldsymbol{n_l}}^{[l]} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n_l} \\ w_{21} & w_{22} & \dots & w_{2n_l} \\ \vdots & \vdots & \dots & \vdots \\ w_{n_{l-1}1} & w_{n_{l-1}2} & \dots & w_{n_{l-1}n_l} \end{bmatrix} \tag{4-20}$$

$$\boldsymbol{b}^{[l]} = \begin{bmatrix} b_1^{[l]} & b_2^{[l]} & \dots & b_{n_l}^{[l]} \end{bmatrix}. \tag{4-21}$$

Luego, podemos expresar el paso hacia adelante (Zhang et al., 2021), es decir, el cálculo de la activación de la l-ésima, capa como:

$$\boldsymbol{Z}^{[l]} = \boldsymbol{A}^{[l-1]}\boldsymbol{W}^{[l]} + \mathbf{1}_{\boldsymbol{n}}\boldsymbol{b}^{[l]}, \text{ donde } \mathbf{1}_{\boldsymbol{n}} \in \mathbf{R}^{n\times 1}, \tag{4-22}$$

donde,

$$\boldsymbol{A}^{[l]} = \boldsymbol{g}^{[l]}(\boldsymbol{Z}^{[l]}), \tag{4-23}$$

con $\boldsymbol{g}^{[l]}(\boldsymbol{Z}^{[l]})$ la matriz resultante de aplicar la función de activación $g^{[l]}$ a cada uno de los elementos de la matriz $\boldsymbol{Z}^{[l]}$, conservando la dimensión de esta última.

A partir de estas ecuaciones se puede realizar el proceso de *forward propagation* (o propagación hacia adelante de la red neuronal) cuyo grafo computacional se presenta en la Figura **4-5**:

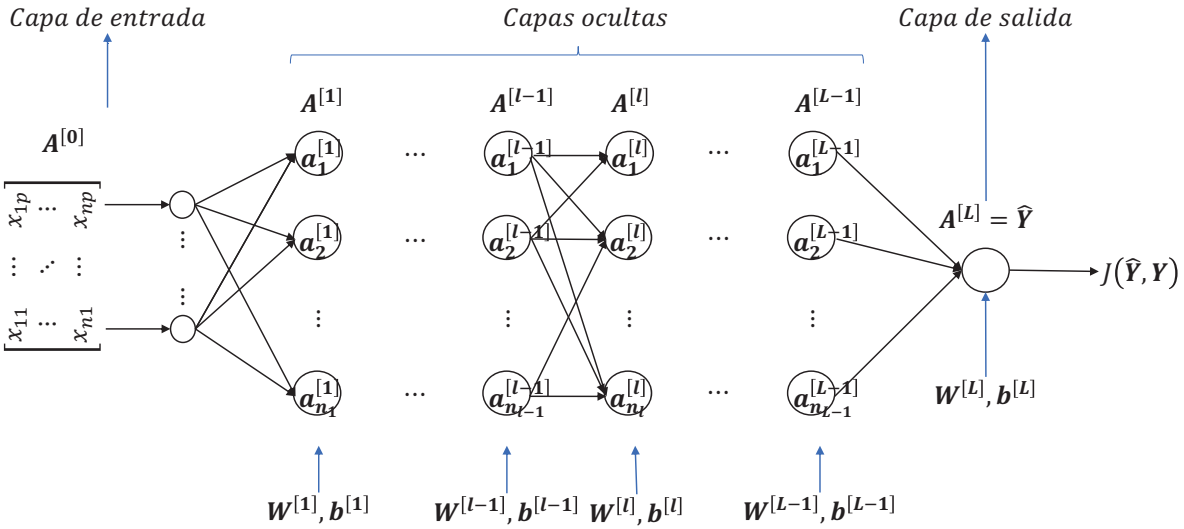

Figura 4-5.: Grafo computacional del perceptrón multicapa. Fuente: Elaboración propia

La propagación hacia adelante permite obtener las matrices $\boldsymbol{Z}^{[l]}$ de variables intermedias y de activaciones $\boldsymbol{A}^{[l]}$ para todas las capas, así como el valor estimado por el modelo $\hat{\boldsymbol{Y}} \in \mathbf{R}^{n\times n_y}$ (de forma general para una salida multivariada o $\hat{\boldsymbol{y}} \in \mathbf{R}^{n\times 1}$ en el caso univariado) el cual es equivalente a la salida de la última capa, es decir, $\boldsymbol{A}^{[L]}$. A partir del valor estimado por

el modelo y la matriz de observaciones de la variable de interés puede calcularse la pérdida $J(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{c}(\hat{\boldsymbol{y}}_{\boldsymbol{i}}, \boldsymbol{y}_{\boldsymbol{i}})$. Este proceso depende del conjunto de parámetros del modelo, dados por las matrices de pesos $\boldsymbol{W}^{[l]}$ y los vectores de sesgos $\boldsymbol{b}^{[l]}$ de cada capa, que se denotará $\boldsymbol{\Theta} = \{(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}) \, \forall \, l = 1, 2, \ldots, L\}$. El proceso de entrenamiento o ajuste consiste en hallar el conjunto de parámetros $\boldsymbol{\Theta}^*$ que minimicen la función de pérdida, es decir:

$$\boldsymbol{\Theta}^* = \underset{\boldsymbol{\Theta}}{\operatorname{argmin}} \quad J(\hat{\boldsymbol{Y}}, \boldsymbol{Y}).$$

Tal y como se mencionó en la sección anterior, para hallar estos valores de los parámetros se utilizará el descenso gradiente estocástico por lotes, lo que requiere para su actualización hallar la derivada de la función de pérdida respecto a cada uno de los parámetros, para lo cual se utiliza el método de *back propagation* o propagación hacia atrás que recorre de forma inversa la estructura computacional de la red neuronal desde la capa de salida hasta la de entrada de acuerdo a la regla de la cadena (Zhang et al., 2021).

Los cálculos realizados para obtener los gradientes dados en (4-9) y (4-10) pueden ser generalizados para obtener las cuatro ecuaciones fundamentales de la propagación hacia atrás (Nielsen, 2015):

$$\frac{\partial J}{\partial \boldsymbol{z}^{[L]}} = \frac{1}{n} \boldsymbol{c}_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \odot \boldsymbol{g}'^{[L]}(\boldsymbol{z}^{[L]}) \tag{4-24}$$

$$\frac{\partial J}{\partial \boldsymbol{Z}^{[l]}} = \left(\frac{\partial J}{\partial \boldsymbol{Z}^{[l+1]}} (\boldsymbol{W}^{[l+1]})^T \right) \odot \boldsymbol{g}'^{[l]}(\boldsymbol{z}^{[l]}), \tag{4-25}$$

$$\frac{\partial J}{\partial \boldsymbol{W}^{[l]}} = \left(\boldsymbol{A}^{[l-1]}\right)^T \frac{\partial J}{\partial \boldsymbol{Z}^{[l]}}, \tag{4-26}$$

$$\frac{\partial J}{\partial \boldsymbol{b}^{[l]}} = (\boldsymbol{1}_n)^T \frac{\partial J}{\partial \boldsymbol{Z}^{[l]}}, \tag{4-27}$$

y al repetir de forma recursiva este procedimiento se pueden obtener los Jacobianos de la función de pérdida respecto a las matrices de pesos $\dfrac{\partial J}{\partial \boldsymbol{W}^{[l]}}$ y los gradientes respecto a los vectores de sesgos $\dfrac{\partial J}{\partial \boldsymbol{b}^{[l]}}$ para las L capas de la red neuronal, que se denotarán por el conjunto $\boldsymbol{\partial\Theta} = \left\{ \left(\dfrac{\partial J}{\partial \boldsymbol{W}^{[l]}}, \dfrac{\partial J}{\partial \boldsymbol{b}^{[l]}} \right) \; \forall \; l = 1, 2, \ldots, L \right\}$. La inclusión del cálculo de los gradientes lleva al grafo computacional que se observa en la Figura **4-6**.

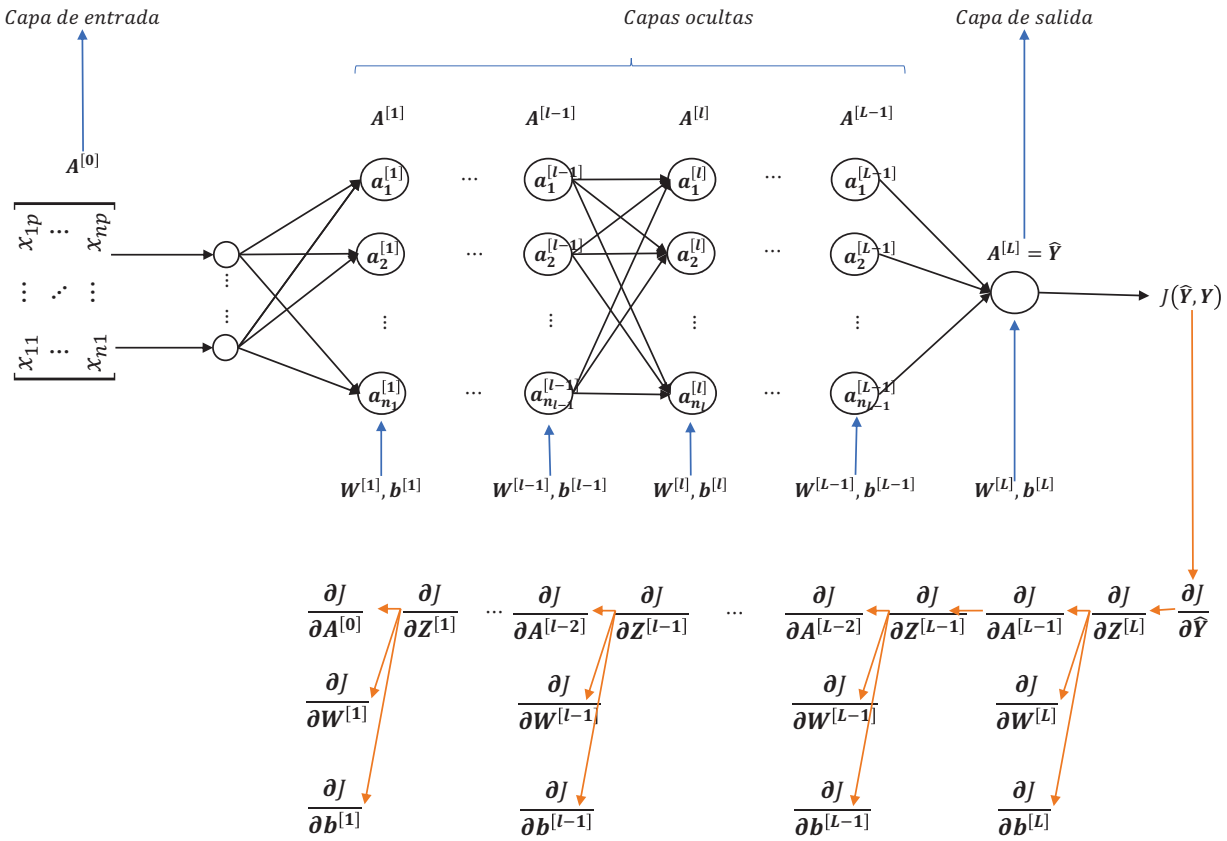

Figura 4-6.: Grafo computacional del perceptrón multicapa con propagación hacia atrás. Fuente: Elaboración propia.

Tal como se mencionó en el caso del ajuste de la regresión Poisson como red neuronal, para el perceptrón multicapa se utiliza el descenso gradiente estocástico por lotes, de modo que el algoritmo utilizando K lotes para el entrenamiento del modelo por E épocas es desarrollado así:

Inicializar $\boldsymbol{\Theta} = \{(\boldsymbol{W}^{[l]}, \boldsymbol{b}^{[l]}) \, \forall \, l = 1, 2, \ldots, l\}$

Repetir E veces:

para k de 1 a B:

Aplicar la propagación hacia adelante para calcular las matrices de variables intermedias $\boldsymbol{Z}^{[l]\{k\}}$ y de activaciones $\boldsymbol{A}^{[l]\{k\}}$ para las L capas.

Aplicar la propagación hacia atrás para calcular
$\boldsymbol{\partial\Theta} = \left\{ \left(\dfrac{\partial J}{\partial \boldsymbol{W}^{[l]}}, \dfrac{\partial J}{\partial \boldsymbol{b}^{[l]}} \right) \; \forall \, l = 1, 2, \ldots, L \right\}$

Actualizar $\boldsymbol{\Theta}$ usando $\boldsymbol{\partial\Theta}$ según algoritmo de descenso gradiente usado.

Donde $\boldsymbol{Z}^{[l]\{k\}} \in \mathbf{R}^{m \times n_l}$ y $\boldsymbol{A}^{[l]\{k\}} \in \mathbf{R}^{m \times n_l}$ denotan, respectivamente, las matrices de funciones afines y de activaciones para el b-ésimo lote de tamaño m.

En este caso, para presentar un ejemplo de la comparación entre el perceptrón multicapa y el modelo de regresión Poisson, se incluye en el repositorio disponible en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast en el notebook "nn_vs_poisson_regression" donde se utilizan ambos modelos sobre los datos del conteo total de bicicletas que cruzan los puentes de la ciudad de Nueva York (of Transportation, 2017). Este conjunto de datos contiene información pública de 214 observaciones del conteo del total diario de ciclistas que cruzan los puentes de Brooklyn, Manhattan, Williamsburg y Queensboro, junto con la información de las temperaturas mínima y máxima del día en que se realizó el conteo, así como el índice de precipitación.

En la Figura **4-7** se exhiben la serie temporal de los datos del conteo de bicicletas, donde se puede evidenciar que en el intervalo de tiempo observado, la serie muestra una varianza constante, pero presenta un cambio de nivel, por otro lado, en la Figura **4-8** se presenta la distribución del conteo de bicicletas respecto a los meses y los días de la semana, lo que muestra que existe un efecto del mes sobre el conteo de bicicletas y un comportamiento periódico en los días de la semana, donde se observa una reducción en el conteo de bicicletas los fines de semana.

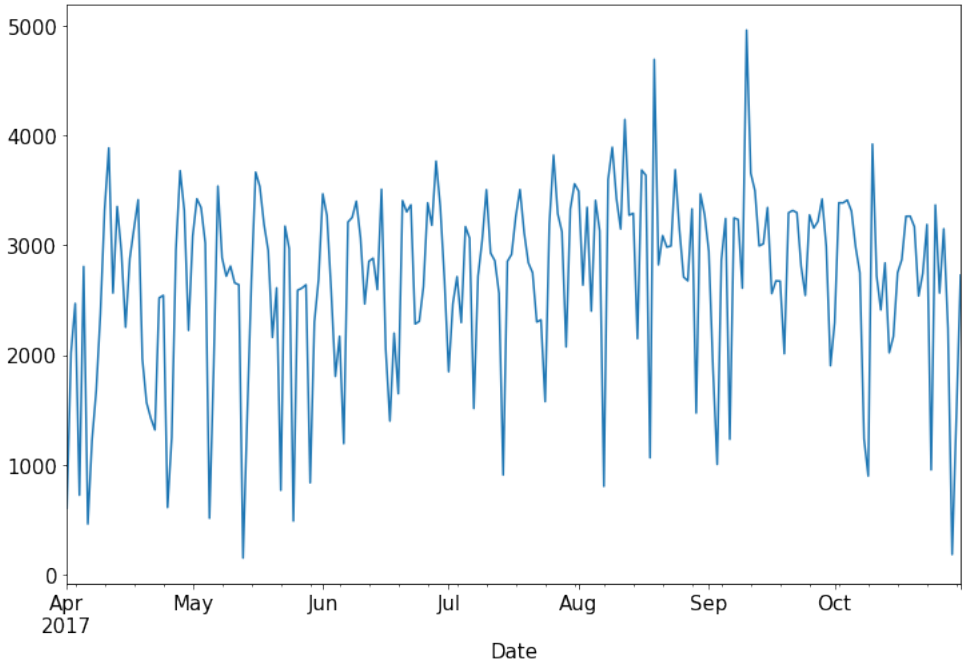

Figura 4-7.: Conteo total de bicicletas que cruzan los puentes de la ciudad de Nueva York

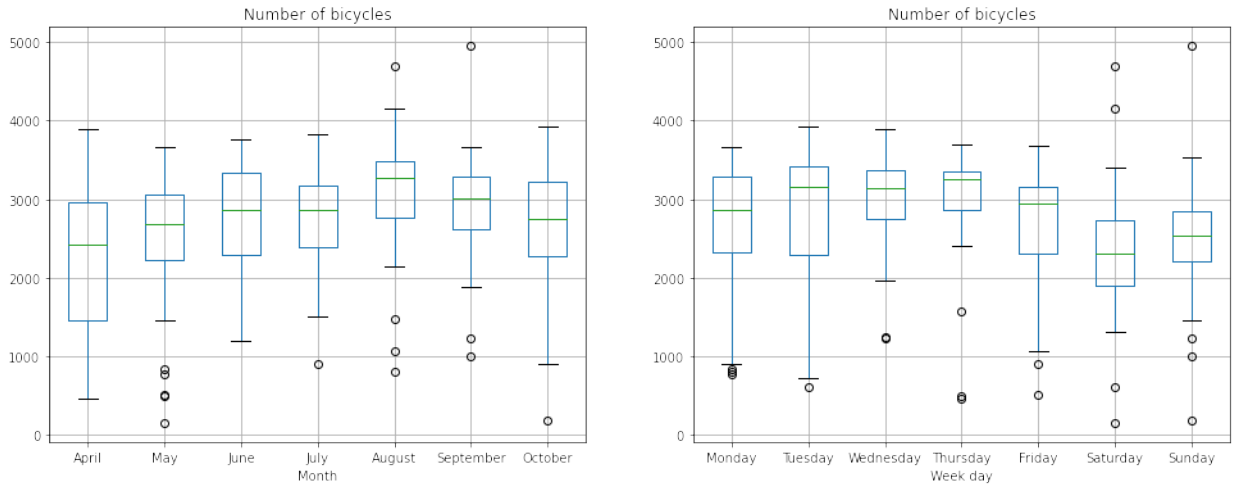

Figura 4-8.: Box plots del conteo total de bicicletas que cruzan los puentes de la ciudad de Nueva York discriminado por mes y día de la semana

Para la modelación se consideran las siguientes variables:

Y : conteo total de bicicletas que cruzan los puentes de la ciudad de Nueva York,

la cual se relaciona con las variables:

- X_1 : Temperatura máxima en el día.
- X_2 : Temperatura mínima en el día.
- X_3 : Índice de precipitación.
- X_4 : Mes al que corresponde la observación.
- X_5 : Día de la semana.
- X_6 : Fecha (día en el mes) de la observación.

Para el ajuste de los modelos se utilizó un total de 171 observaciones, y se dejan 43 observaciones para evaluar las predicciones de modelos. El primer modelo a considerar es el de la regresión Poisson con variable respuesta y predictores previamente descritos, implementada computacionalmente como una red neuronal y obteniendo la estimación de parámetros mediante el descenso gradiente, tal como se plantea en la Sección 4.1.

Por otro lado, para el segundo modelo se seleccionó una red neuronal con 4 capas en total (3 ocultas y la capa de salida) las cuales tienen las siguientes características:

- Primera capa con 8 nodos y función de activación ReLU
- Segunda capa con 16 nodos y función de activación ReLU
- Cuarta capa con 16 nodos y función de activación ReLU
- Capa de salida con 1 nodo y función de activación exponencial

Además, la función de pérdida utilizada corresponde a la ecuación (3-28), asumiendo una distribución Poisson para la variable respuesta.

Al ajustar a los datos los modelos mencionados, se obtienen los resultados presentados en la Tabla **4-4** para el MSE de ajuste, calculado según:

$$\text{MSE}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2, \tag{4-28}$$

con y_i la i-esima observación de la variable respuesta y $\widehat{y_i}$ el valor ajustado por el modelo dado él i-ésimo vector de observaciones $\boldsymbol{x_i} = [x_{i1}, x_{i2}, x_{i3}, x_{i4}, x_{i5}, x_{i6}]$.

Tabla 4-4.: MSE de ajuste para los modelos trabajados

	Regressión Poisson (como NN)	**Perceptrón multicapa**
MSE	335079.97	257442.28

Por otro lado, en la Figura **4-9** se presenta el gráfico de los valores reales del conteo de ciclistas versus valores ajustados por ambos modelos dentro del conjunto de datos de entrenamiento y en la Figura **4-10** se presentan los gráficos de los residuos del ajuste, calculados según (3-48), versus el tiempo y versus los valores ajustados.

Si bien en la Figura **4-9** parecen apreciarse un buen ajuste de ambos modelos, especialmente del perceptrón multicapa (lo que confirma el resultado del MSE de ajuste presentado en la tabla **4-4**), en la Figura **4-10** se observan patrones en los residuos de ambos modelos que algo de carencia de ajuste pero no conduce a una violación fuerte del supuesto de media cero para los errores de ajuste, aunque son menos marcados para el caso del perceptrón multicapa. Por otro lado, para ambos modelos no se observa evidencia fuerte contra el supuesto de varianza constante.

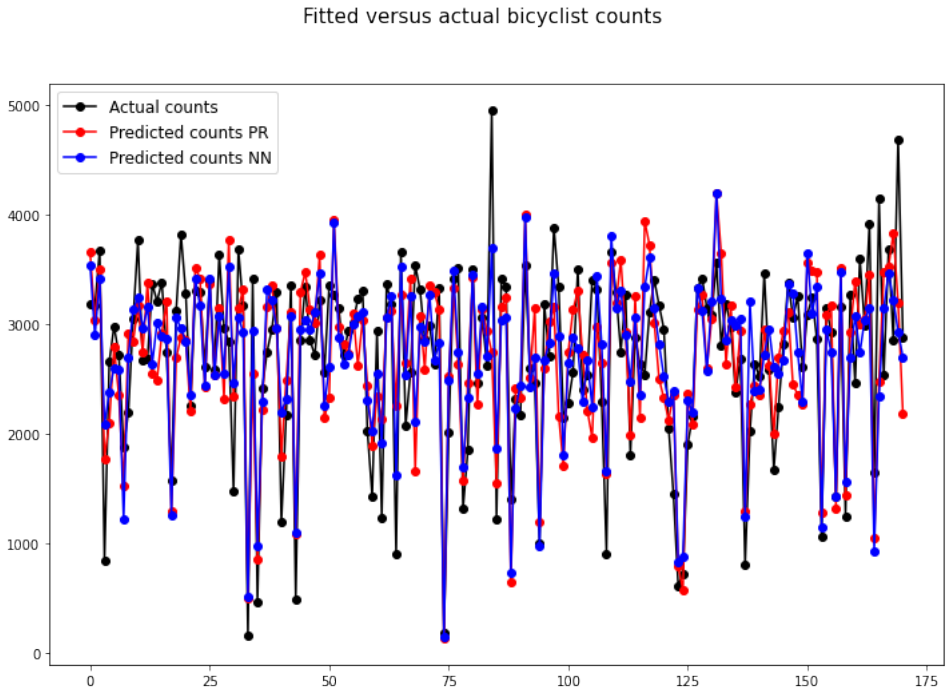

Figura 4-9.: Valores ajustados contra reales del total de ciclistas

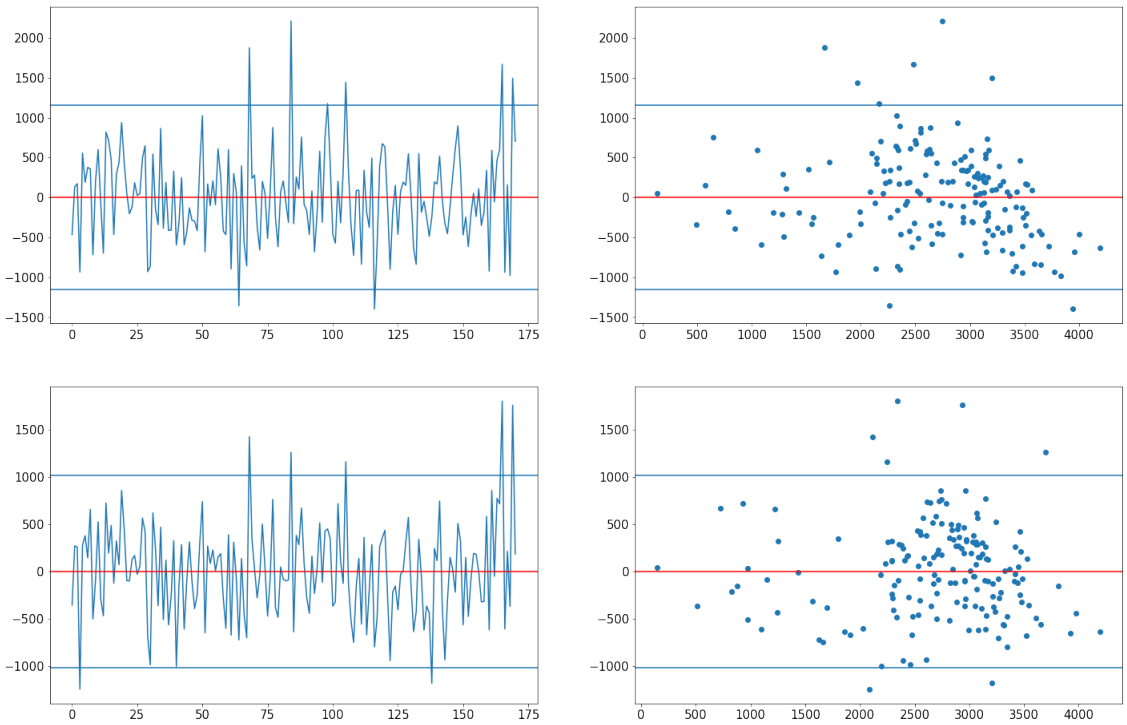

Figura 4-10.: Gráficos de residuales para el modelo de regresión Possion (arriba) y el modelo de perceptrón multicapa (abajo)

En cuanto a los valores predichos por los modelos para el conjunto de evaluación, presentados en la Figura **4-11** junto a los valores reales, donde se observa que los pronósticos se aproximan relativamente bien a los valores reales sin sesgos sistemáticos o tendencias a dar valores siempre mayores o reales que los reales, siendo un poco más preciso el perceptrón multicapa. Este resultado se confirma al calcular el MSE de predicción, el cual se calcula con la fórmula presentada en (4-28), con la salvedad de que las observaciones corresponden al conjunto de evaluación, y cuyos resultados se presentan en la Tabla **4-5**.

Tabla 4-5.: MSE de predicción para los modelos trabajados

	Regressión Poisson (como NN)	**Perceptrón multicapa**
MSE	304356.48	297900.96

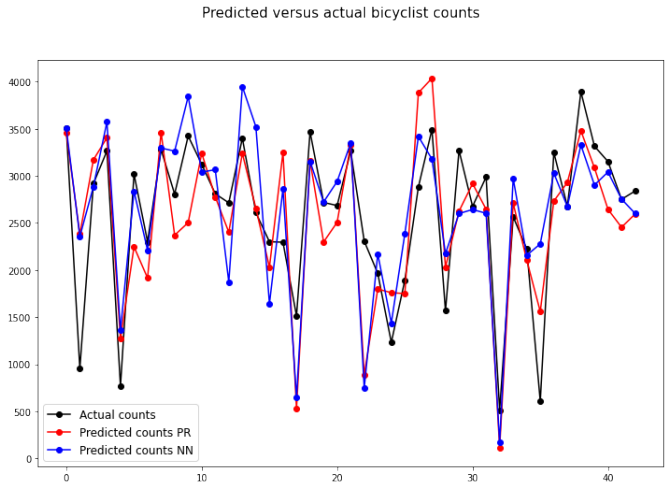

Figura 4-11.: Valores predichos contra reales del total de ciclistas

Con los anteriores resultados se concluye que si bien no se encuentra en el análisis de residuos una gran diferencia entre los dos modelos, en cuanto a la precisión del ajuste y del pronóstico parece mejor el modelo del perceptrón multicapa que el modelo de regresión Poisson como NN. Sin embargo, cabe tener en cuenta que el perceptrón multicapa requiere de un esfuerzo superior en la selección de hiperparámetros como el número de capas, los nodos en estas capas y las funciones de activación, así como tiempos computacionales mayores para el entrenamiento de múltiples épocas conservando los pesos que mejor desempeño presentan a través de las épocas, factores a tener en cuenta para casos como éste donde dada la limitada cantidad de datos, no hay una diferencia muy notoria en el desempeño de ambos modelos, por lo que pueden existir casos para los que recurrir al ajuste con este método no sea recomendable. Si bien este trabajo no explora en profundidad la comparación entre el modelo de regresión y el perceptrón multicapa y sus casos de uso, en futuras comparaciones entre el modelo autorregresivo y el basado en redes neuronales, se volverá sobre este punto.

4.3. Modelamiento secuencial: Redes Neuronales Recurrentes (RNN)

Las Redes neuronales recurrentes (o RNN por su sigla en inglés) son una familia de redes neuronales para el procesamiento de datos secuenciales y pueden escalar a secuencias mucho más largas de lo que sería práctico trabajar arquitecturas que no tienen un diseño adecuado para secuencias de gran longitud (Goodfellow et al., 2016). Las RNN capturan las dinámicas de las secuencias vía conexiones recurrentes (Zhang et al., 2021), que pueden entenderse como ciclos en los nodos de las redes que permiten compartir parámetros entre diferentes partes de un modelo. Estas redes se desenvuelven a través de los pasos de tiempo (o los pasos de una secuencia) utilizando los mismos parámetros para cada paso, de modo que la arquitectura aplicada en la red es ahora aplicada al input correspondiente a cada paso de tiempo.

Para visualizar la idea de la red desenvuelta a través de los pasos de tiempo, comencemos por simplificar la representación de un perceptrón multicapa como se observa en la Figura **4-12**. Allí, el bloque de las capas ocultas corresponde a las distintas capas de la red con los respectivos parámetros que se emplean en el cálculo de las activaciones. Para el caso de las RNN, como se observa en la Figura **4-13**, se tiene un input para cada paso de tiempo de la secuencia, de longitud T_x, y este input pasa por las capas ocultas y arroja una salida para cada elemento de la secuencia utilizando los mismos parámetros. Las conexiones que se observan entre las capas ocultas a través del tiempo corresponden a las conexiones recurrentes que permiten capturar las dinámicas de la secuencia.

Las RNN se han convertido en herramientas prácticas con grandes resultados en tareas como el reconocimiento de escritura, traducción automática y reconocimiento de diagnósticos médicos. Si bien recientemente han cedido un terreno considerable a los modelos Transformers, las RNN se han convertido en el modelo por defecto para el manejo de secuencias complejas en deep learning, y continúan vigentes hasta el día de hoy en el modelamiento secuencial (Zhang et al., 2021). La presente Sección explora la estructura matemática del modelo de las RNN, las celdas LSTM para la actualización de los estados y se presentan ejemplos de su implementación, sin embargo, el detalle del cálculo de los gradientes mediante la propagación hacia atrás no se presentará en esta y Futuras secciones. Si bien en fuentes como Goodfellow et al. (2016) o Zhang et al. (2021) puede consultarse la propagación hacia atrás para este tipo de modelos, las plataformas de software orientadas a la construcción de modelos de deep learning normalmente incluyen algoritmos de diferenciación automática, (Zhang et al., 2021) permitiendo centrar la atención del desarrollo en la generación del modelo, sin preocuparse por la obtención de los gradientes.

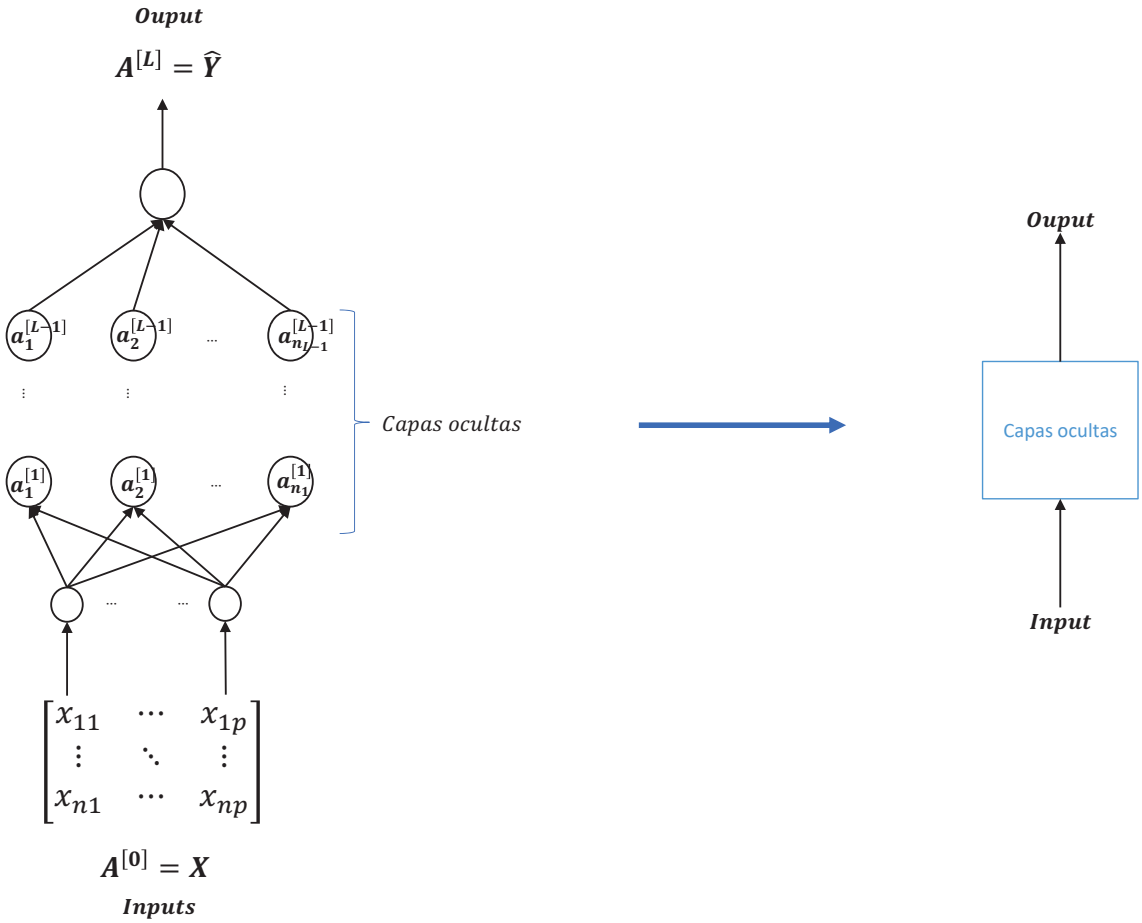

Figura 4-12.: Representación simplificada de una red neuronal. En este caso se presenta un perceptrón multicapa. Fuente: Elaboración propia

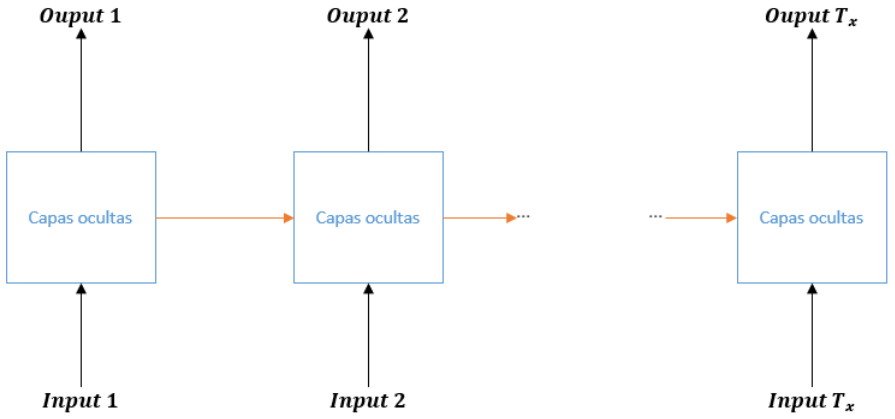

Figura 4-13.: Representación del funcionamiento básico de una red neuronal recurrente. Fuente: Elaboración propia

4.3.1. Redes Neuronales Recurrentes

Como se vio anteriormente, los grafos computacionales pueden utilizarse para formalizar la estructura de un conjunto de cálculos, como aquellos involucrados en la obtención de las salidas de una red neuronal y de la función de pérdida a partir de las entradas y parámetros (Goodfellow et al., 2016). Para entender el modelamiento de las RNN, en esta Sección se explicará la idea de un grafo computacional con una estructura repetitiva, normalmente correspondiente a una cadena de eventos, lo que resulta en los parámetros compartidos a través de la estructura de la red.

En los sistemas dinámicos clásicos (Goodfellow et al., 2016) se tiene el estado del sistema en el momento t:

$$s^{(t)} = f(s^{(t-1)}; \boldsymbol{\theta}), \tag{4-29}$$

que en muchos modelos de redes neuronales recurrentes es transformado en

$$h^{(t)} = f(h^{(t-1)}, \boldsymbol{x^{(t)}}; \boldsymbol{\theta}), \tag{4-30}$$

donde la variable h representa el estado de las capas ocultas (hidden units) y el vector $\boldsymbol{x^{(t)}}$ los datos de entrada en el momento t.

La anterior ecuación puede entenderse desde la perspectiva de los modelos estado espaciales, en los cuales existe un estado de interés latente o no observable, que evoluciona a lo largo del tiempo (Fearnhead, 2011). El estado $s^{(t)}$ permite describir la estructura del modelo para la variable observada a lo largo del tiempo, $y^{(t)}$, como:

$$h^{(t)}|\{(y^{(1)} \dots y^{(t-1)}), (h^{(1)}, \dots, h^{(t-1)})\} \sim P(h^{(t)}|h^{(t-1)}, \boldsymbol{\theta}), \tag{4-31}$$

$$y^{(t)}|\{(y^{(1)} \dots y^{(t-1)}), (h^{(1)}, \dots, h^{(t-1)})\} \sim P(y^{(t)}|h^{(t)}, \boldsymbol{\theta}), \tag{4-32}$$

donde $\boldsymbol{\Theta}$ corresponde a los parámetros del modelo y se tiene el supuesto de que cada observación en el momento t solo depende del estado oculto. Sin embargo, como se observa en la ecuación (4-30), el modelo se generaliza de modo que el estado depende también de variables de entrada $\boldsymbol{x^{(t)}}$ (que pueden ser los valores de la secuencia en el paso inmediatamente anterior).

Las RNN son redes neuronales con estados ocultos (Zhang et al., 2021) que al ser entrenadas para desempeñar una tarea que requiere realizar pronósticos del futuro basados en el pasado, "aprenden" a utilizar a $h^{(t)}$ como un resumen de los aspectos relevantes para la tarea de todas las observaciones pasadas de la secuencia hasta el momento t. Un grafo computacional de esta estructura de cálculos se muestra en la Figura **4-14**, donde los recuadros azules representan las capas ocultas de la red.

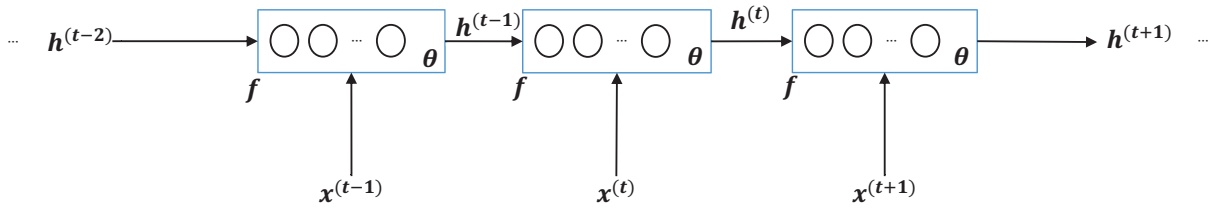

Figura 4-14.: Grafo computacional representando el cálculo recurrente del estado oculto. Fuente: Elaboración propia

Podemos plantear la estructura de las RNN a partir del modelo de la red neuronal incluyendo el cálculo recurrente de los estados ocultos. Como se vio en las definiciones para el perceptrón multicapa, las entradas a la red normalmente se dan por lotes, y en este caso se parte del lote de datos de entrada $\boldsymbol{A}^{[0](t)} \in \mathbb{R}^{m \times p}$ en el momento t (con m el número de ejemplos en cada lote), que por simplificación denotaremos como $\boldsymbol{A}^{(t)}$. Así, para un lote con m ejemplos, cada fila de $\boldsymbol{A}^{(t)}$ corresponde a un ejemplo en el momento t de la secuencia. Por facilidad, se inicia con el caso de una única capa oculta, cuya salida en el momento t denotamos con $\boldsymbol{H}^{(t)} \in \mathbb{R}^{n \times u}$, siendo u el número de nodos (o unidades). Recuerde que en el caso del perceptrón multicapa se tiene la ecuación (4-17) para el cálculo de la activación de una capa, la cual toma por entradas la activación de la capa anterior (o de las entradas del modelo en la primera capa oculta). En el caso de las RNN el cálculo de las salidas de la capa oculta en el momento t (que corresponde al estado oculto) incluye la información de la salida de la capa oculta en el momento anterior, $t-1$, además de la activación de la capa anterior en el momento t, es decir:

$$\boldsymbol{H}^{(t)} = \boldsymbol{g_h}(\boldsymbol{A}^{(t)}\boldsymbol{W_{ah}} + \boldsymbol{H}^{(t-1)}\boldsymbol{W_{hh}} + \boldsymbol{b_h}), \tag{4-33}$$

donde $\boldsymbol{W_{ah}} \in \mathbb{R}^{p \times u}$ es la matriz de los pesos de las entradas sobre la salida de la capa oculta en el momento t, $\boldsymbol{W_{hh}} \in \mathbb{R}^{u \times u}$ es la matriz de pesos de la salida de la capa oculta en el momento $t-1$, sobre la salida de la capa oculta en el momento t, $\boldsymbol{b_h} \in \mathbb{R}^{1 \times h}$ el vector de sesgos de las unidades de la capa oculta y $\boldsymbol{g_h}$ denota la función de activación para obtener la salida de la capa oculta.

La relación entre las salidas de las capas ocultas de tiempos adyacentes, $\boldsymbol{H}^{(t)}$ y $\boldsymbol{H}^{(t-1)}$, captura y retiene la información histórica hasta el momento de tiempo actual, es decir, el

estado o memoria de la red, por lo que a la salida de la capa oculta se le llama estado oculto. Desde que el estado oculto repite el cálculo del paso anterior en cada paso de tiempo, se vuelve un cálculo recurrente y las capas que los ejecutan reciben el nombre de capas recurrentes (Zhang et al., 2021).

Si bien existen muchas formas de construir las RNNs, el estado oculto definido en (4-33) es muy común. En cuanto al cálculo de la salida en el momento t, $\widehat{\boldsymbol{Y}}^{(t)} \in \mathbb{R}^{m \times n_y}$, donde n_y es el número de variables del vector de salida ($n_y > 1$ en el caso de multiples series de tiempo), su cálculo es similar al de la salida en el perceptrón multicapa, es decir,

$$\hat{\boldsymbol{Y}}^{(t)} = \boldsymbol{g_y}(\boldsymbol{H}^{(t)}\boldsymbol{W_{hy}} + \boldsymbol{b_y}), \tag{4-34}$$

con $\boldsymbol{W_{hy}} \in \mathbb{R}^{u \times n_y}$ la matriz de pesos del estado oculto sobre la salida del modelo en el momento t y $\boldsymbol{b_y} \in \mathbb{R}^{1 \times n_y}$ el vector de sesgos.

La Figura **4-15** presenta el grafo computacional del cálculo de esta estructura de RNN. Si bien la arquitectura presentada parece tener la restricción de que el número de pasos de tiempo de las entradas y las salidas debe ser igual, es posible manipular la capa final, donde se calcula el output, para generar un número diferente de pasos de tiempo. Diferentes arquitecturas y estrategias para las RNN se pueden encontrar en Goodfellow et al. (2016) y Zhang et al. (2021).

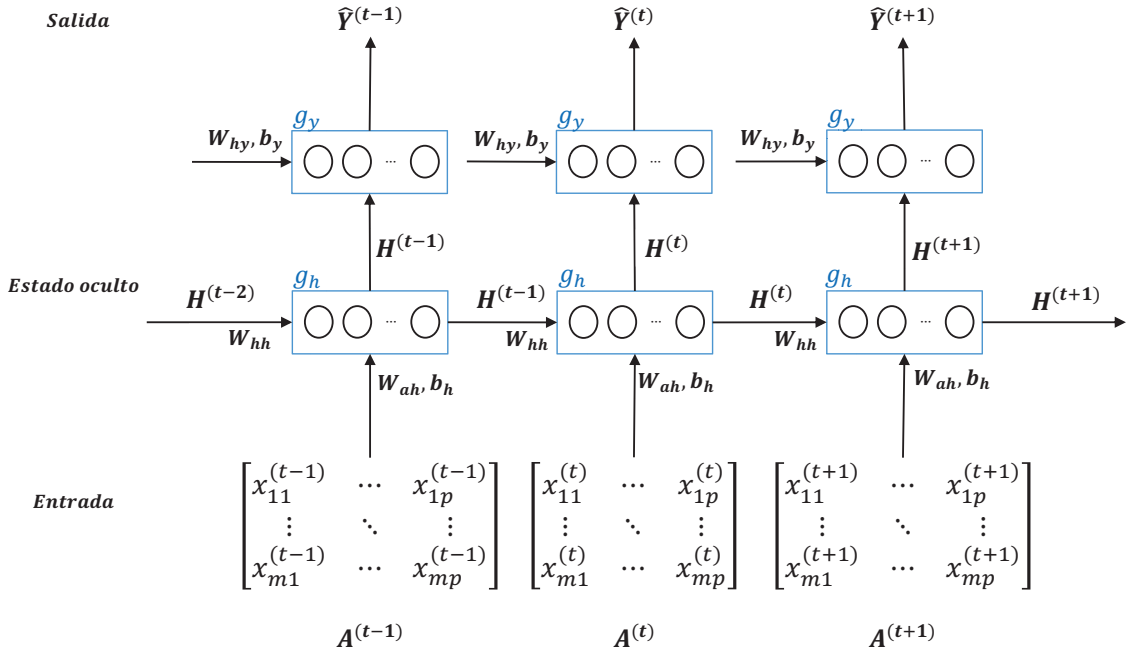

Figura 4-15.: Grafo computacional representando el cálculo de las salidas en una RNN. Fuente: Elaboración propia

4.3.2. Memoria Larga de Corto Plazo (LSTM)

De acuerdo con Goodfellow et al. (2016) los modelos de secuencias más efectivos en aplicaciones prácticas con RNNs son llamados modelos con compuertas (gated RNN), incluyendo los modelos de memoria larga de corto plazo o LSTM por su sigla en inglés (Long Short-Term Memory).

Las RNN con compuertas se basan en la idea de crear caminos a través del tiempo, resultando la LSTM en una de las primeras y más exitosas técnicas para solucionar el problema de los gradientes que se desvanecen o explotan (Zhang et al., 2021), casos en los cuales o se generan gradientes demasiado pequeños que impiden la actualización de los parámetros o valores demasiado grandes que generan la divergencia del algorítmo de optimización. Los modelos LSTM son similares a las RNN estándar vistas en la sección anterior, pero las capas ocultas usuales son remplazados por "celdas de memoria" que contienen un estado interno. El término "memoria larga de corto plazo" significa que aun cuando las RNN estándar poseen una "memoria de largo plazo" en la forma de los pesos, debido a que estos cambian lentamente durante el entrenamiento, también tienen una "memoria de corto plazo" en la forma de las activaciones pasadas de un nodo al siguiente. Sin embargo, el modelo LSTM introduce un tipo de almacenamiento intermedio con la celda de memoria, que es una unidad compuesta, construida de nodos más simples y conexiones, con la idea de acumular información sobre un largo periodo de tiempo, pero manteniendo la opción de "olvidar" el estado una vez que la información ha sido utilizada (Goodfellow et al., 2016).

4.3.3. Celda de memoria con compuertas

De acuerdo con Zhang et al. (2021) las celdas de memoria con compuertas (o gated memory cells) constituyen la diferencia fundamental entre el modelo básico de RNN y el LSTM. Estas celdas cuentan con un estado interno y compuertas multiplicativas que cumplen las funciones de:

- Determinar si una entrada debería impactar o no el estado interno (llamada compuerta de entrada o input gate).
- Determinar si el estado interno debería ser llevado a 0 (compuerta de olvido o forget fate).
- Determinar si el estado interno de una unidad debería poder impactar la salida de la celda (compuerta de salida u output gate).

Estas compuertas funcionan como mecanismos para determinar cuándo un estado oculto debería ser actualizado o reiniciado. Estos mecanismos se aprenden con el entrenamiento de

la red. Para comprender la estructura de las celdas del modelo LSTM y la forma en que se incluyen las compuertas, se presentará el cálculo, los diferentes elementos que la componen y se adicionará al diagrama de la celda

De forma similar a las capas ocultas de las RNN, las celdas del modelo LSTM reciben las entradas en el momento t y el estado oculto del momento anterior, $t-1$, como se muestra en la Figura **4-16**. Dentro de la celda los valores de las compuertas de entrada, olvido y salida se calculan a partir de capas densas (el tipo de capas que componen el perceptrón multicapa) con una función de activación sigmoide (usualmente denotada σ) que hace que los valores de éstas estén en el rango $(0, 1)$. Si se denota la compuerta de entrada como $\boldsymbol{I^{(t)}} \in \mathbb{R}^{m\times u}$, la de olvido como $\boldsymbol{F^{(t)}} \in \mathbb{R}^{m\times u}$ y la de salida como $\boldsymbol{O^{(t)}} \in \mathbb{R}^{m\times u}$, en el momento t, éstas pueden calcularse como:

$$\boldsymbol{I^{(t)}} = \boldsymbol{\sigma}(\boldsymbol{A^{(t)}W_{ai}} + \boldsymbol{H^{(t-1)}W_{hi}} + \boldsymbol{b_i}), \tag{4-35}$$

$$\boldsymbol{F^{(t)}} = \boldsymbol{\sigma}(\boldsymbol{A^{(t)}W_{af}} + \boldsymbol{H^{(t-1)}W_{hf}} + \boldsymbol{b_f}), \tag{4-36}$$

$$\boldsymbol{O^{(t)}} = \boldsymbol{\sigma}(\boldsymbol{A^{(t)}W_{ao}} + \boldsymbol{H^{(t-1)}W_{ho}} + \boldsymbol{b_o}), \tag{4-37}$$

donde $\boldsymbol{W_{ai}, W_{af}, W_{ao}} \in \mathbb{R}^{p\times u}$ son matrices de pesos de las entradas en el momento t sobre las respectivas compuertas; $\boldsymbol{W_{hi}, W_{hf}, W_{ho}} \in \mathbb{R}^{u\times u}$ son matrices de pesos de la salida de la capa oculta en el momento $t-1$ sobre las respectivas compuertas y $\boldsymbol{b_i, b_f, b_o} \in \mathbb{R}^{1\times u}$ son sesgos de las compuertas y $\boldsymbol{\sigma}(\cdot)$ denota a la función de activación sigmoide presentada en (4-14). Cabe resaltar que las dimensiones p y u corresponden al número de covariables en las entradas del modelo y el número de unidades en el estado oculto, respectivamente.

De forma intuitiva, la compuerta de entrada determina qué tanto del valor de las entradas debería agregarse al estado interno de la celda de memoria. Por su parte, la compuerta de olvido determina si debe mantenerse el valor actual del estado interno o si debería o no reiniciarse y la compuerta de salida determina si la celda de memoria debería influenciar o no la salida en el paso de tiempo actual (Zhang et al., 2021).

El siguiente elemento a definir en la celda LSTM es el nodo de entrada, que denotaremos $\boldsymbol{\tilde{C}^{(t)}} \in \mathbb{R}^{m\times u}$, el cual se calcula de forma similar a las compuertas previamente descritas a partir de las entradas del momento t y los estados ocultos del momento $t-1$, pero utiliza una función de activación *tanh* con rango $(-1, 1)$, presentada en (4-15), teniendo como ecuación (Zhang et al., 2021):

$$\boldsymbol{\tilde{C}^{(t)}} = \boldsymbol{tanh}(\boldsymbol{A^{(t)}W_{ac}} + \boldsymbol{H^{(t-1)}W_{hc}} + \boldsymbol{b_c}), \tag{4-38}$$

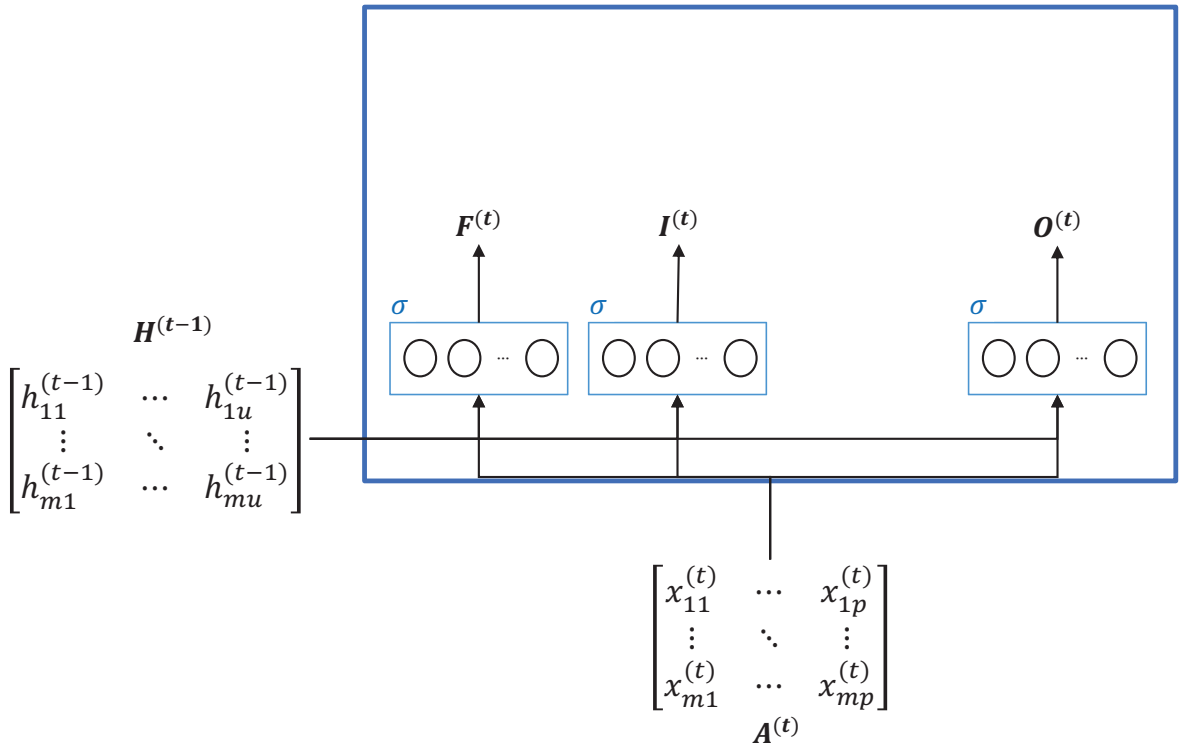

Figura 4-16.: Representación inicial de la celda LSTM al agregar únicamente las compuertas. Fuente: Elaboración propia

donde $\boldsymbol{W_{ac}} \in \mathbb{R}^{p\times u}$ es la matriz de pesos de las entradas en el momento t sobre el nodo de entrada, $\boldsymbol{W_{hc}} \in \mathbb{R}^{u\times u}$ es la matriz de pesos del estado oculto en $(t-1)$ sobre el nodo de entrada y $\boldsymbol{b_c} \in \mathbb{R}^{1\times u}$ es el vector de sesgos. La Figura **4-17** ilustra la celda con el nodo de entrada y las compuertas.

Como se explica en Zhang et al. (2021), con los elementos anteriores definidos, el siguiente paso es actualizar el estado interno de la celda LSTM. Para ello, la compuerta de entrada $\boldsymbol{I^{(t)}}$ determina qué tanto de los datos de entrada, vía $\tilde{\boldsymbol{C}}^{(t)} \in \mathbb{R}^{m\times u}$, se toman en cuenta y la compuerta de olvido $\boldsymbol{F^{(t)}}$ determina qué tanto del estado interno de la celda en el momento anterior, $\boldsymbol{C^{(t-1)}}$, debe tenerse en cuenta. Así, el cálculo del estado interno se realiza mediante el operador de producto Hadamard $\odot$ (elemento a elemento) y se obtiene:

$$\boldsymbol{C^{(t)}} = \boldsymbol{F^{(t)}} \odot \boldsymbol{C^{(t-1)}} + \boldsymbol{I^{(t)}} \odot \tilde{\boldsymbol{C}}^{(t)}. \tag{4-39}$$

Como se observa de lo anterior, las compuertas de entrada y olvido le dan al modelo la capacidad de aprender cuándo mantener el estado interno inalterado y cuándo perturbarlo como respuesta a las entradas, lo que, en la práctica, resulta en modelos más fáciles de entrenar, especialmente para conjuntos de datos con secuencias muy largas (Zhang et al., 2021).

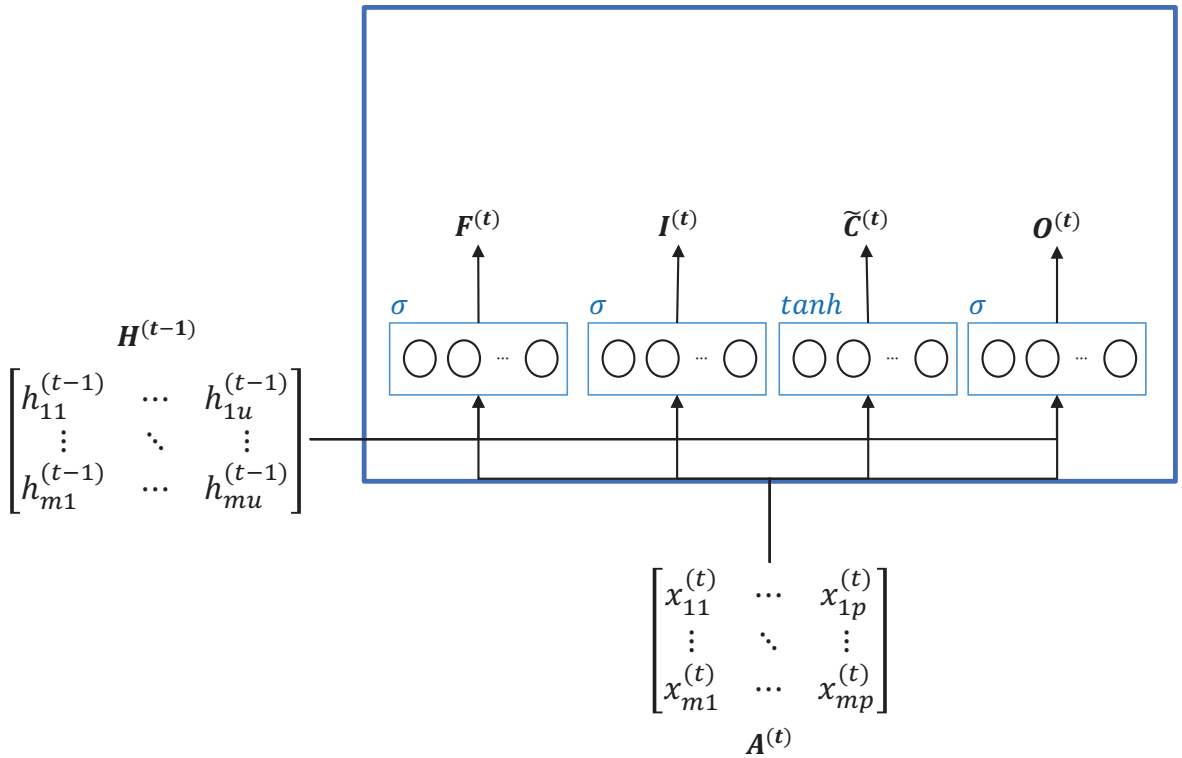

Figura 4-17.: Representación de la celda LSTM con compuertas y nodo de entrada. Fuente: Elaboración propia.

La Figura **4-18** ilustra la celda de memoria una vez se adiciona el cálculo del estado interno y permite apreciar que la celda, además de recibir como entradas el estado oculto en el momento anterior y las entradas en el momento presente, recibe el estado interno del momento anterior. Finalmente, se define cómo calcular la salida de la celda de memoria, es decir, el estado oculto (teniendo en cuenta que la celda de memoria reemplaza a los nodos de la red neuronal recurrente clásica que calculan el estado oculto) denotado $\boldsymbol{H^{(t)}} \in \mathbb{R}^{m \times u}$. Para el cálculo del estado oculto nos valemos de la compuerta de salida y el resultado de aplicar la función $tanh$ al estado interno, garantizando que $\boldsymbol{H^{(t)}}$ pertenezca al intervalo $(-1, 1)$ (Zhang et al., 2021).

$$\boldsymbol{H^{(t)}} = \boldsymbol{O^{(t)}} \odot \boldsymbol{tanh}(\boldsymbol{C^{(t)}}). \tag{4-40}$$

Intuitivamente, cuando la compuerta de salida es cercana a 1, el estado interno de la celda de memoria afecta en las capas siguientes sin ser alterado, mientras que cuando se acerca a 0 se evita que éste tenga un efecto sobre otras capas.

La estructura completa de la celda de memoria del modelo LSTM puede apreciarse en la Figura **4-19** una vez se incluye el cálculo del estado oculto.

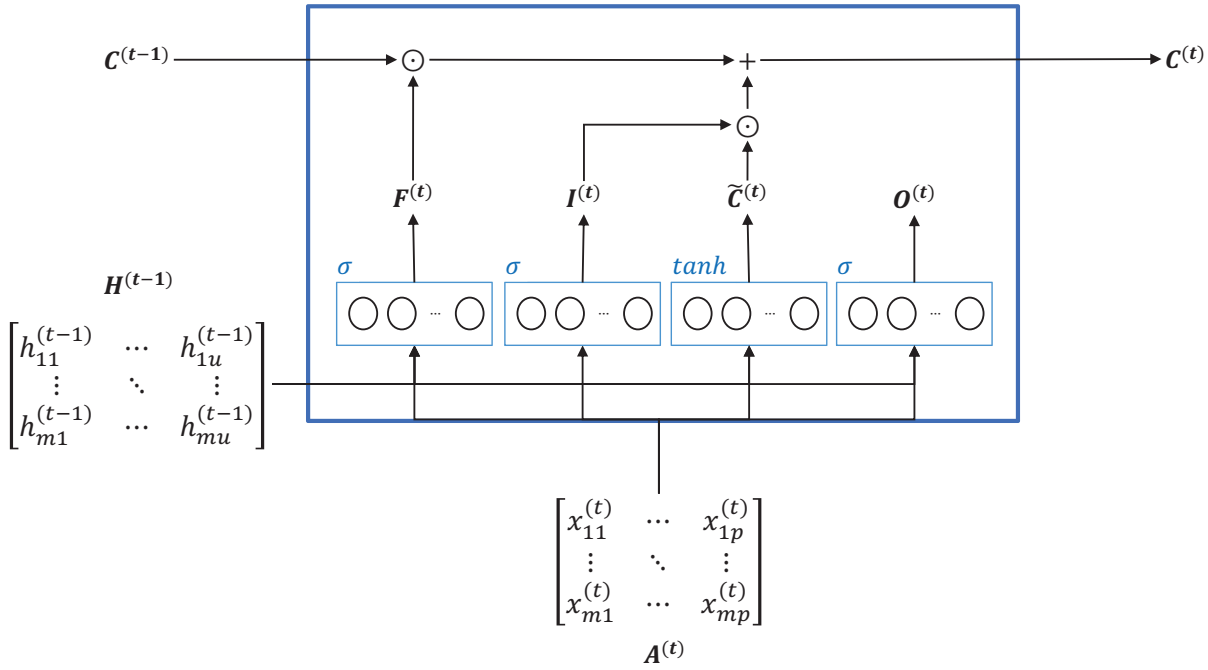

Figura 4-18.: Representación de la celda LSTM con compuertas, nodo de entrada y cálculo del estado oculto. Fuente: Elaboración propia.

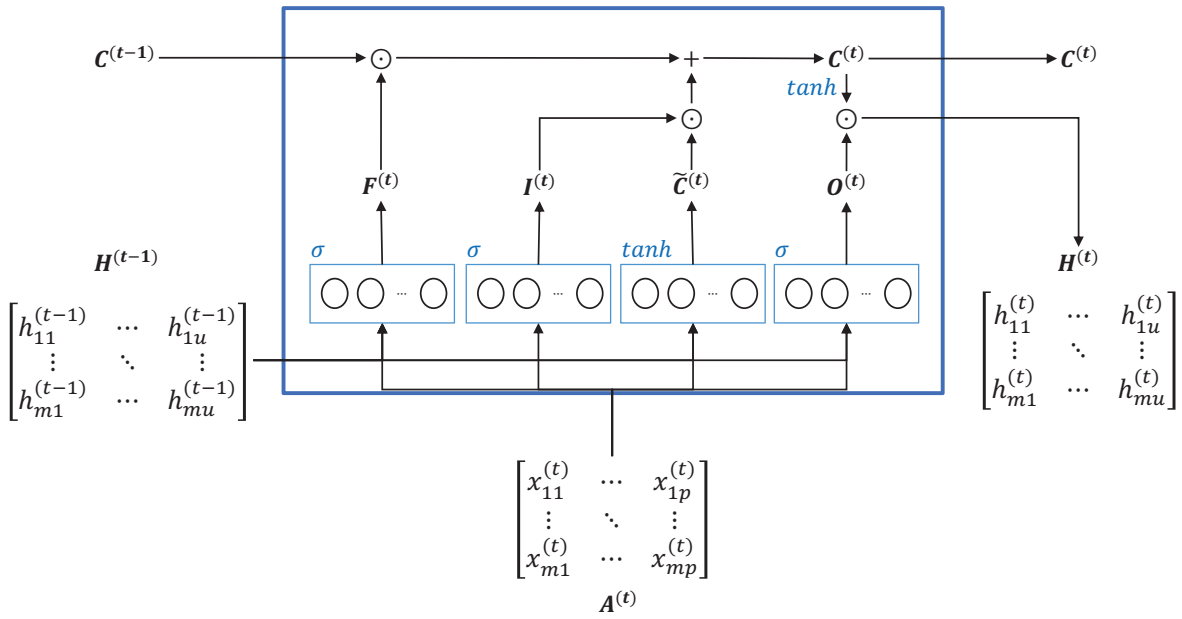

Figura 4-19.: Representación de la estructura completa de la celda LSTM. Fuente: Elaboración propia.

4.3.4. Red neuronal recurrente con celdas LSTM.

En las secciones anteriores se presentó tanto la estructura clásica de las RNN como las celdas de memoria que constituyen el elemento esencial del modelo LSTM. A partir de estos elementos es posible definir el modelo completo de una red neuronal recurrente donde se reemplazan los nodos para el cálculo de los estados ocultos por celdas de memoria LSTM. En este caso, la ecuación para el cálculo de la salida del modelo mediante las capas densas vista en (4-24) permanece igual, sin embargo, el cálculo de los estados ocultos se realiza acorde a las ecuaciones (4-25) a (4-30) para las celdas de memoria LSTM en cada paso de tiempo.

En la Figura **4-20** se puede observar el grafo computacional que representa el cálculo de las salidas del modelo una vez se reemplazan los nodos por las celdas de memoria.

4.3.5. Implementación en Python

En esta sección se retomarán los conjuntos de datos de series de tiempos de conteos utilizados en las implementaciones con **Python** del Capítulo anterior, cuyos códigos, al igual que los utilizados para el desarrollo de esta Sección, están disponibles en el repositorio que acompaña este trabajo en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast.

Si bien las implementaciones presentadas en este Capítulo se valen de herramientas y estrategias de código usuales en el desarrollo de este tipo de modelos en cuanto a la manipulación de los datos para convertirlos en ventanas y el ajuste de las redes neuronales mediante conjuntos de entrenamiento y validación, estos conceptos están fuera del alcance de este trabajo y si bien no se incluyen en el presente marco teórico pueden revisarse en fuentes como Zhang et al. (2021) o Chollet (2017). Finalmente, cabe resaltar que esta Sección se centra en mostrar los ajustes que se obtienen utilizando la metodología de las RNN para el pronóstico de las series, más no se comparan los resultados con los obtenidos en el Capítulo anterior. La comparación del desempeño entre modelos se deja para el Capítulo 6 del presente trabajo.

Caso 1: Infecciones de Campylobacter en Canada

Como se mencionó anteriormente, el conjunto de datos cuenta con 140 observaciones sobre el número de casos de infecciones de campylobacter en el norte de la provincia de Quebec (Canadá) en intervalos de 4 semanas, entre Enero de 1990 y Octubre del 2000. Si bien la variable respuesta se puede definir nuevamente como:

$Y^{(t)}$: Número de casos de infecciones por campylobacter en el norte de la provincia de Quebec en un momento t,

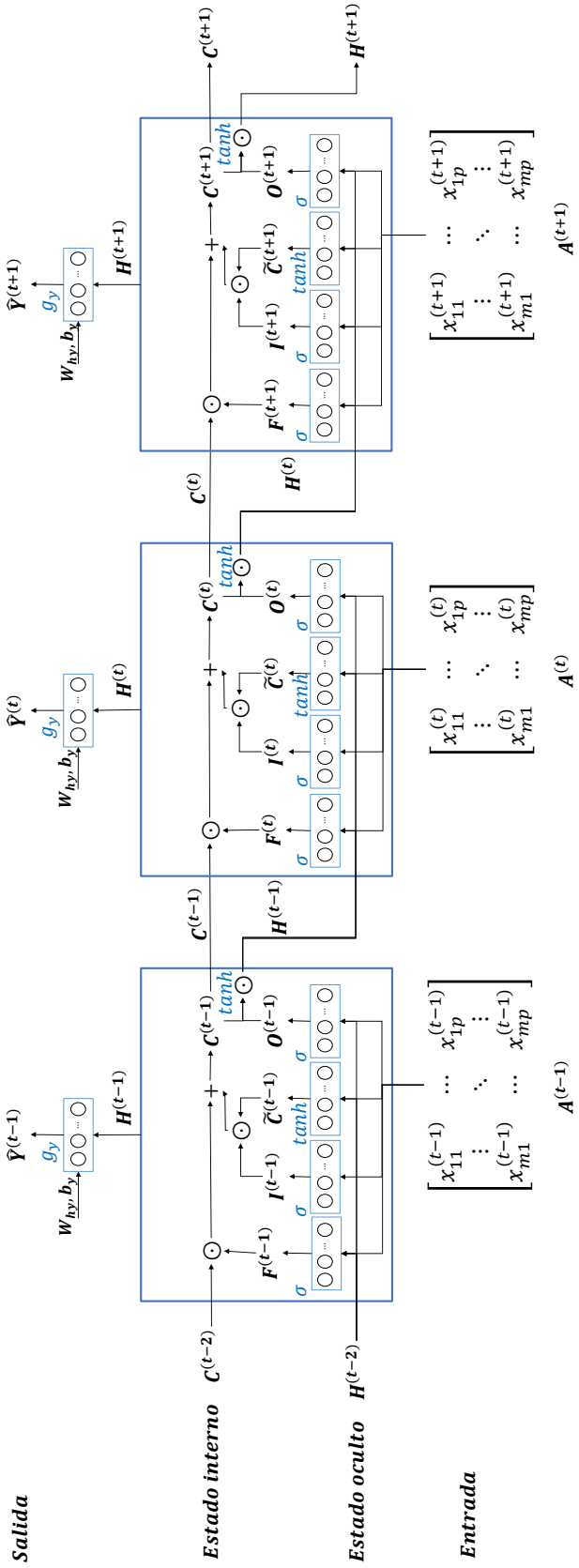

Figura 4-20:. Grafo computacional de la RNN con celdas LSTM. Fuente: Elaboración propia

la generalidad del modelo RNN en cuanto a los inputs permite incluir toda la información respecto a los datos de entrada, como otras covariables, lo que en este caso permite incluir el año y la semana del año a la cual corresponde cada observación, así podemos definir como covariables:

- $X_1^{(t)}$: Número de casos de infecciones por campylobacter en el norte de la provincia de Quebec en un momento t.
- $X_2^{(t)}$: Año al que corresponde la observación del momento t.
- $X_3^{(t)}$: Número de la semana en el año a la cual corresponde la observación del momento t.

Si bien el modelo básico de las RNN tiene la restricción de relacionar una salida a cada entrada del modelo, en este caso se utilizará una versión adaptada que permita longitudes de secuencia diferentes para las entradas y salidas, de modo que se usará la información de las últimas 13 observaciones, que equivale a 1 año en estos datos, según su frecuencia de observación, para pronosticar el número de casos de infección del año inmediatamente posterior. Este segmento de datos de 13 periodos como entrada y el inmediatamente posterior como salida recibe el nombre de ventana y la k-ésima ventana tendrá la estructura:

$$[\boldsymbol{a^{(k)}}, \boldsymbol{a^{(k+1)}}, \boldsymbol{a^{(k+2)}}, \ldots, \boldsymbol{a^{(k+12)}}] \longrightarrow Y^{(k+13)},$$

con $\boldsymbol{a^{(k)}} = [X_1^{(k)}, X_2^{(k)}, X_3^{(k)}]$, y $\longrightarrow$ que indica que los vectores de entradas se mapean a la variable respuesta.

Teniendo en cuenta que hay un total de 140 observaciones en el conjunto de datos, éste puede dividirse en 127 ventanas de datos, respetando el orden temporal al interior de éstas, las cuales son utilizadas para ajustar el modelo, pasándose de forma aleatoria al modelo. En la figura **4-21** se puede observar el gráfico de los valores reales contra los ajustados por el modelo, donde se puede apreciar que el ajuste no aproxima bien los valores mínimos ni las variaciones en el patrón estacional presente en los datos, esto último debido a que el ajuste se asemeja más a una función periódica exacta que no da cuenta de las irregularidades que realmente exhibe el fenómeno estacional observado en la serie. Sin embargo, logra seguir los cambios en el nivel de la serie en el intervalo de tiempo de ajuste. Ahora bien, comparando con los modelos ajustados a estos datos en el Capítulo anterior, de acuerdo al MSE del ajuste, calculado utilizando (3-49), que arroja un valor de 11,2271, este modelo resulta más preciso, además, logró seguir el cambio de nivel de la serie, como ya se señaló, ante lo cual fueron menos sensibles los modelos previos. Una mejor aproximación al nivel de la serie también contribuye a un mejor centramiento de los residuos al rededor de cero y por ende, menor

evidencia en contra del supuesto de que los errores de ajuste tienen media cero, como puede verse en la Figura **4-22**.

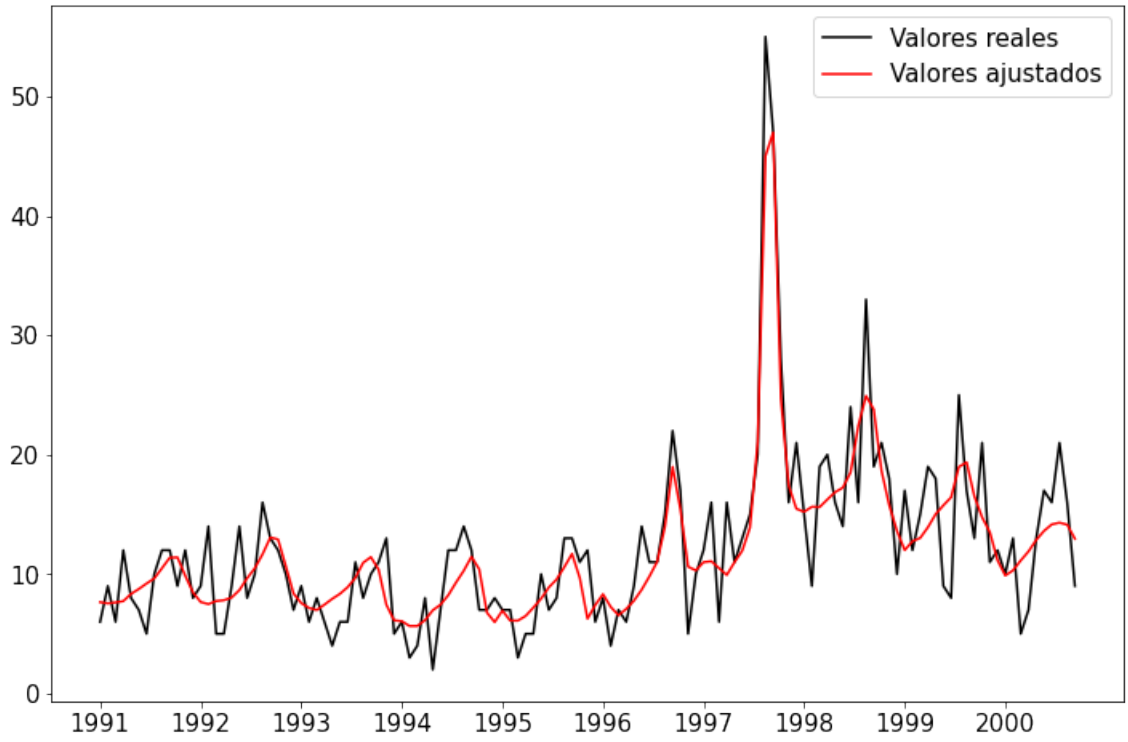

Figura 4-21.: Gráfico de valores reales contra ajustados por el modelo RNN con celdas LSTM, para los datos de Campylobacter.

Por otro lado, podemos analizar los residuales del modelo para verificar si el modelo de pronóstico con la RNN, está capturando toda la información de los datos. En la Figura **4-22** Se puede observar la ACF estimada con los residuos del ajuste, calculados con (3-48), donde se presenta un corte muy cercano al límite crítico en el rezago de orden uno y un valor justo en el límite de orden siete. Sin embargo, teniendo en cuenta que la estimación no excede por mucho el valor crítico y que en general la variación alrededor de cero parece aleatoria, es muy probable que en los rezagos donde aparenta significancia la ACF, se esté incurriendo en error tipo I y en consecuencia que la evidencia muestral en contra de la independencia de los errores no es fuerte. Por otro lado, aun cuando sean observadas autocorrelaciones de orden 1 ó 2 estadísticamente significativas, pero que en términos numéricos no llegan a ser mayores (en valor absoluto) a 0.2, creemos que no representan un serio problema para el pronóstico, por lo menos en el corto plazo. También es importante señalar que con los modelos NN se puede incurrir en sobre ajuste, lo cual contribuye a que se generen autocorrelaciones de orden $k = 1$ entre los residuos, por lo que, en la medida de lo posible, se aconseja utilizar

estrategias para prevenir este problema. Por otro lado, la Figura **4-22** permite evidenciar que no se encuentran patrones en el gráfico de residuales y que estos varían aleatoriamente en torno al cero. Si bien hay varios outliers, no se observa algún patrón que sugiera problemas en el ajuste, y aunque la varianza después de $t = 80$ es mayor a la observada antes de ese tiempo, no hay una violación severa del supuesto de varianza constante.

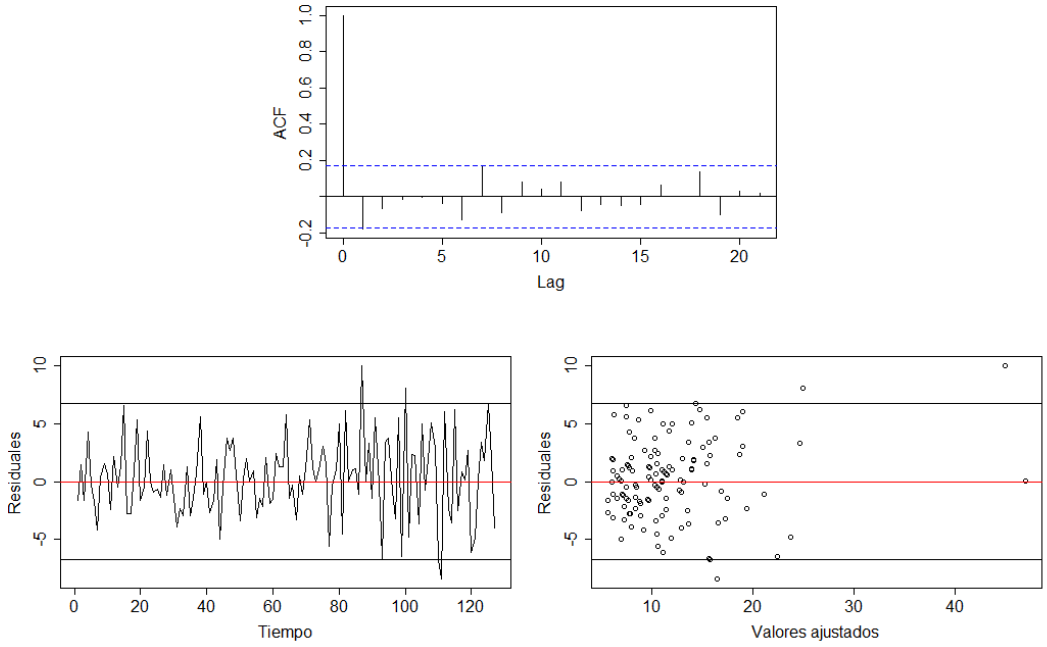

Figura 4-22.: Gráficos para el análisis de supuestos de los errores.

Caso 2: Muertes en accidentes de tránsito en Gran Bretaña

En este caso nuevamente tomaremos toda la información disponible en el conjunto de datos para pronosticar el número de conductores muertos en accidentes de tránsito durante un mes, presentado también en el Capítulo 3. El conjunto de datos disponible en la instalación básica del paquete estadístico **R** (R Core Team, 2023), consiste de 192 observaciones multivariadas mensuales tomadas entre enero de 1969 y diciembre de 1984, sobre las siguientes variables:

- $X_1^{(t)}$: Número de conductores muertos durante el mes t.
- $X_2^{(t)}$: Número de conductores muertos o heridos de gravedad durante el mes t.

- $X_3^{(t)}$: Número de pasajeros del asiento delantero muertos o heridos de gravedad en el mes t.
- $X_4^{(t)}$: Número de pasajeros de los asientos traseros, muertos o heridos de gravedad en el mes t.
- $X_5^{(t)}$: Distancia conducida en el mes t.
- $X_6^{(t)}$: Precio del petróleo en el mes t.
- $X_7^{(t)}$: Conductores de vehículos ligeros de bienes muertos en el mes t.
- $X_8^{(t)}$: Indicador de sí había entrado o no en vigencia la ley de uso obligatorio del cinturón de seguridad en el mes t.

Por otro lado, si incluimos la información del año y mes de cada observación y se denotan éstas por $X_9^{(t)}$ y $X_{10}^{(t)}$ respectivamente, se tiene un conjunto de 10 variables explicativas por registro, de modo que las observaciones del conjunto de entrada tienen la forma:

$$\boldsymbol{a^{(t)}} = [X_1^{(t)}, X_2^{(t)}, \ldots, X_{10}^{(t)}],$$

Sea $Y^{(t)}$ el número de casos de muertes de conductores en accidentes de tránsito en Gran Bretaña en el mes t.

Si utilizamos la información del año anterior para pronosticar el número de muertes del mes inmediatamente posterior, se tiene que la estructura de la k-ésima ventana del modelo es:

$$[\boldsymbol{a^{(k)}}, \boldsymbol{a^{(k+1)}}, \boldsymbol{a^{(k+2)}}, \ldots, \boldsymbol{a^{(k+11)}}] \longrightarrow Y^{(k+12)},$$

En este caso, las 192 observaciones tomadas en el tiempo se traducen en 180 ventanas de entradas y salidas, que respetan el orden temporal, con las cuales se realiza el ajuste del modelo. La Figura **4-23** presenta los valores ajustados por el modelo versus los valores reales, en el cual se puede observar que el modelo parece tener un mejor ajuste que el observado en los modelos aplicados a estos datos en el Capítulo anterior, pues se adapta mejor a los cambios de nivel en la serie y a los valores máximos; sin embargo, no logra ajustar la mayoría de los valores mínimos que ocurren a lo largo de la serie. Su mejor ajuste se confirma al obtenerse el MSE del ajuste, utilizando (3-49), con un valor de 177,6327, notablemente menor al obtenido por los modelos estudiados en el Capítulo anterior para este conjunto de datos.

En cuanto al análisis de los residuos de ajuste, la Figura **4-24** permite observar cortes en los límites críticos para los órdenes de autocorrelación 8 y 10. Por otra parte, en los gráficos de residuos no se observan patrones claros de carencia de ajuste y la variabilidad se da alrededor de cero con varianza constante. De nuevo, en este aspecto, el modelo RNN con celdas LSTM muestra mejor desempeño que los modelos del Capítulo anterior.

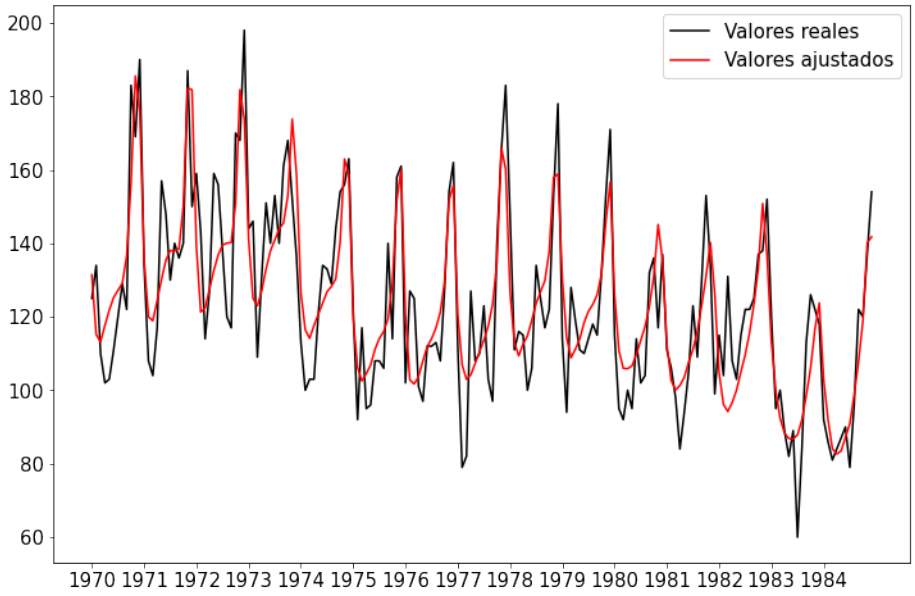

Figura 4-23.: Gráfico de valores reales contra ajustados por el modelo RNN con celdas LSTM, para los datos de muertes de conductores en Gran Bretaña.

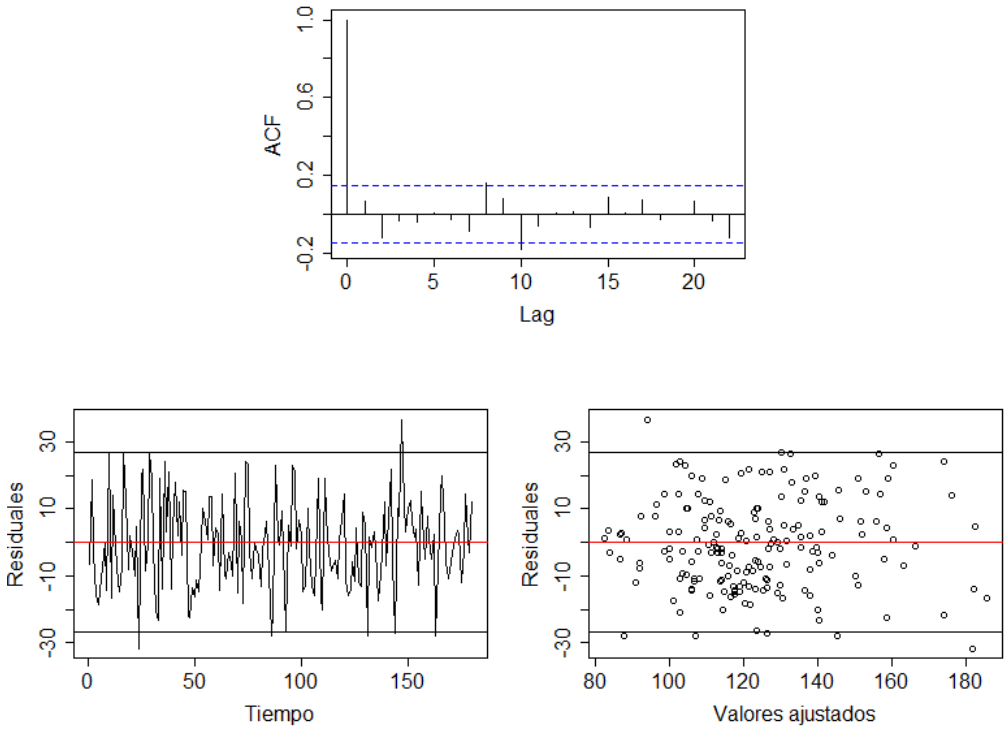

Figura 4-24.: Gráficos para el análisis de supuestos de los errores en el modelo.

5. Transformers para el pronóstico de series de tiempo

De acuerdo con Zhang et al. (2021) durante los primeros años del boom del deep learning fueron protagónicos los resultados producidos utilizando las arquitecturas del perceptrón multicapa, las redes neuronales convolucionales y las redes neuronales recurrentes, produciéndose cambios muy pequeños en las arquitecturas que soportaron muchos de los mayores avances de los 2010s, principalmente en innovaciones metodológicas, como las activaciones ReLU o las capas residuales, pero manteniendo una arquitectura base basada en las ideas clásicas. En cuanto al lenguaje natural, los diseños basados en redes neuronales recurrentes con celdas LSTM dominaron la mayoría de las aplicaciones. Sin embargo, de acuerdo con Phuong y Hutter (2022), desde su aparición hace 6 años, los Transformers han sido modelos muy exitosos para tareas de procesamiento de lenguaje natural y otros dominios, siendo posiblemente la arquitectura más exitosa para el modelamiento secuencial, al demostrar desempeños sin paralelo en aplicaciones como el procesamiento de lenguaje natural, reconocimiento del habla y visión computacional (Zeng et al., 2022).

La idea central detrás de los Transformers es el mecanismo de atención (Zhang et al., 2021), una innovación originalmente planteada para optimizar las RNN con arquitectura codificador-decodificador (encoder-decoder) para traducción automática, donde una red neuronal codificadora (encoder) lee y codifica frases de entrada como vectores de una longitud fija, en tanto que una red neuronal decodificadora (decoder) arroja una traducción a partir de los vectores, con ambas redes entrenadas de manera conjunta para maximizar la probabilidad de una traducción correcta (Bahdanau et al., 2016a). La intuición tras los algoritmos de atención es que en lugar de que el codificador comprima las entradas a vectores, lo ideal es que el decodificador visite la secuencia de entrada en cada paso, enfocándose selectivamente en partes de la secuencia de entrada.

Zhang et al. (2021) mencionan que inicialmente los algoritmos de atención resultaron especialmente exitosos para optimizar las RNN dominando los campos de traducción automática; sin embargo, los mecanismos de atención pronto se convirtieron en centro de atención más allá de la optimización de las RNN de codificador-decodificador. Vaswani et al. (2017) proponen la arquitectura de los Transformer basada en mecanismos de atención con desempeño especialmente bueno y convirtiéndose en el estado del arte para la mayoría de sistemas de procesamiento de lenguaje.

Según Zeng et al. (2022) recientemente han aparecido muchas soluciones basadas en Transformers para el análisis de series de tiempo, algunos de ellos para el problema de los pronósticos de largo plazo. Si bien en dicho trabajo se cuestiona la capacidad de estos modelos para el pronóstico de series de tiempo, el estudio se orienta principalmente a modelos para pronósticos de largo plazo en comparación a otras arquitecturas de redes neuronales. Por otro lado, trabajos como el de Nie et al. (2023) han continuado la exploración de soluciones basadas en Transformers para el pronóstico de series de tiempo, con diversos ajustes a la arquitectura.

Este Capítulo presenta el modelo básico del Transformer tal como se planteó en Vaswani et al. (2017) para luego plantear una arquitectura con ajustes menores al modelo para su aplicación a series de tiempo en lugar de traducción automática y un ejemplo de aplicación de la implementación de la arquitectura ajustada. El objetivo principal es obtener el modelo que se comparará con las metodologías expuestas anteriormente para el pronóstico de múltiples series de tiempo de conteos. Sin embargo, la exploración de otras metodologías basadas en Transformers o la investigación de cuáles de estas tienen un mejor desempeño para el pronóstico de series de tiempo de conteos es un tema de investigación que escapa al alcance de este trabajo.

5.1. Arquitectura del Modelo

De acuerdo con Vaswani et al. (2017) los modelos más competitivos para la transducción de secuencias utilizando redes neuronales tienen una estructura codificador-decodificador. Para estos modelos, el codificador toma como entrada una serie de representaciones simbólicas o *tokens* (elementos que representan una palabra, sílaba, letra, etc.) $(x^{(1)}, \ldots, x^{(T_x)})$ y la mapea a una secuencia de representaciones continuas $\boldsymbol{z} = (z^{(1)}, \ldots, z^{(T_x)})$. Dada dicha representación, el decodificador genera una secuencia de salida $(y^{(1)}, \ldots, y^{(T_y)})$ de representaciones simbólicas o tokens una a la vez, con una naturaleza autorregresiva al utilizar el símbolo generado anteriormente como entrada adicional para general el siguiente.

La arquitectura original del Transformer presentada por Vaswani et al. (2017) sigue la estructura de codificador-decodificador, utilizando capas de "auto-atención" y capas densas (las mismas que componen el perceptrón multicapa) para tanto el codificador como el decodificador, tal como se presenta en la Figura **5-1**.

A continuación, se presentan brevemente los elementos de la arquitectura del transformer según Vaswani et al. (2017). Mayor profundidad sobre estos puede obtenerse en Zhang et al. (2021) o acudiendo a las referencias del artículo original.

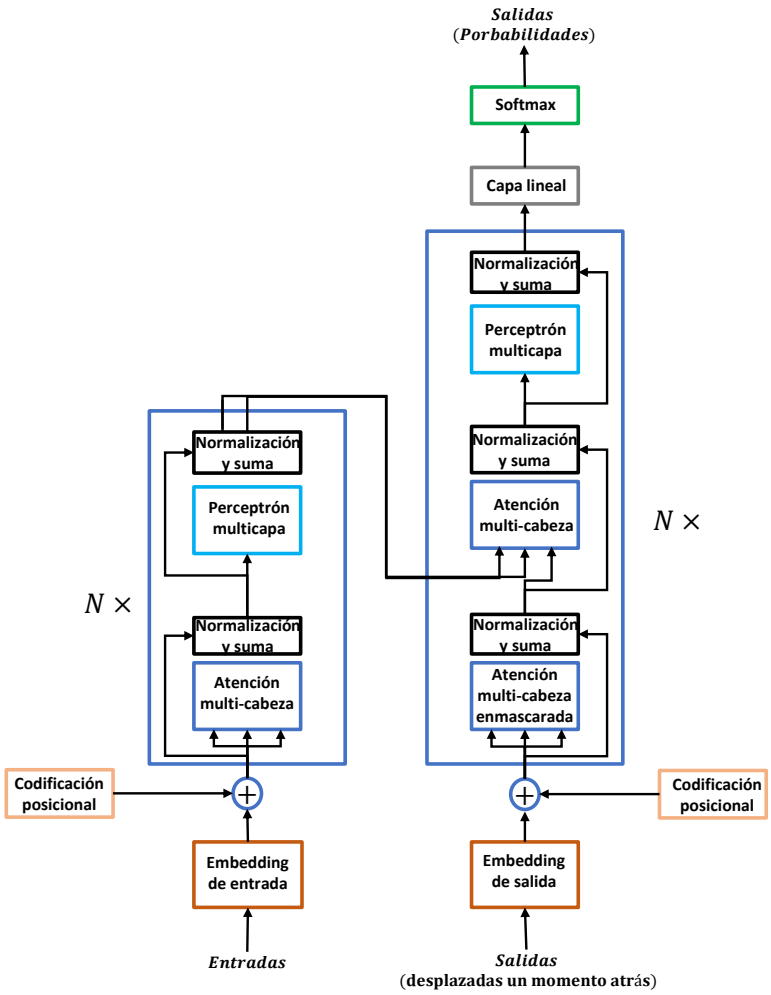

Figura 5-1.: Arquitectura del modelo Transformer. Fuente: Elaboración propia, basado en Vaswani et al. (2017).

5.1.1. Codificador y Decodificador

Codificador: En la arquitectura del modelo presentada en la Figura **5-1** el codificador corresponde al bloque en la parte izquierda, donde el multiplicador $N\times$ determina el número de capas idénticas que se apilan para conformarlo. En Vaswani et al. (2017) se presenta el codificador de $N = 6$ capas idénticas apiladas, cada una con dos sub-capas. La primera consiste de un mecanismo de atención multi-cabeza, y la segunda de un perceptrón multicapa. Los autores utilizan también una conexión, presentada en He et al. (2016), y una normalización de capa, que estandariza a través de las p variables de cada observación (Ba

et al., 2016), denotada LayerNorm, de forma que la salida en cada una de las sub-capas se calcula como LayerNorm(x + Sublayer(x)), donde Sublayer(x) es la función implementada por la sub-capa. Para facilitar la conexión residual, todas las sub-capas en el modelo utilizan salidas de una misma dimensión d_{model}, que para el caso presentado en Vaswani et al. (2017) corresponden también a la dimensión de los embedding, que son representaciones vectoriales de los símbolos de entrada del modelo (Mikolov et al., 2013), $d_{model} = 512$.

Decodificador: En este caso, al referirse a la arquitectura del modelo en la Figura **5-1** el decodificador está representado por el bloque de la derecha, donde nuevamente él $N\times$ determina el número de capas idénticas que se apilan para conformarlo. En Vaswani et al. (2017) se utilizan también $N = 6$ capas idénticas para el decodificador, sin embargo, éstas incluyen una tercera sub-capa adicional que ejecuta un mecanismo de atención "multi-cabeza" sobre las salidas del codificador. Similar al codificador, se emplean también conexiones residuales en cada sub-capa y normalización de capa. Además, se modifica el algoritmo de la sub-capa de auto atención para evitar que las posiciones de la secuencia atiendan a posiciones posteriores en las entradas del codificador, garantizando que las predicciones en la posición t pueden depender únicamente de las salidas conocidas en las posiciones anteriores.

5.1.2. Atención

El desarrollo de los conceptos de atención parte de la lógica de obtención de información utilizada en bases de datos, las cuales, en su forma más simple, son colecciones de pares clave-valor (k, v por sus iniciales en inglés), sobre las cuales podemos operar con consultas (q por su inicial en inglés) que devuelven la información de los valores cuyas llaves cumplen con determinadas condiciones (Zhang et al., 2021).

Vaswani et al. (2017) mencionan que las funciones de atención pueden describirse como el mapeo de una consulta y un conjunto de pares llave-valor con un resultado, donde la consulta, las llaves, los valores y la salida son todos vectores. El concepto de atención introducido en Bahdanau et al. (2016b) genera pesos para cada valor a partir de la relación entre la consulta y la llave correspondiente, llamada función de compatibilidad, y la salida se calcula como la suma ponderada de los valores utilizando los pesos hallados (ver Figura **5-2**). El nombre "atención" se deriva del hecho de que la operación presta particular atención a los valores para los cuales el peso generado por dicha relación es significativo. Según Zhang et al. (2021) existen diferentes formas para el algoritmo de atención y algunas se presentan en dicho trabajo, sin embargo, en Vaswani et al. (2017) se propone particularmente la atención del producto punto escalado.

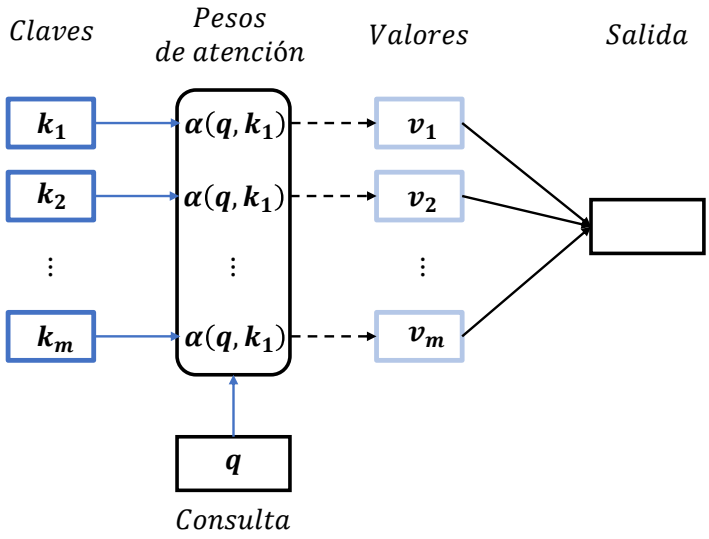

Figura 5-2.: Representación del algoritmo de atención. Fuente: Elaboración propia, basado en Zhang et al. (2021).

5.1.3. Atención del producto punto escalado

El algoritmo o función de atención presentado en Vaswani et al. (2017) para la arquitectura original del Transformer se introduce con el nombre de "Scaled Dot-Product Attention" (o atención del producto punto escalado), computacionalmente representado en la Figura **5-3**. Para esta función de atención, el cálculo de una determinada salida $\boldsymbol{o_j} \in \mathbb{R}^{d_v \times 1}$ toma como entradas la consulta $\boldsymbol{q_j} \in \mathbb{R}^{d_k \times 1}$, y las m llaves $\boldsymbol{k_i} \in \mathbb{R}^{d_k \times 1}$ y valores $\boldsymbol{v_i} \in \mathbb{R}^{d_v \times 1}$, con $i \in 1, \ldots, m$. Sin embargo, Vaswani et al. (2017) mencionan que en la práctica el cálculo de la función de atención sé realiza sobre un conjunto de consultas de forma simultánea empaquetadas en una matriz $\boldsymbol{Q} \in \mathbb{R}^{q \times d_k}$, con q el número de consultas. Las llaves y valores también se empaquetan en las matrices $\boldsymbol{K} \in \mathbb{R}^{m \times d_k}$ y $\boldsymbol{V} \in \mathbb{R}^{m \times d_v}$, y la matriz de salidas se calcula como:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\boldsymbol{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}, \tag{5-1}$$

donde la función softmax corresponde a la operación sobre un vector cualesquiera $\boldsymbol{x} \in \mathbb{R}^{k \times 1}$, con elementos $\{x_j\}, j = 1, 2, \ldots, k$, dada por

$$\text{softmax}(\boldsymbol{x}) = \begin{bmatrix} \frac{\exp(x_1)}{\sum_{j=1}^{k} \exp(x_j)} \\ \frac{\exp(x_2)}{\sum_{j=1}^{k} \exp(x_j)} \\ \vdots \\ \frac{\exp(x_k)}{\sum_{j=1}^{k} \exp(x_j)} \end{bmatrix}. \tag{5-2}$$

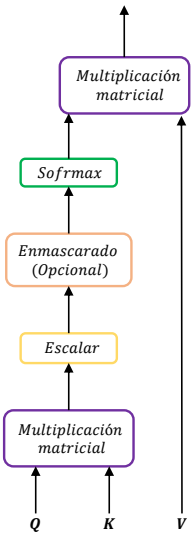

Figura 5-3.: Grafo computacional de la atención del producto punto escalado. Fuente: Elaboración propia, basado en Vaswani et al. (2017).

5.1.4. Atención Multi-Cabeza

De acuerdo con Zhang et al. (2021) en la práctica queremos que dado un mismo conjunto de consultas, llaves y valores, nuestro modelo combine conocimiento de diferentes comportamientos del mismo mecanismo de atención, como podría ser capturar dependencias de diversos rangos en la secuencia. Para ello puede ser beneficioso que el mecanismo de atención utilice representaciones en subespacios diferentes de las consultas, llaves y valores. En este orden ideas, en Vaswani et al. (2017) mencionan que en lugar de calcular una única

función de atención con consultas, llaves y valores de dimensión d_{model}, es beneficioso realizar h proyecciones lineales a espacios de dimensiones d_k, d_k y d_v, respectivamente, con los parámetros de la proyección siendo parte de los parámetros que aprende el modelo. Con cada una de dichas proyecciones lineales se calcula la salida de la función de atención de forma paralela, obteniendo las salidas $\boldsymbol{O_i} \in R^{q \times d_v}$, con $i \in 1, \ldots, h$. Estas salidas se concatenan y proyectan una vez más, como se observa en la Figura **5-4**.

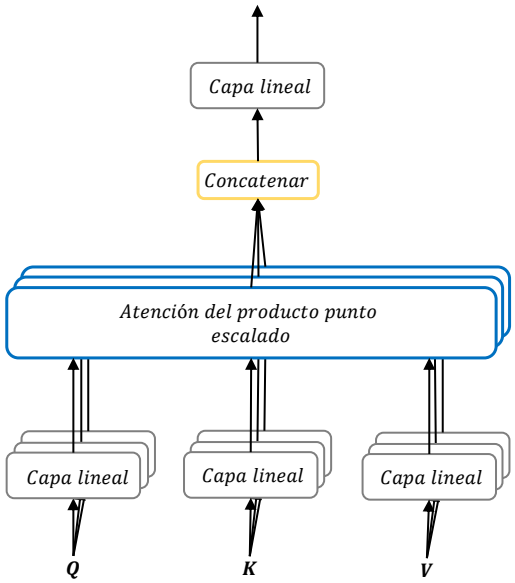

Figura 5-4.: Representación de la atención multi-cabeza. Fuente: Vaswani et al. (2017).

Para el cálculo de la atención multi-cabeza, Vaswani et al. (2017) presentan:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{k}, \boldsymbol{V}) = \quad \text{Concat}(head_1, \ldots, head_h)\boldsymbol{W^O} \tag{5-3}$$

$$\text{where head}_\text{i} = \quad \text{Attention}(\boldsymbol{QW_i^Q}, \boldsymbol{KW_i^K}, \boldsymbol{VW_i^V}), \tag{5-4}$$

Donde las proyecciones son las matrices de parámetros $\boldsymbol{W_i^Q} \in \mathbb{R}^{d_{model} \times d_k}$, $\boldsymbol{W_i^K} \in \mathbb{R}^{d_{model} \times d_k}$, $\boldsymbol{W_i^V} \in \mathbb{R}^{d_{model} \times d_v}$ y $\boldsymbol{W_i^O} \in \mathbb{R}^{hd_v \times d_{model}}$. Los autores mencionan utilizar $h = 8$ cabezas de atención y $d_k = d_v = d_{model}/h = 64$.

La atención multi-cabeza se utiliza de tres formas distintas en el Transformer (ver Figura **5-1**:

- **Atención codificador-decodificador:** En estas sub capas las consultas provienen de la capa anterior del decodificador y las parejas llave-valor provienen de las salidas del codificador, permitiendo a cada posición del decodificador atender a todas las posiciones de la secuencia de entrada.
- **Auto-atención en el codificador:** En una sub-capa de auto-atención, tanto las consultas como los pares llave-valor provienen del mismo lugar. En este caso, todos provienen de las salidas de la capa anterior del codificador, de forma que cada posición en el codificador pueda atender a todas las posiciones de la capa anterior.
- **Auto-atención en el decodificador:** De forma similar, el mecanismo permite al decodificador atender a todas las posiciones en el decodificador hasta, e incluyendo, esa posición.

5.1.5. Redes de alimentación hacia adelante

Además de las sub-capas de atención, las capas del codificador y decodificador contienen una red densa de alimentación hacia adelante (perceptrón multicapa). La red consiste de dos transformaciones lineales con una activación ReLU entre ellas, como se observa en la ecuación (5-5) (Vaswani et al., 2017).

$$\text{FFN}(x) = \text{máx}(0, x\boldsymbol{W_1} + b)\boldsymbol{W_2} + b_2. \tag{5-5}$$

En Vaswani et al. (2017) mencionan que aunque las transformaciones lineales son las mismas a través de diferentes posiciones, utilizan diferentes parámetros de una capa a otra. En el artículo se tiene una dimensionalidad de $d_{ff} = 2048$ nodos en la capa interior de la red y tanto las entradas como las salidas tienen dimensionalidad $d_{model} = 512$.

5.1.6. Codificación Posicional

A diferencia de las RRNs, que procesan recurrentemente los tokens de una secuencia uno a uno, las funciones de atención realizan cálculos en paralelo y no preservan el orden de la secuencia. Sin embargo, el orden de las entradas puede ser un elemento de interés para el modelo (Zhang et al., 2021). La manera dominante de aproximarse a la preservación de la información del orden de los tokens en la secuencia es la representación del orden como una entrada adicional asociada a cada token. Estas entradas reciben el nombre de codificadores posicionales.

En Vaswani et al. (2017) se agregan codificadores posicionales a las entradas de los embeddings tanto en el codificador como en el decodificador, dando una dimensión d_{model} a los codificadores posicionales de manera que los dos puedan sumarse. Los autores mencionan el uso de codificadores posicionales con funciones seno y coseno de diferentes frecuencias. Zhang et al. (2021) describen el codificador posicional partiendo de una representación $\boldsymbol{X} \in \mathbb{R}^{T_x \times d_{model}}$ que contiene los embeddings para una secuencia con T_x tokens. El codificador posicional tiene por resultado $\boldsymbol{X} + \boldsymbol{P}$ utilizando una matriz de posicionamiento $\boldsymbol{P} \in \mathbb{R}^{T_x \times d_{model}}$ cuyos elementos en la i-ésima fila y $2j$-ésima o $(2j+1)$-ésima columna están dados por:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d_{model}}}\right), \tag{5-6}$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d_{model}}}\right). \tag{5-7}$$

De esta forma, cada índice i indica la posición en la secuencia y el índice j está relacionado con la dimensión del embedding (dimensión del vector de representación de cada token). De este forma la longitud de onda forma una progresión geométrica de 2π hasta $10000 * 2\pi$. Según Vaswani et al. (2017) esta función permite al modelo aprender fácilmente a atender a las posiciones relativas.

5.2. Implementación propuesta para series de tiempo

Si bien trabajos como los presentados por Nie et al. (2023) y Zeng et al. (2022) presentan arquitecturas para la aproximación al pronóstico de series de tiempo, las implementaciones en dichos trabajos se alejan del modelo base del Transformer y están orientadas a pronósticos de largo plazo. Para el presente trabajo se desea comparar el pronóstico del modelo Transformer para el pronóstico de series de tiempo de conteos con otros modelos cuyo objeto de diseño es el ajuste a las series y el pronóstico a corto y mediano plazo, por lo que en este trabajo de tesis se propone un ajuste a la arquitectura planteada de Vaswani et al. (2017) para el Transformer dada la naturaleza de los datos de entrada y salida, ya que el modelo originalmente se diseñó para la traducción automática.

5.2.1. Entradas y salidas del modelo

Como se puede verificar en Zhang et al. (2021), los algoritmos de atención dependen para su correcto funcionamiento de la aplicación de la normalización por capas y los rangos normalizados para las entradas. La arquitectura propuesta en Vaswani et al. (2017) incluye la

normalización de capas alrededor de las sub-capas del modelo y logra los rangos estandarizados en las entradas mediante la obtención de los embbedings de las secuencias de tokens de entrada. Para brindar claridad de este proceso, se puede utilizar como ejemplo el ingresar el texto "esto es un ejemplo" al modelo Transformer. Si se utiliza como token cada palabra, se tiene:

$$\text{Tokens} = [esto, es, un, ejemplo].$$

La obtención de los embeddings, corresponde a obtener las representaciones de cada palabra en un espacio vectorial de dimensión d_{model}. De esta forma, si denotamos al primer token $x^{(1)} = \text{esto}$, tenemos que:

$$\text{Embedding}(x^{(1)}) = [e_1^{(1)}, e_2^{(1)}, \ldots, e_{d_{model}}^{(1)}], \text{ con } 0 \leq e_i^{(1)} \leq 1 \text{ y } \sum_{i=1}^{d_{model}} e_i^{(1)} = 1 \tag{5-8}$$

y la secuencia de tokens se traduce en la matriz:

$$\begin{bmatrix} e_1^{(1)} & e_2^{(1)} & \ldots & e_{d_{model}}^{(1)} \\ e_1^{(2)} & e_2^{(2)} & \ldots & e_{d_{model}}^{(2)} \\ e_1^{(3)} & e_2^{(3)} & \ldots & e_{d_{model}}^{(3)} \\ e_1^{(4)} & e_2^{(4)} & \ldots & e_{d_{model}}^{(4)} \end{bmatrix}, \text{ con } 0 \leq e_i^{(t)} \leq 1 \text{ y } \sum_{i=1}^{d_{model}} e_i^{(t)} = 1 \text{ para } t = 1, 2, 3, 4.$$

De esta forma, si se tienen como entradas del modelo una lista de n frases que pueden separarse en tokens, las entradas corresponden a la matriz:

$$\boldsymbol{A} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(T_x)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(T_x)} \\ \vdots & \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \ldots & x_n^{(T_x)} \end{bmatrix},$$

y al obtenerse los embeddings para los tokens se obtiene un tensor, el cual es un objeto matemático que generaliza el concepto de matrices, de dimensión 2, a dimensiones superiores, $\boldsymbol{E} \in \mathbb{R}^{n \times T \times d_{model}}$ donde cada uno de sus elementos corresponde a la matriz de embeddings correspondiente a los tokens de cada una de las n frases.

En cuanto a las salidas del modelo, el Transformer se entrena relacionando cada frase convertida en una secuencia de tokens $(x^{(1)}, \ldots, x^{(T)})$ con su traducción $(y^{(1)}, \ldots, y^{(T_y)})$. Para ello,

las capas finales del modelo, denotadas "Linear" y "Softmax" en la arquitectura propuesta en Vaswani et al. (2017) toman las representaciones continuas generadas por el decodificador y las mapean a una secuencia de probabilidades para cada token posible en el vocabulario (por ejemplo las palabras del idioma). De esta forma, si llamamos N_T al número total de tokens posibles, el modelo mapea:

$$\begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(T)} \end{bmatrix} \longrightarrow \begin{bmatrix} p_1^{(1)} & p_1^{(2)} & \dots & p_1^{(T)} \\ p_2^{(1)} & p_2^{(2)} & \dots & p_2^{(T)} \\ \vdots & \vdots & \vdots & \vdots \\ p_{N_T}^{(1)} & p_{N_T}^{(2)} & \dots & p_{N_T}^{(T)} \end{bmatrix},$$

donde $p_i^{(t)}$ denota la probabilidad del i-ésimo token en la posición t de la secuencia.

Para obtener el token en cada posición se toma aquél con la máxima probabilidad cuando se realizan inferencias con el modelo y en el entrenamiento la función de pérdida utilizada es la cross-entropía categórica dispersa (Terven et al., 2023) que contempla esta forma de realizar la inferencia.

A diferencia de las aplicaciones típicas del modelo Transformer, para adaptar la arquitectura de este para el pronóstico de series de tiempo, se debe tener en cuenta la naturaleza numérica de los datos. De manera que este trabajo propone adaptar directamente la manera en que se pasan los datos de entrada y salida por ventanas de tiempo para lograr una estructura en los inputs análoga a las matrices de embeddings que se obtienen a partir de las secuencias de tokens en el modelo Transformer y una estructura de salidas del modelo que se pueda relacionar a estos. Suponga que se tiene un conjunto de p series de tiempo univariadas dado por $\mathbb{S} = \{\boldsymbol{y_i} \in \mathbf{R}^K, i = 1, \dots, p\}$. Si el objetivo es utilizar el modelo Transformer para pronosticar T_y periodos de tiempo dados los últimos T_x periodos, se pueden crear parejas de ventanas de datos de entrada y salida, recorriendo el conjunto de series de tiempo, particionando en $n = K - (T_x + T_y) + 1$ ventanas de tamaño $T_x + T_y$, donde los datos de entrada para cada ventana están dados por:

$$\boldsymbol{A_i} = \begin{bmatrix} y_1^{(i+1)} & y_2^{(i+1)} & \dots & y_p^{(i+1)} \\ y_1^{(i+2)} & y_2^{(i+2)} & \dots & y_p^{(i+2)} \\ \vdots & \vdots & \vdots & \vdots \\ y_1^{(i+T_x)} & y_2^{(i+T_x)} & \dots & y_p^{(i+T_x)} \end{bmatrix}, \tag{5-9}$$

y las salidas están dadas por:

$$\boldsymbol{Y_i} = \begin{bmatrix} y_1^{(i+T_x+1)} & y_2^{(i+T_x+1)} & \dots & y_p^{(i+T_x+1)} \\ y_1^{(i+T_x+2)} & y_2^{(i+T_x+2)} & \dots & y_p^{(i+T_x+2)} \\ \vdots & \vdots & \vdots & \vdots \\ y_1^{(i+T_x+T_y)} & y_2^{(i+T_x+T_y)} & \dots & y_p^{(i+T_x+T_y)} \end{bmatrix}, \tag{5-10}$$

con $i \in 0, 2, \dots n-1$.

Para lograr que las entradas se encuentren en el rango $[0, 1]$ se propone que cada serie de tiempo $\boldsymbol{y_i}$ se escale con la transformación min-máx dada por:

$$\text{MinMax}(\text{y}_\text{i}^{(\text{t})}) = \frac{y_i^{(t)} - \text{mín}\, \boldsymbol{y_i}}{\text{máx}\, \boldsymbol{y_i} - \text{mín}\, \boldsymbol{y_i}}, \tag{5-11}$$

y posteriormente se aplica una normalización de capa, que estandariza a través de las p variables de cada observación y beneficia el entrenamiento para redes recurrentes con tamaños de mini-lote pequeños (Ba et al., 2016), lo que resulta conveniente en este caso, dada la limitante en la cantidad de datos que se tiene usualmente en las series de tiempo. De la aplicación de estas transformaciones se tiene que las entradas del codificador, al pasar todas las ventanas de datos a la vez, están dadas por $\boldsymbol{A} \in \mathbb{R}^{b \times T_x \times d_{model}}$, con todos los elementos de las matrices de entradas $\boldsymbol{A_i}$, que conforman el tensor, en el rango $[0, 1]$, logrando un símil con el tensor $\boldsymbol{E}$ que resulta de los embeddings en el modelo del Transformer propuesto en Vaswani et al. (2017) al hacer $p = d_{model}$. De modo que en la arquitectura propuesta se elimina la capa de embeddings y se incluye la normalización de capa para entradas con la estructura descrita.

Por otro lado, para obtener las salidas del modelo como matrices $\widehat{\boldsymbol{Y_i}} \in \mathbb{R}^{T_y \times p}$ se elimina la capa **Softmax** del modelo descrito en Vaswani et al. (2017) y se utiliza una dimensión de $p = d_{model}$ para las salidas de la capa **lineal** en lugar de utilizar el número total del tokens. Dado que en este caso se trata de una salida de regresión en lugar de un problema de clasificación con los tokens, se utiliza el MSE como función de pérdida en lugar de la cross-entropía categórica dispersa.

5.2.2. Arquitectura propuesta

Los ajustes descritos en la sección anterior orientados al manejo de los datos de series de tiempo de conteos (y aplicables a series de tiempo en general) respetan la arquitectura

general del modelo Transformer y su funcionamiento basado en mecanismos de atención, sin embargo, se generan cambios en las entradas y salidas. La arquitectura propuesta con dichos ajustes se puede apreciar en la Figura **5-5**.

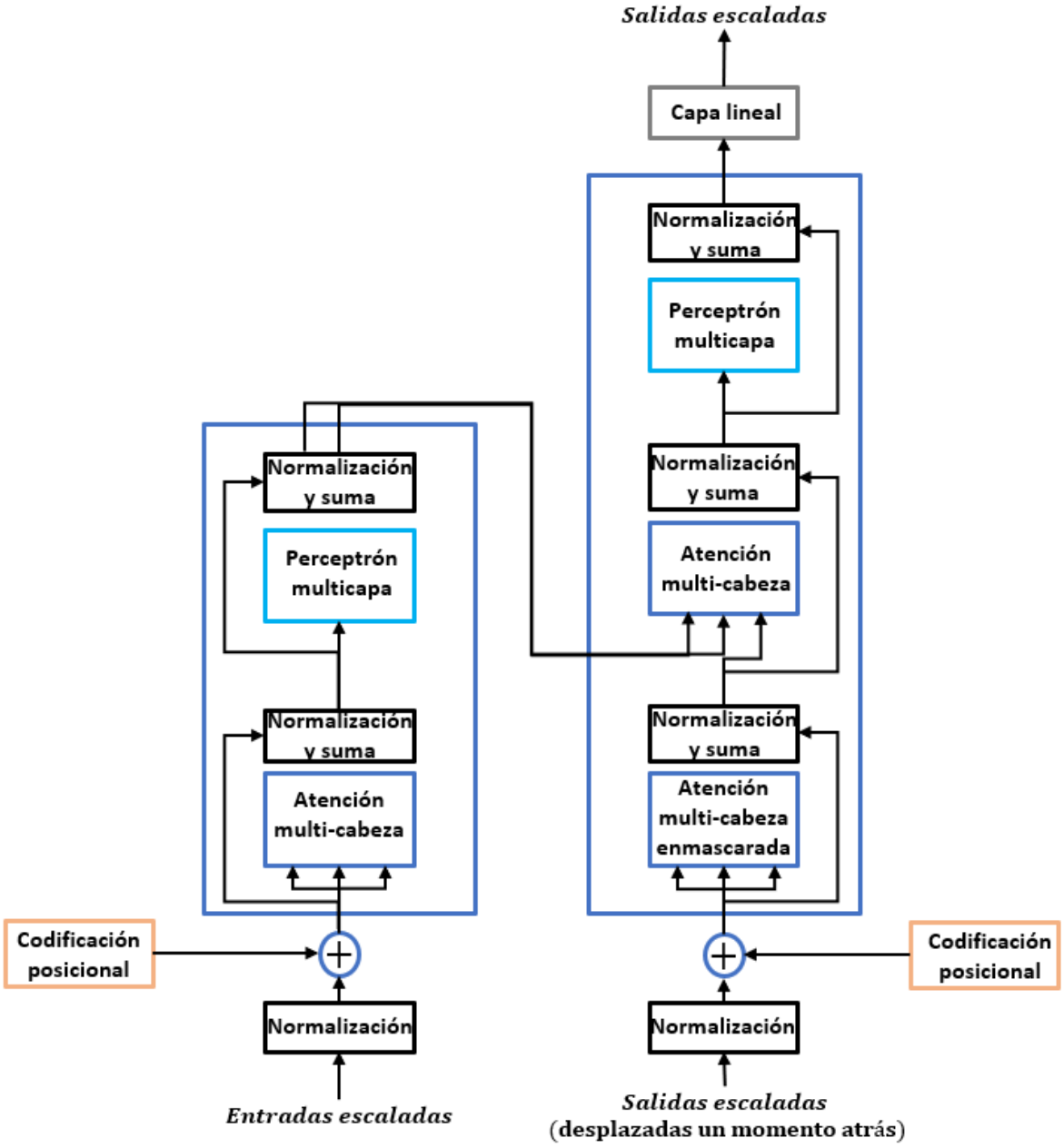

Figura 5-5.: Arquitectura adaptada del modelo Transformer para series de tiempo. Fuente: Elaboración propia.

5.2.3. Implementación en Python

Esta sección retoma el ejemplo de las muertes en accidentes de tránsito en Gran Bretaña, utilizado como segundo caso en las implementaciones de los capítulos pasados, con el objetivo de observar la efectividad para el ajuste de series de tiempo de la arquitectura ajustada del Transformer. Dado el diseño del modelo Transformer para salidas multivariadas (aunque pueda ajustarse el número de unidades de la capa lineal de salida para ajustar la dimensión del output) y el uso como modelo de pronóstico global que tendrá en la comparación entre métodos de pronóstico del próximo capítulo, en esta Sección se realizará el pronóstico paralelo de todas las covariables del conjunto de datos, pero solo se observarán los resultados de la serie del número de muertes en accidentes de tránsito cómo se hizo en los capítulos anteriores. El ajuste utilizando la serie de infecciones de campylobacter se omite en este capítulo dada la naturaleza univariada de la serie.

En el repositorio que acompaña este trabajo (disponible en la URL https://github.com/dbeta95/On_time_series_of_counts_forecast), donde se encuentran los conjuntos de datos, se encuentra el código utilizado para la manipulación de los datos para lograr las ventanas descritas en la propuesta de arquitectura del modelo, la definición de todas las sub-capas, capas y modelo del Transformer propuesto y la implementación cuyos resultados se exponen en esta sección.

Si bien en este caso se utilizan el mismo conjunto de variables de entrada definido en la implementación del capítulo anterior, en este caso en lugar de tener una única variable como salida del modelo, se tiene la estructura de entradas y salidas presentada en (5-9) y (5-10), haciendo $K = 192$ observaciones para cada serie de tiempo en el conjunto de datos, $n = 180$ ventanas de entradas y salidas, $T_x = 12$ periodos como insumo para pronosticar $T_y = 1$ periodo de todas las covariables. La notación en este caso pasa a ser:

- $y_1^{(t)}$: Número de conductores muertos durante el mes t.
- $y_2^{(t)}$: Número de conductores muertos o heridos de gravedad durante el mes t.
- $y_3^{(t)}$: Número de pasajeros del asiento delantero muertos o heridos de gravedad en el mes t.
- $y_4^{(t)}$: Número de pasajeros de los asientos traseros, muertos o heridos de gravedad en el mes t.
- $y_5^{(t)}$: Distancia conducida en el mes t.
- $y_6^{(t)}$: Precio del petróleo en el mes t.

- $y_7^{(t)}$: Conductores de vehículos ligeros de bienes muertos en el mes t.
- $y_8^{(t)}$: Indicador de si había entrado o no en vigencia la ley de uso obligatorio del cinturón de seguridad en el mes t.
- $y_9^{(t)}$: Año al que corresponde la observación t.
- $y_{10}^{(t)}$: Mes al que corresponde la observación t.

La Figura **5-6** presenta los valores ajustados por el modelo contra los valores reales y la Figura **5-7** presenta los gráficos para el análisis de residuos del ajuste, calculados según (3-48). Si bien el modelo parece seguir el nivel medio de la serie y aproximar el comportamiento periódico de la serie, se equivoca en el ajuste de los valores máximo y mínimo de cada periodo estacional. Por otro lado, en la Figura **5-7** se observa que los residuales no parecen seguir ningún patrón y se mueven al rededor del cero con varianza homogénea, lo que sugiere un buen ajuste del modelo, en la Figura **5-7**, que presenta el análisis de autocorrelaciones, se observa un patrón de cola que habla de correlación entre los residuos del ajuste. sin embargo, los métodos de ajuste no paramétrico generalmente violan el supuesto de independencia en errores de ajuste, lo que no es muy preocupante en términos prácticos si las correlaciones no son muy grandes, como pasa en este caso.

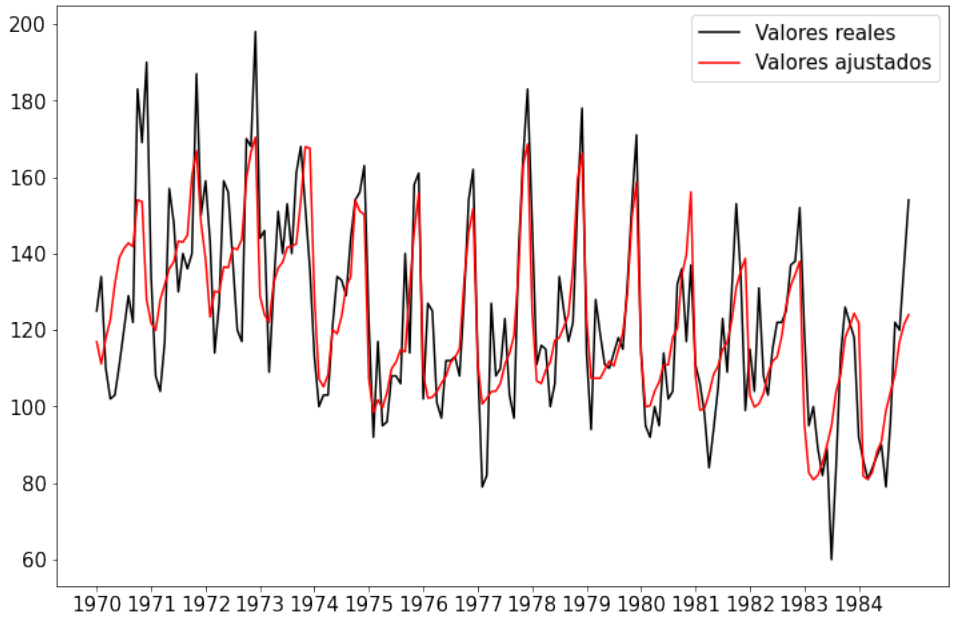

Figura 5-6.: Gráfico de valores reales contra ajustados por el modelo Transformer.

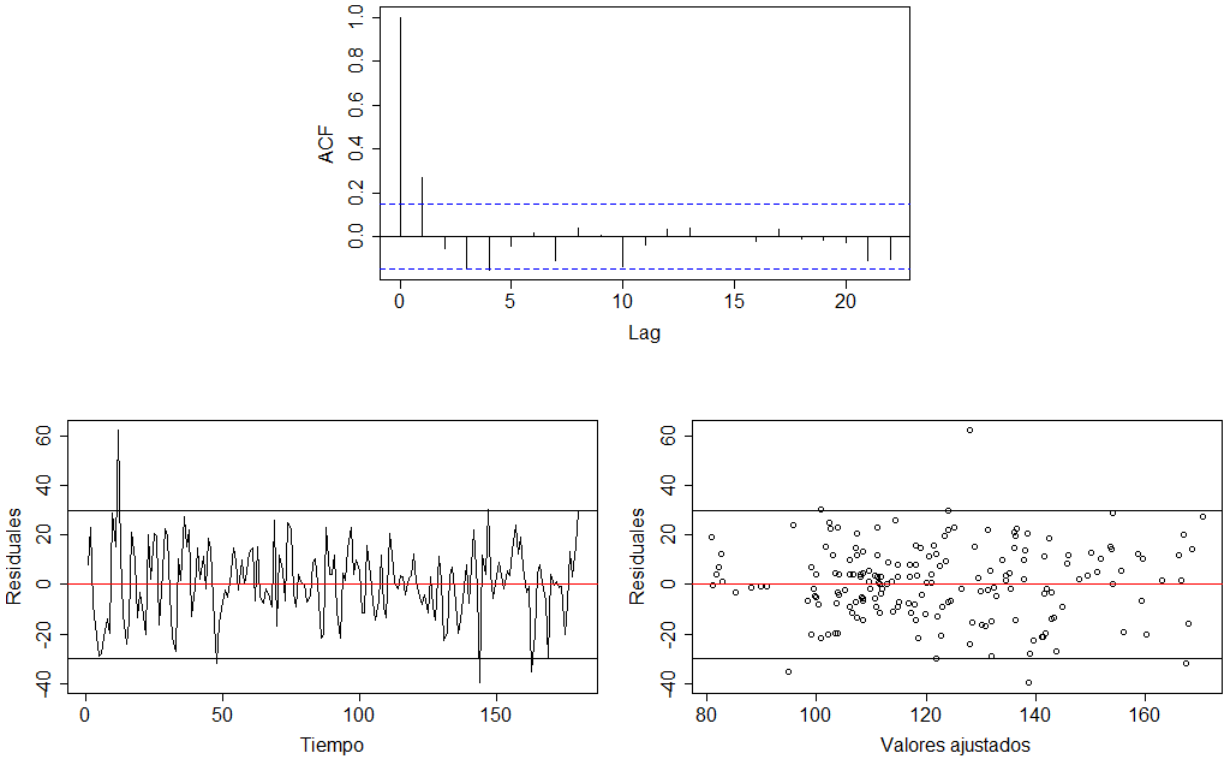

Figura 5-7.: Gráficos para el análisis de supuestos de los errores.

6. Comparación de metodologías de pronóstico

Este capítulo presenta un estudio de la precisión del pronóstico de series de tiempo de conteos mediante simulación, para abordar el problema de entender las circunstancias en las cuales las aproximaciones globales o locales y los diferentes tipos de modelo funcionan, y cuál es la complejidad ideal del modelo para conjuntos de datos de ciertos tamaños y particularidades.

Si bien los hallazgos de Makridakis et al. (2020) sugieren que en el presente caso se podrían lograr mejores resultados con metodologías híbridas, tal como mencionan los autores, la aplicación de los modelos globales utilizando machine learning está abierta a mejoras, lo que sugiere que una exploración más profunda de los modelos puede servir como base para el trabajo a seguir en el caso de las series de tiempo de conteos. Así, el presente trabajo se centra en la exploración de algunas metodologías estadísticas basadas en algoritmos autorregresivos y adecuadas para el pronóstico de series de conteo, incluyendo estrategias propias del machine learning en el ajuste de parámetros, y en metodologías de machine learning adecuadas para el manejo de datos secuenciales, con el objetivo de obtener resultados que sirvan como punto de partida para la exploración futura de ajustes sobre los métodos aplicados y de aproximaciones híbridas al problema.

Por lo anterior, se realizan pronósticos utilizando la aproximación local y global, utilizando las metodologías planteadas en los Capítulos anteriores, con el objetivo de obtener hallazgos sobre cuáles son los escenarios en que los distintos modelos tienen un mejor desempeño para el pronóstico de múltiples series de tiempo de conteos y analizar las implicaciones prácticas del pronóstico bajo estas circunstancias mediante las diferentes metodologías. Se tendrá en cuenta que en los casos de aplicación en los cuales el volumen de datos puede aumentar significativamente, y se den escenarios que requieran el análisis conjunto de un gran número de series de tiempo, resulta poco práctico el análisis individual, y en su lugar, se hacen necesarios algoritmos de selección automática de parámetros basados en datos para los modelos (Montero-Manso & Hyndman, 2021). Otro elemento a contemplar a nivel práctico, son los tiempos computacionales para el ajuste de modelos.

A continuación, se presenta inicialmente la metodología del estudio de comparación mediante simulación, incluyendo los escenarios a probar, la generación de datos y las métricas de

comparación, y posteriormente los resultados obtenidos y las conclusiones alcanzadas a partir de estos.

6.1. Metodología experimental

El presente estudio de simulación consta de tres etapas fundamentales para la comparación entre metodologías de pronóstico de múltiples series de tiempo de conteos:

1. Generación de datos mediante simulación.
2. Ajuste de modelos utilizando conjuntos de entrenamiento.
3. Obtención de métricas de desempeño de los pronósticos.

Estas etapas se repiten para diferentes casos experimentales donde se controla la complejidad de la relación autorregresiva, la heterogeneidad (o relación) entre las series, el número de series de tiempo y la longitud de las series en el conjunto de datos. Estos elementos son análogos a los controlados en Hewamalage et al. (2022) en un estudio de simulación con fines similares. A continuación se profundiza sobre los elementos expuestos.

6.1.1. Generación de datos

Cómo se mencionó anteriormente, en el estudio se controlaron cuatro elementos fundamentales en la generación de los datos, a partir de los cuales se generan los diferentes casos para la simulación, entrenamiento de modelos y obtención de métricas de precisión del pronóstico.

Complejidad de la relación autorregresiva y heterogeneidad

Los primeros dos factores se abordaron como un único elemento, definiendo dos escenarios de generación de datos. El primero de ellos, genera de forma independiente las series a colocar dentro de los grupos, a partir de modelos autorregresivos Poisson, siguiendo la idea hallada comúnmente en la literatura revisada (por ejemplo, en Hewamalage et al., 2022) de simular datos temporales desde modelos autorregresivos. El segundo escenario, genera series de tiempo estacionales con características similares a algunas series de tiempo encontradas en contextos prácticos de la economía e industria, tales como aquellas usadas en los benchmarks realizados en la evaluación de métodos de pronóstico como la competencia M4 (Makridakis et al., 2020), y además relacionadas entre sí. La selección del primer escenario se dio con el

objetivo de evaluar que tal es el desempeño de los diferentes modelos cuando se cumplen los supuestos del modelo de Autorregresión Poisson (ver Capítulo 3), con órdenes de autocorrelación generados aleatoriamente en cada serie y bajo independencia entre éstas, esto último conduce a escenarios en los cuales no es posible obtener información para el pronóstico de cada serie a partir de las demás, lo cual resulta complejo para el ajuste con los modelos de machine learning. En cuanto al segundo, se seleccionó con el objetivo de tener casos en los cuales los patrones entre las series ofrecen información para el pronóstico conjunto y están basados en datos de una naturaleza similar a los observados en la industria, por lo que no se parte del cumplimiento de supuestos del modelo de autorregresión Poisson.

Con base en las ideas vistas en la literatura de generación de series de tiempo a partir de modelos autorregresivos, y ajustando la estrategia para generar datos acordes a los supuestos distribucionales en que se enfoca este trabajo de tesis, se diseña el siguiente algoritmo para la generación de las series bajo el primer tipo de escenario, descrito utilizando la notación para el modelo de autorregresión Poisson log-lineal del Capítulo 3, donde K es el número de series a generar en el conjunto de datos, n la longitud total de cada serie de tiempo y R el máximo para los órdenes p y q del modelo.

Repetir K veces:

Obtener los órdenes de autorregresión p y q de distribuciones uniformes discretas $U\{1, R\}$

Obtener los p coeficientes sin escalar a_i^* de distribuciones uniformes continuas $U(-1, 1)$

Obtener los q coeficientes sin escalar b_j^* de distribuciones uniformes continuas $U(-1, 1)$

Normalizar los coeficientes haciendo

$$a_i = \frac{a_i^*}{\sum_{i=1}^{p} |a_i^*| + \sum_{j=1}^{q} |b_j^*|}, \text{ con } i = 1, 2, \ldots, p,$$

$$b_i = \frac{b_i^*}{\sum_{i=1}^{p} |a_i^*| + \sum_{j=1}^{q} |b_j^*|}, \text{ con } j = 1, 2, \ldots, q$$

para garantizar la estacionariedad de las series generadas.

Obtener él intercepto d de una distribución uniforme continua $U(0, 5)$

Inicializar la media del proceso λ_{ini} de una distribución uniforme discreta $U\{0, 10\}$

Generar la varianza del ruido σ^2 de una distribución uniforme continua $U(0, 1)$

Generar p valores iniciales para $\lambda^{(t)} \sim Poisson(\lambda_{ini})$ para $t = -(p-1), -(p-2), \ldots, 0$

Obtener $\nu^{(t)} = \log(\lambda^{(t)})$ para $t = -(p-1), -(p-2), \ldots, 0$

Generar q valores iniciales para $y^{(t)} \sim Poisson(\lambda_{ini})$ para $t = -(q-1), -(q-2), \ldots, 0$
Para t entre 1 y n:

$\nu^{(t)} = d + \sum_{i=1}^{p} a_i \nu^{(t-i)} + \sum_{j=1}^{q} b_j \log(Y^{(t-j)} + 1) + e^{(t)}, \quad e^{(t)} \sim N(0, \sigma^2)$

$\lambda^{(t)} = \exp(\nu^{(t)})$

Obtener $y^{(t)} \sim Poisson(\lambda^{(t)})$

Devolver $\boldsymbol{y} = [y^{(1)}, y^{(2)}, \ldots, y^{(n)}]$.

El código utilizado para la generación de datos utilizando dicho algoritmo se encuentra disponible en el repositorio que acompaña este trabajo accesible en https://github.com/dbeta95/On_time_series_of_counts_forecast en el módulo *DataSimulation* y es utilizado en los diferentes casos de generación de datos en los notebooks en los que se ejecutan los experimentos.

Por otro lado, para la generación de los datos del segundo escenario se utilizó la aproximación a la generación de series de tiempo GRATIS propuesta por Kang et al. (2020), cuyo nombre proviene del titulo de dicho trabajo: "GeneRAting TIme Series wiht diverse and controllable characteristics", la cual es una aproximación a la generación de series de tiempo basada en mezclas Gaussianas autorregresivas (MAR) para generar un amplio rango de series de tiempo no Gaussianas y no lineales. Una ventaja de dicha metodología es que permite establecer características objetivo para las series de tiempo a generar, entendiendo dichas características como funciones que se calculan a partir de la serie de tiempo y que proveen información útil sobre la naturaleza de ésta. Algunas características estudiadas por los autores son la longitud, los periodos estacionales, la entropía, la linealidad, la estacionariedad, entre otras.

La aproximación GRATIS permite simular series de tiempo apuntando a un espacio específico de características, para ello, utiliza algoritmos genéticos para ajustar los parámetros del modelo MAR hasta lograr que la distancia entre el vector de características objetivo y el vector de características de una serie de tiempo simulada sea cercana a cero (Kang et al., 2020). Este trabajo calcula las características de las series de tiempo utilizando el paquete **tsfeatures** (Hyndman et al., 2023) en **R** (R Core Team, 2023) y se agregan el lambda de BoxCox y la entropía para construir el vector de características objetivo de las series de tiempo para utilizar en la generación de datos. Este proceso se ejecutó siguiendo la metodología presentada por Hyndman (2021) en una charla para la *Australian Data Science Network*.

Como series de tiempo de referencia se toman la serie mensual del total de conductores muertos en Gran Bretaña entre enero de 1969 y diciembre de 1984 con un total de 192

observaciones, incluido en **R** (R Core Team, 2023), y la serie horaria "H381" del conjunto de datos de la competencia M4 (Makridakis et al., 2020), seleccionada aleatoriamente entre aquellas series de periodicidad horaria en el set de datos con máxima longitud (961 observaciones), con valores de naturaleza discreta y ocurrencia del 10 como valor mínimo (el cual correspondía al cero acorde a los ajustes realizados por los organizadores de la competencia). Una vez simuladas las series de tiempo se aplicó un desplazamiento que garantizara un porcentaje aleatorio entre 5 % y 10 % de observaciones menores o iguales a cero y las convertía en cero, además de aproximar al entero más cercano los valores de la serie, para lograr series de conteos, desde que la generación por parte del algoritmo GRATIS es en los reales.

En el repositorio que acompaña este trabajo accesible en https://github.com/dbeta95/On_time_series_of_counts_forecast están disponibles los códigos en **R** (R Core Team, 2023) utilizados para la generación de datos, así como los conjuntos generados y la función de procesamiento, disponible en el módulo *DataSimulation*.

Número de series de tiempo

En este trabajo se establecieron dos magnitudes para el conjunto de series de tiempo: 10 y 100 series de tiempo respectivamente. La selección se dio basada en el número máximo de series de tiempo presentado en Hewamalage et al. (2022) de 100 series para los diferentes escenarios. Sin embargo, para evaluar el efecto de una reducción significativa del tamaño de los grupos de series sobre el desempeño de los modelos de pronósticos locales y globales, se decidió considerar un valor mínimo de 10 para el tamaño de los grupos.

Longitud de las series de tiempo en el conjunto de datos

Como se mencionó anteriormente, la generación de datos mediante el algoritmo GRATIS tomó como referencia observaciones de dos series de tiempo de conteos, una de naturaleza mensual y con un periodo estacional de 12 y la otra con una naturaleza horaria y un periodo estacional de 24. Dichas series tienen una longitud de 192 y 961 observaciones respectivamente, por lo que se decidió generar longitudes similares de 200 y 1000 observaciones, para los datos simulados.

Casos experimentales

La combinación de los tres factores descritos lleva a los siguientes casos de datos generados para la experimentación:

- **Caso 1:** Series de tiempo independientes generadas como un proceso autorregresivo Poisson con un orden de autocorrelación máximo de 12. Un total de 10 series de tiempo en el conjunto de datos, cada una con una longitud de 200.
- **Caso 2:** Series de tiempo independientes generadas como un proceso autorregresivo Poisson con un orden de autocorrelación máximo de 12. Un total de 100 series de tiempo en el conjunto de datos, cada una con una longitud de 200.
- **Caso 3:** Series de tiempo independientes generadas como un proceso autorregresivo Poisson con un orden de autocorrelación máximo de 24. Un total de 10 series de tiempo en el conjunto de datos, cada una con una longitud de 1000.
- **Caso 4:** Series de tiempo independientes generadas como un proceso autorregresivo Poisson con un orden de autocorrelación máximo de 24. Un total de 100 series de tiempo en el conjunto de datos, cada una con una longitud de 1000.
- **Caso 5:** Series de tiempo generadas con el método GRATIS con un conjunto único de características objetivo de naturaleza mensual y periodo estacional 12. Un total de 10 series de tiempo en el conjunto de datos, cada una con una longitud de 200.
- **Caso 6:** Series de tiempo generadas con el método GRATIS con un conjunto único de características objetivo de naturaleza mensual y periodo estacional 12. Un total de 100 series de tiempo en el conjunto de datos, cada una con una longitud de 200.
- **Caso 7:** Series de tiempo generadas con el método GRATIS con un conjunto único de características objetivo de naturaleza horaria y periodo estacional 24. Un total de 10 series de tiempo en el conjunto de datos, cada una con una longitud de 1000.
- **Caso 8:** Series de tiempo generadas con el método GRATIS con un conjunto único de características objetivo de naturaleza horaria y periodo estacional 24. Un total de 100 series de tiempo en el conjunto de datos, cada una con una longitud de 1000.

Cabe resaltar que la decisión de tomar como un único elemento la independencia y el algoritmo generador de los datos y utilizar el periodo estacional 12 únicamente para las series de 200 observaciones y el periodo 24 para las de 1000 observaciones, en lugar de generar ambos casos para cada longitud, fue orientada por las restricciones computacionales derivadas tiempo de entrenamiento, ya que como se evidenciará en los resultados respecto a la eficiencia computacional, para algunos casos el tiempo de ajuste de todos los modelos en total superó las 24 horas, de modo que si se generaban todos los 128 casos posibles considerando independencia, algoritmo generador por separado, los periodos estacionales, longitud de las series, el número de series y la forma de dividir los datos en entrenamiento y evaluación (como se ve en la siguiente Subsección), hubiera sido imposible la realización de este estudio

de simulación dentro del cronograma establecido para este trabajo de tesis. Así, se buscó acotar el número de casos a uno que permitiera realizar los ajustes e iterar y corregir de ser necesario.

6.1.2. Ajuste de modelos

Para comparar el pronóstico se ajustaron cuatro tipos modelos para cada caso, los modelos presentados en este trabajo: autorregresivos Poisson (ver Capítulo 3), modelos de pronóstico de series de tiempo con RNN (ver Capítulo 4) y modelos de pronóstico de series de tiempo con una arquitectura basada en el Transformer (ver Capítulo 5), y modelos SARIMA que se toman como referencia pese al incumplimiento de supuestos, dado el amplio uso que tienen este tipo de modelos. En todos los casos se utilizaron rutinas programadas en **Python** que seleccionarán los parámetros de los modelos de forma automática a partir de los datos y que se corren de forma vectorial para garantizar eficiencia computacional, de manera que se puedan realizar conclusiones respecto a los tiempos de corrida de los algoritmos.

Se implementó la estrategia de ajuste con validación cruzada así: en series de tiempo mensuales de longitud 200 se utilizaron 188 observaciones en el ajuste y un único periodo estacional de 12 observaciones para evaluar la calidad de predicción, en tanto que en las series horarias, de longitud 1000, se usaron 904 observaciones en el ajuste, dejando 96 observaciones (cuatro periodos estacionales) para evaluar los pronósticos. La sub-partición de los datos usados para el ajuste entre entrenamiento y validación varió según la forma de realizar la validación cruzada en cada modelo. A continuación se detalla el procedimiento seguido para el ajuste en cada caso.

Modelos SARIMA

Este trabajó utilizó la implementación de los modelos ARIMA$(p, d, q)(P, D, Q)[s]$ disponible en el paquete **pmdarima** (Smith, 2017) en **Python**, el cual cuenta con un algoritmo de selección automática de los órdenes autorregresivos y de la media móvil para el modelo, tanto en su parte regular como estacional. En el repositorio de este trabajo dentro del módulo SARIMA se puede encontrar el código para la generación de un objeto que permite ajustar modelos independientes para conjuntos de series de tiempo y generar pronósticos de todos los modelos en paralelo. Para el ajuste se definen los órdenes autorregresivos, p, y de media móvil, q, máximos para la parte regular y si el algoritmo debe considerar que las series tienen una naturaleza estacional, se deja que el algoritmo seleccione libremente los órdenes de la parte estacional. En los casos de las series de longitud 200 se tomaron p y q máximos de 12 y en las de longitud 1000 un máximo de 24. Además, para las series generadas como procesos autorregresivos Poisson no se tomaron modelos estacionales, pero sí para las generadas con el método GRATIS.

Modelos autorregresivos Poisson

El detalle de estos modelos, también llamados INGARCH, la optimización para obtener los valores de los parámetros del modelo y el algoritmo de selección automática del orden autorregresivo, así como la implementación construida en **Python** se presentó en el Capítulo 3 de este trabajo. Para el ajuste en paralelo de cada una de las series de tiempo integrantes de un conjunto dado se creó un objeto en **Python** disponible en el repositorio que acompaña este trabajo, el cual permite la aplicación del algoritmo de selección automática de los órdenes en el modelo INGARCH para cada serie de tiempo, el entrenamiento del modelo con los órdenes seleccionados y el pronóstico paralelo de todos los modelos. De forma similar al caso de los modelos SARIMA, en los casos de las series de longitud 200 se tomaron p y q máximos de 12 y en las de longitud 1000, con órdenes máximos de 24.

Modelos de pronóstico de series de tiempo con RNN

Las redes neuronales y redes neuronales recurrentes (RNN), así como la arquitectura del modelo para el pronóstico de series de tiempo, como se plantea para este trabajo, se abordaron en el Capítulo 4. En las aplicaciones vistas en dicho capítulo, se vio la tendencia al sobre ajuste de estos modelos, y la necesidad de tomar medidas para ello en el entrenamiento.

A partir de lo mencionado, el ajuste de las RNN aborda modelos de pronóstico de ventanas temporales que pronostican de forma global todas las series del conjunto de datos y se generó un objeto de selección automática de hiperparámetros y de la arquitectura del modelo. Las arquitecturas consideradas fueron:

- **Tipo 1:** Una única capa recurrente, de celdas LSTM, seguida de una capa densa (capas completamente conectadas del perceptrón multicapa) y pronósticos en cada momento t utilizando la información del estado latente $h^{(t)}$ en dicho momento, de forma que las ventanas de entrada y salida tienen la misma longitud.

- **Tipo 2:** Dos capas recurrentes, de celdas LSTM, seguidas de una capa densa y pronósticos en cada momento t utilizando la información del estado latente $h^{(t)}$ en dicho momento, de forma que las ventanas de entrada y salida tienen la misma longitud.

- **Tipo 3:** Una única capa recurrente, de celdas LSTM, seguida de una capa densa y todos los pronósticos generados utilizando la información del estado latente $h^{(T_x)}$ en el último paso de tiempo de las entradas, de forma que las ventanas de entrada y salida pueden tener diferente longitud, y se utilizan ventanas de entrada con una longitud igual al doble de las ventanas de salida.

- **Tipo 4:** Dos capas recurrentes, de celdas LSTM, seguidas de una capa densa y todos los pronósticos generados utilizando la información del estado latente $h^{(T_x)}$ en el último paso de tiempo de las entradas, de forma que las ventanas de entrada y salida pueden tener diferente longitud, y se utilizan ventanas de entrada con una longitud igual al doble de las ventanas de salida.

Por otro lado, en cuanto a los hiperparámetros se variaron en los siguientes valores:

- **Unidades en la capa recurrente:** 16, 32 ó 64.
- **Tasa de aprendizaje en el descenso gradiente:** 0.01, 0.001 ó 0.0001.
- **Tamaño de lote para el descenso gradiente por mini-lotes:** 8 (únicamente para las series de longitud 200), 32, 64 ó 128 (únicamente para las series de longitud 1000).

A partir de la combinación de las arquitecturas se generaron 108 modelos para cada caso, los cuales se entrenaron por 500 épocas sobre conjuntos de entrenamiento correspondientes al 80 % de observaciones utilizadas para el ajuste y se evaluaban sobre el conjunto de validación con el 20 % restante. Luego, en cada modelo se seleccionaba el mínimo valor de la pérdida sobre el conjunto de validación, a través de los registrados en todas las épocas, lo que permitió controlar el sobre ajuste. Finalmente, el modelo seleccionado es aquel con la mejor métrica de validación mínima, de los 108 evaluados, y corresponde a la versión entrenada con el número de épocas para los cuales el modelo tenía los parámetros con los cuales alcanzó este mínimo valor.

Modelos de pronóstico de series de tiempo con arquitectura basada en transformers

En el Capítulo 5 de este trabajo se presentó el modelo Transformer, una propuesta de arquitectura ajustada para el pronóstico de series de tiempo y su implementación. Para el ajuste de los modelos basados en dicha propuesta se siguió una lógica similar a la mencionada para las RNN, con la salvedad de que la arquitectura a probar en este caso es única. En la selección automática de hiperparámetros se probaron:

- **Número de capas de codificador y decodificador:** 2.
- **Número de cabezas de atención:** 4 ó 6.
- **Unidades en la capa densa final:** 8, 16 ó 32.
- **Tasa de dropout:** 0.1 ó 0.3.

- **Tamaño de lote para el descenso gradiente por mini-lotes:** 8, 16 ó 32.

Donde la tasa *dropout* corresponde al porcentaje de nodos (y sus respectivas conexiones) que se apagan de forma aleatoria durante el entrenamiento de la red, lo que previene la co-adaptación excesiva de los nodos, estrategia que se utiliza para evitar el sobre ajuste del modelo (Srivastava et al., 2014). En consecuencia, se probó un total de 72 modelos, en cada caso entrenando por 500 épocas y nuevamente seleccionando el mejor a partir de la métrica de la pérdida en el conjunto de validación.

6.1.3. Métricas de desempeño de pronóstico

En este trabajo se calculan el sMAPE (Makridakis, 1993) y el MASE (Hyndman & Koehler, 2006) como medidas de desempeño de pronóstico, tal como se presentaron en (2-19) y (2-20) para el pronóstico obtenido con cada modelo en cada serie de tiempo y se reportan el mínimo, máximo, promedio y desviación estándar para ambas medidas en cada conjunto de datos, es decir sobre todas las series de tiempo generadas para cada caso. Sin embargo, de acuerdo con Hyndman y Koehler (2006) las medidas basadas en errores porcentuales, como el sMAPE, son sesgadas y se recomienda mejor el uso del MASE. Por lo tanto, los análisis en torno al desempeño de los modelos para el pronóstico se realizarán únicamente con base en el MASE y los valores del sMAPE solo son reportados como información de referencia sobre la magnitud del error de predicción respecto a los valores de la serie de tiempo.

6.2. Comparación del desempeño de los modelos

Cómo se mencionó anteriormente, la comparación del desempeño de los modelos para el pronóstico se realiza usando el MASE, métrica que puede interpretarse como el error del modelo en relación con el error en que incurre el pronóstico "ingenuo" de tomar como pronóstico para cada el valor el inmediatamente anterior (Hyndman & Koehler, 2006). En la tabla **6-1** se presentan el promedio, la desviación estándar y el valor mínimo y máximo de las métricas obtenidas con cada modelo en los diferentes escenarios. El código utilizado para la obtención de estos valores, así como los valores de las métricas guardados como archivos *.json*, se pueden acceder en el repositorio que acompaña este trabajo disponible en https://github.com/dbeta95/On_time_series_of_counts_forecast.

Tabla 6-1.: Resumen estadístico de las métricas de pronóstico de los modelos en los diferentes escenarios.

		SARIMA		**INGARCH**		**RNN**		**Transformer**	
		sMAPE	MASE	sMAPE	MASE	sMAPE	MASE	sMAPE	MASE
case_1	**mean**	56.1827	0.7229	54.3988	0.6597	54.8128	0.6773	53.0521	0.6338
	sd	26.1570	0.2439	27.2834	0.1209	27.4535	0.1478	26.5891	0.1213
	min	14.5745	0.4076	18.6374	0.3863	20.4078	0.3796	18.5374	0.4075
	max	97.3605	1.3101	104.4552	0.8219	103.5318	0.8777	98.0667	0.7892
case_2	**mean**	61.3135	0.7937	61.5685	0.8276	58.7354	0.7298	59.3083	0.7374
	sd	32.2182	0.2982	33.8748	0.6309	31.2012	0.2127	31.4378	0.2182
	min	7.4553	0.3100	8.9668	0.3263	9.2292	0.3599	9.0952	0.3480
	max	148.1599	2.4662	156.3989	6.4920	149.5699	1.5696	147.2267	1.5781
case_3	**mean**	60.6527	0.7277	59.8804	0.7363	60.4105	0.7256	60.8379	0.7293
	sd	20.3118	0.0716	20.9177	0.0852	20.0800	0.0689	20.4042	0.0783
	min	22.2805	0.6416	22.1607	0.6228	22.7956	0.6373	22.1346	0.6245
	max	89.0408	0.8892	89.5387	0.8708	89.8535	0.8583	89.4845	0.8897
case_4	**mean**	59.5166	0.7599	57.9241	0.7804	57.6525	0.7274	57.7086	0.7267
	sd	25.9861	0.2572	26.1820	0.3363	24.5581	0.0963	24.3781	0.0893
	min	3.6164	0.3635	7.1540	0.5249	3.7128	0.5236	3.6034	0.5339
	max	124.2347	3.1425	138.3450	3.6145	118.7633	1.2203	109.8548	1.2399
case_5	**mean**	106.2827	1.3450	84.1154	0.5564	89.5761	0.6081	85.8889	0.8451
	sd	30.7537	0.7159	20.2122	0.4556	19.8508	0.1552	12.4394	0.1490
	min	64.7055	0.5045	59.6577	0.1164	61.2262	0.3432	66.9535	0.4805
	max	150.0000	2.5593	123.9727	1.5068	129.8362	0.8517	104.4420	1.0619
case_6	**mean**	101.1775	1.3130	82.7722	0.5962	89.4558	0.8263	87.8465	0.7916
	sd	32.8984	0.8150	23.7307	0.6496	20.5163	0.3591	18.9369	0.3540
	min	45.5837	0.0494	36.8266	0.0597	54.2238	0.3623	46.0128	0.2966
	max	200.0000	5.2741	132.7777	4.5110	160.2319	2.5124	142.0779	2.3686
case_7	**mean**	73.3482	1.8753	104.0152	1.3741	74.8386	1.5356	82.8930	2.0026
	sd	32.4801	1.1787	49.7122	1.2119	21.1346	0.7236	25.6233	0.6881
	min	31.8030	0.2036	50.2502	0.1637	51.6456	0.3276	50.2687	1.2030
	max	141.9237	4.6363	176.8909	3.3597	107.9169	2.7241	116.8467	3.5181
case_8	**mean**	81.1212	1.7173	139.8679	2.5205	72.9551	1.5682	75.4747	1.5626
	sd	40.7117	1.2707	42.1129	2.0115	23.1517	0.8806	23.8968	0.8549
	min	27.2761	0.0837	32.0528	0.5880	36.4421	0.4329	33.3453	0.3213
	max	200.0000	8.8253	195.6578	16.3600	141.0525	6.3820	153.6907	5.7436

Un hallazgo durante la obtención de los pronósticos para el cálculo de las métricas es que para el caso de los pronósticos de más largo plazo, es decir, los pronósticos de 4 periodos estacionales horarios de 24 horas, el modelo de autorregresión Poisson generó divergencias computacionales e incluso falló en obtener el pronóstico para dos de los modelos en el caso 4, dónde se presentaron casos de divergencia por valores muy altos en los gradientes durante el descenso gradiente, que llevó a la función de pérdida a fallos computacionales y generó valores nulos para todos los parámetros. Si bien estos problemas podrían solucionarse con la selección manual de órdenes de autorregresión y el testeo hasta lograr resultados más convenientes, esto implicaría un análisis particular que imposibilita la escalabilidad del uso del modelo. Por ello, para las métricas reportadas y a comparar se debe tener en cuenta que para el modelo de autorregresión Poisson se omitieron los resultados obtenidos para uno de los pronósticos del caso 3, cuatro del caso 4 y uno del caso 7. En la Figura **6-2** se observa el caso de la divergencia numérica en el pronóstico en el caso 3 y en la Figura **6-1** tanto uno de los pronósticos divergentes como una de las series sin pronósticos para el caso 4.

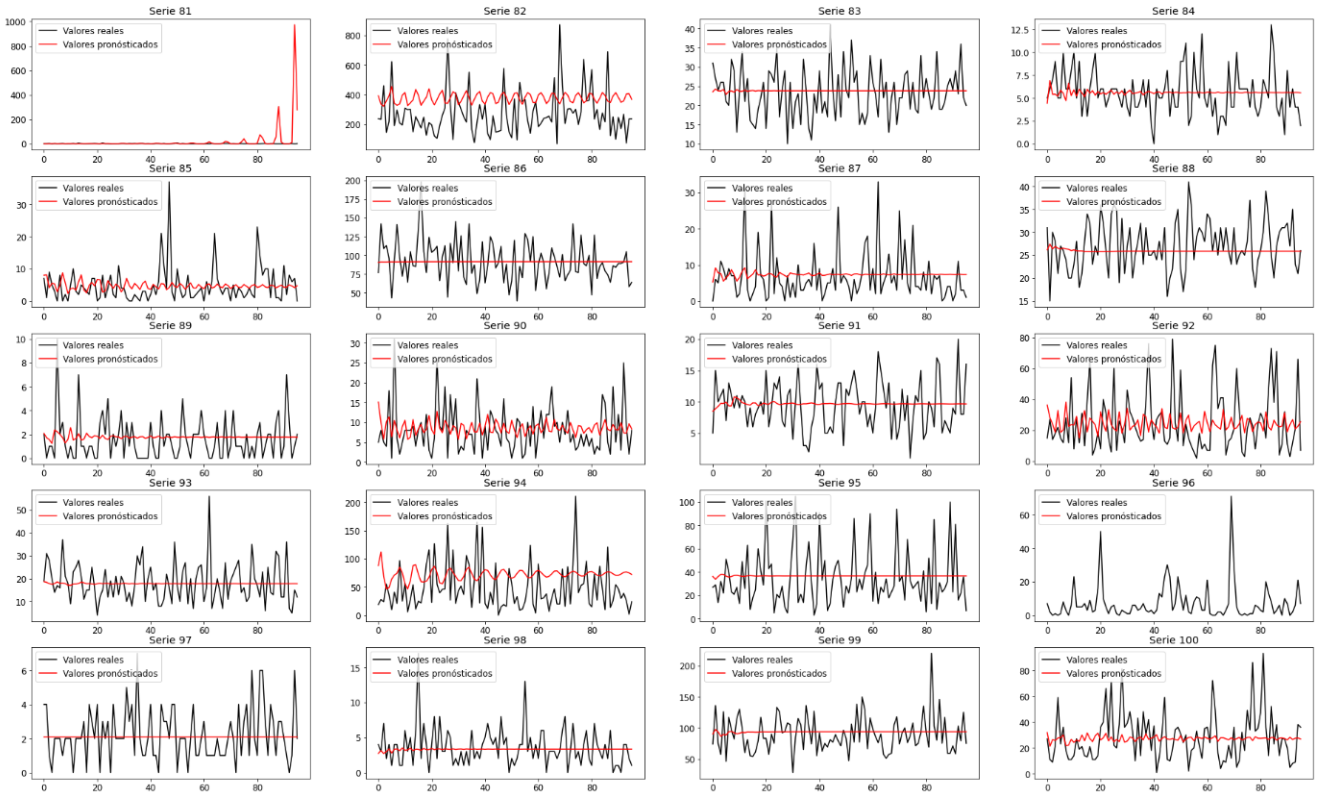

Figura 6-1.: Valores reales contra pronosticados de las últimas 20 series del caso 4 pronosticadas con modelo de autorregresión Poisson.

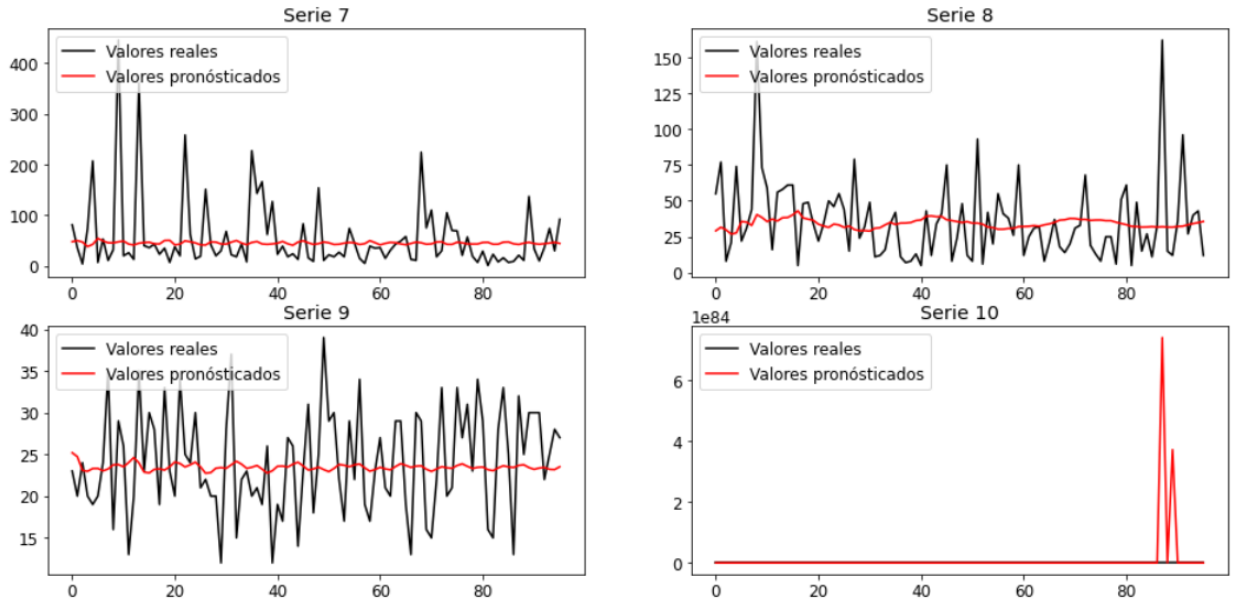

Figura 6-2.: Valores reales contra pronosticados de las últimas 4 series del caso 3 pronosticadas con modelo de autorregresión Poisson.

En la Figura **6-3** se presenta el perfil de medias del MASE que obtuvo cada modelo en los diferentes casos simulados. Se puede apreciar que ninguno de los modelos domina de forma general en términos de precisión de pronóstico, pero que el modelo SARIMA se desempeña de forma pobre para todos los casos, siendo el de peor o segundo peor desempeño siempre. Se puede observar, también, que mientras que los modelos globales y de machine learning tienen un comportamiento estable y cercano en la mayoría de casos (exceptuando el mal pronóstico del Transformer para el caso 7 donde se dispara el error), el modelo de autorregressión Poisson tiene un comportamiento más disperso al estar entre los mejores o peores modelos, lo que en parte concuerda con la información de la Tabla **6-1** donde se puede ver que la desviación estándar del MASE del modelo de autorregresión Poisson es superior a la de los modelos de machine learning de pronóstico global, en todos los casos donde su MASE promedio es elevado, además tiene errores máximos muy superiores. Lo anterior sugiere que el modelo puede generar un mejor ajuste para la serie de forma independiente cuando logra capturar el comportamiento autorregresivo, pero en los casos que falla se generan errores muy altos que suben el promedio del error, mientras que los modelos de machine learning logran mantener un error más uniforme en todos los casos, aunque superior al del modelo de regresión Poisson cuando este logra capturar el comportamiento de la serie, como se presenta con las series de los casos 5 y 6, donde se observan los valores promedio del error de pronóstico inferiores pero con altas desviaciones estándar para el modelo. Por otro lado, en algunos casos el modelo falló por la naturaleza no estacionaria del conjunto de datos, de modo que al no contemplar los cambios de nivel de la serie los pronósticos divergían del valor de la serie, siendo más notable este comportamiento mientras más prolongado fuera el periodo pronosticado.

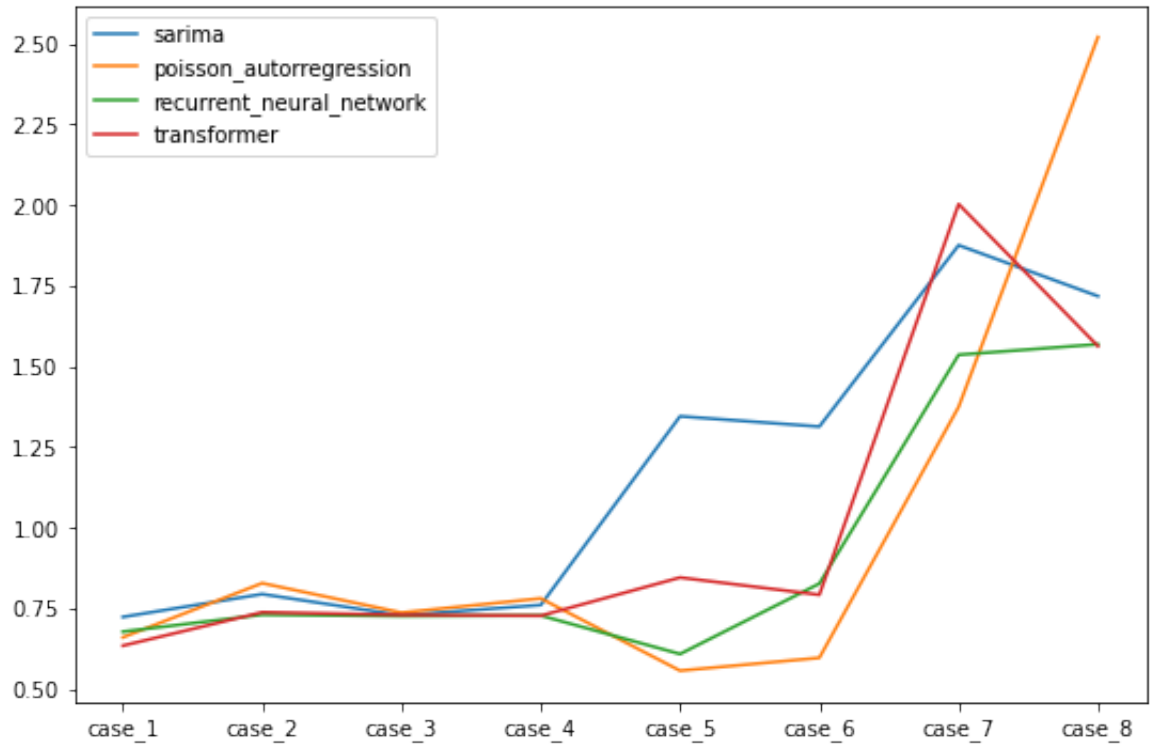

Figura 6-3.: Perfiles de medias del MASE para cada modelo en los diferentes casos.

En cuanto al modelo Transformer en el caso 7, un análisis adicional permitió observar que si bien el modelo aproximaba los comportamientos autorregresivos y estacionales de las series, presentaba errores de ajuste, subestimando o sobreestimando de forma recurrente la magnitud de estas componentes, lo que se atenuó para el caso 8; de lo anterior se infiere que probablemente el elevado error del modelo para el caso 7 se debió a la menor disponibilidad de series en los grupos (10 series por grupo), pero al aumentar este número a 100, mejora el desempeño de este modelo.

Otro hallazgo interesante es el hecho de que para todos los casos en los cuales los datos se generaron con el algoritmo propuesto para obtener series independientes a partir del modelo de autorregresión Poisson, y si además los procesos son estacionarios, el desempeño de todos los modelos resulta muy similar, lo que es consecuencia de que para estos casos los modelos generan como pronóstico valores cercanos a la media de cada serie. Sin embargo, los dos modelos estadísticos realizaron pronósticos más variables y de menor precisión, mientras que los pronósticos de los modelos machine learning se acercaron más a la media del proceso y con menor variabilidad.

Por otro lado, para analizar la información por casos, y profundizar sobre los hallazgos mencionados en términos de la desviación estándar de los modelos, se presentan, en la Figura **6-4**, los box plot de cada uno de los modelos para cada caso.

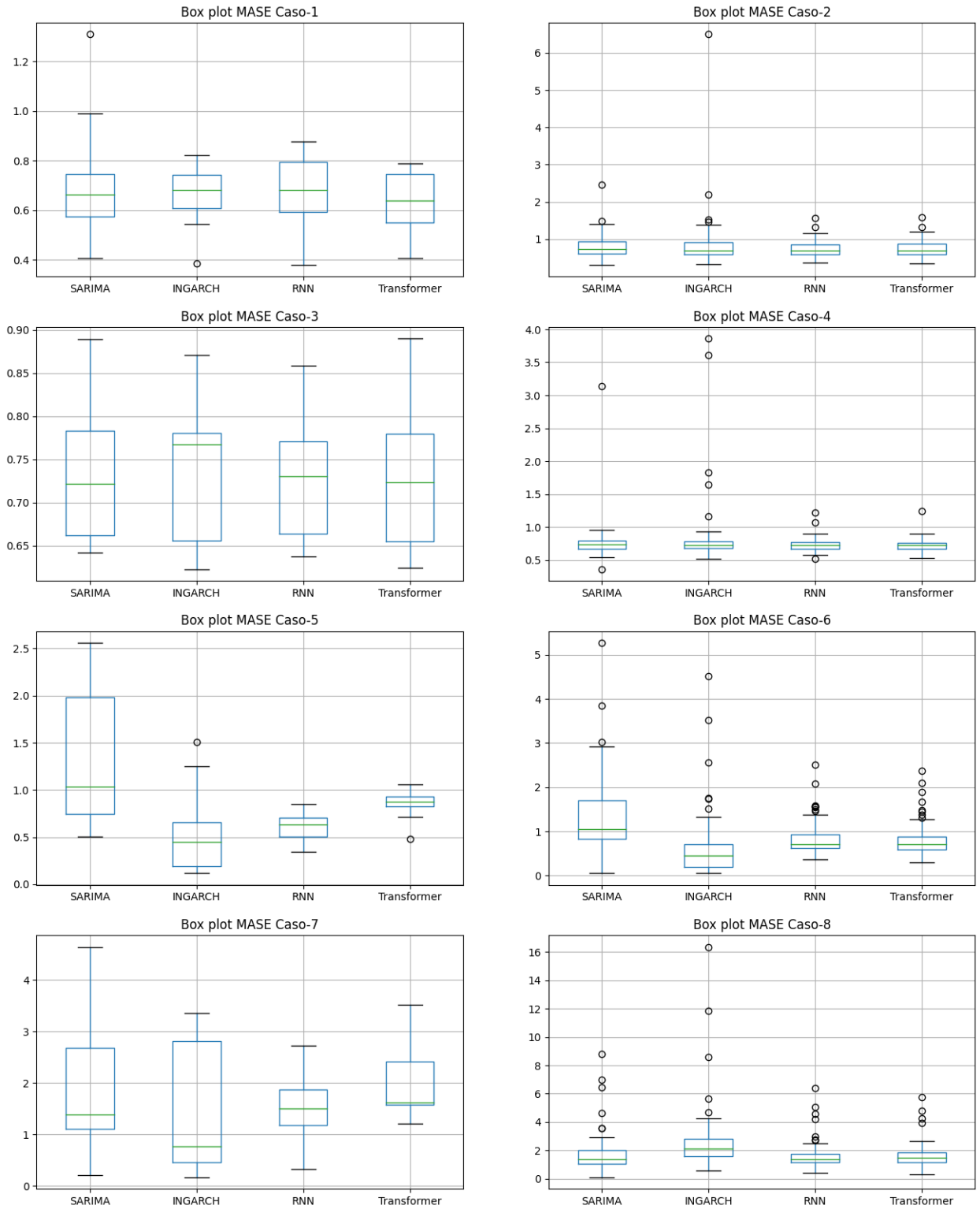

Figura 6-4.: Box plots del MASE de los modelos para cada caso.

En la Figura **6-4**, se puede confirmar que para todos los casos en los cuales los datos se generaron con el algoritmo propuesto a partir del modelo de autorregresión Poisson (casos 1 al 4), los desempeños de todos los modelos son muy similares y los resultados promedio superiores en la calidad del pronóstico (es decir menores errores promedio presentados en la Tabla **6-1** y la Figura **6-4**) de los modelos de machine learning, se deben principalmente a que no se generan errores atípicamente altos que arrastren el MASE promedio.

Por otro lado, para los casos donde los datos se generaron con el algoritmo GRATIS, se observa que sí existen diferencias más notables en los pronósticos de los modelos, y se nota una mejora en los errores cometidos por los modelos de machine learning respecto a los demás modelos, cuando los conjuntos tienen un mayor número de series de tiempo y las series son de mayor longitud. Se evidencia también que el modelo SARIMA tiene un mal desempeño en la mayoría de los casos (y nunca resulta ser el mejor de todos los modelos, aunque es el mejor de los dos modelos estadísticos en los casos 7 y 8) mientras que para el modelo INGARCH se presentan datos atípicos de errores muy elevados para los conjuntos de 100 series de tiempo.

Dada la no robustez de la media a valores atípicos, en la Figura **6-5** se presentan los perfiles de las medianas por modelo versus caso, excluyendo el modelo SARIMA debido a su inferior desempeño.

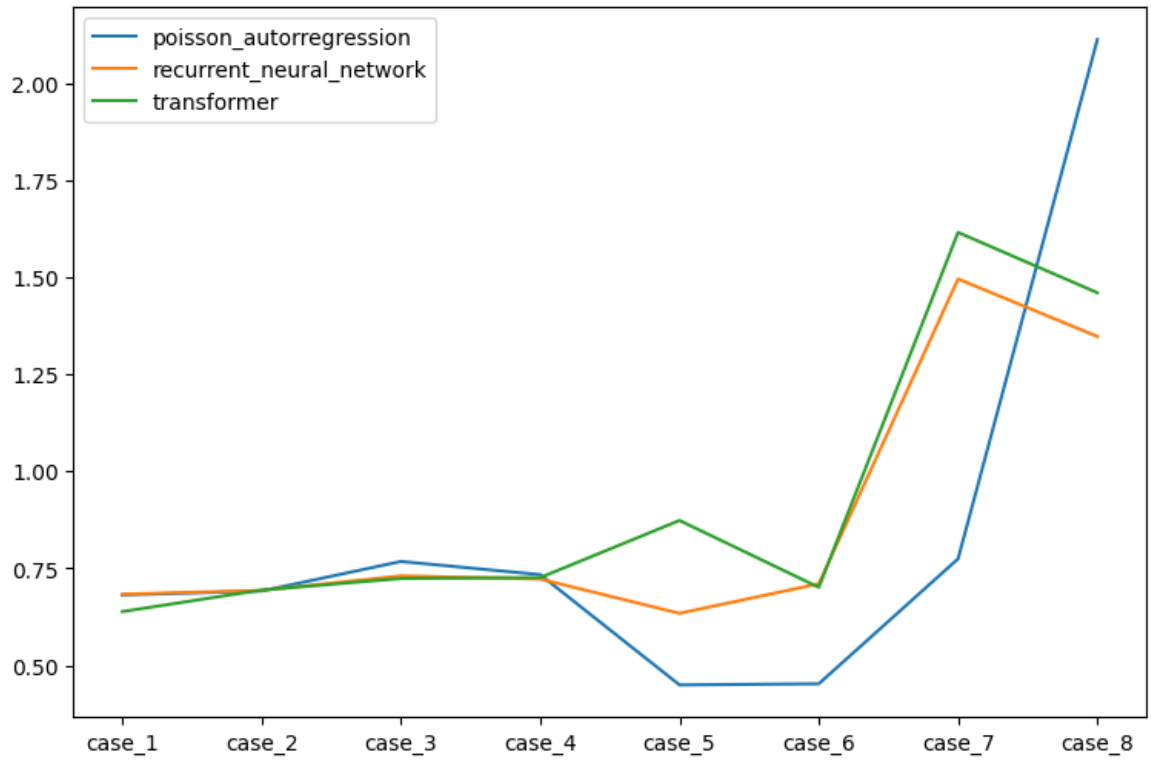

Figura 6-5.: Mediana del MASE de los modelos para cada caso.

En la Figura **6-5** se puede observar que si bien los resultados en los casos generados con la autorregresión Poisson siguen siendo sumamente similares, para los generados con el algoritmo GRATIS se presentó un desempeño muy superior del modelo de autorregresión Poisson en todos los casos, exceptuando el caso 8 para el cual se tienen 100 series de tiempo de longitud 1000 y un horizonte de pronóstico más largo. Este hallazgo resulta de gran interés, ya que en la práctica, cuando los conjuntos no contengan muchas series de tiempo, es factible la intervención humana para refinar los modelos autorregresivos en los casos donde su desempeño resulte pobre y potencialmente pueden conducir a mejores resultados que los modelos de machine learning. Por otro lado, para conjuntos con muchas series de tiempo, los modelos de machine learning no solo no presentaron valores atípicamente altos en sus errores de predicción, sino que también arrojan mejor desempeño en el pronóstico.

Para evaluar la relación del desempeño de los diferentes modelos versus las características de las series de tiempo en los conjuntos de datos simulados, a continuación, se analizarán los efectos individuales de los factores controlados en el experimento de simulación, previamente mencionados y que corresponden a:

- **La naturaleza de los datos:** Complejidad de la relación autorregresiva y heterogeneidad.
- **La longitud de las series de tiempo en el conjunto de datos**
- **El número de series de tiempo**

6.2.1. La naturaleza de los datos

En primera instancia, se presenta en la Figura **6-6** el MASE promedio cuando solo se varía la naturaleza de la generación de los datos: si se trata de procesos autorregresivos Poisson cuyos coeficientes se generan de forma aleatoria, al igual que los órdenes autorregresivos, y las series son independientes entre sí o de la generación con el método GRATIS con series dependientes con las mismas características objetivo.

En los gráficos se puede observar que la mayoría de los modelos se desempeñan peor pronosticando sobre los conjuntos de series generadas con el algoritmo GRATIS, en el cual no se garantiza la estacionariedad de los procesos temporales, mientras que el desempeño fue mejor y más cercano entre los modelos, cuando los datos fueron generados por el primer algoritmo, el cual garantiza la estacionariedad y la ausencia de patrones estacionales. Se infiere, por tanto, que mientras los procesos son estacionarios y no estacionales, los distintos modelos probados parecen tener menor error de predicción al producir pronósticos cercanos a los valores promedios de las series. Ahora bien, el modelo autorregresivo Poisson mostró la peculiaridad de superar a los demás modelos cuando los conjuntos de series fueron simuladas

a partir del algoritmo GRATIS, con longitud de 200 y plazo de pronósticos más cortos, en cambio, resulta peor con series largas y a la vez horizontes de pronósticos largos, muy probablemente porque a pesar de un mayor tamaño de muestra en el ajuste, el pronóstico pierde precisión si el horizonte de tiempo es más largo, y también porque el modelo autorregresivo puede no ajustar bien a los datos generados por procesos no estacionarios.

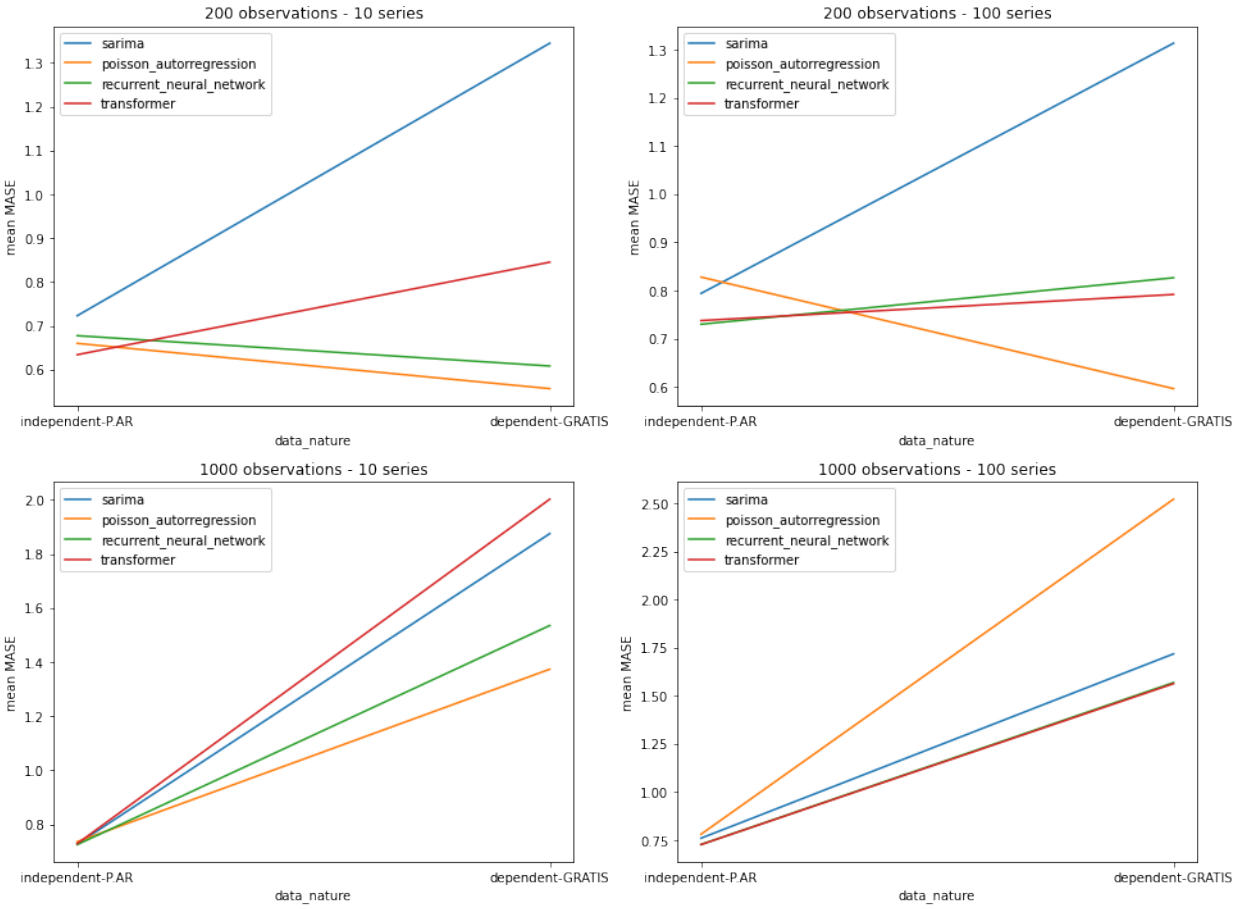

Figura 6-6.: MASE promedio obtenido para cada modelo según naturaleza de los datos.

Por otro lado, se debe tener en cuenta que para los modelos de machine learning los horizontes más prolongados y mayores números de series en el conjunto de datos, hacen también más complejas las funciones de pérdida y aumentan el número de gradientes, lo que demanda una mayor cantidad de datos. Especialmente el modelo Transformer, con muchos más parámetros que la red recurrente, falló en capturar el comportamiento de las series generadas con el algoritmo GRATIS en los conjuntos de menos series paralelas y con menor disponibilidad de datos, por lo que se hizo evidente el aumento del error predictivo en estos escenarios versus el error cometido en los casos donde los datos provinieron del primer algoritmo. La excepción fue el caso de la RNN para el horizonte más corto, de 200 observaciones y 10 series

de tiempo, donde la red logró aproximar el comportamiento de las series generadas con el algoritmo GRATIS pese a la menor disponibilidad de datos, probablemente también por ser el modelo de machine learning con la arquitectura más simple, lo que representa menos parámetros a ajustar.

6.2.2. La longitud de las series de tiempo

En la Figura **6-7** se presentan ahora los promedio del MASE cuando solo varía la longitud de las series de tiempo en el conjunto de datos de 200 a 1000 observaciones en total, lo que implica también un cambio en el horizonte de pronóstico, de 12 periodos en el primer caso y 96 en el segundo.

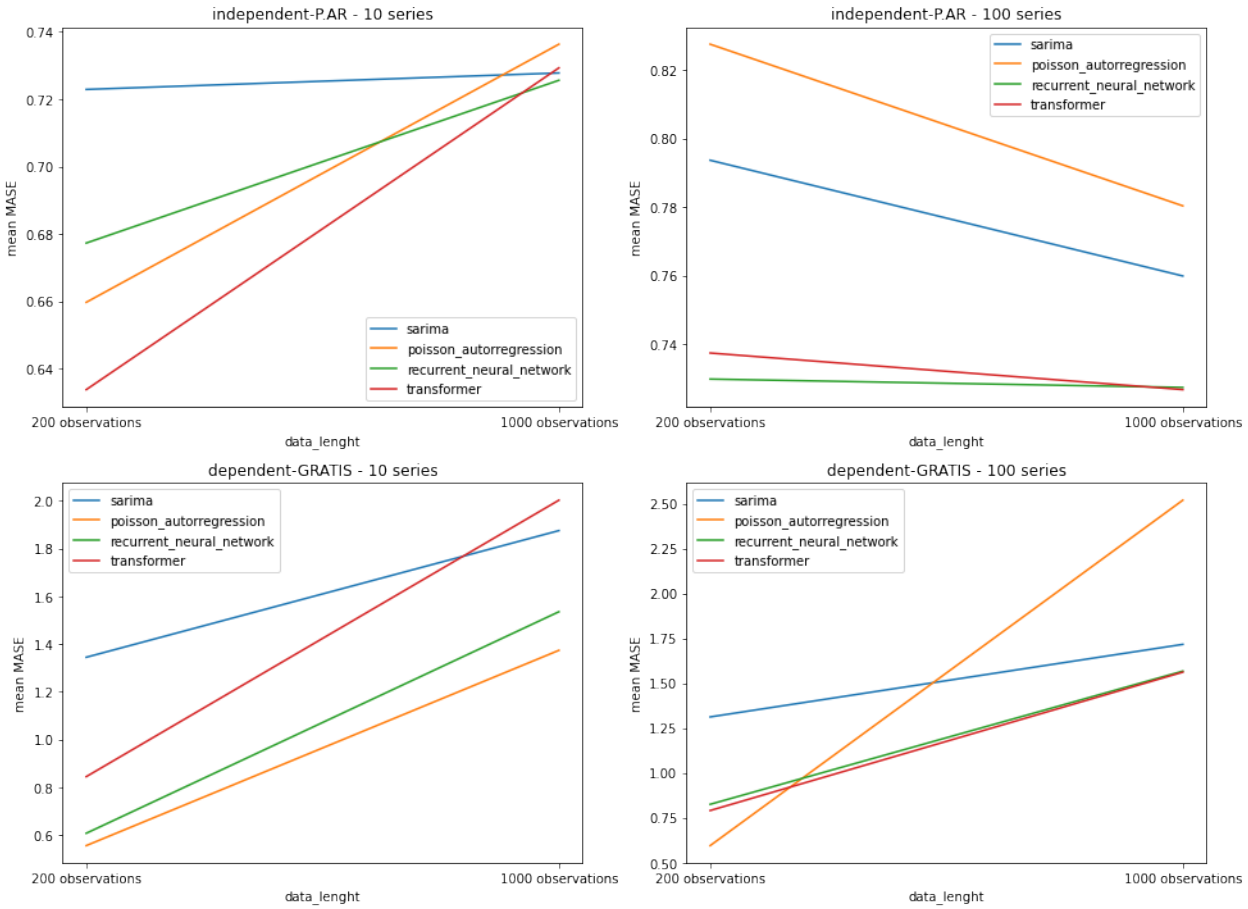

Figura 6-7.: MASE promedio obtenido para cada modelo según la longitud de las series de tiempo.

Se puede observar que para el caso de las series independientes y solo 10 series a la vez, el tamaño de las series conjuntamente con el horizonte de pronóstico (recuerde que en series de longitud 200 se ajusta con 188 observaciones y se pronostican las 12 últimas, en tanto que en las series de longitud 1000 se ajusta con 904 observaciones y se pronostican 96), empeora la calidad de los pronósticos de los modelos, pues a pesar de la mayor información en el ajuste, esto no contribuye a una mayor precisión en el pronóstico, si el horizonte de predicción es muy largo. Sin embargo, al aumentar el número de series de tiempo paralelas, de 10 a 100, la precisión de pronóstico de los modelos parece mejorar. Como se mencionó anteriormente, un análisis adicional permitió establecer que, en general, los modelos no logran capturar el comportamiento autorregresivo de la mayoría de las series generadas de forma independiente con el algoritmo autorregresivo propuesto, y sus pronósticos terminan siendo cercanos al promedio global de las series, desempeñándose mejor en esos escenarios, los modelos globales cuyos pronósticos se acercaron más a las medias. Por otro lado, los casos de 1000 datos fueron procesos de simulación más prolongados y estables, en los cuales sus valores oscilaron más cerca a su media, lo que generó el resultado observado. Esto último es de esperarse para los modelos autorregresivos, dada la estacionariedad de las series del conjunto, y resulta interesante que para los modelos de Machine learning se alcance un ajuste más preciso de la media global del proceso. Lo anterior plantea la pregunta de cuál comportamiento podrían tener los modelos de machine learning planteados en escenarios estacionarios, pero con series relacionadas entre sí y con los mismos órdenes de autocorrelación, lo que podría ser un problema de interés para futuras investigaciones.

Para el caso de las series dependientes, se observa que nuevamente para todos los modelos empeora la calidad del pronóstico al aumentar su horizonte. Para el modelo de regresión Poisson el aumento del error de pronóstico en los horizontes de pronósticos más prolongados se hizo especialmente notorio para el conjunto de 100 series, pues se presentaron más casos en los cuales el modelo falló en capturar el comportamiento de la serie y se generaron errores muy altos que afectaron el promedio global, como puede verificarse en la Tabla **6-1**, donde se observa una desviación estándar del MASE y un valor máximo del modelo muy superior a los de los demás en el caso 8. En cuanto a los modelos globales, se observa que en el conjunto de 100 series hay un incremento menor en el error de pronóstico a medida que aumenta el horizonte, evidenciándose que para las series dependientes estos modelos sacan provecho de la mayor cantidad de datos, especialmente el modelo Transformer.

6.2.3. El número de series de tiempo

La Figura **6-8** presenta el promedio del MASE al variar el número de series de tiempo a pronosticar en el conjunto de datos de 10 a 100. En la figura se puede observar que al aumentar el número de series pronosticadas y cuando éstas son independientes, y generadas con el algoritmo propuesto, los modelos en general tienen un peor desempeño para el pronóstico,

al generarse más casos en los que el modelo falla en capturar el comportamiento de la serie. La salvedad es el caso del modelo transformer, que con más datos logró aproximar mejor las medias globales (acorde al análisis de la subsección anterior) más no aproxima bien el comportamiento autorregresivo de las series al aumentar la cantidad de éstas.

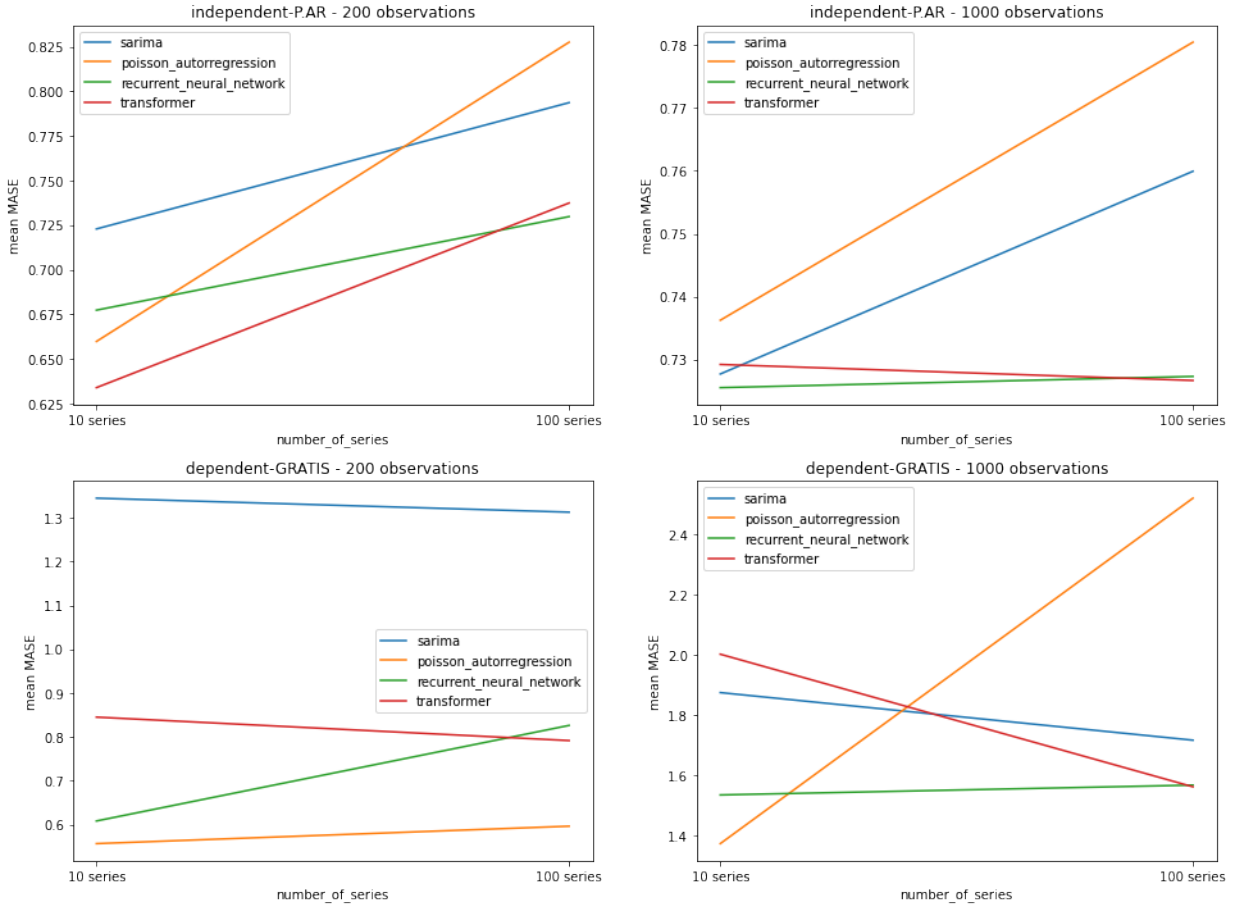

Figura 6-8.: MASE promedio obtenido para cada modelo según el número de series de tiempo.

Sin embargo, el comportamiento cambia para las series dependientes. El modelo de regressión Poisson logra mejores pronósticos para menor cantidad de series en el conjunto de datos, donde se presentan menos casos en los cuales falla en el ajuste de las series. Por otro lado, la RNN empeora la calidad de sus pronósticos al tener más series en el conjunto de datos, posiblemente porque su arquitectura simple y estructura autorregresiva no logran obtener suficiente provecho de la información de las series paralelas para mitigar el efecto de la complejidad paramétrica que genera la función de pérdida para más series de tiempo. Por su parte, con el Transformer aumentar la cantidad de series paralelas, y por lo tanto, la información a ser explotada por los algoritmos de atención, conduce a un mayor beneficio

en la calidad del pronóstico. Cabe resaltar que el caso de 1000 observaciones y 100 series de tiempo con series dependientes, es el único donde los algoritmos globales no solo son en promedio los mejores, sino que logran, también, errores mínimos notablemente inferiores a los del modelo de autorregresión Poisson.

6.2.4. Eficiencia computacional

Finalmente, se presentan algunos hallazgos en términos del costo computacional, principalmente para el tiempo demandado por los diferentes modelos. En la Tabla **6-2** se presenta el resumen de los tiempos de corrida para los entrenamientos de todos los modelos. Como se puede observar, para los casos en los que los conjuntos de datos son pequeños (pocas series de tiempo y de pocas observaciones) los modelos locales tienen tiempos computacionales mucho menores que los globales, sin embargo, a medida que la cantidad de datos aumenta, se incrementa el tiempo de corrida, pero este incremento es bastante significativo con el modelo autorregresivo Poisson, en tanto que el modelo SARIMA necesitó en general menores tiempos que los demás modelos.

Tabla 6-2.: Tiempos computacionales para el entrenamiento de cada modelo

	SARIMA	**INGARCH**	**RNN**	**Transformer**
Caso 1	23.6s	43m, 31.8s	173m, 25.32s	415m, 55.8s
Caso 2	4m, 6s	415m,38.9	175m, 14.7s	420m, 51.3s
Caso 3	2m, 17.9s	732m, 42.1s	214m, 30.5s	719m, 44s
Caso 4	30m, 33.9s	1102m, 15.4s	237m, 32.6s	715m, 23.3s
Caso 5	42.8s	41m, 57.1s	147m, 43.2s	410m, 5.7s
Caso 6	6m, 12.1s	415m, 38.9s	150m, 21.3s	398m, 23.8s
Caso 7	4m, 3.5s	727m, 30.3s	212m, 9.9s	690m, 49.0s
Caso 8	53m, 40.2s	1058, 28.6s	246m, 46.1s	704m, 26.2s

Los resultados presentados se obtuvieron con un computador portátil con un procesador Intel(R) Core(TM) i7-10750H de 6 núcleos y 12 procesadores lógicos, con velocidad de base de 2.60GHz, 16GB de memoria RAM y una tarjeta gráfica NVIDIA GeForce GTX 1650 Ti de 4GB.

7. Conclusiones y recomendaciones

7.1. Conclusiones

El pronóstico de series de tiempo de conteos requiere de aproximaciones a dichos procesos con modelos cuyos supuestos sean soportados por conjuntos numéricos adecuados para valores discretos y ocurrencias de ceros, o modelos basados en datos que no tengan supuestos distribucionales. En este trabajo se mostró que al usar los modelos comúnmente empleados para el pronóstico de series de tiempo, pero cuyos supuestos se basan en distribuciones Gausianas, como es el caso de los modelos ARIMA ó SARIMA, los pronósticos resultantes pueden ser de muy mala precisión.

Por otro lado, este trabajo abordó la aplicación de modelos locales, que modelan y pronostican las series de forma individual y que son de naturaleza estadística, hallando que logran mejores resultados para el pronóstico de cada serie de tiempo en los casos en los cuales los conjuntos de datos son pequeños (pocas series de tiempo paralelas), y cuando se logra capturar la naturaleza autorregresiva de las series y estas obedecen a los supuestos de estacionariedad que hace el modelo. Sin embargo, esto genera una restricción en términos de la escabilidad del proceso de pronóstico, pues requiere intervención humana para los casos en los cuales las selecciones automáticas fallen y sea necesario el refinamiento de los modelos, con el fin de mejorar su calidad de ajuste y de predicción.

Por otro lado, con los modelos de machine learning utilizados para pronósticos globales, se logran mejores resultados cuando se tienen series de tiempo largas y conjuntos de series grandes, particularmente en los modelos complejos como el Transformer propuesto. Sin embargo, para las redes neuronales recurrentes, la complejidad de la función de pérdida al aumentar el número de series a pronosticar afecta su capacidad de pronóstico.

Los experimentos también encontraron que al aumentar el número de serie de tiempo en paralelo, cuando éstas son independientes, los modelos globales no presentan una buena capacidad de pronóstico, dado que no se puede extraer información de una serie para pronosticar otra, y aproximan las series estacionarias como sus medias, pese a existir patrones autorregresivos.

Además de la calidad de pronóstico, a medida que aumenta la cantidad de datos, también mejoran los tiempos computacionales de los modelos globales de machine learning, respecto

a los locales estadísticos, pues estos últimos crecen a una proporción mucho mayor. Si bien en este trabajo no se observaron nunca tiempos mayores del modelo SARIMA, para conjuntos de dimensiones mucho mayores, es de esperar que los tiempos computacionales de los modelos de machine learning sean menores a los de este modelo.

De acuerdo a los factores considerados en el estudio de simulación, se concluye que en los conjuntos de pocas series de tiempo, donde resultara viable la supervisión humana para garantizar el cumplimiento de supuestos y verificar la calidad del pronóstico individual de las series, parece más recomendable el uso de las metodologías estadísticas. Sin embargo, cuando los conjuntos de datos son grandes, la aplicación de estos modelos no solo presenta dificultades prácticas, sino que también muestra un desempeño inferior para el pronóstico que los modelos de machine learning.

Por otro lado, los resultados de este estudio, en términos de las métricas usadas sobre la precisión de pronóstico, son similares a los encontrados por Montero-Manso y Hyndman (2021) sobre el pronóstico de múltiples series de tiempo de valores continuos (no discretas). Esto sugiere que lo desarrollado en la literatura sobre el pronóstico de múltiples series de tiempo continuas puede extenderse al campo de las series de conteo y orientar el trabajo futuro sobre el problema abordado en esta tesis.

7.2. Trabajo futuro

Si bien este trabajo presentó con cierto nivel de profundidad los modelos Autorregresivos Poisson, las Redes Neuronales Recurrentes y los Transformers para el pronóstico de series de tiempo de conteos, existen desarrollos sobre estos modelos que no se profundizaron e implementaron y que pueden mejorar la capacidad de pronóstico de los mismos. En primera instancia, el ajuste de los modelos autorregresivos Poisson para incluir estacionalidades e innovaciones puede resultar de interés, así como considerar variaciones al modelo que contemplen cambios de nivel y órdenes de integración tal como en los modelos ARIMA.

Por otro lado, arquitecturas más complejas para las redes neuronales recurrentes que incluyan conexiones residuales, estructuras codificador-decodificador y otros elementos para optimizar el modelo son otra área de interés a explorar. En cuanto a los Transformers, se trata de un tema de investigación actual y otras adaptaciones para el pronóstico de series de tiempo pueden estudiarse.

Teniendo en cuenta además, que en los hallazgos de este trabajo se encontró que los modelos locales pueden capturar mejor el comportamiento de las series, pero carecen de la estabilidad de los globales, se dan indicios de que aproximaciones mixtas pueden tener mejores resultados, tal como se comprobó para series continuas en Makridakis et al., 2018. En esta

línea, está el analizar, también, el resultado de modelos de redes neuronales recurrentes que tomen conjuntos de múltiples series, pero ajusten funciones de pérdida sobre una serie a la vez, generando una red para el pronóstico de cada una, de forma local, pero con una entrada global de datos. Otra línea en este sentido es replicar modelos con resultados superiores para el pronóstico de series continuas con las debidas adaptaciones a las series de conteos.

Finalmente, teniendo en cuenta que en los modelos de lenguaje basados en Transformers se ha comprobado el beneficio de reutilizar modelos entrenados sobre grandes cuerpos de texto, para adaptar una capa final que se entrene sobre conjuntos limitados de datos en aplicaciones específicas (práctica denominada "fine-tunning"), podría resultar de interés experimentar una adaptación similar de tomar modelos entrenados para el pronóstico de series de tiempo sobre conjuntos de referencia con gran cantidad de series, como los de las competencias M, y adaptarlos en aplicaciones específicas.

A. Descenso Gradiente

Según Ruder, 2016 El descenso gradiente es uno de los algoritmos más populares para la optimización y por mucho el más común para la optimización de redes neuronales, de modo que casi todas las librerías de referencia de Deep learning contienen implementaciones de varios algoritmos de optimización por descenso gradiente. Este algoritmo es una forma de optimizar una función $J(\boldsymbol{\theta})$, con $\boldsymbol{\theta} \in \mathbb{R}^d$ un vector de parámetros, al actualizar los parámetros en la dirección opuesta al gradiente de la función objetivo, dado por $\nabla_\theta J(\boldsymbol{\theta})$.

Algunos de los algoritmos más ampliamente utilizados en el descenso gradiente son:

- Momentum
- Nesterov accelerated gradient
- Adagrad
- Adadelta
- RMSprop
- Adam
- AdaMax
- Nadam

A continuación, se explica el algoritmo Adam, el cual fue implementado en este trabajo.

Según Ruder, 2016, la estimación de momentos adaptativa (Adam) es un método que calcula tasas de aprendizaje adaptativas para cada parámetro, calculando un decrecimiento exponencial medio de los gradientes anteriores,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{A-1}$$

y un decrecimiento exponencial medio de los gradientes cuadráticos anteriores,

$$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) g_t^2, \tag{A-2}$$

donde g_t es el valor del gradiente evaluado en el valor del vector de parámetros en el paso t y m_t y ν_t son estimaciones del primer y segundo momento de los gradientes, respectivamente y se inicializan como vectores de $0's$. Este algoritmo incluye una corrección de sesgo para los estimadores dadas respectivamente por

$$\widehat{m_t} = \frac{m_{t-1}}{1 - \beta_1^t}, \tag{A-3}$$

$$\widehat{\nu_t} = \frac{\nu_{t-1}}{1 - \beta_2^t}, \tag{A-4}$$

donde β_1^t, β_2^t corresponden a las t-ésimas potencias de estos parámetros.

Los valores calculados en (A-3) y (A-4) se utilizan para actualizar el vector de parámetros $\boldsymbol{\theta}$ en cada paso utilizando la regla:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{\nu}_t} + \epsilon} \hat{m}_t, \tag{A-5}$$

Para este algoritmo se proponen valores por defecto de 0,9 para β_1, 0,999 para β_2 y 10^{-8} para ϵ. El algoritmo está dado por:

$\alpha = 0{,}001, \beta_1 = 0{,}9, \beta_2 = 0{,}999, \epsilon = 10^{-8}$ (Valores por defecto).
$m_0 = 0$
$\nu_0 = 0$
$t = 0$
Mientras $\boldsymbol{\theta}_t$ no converja, hacer:
  $t = t + 1$
$\boldsymbol{g_t} = \nabla_\theta J(\boldsymbol{\theta}_{t-1})$
$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
$\nu_t = \beta_2 \nu_{t-1} + (1 - \beta_2) g_t^2$
$\hat{m}_t = \frac{m_{t-1}}{1 - \beta_1^t}$
$\hat{\nu}_t = \frac{\nu_{t-1}}{1 - \beta_2^t}$
$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\hat{\nu}_t} + \epsilon} \hat{m}_t$
Terminar ciclo
retornar $\boldsymbol{\theta}_t$

El ciclo se repite un número de iteraciones definido o hasta que deja de existir reducción en la función $J(\boldsymbol{\theta})$ al evaluar en $\boldsymbol{\theta} = \boldsymbol{\theta}_t$.

B. Cálculo vectorial y matricial

En este Anexo se presentan elementos del cálculo vectorial y matricial que se aplican en este trabajo. Mayor detalle del cálculo utilizado en deep learning puede verse en Parr y Howard, 2018, fuente utilizada para la creación de este Anexo.

Gradientes y Jacobianos

Sea $\boldsymbol{x} \in \mathbb{R}^{n\times 1} = [x_1, x_2, \ldots, x_n]^T$ y $y = f(\boldsymbol{x}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}$ El gradiente de y respecto a $\boldsymbol{x}$ esta dado por:

$$\frac{\partial y}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} & \frac{\partial f(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{bmatrix}. \tag{B-1}$$

Si se denota $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}^{m\times 1}$ al vector

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{bmatrix}$$

El Jacobiano es la matriz conteniendo todas las posibles derivadas parciales escalares:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial \boldsymbol{x}} \\ \frac{\partial f_2(\boldsymbol{x})}{\partial \boldsymbol{x}} \\ \vdots \\ \frac{\partial f_m(\boldsymbol{x})}{\partial \boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_1(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_n} \\ \frac{\partial f_2(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_2(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\boldsymbol{x})}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_m(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_m(\boldsymbol{x})}{\partial x_n} \end{bmatrix}. \tag{B-2}$$

Operaciones elemento a elemento entre vectores

Sean $\boldsymbol{f}(\boldsymbol{w}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}^{n\times 1}$ y $\boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}^{n\times 1}$ funciones vectoriales tales que cada $f_i(\boldsymbol{w}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}$ accede únicamente a w_i y cada $g_i(\boldsymbol{x}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}$ únicamente a x_i, es decir:

$$\boldsymbol{f}(\boldsymbol{w}) = \begin{bmatrix} f_1(\boldsymbol{w}) \\ f_2(\boldsymbol{w}) \\ \vdots \\ f_n(\boldsymbol{w}) \end{bmatrix} = \begin{bmatrix} f_1(w_1) \\ f_2(w_2) \\ \vdots \\ f_n(w_n) \end{bmatrix}, \boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \\ \vdots \\ g_n(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} g_1(x_1) \\ g_2(x_2) \\ \vdots \\ g_n(x_n) \end{bmatrix}. \tag{B-3}$$

Se define la forma genérica de una operación elemento a elemento en los vectores $\boldsymbol{w} \in \mathbb{R}^{n\times 1}$ y $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ usando el operador $\bigcirc$ como $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{w}) \bigcirc \boldsymbol{g}(\boldsymbol{x}) \in \mathbb{R}^{n\times 1}$, tal que

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{w}) \bigcirc g_1(\boldsymbol{x}) \\ f_2(\boldsymbol{w}) \bigcirc g_2(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{w}) \bigcirc g_n(\boldsymbol{x}) \end{bmatrix} \tag{B-4}$$

Un ejemplo es el producto elemento a elemento o producto Hadamard dado por $\boldsymbol{y} = \boldsymbol{w} \odot \boldsymbol{x}$, donde $f_i(\boldsymbol{w}) = w_i$, $g_i(\boldsymbol{x}) = x_i$ y $f_1(\boldsymbol{w}) \odot g_1(\boldsymbol{x}) = w_i \times x_i$.

De forma general, el jacobiano de la función vectorial $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{w}) \bigcirc \boldsymbol{g}(\boldsymbol{x})$ con respecto a $\boldsymbol{x}$ es:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial (f_1(\boldsymbol{w}) \bigcirc g_1(\boldsymbol{x}))}{\partial x_1} & \frac{\partial (f_1(\boldsymbol{w}) \bigcirc g_1(\boldsymbol{x}))}{\partial x_2} & \cdots & \frac{\partial (f_1(\boldsymbol{w}) \bigcirc g_1(\boldsymbol{x}))}{\partial x_n} \\ \frac{\partial (f_2(\boldsymbol{w}) \bigcirc g_2(\boldsymbol{x})}{\partial x_1} & \frac{\partial (f_2(\boldsymbol{w}) \bigcirc g_2(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial (f_2(\boldsymbol{w}) \bigcirc g_2(\boldsymbol{x})}{\partial x_n} \\ \vdots & & & \\ \frac{\partial (f_m(\boldsymbol{w}) \bigcirc g_m(\boldsymbol{x}))}{\partial x_1} & \frac{\partial (f_m(\boldsymbol{w}) \bigcirc g_m(\boldsymbol{x}))}{\partial x_2} & \cdots & \frac{\partial (f_m(\boldsymbol{w}) \bigcirc g_m(\boldsymbol{x}))}{\partial x_n} \end{bmatrix}. \tag{B-5}$$

Teniendo en cuenta la restricción de que $f_i(\boldsymbol{w})$ accede únicamente a w_i y $g_i(\boldsymbol{x})$ únicamente a x_i, el jacobiano se simplifica a la matriz diagonal:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \operatorname{diag}\left(\left\{ \frac{\partial (f_i(w_i) \bigcirc g_i(x_i))}{\partial x_i} \right\}_{i=1}^{n} \right). \tag{B-6}$$

Para el caso del producto Hadamard se tiene:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \text{diag}\left(\left\{\frac{\partial(w_i \times x_i)}{\partial x_i}\right\}_{i=1}^{n}\right) = \text{diag}\left(\left\{w_i\right\}_{i=1}^{n}\right) = \text{diag}(\boldsymbol{w}).$$

Operación elemento a elemento

Una operación elemento a elemento en el vector $\boldsymbol{x} \in \mathbb{R}^n$ usando la función escalar $g(\cdot) : \mathbb{R} \longrightarrow \mathbb{R}$ está dada por la función vectorial $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}^{n\times 1}$ tal que,

$$\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{x}) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} g(x_1) \\ g(x_2) \\ \vdots \\ g(x_n) \end{bmatrix}. \tag{B-7}$$

Si se denota $g'(x) = \frac{dg(x)}{dx}$, el jacobiano con respecto a $\boldsymbol{x}$ es:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \text{diag}\left(\left\{g'(x_i)\right\}_{i=1}^{n}\right). \tag{B-8}$$

Expansión escalar

Sumar el escalar z al vector $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$, $\boldsymbol{y} = \boldsymbol{x} + z$, realmente corresponde a $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{z})$ donde $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}$ y $\boldsymbol{g}(z) = \mathbf{1}_{\boldsymbol{n}} z$, donde $\mathbf{1}_{\boldsymbol{n}} \in \mathbb{R}^{n\times 1}$ es un vector de unos.

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{x} + z) = \text{diag}(\mathbf{1}_{\boldsymbol{n}}) = \boldsymbol{I}_{\boldsymbol{n}}, \tag{B-9}$$

con $\boldsymbol{I}_{\boldsymbol{n}} \in \mathbb{R}^{n\times n}$ una matriz identidad.

$$\frac{\partial}{\partial z}(\boldsymbol{x} + z) = \mathbf{1}_{\boldsymbol{n}}, \tag{B-10}$$

con $\boldsymbol{I}_{\boldsymbol{n}} \in \mathbb{R}^{n\times 1}$ un vector de unos.

Reducción de vectores

Sean $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ y $\boldsymbol{w} \in \mathbb{R}^{n\times 1}$ vectores columna, se tienen las siguientes reglas para la reducción de estos.

Para $y = \text{sum}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i$.

$$\frac{\partial y}{\partial \boldsymbol{x}} = \mathbf{1}_{\boldsymbol{n}}^{T}.$$

Sobre el producto punto $y = \text{sum}(\boldsymbol{x} \odot \boldsymbol{w}) = \sum_{i=1}^{n} x_i w_i = \boldsymbol{x}^T \boldsymbol{w}$.

$$\frac{\partial y}{\partial \boldsymbol{x}} = \boldsymbol{w}^T,$$

$$\frac{\partial y}{\partial \boldsymbol{w}} = \boldsymbol{x}^T.$$

Regla de la cadena

Sean $\boldsymbol{f}(\cdot) : \mathbb{R}^{m\times 1} \longrightarrow \mathbb{R}^{k\times 1}$, $\boldsymbol{g}(\cdot) : \mathbb{R}^{n\times 1} \longrightarrow \mathbb{R}^{m\times 1}$ funciones vectoriales, y $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ un vector columna

$$\frac{\partial \boldsymbol{f}(\boldsymbol{g}(\boldsymbol{x}))}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{g}} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}}. \tag{B-11}$$

C. Cálculo de los gradientes en la regresión como red neuronal

El proceso de obtención de las derivadas parciales necesarias para la actualización del gradiente en cada paso de la optimización mediante descenso del gradiente (ver anexo A) requiere la aplicación de la regla de la cadena vectorial (ver anexo B), y toma el nombre de back propagation o propagación hacia atrás (Zhang et al., 2021), ya que en éste la relación de la función de costo con los parámetros sigue la lógica:

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) \longrightarrow \hat{\boldsymbol{y}}(\boldsymbol{z}) \longrightarrow \boldsymbol{z}(\boldsymbol{w}, b).$$

La cual es contraria al forward pass, o propagación hacia adelante, donde se obtuvo la estimación a partir de los datos de entrada y los valores de los parámetros. Las ecuaciones que presentan este cálculo pueden observarse en (4-3), (4-4), (4-5) y (4-6).

Inicialmente, obtendremos las derivadas parciales de las diferentes funciones aplicadas en el forward pass (las reglas para el cálculo utilizado en esta sección pueden revisarse en el anexo B), comenzando con las derivadas parciales de la función afín $\boldsymbol{z}$ respecto a los parámetros del modelo:

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial z_1}{\partial \boldsymbol{w}} \\ \frac{\partial z_2}{\partial \boldsymbol{w}} \\ \vdots \\ \frac{\partial z_n}{\partial \boldsymbol{w}} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{w}}(\boldsymbol{x_1^T w} + b) \\ \frac{\partial}{\partial \boldsymbol{w}}(\boldsymbol{x_2^T w} + b) \\ \vdots \\ \frac{\partial}{\partial \boldsymbol{w}}(\boldsymbol{x_n^T w} + b) \end{bmatrix} = \begin{bmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots x_{1p} \\ x_{21} & x_{22} & \dots x_{2p} \\ \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots x_{np} \end{bmatrix} = \boldsymbol{A}, \tag{C-1}$$

$$\frac{\partial \boldsymbol{z}}{b} = \begin{bmatrix} \frac{\partial z_1}{\partial b} \\ \frac{\partial z_2}{\partial b} \\ \vdots \\ \frac{\partial z_n}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial b}(\boldsymbol{x_1^T w} + b) \\ \frac{\partial}{\partial b}(\boldsymbol{x_2^T w} + b) \\ \vdots \\ \frac{\partial}{\partial b}(\boldsymbol{x_n^T w} + b) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \boldsymbol{1_n}. \tag{C-2}$$

Por otro lado, se obtiene la derivada del vector de valores estimados respecto a la función afín. Teniendo en cuenta que la estimación consiste en aplicar la función de activación (la cual es una operación elemento a elemento) a la función afín, si

$$g'(x) = \frac{d}{dx}g(x)$$

es la derivada de la función de activación, entonces la derivada del vector de valores estimados respecto a la función afín está dada por:

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} = \frac{\partial \boldsymbol{g}(\boldsymbol{z})}{\partial \boldsymbol{z}} = \operatorname{diag}\left(\left\{g'(z_i)\right\}_{i=1}^{n}\right). \tag{C-3}$$

Finalmente, se obtiene la derivada parcial de la función de costo respecto a los valores ajustados. Teniendo en cuenta que la función de costo puede representarse como:

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n} c(\hat{y}_i, y_i),$$

y si denotamos como $\boldsymbol{c}$ el vector resultante de la operación elemento a elemento entre los vectores $\widehat{\boldsymbol{y}}$ y $\boldsymbol{y}$, con la función $c(\widehat{y}_i, y_i)$, es decir,

$$\boldsymbol{c} = \begin{bmatrix} c(\hat{y_1}, y_1) & c(\hat{y_2}, y_2) & \dots & c(\hat{y_n}, y_n) \end{bmatrix}^T,$$

entonces, la función de costo puede escribirse como la reducción vectorial:

$$J(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n} c(\hat{y}_i, y_i) = \frac{1}{n}\operatorname{sum}(\boldsymbol{c}). \tag{C-4}$$

Ahora bien, utilizando la regla de la cadena, se obtiene que la derivada parcial de la función de costo respecto al vector de valores ajustados está dada por:

$$\frac{\partial J}{\partial \hat{\boldsymbol{y}}} = \frac{\partial J}{\partial \boldsymbol{c}}\frac{\partial \boldsymbol{c}}{\partial \hat{\boldsymbol{y}}} = \frac{1}{n}\mathbf{1}_{\boldsymbol{n}}^T \operatorname{diag}\left(\left\{c_{\hat{y}}(\hat{y}_i, y_i)\right\}_{i=1}^{n}\right), \tag{C-5}$$

donde $c_{\hat{y}}(\hat{y}, y) = \frac{\partial c(\hat{y}, y)}{\partial \hat{y}}$.

De otro lado, por medio de la regla de la cadena, podemos utilizar las derivadas parciales calculadas previamente para obtener las derivadas parciales (gradiente en el caso de $\boldsymbol{w}$) de la función de costo respecto a los parámetros del modelo:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{w}} = \frac{1}{n}\mathbf{1}_{\boldsymbol{n}}^T \operatorname{diag}\left(\left\{c_{\hat{y}}(\hat{y}_i, y_i)\right\}_{i=1}^{n}\right) \operatorname{diag}\left(\left\{g'(z_i)\right\}_{i=1}^{n}\right) \boldsymbol{A},$$

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{1}{n}\left[c_{\hat{y}}(\hat{y}_1, y_1)g'(z_1), c_{\hat{y}}(\hat{y}_2, y_2)g'(z_2), \ldots, c_{\hat{y}}(\hat{y}_n, y_n)g'(z_n)\right]^T \boldsymbol{A}, \tag{C-6}$$

y denotando $\boldsymbol{c_{\hat{y}}(\hat{y}, y)}$ al vector resultante dado por,

$$\boldsymbol{c_{\hat{y}}(\hat{y}, y)} = \left[c_{\hat{y}}(\hat{y_1}, y_1), \quad c_{\hat{y}}(\hat{y_2}, y_2), \quad \ldots \quad c_{\hat{y}}(\hat{y_n}, y_n)\right]^T, \tag{C-7}$$

y $\boldsymbol{g'(z)}$ a la operación elemento a elemento:

$$\boldsymbol{g'(z)} = \left[g'(z_1), \quad g'(z_2), \quad \ldots \quad g'(z_n)\right]^T, \tag{C-8}$$

entonces se puede utilizar el producto Hadamard para expresar de forma más simple a (C-6)

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{1}{n}\left(\boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)}\right)^T \boldsymbol{A}. \tag{C-9}$$

De forma similar, para la derivada parcial respecto al sesgo tenemos:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial b} = \frac{1}{n}\mathbf{1}_{\boldsymbol{n}}^T \text{diag}\left(\left\{c_{\hat{y}}(\hat{y}_i, y_i)\right\}_{i=1}^n\right)\text{diag}\left(\left\{g'(z_i)\right\}_{i=1}^n\right)\mathbf{1}_{\boldsymbol{n}},$$

$$\frac{\partial J}{\partial b} = \frac{1}{n}\text{sum}\left(\boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)}\right). \tag{C-10}$$

Como previamente se mencionó, las derivadas parciales con respecto a los parámetros se utilizan en la actualización de los valores de los parámetros en el descenso gradiente (ver Anexo A para mayor detalle), sin embargo, la notación utilizada para obtener estas derivadas conduce a gradientes con dimensiones transpuestas a las del vector de parámetros. Para facilitar la aplicación de los algoritmos, las derivadas parciales se presentarán en una notación equivalente a trasponer los cálculos presentados hasta ahora. Así, se obtienen las siguientes ecuaciones:

$$\frac{\partial J}{\partial \boldsymbol{z}} = \frac{\partial J}{\partial \boldsymbol{\hat{y}}}\frac{\partial \boldsymbol{\hat{y}}}{\partial \boldsymbol{z}} = \frac{1}{n}\left(\left(\boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)}\right)^T\right)^T = \frac{1}{n}\boldsymbol{c_{\hat{y}}(\hat{y}, y)} \odot \boldsymbol{g'(z)}, \tag{C-11}$$

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{w}} = \boldsymbol{A^T}, \tag{C-12}$$

$$\frac{\partial \boldsymbol{z}}{\partial b} = \mathbf{1}_{\boldsymbol{n}}^{\boldsymbol{T}}, \tag{C-13}$$

$$\frac{\partial J}{\partial \boldsymbol{w}} = \boldsymbol{A}^{\boldsymbol{T}} \frac{1}{n} \boldsymbol{c}_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \odot \boldsymbol{g}'(\boldsymbol{z}), \tag{C-14}$$

$$\frac{\partial J}{\partial b} = sum\left(\frac{1}{n} \boldsymbol{c}_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) \odot \boldsymbol{g}'(\boldsymbol{z}).\right) \tag{C-15}$$

Bibliografía

Aghababaei Jazi, M., & Alamatsaz, M. (2012). Two new thinning operators and their applications. *Global Journal of Pure and Applied Mathematics*, *8*, 13-28.

Allende, H., Moraga, C., & Salas, R. (2002). Artificial neural networks in time series forecasting: a comparative analysis. *Kybernetika*, *38*(6), 685-707.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer Normalization.

Bahdanau, D., Cho, K., & Bengio, Y. (2016a). Neural Machine Translation by Jointly Learning to Align and Translate.

Bahdanau, D., Cho, K., & Bengio, Y. (2016b). Neural Machine Translation by Jointly Learning to Align and Translate.

Bandara, K., Shi, P., Bergmeir, C., Hewamalage, H., Tran, Q., & Seaman, B. (2019). Sales Demand Forecast in E-commerce Using a Long Short-Term Memory Neural Network Methodology. En T. Gedeon, K. W. Wong & M. Lee (Eds.), *Neural Information Processing* (pp. 462-474). Springer International Publishing.

Byrd, R. H., Schnabel, R. B., & Shultz, G. A. (1987). A Trust Region Algorithm for Nonlinearly Constrained Optimization. *SIAM Journal on Numerical Analysis*, *24*(5), 1152-1170. Consultado el 7 de mayo de 2023, desde http://www.jstor.org/stable/2157645

Chollet, F. (2017). *Deep Learning with Python* (1st). Manning Publications Co.

Christou, V., & Fokianos, K. (2015). On count time series prediction. *Journal of Statistical Computation and Simulation*, *85*(2), 357-373. https://doi.org/10.1080/00949655.2013.823612

Davis, R. A., Fokianos, K., Holan, S. H., Joe, H., Livsey, J., Lund, R., Pipiras, V., & Ravishanker, N. (2021). Count Time Series: A methodological Review. *Journal of the American Statistical Association, 116*, 1533-1547. https://doi.org/10.1080/01621459.2021.1904957

Dufour, J.-M. (2008). Estimation of ARMA models by maximum likelihood. https://jeanmariedufour.github.io/ResE/Dufour_2008_C_TS_ARIMA_Estimation.pdf

Dunsmuir, W. T. (2016). Generalized Linear Autoregressive Moving Average Models. En R. A. Davis, S. H. Holan, R. Lund & N. Ravishanker (Eds.). CRC Press.

Excoffier, M., Gicquel, C., & Jouini, O. (2016). A joint chance-constrained programming approach for call center workforce scheduling under uncertain call arrival forecasts. *Computers & Industrial Engineering*, *96*, 16-30. https://doi.org/https://doi.org/10.1016/j.cie.2016.03.013

Farsani, R., Pazouki, E., & Jecei, J. (2021). A Transformer Self-Attention Model for Time Series Forecasting. *Journal of Electrical and Computer Engineering Innovations*, *9*, 1-10. https://doi.org/10.22061/JECEI.2020.7426.391

Fearnhead, P. (2011). MCMC for State Space Models. En S. Brooks, A. Gelman, G. Jones & X.-L. Meng (Eds.). Chapman; HALL/CRC.

Feng, C., Li, L., & Sadeghpour, A. (2020). A comparison of residual diagnosis tools for diagnosing regression models for count data. *BMC Medical Research Methodology*, *20*, 1-21. https://doi.org/10.1186/s12874-020-01055-2

Ferland, R., Latour, A., & Oraichi, D. (2006). Integer-Valued GARCH Process. *Journal of Time Series Analysis*, *27*(6), 923-942. https://doi.org/https://doi.org/10.1111/j.1467-9892.2006.00496.x

Fokianos, K. (2012). Count Time Series Models. *Handbook of Statistics*, *30*, 315-347. https://doi.org/10.1016/B978-0-444-53858-1.00012-0

Fokianos, K., Rahbek, A., & Tjøstheim, D. (2009). Poisson Autoregression. *Journal of the American Statistical Association*, *104*(488), 1430-1439. Consultado el 15 de marzo de 2023, desde http://www.jstor.org/stable/40592351

Fokianos, K., & Tjøstheim, D. (2011). Log-linear Poisson autoregression. *Journal of Multivariate Analysis*, *102*(3), 563-578. https://doi.org/https://doi.org/10.1016/j.jmva.2010.11.002

Gamerman, D., Abanto-Valle, C., Silva, R., & Martins, T. (2016). Dynamic Bayesian Models for Discrete-Valued Time Series. En R. A. Davis, S. H. Holan, R. Lund & N. Ravishanker (Eds.). CRC Pess.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* [http://www.deeplearningbook.org]. MIT Press.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770-778. https://doi.org/10.1109/CVPR.2016.90

Hewamalage, H., Bergmeir, C., & Bandara, K. (2022). Global models for time series forecasting: A Simulation study. *Pattern Recognition*, *124*, 108441. https://doi.org/https://doi.org/10.1016/j.patcog.2021.108441

Hoppe, R. W. (2006). Chapter 4 Sequential Quadratic Programming. https://www.math.uh.edu/~rohop/fall_06/Chapter4.pdf

Hyndman, R. J. Focused Workshop: Synthetic Data — Generating time series. En: 2021. https://www.youtube.com/watch?v=F3lWECtFa44&ab_channel=AustralianDataScienceNetwor

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3rd). OTexts.

Hyndman, R. J., Kang, Y., Montero-Manso, P., O'Hara-Wild, M., Talagala, T., Wang, E., & Yang, Y. (2023). *tsfeatures: Time Series Feature Extraction* [https://pkg.robjhyndman.com/tsfeat https://github.com/robjhyndman/tsfeatures].

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679-688. https://doi.org/https://doi.org/10.1016/j.ijforecast.2006.03.001

Jia, Y. (2018). *Some Models for Count TimeSeries* (Tesis doctoral). Clemson University. 105 Sikes Hall, Clemson, SC 29634, Estados Unidos. https://tigerprints.clemson.edu/all_dissertations/2213

Kang, Y., Hyndman, R. J., & Li, F. (2020). GRATIS: GeneRAting TIme Series with diverse and controllable characteristics. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, *13*(4), 354-376. https://doi.org/10.1002/sam.11461

Liboschik, T., Fokianos, K., & Fried, R. (2017). tscount: An R Package for Analysis of Count Time Series Following Generalized Linear Models. *Journal of Statistical Software*, *82*(5), 1-51. https://doi.org/10.18637/jss.v082.i05

Lund, R., & Livsey, J. (2016). Renewal-Based Count Time Series. En R. A. Davis, S. H. Holan, R. Lund & N. Ravishanker (Eds.). CRC Press.

Makridakis, S. (1993). Accuracy measures: theoretical and practical concerns. *International Journal of Forecasting*, *9*(4), 527-529. https://doi.org/https://doi.org/10.1016/0169-2070(93)90079-3

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward. *Plos One*. https://doi.org/10.1371/journal.pone.0194889

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 Competition: 100,000 time series and 61 forecasting methods [M4 Competition]. *International Journal of Forecasting*, *36*(1), 54-74. https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.04.014

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems [Software available from tensorflow.org]. https://www.tensorflow.org/

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*. https://doi.org/10.1007/BF02478259

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

Montero-Manso, P., & Hyndman, R. J. (2021). Principles and algorithms for forecasting groups of time series: Locality and globality. *International Journal of Forecasting*, *37*(4), 1632-1653. https://doi.org/https://doi.org/10.1016/j.ijforecast.2021.03.004

Nariswari, R., & Pudjihastuti, H. (2019). Bayesian Forecasting for Time Series of Count Data [The 4th International Conference on Computer Science and Computational

Intelligence (ICCSCI 2019) : Enabling Collaboration to Escalate Impact of Research Results for Society]. *Procedia Computer Science*, *157*, 427-435. https://doi.org/https://doi.org/10.1016/j.procs.2019.08.235

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)*, *135*(3), 370-384. Consultado el 13 de enero de 2024, desde http://www.jstor.org/stable/2344614

Ng, A. Y., Katanforoosh, K., & Mourri, Y. B. (2023). Neural Networks and Deep Learning [MOOC]. Coursera. https://www.coursera.org/learn/neural-networks-deep-learning

Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023). A Time Series is Worth 64 Words: Long-term Forecasting with Transformers.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

of Transportation, N. D. (2017). Bicycle Counts for East River Bridges (Historical) [Daily total of bike counts conducted monthly on the Brooklyn Bridge, Manhattan Bridge, Williamsburg Bridge, and Queensboro Bridge. https://data.cityofnewyork.us/Transportation/Bicycle-Counts-for-East-River-Bridges-Historical-/gua4-p9wg].

Parr, T., & Howard, J. (2018). The Matrix Calculus You Need For Deep Learning.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825-2830.

Phuong, M., & Hutter, M. (2022). Formal Algorithms for Transformers.

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. https://www.R-project.org/

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, *abs/1609.04747*. http://arxiv.org/abs/1609.04747

Rue, H., Martino, S., & Chopin, N. (2009). Approximate Bayesian Inference for Latent Gaussian models by using Integrated Nested Laplace Approximations. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *71*(2), 319-392. https://doi.org/10.1111/j.1467-9868.2008.00700.x

Sathish, V., Mukhopadhyay, S., & Tiwari, R. (2020). ARMA Models for Zero Inflated Count Time Series. https://doi.org/10.48550/ARXIV.2004.10732

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85-117. https://doi.org/10.1016/j.neunet.2014.09.003

Seabold, S., & Perktold, J. (2010). statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference*.

Shenstone, L., & Hyndman, R. J. (2005). Stochastic models underlying Croston's method for intermittent demand forecasting. *Journal of Forecasting*, *24*(6), 389-402. https://doi.org/https://doi.org/10.1002/for.963

Shmueli, G., Bruce, P. C., Yahav, I., Patel, N. R., & Lichtendahl Jr., K. C. (2018). *Data mining for business analytics*. Wiley.

Shmueli, G., & Lichtebdahl, K. C. (2018). *Practical time series forecasting with R*. Axelrod Schnall Publishers.

Shrivastava, S. (2020). Cross Validation in Time Series. https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4

Smith, T. G. (2017). *pmdarima: ARIMA estimators for Python*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929-1958. http://jmlr.org/papers/v15/srivastava14a.html

Terven, J., Cordova-Esparza, D. M., Ramirez-Pedraza, A., & Chavez-Urbiola, E. A. (2023). Loss Functions and Metrics in Deep Learning.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998-6008. http://arxiv.org/abs/1706.03762

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261-272. https://doi.org/10.1038/s41592-019-0686-2

Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., & Sun, L. (2023). Transformers in Time Series: A Survey.

Zeng, A., Chen, M.-H., Zhang, L., & Xu, Q. (2022). Are Transformers Effective for Time Series Forecasting? *AAAI Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:249097444

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into Deep Learning. *arXiv preprint arXiv:2106.11342*.