



TÓPICOS DE GEOFÍSICA II

513437-1

Informe de resultados

**Pronóstico de producción de energía
fotovoltaica a corto-plazo mediante redes
neuronales artificiales.**

Tamara Paz García Latorre

Matrícula: (2020427330)

Profesor: Héctor Hito Andrés Sepúlveda Allende

18 de diciembre, 2023.

Índice

Resumen	2
Introducción	3
Marco teórico	4
Energía fotovoltaica	4
Modelos de pronóstico	6
Redes neuronales	7
Error de predicción	9
Overfitting	10
Librerías y herramientas esenciales de Python	10
Objetivos	12
Objetivo general	12
Objetivos específicos	12
Metodología	13
Pre-procesamiento de los datos	13
Construyendo la red neuronal	18
Post-procesamiento de datos	23
Resultados	24
Modelo de Red Neuronal Recurrente	24
Modelo de Red Neuronal Long-Short Term Memory	26
Discusión	29
Conclusión	30
Referencias	31

Resumen

Mediante el transcurso del curso y elaboración de este informe se espera conocer y entender el funcionamiento base de las redes neuronales (RN) junto con su aplicación en el contexto de pronóstico de energía solar fotovoltaica a corto plazo utilizando una variable meteorológica independiente. También, inspeccionar las problemáticas que conlleva utilizar energía solar fotovoltaica (PV) para inyectarla a la red eléctrica y qué ventajas puede aportar el uso de técnicas de inteligencia artificial, en específico redes neuronales recurrentes en este ámbito para el funcionamiento del sistema.

Como este tópico es un primer acercamiento al tema es necesario introducir y entender ciertos conceptos y metodologías de la materia. En primera instancia se busca crear o adaptar un código base en Python de una red neuronal para datos de energía PV de salida y cobertura nubosa en una zona específica para realizar un pronóstico a corto plazo (1 día) de la producción de energía del sistema.

Introducción

La producción de energía es uno de los pilares fundamentales en el desarrollo y la sostenibilidad de un país. En la actualidad, Chile depende de diferentes fuentes de producción eléctrica, entre ellas, el gas natural, el petróleo y la energía hidroeléctrica las cuales abastecen en diferente medida la red nacional. Debido a la gran diversidad geográfica y a una creciente demanda energética, la transición hacia fuentes de energía renovable en el contexto de la lucha contra el cambio climático plantea diversas problemáticas que deben abordarse. Una de estas es el aumento significativo en el **uso de la energía fotovoltaica (PV)** que trae consigo la **necesidad de prever cuánta energía se generará** en cada instante de tiempo para su incorporación en la red eléctrica (1), la planificación operativa, el uso de reservas, adaptación de cargas máximas (2), una gestión eficiente y para promover su integración a gran escala (3). Siendo crucial la **predicción a corto plazo** debido a que la producción de energía PV depende de las **condiciones meteorológicas** que varían en cada instante y además es necesario adaptarse a la potencia requerida de la red.

Lo que trae consigo, la elección de un **modelo de pronóstico a corto plazo basado en redes neuronales**, siendo este uno de los métodos con mayor precisión dada la flexibilidad de la estructura (4), su capacidad de aprender de datos históricos y adaptación ante patrones cambiantes (5). Permitiendo utilizarlo para la predicción en tiempo real sin necesidad de tener información del sistema, resolver ecuaciones físicas complejas o suponer condiciones ideales, como lo hacen los modelos meteorológicos que no logran responder todas las incertidumbres relacionadas a las variables (3).

Adentrándonos más en el tema, el horizonte de pronóstico a corto plazo se refiere a la obtención de valores de producción de energía PV esperados en un rango de tiempo específico que va desde 1 hora hasta los 7 días (3). Por otro lado, el modelo de pronóstico se basa en el método de redes neuronales recurrentes, el cual está construido en **Python** (6) y el objetivo consiste en obtener valores de energía fotovoltaica de salida utilizando datos históricos de la misma y variables meteorológicas, tal como la cobertura nubosa. Una red neuronal trata de imitar el funcionamiento de las neuronas biológicas y en aspectos generales trabaja de la siguiente forma:

- Se constituye por una serie de capas, que dentro de ellas contienen neuronas y están conectadas entre sí mediante pesos ajustables.
- La estructura se divide en 3; capa de entrada, en donde se ingresan los inputs; capas ocultas en donde se produce el aprendizaje y ajuste; capa de salida, en la que se entrega el output o valor estimado.
- Se le entrega un conjunto de datos al modelo el cual es dividido en set de entrenamiento y testeo (cada uno cuenta con datos de las variables dependientes e independientes). Con el primero la red empieza a cambiar los valores de los enlaces o pesos entre las capas para aproximarse al valor esperado de salida. Con el segundo, se puede evaluar la precisión del valor de salida con el valor esperado y así conocer el error.

Existen también distintos tipos de redes neuronales, en este caso se utiliza una red neuronal recurrente (RNN) que se identifican por sus bucles de retroalimentación. Estos algoritmos de aprendizaje se utilizan principalmente con datos de series temporales para hacer predicciones sobre resultados futuros (7). Una gran dificultad es la elección de los parámetros que construyen la red, mayores detalles sobre el tema se dan a conocer en la sección de “Marco teórico”.

Los datos utilizados son promedios diarios de energía solar (no se especifica su unidad) del Parque solar Urbana-Champaign (tampoco se especifica su ubicación exacta) y la cobertura nubosa tomados de la plataforma **GitHub** del trabajo “Machine Learning for Solar Energy Prediction” realizado por Adele Kuzmiakova, Gael Colas y Alex McKeethan, estudiantes de posgrado de la Universidad de Stanford (8). Como el objetivo es familiarizarse con la metodología no es relevante el conjunto de datos que se utilicen. En este caso los datos de energía PV son dependientes del valor de la cobertura nubosa, siendo esta última la variable independiente con rango de 0 a 1, indicando un día sin nubosidad o nublado respectivamente.

Marco Teórico

Energía fotovoltaica

La energía solar PV es una fuente de energía limpia y sostenible. No produce emisiones de gases de efecto invernadero durante su operación y ayuda a reducir la dependencia de fuentes de energía no renovables.

Este tipo de energía consiste en captar la radiación solar directa que llega desde el sol a la superficie terrestre usándola para producir energía eléctrica mediante el efecto fotovoltaico, el cual ocurre cuando electrones de valencia de átomos de materiales semiconductores presentan cierta ligazón con el núcleo y son arrancados por la energía de los fotones o radiación incidente. Esto a través del uso de celdas solares fabricadas con materiales semiconductores cristalinos, como el silicio, que se conectan en serie y en paralelo logrando determinados niveles de tensión e intensidad eléctrica. Existen distintas aplicaciones de la energía producida, siendo una de ellas la inyección directa a la red eléctrica (9).

En los siguientes puntos se explica de forma general cómo funciona y se utiliza el efecto fotovoltaico.

- Las celdas solares son fabricadas con materiales semiconductores, como el silicio, estos átomos tienen su orbital externo incompleto con solo cuatro electrones denominados electrones de valencia. Estos átomos forman una red cristalina en la que cada uno comparte sus cuatro electrones de valencia con los cuatro átomos vecinos formando enlaces covalentes. Cualquier aporte de energía, como una elevación de la temperatura o la iluminación del semiconductor, provoca que algunos electrones de valencia absorban suficiente energía para librarse del enlace covalente y moverse a través de la red cristalina, convirtiéndose en electrones libres (10).
- Cuando un electrón abandona el átomo de un cristal de silicio, deja en la red cristalina una vacante (hueco). Así, cuando un fotón incide sobre un semiconductor, si tiene suficiente energía, genera un par electrón-hueco.
- Para que este semiconductor pueda ser un productor de energía se introducen impurezas (átomos) en el silicio, proceso denominado “dopaje” con el fin de cambiar sus propiedades eléctricas (11).

Estos átomos añadidos son de dos tipos; Los que tienen un electrón más que el silicio en su capa de valencia externa (como el fósforo), que formarán la zona tipo N, y aquellos que tienen un electrón menos que el silicio en su capa de valencia externa (como el boro), que forman la zona tipo P. En la figura (1) se puede observar esta estructura de la celda solar y las capas.

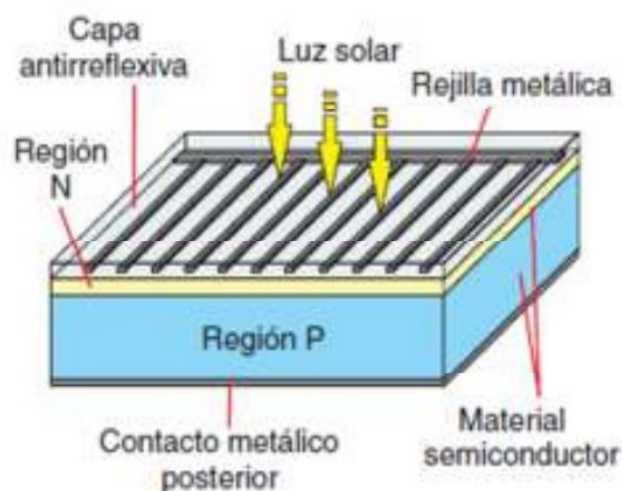


Figura 1: Célula solar convencional construida a partir de una oblea de material semiconductor como el silicio en la que se ha difundido boro (región P) y sobre la que se difunde una capa muy fina de fósforo (región N), para obtener una unión P-N (12)

- La celda solar de este modo tiene esta estructura de capas; la capa P contiene átomos de con deficiencia de electrones (huecos), mientras que la capa N tiene un exceso de electrones. La interfaz entre estas capas se llama una unión P-N.
- Cuando los fotones de la luz solar inciden sobre la estructura, pueden ser absorbidos por los átomos en la capa N, proporcionando suficiente energía para liberar electrones. Este proceso se llama excitación y se crean pares electrón-hueco en la región de la unión P-N. Los electrones liberados se mueven hacia la capa N y los huecos se mueven hacia la capa P.
- Cuando el semiconductor P-N absorbe la luz de suficiente energía (en el caso del silicio, luz visible e infrarroja corta), cada fotón produce un electrón negativo y un hueco positivo. Al existir una barrera de potencial en la unión, los electrones producidos en la capa N se mueven hacia el electrodo y los huecos positivos se mueven hacia el otro electrodo. Cuando los electrones y los agujeros se concentran en los electrones se produce una potencia eléctrica positiva, y si se unen los electrodos con un conductor, una corriente eléctrica circulará por este.

La estructura base de los sistemas fotovoltaicos como se presenta en la figura (2) consisten en un subsistema de captación constituido por un panel fotovoltaico que convierte la radiación solar en electricidad, un subsistema de almacenamiento con un acumulador o batería que guarda la energía que no esta siendo utilizada por el consumidor, un subsistema de regulación que evita que las baterías tengan una sobrecarga y un subsistema convertidor de corriente que adapta la energía producida de tipo continuo a alterna si es necesario mediante un inversor (9).

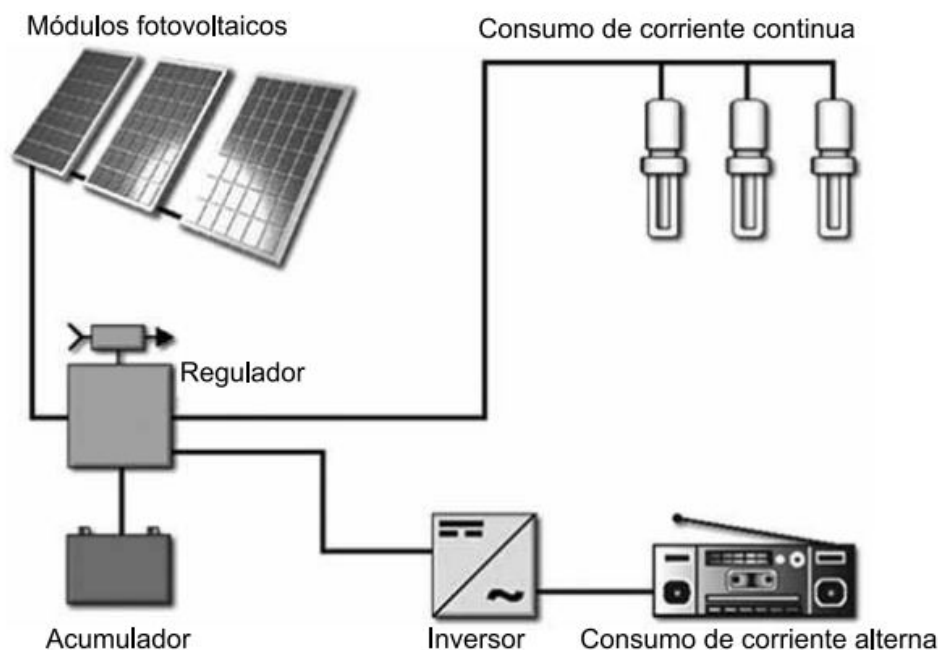


Figura 2: Esquema conceptual de un sistema de energía solar fotovoltaica (9).

Pese a que la radiación solar puede considerarse dispersa es muy abundante, siendo una fuente primaria gratis, limpia e inagotable (a escala humana) de energía (13). Actualmente la eficiencia de conversión de los paneles ronda el 20 % (5) habiendo aumentado más del doble comparada con una eficiencia del 7 % que tenían los primeros sistemas fotovoltaicos construidos (9).

Sin embargo, uno de los problemas que presenta este proceso es la característica intrínseca de producción de potencia intermitente dependiente de las condiciones atmosféricas, especialmente de la radiación solar y la temperatura ambiente. Aquella generación no controlada puede causar problemas de control y operación en las redes eléctricas, dado que estas tienen que lidiar frecuentemente con rápidos excesos o caídas en cuanto a la producción generada. Una posible solución a esto es aplicar modelos de pronóstico de la potencia, la cual puede ayudar a monitorizar el rendimiento, detectar fallas y ayudar a escoger el mejor momento el mantenimiento. Para los operadores del sistema, una herramienta de predicción puede

ayudar tanto al pre-despacho (horizonte 24 h) como a la estabilidad del sistema (predicciones intra-diarias, generalmente 6 h) (5).

Modelos de pronóstico

Un modelo de pronóstico sirve como una herramienta con la cual se predice algún evento o valor futuro en función de datos, patrones históricos o estadísticos. Existen diversos métodos para predecir, en la figura (3) se observa una clasificación general en la cual los modelos físicos utilizan ecuaciones matemáticas que describen el estado físico y la dinámica de la atmósfera, los modelos estadísticos predicen la variable de salida mediante un análisis estadístico de las series temporales, y por último, los modelos híbridos combinan dos o más técnicas para aprovechar las ventajas de las diversas técnicas.

Dentro de la segunda categoría se encuentra el aprendizaje de máquina o *Machine learning*, el cual se centra en el desarrollo de algoritmos y modelos informáticos que pueden aprender y mejorar su rendimiento en tareas específicas a partir de datos y experiencia previa. De acá nace la subcategoría de **Redes neuronales** a utilizar, que se basan en imitar el funcionamiento de las neuronas biológicas y entre sus ventajas destaca el uso de patrones de datos no lineales y aprendizaje mediante experiencia sin la necesidad de resolver ecuaciones complejas.

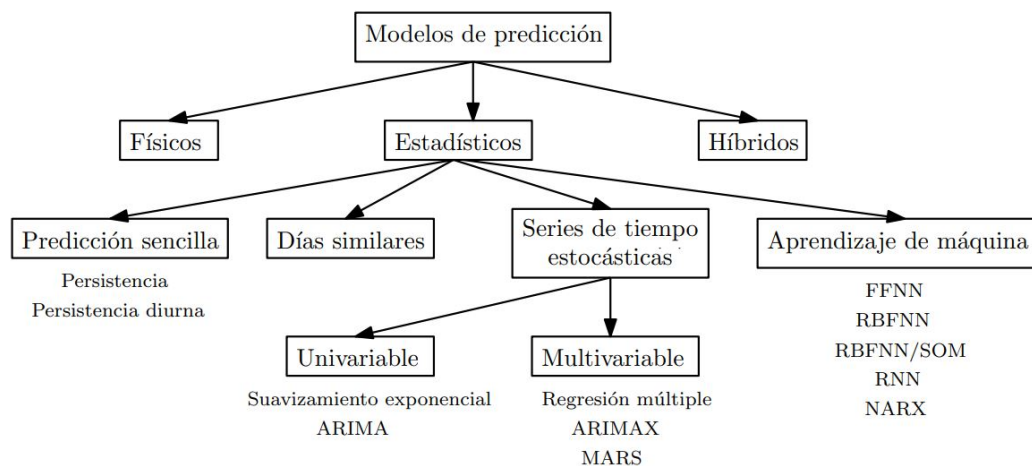


Figura 3: Distintos métodos para generar pronósticos de energía fotovoltaica. Obtenido de Carrasco Catalán, N. A. (2017). *Predicción de potencia a corto plazo para plantas fotovoltaicas utilizando redes neuronales artificiales (Tesis de pregrado)* (5).

El *Machine learning* comprende gran variedad de algoritmos con diversas clasificaciones según sus características, una de ellas comprende el aprendizaje supervisado o no supervisado utilizado por el modelo que son explicadas a continuación.

- **Aprendizaje supervisado:** Se le presenta a la red una serie de vectores de entrada y salida. La red procesa el vector de entrada y la salida obtenida se compara con la salida deseada. Luego se modifican los pesos para hacer que los outputs sean lo más similares posible. En otras palabras, en este modo de aprendizaje se conocen las clases o valores de salida deseados.

De acá se desprenden dos sub-categorías más para la etapa de entrenamiento del modelo; **Incremental**, en cada patrón mostrado a la red evalúa la salida y se compara con la deseada. De este modo se ajustan los pesos de las conexiones para minimizar el error, pasando al siguiente patrón y así sucesivamente hasta que todos los pesos sinápticos se estabilicen en torno a valores óptimos; **Por lote**, se le presenta al modelo un conjunto con los pares de entrada y salida haciendo que los pesos cambien en cada iteración hasta llegar a una red óptima.

- **Aprendizaje no supervisado:** Se caracteriza por no tener salidas deseadas para los datos de entrada. El objetivo de este aprendizaje es descubrir relaciones y estructuras entre los datos que se le presentan a la red, sin presentar algún criterio o ayuda externa que pueda ayudar a la clasificación.

Los modelos a parte del método de pronóstico también se clasifican según el **Horizonte de pronóstico**, es decir, según el periodo de tiempo en el futuro en que son pronosticados los valores de salida, tal como la producción de energía PV (14). Existen distintos rangos del período de tiempo para la categorización, pero por lo general se dividen en cuatro:

1. **Muy corto plazo:** Pronósticos que son menores al rango de 1 hora, por ejemplo 10, 15, 30 minutos. Este tipo de pronóstico se ha realizado para suavización de energía, despacho de electricidad en tiempo real y reservas óptimas.
2. **Corto plazo:** Pronóstico desde 1 hora hasta 7 días. Estos pueden garantizar el compromiso unitario, la programación y el despacho de la energía eléctrica. También mejora la seguridad de la operación de la red.
3. **Medio plazo:** Pronóstico desde los 7-30 días. Este facilita la planificación del sistema eléctrico y el programa de mantenimiento al predecir la disponibilidad de energía eléctrica en el futuro.
4. **Largo plazo:** Pronóstico desde 1 mes hasta 1 año. Es útil para la planificación de la organización de generación, transmisión y distribución de electricidad, además de la licitación de energía y la seguridad de la operación.

Por otro lado, es crucial considerar el uso de variables que presenten cierta correlación. Se recomienda preprocesar el conjunto de datos para aumentar la precisión del modelo. Algunos métodos comunes de preprocesamiento incluyen la normalización o la aplicación de la transformada de Wavelet. Si se elige aplicar alguna de estas técnicas, es importante realizar también un postprocesamiento después de obtener los resultados del modelo. Por ejemplo, si se aplica la transformada de Wavelet, se sugiere posteriormente llevar a cabo una reconstrucción de Wavelet. Este paso adicional facilita el análisis del rendimiento del modelo. (14).

Redes neuronales

Las redes neuronales son un conjunto de algoritmos, que emulan el funcionamiento de las neuronas biológicas. Estas son utilizadas para realizar tareas específicas de aprendizaje automático y reconocimiento de patrones, tienen alta adaptabilidad al entorno, aprenden de la experiencia y pueden generalizar. Las redes están compuestas por unidades llamadas “nodos” o “neuronas”, organizadas en capas e interconectadas entre sí mediante pesos sinápticos ajustables. Cada neurona de una capa esta conectada a todas las neuronas de la siguiente capa, en los pesos se modifica la importancia de la información transmitida a neuronas de la capa siguiente.

La estructura general se divide en 3 partes, que se puede observar en la figura (4). El lado derecho representa un diagrama en cuanto a la estructura en sí del modelo, el cual consiste en una **capa de entrada** donde ingresan las variables independientes, las **capas ocultas** donde se procesa la información, se reconocen patrones y modifican los pesos, y por último, la **capa de salida** en la que se entrega el valor pronosticado. El lado derecho de la figura corresponde a una representación matemática del proceso que ocurre en una capa de la red neuronal, su explicación se da a continuación.

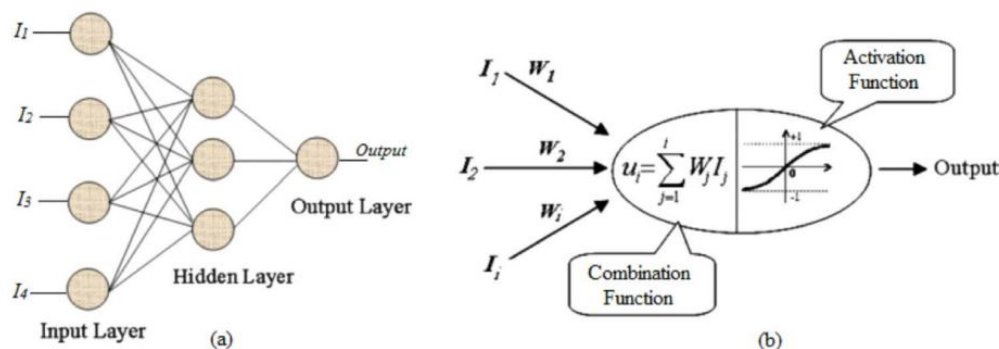


Figura 4: (a) Diagrama esquemático de la estructura de una red neuronal, que consiste en una capa de entrada, capa oculta y capa de salida. (b) Modelo matemático de una célula de red neuronal. Obtenido de *Forecasting of photovoltaic power generation and model optimization: A review* (14)

En primer lugar, a una capa ingresan las entradas de la variable independiente del set de datos. Aquí, los valores se multiplican por el peso sináptico, al que se suma un término que representa el sesgo. Luego, se realiza una sumatoria de todos ellos mediante una **función de combinación** (inicialmente, los pesos son aleatorios). Posteriormente, se aplica una **función de activación** que determina la salida de una neurona después de recibir la suma ponderada de sus entradas. Esta función introduce no linealidades en el modelo, lo que permite a la red aprender y representar patrones complejos en los datos. La salida de esta capa se convierte en la entrada para la siguiente capa, y este proceso se repite hasta llegar a la capa de salida final, donde el resultado es el valor pronosticado.

En términos simples, cada neurona en la red está conectada a otras, y cada conexión tiene asignado un **peso** y un **umbral**. Cuando la salida de una neurona supera el umbral especificado, la neurona se activa y envía datos a la siguiente capa de la red. En caso contrario, no se transmiten datos a la siguiente capa. A medida que el modelo ajusta sus pesos y sesgos, utiliza la **función de pérdida**, que mide la discrepancia entre la predicción y la verdad conocida. La red emplea el algoritmo de **retropropagación** para ajustar los pesos y sesgos con el objetivo de minimizar la función de pérdida, indicando la dirección para reducir los errores. Este proceso implica el cálculo de gradientes para buscar el punto de convergencia o mínimo local. Con cada iteración del proceso de entrenamiento, los parámetros del modelo se ajustan gradualmente para converger hacia el mínimo. (15)

Las redes neuronales han demostrado ser herramientas poderosas en diversas aplicaciones, pero es importante tener en cuenta que su éxito a menudo depende de la calidad y cantidad de datos utilizados para su entrenamiento, así como de la elección adecuada de su arquitectura y parámetros.

Respecto a la construcción de una red neuronal, el número de neuronas y capas que se utilizan deben ser elegidos según su conveniencia y la tarea a resolver. El proceso de aprendizaje empieza con la división del set de datos en dos partes, set de entrenamiento y testeo (por lo general, se dejan en un porcentaje de 80 % y 20 %). con el **set de entrenamiento**, la red ajusta sus pesos en función de la diferencia entre la predicción actual y la salida deseado. Esto se hace utilizando un algoritmo de optimización que minimiza la función de pérdida o error. El algoritmo de retropropagación es utilizado para calcular cómo los pesos deben ser ajustados en todas las capas de la red para reducir la pérdida. Se propaga el error desde la capa de salida hacia atrás a través de la red, y los pesos se actualizan en consecuencia.

Este proceso se repite a lo largo de múltiples iteraciones (épocas) hasta que la red neuronal alcanza un rendimiento deseado en la tarea específica para la cual fue entrenada. Por último, se utiliza el **set de testeo** para obtener la posición del modelo, estos valores al no proceder del set de entrenamiento hacen que se pueda estimar el error entre los datos de salida y los esperados, pues es información nueva con la que la red no ajustó sus pesos.

A continuación, se describen algunas consideraciones adicionales clave para el aprendizaje del modelo:

1. **Normalización de Datos:** Antes de alimentar los datos a la red neuronal, es una práctica común normalizarlos. La normalización ayuda a estabilizar el proceso de aprendizaje al garantizar que las características estén en una escala comparable, evitando que ciertas características dominen sobre otras.
2. **Elección de Arquitectura de Red:** La arquitectura de la red, incluido el número de capas y neuronas en cada capa, es un factor crítico en el rendimiento de la red. La arquitectura debe ser elegida según la complejidad de la tarea y la cantidad de datos disponibles. Experimentar con diferentes arquitecturas puede ser necesario para encontrar la más adecuada (14).
3. **Hiperparámetros:** Ajustar cuidadosamente los hiperparámetros es esencial. Elementos como la tasa de aprendizaje, el tamaño del lote y la tasa de regularización deben ser sintonizados para optimizar el rendimiento de la red, mayor detalles de estos se dan más adelante.
4. **Inicialización de Pesos:** La inicialización de pesos es crucial para el éxito del entrenamiento. Se deben elegir métodos de inicialización adecuados para evitar problemas como el estancamiento del aprendizaje, en general, se utilizan pesos iniciales aleatorios o con distribución normal.

Existen diferentes estructuras de redes neuronales, y su elección depende en gran medida de la tarea específica que se va a realizar. Aquí se describen algunas de las arquitecturas más comunes:

- **Redes neuronales artificiales (ANN):** Son la forma más básica de redes neuronales. Se componen de capas de entrada, capas ocultas y capas de salida. Cada conexión entre las neuronas tiene un peso que se ajusta durante el proceso de entrenamiento. Las ANN son versátiles y pueden aplicarse a una variedad de tareas.
- **Redes neuronales convolucionales (CNN):** Diseñadas especialmente para procesar datos de cuadrícula, como imágenes, las CNN son altamente eficientes en la identificación y extracción de características espaciales. Utilizan filtros convolucionales para analizar regiones locales de la entrada, permitiendo un procesamiento más eficiente de datos multidimensionales como imágenes.
- **Redes neuronales recurrentes (RNN):** A diferencia de las ANN, las RNN poseen conexiones que forman bucles, permitiendo la retroalimentación de la información. Esto las hace aptas para el procesamiento de secuencias temporales. Un subgrupo que deriva de este son las redes neuronales LSTM (Long Short-Term Memory) que tienen capacidad para recordar y olvidar información a lo largo del tiempo, lo que las hace particularmente efectivas en la modelización de secuencias largas.

El enfoque central de este informe es la aplicación de Redes Neuronales Recurrentes para el pronóstico en tareas que involucran dependencias temporales. Las RNN, con su capacidad única de capturar patrones secuenciales, se utilizan específicamente para mejorar la precisión en la predicción de eventos futuros.

Error de predicción

El error de predicción, también conocido como error de pronóstico o residuo, representa la discrepancia entre el valor predicho por un modelo y el valor real observado en los datos, representa un valor del **rendimiento del modelo**. Es un indicador crítico que permite evaluar qué tan precisa es una predicción o modelo en comparación con los resultados reales. Comprender y gestionar el error de predicción es esencial en cualquier tarea de pronóstico para mejorar y ajustar continuamente los modelos. También sirve para poder comparar un modelo de predicción con los demás. Existen varias formas de medirlo, algunas de ellas se pueden observar en la tabla a continuación.

Siglas	Nombre	Fórmula
MSE	Error cuadrático medio.	$\frac{1}{N} \sum_{i=1}^N (y_{\text{pred}} - y_{\text{med}})^2$
RMSE	Raíz del error cuadrático medio o distancia media cuadrática mínima.	$\sqrt{\frac{1}{N} \sum_{i=1}^N (y_{\text{pred}} - y_{\text{med}})^2}$
nRMSE	Distancia media cuadrática mínima normalizada.	$\left(\sqrt{\frac{1}{N} \sum_{i=1}^N (y_{\text{pred}} - y_{\text{med}})^2} \right) \cdot 100\% / y_{\text{med}_{\text{máx}}}$
MAE	Error absoluto medio.	$\frac{1}{N} \sum_{i=1}^N y_{\text{pred}} - y_{\text{med}} $
MAPE	El error porcentual absoluto medio.	$\frac{1}{N} \sum_{i=1}^N \frac{ y_{\text{med}} - y_{\text{pred}} }{ y_{\text{med}} } \cdot 100\%$
MRE	Error medio relativo.	$\frac{1}{N} \sum_{i=1}^N \frac{y_{\text{pred}} - y_{\text{med}}}{y_{\text{total}}} \cdot 100\%$
MBE	Sesgo promedio.	$\frac{1}{N} \sum_{i=1}^N (y_{\text{pred}} - y_{\text{med}})$

Figura 5: Fórmulas para los distintos tipos de error de predicción. Obtenido de *Romero Rodríguez, J. M. (2020). Tesis de maestría, Universidad del Norte*(16).

Donde N es el número de observaciones, y_{pred} es la potencia predicha por el modelo, y_{med} es la potencia medida, y_{medmax} es el valor máximo medido de potencia. Finalmente, y_{total} es la capacidad instalada del sistema fotovoltaico. Cabe resaltar que el RMSE es frecuentemente utilizado en la literatura académica a pesar de no ser un porcentaje. En los modelos de potencia fotovoltaica el RMSE tiene las unidades de la potencia (16).

En los siguientes puntos se da una breve descripción de algunos ellos:

1. **Error Cuadrático Medio (MSE):** Calcula la media de las diferencias al cuadrado entre las predicciones y los valores reales. MSE penaliza más fuertemente los errores grandes y es útil cuando se quieren destacar las discrepancias significativas.
2. **Raíz del Error Cuadrático Medio (RMSE):** Es la raíz cuadrada del MSE y tiene la ventaja de estar en la misma escala que los datos originales, facilitando la interpretación.
3. **Error Absoluto Medio (MAE):** Es la media de las diferencias absolutas entre las predicciones y los valores reales. MAE proporciona una medida promedio del error absoluto y es fácil de interpretar.
4. **Error Porcentual Absoluto Medio (MAPE):** Expresa el error como un porcentaje del valor real y es útil cuando se desea evaluar el error en términos relativos.
5. **Error Biased Medio (MBE):** Esta métrica evalúa el sesgo medio establecido del modelo de pronóstico (17). El sesgo se refiere a la tendencia sistemática del modelo a sobrestimar o subestimar consistentemente los valores reales.

Overfitting

El **sobreajuste**, también llamado overfitting, sucede cuando entrenamos un algoritmo demasiado específicamente con ciertos datos para los cuales ya conocemos los resultados deseados. La idea es que el algoritmo pueda aprender lo suficiente durante el entrenamiento para hacer predicciones en otros casos. Queremos que sea como un aprendiz que puede resolver diferentes situaciones, no solo las que ha visto antes.

Sin embargo, si entrenamos demasiado o con datos extraños, el algoritmo puede enfocarse demasiado en detalles específicos de los datos de entrenamiento que no representan bien el problema en general. En este estado, el algoritmo puede funcionar muy bien al responder a ejemplos del conjunto de entrenamiento, pero cuando se enfrenta a nuevos casos, su rendimiento puede empeorar (18).

Librerías en Python

Python cuenta con librerías para la carga de datos, visualización, estadística, procesamiento de imágenes y más. Una de las ventajas principales de este lenguaje de programación es su habilidad para interactuar directamente con el código, usando una terminal u otras herramientas como *Jupyter Notebook*. Por lo que facilita el uso del *Machine Learning* y el análisis de datos pues son procesos fundamentalmente iterativos en los cuales los datos dirigen el análisis, siendo esencial para estos procesos la rápida iteración y fácil interacción (19).

Para quienes no están familiarizados con las librerías a utilizar, se entrega una breve descripción de cada una de ellas:

1. **Pandas:** Es una librería para el manejo y análisis de datos. Está construido a partir de una estructura de datos llamada **DataFrame**, que en palabras simples es una tabla. *Pandas* brinda un gran rango de métodos para modificar y operar sobre esta tabla, además, cada columna puede ser de un tipo distinto de entrada (Enteros, fechas, letras). También tiene la habilidad para leer una gran variedad de formatos de archivos y base de datos, como SQL, Excel y CSV.
2. **Numpy:** Añade una funcionalidad para vectores multidimensionales, funciones matemáticas de gran nivel como operaciones de álgebra lineal y la transformada de Fourier, y generación pseudo-aleatoria de números.

3. **SciPy:** Colección de funciones para la computación científica. provee muchas otras funcionalidades, rutinas de álgebra lineal avanzada, funciones matemáticas de optimización, procesamiento de señales, funciones matemáticas especiales y distribuciones estadísticas.
4. **Matplotlib:** Es la librería principal para gráficos científicos. Provee funciones para realizar visualizaciones con calidad de publicación, como gráficos de línea, histogramas, diagramas de dispersión, etc. Visualizar los datos y diferentes aspectos del análisis puede brindar información importante.
5. **Sklearn:** Es un proyecto de código abierto, el cual contiene un gran número de “estados del arte” de algoritmos de *Machine Learning*, así como documentación de ello. El vector de *NumPy* es la estructura fundamental de datos para su uso.
6. **Keras:** Librería de redes neuronales escrita en Python y compatible con TensorFlow y Theano, se distingue por su facilidad y rapidez en el prototipado gracias a su modularidad y minimalismo. Ofrece soporte nativo para redes neuronales convolucionales y recurrentes, permitiendo la combinación eficiente de ambas arquitecturas en un mismo modelo. Su diseño versátil y extensible facilita a los desarrolladores la experimentación ágil con diversas configuraciones, lo que la convierte en una herramienta poderosa para el desarrollo eficiente de modelos de aprendizaje profundo (20).
7. **Jupyter Notebook:** Ambiente interactivo para correr un código en el navegador. Esto hace fácil incorporar código, texto e imágenes, y todo esto compilado como un “cuaderno”.

Objetivos

Objetivo general

Aplicar técnicas de redes neuronales a la predicción de la energía solar fotovoltaica en Chile.

Objetivos específicos

1. Familiarizarse con las técnicas de RN aplicadas a pronóstico de corto plazo de la energía solar.
2. Adaptar un código de RN existente a la información disponible para un lugar Chile.
3. Evaluar estadísticamente del desempeño de dicho código.
4. Comparar diferentes tipos de RN aplicados al problema de pronóstico.

Metodología

Pre-procesamiento de los datos

En primer lugar es necesario estudiar y visualizar el set de datos que se utilizará lo, esto que permite identificar características u errores que presenten y así, establecer un enfoque para trabajar y desarrollar el problema de forma más eficaz para llevarlo a la inteligencia artificial.

1) Set de datos:

Es recomendable utilizar un set de datos de gran extensión, independiente de la resolución temporal, para que el modelo pueda generalizar de mejor forma e identificar patrones en los datos.

Los datos a utilizar son promedios diarios de la energía solar (no se especifica su unidad) del Parque solar Urbana-Champaign y la cobertura nubosa tomados de la plataforma **GitHub** del trabajo “Machine Learning for Solar Energy Prediction” realizado por Adele Kuzmiakova, Gael Colas y Alex McKeehan, estudiantes de posgrado de la Universidad de Stanford (8). En este caso los datos de energía solar son dependientes del valor de la cobertura nubosa, siendo esta última la variable independiente con rango desde el 0 al 1.

Observar la serie de tiempo de estas variables permite ubicar intervalos donde no existen datos, hay posibles errores de medición, etc. Tales valores debiesen ser eliminados del set. Otro aspecto a considerar son los datos que tiene un valor de 0, pues al ingresar al modelo al producirse multiplicaciones estos no aportarían información para el aprendizaje, por ejemplo el horario nocturno.

2) Series de tiempo:

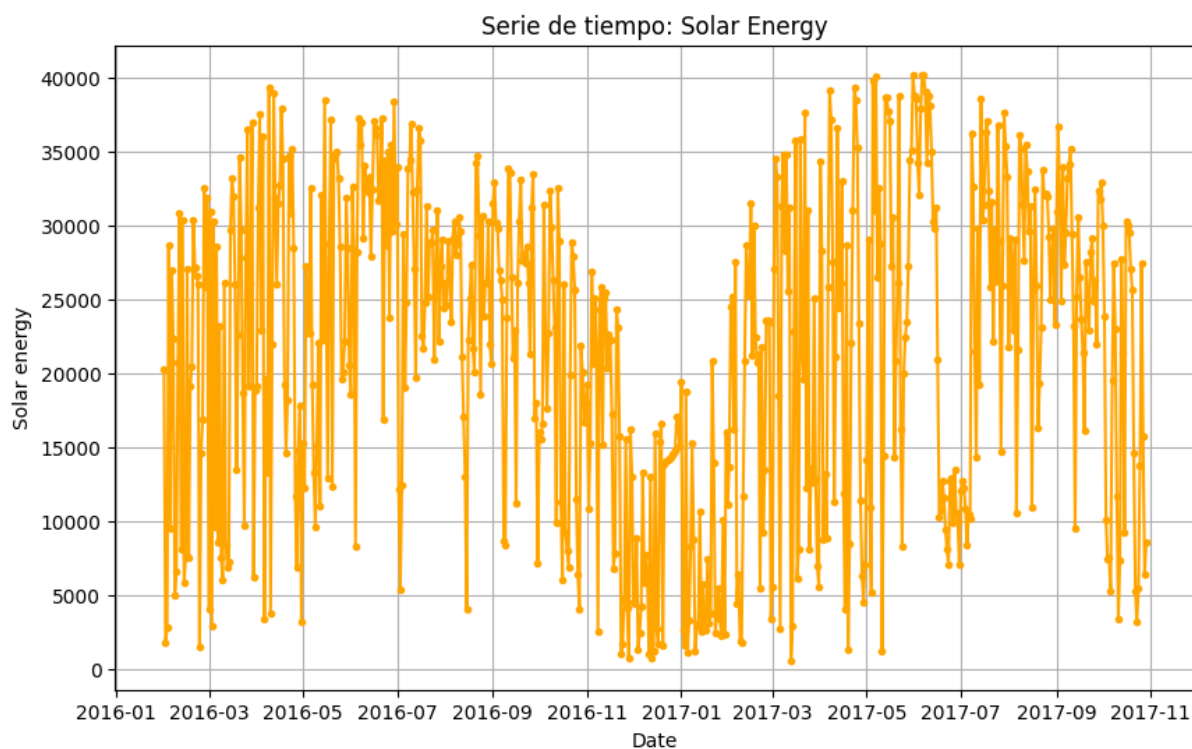


Figura 6: Serie de tiempo de la variable dependiente “Solar energy”, promedio diario, entre el año 2016-2017. Gráfico de elaboración propia mediante Python.

De este gráfico se pueden observar valores tal vez un poco bajos de lo normal en el mes de julio del 2017.

A continuación se entrega un zoom de esta variable del último mes de cual que se registran datos.

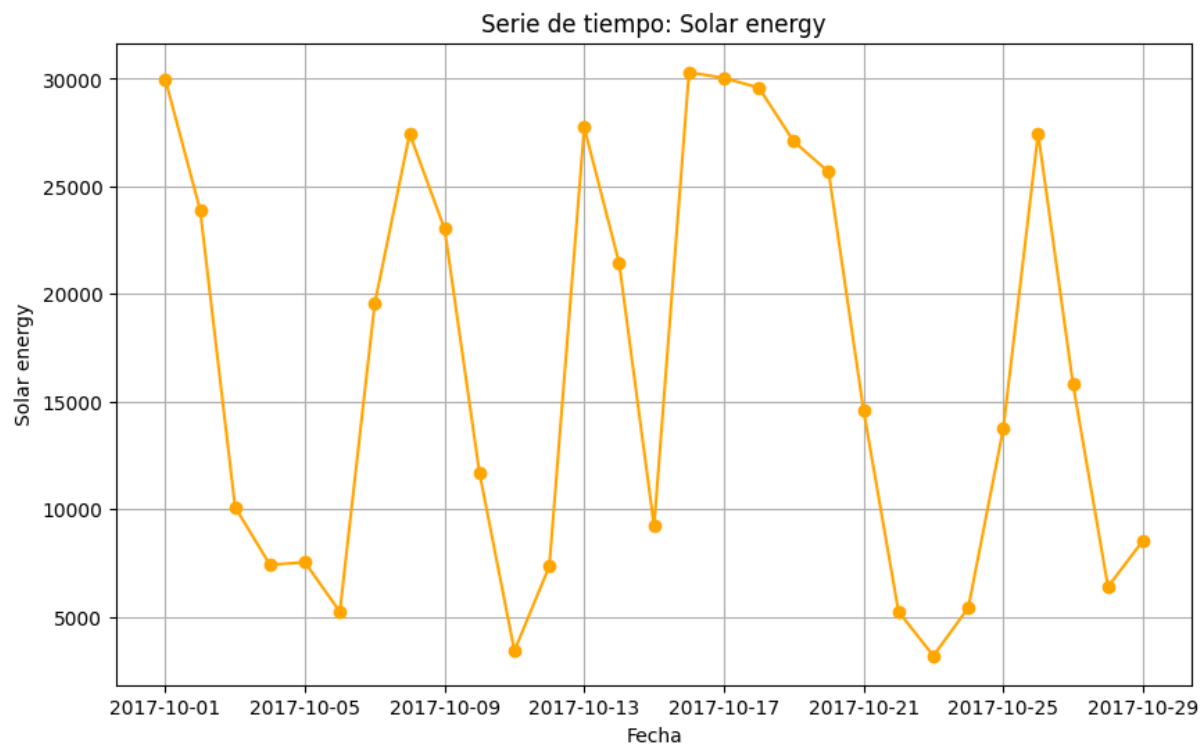


Figura 7: Serie de tiempo de la variable dependiente “Solar energy”, promedio diario durante el mes de octubre, 2017. Gráfico de elaboración propia mediante Python.

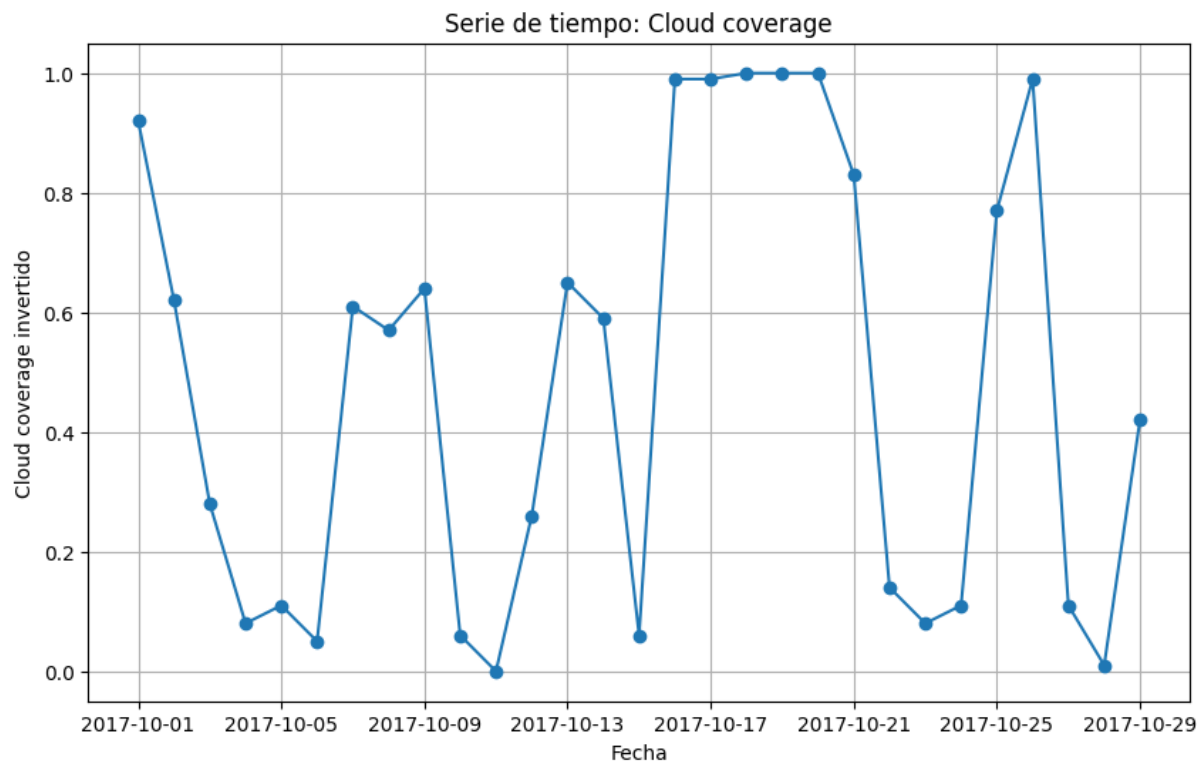


Figura 8: Serie de tiempo de la variable independiente “Cloud coverage”, promedio diario durante el mes de octubre, 2017. Gráfico de elaboración propia mediante Python.

En este último gráfico (8) la variable “Cloud coverage” tiene su rango invertido, esto será explicado en el siguiente apartado.

3) Correlación:

Se podría utilizar cualquier variable meteorológica para tratar de pronosticar la energía de salida, pero una mayor precisión podría lograrse utilizando las variables que más influyan en el valor de salida. Por ello, para vislumbrar que variables pueden tener mayor relación se puede calcular la correlación de estas. Por ejemplo, utilizando la correlación de Pearson que dice cuan fuerte es la relación directa o lineal entre ambas variables. En este caso la correlación de Pearson entre “Solar energy” y “Cloud coverage” es de -0.6879 . Esto indica que a medida que hay mayor cobertura nubosa menor es la energía que se produce. Si bien pueden existir relaciones no lineales o con algún desfase depende del caso averiguar estas características para seleccionar los inputs del modelo.

También se puede visualizar esta relación con un gráfico de dispersión correspondiente a la figura (9).

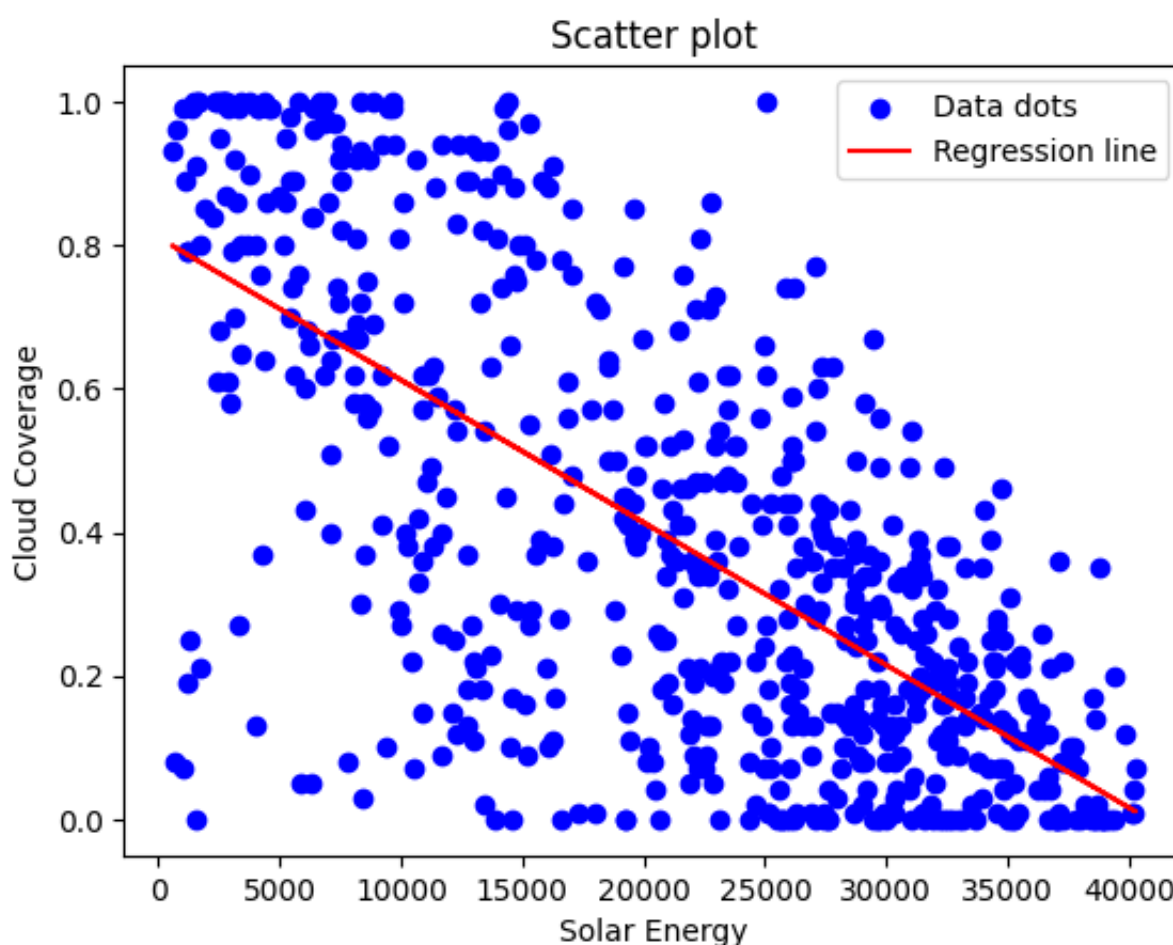


Figura 9: Gráfico de dispersión de las variables utilizadas entre 2016-2017. Elaboración propia mediante Python.

En los datos elegidos existían otras variables como la temperatura y la humedad, pero estos presentaban una menor correlación con la energía solar. Si bien se podrían utilizar todas estas variables, por ahora solo se usará una.

Mediante un histograma podemos identificar en que valores se concentran los datos. En la figura (10) se puede observar que la cobertura nubosa presenta la mayor cantidad en el valor 0, que se refiere a que hay un cielo sin cobertura nubosa o despejado. Pero como se mencionó hay un problema relacionado con valores correspondientes a 0 por lo que se invertirá el rango de esta variable. Originalmente un valor de 0 indicaba un cielo despejado y un valor de 1 un cielo nublado, ahora se tendrá a lo contrario simplemente por que nuestros datos tienen mayor frecuencia en el primer valor. Quedando entonces con un rango de $[1,0]$, donde un valor de 1 se refiere a un cielo despejado y un valor de 0 a un cielo nublado. El gráfico de dispersión con esta variable invertida se representa en la figura (11).

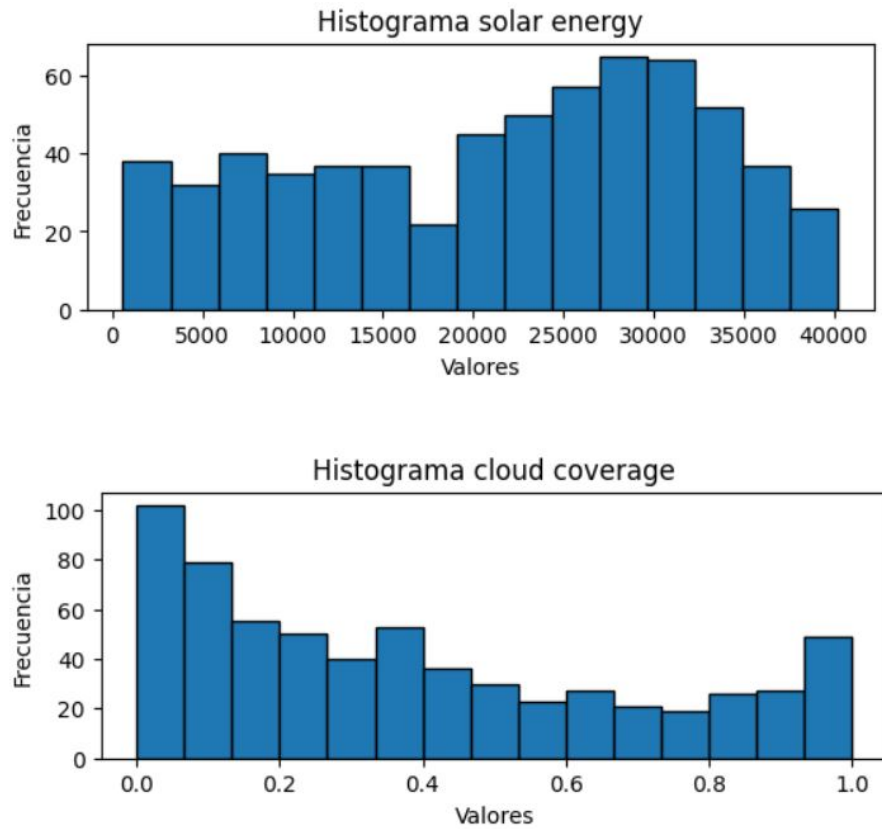


Figura 10: Histograma de las variables utilizadas, Solar energy y Cloud coverage, entre el año 2016-2017. Elaboración propia mediante Python.

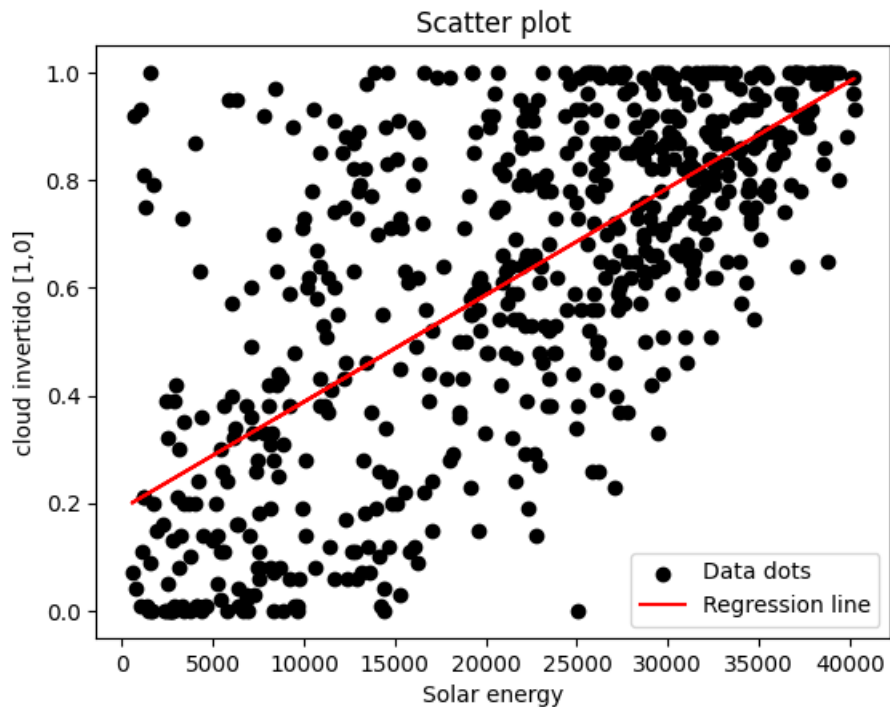


Figura 11: Gráfico de dispersión de las variables utilizadas entre 2016-2017 con la variable 'Cloud coverage' invertida. Elaboración propia mediante Python.

4) Normalizar:

Otro paso importante es la normalización de los datos para que el modelo pueda converger más rápido durante el proceso de entrenamiento en donde se optimizan los pesos, haciendo que este proceso sea más eficiente. Además se puede evitar problemas numéricos y ayuda a la comparación si hay más variables.

	Fecha	Valores	Cloud
0	2016-02-01	0.496054	0.90
1	2016-02-02	0.029774	0.20
2	2016-02-03	0.055338	0.13
3	2016-02-04	0.708811	0.63
4	2016-02-05	0.225312	0.48
..
632	2017-10-25	0.331703	0.77
633	2017-10-26	0.677802	0.99
634	2017-10-27	0.383638	0.11
635	2017-10-28	0.146729	0.01
636	2017-10-29	0.200605	0.42

[637 rows x 3 columns]

Figura 12: Extracto del *Jupyter notebook* de algunos valores de las variables utilizadas una vez normalizadas.

5) Datos supervisados:

Ahora se transforma este set de datos en supervisados en donde se proporciona al modelo un conjunto de datos etiquetado, donde la etiqueta es la salida deseada y se tiene clara la relación entre las variables. En este caso se hace por lotes.

	Cloud(t-20)	Cloud(t-19)	Cloud(t-18)	Cloud(t-17)	Cloud(t-16)	\
0	0.90	0.20	0.13	0.63	0.48	
1	0.20	0.13	0.63	0.48	0.87	
2	0.13	0.63	0.48	0.87	0.79	
3	0.63	0.48	0.87	0.79	0.13	
4	0.48	0.87	0.79	0.13	0.00	
..	
612	0.11	0.05	0.61	0.57	0.64	
613	0.05	0.61	0.57	0.64	0.06	
614	0.61	0.57	0.64	0.06	0.00	
615	0.57	0.64	0.06	0.00	0.26	
616	0.64	0.06	0.00	0.26	0.65	

	Cloud(t-15)	Cloud(t-14)	Cloud(t-13)	Cloud(t-12)	Cloud(t-11)	\
0	0.87	0.79	0.13	0.00	0.75	
1	0.79	0.13	0.00	0.75	0.66	
2	0.13	0.00	0.75	0.66	0.38	
3	0.00	0.75	0.66	0.38	0.92	
4	0.75	0.66	0.38	0.92	0.00	
..	
612	0.06	0.00	0.26	0.65	0.59	
613	0.00	0.26	0.65	0.59	0.06	
614	0.26	0.65	0.59	0.06	0.99	
615	0.65	0.59	0.06	0.99	0.99	
616	0.59	0.06	0.99	0.99	1.00	

614	0.383638
615	0.146729
616	0.200605

Name: Valores(t+1), Length: 617, dtype: float64

Figura 13: Extracto del *Jupyter notebook* de algunos valores del set de datos supervisados de la variable independiente.

6) Set de entrenamiento y testeo:

Por último, dividimos los datos en dos partes el conjunto de datos. Con un porcentaje de 80 % para los de entrenamiento y un 20 % para los de testeo. Estos valores pueden modificarse según se requiera. Por lo general se utiliza un porcentaje 70-30, pero como no contamos con una gran cantidad de datos dejamos más en el entrenamiento.

Construyendo la red neuronal

Para la construcción del modelo se eligen las redes neuronales recurrentes (RNN) y un segundo tipo de red long-short term memory (LSTM) con memoria a largo plazo, debido a que tienen un buen desempeño para el pronóstico mediante el uso de series de tiempo. Al utilizar dos tipos se busca comparar cual resuelve mejor el problema.

A continuación se muestran algunas líneas del código utilizado para la Red Neuronal Recurrente y una breve explicación de la elección o significado de los parámetros y funciones utilizadas. El código completo puede ser consultado en la plataforma GitHub (21).

1) Librerías:

Las librerías utilizadas para generar las redes neuronales se muestran en el extracto de código que sigue:

```

1  # RNN
2  from keras.models import Sequential
3  from keras.layers import SimpleRNN, Dense
4  from keras.optimizers import Adam
5  from sklearn.model_selection import train_test_split
6  from sklearn.metrics import mean_squared_error
7  import numpy as np
8
9  # Red LSTM
10 from math import sqrt
11 from numpy import concatenate
12 from matplotlib import pyplot
13 from pandas import DataFrame
14 from pandas import concat
15 from sklearn.preprocessing import MinMaxScaler
16 from sklearn.preprocessing import LabelEncoder
17 from sklearn.metrics import mean_squared_error
18 from keras.models import Sequential
19 from keras.layers import Dense, Dropout
20 from keras.layers import LSTM

```

Algunas de ellas están definidas en el apartado de “Marco teórico” y las funciones utilizadas sirven en general para seleccionar el tipo de red, dividir el set de datos en entrenamiento-testeo y para definir ciertos parámetros de estructura de red.

2) Conjunto de entrenamiento y testeo:

Las siguientes líneas de código corresponden a la función utilizada que divide el total de datos en dos partes, un set de entrenamiento y otro set de testeo en una proporción de 0.9 y 0.1 respectivamente. Este tamaño de los set se regula con **test size**, por otro lado, el parámetro **random state** se utiliza para establecer una semilla (seed) para la generación de números aleatorios para la división y selección de datos en cada set. Esto significa que si se proporciona el mismo valor de semilla en ejecuciones diferentes, se obtiene la misma división de datos.

```

1  # Dividir los datos en conjuntos de entrenamiento y prueba
2  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
    random_state=42)

```

3) Modelo y arquitectura de la red:

La elección de los parámetros de la estructura que utilizan las redes neuronales constituye una de las mayores dificultades debido al sin número de posibilidades siendo complejo elegir combinaciones que funcionen bien o distinguir que son mejores que otras. Existen algunos algoritmos para encontrar valores de parámetros óptimos, pero en este caso se utiliza el método de prueba-error en el que se cambian valores y se observa el comportamiento de la red durante los procesos para seleccionar aquellos que entreguen un menor error de del valor estimado.

En cuanto a los **tipos de modelos**, existen diferentes opciones de tipos de capas recurrentes y Keras se utilizan tres:

- a) **SimpleRNN**: Red recurrente estándar totalmente conectada cuya salida realimenta la entrada diseñada para trabajar con datos secuenciales.
- b) **LSTM**: Red Long-Short Term Memory, es un tipo de red neuronal recurrente diseñada para abordar el problema de las dependencias a largo plazo en las secuencias de datos. A diferencia de la red recurrente simple, utiliza una estructura de celda que permite retener y olvidar información a lo largo del tiempo. Esto le permite aprender dependencias a largo plazo y es especialmente útil en tareas donde se requiere recordar información relevante de manera más efectiva.

La arquitectura incluye unidades de memoria llamadas celdas de memoria y puertas (entrada, salida y olvido) que regulan el flujo de información a través de la red. A menudo superan a las RNN tradicionales en la predicción de series temporales debido a su capacidad para manejar eficazmente dependencias temporales complejas.

- c) **GRU**: Gated Recurrent Unit, es otro tipo de red neuronal recurrente que es muy similar a las LSTM pero con una estructura más simplificada. A diferencia de LSTM, GRU tiene dos puertas lógicas en lugar de tres (puerta de actualización y puerta de reinicio) (20).

En este caso se utilizan las SimpleRNN y LSTM como se mencionó anteriormente. A continuación se describen algunos de los parámetros que se utilizan para modificar la estructura la red.

Activadores: Se refiere a la función de activación que se aplicarán a las neuronas. El modo de activación puede ser implementado mediante una capa del modo:

```
model.add(Dense(64, activation='tanh')).
```

Existen diversas funciones de activación y el rango del valor de salida de la neurona depende de esta.

- **Linear**: La función de activación lineal, aplicada mediante la capa con `activation='linear'`, realiza una transformación simple, proporcionando una salida proporcional a la entrada. No impone restricciones en el rango de salida, permitiendo que la red neuronal genere cualquier número real como salida.
- **Relu**: (Rectified Linear Unit) La función de activación ReLU, implementada con `activation='relu'`, actúa anulando las entradas negativas y manteniendo inalteradas las entradas no negativas. Su rango de salida es $[0, +\infty)$, suprimiendo la parte negativa de la entrada y promoviendo activaciones más dispersas y no lineales.
- **Tanh**: (Tangente Hiperbólica) La función de activación tangente hiperbólica, empleada mediante `activation='tanh'`, es similar a la sigmoide pero produce salidas en el rango $[-1, 1]$. Es útil para centrar los datos alrededor de cero, siendo especialmente beneficioso en problemas en los que se requiere una salida que oscile entre valores positivos y negativos.
- **Sigmoid**: La función de activación sigmoide, utilizada con `activation='sigmoid'`, transforma las entradas en valores dentro del rango $[0, 1]$. Esta función es comúnmente aplicada en la capa de salida de modelos de clasificación binaria, ya que proporciona una interpretación probabilística de las predicciones, asignando probabilidades a las clases.

La función ReLU o rectificadora es conveniente de usar en este caso, ya que su rango se encontrará entre 0 y 1 que es el mismo formato que tienen los datos. En la figura (14) se observan visualmente estas funciones descritas.

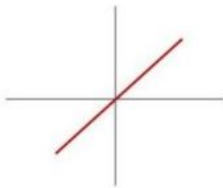
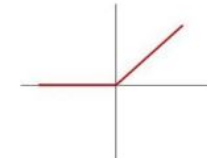
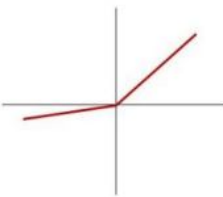
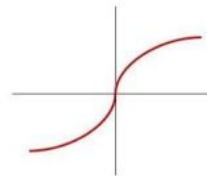
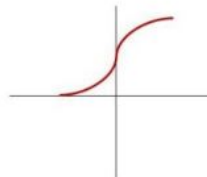
Activation Function	Equation	Plot
Linear	$f(x) = x$	
ReLU	$f(x) = \max(0, x)$	
Leaky ReLU	$f(x) = \max(0.1 \cdot x, x)$	
Tanh	$f(x) = \tanh(x)$	
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	

Figura 14: Funciones de activación de una red neuronal. Fuente: Rajagukguk, R. A. (2020)(22).

4) Compilar el modelo:

Para la compilación del modelo se debe especificar la función de pérdida para evaluar los pesos, el optimizador usado para elegir entre los diferentes pesos, y métricas opcionales para recoger y mostrar los datos durante el entrenamiento.

Función de pérdida: Entrega el error entre el valor estimado y el esperado, con esta diferencia se transmite la información para actualizar los pesos. Las fórmulas de estas funciones de pérdida se muestran en la figura (5) junto con sus respectivas definiciones en la sección de “Marco teórico”.

Optimizador: El optimizador es otro de los dos valores necesarios para compilar un modelo (junto con la función de pérdida).

- **Adam (Adaptive Moment Estimation):** Adapta las tasas de aprendizaje de cada parámetro de la red de manera individual, basándose en estimaciones del primer momento (media) y el segundo momento (varianza) de los gradientes. Esto ayuda a que el modelo converja de manera eficiente y a lidiar con diferentes tasas de aprendizaje para cada parámetro. Este método es sencillo de implementar, es eficiente, necesita poca capacidad de memoria, y es muy bueno en problemas con una gran cantidad de datos y/o parámetros. Es utilizado por defecto en los modelos de Keras con un valor default de 0.001.
- **Adagrad (Adaptive Gradient Algorithm):** Adapta las tasas de aprendizaje para cada parámetro según la magnitud de las actualizaciones anteriores. Puede funcionar bien para

conjuntos de datos dispersos, pero a veces puede llevar a tasas de aprendizaje que disminuyen demasiado rápido.

- **SGD (Stochastic Gradient Descent):** Este es el optimizador más básico. Actualiza los pesos en dirección opuesta al gradiente de la función de pérdida con respecto a los pesos. Puede ser estocástico (utilizando un subconjunto aleatorio de datos en cada iteración) o por lotes (utilizando todo el conjunto de datos).

Como optimizador para ambas redes se utilizará el algoritmo de gradiente descendiente Adam con su valor default.

Tasa de aprendizaje: Su función se expresa como **learning rate** y corresponde a un parámetro que determina la velocidad a la que van a cambiar los pesos de las conexiones de la red. Tiene generalmente un rango $[0,1]$, siendo los valores más cercanos a cero los que hacen que los pesos cambien poco a poco, acercándose lentamente a la convergencia, y los cercanos a uno los que hacen que la red converja mas rápidamente al principio, pero siendo posible que los pesos oscilen demasiado al encontrar el peso óptimo final.

Dropout: Corresponde a una técnica de regularización en la que en cada paso de entrenamiento existe la probabilidad de que alguna neurona sea “apagada” temporalmente, por lo que será ignorada completamente durante esa iteración, pero puede estar activa en la siguiente. Este algoritmo puede aumentar la precisión del modelo entre un 1-2% (23). Si se observa que hay *overfitting* o sobreentrenamiento se puede aumentar la tasa de dropout.

```

1 #Compilar el modelo con una funcion de perdida y un optimizador
2 model.compile(loss='mean_squared_error',
3               optimizer=Adam(learning_rate=0.003)) #: Calcula el error cuadratico
4               medio.
5
6 # Imprimir un resumen del modelo
7 model.summary()
```

5) Entrenar el modelo:

Una vez compilado, el siguiente paso es ejecutar el modelo con los datos de entrada. Podemos entrenar o ajustar nuestro modelo cargando los datos del set de entrenamiento utilizando la función `fit()`.

El proceso de entrenamiento se ejecutará para un número de iteraciones definido mediante el argumento `epoch`. También podemos definir el número de entradas que serán evaluadas antes de que haya una actualización de los pesos de la red mediante el argumento `batch size`.

```

1 # Entrenar el modelo
2 model.fit(X_train, y_train, epochs=300, batch_size=64, verbose=2)
3 #Entrena el modelo para un numero fijado de ciclos (nb_epoch), con
4   actualizaciones de gradiente cada batch_size ejemplos.
```

- 6) **Predicciones:** Una vez realizado el proceso de entrenamiento se puede realizar la validación del modelo utilizando el set de datos de testeo para realizar las predicciones y posteriormente calcular el error de la estimación. El siguiente código es un ejemplo de ello.

```

1 # Realizar predicciones en el conjunto de prueba
2 y_pred = model.predict(X_test)
3
4 # Calcular el error cuadratico medio entre el valor de salida y el esperado
5 mse = mean_squared_error(y_test, y_pred)
6 print(f"Error Cuadratico Medio en el conjunto de prueba: {mse:.2f}")
7 history = model.fit(X_train, y_train, epochs=300, batch_size=64, verbose=2)
```

Además de los parámetros que definen el modelo, durante los diferentes procesos se tienen otras herramientas para saber cómo está trabajando o aprendiendo la red.

Callbacks: Son una serie de funciones que pueden ser aplicadas en ciertos momentos del proceso de entrenamiento o testeo y son personalizables. Estas pueden ser utilizados para echar un vistazo a los estados internos así como realizar acciones, como detener el entrenamiento prematuramente, guardar el mejor modelo, ajustar la tasa de aprendizaje, etc.

- **ModelCheckpoint:** es útil para guardar el modelo después de cada época si se observa una mejora en una métrica específica, como la precisión en el conjunto de validación. Este callback es esencial al entrenar modelos complejos durante largos periodos, ya que permite retener el mejor modelo en términos de rendimiento en lugar de simplemente el último.
- **TensorBoard** Este callback proporciona una interfaz gráfica para visualizar métricas de entrenamiento y otros aspectos del modelo a lo largo del tiempo. Al integrar **TensorBoard** en el proceso de entrenamiento, se puede monitorear el progreso del modelo de manera interactiva. Un uso común es visualizar las curvas de aprendizaje, inspeccionar la arquitectura del modelo y comparar múltiples ejecuciones.
- **History():** Aplicado en cada modelo Keras que para almacenar eventos en un objeto de tipo History. Puede ser utilizado posteriormente para visualizar las curvas de aprendizaje y realizar análisis de rendimiento.

Con la información almacenada en History, se pueden graficar los valores de pérdida (loss) durante el entrenamiento obteniéndose una curva de entrenamiento o durante la validación para llamándose curva de validación. Estas sirven para obtener información sobre cómo está aprendiendo el modelo en el conjunto de entrenamiento y cómo generaliza en el conjunto de validación.

Si la curva de entrenamiento está disminuyendo pero la curva de validación comienza a aumentar o se estabiliza, podría ser un signo de sobreajuste, lo que significa que el modelo está aprendiendo demasiado los detalles específicos del conjunto de entrenamiento y no está generalizando bien. Por otro lado, si ambas curvas disminuyen, es un buen indicativo de que el modelo está aprendiendo correctamente. Para visualizar esto se puede observar la figura (16) en la sección de “Resultados”.

La función `model.summary()`, por ejemplo, proporciona un resumen conciso de la arquitectura del modelo. Muestra información detallada sobre las capas, el número de parámetros entrenables y la forma de salida de cada capa.

Al ejecutar el código de testeo se arrojará entonces la información en la terminal entregando un mensaje por cada iteración mostrando su respectivo valor de pérdida. Esto se puede observar en las figuras (15) y (18) que muestran la información tras correr ambos modelos.

7) Código completo de ambas redes:

El siguiente extracto de código corresponde a la estructura utilizada para los tipos de redes, todos estos códigos y gráficos mostrados a lo largo del informe pueden revisarse en (21).

```

1      # Definir la arquitectura de la red neuronal
2
3      #red neuronal LSTM
4      model = Sequential()
5      model.add(LSTM(64, input_shape=(X_train.shape[1],1)))
6      model.add(Dense(64, activation='relu')) #capa oculta #relu para que salga
          con valores de 0 a 1
7      model.add(Dropout(0.07))
8      model.add(Dense(1))
9      model.compile(loss='mae', optimizer='adam')
10     # fit network
11     history = model.fit(X_train, y_train, epochs=500, batch_size=64,
          validation_data=(X_test, y_test), verbose=2, shuffle=False)
12     # plot history
13     y_pred6= model.predict(X_test)
14     mse = mean_squared_error(y_test, y_pred6)
15     print(f"Error Cuadrático Medio en el conjunto de prueba: {mse:.2f}")
16     pyplot.plot(history.history['loss'], label='train')
17     pyplot.plot(history.history['val_loss'], label='test')
18     pyplot.legend()

```

```
19 | pyplot.show()
20 |
21 | #red neuronal recurrente
22 | model = Sequential()
23 | model.add(SimpleRNN(64, input_shape=(X_train.shape[1], 1)))
24 | model.add(Dense(64, activation='relu')) #capa oculta #relu para que salga
    |     con valores de 0 a 1
25 | model.add(Dropout(0.4))
26 | model.add(Dense(1))
27 | model.compile(loss='mean_squared_error', optimizer='adam')
28 | model.summary()
29 | history = model.fit(X_train, y_train, epochs=500, batch_size=64,
    |     validation_data=(X_test, y_test), verbose=2, shuffle=False)
30 | y_pred = model.predict(X_test)
31 | mse = mean_squared_error(y_test, y_pred)
```

Post-procesamiento de datos

El postprocesamiento se realiza una vez que el modelo ya arroja los datos pronosticados. Generalmente, si en el pre-procesamiento de los datos se realizó una normalización a estos se recomienda que al final se aplique una desnormalización para obtener los valores en su correspondiente unidad de medida y puedan ser interpretados fácilmente. Existen diversas técnicas para el pre y post procesado de los datos, tal como la normalización utilizada anteriormente, la transformada de Wavelet, transformada logarítmica, estandarización, etc. La utilidad de cada una depende de las características del set de datos y del problema a resolver.

En este estudio al no contar con unidades de medida de las variables no es esencial aplicar el post-procesamiento, centrandose en el procedimiento más que en la interpretación de los resultados.

Resultados

1. Modelo de Red Neuronal Recurrente:

a) Información al correr el modelo

El código de este modelo se encuentra en la sección 7) en la parte inferior. La información que se muestra en la siguiente captura es entregada en la terminal al correr el modelo de red neuronal. En la parte superior se encuentra un “summary” o resumen de las capas y sus tamaños. En la parte inferior la pérdida durante el entranamiento y testeo de la red.

```
Model: "sequential_3"
Layer (type)                Output Shape              Param #
=====
simple_rnn_3 (SimpleRNN)     (None, 64)                4224
dense_3 (Dense)              (None, 64)                4160
dropout (Dropout)           (None, 64)                0
dense_4 (Dense)              (None, 1)                 65
=====
Total params: 8449 (33.00 KB)
Trainable params: 8449 (33.00 KB)
Non-trainable params: 0 (0.00 Byte)
=====
Epoch 1/500
9/9 - 2s - loss: 0.2601 - val_loss: 0.0799 - 2s/epoch - 198ms/step
Epoch 2/500
9/9 - 0s - loss: 0.1132 - val_loss: 0.0824 - 70ms/epoch - 8ms/step
Epoch 3/500
9/9 - 0s - loss: 0.0995 - val_loss: 0.0821 - 74ms/epoch - 8ms/step
Epoch 4/500
9/9 - 0s - loss: 0.0834 - val_loss: 0.0760 - 113ms/epoch - 13ms/step
...
Epoch 500/500
9/9 - 0s - loss: 0.0107 - val_loss: 0.0854 - 63ms/epoch - 7ms/step
2/2 [=====] - 0s 3ms/step
Error Cuadrático Medio en el conjunto de prueba: 0.09
```

Figura 15: Información de la pérdida durante las iteraciones de entranamiento y testeo. MSE=0.09 . Elaboración propia mediante Python.

b) Gráfico de pérdida

En el eje horizontal del siguiente gráfico de pérdida se encuentran las épocas o iteraciones y en el eje vertical el valor de la pérdida. La pérdida es una medida de cuánto difieren las predicciones del modelo de las etiquetas reales. Recordando, el objetivo del entranamiento es minimizar esta pérdida.

La curva de entranamiento “**train**”, muestra cómo la pérdida cambia en el conjunto de entranamiento a medida que el modelo se entrena. Idealmente, debería disminuir a medida que el modelo aprende y se ajusta a los datos de entranamiento. Por otro lado, la curva de validación “**test**” muestra cómo la pérdida cambia en el conjunto de validación a medida que el modelo se entrena. Es importante observar esta curva para asegurarse de que el modelo no esté sobreajustando los datos de entranamiento y pueda generalizar bien a nuevos datos.

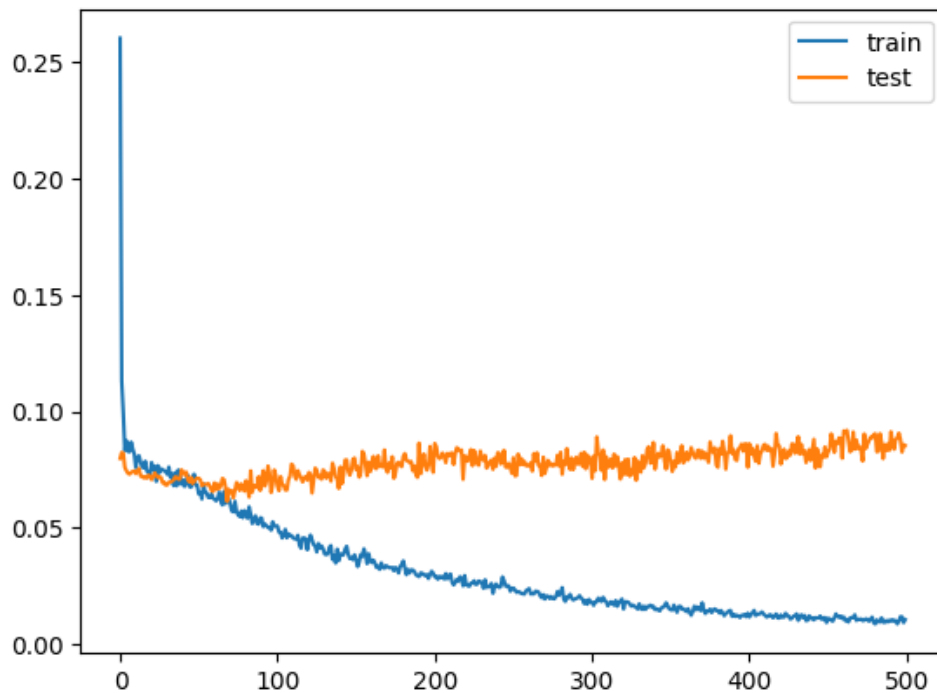


Figura 16: Gráfico de pérdida durante las iteraciones de entrenamiento y testeo. Elaboración propia mediante Python.

En el caso de la figura anterior en la curva de validación se observa un ligero aumento a lo largo de las iteraciones (a partir de la número 100), por lo que hay un sobreajuste.

c) **Valores esperados y pronosticados**

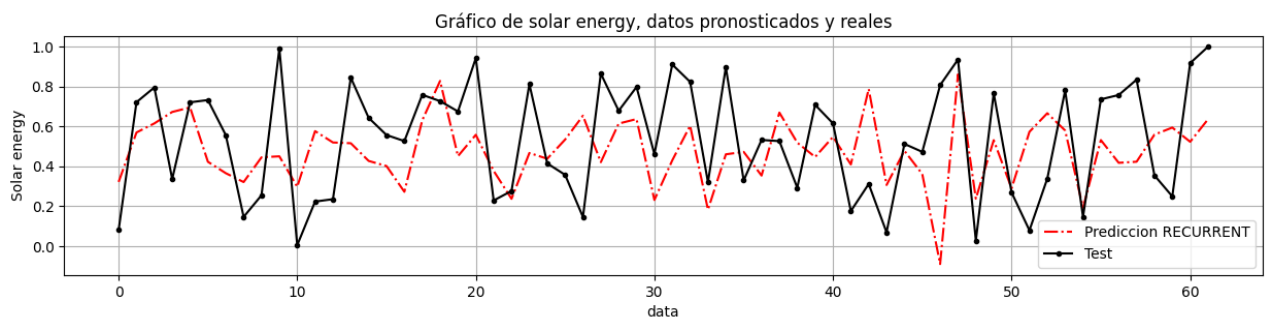


Figura 17: Gráfico de los valores esperados y los pronosticados por la Red Neuronal Recurrente Simple. Elaboración propia mediante Python.

Este gráfico muestra los valores esperados (color negro) y los estimados (color rojo punteado) entregados por el modelo. Visualmente se observa que el resultado no se ajusta y tiende a subestimar los valores esperados. El error cuadrático medio (MSE) tiene un valor de 0.09 en este caso.

2. Modelo de Red Neuronal Long-Short Term Memory:

En esta sección se muestra información del modelo LSTM y gráficos generados, cuyo código se encuentra en la sección 7) en la parte superior.

a) Información al correr el modelo

```
Epoch 1/500
9/9 - 3s - loss: 0.3567 - val_loss: 0.2403 - 3s/epoch - 278ms/step
Epoch 2/500
9/9 - 0s - loss: 0.2420 - val_loss: 0.2407 - 122ms/epoch - 14ms/step
Epoch 3/500
9/9 - 0s - loss: 0.2271 - val_loss: 0.2397 - 134ms/epoch - 15ms/step
Epoch 4/500
9/9 - 0s - loss: 0.2204 - val_loss: 0.2363 - 140ms/epoch - 16ms/step
Epoch 5/500
9/9 - 0s - loss: 0.2247 - val_loss: 0.2376 - 122ms/epoch - 14ms/step
Epoch 6/500
9/9 - 0s - loss: 0.2221 - val_loss: 0.2362 - 126ms/epoch - 14ms/step
Epoch 7/500
9/9 - 0s - loss: 0.2245 - val_loss: 0.2356 - 129ms/epoch - 14ms/step
Epoch 8/500
9/9 - 0s - loss: 0.2193 - val_loss: 0.2355 - 121ms/epoch - 13ms/step
Epoch 9/500
9/9 - 0s - loss: 0.2219 - val_loss: 0.2351 - 120ms/epoch - 13ms/step
Epoch 10/500
9/9 - 0s - loss: 0.2170 - val_loss: 0.2350 - 120ms/epoch - 13ms/step
Epoch 11/500
9/9 - 0s - loss: 0.2228 - val_loss: 0.2337 - 127ms/epoch - 14ms/step
Epoch 12/500
9/9 - 0s - loss: 0.2213 - val_loss: 0.2343 - 121ms/epoch - 13ms/step
Epoch 13/500
...
Epoch 500/500
9/9 - 0s - loss: 0.1170 - val_loss: 0.1842 - 129ms/epoch - 14ms/step
2/2 [=====] - 0s 7ms/step
Error Cuadrático Medio en el conjunto de prueba: 0.05
```

Figura 18: Información al correr el modelo LSTM con dropout 0.08. Donde se visualizan los valores de pérdida durante el entrenamiento y testeo en cada iteración, al final subrayado de rojo se muestra el error MSE. Captura de pantalla.

b) Gráfico de pérdida

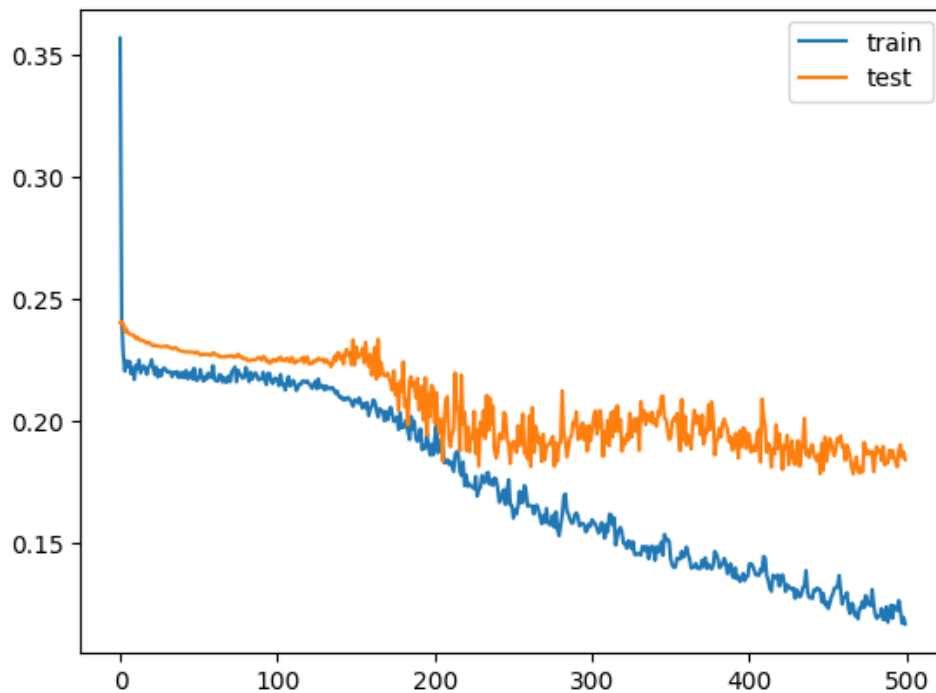


Figura 19: Gráfico de pérdida durante las iteraciones de entrenamiento y testeo. Con dropout=0.08. Elaboración propia mediante Python.

Las curvas de entrenamiento y validación indican un buen aprendizaje del modelo al disminuir los valores de pérdida. Este comportamiento es considerado más correcto que el caso de la RNN.

c) Valores esperados y pronosticados

En el gráfico comparativo de los valores esperados se visualizan dos líneas (azul y roja) de pronósticos en los que se utilizan distintos valores del parámetro “dropout”. Esto demuestra la gran variabilidad y complejidad en la elección de los parámetros de la estructura, ya que hace que las predicciones varíen bastante. En este caso un valor más alto de dropout hace que el modelo tenga menos variabilidad en cuanto máximos y mínimos de la variable a estimar.

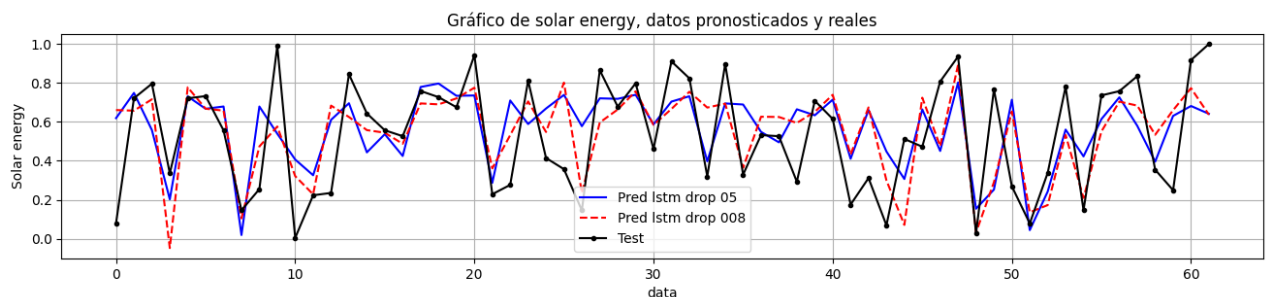


Figura 20: Gráfico de los valores esperados y los pronosticados por la red LSTM con dos valores de “dropout”. Elaboración propia mediante Python.

La línea roja que representa un modelo con dropout de 0.08 se ajusta mejor a los valores esperados con un error MSE de 0.05.

3. Serie comparativa:

La siguiente figura compara los resultados del modelo de RNN simple y el LSTM junto con los datos reales, de color cyan, azul y negro respectivamente. El eje x, correspondiente al dato tiene pocos valores pues el set de testeó sólo cuenta con un 10 % del conjunto total de los datos. Notar que la línea negra de valores esperados es la misma para los dos casos anteriores al utilizar un misma semilla para definir los set de entrenamiento-testeo.

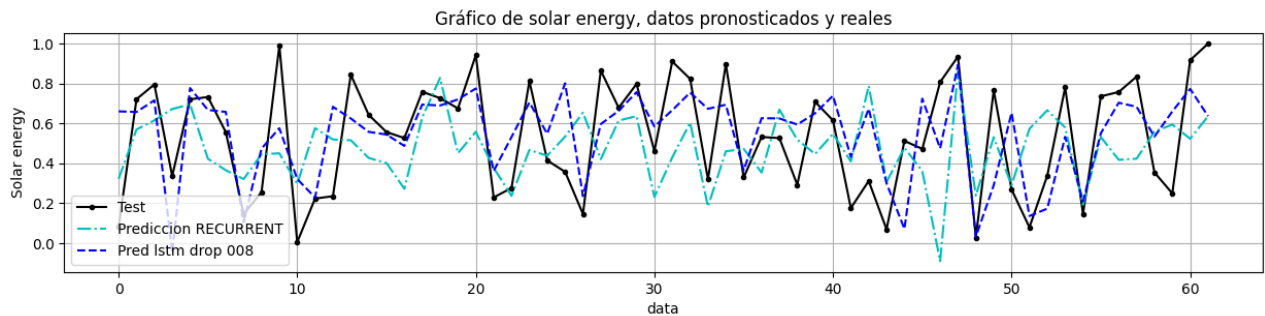


Figura 21: Gráfico compartivo de los valores esperados de energia fotovoltaica en color negro y los valores pronosticados por los dos modelos. Elaboración propia mediante Python.

En este último gráfico se observa un mejor ajuste de los valores estimados por la red neuronal LSTM. Recordando el Error Cuadrático Medio de la RNN simple es de 0.09 y el de la red LSTM de 0.05.

Discusión

Según el gráfico de pérdida del modelo recurrente simple (16) en el entrenamiento la red neuronal aprende bien de la serie de tiempo, sin embargo en el proceso de validación se ve que hay sobreajuste pues en vez de disminuir la pérdida esta aumenta. Según la tasa de disminución durante el entrenamiento se observa que unas 400 iteraciones serían suficientes para el aprendizaje. Por otro lado, en la serie comparativa de los datos esperados y el output de este mismo modelo muestra una subestimación de pronóstico debido a que el modelo aprendió detalles más específicos del set de entrenamiento dificultando la generalización para obtener nuevos valores a partir de datos no usados en el entrenamiento.

Ahora, para el caso del modelo LSTM hay un buen aprendizaje de acuerdo a lo mostrado en las curvas de entrenamiento y validación (19), no se observa el overfitting aunque podrían añadirse más iteraciones para ver cuánto más disminuye la pérdida. De acuerdo al gráfico comparativo (20) se deduce que un dropout mayor hace que el modelo subestime más, por ello un valor ligeramente menor, tal como 0.08 funciona mejor en este caso. La elección de cualquiera de los parámetros que definen la estructura y optimización de la red pueden cambiar el comportamiento de los valores estimados completamente, al cambiar un valor puede que otros también deban ajustarse.

Por último, al aplicar dos tipos de redes recurrentes al problema (una simple y otra con memoria a largo plazo) permite comparar el desempeño de ambos modelos y cuál resuelve de mejor forma el problema que en este caso consiste en el pronóstico de energía fotovoltaica. A pesar de que la serie de datos no es tan extensa como se quisiera la red LSTM entrega de todas formas un mejor resultado de acuerdo a la figura (21) y que presenta un menor error MSE (0.05 en comparación con 0.09).

Sería interesante trabajar con variables de entrada que muestren una mayor correlación entre sí, e incluso utilizar más de una variable independiente (irradiancia, temperatura, etc) para poder aportar más información al modelo a expensas de aumentar la complejidad. Cabe destacar que se utilizaron diversos parámetros para encontrar un resultado aceptable, donde el MSE fuese menor que 1. Estos modelos y sus diversos parámetros de estructura bien pudiesen ser comparados tal como en la figura (20) donde se superponen pronósticos utilizando una tasa de dropout distinta.

Conclusión

En síntesis, la red neuronal LSTM muestra un menor error cuadrático medio de 0.05 en comparación con la red neuronal recurrente simple que entrego uno de 0.08, además de este último mostrar overfitting durante el proceso de validación. De este modo, se lograron aplicar dos tipos de redes neuronales recurrentes al problema de predicción de energía solar fotovoltaica con un conjunto de datos internacional, de modo que queda pendiente aplicarlo a una zona de Chile. Los estadísticos, gráficos y errores permiten ver el comportamiento de los modelos durante su funcionamiento e identificar aquel que resuelva el problema con un menor error. El conocimiento del funcionamiento base de una red neuronal junto con el entendimiento de los diferentes parámetros y estructuras que componen el modelo es crucial para poder elegir los valores que permitan encontrar la mejor aproximación de los datos estimados ante los esperados.

Las redes neuronales y la metodología utilizada puede generalizarse para trabajar con cualquier set de datos, solamente es necesario contar con un serie de tiempo de variables independientes y dependientes entre sí. Un caso de ejemplo, sería el pronóstico de irradiancia solar utilizando otras variables meteorológicas como la temperatura, o también el pronóstico de precipitación caída utilizando datos de humedad. En la actualidad la inteligencia artificial sigue aumentando sus aplicaciones gracias a los avances computacionales que permiten tratar problemas con mayor complejidad o que requieran una gran estructura con muchas mas capas ocultas o numero de neuronas.

En en un futuro trabajo se espera recabar una mayor cantidad de datos junto con una seleccion de variables con mayor correlación en una zona de Chile aplicando la misma metodología y profundizando más en los tipos de redes y parámetros que pueden aportar otras ventajas a la resolución del problema.

Referencias

- [1] Alcañiz, A., Grzebyk, D., Ziar, H., & Isabella, O. (2023). Trends and gaps in photovoltaic power forecasting with machine learning. *Energy Reports*, 9, 447-471. Recuperado el 21-10-23 de <https://www.sciencedirect.com/science/article/pii/S2352484722025975>
- [2] Reikard, G. (2009). Predicting solar radiation at high resolutions: A comparison of time series forecasts. *Solar Energy*, 83(3), 342-349. Recuperado el 23-10-23 de <https://www.sciencedirect.com/science/article/pii/S0038092X08002107>
- [3] Dasa, U. K., Tey, K. S., Seyedmahmoudi, M., Mekhilef, S., Idris, M. Y., Deventer, W. V., Horan, B., & Stojcevski, A. (2018). Forecasting of photovoltaic power generation and model optimization: A review, *Renewable and Sustainable Energy Reviews*, Volume 81, Part 1, 2018, Pages 912-928, ISSN 1364-0321. Recuperado el 21-10-23 de <https://www.sciencedirect.com/science/article/pii/S1364032117311620?via%3Dihub#ab0010>
- [4] Ahmed, R., Sreeram, V., Mishra, Y., & Arif, M. D. (2020). A review and evaluation of the state-of-the-art in PV solar power forecasting: Techniques and optimization. *Energy Reports*, 9, 447-471. Recuperado el 21-10-23 de <https://www.sciencedirect.com/science/article/pii/S1364032120300885>
- [5] Carrasco Catalán, N. A. (2017). Predicción de potencia a corto plazo para plantas fotovoltaicas utilizando redes neuronales artificiales (Tesis de pregrado). Universidad Técnica Federico Santa María, Peumo Repositorio Digital USM. Recuperado el 21-10-23 de <https://repositorio.usm.cl>
- [6] Python. (s.f.). Recuperado el 11 de nov. 2023 de <https://www.python.org/>
- [7] IBM. (s.f.). ¿Qué son las redes neuronales?. Recuperado el 11 de nov. 2023 de <https://www.ibm.com/es-es/topics/neural-networks>
- [8] Kuzmiakova, A., Colas, G., & McKeenhan, A. (2019) Machine-Learning-for-Solar-Energy-Prediction. GitHub. Recuperado el 18 de nov. 2023 de <https://github.com/usuario/Machine-Learning-for-Solar-Energy-Prediction>
- [9] Carta, J. Calero, Colmenar, A. Castro, M-A. (2009) Centrales de energías renovables, generación eléctrica con energías renovables (pp 261-264). Pearson Educacion, S.A.
- [10] Mesa Issa, J. D. (2009). Descripción y análisis del efecto fotovoltaico: Estudio de factibilidad para el desarrollo e implementación en el área metropolitana de la ciudad de Pereira (Tesis de pregrado). Universidad Tecnológica de Pereira, Facultad de Ingenierías Eléctrica, Electrónica, de Física y Ciencias de la Computación, Programa de Ingeniería Eléctrica. Recuperado el 24 de nov. 2023, de <https://repositorio.utp.edu.co/items/018b3d68-14b6-4206-ba46-ad72b8d3eb84>
- [11] EOI Escuela de Organización Industrial. (2011). Master profesional en Ingeniería y Gestión Medioambiental 2011: Energía Solar Fotovoltaica. Profesor: Carlos Montoya Rasero. Recuperado el 24 de nov. de 2023, de <https://www.eoi.es>
- [12] Adler, F., Berardi, M., García Pedrosa, M., Monticelli, F., & Morquecho, M. (2013). Energía solar fotovoltaica: Instalaciones Industriales. Recuperado el 24 de nov. 2023, de <http://www3.fi.mdp.edu.ar/dtoelectrica/files/instalaciones-industriales/material/autogeneracion/renovables/solar/energia%20solar%202013.pdf>
- [13] Ovalle, R. (2014) Sociedad fotovoltaica. *Ingeniare. Revista chilena de ingeniería*, vol. 22 N° 1, 2014, pp. 4-5. Recuperado el 08 de octubre del 2023 de https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S071833052014000100001&lng=en&nrm=iso&tlng=en
- [14] Utpal Kumar Das, Kok Soon Tey, Mehdi Seyedmahmoudian, Saad Mekhilef, Moh Yamani Idris, Willem Van Deventer, Bend Horan, Alex Stojcevski. Forecasting of photovoltaic power generation and model optimization: A review, *Renewable and Sustainable Energy Reviews*, Volume 81, Part 1, 2018, Pages 912-928, ISSN 1364-0321. Recuperado el 01 de oct. 2023 de <https://www.sciencedirect.com/science/article/pii/S1364032117311620?via%3Dihub#ab0010>
- [15] Rashid, T (2016). Make your own neuronal network. CreateSpace Independent Publishing Platform.

- [16] Romero Rodríguez, J. M. (2020). Modelo para predicción de potencia de paneles fotovoltaicos utilizando técnicas de clasificación no supervisada y redes neuronales artificiales (Tesis de maestría, Universidad del Norte).
- [17] Assaf, A.M. Haron, H. Abdull Hamed, H.N. Ghaleb, F.A. Qasem, S.N. Albarrak, A.M. A Review on Neural Network Based Models for Short Term Solar Irradiance Forecasting. Appl. Sci.2023, 13, 8332. Recuperado el 26 de nov. 2023 de <https://doi.org/10.3390/app13148332>
- [18] Antona Cortés, C. (2017). Herramientas modernas en redes neuronales: La librería Keras. Trabajo de Fin de Grado, Universidad Autónoma de Madrid, Escuela Politécnica Superior, Grado en Ingeniería Informática. Tutor: Dorronsoro Ibero, J. R. Recuperado el 01 de dic. 2023, de https://repositorio.uam.es/bitstream/handle/10486/677854/antona_cortes_carlos_tfg.pdf?sequence=1&isAllowed=y
- [19] Müller, A. C., & Guido, S. (2018). Chapter 1. Introduction to Machine Learning with Python: A Guide for Data Scientists (pp. 1-25). O'Reilly Media Inc.
- [20] Antona Cortés, C. (2017). Herramientas modernas en redes neuronales: La librería Keras. Trabajo de Fin de Grado, Universidad Autónoma de Madrid, Escuela Politécnica Superior, Grado en Ingeniería Informática. Tutor: Dorronsoro Ibero, J. R. Recuperado el 26 de nov. 2023 de https://repositorio.uam.es/bitstream/handle/10486/677854/antona_cortes_carlos_tfg.pdf?sequence=1&isAllowed=y
- [21] Latorre, T. (2023). Topico Pronostico de energia fotovoltaica mediante redes neuronales. GitHub: <https://github.com/TamiLatorre/Topico-Pronostico-de-energia-fotovoltaica-mediante-redes-neuronales>
- [22] Rajagukguk, R. A., Ramadhan, R. A. A., & Lee, H.-J. (2020). A Review on Deep Learning Models for Forecasting Time Series Data of Solar Irradiance and Photovoltaic Power. Energies, 13(24), 6766.
- [23] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly.