UNMDP - Facultad de Ingeniería - Departamento de Informática

Desarrollo de un modelo predictivo de Aprendizaje Profundo para aproximar la potencia eléctrica de una planta fotovoltaica

Autores

Marinucci, Lorenzo (lorenmarinucci@gmail.com)
Palomeque, Lucía Lourdes (lucialourdespalomeque@gmail.com)

Directora

Dra. Seijas, Leticia María

Co-Director

Dr. Ing. González, Sergio Alejandro

Proyecto final para optar al grado de Ingeniero/a en Informática

Mar del Plata, 14 de diciembre de 2023

UNMDP - Facultad de Ingeniería - Departamento de Informática

Desarrollo de un modelo predictivo de Aprendizaje Profundo para aproximar la potencia eléctrica de una planta fotovoltaica

Autores

Marinucci, Lorenzo (lorenmarinucci@gmail.com)
Palomeque, Lucía Lourdes (lucialourdespalomeque@gmail.com)

Directora

Dra. Seijas, Leticia María

Co-Director

Dr. Ing. González, Sergio Alejandro

Proyecto final para optar al grado de Ingeniero/a en Informática

Mar del Plata, 14 de diciembre de 2023

Índice general

Agradecimientos

En primer lugar, queremos agradecer a nuestras familias y amigos por el apoyo incondicional durante todo el transcurso de la carrera. Muchas gracias por acompañarnos en este camino y por darnos fuerzas en cada momento, nada de esto hubiese sido posible sin ustedes a nuestro lado.

También agradecemos a nuestros directores, la Dra. Leticia María Seijas y el Dr. Ing. Sergio Alejandro González, por guiarnos durante toda la extensión del proyecto. Gracias por su ayuda y excelente predisposición a lo largo de este camino.

A su vez, le queremos agradecer a todos los docentes de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata por su gran desempeño a la hora de formarnos como ingenieros.

Finalmente, hacemos una mención especial a nuestras mascotas: Chicha, Oso, Ciro y Perla. Gracias por todo su amor y por estar siempre a nuestro lado durante estos años de estudio.

Resumen

El presente trabajo expone el proceso de construcción de un modelo predictivo de *Deep Learning*, aplicado a la predicción de la potencia eléctrica generada por una planta fotovoltaica. El caso de estudio es la planta del Instituto de Investigaciones Científicas y Tecnológicas en Electrónica (ICyTE), situada en la terraza de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata.

En primer lugar, se realizó un estudio abarcativo sobre los modelos pertenecientes al estado del arte de la problemática. Luego, a través de diversas etapas de experimentación, se optimizaron los modelos hasta minimizar el error predictivo. Se obtuvo una gran variedad de modelos de buena *performance*, entre los cuales el mejor se constituyó de una arquitectura *Long Short-Term Memory* (LSTM) de 3 capas ocultas. El modelo minimizó el error de porcentaje medio absoluto (MAPE) a un 9,46 % (±0,20 %), lo que representa una mejora del 6,8 % respecto a los reportados previamente por el ICyTE. A su vez, se encuentran cercanos a aquellos hallados en la literatura para ciudades con condiciones climáticas similares a Mar del Plata.

Por otra parte, se llevó a cabo el proceso de desarrollo de un producto de software que permite utilizar el modelo implementado, modificar sus características y visualizar y exportar los resultados generados. Asimismo, la herramienta cuenta con las capacidades necesarias para realizar un análisis del comportamiento y las propiedades de las variables climáticas que influyen en la predicción de potencia eléctrica.

Además de describir los procesos técnicos realizados, se incluye una reflexión acerca de las diferencias con la planificación original, los desvíos, el grado de cumplimiento de los objetivos establecidos y el aprendizaje experimentado al desarrollar el trabajo final de la carrera.

Palabras clave: Aprendizaje Profundo, LSTM, GRU, Bi-LSTM, mecanismos de atención, predicción de la potencia fotovoltaica.

Abstract

This paper presents the building process of a Deep Learning predictive model, applied to the prediction of the electrical power generated by a photovoltaic plant. The study case is the plant of the Instituto de Investigaciones Científicas y Tecnológicas en Electrónica (ICyTE), located at the terrace of the Faculty of Engineering of the Universidad Nacional de Mar del Plata.

First, a comprehensive study of the state of the art models of the problem was carried out. Then, through several experimentation stages, the models were optimized until the predictive error was minimized. A wide variety of good-performing models were obtained, the best of which consisted of a 3-hidden-layer Long Short-Term Memory (LSTM) architecture. The model minimized the mean absolute percentage error (MAPE) to 9.46 %, a 6.8 % improvement over those previously reported by the ICyTE. At the same time, they are close to those found in the literature for cities with climatic conditions similar to Mar del Plata.

On the other hand, the development process of a software product that allows using the implemented model, modifying its characteristics, and visualizing and exporting the generated results was carried out. Likewise, the tool has the necessary capabilities to perform an analysis of the behavior and properties of the climatic variables that influence the prediction of electric power.

In addition to describing the technical processes carried out, a reflection about the differences with the original planning, the deviations, the degree of fulfillment of the established objectives, and the acquired knowledge in the development of the final project of the career is included.

Keywords: *Deep Learning, LSTM, GRU, Bi-LSTM, attention, photovoltaic power prediction.*

Nomenclatura

ARIMA	Modelo autorregresivo integrado de media móvil
CNN	Red neuronal convolucional
CPU	Unidad central de procesamiento
GPU	Unidad de procesamiento gráfico
GRU	*Gated Recurrent Unit*
ICyTE	Instituto de Investigaciones Científicas y Tecnológicas en Electrónica (UNMDP-CONICET)
LSTM	*Long Short-Term Memory*
MAE	Error absoluto medio
MAPE	Error de porcentaje medio absoluto
MLP	Perceptrón Multicapa
MSE	Error cuadrático medio
NARX	Red autorregresiva no lineal con entradas exógenas
NWP	*Numerical Weather Prediction*
PVUSA	*Photovoltaics for Utility Scale Applications*
RBF	*Radial basis function*
ReLU	Unidad lineal rectificada
RMSE	Raíz del error cuadrático medio
RNA	Red neuronal artificial
RNN	Red neuronal recurrente
SVM	*Support Vector Machine*
SVR	*Support Vector Regression*
TPU	Unidad de procesamiento tensorial
XLA	*Accelerated Linear Algebra*

Capítulo 1

Introducción

Una planta fotovoltaica es una central eléctrica encargada de la conversión de luz solar en electricidad. Este proceso se realiza mediante el efecto fotovoltaico que se produce en los módulos o celdas de los paneles solares. En el año 2015, el ICyTE instaló una planta fotovoltaica experimental en la terraza del edificio principal de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata, la cual es utilizada para producir electricidad que se consume dentro del establecimiento.

En el último tiempo, los sistemas de energía han cambiado notablemente su forma de ser diseñados y administrados. La posibilidad de generar energía de forma particular e independiente ha permitido un consumo flexible del recurso. Además de su uso interno, es posible inyectar los excedentes a la red eléctrica pública, de forma que puedan ser aprovechados por otro consumidor. Para una gestión eficiente del flujo de potencia generado, las plantas fotovoltaicas utilizan un proceso conocido como *photovoltaic power forecasting* (predicción de energía fotovoltaica), en donde se busca predecir la producción de una planta en un horizonte temporal determinado.

Existen diferentes horizontes en la predicción de potencia eléctrica. Uno de ellos es el de *very short-term* (muy corto plazo), que incluye a los procesos capaces de pronosticar la potencia generada entre 5 y 60 minutos de antelación, lapso que podría extenderse hasta las 4 horas. Por otra parte, existe el horizonte *short-term* (corto plazo), que a diferencia del anterior se puede extender hasta las 24 horas. Finalmente, se encuentran el *medium-term* (mediano plazo) y el *long-term* (largo plazo), donde el objetivo es predecir la potencia en el rango de semanas y meses, respectivamente.

Las predicciones son realizadas a través de modelos que realizan sus pronósticos a partir de variables climáticas como la irradiancia solar, la humedad, la temperatura del aire y la velocidad del viento, entre otras. Para la obtención de las condiciones climáticas del horizonte a predecir se utiliza una metodología conocida como *Numerical Weather Prediction* (NWP), que realiza pronósticos meteorológicos en base a modelos físicos.

1.1. Estado del arte

En la actualidad, existe un gran número de modelos predictivos que se utilizan para la resolución de la problemática. Entre ellos, se encuentran los modelos clásicos (en particular los estadísticos) y aquellos más novedosos, basados en *Machine Learning*. Dentro de estos últimos, el foco está puesto sobre el área del aprendizaje profundo o *Deep Learning*.

1.1.1. Modelos estadísticos

Uno de los modelos más referidos en la literatura es ARIMA (*Autoregressive Integrated Moving Average*). ARIMA únicamente utiliza valores de potencia anteriores en el tiempo para generar predicciones. En caso de requerir del uso de más variables climáticas, se debe recurrir a la utilización de otras variantes de mayor complejidad. Esto conlleva a un gran aumento del costo y tiempo de ejecución del modelo. Adicionalmente, se ha demostrado que es muy ineficiente al utilizar secuencias de datos que no son lineales [1].

Si bien ARIMA es utilizado para la resolución de la problemática, es superado notoriamente por otros tipos de modelos, como los basados en *Machine Learning*. Para el caso de *"Short-Term Photovoltaic Power Forecasting Based on Long Short Term Memory Neural Network and Attention Mechanism"* de Zhou *et al.* [2], se superó en un 15,21 % al modelo ARIMA con la utilización del Perceptrón Multicapa (MLP),

un modelo de *Machine Learning*. Por otra parte, en la publicación *"Performance Analysis of Statistical, Machine Learning and Deep Learning Models in Long-Term Forecasting of Solar Power Production"* de Sedai *et al.* [3], un modelo MLP supera en un 83,33 % al modelo ARIMA planteado.

Otro de los modelos estadísticos de más renombre es PVUSA (*Photovoltaics for Utility Scale Applications*), que fue desarrollado durante la década del 90 [4]. PVUSA aplica una regresión para obtener la potencia eléctrica generada por una planta fotovoltaica, la cual es dependiente de parámetros climáticos y ambientales, como la irradiancia solar, el viento y la temperatura. El ICyTE ha utilizado este modelo para el cálculo del PR (*Performance Ratio*) estimado de la planta fotovoltaica de la Facultad de Ingeniería [5]. Esta es una medida que se encarga de determinar la eficacia con la cual una planta convierte energía solar en energía eléctrica a partir de la potencia nominal de los módulos utilizados [6].

Si bien el modelo predictivo hace uso de múltiples variables climáticas, no tiene en consideración la humedad relativa, ya que fue desarrollado para localizaciones con climas secos. Dado que se ha demostrado que los altos niveles de humedad pueden afectar el rendimiento de los paneles fotovoltaicos, se considera que el modelo es insuficiente para las predicciones en este tipo de ambientes, como presenta la ciudad de Mar del Plata.

1.1.2. Modelos basados en *Machine Learning*

Los modelos basados en *Machine Learning* se centran en imitar el comportamiento inteligente de los humanos a partir de modelos provenientes de distintas disciplinas. Dentro de este tipo de modelos existen técnicas clásicas, como el MLP y las *Support Vector Regression* (SVR). Se tratan de los modelos más básicos de *Machine Learning*. Una de las ventajas que proveen es la utilización de más de una variable para la predicción de potencia.

Si bien los modelos clásicos continúan siendo aplicados para la resolución del problema, se han visto superados por el rendimiento de los modelos de *Deep Learning*. Para la publicación *"Short-Term Photovoltaic Power Forecasting Based on Long Short Term Memory Neural Network and Attention Mechanism"* de Zhou *et al.* [2], se superó a un modelo MLP con uno de *Deep Learning* en un 3,65 %. Por parte de *"Forecasting Solar Power Using Long-Short Term Memory and Convolutional Neural Networks"* de Lee *et al.* [7], se estableció un modelo SVR que fue superado en un 44,54 % por otro modelo de *Deep Learning*. Por último, en la publicación *"Solar Power Forecasting Using Deep Learning Techniques"* de Elsaraiti y Merabet [8], se ha superado también a un modelo MLP con uno de *Deep Learning* en un 63,58 %.

1.1.3. Modelos de *Deep Learning*

Los modelos de *Deep Learning* constituyen una mejora representativa sobre los modelos de *Machine Learning*. Su beneficio es el hecho de que permiten procesar altos volúmenes de datos y modelar sus conceptos en un alto nivel de abstracción a partir de la utilización de arquitecturas complejas, con numerosas capas, neuronas y conexiones. En la última década, los modelos de aprendizaje profundo han acaparado los mejores resultados en un sin fin de áreas, tales como el procesamiento del lenguaje natural, la visión artificial y el reconocimiento del habla, entre otras.

Existen modelos de *Deep Learning*, como las redes neuronales recurrentes (RNN), que han sido aplicados con éxito para la predicción de potencia eléctrica. Algunos ejemplos de RNN son las redes NARX (*Nonlinear Autoregressive with eXogenous input*) [9] y las *Long Short Term Memory* (LSTM) [7] [10] [11], estas últimas con mayor relevancia en la actualidad. Otro modelo ampliamente referido en la literatura es *Gated Recurrent Unit* (GRU) [12] [13], una versión modificada de LSTM.

Si bien las redes LSTM y GRU proveen resultados de alta precisión, existen variantes adicionales que han logrado superarlos. Dentro ellas, se encuentran las arquitecturas bidireccionales (Bi-LSTM y Bi-GRU) [14] [15]. A su vez, se incluyen también las que incorporan mecanismos de atención. Para el caso de la publicación *"Short-Term Photovoltaic Power Forecasting Based on Long Short Term Memory Neural Network and Attention Mechanism"* de Zhou *et al.* [2], se logra superar a un modelo LSTM en un 5,73 % con la implementación de un mecanismo de atención.

Adicionalmente, un amplio campo de estudio se encuentra en el dominio de las redes convolucionales. Se observa en la literatura que este tipo de modelos ofrecen gran precisión en los resultados obtenidos [16] [17] [18], pero debido a su inherente complejidad y a la extensión determinada para el proyecto, su estudio excede el alcance del presente trabajo.

1.2. Problema a resolver

En el pasado, el ICyTE ha intentado generar un método predictivo a través de la aplicación de modelos estadísticos y *Machine Learning*. Si bien el instituto ha realizado experimentación preliminar, el primer trabajo formal fue desarrollado en el presente año y contó con el co-director del proyecto S. A. González como uno de sus integrantes [19]. Mediante el uso de modelos de *Deep Learning*, lograron alcanzar un error de porcentaje medio absoluto (MAPE) de 10.15 % para predicciones con horizonte temporal de un día. Como siguiente paso en la experimentación, el instituto está en búsqueda de un modelo predictivo que también utilice técnicas de aprendizaje profundo y que provea aún más precisión.

La obtención de este modelo permitiría una toma de decisiones más acertada respecto a la arbitración y la gestión del flujo de la potencia eléctrica generada por la planta. Dichas actividades se enmarcan dentro del contexto del mercado energético, ya que el conocimiento obtenido sería de gran utilidad para decidir si resulta más beneficioso autoconsumir la potencia generada o efectuar la venta de la misma al mercado. Minimizar el error porcentual en la estimación es muy importante, dado que este resulta directamente proporcional al costo que afronta el distribuidor por la energía generada en el horizonte temporal analizado.

Adicionalmente, es de interés que el modelo obtenido logre resultados cercanos o superiores a los publicados en la literatura. Para ello, se necesita un relevamiento sobre el estado del arte de las distintas variantes que son aplicados en la actualidad para la predicción de la potencia eléctrica generada. Estas publicaciones serían de utilidad a modo de base comparativa para los modelos construidos a lo largo del proyecto.

Con los productos propuestos para el proyecto, el ICyTE contará con amplia información sobre modelos predictivos y una herramienta de gran utilidad para la gestión del flujo de potencia eléctrica generada. De esta forma, el instituto tendrá una mayor previsión sobre la cantidad de electricidad que la planta generará dentro de un horizonte temporal determinado. El modelo predictivo permitirá tomar con mayor certeza decisiones respecto al autoconsumo o la venta de la potencia producida.

1.3. Objetivos del proyecto

1.3.1. Objetivo global

El objetivo global del proyecto es la construcción de un modelo predictivo de *Deep Learning*, que permita aproximar la potencia eléctrica que generará una planta fotovoltaica. Se espera que dicho modelo obtenga predicciones superiores a las producidas actualmente por el ICyTE, y cercanas o superiores a las expuestas en el estado del arte. Por último, se debe proveer al ICyTE de un software que permita la utilización del modelo predictivo final y la visualización de sus resultados. A su vez, el software deberá brindar las herramientas necesarias para analizar el comportamiento de las variables climáticas registradas en la base de datos del ICyTE.

1.3.2. Objetivos específicos

- Adquirir las competencias necesarias para la gestión de un proyecto que incorpora tareas de investigación, experimentación y desarrollo de software.
- Analizar publicaciones y bibliografía relacionadas a la temática y adquirir conocimientos acerca de modelos de aprendizaje profundo, su funcionamiento y los resultados obtenidos con ellos. Por otra parte, a partir de la ejecución de conjuntos de pruebas, obtener habilidades para su óptima puesta a punto.
- Investigar acerca del estado del arte relacionado al software que se utiliza para trabajar con modelos predictivos.
- Analizar y realizar la puesta a punto de la base de datos otorgada por el ICyTE. A su vez, detectar los parámetros con mayor influencia en las predicciones a partir del análisis de la dependencia de la potencia generada sobre las variables climáticas registradas.

1.4. Alcance del proyecto

Dentro de los objetivos planteados, se definió el desarrollo de un software para la predicción de potencia eléctrica. Según lo estipulado al inicio del proyecto, el producto no contemplará la incorporación de librerías para el proceso de NWP, que es requerido para el uso diario del modelo. De acuerdo a lo establecido con el ICyTE, su incorporación será realizada en un proyecto futuro. Por este motivo, la herramienta únicamente proveerá el acceso al modelo predictivo y a los resultados que este produzca.

1.5. Institución solicitante

El ICyTE dio sus primeros pasos en el año 1983 como una asociación informal de 8 grupos de investigación de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata. Entre sus líneas de investigación se encuentran la Ingeniería Eléctrica, Electrónica y de la Información. En el año 2016, el CONICET aprobó la creación del instituto de investigación de forma oficial y en la actualidad cuenta con 10 grupos de investigación y 84 integrantes, entre los cuales se encuentran investigadores y becarios de diferentes áreas.

Capítulo 2

Estimaciones iniciales

2.1. Análisis FODA

Previo al inicio del proyecto, se realizó un análisis FODA. Este se encarga de establecer, como indican sus siglas, las fortalezas, oportunidades, debilidades y amenazas para un proyecto. Es de vital importancia, ya que permite identificar ventajas del equipo de trabajo y de la confección del proyecto, y a su vez encontrar posibles problemáticas o falencias internas y externas.

Fortalezas

- El equipo de trabajo es dirigido por la Dra. Leticia M. Seijas, quien posee un amplio recorrido en los campos de las Redes Neuronales Artificiales y el *Deep Learning*.
- El Dr. Sergio Alejandro González, co-director y referente funcional del proyecto, cuenta con gran experiencia dentro del área de la electrónica de potencia y su aplicación sobre energías renovables.
- Para el desarrollo del proyecto, el ICyTE pone a disposición del equipo una base de datos con distintas mediciones efectuadas sobre el arreglo de paneles fotovoltaicos desde el año 2015 hasta la actualidad. La misma posee todos los parámetros ambientales relevantes para la generación de predicciones sobre la potencia eléctrica producida, de acuerdo a lo establecido en la metodología PVUSA. Este recurso es de invaluable utilidad para el entrenamiento de los modelos de *Machine Learning*.
- El Dr. Sergio González pone a disposición del equipo el laboratorio del ICyTE como espacio de consulta y de trabajo durante la extensión del proyecto. De esta forma, la información que sea requerida será de sencillo acceso.
- La planta fotovoltaica se sitúa sobre la propia sede principal de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata. Esto la vuelve de fácil acceso en caso de ser requerido.
- Previo al comienzo del proyecto, el equipo de trabajo ha recabado una gran variedad de literatura y material informativo, tanto en el campo de la Redes Neuronales Artificiales y *Deep Learning*, como en el estado del arte sobre la predicción de potencia eléctrica en plantas fotovoltaicas.
- El ICyTE ha realizado un trabajo preliminar sobre modelos predictivos basados en *Machine Learning*, por lo que el equipo de trabajo cuenta con una base previa extremadamente útil para el planteamiento inicial de los modelos y su posterior comparación de resultados.

Oportunidades

- Un modelo predictivo más preciso permitiría al ICyTE gestionar de forma más eficiente la potencia eléctrica generada por la planta fotovoltaica, solidificando su participación dentro del mercado energético.
- A nivel mundial, existe una tendencia a la generación de energía distribuida y al autoconsumo, dando lugar a las *Smart Grids*, donde se utilizan herramientas informáticas (tales como los modelos predictivos) para mejorar la eficiencia en la producción y distribución de la potencia eléctrica

generada. Esto da lugar a una gran cantidad de modelos e información de referencia que puede ser aprovechada durante el desarrollo del proyecto.

- A nivel nacional, la generación distribuida de energía es fomentada a través del Régimen de Fomento a la Generación Distribuida de Energías Renovables, establecido por la Ley N° 27.424. Dicho régimen promueve la incorporación de sistemas generadores tales como los que posee el ICyTE. Esto genera en consecuencia un alto interés en temáticas como la predicción de potencia eléctrica generada. De esta manera, el proyecto adquiere una alta relevancia dentro del contexto energético sobre el cual se desarrolla.

Debilidades

- Los alumnos se enfrentan por primera vez a los modelos de *Machine Learning* y *Deep Learning* como herramientas para la predicción de potencia eléctrica.
- De la misma forma, este proyecto es su primer acercamiento al dominio de las plantas fotovoltaicas y su generación de potencia eléctrica.

Amenazas

- La metodología PVUSA ha sido aplicada de forma predominante en localizaciones con clima seco y de baja humedad. La ciudad de Mar del Plata difiere en estos aspectos, presentando un clima costero y con humedad alta. Dichas diferencias podrían dificultar la comparativa con modelos presentes en la literatura, donde la humedad como parámetro ambiental no es considerada relevante para efectuar los pronósticos.

2.2. Planificación

Previo al armado del protocolo del proyecto, se realizaron reuniones para presentar diversos temas centrales a modo de introducción a la problemática a tratar. Esto fue de gran ayuda para la comprensión del alcance y los beneficios que traería el desarrollo del proyecto.

Una vez obtenidos los conocimientos centrales requeridos para una comprensión general del problema a resolver, se procedió al armado de un diagrama de Gantt con las tareas a desarrollar durante el proyecto (ver figura 2.1). Para ello, se establecieron bloques generales que agrupaban tareas más específicas. El reparto de tareas se realizó de forma equitativa, de manera que se logre una división del trabajo eficiente. Ciertas tareas, como la escritura de documentación y la incorporación de conceptos teóricos, fueron asignadas a ambos integrantes. Asimismo, con el objetivo de dividir las cargas de trabajo, otras se planificaron de forma individual.

Inicialmente, el flujo principal de tareas fue diagramado en formato cascada. Si bien existían etapas que requerían necesariamente de la completitud de las anteriores (principalmente la 4), podía existir solapamiento entre otras. El único bloque que se estableció continuamente en paralelo fue el 3, ya que el análisis de la base de datos otorgada por el ICyTE y sus herramientas asociadas eran de utilidad durante el avance en el estudio de los modelos. Por otra parte, se estableció que las tareas de generación de conclusiones y finalización de la redacción del informe del bloque 7 podían comenzar a la vez que las pertenecientes a los bloques 5 y 6.

Al ser un proyecto con componentes de investigación y experimentación, ciertas tareas se asociaban a un alto grado de incertidumbre. En primer lugar, existía el riesgo de extenderse en aquellas relacionadas al estudio de los modelos y el estado del arte. Si bien se había realizado un primer acercamiento a la temática, el proyecto representaba la primera experiencia de los alumnos en el trabajo con modelos predictivos. Adicionalmente, podían surgir dificultades en la implementación de los modelos, dado que se desconocía cómo iba a ser el comportamiento de los conjuntos de datos a utilizar. Por los motivos anteriormente expuestos, existía la posibilidad de que la estimación de tiempos fuera optimista.

En la organización de las tareas a desarrollar se priorizó la documentación continua, de manera que en cada bloque se mantenga registro de lo realizado. El objetivo fue disminuir el tiempo de escritura total del presente informe, ya que su redacción sería realizada iterativamente y a partir de conocimientos recientemente adquiridos.

Una vez finalizado el diagrama, se estimó una dedicación semanal de 13 horas por cada integrante del equipo. A su vez, se definió una dedicación total de 41 semanas para el proyecto (1066 horas totales entre ambos estudiantes). Una vez obtenida una respuesta del protocolo por parte de la mesa directiva,

Identificador	Título de la tarea	Responsable	Duración (semanas)
1	**Estudio y relevamiento del estado del arte: modelos clásicos de Machine Learning (ML) y estadísticos**		
1.1	Definición y estudio de las características de las series temporales	Lucía y Lorenzo	3
1.2	Perceptrón multicapa: estudio del modelo y sus características	Lucía y Lorenzo	2
1.3	Perceptrón multicapa: estudio del estado del arte y aplicación sobre la temática	Lucía	2
1.4	Support Vector Regression (SVR): estudio del modelo y sus características	Lucía y Lorenzo	2
1.5	Support Vector Regression (SVR): estudio del estado del arte y aplicación sobre la temática	Lorenzo	2
1.6	Autorregresive integrated moving average (ARIMA): estudio del modelo y sus características	Lucía y Lorenzo	2
1.7	Autorregresive integrated moving average (ARIMA): estudio del estado del arte y aplicación sobre la temática	Lucía	1
1.8	Documentación y escritura	Lucía y Lorenzo	8
2	**Estudio y relevamiento del estado del arte: modelos de Deep Learning**		
2.1	Redes Neuronales Recurrentes (RNN): estudio de modelos, características y variantes de interés para el problema a resolver	Lucía y Lorenzo	4
2.2	Long Short Term Memory (LSTM): estudio del modelo y sus características y variantes	Lucía y Lorenzo	4
2.3	Long Short Term Memory (LSTM): estudio del estado del arte y aplicación sobre la temática	Lucía	5
2.6	Documentación y escritura	Lucía y Lorenzo	10
3	**Análisis de datos y herramientas a aplicar**		
3.1	Análisis de datos del conjunto de mediciones ambientales, provisto por el ICYTE. Definición de uso de los datos para el entrenamiento	Lucía y Lorenzo	7
3.2	Documentación y escritura	Lucía y Lorenzo	7
4	**Aplicación de estudios e investigaciones a la problemática**		
4.1	Selección y desarrollo de los modelos a utilizar	Lucía y Lorenzo	7
4.2	Optimización de los modelos y sus hiperparámetros	Lucía y Lorenzo	9
4.3	Ejecución y recopilación de resultados	Lucía y Lorenzo	5
4.4	Construcción de gráficos	Lucía	3
4.5	Análisis de los resultados obtenidos	Lorenzo	4
4.6	Documentación y escritura	Lucía y Lorenzo	13
5	**Resultados finales y comparativa con la literatura**		
5.1	Estudio y recopilación de bases de datos estándar para la comparativa de resultados	Lucía y Lorenzo	3
5.2	Ejecución de los modelos sobre las bases de datos seleccionadas	Lucía y Lorenzo	4
5.3	Construcción de gráficos	Lucía	2
5.4	Análisis de los resultados y comparativa. Conclusiones	Lorenzo	3
5.5	Documentación y escritura	Lucía y Lorenzo	7
6	**Desarrollo de software para la predicción de potencia eléctrica**		
6.1	Análisis de requerimientos funcionales y no funcionales	Lucía	1
6.2	Diseño de la solución	Lorenzo	1
6.3	Desarrollo del software	Lucía y Lorenzo	2
6.4	Testing y puesta a punto de la solución	Lucía y Lorenzo	1
6.5	Documentación y escritura	Lucía y Lorenzo	3
7	**Conclusiones finales y cierre del proyecto**		
7.1	Conclusiones finales del proyecto	Lucía y Lorenzo	5
7.2	Finalización del escrito	Lucía y Lorenzo	10

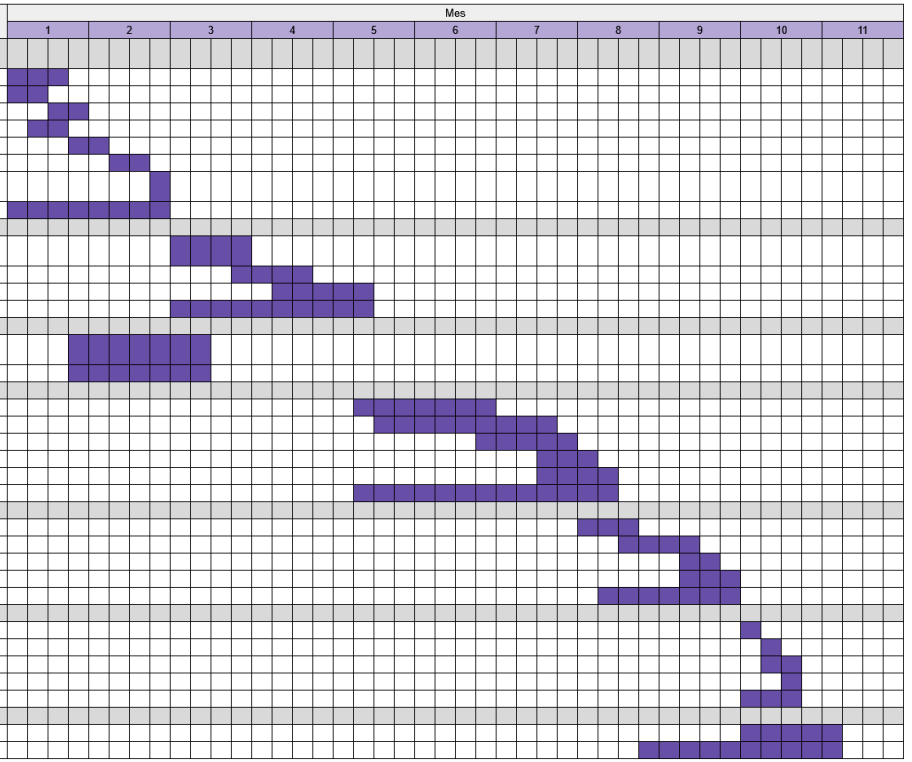


Figura 2.1: Diagrama de Gantt inicial.

se definió el día 03/10/2022 como fecha de inicio del proyecto y una finalización estimada para el día 17/07/2023.

2.2.1. Modificación sobre productos entregables

Como parte del trabajo a realizar en la Etapa 3, se definió el desarrollo de una herramienta de uso interno para el trabajo con la base de datos climática. El objetivo era poder realizar un análisis detallado de la información contenida en ella, de manera que pueda utilizarse de forma óptima durante la experimentación con los modelos predictivos.

Al comenzar el proceso de desarrollo, fue fundamental la ayuda del co-director S. A. González, quien definió los requerimientos necesarios para facilitar el estudio de la base de datos. A medida que avanzó el análisis, los requerimientos aumentaron de forma iterativa, al punto donde el desarrollo se volvió un producto de software completo.

En la diagramación inicial de los objetivos del proyecto, únicamente se había estipulado la construcción de un software con capacidades predictivas. Sin embargo, dado que el ICyTE no contaba con una herramienta de análisis de su base de datos, se decidió incorporar la funcionalidad desarrollada como parte del producto final. Dicha modificación únicamente requirió del desarrollo adicional de una interfaz de usuario, lo cual demandó de poca carga de trabajo por sobre la estimada originalmente (6 horas).

2.3. Análisis de riesgos

Se ha realizado un análisis acerca de los riesgos que pueden afectar el desarrollo y finalización del proyecto. Para ello, se han recopilado cada uno de ellos junto con su consecuencia. A su vez, se le ha asociado una probabilidad y un impacto a su ocurrencia. Ambas características pueden tomar un valor entre 1 y 3, siendo este último el mayor. Por otra parte, se calculó el peso para cada uno de los riesgos. Su valor se obtuvo a partir del producto entre la probabilidad y el impacto (ver tabla 2.1).

2.3.1. Planes de contingencia

Para los casos en donde el peso resultase mayor o igual a 6, se establecieron planes de contingencia (ver tabla 2.2). Un plan de contingencia es el curso de acción a realizar en caso de que un riesgo se torne realidad en algún punto del proyecto. Su definición es de extrema necesidad, ya que existe una alta probabilidad de ocurrencia del riesgo y su impacto sería grave para el proyecto.

Riesgo	**Consecuencia**	**Probabilidad de ocurrencia**	**Impacto**	**Peso**
Un integrante del equipo abandona el proyecto	Pérdida de recurso humano y conocimientos	1	3	3
Un integrante del equipo se ausenta del proyecto por un período largo de tiempo	Demoras en la finalización del proyecto y pérdida de mano de obra momentánea	1	3	3
Obtención de resultados que no superen los obtenidos por el ICyTE	El modelo obtenido no es de utilidad para el instituto	2	3	6
La planta fotovoltaica de la Facultad de Ingeniería deja de estar en funcionamiento	El proyecto deja de ser útil para el ICyTE	1	3	3
Limitaciones en las librerías para la implementación de modelos predictivos	Imposibilidad del armado de los modelos con los lenguajes y herramientas conocidos por el equipo de trabajo	2	3	6
Dificultades por inexperiencia en el estudio y trabajo con modelos predictivos	Demoras en el aprendizaje y retraso en la completitud de las tareas	3	2	6
El hardware de las computadoras personales de los integrantes no posee la potencia suficiente para la ejecución de conjuntos de pruebas complejos	Demoras excesivas en la ejecución de los modelos predictivos	3	3	9
Ausencia presencial del co-director	Imposibilidad de reuniones presenciales y de utilización del laboratorio del ICyTE como espacio de trabajo	3	2	6
No encontrar bases de datos comparativas de ciudades con clima similar a Mar del Plata para comprobar la robustez del modelo predictivo final	Complicaciones en la evaluación de la robustez del modelo con datos externos	3	2	6
Pérdida de interés en el proyecto por parte del ICyTE	Cancelación del proyecto	1	3	3

Tabla 2.1: Análisis de riesgos

Riesgo	**Plan de contingencia**
Obtención de resultados que no superen los obtenidos por el ICyTE	Repriorizar el análisis de ciertos modelos y analizar la posibilidad de incluir modelos adicionales al estudio (sin extender los plazos del proyecto y de acuerdo a la planificación inicial).
Limitaciones en las librerías para la implementación de modelos predictivos	Trabajar con otro tipo de herramientas o lenguajes de programación que se ajusten a dichos modelos. Esto implicaría una pequeña demora en la finalización del proyecto en caso de que estas sean desconocidas por el equipo.
Dificultades por inexperiencia en el trabajo con modelos predictivos	Como primeros pasos del proyecto, dedicarle tiempo prudencial al estudio de un modelo sencillo, con el objetivo de sentar los fundamentos para el trabajo con opciones más complejas
El hardware de las computadoras personales de los integrantes no posee la potencia suficiente para la ejecución de conjuntos de pruebas complejos	Investigar y experimentar con el uso de entornos de software en la nube que puedan proveer las características necesarias.
Ausencia presencial del co-director	Plantear reuniones de forma virtual, y en caso de ser necesaria la utilización del laboratorio, recurrir a algún integrante adicional del ICyTE.
No encontrar bases de datos comparativas de ciudades con clima similar a Mar del Plata para comprobar la robustez del modelo predictivo final	Investigar acerca de publicaciones relacionadas a la predicción de potencia eléctrica en ciudades con clima similar a Mar del Plata y realizar la comparativa en base a ellas.

Tabla 2.2: Planes de contingencia

Capítulo 3

Metodologías

3.1. Metodología de reunión

En el comienzo del proyecto, se planteó realizar de forma semanal reuniones presenciales en la Facultad de Ingeniería. Las mismas contaron con un esquema presentacional, en donde tanto los alumnos como los directores del proyecto exponían de forma oral y con soporte visual diferentes temas de índole teórico y experimental. Este tipo de dinámica fue de gran importancia, ya que permitió generar un espacio para profundizar aún más sobre distintas temáticas y resolver dudas de diferente índole.

Sin embargo, poco antes de la finalización del año 2022, el co-director del proyecto S. A. González debió viajar al exterior por motivos laborales. Debido a ello, el esquema presencial para las reuniones debió modificarse hacia uno virtual con una frecuencia semanal y en ocasiones quincenal. A pesar de este cambio en la dinámica de trabajo, se continuó realizando presentaciones teóricas a medida que nuevos temas eran introducidos al proyecto. A su vez, se mantuvo presente la posibilidad de reuniones presenciales a requerimiento con la directora del proyecto L. M. Seijas.

Finalmente, con el avance del trabajo y con cierta madurez en los conocimientos adquiridos, las reuniones se establecieron de forma más espaciada según la necesidad de los alumnos. En caso de requerir la resolución de dudas o exponer avances de gran relevancia sobre el proyecto, se generaba una reunión a demanda.

3.2. Metodología de estudio, desarrollo y experimentación de modelos

Al momento de incursionar sobre cada modelo predictivo, se realizó un estudio completo sobre su funcionamiento y sus capacidades. Por otra parte, se procedió a realizar un análisis sobre el estado del arte. Esta etapa fue de gran utilidad, ya que permitió obtener un panorama general sobre la estructura de los modelos y los resultados que se podían obtener con su utilización.

Una vez adquirida la base de conocimientos teóricos, se desarrolló cada modelo a partir de la utilización de librerías o herramientas de software. Posteriormente, se establecieron diferentes conjuntos de pruebas que fueron aplicados a los modelos desarrollados para comprobar su funcionamiento y establecer conclusiones en base a los resultados obtenidos. En un primer momento, las pruebas fueron realizadas de forma manual, con el objetivo de lograr una mayor comprensión del modelo y del impacto generado al modificar sus características. Posteriormente, se aplicaron metodologías más óptimas y automatizadas para aumentar la eficiencia en la ejecución de extensos conjuntos de pruebas.

Al momento de decidir si seguir profundizando sobre el modelo de estudio o virar hacia nuevas alternativas, se realizó un análisis de las capacidades observadas en la literatura y de los resultados obtenidos durante la experimentación.

3.3. Metodología de desarrollo de software

Como paso inicial en el desarrollo de software para el proyecto, se realizó un análisis de requerimientos con el co-director y referente funcional S. A. González del ICyTE. El establecimiento de las funcionalidades se realizó durante las instancias de reunión, en donde se mencionaron las necesidades fundamentales

que debían satisfacerse en los desarrollos pactados.

El proceso de construcción del software predictivo se realizó con una metodología tipo cascada, mientras que el software de análisis de datos se realizó de forma iterativa. Se utilizó Python como lenguaje en el desarrollo y se aplicó el uso de diversas librerías, tanto para el desarrollo de interfaces visuales como para el manejo de datos, entre otras. Por otra parte, se utilizó la plataforma Github para el manejo del versionado del proyecto.

En cuanto a la modalidad de *testing*, se realizaron diversos conjuntos de prueba para evaluar el funcionamiento de las partes centrales del software. En primer lugar, las funcionalidades granulares fueron cubiertas a partir de *tests* unitarios. Posteriormente, se realizaron pruebas de integración para verificar la interacción entre módulos. El objetivo del proceso de *testing* fue lograr el mayor porcentaje posible de cobertura sobre el código de la aplicación.

Por último, con el objetivo de facilitar posibles modificaciones posteriores, las funcionalidades fueron documentadas a lo largo del código. Adicionalmente, se creó un archivo explicativo para facilitar el uso de la herramienta por parte del personal del ICyTE.

Capítulo 4

Marco teórico

4.1. Series temporales

Una serie temporal se define como un conjunto de observaciones x_t, cada una registrada en un momento específico t. En base a su conjunto de tiempos T_0, se las caracteriza como discretas o continuas. Las series temporales discretas poseen observaciones espaciadas (por ejemplo, cada cinco minutos). En contraposición, las continuas las realizan continuamente a lo largo de un intervalo, como podría ser $T_0 = [0, 1]$ [20].

Se caracterizan por las siguientes componentes [21]:

- **Tendencia**: define si, a lo largo de un período, la serie temporal aumenta o decrece. Puede ser ascendente o descendente.
- **Estacionalidad**: patrón que se repite periódicamente.
- **Ruido**: fluctuaciones aleatorias en los datos.

A la hora de trabajar en la inferencia de series temporales, primero se debe establecer un modelo probabilístico para representar los datos. A partir del modelo, se deben estimar sus parámetros y evaluar la correctitud de las predicciones [20].

4.1.1. Series temporales en la predicción de potencia eléctrica

El problema de la predicción de potencia eléctrica se encuentra intrínsecamente relacionado con la temática de las series temporales. En primer lugar, lo que se busca aproximar es la evolución de la potencia a lo largo del día, es decir, una serie temporal. A su vez, las variables utilizadas para su predicción también conforman series temporales.

4.1.1.1. Variables predictivas

La predicción de potencia eléctrica se realiza en base a un conjunto de datos climáticos: la irradiancia solar, la temperatura del aire, la velocidad del viento y la humedad. A su vez, otra variable de interés es la declinación solar, que provee información temporal. Al igual que la potencia, la evolución de cada medición a lo largo del tiempo representa una serie temporal. A continuación, se detalla cada una de las variables disponibles:

- **Irradiancia solar** $[\frac{W}{m^2}]$**:** es la cantidad de potencia por unidad de área que recibe una superficie desde el Sol en forma de radiación electromagnética.
- **Temperatura del aire** $[°C]$**:** como su nombre indica, es la medición que especifica qué tan frío o caluroso se encuentra el aire circundante en un ambiente.
- **Velocidad del viento** $[\frac{m}{s}]$**:** es la magnitud obtenida con el movimiento del aire de alta a baja presión. Usualmente se debe a los cambios de temperatura del ambiente.
- **Humedad** $[\%]$**:** es la cuantificación de la cantidad de vapor de agua presente en el aire. Generalmente se encuentra expresada en términos de "humedad relativa", la cual indica un porcentaje de la cantidad presente sobre el máximo que puede llegar a contener el aire a la misma temperatura.

- **Declinación** [°]**:** describe el ángulo en grados que forma el Sol con la Tierra, por lo que varía continuamente a lo largo de un año. Se considera de relevancia ya que permite identificar un mismo instante temporal en diferentes años.

4.1.1.2. Planta fotovoltaica y sensores climáticos

La planta instalada en la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata (ver figura 4.1) incluye 18 módulos fotovoltaicos, donde cada uno de ellos genera un máximo de 290 W. Se encuentran instalados 3 *strings* de 6 módulos cada uno, que colectivamente generan una cantidad de 5,22 kWp y ocupan una superficie de 35 m2 aproximadamente. Los módulos se encuentran apuntando hacia el norte dentro del perímetro de la facultad y poseen una inclinación de 30°. La planta cuenta con un PR de 0,65, calculado como media anual [5].

Figura 4.1: Planta fotovoltaica instalada en la terraza del edificio principal de la Facultad de Ingeniería de la Universidad Nacional de Mar del Plata.

El ICyTE cuenta con una extensa base de datos en donde se almacenan los valores de potencia generada y las diferentes variables climáticas en cada instante temporal. Los detalles de diseño de la base de datos son explicados en la sección 5.1.1.

Para medir cada una de las variables, la planta fotovoltaica cuenta con diversos sensores instalados. Para medir la irradiancia, se utilizan 2 sensores. El primero forma parte de una estación climática Davis Vantage Pro II, ubicada a 5 m de los paneles. El segundo se encuentra en la Facultad, sobre la superficie inclinada en donde se encuentran los módulos. De esta forma, se registran datos tanto de la irradiancia inclinada como de la horizontal.

La estación Davis también cuenta con sensores adicionales para tomar mediciones sobre la temperatura del aire, la velocidad del viento y la humedad del ambiente. A su vez, se utiliza el software Weatherlink [22], propio de la estación, para la obtención de dichos valores recopilados cada 1 minuto.

Por último, la obtención de los datos de potencia de los tres *strings* es realizada por un dispositivo PQube [23]. Se encarga de promediar los datos obtenidos cada 10 segundos y los almacena cada 5. Todos los sensores están sincronizados vía NTP (*Network Time Protocol*) con la zona horaria argentina (ART).

4.2. Modelos conexionistas y *Deep Learning*

El conexionismo es un paradigma que hace referencia a los procesos cerebrales humanos. Se encarga de explicar la estructura de una red neuronal y el procesamiento de información a través de ella. El concepto de conexionismo aplicado a neuronas artificiales fue introducido por McCulloch y Pitts en su trabajo de 1943 titulado *"A Logical Calculus of the Ideas Immanent in Nervous Activity"* [24]. Este trabajo sentó las bases para el desarrollo posterior de redes de neuronas artificiales. De esta forma, los modelos conexionistas computacionales comenzaron a denominarse redes neuronales artificiales (RNA), debido a su similitud con el funcionamiento del cerebro humano [25].

Con el paso del tiempo y el avance de la tecnología, la humanidad se vio en la necesidad de administrar en su día a día una mayor cantidad de información para resolver problemas cada vez más complejos. A su vez, las redes neuronales artificiales también debieron ser mejoradas para poder hacer uso de una gran cantidad de datos. Para ello, se diseñaron modelos de aprendizaje profundo o de *Deep Learning*, los cuales se caracterizan por ser redes con una estructura de alta complejidad y con un gran volumen de neuronas en su interior [26].

Dentro de los modelos de aprendizaje profundo, se destacan las RNN, que son capaces de retener información del pasado para utilizarla en predicciones futuras. Un ejemplo clásico de RNN es la red NARX (ver sección 5.3).

A su vez, en las últimas décadas han surgido modelos recurrentes de capacidades aun mayores, como el LSTM y la GRU (ver secciones 4.2.3 y 4.2.3.3). Al ser redes con una alta complejidad arquitectónica, requieren de mucho más tiempo de entrenamiento y de la utilización de unidades de procesamiento dedicadas, como las GPU o TPU.

4.2.1. Redes neuronales artificiales

Las redes neuronales artificiales son un modelo diseñado para imitar la forma en la que el cerebro humano realiza una tarea en particular [27]. Al igual que el cerebro, las RNA se constituyen a partir de la interconexión de unidades básicas: las neuronas, que representan unidades de procesamiento.

Cada conexión de neuronas posee un nivel de importancia o fuerza, que es determinado por un valor conocido como peso sináptico. Cuanto mayor sea el peso de una conexión, más influencia tendrá en el resultado de la computación que se desee realizar [27].

Al nacer, el cerebro humano cuenta con una estructura establecida. Además, posee la capacidad de construir sus propias reglas de inferencia a través de la experiencia. Al igual que el cerebro, las RNA adquieren conocimiento a través de un proceso de aprendizaje. El conocimiento adquirido se representa a través de los pesos sinápticos de sus conexiones [27].

El procedimiento utilizado para efectuar el proceso de aprendizaje es conocido como el algoritmo de aprendizaje, una función que es capaz de modificar los pesos de una RNA. El algoritmo busca que la red alcance su objetivo de diseño: resolver un problema determinado modelando la forma en que opera el cerebro humano [27].

4.2.1.1. El cerebro humano

Como puede observarse en la figura 4.2, el sistema nervioso humano está compuesto por tres partes principales: los receptores, el cerebro y los efectores. El cerebro es la parte central del sistema y puede representarse como una red neuronal. La red recibe información constantemente y toma decisiones a partir de un proceso de inferencia [28].

El sistema nervioso cuenta con conexiones hacia adelante y hacia atrás. Las primeras representan señales que portan información a través del sistema. Por otro lado, las segundas indican conexiones de retroalimentación [28].

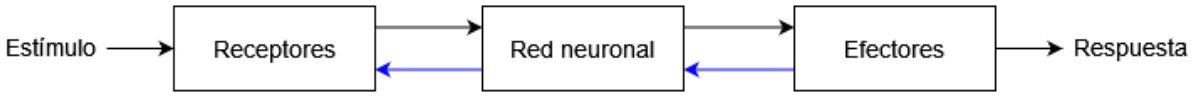


Figura 4.2: Modelo esquemático del sistema nervioso (adaptación de Haykin [27]).

La primera parte está formada por receptores. Son los encargados de captar estímulos, ya sea del ambiente o internos al cuerpo. Los estímulos se transforman en impulsos eléctricos que trasladan la información a la red neuronal. Los efectores, por su parte, forman la parte final del sistema. Su tarea es convertir las salidas de la red neuronal (impulsos eléctricos) en una respuesta determinada (por ejemplo, mover un músculo) [28].

A pesar de las limitaciones que puede tener, el cerebro es una estructura altamente eficiente. La velocidad en la que actúan las neuronas ronda la escala de los milisegundos, mientras que las compuertas lógicas de un procesador operan en el orden de los nanosegundos. La diferencia está en la cantidad de interconexiones. La corteza cerebral contiene aproximadamente 10 mil millones de neuronas, las cuales forman 60 billones de conexiones [29].

Otra parte importante dentro del cerebro es la sinapsis. Se trata de la estructura elemental que media las interacciones entre neuronas. Las más comunes son de tipo químico. Puede pensarse como una conexión que puede mostrar excitación o inhibición, pero no ambas en simultáneo [29]. La mayoría de las neuronas transmiten sus salidas como una serie de pulsos voltaicos breves, conocidos como potenciales de activación, que se propagan a través de la red [27].

A su vez, el sistema nervioso posee la capacidad de adaptarse a su entorno a través de la creación de nuevas conexiones sinápticas entre neuronas y la modificación de las sinapsis existentes. A esta característica se la conoce como plasticidad [27].

4.2.1.2. Perceptrón

Luego de la introducción de la neurona artificial en el año 1943 por McCulloch y Pitts [24], Frank Rosenblatt amplió el modelo en el año 1958 en su libro *"The Perceptron: A Probabilistic Model For Information Storage and Organization in the Brain"* [30]. En esta publicación se puede encontrar el primer modelo conexionista computacional diseñado por el ser humano: el perceptrón simple o perceptrón de Rosenblatt, la unidad básica de las redes neuronales [30]. En la figura 4.3, se lo puede visualizar esquemáticamente.

Cada red neuronal posee una o múltiples capas, que están conformadas por un número determinado de neuronas. Dentro de una capa, las neuronas trabajan en conjunto para procesar la información recibida y generar una respuesta, que luego es transmitida a la siguiente capa de la red [27].

Matemáticamente, el perceptrón se modela en base al funcionamiento de su equivalente en el sistema nervioso. Se compone de cuatro elementos: [27]:

1. **Sinapsis** o **conexiones**: están caracterizadas por su peso sináptico, que representa el grado de importancia de la conexión. El peso puede ser un número positivo o negativo. Una señal x_j en la entrada de la sinapsis j conectada a la neurona k se multiplica por el peso sináptico w_{kj}.
2. **Sumador**: encargado de adicionar las señales de entrada que fueron previamente ponderadas por su respectivo peso sináptico.
3. **Función de activación**: limita la amplitud de las salidas de la neurona.
4. **Sesgo** o ***bias***: aumenta o disminuye la entrada neta de la función de activación. Permite desplazar la función de activación. Es representado por el término b_k y es equivalente a la ordenada de origen en una función lineal [31].

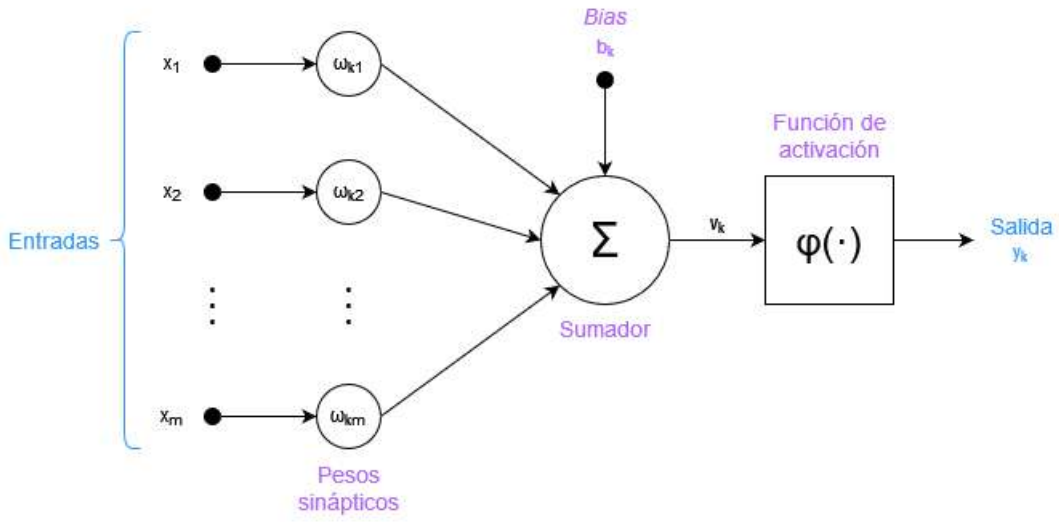


Figura 4.3: Modelo esquemático del perceptrón de Rosenblatt (adaptación de Haykin [27]).

A su vez, los elementos de una neurona artificial pueden representarse por medio las ecuaciones:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{4.1}$$

y

$$y_k = \varphi(u_k + b_k) \tag{4.2}$$

donde:

- $x_1, x_2, ..., x_m$ representan las entradas.
- $w_{k1}, w_{k2}, ..., w_{km}$ son los pesos sinápticos de la neurona k.

- u_k es la salida del combinador lineal.
- b_k representa la unidad de *bias*.
- y_k indica el valor de salida de la neurona k.
- φ es la función de activación.

4.2.1.3. Funciones de activación

El agregado de una función de activación le otorga a la RNA la capacidad de analizar datos de forma no lineal [32]. En caso de no contar con ellas, las neuronas únicamente producirían transformaciones lineales de sus entradas. A continuación, se mencionan algunas de las más relevantes:

- **Función lineal:** produce una salida igual al resultado de la combinación lineal entre los pesos, las entradas y el *bias*. Está determinada por la ecuación $f(x) = x$. Generalmente solo se la utiliza en una sola capa, como puede ser la de salida [32]. No suele utilizarse en capas ocultas, dado que solo produce transformaciones lineales.
- **Sigmoide:** dada por la ecuación $f(x) = \frac{1}{1+e^{-x}}$. Produce valores entre 0 y 1. Es una función no lineal que permite mantener la salida dentro de un rango fijo. Puede visualizarse gráficamente en la figura 4.4.
- **Tangente hiperbólica:** es similar a la función sigmoide. Sus valores de salida están dentro del rango -1 a 1. Ayuda a centrar la media cerca del 0 y también aporta no-linealidad [33]. Dada por la ecuación $f(x) = tanh(x)$. A modo de ejemplo, se puede observar la figura 4.5.
- **ReLU:** su nombre se deriva de unidad lineal rectificada (*rectified linear unit*). Es una función por tramos determinada por la ecuación $f(x) = max(0, x)$ y puede visualizarse en la figura 4.6. Es más eficiente computacionalmente que la sigmoide y la tangente hiperbólica [33]. Sin embargo, su gradiente es 0 en las regiones negativas, lo que provoca que los pesos no se actualicen durante el descenso y su salida sea siempre 0. En caso de que la condición se presente en la mayoría de neuronas, la red se volvería incapaz de aprender y retornaría siempre un valor constante. Este problema es conocido como *"Dying ReLU"* [34].

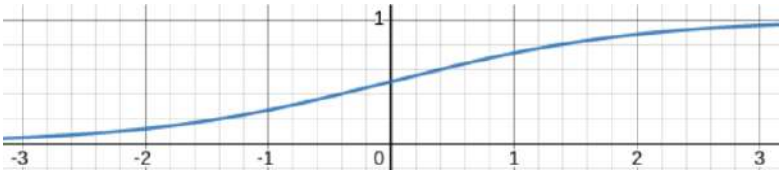


Figura 4.4: Gráfico de la función sigmoide en el intervalo $[-3, 3]$.

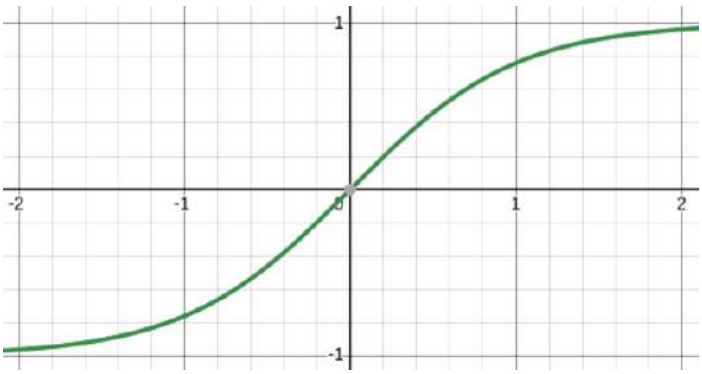


Figura 4.5: Gráfico de la tangente hiperbólica en el intervalo $[-2, 2]$.

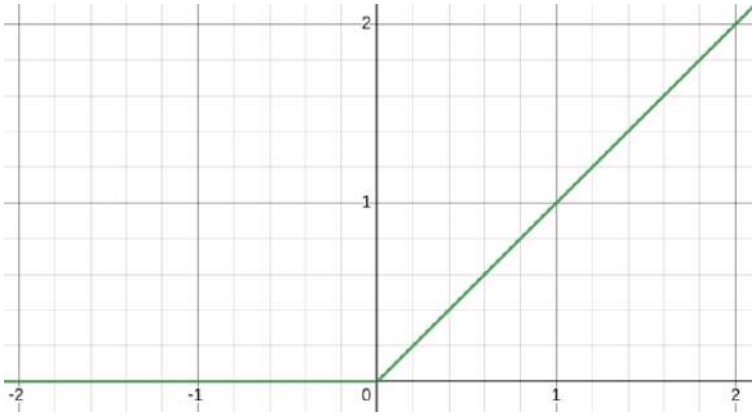


Figura 4.6: Gráfico de la función ReLU en el intervalo $[-2, 2]$.

4.2.1.4. Arquitecturas

La arquitectura se define como la forma en la que las neuronas de una RNA están estructuradas [27]. Dentro de las tratadas en el proyecto, se encuentran las redes *feedforward* de capa única o multicapa, así como las redes recurrentes.

Redes *feedforward* de capa única

Cuentan con una única capa de neuronas. En primer lugar, está la capa de entrada, encargada de recibir los valores introducidos al modelo. Posteriormente, se los transfiere a la capa de cómputo, donde sus neuronas se encargan de calcular el resultado de la red. Al ser la última capa, también es conocida como capa de salida. En la figura 4.7, puede observarse un modelo esquemático.

Se las conoce como redes *feedforward* (o prealimentadas), dado que sus nodos no forman ciclos. La información únicamente se mueve hacia adelante y no hay conexiones que puedan enviar información hacia atrás [35]. También se las denomina perceptrón de capa única (o *single-layer perceptron*).

Al contar con una única capa, este tipo de redes funcionan como combinadores lineales en problemas de regresión. Por este motivo, su utilidad práctica es limitada.

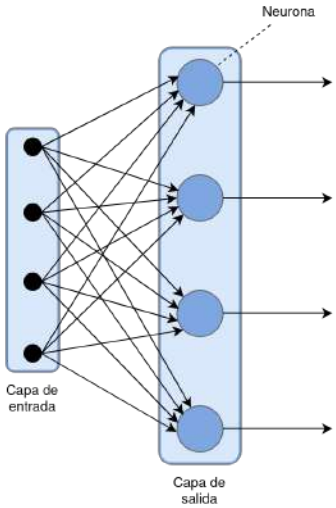


Figura 4.7: Red neuronal *feedforward* de capa única. Cada neurona es equivalente a la representada en la figura 4.3.

Redes *feedforward* multicapa

Incluyen una o más capas de neuronas adicionales entre medio de las capas de entrada y salida, denominadas como capas ocultas. El objetivo de añadir más dimensiones a las interacciones entre neuronas es

que la red logre extraer patrones de mayor complejidad presentes en los datos [27]. También se la conoce como perceptrón multicapa.

Una representación de la arquitectura puede observase en la figura 4.8. El MLP observado se encuentra completamente conectado (o *fully conected*). Cada neurona en cada capa de la red está conectada a todos los nodos de la capa posterior [27].

Al igual que en el modelo *single-layer*, la información introducida a la red fluye hacia la salida. Cada capa produce una respuesta a partir de los datos que recibe, que luego es transmitida hacia adelante. Finalmente, la última capa es la encargada de computar la respuesta final.

A las señales que se transmiten hacia adelante se las denominada "señales de función" [27]. A su vez, existen otras que inician en la capa de salida y se propagan hasta llegar a la entrada. Se las denomina "señales de error", ya que se producen a partir de la diferencia entre las predicciones de la red y los resultados reales. En la figura 4.9, puede observarse una ilustración de ambos tipos de señales.

El flujo de errores es utilizado para actualizar los pesos sinápticos y el *bias* de la red, de manera que el error se reduzca. A este procedimiento se lo conoce como *backpropagation* y es una componente fundamental del entrenamiento de redes neuronales [27].

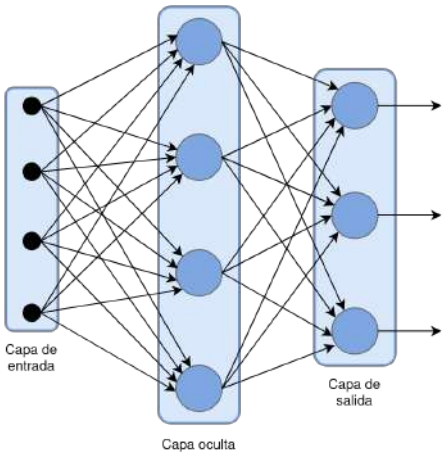


Figura 4.8: MLP con una capa oculta.

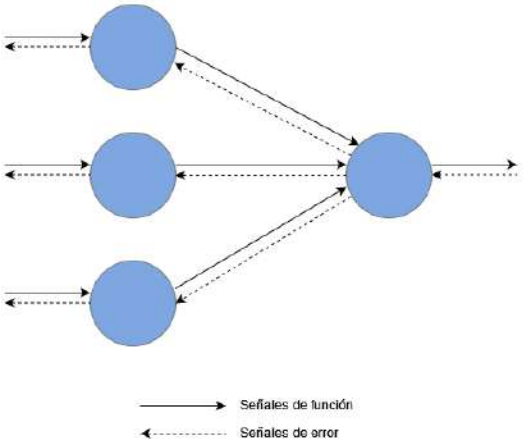


Figura 4.9: Propagación de señales para una red neuronal *feedforward* (adaptación de Haykin [27]).

Redes recurrentes

Se caracterizan por la presencia de conexiones de retroalimentación, una propiedad que no poseen las redes *feedforward*. Las neuronas no solo alimentan a la capa posterior, sino que su salida también es utilizada como retroalimentación de otras unidades o de sí misma. Su capacidad de persistencia es particularmente importante en problemas de series temporales, donde cada instante está condicionado por aquellos que han ocurrido antes.

Una visualización de una red neuronal recurrente básica se presenta en la figura 4.10. A su vez, en la figura 4.11 se puede observar un desdoblamiento en el tiempo de una neurona recurrente.

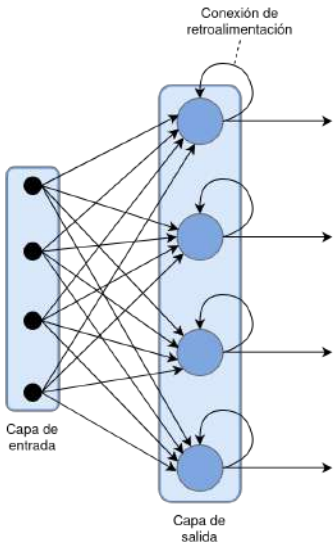


Figura 4.10: Red neuronal recurrente.

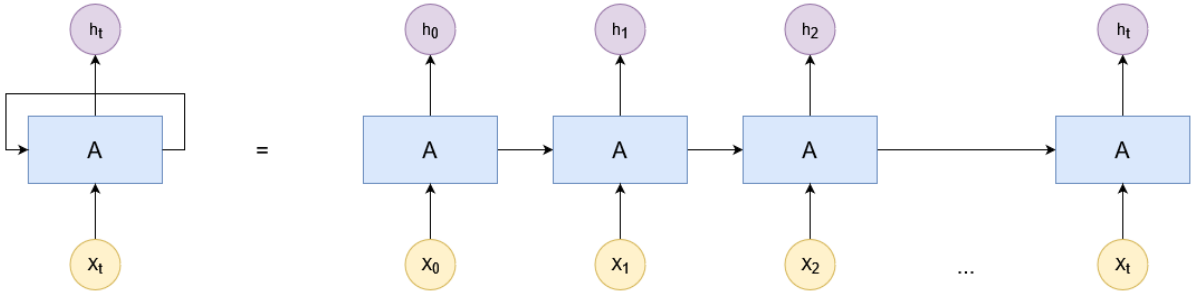


Figura 4.11: Neurona de una RNN desdoblada en el tiempo. En cada instante t, la unidad calcula la salida h_t en base a la entrada x_t y el estado anterior h_{t-1}. Posteriormente, el valor es retroalimentado a la siguiente iteración (adaptación de Olah [36]).

4.2.1.5. Proceso de aprendizaje

Existen múltiples enfoques a la hora de llevar a cabo el proceso de aprendizaje de una red neuronal. Los más importantes son el aprendizaje supervisado y el no supervisado, aunque existen otras alternativas como el aprendizaje por refuerzo y el semi-supervisado. La diferencia principal entre un proceso supervisado y otro no supervisado es que el primero utiliza datos rotulados para ayudar a predecir la salida, mientras que el segundo no lo hace.

Al ser un problema de regresión, la predicción de la potencia eléctrica se enmarca dentro del aprendizaje supervisado. Los problemas de regresión se basan en la aproximación de una variable dependiente en base a un conjunto de variables independientes. Para el presente proyecto, las variables independientes son los valores correspondientes a las variables climáticas, mientras que la variable dependiente es la propia potencia eléctrica.

El objetivo del aprendizaje supervisado es generar un modelo capaz de realizar predicciones correctas ante nuevas entradas. Para llegar a este fin, las RNA pasan a través de un proceso de aprendizaje, donde se trabaja con un conjunto de ejemplos para los cuales se conoce la salida esperada. Al contar con los valores asociados al conjunto de entradas, se dice que los datos se encuentran rotulados [37].

El uso de datos rotulados le permite al modelo calcular una métrica de error, que se define como la diferencia entre la respuesta real y la que provee la red. El cálculo del error es una función de los pesos del sistema y puede visualizarse como una superficie de error multidimensional.

A partir del algoritmo de *backpropagation*, los pesos sinápticos de la red se ajustan en base a la métrica de error calculada. El proceso se realiza iterativamente, con el objetivo de reducir al mínimo posible la diferencia entre los resultados [27]. A modo de ejemplo, puede observarse la figura 4.12.

La actualización de los pesos se efectúa a través del cálculo del gradiente de la superficie en el punto de la iteración actual. El gradiente es un vector que apunta hacia la dirección del ascenso más pronunciado, por lo que el ajuste se hace en la dirección inversa [27]. Una vez alcanzado el mínimo en la superficie de error, se considera que el proceso de aprendizaje ha finalizado. Finalmente, la red se encuentra lista para realizar predicciones en base a nuevos datos de entrada.

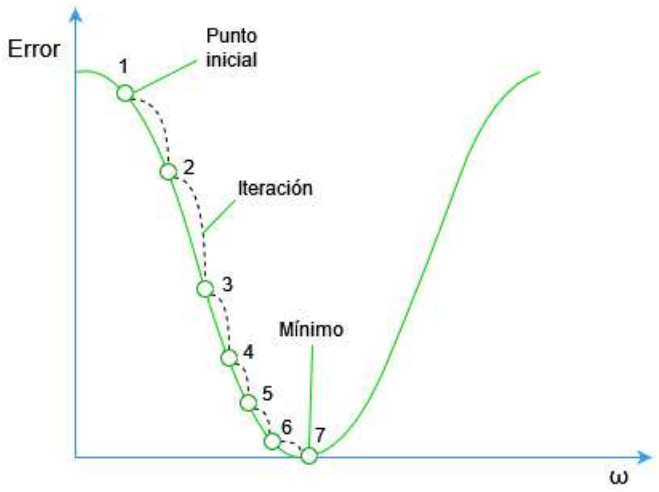


Figura 4.12: Descenso por gradiente con una única variable independiente ω (adaptación de Sancho Caparrini [38]).

4.2.2. NARX

Los modelos conexionistas clásicos han sido de gran utilidad como base del funcionamiento de otro tipo de redes. Sin embargo, existen arquitecturas que proveen una *performance* mayor, como lo son las RNN. Uno de los tipos tratados en el proyecto es NARX, un modelo sencillo orientado a la predicción de series temporales [27].

4.2.2.1. Arquitectura

Una red NARX es un tipo de RNN que contiene unidades de memoria tanto para valores de entrada como de salida, las cuales actúan como retroalimentadoras de la red [27]. Se encargan de realizar una predicción de una serie temporal dependiente a partir de otra independiente. El modelo fue propuesto en el año 1990 por K. S. Narendra y K. Parthasarathy en el trabajo titulado *"Identification and control of dynamical systems using neural networks"* [39]. A partir de entonces, se volvió la base para posteriores modelos de RNN.

Las redes NARX son reconocidas por su desempeño para la predicción de series temporales. Su principal característica es la capacidad para persistir valores anteriores de entrada y de salida, que luego son utilizados para predecir el siguiente instante temporal de la serie [40]. A dichos valores persistidos se los denomina ventanas temporales.

Como se puede observar en la figura 4.13, la representación de una red NARX es similar a la de una red neuronal *feedforward*. La única diferencia es que se añade una entrada exógena y otra endógena [27]:

- **Entrada exógena:** es el valor asociado a una serie temporal independiente. Para generar una predicción en un momento temporal t, se puede ampliar la ventana temporal a n instantes temporales hacia atrás en el tiempo e ingresar las variables de entrada en $t, t-1, ..., t-n$.
- **Entrada endógena:** es el valor asociado a una serie temporal dependiente. Es la predicción obtenida en un instante de tiempo t y que será usada en la próxima predicción como un valor de entrada y_{t-1}. Para este caso, también se puede ampliar la ventana temporal a m instantes temporales anteriores e ingresar las variables de entrada en $t, t-1, ..., t-m$.

Una red NARX puede ser abierta o cerrada, lo que depende de si su retroalimentación es externa o interna, respectivamente. Una cerrada se retroalimenta con el mismo valor predicho por el modelo, mientras que una abierta lo hace a partir del valor real [41]. Por otra parte, una característica adicional a destacar sobre este tipo de redes, es que se pueden entrenar por *backpropagation*, como lo hace también un MLP.

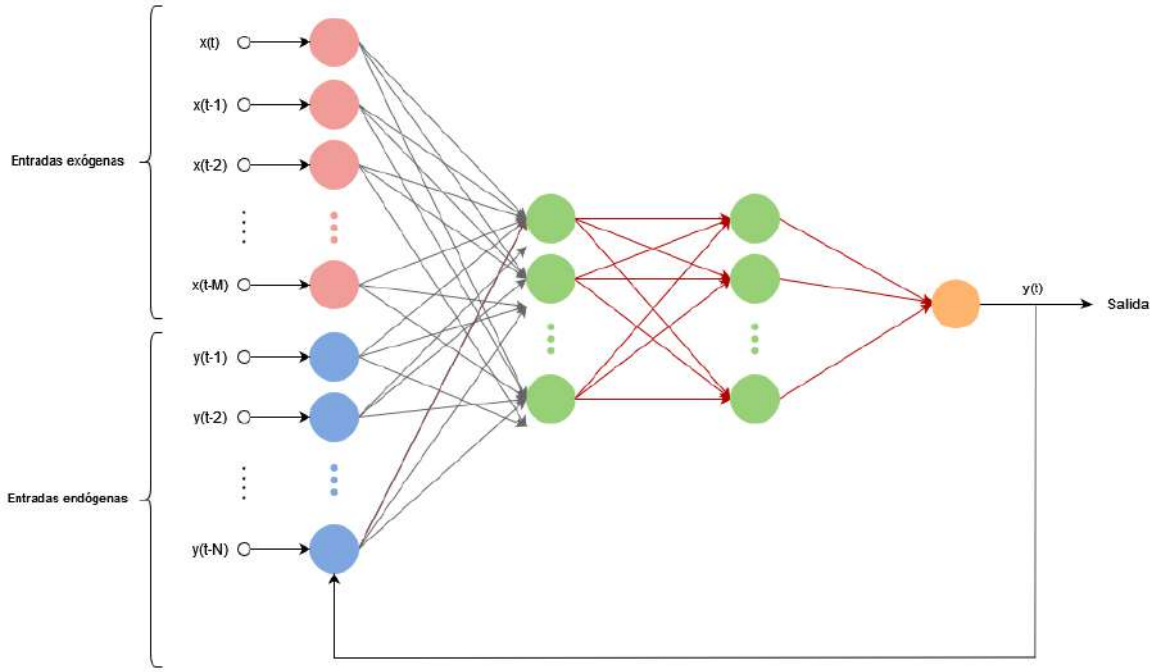


Figura 4.13: Gráfico de la estructura de una red NARX.

4.2.3. LSTM y GRU

4.2.3.1. Origen

Si bien las RNN almacenan información del pasado, los modelos más simples son incapaces de retener las dependencias a largo plazo de los datos. Por este motivo, las variantes clásicas no suelen ser suficiente para trabajar sobre problemas con alta complejidad. La razón de este comportamiento es el problema del desvanecimiento del gradiente.

El cálculo de gradientes en una red neuronal se realiza a través de la regla de la cadena. Debido a que la propagación hacia atrás en las RNN es efectuada a través de las capas y el tiempo, la señal de error disminuirá exponencialmente al aumentar la cantidad de iteraciones en el entrenamiento, lo que provocará que las actualizaciones sobre los pesos sinápticos sean insignificantes [42]. Por otro lado, también existe la posibilidad de que ocurra el caso inverso: las multiplicaciones de gradientes pueden crecer exponencialmente y tender a infinito. A esta situación se la denomina como el problema de la explosión del gradiente [43].

Para solucionar los problemas mencionados, Hocreiter y Schmidhuber idearon la arquitectura *Long Short-Term Memory* en 1997 [44]. El modelo posee una estructura interna que permite que el estado de cada celda se actualice a través de una función aditiva. A su vez, sus compuertas son capaces de nivelar el valor del gradiente, de manera que se evite el desvanecimiento.

4.2.3.2. Estructura LSTM

Las capas de un modelo LSTM poseen una estructura en forma de cadena, donde las celdas se enlazan iteración a iteración. Cada una de ellas está conformada por los siguientes componentes:

- **Estado**: representa el flujo de información a través de la celda. La adición o remoción de información se regula a partir de compuertas.

- **Compuerta *forget***: decide qué información se removerá del estado de la celda. Recibe como parámetros las entradas a la capa y el estado oculto de la iteración anterior. A través de una función sigmoide, produce un vector con valores entre 0 y 1, que se multiplica componente a componente con el estado de la celda anterior.
- **Compuerta *input***: regula qué nueva información se almacenará en el estado de la celda. En primer lugar, decide qué componentes del estado actualizar, lo cual se realiza a partir de una función sigmoide. Posteriormente, se utiliza una función *tanh* para calcular el vector de candidatos. Ambos vectores se multiplican componente a componente y el resultado se adiciona al estado de la celda.
- **Estado oculto o salida**: representa una versión filtrada del estado de la celda. En primer lugar, el estado de la celda pasa a través de una función *tanh*. Luego, se lo multiplica componente a componente por el resultado de una función sigmoide, que define qué partes del vector compondrán la salida. Su valor no solo pasará a la siguiente capa, sino que también se utilizará en la siguiente iteración temporal.

Los componentes de la celda LSTM pueden visualizarse en la figura 4.14. Matemáticamente, se la representa mediante las ecuaciones:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{4.3}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{4.4}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{4.5}$$

$$\tilde{C}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{4.6}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4.7}$$

$$h_t = o_t \odot tanh(C_t) \tag{4.8}$$

en donde:

- t representa un instante temporal.
- h es la cantidad de unidades de la celda.
- d es la cantidad de entradas.
- $x_t \in R^d$ es el vector de entradas.
- $f_t \in (0,1)^h$ es el vector de activación de la compuerta *forget*.
- $i_t \in (0,1)^h$ es el vector de activación de la compuerta *input*.
- $o_t \in (0,1)^h$ es el vector de activación de la salida.
- $h_t \in (-1,1)^h$ es el vector del estado oculto.
- $\tilde{C}_t \in (-1,1)^h$ es el vector de activación de la celda.
- $C_t \in R^h$ es el vector de activación de la entrada de la celda.
- $W \in R^{h \times d}$ y $U \in R^{h \times h}$ son matrices de pesos.
- $b \in R^h$ es el vector de *bias*.
- σ es la función sigmoide.
- $\odot$ representa el producto elemento a elemento (también conocido como producto de Hadamard [45]).

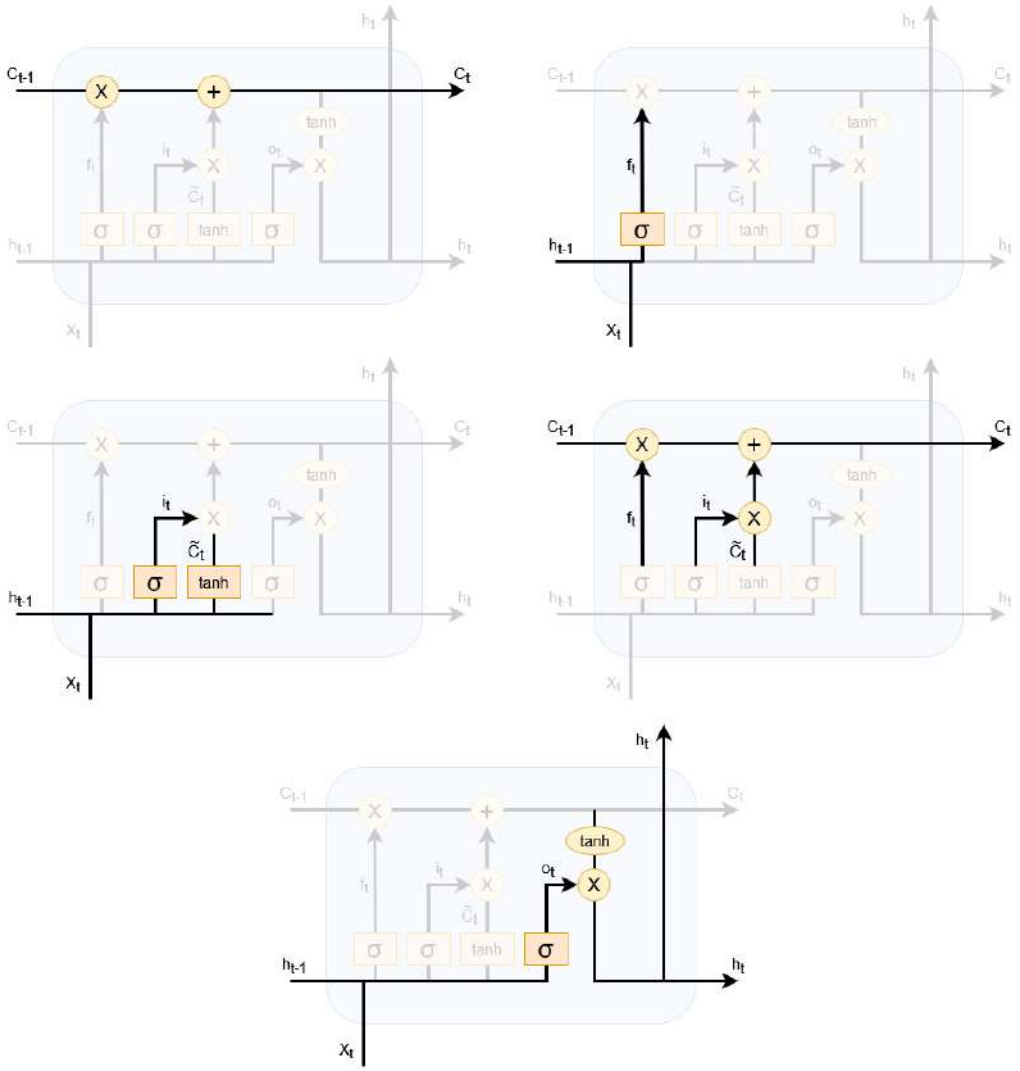


Figura 4.14: Componentes de una celda LSTM. De izquierda a derecha y de arriba hacia abajo: *(I)* recorrido del estado oculto dentro de la celda, *(II)* compuerta *forget*, *(III)* compuerta *input*, *(IV)* actualizaciones al estado de la celda y *(V)* cálculo del estado oculto (adaptación de Olah [36]).

4.2.3.3. GRU

La estructura de una celda LSTM puede modificarse de múltiples maneras, por lo que existen diversas variantes sobre la estructura original propuesta por Hocreiter y Schmidhuber. Una de las variantes más populares es GRU, que fue propuesta por Cho *et al.* en 2014 [46].

GRU posee una estructura más simple que el modelo LSTM original, lo que reduce la cantidad de operaciones matriciales a realizar. A pesar de poseer una menor complejidad, la *performance* de GRU no decrece considerablemente en comparación a LSTM. Por ende, ofrece una buena relación entre *performance* y costo computacional [47].

Una de las diferencias principales es la combinación del estado oculto y el estado de la celda. Se caracteriza por las siguientes compuertas.

- **Compuerta *update***: actúa como una combinación de las compuertas *forget* e *input* del modelo LSTM, ya que determina qué información del pasado debe ser trasladada al futuro. Para sus cálculos, utiliza el estado del instante anterior y la entrada a la celda. A través de una función sigmoide, produce un vector que establece las componentes del estado que serán actualizadas. Las actualizaciones y remociones sobre el estado funcionan de manera complementaria. La diferencia entre un vector de unos y el resultado de la compuerta es utilizado para determinar qué componentes del estado se deben olvidar.
- **Compuerta *reset***: indica la relevancia del estado anterior a la hora de efectuar el cálculo del vector de candidatos.

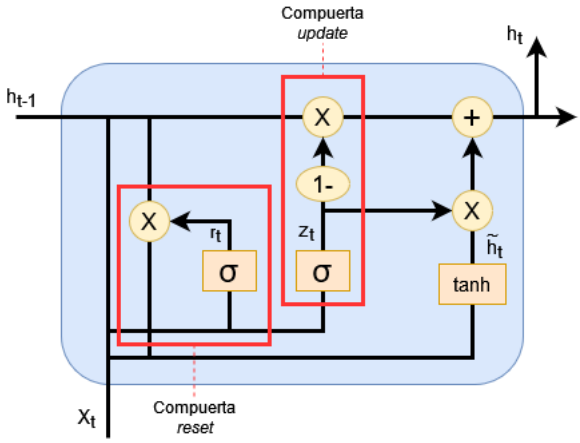


Figura 4.15: Estructura de una celda GRU (adaptación de Olah [36]).

El modelo de la celda GRU puede observarse en la figura 4.15. Se la representa a través de las ecuaciones:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{4.9}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{4.10}$$

$$\tilde{h}_t = tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \tag{4.11}$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{4.12}$$

en donde:

- $x_t \in R^d$ es el vector de entradas.
- $z_t \in R^h$ es el vector de la compuerta *update*.
- $r_t \in R^h$ es el vector de la compuerta *reset*.

- $h_t \in R^h$ es el vector de salida.
- $\tilde{h}_t$ es el vector de candidatos.
- $W \in R^{h \times d}$ y $U \in R^{h \times h}$ son matrices de pesos.
- $b \in R^h$ es el vector de *bias*.

4.2.3.4. Capas bidireccionales

Una capa bidireccional es generada a partir de la combinación en paralelo de dos celdas LSTM o GRU independientes. Los modelos que las utilizan son denominados Bi-LSTM o Bi-GRU, lo cual depende de la implementación escogida.

En un modelo bidireccional, una de las celdas recibe la secuencia de entrada ordenada cronológicamente, mientras que su contraparte la recibe de forma inversa. Una vez que cada celda realiza sus respectivas operaciones, las salidas se combinan y pasan a la siguiente capa de forma concatenada. Otra opción es promediarlas o adicionarlas.

Las capas bidireccionales se utilizan principalmente en problemas como el procesamiento del lenguaje natural. Sin embargo, han logrado obtener resultados competitivos al utilizarse en la predicción de la potencia eléctrica generada por una planta fotovoltaica [48].

4.2.3.5. Mecanismos de atención

Los modelos LSTM y GRU comprimen la información de los instantes temporales previos en un vector. La compresión puede llevar a que cierta parte de la información se pierda, dado que toda la secuencia se debe reducir a un vector de tamaño fijo. Este problema dio origen a los mecanismos de atención, que fueron introducidos por Bahdanau *et al.* en su trabajo *"Neural Machine Translation by Jointly Learning to Align and Translate"* [49].

La atención surge como otra forma de emular el comportamiento del cerebro humano. Se utiliza para que los modelos de *Deep Learning* prioricen la información más relevante de un conjunto de datos. La forma más común de implementar el mecanismo es a través de la inclusión de una nueva capa en la red neuronal, cuya tarea es aprender qué porciones de los datos son las más relevantes en cada instante temporal en particular.

El mecanismo utiliza pesos para asignarle una determinada importancia a cada estado oculto, de manera similar a los pesos sinápticos de una red neuronal. Los pesos de atención se calculan en base a una función de puntaje. Pueden utilizarse una amplia variedad de funciones para su cálculo, entre ellas el estilo multiplicativo de Luong, introducido en su trabajo *"Effective Approaches to Attention-based Neural Machine Translation"* [50].

Durante la ejecución, la capa de atención recibe los estados ocultos producidos por la capa recurrente. Posteriormente, computa el vector de contexto, que contiene la información más relevante de la secuencia. El vector es la salida de la capa de atención y se lo calcula como la suma ponderada de los pesos y los estados ocultos.

4.3. Otros modelos

Además del conexionismo, existen modelos de *Machine Learning* basados en otros paradigmas. Un ejemplo son aquellos asociados a la teoría del aprendizaje estadístico, cuyos referentes clásicos son los modelos SVM (*Support Vector Machine*) y SVR [51].

Se escogió describir brevemente el modelo SVR, dada su importancia el la literatura de la predicción de potencia eléctrica generada por una planta fotovoltaico. Es uno de los modelos clásicos más referenciados y será aplicado con fines comparativos durante la fase de experimentación.

4.3.1. SVR

Un SVM es un modelo predictivo que se encarga de resolver problemas relacionados con la clasificación de patrones. Permite lidiar con casos complejos, donde los patrones no son linealmente separables [27]. Una de sus variantes más conocidas es el SVR, que se dedica exclusivamente a resolver problemas de regresión.

4.3.1.1. Arquitectura

Las componentes más importantes del modelo pueden visualizarse en la figura 4.16. A continuación, se explica cada una de ellas:

- **Hiperplano:** es la línea que mejor se ajusta a los datos y que es utilizada para realizar las predicciones. A los valores más cercanos al hiperplano se los llama "vectores de soporte" y sirven como referencia para la obtención de predicciones [52].
- **Líneas límite:** son dos líneas que se encuentran a una distancia ϵ del hiperplano. Funcionan como una limitante entre los datos cercanos al hiperplano y posibles *outliers* [52].
- ***Kernel*:** es un conjunto de funciones matemáticas que toman un valor de entrada y lo transforman, de manera similar al funcionamiento de una función de activación de un perceptrón multicapa. Se utilizan generalmente para encontrar un hiperplano. Algunos de los más utilizados son el lineal, el polinomial y la *radial basis function* (RBF) [52].

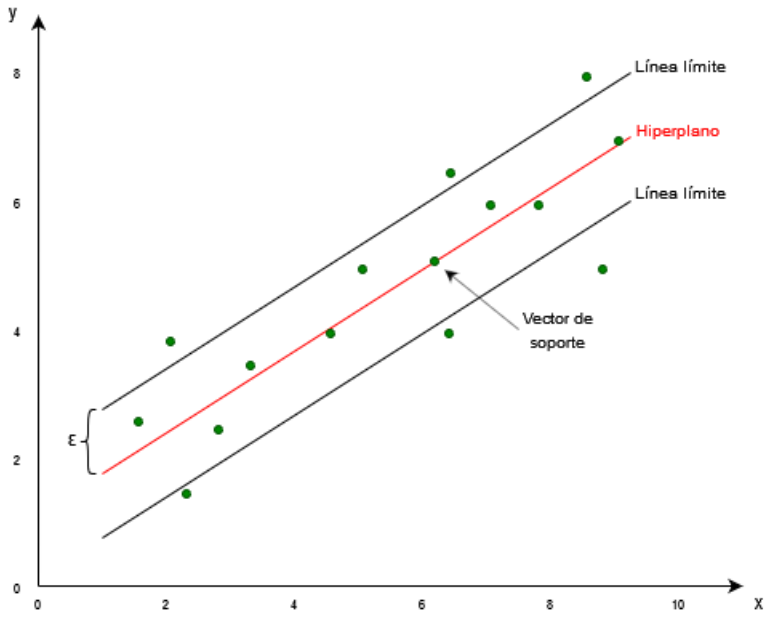


Figura 4.16: Visualización gráfica de los componentes de un modelo SVR bidimensional y con *kernel* lineal.

El *kernel* reemplaza a las celdas o neuronas en el paradigma conexionista. A diferencia de los modelos regresivos, que intentan minimizar el error entre el valor predicho y el real, SVR se basa en encontrar un hiperplano que se ajuste a la mayor cantidad posible de los valores que el modelo observa [53]. RBF es el *kernel* más recomendado y utilizado. Se encarga de determinar qué tanta similitud hay entre dos puntos x_1 y x_2 a partir de la distancia entre ellos. Mientras más cercanos sean los dos puntos, más similares son considerados [54]. Su funcionamiento se demuestra con la siguiente ecuación:

$$K(x_1, x_2) = exp(-\frac{||x_1 - x_2||^2}{2\sigma^2}) \tag{4.13}$$

en donde σ representa la varianza y $||x_1 - x_2||$ es la distancia euclidiana entre los dos puntos. Dentro de la ecuación del kernel, el factor $\frac{1}{2\sigma^2}$ suele expresarse como γ, el cual representa uno de los parámetros de configuración del modelo [55].

Posteriormente, se incorpora un parámetro adicional denominado C, encargado de determinar la penalización a los valores que se excedan de las líneas límite. El nivel de penalización determinará qué tan preciso será el ajuste realizado por el modelo durante el proceso de aprendizaje. Es un valor de alta importancia, dado que de establecerlo incorrectamente el modelo puede tender tanto al *overfitting* como al *underfitting* (ver sección 4.4.2) [54].

4.4. Construcción y entrenamiento de modelos

A la hora de construir un modelo, uno de los objetivos principales es obtener un buen nivel de generalización. La generalización se define como la capacidad del modelo para desempeñarse satisfactoriamente ante entradas que no ha procesado con anterioridad [56]. Dentro de un problema de regresión, por ejemplo, el objetivo es construir un modelo capaz de realizar predicciones acertadas, incluso cuando la red se enfrenta a nuevas entradas. Algunos conceptos asociados a la generalización son la partición de los datos disponibles, el *underfitting* y el *overfitting*.

A su vez, para lograr una generalización adecuada, es importante definir correctamente los hiperparámetros asociados al modelo. Los hiperparámetros son variables externas que se utilizan para configurar el entrenamiento y la estructura de la red. Se dividen en dos tipos: de arquitectura y de entrenamiento.

4.4.1. Partición de datos

El cálculo del nivel de generalización está estrechamente vinculado a una técnica basada en la partición de datos. Al momento de construir un modelo de regresión, se debe contar con entradas rotuladas sobre las cuales la red pueda aprender. Normalmente, el conjunto completo de datos se divide en tres partes: entrenamiento, validación y *testing*.

El conjunto de entrenamiento es utilizado durante la fase de aprendizaje del modelo. A lo largo del proceso, la red es expuesta a un conjunto de entradas cuyas salidas son conocidas. A partir de técnicas como el descenso por gradiente (ver sección 4.4.4.2), los pesos de la red se ajustan en cada iteración para reducir la diferencia entre las salidas producidas y las esperadas. A esta diferencia se la conoce como error de entrenamiento. Generalmente, un conjunto de entrenamiento más grande y representativo facilitará el aprendizaje de la red, ya que se la expondrá a una variedad más amplia de valores [57].

El conjunto de validación también es utilizado durante el aprendizaje. Sin embargo, el modelo no actualiza los pesos en base a su error. Al final de una iteración por todo el conjunto de datos de entrenamiento (lo que se denomina como época), se calcula el error de validación, que representa una estimación del error del modelo ante datos que no ha observado anteriormente [58]. Por este motivo, se suele detener el proceso de entrenamiento si el error de validación aumenta de forma desmedida.

Una vez finalizado el proceso de entrenamiento, se utiliza el conjunto de *testing* para calcular la *performance* real del modelo. El uso de un conjunto completamente independiente al período de aprendizaje permite evaluar su error de generalización real. Como se mencionó anteriormente, el error de validación suele tener injerencia en el entrenamiento, por lo que utilizar su valor como medida de *performance* sería incurrir en un sesgo.

4.4.2. *Overfitting* y *underfitting*

Por lo general, existe una brecha entre el error de *testing* y el error de entrenamiento. Naturalmente, la red logrará mejor precisión sobre los ejemplos utilizados durante el ajuste de los pesos sinápticos que sobre entradas que no ha visto anteriormente. Por este motivo, para mejorar la generalización del modelo, se hace énfasis en los siguientes objetivos [56]:

1. Reducir el error de entrenamiento.
2. Reducir la brecha entre el error de *testing* y el de entrenamiento.

Ambos factores se asocian a dos situaciones que buscan evitarse durante el entrenamiento de modelos: el *underfitting* y el *overfitting*. El *underfitting* ocurre cuando la red no es capaz de reducir el error de entrenamiento a un nivel satisfactorio. Por otro lado, el *overfitting* sucede cuando la brecha entre el error de entrenamiento y el de *testing* es muy grande [56].

El *underfitting* y el *overfitting* están estrechamente relacionados con el concepto de capacidad, que es un término informal para referirse a la complejidad de un modelo y al nivel de abstracción de los patrones que puede captar [59].

Al aumentar de forma excesiva la capacidad, los modelos tenderán a memorizar propiedades del conjunto de entrenamiento que no están presentes en el conjunto de *testing*, lo cual llevará a que ocurra *overfitting*. Por otra parte, una capacidad baja tendrá dificultades a la hora de aprender acerca del conjunto de entrenamiento y tampoco ofrecerá un buen nivel de generalización, ocasionando *underfitting* [56]. Durante la construcción del modelo, el objetivo es encontrar una capacidad óptima que minimice la brecha entre el error de entrenamiento y de generalización (ver figura 4.17).

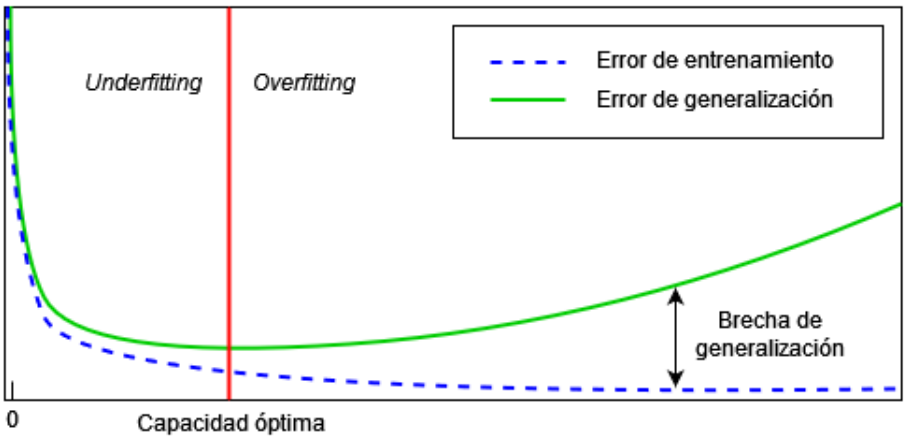


Figura 4.17: Relación entre la capacidad un modelo y los errores de entrenamiento y generalización (adaptación de Goodfellow, *et al.* [56]).

A modo de ejemplo, en la figura 4.18, pueden observarse predicciones generadas por modelos con distintos niveles de capacidad. Los puntos a predecir fueron generados por una función cuadrática con un nivel bajo de ruido. En primer lugar, el modelo que presenta *underfitting* únicamente es capaz de producir una recta para aproximar los datos. En contraposición, el modelo de alta capacidad hace un sobre-ajuste y los aproxima con un polinomio de grado elevado. Finalmente, el modelo con capacidad adecuada es capaz de predecir correctamente el patrón cuadrático, de manera que su nivel de generalización para datos no observados será mayor [56].

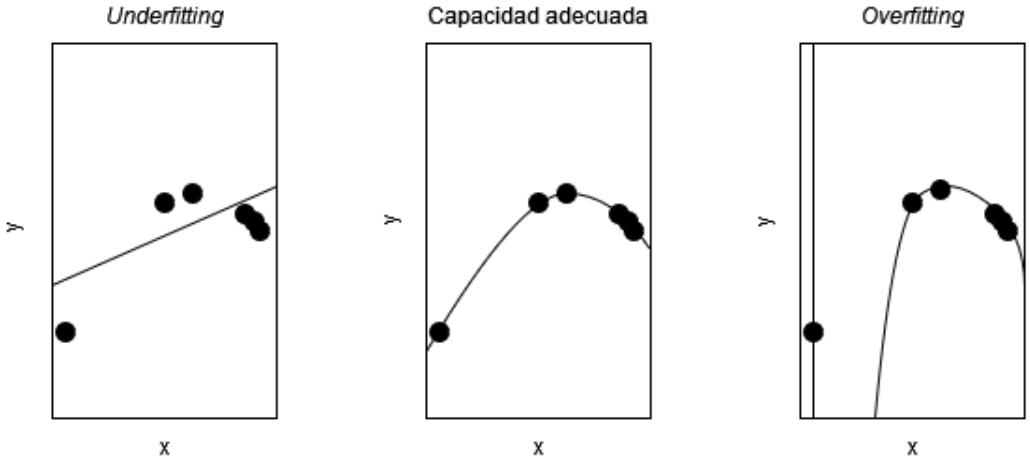


Figura 4.18: Predicciones generadas por redes neuronales con diferentes niveles de capacidad (adaptación de Goodfellow, *et al.* [56]).

Existen diversas formas de alterar la capacidad de un modelo. Una de ellas es aumentar la cantidad de entradas a la red. Para el caso del problema a tratar, esta alternativa no resulta viable, ya que solo se tiene registro temporal de un grupo reducido de variables. Otra opción es la modificación de ciertos hiperparámetros del modelo, principalmente los vinculados a su arquitectura [56].

4.4.3. Hiperparámetros de arquitectura

Los hiperparámetros de arquitectura son aquellos asociados a la estructura de la red. Generalmente, un modelo más grande y complejo lleva a un aumento de la capacidad [56]. Las forma más sencilla de alterar la estructura es modificar la cantidad de unidades por capa, el número de capas ocultas o las funciones de activación de cada neurona. A su vez, existe una técnica especialmente vinculada a reducir las posibilidades de sobre-ajuste: el *dropout*.

En una red *fully connected*, los nodos desarrollan dependencias entre sí, lo cual puede llevar al *overfitting*. Por ejemplo, es posible que una neurona altere sus pesos para corregir los errores de otra. Una adaptación de este estilo suele fallar a la hora de generalizar sobre un conjunto de datos que no ha sido visto [60].

El *dropout* consiste en la eliminación o abandono de ciertos nodos de una red neuronal (ver figura 4.19). De forma temporal, todas las conexiones de un nodo son removidas, de manera que se crea una nueva arquitectura a partir de la red anterior. La eliminación de nodos se realiza aleatoriamente al comienzo de cada iteración y se calcula a partir de una probabilidad establecida durante el armado del modelo [61].

A través del uso del *dropout*, se previenen situaciones como la dependencia entre neuronas, ya que la aparición de ambas en simultáneo dentro de una iteración se vuelve menos probable. La técnica fuerza a la red a aprender patrones más robustos y ayuda a reducir la posibilidad del *overfitting* [60].

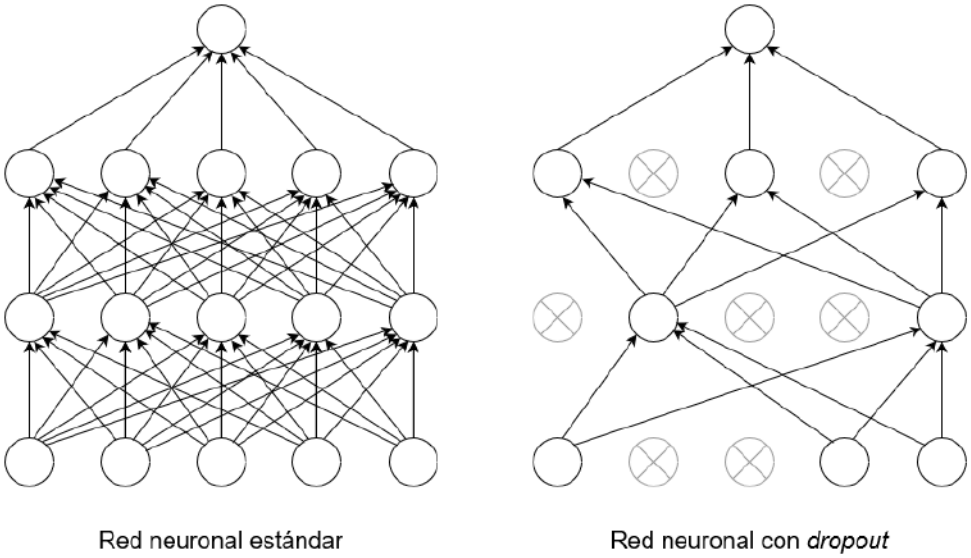


Figura 4.19: A la izquierda, esquematización de una red neuronal estándar con dos capas ocultas. A la derecha, la reducción de la red al aplicar *dropout*. Las unidades tachadas representan aquellas eliminadas (adaptación de Srivastava, *et al.* [61]).

El *dropout* no ha sido aplicado de forma exitosa sobre capas recurrentes [62]. Por esta razón, ha surgido una variante conocida como *dropout* recurrente, que es aplicable sobre este tipo de arquitecturas. Una versión de *dropout* recurrente es la propuesta por Semeniuta, Severyn y Barth, que es utilizada en los modelos LSTM y GRU. En su técnica, la eliminación se aplica sobre las conexiones que realizan la actualización del estado de la celda (LSTM) y el estado oculto (GRU) [63].

4.4.4. Hiperparámetros de entrenamiento

Los hiperparámetros de entrenamiento son los encargados de configurar la forma en la que se lleva a cabo el proceso de aprendizaje de la red neuronal.

4.4.4.1. Cantidad de épocas

Como se observa el figura 4.20, con el correr de las iteraciones el error de entrenamiento tiende a disminuir hasta llegar a un punto de convergencia. Al mismo tiempo, es posible que llegue un momento donde el error de validación comience a crecer, lo que muestra que la red comienza a presentar *overfitting* [64].

Por este motivo, es importante establecer un límite de épocas sobre las cuales se desarrollará el entrenamiento. Es posible definir un límite fijo, aunque el número dependerá de la capacidad del modelo y la cantidad de datos disponibles.

Otra opción es establecer un mecanismo de *early stopping*, que se basa en establecer un conjunto de condiciones para continuar con el entrenamiento de la red. Las reglas pueden ajustarse de múltiples

maneras y pueden ser más permisivas o estrictas, lo cual dependerá de los recursos disponibles a la hora de entrenar (principalmente el tiempo).

Por ejemplo, puede establecerse que el entrenamiento solo continúe si el error de validación se redujo en una cierta cantidad durante la última época. Si se quiere ser más permisivo, es posible establecer un valor de paciencia, en donde se establece un número de épocas máximo para el cual el error deberá ser reducido. En caso de no lograrlo, el entrenamiento se detendrá.

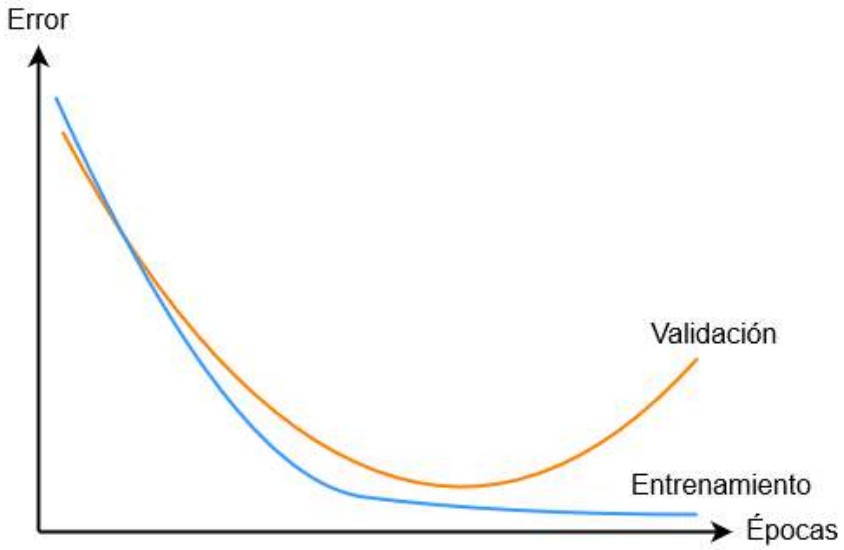


Figura 4.20: Evolución del error de entrenamiento y validación al aumentar la cantidad de épocas (adaptación de [64]).

4.4.4.2. Optimizadores

El entrenamiento de una red neuronal tiene como objetivo minimizar una función de error, también denominada como función de coste. Como se mencionó anteriormente, el error se calcula en base a la diferencia entre las predicciones y los valores reales presentes en el conjunto de entrenamiento. Se trata de una función de los pesos sinápticos de la red.

Los optimizadores son algoritmos encargados de encontrar el conjunto de pesos sinápticos que minimice el error de la red. Matemáticamente, buscan el punto mínimo en la superficie de la función de coste. Son sumamente importantes a la hora de determinar la precisión y la velocidad de entrenamiento del modelo [65].

Descenso por gradiente

Se basa en del cálculo del vector gradiente de la función de coste. Dado que el vector indica la dirección del ascenso más pronunciado sobre la superficie de la función, la técnica consiste en avanzar en la dirección contraria, es decir, aquella hacia donde el descenso sea más pronunciado.

Para un determinado peso i, el ajuste se define como:

$$w_i = w_i - \alpha \frac{\partial J}{\partial w_i} \tag{4.14}$$

donde J es la función de coste y α es la tasa de aprendizaje, un escalar que determina el tamaño del paso en la dirección de descenso. Su elección es fundamental para que el modelo converja rápidamente. Como se ilustra en la figura 4.21, a mayor tasa de aprendizaje, más grande será el paso hacia el mínimo. Sin embargo, un α muy grande puede llevar a que el modelo diverja [66].

Otra forma de acelerar el proceso de convergencia es a través de la técnica de *momentum*. Se basa en la inclusión de un término adicional en el descenso por gradiente, donde se considera el ajuste efectuado en el paso anterior [66]. La ecuación de ajuste modificada queda definida de la siguiente manera:

$$v_i^t = \gamma \cdot v_i^{t-1} + \alpha \frac{\partial J}{\partial w_i} \tag{4.15}$$

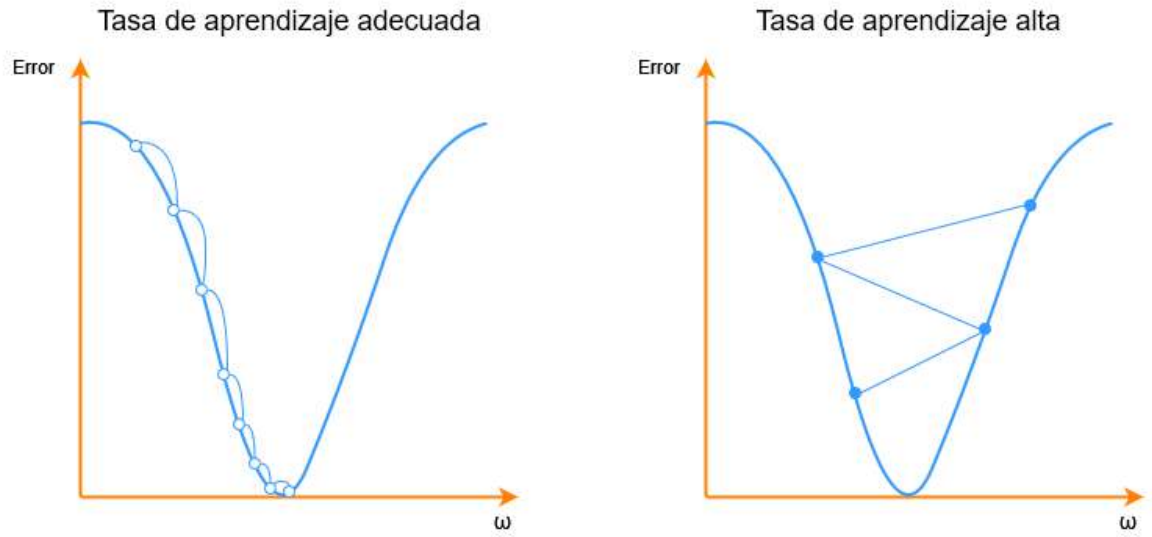


Figura 4.21: Evolución del descenso por gradiente en base a la tasa de aprendizaje (adaptación de [67]).

$$w_i^{t+1} = w_i^t - v_i^t \tag{4.16}$$

donde la velocidad v es el término que acumula los gradientes anteriores, γ es un escalar que ajusta la cantidad de *momentum* aplicado y t representa la iteración del algoritmo.

Descenso por gradiente con *mini-batch*

El costo computacional de cada iteración del descenso por gradiente es de $O(dn)$, donde d es la cantidad de parámetros en la red y n el tamaño del conjunto de entrenamiento [68]. Generalmente, se requiere de conjuntos grandes para lograr una mejor generalización, por lo que el costo acumulado de las operaciones puede volverse prohibitivo. Para solucionar este problema, existen diversas variantes de la técnica original, una de las cuales es el descenso por gradiente con *mini-batch*.

En cada paso del algoritmo, se toma un subconjunto de los ejemplos de entrenamiento, denominado *mini-batch*. Se lo puede expresar como $B = \{x(1), ..., x(m')\}$. El cómputo del gradiente no se efectúa sobre el conjunto de datos completo, sino que la actualización de los pesos solo tendrá en consideración los ejemplos incluidos en el *mini-batch*. Esto puede llevar a que el gradiente muestre fluctuaciones [69], tal como se observa en la figura 4.22. Sin embargo, la eficiencia computacional obtenida resulta importante al entrenar problemas con extensos conjuntos de datos. El descenso por gradiente original es un caso especial del *mini-batch*, donde $m' = m$.

La elección del tamaño del *mini-batch* m' (también denominado como *batch size*) es fundamental a la hora de implementar el algoritmo. Se debe equilibrar la mejora de eficiencia con el riesgo asociado a las fluctuaciones, ya que es posible que el optimizador no logre llegar a la convergencia [69]. Adicionalmente, la relación entre la eficiencia y el *batch size* no es lineal. Cuanto menor sea el tamaño de *batch*, aumentará el *overhead* vinculado a la carga de datos y a la actualización de los pesos sinápticos. A su vez, puede llevar a utilizaciones menos eficientes de ciertos aceleradores de hardware, como las GPU (unidad de procesamiento gráfico) o TPU (unidad de procesamiento tensorial) [56].

RMSprop

Es un algoritmo de optimización propuesto por Geoffrey Hinton en su curso *"Neural Networks for Machine Learning"*. Es una variante de la técnica del descenso por gradiente e introduce modificaciones para mejorar la *performance* del proceso de optimización.

Los gradientes para cada peso sináptico pueden variar en magnitud, lo cual puede dificultar la búsqueda del mínimo global. Para resolver este problema, RMSprop calcula la media móvil de los cuadrados del gradiente de cada peso de la red. El objetivo de la media móvil es compensar el cálculo del gradiente, de manera que los ajustes se mantengan en una magnitud regular. Las magnitudes consistentes ayudan al modelo a evitar los puntos de ensilladura de la función de coste, ya que hasta un gradiente pequeño producirá un paso considerable [70].

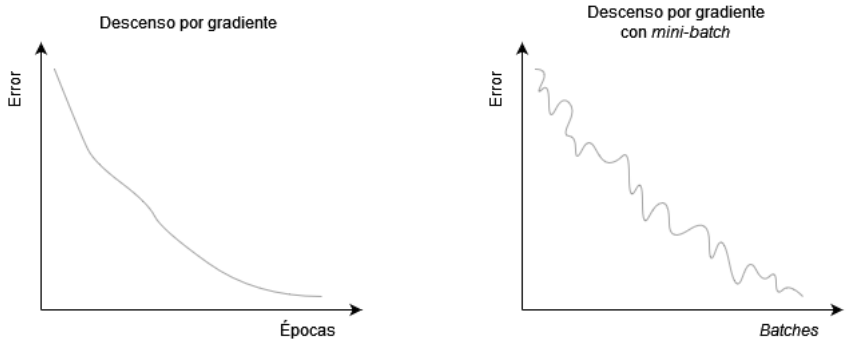


Figura 4.22: Evolución de la función de coste al utilizar descenso por gradiente y la variante con *mini-batch* (adaptación de [69]).

Al calcular la media móvil, el optimizador mantiene tasas de aprendizaje adaptables e individuales para cada peso de la red. Puede resumirse a las siguientes ecuaciones:

$$E[g^2]_i^t = \beta E[g^2]_i^{t-1} + (1-\beta)(\frac{\partial J}{\partial w_i})^2 \tag{4.17}$$

$$w_i^{t+1} = w_i^t - \frac{\alpha}{\sqrt{E[g^2]_i^t}} \frac{\partial J}{\partial w_i} \tag{4.18}$$

Adam

El optimizador Adam es una combinación del descenso por gradiente con *momentum* y el algoritmo RMSProp. Fue introducido por Kingma y Ba en su trabajo *"Adam: A Method for Stochastic Optimization"* [71]. Según sus autores, algunos beneficios de Adam son que:

- Es fácil de implementar.
- Posee alta eficiencia computacional y bajos requerimientos de memoria.
- Es apropiado para funciones de error no estacionarias, donde la superficie es compleja.
- Posee hiperparámetros intuitivos y que requieren poca optimización.

Al igual que RMSProp, Adam computa tasas de aprendizajes adaptables para cada parámetro. El algoritmo calcula medias móviles exponenciales del gradiente y de su cuadrado. La media cuadrática sirve como estimación de la varianza no centrada del gradiente [71]. A continuación se listan sus parámetros de configuración, con la inclusión de los valores recomendados por sus autores e implementados *default* en la librería Keras [72]:

- α: la tasa de aprendizaje. Regula la proporción en que los pesos son actualizados. Valor por *default*: 0,001.
- β_1: tasa de decaimiento exponencial de la media móvil. Valor por *default*: 0,9.
- β_2: tasa de decaimiento exponencial de la media móvil cuadrática. Valor por *default*: 0,999.
- ϵ: constante de estabilidad numérica. Valor por *default*: 10^{-8}.

Se lo representa por medio de las siguientes ecuaciones:

$$m_i^t = \beta_1 \cdot m_i^{t-1} \cdot (\frac{\partial J}{\partial w_i}) \tag{4.19}$$

$$v_i^t = \beta_2 \cdot v_i^{t-i} \cdot (\frac{\partial J}{\partial w_i}) \tag{4.20}$$

$$\hat{m}_i^t = \frac{m_i^t}{1-\beta_1} \tag{4.21}$$

$$\hat{v}_i^t = \frac{v_i^t}{1-\beta_2} \tag{4.22}$$

$$w_i^{t+1} = w_i^t - \frac{\alpha}{\sqrt{\hat{v}_i^t + \epsilon}} \cdot m_i^t \tag{4.23}$$

Levenberg-Marquardt

Levenberg-Marquardt fue publicado por primera vez por Levenberg en 1944, para luego ser redescubierto por Marquardt en 1963. Se implementó por primera vez para el entrenamiento de redes neuronales en el trabajo *"Training feedforward networks with the Marquardt algorithm"* [73] de Hagan y Menhaj, donde recomiendan su uso para redes pequeñas (por debajo de los miles de pesos sinápticos). Es un método utilizado para resolver problemas de mínimos cuadrados no lineales. Proviene de la combinación de dos métodos de minimización: Gauss-Newton y el descenso por gradiente.

El comportamiento del algoritmo se define a través del parámetro μ. Cuando μ es grande, el método se aproxima a un descenso por gradiente con una tasa de aprendizaje pequeña. Cuando es 0, pasa a utilizar el método de Gauss-Newton, el cual es más rápido y preciso. El objetivo es utilizar dicho método lo antes posible, por lo que μ se reduce tras cada paso realizado con éxito (reducción de la métrica de error). Su valor únicamente aumenta cuando un paso tentativo lleva a un mayor error. Por esta razón, la función de coste siempre se reduce en cada iteración [74].

El software MATLAB [75] implementa el algoritmo y lo considera como el método más rápido de optimización, a pesar de tener un mayor requerimiento de memoria. En MATLAB, el valor de μ por *default* es 0.001 [76].

4.4.5. Ajuste de hiperparámetros

A la hora de construir un modelo, existe una fase en donde se realizan diversas pruebas para encontrar el mejor conjunto de hiperparámetros para resolver un problema determinado. Este proceso es denominado ajuste de hiperparámetros (o *hyperparameter tuning*) y puede realizarse de forma manual o automática. Las comparaciones entre grupos de hiperparámetros se realiza a partir del error de validación asociado a cada modelo, el cual es denominado como *score*.

4.4.5.1. *Hyperband*

Existen diversos algoritmos para el ajuste de hiperparámetros. Entre los más básicos, se encuentran la búsqueda por grilla y la aleatoria. Ambos algoritmos son similares, ya que toman rangos de valores para cada hiperparámetro y ejecutan un entrenamiento para cada una de las posibles combinaciones. La única diferencia es que la búsqueda por grilla realiza pruebas sobre los conjuntos de hiperparámetros de forma ordenada, mientras que la búsqueda aleatoria lo hace aleatoriamente. Debido a que el ajuste requiere explorar una gran parte del espacio de posibilidades, son algoritmos de gran ineficiencia y que maximizan los tiempos de optimización.

Con el paso del tiempo, han surgido variantes que permiten realizar una optimización de la búsqueda. En particular, debido a que son capaces de asignar un mayor volumen de recursos (como cantidades de épocas a entrenar) a las configuraciones más prometedoras. Un ejemplo de ello es el algoritmo *Hyperband*.

Hyperband es una variante del algoritmo de búsqueda aleatoria que permite optimizar los tiempos de evaluación de configuraciones. Utiliza el método de *early stopping* para detener entrenamientos con un alto *score* de forma temprana, sin efectuar un entrenamiento con una alta cantidad de épocas [77].

Para su funcionamiento, se debe indicar el rango de valores a tomar para cada hiperparámetro a contemplar en la búsqueda. Por otra parte, se debe establecer una cantidad máxima de configuraciones sobre las cuales se va a realizar la búsqueda y el máximo de épocas a entrenar.

El algoritmo, por su parte, es similar a un proceso de reducción sucesiva en mitades:

1. Se realiza un entrenamiento con una cantidad de épocas mínimo. Una vez finalizado, se descarta la mitad de configuraciones con las que se obtuvo un bajo nivel de *score*.

2. A continuación, se realiza el mismo procedimiento con un aumento en la cantidad de épocas.

3. Se repiten los pasos 1 y 2 hasta que se llega al máximo de épocas de entrenamiento. De esta forma, se genera un podio con los mejores modelos obtenidos.

La gran ventaja que provee el algoritmo es que asigna mayor cantidad de recursos a configuraciones prometedoras que hayan obtenido un valor mínimo de *score*. Los modelos con hiperparámetros que obtuvieron un valor alto de *score* son descartados de forma temprana. De esta forma, no se desperdicia tiempo de búsqueda para hiperparámetros con los cuales no se obtienen buenos resultados [77].

4.4.6. Métricas de regresión

Las métricas son utilizadas para evaluar la *performance* de un modelo con respecto a un conjunto de datos. Se aplican para todas las fases: entrenamiento, validación y testeo. En la tabla 4.1 se listan aquellas utilizadas en el trabajo. Si bien todas las métricas contaron con injerencia en la toma de decisiones, aquella a la que se le dio la mayor relevancia durante el proyecto fue el MAPE. La razón fue que numerosos trabajos relacionados a la predicción de potencia eléctrica utilizan al MAPE como medida del rendimiento de sus modelos.

4.5. Conclusión

A lo largo del capítulo, se presentaron diversos conceptos vinculados al marco teórico dentro del cual se desarrollará el presente trabajo final de carrera.

En primer lugar, se introdujo la temática de las series temporales y su relación con la predicción de potencia eléctrica. Las variables climáticas presentadas serán foco de la experimentación desarrollada en el siguiente capítulo. Principalmente, se analiza el grado de mejora en los resultados ante su inclusión en las entradas de los modelos predictivos.

Posteriormente, se presentó el paradigma conexionista y sus modelos derivados. Debido al análisis realizado sobre el estado del arte (ver sección 1.1), esta tesis se enfocará en la experimentación con las arquitecturas de *Deep Learning*, principalmente LSTM y GRU, junto con sus variantes bidireccionales y los mecanismos de atención. El modelo MLP será utilizado como vía introductoria a la temática. Por otra parte, SVR será principalmente de interés comparativo, sin hacer demasiado énfasis sobre su aplicación práctica.

Además de la presentación de arquitecturas, se introdujeron las metodologías de construcción y entrenamiento de modelos predictivos. En el siguiente capítulo, se presentará el proceso realizado en el marco del proyecto, donde se hará énfasis en la optimización de cada uno de los hiperparámetros mencionados.

Por último, se presentaron las métricas de regresión que serán referenciadas a lo largo del presente informe. Como se mencionó anteriormente, aquella de mayor relevancia en la literatura es el MAPE. Por este motivo, será la más referida durante el desarrollo del trabajo.

Nombre	Ecuación	Descripción
MSE	$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$	Promedia el error cuadrático entre los valores reales y los predichos. Penaliza en mayor medida a los errores grandes, por lo que no es una buena métrica ante datos ruidosos [78].
MAE	$\frac{1}{N}\sum_{i=1}^{N}\lvert y_i - \hat{y}_i\rvert$	Promedia el error absoluto de las diferencias. Más robusto que el MSE al tratar con valores atípicos [78].
MAPE	$\frac{100\,\%}{N}\sum_{i=1}^{N}\left\lvert\frac{\hat{y}_i - y_i}{\hat{y}_i}\right\rvert$	Expresa la precisión del modelo de forma porcentual. Penaliza mayormente errores negativos, donde $\hat{y}_i < y_i$, por lo que puede sesgar el modelo para que produzca predicciones bajas [78].
RMSE	$\sqrt{MSE}$	Calcula la raíz cuadrada del MSE, de manera que el resultado se expresa en la unidad de la variable objetivo. Representa la desviación estándar de los residuos. Al igual que MSE, es sensible a los datos ruidosos [78].
R^2	$1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$	Representa la variación de la variable dependiente que se puede predecir a través del modelo [78].
R^2 ajustado	$1 - \frac{(1-R^2)\cdot(N-1)}{(N-k-1)}$	Toma en cuenta la cantidad de variables independientes del modelo. Permite determinar si las variables añadidas efectivamente ayudan en la predicción [79].

Tabla 4.1: Descripción de las métricas utilizadas a lo largo del proyecto. Las variables utilizadas son:

- N: cantidad de datos en el conjunto.
- y: predicción del modelo.
- $\hat{y}$: valor verdadero.
- $\bar{y}$: media de los valores verdaderos.
- k: número de variables independientes.

Capítulo 5

Experimentación y resultados

En este capítulo, se detallará el proceso de experimentación realizado durante el proyecto. En primer lugar, en la sección 5.1, se explicará el análisis llevado a cabo sobre la base de datos otorgada por el ICyTE y las series temporales almacenadas en ella. Luego, se detallarán los procesos experimentales con los modelos NARX, LSTM y GRU, caracterizados por la utilización de la base de datos (ver secciones 5.2, 5.3 y 5.4). Por último, en la sección 5.5, se realiza una comparativa entre el modelo final propuesto y diferentes alternativas halladas en la literatura.

Los temas serán presentados de forma tal de documentar el proceso de desarrollo y puesta a punto de los modelos, el cual llevó a una implementación final exitosa. A lo largo de la experimentación, se aplicó una estrategia incremental en cuanto a la complejidad de las arquitecturas y las técnicas de optimización de hiperparámetros utilizadas.

Adicionalmente, se incluyen apéndices con las etapas complementarias de experimentación. En los apéndices A.1 y A.2, se presenta el trabajo introductorio realizado con los modelos MLP y NARX. Por otra parte, en el apéndice B, se detallan las pruebas llevadas a cabo con la arquitectura SVR, las cuales fueron de escala reducida en comparación a las de otros modelos.

5.1. Base de datos y series temporales

Con el objetivo de interiorizarse en la problemática, se realizó un análisis sobre la base de datos de series temporales provista por el ICyTE. Se estudió el comportamiento de las mediciones almacenadas, así como la correlación entre las variables predictivas y la potencia eléctrica. A su vez, se explica el proceso de optimización de la base de datos, que fue realizado iterativamente a lo largo del proyecto. Cada una de las tareas fue posible gracias al software de análisis de datos desarrollado en el proyecto (ver sección 6.1).

5.1.1. Base de datos climática

La base de datos provista por el ICyTE contiene datos para 7 variables climáticas: irradiancia, temperatura del aire, velocidad del viento, humedad, declinación y potencia. Posee una resolución de 5 minutos y cada una de sus columnas tiene una precisión de 10 dígitos. A su vez, debido a la gran diferencia entre las dimensiones y el rango de valores posibles, la información se encuentra normalizada. El rango de datos está acotado entre $[-1, 1]$ para la declinación y $[0, 1]$ para el resto de variables. Los factores utilizados en el proceso de normalización fueron los siguientes:

- **Irradiancia:** $2000 \frac{W}{m^2}$.
- **Temperatura del aire:** $50°C$.
- **Velocidad del viento:** $100 \frac{m}{s}$.
- **Humedad:** $100\,\%$.
- **Declinación:** $30°$.
- **Potencia:** $2000W$.

Como última característica, la base de datos cuenta con una columna adicional llamada *datenum*, compuesta por el valor de la fecha y hora de cada medición en un formato numérico manejable por MATLAB [75] [80]. Con este añadido, es posible identificar unívocamente cada una de las mediciones a lo largo del tiempo, con precisión de $\pm 1s$.

Al inicio del presente trabajo, el ICyTE proveyó la información recolectada entre el 1/1/2019 y el 31/1/2020. Posteriormente, se incorporaron los años 2016, 2017 y 2018. El total de datos proporcionado se compuso de 158019 registros. Adicionalmente, la base de datos se encontraba curada, ya que previamente se habían eliminado los datos que indicaban fallos en los sensores. Este procedimiento era necesario para no generar problemas en las predicciones posteriores.

5.1.2. Análisis diario

Al iniciar la experimentación, se decidió analizar el comportamiento de los datos en un período diario. El objetivo era obtener información acerca de la evolución de las series temporales en una escala reducida. El estudio se centró en dos tipos de días: uno de cielo despejado y otro nuboso.

5.1.2.1. Día de cielo despejado

Para comprender la problemática, se comenzó analizando un día despejado (o *clear-sky day*): el día 11 de enero del 2019. Esta clase de días se caracterizan por la ausencia de nubosidad. Dado que la irradiancia solar no es atenuada antes de alcanzar los paneles fotovoltaicos, las condiciones son ideales para la producción de potencia eléctrica [81].

En la figura 5.1, se pueden visualizar las series temporales del día analizado. En base a los gráficos provistos y otros días de cielo despejado examinados, fue posible establecer las siguientes conclusiones acerca del comportamiento diario de las series temporales:

- La irradiancia y la potencia eléctrica siguen series temporales similares, lo cual representa un alto grado de correlación para este tipo de días. Para ambas mediciones, el amanecer marca el comienzo de una tendencia ascendente, mientras que el atardecer una descendente.
- La temperatura del aire muestra un crecimiento sostenido durante el día. Con el correr de las horas, aumenta debido a la irradiancia y la convección con la superficie. Sobre las horas de la tarde, la serie temporal se aplana. Las razones son la caída de la irradiancia y la propia convección, ya que también existe transferencia de calor hacia las capas superiores de aire. La irradiancia comienza a perder poder en comparación al calor perdido, lo cual lleva al enfriamiento del aire [82].
- La serie temporal correspondiente a la velocidad del viento es altamente variable, aunque muestra valores más altos sobre la tarde.
- Para el 11 de enero, la declinación solar muestra una tendencia ascendente. Seguirá creciendo hasta el solsticio de junio.
- La humedad posee una tendencia descendente. Los valores de humedad suelen ser mayores por la mañana, cuando la temperatura del aire es más fría. Por otro lado, el decrecimiento se presenta cuando la temperatura del aire aumenta [83].

5.1.2.2. Día nuboso

Al ser la contraparte de un día de cielo despejado, un día nuboso se caracteriza por dificultar notablemente la producción de potencia eléctrica. Esto sucede ya que la irradiancia solar no impacta de forma directa sobre los paneles fotovoltaicos, debido a las nubes interpuestas.

Para analizar la respuesta de la potencia ante estas condiciones, se analizó la base de datos y se tomó como referencia visual el día 30/06/2019 (ver figura 5.2). Para ello, se utilizó el software de análisis de datos que fue desarrollado en el proyecto. A partir del análisis general, se obtuvieron las siguientes conclusiones:

- Para este caso, las series temporales de potencia e irradiancia se mantienen similares, por lo que se observa que se mantiene la correlación. Sin embargo, se dificulta la apreciación de cierta tendencia, aunque los valores mayores se encuentran cerca del mediodía y los menores a comienzos y finales del día.

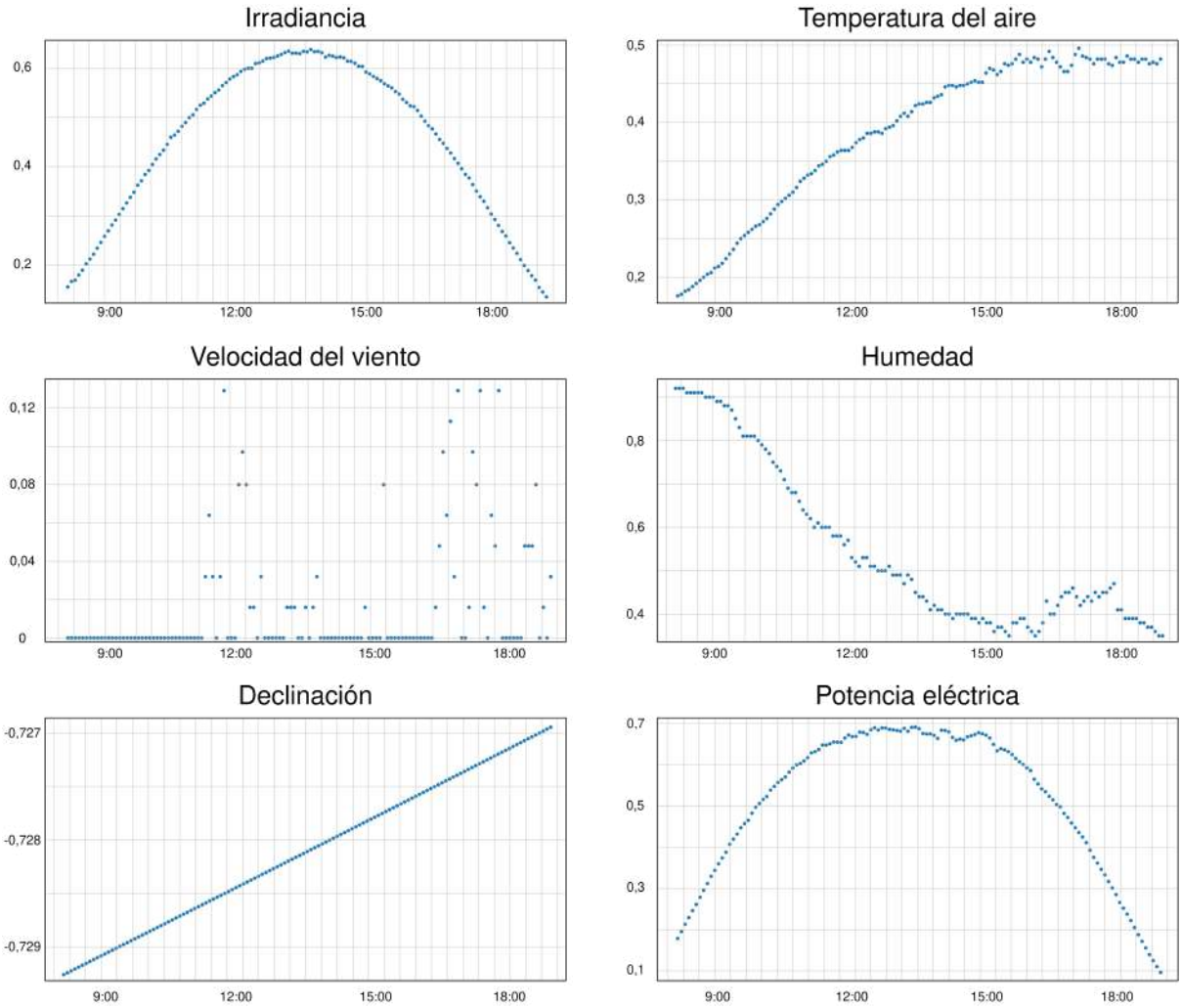


Figura 5.1: Series temporales correspondientes al 11/01/2019.

- El comportamiento de la temperatura del aire es similar que para un día despejado, aunque se puede observar que toma valores menores o dispersos por la baja o difusa presencia de irradiancia.
- La serie temporal de la velocidad del viento se observa variable, de igual forma que para un día de cielo despejado.
- Para el 30 de junio, la serie temporal correspondiente a la declinación solar tiene una tendencia descendente y seguirá en disminución hasta el solsticio de verano.
- Para días con presencia de nubosidad, se observa que la humedad posee una tendencia descendente. Sin embargo, su reducción no es sustancial, debido a que la radiación solar no alcanza la superficie terrestre y la temperatura del aire se mantiene baja. Al imposibilitar el paso de la irradiancia, la relación entre los niveles de humedad y la producción de potencia eléctrica es negativa.

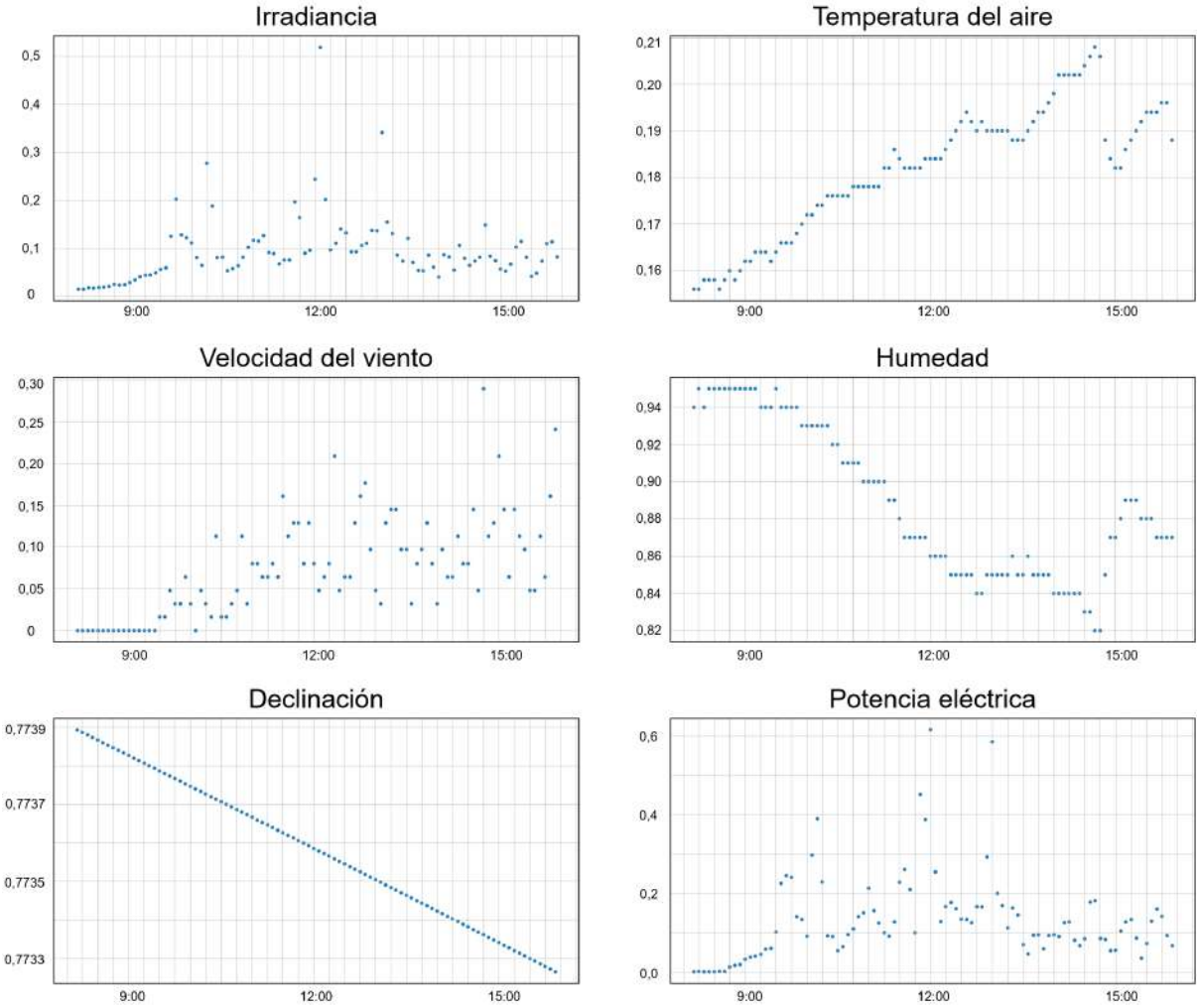


Figura 5.2: Series temporales correspondientes al 30/06/2019.

Por otra parte, se encontraron algunos casos en donde el día comienza siendo despejado y, en cierto momento, la potencia sufre una drástica caída. Para ejemplificar, se menciona el día 16/12/2017 (ver figura 5.3). En este caso, cerca del mediodía, se observa una repentina disminución de la potencia hasta un valor umbral, que es determinado por el sensor medidor. Este tipo de situaciones se suelen dar por fuertes tormentas (por ejemplo, tormentas de verano) que se presentan en un día de cielo despejado. En estos casos, se puede apreciar una nubosidad densa y oscura que imposibilita el paso de radiación solar, de manera que los sensores registran muy bajos niveles de potencia e irradiancia.

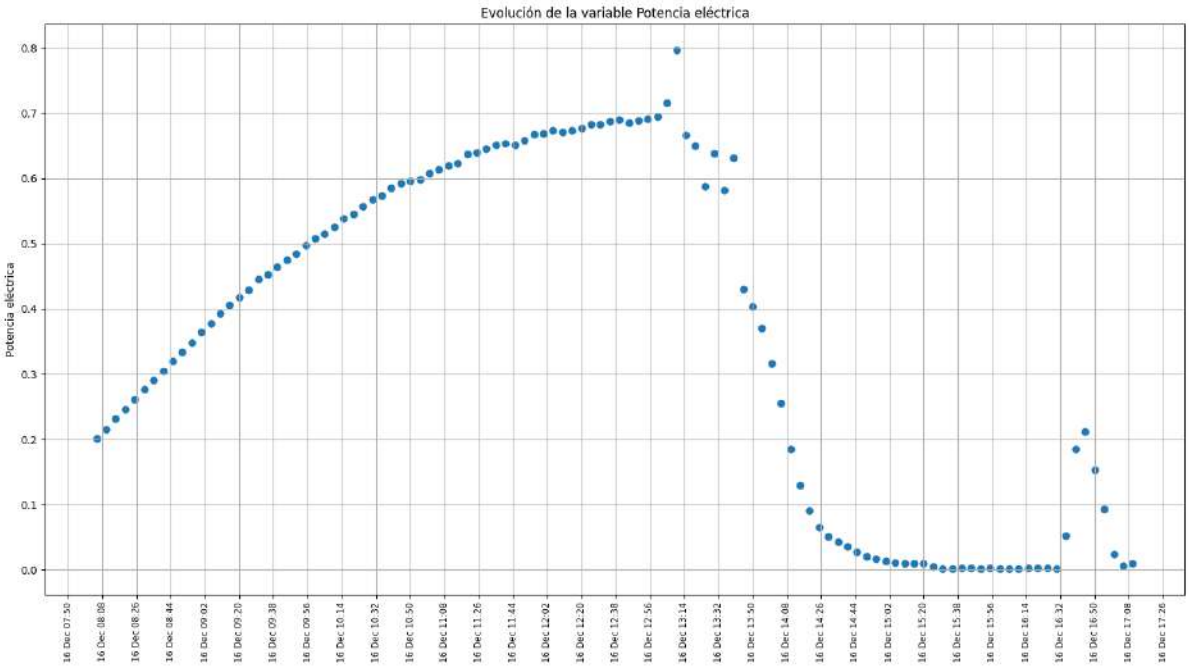


Figura 5.3: Serie temporal correspondientes a la potencia eléctrica generada el 16/12/2017. Para la irradiancia, se observó un comportamiento similar.

5.1.2.3. Periodicidad estacional

Dentro del análisis realizado previamente, se detectó que el período en el cual se registran los datos varía de forma estacional. Las mediciones climáticas son realizadas durante los momentos del día en donde Mar del Plata recibe irradiancia solar y la planta es capaz de producir potencia eléctrica. Generalmente, la recopilación comienza a las 08:00 h. En verano, esta finaliza alrededor de las 19:00 h. Sin embargo, el intervalo en invierno suele ser más reducido. Por este motivo, el período de la función no es constante a lo largo del año. Este comportamiento se puede observar en la figura 5.4, en donde se evidencia que las mediciones del día invernal finalizan con una hora de antelación.

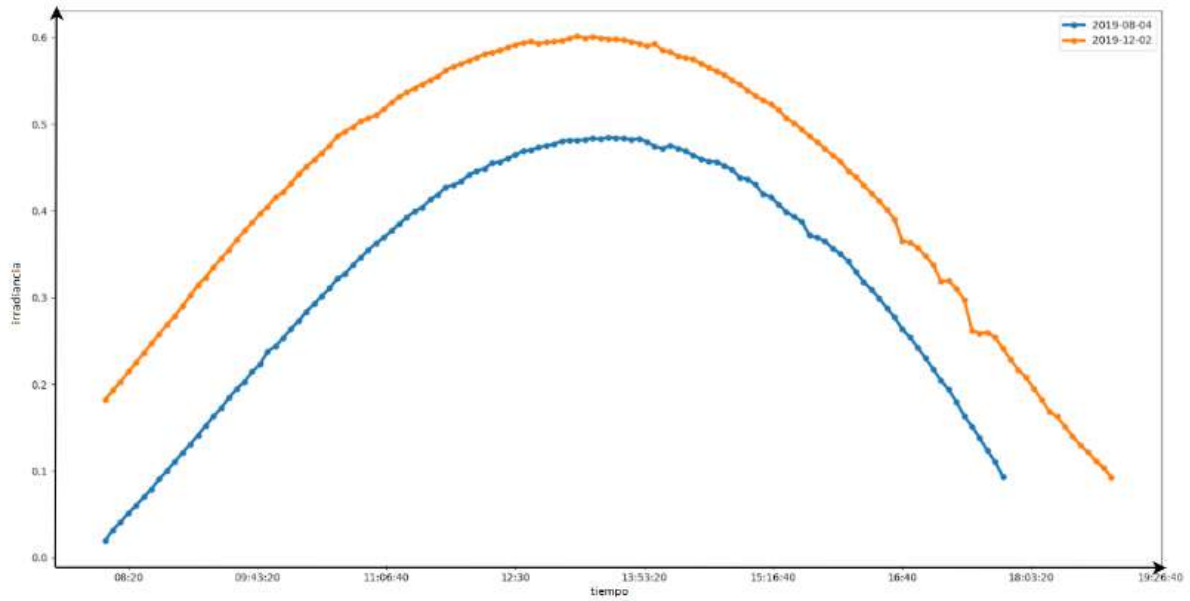


Figura 5.4: Gráfico comparativo de la irradiancia detectada en un día de verano y uno de invierno.

5.1.3. Análisis anual

El siguiente paso fue efectuar un análisis que abarque la totalidad de un año. Se tomó como referencia el 2019, dado que fue el primer año provisto por el ICyTE como base de la experimentación.

5.1.3.1. Evolución de las series temporales

Como puede visualizarse en la figura 5.5, el aumento del intervalo temporal hace evidentes los patrones estacionales que afectan a las variables climáticas.

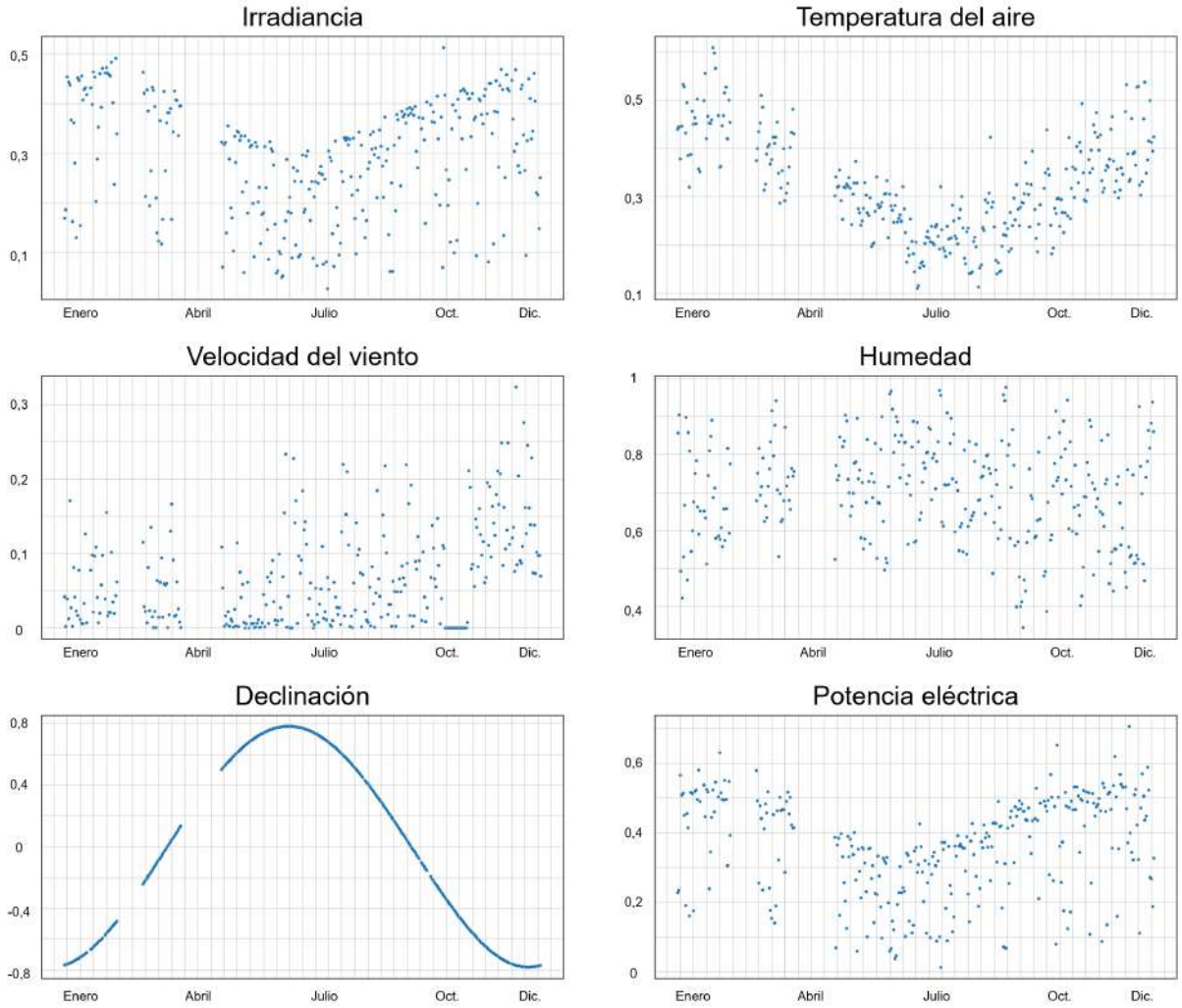


Figura 5.5: Series temporales correspondientes al año 2019. Cada punto representa el promedio diario de la medición. Los espacios vacíos indican adquisiciones faltantes en la base de datos.

En primer lugar, puede observarse que la potencia eléctrica decrece durante los meses invernales. Este hecho se explica observando los gráficos de irradiancia y temperatura del aire. Sus series temporales marcan un patrón descendente cuando la incidencia solar disminuye, por lo que los paneles fotovoltaicos reducen su producción. A su vez, se visualiza un leve aumento de la humedad en los meses fríos, justificado por el decrecimiento de la temperatura del aire.

Por otro lado, la gráfica anual permite ilustrar con exactitud la serie temporal correspondiente a la declinación solar. Como se mencionó anteriormente, el valor máximo se sitúa en el solsticio de junio. Antes del invierno, la declinación crece hasta alcanzar su pico. Posteriormente, decrece hasta el solsticio de verano, donde el ciclo vuelve a comenzar.

A contraposición del resto de variables, la velocidad del viento se caracteriza por sus promedios irregulares. La serie temporal no muestra evidencias de poseer una componente estacional en el período analizado.

5.1.3.2. Análisis de correlaciones

Como complemento del análisis realizado sobre la evolución de las series temporales, se procedió a calcular el coeficiente de correlación entre cada una de ellas y la potencia eléctrica. El objetivo fue determinar la dependencia de la potencia generada con respecto al resto de mediciones. A continuación, se presentan los valores obtenidos para cada variable:

- Irradiancia: 0,89.
- Humedad: -0,41.
- Temperatura del aire: 0,28.
- Declinación: -0,28.
- Velocidad del viento: 0,18.

En base a los resultados obtenidos, pudieron extraerse diversas conclusiones:

- En primer lugar, se confirmó lo observado en el estudio de la evolución de las series temporales: la irradiancia es la variable de mayor impacto en la producción de potencia eléctrica (ver figura 5.6). Posee una correlación positiva fuerte, muy cercana a 1. Por este motivo, la potencia generada será mayor en meses de verano, donde la incidencia solar aumenta.
- Nuevamente, se confirma que la humedad es una variable que influye negativamente en la generación de potencia. Es la medición con correlación negativa más fuerte en la base de datos (ver figura 5.7), lo que pudo visualizarse en el análisis del día nuboso. Dado que los altos niveles de humedad dificultan el paso de la radiación solar, la producción de los paneles será menor.
- La temperatura del aire, la declinación y la velocidad del viento no influyen de forma significativa en la producción de potencia eléctrica. Sin embargo, podrían añadir información adicional que sea de utilidad al modelo. En el caso de la declinación, por ejemplo, se puede observar una correlación negativa débil, dado que sus valores máximos ocurren en el invierno del hemisferio sur. A diferencia del resto de variables, la declinación puede ofrecer información temporal al modelo, ya que establece la época del año sobre la cual se efectúan las predicciones.

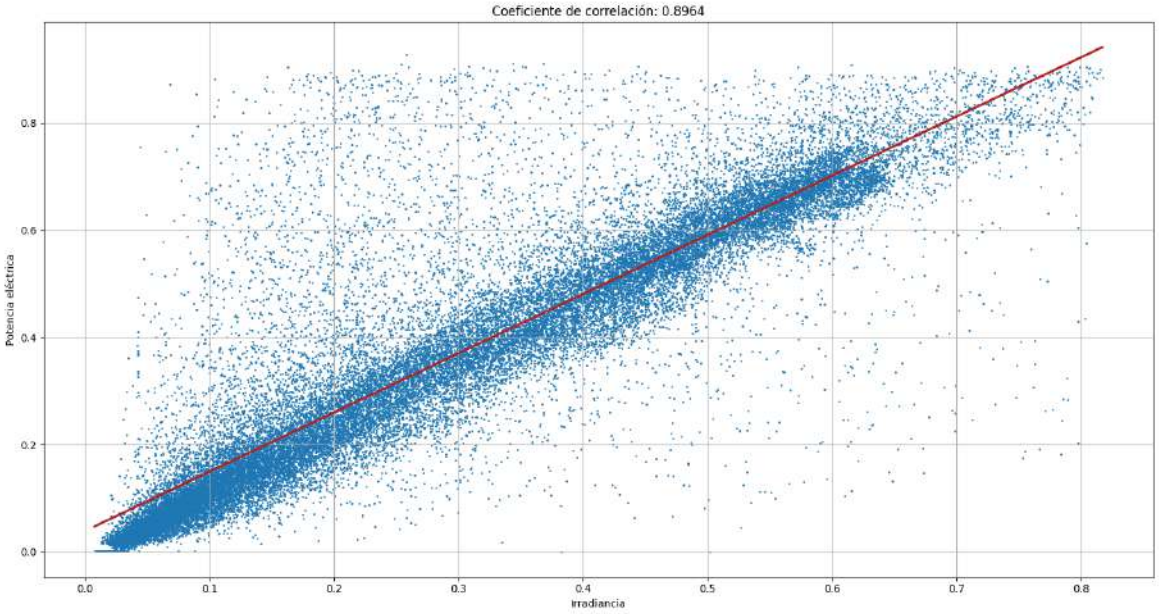


Figura 5.6: Correlación de la irradiancia con respecto a la potencia para el año 2019.

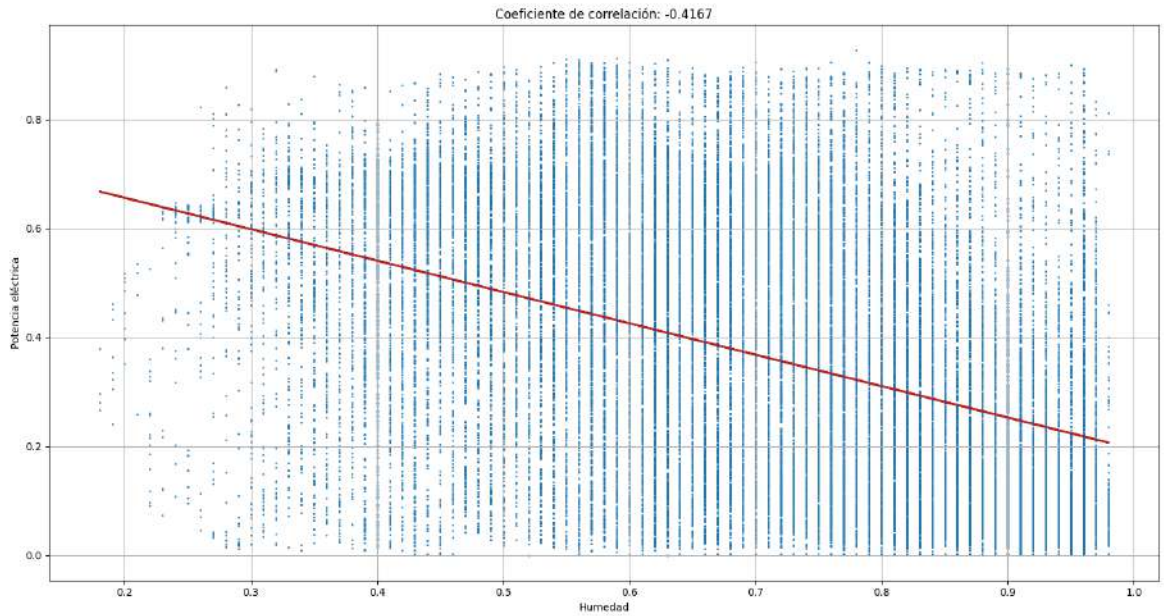


Figura 5.7: Correlación de la humedad con respecto a la potencia para el año 2019.

5.1.4. Limpieza de datos

A lo largo del proceso de experimentación, se detectaron anomalías de diferente índole en la base de datos otorgada por el ICyTE. A continuación, se detalla cada una de ellas, su impacto en los datos y las acciones llevadas a cabo.

5.1.4.1. Valores inválidos (NaN)

Al trabajar con el año 2019, se identificó la presencia de valores NaN (*Not a Number*) en el conjunto de datos. En total, el año contaba con 50 mediciones cuyo valor de irradiancia se almacenaba como NaN. 49 de ellas ocurrían consecutivamente en el día 30/9/2019, entre las 13:55 y las 17:55 (último valor del día). La medición adicional correspondía al 7/10/2019 a las 11:40, donde los registros anterior y posterior se situaban a la distancia típica de 5 minutos. Para solucionar este problema, se optó por eliminar las mediciones mencionadas del conjunto de datos. Posteriormente, situaciones similares se detectaron en los conjuntos correspondientes al 2016, 2017 y 2018, donde se procedió de la misma forma.

5.1.4.2. Valores atípicos

Durante los procesos de experimentación con modelos, se utilizó el software de análisis para verificar la consistencia de la base de datos. Se encontró un conjunto de valores atípicos, caracterizados por tener valores de potencia cercanos a 0 y niveles de irradiancia altos.

Un ejemplo de día con mediciones atípicas es el 9/12/2019. En la figura 5.8, puede observarse que los niveles de irradiancia decaen de manera natural durante la tarde. Sin embargo, la medición de potencia eléctrica a mitad del período fue cercana a 0. Este hecho, sumado a que no se cuenta con las mediciones posteriores, permite suponer que el medidor tuvo un error durante el lapso analizado. A su vez, puede observarse que a partir de las 16:05 h los valores de potencia vuelven a la normalidad.

Para preservar la consistencia de la base de datos, se decidió eliminar las mediciones consideradas como atípicas. A su vez, 11 días con largos períodos de datos faltantes fueron eliminados. Para aquellos con períodos entrecortados, como el 9/12/2019, se decidió únicamente eliminar la porción final de datos.

5.1.5. Conclusión sobre la base de datos y sus series temporales

En base al análisis de la evolución de las series temporales, puede destacarse que el problema de la predicción de potencia eléctrica se enfrenta a series temporales con componentes mayormente estacionales. Los gráficos anuales evidencian la existencia de este tipo de patrones, distinguibles por la estación del año en la cual se realizó la medición. En promedio, la generación de potencia eléctrica evidencia un

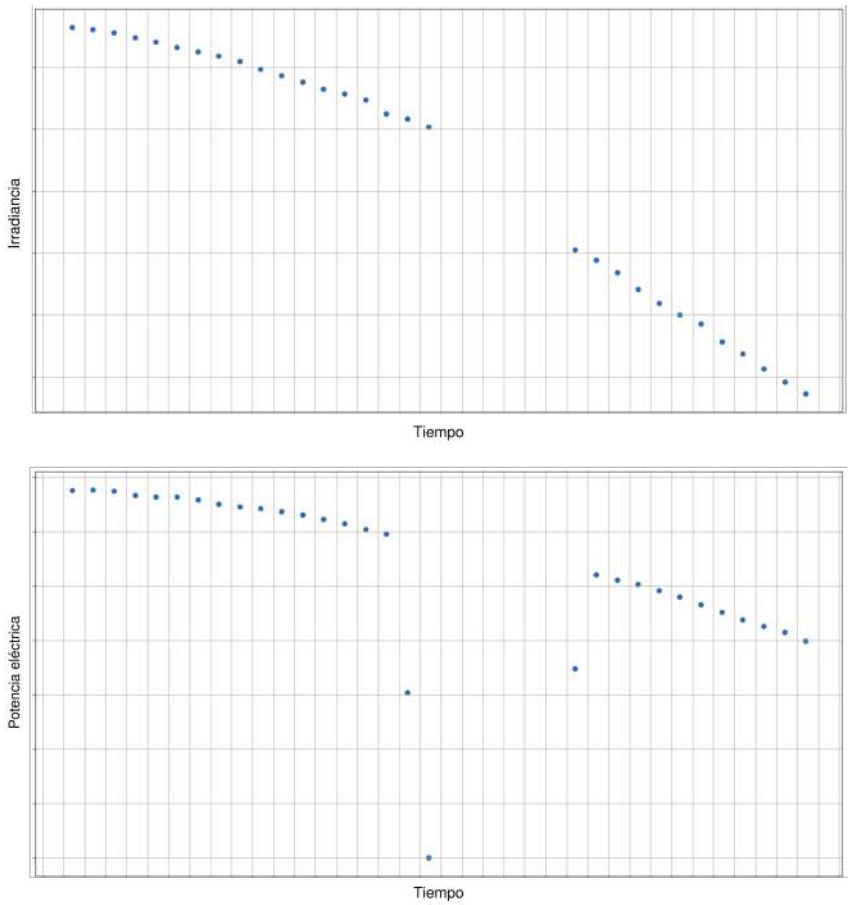


Figura 5.8: Evolución de la irradiancia (arriba) y la potencia eléctrica entre las 14:00 h y las 17:00 h del día 9/12/2019.

mayor rendimiento durante los meses calurosos, en contraposición a la temporada de invierno donde la incidencia solar es menor. El análisis de correlaciones dejó en evidencia este hecho, dado que la irradiancia es la variable de mayor correlación con la potencia eléctrica. Si bien es posible que ciertas variables también posean una componente de tendencia (por ejemplo, debido al cambio climático), su efecto no es significativo para los períodos analizados durante el desarrollo del proyecto.

Posteriormente, el análisis de correlaciones fue de utilidad para comprender la relación de las mediciones climáticas con la potencia eléctrica. Se identificó a la irradiancia como la variable de mayor importancia en la producción de energía, seguida de los niveles de humedad.

Finalmente, las tareas de limpieza de datos fueron necesarias para la preservación de la consistencia de la base de datos provista por el ICyTE. Se identificaron diversos valores inválidos, que fueron removidos de los conjuntos de datos utilizados durante la fase de experimentación.

Ya comprendido el modelo físico-meteorológico y las variables presentes en la base de datos, comenzaremos con el problema de encontrar un modelo predictivo capaz de aproximar con exactitud la potencia eléctrica generada por la planta fotovoltaica del ICyTE.

5.2. Perceptrón multicapa

5.2.1. Motivación de la investigación

Debido a su simplicidad, el perceptrón multicapa fue escogido como puntapié de la investigación. En primer lugar, se estudió el estado del arte relacionado a los MLP en problemas de predicción similares al de la potencia eléctrica. Si bien existen trabajos que utilizan el modelo [84] [85] [86] [87], sus resultados han sido superados por otros con mayor relevancia en la actualidad, como LSTM o GRU [7] [88] [2] [19]. A pesar de lo mencionado anteriormente, se consideró de utilidad estudiar y experimentar sobre el MLP, ya que representaba un primer acercamiento a los modelos de redes neuronales.

5.2.2. Implementación

El paso previo a la experimentación fue desarrollar la implementación del modelo. Para ello, se decidió utilizar el lenguaje de programación Python [89], junto con las librerías Keras [90] y TensorFlow [91]. Por otra parte, se optó por utilizar la biblioteca Matplotlib [92] para la construcción de gráficos.

Las librerías fueron escogidas debido a que se encuentran entre las más populares y completas para proyectos de desarrollo orientados a *Machine Learning* [93]. Su facilidad de uso y el acceso a gran cantidad de documentación permitió que la implementación fuera desarrollada rápidamente y sin complicaciones.

5.2.3. Pruebas realizadas

Para concretar un primer acercamiento práctico a las redes neuronales, se decidió utilizar el MLP para trabajar con un problema sencillo: la predicción de funciones $sen(x)$ y $tan(x)$. La elección se debió a sus características periódicas, similares a las encontradas en la serie temporal de la potencia eléctrica. Las pruebas realizadas se encuentran en el apéndice A.1.

5.2.4. Conclusión sobre el perceptrón multicapa

La experimentación con el perceptrón multicapa fue de utilidad para introducirse a la construcción de modelos predictivos. Sin embargo, como se observó en los resultados obtenidos, es una red incapaz de extrapolar una función periódica. Por este motivo, junto con la existencia de arquitecturas con mejores resultados en la literatura, se descartó realizar un análisis posterior con datos reales de la problemática analizada.

5.3. NARX

5.3.1. Recopilación de entornos y librerías

El primer paso para el análisis del funcionamiento de una red NARX fue construir una implementación del modelo. Como primera opción, se optó por recurrir a la librería Keras, que había sido utilizada en la etapa anterior. Sin embargo, dentro de sus herramientas no se cuenta con una implementación directa del modelo NARX.

Si bien Keras posee una implementación genérica de una red neuronal recurrente, llamada SimpleRNN [94], no es posible interactuar con el tamaño de las ventanas temporales aplicadas por el modelo. Por otra parte, la red mantenía una memoria en cada neurona, pero no se permitía la retroalimentación de la potencia eléctrica. Esto hacía imposible el armado de un modelo NARX a partir del uso de SimpleRNN. Posteriormente se investigó acerca de TensorFlow, pero tampoco se logró encontrar una implementación.

Por los motivos anteriormente expuestos, se decidió recurrir a un nuevo abanico de librerías: SysIdenPy [95], Pyneurgen [96] y FireTS [97]. Por otra parte, se añadió como una de las posibilidades al entorno MATLAB [75], el cual da soporte para el trabajo con redes NARX.

5.3.2. Pruebas introductorias

Antes de trabajar con la base de datos del ICyTE, fue necesario realizar un conjunto de pruebas preliminares para definir la librería o entorno a utilizar. A su vez, se deseaba efectuar un primer acercamiento al modelo NARX y comprobar sus capacidades. Al igual que para el modelo MLP, se decidió experimentar con la predicción de la función $sen(x)$.

Una vez llevadas a cabo las pruebas planteadas, se concluyó que todas los requerimientos para trabajar con la predicción de potencia eran cumplidos por MATLAB. Además de contar con una completa capacidad de optimización de hiperparámetros, fue de utilidad la gran cantidad de recursos y documentación disponibles acerca del entorno.

La totalidad de pruebas realizadas y el análisis completo de librerías se puede encontrar en el apéndice A.2.

5.3.3. Predicción de un día

Luego de la experimentación preliminar, se procedió a trabajar con un conjunto de pruebas formado a partir de la base de datos provista. Inicialmente, se estableció como tarea lograr la predicción de un único día del año 2019: el 3/11/2019. La razón de su elección fue que, como se observa en la figura 5.9, es un *clear-sky day*, condición que facilitaba las tareas predictivas. Para ello, se definió un modelo inicial sobre el cual trabajar, detallado a continuación:

- **Arquitectura:** una única capa oculta con 10 neuronas y función de activación de tipo sigmoidea. Se establecieron ventanas temporales endógena y exógena (ver sección 4.2.2.1) con un tamaño de 10.
- **Función de activación:** sigmoidea.
- **Método de ajuste:** se utilizó el método Levenberg-Marquardt, recomendado por MATLAB para una amplia variedad de problemas [98].
- **Conjunto de datos:** la red fue entrenada con 124 datos de un único día (2/11/2019) y se utilizó un *split* fijo del 80 % para los datos de entrenamiento y el 20 % restante para la validación. El *split* definido se basó en los resultados obtenidos en las pruebas iniciales con modelos MLP (ver sección A.1).
- **Variables de entrada:** inicialmente, se utilizó únicamente la irradiancia.
- **Retroalimentación:** interna (red cerrada).

Con la configuración planteada (ver figura 5.10) se obtuvo un valor de MAPE del 33,35 % para el día 3/11/2019. Posteriormente, para lograr un ajuste más preciso y disminuir el nivel de error, se realizó un *tuning* de la estructura de la red. Al ser las primeras pruebas, se decidió realizar los ajustes manualmente. A continuación, se detalla cómo respondió el modelo ante cada uno los cambios:

- **Ventanas temporales:** reducir la ventana llevó a mejores resultados. Un tamaño de 3 logró un MAPE del 19 %, junto con una gráfica suave. Por otra parte, los aumentos llevaron a resultados peores.
- **Función de activación:**
  - **ReLU:** el modelo se desestabilizó más allá de una ventana de tamaño 5. Con tamaños pequeños (2 o 3), sus resultados fueron ligeramente peores que los obtenidos con una función de activación sigmoidea.

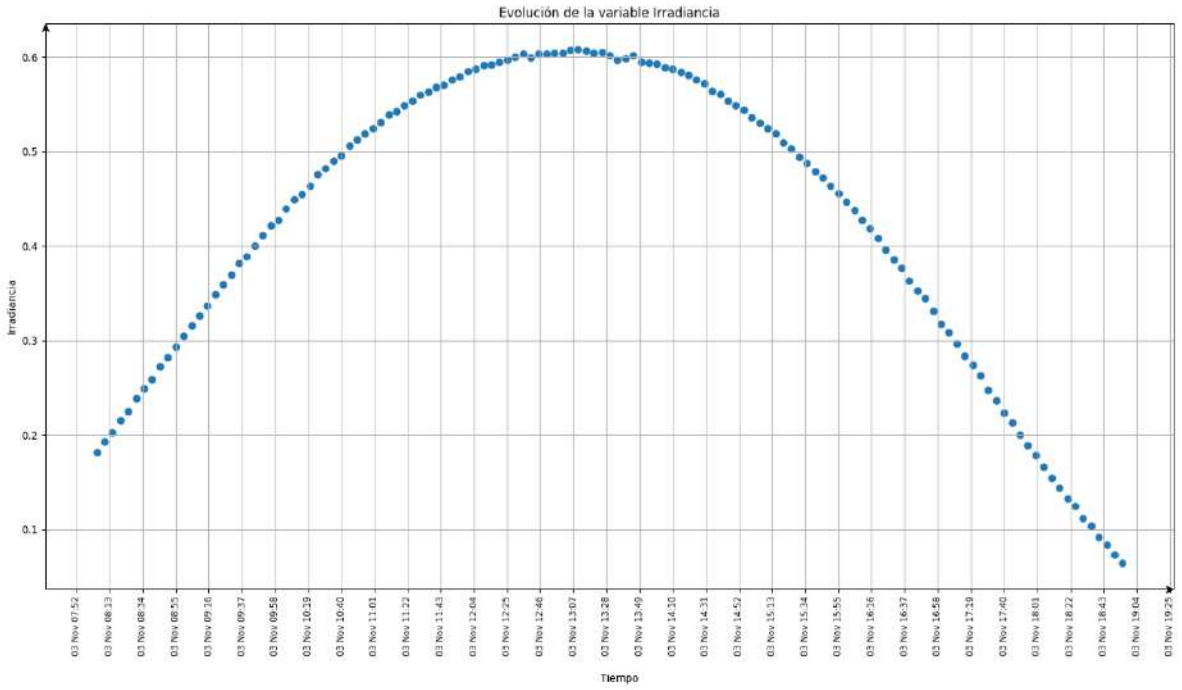


Figura 5.9: Evolución de la irradiancia para el *clear-sky day* correspondiente al 3/11/2019.

- ***tanh:*** se obtuvieron resultados sustancialmente peores, por el orden del 30 % de MAPE.

- **Cantidad de neuronas:** tanto aumentar como disminuir su cantidad llevó a resultados visiblemente peores.
- **Optimizador:** al utilizar gradiente con *momentum*, se requirió de mayor tiempo para converger y no mejoraron los resultados.
- **Expansión de la cantidad de días de entrenamiento de 1 a 7 días (27/10 - 2/11):**
  - En primera instancia, mantener la configuración anterior implicó la obtención de peores resultados.
  - Tras varias pruebas, se estableció en 8 la cantidad de neuronas y en 5 el tamaño de las ventanas. Con este modelo, se obtuvo un MAPE del 9,58 %.
  - Expandir a 2 semanas no logró mejorar el entrenamiento.

En conclusión, el modelo que minimizó el error obtenido utilizó una configuración de 8 neuronas en su capa oculta, ventanas temporales de tamaño 5 y optimizador Levenberg-Marquardt. Con un rango de entrenamiento del 27/10 al 2/11, el MAPE final para el 3/11 fue 9,58 %.

5.3.3.1. Agregado de nuevas variables

A continuación, se procedió a realizar un análisis sobre las variables de entrada adicionales que podían añadirse al modelo. Para ello, se calculó la correlación de las mismas con la potencia durante el rango de fechas utilizado. A partir del uso del software de análisis de datos, se obtuvieron los siguientes coeficientes de correlación:

1. Irradiancia: 0,91.
2. Temperatura del aire: 0,37.
3. Humedad: -0,31.
4. *Datenum*: 0,22.
5. Declinación: -0,22.

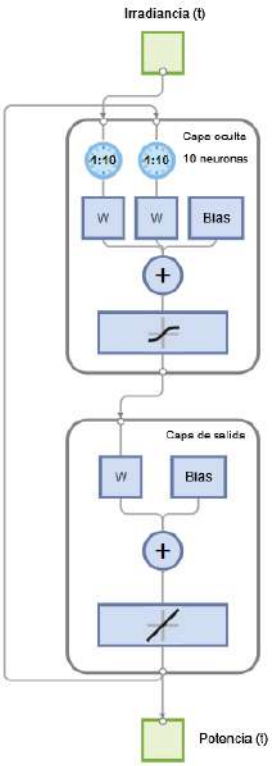


Figura 5.10: Arquitectura de la red NARX obtenida para el primer modelo planteado.

6. Velocidad del viento: sin datos en el período.

Las variables se añadieron de forma unitaria en el orden enunciado para poder analizar la respuesta del modelo ante cada una de ellas. Los resultados obtenidos fueron los siguientes:

- **Temperatura del aire:** con su añadido al modelo se logró una muy pequeña disminución en el valor del MAPE, a un 9,32 %.
- **Humedad:** al añadir la variable junto con la irradiancia y la temperatura del aire, los resultados empeoraron y se obtuvo un MAPE del 19,43 %.
  - Para disminuir el nivel de error, se estableció la cantidad de neuronas en 6. El ajuste llevó a un MAPE del 7,4 %. No se lograron mejoras al establecer en otros valores la cantidad de neuronas.
  - Al aumentar el rango de entrenamiento a 3 semanas, los resultados empeoraron. Tras aumentar la cantidad de neuronas y establecer las ventanas temporales en 3, se logró un MAPE del 8,8 %.
- ***Datenum*:** su añadido al modelo empeoró los resultados a un MAPE del 17,52 %. Se realizó un cambio de la cantidad de neuronas a 8 y se obtuvo un MAPE del 11,6 %, sin mejorar los resultados con ningún cambio adicional.
- **Declinación:** al añadirla al mejor modelo previo a la inclusión de *datenum*, el error aumentó a un 22,93 %. Con un cambio en la cantidad de neuronas a 7 se obtuvo el mejor resultado para 3 semanas de entrenamiento: 8,1 %. De todas formas, como se observa en la figura 5.11, la predicción resultante es escalonada.

A modo de conclusión, se obtuvo el menor nivel de error (7,4 %) con una red que utilizó irradiancia, temperatura del aire y humedad como variables de entrada para un entrenamiento de 2 semanas. La arquitectura de la red consistió en 8 neuronas en su capa oculta, un tamaño de ventanas temporales de 6 y optimizador de tipo Levenberg-Marquardt.

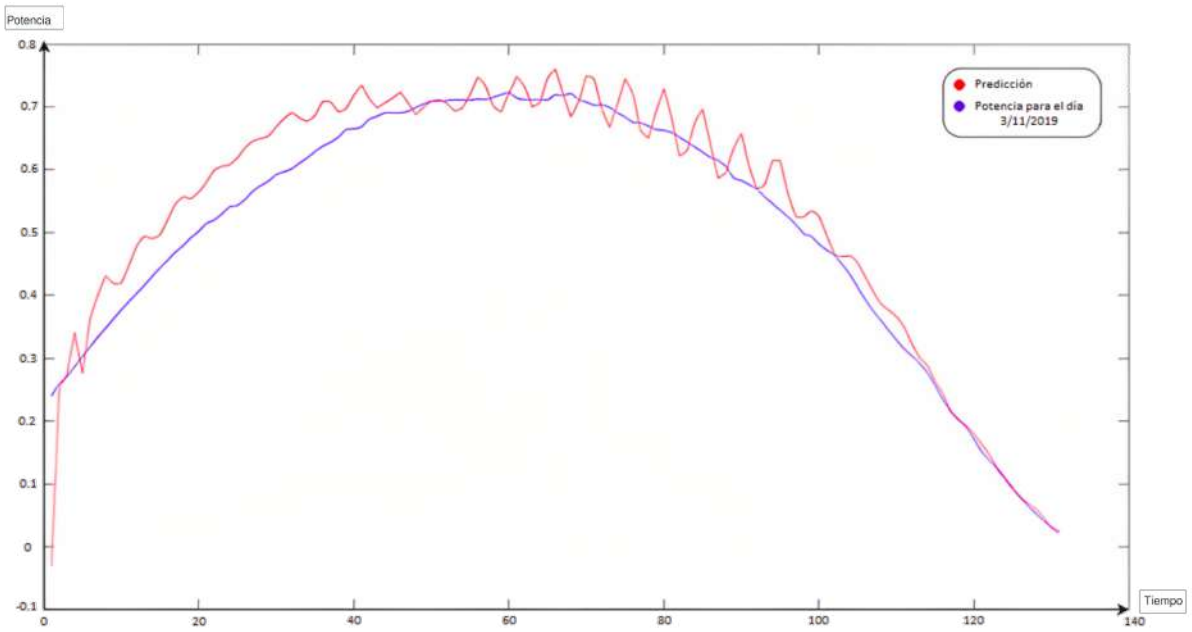


Figura 5.11: Predicción de potencia eléctrica para el día 3/11/2019 con el modelo con el cual se obtuvo el menor MAPE.

Por otra parte, la estructura de la red con la cual se obtuvo el menor nivel de error (8,1 %) para 3 semanas de entrenamiento utilizó irradiancia, humedad, temperatura del aire y declinación. La arquitectura final del modelo consistió en una capa oculta de 7 neuronas, ventanas temporales de tamaño 3 y optimizador Levenberg-Marquardt.

5.3.4. Predicción de un mes

Como siguiente paso, se estableció el objetivo de realizar predicciones para cada día a lo largo de un mes entero. Se determinó que el conjunto de entrenamiento fuera el correspondiente al año 2019 completo. Para la etapa de *testing*, se utilizó únicamente el mes de enero de 2020. A través de la experimentación, fue posible determinar con mayor robustez el error diario del modelo, dado que el mes analizado contó con una amplia variedad de días soleados y nubosos.

Se estableció la siguiente estructura inicial:

- **Arquitectura:** una única capa oculta compuesta por 8 neuronas y función de activación sigmoidea. Se estableció el tamaño de las ventanas temporales endógena y exógena en 5.
- **Método de ajuste:** Levenberg-Marquardt.
- **Conjunto de datos:** se utilizó un total de 36322 datos, con un *split* del 80 % para los datos de entrenamiento y el 20 % restante para la validación.
- **Variables de entrada:** debido a la inclusión de un alto volumen de datos, se decidió comenzar el análisis con la irradiancia únicamente.

Con la estructura planteada se obtuvo un MAPE promedio diario del 17,7 %. Las predicciones obtenidas se observan en la figura 5.12.

Para llegar a un modelo óptimo y minimizar el nivel de error, se realizaron ajustes a la arquitectura de la red. Los cambios planteados fueron los siguientes:

- **Tamaño de ventanas temporales:** utilizar valores entre 2 y 15 generó resultados similares. Con un valor de 8, se logró mejorar el valor de MAPE al 17,04 %. Al aumentar a valores mayores de 20, el entrenamiento se volvió más lento y no se obtuvo mejora alguna.
- **Función de activación:**

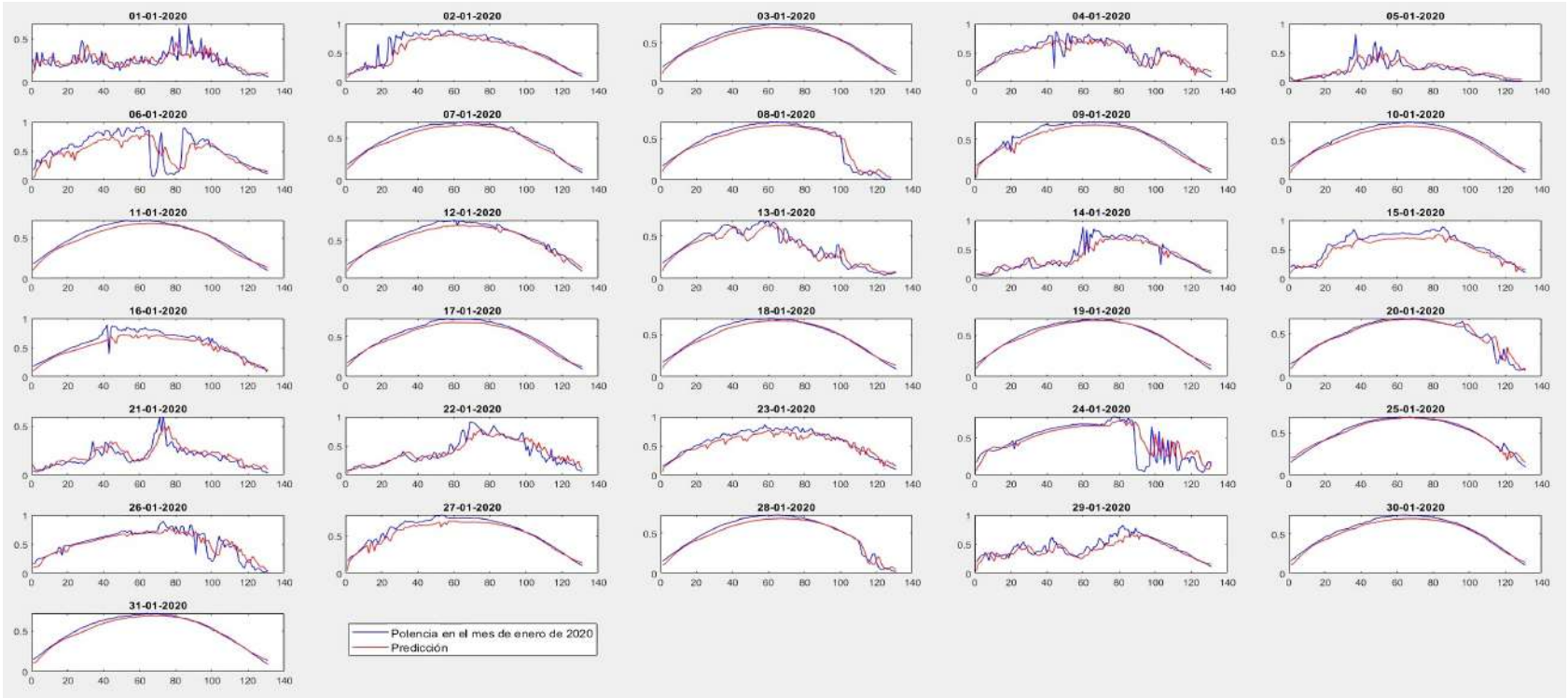


Figura 5.12: Predicción de la potencia para el mes de enero del año 2020.

  - **ReLU:** implicó un aumento notable del MAPE, incluso con la alteración del tamaño de las ventanas temporales y la cantidad de neuronas en la capa oculta.
  - ***tanh:*** los resultados obtenidos mejoraron levemente los expuestos en el modelo base (por ejemplo, se disminuyó el valor de MAPE promedi diario a un 16,78 %). Se decidió continuar el análisis con su uso.

- **Cantidad de neuronas:** valores entre 3 y 8 mantuvieron el valor de MAPE en 17 % aproximadamente. Más allá de 9, el modelo presentó inestabilidad en ciertas pruebas. A partir de las 25 neuronas, la inestabilidad se volvió más marcada.

- **Optimizadores:**

  - **Gradiente con *momentum*:** fue extremadamente más lento para converger, sin mejorar los resultados obtenidos con Levenberg-Marquardt. Aumentar el *learning rate* llevó a una mayor inestabilidad.
  - **Gradiente con LR adaptable:** el modelo logró converger de forma muy rápida, aunque con un MAPE del 24,79 %.
  - Otros optimizadores, como gradientes conjugados y la regularización bayesiana, tampoco reportaron mejoras sobre Levenberg-Marquardt.

En conclusión, se logró minimizar el error para el mes de enero hasta llegar a un error diario del 16,78 %. El mejor modelo se encontró al modificar la función de activación a *tanh* y aumentar el tamaño de las ventanas temporales a 8. Los hiperparámetros restantes se mantuvieron iguales al caso base.

5.3.4.1. Agregado de nuevas variables

Las nuevas variables fueron añadidas en base al análisis de correlaciones realizado para el año 2019 en la sección 5.1. Si no se menciona lo contrario, los cambios realizados en los tamaños de las ventanas temporales fueron para todas las variables en simultáneo. A continuación, se detallan los resultados obtenidos para cada una:

- **Añadido de humedad:**

  - Agregar la variable al mejor modelo anterior no reportó mejoras.
  - El uso de ventanas con tamaño mayor a 13 generó resultados similares, que rondaron un valor de MAPE del 17,5 %.
  - Posteriormente, disminuir el tamaño de las ventanas temporales a 4 logró reducir el MAPE a un 16,5 %.

- **Añadido de temperatura del aire:** añadir la variable al modelo anterior produjo un MAPE del 17,62 %. Los cambios realizados en la configuración de la red no resultaron en mejoras.

- **Añadido de declinación:**

  - Al añadirla al modelo anterior se obtuvo un MAPE del 15,67 %.
  - Aumentar el tamaño de ventanas temporales a 5 llevó a un valor del 15,47 %.
  - Los cambios en la cantidad de neuronas no mejoraron los resultados.
  - Las modificaciones únicamente en la ventana temporal de la declinación no representaron mejoras.

- **Remoción de la temperatura del aire:** generó un MAPE del 18,16 %. Los cambios sobre los hiperparámetros no implicaron ninguna mejora, por lo que se continuó con el uso de la variable.

- **Añadido de velocidad del viento:** se obtuvo un MAPE del 19,6 %. Los cambios sobre los hiperparámetros no generaron ninguna mejora, por lo que se procedió a quitarla del conjunto de variables de entrada.

- **Añadido de *datenum*:**

  - Llevó a una clara desmejora de los resultados. El MAPE obtenido fue del 28,98 %.

- Ninguna modificación sobre los hiperparámetros implicó una mejora.
- Una reducción en las ventanas temporales disminuyó levemente el nivel de error, pero no superó el mejor modelo obtenido hasta el momento.
- Se concluyó que no es beneficioso el uso de *datenum* como variable y se determinó el uso de la declinación en su lugar.

Finalmente, se obtuvo un MAPE del 15,47 % como mejor resultado para un modelo con ventanas temporales de tamaño 5 y 8 neuronas en su capa oculta. A modo de conclusión sobre el conjunto de pruebas desarrollado, las variables a contemplar para las predicciones fueron la irradiancia, temperatura del aire, humedad y declinación. Como se mencionó anteriormente, las inclusiones de la velocidad del viento y *datenum* no fueron beneficiosas para el modelo.

5.3.5. Entrenamiento con búsqueda por grilla

Finalmente, se decidió realizar un ajuste de hiperparámetros en base al método de búsqueda por grilla. Como se mencionó anteriormente, la técnica consiste en la iteración sucesiva de ejecuciones, donde se comparan los resultados obtenidos con distintas combinaciones de hiperparámetros. El objetivo de la etapa fue realizar un *tuning* de la estructura de la red de forma más precisa. Como base, se utilizó el mejor modelo obtenido en la etapa anterior.

El primer conjunto de pruebas realizado se basó en la modificación del tamaño de la ventana temporal para cada variable de entrada en el rango $[1, 30]$. A su vez, se estableció entre 1 y 20 la cantidad de neuronas, dado que valores más altos no habían reportado mejoras anteriormente. A continuación, se detalla los resultados obtenidos para cada una de ellas:

- **Irradiancia:** las mejores métricas se obtuvieron para tamaños de 5 y 27 (ver figura 5.13). Adicionalmente, al incrementar el tamaño de la ventana temporal, no se observó un claro aumento del tiempo de entrenamiento (ver figura 5.14).
- **Humedad:** el mejor resultado fue obtenido con una ventana de tamaño 14, lo que reportó un MAPE del 14,81 %. A diferencia del caso anterior, sí se apreció un aumento del tiempo de entrenamiento al aumentar el tamaño de la ventana.
- **Temperatura del aire, declinación y potencia:** el MAPE mínimo fue reportado al mantener el tamaño de ventana de 5.
- **Cantidad de neuronas:** el mejor resultado se obtuvo al mantener su valor en 8. Al igual que con la ventana de humedad, el crecimiento de la cantidad de neuronas se asoció a tiempos mayores de entrenamiento (ver figura 5.15).
- **Velocidad del viento:** el mejor MAPE fue obtenido con una ventana de tamaño 6. Sin embargo, no superó los resultados obtenidos con el modelo que no incluía la variable.

A modo de conclusión, el modelo con el cual se obtuvieron los mejores resultados fue una red con una capa oculta de 8 neuronas. Los valores para las ventanas temporales fueron de tamaño 27 para la irradiancia, 14 para la humedad y 5 para la temperatura del aire, declinación y potencia. En la figura 5.16 se puede observar la estructura del modelo obtenido.

5.3.6. Conclusión sobre NARX

Inicialmente, se realizaron pruebas introductorias para determinar que el entorno adecuado para trabajar con una red NARX era MATLAB. Posteriormente, se experimentó para encontrar modelos que minimicen el error al predecir un día y un mes. Por último, se realizaron los ajustes finales de hiperparámetros a través de una búsqueda por grilla.

El MAPE mínimo obtenido para el mes de enero de 2020 fue 14,81 %. El modelo utilizó únicamente 8 neuronas en su capa oculta. Los valores para las ventanas temporales de cada variable fueron de 27 para la irradiancia, 14 para la humedad y 5 para la temperatura del aire, declinación y potencia. El optimizador utilizado fue Levenberg-Marquardt, mientras que la función de activación escogida fue *tanh*.

De las variables disponibles en la base de datos, las únicas no incluidas como entradas del modelo fueron la velocidad del viento y *datenum*. A pesar de no aportar beneficios en la predicción, esta última fue aplicada para el filtrado de datos y la generación de los conjuntos de entrenamiento, validación y *testing*. La velocidad del viento, por otra parte, no fue de utilidad durante la experimentación con los modelos NARX.

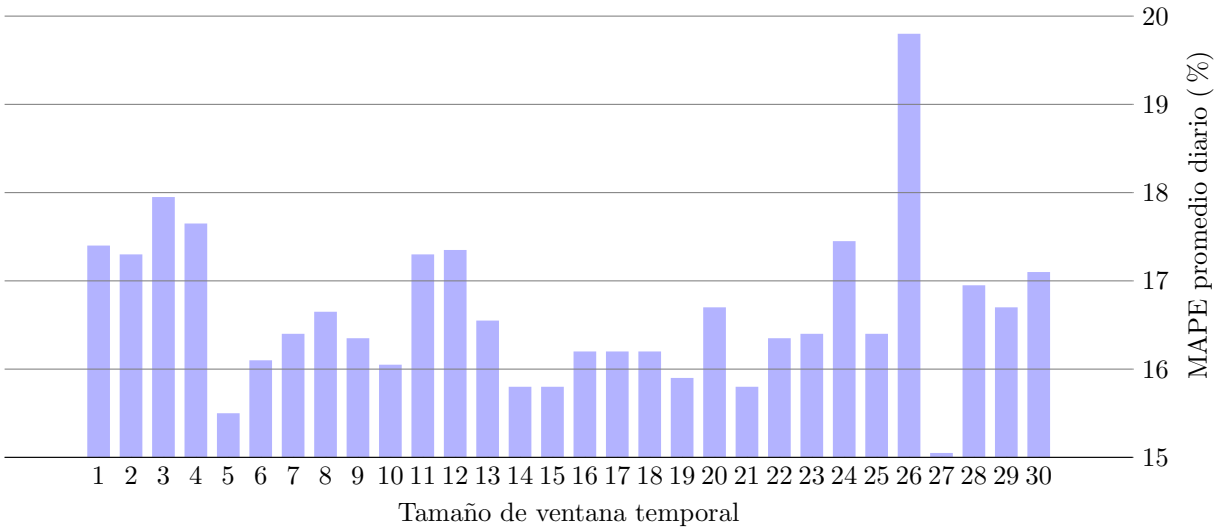


Figura 5.13: Valores de MAPE obtenidos durante la búsqueda por grilla, con variaciones en el tamaño de ventana temporal para la variable irradiancia.

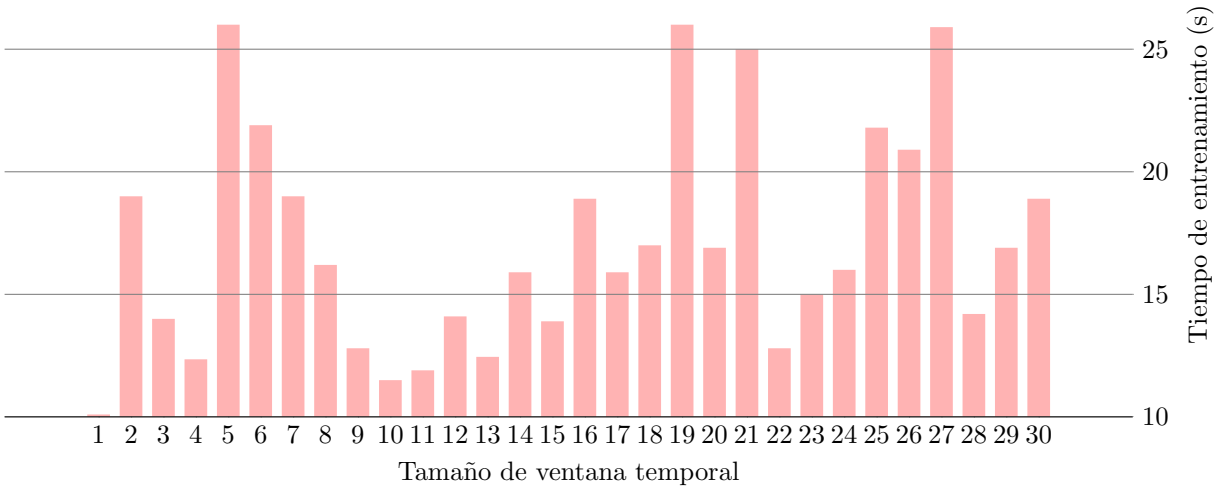


Figura 5.14: Tiempos de entrenamiento obtenidos durante la búsqueda por grilla, con variaciones en el tamaño de ventana temporal para la variable irradiancia.

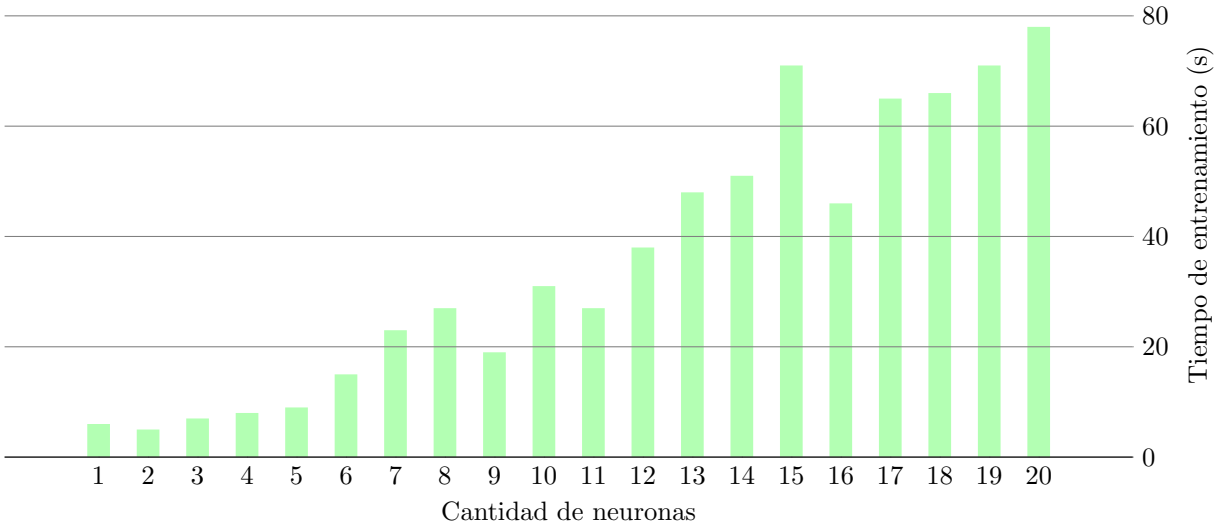


Figura 5.15: Tiempos de entrenamiento obtenidos durante la búsqueda por grilla, con variaciones en la cantidad de neuronas en la capa oculta.

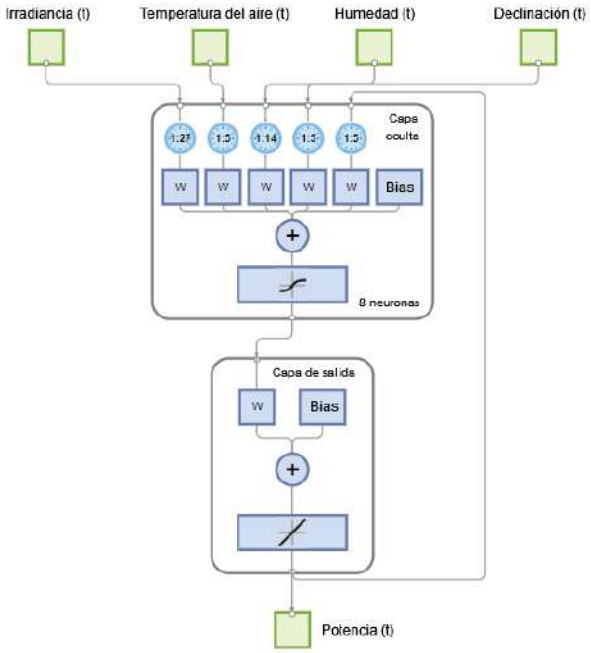


Figura 5.16: Arquitectura del modelo NARX final.

5.4. LSTM y GRU

5.4.1. Implementación

Al igual que para el perceptrón multicapa, se utilizó Keras para la construcción de modelos. La librería incluye la implementación básica de cada arquitectura, junto con la posibilidad de realizar un apilado o *stacking* de capas recurrentes. A su vez, es posible utilizar capas bidireccionales.

Adicionalmente, Keras ofrece una implementación de la técnica de *dropout* recurrente de Semeniuta (ver sección 4.4.3), junto con la posibilidad de realizar ajustes de hiperparámetros automatizados en base al algoritmo *Hyperband* (ver sección 4.4.5.1).

5.4.1.1. Generación de secuencias

Debido a los resultados positivos obtenidos con las redes NARX, donde se incluyó el uso de ventanas temporales, se decidió realizar una implementación similar con Keras. Con el uso de la librería, fue posible implementar un preprocesamiento de los datos a partir de un mecanismo de ventana deslizante. Cada ventana (también denominada secuencia) es un arreglo bidimensional, donde una dimensión corresponde al tipo de medición (irradiancia, temperatura del aire, etc.) y la otra al instante temporal en el cual han sido realizadas [99]. Cada secuencia es asociada a un objetivo o *target* que representa el valor que se busca predecir: la potencia eléctrica.

Además de incluir ventanas temporales para las variables climáticas, las redes NARX también permiten retroalimentar las predicciones generadas y utilizarlas como entradas al modelo. Para implementar esta funcionalidad con Keras, era necesario incluir una nueva variable para cada instante temporal de la secuencia, correspondiente a la potencia eléctrica en dicho momento. A la hora de utilizar el modelo en una situación real, las variables climáticas del día a predecir pueden ser obtenidas mediante librerías de NWP. Sin embargo, al incluir a la potencia en la secuencia de entrada, sus valores deben ser generados a partir de retroalimentaciones con el modelo.

La dependencia sobre un mecanismo de retroalimentación era una limitante sobre los instantes temporales que podían incluirse en la secuencia de entrada. Dado que para el instante a predecir se desconoce la potencia eléctrica, no posible incorporar su información temporal completa. Por este motivo, originalmente se definió que las mediciones climáticas incluidas en la secuencia sean las N anteriores al instante de predicción. Por ejemplo, para obtener la potencia eléctrica a las 15:00 h con $N = 3$, la secuencia estaría formada por las mediciones efectuadas a las 14:55, 14:50 y 14:45 h.

Inicialmente, por diferentes limitaciones existentes en Keras, se dificultó el desarrollo del mecanismo de retroalimentación. Por esta razón, las primeras experimentaciones con modelos LSTM aplicaron la configuración de secuencia anteriormente mencionada, pero sin incluir a la potencia eléctrica como variable de entrada. El razonamiento fue dejar abierta la posibilidad de una inclusión futura de la retroalimentación con el modelo.

A pesar de múltiples intentos, no se logró concretar el mecanismo y se desistió con la idea de incorporar a la potencia como variable de entrada. A partir de entonces, se realizó un ajuste a la generación de secuencias, de manera que se incluyeran las variables climáticas asociadas al propio instante que se deseaba predecir (ver sección 5.4.2.6). De esta manera, para el ejemplo de la predicción de la potencia a las 15:00 h mencionado anteriormente, también se incluirían las mediciones climáticas de ese momento.

Parámetros de configuración

La generación de secuencias en Keras puede configurarse a partir de las siguientes variables [99]:

- **Tasa de muestreo (*sampling rate*):** período entre las mediciones incluidas en la secuencia. Se utiliza cuando los instantes de tiempo entre cada medición son muy cortos, de manera que los valores resultan muy similares entre sí. Para la base de datos utilizada durante el proyecto, las mediciones fueron realizadas en intervalos de 5 minutos. Por ende, un valor de 12 incluiría mediciones espaciadas en lapsos de una hora. Durante todas las pruebas realizadas el parámetro, fue establecido en 1, por lo que las secuencias mantuvieron el intervalo presente en la base de datos.
- **Longitud de secuencia (*sequence length*):** cantidad de instantes temporales a incluir en la ventana. Fue uno de los hiperparámetros de mayor interés durante la experimentación.
- ***Shuffle*:** valor booleano, establece si las secuencias deben ser generadas en orden o aleatoriamente. No alteran la disposición de las mediciones que componen la secuencia, sino el orden en el cual

se retorna cada secuencia independiente. Al tratar con un problema de series temporales, fue establecido en *false*.

- **Tamaño de *batch*:** cantidad de secuencias que se introducen al modelo en un *batch* de datos. Es utilizado para establecer la duración de la memoria de las capas LSTM y GRU, dado que por defecto su estado es reiniciado al finalizar cada *batch* [100] [101]. Al igual que la longitud de secuencia, fue estudiado con profundidad.

5.4.2. Predicción del mes de enero de 2020

Una vez demostrada la importancia de la inclusión del año 2019 completo para realizar entrenamientos en las pruebas con modelos NARX, se decidió hacer uso de todos sus registros desde el comienzo de la experimentación. Debido a la experiencia adquirida, la manipulación de altos volúmenes de datos se realizó sin inconvenientes.

El objetivo de la primera etapa de pruebas con las redes LSTM y GRU fue la determinación de las variables de entrada a incluir en el modelo. A su vez, se buscó comprender la reacción de las redes ante las variaciones en sus arquitecturas.

5.4.2.1. LSTM de capa única

Para comenzar la experimentación, se definió un modelo LSTM base con las siguientes características:

- **Arquitectura:** capa oculta LSTM con 150 unidades.
- **Variable de entrada:** irradiancia.
- **Optimizador:** Adam.
- **Longitud de secuencia:** se estableció una longitud de 120, es decir 10 horas.
- ***Batch size*:** se utilizó un tamaño de 128.
- **Función de pérdida:** MSE.

El período de entrenamiento utilizado fue desde enero a octubre, mientras que el conjunto de validación estuvo compuesto por los meses de noviembre y diciembre. A pesar de ser el primer modelo analizado y contar con una única variable de entrada, se obtuvo un MAPE promedio diario del 17,81 % al predecir el mes de enero de 2020, un valor cercano al mejor resultado obtenido para las redes NARX.

Posteriormente, con el objetivo de equilibrar la cantidad de épocas y el tiempo de entrenamiento de cada modelo, se implementó un mecanismo de *early stopping* basado en la pérdida del conjunto de validación. El máximo de épocas fue establecido en 50 y la paciencia en 5. A su vez, en caso de que el entrenamiento se detuviera de forma temprana, los pesos sinápticos de la red eran restablecidos a la época donde se obtuvo el menor error de validación.

Para continuar con el análisis, se realizaron las siguientes modificaciones a los hiperparámetros y la arquitectura de la red:

- **Cantidad de unidades:** dado que el modelo base ya contaba con un valor elevado en comparación a los utilizados en la etapa de experimentación con modelos NARX, se procedió a realizar pruebas sobre una posible reducción. Se observó que las métricas de error del modelo no aumentaron considerablemente al reducir su cantidad. Al mismo tiempo, la mejoría en los tiempos de entrenamiento fue notoria. El valor escogido para continuar la experimentación fue de 10, el cual reportó un MAPE promedio diario del 18,14 %. Para una cantidad de unidades menor, las métricas de la red empeoraron notoriamente.
- **Longitud de secuencia:** al igual que para la cantidad de unidades, se investigó el comportamiento de la red ante una secuencia reducida. En términos de *performance*, el mejor valor obtenido fue para un tamaño de 20 (17,84 %), el cual representa un lapso temporal de 100 minutos. A medida que se aumenta el tamaño de la secuencia, los valores de irradiancia reportan una menor correlación con la potencia eléctrica del instante actual, por lo que su inclusión en el modelo fue innecesaria. A su vez, una secuencia reducida mejoró los tiempos de entrenamiento de la red, dado que la cantidad de pesos sinápticos requeridos se redujo.

- **Tamaño de *batch*:** valores altos reportaron un menor tiempo de entrenamiento. Sin embargo, tamaños menores mejoraron la *performance* del modelo. La razón pudo haber estado relacionada al lapso temporal en el cual la memoria del modelo era restablecida. Es posible que una memoria longeva incluya información poco relevante para la predicción actual. El mejor resultado fue del 17,17 % y fue obtenido con un *batch* de tamaño 32, lo que representa una memoria de 160 minutos.
- ***Dropout*:**
  - Después de la capa de entrada: su inclusión llevó a una clara reducción de la *performance*. Ante cada aumento en la probabilidad de eliminación de una unidad, el error empeoró. Para un valor de 0,1, se reportó un MAPE de 19,71 %. Al establecerlo en 0,5, el error ascendió a 44,74 %. Debido a la clara degradación de las métricas, la inclusión *dropout* antes de la capa recurrente fue descartada para pruebas posteriores.
  - Después de la capa LSTM: el mejor resultado fue obtenido con una probabilidad de 0,1, donde se logró un MAPE del 18,91 %. Si bien no logró mejorar las métricas obtenidas previamente, se decidió que las futuras experimentaciones con *dropout* sean luego de la capa recurrente.
  - *Dropout* recurrente: su inclusión con probabilidades bajas reportó resultados similares a los obtenidos con anterioridad. Con un valor de 0,2 el mejor MAPE obtenido fue de 17,67 %.
  - La combinación de los tipos de *dropout* tampoco logró reducir las métricas de error.
  - Debido a los resultados obtenidos, el análisis prosiguió sin la inclusión de ningún tipo de *dropout*.
- **Optimizadores:**
  - Adam: hasta el momento había sido utilizado con los parámetros por defecto (especificados en la sección 4.4.4.2). Aumentar o disminuir la tasa de aprendizaje no presentó mejoras.
  - Descenso por gradiente: ninguna variante logró mejorar los resultados obtenidos anteriormente. Los ajustes se realizaron sobre la tasa de aprendizaje y se incluyó la técnica de *momentum*.
  - RMSprop: reportó resultados muy similares a los obtenidos con Adam.

Como conclusión de la primera etapa de experimentación, se logró un MAPE promedio diario del 17,17 % para el mes de enero de 2020. Como se observa en la figura 5.17, el modelo final de esta etapa se compuso de una única capa oculta LSTM de 10 unidades, no contó con *dropout* y su longitud de secuencia fue establecida en 20. El optimizador definido fue Adam, con los parámetros por defecto. Por otra parte, el *batch size* se estableció en 32.

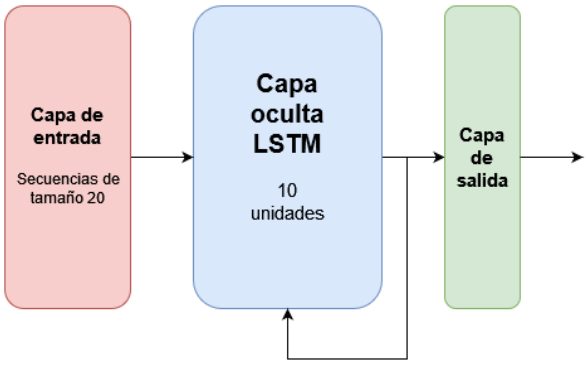


Figura 5.17: Arquitectura del modelo LSTM de capa única obtenido en la primera etapa de experimentación.

5.4.2.2. *Stacked* LSTM

La siguiente etapa del análisis fue la incorporación de una capa recurrente adicional. Como base, se partió del modelo obtenido en las pruebas anteriores. Se le añadió una nueva capa LSTM, de manera que el modelo incluyera dos capas ocultas, ambas de 10 unidades. La nueva configuración obtuvo como resultado un MAPE del 17,46 %, cercano al obtenido con una única capa. Posteriormente, se procedió a evaluar nuevos ajustes de hiperparámetros:

- **Cantidad de unidades:** se incrementó el valor a 20 y posteriormente a 50. Con esta última modificación se logró una mejora leve en los resultados. Ninguna modificación adicional logró mejoras en el modelo.
- ***Dropout***:
  - Debido a que la inclusión de múltiples capas aumenta la capacidad de la red, existía la posibilidad de que el modelo presentase *overfitting*.
  - Para contrarrestarlo, se decidió experimentar nuevamente con la inclusión de *dropout*. Si bien el *dropout* recurrente no reportó mejoras, la inclusión de *dropout* estándar luego de las capas ocultas logró mejorar notoriamente los resultados. El mejor MAPE obtenido fue del 15,94 % para una probabilidad de 0,2. Otros valores no reportaron mejoras.
  - Con la inclusión de *dropout*, se experimentó nuevamente sobre las variaciones en la cantidad de unidades y la longitud de secuencia. Los aumentos de unidades no lograron reducir las métricas de error. Sin embargo, una secuencia de tamaño 40 resultó beneficiosa para el modelo. El valor de MAPE obtenido fue del 15,90 %, lo que mejoró ligeramente el resultado anterior.
- **Función de activación:** se experimentó con el uso de ReLU, lo cual no logró mejorar los resultados previamente obtenidos.

En conclusión, añadir una capa LSTM adicional logró mejorar los resultados obtenidos previamente. Fue determinante la inclusión de *dropout* con probabilidad 0,2, el cual no había sido de utilidad en la fase anterior. Finalmente, el modelo *stacked* LSTM que minimizó el valor de MAPE para esta etapa contó con capas de 50 unidades y una longitud de secuencia de 40.

5.4.2.3. LSTM bidireccional

Posteriormente, se experimentó sobre el reemplazo de las capas LSTM estándar por capas bidireccionales, de manera que las secuencias introducidas se analicen en ambos órdenes cronológicos.

Las implementaciones Bi-LSTM de capa única no lograron superar los resultados expuestos anteriormente. Las pruebas incluyeron el añadido de *dropout*, junto con variaciones en la longitud de secuencia y los optimizadores. El modelo LSTM bidireccional que logró minimizar el valor de MAPE a un 16,07 % para esta etapa se compuso de una capa de 50 unidades, *dropout* de 0,2 y longitud de secuencia de 50 (ver figura 5.18). A su vez, el optimizador utilizado fue nuevamente Adam. Sin embargo, al añadir una segunda capa bidireccional con características idénticas a la anterior, el modelo logró mejorar su *performance* y alcanzó un MAPE del 15,28 %.

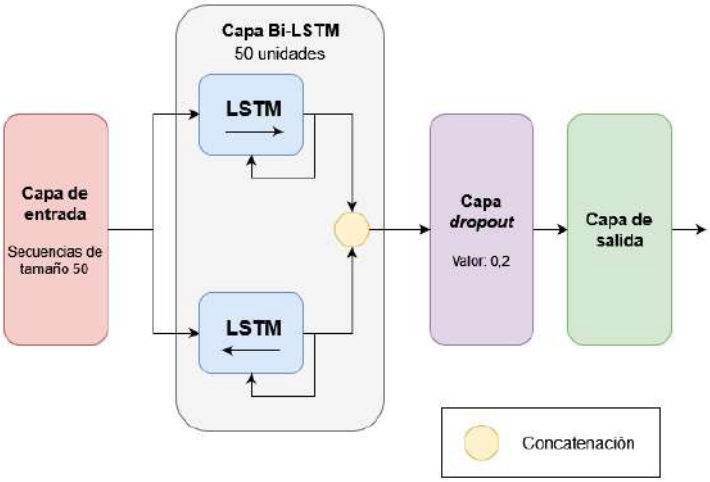


Figura 5.18: Arquitectura de un modelo Bi-LSTM de capa única.

Finalmente, se experimentó sobre la combinación de capas estándar y bidireccionales. Se reemplazó la primera capa bidireccional por una estándar, lo cual reportó un MAPE del 15,63 %. Por otro lado, utilizar primero una capa bidireccional y luego una estándar alcanzó un resultado de 15,29 %, ligeramente peor que el obtenido con dos capas bidireccionales. Ante la ausencia de mejoras y con el objetivo de reducir la carga de trabajo, se decidió descartar el estudio posterior de modelos con capas combinadas.

En la tabla 5.1, se muestra un resumen de los mejores modelos LSTM producidos hasta este punto de la experimentación.

Modelo	Arquitectura	*Dropout*	Longitud de secuencia	*Batch size*	Función de pérdida	MAPE promedio
LSTM	Capa única con 10 unidades	-	20	32	MSE	17,17 %
LSTM	2 capas de 50 unidades	0,2	40	32	MSE	15,90 %
Bi-LSTM	2 capas de 50 unidades	0,2	50	32	MSE	15,28 %

Tabla 5.1: Mejores modelos LSTM obtenidos en cada fase de experimentación con una única variable de entrada.

5.4.2.4. Inclusión de variables de entrada adicionales

Al igual que para las redes NARX, el añadido de mediciones climáticas se realizó progresivamente en base a su grado de correlación con la potencia eléctrica. Además de la inclusión de nuevas entradas, se realizaron diferentes ajustes sobre los hiperparámetros analizados en fases anteriores. Los modelos utilizados para la experimentación fueron el LSTM estándar y el Bi-LSTM, ambos en sus variantes *stacked*. A continuación, se mencionan las principales observaciones para cada variable:

- **Humedad:**
  - Luego de algunos ajustes, su incorporación reportó mejoras en ambos modelos. El *stacked* LSTM logró reducir su MAPE de 15,90 % a 15,60 %.
  - La red Bi-LSTM pasó de 15,28 % a 14,60 %.
- **Temperatura del aire:** nuevamente, ambas arquitecturas reportaron mejoras al incorporar la variable. Los mejores resultados fueron 14,51 % para la red *stacked* y 14,24 % para la Bi-LSTM.
- **Declinación:** si bien se logró reducir el MAPE de Bi-LSTM a 14,12 %, el modelo *stacked* obtuvo como mejor resultado un 14,76 %, lo cual no mejoró lo obtenido anteriormente.
- **Velocidad del viento:**
  - Incorporarla al resto de variables logró reducir el MAPE del modelo *stacked* a 14,26 %. Posteriormente, se experimentó con quitar la declinación, dado que anteriormente había perjudicado al modelo. Sin embargo, al removerla no se logró mejorar el 14,26 % reportado con la totalidad de las variables.
  - Bi-LSTM también reportó mejoras y redujo su MAPE a 13,96 %.
- ***Datenum*:** debido a su baja correlación y los resultados obtenidos previamente, no fue analizada.

Finalmente, a diferencia de la arquitectura NARX, los modelos implementados demostraron la utilidad de todas las mediciones presentes en la base de datos. El mejor modelo obtenido en esta etapa de experimentación para una red *stacked* LSTM incluyó la totalidad de las variables (a excepción de *datenum*) y logró un MAPE del 14,26 %. Su arquitectura se compuso de 2 capas de 100 unidades, una longitud de secuencia de 20 y *dropout* con probabilidad 0,5.

Por otra parte, para la variante Bi-LSTM, el modelo de mejor *performance* que se obtuvo en esta estapa reportó un MAPE del 13,96 % y también utilizó todas las variables. Sus 2 capas se compusieron de 80 unidades, con *dropout* recurrente de 0,1. La longitud de secuencia se estableció en 150, mientras que se incluyó *dropout* de probabilidad 0,3 luego de las capas recurrentes.

En la tabla 5.2, puede observarse una comparativa entre los modelos producidos en esta fase y aquellos que utilizaron una única variable de entrada.

5.4.2.5. *Stateful* LSTM

Como se mencionó anteriormente, los modelos implementados restablecen su memoria al finalizar un *batch* de datos. La librería Keras ofrece la posibilidad de evitar este comportamiento, de manera que el estado oculto y el estado de la celda de la capa LSTM se mantengan intactos durante toda la ejecución. A este tipo de modelos se los denomina *stateful*.

Modelo	Arquitectura	Variables de entrada	*Dropout*	*Dropout* recurrente	Longitud de secuencia	*Batch size*	Función de pérdida	MAPE promedio
LSTM	2 capas de 50 unidades	Irradiancia	0,2	-	40	32	MSE	15,90 %
LSTM	2 capas de 100 unidades	Todas (excepto *datenum*)	0,5	-	20	32	MSE	14,26 %
Bi-LSTM	2 capas de 50 unidades	Irradiancia	0,2	-	50	32	MSE	15,28 %
Bi-LSTM	2 capas de 80 unidades	Todas (excepto *datenum*)	0,3	0,1	150	32	MSE	13,96 %

Tabla 5.2: Comparativa entre modelos LSTM con la inclusión de nuevas variables de entrada.

A pesar de sus potenciales ventajas, las implementaciones de los modelos *stateful* poseen limitaciones. La principal es que demandan el establecimiento de un tamaño de *batch* fijo. Generalmente, la cantidad de ejemplos de cada conjunto no es un valor múltiplo del *batch size*. Por este motivo, el último *batch* suele incluir una cantidad menor al tamaño establecido, lo cual no está permitido para modelos *stateful*.

Para llevar a cabo las pruebas, fue necesario recortar ejemplos de los conjuntos de entrenamiento y *testing*, de manera que la cantidad total de datos sea múltiplo del *batch size* utilizado (32).

Se analizó la alternativa de utilizar tamaños pequeños, de manera que se reduzca la cantidad de ejemplos a recortar. Sin embargo, reducir el tamaño del *batch* aumentó considerablemente el tiempo de entrenamiento del modelo. Por este motivo, se descartó dicha posibilidad.

Si bien las limitaciones perjudicaban la viabilidad práctica del modelo, las pruebas efectuadas lograron reducir el MAPE de la arquitectura *stacked*. Se obtuvo un valor del 13,81 %, en comparación al 14,26 % reportado anteriormente.

5.4.2.6. Ajuste de secuencia

Tal como se explicó en la sección 5.4.1.1, las secuencias introducidas al modelo no contaban con el instante temporal actual. Durante el transcurso de las pruebas anteriores, se investigó sobre la posibilidad de incluir una retroalimentación de la potencia eléctrica como una variable adicional de la secuencia, de forma similar a las redes NARX cerradas. Sin embargo, a pesar de los esfuerzos detrás de la investigación, no se logró encontrar una implementación que se ajuste a las características especificadas.

Una vez eliminada la posibilidad de incluir a la potencia eléctrica como variable de entrada del modelo, se realizó un ajuste a las secuencias de entrada introducidas, de manera que se incluyan las mediciones climáticas del instante actual. La nueva inclusión logró una gran mejora en los resultados obtenidos hasta el momento. A partir de nuevos ajustes de hiperparámetros, se lograron los siguientes resultados:

- ***Stacked* LSTM:** el MAPE se redujo de 14,26 % a 11,35 % con un modelo que contó con 2 capas con 50 unidades de procesamiento, *dropout* con valor 0,5 y longitud de secuencia de 40.
- ***Stateful* LSTM:** se obtuvo un MAPE del 11,27 %, en comparación al 13,81 % previo. Al igual que para el modelo anterior, se utilizaron dos capas con 50 unidades y 40 de longitud de secuencia. El modelo no contó con *dropout*.
- **Bi-LSTM:** se logró reducir el MAPE de 13,96 % a 11,42 %. El modelo utilizó 2 capas bidireccionales de 100 unidades, longitud de secuencia de 80, *dropout* con valor 0,1 y *dropout* recurrente de 0,1.

En conclusión, el ajuste de la secuencia fue beneficioso para la *performance* de los modelos. En la tabla 5.3 puede observarse la comparativa entre los modelos anteriores y los producidos en esta etapa.

Al finalizar la primera fase de experimentación con modelos LSTM, las arquitecturas *stacked* y bidireccional demostraron gran potencial para la resolución del problema, dado que lograron superar los resultados obtenidos con las redes NARX (14,81 %). Por este motivo, se definió que ambas variantes fueran foco de pruebas futuras. Debido a la similitud en los resultados obtenidos y las restrictivas limitaciones detrás de su implementación, se decidió descartar el modelo *stateful* para las etapas siguientes.

Modelo	Arquitectura	*Dropout*	*Dropout* recurrente	Longitud de secuencia	*Batch size*	Función de pérdida	Ajuste de secuencia	MAPE promedio
LSTM	2 capas de 100 unidades	0,5	-	20	32	MSE	No	14,26 %
LSTM	2 capas de 50 unidades	0,5	-	40	32	MSE	Sí	11,35 %
Stateful LSTM	2 capas de 100 unidades	0,5	-	20	32	MSE	No	13,81 %
Stateful LSTM	2 capas de 50 unidades	-	-	40	32	MSE	Sí	11,27 %
Bi-LSTM	2 capas de 80 unidades	0,3	0,1	150	32	MSE	No	13,96 %
Bi-LSTM	2 capas de 100 unidades	0,1	0,1	80	32	MSE	Sí	11,42 %

Tabla 5.3: Comparativa entre los mejores modelos LSTM obtenidos antes y después del ajuste de secuencia.

5.4.2.7. GRU de capa única

Una vez concluidas las experimentaciones con LSTM, se decidió proseguir con su variante más popular: GRU. Las pruebas anteriores llevaron a la conclusión de que la inclusión de todas las variables climáticas era beneficiosa. Por ende, los modelos analizados incluyeron todas las entradas posibles, a excepción de la variable *datenum*.

Se decidió comenzar el proceso de experimentación con un modelo GRU de baja capacidad, con el objetivo de incrementarla progresivamente. El modelo base contó con las siguientes especificaciones:

- **Arquitectura:** una única capa oculta GRU, de 20 unidades.
- **Optimizador:** Adam, con parámetros por defecto.
- **Tamaño de *batch*:** 32.
- **Longitud de secuencia:** 20.
- ***Dropout***: no se utilizó.

A partir del modelo definido, se logró obtener un MAPE del 13,79 %. Posteriormente, se incursionó en la experimentación sobre los hiperparámetros y la arquitectura de la red. El proceso fue similar al utilizado durante las pruebas sobre los modelos LSTM:

- **Cantidad de unidades:** se realizaron ejecuciones con 20, 50, 80, 100 y 120. El mejor resultado fue obtenido con 50 unidades, donde el MAPE se redujo a 11,16 %.
- ***Dropout***: las pruebas se realizaron con valores de 0,2 y 0,5, dado que anteriormente habían reportado la mejor *performance*. Sin embargo, a pesar de variar en conjunto la cantidad de unidades, su inclusión no logró mejorar los resultados obtenidos anteriormente.
- ***Dropout* recurrente**: al igual que en su versión estándar, se experimentó sobre los valores 0,2 y 0,5. Nuevamente, las métricas obtenidas no reportaron mejoras.
- **Longitud de secuencia:** su modificación fue analizada al mismo tiempo que las pruebas anteriores. Los valores utilizados fueron 20, 40, 60, 80 y 100.

En definitiva, el mejor modelo GRU de capa única obtuvo un MAPE del 11,16 % y consistió de 50 unidades y longitud de secuencia de 40. La inclusión de *dropout* y su variante recurrente no fueron de ayuda para el modelo.

5.4.2.8. *Stacked* GRU

Una vez realizadas las pruebas sobre una única capa, se procedió a experimentar sobre un modelo de dos capas GRU. Nuevamente, la experimentación se basó en la modificación de la cantidad de unidades, el tamaño de la secuencia y el uso de *dropout*.

A pesar de las modificaciones realizadas, ningún modelo *stacked* logró superar los resultados obtenidos con una única capa oculta. El mejor modelo para este tipo de arquitectura consistió de 2 capas con 50 unidades y longitud de secuencia de 20. A su vez, las capas contaron con *dropout* recurrente de valor 0,2. El MAPE promedio diario resultante para el mes de enero 2020 fue 11,62 %.

A modo de resumen, en la tabla 5.4 pueden observarse los dos mejores modelos GRU producidos durante la experimentación inicial.

Modelo	Arquitectura	*Dropout* recurrente	Longitud de secuencia	*Batch size*	Función de pérdida	MAPE promedio
GRU	Capa única con 50 unidades	-	40	32	MSE	11,16 %
GRU	2 capas de 50 unidades	0,2	20	32	MSE	11,62 %

Tabla 5.4: Mejores modelos GRU obtenidos en cada fase de experimentación con la totalidad de las variables de entrada (excepto *datenum)*.

5.4.2.9. Conclusión de la predicción del mes de enero de 2020

Al finalizar la primera etapa de experimentación, se realizó un análisis sobre los resultados obtenidos. La conclusión principal fue que los modelos LSTM y GRU lograron superar la *performance* observada en las redes NARX. Por este motivo, se decidió descartar el uso futuro de esta arquitectura y continuar con la experimentación sobre LSTM y GRU.

Adicionalmente, se observó que los modelos implementados se vieron beneficiados por la inclusión de todas las variables en la base de datos (excepto *datenum*). Esto no había sucedido en las redes NARX, donde la velocidad del viento no había sido de utilidad.

En cuanto a las arquitecturas analizadas, se observó que los resultados obtenidos en las implementaciones *stacked* y de capa única resultaron similares, por lo que se decidió continuar experimentando sobre ambos tipos de modelos. A su vez, los llamativos resultados obtenidos con las arquitecturas bidireccionales obligaron a incorporarlas a los análisis futuros, lo cual no estaba previsto anteriormente.

5.4.3. Implementación del *tuning* de hiperparámetros

Ante los buenos resultados obtenidos en la experimentación previa, se decidió implementar un ajuste automatizado de los hiperparámetros. Anteriormente, las modificaciones realizadas habían sido manuales. Con el *tuning* automatizado, el objetivo fue experimentar de forma más eficiente sobre las múltiples combinaciones de valores posibles.

5.4.3.1. Keras Tuner

Keras Tuner [102] es una librería de Python utilizada para realizar ajustes de hiperparámetros automatizados sobre modelos predictivos. Para realizar el *tuning*, es necesario construir un hipermodelo. El hipermodelo funciona como una plantilla de un modelo real. Incluye definiciones básicas sobre su arquitectura (como la cantidad de capas), junto con los rangos posibles que pueden tomar los hiperparámetros durante el ajuste (tipo de optimizador, cantidad de unidades por capa, valor de *dropout*, entre otros).

Para realizar la búsqueda del conjunto de hiperparámetros óptimo, se utiliza un algoritmo de búsqueda o *tuner*, que es encargado de establecer una estrategia para la evaluación de alternativas. Keras Tuner ofrece diferentes tipos de *tuners*, entre los cuales se encuentran *RandomSearch*, *Hyperband* y *BayesianOptimization*.

Para el proyecto, se definió utilizar el algoritmo *Hyperband* (ver sección 4.4.5.1), dado que se lo considera más eficiente que el resto de *tuners* disponibles [103].

5.4.3.2. Entornos de ejecución y aceleradores

Hasta el momento, las implementaciones y ejecuciones habían sido realizadas en entornos locales. Si bien la mayoría de modelos tenían tiempos de ejecución en el orden de los 15 minutos, ciertas arquitecturas demandaban hasta una hora de entrenamiento.

Los tiempos de ejecución varían en base a la cantidad de pesos sinápticos de la red, los cuales dependen de la arquitectura escogida. Los modelos bidireccionales, por ejemplo, duplican la cantidad de pesos requeridos. A su vez, un aumento en la cantidad de unidades y el tamaño de la secuencia lleva a una mayor demanda de parámetros. Debido a que se deseaba experimentar sobre modelos que incluyeran hasta 3 o 4 capas ocultas, la alta duración del entrenamiento podía convertirse en una limitación del proceso de experimentación. A su vez, el problema iba a empeorar a la hora de realizar los ajustes de hiperparámetros, dado que la cantidad de ejecuciones requerida era alta.

Por estos motivos, se decidió investigar acerca de los entornos en la nube disponibles para llevar a cabo el proceso. El análisis se centró en aquellos considerados de mayor relevancia: Google Colab [104] y Kaggle [105].

Características de los entornos

Ambos entornos son un software como servicio que permiten la escritura y ejecución de código Python. Ofrecen una estructura denominada como *notebook*, donde el código se estructura en celdas.

Su principal ventaja es la posibilidad de realizar ejecuciones con aceleradores de hardware, como las GPU y TPU. La utilización de aceleradores es muy común en el ámbito de las redes neuronales, ya que permiten la optimización de las operaciones matriciales voluminosas [106]. En sus versiones gratuitas, la disponibilidad de los aceleradores varía en base a la demanda. A su vez, existen límites para la utilización de los recursos.

En el caso de Colab, las características de sus aceleradores no se encuentran publicadas. A su vez, sus límites de uso tampoco están definidos, ya que fluctúan a lo largo del tiempo [104]. La escritura de datos al utilizar TPUs está limitada a entornos en la nube. Por ende, fue necesario crear un *bucket* en Google Cloud para almacenar los registros creados durante el ajuste de hiperparámetros. Google Cloud ofrece una prueba gratuita de 300 U$S, lo que permitió cubrir los gastos generados por el *bucket*.

A diferencia de Colab, Kaggle sí ofrece documentación acerca de sus aceleradores, de manera que es posible saber con certeza sus especificaciones. Además, los límites de uso se encuentran publicados. Al momento del desarrollo del proyecto, el límite de uso era de 20 horas semanales para cada acelerador [107].

Comparativa de aceleradores

Según el trabajo *"Benchmarking TPU, GPU, and CPU Platforms for Deep Learning"* [108] publicado en 2019, las TPU pueden ofrecer un rendimiento hasta 20 veces mejor que las GPU. Sin embargo, los resultados varían en base al tamaño de la red, el conjunto de datos y el *batch size* utilizado.

Adicionalmente, TensorFlow ofrece una versión específica de los modelos recurrentes para las GPU NVIDIA con *kernels* cuDNN [109]. Si bien la implementación posee la ventaja de maximizar el rendimiento de la red, también impone algunas limitaciones sobre su arquitectura. Por ejemplo, no se permite el uso de *dropout* recurrente [100].

Para decidir el acelerador a utilizar durante la etapa de *tuning*, se decidió experimentar sobre ambos entornos y sus opciones ofrecidas. La comparación se realizó a partir de la ejecución de un ajuste de hiperparámetros automatizado, cuyas características se asemejaron a los *tunings* planificados para las etapas siguientes.

Modelo base y espacio de búsqueda

El modelo utilizado como base contó con las siguientes especificaciones:

- **Arquitectura:** para profundizar sobre redes de mayor capacidad, se decidió utilizar 3 capas ocultas LSTM. Con el objetivo de que la comparación sea justa con los modelos *stacked* analizados anteriormente, se definió que la cantidad de unidades en cada capa sea la misma.
- **Función de pérdida:** MSE.
- ***Dropout*:** posterior a la capa recurrente.
- **Longitud de secuencia:** 40.
- ***Early stopping*:** implementado sobre el error de validación. Se estableció un valor de 5 para la paciencia y una reducción mínima de 0,0002 para definir una mejora en la métrica.
- **Máximo de épocas:** 40.
- ***Batch size*:** 32.

A su vez, el espacio de búsqueda consistió de 42 combinaciones de hiperparámetros:

- **Cantidad de unidades:** de 16 a 112, con paso de 16.
- ***Dropout*:** valor entre 0 y 0,4. El paso utilizado fue de 0,2.
- **Optimizadores:** se permitió la elección entre Adam y RMSprop. No se alteraron los parámetros de cada uno.

Los valores establecidos para el modelo base y el espacio de búsqueda se basaron en los resultados obtenidos para la experimentación manual realizada en la fase anterior. Con el objetivo de cumplir los requerimientos de la implementación cuDNN, no se incluyó el uso de *dropout* recurrente.

Experimentación

La primera ejecución fue realizada en un entorno local, con un procesador Intel Core i7-1165G7 de undécima generación. El ajuste de hiperparámetros demandó un total de 7 horas, 3 minutos y 11 segundos, por lo que se confirmó que la ejecución local era inviable.

Posteriormente, se experimentó sobre el uso de los aceleradores disponibles en Kaggle y Colab. Su incorporación fue un éxito, dado que logró reducir los tiempos de ejecución a una hora aproximadamente.

Para la utilización de GPU, las pruebas se dividieron en dos partes. En primer lugar, se realizó una ejecución sin realizar cambios adicionales a la configuración de Tensorflow y Keras. Posteriormente, se incorporaron diversas modificaciones sugeridas por la documentación de las librerías:

- **Esquema de precisión mixta:** por defecto, los cálculos del entrenamiento se realizan con tipos de coma flotante de 32 bits. Al activar la precisión mixta, se permite que las operaciones menos sensibles se realicen en 16 bits, de manera que el entrenamiento se ejecute más rápido y utilice menos memoria [110].
- ***Accelerated Linear Algebra* (XLA):** es un compilador específico para álgebra lineal, que permite acelerar los modelos de TensorFLow. XLA se encarga de optimizar el grafo de ejecución, de manera que sea optimizado y se mejore el rendimiento [111].
- **Estrategia *mirrored*:** permite el entrenamiento sincrónico entre múltiples GPUs de una misma máquina [112]. Únicamente fue utilizado con las GPU T4 de Kaggle, dado que existía la posibilidad de utilizar dos unidades en paralelo.

A pesar de la incorporación de las modificaciones mencionadas, no se logró mejorar los tiempos obtenidos anteriormente.

Por parte de las TPU, su uso reportó los mejores tiempos de entrenamiento en cada entorno. El mejor resultado fue obtenido en Kaggle, donde el acelerador VM v3-8 logró completar el ajuste en 50 minutos y 51 segundos. Por otra parte, la TPU de Colab alcanzó un tiempo de 1 hora, 2 minutos y 3 segundos.

La duración del proceso de ajuste con cada entorno y acelerador pueden observarse en la tabla 5.5, donde también se especifica la cantidad de iteraciones del algoritmo *Hyperband* para decidir las mejores combinaciones de hiperparámetros. A su vez, se puede observar una comparativa gráfica en la figura 5.19.

Entorno	**Acelerador**	**Especificaciones adicionales**	**Tiempo**	***Iteraciones***
Colab	Ninguno (CPU)	-	7 h 3 min	59
Colab	GPU	-	1 h 6 min	58
Colab	GPU	*Mixed precision* y XLA	1 h 4 min	55
Colab	TPU	-	1 h 2 min	59
Kaggle	GPU T4 X2	-	59 min	54
Kaggle	GPU T4 X2	*Mixed precision*, XLA y *mirrored*	1 h 59 min	59
Kaggle	GPU P100	-	1 h 8 min	59
Kaggle	GPU P100	*Mixed precision* y XLA	1 h 11 min	53
Kaggle	TPU VM v3-8	-	50 min	60

Tabla 5.5: Comparativa de la duración de ajuste entre los entornos de ejecución y sus aceleradores.

Conclusiones sobre entornos de ejecución y aceleradores

Finalmente, se concluyó que una TPU es el mejor acelerador para las redes utilizadas en el proyecto. A pesar de que las GPU ofrecen tiempos de entrenamiento similares, su viabilidad es limitada ante las restricciones impuestas por la implementación cuDNN.

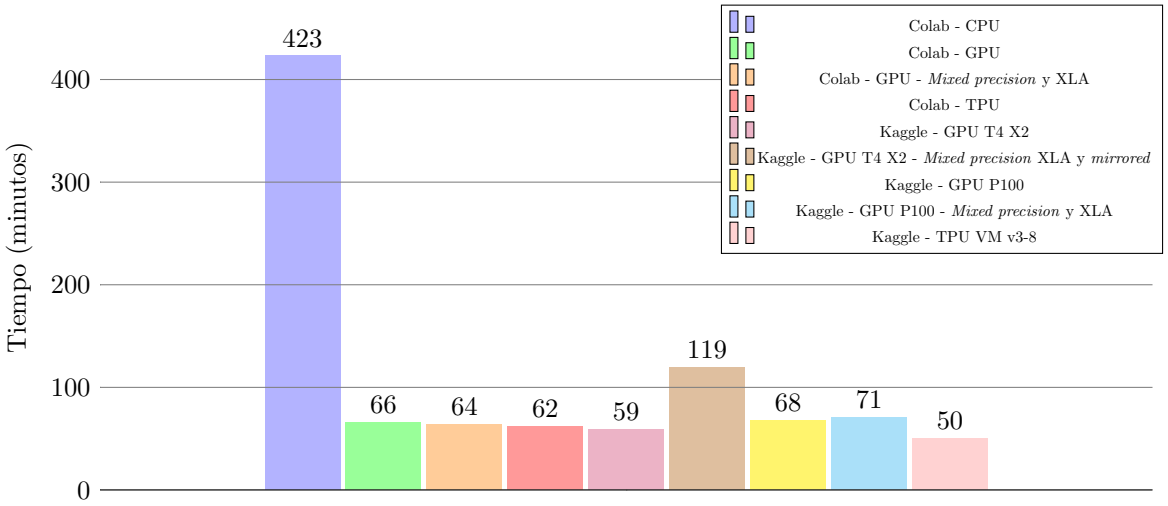


Figura 5.19: Gráfico comparativo entre la duración del ajuste y los aceleradores de cada entorno.

En cuanto a la elección del entorno, si bien Kaggle reportó el mejor rendimiento, se encontró que la disponibilidad de recursos solía ser más limitada que en Colab. Por dicha razón, los tiempos de espera al solicitarlos podían ser altos.

Debido a que los dos entornos contaban con fortalezas y debilidades, se decidió continuar utilizando ambos para el resto del proyecto. La combinación de entornos permitió ejecutar múltiples pruebas en simultáneo, además de ofrecer la posibilidad de evitar los límites de uso.

5.4.4. Delimitación de arquitecturas e hiperparámetros

Una vez realizada la implementación del ajuste, se lo utilizó para evaluar diversas modificaciones sobre los modelos LSTM y GRU. El objetivo fue establecer un conjunto delimitado de arquitecturas e hiperparámetros, de forma que se facilite el estudio al momento de incorporar una mayor cantidad de años al proceso de entrenamiento. Dentro de los aspectos analizados se encontraron el número de capas ocultas a utilizar, las funciones de pérdida, el período de validación y la incorporación de mecanismos de atención.

5.4.4.1. Proceso de evaluación de modelos

Selección de modelos

Luego de definir los entornos y el acelerador a utilizar, se estableció la metodología para evaluar los modelos producidos por el algoritmo *Hyperband*. Si bien los procesos de *tuning* son de utilidad para la determinación de la cantidad de unidades por capa, el optimizador y los valores de *dropout*, existen ciertas características que no pueden ser modificadas y evaluadas durante el ajuste. Principalmente, estas son la longitud de secuencia, el tamaño de *batch*, la función de pérdida y la cantidad de capas del modelo. Por este motivo, a la hora de estudiar modificaciones en alguna de estas propiedades, fue necesario realizar una ejecución independiente del algoritmo *Hyperband*.

En cada proceso de ajuste, *Hyperband* analiza múltiples modelos a partir de diferentes combinaciones de hiperparámetros definidas en el espacio de búsqueda. Tal como se especificó en la sección 4.4.5.1, al finalizar su ejecución, los modelos generados son ordenados en base a su *score*: el error obtenido sobre el conjunto de validación.

En base a los resultados obtenidos previamente, se observó que múltiples arquitecturas y longitudes de secuencia habían reportado resultados positivos. Por este motivo, era necesario ejecutar una alta cantidad de procesos de ajuste independientes. Para delimitar la cantidad total de modelos a analizar, se definió únicamente evaluar con el conjunto de *testing* a los 5 de mejor *score* en cada proceso. Se consideró que dicha cantidad era lo suficientemente representativa para evaluar cada combinación de función de pérdida, longitud de secuencia y cantidad de capas a estudiar, sin excederse en cuanto a la carga de trabajo establecida.

Reentrenamiento y evaluación sobre el conjunto de *testing*

Por recomendación de Keras, los modelos seleccionados deben ser entrenados nuevamente a partir de una configuración de *early stopping* más permisiva [102]. A la hora de realizar los procesos de *tuning*, la estrategia suele ser agresiva, de forma que se descarten rápidamente los modelos de baja *performance*. Sin embargo, una vez realizada la selección, es recomendable optimizar al máximo cada modelo para evitar el *underfitting* [113].

Adicionalmente, al tratarse de un problema de series temporales, es recomendable incorporar el conjunto de validación al conjunto de entrenamiento a utilizar. La razón es que el conjunto de validación se compone de instantes temporales más cercanos a aquellos que se desean predecir, por lo que se considera relevante que el modelo acceda a dichos datos durante la fase de entrenamiento. Dado que los hiperparámetros de cada modelo no serán ajustados nuevamente, no es necesario evaluar la *performance* en un conjunto separado [113]. Sin embargo, sí hace falta establecer la cantidad de épocas que durará el entrenamiento completo, con el objetivo de evitar el *overfitting*. Por este motivo, Chollet recomienda dividir el entrenamiento en dos fases [113]:

1. Primer entrenamiento: cada modelo seleccionado es entrenado nuevamente, con el mismo conjunto de entrenamiento y validación utilizados durante el proceso de ajuste. Para modificar la estrategia de *early stopping* a una más permisiva, la paciencia se aumentó a 10, en comparación al valor de 5 utilizado en los procesos de *tuning*. Adicionalmente, se eliminó el establecimiento de un valor mínimo, de manera que toda reducción en las métricas de error califique como una mejora. Una vez finalizado el entrenamiento, se almacena la época en la cual el error del conjunto de validación fue mínimo.

2. Reentrenamiento: en primer lugar, se incorpora el conjunto de validación al conjunto de entrenamiento utilizado previamente. Dado que el proceso no contará con un conjunto de validación, se utiliza el número de época obtenido en el paso anterior para determinar el final del entrenamiento. Sin embargo, dado que se está aumentando la cantidad de datos utilizados, el modelo requerirá de un mayor tiempo de convergencia. Por este motivo, el número de época es ajustado en un determinado porcentaje, de manera que se garantice la convergencia del modelo. Como valor inicial, se utilizó el aplicado por Chollet: un aumento del 30 % por sobre aquel obtenido [113].

Una vez completados ambos pasos, se obtiene el modelo entrenado para realizar predicciones sobre el conjunto de *testing*. Al igual que en etapas anteriores, se utilizó el mes de enero de 2020 como período de evaluación. Los conjuntos de entrenamiento y validación también se mantuvieron iguales a los utilizados anteriormente: enero-octubre y noviembre-diciembre de 2019, respectivamente.

5.4.4.2. Ajuste por función de pérdida y tamaño de *batch*

Tal como se mencionó anteriormente, los *tuners* no pueden optimizar la longitud de secuencia, la función de pérdida, el tamaño de *batch* ni la cantidad de capas ocultas del modelo. Por este motivo, se decidió realizar una serie de *tunings* previos, con el objetivo de encontrar el *batch size* y la función de pérdida que ofrecieran el mejor balance entre la duración del ajuste y la *performance* de los modelos. Debido a que las pruebas de las etapas anteriores mostraron que la longitud de secuencia óptima podía variar en base a la arquitectura, su análisis se postergó a la siguiente fase.

En base los valores utilizados en la literatura, se decidió experimentar sobre tamaños de *batch* entre 16 y 256, con un paso de 16. A su vez, además de utilizar MSE, se incorporó el estudio de la función MAE como pérdida del modelo. Si bien la ecuación de MAE posee un valor absoluto (ver tabla 4.1) y no es diferenciable cuando el valor predicho es igual al real, este hecho no es una limitante en la práctica, dado que Keras implementa contramedidas para manejar el caso [114].

En cuanto al espacio de búsqueda utilizado, se realizaron diversas modificaciones respecto al definido en el estudio de entornos y aceleradores (ver sección 5.4.3.2). Principalmente, se decidió ampliar determinados intervalos, con el objetivo de abarcar una mayor cantidad de combinaciones de hiperparámetros:

- La cantidad máxima de unidades se aumentó a 192.
- Se incorporó *dropout* recurrente. Se utilizaron valores idénticos en cada capa para evitar excederse en la cantidad de opciones a evaluar en el ajuste.
- Se ampliaron los valores para ambos tipos de *dropout* hasta 0,8, con el objetivo de analizar si las probabilidades elevadas podían ser beneficiosas.

- El máximo de épocas se estableció en 80, el doble al utilizado anteriormente.

La arquitectura de 3 capas LSTM, la longitud de secuencia de 40 y los optimizadores Adam y RMSprop se mantuvieron iguales a los utilizados anteriormente.

Resultados

En total se ejecutaron 10 procesos de *tuning*, cuyos resultados pueden encontrarse en la tabla 5.6. Los ajustes con *batch size* de 16 y 32 fueron realizados en Colab, mientras que el resto fue ejecutado en Kaggle.

Entorno	*Batch size*	Costo	Tiempo total	Iteraciones	MAPE promedio
Colab	16	MAE	8 h 47 min	90	13,11 %
Colab	16	MSE	6 h 1 min	90	17,02 %
Colab	32	MAE	4 h 53 min	90	12,46 %
Colab	32	MSE	4 h 14 min	90	15,42 %
Kaggle	64	MAE	2 h 46 min	90	13,45 %
Kaggle	64	MSE	3 h 4 min	90	17,70 %
Kaggle	128	MAE	2 h 17 min	90	14,92 %
Kaggle	128	MSE	2 h 36 min	90	17,89 %
Kaggle	256	MAE	2 h 29 min	90	15,23 %
Kaggle	256	MSE	2 h 15 min	90	14,96 %

Tabla 5.6: Comparativa entre la duración total de cada proceso (*tuning* y *testing*), junto con el MAPE de *testing* promediado entre sus 5 modelos seleccionados.

Al analizar los resultados, se pudo observar que las capacidades de la TPU se aprovecharon mejor a medida que se incrementó el tamaño de *batch*. Sin embargo, los resultados promedio sobre el conjunto de *testing* no evidenciaron una mejoría ante su aumento.

Por otra parte, en la figura 5.20 pueden visualizarse los resultados individuales para los 50 modelos seleccionados. La imagen deja en evidencia la superioridad de la función MAE en comparación a MSE. En promedio, los 25 modelos que utilizaron MAE como función de pérdida reportaron un MAPE de 13,84 %. En contraposición, los 25 modelos que aplicaron MSE reportaron un MAPE promedio de 16,59 %. Adicionalmente, el mejor de los 50 modelos también utilizó MAE y reportó un error del 11,07 %. En cuanto a los tiempos de ajuste, los obtenidos para ambas funciones resultaron similares entre sí.

Análisis del espacio de búsqueda utilizado

Si bien el objetivo principal de las pruebas fue la determinación de la función de error y el tamaño de *batch*, se aprovechó para realizar un análisis del comportamiento de los modelos ante ciertos hiperparámetros dentro del espacio de búsqueda. Las observaciones se basaron en los hiperparámetros de los 10 modelos obtenidos en los *tunings* con función MAE y *batch size* de 16 y 32, dado que su promedio de resultados fue el mejor.

En primer lugar, se observó que la elección de un número bajo en la cantidad de unidades no era beneficiosa. De los 10 modelos analizados, solamente 2 contaron con 32 unidades o menos y su MAPE promedio fue del 13,09 %. En contraposición, los 2 mejores promediaron un MAPE de 11,16 % y utilizaron una alta cantidad de unidades (144 y 176).

Posteriormente, se realizó un análisis acerca del impacto de la técnica de *dropout* en los modelos producidos. Se observó que 4 de los 5 mejores modelos utilizaron un valor de 0 para *dropout* luego de las capas recurrentes. A su vez, 2 de los 3 peores utilizaron un valor alto de *dropout* (0,6).

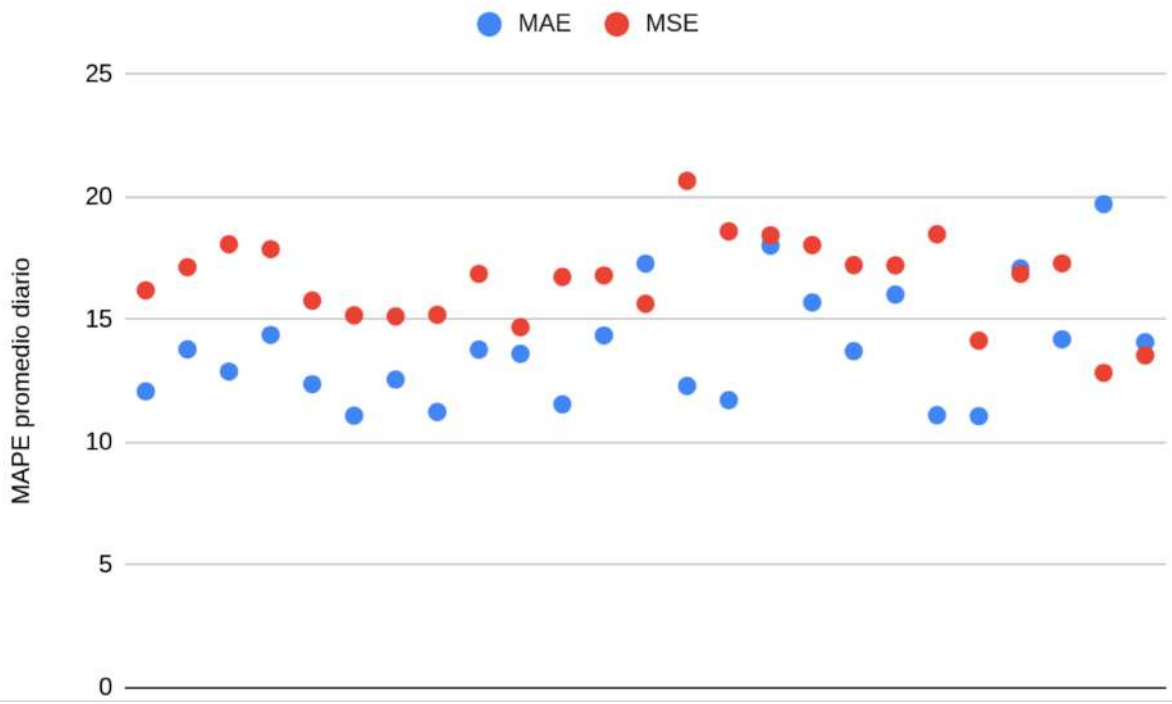


Figura 5.20: MAPE promedio diario durante el mes de enero de 2020 en cada uno de los 50 modelos seleccionados.

En cuanto al *dropout* recurrente, no se logró observar una relación clara con la *performance*. De los 10 modelos analizados, los 2 mejores no lo utilizaron. Sin embargo, el tercero empleó un valor de 0,6. De todas formas, se pudo observar que el valor 0,8 fue seleccionado únicamente en uno de los 50 modelos producidos a lo largo de toda la etapa. Además, fue el que reportó el peor MAPE (19,72 %).

Conclusiones del ajuste por función de pérdida y tamaño de *batch*

Como conclusión de la primera etapa de ajuste de hiperparámetros, se decidió mantener a la función MAE como métrica de error durante el entrenamiento. A su vez, se estableció el uso de un *batch* de tamaño 32, dado que sus modelos combinaron la mejor *performance* en las métricas analizadas y tiempos de entrenamiento aceptables.

Por otra parte, el análisis realizado sobre el espacio de búsqueda seleccionado permitió detectar ciertos hiperparámetros que no estaban siendo óptimos para la construcción de los modelos. Se decidió que en las fases posteriores se realice una modificación al conjunto, de manera que no se incluyan las opciones vinculadas a una baja *performance*.

5.4.4.3. Selección de arquitecturas y longitudes de secuencia

Una vez establecido el tamaño de *batch* y la función de pérdida a utilizar, se procedió a realizar *tunings* para analizar el impacto de la longitud de secuencia en diversas arquitecturas. El objetivo era determinar el conjunto de longitudes óptimo a estudiar durante el *tuning* final, así como analizar la posibilidad de descartar las arquitecturas con altos tiempos de entrenamiento y baja *performance*. Debido a su alta capacidad y los resultados obtenidos previamente, las arquitecturas escogidas para el análisis fueron las *stacked* y las bidireccionales, tanto para modelos LSTM como GRU.

Se estableció que las redes *stacked* a analizar contasen con 3 o 4 capas ocultas. En primer lugar, se deseaba incorporar una tercera capa a los modelos GRU, dado que no habían sido analizados en los ajustes previos. A su vez, se deseaba estudiar nuevamente los modelos LSTM al ajustarse en base a un espacio de búsqueda optimizado.

Debido a que los *tunings* preliminares con *stacked* LSTM no habían demostrado beneficios notorios a la hora de utilizar 3 capas ocultas, se decidió no analizar modelos con más de 4. Al mismo tiempo, los modelos observados en la literatura no contaban con una cantidad mayor a la establecida [115] [116].

Para el caso de los modelos bidireccionales, aquellos analizados en la primera fase de experimentación

contaron con un máximo de 2 capas. Debido a sus resultados positivos, se decidió utilizar modelos de 3 capas durante esta etapa, de manera que pueda determinarse si la inclusión de una capa adicional era de ayuda para la predicción. En vista de su alto costo computacional, no se analizaron modelos bidireccionales con 4 capas ocultas.

Adicionalmente, se determinó que las longitudes de secuencia a analizar fuesen 20, 50, 80, 110 y 140. En pruebas anteriores, las longitudes de menor tamaño habían reportado los mejores resultados. Además, el alto tiempo de entrenamiento que implicaba el aumento de la longitud hacía difícil su estudio. Sin embargo, el hecho de contar con aceleradores volvió factible realizar un análisis de mayor profundidad sobre dichas longitudes.

En base a las conclusiones obtenidas en la fase anterior (ver sección 5.4.4.2), se realizaron las siguientes modificaciones al espacio de búsqueda del algoritmo *Hyperband*:

- Se estableció que cada capa cuente con una cantidad de unidades y un valor de *dropout* recurrente independiente a las demás. Anteriormente, con el objetivo de limitar el alcance de las pruebas preliminares, se había establecido que todas las capas cuenten con los mismos hiperparámetros.
- El rango de unidades cambió de $[16, 192]$ a $[48, 208]$, dado que se observaron mejores resultados para modelos con una alta cantidad de unidades.
- Se eliminó el uso de *dropout* estándar, debido a su baja *performance* asociada.
- El valor posible de *dropout* recurrente se delimitó en 0, 0,2 y 0,5. Valores mayores no habían sido beneficiosos.

Resultados

A continuación, se listan las arquitecturas junto con las observaciones más relevantes:

- ***Stacked* LSTM:**
  - **Modelos de 3 capas:**
    - El mejor MAPE fue reportado por un modelo con longitud de secuencia de 20. El resultado obtenido fue de 10,75 %, el cual superó aquellos en las etapas de experimentación manual (11,35 % con 2 capas).
    - Los posteriores aumentos en la longitud de secuencia no lograron superar el resultado mencionado.
  - **Modelos de 4 capas:**
    - Debido a los resultados anteriores, se decidió limitar la experimentación con 4 capas a longitudes de 20 y 50.
    - Al analizar el *tuning* con longitud de secuencia 50, el MAPE promedio de los modelos seleccionados fue de 12,54 %. En comparación, había sido de 13,07 % para 3 capas y el mismo tamaño de secuencia.
    - Para longitud de secuencia de 20, el promedio en los modelos seleccionados aumentó de 11,29 % a 11,66 %.
    - El mejor MAPE (11,08 %) fue obtenido nuevamente con longitud de 20, aunque no logró superar al mejor modelo de 3 capas.
- ***Stacked* GRU:**
  - **Modelos de 3 capas:**
    - Se obtuvieron resultados similares a los observados para el modelo LSTM. El mejor modelo reportó un MAPE de 10,80 % y fue obtenido con longitud de secuencia de 20.
    - Nuevamente, los aumentos en la longitud de secuencia no reportaron mejoras. Inclusive, para un valor de 140, el error promedio entre los modelos seleccionados ascendió al 17,55 %, en comparación al promedio de 11,56 % para una longitud de 20.
  - **Modelos de 4 capas:**
    - Nuevamente, las experimentaciones se limitaron a longitudes de secuencia de 20 y 50.
    - Los resultados fueron similares al caso de LSTM, ningún modelo logró superar al GRU obtenido con 3 capas.

- Adicionalmente, los resultados promedio para longitud de secuencia de 20 empeoraron respecto a los anteriores. Sin embargo, aquellos para 50 mejoraron levemente (de 12,87 % a 12,65 %).

- **Bi-LSTM:**
  - Debido a que los altos tamaños de secuencia no habían sido beneficiosos en los modelos anteriores, solo se realizaron pruebas con valores de 20, 50 y 80.
  - Los mejores resultados fueron obtenidos con una longitud de 50. En particular, el mejor modelo reportó un MAPE de 10,44 %, mejor a los obtenidos con los modelos *stacked*.
  - El mejor promedio de resultados entre los modelos seleccionados fue para el *tuning* con longitud de secuencia 50 (11,10 %).
- **Bi-GRU:**
  - El mejor modelo obtuvo un MAPE de 10,79 %, peor al 10,44 % reportado con Bi-LSTM.
  - A su vez, los resultados promedio para cada longitud de secuencia también empeoraron en comparación a Bi-LSTM. Para 20, el MAPE aumentó de 11,58 % a 12,13 %. En el caso de 50, el aumento fue de 11,10 % a 14,31 %.

En la tabla 5.7, se pueden observar los mejores resultados por cada arquitectura analizada.

Modelo	**Arquitectura**	**Longitud de secuencia**	***Batch size***	**Función de pérdida**	**MAPE promedio**
LSTM	3 capas	20	32	MAE	10,75 %
GRU	3 capas	20	32	MAE	10,80 %
Bi-LSTM	3 capas	50	32	MAE	10,44 %
Bi-GRU	3 capas	20	32	MAE	10,79 %

Tabla 5.7: Mejores modelos LSTM y GRU obtenidos en la fase de selección de arquitecturas y longitud de secuencia.

Conclusiones sobre la selección de arquitecturas y longitudes de secuencia

Los resultados obtenidos permitieron delimitar las longitudes de secuencia a analizar en el futuro. Se determinó que tamaños mayores a 50 no son de ayuda para la predicción, de manera que fueron descartados de las experimentaciones futuras.

Los modelos de 3 capas LSTM y GRU lograron superar los resultados obtenidos durante la experimentación manual. Sin embargo, la inclusión de una capa adicional no demostró ser de utilidad. Por este motivo, los modelos con más de 3 capas fueron relegados de las siguientes etapas.

Finalmente, se determinó que las arquitecturas bidireccionales a analizar sean de tipo Bi-LSTM. Si bien Bi-GRU reportó buenos resultados, fueron inferiores a los de su contraparte. Debido a que los modelos bidireccionales resultaron los más costosos en tiempo de entrenamiento, se decidió continuar las experimentaciones únicamente con la variante LSTM.

5.4.4.4. Análisis del conjunto de validación

Hasta el momento, el conjunto de validación estaba compuesto por las mediciones de los meses de noviembre y diciembre de 2019. Dado que el objetivo era predecir el mes de enero de 2020, se había decidido que el conjunto se conforme por los meses más cercanos. Los modelos seleccionados para *testing* eran aquellos de mejor *score* durante el ajuste de hiperparámetros. A su vez, a la hora de realizar el reentrenamiento, el conjunto era utilizado para determinar la época exacta hasta la cual entrenar el modelo (que posteriormente era ajustada en un 30 %).

Por estos motivos, se consideró importante analizar posibles modificaciones sobre el conjunto de validación utilizado. Se decidió realizar una serie de pruebas para evaluar posibles cambios en el período de validación, así como en el valor de 30 % utilizado para realizar el ajuste de épocas.

Ajuste de la cantidad de épocas

Como base de la comparativa, se seleccionaron dos procesos de *tuning* ejecutados previamente: LSTM de 3 capas y secuencia de 20, y Bi-LSTM de 3 capas y secuencia de 50. La elección se debió a que, hasta el momento, habían sido las arquitecturas con mejor promedio de resultados.

En base a los modelos de mejor *score* obtenidos en la fase anterior, se realizaron nuevos reentrenamientos a la vez que se varió el valor utilizado para el ajuste de épocas. Se decidió experimentar con valores de 0 %, 10 %, 20 %, 40 %, 60 %, 100 %, además del 30 % utilizado anteriormente. A su vez, se incluyó en la comparativa un ajuste del 400 %, con el objetivo de demostrar la tendencia de ciertas métricas ante un aumento exagerado de la cantidad de épocas. A continuación, se detallan los resultados obtenidos:

- **LSTM:**
  - Al aumentar el porcentaje del ajuste, se observó que el MAE de entrenamiento se redujo notablemente (ver figura 5.21).
  - El MAPE de *testing* no se comportó de la misma forma. No se observó una mejora al aumentar el valor del ajuste (ver figura 5.22).
  - El mejor resultado se obtuvo con un ajuste del 10 % (11,23 %), aunque sin una mejora notoria sobre el 30 % utilizado previamente (11,29 %).
  - Al utilizar el ajuste del 400 %, se obtuvo el peor promedio de resultados: MAPE de 12,26 %.
- **Bi-LSTM:**
  - Los resultados obtenidos fueron similares a la arquitectura LSTM.
  - El MAE nuevamente se redujo al entrenar, aunque no hubo una correlación con las métricas de *testing*.
  - El mejor resultado (MAPE de 11,11 %) fue obtenido con el ajuste original de 30 %. El segundo mejor fue 11,13 %, al utilizar un 40 %.
  - Nuevamente, el ajuste del 400 % retornó el peor resultado (13,01 %).

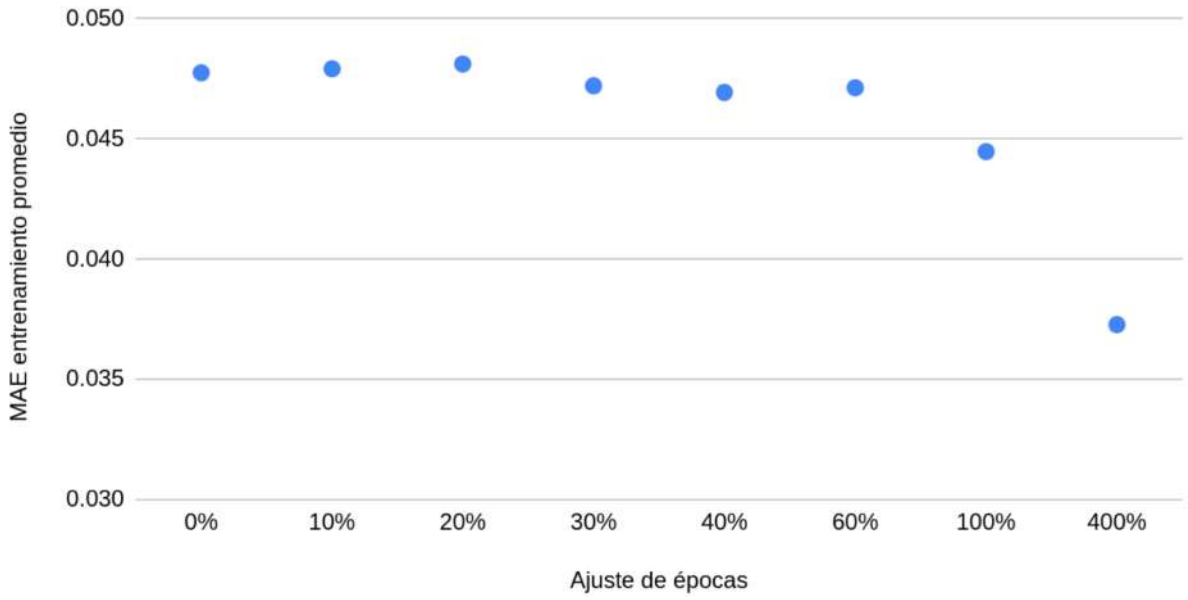


Figura 5.21: MAE de entrenamiento promedio entre los modelos LSTM seleccionados, en base al valor utilizado para el ajuste de épocas.

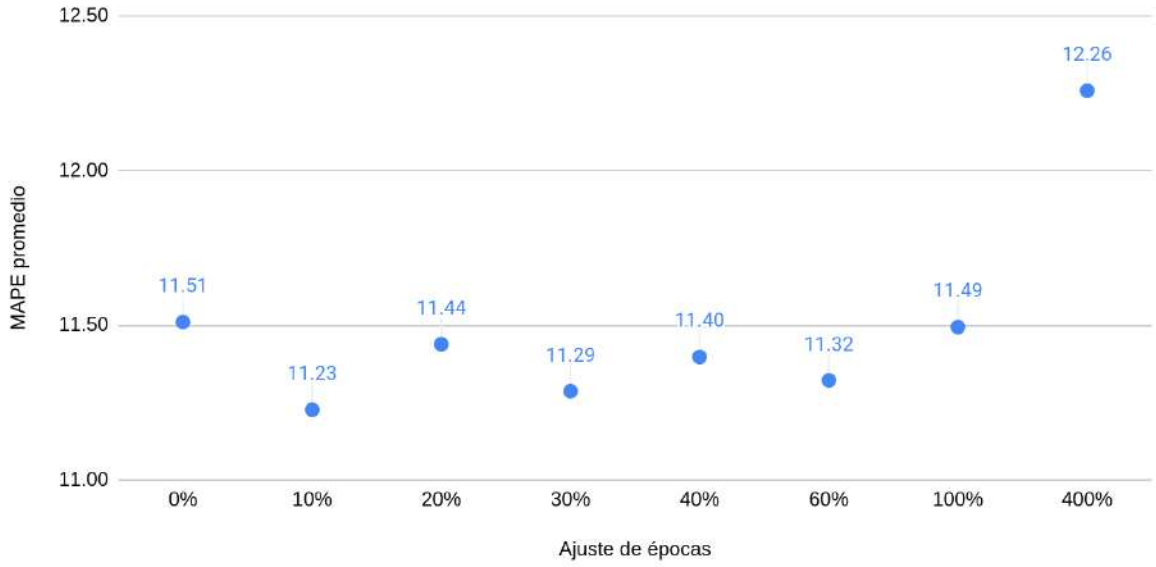


Figura 5.22: MAPE de *testing* promedio entre los modelos LSTM seleccionados, en base al valor utilizado para el ajuste de épocas.

Ajuste del período de validación

Para esta etapa de pruebas, se decidió modificar el intervalo de 2 meses utilizado como conjunto de validación. En base a los datos disponibles del 2019, los períodos seleccionados fueron: enero-febrero, mayo-junio, agosto-septiembre, octubre-noviembre y noviembre-diciembre, aquel utilizado originalmente.

Además de utilizar el *tuning* original como base de la comparativa, se ejecutaron nuevos procesos sobre la arquitectura LSTM de 3 capas. Se decidió no incluir nuevamente el modelo bidireccional, ya que las conclusiones extraídas en la prueba anterior habían sido iguales a las obtenidas con el modelo LSTM. Los resultados obtenidos fueron los siguientes:

- El MAE de entrenamiento fue similar con todos los períodos. Octubre-noviembre reportó el promedio más bajo, seguido de noviembre-diciembre.
- En cuanto al MAPE de *testing*, los períodos más cercanos al mes de estudio fueron los que lograron los mejores resultados (ver figura 5.23).
- En promedio, noviembre-diciembre había reportado un 11,28 %. Por otra parte, octubre-noviembre logró un 11,57 %.
- El mejor modelo fue obtenido con el uso de agosto-septiembre (10,42 %), sin embargo, el promedio de error de los modelos seleccionados ascendió al 12,09 %.

Conclusiones sobre el análisis del conjunto de validación

Ninguna de las modificaciones introducidas logró mejorar los resultados de forma clara. En promedio, el período de noviembre-diciembre reportó las mejores métricas. A su vez, las variaciones en el porcentaje de ajuste no demostraron una tendencia para ninguna de las arquitecturas analizadas. Por estos motivos, se decidió mantener el período y el ajuste que venían siendo utilizados hasta el momento: noviembre-diciembre y 30 %, respectivamente.

5.4.4.5. Mecanismos de atención

Posteriormente, se decidió evaluar arquitecturas que cuenten con el mecanismo de atención introducido en la sección 4.2.3.5. Para la implementación, se decidió utilizar la librería *Keras Attention* [117], dado que fue la opción con la mayor cantidad de documentación disponible encontrada. La librería permite definir la cantidad de unidades en la capa de atención, así como el tipo de función de puntaje, que por

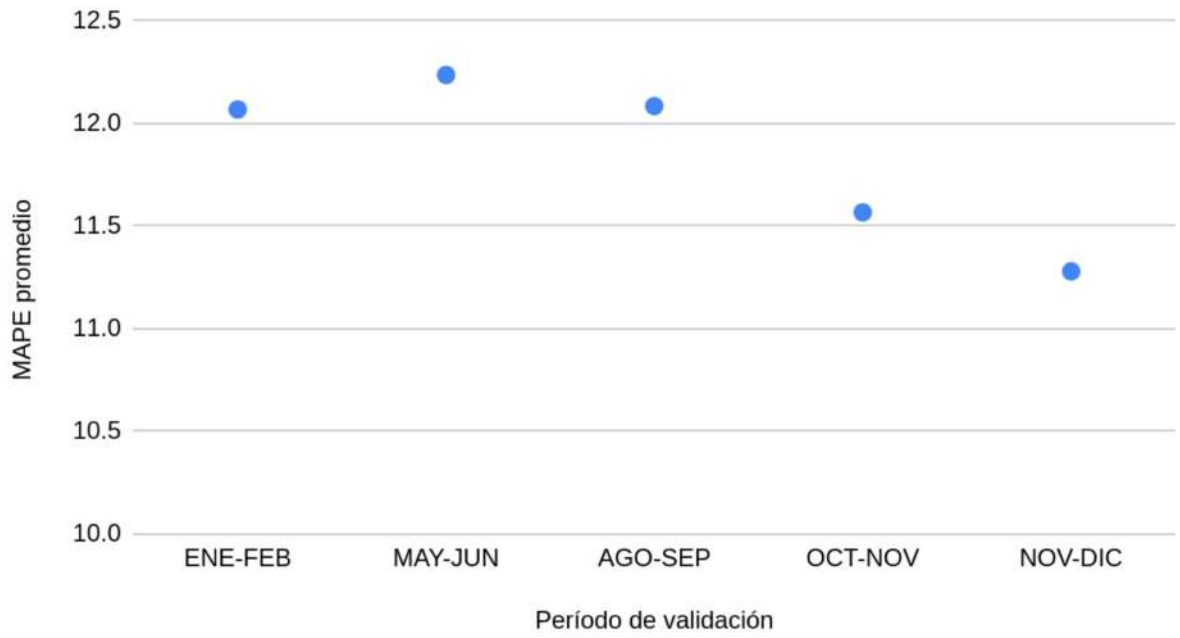


Figura 5.23: MAPE de *testing* promedio entre los modelos LSTM seleccionados, en base al período de validación utilizado en el *tuning*.

defecto es el estilo multiplicativo de Luong. Otra opción analizada fue la capa *Attention* de Keras [118]. Sin embargo, no se encontraron ejemplos adecuados a los modelos utilizados durante el proyecto.

Si bien en etapas anteriores se habían descartado las longitudes de secuencia de mayor tamaño, se decidió volver a incorporarlas para realizar esta experimentación. El razonamiento fue que, dada su capacidad para seleccionar la información relevante, los mecanismos de atención podían llegar a ser beneficiosos para las secuencias extensas.

Debido a los resultados obtenidos previamente, se estudiaron las arquitecturas LSTM, GRU y Bi-LSTM, cada una con 3 capas ocultas. Fue necesario incluir un nuevo hiperparámetro en el espacio de búsqueda: la cantidad de unidades en la capa de atención. Se definió que los posibles valores se encuentren entre 16 y 128, con pasos de 16. A su vez, con el objetivo de no sobreextenderse en las experimentaciones, se decidió únicamente utilizar la función de puntaje por *default*. Los valores posibles para el resto de hiperparámetros se mantuvo igual a la etapa anterior.

Resultados

- **LSTM:** se realizaron ejecuciones con valores de 20, 50, 80 y 110 para la longitud de secuencia. Los mejores resultados promedio fueron obtenidos con un tamaño de 50 (MAPE de 11,49 %), mientras que 110 reportó las peores métricas (12,94 %). El mejor modelo utilizó un tamaño de secuencia de 20 junto con una cantidad de unidades de atención de 16. Logró un MAPE de 10,66 %, por lo que superó el mejor resultado obtenido en modelos LSTM sin atención (10,75 %).
- **Bi-LSTM:** debido a los malos resultados y la duración del entrenamiento al utilizar secuencias de tamaño 110, se decidió únicamente utilizar 20, 50 y 80. Para esta arquitectura, el mejor promedio de resultados fue obtenido con longitud de 20 (10,90 %), mientras que 80 reportó los peores con notoria diferencia (14,01 %). El mejor modelo (ver figura 5.24), volvió a utilizar longitud de 20 y contó con 64 unidades en la capa de atención. Alcanzó un MAPE de 10,53 %, ligeramente peor del mejor valor obtenido en modelos bidireccionales sin atención (10,44 %).
- **GRU:** al igual que para la arquitectura anterior, se decidió descartar las longitudes de secuencia altas, de manera que solo se experimentó con 20 y 50. Si bien el mejor resultado fue obtenido en un modelo con secuencia de tamaño 20, el MAPE promedio entre los 5 seleccionados aumentó en relación al resto de arquitecturas (13,45 %). Para tamaños de 50, el aumento fue más pronunciado y el error alcanzó un promedio de 15,36 %. El mejor modelo obtenido reportó un MAPE de 11,96 %, lejos del 10,80 % reportado sin atención. Utilizó una cantidad de 64 unidades de atención.

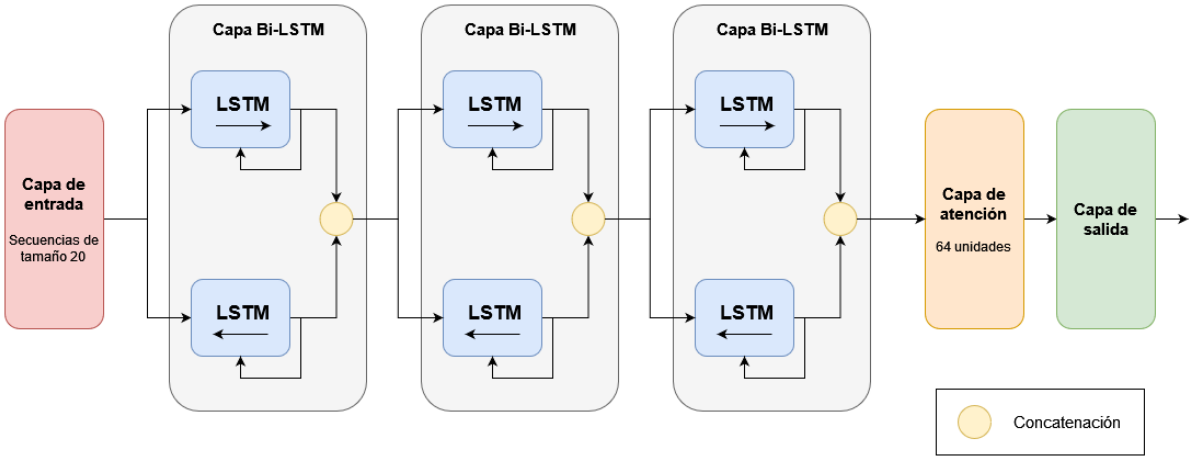


Figura 5.24: Arquitectura del mejor modelo obtenido a partir de la utilización de 3 capas Bi-LSTM y la incorporación de un mecanismo de atención.

En la tabla 5.8 se observa una comparativa entre los mejores modelos obtenidos en esta etapa de experimentación y los recopilados hasta el momento que no incluían mecanismos de atención.

Modelo	Arquitectura	Longitud de secuencia	*Batch size*	Función de pérdida	Atención	MAPE promedio
LSTM	3 capas	20	32	MAE	No	10,75 %
LSTM	3 capas	20	32	MAE	Sí	10,66 %
Bi-LSTM	3 capas	20	32	MAE	No	10,44 %
Bi-LSTM	3 capas	20	32	MAE	Sí	10,53 %
GRU	3 capas	50	32	MAE	No	10,80 %
GRU	3 capas	20	32	MAE	Sí	11,96 %

Tabla 5.8: Comparativa entre los mejores modelos LSTM y GRU obtenidos antes y después del añadido de mecanismos de atención.

Conclusiones sobre mecanismos de atención

La incorporación de los mecanismos de atención fue positiva en los modelos LSTM y Bi-LSTM. Sin embargo, los resultados obtenidos con GRU no reflejaron mejoras sobre aquellos logrados anteriormente. En base a las métricas obtenidas, se decidió continuar experimentando con mecanismos de atención en etapas futuras, pero sin descartar arquitecturas que no lo incorporen.

5.4.4.6. Análisis de MAPE como función de pérdida

Para las pruebas realizadas hasta el momento, únicamente se experimentó con el uso de MSE o MAE como funciones de pérdida. Sin embargo, la métrica principal sobre la cual se evaluaba la *performance* era el MAPE. Por este motivo, surgió la idea de utilizar dicha métrica como la función de costo del modelo. Al igual que con MAE, Keras permite su uso, a pesar de que la ecuación posee un valor absoluto [114].

Anteriormente, se había experimentado brevemente con el uso de MAPE en modelos recurrentes. Sin embargo, no se había logrado obtener resultados prometedores, dado que las métricas se disparaban durante el entrenamiento y no era posible la convergencia. Por este motivo, antes de comenzar a trabajar sobre los conjuntos de pruebas, se realizó un análisis sobre la base de datos para analizar posibles anomalías. Como se explica en la sección 5.1.4.2, se identificaron diversos valores atípicos que fueron depurados de la base de datos. Una vez efectuada la limpieza, fue posible entrenar correctamente a los modelos utilizando MAPE.

Al igual que en etapas anteriores, se decidió experimentar sobre modelos con 3 capas ocultas: LSTM, GRU y Bi-LSTM. El espacio de búsqueda se mantuvo igual al utilizado en las fases previas. No se

experimentó sobre modelos con atención, dado que ambas fases fueron desarrolladas paralelamente. Los modelos fueron analizados con secuencias de tamaño 20 y 50.

Resultados

A continuación, se detallan las pruebas realizadas y la comparativa con arquitecturas anteriores que no incluyeron atención:

- **LSTM:**
  - En las pruebas con MAE, el MAPE promedio entre los 5 modelos seleccionados había sido de 11,29 % y 13,07 % para las longitudes 20 y 50, respectivamente. Con el uso de MAPE como pérdida, los promedios fueron de 12,94 % y 10,86 %.
  - Al aplicar función de costo MAPE, se obtuvo el mejor modelo hasta el momento, que contó con longitud de secuencia 50 y reportó un error de *testing* del 10,36 %. Anteriormente, el mejor modelo LSTM había obtenido 10,75 %.
- **Bi-LSTM:**
  - Los resultados promedio al utilizar MAPE fueron de 10,70 % y 13,24 % para longitudes de secuencia de 20 y 50, respectivamente. En comparación, el uso de MAE había reportado 11,59 % y 11,11 %.
  - Nuevamente, el uso de MAPE como función de pérdida llevó a obtener el mejor modelo para la arquitectura. Uno de los modelos con tamaño de secuencia de 20 obtuvo un 10,30 %. Previamente, el mejor resultado había sido de 10,44 %.
- **GRU:**
  - Al utilizar MAPE, las pruebas reportaron un error de *testing* promedio del 13,05 % para los modelos seleccionados con longitud de secuencia 20. Para tamaños de 50, el error promedio fue del 11,04 %. En los modelos GRU entrenados con MAE, los promedios habían sido de 11,57 % y 12,88 %, respectivamente.
  - Al igual que con las arquitecturas anteriores, el uso de MAPE reportó el mejor modelo, que alcanzó un error del 10,68 % con una secuencia de tamaño 50. Anteriormente, el mejor resultado había sido de 10,80 %.

Si bien el uso de MAPE reportó los mejores resultados en todas las arquitecturas, se observó una degradación del orden del 3 % en otras métricas calculadas durante la experimentación. Dentro de ellas se encontraron el MAE o el R^2 de *testing*, que fueron calculados complementariamente al MAPE para el mes de enero de 2020. Por ejemplo, para el mejor modelo Bi-LSTM con longitud de secuencia de 20 mencionado anteriormente, el MAE aumentó de 0,037 a 0,039, mientras que el R^2 decreció de 0,90 a 0,88.

En la tabla 5.9, se observa una comparativa entre los mejores modelos obtenidos en esta etapa y los recopilados hasta el momento que utilizaban MAE como función de pérdida.

Modelo	**Arquitectura**	**Longitud de secuencia**	***Batch size***	**Función de pérdida**	**MAPE promedio**
LSTM	3 capas	20	32	MAE	10,75 %
LSTM	3 capas	50	32	MAPE	10,36 %
Bi-LSTM	3 capas	20	32	MAE	10,44 %
Bi-LSTM	3 capas	20	32	MAPE	10,30 %
GRU	3 capas	50	32	MAE	10,80 %
GRU	3 capas	50	32	MAPE	10,68 %

Tabla 5.9: Comparativa entre los mejores modelos LSTM y GRU obtenidos antes y después de la utilización de MAPE como función de pérdida

Conclusiones sobre el análisis de MAPE como función de pérdida

El uso de MAPE como función de pérdida fue de utilidad para los modelos. El promedio de resultados fue variable, pero todas las arquitecturas superaron ligeramente su mejor MAPE obtenido previamente. Si bien hubo mejoras alentadoras, también se detectó una pequeña degradación en otras métricas de *testing*. Por este motivo, se decidió utilizar tanto a MAE como MAPE en las experimentaciones futuras.

5.4.4.7. Conclusión de la delimitación de arquitecturas e hiperparámetros

Se utilizó de forma exitosa el *tuning* de hiperparámetros para evaluar diferentes alternativas y modificaciones sobre los modelos de *Deep Learning*, de manera que se facilite la experimentación en la fase posterior:

- Se estableció que las arquitecturas a estudiar fueran LSTM, GRU y Bi-LSTM. La cantidad máxima de capas ocultas se definió en 3, dado que la experimentación con 4 no fue exitosa.
- Se estableció el uso de longitudes de secuencia pequeñas (20 y 50). Valores mayores no reportaron mejoras y aumentaron el tiempo de entrenamiento.
- Se eliminó el uso de *dropout* estándar. Se decidió mantener su variante recurrente.
- Se descartó el uso de MSE como función de costo, dado que sus resultados fueron inferiores a MAE.
- Ante sus resultados positivos, se incorporó la opción de utilizar MAPE como función de pérdida.
- La inclusión de los mecanismos de atención fue exitosa, por lo que se decidió evaluarlos como alternativa adicional.
- El período de validación fue mantenido en noviembre-diciembre, mientras que el ajuste de épocas del reentrenamiento se estableció en 30 %.

5.4.5. Incorporación del período 2016-2018

Una vez completadas las experimentaciones con un único año de entrenamiento, se procedió a incorporar el resto de datos disponibles a los procesos de ajuste. Debido a que existieron dificultades en la adquisición de datos actuales, el ICyTE proveyó el período 2016-2018, que se consideró suficiente para proseguir con la experimentación.

En base a los resultados obtenidos en la fase previa, fue posible delimitar las arquitecturas y los hiperparámetros de estudio. El objetivo de la etapa fue la determinación del modelo predictivo final.

5.4.5.1. Definición del tamaño de *batch*

Durante las fases previas, el tamaño de *batch* del modelo había sido establecido en 32. La decisión había sido tomada en base a las métricas obtenidas con ese valor, así como el tiempo de duración de los procesos de *tuning*. La incorporación de 3 años adicionales de datos implicó una ralentización general de los procesos, de manera que fue necesario reevaluar el tamaño utilizado.

A modo experimental, se decidió realizar una batería de *tunings* con distintos tamaños de *batch*. El modelo utilizado como base fue el LSTM de 3 capas ocultas y función de pérdida MAE. El espacio de búsqueda fue modificado en base al análisis de los resultados de la fase anterior:

- La cantidad máxima de unidades se elevó a 368.
- El paso en el intervalo de la cantidad de unidades se modificó de 16 a 32. El objetivo fue balancear la ampliación del intervalo con una reducción en la cantidad total de opciones.
- Se decidió remover el optimizador RMSprop de las opciones, de manera que solo se utilice Adam. Solo 1 de los 15 mejores modelos con MAPE como función de pérdida habían utilizado RMSprop. Lo mismo sucedió en aquellos que utilizaron MAE.

En total, se realizaron 6 procesos de *tuning*. Los tamaños de *batch* utilizados fueron 32, 64, 128, 256, 512 y 1024. A continuación, se exponen las principales observaciones:

- Los modelos que utilizaron 32, el tamaño que se utilizó durante la fase anterior, reportaron un MAPE promedio de 11,10 % (ver figura 5.25).

- Los mejores resultados se obtuvieron con tamaño de 64. Sus modelos lograron un MAPE de 10,74 %. A su vez, su uso redujo la duración del proceso de *tuning* en un 34 %, de 378 a 242 minutos (ver figura 5.26).
- A pesar de que los tamaños altos reportaron una clara mejoría en los tiempos de entrenamiento, la reducción en *performance* también fue notoria. Para 1024, el MAPE promedio ascendió a 22,50 %. En comparación al *tuning* de 32, su duración se redujo en un 68 % (122 minutos).

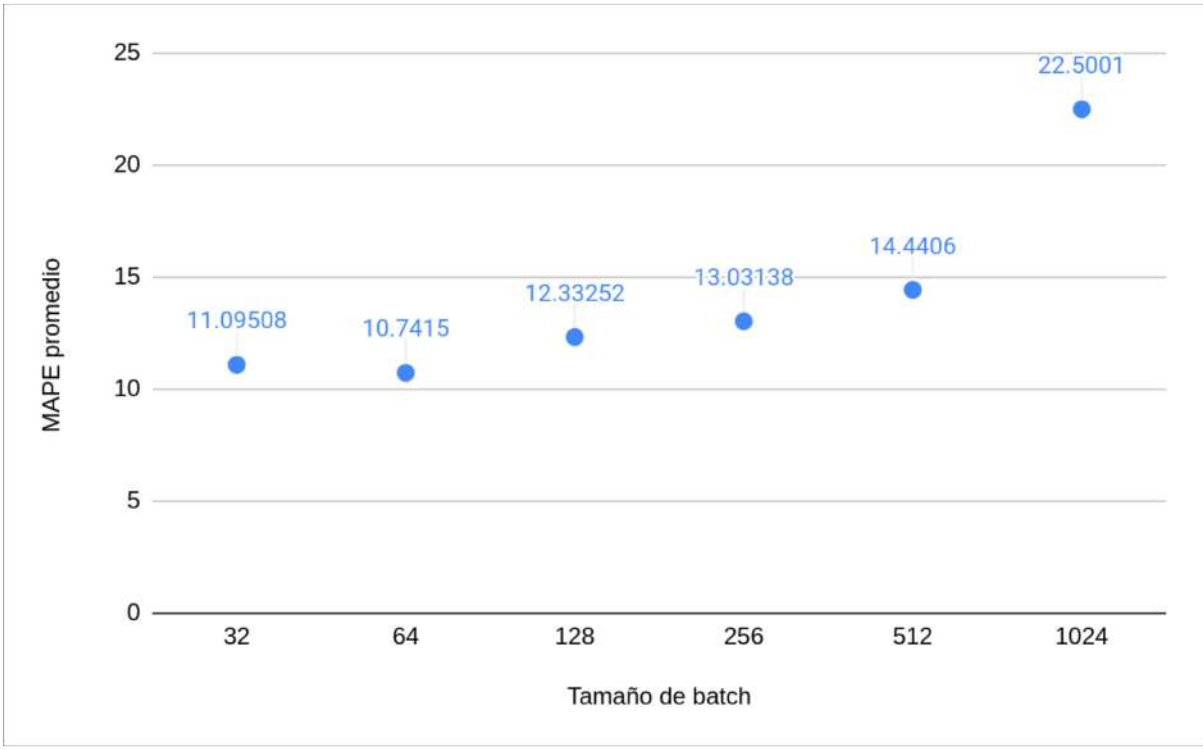


Figura 5.25: MAPE promedio para los 5 mejores modelos de cada proceso de *tuning*, en base al *batch size* y para una arquitectura LSTM de 3 capas.

En base a las observaciones realizadas, se decidió continuar las experimentaciones con un tamaño de *batch* de 64. Su uso reportó los mejores resultados en la etapa, mientras que la duración total del proceso (242 minutos) se consideró aceptable para el tipo de arquitectura analizada y las cargas de trabajo del proyecto.

5.4.5.2. Incorporación de la potencia eléctrica del día anterior

Hasta el momento, no había sido posible incorporar la potencia eléctrica como variable de entrada del modelo. En etapas anteriores, se había intentado implementar un mecanismo de retroalimentación para las predicciones generadas. Sin embargo, la búsqueda de una solución no fue exitosa y se desistió de la idea.

Durante la planificación de los procesos de ajuste finales y en base al trabajo *"Short-term photovoltaic power production forecasting based on novel hybrid data-driven models"* de Alrashidi y Raham [119], se logró encontrar una forma de incorporar a la potencia eléctrica en las secuencias de entrada: que cada instante temporal incluya la medición de 24 h antes. Para determinar la utilidad de su inclusión, se decidió realizar una experimentación breve y analizar sus resultados.

Experimentación

La obtención de cada valor fue dificultosa, debido a que la cantidad de mediciones para cada día no es constante en la base de datos (ver sección 5.1.2.3). A su vez, al contar con días y períodos incompletos, la existencia del día anterior a la medición no estaba asegurada.

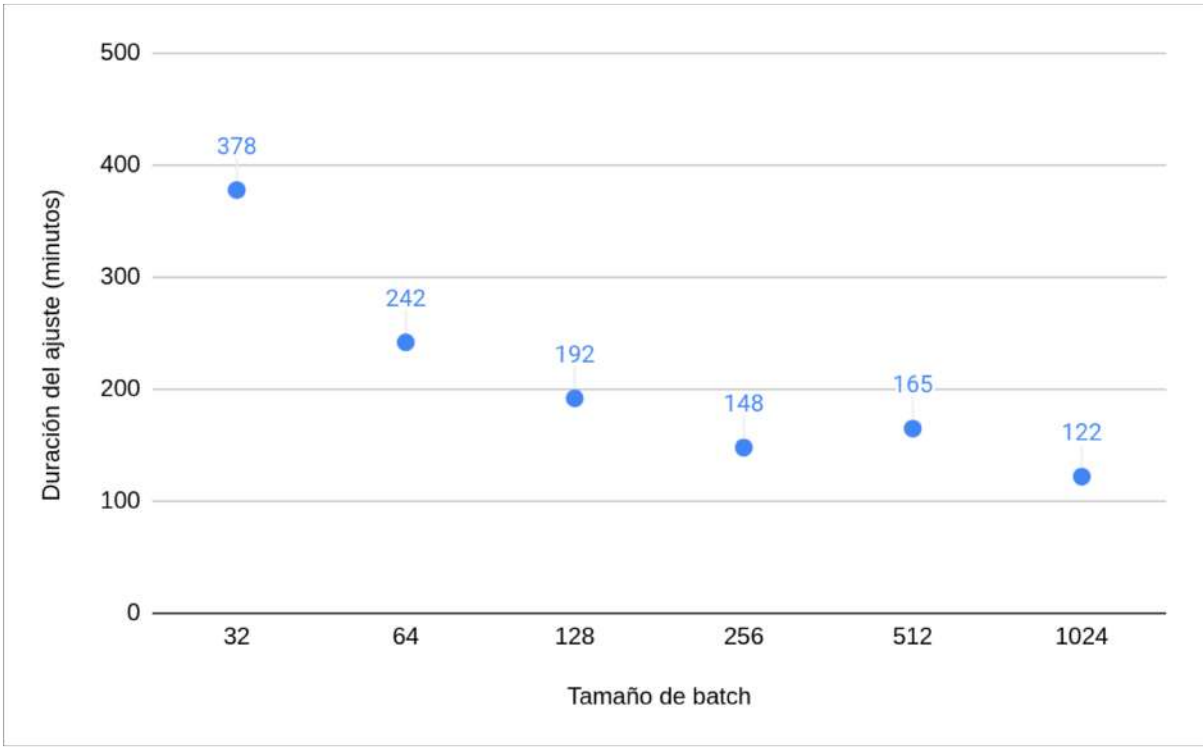


Figura 5.26: Duración del proceso de *tuning* en base al *batch size*.

Para solucionar el problema, se experimentó con la utilización de intervalos para la obtención de la medición más cercana a las 24 h anteriores de cada instante temporal. Para asociar a cada instante con una potencia eléctrica, se estableció el siguiente orden de prioridad:

1. Una medición de exactamente 24 h antes.
2. Una medición del día anterior, hasta con un máximo de 40 minutos de diferencia.
3. De no contar con el día anterior o con mediciones en el período de 40 minutos, retroceder iterativamente un día adicional hacia atrás hasta encontrar un día con una medición a ±40 minutos de la hora original.

El período de 40 minutos fue establecido en base a una serie de pruebas preliminares, en donde se evaluó el nivel de correlación entre la nueva variable y la potencia eléctrica actual para distintos valores. Para limitar la duración de la experimentación, el retroceso se limitó temporalmente a un máximo de 2 días. De forma iterativa, se utilizaron intervalos de 5, 10, 15, 20, 25, 30, 35, 40, 45 y 50 minutos. Se encontró que el período de 40 minutos maximizaba la correlación entre las variables, hasta alcanzar un valor de 0,5159. Por este motivo, fue seleccionado como el intervalo a utilizar durante el resto del proyecto.

Una vez definido el período de búsqueda, se realizó un ajuste de hiperparámetros de prueba para evaluar la inclusión de la nueva variable. Se utilizó como base la arquitectura LSTM de 3 capas, dado que había sido una con las que se obtuvo mejor *performance* hasta el momento.

Un único proceso de ajuste fue suficiente para evaluar la incorporación de la variable, dado que se obtuvo un modelo que registró un MAPE de 9,81 % y mejoró a todos los obtenidos hasta el momento. Ante los resultados reportados, se decidió mantener a la potencia eléctrica como variable de entrada y utilizarla durante los ajustes de hiperparámetros finales.

5.4.5.3. Ajustes de hiperparámetros finales

Para la etapa final de *tunings*, se incluyó cada una de las arquitecturas que habían demostrado resultados prometedores en las fases anteriores: LSTM, GRU y Bi-LSTM. En base a las observaciones

efectuadas previamente, los modelos incluidos utilizaron 1, 2 o 3 capas ocultas. A su vez, algunas de las variantes analizadas incluyeron mecanismos de atención. Únicamente se estudió el uso de longitudes de secuencia de tamaño 20 y 50.

El espacio de búsqueda final fue definido de la siguiente forma:

- Cantidad de unidades por capa: entre 48 y 368, con paso de 32.
- Cantidad de unidades en la capa de atención: 16, 32, 64 y 128.
- *Dropout* recurrente: valores de 0, 0,2 y 0,5.

Al mismo tiempo, el único optimizador definido fue Adam, con valores *default* para sus parámetros. Las funciones de costo utilizadas fueron MAE y MAPE.

En la tabla 5.10 se listan las 10 arquitecturas con el mejor promedio de resultados entre sus 5 modelos seleccionados. Como puede observarse, con las que se obtuvo un mejor rendimiento se incluyó a MAPE como función de pérdida.

La mejor arquitectura que utilizó MAE fue GRU con 2 capas ocultas. Sin embargo, reportó un MAPE promedio de 10,41 %, ligeramente por encima del resto. Al compararlo con el mejor modelo de la tabla, se observó que el GRU mencionado aumentó el MAPE promedio en un 6,12 %. A su vez, la degradación de las métricas MAE y R^2 fue del 3,86 % y 1,76 %, respectivamente. Dado que los niveles de degradación fueron muy bajos, se optó por descartar el modelo que utilizó MAE, ya que MAPE había sido la métrica definida como de mayor importancia en el proyecto. Al ser inferiores al modelo GRU, también se descartó al resto de modelos que utilizaron MAE como función de costo.

Red	Cantidad de capas	Longitud de secuencia	Atención	Función de coste	MAPE promedio
LSTM	2	20	No	MAPE	9,81 %
Bi-LSTM	2	50	No	MAPE	10,08 %
LSTM	2	50	No	MAPE	9,85 %
LSTM	3	20	No	MAPE	9,88 %
GRU	2	50	No	MAPE	10,29 %
Bi-LSTM	1	20	No	MAPE	10,13 %
Bi-LSTM	3	50	No	MAPE	10,10 %
LSTM	3	50	No	MAPE	10,25 %
LSTM	2	20	Sí	MAPE	10,22 %
LSTM	2	50	Sí	MAPE	10,15 %

Tabla 5.10: Listado de las 10 arquitecturas con mejor promedio de MAPE durante el proceso de *tuning* final.

5.4.5.4. Definición del tamaño del conjunto de validación

Como último paso antes de la selección del modelo final, se decidió realizar una nueva experimentación relacionada al conjunto de validación. Las pruebas consistieron en analizar la variación de los resultados en base a la cantidad de meses incluidos en el conjunto.

Al igual que en etapas anteriores, los *tunings* realizados habían utilizado los meses de noviembre y diciembre como período. Se decidió realizar una serie de pruebas que analicen los resultados al utilizar 1 o 3 meses dentro del conjunto, de manera que pueda determinarse un valor final.

Como base, se decidió utilizar las 10 mejores arquitecturas obtenidas durante el proceso de ajuste final. Para reducir el sesgo, los *tunings* de las arquitecturas mencionadas fueron ejecutados nuevamente. Posteriormente, se realizaron ajustes donde los períodos de validación utilizados fueron diciembre y octubre-noviembre-diciembre, respectivamente.

En la tabla 5.11 se muestran los resultados finales obtenidos para cada arquitectura. Si bien no se observó una tendencia clara en los resultados, se apreció que el mejor modelo en diversas arquitecturas únicamente requirió de un mes en el conjunto de validación. Un ejemplo fue el modelo LSTM de 3 capas, que reportó el mejor MAPE de la fase: 9,18 %.

Red	Capas	Tamaño de secuencia	Atención	1 mes	2 meses	3 meses
LSTM	2	20	No	9,89 % (9,66 %)	9,84 % (9,40 %)	10,19 % (9,62 %)
Bi-LSTM	2	50	No	10,01 % (9,63 %)	10,38 % (9,98 %)	10,28 % (9,75 %)
LSTM	2	50	No	10,59 % (9,46 %)	10,01 % (9,41 %)	9,91 % (9,37 %)
LSTM	3	20	No	9,67 % (9,48 %)	9,81 % (9,60 %)	9,63 % (9,39 %)
GRU	2	50	No	11,78 % (9,42 %)	10,14 % (9,22 %)	10,76 % (10,09 %)
Bi-LSTM	1	20	No	9,77 % (9,50 %)	9,59 % (9,25 %)	9,81 % (9,70 %)
Bi-LSTM	3	50	No	10,39 % (10,12 %)	9,84 % (9,72 %)	11,43 % (9,97 %)
LSTM	3	50	No	9,65 % (9,18 %)	9,79 % (9,65 %)	9,73 % (9,53 %)
LSTM	2	20	Sí	9,98 % (9,62 %)	9,81 % (9,75 %)	11,03 % (9,86 %)
LSTM	2	50	Sí	9,83 % (9,57 %)	9,91 % (9,55 %)	10,47 % (9,39 %)

Tabla 5.11: Promedio de resultados entre los 5 mejores modelos de cada arquitectura, en base a la cantidad de meses en el conjunto de validación. Entre paréntesis, se especifica el mejor resultado obtenido dentro de cada arquitectura.

5.4.5.5. Selección de modelos y resultados finales

Finalmente, se procedió a realizar la selección del modelo predictivo final. A modo de base, se utilizaron los modelos generados durante el análisis del conjunto de validación (ver tabla 5.11). Se decidió preseleccionar a los 10 mejores modelos producidos, cada uno asociado a una cantidad determinada de meses dentro de su conjunto de validación.

Para determinar los resultados finales, cada modelo fue ejecutado un total de 10 veces. En la tabla 5.12, se encuentran las métricas finales de cada uno.

En base a los resultados obtenidos, se decidió seleccionar como modelo final (ver figura 5.27) al LSTM de 3 capas y longitud de secuencia de tamaño 50. El modelo contó con capas de 336, 304 y 208 unidades. A su vez, utilizó *dropout* recurrente con valor 0,5 en la última capa oculta. Como puede observarse, el modelo no solo minimizó el MAPE obtenido, sino que también reportó las mejores métricas de MAE, R^2, R^2 ajustado y RMSE. Si bien sus tiempos de ejecución fueron los más elevados, el Dr. Ing. S. A. González consideró que no imponían una traba para el uso del modelo. En la figura 5.28, se observan las predicciones para cada día del mes de enero de 2020.

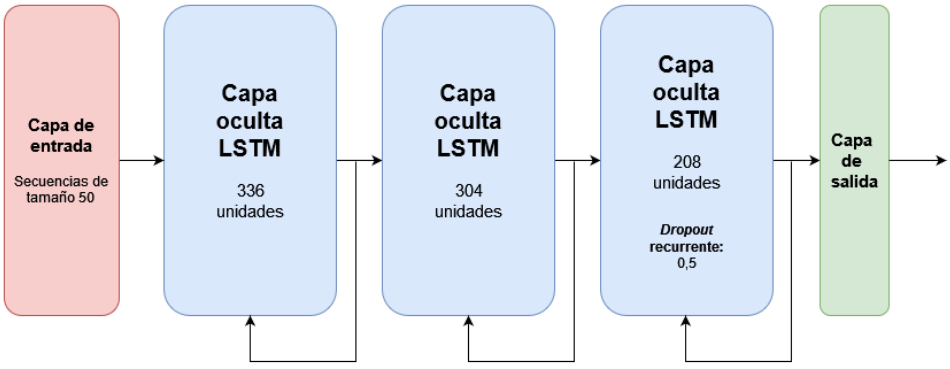


Figura 5.27: Modelo LSTM final.

5.4.5.6. Comparativa estacional

Una vez seleccionado el modelo final, se decidió evaluar su *performance* ante distintos conjuntos de *testing*. El proyecto priorizó el uso del mes de enero de 2020, dado que los meses de verano son considerados como los más importantes para evaluar las métricas de los modelos. Sin embargo, resultaba de interés evaluar su capacidad al enfrentarse a conjuntos de datos pertenecientes a otras estaciones.

Se decidió realizar 10 ejecuciones en las estaciones de primavera, invierno y otoño. Debido a que el proceso de *testing* de verano había utilizado el mes de enero, los meses seleccionados para la evaluación

Red	Cantidad de capas	Función de pérdida	Tamaño de secuencia	Atención	Meses de validación	MAPE promedio	MAPE mín.	MAPE máx.	MAE	R^2	R^2 ajustado	RMSE	Tiempo de ejecución (s)
LSTM	3	MAPE	50	No	1	9,46 (±0,20)	1,42 (±0,22)	33,98 (±3,53)	0,032 (±0,0012)	0,903 (±0,0048)	0,895 (±0,0052)	0,069 (±0,0017)	3279 (±724)
GRU	2	MAPE	50	No	2	9,58 (±0,36)	1,52 (±0,31)	26,70 (±1,37)	0,034 (±0,0018)	0,898 (±0,0062)	0,889 (±0,0067)	0,074 (±0,0017)	1591 (±486)
Bi-LSTM	1	MAPE	20	No	2	9,67 (±0,31)	1,81 (±0,44)	26,33 (±0,98)	0,035 (± 0,0017)	0,888 (±0,0053)	0,884 (±0,0055)	0,075 (±0,0018)	883 (±134)
LSTM	2	MAPE	50	No	3	10,10 (±0,71)	2,17 (±0,60)	28,31 (±1,80)	0,036 (±0,0035)	0,889 (±0,0123)	0,880 (±0,0133)	0,076 (±0,0030)	1081 (±513)
LSTM	2	MAPE	50	Sí	3	11,13 (±1,01)	2,67 (±0,64)	27,91 (±1,35)	0,041 (±0,0037)	0,871 (±0,0158)	0,861 (±0,0171)	0,079 (±0,0050)	1119 (±653)
LSTM	3	MAPE	20	No	3	9,88 (±0,50)	1,82 (±0,55)	27,58 (±1,37)	0,035 (±0,0035)	0,891 (±0,0115)	0,888 (±0,0118)	0,071 (±0,0036)	937 (±385)
LSTM	2	MAPE	20	No	2	10,04 (±0,48)	2,09 (±0,85)	28,79 (±2,10)	0,037 (±0,0029)	0,892 (±0,0060)	0,889 (±0,0061)	0,073 (±0,0032)	1067 (±246)
LSTM	2	MAPE	50	No	2	9,95 (±0,86)	1,66 (±0,42)	31,11 (±2,57)	0,035 (±0,0032)	0,898 (±0,0160)	0,887 (±0,0173)	0,073 (±0,0068)	2086 (±729)
GRU	2	MAPE	50	No	1	9,96 (±0,51)	1,96 (±0,57)	25,44 (±0,73)	0,036 (±0,0023)	0,888 (±0,0066)	0,880 (±0,0071)	0,075 (±0,0021)	1528 (±237)
LSTM	2	MAPE	50	No	1	9,94 (±0,61)	1,89 (±1,01)	31,67 (±5,01)	0,035 (±0,0043)	0,897 (±0,0118)	0,889 (±0,0127)	0,071 (±0,0029)	2509 (±1229)

Tabla 5.12: Métricas finales para los 10 mejores modelos. Entre paréntesis, se especifica la desviación estándar.

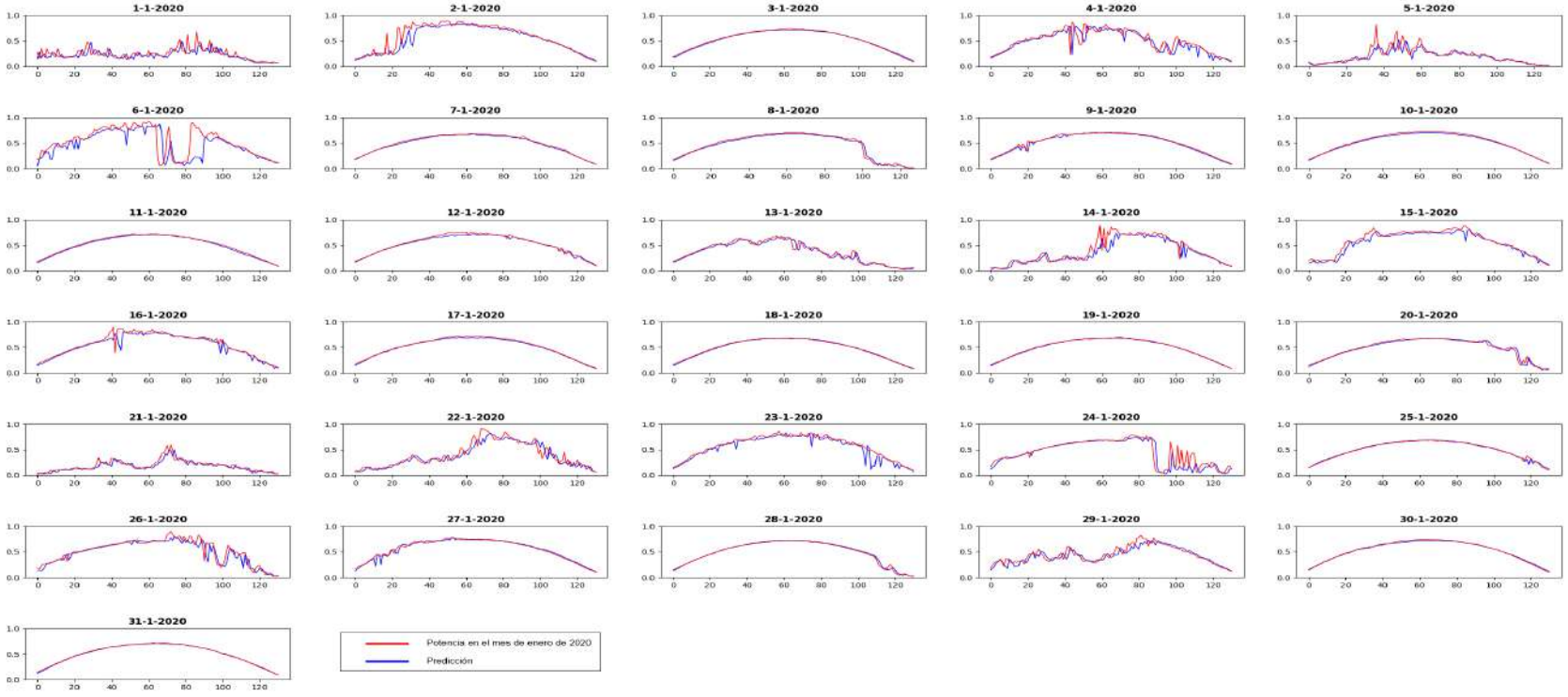


Figura 5.28: Predicciones generadas con el modelo LSTM final.

fueron octubre, julio y abril. A su vez, los meses utilizados para formar parte del conjunto de validación fueron septiembre, junio y marzo, respectivamente.

Dado que no fue posible obtener datos del ICyTE posteriores a enero de 2020, se decidió analizar el período 2016-2019 utilizado hasta el momento. El objetivo fue encontrar años que cuenten con los períodos completos de validación y *testing* seleccionados. En base a la búsqueda realizada, se seleccionó el año 2019 para ser utilizado en las ejecuciones de primavera e invierno. Dado que el 2019 no contaba con datos en el mes de abril, fue necesario recurrir al 2018 para realizar el *testing* otoñal.

Luego de definir los períodos a utilizar, se realizaron las ejecuciones por cada estación. Para cada una de ellas, el modelo no accedió a datos posteriores al mes de *testing* escogido. El entrenamiento fue realizado únicamente con datos previos al mes de validación seleccionado. Los resultados obtenidos se encuentran en la tabla 5.13.

Estación	MAPE promedio	MAPE mín.	MAPE máx.	MAE	R^2	R^2 ajustado	RMSE	Tiempo de ejecución (s)
Primavera	20,43 (±0,53)	6,25 (±0,67)	48,10 (±3,56)	0,069 (±0,0031)	0,810 (±0,0089)	0,792 (±0,0097)	0,112 (±0,0026)	4166 (±1267)
Invierno	26,32 (±1,30)	9,12 (±0,36)	90,67 (±6,68)	0,054 (±0,0017)	0,732 (±0,0091)	0,704 (±0,0101)	0,099 (±0,0017)	2985 (±1202)
Otoño	30,85 (±2,48)	2,37 (±0,35)	108,04 (±36,12)	0,069 (±0,0020)	0,723 (±0,0161)	0,694 (±0,0179)	0,134 (±0,0040)	1897 (±1064)

Tabla 5.13: Métricas obtenidas para el *testing* estacional del modelo producido en el proyecto, con la desviación estándar entre paréntesis.

Como puede observarse, todas las métricas empeoraron al analizar meses en otras estaciones. Los resultados eran esperables, dado que la estación de verano cuenta con una mayor cantidad de días con alta irradiancia, lo cual los hace más sencillos de predecir. A su vez, durante el proyecto se trabajó con procesos de *tuning* enfocados a los meses veraniegos, de manera que había una alta incertidumbre a la hora de analizar los resultados del modelo sobre otros períodos temporales.

Por otra parte, se hace énfasis en el hecho de que los entrenamientos se realizaron con una cantidad de datos menor a la de etapas anteriores. Por ejemplo, para la estación con la cual se obtuvieron los peores resultados (otoño), se debieron remover 65012 registros, lo cual representó un 42 % menos de datos respecto al proceso de aprendizaje para las pruebas de verano.

A futuro, se sugiere un estudio de mayor profundidad sobre los resultados estacionales, dado que existe la posibilidad de que otras arquitecturas o hiperparámetros maximicen la *performance* en las estaciones mencionadas.

5.4.6. Conclusión de LSTM y GRU

La experimentación sobre modelos de *Deep Learning* fue exitosa. Se analizaron las arquitecturas LSTM y GRU, junto con variantes como los modelos bidireccionales y los mecanismos de atención. En primera instancia, se observó que los resultados obtenidos fueron superiores a los reportados durante las pruebas con NARX y SVR. Por este motivo, se prosiguió en la implementación y ejecución de procesos de *tuning* automatizados.

La automatización y el uso de entornos en la nube permitió la ejecución de una alta cantidad de pruebas, donde el objetivo fue la obtención del conjunto de hiperparámetros óptimo. A partir de la incorporación de nuevos períodos de datos, se logró obtener el modelo LSTM que minimizó el error obtenido en la totalidad de pruebas ejecutadas con cada uno de los modelos analizados (ver tabla 5.14). Adicionalmente, se logró evaluar las capacidades estacionales al utilizarlo para predecir meses en diferentes estaciones.

5.5. Comparativa y análisis de resultados

Luego de haber encontrado el modelo predictivo con el cual se obtuvieron los niveles más bajos de error, se consideró de importancia evaluarlo contra otros resultados obtenidos en la literatura. A su vez, se investigó acerca de la existencia de bases de datos similares a la utilizada durante el proyecto. El objetivo fue contar con nuevos conjuntos de datos sobre los cuales evaluar el funcionamiento del modelo.

5.5.1. Bases de datos externas

En primer lugar, se realizó una búsqueda de bases de datos públicas con características similares a la utilizada. Se requirió que estén incluidas las variables existentes en la base de datos provista por el

Modelo	MAPE promedio diario
NARX 5.3.5	14,81 %
SVR B.1	11,04 %
LSTM 5.12	9,46 %
Bi-LSTM 5.12	9,67 %
LSTM con atención 5.12	11,13 %
GRU 5.12	9,58 %

Tabla 5.14: Mejor resultado obtenido en la predicción de cada día del mes enero del 2020, en base a la arquitectura. Para los modelos NARX y SVR, el resultado reportado es de una única ejecución.

ICyTE y que la resolución de los datos almacenados sea como mínimo de 5 minutos. Estos requisitos se establecieron para lograr una comparación fidedigna a las pruebas realizadas a lo largo del proyecto.

Se tenía conocimiento de antemano sobre la dificultad existente en encontrar conjuntos de datos que cumplan con dichas condiciones y que sean provistos para uso libre. Aún con tales dificultades presentes, se investigó al respecto y los mejores resultados obtenidos fueron los siguientes:

- **Base de datos de competición:** en la plataforma Kaggle se encontró un conjunto de datos asociado a observaciones de potencia generada entre el 1/7/2010 y el 1/7/2013. Contenía 16 variables distintas, incluyendo las requeridas. Sin embargo, no fue posible su uso ya que pertenecía a una competición de participación limitada y los archivos únicamente podían ser accedidos por usuarios invitados. Tampoco se logró obtener la información de contacto del usuario organizador, dado que se trataba de una competición antigua.
- **Sistema geográfico de información fotovoltaica de la Comisión Europea:** esta herramienta contiene bases de datos de potencia generada para casi todos los países del mundo. Su última actualización fue el 1/3/2022, pero contiene información desde el año 2005. Las bases de datos se pueden descargar de forma libre y también contienen todas las variables requeridas. El problema hallado fue que los datos contaban con una resolución de 1 hora. Esto implicaba un problema de compatibilidad con aquella utilizada durante el desarrollo del proyecto.
- **Bases de datos *open-source* de energía fotovoltaica:** se encontró un listado de diferentes bases de datos *open-source* provistas por entidades, empresas y usuarios particulares. Cada una de ellas provenía de diferentes países del mundo. Se encontraron 9 en las cuales la resolución de los datos era cada 1 o 5 minutos. Sin embargo, con todas hubo problemas tanto de acceso como de falta de compatibilidad en cuanto a las variables climáticas disponibles. Muchas de las bases de datos tampoco contaban con la irradiancia, la variable de mayor correlación del problema.

Debido a la inexistencia de bases de datos externas para la realización de pruebas, se decidió proceder con la comparación de resultados en publicaciones relacionadas a la predicción de potencia eléctrica.

5.5.2. Comparativa con publicaciones

Al proceso de generación de potencia eléctrica se le suele asociar una gran incertidumbre debido a varios factores climáticos, entre ellos la humedad [120]. Al ser una ciudad costera, Mar del Plata cuenta con altos niveles de humedad (aproximadamente un 80 % de humedad anual promedio [121]). Es por ello que se decidió buscar publicaciones en donde se analicen ciudades con una humedad relativa mensual similar. De esta forma, fue posible presentar un análisis acorde a las condiciones climáticas de la planta fotovoltaica de la Facultad de Ingeniería.

5.5.2.1. Publicación relacionada a la planta fotovoltaica de la Facultad de Ingeniería

La primera comparativa fue realizada sobre el trabajo *"A Photovoltaic Generation Forecasting using Convolutional and Recurrent Neural Networks"* del año 2023 [19]. La publicación resultó de profundo interés, dado que se báso en la planta fotovoltaica del ICyTE y utilizó la misma base de datos que fue

suministrada para el presente proyecto. Algunas de las características principales expuestas en el trabajo se detallan a continuación:

- El rango de los datos fue acotado desde el 01/01/2019 al 31/01/2020.
- Las variables incluidas fueron irradiancia, humedad, temperatura del aire y velocidad del viento.
- Se utilizó un tamaño de secuencia de 130, con resolución de 5 minutos.
- Se analizaron modelos LSTM y GRU junto con la inclusión de preprocesamiento de datos a través de redes convolucionales (CNN, no desarrolladas en el proyecto).
- Para la etapa de *testing*, se realizaron 10 ejecuciones independientes por modelo.
- Se utilizó un horizonte temporal de 1, 2, 3 y 4 días. La comparativa se realizó para 1 día, el mismo utilizado en el presente proyecto.
- Los valores más bajos para las métricas de error se obtuvieron con un modelo CNN-GRU de 2 capas.

Modelo	MAPE	RMSE	MAE	R^2
CNN-GRU de 2 capas (Babalhavaeji, *et al.*, 2023 [19])	10,15 % (+- 1,30 %)	0,093 (+- 0,017)	0,076 (+- 0,015)	0,93 (+- 0,028)
LSTM de 3 capas (modelo propuesto)	9,46 % (+- 0,20 %)	0,069 (+- 0,0017)	0,032 (+- 0,0012)	0,903 (+- 0,0048)

Tabla 5.15: Comparativa entre los resultados obtenidos en el proyecto y la publicación *"A Photovoltaic Generation Forecasting using Convolutional and Recurrent Neural Networks"* [19].

Tal como se observa en la tabla 5.15, el proyecto consiguió disminuir el nivel de error obtenido en diversas métricas de la publicación mencionada. Particularmente, las reducciones fueron del 6,80 % para el MAPE, 25,81 % para el RMSE y 57,89 % para el MAE. En contraposición al resto de métricas, el valor de R^2 empeoró en un 2,90 %.

En base a los resultados expuestos, es posible concluir que, con un modelo menos complejo que una red convolucional, se lograron obtener métricas que mejoraron a las de la publicación analizada. La única excepción fue R^2, aunque con un degradamiento menor en comparación a las métricas mejoradas.

5.5.2.2. Publicaciones externas

Por otra parte, con el objetivo de documentar escenarios similares a este estudio y los resultados obtenidos, se recopilaron diversos trabajos adicionales desarrollados en zonas donde la humedad media anual se mantiene cercana a la de la ciudad de Mar del Plata. Se realizaron comparaciones únicamente con sus valores de MAPE, ya que las plantas fotovoltaicas y sus bases de datos difieren en gran medida con la utilizada en este proyecto. Los trabajos analizados junto con sus características, modelos utilizados y MAPE promedio diario se listan a continuación:

1. ***Forecasting Solar Power Using Long-Short Term Memory and Convolutional Neural Networks* [7]:** se investigaron diversas ciudades de Corea del Sur. Se utilizaron datos entre 2012 y 2016, de los cuales un 25 % fueron datos de *testing* (aproximadamente un año). Las variables de entrada fueron fecha, hora, temperatura del aire, velocidad del viento y humedad, entre otras. Para los modelos LSTM, se ingresaron secuencias de 1 a 6 horas. El horizonte temporal fue de 24 horas. A continuación, se expresa el MAPE promedio diario para cada arquitectura:

   a) **SVR con RBF como *kernel*:** 40,03 %.

   b) **LSTM con *autoencoder* (secuencia de 1 hora):** 22,2 %.

   c) **LSTM convolucional (secuencia de 1 hora):** 13,42 %.

2. ***Solar Power Forecasting Using Deep Learning Techniques* [8]:** se analizó una planta fotovoltaica de Halifax, Canadá, cuya humedad relativa anual es similar al caso de Mar del Plata. Se recopilaron datos entre las 8 y 17 h del año 2017. Se utilizó una resolución de 30 minutos y se realizaron predicciones sobre un día de verano (30/06/17). No existe información sobre el tamaño de secuencia y el horizonte temporal utilizado es de 24 horas. Se obtuvieron los siguientes MAPE:

   a) **LSTM:** 27,5 %.
   b) **MLP:** 75,5 %.

3. ***Ultra-short-term PV prediction based on LSTM with a multi-head attention mechanism*** **[88]:** se recopilaron datos de una planta en Hangzhou, China desde el 15 de enero al 15 de septiembre del año 2022. Su resolución es de 15 minutos. El *testing* del modelo se realizó en los meses de agosto y septiembre, por lo que la estación fue el verano. El horizonte temporal es de 4 horas. A continuación, se listan los MAPE promedio diario de la publicación:

   a) **LSTM:** 47 %.
   b) **LSTM con mecanismo de atención:** 38 %.
   c) **LSTM con mecanismo *multi-head* de atención:** 23 %.

4. ***Short-Term Photovoltaic Power Forecasting Based on Long Short Term Memory Neural Network and Attention Mechanism*** **[2]:** se utilizaron datos desde octubre de 2014 a septiembre de 2018 de Shaoxing, China. Su resolución fue de 7,5 minutos. Se utilizó el período de octubre de 2017 a septiembre de 2018 para *testing*. Utilizó como variables de entrada la potencia y la temperatura. No se especificó el tamaño de ventanas temporales. El horizonte temporal definido fue de 60 minutos. Los MAPE obtenidos fueron:

   a) **ARIMA:** 49,11 %.
      1) **Otoño:** 52,6 %.
      2) **Invierno:** 49,8 %.
      3) **Primavera:** 49,56 %.
      4) **Verano:** 44,49 %.
   b) **MLP:** 41,64 %.
      1) **Otoño:** 40,36 %.
      2) **Invierno:** 47 %.
      3) **Primavera:** 39,41 %.
      4) **Verano:** 39,78 %.
   c) **LSTM:** 40,12 %.
      1) **Otoño:** 38,27 %.
      2) **Invierno:** 43,3 %.
      3) **Primavera:** 38,92 %.
      4) **Verano:** 39,98 %.
   d) **LSTM con un mecanismo de atención:** 37,82 %.
      1) **Otoño:** 36,6 %.
      2) **Invierno:** 40,47 %.
      3) **Primavera:** 38,12 %.
      4) **Verano:** 36,1 %.

5. ***Solar PV Power Forecasting Using Modified SVR with Gauss-Newton Method*** **[122]:** se utilizaron datos de Brisbane, Australia desde las 8 a 18 h de un mes. Se escogieron días aleatorios de diferentes condiciones climáticas del mismo mes del año siguiente para *testing*. Las variables de entrada utilizadas fueron irradiancia y potencia. La resolución de los datos fue de 1 hora, mientras que el horizonte temporal fue de 24 horas. A continuación, se listan los resultados obtenidos:

   a) **SVR:** 9,89 %.
      1) **Día soleado:** 8,56 %.
      2) **Día parcialmente nublado:** 12,3 %.
      3) **Día mayormente nublado:** 8,82 %.
   b) **SVR modificado con un método Gauss-Newton:** 9,25 %.
      1) **Día soleado:** 8,35 %.

2) **Día parcialmente nublado:** 10,84 %.
3) **Día mayormente nublado:** 8,56 %.

Luego de observar los resultados publicados para escenarios similares a los de este proyecto, se puede afirmar que los resultados obtenidos en el trabajo se encuentran dentro de lo presentado en la literatura para el estado del arte.

5.5.3. Conclusión acerca de la comparativa y el análisis de resultados

Finalmente, se demostró que el modelo propuesto mejoró los resultados de una publicación basada en la planta fotovoltaica del ICyTE. El trabajo referenciado fue publicado en un congreso europeo del presente año, lo que demuestra que el modelo desarrollado se encuentra a la altura del estado del arte para la predicción de la potencia eléctrica. Se obtuvo una mejora en el valor de MAPE del 6,8 %, un 25,81 % para el RMSE y un 57,89 % para el MAE. La única excepción fue el R^2, donde el resultado de la publicación fue un 2,9 % mejor. Adicionalmente, se destaca que se logró disminuir la complejidad del modelo predictivo al no utilizar redes convolucionales.

En cuanto a los trabajos externos, las comparativas resultaron dificultosas, dado que las bases de datos aplicadas fueron distintas. Sin embargo, se observó que los resultados obtenidos en el presente proyecto se asemejan a los de la literatura para ciudades con condiciones climáticas similares a Mar del Plata. Nuevamente, esto demuestra que el modelo propuesto está al nivel del estado del arte.

Capítulo 6

Productos

6.1. Software predictivo y de análisis de datos

Durante las primeras fases del proyecto, surgió la necesidad de contar con una herramienta que facilite el análisis sobre las mediciones registradas. El objetivo principal del software fue la visualización gráfica de las series temporales presentes en la base de datos del ICyTE.

A través de la utilización de la herramienta, se propuso analizar períodos de tiempo acotados dentro del total de datos almacenados, así como visualizar la evolución de las variables durante el lapso seleccionado. A partir de su construcción, se facilitaron las tareas de selección de datos para el entrenamiento de los modelos.

En un primer momento, la herramienta fue concebida para uso interno al proyecto. Sin embargo, ante el crecimiento de sus funcionalidades y la utilidad demostrada, se decidió ofrecerla como un producto adicional.

En cuanto a la funcionalidad predictiva, fue necesario desarrollar una herramienta que permitiese una sencilla operación del modelo obtenido en la etapa final. Además de ofrecer acceso a la predicción, se requería poder realizar ejecuciones con datos de entrenamiento y *testing* a elección.

6.1.1. Tecnologías aplicadas

Se optó por utilizar el lenguaje de programación Python para el desarrollo del software. Uno de los motivos de su elección fue que cuenta con una de los paquetes más completos para la manipulación de tablas y series temporales: Pandas [123]. Debido a la limitada experiencia en el uso de la librería, resultó de crucial importancia la amplia cantidad de documentación y soporte disponible sobre ella. Adicionalmente, el uso de Python permitió utilizar la librería matemática Numpy [124], que fue de extrema utilidad para el armado de los conjuntos de datos.

Para el desarrollo de la interfaz gráfica, se realizó una investigación sobre los distintos paquetes disponibles para Python. Ante la carencia de experiencia previa en el desarrollo de interfaces con dicho lenguaje, la búsqueda se centró en una librería sencilla y rápida de aprender. Las principales candidatas fueron PyQt5 [125] y Tkinter [126], las más populares dentro de su área. Si bien ambas contaban con grandes capacidades y numerosos recursos para el aprendizaje, se optó por Tkinter debido a que su simplicidad ofrecía una curva de aprendizaje corta.

Adicionalmente, se utilizaron diversas librerías que fueron mencionadas a lo largo del informe. Entre ellas, se encuentran Keras para la construcción del modelo, Matplotlib para la generación de gráficos. A su vez, fue de extrema utilidad la biblioteca scikit-learn para el cálculo de ciertas métricas.

Cabe aclarar que todas las librerías seleccionadas para la construcción del software son de código abierto.

6.1.2. Requerimientos esperados

Los requerimientos del software predictivo fueron establecidos a comienzos del proyecto por el Dr. Ing. S. A. González del ICyTE. En cuanto a la funcionalidad de análisis, todo el equipo de trabajo se volvió partícipe de la definición de nuevas funcionalidades. La fase de experimentación motivó constantemente el desarrollo de mejoras sobre la herramienta, dado que su uso fue importante para facilitar la selección y el análisis de datos.

Uno de los requerimientos generales fue que tanto la herramienta de análisis como el software predictivo coexistan en una misma interfaz visual, de manera que se facilite su acceso (ver figuras 6.1 y 6.2). En las secciones posteriores, se especifican los requerimientos específicos de cada funcionalidad.

6.1.2.1. Software de análisis de datos

1. **Filtrado de datos**: debía ser posible filtrar las mediciones en base a un período de días u horas. Por ejemplo: obtener los datos referentes al mes de enero del año 2020, entre las 16:00 y 18:00 h. De esta manera, podría analizarse con mayor facilidad un período de tiempo específico.

2. **Elaboración de gráficos**:

   a) **Evolución de las series temporales**: el sistema debía realizar gráficos con la evolución temporal de las siete variables almacenadas en la base de datos. Se debía contar con dos tipos de evoluciones: individual (medición a medición) y promediada (por hora y día).

   b) **Diagramas de caja**: su finalidad fue la representación visual de la distribución de los datos, de manera que pueda detectarse su agrupamiento y los valores atípicos de cada medición.

   c) **Correlación**: se debían construir gráficos que muestren la relación entre cada una de las variables de entrada y la potencia eléctrica generada. A través de la correlación calculada, sería posible detectar aquellas que debían contar con mayor influencia dentro del modelo. Posteriormente, se planteó la posibilidad de visualizar el valor del resto de mediciones asociadas a cada punto del gráfico. Para ello, el usuario debía ubicar el cursor sobre el punto que deseara consultar. De esta manera, podría asociarse rápidamente un punto con sus valores en el resto de series temporales.

   d) **Valores atípicos**: se debía representar la evolución temporal de cada variable y resaltar los valores alejados a un número de desviaciones estándar especificado por el usuario.

   e) **Gráficos de ensamble**: su objetivo fue exponer la superposición diaria de las series temporales asociadas a una medición. De esta manera, se permitiría una rápida comparación entre la evolución de las variables durante un conjunto de días.

3. **Almacenamiento de los datos procesados**: el software debía ser capaz de almacenar los datos filtrados, junto con un cuadro descriptivo de cada una de las mediciones. La descripción debía especificar (para el período seleccionado) su media, valor máximo y mínimo, varianza y desviación estándar, entre otros valores.

4. **Interfaz de usuario**: el sistema debía contar con una interfaz sencilla, a través de la cual podrían especificarse los datos con los cuales realizar el filtrado y los gráficos requeridos. En primer lugar, la interacción con el usuario se realizó por medio de línea de comandos. Posteriormente, con el objetivo de aumentar la usabilidad de la herramienta, se construyó una interfaz visual.

6.1.2.2. Software predictivo

1. **Almacenamiento del modelo:** el sistema debía contemplar el almacenamiento de un archivo que contenga la arquitectura e hiperparámetros utilizados en el modelo final obtenido.

2. **Configuración de la etapa de entrenamiento:** el software debía ser capaz de presentar al usuario los archivos disponibles para utilizar durante la etapa de entrenamiento del modelo. A su vez, se debía permitir el ingreso de una fecha de inicio para la etapa de validación. El conjunto de entrenamiento debía conformarse por las mediciones del archivo anteriores a la fecha introducida, mientras que el conjunto de validación se formaría por el resto de datos.

3. **Configuración de la etapa de *testing*:** el sistema debía presentar al usuario los archivos disponibles para ser utilizados en la etapa de *testing*. A su vez, si el usuario así lo deseara, se le permitiría ingresar las fechas de inicio y fin para la etapa. En caso de no ingresar dicha información, se utilizaría el archivo completo. Para permitir el inicio de esta etapa, debía ser necesario haber completado la etapa de entrenamiento.

4. **Elaboración de gráficos:** se requería que el sistema fuera capaz de elaborar gráficos en donde se observen los valores de potencia eléctrica predichos para los días determinados por el usuario en la etapa de *testing* del modelo. Los gráficos para múltiples días debían organizarse en un formato de

grilla para mejorar la visualización. A su vez, se debía poder visualizar la evolución de los errores de entrenamiento y validación durante la fase de aprendizaje.

5. **Cálculo de métricas:** el sistema debía ser capaz de calcular diversas métricas, tanto en la etapa de entrenamiento como en la de *testing*. Para cada predicción, se debía determinar el MAE, MAPE promedio diario, día con MAPE mínimo, día con MAPE máximo, R^2, R^2 ajustado, RMSE y tiempo de ejecución.

6. **Interfaz de usuario:** el sistema debía contar con una interfaz visual sencilla en donde el usuario fuese capaz de ingresar toda la información necesaria para completar cada una de las etapas mencionadas.

7. **Exportación de resultados:** el sistema debía ser capaz de exportar los resultados de las predicciones y las métricas obtenidas en el proceso predictivo.

8. **Uso de aceleradores:** en casos donde el entorno de ejecución contara con un acelerador tipo TPU, el modelo debía ser capaz de utilizar sus capacidades para las fases de entrenamiento y *testing*.

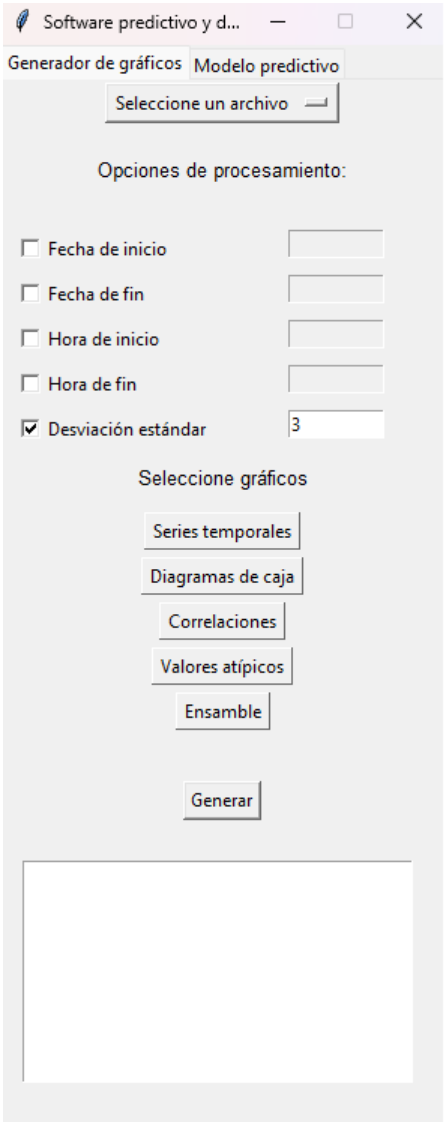


Figura 6.1: Interfaz de usuario del software de análisis de datos.

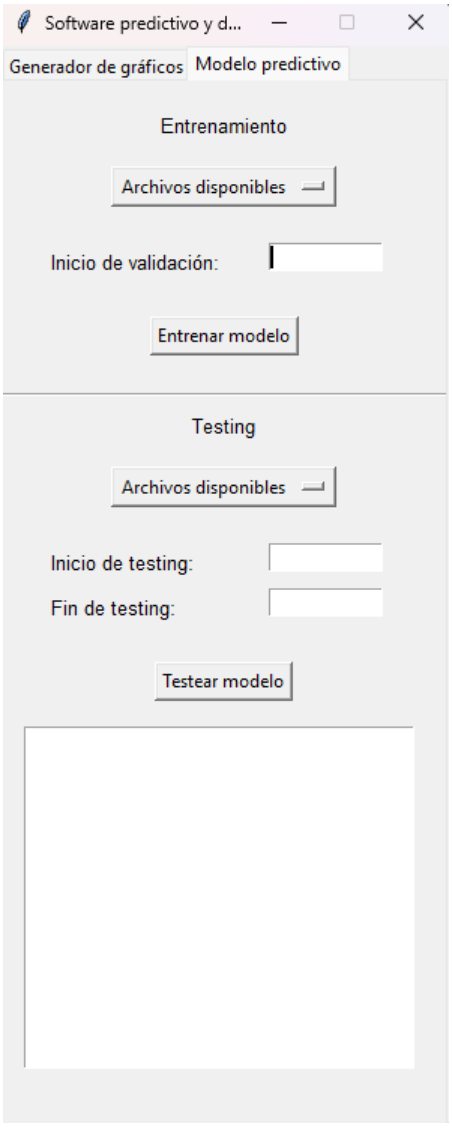


Figura 6.2: Interfaz de usuario del software predictivo.

6.1.3. Arquitectura

La arquitectura del software fue diseñada a partir de módulos con responsabilidades limitadas. A continuación, se especifica la función de cada uno de ellos:

- **Interfaz:** actúa como el mediador entre el usuario y las funcionalidades del sistema. Permite la configuración de las facetas predictivas y de análisis de datos.
- **Predicción:** gestiona la ejecución de los módulos asociados a las funcionalidades de predicción de potencia.
  - **Evaluador:** calcula las métricas de *performance* del modelo. A su vez, construye un gráfico con la evolución diaria de la potencia eléctrica para cada día solicitado.
  - **Exportador:** exporta los resultados obtenidos, junto con las métricas calculadas para la etapa de *testing*.
  - **Modelo:** gestiona la construcción, el entrenamiento y la generación de pronósticos a partir del modelo LSTM configurado. De estar disponible, utiliza un acelerador TPU durante las ejecuciones. Adicionalmente, produce gráficos con la evolución de los errores de entrenamiento y validación.
  - ***Datasets*:** se encarga de construir los conjuntos de datos de entrenamiento, validación y *testing*.
- **Procesamiento:** encargado de gestionar las funcionalidades de análisis de datos.
  - ***DataFrames*:** realiza el filtrado de datos en base a la fecha y hora introducidas por el usuario. A su vez, se encarga de producir el cuadro descriptivo del período seleccionado.
  - **Graficador:** responsable de la generación de los gráficos de análisis.

Las principales dependencias entre cada uno de los módulos pueden visualizarse en la figura 6.3.

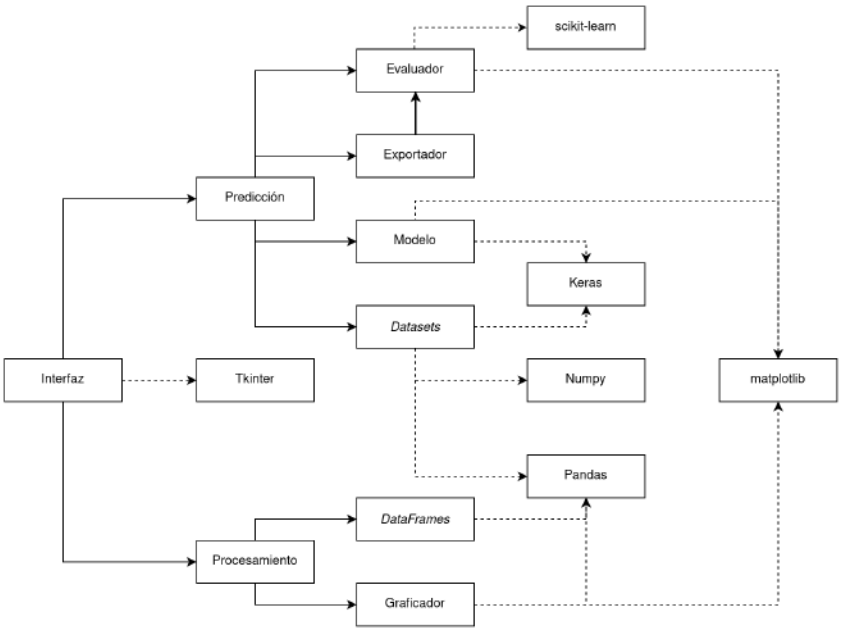


Figura 6.3: Grafo de dependencias para cada uno de los módulos del software. Las líneas sólidas representan dependencias a módulos propios, mientras que las punteadas son hacia librerías externas.

6.1.3.1. Funcionalidades adicionales

Además de los módulos anteriormente mencionados, se desarrollaron funcionalidades de soporte para facilitar el uso de la herramienta. En primer lugar, se estableció el uso de *logs* para registrar las excepciones no contempladas. De esta manera, será sencillo identificar problemas futuros en caso de que surjan errores

no detectados durante el proceso de desarrollo. Adicionalmente, también se registran las métricas de error para cada *batch* durante las etapas de entrenamiento.

Como utilidad complementaria a la herramienta, se incluye un *script* desarrollado durante las fases de experimentación, que permite eliminar los valores nulos de un archivo de datos, así como generar una entrada adicional para la potencia eléctrica del día anterior de cada instante temporal. El *script* será de utilidad en caso de que el ICyTE haga uso de archivos de datos anteriores o posteriores a los utilizados durante el proyecto. Dado que su ejecución es costosa y solo es necesaria una única vez, se decidió mantenerlo como funcionalidad separada al software.

6.1.4. *Testing*

El objetivo del proceso de *testing* fue validar el correcto funcionamiento de la herramienta. Bajo esta premisa, se decidió desarrollar un conjunto de pruebas automatizadas, las cuales permitieron cubrir un alto porcentaje de las características del software. Como ventaja sobre el *testing* manual, la automatización permitió realizar cambios sobre el código sin necesidad de realizar pruebas reiterativas sobre funcionalidades ya desarrolladas.

Para las características granulares, se desarrollaron pruebas unitarias a través del *framework* unittest [127]. Las pruebas evaluaron todos los posibles escenarios de las siguientes operaciones principales:

- Lectura de los archivos de datos y eliminación de valores nulos.
- Conversión de valores tipo *datenum* a fechas manejables por Pandas.
- Filtrado de datos por día, hora, minuto y segundo.
- Generación de estructuras de datos de tipo *DataFrame* para el manejo de conjuntos de entrenamiento, validación y *testing*.
- Exportación de resultados a archivos.
- Generación de una estructura de tipo grilla para la presentación de los resultados de las predicciones de potencia.
- Cálculo de métricas.

Posteriormente, se realizaron *tests* de integración, cuyo objetivo fue corroborar la interacción entre módulos. Para estas pruebas, se volvió a utilizar el *framework* unittest. A su vez, se incorporó el uso de *mocks*, clases con funcionalidad limitada que permiten evitar el uso de otras de mayor complejidad a la hora de testear. Por ejemplo, se crearon *mocks* sobre la librería matplotlib, de manera que los gráficos no se muestren por pantalla a la hora de efectuar las pruebas.

Los *tests* de integración incluyeron la automatización de ejecuciones completas de la herramienta, con excepción de la interfaz de usuario. Cada prueba analizó que el flujo de ejecución sea acorde a lo introducido. Por ejemplo, que únicamente se generen los gráficos que fueron solicitados.

A partir de los *tests* unitarios y de integración, se logró cubrir un 95 % del código del software, sin contar la interfaz de usuario ni las funcionalidades de despliegue de gráficos. Si se la contabiliza, el porcentaje se reduce al 60 %. El reporte completo, generado mediante el paquete Coverage.py [128], puede visualizarse en la figura 6.4.

Si bien se priorizó el armado de pruebas automatizadas, ciertas funcionalidades fueron difíciles de testear por esta vía. Algunos ejemplos son la interfaz de usuario y la correcta visualización de los gráficos. Al depender íntegramente de librerías externas, su testeo habría requerido una gran dedicación de horas de trabajo. Por este motivo, se decidió que estas funcionalidades fueran testeadas únicamente de forma manual.

6.1.5. Configuración y documentación

El software cuenta con tres archivos de configuración de tipo JSON que son utilizados para establecer valores a diferentes variables utilizadas por el sistema. Se encuentran en el directorio *"./config"* del proyecto. A continuación, se detalla cada uno de ellos y su utilidad:

- **modelo_config.json:** contiene la estructura de la red y los valores para la cantidad de unidades y el *dropout* recurrente de cada capa. Actualmente, contiene los valores correspondientes al mejor modelo obtenido durante el proyecto.

Coverage report: 95%

coverage.py v7.3.1, created at 2023-09-24 20:33 -0300

Module	statements	missing	excluded	coverage
forecasting\datasets.py	44	0	0	100%
forecasting\evaluator.py	72	0	0	100%
forecasting\exception\forecast_exception.py	2	0	0	100%
forecasting\exporter.py	25	0	0	100%
forecasting\handler.py	41	0	0	100%
forecasting\model.py	41	9	0	78%
forecasting\plotter.py	15	13	0	13%
forecasting\tpu\strategy_generator.py	10	3	0	70%
forecasting\utils\grid.py	11	0	0	100%
forecasting\utils\metrics.py	6	0	0	100%
processing\dataframe\utils.py	13	0	0	100%
processing\dataset_processing.py	26	0	0	100%
processing\plots\constants.py	6	0	0	100%
processing\plots\plots.py	209	3	0	99%
utils\constants\config\model.py	4	0	0	100%
utils\constants\config\training.py	8	0	0	100%
utils\constants\dataset.py	13	0	0	100%
utils\constants\files.py	14	0	0	100%
utils\constants\values.py	3	0	0	100%
utils\csv.py	3	0	0	100%
utils\datasets.py	5	0	0	100%
utils\datenum.py	3	0	0	100%
utils\loader.py	11	0	0	100%
Total	585	28	0	95%

Figura 6.4: Reporte de cobertura generado por Coverage.py.

- **graficos_config.json:** posee las variables relacionadas a la elaboración de la grilla de gráficos de las predicciones.
- **entrenamiento_config.json:** contiene todas las variables relacionadas al armado de las secuencias de entrada al modelo. Se incluye la posibilidad de modificar el tamaño de *batch*, la tasa de muestreo, la longitud de secuencia y la cantidad de pasos adelante de la predicción.

Adicionalmente, el software cuenta con un archivo de documentación donde se especifica la modalidad de uso, el formato para la introducción de fechas y todas las opciones de configuración disponibles.

6.1.6. Conclusión

Finalmente, se logró desarrollar de forma exitosa una herramienta de software con capacidades analíticas y predictivas.

El software de análisis fue de suma utilidad durante el transcurso del proyecto, por lo que cumplió con su objetivo inicial. Su capacidad para visualizar la evolución de las mediciones y sus valores atípicos fue muy beneficiosa para la elección de los períodos de entrenamiento y *testing* de los modelos. A su vez, el análisis de correlaciones demostró su utilidad durante las tareas relacionadas al agregado de nuevas variables. A modo de ejemplo, ambas funcionalidades fueron utilizadas durante la predicción de un día con el modelo NARX (ver sección 5.3.3).

En cuanto a la faceta predictiva, se considera que la herramienta será de gran utilidad para el personal del ICyTE. A partir de su uso, es posible ejecutar las fases de entrenamiento y *testing* de los períodos especificados. A su vez, se ofrece la posibilidad de manipular los hiperparámetros y la arquitectura del modelo, de manera que sea posible realizar experimentación adicional.

6.2. Estudio del estado del arte

Durante el proyecto, se llevó a cabo un extenso estudio acerca del estado del arte en la predicción de potencia eléctrica.

En primer lugar, se realizó una investigación acerca de los modelos conexionistas y de *Deep Learning* relevantes en la resolución del problema. En la fase de experimentación se analizaron las diferentes

librerías y entornos disponibles para la construcción y ejecución de modelos predictivos. A su vez, se realizaron pruebas exhaustivas sobre distintas arquitecturas y sus conjuntos de hiperparámetros. Finalmente, se incluyó una comparativa con trabajos de relevancia asociados a ciudades con condiciones climáticas similares a las de Mar del Plata.

El estudio del estado del arte y las conclusiones extraídas en el proyecto conforman un producto de gran utilidad para trabajos futuros basados en la problemática.

6.3. Optimización de la base de datos

La experimentación práctica requirió de un uso extensivo de la base de datos provista por el ICyTE. Si bien la información había sido previamente depurada y se encontraba en un estado consistente, se localizaron y eliminaron períodos con mediciones incompletas o valores atípicos. Como producto adicional del proyecto, se incluye la base de datos final utilizada para el entrenamiento de los modelos, que también incluye la potencia eléctrica del día anterior como columna adicional.

6.4. Conclusión

El proyecto cumplió con los objetivos establecidos y entregó como resultado final una herramienta de software para el análisis de períodos de datos y la predicción de potencia eléctrica. A su vez, se destacan los productos adicionales referidos al estudio del estado de arte y la optimización de la base de datos, que son de utilidad para trabajos futuros vinculados al problema.

Capítulo 7

Memoria del proyecto

En el presente capítulo, se reflexiona acerca del grado de cumplimiento de los objetivos establecidos. A su vez, se analiza una de las competencias de mayor importancia a desarrollar en la asignatura de Trabajo Final: la gestión de proyectos. Se evalúa el grado de exactitud de la planificación original, los desvíos presentados y las herramientas colaborativas utilizadas.

7.1. Cumplimiento de objetivos

7.1.1. Objetivo global

Construir un modelo predictivo de *Deep Learning*, que permita aproximar la potencia eléctrica que generará una planta fotovoltaica. Obtener predicciones superiores a las del ICyTE, y cercanas o mejores a las de la literatura. A su vez, ofrecer un software que permita utilizar el modelo, acceder a sus resultados y realizar un análisis de la base de datos.

Se logró implementar un modelo LSTM capaz de realizar predicciones precisas sobre la potencia eléctrica de la planta fotovoltaica situada en la Facultad de Ingeniería. Tal como se presentó en la sección 5.5.2, los resultados obtenidos fueron superiores al modelo de mejor *performance* con el cual cuenta el ICyTE. A su vez, se observó que los niveles de error se asemejaron a aquellos encontrados en la literatura para ciudades con condiciones climáticas similares a Mar del Plata.

Para su utilización, se construyó un software capaz de administrar las etapas de entrenamiento, validación y *testing*. Adicionalmente, los resultados obtenidos pueden ser visualizados a través de las métricas y los gráficos de evolución diaria de la potencia. Además de la funcionalidad predictiva, se desarrollaron componentes de análisis para facilitar el estudio de la base de datos. La herramienta fue de suma utilidad durante el proyecto y se considera que aportará valor a las actividades futuras del ICyTE.

7.1.2. Objetivos específicos

Adquirir las competencias necesarias para la gestión de un proyecto que incorpora tareas de investigación, experimentación y desarrollo de software.

Al ser un proyecto enmarcado dentro de numerosas áreas, se presentaron desafíos muy interesantes respecto a la gestión. Un ejemplo fueron las decisiones para determinar la profundidad de la investigación sobre cada arquitectura. El área del *Deep Learning* es extremadamente amplia y se encuentra en constante crecimiento, por lo que la selección de variantes de estudio representó un reto durante la planificación y experimentación. Dado que se debió priorizar un conjunto pequeño de modelos, se debieron tomar decisiones justificadas por los resultados obtenidos y por aquellos observados en la literatura. En retrospectiva, se considera que las determinaciones realizadas fueron acertadas.

Si bien existieron desvíos por fuera de lo planificado originalmente, la experiencia fue de gran utilidad para la adquisición de las competencias esperadas en la asignatura. El proyecto fue una instancia única en la carrera. Por este motivo, se considera que la experiencia obtenida será de un alto valor a lo largo del futuro profesional como ingenieros.

Analizar publicaciones y bibliografía relacionadas a la temática, con el objetivo de adquirir conocimiento abarcativo en el área del aprendizaje profundo.

Al ser un área no tratada en la carrera, la inmersión sobre los modelos conexionistas y el *Deep Learning* fue un desafío adicional. Por este motivo, la lectura de bibliografía y la experimentación fueron de suma utilidad para la comprensión acerca de la temática. Esto fue complementado con clases explicativas dadas por la directora del proyecto, especialmente durante la primera etapa.

A lo largo de las fases del proyecto, se logró adquirir conocimiento teórico, técnico y metodológico acerca de un tópico de alta complejidad y con profunda relevancia en la actualidad. Adicionalmente, fue necesario buscar y analizar publicaciones científicas vinculadas al estado del arte del problema desarrollado. Estas actividades no se habían realizado anteriormente en la carrera, por lo que nutrieron competencias adicionales vinculadas a tareas de investigación.

Investigar acerca del estado del arte relacionado al software que se utiliza para trabajar con modelos predictivos.

La confección del trabajo permitió el acercamiento a diferentes lenguajes, librerías, herramientas y entornos de desarrollo que no habían sido utilizados anteriormente y que son de amplia relevancia en la temática del aprendizaje profundo.

Además de adquirir conocimiento acerca de su funcionamiento, existieron etapas donde fue necesario evaluar diferentes alternativas y analizar sus ventajas y desventajas. Se considera que las decisiones tomadas fueron acertadas y que el análisis permitió profundizar adecuadamente sobre cada tecnología.

Realizar una puesta a punto de la base de datos otorgada por el ICyTE. Detectar los parámetros de mayor influencia en la generación de potencia eléctrica.

En primer lugar, se realizó un análisis acerca del grado de correlación entre las mediciones almacenadas y la potencia eléctrica. Tal como se demostró en la sección 5.1.3.2, la irradiancia es la variable de mayor importancia a la hora de realizar las predicciones. Sin embargo, a lo largo de las fases de experimentación, se observó la utilidad de cada una de las mediciones almacenadas.

El modelo LSTM propuesto utiliza como entradas a 6 de las 7 variables presentes en la base de datos. La excepción es *datenum*, que de todas formas es indispensable a la hora de filtrar los períodos temporales. Por este motivo, es aplicada para generar los conjuntos de entrenamiento, validación y *testing*, además de ser necesaria para el funcionamiento del software de análisis.

Adicionalmente, a través del desarrollo del proyecto, se realizaron múltiples estudios de la base de datos otorgada por el ICyTE. Esto permitió detectar y eliminar diferentes mediciones atípicas o inválidas. A su vez, también se removieron días con mediciones incompletas. Además de las herramientas de software, se hace entrega al ICyTE del conjunto de datos final optimizado con las depuraciones mencionadas.

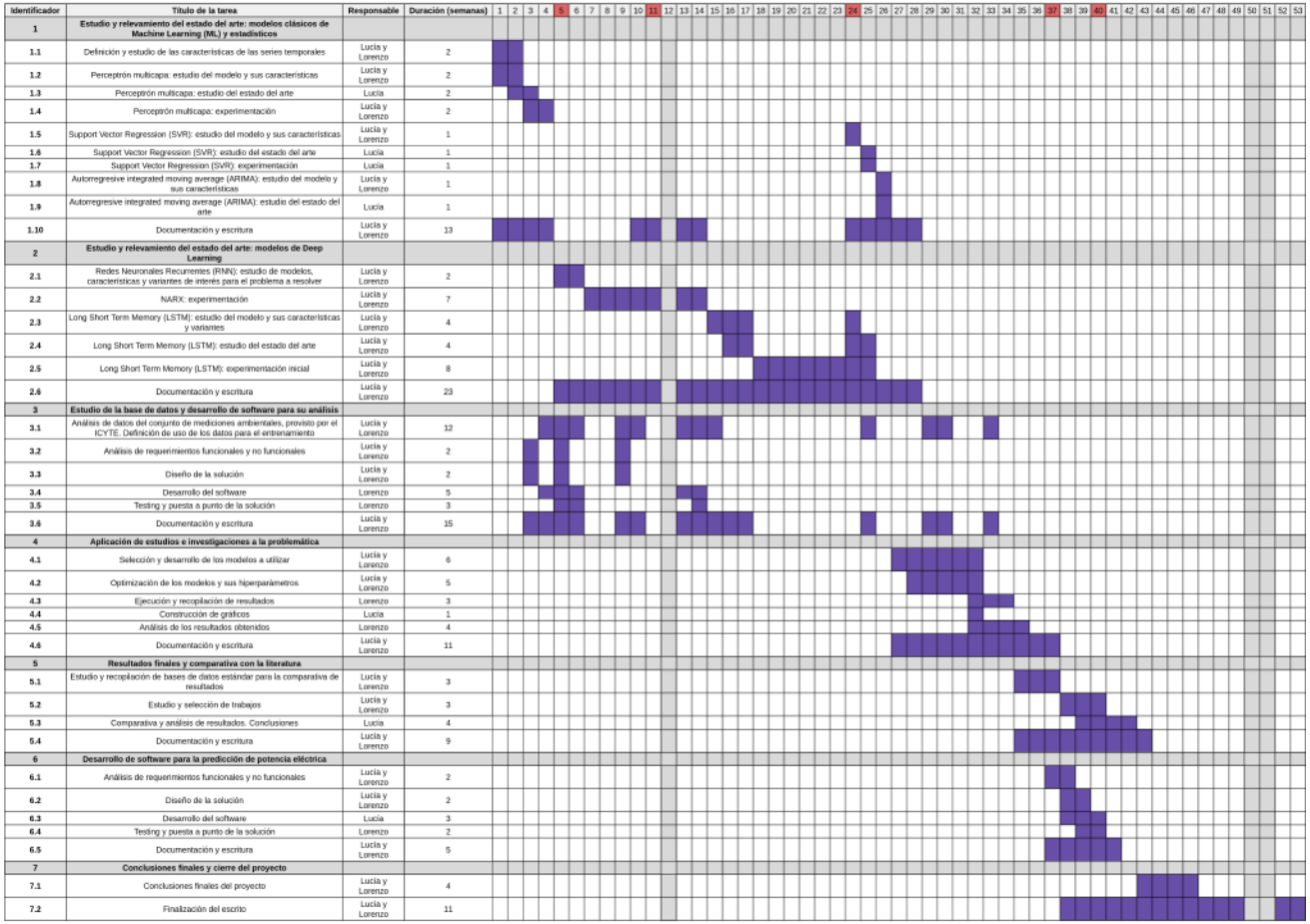
7.2. Gestión del proyecto

7.2.1. Planificación y ejecución

Como se mencionó en el capítulo 2, el proyecto se estableció dentro del marco de una investigación científico-tecnológica. Dado que el 44 % de la duración del trabajo se planificó para tareas de estudio y experimentación preliminar, fue de vital importancia establecer prioridades sobre la dedicación a cada modelo. La profundidad del área obligó a seleccionar cuidadosamente las tareas a desarrollar, con el objetivo de no extender los plazos del proyecto. Por estos motivos, se realizaron diversos puntos de control a lo largo del desarrollo.

Si bien el flujo de fases general se enmarcó dentro de las estipuladas originalmente, los puntos de control fueron utilizados para dictaminar el mejor curso de acción en el corto y mediano plazo. Las decisiones tomadas se basaron en los resultados obtenidos durante las fases de experimentación. Debido a la alta carga de trabajo y los plazos establecidos, fue necesario ajustar la dedicación a cada una de las arquitecturas y priorizar aquellas de mejor *performance*.

En la figura 7.1, puede observarse el diagrama de Gantt llevado a cabo en la práctica. Con el objetivo de lograr una mayor clarificación, se modificaron algunas de las tareas de la planificación inicial (ver sección 2.2):



Identificador	Título de la tarea	Responsable	Duración (semanas)
1	**Estudio y relevamiento del estado del arte: modelos clásicos de Machine Learning (ML) y estadísticos**		
1.1	Definición y estudio de las características de las series temporales	Lucía y Lorenzo	2
1.2	Perceptrón multicapa: estudio del modelo y sus características	Lucía y Lorenzo	2
1.3	Perceptrón multicapa: estudio del estado del arte	Lucía	2
1.4	Perceptrón multicapa: experimentación	Lucía y Lorenzo	2
1.5	Support Vector Regression (SVR): estudio del modelo y sus características	Lucía y Lorenzo	1
1.6	Support Vector Regression (SVR): estudio del estado del arte	Lucía	1
1.7	Support Vector Regression (SVR): experimentación	Lucía	1
1.8	Autoregresive integrated moving average (ARIMA): estudio del modelo y sus características	Lucía y Lorenzo	1
1.9	Autoregresive integrated moving average (ARIMA): estudio del estado del arte	Lucía	1
1.10	Documentación y escritura	Lucía y Lorenzo	13
2	**Estudio y relevamiento del estado del arte: modelos de Deep Learning**		
2.1	Redes Neuronales Recurrentes (RNN): estudio de modelos, características y variantes de interés para el problema a resolver	Lucía y Lorenzo	2
2.2	NARX: experimentación	Lucía y Lorenzo	7
2.3	Long Short Term Memory (LSTM): estudio del modelo y sus características y variantes	Lucía y Lorenzo	4
2.4	Long Short Term Memory (LSTM): estudio del estado del arte	Lucía y Lorenzo	4
2.5	Long Short Term Memory (LSTM): experimentación inicial	Lucía y Lorenzo	8
2.6	Documentación y escritura	Lucía y Lorenzo	23
3	**Estudio de la base de datos y desarrollo de software para su análisis**		
3.1	Análisis de datos del conjunto de mediciones ambientales, provisto por el ICYTE. Definición de uso de los datos para el entrenamiento	Lucía y Lorenzo	12
3.2	Análisis de requerimientos funcionales y no funcionales	Lucía y Lorenzo	2
3.3	Diseño de la solución	Lucía y Lorenzo	2
3.4	Desarrollo del software	Lorenzo	5
3.5	Testing y puesta a punto de la solución	Lorenzo	3
3.6	Documentación y escritura	Lucía y Lorenzo	15
4	**Aplicación de estudios e investigaciones a la problemática**		
4.1	Selección y desarrollo de los modelos a utilizar	Lucía y Lorenzo	6
4.2	Optimización de los modelos y sus hiperparámetros	Lucía y Lorenzo	5
4.3	Ejecución y recopilación de resultados	Lorenzo	3
4.4	Construcción de gráficos	Lucía	1
4.5	Análisis de los resultados obtenidos	Lorenzo	4
4.6	Documentación y escritura	Lucía y Lorenzo	11
5	**Resultados finales y comparativa con la literatura**		
5.1	Estudio y recopilación de bases de datos estándar para la comparativa de resultados	Lucía y Lorenzo	3
5.2	Estudio y selección de trabajos	Lucía y Lorenzo	3
5.3	Comparativa y análisis de resultados. Conclusiones	Lucía	4
5.4	Documentación y escritura	Lucía y Lorenzo	9
6	**Desarrollo de software para la predicción de potencia eléctrica**		
6.1	Análisis de requerimientos funcionales y no funcionales	Lucía y Lorenzo	2
6.2	Diseño de la solución	Lucía y Lorenzo	2
6.3	Desarrollo del software	Lucía	3
6.4	Testing y puesta a punto de la solución	Lorenzo	2
6.5	Documentación y escritura	Lucía y Lorenzo	5
7	**Conclusiones finales y cierre del proyecto**		
7.1	Conclusiones finales del proyecto	Lucía y Lorenzo	4
7.2	Finalización del escrito	Lucía y Lorenzo	11

Figura 7.1: Diagrama de Gantt ejecutado. La semanas en gris (12, 50 y 51) fueron de inactividad. Por otra parte, los números marcados en rojo representan semanas donde se realizó un punto de control sobre la planificación.

- Las tareas de estudio del estado del arte y aplicación sobre la temática fueron divididas. Para cada uno de los modelos (a excepción de ARIMA), se realizaron tareas de experimentación práctica. A continuación, se listan aquellas afectadas:
  - **Tarea 1.3**: subdividida en **1.3** y **1.4**.
  - **Tarea 1.5**: subdividida en **1.6** y **1.7**.
  - **Tarea 2.1**: subdividida en **2.1** y **2.2**.
  - **Tarea 2.3**: subdividida en **2.4** y **2.5**.
- La tarea **1.7** (ahora **1.9**) fue renombrada a *"Autorregresive integrated moving average* (ARIMA): estudio del estado del arte", dado que se optó por no realizar experimentación con el modelo.
- Se renombró la Etapa 3 y se ramificaron sus tareas asociadas. Originalmente, solo se había definido la tarea **3.1** para el análisis de datos. Sin embargo, también se había contemplado el desarrollo de una herramienta para facilitar el estudio de la base de datos del ICyTE. Dado que para su implementación se siguió una metodología de desarrollo de software, se especificaron las tareas **3.2**, **3.3**, **3.4** y **3.5** para representar el proceso de forma adecuada.
- Debido a la dificultad en la búsqueda de bases de datos externas de mediciones, la Etapa 5 fue reformulada para realizar comparativas en base a trabajos seleccionados. Las tareas **5.2** y **5.3** fueron reducidas a la tarea **5.2** actual.

7.2.2. Análisis de tiempos

Si bien existieron desvíos a lo largo del proyecto, la carga de trabajo total se mantuvo cercana a la esperada. La dedicación fue de 1171 h, un 10 % más de la estimada inicialmente (1066 h). Por otra parte, la planificación fue realizada sobre 41 semanas de trabajo. Sin embargo, el desarrollo demandó 53 semanas, lo que representa un aumento del 29 %.

Se considera que uno de los factores principales en la extensión fue la carga horaria semanal estipulada. Durante el proyecto, el promedio de trabajo fue de 11,05 h por persona, menor a las 13 h planificadas. A su vez, cabe destacar que la carga horaria varió semanalmente y no fue constante. Paralelamente al proyecto, se debió dedicar esfuerzo diario a empleos de tiempo completo. El cansancio mental de llevar ambas responsabilidades en simultáneo fue originalmente subestimado, por lo que la dedicación en las primeras 11 semanas fue de aproximadamente 9,74 h por persona. En contraposición, la carga horaria creció hasta las 11,67 h entre la semana 13 y el final del proyecto. La experiencia adquirida a lo largo del trabajo fue importante para optimizar las tareas y el tiempo productivo dedicado.

7.2.3. División del trabajo

La división del trabajo fue de gran importancia a lo largo del proyecto. Sin embargo, aquella establecida inicialmente fue una de las falencias de la planificación original. Si bien se intentó respetarla, existieron situaciones donde no fue posible.

Una de las razones fue que las tareas diagramadas contaron con una baja granularidad. En el día a día, fue necesario establecer tareas más reducidas (definidas en términos de horas) para realizar una gestión adecuada. A modo de ejemplo, la tarea **2.2** implicó la experimentación con el modelo NARX. Al abarcar un total de 7 semanas, en la práctica fue subdividida en múltiples otras de menor envergadura. Por ejemplo: la predicción diaria, la predicción de un mes, la optimización mediante búsqueda por grilla, entre otras.

La subdivisión sobre tareas más granulares permitió gestionar el trabajo de forma eficiente, para lo cual fue de mucha utilidad el uso del sistema Kanban. A medida que transcurrió el trabajo, las estimaciones reducidas se volvieron más precisas debido a la obtención de experiencia en el problema y en las tareas de gestión.

7.2.4. Documentación y escritura

Originalmente, las tareas referidas a la documentación y escritura se situaron en paralelo a cada etapa. Su objetivo era producir una versión inicial del escrito asociado a la fase. La razón de esta diagramación fue reducir la cantidad de tiempo entre cada tarea y su respectiva plasmación en el informe, de manera que los detalles vinculados a cada una estuvieran más presentes durante la escritura.

Si bien la documentación fue acorde a lo planificado, la escritura debió extenderse en cada una de las etapas. De todas formas, se considera que la carga de trabajo sobre la Etapa 7 (vinculada a la finalización del escrito) fue mejor aprovechada al trabajar con esta metodología.

Adicionalmente, por recomendación de los directores, se utilizó el sistema de composición de textos LaTeX para el desarrollo del informe. Si bien inicialmente se debió dedicar esfuerzo adicional para su aprendizaje, se lo consideró de gran utilidad para la gestión de la bibliografía, las ecuaciones matemáticas y las figuras presentadas.

7.2.5. Uso de herramientas colaborativas

A comienzos del proyecto, se realizaron diversas reuniones presenciales para establecer la dinámica de trabajo y realizar las planificaciones iniciales. Sin embargo, la presencialidad se redujo paulatinamente con el correr de los meses, especialmente debido a que el Dr. Ing. S. A. González, co-director del proyecto, no estuvo presente en el país. Por este motivo, sumado a que el proyecto fue desarrollado de forma remota, fue de especial importancia establecer el uso de herramientas colaborativas para llevar adelante una gestión adecuada de la información:

- La gestión de tareas fue realizada en Trello [129], una herramienta para el armado de tableros Kanban. Su uso permitió granularizar y asignar de forma simple las responsabilidades. Asimismo, la visualización de la información mediante tableros fue de utilidad para tener constancia del grado de avance del proyecto, así como de las tareas pendientes y en desarrollo.
- El código desarrollado quedó a disposición en un repositorio de GitHub [130], lo que facilitó su almacenamiento y versionado.
- Se utilizó el editor de LaTeX colaborativo Overleaf [131] para la escritura del presente informe, junto con las herramientas de Google Drive [132] para el guardado de tablas y documentación inicial.
- Las reuniones entre alumnos y directores se realizaron por medio de videollamadas en Google Meet [133].

7.2.6. Análisis de etapas, desvíos y puntos de control

En la figura 7.2, se puede visualizar una comparativa entre los tiempos asociados a cada fase y los estimados originalmente. Si bien los valores reales fueron registrados durante el proyecto, la planificación inicial se había realizado por semana y no por hora, por lo que el valor del tiempo original es aproximado.

Para el cálculo, se utilizó la dedicación semanal estipulada originalmente y se la consideró constante a lo largo del proyecto. En semanas con superposición de tareas, la carga de trabajo se dividió equitativamente entre cada una de ellas. A su vez, debido a su dedicación altamente variable, no se subdividió por las tareas de documentación y escritura (a excepción de la **7.2**). Cabe aclarar que la aproximación no tiene en consideración el peso o la prioridad de cada tarea en la semana, por lo que los valores no son exactos. Un ejemplo son aquellos referidos a la Etapa 6, que son subestimados por el cálculo al superponerse constantemente a tareas de la Etapa 7. A pesar de estos problemas, se lo consideró la forma más objetiva de establecer la aproximación.

Mas allá de las observaciones realizadas, se pueden apreciar diferencias considerables entre lo planificado para ciertas etapas y su carga de trabajo real. A continuación, se realiza un análisis acerca de los tiempos obtenidos para cada fase, los desvíos presentados y los puntos de control que existieron a lo largo del proyecto.

7.2.6.1. Etapa 1: Estudio y relevamiento del estado del arte: modelos clásicos de *Machine Learning* y estadísticos

La primera etapa del proyecto implicaba la combinación del estudio y el análisis de los bloques fundacionales relacionados a la temática: las series temporales y los modelos clásicos de *Machine Learning* y estadísticos. Las tareas iniciales se desempeñaron con normalidad y acorde a lo estipulado en el diagrama inicial. Sin embargo, luego ocurrió una desviación del plan original.

Además del estudio específico de MLP, SVR y ARIMA, uno de los objetivos de la etapa era adquirir los conocimientos fundamentales requeridos para el trabajo con modelos de aprendizaje automático. De

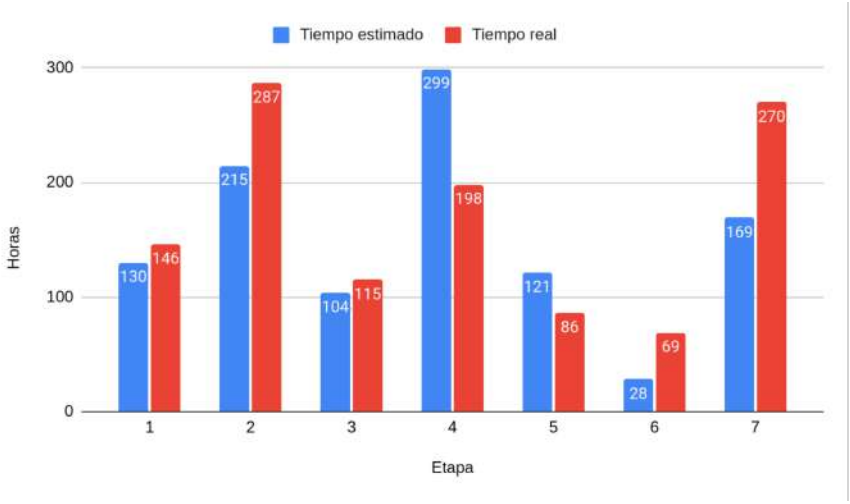


Figura 7.2: Comparativa entre las horas estimadas y las reales de cada etapa del proyecto.

esta forma, sería posible ahondar en arquitecturas de mayor complejidad y con resultados superiores en el estado del arte (como las RNN, ver sección 1.1).

En la semana 5, se realizó un punto de control para analizar el progreso en el primer mes de trabajo. Como observación principal, se consideró que la experimentación y la lectura realizada acerca del modelo MLP había sido suficiente para establecer las bases de las metodologías de entrenamiento y ajuste de hiperparámetros a utilizar en el proyecto. Por este motivo, se decidió aplazar el estudio del resto de modelos clásicos y priorizar los vinculados al *Deep Learning*.

Adicionalmente, la escritura referente a las series temporales y la experimentación con MLP debió ser demorada, de manera que fue realizada 6 semanas después. El motivo fue la alta prioridad asociada a las tareas de NARX, junto con una demanda de tiempo adicional por el aprendizaje del sistema LaTeX.

El análisis y la experimentación con SVR fueron realizados en paralelo al *tuning* de hiperparámetros para el año 2019 sobre las redes LSTM y GRU (Etapa 2). Se observó que sus resultados no superaron aquellos obtenidos con los modelos de *Deep Learning*, por lo que se descartó su uso en la Etapa 4.

Para el caso de ARIMA, se decidió no dedicar carga de trabajo a la experimentación e implementación del modelo. La razón principal fue que se lo consideró anticuado y con resultados inferiores a los de otros modelos de mayor complejidad utilizados en el proyecto. Adicionalmente, se estimó que la experiencia que podía adquirirse con su uso no era suficiente para justificar la dedicación.

En total, se habían planificado 7 semanas de estudio y análisis de los modelos SVR y ARIMA, mientras que el proyecto dedicó únicamente 3 semanas. El tiempo restante fue utilizado para priorizar la implementación y experimentación con modelos de *Deep Learning*, junto con la incorporación de variantes adicionales que reportaron resultados prometedores a lo largo del proyecto (como los modelos bidireccionales y los mecanismos de atención).

- **Tiempo estimado**: 8 semanas (130 h aproximadamente).
- **Tiempo real**: 13 semanas (146 h).

7.2.6.2. Etapa 2: Estudio y relevamiento del estado del arte: modelos de *Deep Learning*

La segunda etapa del proyecto se caracterizaba por ser el primer acercamiento a las redes neuronales recurrentes. Principalmente, el foco se había colocado sobre las arquitecturas LSTM y GRU. Sin embargo, también fue considerado relevante estudiar variantes de interés al problema, como el modelo NARX. Debido al grado de complejidad asociado a los modelos utilizados, fue estimada como la segunda fase de mayor dedicación.

Originalmente, el estudio inicial de las RNN y sus variantes se estipuló para 4 semanas. Sin embargo, los tiempos reales de análisis y experimentación se extendieron a un total de 9 (sin contar una semana de inactividad). Una de las razones asociadas al desvío fue la dificultad a la hora de encontrar una herramienta de software que cuente con las características deseadas para la experimentación con redes NARX. En un primer momento, se priorizó el uso de librerías Python para el desarrollo, dado que se había adquirido importante conocimiento durante la experimentación con MLP. Sin embargo, las

limitaciones de las opciones encontradas obligaron a la migración a un entorno completamente distinto como MATLAB, lo cual demandó tiempo fuera de lo previsto.

Adicionalmente, se considera en retrospectiva que habría sido óptimo incursionar con mayor rapidez sobre la predicción de potencia a partir de los datos del ICyTE. Si bien la experimentación con las funciones seno fue importante como acercamiento al modelo, se cree que el tiempo invertido en la realización de las pruebas se extendió más allá de lo requerido.

En la semana 11, se realizó una reunión entre todos los integrantes del equipo de trabajo, donde se determinaron las tareas a seguir durante las vacaciones de verano de los directores. Al igual que como se planificó originalmente, se decidió proseguir con la investigación de LSTM y GRU como paso siguiente a la experimentación con NARX.

La semana 12 fue la última del año 2022, en donde se contó con 7 días de vacaciones laborales. Debido a que el cansancio mental estaba dificultando el avance sobre el proyecto, se optó por cesar las actividades durante dicho período. En retrospectiva, la semana de descanso fue de gran importancia, dado que permitió reanudar el trabajo con esfuerzos renovados y dedicar una mayor cantidad de tiempo productivo.

En la semana 24, una vez retomadas las actividades de los directores, se realizó una nueva reunión grupal. En base a los resultados que se habían obtenido en la experimentación, se decidió mantener a los modelos bidireccionales como variantes adicionales de la investigación. A su vez, se planificó incluir a los mecanismos de atención como otra alternativa dentro de los modelos de *Deep Learning*. Por este motivo, fue necesario dedicar tiempo adicional a la etapa de estudio y experimentación con dichas arquitecturas. Como se mencionó en el análisis de la Etapa 1, la horas de trabajo fueron extraídas del tiempo restante de los modelos clásicos.

Si bien el modelo final no utilizó una arquitectura bidireccional ni mecanismos de atención, se considera que la desviación establecida sobre la planificación original fue acertada. El 50 % de las arquitecturas seleccionadas para el análisis final de los conjuntos de validación contó con alguna de dichas características. A su vez, el tercer mejor modelo final (en base al valor de MAPE obtenido) fue un modelo Bi-LSTM. Además de ofrecer resultados cercanos a los óptimos, su análisis y experimentación puede ser de utilidad para trabajos futuros del ICyTE.

- **Tiempo estimado**: 10 semanas (215 h aproximadamente).
- **Tiempo real**: 23 semanas (287 h).

7.2.6.3. Etapa 3: Estudio de la base de datos y desarrollo de software para su análisis

La Etapa 3 del proyecto consistía en el desarrollo de una herramienta para el análisis de la base de datos provista por el ICyTE, así como el posterior estudio acerca de las mediciones involucradas, su evolución temporal y su grado de correlación con la potencia eléctrica.

El comienzo de la fase se dio en concordancia a lo estipulado inicialmente. A su vez, si bien no se había planificado ofrecer la herramienta como un software completo, solo fue necesario el desarrollo adicional de una interfaz de usuario simple. Por este motivo, los costos de su implementación estuvieron acorde a lo planificado.

La principal diferencia respecto al diagrama de Gantt original fue el uso reiterado que se le debió dar a la herramienta. Durante diferentes etapas de la experimentación, fue necesario volver a realizar un análisis de ciertos períodos de datos con el objetivo de eliminar valores atípicos o inválidos. A su vez, nuevamente fue utilizada durante las pruebas estacionales con el modelo final, donde se la aplicó para la selección de los períodos de entrenamiento, validación y *testing*.

Las tareas de análisis y diseño de la herramienta se realizaron en tres instancias, dado que los requerimientos crecieron a medida que transcurrió el proyecto. Adicionalmente, el desarrollo y *testing* fue completado en la semana 14, donde se terminaron de implementar las funcionalidades menos prioritarias.

- **Tiempo estimado**: 7 semanas (104 h aproximadamente).
- **Tiempo real**: 15 semanas (115 h).

7.2.6.4. Etapa 4: Aplicación de estudios e investigaciones a la problemática

La fase estaba dedicada a la selección y la optimización de los modelos finales, así como la ejecución y evaluación de sus métricas. Dado que estuvo estrechamente relacionada al proceso de experimentación

con LSTM y GRU de la Etapa 2, se considera que su inicio fue la incorporación del período 2016-2018 a los ajustes automatizados.

Se observó que la planificación previa sobreestimó la cantidad de horas requeridas. La razón principal fue el profundo trabajo realizado en la experimentación inicial, donde se logró limitar la cantidad de arquitecturas y variantes de estudio, junto con el rango de los hiperparámetros a optimizar. Adicionalmente, diversas tareas realizadas en la fase previa (como la configuración de entornos de ejecución y la implementación del *tuning* automatizado) permitieron reducir la cantidad de horas dedicadas.

- **Tiempo estimado**: 13 semanas (299 h aproximadamente).
- **Tiempo real**: 11 semanas (198 h).

7.2.6.5. Etapa 5: Resultados finales y comparativa con la literatura

Originalmente, el objetivo de la etapa era recopilar bases de datos externas y realizar ejecuciones con el modelo final. Sin embargo, no se logró encontrar ejemplos para una comparativa acorde en términos de resolución y cantidad de variables. Por este motivo, fue requerido establecer un desvío por sobre la planificación original.

Se realizó un punto de control en la semana 37, donde se reformuló la etapa para realizar comparativas en base a trabajos recopilados. La selección fue realizada en base a las condiciones de sus localizaciones, con el objetivo de efectuar una comparación acorde al clima de la ciudad de Mar del Plata. Al no poder realizar ejecuciones, los tiempos estimados para la fase resultaron superiores a los obtenidos.

Si bien no se cumplió el objetivo inicial, se logró reacondicionar la fase de forma exitosa. Se considera que la comparativa realizada fue de utilidad para establecer el grado de exactitud del modelo, no solo respecto a aquellos utilizados por el ICyTE, sino también ante los encontrados en la literatura.

- **Tiempo estimado**: 7 semanas (121 h aproximadamente).
- **Tiempo real**: 9 semanas (86 h).

7.2.6.6. Etapa 6: Desarrollo de software para la predicción de potencia eléctrica

La etapa consistía en la implementación de una herramienta de software para la predicción de potencia, basada en el modelo construido. Los resultados obtenidos debían poder ser visualizados y exportados por el usuario.

En base a los comentarios de la cátedra, las actividades de análisis y diseño fueron realizadas de forma conjunta. Asimismo, con el objetivo de ofrecer una solución ingenieril completa y acorde a la esperada, el software contó con todas las capacidades demandadas por el ICyTE. Además de la ejecución del modelo, se ofrece la posibilidad de configurar sus hiperparámetros, el formato de las secuencias de entrada, su rango de validación, el período de *testing* y la visualización de gráficos, entre otras funcionalidades. Al mismo tiempo, el software provee el acceso a la herramienta de análisis de la base de datos desarrollada en la Etapa 3.

Se considera que todas las características mencionadas serán de utilidad para el ICyTE, no solo para la predicción de potencia eléctrica, sino también para tareas futuras de experimentación y estudio.

- **Tiempo estimado**: 3 semanas (28 h aproximadamente).
- **Tiempo real**: 5 semanas (69 h[1]).

7.2.6.7. Etapa 7: Conclusiones finales y cierre del proyecto

La Etapa 7 estaba enfocada en establecer las conclusiones del proyecto y llevar a cabo la finalización del informe. Su inicio se estableció en la semana 38, donde los directores reportaron las primeras correcciones sobre los escritos realizados anteriormente. Las conclusiones finales fueron establecidas una vez finalizada la etapa de experimentación y comparativa. A su vez, en la semana 40, se realizó un último punto de control previo a las vacaciones de invierno de los directores, donde se diagramaron las tareas finales de escritura y los plazos de corrección para la entrega del presente informe.

[1] A modo de aclaración, se menciona que la implementación del modelo había sido realizada en las Etapas 3 y 4. El tiempo invertido en su desarrollo no fue considerado para la presente etapa, que se centró en ofrecer las funcionalidades de acceso, exportación de resultados y configuración.

La confección del escrito fue una tarea ardua, debido a que durante la experimentación se generó un amplio volumen de información y resultados. Esto obligó a realizar múltiples iteraciones del informe, donde el objetivo fue realizar una presentación adecuada de todo el trabajo realizado a lo largo del proyecto. La dedicación horaria a lo largo de la fase fue variable, dado que el proceso requirió de múltiples instancias de corrección por parte de los directores. Por este motivo, ciertas semanas contaron con una menor y hasta nula carga de trabajo (como las número 50 y 51).

Finalmente, se considera que el informe presentado es acorde a lo esperado por la cátedra y representa adecuadamente la dedicación al proyecto. Se considera que la decisión de realizar documentación y escritura en simultáneo fue de utilidad, debido a que redujo los esfuerzos requeridos para escribir acerca de fases anteriores. Sin embargo, debido a los motivos mencionados anteriormente, la duración final de la etapa fue mayor a la esperada.

- **Tiempo estimado**: 10 semanas (169 h aproximadamente).
- **Tiempo real**: 16 semanas (270 h).

7.3. Conclusión

Se considera que los objetivos planteados al inicio del proyecto fueron alcanzados satisfactoriamente. Como producto central del trabajo, se construyó un modelo predictivo con resultados superiores a aquellos obtenidos por el ICyTE. El modelo puede ser operado a través de una herramienta de software, que posee capacidades de configuración, exportación de resultados y análisis de series temporales. Adicionalmente, se considera que el desarrollo del trabajo fue de suma utilidad para la adquisición de las competencias profesionales esperadas.

Si bien la carga horaria total solo fue superada en un 10 %, la entrega final del proyecto se extendió un 29 % por sobre la fecha estimada. Se detallaron las razones principales en las diferencias, una de las cuales fue la dedicación semanal definida inicialmente. A su vez, ciertas etapas (como la finalización del escrito), demandaron de tiempo adicional por sobre el estimado. Si bien se considera que las estimaciones fueron aceptables en base a la complejidad del trabajo y su elevada duración e incertidumbre, se reflexionó sobre los errores cometidos y se los considera parte del aprendizaje para las instancias profesionales futuras.

Como se mencion o a lo largo del capítulo, la planificación inicial sufrió diferentes modificaciones durante el desarrollo del proyecto. Al enmarcarse dentro de un proceso de investigación y experimentación, existía el riesgo de incurrir en desvíos respecto a lo diagramado inicialmente. La mayor parte de las diferencias se observaron en las fases de estudio y experimentación inicial, donde la incertidumbre era mayor ante la falta de experiencia en la temática.

Si bien existieron desvíos no intencionales, como los sufridos en la etapa de NARX, se considera que se realizó una adecuada gestión de los problemas que se presentaron a lo largo del trabajo. El establecimiento de puntos de control fue útil para revisar la planificación inicial y establecer las tareas de mayor relevancia. El ejemplo más claro fue la priorización del estudio de las RNN por sobre los modelos clásicos, una decisión que en retrospectiva se considera acertada. Adicionalmente, el uso de herramientas colaborativas fue crucial para el desarrollo de todo el proyecto, que se realizó mayormente de forma remota.

Finalmente, se concluye que la experiencia a lo largo del año fue muy enriquecedora. El aprendizaje obtenido durante el proceso permitió fomentar capacidades fundamentales del futuro profesional, especialmente en competencias sobre las cuales no se había trabajado anteriormente.

Capítulo 8

Conclusiones y trabajo futuro

A continuación, se detalla cada una de las conclusiones obtenidas a lo largo del proyecto. A su vez, se presentaran distintos puntos de evolución para el proyecto que podrían realizarse en un futuro.

8.1. Conclusiones

Para concluir el proyecto, se considera que los objetivos planteados inicialmente se han cumplido. En primer lugar, se realizó un estudio abarcativo sobre las temáticas de las series temporales, los modelos conexionistas y el *Deep Learning*. Adicionalmente, se analizaron diversas publicaciones para tener conocimiento del estado del arte de las arquitecturas en la problemática de la predicción de potencia eléctrica.

Durante la construcción de los modelos, fue necesario investigar acerca de los diferentes entornos de desarrollo y lenguajes disponibles. Se analizaron múltiples librerías y herramientas que fueron de utilidad para llevar a cabo cada una de las implementaciones. Adicionalmente, se debió incursionar en diversos entornos de ejecución en la nube, que permitieron acelerar notablemente los tiempos de entrenamiento y predicción de los modelos.

Dentro de la etapa de experimentación, se realizó un primer acercamiento a la temática a partir de las pruebas con el modelo MLP. Con el avance del proyecto, se incorporó el análisis de modelos de mayor complejidad, como las redes NARX, LSTM y GRU, junto a sus respectivas variantes bidireccionales y con mecanismos de atención. Las pruebas se centraron en encontrar el mejor modelo predictivo para el problema a partir de la experimentación con sus hiperparámetros, la generación de secuencias de entrada y los conjuntos de datos utilizados, entre otras características. Los procesos de *tuning* fueron ejecutados en base a meses calurosos, por lo que los modelos producidos fueron optimizados sobre dicho período.

El modelo que minimizó el MAPE promedio diario para la predicción de cada día del mes de enero de 2020 a un 9,46 % (±0,20 %) fue una red LSTM con capas ocultas de 336, 304 y 208 unidades. Utilizó función de pérdida MAPE, optimizador Adam, secuencias de entrada de tamaño 50 y *dropout* recurrente de valor 0,5 en la tercera capa. El modelo fue generado al utilizar el período 2016-2019 en la etapa de entrenamiento, con el mes de diciembre de 2019 como conjunto de validación.

Además del análisis de la *performance* en enero, se realizaron pruebas con conjuntos de *testing* pertenecientes a diferentes estaciones climáticas. Dado que el modelo fue construido en base a una optimización sobre el verano, se observó que los niveles de MAPE no se asemejaron a los reportados previamente. Al predecir un mes completo de cada estación, se obtuvieron errores del 20,43 % (±0,53 %) para primavera, un 26,32 % (±1,30 %) para invierno y un 30,85 % (±2,48 %) para otoño.

Para comprobar la robustez del modelo obtenido, se inició una búsqueda de bases de datos públicas con las variables climáticas y los niveles de resolución utilizados en el proyecto. Sin embargo, debido a numerosos factores externos, no se halló ninguna con la compatibilidad requerida. Por este motivo, se realizaron comparativas en base a diferentes publicaciones desarrolladas en condiciones climáticas similares a la ciudad de Mar del Plata. Particularmente, al analizar un trabajo desarrollado sobre la planta fotovoltaica del ICyTE, se obtuvieron mejoras del 6,80 % en términos de MAPE, 25,81 % para el RMSE y 57,89 % para el MAE. La única métrica no superada fue el R2, que aumentó en un 2,90 %. Cabe destacar que el modelo propuesto posee una complejidad menor al de la publicación analizada.

Respecto a trabajos externos, no fue posible realizar una comparativa exhaustiva debido a las diferentes bases de datos utilizadas. Sin embargo, se observó que. A partir de las comparativas, se concluyó

que el modelo LSTM obtenido cumplió con los objetivos iniciales del proyecto. A su vez, se logró comprobar la superioridad de los modelos de *Deep Learning* por sobre los estadísticos y los de aprendizaje automático.

En cuanto al proceso de desarrollo de software, el trabajo concluyó en una herramienta capaz de efectuar análisis de datos y llevar a cabo las tareas predictivas. Las funcionalidades de análisis incluyen la visualización de la evolución temporal de cada una de las variables climáticas almacenada en la base de datos del ICyTE, la determinación su grado de correlación con la potencia eléctrica y la detección de valores atípicos, entre otras. Adicionalmente, el módulo predictivo permite acceder al modelo obtenido durante el desarrollo del proyecto y realizar predicciones a partir de su uso. Para ofrecer un producto completo y de uso futuro en la experimentación del ICyTE, también se ofrece la posibilidad de modificar sus hiperparámetros, gestionar la generación de datos de entrada, configurar las etapas de entrenamiento y *testing* y exportar los resultados generados.

Como trabajo adicional, se realizó un análisis de la base de datos provista por el ICyTE. Se pudieron extraer diferentes características de las series temporales almacenadas, como por ejemplo el grado de correlación entre las diferentes mediciones y la potencia eléctrica. Finalmente, el proyecto demostró la utilidad de cada una de las variables almacenadas, por lo que se recomienda no alterar el conjunto de mediciones. A su vez, los análisis realizados permitieron identificar diversos datos erróneos que resultaban en cálculos de métricas desfavorables. Por este motivo, se realizó una limpieza de la base de datos y se hace entrega de ella al ICyTE.

En base a la planificación inicial, la estimación de horas totales fue acertada. Sin embargo, la fecha de finalización establecida se extendió en un 29 %, principalmente debido a que la dedicación semanal estimada fue optimista. A pesar de las diferencias, se considera que la gestión realizada fue acorde a los riesgos, la envergadura y el contexto del proyecto. Si bien existieron diferencias de acuerdo a la planificación inicial, algunos desvíos resultaron por decisiones propias del equipo y fueron justificados en base a los resultados obtenidos. Adicionalmente, se destaca el buen manejo de herramientas colaborativas durante las fases con trabajo remoto y la utilidad de los planes de contingencia desarrollados, especialmente aquel planteado para las limitaciones en el hardware durante la fase de experimentación.

Se considera que el trabajo final de la carrera ha sido una instancia realmente fructífera. Se incursionó de forma exitosa sobre un proyecto de índole científico-tecnológica y se trabajó sobre diferentes temáticas de amplia relevancia en la actualidad, como el aprendizaje profundo, las series temporales y la predicción de potencia eléctrica en el marco de las energías renovables. Adicionalmente, al enfrentarse a una problemática real, el proyecto resultó de gran utilidad para desarrollar competencias propias de un ingeniero informático. Principalmente, aquellas referidas a la gestión de proyectos y los procesos de desarrollo de software.

Por otra parte, se destaca la interacción interdepartamental entre las áreas de Informática, Computación y Electrónica. No es común encontrar experiencias de esta índole y consideramos que trabajos colaborativos pueden afianzar las interacciones entre departamentos de la institución.

Finalmente, al analizar el trabajo desde el punto de vista personal, si bien resultó dificultoso llevar a cabo un proyecto de alta complejidad a la par de empleos de tiempo completo, fue muy gratificante acercarse a una temática de gran interés como el *Deep Learning*. A su vez, se destaca el sentimiento de satisfacción obtenido al aplicar conocimientos adquiridos en una gran cantidad de asignaturas cursadas a lo largo de la carrera universitaria.

8.2. Trabajo futuro

Como trabajo adicional fuera del alcance del presente proyecto, se sugiere al ICyTE profundizar sobre los siguientes ejes:

- **Modelos estacionales:** se recomienda la búsqueda de un modelo optimizado y validado para cada estación climática, con el objetivo de determinar si es posible reducir el MAPE.
- **Inclusión de librerías climáticas:** para este proyecto, no se contempló la inclusión de librerías NWP en el software predictivo. Estas permiten predecir las condiciones climáticas de un día determinado y serían necesarias para el uso futuro del modelo.
- **Redes convolucionales:** se recomienda experimentar sobre este tipo de redes, dado que se encontraban por fuera de la planificación original. Puede observarse en numerosos trabajos que su inclusión reduce las métricas de error de los modelos.

- **Investigación sobre mecanismos de atención:** existen también mecanismos de atención de mayor robustez que el utilizado en el proyecto (como el mecanismo *multi-head*, o mecanismos de atención aplicados a modelos Transformer, entre otros), que merecen un estudio en profundidad ya que podrían beneficiar notablemente a los procesos de predicción [88] [2].

Referencias

[1] Xia, Y., Wang, J., Zhang, Z., Wei, D. and Yin, L., "Short-term PV power forecasting based on time series expansion and high-order fuzzy cognitive maps," 2023. DOI: 10.1016/j.asoc.2023.110037.

[2] Zhou, H., Zhang, Y., Yang, L., Liu, Q., Yan, K. and Du, Y., "Short-Term Photovoltaic Power Forecasting Based on Long Short Term Memory Neural Network and Attention Mechanism," 2019. DOI: 10.1109/ACCESS.2019.2923006.

[3] Sedai, A., Dhakal, R., Gautam, S., Dhamala, A., Bilbao, A., Wang, Q., Wigington, A. and Pol, S., "Performance Analysis of Statistical, Machine Learning and Deep Learning Models in Long-Term Forecasting of Solar Power Production," 2023. DOI: 10.3390/forecast5010014.

[4] Myers, Daryl, "Evaluation of the Performance of the PVUSA Rating Methodology Applied to DUAL Junction PV Technology," 2009. URL: https://www.nrel.gov/docs/fy09osti/45376.pdf.

[5] González, S. A., Murcia, G., Garín, E., Branda, J., Lanson, A. and Aristegui, R., "Performance Ratio Calculation of a Grid-connected PV Plant Through Different Techniques," 2019. URL: http://sedici.unlp.edu.ar/handle/10915/109741.

[6] Khalid, A., Mitra, I., Warmuth, W. and Schacht, V., "Performance ratio – Crucial parameter for grid connected PV plants," 2016. DOI: 10.1016/j.rser.2016.07.066.

[7] Lee, W., Kim, K., Park, J., Kim, J. and Kim, Y., "Forecasting Solar Power Using Long-Short Term Memory and Convolutional Neural Networks," 2018. DOI: 10.1109/ACCESS.2018.2883330.

[8] Elsaraiti, Meftah and Merabet, Adel, "Solar Power Forecasting Using Deep Learning Techniques," 2022. DOI: 10.1109/ACCESS.2022.3160484.

[9] Nkuriyingoma, Obed and Selcuklu, Saltuk, "Solar power plant generation forecasting using NARX neural network model: A case study," 2021. DOI: 10.31593/ijeat.870088.

[10] Gensler, A., Henze, J., Sick, B. and Raabe, N., "Deep Learning for solar power forecasting — An approach using AutoEncoder and LSTM Neural Networks," 2016. DOI: 10.1109/SMC.2016.7844673.

[11] Chen, Hailang and Chang, Xianfa, "Photovoltaic power prediction of LSTM model based on Pearson feature selection," 2021. DOI: 10.1016/j.egyr.2021.09.167.

[12] Gong, G., Lou, K., Yin, J. and Li, D., "Forecast of photovoltaic Power Generation Based on GRU," 2023. DOI: 10.1145/3573428.3573477.

[13] Sodsong, N., Yu, K. and Ouyang, W., "Short-Term Solar PV Forecasting Using Gated Recurrent Unit with a Cascade Model," 2019. DOI: 10.1109/ICAIIC.2019.8668970.

[14] El Bourakadi, D., Ramadan, H., Yahyaouy, A. and Boumhidi, J., "A novel solar power prediction model based on stacked BiLSTM deep learning and improved extreme learning machine," 2022. DOI: 10.1007/s41870-022-01118-1.

[15] Wang, S., Li, D., Liu, G., Ling, Z. and Wang, D., "Short-term PV Power Prediction Based on Bi-directional Gated Recurrent Unit Network and Adaptive Chirp Mode Decomposition," 2023. DOI: 10.1109/NNICE58320.2023.10105683.

[16] Sabri, Mohammed and El Hassouni, Mohammed, "A Novel Deep Learning Approach for Short Term Photovoltaic Power Forecasting Based on GRU-CNN Model," 2021. DOI: 10.1051/e3sconf/202233600064.

[17] Wang, K., Qi, X. and Liu, H., "Photovoltaic power forecasting based LSTM-Convolutional Network," 2019. DOI: 10.1016/j.energy.2019.116225.

[18] Zhang, C., Peng, T. and Nazir, M., "A novel integrated photovoltaic power forecasting model based on variational mode decomposition and CNN-BiGRU considering meteorological variables," 2022. DOI: 10.1016/j.epsr.2022.108796.

[19] Babalhavaeji, A., Radmanesh, M., Jalili, M. and González, S. A., "A Photovoltaic Generation Forecasting using Convolutional and Recurrent Neural Networks," 2023.

[20] P. Brockwell y R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 2002.

[21] P. P. Pathak, *Time Series Forecasting — A Complete Guide*, Accedido: 2022-03-22, 2021. URL: https://medium.com/analytics-vidhya/time-series-forecasting-a-complete-guide-d963142da33f.

[22] D. Instruments, *Weatherlink*, 2023. URL: https://www.davisinstruments.com/pages/weatherlink-cloud.

[23] Powerside, *PQube 3 Power Analyzers*, 2023. URL: https://powerside.com/products/power-quality-measurement-solutions/pqube-3-power-analyzers/.

[24] W. Mcculloch y W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, vol. 5, págs. 127-147, 1943.

[25] Berkeley, Istvan, "The Curious Case of Connectionism," 2019. DOI: 10.1515/opphil-2019-0018.

[26] N. D. Lewis, *Deep Time Series Forecasting With Python*. CreateSpace, 2016.

[27] S. Haykin, *Neural Networks and Learning Machines*. Pearson, 2009.

[28] M. A. Arbib, *Brains, Machines, and Mathematics*. Springer, 1987.

[29] G. M. Shepherd y C. Koch, *The Synaptic Organization of the Brain*. Oxford University Press, 1990.

[30] Rosenblatt, Frank, "The perceptron: A probabilistic model for information storage and organization in the brain," 1958. DOI: 10.1037/h0042519.

[31] *The role of bias in Neural Networks*, Accedido: 2023-04-02. URL: https://www.pico.net/kb/the-role-of-bias-in-neural-networks/.

[32] Y. Tiwari, *Activation functions in Neural Networks*, Accedido: 2023-04-02, 2023. URL: https://www.geeksforgeeks.org/activation-functions-neural-networks/.

[33] P. Baheti, *Activation Functions in Neural Networks [12 Types & Use Cases]*, Accedido: 2023-04-02, 2023. URL: https://www.v7labs.com/blog/neural-networks-activation-functions.

[34] L. Lu, Y. Shin, Y. Su y G. Karniadakis, "Dying ReLU and Initialization: Theory and Numerical Examples," *Communications in Computational Physics*, vol. 28, págs. 1671-1706, nov. de 2020. DOI: 10.4208/cicp.OA-2020-0165.

[35] *Understanding Feed Forward Neural Networks With Maths and Statistics*, Accedido: 2023-04-02. URL: https://www.turing.com/kb/mathematical-formulation-of-feed-forward-neural-network.

[36] C. Olah, *Understanding LSTM Networks*, Accedido: 2023-05-12. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[37] J. Delua, *Supervised vs. Unsupervised Learning: What's the Difference?* Accedido: 2023-04-02, 2021. URL: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

[38] F. Sancho Caparrini, *Entrenamiento de Redes Neuronales: mejorando el Gradiente Descendente*, Accedido: 2023-04-02, 2020. URL: https://www.cs.us.es/~fsancho/?e=165.

[39] Narendra, K. S. and Parthasarathy K., "Identification and control of dynamical systems using neural networks," 1990. DOI: 10.1109/72.80202.

[40] *Design Time Series NARX Feedback Neural Networks*, Accedido: 2022-05-21. URL: https://la.mathworks.com/help/deeplearning/ug/design-time-series-narx-feedback-neural-networks.html.

[41] *Time Series Forecasting Using Deep Learning*, Accedido: 2022-05-21. URL: https://www.mathworks.com/help/deeplearning/ug/time-series-forecasting-using-deep-learning.html.

[42] J. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," abr. de 1991.

[43] R. Pascanu, T. Mikolov e Y. Bengio, *On the difficulty of training Recurrent Neural Networks*, nov. de 2012.

[44] S. Hochreiter y J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, págs. 1735-80, dic. de 1997. DOI: 10.1162/neco.1997.9.8.1735.

[45] E. Million, "The Hadamard Product." URL: https://api.semanticscholar.org/CorpusID:3086933.

[46] K. Cho, B. van Merrienboer, C. Gulcehre et al., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014. arXiv: 1406.1078 [cs.CL].

[47] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink y J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, n.º 10, págs. 2222-2232, oct. de 2017. DOI: 10.1109/tnnls.2016.2582924.

[48] H. Zhen, D. Niu, K. Wang, Y. Shi, Z. Ji y X. Xu, "Photovoltaic Power Forecasting Based on GA improved Bi-LSTM in Microgrid without Meteorological Information," *Energy*, vol. 231, pág. 120 908, mayo de 2021. DOI: 10.1016/j.energy.2021.120908.

[49] Bahdanau, D., Cho, K. and Bengio, Y., "Neural Machine Translation by Jointly Learning to Align and Translate," 2014. DOI: 10.48550/arXiv.1409.0473.

[50] Luong, M., Pham, H. and Manning, C., "Effective Approaches to Attention-based Neural Machine Translation," 2015. DOI: 10.48550/arXiv:1508.04025.

[51] D. Nasien, S. Yuhaniz y H. Haron, "Statistical Learning Theory and Support Vector Machines," *Computer Research and Development, International Conference on*, vol. 0, págs. 760-764, mayo de 2010. DOI: 10.1109/ICCRD.2010.183.

[52] A. Raj, *Unlocking the True Power of Support Vector Regression*, Accedido: 2023-06-14. URL: http://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0.

[53] T. Sharp, *An Introduction to Support Vector Regression (SVR)*, Accedido: 2023-06-14. URL: https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2.

[54] S. Sreenivasa, *Radial Basis Function (RBF) Kernel: The Go-To Kernel*, Accedido: 2023-06-14. URL: https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a.

[55] *scikit-learn SVM Documentation*, Accedido: 2023-06-14. URL: https://scikit-learn.org/stable/modules/svm.html.

[56] B. Y. Goodfellow Ian y C. Aaron, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[57] M. Ansaldo, *When training AI models, is a bigger dataset better?* Accedido: 2023-05-24, 2022. URL: https://www.hpe.com/us/en/insights/articles/when-training-ai-models-is-a-bigger-dataset-better-2207.html.

[58] R. Pramoditha, *On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning*, Accedido: 2023-05-12, 2022. URL: https://towardsdatascience.com/why-do-we-need-a-validation-set-in-addition-to-training-and-test-sets-5cf4a65550e0.

[59] W. Wang, *Quantifying Model Capacity: The VC Dimension*, Accedido: 2023-05-12, 2022. URL: https://towardsdatascience.com/quantifying-model-capacity-the-vc-dimension-d4eb76dd26f7.

[60] A. Budhiraja, *Dropout in (Deep) Machine learning*, Accedido: 2023-05-20, 2016. URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, págs. 1929-1958, jun. de 2014.

[62] Gal, Yarin and Ghahramani, Zoubin, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," 2015. DOI: 10.48550/arXiv.1512.05287.

[63] Semeniuta, S., Severyn, A. and Barth, E., "Recurrent Dropout without Memory Loss," 2016. DOI: 10.48550/arXiv.1603.05118.

[64] *Epoch in Neural Networks*, Accedido: 2023-05-20, 2023. URL: https://www.baeldung.com/cs/epoch-neural-networks.

[65] A. Gupta, *A Comprehensive Guide on Optimizers in Deep Learning*, Accedido: 2023-05-25, 2023. URL: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/.

[66] *Descenso de gradiente*, Accedido: 2023-05-25. URL: https://es.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/what-is-gradient-descent.

[67] *Gradient Descent: An Optimization Technique in Machine Learning*, Accedido: 2023-05-25. URL: https://www.turing.com/kb/gradient-descent-optimization-technique-in-machine-learning.

[68] Santucci, Andreas, *Introduction to Optimization for Machine Learning*, Accedido: 2023-05-21, 2020. URL: https://stanford.edu/~rezab/dao/.

[69] *Batch , Mini Batch and Stochastic gradient descent*, Accedido: 2023-05-21, 2020. URL: https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461.

[70] V. Bushaev, *Understanding RMSprop — faster neural network learning*, Accedido: 2023-05-25, 2018. URL: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a.

[71] Kingma, Diederik and Ba, Jimmy, "Adam: A Method for Stochastic Optimization," 2014. DOI: 10.48550/arXiv.1412.6980.

[72] *Adam Documentation*, Accedido: 2022-05-21. URL: https://keras.io/api/optimizers/adam/.

[73] Hagan, Martin and Menhaj, Mohammad, "Training feedforward networks with the Marquardt algorithm," 1994. DOI: 10.1109/72.329697.

[74] Gavin, Henri, "The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems," 2022.

[75] T. M. Inc., *MATLAB version: 9.13.0 (R2022b)*, Natick, Massachusetts, Estados Unidos, 2022. URL: https://www.mathworks.com.

[76] *MathWorks*, Accedido: 2023-05-25, 2022. URL: https://la.mathworks.com/help/deeplearning/ref/trainlm.html.

[77] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. and Talwalkar, A., "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," 2016. DOI: 10.48550/arXiv.1603.06560.

[78] A. Botchkarev, "A New Typology Design of Performance Metrics to Measure Errors in Machine Learning Regression Algorithms," *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 14, págs. 045-076, 2019. DOI: 10.28945/4184.

[79] S. Gwowen, "Improved Shrinkage Estimation of Squared Multiple Correlation Coefficient and Squared Cross-Validity Coefficient," *Organizational Research Methods*, vol. 11, págs. 387-407, 2008. DOI: 10.1177/1094428106292901.

[80] *Datenum*, Accedido: 2023-05-25, 2022. URL: https://la.mathworks.com/help/matlab/ref/datenum.html.

[81] Murcia, G., González, S. A., Echeverría, N., Garín, E., Branda, J. and Jacob, S., "Efecto de las Condiciones Meteorologicas de Mar del Plata en la Produccion Fotovoltaica," 2017. URL: http://www3.fi.mdp.edu.ar/clagtee/2017/articles/07-037.pdf.

[82] M. LeMone, *How the Temperature Varies During the Day and Night*, Accedido: 2022-03-23, 2008. URL: https://www.globe.gov/explore-science/scientists-blog/archived-posts/sciblog/2008/02/27/how-the-temperature-varies-during-the-day-and-night/comment-page-1/index.html.

[83] National Weather Service, *A Discussion of Water Vapor, Humidity, and Dewpoint, and Relationship to Precipitation*, Accedido: 2022-03-23. URL: https://www.weather.gov/lmk/humidity.

[84] Lima, M., Carvalho, P., Braga, A. Fernández, L. and Leite, J., "MLP Back Propagation Artificial Neural Network for Solar Resource Forecasting in Equatorial Areas," 2018. DOI: 10.24084/repqj16.253.

[85] Parvez, I., Sarwat, A., Debnath, A., Olowu, T., Dastgir, M. and Riggs, H., "Multi-layer Perceptron based Photovoltaic Forecasting for Rooftop PV Applications in Smart Grid," 2020. DOI: 10.1109/SoutheastCon44009.2020.9249681.

[86] Keerthisinghe, C., Mickelson, E., Kirschen, D., Shih, N. and Gibson, S., "Improved PV Forecasts for Capacity Firmingd," 2020. DOI: 10.1109/ACCESS.2020.3016956.

[87] Omar, M., Dolara, A., Magistrati, G., Mussetta, M., Ogliari, E. and Viola, F., "Day-ahead forecasting for photovoltaic power using artificial neural networks ensembles," 2016. DOI: 10.1109/ICRERA.2016.7884513.

[88] Zhu, X., Wang, L., Gu, J., Wang, P., Tang, J., Hao, J., Ziang, M., Lu, Y. and Lu, X., "Ultra-short-term PV prediction based on LSTM with a multi-head attention mechanism," 2023. DOI: 10.1088/1742-6596/2428/1/012013.

[89] G. Van Rossum y F. L. Drake, *Python 3 Reference Manual*, Scotts Valley, CA, 2009.

[90] F. Chollet et al., *Keras*, https://keras.io, 2015.

[91] Martín Abadi, Ashish Agarwal, Paul Barham et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. URL: https://www.tensorflow.org/.

[92] *Matplotlib Documentation*, Accedido: 2022-09-13. URL: https://matplotlib.org/stable/index.html.

[93] Menon, Aishwarya and Dr. Sarathambekai, "Python libraries and packages for Deep Learning - A survey," 2019. URL: https://www.irjet.net/archives/V6/i5/IRJET-V6I5410.pdf.

[94] *SimpleRNN Documentation*, Accedido: 2022-05-21. URL: https://keras.io/api/layers/recurrent_layers/simple_rnn/.

[95] *SysIdentPy Documentation*, Accedido: 2022-05-21. URL: https://sysidentpy.org/.

[96] *PyNeurGen Documentation*, Accedido: 2022-05-21. URL: https://pyneurgen.sourceforge.net/.

[97] *FireTS Repository and Documentation*, Accedido: 2022-05-21. URL: https://github.com/jxx123/fireTS.

[98] *Choose a Multilayer Neural Network Training Function*, Accedido: 2023-05-25, 2022. URL: https://la.mathworks.com/help/deeplearning/ug/choose-a-multilayer-neural-network-training-function.html.

[99] *TensorFlow Timeseries Dataset From Array*, Accedido: 2023-03-12. URL: https://www.tensorflow.org/api_docs/python/tf/keras/utils/timeseries_dataset_from_array.

[100] *LSTM layer*, Accedido: 2023-03-12. URL: https://keras.io/api/layers/recurrent_layers/lstm/.

[101] *GRU layer*, Accedido: 2023-03-12. URL: https://keras.io/api/layers/recurrent_layers/gru/.

[102] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi et al., *Keras Tuner*, https://github.com/keras-team/keras-tuner, 2019.

[103] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh y A. Talwalkar, *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*, 2018. arXiv: 1603.06560 [cs.LG].

[104] *Colaboratory*, https://research.google.com/colaboratory/intl/es/faq.html, Verified: 2023-06-12.

[105] *Kaggle*, https://www.kaggle.com/docs/notebook, Verified: 2023-06-12.

[106] *Cloud Tensor Processing Unit (TPU)*, https://cloud.google.com/tpu/docs/tpus?hl=es-419, Verified: 2023-06-12.

[107] *Cloud Tensor Processing Unit (TPU)*, https://www.kaggle.com/docs/tpu, Verified: 2023-06-12.

[108] Y. E. Wang, G.-Y. Wei y D. Brooks, *Benchmarking TPU, GPU, and CPU Platforms for Deep Learning*, 2019. arXiv: 1907.10701 [cs.LG].

[109] NVIDIA, *NVIDIA cuDNN*, Verified: 2023-06-12. URL: https://developer.nvidia.com/cudnn.

[110] *Precisión mixta*, https://www.tensorflow.org/guide/mixed_precision?hl=es-419, Verified: 2023-06-12.

[111] *XLA: Optimización del compilador para el aprendizaje automático*, https://www.tensorflow.org/xla?hl=es-419, Verified: 2023-06-12.

[112] *Mirrored Strategy*, https://www.tensorflow.org/api_docs/python/tf/distribute/MirroredStrategy, Verified: 2023-06-12.

[113] F. Chollet y F. Chollet, *Deep Learning with Python, Second Edition*. Manning, 2021, ISBN: 9781617296864. URL: https://books.google.com.ar/books?id=XHpKEAAAQBAJ.

[114] *Losses Documentation*, Accedido: 2022-05-21. URL: https://keras.io/api/losses/.

[115] Nguyen, N., Bui, L., Doan, B., Sanseverino, E., Di Cara, D. and Nguyen, Q., "A new method for forecasting energy output of a large-scale solar power plant based on long short-term memory networks a case study in Vietnam," 2021. DOI: 10.1016/j.epsr.2021.107427.

[116] Gao, M., Li, J., Hong, F. and Long, D., "Day-ahead power forecasting in a large-scale photovoltaic plant based on weather classification using LSTM," 2019. DOI: 10.1016/j.energy.2019.07.168.

[117] P. Rémy, *Keras Attention Layer*, 2023. URL: https://github.com/philipperemy/keras-attention.

[118] Keras, *Attention layer*, 2023. URL: https://keras.io/api/layers/attention_layers/attention/.

[119] Alrashidi, Musaed and Rahman, Saifur, "Short-term photovoltaic power production forecasting based on novel hybrid data-driven models," 2023. DOI: 10.1186/s40537-023-00706-7.

[120] Sobri, S., Koohi-Kamali, S. and Rahim, N., "Solar photovoltaic generation forecasting methods: A review," 2017. DOI: 10.1016/j.enconman.2017.11.019.

[121] *Average Monthly Humidity - Mar del Plata*, Accedido: 2023-08-15. URL: https://weather-and-climate.com/average-monthly-Humidity-perc,Mar-Del-Plata,Argentina.

[122] Pawar, P., Mithulananthan, N. and Raza, M., "Solar PV Power Forecasting Using Modified SVR with Gauss-Newton Method," 2020. DOI: 10.1109/GPECOM49333.2020.9247935.

[123] T. pandas development team, *pandas-dev/pandas: Pandas*, ver. 1.4.3, 2022. DOI: 10.5281/zenodo.3509134.

[124] C. R. Harris, K. J. Millman, S. J. van der Walt et al., "Array programming with NumPy," *Nature*, vol. 585, n.º 7825, págs. 357-362, sep. de 2020. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[125] T. Q. Company, *PyQt5*, 2022. URL: https://www.riverbankcomputing.com/software/pyqt/.

[126] *Tkinter*, ver. 8.6.12, 2021. URL: https://docs.python.org/es/3/library/tkinter.html#module-tkinter.

[127] *Unittest framework documentation*, https://docs.python.org/3/library/unittest.html, Verified: 2023-08-12.

[128] N. Batchelder, *Coverage.py*, ver. 7.1.0, 2022. URL: https://github.com/nedbat/coveragepy.

[129] T. Inc, *Trello*, 2011. URL: trello.com.

[130] github, *GitHub*, 2008. URL: https://github.com/.

[131] W. Limited, *Overleaf*, 2013. URL: overleaf.com.

[132] Google, *Google Drive*, 2012. URL: drive.google.com.

[133] Google, *Google Meet*, 2017. URL: meet.google.com.

[134] *SGD Documentation*, Accedido: 2023-05-16. URL: https://keras.io/api/optimizers/sgd/.

[135] *scikit-learn Documentation*, Accedido: 2023-06-14. URL: https://scikit-learn.org/stable/.

[136] Sauer, J., Kramer, O., Parisi, J. and Bremer, J., "Support Vector Regression for Solar Power Prediction," 2017. URL: https://d-nb.info/1138027561/34.

[137] Wolff, B., Kühnert, J., Lorenz, E., Kramer, O. and Heinemann, D., "Comparing support vector regression for PV power forecasting to a physical modeling approach using measurement, numerical weather prediction, and cloud motion data," 2016. DOI: 10.1016/j.solener.2016.05.051.

Apéndice A

Pruebas preliminares

A.1. Pruebas con perceptrón multicapa

A.1.1. Pruebas con $sen(x)$

Se decidió comenzar el análisis con la función seno, dado que se consideró que su continuidad facilitaría las tareas predictivas.

A.1.1.1. Caso base

Para la búsqueda del modelo MLP óptimo, se planteó un caso base o inicial con una arquitectura e hiperparámetros recomendables para el problema determinado [134]. A continuación, se detalla la estructura de la red construida:

- **Arquitectura:** se utilizó una capa de entrada para un único valor (x), seguida por una capa oculta con 100 neuronas y función de activación *tanh*. La capa de salida contó con activación lineal.
- **Conjunto de datos:** en base a una distribución uniforme, se generaron aleatoriamente 1000 puntos en el rango $[-\pi, \pi]$.
- **Estructura de datos**: se decidió utilizar el 10 % de los datos (100 valores) como conjunto de *testing*. El 90 % restante fue dividido de la siguiente manera:
  - **Entrenamiento**: 720 valores (80 %).
  - **Validación**: 180 valores (20 %).
- **Optimización:** se utilizó descenso por gradiente con *mini-batch*, con *learning rate* 0,01. No se utilizó *momentum*.
- **Función de costo:** MSE.
- **Capa *dropout*:** no se utilizó.
- **Tamaño de *batch*:** 32.
- **Duración del entrenamiento:** 500 épocas.

El primer acercamiento logró buenos resultados. Como se puede ver en la figura A.1, el modelo logró generar predicciones con una gran precisión dentro del rango $[-\pi, \pi]$. Para la etapa de entrenamiento, se llegó a un MSE de 0,053, mientras que el error de validación alcanzó un valor de 0,035. A su vez, se obtuvo 0,9897 y 0,9961 para el cálculo de R^2, respectivamente. Por último, en la etapa de *testing*, se obtuvo un MSE de 0,075 y un R^2 de 0,9851.

Posteriormente, se analizó la evolución por época de los errores de entrenamiento y validación del modelo (ver figura A.2). Se observó que el MSE de entrenamiento se redujo hasta alrededor de la época 40, donde la curva converge y comienza a oscilar levemente. Para el caso del MSE de validación, el comportamiento fue aún más pronunciado. Por este motivo, se decidió disminuir la cantidad de épocas de entrenamiento, de manera que la ejecución finalice cerca del punto de convergencia.

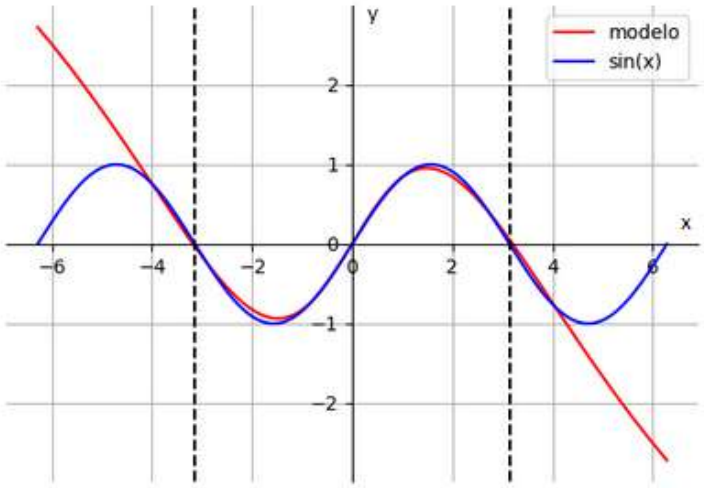


Figura A.1: Gráfico de la predicción de $sen(x)$ con el modelo base, entre $[-\pi, \pi]$.

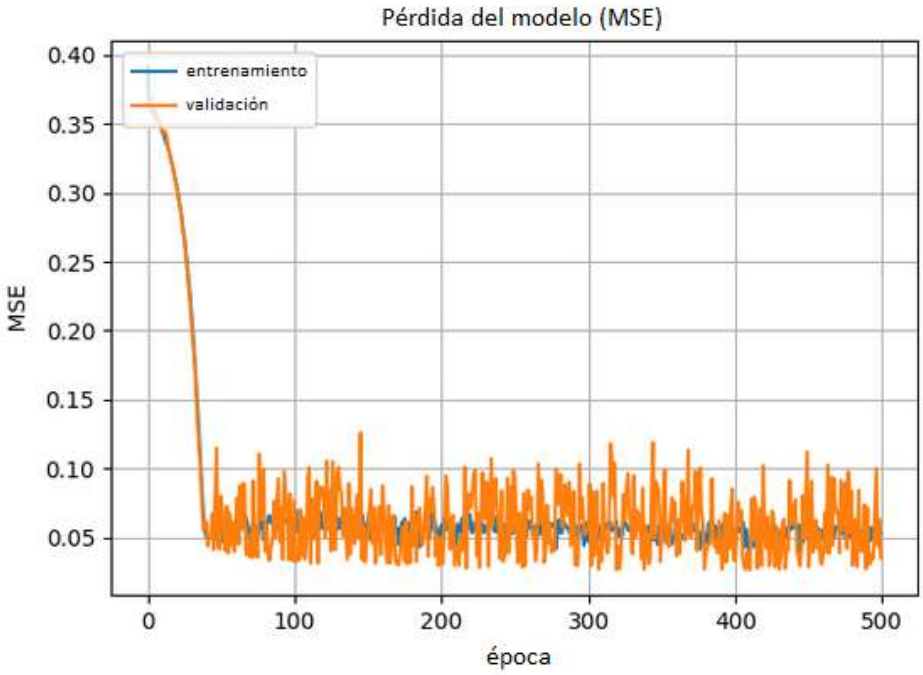


Figura A.2: Gráfico de la pérdida del modelo (MSE) en cada época de entrenamiento para el caso base.

A.1.1.2. Ajuste de hiperparámetros

Una vez finalizado el análisis sobre el caso base, se procedió a experimentar sobre cambios en la arquitectura y los hiperparámetros del modelo. El objetivo fue obtener mejoras sobre los resultados anteriores.

Tras finalizar la experimentación, el modelo con el cual se obtuvieron los resultados más precisos contó con una única capa oculta compuesta por 10 neuronas y función de activación *ReLU*. Además, la capa de salida se mantuvo con activación lineal. En la figura A.3 se puede visualizar la arquitectura del modelo.

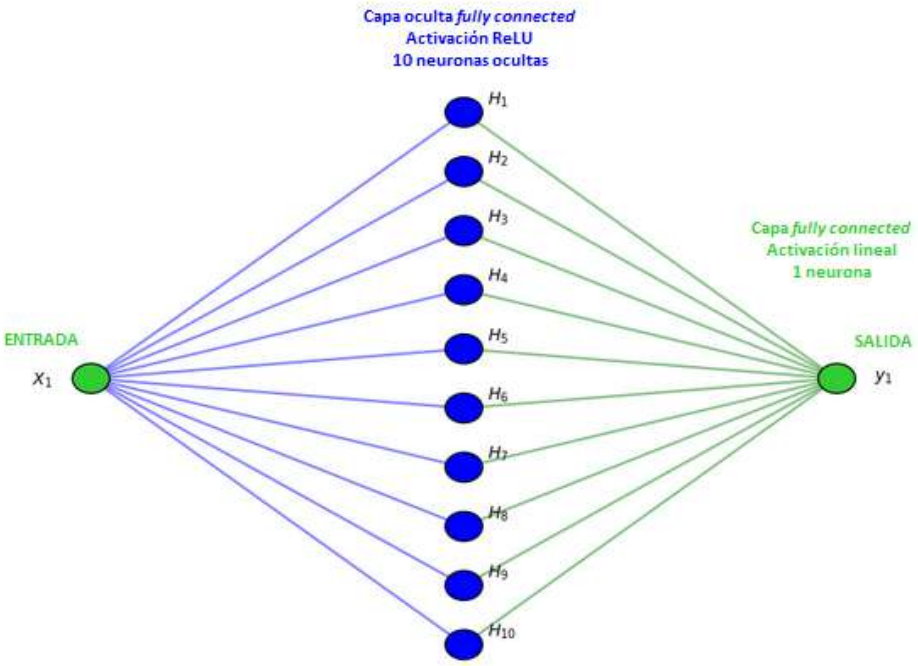


Figura A.3: Estructura del MLP utilizado como modelo final.

El conjunto de datos inicial, el optimizador, el tamaño de *batch* y el resto de hiperparámetros utilizados también se mantuvieron iguales al caso base. No se requirió del uso de una capa *dropout* y el entrenamiento del modelo tuvo una duración de 200 épocas.

Como se puede observar en la figura A.4, alrededor de la época 75 se llega al punto de convergencia de la red y el error se estabiliza. En la etapa de *testing*, se obtuvo un MSE de 0,016 y un R^2 de 0,999.

A continuación, se detalla el proceso de búsqueda previo a hallar la estructura óptima y cómo respondió el modelo a partir de los cambios propuestos:

- **Cantidad de épocas:** disminuir su valor a menos de 30 generó predicciones notablemente peores. Por otro lado, aumentar su cantidad no se tradujo en mejores métricas. El tiempo de entrenamiento creció en base a la cantidad de épocas utilizada.
- **Tamaño de *batch*:** disminuirlo implicó un mayor tiempo al entrenar el modelo. A su vez, al aumentarlo no se logró conseguir buenas predicciones a menos que se también se aumentase la cantidad de épocas.
- **Arquitectura:** se observó que aumentar la cantidad de capas ocultas no mejoraba los resultados y el tiempo de entrenamiento era mayor. El mismo comportamiento fue observado al aumentar la cantidad de neuronas, mientras que al disminuirlas el sistema se tornó inestable.
- **Función de activación:** se estudiaron las funciones lineal, sigmoidea y ReLU, siendo esta la última la que mejores resultados obtuvo.
- **Cantidad de datos:**
  - Disminuir la cantidad de datos generó una clara disminución del tiempo de entrenamiento, pero la precisión de las predicciones se redujo.

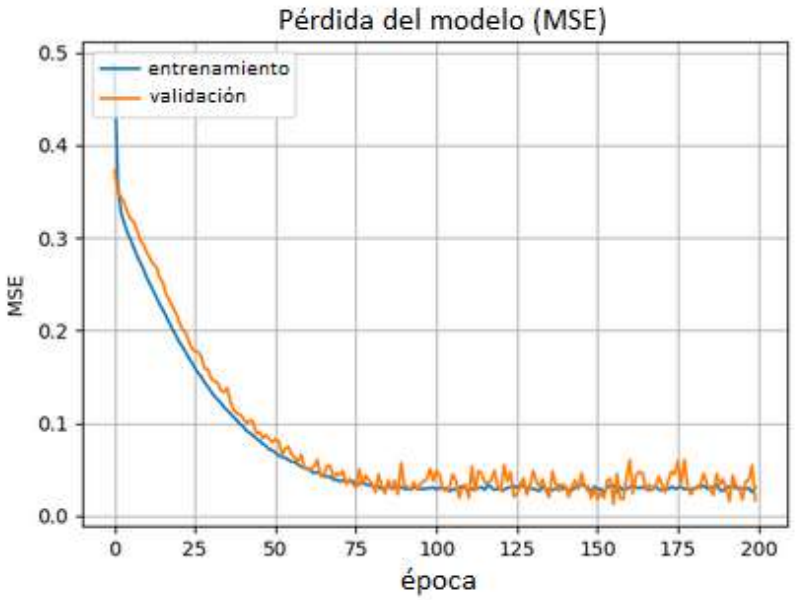


Figura A.4: Gráfico de la pérdida del modelo (MSE) en cada época de entrenamiento para el modelo final.

- Temporalmente, se amplió la cantidad de períodos de la función seno a predecir. Se observó que, a mayor cantidad de períodos, se requirió de un volumen de datos más grande para lograr buenas predicciones. A su vez, también fue necesario aumentar la capacidad del modelo.

- **Optimizadores e hiperparámetros:**
  - Se identificó que el valor ideal para utilizar como *learning rate* era 0,01. Valores menores implicaron un mayor tiempo de entrenamiento y requirieron aumentar la cantidad de épocas. Por otro lado, un aumento generó una convergencia más rápida. Sin embargo, los resultados del modelo comenzaron a experimentar una alta inestabilidad.
  - La inclusión de un valor de 0,9 de *momentum* generó una convergencia veloz, aunque los resultados estuvieron levemente por debajo de los reportados anteriormente (MSE en *testing* de 0,0522 y R^2 de 0,9925).
  - La inclusión de una capa de *dropout* empeoró notablemente los resultados.
  - Se realizaron pruebas con el optimizador *Adam*. Los resultados obtenidos fueron buenos, pero no mejoraron los alcanzados anteriormente.

A.1.1.3. Pruebas de extrapolación

Una vez finalizado el entrenamiento con el conjunto de pruebas, se decidió incursionar en las capacidades de extrapolación del modelo. Al momento de interpolar, el modelo logró obtener excelentes resultados. Sin embargo, debido a las características periódicas del problema, se deseaba contar con una red capaz de predecir más allá del rango observado.

Para este fin, se realizaron pruebas en donde el modelo era entrenado con datos dentro de un rango establecido (por ejemplo, $[-5\pi, 5\pi]$) y posteriormente era testeado con valores fuera del rango.

Debido a que los primeros modelos no lograron extrapolar de la forma esperada, se decidió aumentar progresivamente la cantidad de períodos utilizados durante el entrenamiento, hasta llegar a un máximo de 30. A pesar de los cambios introducidos, ninguna modificación fue suficiente para lograr la extrapolación deseada. En la figura A.5 puede observarse la diferencia entre las predicciones de interpolación y extrapolación.

A.1.2. Pruebas con $tan(x)$

Posteriormente, se propuso realizar pruebas adicionales para generar una predicción de la función $tan(x)$. Se deseaba analizar las capacidades del modelo MLP ante una función que presentase asíntotas

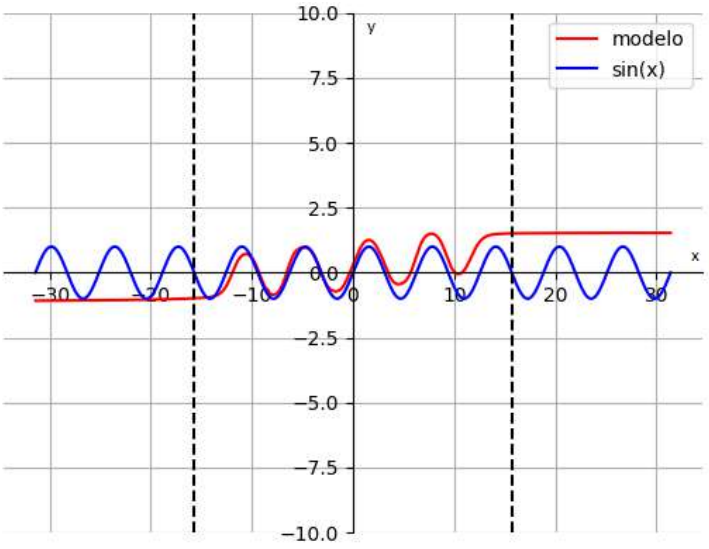


Figura A.5: Gráfico de la predicción de $sen(x)$, con interpolación para $[-5\pi, 5\pi]$ y extrapolación entre $[-10\pi, -5\pi]$ y $[5\pi, 10\pi]$.

en su dominio. De esta manera, se podría observar el comportamiento de la red ante una función de mayor complejidad que la analizada anteriormente.

A modo de ejemplificación, una de las pruebas con la que se obtuvieron los mejores resultados es la representada en la figura A.6. El modelo se compuso de una capa oculta de 100 neuronas y una función de activación $ReLU$. Se entrenó la red con 10000 datos entre $[-\pi, \pi]$.

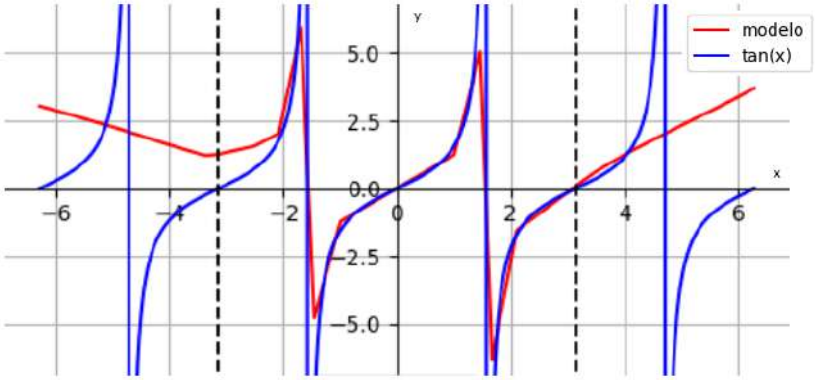


Figura A.6: Gráfico de la predicción de $tan(x)$, con interpolación para$[-\pi, \pi]$ y extrapolación entre $[-2\pi, -\pi]$ y $[\pi, 2\pi]$.

En la etapa de entrenamiento, se llegó a un MSE de 5,3043 y un R^2 de 0,1082. Los valores obtenidos durante durante la validación fueron muy similares (MSE de 4,2692 y R^2 de 0,0868). Posteriormente, se obtuvo un MSE de *testing* de 2,888 y un R^2 de 0,398.

En base a las pruebas realizadas, pudo observarse que la red encontró dificultades a la hora de realizar predicciones en la cercanía de las asíntotas. A su vez, al calcular valores fuera del rango entrenado, el modelo volvió a ser incapaz de extrapolar.

A.2. Pruebas con NARX

A.2.1. PyNeurGen

Mediante el uso de la librería PyNeurGen, se realizaron las primeras pruebas para comenzar a indagar sobre las capacidades de una red NARX. Sin embargo, se encontró que su última actualización data del año 2017 y no cuenta con una versión para Python3. Para lograr trabajar con ella, se debía realizar

pequeñas modificaciones a la librería. A su vez, se carecía de documentación por no ser una librería actualizada y había mucha limitación en cuanto al *tuning* de hiperparámetros. Debido las desventajas mencionadas, se decidió no continuar con su análisis.

A.2.2. SysIdentPy

SysIdentPy es una librería utilizada mayoritariamente para la identificación de sistemas dinámicos. Una de sus ventajas es que permite la personalización de un amplio rango de hiperparámetros. A su vez, otro beneficio considerado es que su versión actual trabaja sobre Python3 y se encuentra en constante actualización por parte de los desarrolladores.

A pesar de las ventajas mencionadas, se presentaron diversos conflictos al momento de desarrollar el modelo. Debido a la carencia de documentación, se dificultó la parametrización de la red, principalmente en el uso de funciones base (característica solo utilizada por esta librería). Una función base permite que el modelo NARX capture diferentes tipos de no linealidades y provee cierta flexibilidad al modelar sistemas complejos. Sin embargo, al no contar con ejemplos claros a la hora de utilizar las funciones, no se logró aprovechar las capacidades de la librería.

A.2.3. FireTS

Posteriormente, se procedió a analizar la última librería recopilada para trabajar con Python. FireTS está basada en la librería scikit-learn [135] y es utilizada para la predicción de series temporales multivariables.

A pesar de que una gran porción de los hiperparámetros de NARX no eran configurables, fue posible construir un modelo y proceder a la etapa de evaluación de las capacidades de extrapolación.

A.2.3.1. Análisis de una función seno con ruido

Antes de comenzar el análisis, se realizó un ajuste a la problemática planteada. Debido a la posible existencia de ruido en las mediciones climáticas, se consideró de relevancia incorporar dicha característica en la función a extrapolar. El conjunto de datos de entrenamiento se generó a partir de la función $f(x) = sen(x)+e$, donde e era un valor modificado durante cada período. Los valores de e se establecieron en el rango $[-0,3, 0, 3]$, con un paso de $0,1$.

FireTS resultó suficiente para comprobar el objetivo planteado. Se detectó que la extrapolación de una función seno con ruido era posible. En todos los casos, se entrenó el modelo con 3500 datos distribuidos en 7 períodos. Al momento de predecir valores fuera del rango entrenado, se ingresaron a la red 3500 entradas entre el 8vo y el 14vo período. Finalmente, como se observa en la figura A.7, el modelo logró predecir una función $f(x) = sen(x)$ casi perfecta para todo el rango de predicción.

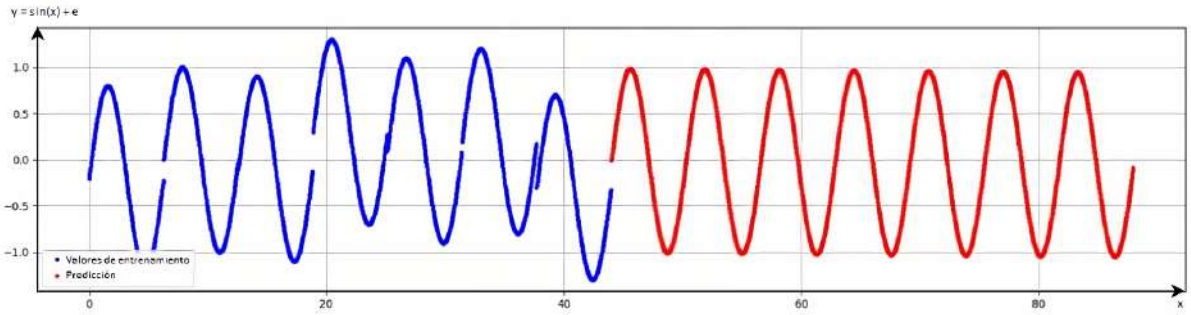


Figura A.7: Predicción de una función seno a partir de un entrenamiento con funciones seno con un error e.

A.2.3.2. Conclusión

A pesar de que las capacidades de FireTS fueron suficientes para resolver el objetivo planteado, se consideró que las opciones vinculadas a la modificación de hiperparámetros era muy limitadas. Por este motivo, se descartó el uso futuro de la librería.

A.2.4. MATLAB

Desde un inicio, se conocía que el sistema MATLAB contiene un módulo específico para trabajar con redes NARX. De todas formas, se prefirió priorizar la profundización sobre las opciones en Python, debido a la experiencia y los desarrollos previos realizados con el lenguaje.

Si bien la transición requirió de un mayor tiempo de trabajo, MATLAB se convirtió en la opción escogida. El entorno ofreció una alta capacidad de personalización y optimización de sus modelos, junto con la disponibilidad de ejemplos para su sencilla implementación.

A.2.4.1. Análisis de una función seno con período variable

En primer lugar, se utilizó MATLAB para replicar las pruebas realizadas con FireTS. Posteriormente, con el objetivo de adquirir aprendizaje adicional, se realizó una nueva modificación sobre el problema predictivo.

En base a la periodicidad variable observada en los datos del ICyTE (ver sección 5.1.2.3), se decidió realizar pruebas para predecir una función $f(x) = sen(\alpha x)$, donde α fue modificado a lo largo de la generación de datos de entrenamiento.

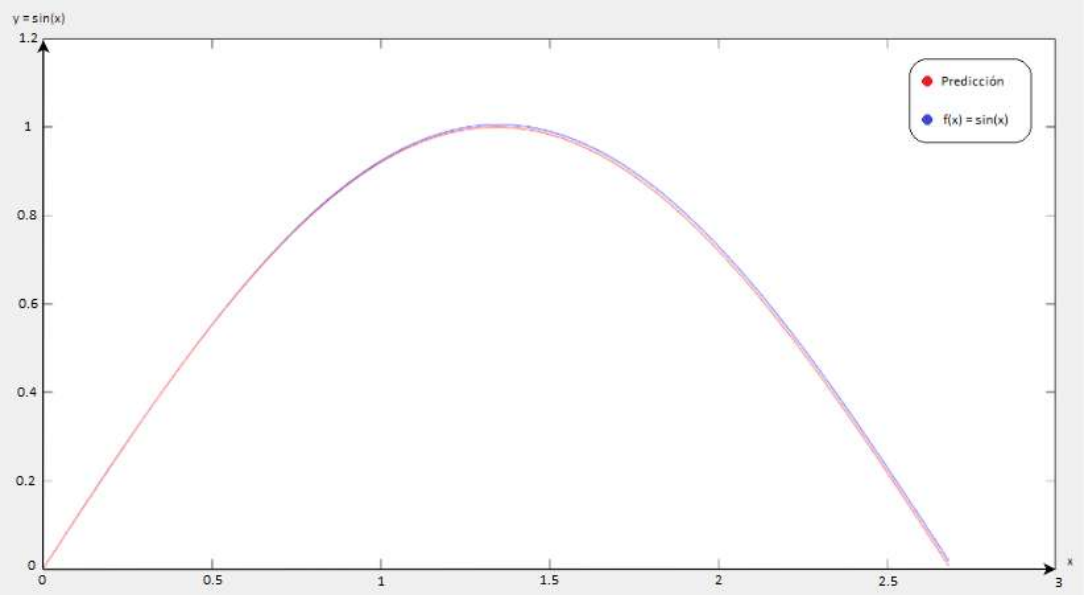


Figura A.8: Extrapolación de una función seno con el entorno MATLAB.

Luego de realizar diversas pruebas, los resultados fueron exitosos y fue posible realizar la extrapolación de la función seno (ver figura A.8). Debido a las conclusiones obtenidas con respecto a las librerías utilizadas en Python y las diversas ventajas de la utilización del entorno MATLAB, se decidió continuar con este último para la ejecución de pruebas sobre los datos otorgados por el ICyTE.

Apéndice B

Experimentación con SVR

Como complemento de la fase de experimentación, se estudiaron las capacidades de la arquitectura SVR para la predicción de potencia eléctrica. Para su implementación, se decidió utilizar la librería scikit-learn [135] de Python. De las opciones analizadas, se la consideró la más completa y aquella con mejor disponibilidad de documentación.

Inicialmente, se llevó a cabo una investigación acerca de trabajos relacionados a la predicción de potencia eléctrica que hayan utilizado SVR como modelo predictivo [136] [137]. Se encontró que, como se mencionó previamente, el *kernel* más referido es RBF. A su vez, los valores más utilizados para cada uno de sus hiperparámetros son los siguientes:

- γ: $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$
- C: $[1, 10, 100, 1000, 10000, 100000]$
- ϵ: $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$

Debido al análisis realizado sobre el estado del arte para SVR, se decidió no incursionar en otros tipos de *kernel*.

B.1. Entrenamiento de un año

Se planteó el siguiente modelo para comenzar el análisis:

- **Conjunto de datos:** al igual que en otras arquitecturas, se utilizó el año 2019 para entrenar el modelo y el mes de enero de 2020 para la etapa de *testing*. Además, se definió que el conjunto de validación sea conformado por los meses de noviembre y diciembre.
- **Ingreso de secuencias:** se ajustaron las entradas del modelo para permitir el ingreso de secuencias, al igual que en las redes LSTM y GRU. Se determinó su longitud inicial en 20, dado que había sido uno de los valores con mejores resultados en experimentaciones anteriores.
- **Variables de entrada:** debido a la experiencia adquirida durante la experimentación con otros modelos, se decidió obviar la etapa de selección de variables. Desde un inicio, las pruebas incluyeron al conjunto completo (irradiancia, humedad, temperatura del aire, velocidad del viento y declinación).
- **Hiperparámetros iniciales:**
  - ϵ: 0,1
  - γ: 0,1
  - C: 1

Con el modelo inicial, se obtuvo un MAPE promedio diario del 17,18 %. A continuación, se procedió a realizar un ajuste de hiperparámetros con el objetivo de mejorar las métricas reportadas. Los mejores resultados fueron obtenidos a partir del uso de $C = 1$ o $C = 10$ y con γ de 0,1 o 0,001. En cuanto a ϵ, se optimizaron los resultados al utilizar valores bajos.

Finalmente, el mejor modelo obtenido logró minimizar el MAPE promedio diario del mes de enero a un 11,04 %, resultados similares a los reportados con NARX. Los hiperparámetros utilizados fueron $C = 10$, $\gamma = 0,01$ y $\epsilon = 0,000001$.

B.2. Entrenamiento completo

Para evaluar el uso futuro de SVR, se decidió realizar un conjunto de pruebas finales que incluyan un mayor volumen de datos. La experimentación consistió en reentrenar a los 5 mejores modelos obtenidos anteriormente, utilizando el período completo de datos disponibles (2016 a 2019). Los conjuntos de validación y *testing* se mantuvieron iguales.

Para cada uno de los modelos, se observó un aumento del MAPE en comparación al reportado en la etapa previa. Las pruebas realizadas no mostraron mejoras y nuevamente se asemejaron a los resultados obtenidos con NARX, que ya habían sido superados por los modelos de *Deep Learning*.

B.3. Conclusión

Si bien los resultados obtenidos con los modelos SVR fueron considerados aceptables, no lograron alcanzar los reportados por LSTM y GRU. Ante la falta de mejoras, se decidió descartar el uso futuro de la arquitectura y se centraron los esfuerzos en el *tuning* de los modelos de *Deep Learning*.