TRABAJO FIN DE MÁSTER

TECNOLOGÍA ENERGÉTICA PARA DESARROLLO SOSTENIBLE

Predicción de demanda y producción de energía eléctrica mediante redes neuronales y validación de los resultados mediante ensayos realizados en el laboratorio de recursos energéticos distribuidos de la UPV

AUTOR: **MARTINEZ Aurélien**

TUTOR: VARGAS SALGADO Carlos

Curso Académico: 2020-21

02/2021

Agradecimientos

A mis padres por haberme apoyado toda mi vida.

A mi novia por su ayuda con la gramática española.

A mi hermano por escucharme practicar mi exposición aunque no entiende mucho el castellano.

A mi tutor por dejarme la libertad de juntar mi gusto por la inteligencia artificial y el desarrollo sostenible.

A mis escuelas, CentraleSupélec y la ETSII por la oportunidad de realizar una doble titulación Franco-Española.

Resumen

El objetivo de ese TFM es, mediante redes neuronales artificiales, predecir la producción y el consumo de una microrred eléctrica. El modelo se aplica a la microrred del laboratorio de recursos energéticos distribuidos de la UPV (*LabDER).* La microrred se compone de un sistema fotovoltaico, un generador eólico, una planta de gasificación de biomasa y un sistema de almacenamiento de energía mediante baterías de ácido-plomo. El sistema de adquisición de datos de la microrred mide y almacenas variables eléctricas de generación, consumo y almacenamiento de energía.

Para predecir la producción del sistema solar, se utilizan datos de irradiancia, de producción de energía, y de temperatura de las 48 horas anteriores. Con esta información, se predice la curva de producción del día siguiente mediante una red de convolución CNN. Con los parámetros elegidos, se obtiene una precisión de **71%.**

Para predecir la producción del sistema eólico, se utilizan datos de producción de las 10 horas anteriores. Con dicha información se predice la cantidad de energía producida durante las próximas dos horas mediante un método híbrido de clasificación con *RandomTreeForest* y una regresión con una red recurrente RNN. Se obtiene una precisión de **72.1%**.

La demanda del laboratorio es constante si no hay actividad (cerca de 400W) e impredecible cuando hay actividad. Así, para predecir la demanda se utilizó datos de consumo del edificio 3P. Con una red CNN se obtiene una precisión de **53.5%** sobre la curva de consumo y **74%** sobre la cantidad total de energía consumida**.**

Se predice además la cantidad de energía en las baterías utilizando un balance energético. Ya que el laboratorio está conectado a la red de suministro local, las baterías solamente se utilizan para ensayos puntuales, cuando hay pruebas en el laboratorio. Utilizando datos de pruebas se obtiene una precisión de predicción de **94.3%.**

Palabras clave: redes neuronales artificiales, red inteligente, Microred, inteligencia artificial, redes de convolución, redes recurrentes, clasificación, regresión, Deep Learning, Machine Learning, ANN, CNN, RNN.

Resum

L'objectiu d'aqueix TFM és, mitjançant xarxes neuronals artificials, predir la producció i el consum d'una microxarxa elèctrica. El model s'aplica a la microxarxa del laboratori de recursos energètics distribuïts de la UPV (LabDER). La microxarxa es compon d'un sistema fotovoltaic, un generador eòlic, una planta de gasificació de biomassa i un sistema d'emmagatzematge d'energia mitjançant bateries d'àcid-plom. El sistema d'adquisició de dades de la microxarxa mesura variables elèctriques de generació, consum i emmagatzematge d'energia.

Per a predir la producció del sistema solar, s'utilitzen dades d'irradiància, de producció d'energia, i de temperatura de les 48 hores anteriors. Amb aquesta informació, es prediu la corba de producció de l'endemà mitjançant una xarxa de convolució CNN. Amb els paràmetres triats, s'obté una precisió de **71%.**

Per a predir la producció del sistema eòlic, s'utilitzen dades de producció de les 10 hores anteriors. Amb aquesta informació es prediu la quantitat d'energia produïda durant les pròximes dues hores mitjançant un mètode híbrid de classificació amb RandomTreeForest i una regressió amb una xarxa recurrent RNN. Segon els resultats s'obté una precisió del **72,1%.**

La demanda del laboratori és constant si no hi ha activitat (prop de 400W) i impredictible quan hi ha activitat. Així doncs, per a predir la demanda es va utilitzar dades de consum de l'edifici 3P. Amb una xarxa CNN s'obté una precisió de **53,5%** sobre la corba de consum i **74%** sobre la quantitat total d'energia consumida.

Es prediu a més la quantitat d'energia en les bateries utilitzant un balanç energètic. Ja que el laboratori està connectat a la xarxa de subministrament local, les bateries solament s'utilitzen per a assajos puntuals, quan hi ha proves en el laboratori. Utilitzant dades de proves s'obté una precisió de predicció de **94,3%.**

Paraules clau: xarxes neuronals artificials, xarxa intelligent, microxarxa, intelligència artificial, xarxes de convolució, xarxes recurrents, classificació, regressió, Deep Learning, Machine Learning, ANN, CNN, RNN.

Abstract

The goal of this master's thesis is to predict the production and the consumption of a smart grid using artificial neural networks. The model will be applied to the smart grid of the distributed energy resources laboratory (*LabDER).* The smart grid is composed of a photovoltaic system, a wind power generator, a gasification plant of biomass and an energy storage system with lead-acid batteries. The data acquisition system, measure and store electrical variables of generation, consumption, and energy storage.

To predict the production of the solar system, irradiance, production, and temperature of the last 48 hours are used. With this information, a convolutional network CNN predicts the production of the next day. With the chosen parameters, the precision obtained is **71%.**

To predict the production of the wind system, the production of the last 10 hours is used. With this information the amount of energy production of the next 2 hours is predicted using a hybrid method of classification with *RandomTreeForest* and a regression with a recurrent network RNN. The obtained precision is **72.1%.**

The demand of the laboratory is constant when there is no activity (around 400W) and unpredictable when there is activity. Therefore, to predict the demand, consumption data of the 3P building have been used. With a CNN network, the precision is about **53.3%** on the consumption curve and **74%** for the total energy consumed.

Moreover, the amount of energy inside the batteries is predicted with an energy balance. As the laboratory is connected to the local supply grid, the batteries are only used during punctual experiments, when there are tests in the lab. With the data during experiments, the precision obtained is about **94.3%**.

Keywords: artificial neural networks, smart grid, Artificial intelligence, convolutional network, recurrent network, classification, regression, deep learning, machine learning, ANN, CNN, RNN.

Índice

I. Memoria

1. Introducción

Las fuentes renovables se presentan como el futuro de la producción de energía. Permiten producir energía sin combustibles fósiles, con bajas emisiones de CO_2. Las fuentes renovables engloban todas las formas de producir energía a partir de fenómenos naturales sostenibles. Se puede utilizar la energía del sol, del viento, del calor de la tierra, de la energía de las aulas, de las mareas o de los ríos con las presas.

Si la energía de las mareas y de las presas son más o menos predictibles, no lo es para la mayoridad de las fuentes renovables. El mayor problema con estas fuentes es que no producen energía de manera constante come lo hacen las plantas nucleares o de carbón.

En efecto, la producción depende de las condiciones exteriores. Como todos no tienen acceso a mareas o unas presas cerca, la producción con fuente renovables es muy variable. De este aspecto de variabilidad derivan muchos problemas:

- No se puede adaptar la producción al consumo. Es decir que a veces se produce más que necesario y la energía se pierde. Al contrario, a veces se produce menos de lo que se necesita, y hay que recurrir a fuentes no renovables auxiliares o sistemas de almacenamiento.
- Se necesita un sistema de almacenamiento porque la producción no se hace de forma simultánea a la demanda. Por ejemplo, si se considera una planta fotovoltaica, la producción se hace durante el día mientras que la demanda eléctrica para las iluminaciones ocurre durante la noche. Por eso se necesita un sistema de almacenamiento. Pero durante el día hay que elegir cuanta energía se consume directamente y cuanta energía se va a almacenar. Lo que es muy difícil que elegir sin saber lo que ocurrirá mañana al nivel de la producción. Eso se puede traducir en sistemas de almacenamiento sobredimensionados, que se traducen en sobrecoste y sobre contaminación.
- La producción variable de las fuentes renovables puede ser también un freno a la implementación de tecnologías de producción verdes.

Con la descentralización de la producción de energía, la democratización del uso de energías renovables y el desarrollo de la inteligencia artificial, surgió el concepto de *Smartgrids* o red inteligente en castellano. Son redes eléctricas que pueden ser aisladas o no, donde además de una red de energía hay una red informática con medidores para seguir en tiempo real el comportamiento de la red eléctrica.

Una red inteligente almacena datos: de consumo, de producción, de almacenamiento de energía y mucho más. Además de medidores, una red inteligente puede disponer de actuadores inteligentes que permiten conectar o desconectar fuentes y cargas.

La multitud de datos almacenados por las redes inteligentes permiten usar algoritmos de *Machine Learning* y *Deep Learning* para estudiar los datos.

Con la emergencia de la inteligencia artificial (*IA)* y el aumento de la potencia computacional de los ordenadores, ahora es posible hacer cosas impensables 20 años antes.

1.1. Objetivos

El objetivo general del TFM es utilizar los datos de una microrred inteligente para predecir la producción de las fuentes renovables, mediante redes neuronales artificiales. Eso permitirá una mejor gestión de la energía, prediciendo también la cantidad de energía en el sistema de almacenamiento y el consumo futuro.

Entre los objetivos específicos del trabajo se encuentran:

- Predecir la producción de un sistema fotovoltaico
- Predecir la producción de un sistema eólico
- Predecir el consumo de una microrred
- Predecir el estado de carga de las baterías de una microrred
- Aplicar los algoritmos a la red LabDER de la UPV

1.2. Herramientas

Entre las herramientas utilizadas se encuentran:

- Algoritmos de *Deep Learning*
- *Python*
- *Notebooks*
- *TensorFlow*
- *Google Colab*
- *Excel*
- *Homer*

A continuación, se explica cómo se utilizan dichas herramientas en el método utilizado.

Los algoritmos de *Deep Learning* y la parte de procesamiento de datos se escribirá en *Python*. Es un lenguaje informático, muy utilizado en el área de las ciencias de los datos y que es muy modulable.

Para hacer los prototipos de los algoritmos se usará los *notebooks.* Los *notebooks* son como cuadernos virtuales donde se puede poner títulos, imágenes, gráficos y células con código *Python*.

Esos notebooks tienen la ventaja de poder ejecutar el código célula por célula mientras que con un código *Python* clásico, se tiene que ejecutar todo el archivo y en el caso de *Deep Learning* eso puede tardar mucho.

Así, con los notebooks se puede construir el código paso a paso y ejecutar las células cada vez para ver si el código hace lo que se quiere.

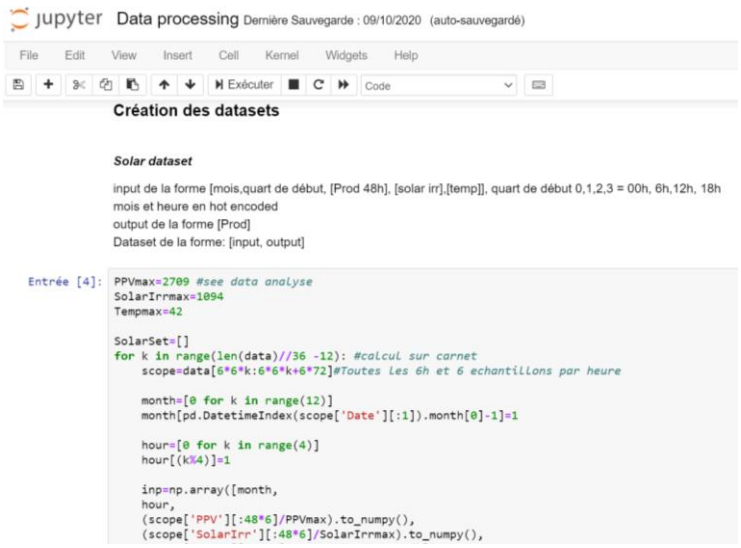


Figura 1: Ejemplo Notebook

Se utiliza la herramienta *Jupyter Notebook* que permite abrir el notebook de manera local usando los navegadores web. Aunque se abre con los navegadores internet, la ejecución se hace localmente, es decir que se utiliza los componentes del ordenador. Así el desempeño depende de las características de la máquina.

Todos los programas se han ejecutados sobre mi portátil. Y este no tiene muy buenas características. Así cuando los cálculos empiezan a ser grandes, la computación dura mucho tiempo.

Por eso se encontró una solución. *Google* pone a disposición, de la misma manera que los *Google Drive* o *los Google doc* compartidos, una herramienta de *notebooks* compartidos que permite ejecutar el código a distancia sobre máquinas de *Google* que son mucho más potentes.

Además, mi portátil no dispone de una verdadera carta gráfica. Y en el caso de *Deep Learning* se suele utilizar los *GPU* para hacer tratamientos de forma mucho más eficaz, paralelizando los cálculos. Los notebooks de *Google* permiten la utilización de una *GPU*, aunque no hay que abusar sino se puede cortar el acceso.

Por eso se usará la herramienta *Google Colab* cuando los cálculos serán demasiados grandes.

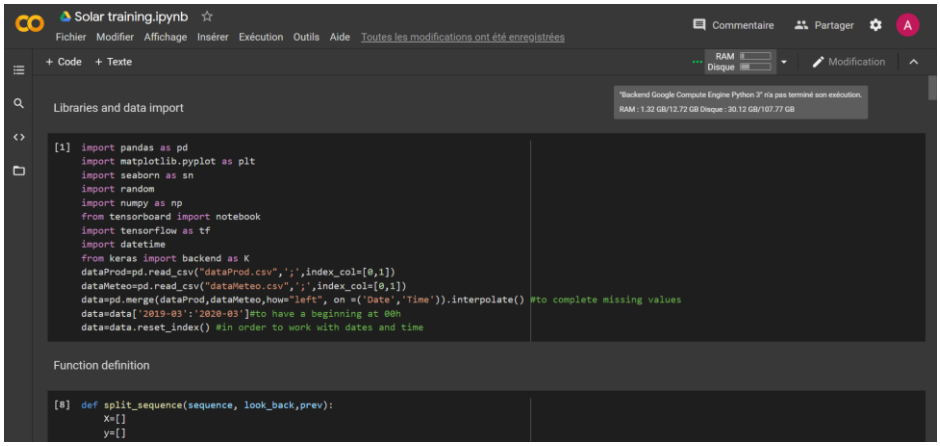


Figura 2: Ejemplo de Google Colab

Como se dijo antes, *Python* es un lenguaje muy modulable, se puede importar muchas bibliotecas de herramientas para hacer diversas cosas, del tratamiento de imagen a la creación de sitios web pasando por la creación de programas.

En este TFM se utilizará la herramienta *TensorFlow* que permite hacer *Deep Learning* de manera muy optimizada con *Python*. En efecto todos los algoritmos de *Deep Learning* son algoritmos matemáticos y se pueden escribir desde cero. Pero eso en práctica, no se hace. Hay bibliotecas que permiten construir de manera bastante simple las redes sin tener que empezar de cero. Además, la herramienta *TensorFlow* usa los tensores para hacer los cálculos, eso permite ahorrar mucho tiempo de computación, optimizando los cálculos.

TensorFlow tiene una herramienta de visualización que se llama *Tensorboard*. Esa herramienta permite trazar curvas, gráficos de redes, y seguir el entrenamiento de los modelos de manera muy visual.

Se utilizará esa herramienta para comparar la eficiencia entre los modelos creados.

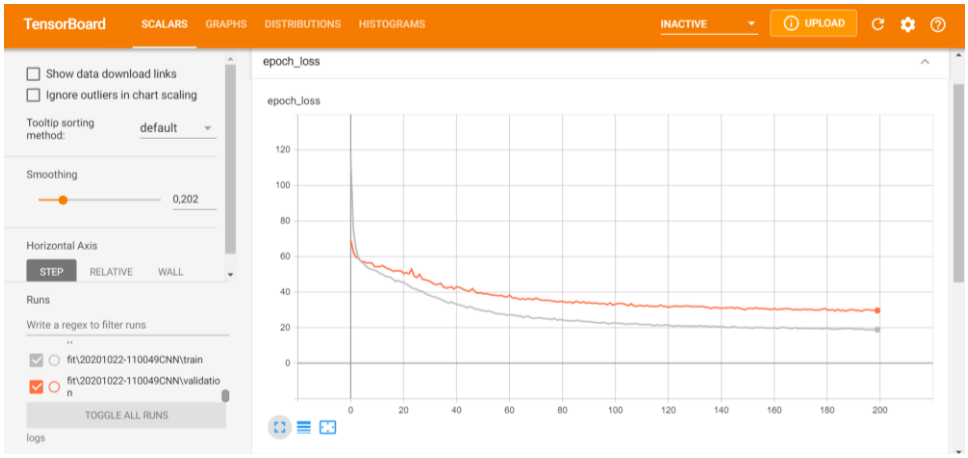


Figura 3: Ejemplo Tensorboard

1.3. Estructura del TFM

Primeramente, se expondrá lo que se hace actualmente en la investigación para predecir la producción de energía con fuentes renovables usando *Machine Learning* y *Deep Learning*.

Después del estado del arte, se estudiará los datos disponibles para predecir el comportamiento de la Microred. En esa parte se hará también un preprocesamiento de los datos.

Luego se intentará predecir la producción de la planta fotovoltaica de varias maneras. Entonces se hará lo mismo para la producción con el sistema eólico.

Acto seguido, se hablará de la predicción de la demanda y del estado de carga de las baterías y por fin se concluirá sobre la eficiencia de las redes neuronales para predecir el comportamiento de la Microred *LabDER*.

2. Estado del arte

Hoy en día, el mundo intenta utilizar más energías renovables. Es crucial reducir las emisiones de CO_2 para frenar el calentamiento global. Además, las energías fósiles no son infinitas y la energía nuclear tiene riesgos y produce gastos radioactivos. Por eso, cada día se utilizan cada vez más fuentes renovables. En 2019, la producción de electricidad con fuentes renovables subo de 10% en España. [1]

Esas fuentes utilizan los fenómenos naturales como el sol, el viento, las aulas, los ríos y otros para producir energía. Tienen muchas ventajas como la ausencia de emisiones de CO_2, pero un gran inconveniente es que no son "fiables".

En efecto, con una central de carbón o una central nuclear, es posible adaptar la producción al consumo de energía para producir en tiempo real lo que se necesita para el consumo mientras que, con las energías renovables, la producción depende de las condiciones naturales: el sol, el viento… Por eso, la predicción de producción de las fuentes renovables es un tema muy importante y que forma parte del tema de las redes inteligentes.

Con las innovaciones en el campo del M*achine Learning* y *Deep Learning*, muchas nuevas posibilidades se han abierto. Pero es solamente el principio. El área de la inteligencia artificial es muy grande, así que hay muchas posibilidades diferentes para intentar predecir la producción de las fuentes renovables o el consumo de energía. Para el objetivo de este TFM, en la predicción de las curvas de producción o de consumo de una red inteligente, se trata de series temporales. Es decir, valores que cambian con el tiempo.

Se va a trabajar con datos sacados de medidores de potencia eléctrica, de irradiancia, de velocidad de viento y otros. Esos variables son temporales.

Esta temática de series temporales es un tema muy común en el área de inteligencia artificial. Es lo mismo para estudiar los electrocardiogramas por ejemplo o para el tratamiento de grabación de voz. Primeramente, se presentará los diferentes algoritmos que se pueden utilizar para trabajar sobre las series temporales. Después se estudiará las diferentes técnicas de pretratamiento de los datos.

2.1. Redes neuronales artificiales

Las redes neuronales artificiales o *"Artificial neural network"* en inglés son las redes más básicas del *Deep Learning* (aprendizaje profundo).

Intentan reproducir lo que pasa en el cerebro humano. Se llaman redes neuronales porque funcionan con neuronas artificiales. La neurona es el elemento fundamental del cerebro humano y también es el de los ANN.

El modelo de la neurona artificial se basa más o menos sobre el de la neurona biológica. Es decir, tiene múltiples entradas y una salida. El valor de la salida depende de los valores de las entradas. En el caso biológico, si las entradas superan cierto umbral, la salida se activa y la información pasa sino la señal se bloquea y nada se transmite.

En informática, la salida depende de las entradas, pero no es obligatoriamente una repuesta binaria.

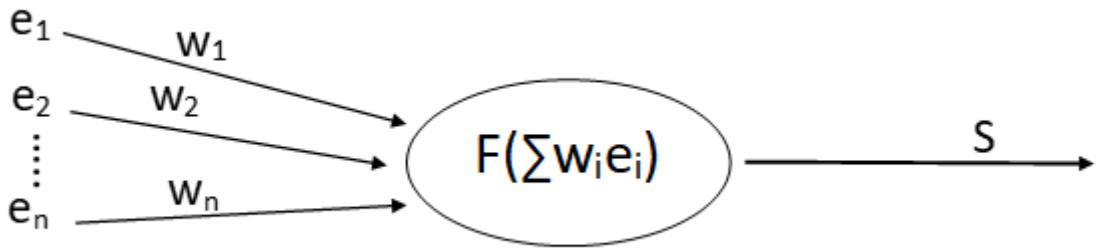


Figura 4: Esquema de una neurona artificial

Como se puede ver en la Figura 4, las entradas tienen pesos w_i. Los valores recibidos por la neurona son el producto de la entrada e_i por del peso w_i. Dentro de la neurona se aplica una función matemática llamada función de activación. La salida de la neurona dependerá de esta función. Desde un punto de vista matemático, hay la siguiente expresión:

$$S = F\left(\sum_{i=0}^{n} w_i e_i\right)$$

Ecuación 1: Activación de una neurona

Donde n es el número de entradas, w_i el peso de la entrada, i e_i el valor de la entrada i y F la función de activación.

La función de activación puede cualquiera función. Puede ser la función identidad, es decir que la salida corresponde a la suma producto de los pesos y entrada. O se puede utilizar distintas funciones, las más usadas son la función llamada *ReLu* y la función *sigmoide*:

$$f(x) = \begin{Bmatrix} x \; si \; x > 0 \\ 0 \; si \; no \end{Bmatrix}$$

Ecuación 2: Formula ReLu

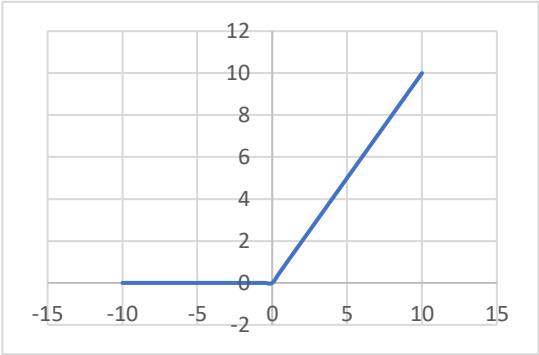


Figura 5: Grafica ReLu

La función *ReLu* permite tener más estabilidad en la red quitando los valores negativos.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 3: Formula sigmoide

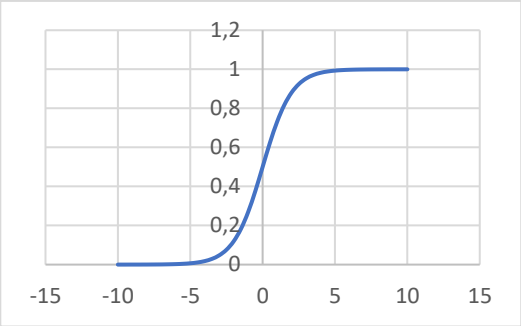


Figura 6: Grafica sigmoide

La función *sigmoide* permite tener una salida entre 0 y 1 cual sean las entradas, lo que puede ser muy útil en algunas situaciones. Además, la función está derivable muy fácilmente, lo que permite ahorrar tiempo de computación durante el entrenamiento de la red.

Aquí están dos funciones de activación típicas, pero hay muchas otras.

Una red neuronal artificial se compone de capas de neuronas interconectadas. Cada neurona de una capa está conectada con todas las neuronas de la capa anterior y de la capa siguiente. Es decir que una neurona de la capa i coge como entrada las salidas de las neuronas de la capa i-1 y manda su salida a las neuronas de la capa i+1.

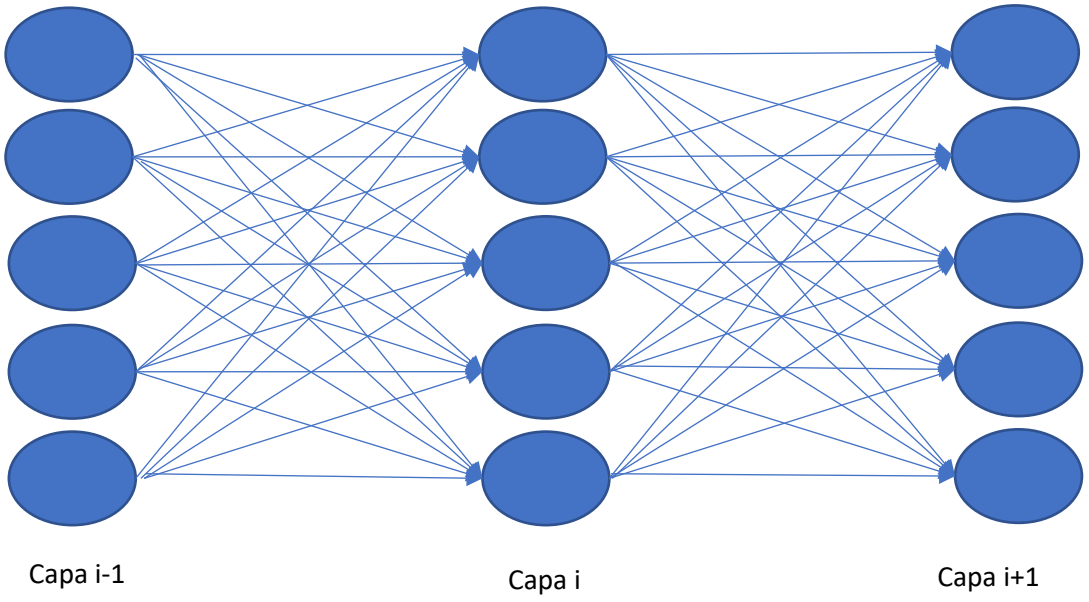


Figura 7: Esquema de una red neuronal

Como se puede ver en la Figura 7, un pequeño número de neuronas da un gran número de enlaces y así un gran número de pesos.

Es bien tener una red neuronal, pero si no se la entrena, no sirve para nada. Entrenar una red significa parametrizarla para que cumpla el objetivo. Por ejemplo, clasificar imágenes o predecir el precio de la gasolina. Por eso hay que entrenar la red sobre muchos datos donde se conoce la salida deseada.

Por ejemplo, para clasificar imágenes y definir si son gatos o perros. Se dispone de muchas imágenes de gatos y perros y cada imagen tiene una "*etiqueta*" que indica la naturaleza de la imagen.

Se pone en entrada la imagen en la primera capa de neuronas y en salida se quiere obtener el valor gato que es por ejemplo 0 y 1 para un perro.

Durante el entrenamiento, la red compara la salida obtenida con la deseada y adapta los pesos de manera a reducir el error entre la realidad y la predicción. Eso se hace calculando el gradiente de la función de error y cambiando los pesos en la buena dirección.

Para cuantificar el rendimiento de la red, se utilizan funciones de error. Estas funciones son el valor que hay que minimizar para entrenar la red. La elección de esta función es muy importante. En efecto, dos redes idénticas en su estructura con diferentes funciones de error darán resultados muy diferentes. De esa función puede depender la veracidad de la predicción.

De forma parecida a las funciones de activación, existen varias funciones de error. Se pueden distinguir en dos categorías:

- Las funciones de error para los problemas de clasificación. Son los problemas donde hay que clasificar los datos en diferentes categorías.
- Las funciones de error para los problemas de regresión. Son los problemas donde se quiere predecir un valor.

Como el tema de este TFM es un problema de regresión sola se hablará de la última categoría.

La función de error más común y utilizada es la *MSE: "Mean Square Error"* o en castellano el error cuadrático medio cuya expresión es la siguiente:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Ecuación 4: Formula MSE

Donde *n* es la dimensión del vector de salida, *y* la salida deseada (verdadera), $\tilde{y}$ la predicción de *y.*

También, existe la función *RMSE: Root Mean Square Error* que permite tener un error en la misma unidad que la salida de la red.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2}$$

Ecuación 5: Formula RMSE

Se usa la MSE o RMSE en los papeles [2]–[7].

O también existe la función *MAPE: Mean Absolute Percentage Error.*

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{y_i - \tilde{y}_i}{y_i}$$

Ecuación 6: Formula MAPE

Esa función es más sencilla que las otras dos. Pero en el caso de valores positivos y negativos, los errores pueden compensarse y dar un error global pequeño mientras que la predicción sea lejana de la realidad.

En práctica, se utilizan diferentes variables para seguir la evolución del aprendizaje. Estas variables se calculan con las funciones de error estudiadas. Una es la "*loss function*" que se minimiza a través del entrenamiento y las otras sirven para seguir controlando el desempeño de la red.

Se suele emplear al mínimo dos funciones de error. Pero se puede emplear muchas más como H. Zang [3] que tiene ocho indicadores: MSE, MAE, MSLE, MASE, NIA U1, U2, MHE.

Se puede también escribir funciones personalizadas para evaluar el error para adaptarse al problema estudiado.

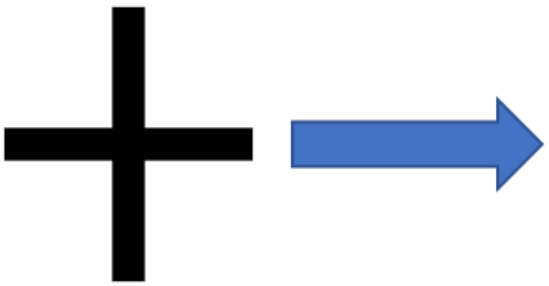
Los ANN son las redes neuronales más simples. Suelen ser usados en algunos estudios como referencias frente a otras soluciones [5]. También se utilizan combinados con otras redes más complejas. En efecto como los ANN son las redes más básicas del *Deep Learning*, se utilizan capas de esos al final de redes como CNN, RNN, u otros.

2.2. Redes de convolución

Las redes de convolución son una categoría de redes inteligentes que son muy bien adaptados al reconocimiento de imágenes y de formas. Se ha descubierto también su potencia de aplicación a series temporales [2], [3]. En efecto, se puede modificar los datos temporales para transformarlas en imágenes y después aplicar un CNN (*Convolucional Neural Network).*

Pero antes de todo, hay que explicar que es una imagen al nivel informático.

Una imagen en negro y blanco corresponde a una tabla de valores. Cada valor de la tabla es un píxel. Cada uno de estos píxeles se representa con 8 bits. Así, hay 2^8 possibilidades es decir 256 possibilidades: entre 0 y 255. 0 corresponde a un pixel blanco y 255 un negro. Se reprensenta un ejemplo en la Figura 8.



0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
255	255	255	255	255	255	255
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0

Figura 8: Representación informática de imagen

Una ventaja de las redes de convolución es que extraen ellos mismos las características de los datos. En una red de convolución, un filtro se aplica sobre una imagen. Un filtro es una tabla que se mueve

sobre la imagen y que hace cálculos. Eso se traduce matemáticamente por un producto de convolución. Por eso se llaman esas redes las redes de convolución.

El filtro contiene valores que se multiplican por los píxeles de la imagen donde se superponen y despues se suman para obtener un elemento de la salida.

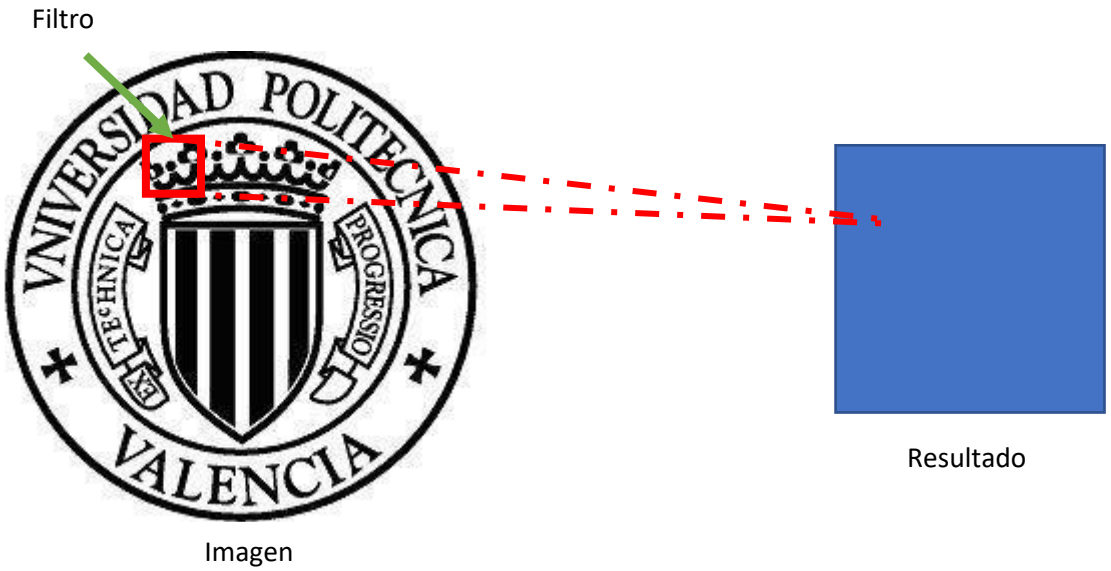


Figura 9: Convolución grafica

La Figura 9 se traduce matemáticamente por la Ecuación 7.

$$Imagen * Filtro = Resultado$$

Ecuación 7: Convolución

En una red de convolución, se aplican muchos filtros a la imagen y se adaptan los valores de esos filtros para entrenar el modelo. Puede tambien aparecer capas de *pooling* en adición de las capas de convolución. Las capas de *pooling* permiten reducir la dimensión haciendo el promedio de algunos píxeles o eligiendo el máximo.

Las redes de convolución suelen acabar por unas capas de ANN.

Se utiliza cada más vez estos redes para trabajar sobre la predicción de series temporales. En el caso de la predicción de producción de electricidad por fuentes renovables, se puede aplicar estos redes a los paneles fotovoltaicos como a las turbinas eólicas.[2][3]

Así, se intentará definir una red de convolución para predecir la producción del *LabDER*.

2.3. Redes recurrentes

Otro método de *Deep Learning* es la red recurrente (Recurent Neural Network o RNN en inglés). Las redes recurrentes suelen ser útiles para trabajar con las series temporales. En efecto, el principio de base de estas redes es que se guarda la información de los previos pasos para predecir el siguiente. Es decir que para predecir el estado t=n+1, se utiliza el paso t=n y un valor de memoria sobre los pasos anteriores.

Los RNN se basan en celdas primitivas donde se calcula la salida y la memoria en cada paso. Existen diferentes tipos de celdas, pero las más conocidas son ***GRU*** *(Gated Recurrent Unit)* y ***LSTM*** *(Long Short Term Memory).*

Hoy en día, se utiliza cada vez más los LSTM en detrimento del GRU. En el campo de la predicción de fuentes renovables, los RNN permiten predecir tanto la producción de energía solar como la producción de energía eólica [4]–[7].

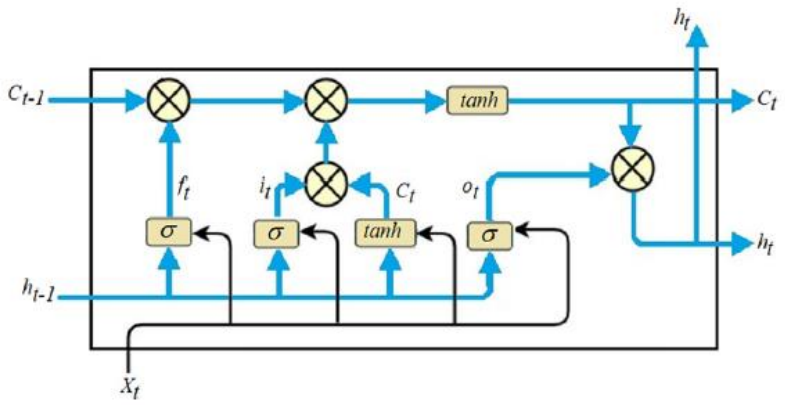


Figura 10: Estructura LSTM, sacado de [4]

En la Figura 10, se puede ver la estructura de una celda *LSTM*.

- h_{t-1} corresponde a la salida de la última celda LSTM
- C_{t-1} es la memoria de la última celda LSTM
- X_t es el dato de entrada de la serie temporal para la celda t
- h_t es la salida de la celda LSTM
- C_t es la memoria de la celda LSTM
- σ corresponde a la función sigmoide
- *tanh* es la función tangente hiperbólica
- ⊗ Corresponde a una suma

Para explicar el funcionamiento de manera bastante sencilla, se puede decir que hay dos cadenas:

- Una que tiene un papel de memoria
- Una que da la salida de cada celda

Para cada celda t, la salida es obtenida en función de la salida anterior, de la memoria anterior y del dato al tiempo t. Al mismo tiempo, la memoria al tiempo t se calcula a partir de la memoria de t-1, del dato t y de la salida t-1.

Se puede ver que, en cada celda elementaría, hay muchos cálculos que hacer con un LSTM. Por eso el tiempo de entrenamiento de estas redes es muy largo y necesita una potencia de cálculo más importante que para los ANN, por ejemplo.

En las investigaciones más recientes, no se utiliza una red con una única técnica, sino una red híbrida. En efecto, los diferentes tipos de redes tienen sus ventajas e inconvenientes. Combinarlos permite tener las ventajas de cada tipo sin los inconvenientes, pero hay que tener cuidado de no tener los inconvenientes de los dos sin las ventajas.

Por ejemplo, Y. Y. Hong [2] combina un CNN con un RBFNN, que es un tipo de ANN. Z. O. Olaofe [7] combina un ANN clásico con un RNN para predecir la potencia eólica de una turbina.

Pero no se puede obtener buenos resultados en *Deep Learning* sin pretratamiento de los datos. Hay muchos métodos diferentes para extraer la información de los datos y elegir los datos interesantes para las predicciones.

2.4. Preprocesado de los datos y datos elegidos

Como en todos los proyectos de *Machine Learning*, lo más importante son los datos utilizados. Es lo mismo para la predicción de producción solar o eólica. Los datos de salida son los mismos, son datos de producción, pero lo más importante son los datos de entrada.

Hay que elegir las informaciones más útiles para que la red prediga la producción de energía. Los datos de entrada dependen esencialmente de lo que se mide. En efecto si se mide la temperatura, la irradiancia o la velocidad del viento, se puede utilizar estos datos. Mientras que en algunas situaciones no se dispone de estas informaciones.

No hay una única solución para predecir la producción de energía. Esto depende de los datos accesibles. Hay solamente una regla, es tener datos sobre la producción de energía, para que se utilicen como referencia para el entrenamiento. Pero también se pueden utilizar como datos de entrada. En efecto se puede solamente utilizar los datos de los días anteriores para predecir la producción de mañana [2], [4].

Pero también se pueden utilizar datos meteorológicos: temperatura, humedad, irradiancia directa, irradiancia global, velocidad de viento, direcciones del viento, precipitaciones [3], [5]–[8] . Hay que tener en cuenta que muchos datos significan un gran tiempo de computación. Además, los datos generalmente no se utilizan de forma directa. Hay que pretratar los datos, cambiar su forma, extraer las informaciones relevantes.

Se pueden utilizar las serie temporales así, con un pretratamiento con normalización por ejemplo [6]. Pero se pueden también descomponer los señales para extraer las informaciones necesarias [3], [4]. Para eso se puede utilizar técnicas de tratamiento de señales como el *VMD* o *SSA* que permiten descomponer las señales en diferentes componentes quitando la parte aleatoria de los datos.

En conclusión, no existe un método mágico para predecir los datos de producción de una red renovable. Todo depende de la arquitectura de la red, de los datos disponibles, y de la potencia de cálculo disponible. En este TFM, habrá que tratar los datos medidos por el *LabDER* y utilizarlos de la manera más optimizada para obtener buenas predicciones.

2.5. Dispositivo experimental – Microrred *LabDER*

El *LabDER*, *Distributed Energy Resources Laboratory,* es una Microred experimental ubicada en el campus de la UPV, Universidad Politécnica de Valencia. Los objetivos del *LabDER* son múltiples y el laboratorio trabaja a nivel de Investigación, Desarrollo e Innovación. De acuerdo con el sitio web del **IIE**: Instituto de Ingeniería Energética, los objetivos del *LabDER* son:

- *Desarrollar sistemas híbridos renovables en el rango del kW, optimizando su fiabilidad mediante técnicas innovadoras de interconexión y almacenamiento.*
- *Desarrollar técnicas de control que garanticen la fiabilidad del suministro eléctrico mediante la combinación de fuentes de energía renovable.*
- *Aplicar dichos sistemas híbridos en experiencias de campo en zonas no interconectadas.*
- *Estudiar el potencial del hidrógeno como vector energético y de almacenamiento de energía en sistemas renovables.*
- *Mejorar la eficiencia de los sistemas renovables, tanto en funcionamiento individual como en sistemas híbridos.*
- *Desarrollar e investigar redes de distribución de energía eléctrica de alta fiabilidad y eficiencia, alimentadas de fuentes renovables múltiples interconectadas*

El laboratorio se compone de fuentes renovables, de sistemas de almacenamiento de energía y de sistemas de control y diagnostico SCADA.

Las fuentes renovables se componen de:

- Una planta fotovoltaica de **2.1kWp** con un inversor para conectar los paneles a la red.

PV panel model	BP7190S
ELECTRICAL CHARACTERISTICS	
Cell Type	Monocrystalline
STC Power Rating Pmp	190 W
Open Circuit Voltage V_{oc}	44.8 V
Short Circuit Current I_{sc}	5.5 A
Voltage at Maximum Power V_{mp}	36.6 V
Current at Maximum Power I_{mp}	5.2 A
Panel Efficiency	15.1%

Grid-tie inverter model	GT2.8-SP
INPUT	
Maximum array open circuit voltage DC	600 V
Maximum input current DC	15.7 A
Maximum array short circuit current DC	24 A
OUTPUT	
Maximum output power AC	2,500 W
AC Voltage	230 V
Weight	20.5 kg

Figuras 11: Panel e inversor de conexión a la red, sacado de los datasheets

- Un aerogenerador de **5kWp** conectado al inversor de red para convertir el corriente del generador en corriente alterna 230V y 50Hz.

Wind turbine model	BORNAY 25.3+
Number of blades	3
Diameter	4 m
Material	Fiberglass / carbon
ELECTRICAL SPECIFICATIONS	
Alternator	3-Ph PMSG* Neodymium
Rated Operating Speed	400 RPM
Power rating (at 12 m/s)	5 kW
WIND SPEED	
For turn on	3 m/s
For automatic braking	14 m/s
Survival	60 m/s

RECTIFIER MODEL	INTERFACE WIND +
INPUT (Three phases AC)	
Operating Voltage Range AC	80 - 480 V
Maximum Voltage AC	510 V
Maximum power	6,000 W
Braking Resistance	10,000 W
OUTPUT	
Voltage range DC	100 - 450 V

GRID-TIE INVERTER	ABB PVI-6000-TL-W
INPUT (Operating voltage range 50...580 V, Max 600V)	
Rated DC input voltage (V DC)	360 V
Max DC input current (I DCmax)	36 A
OUTPUT SIDE (230 V AC Single phase 50 Hz)	
Rated AC power (@cosϕ=1)	6,000 W
Max efficiency (ηmax)	97.0%

Figuras 12: Aerogenerador, rectificador e inversor del sistema eólico, , sacado de los datasheets

- Dos plantas de gasificación de Biomasa de **10KWe** cada una con también una caldera de biomasa de **15kW**. A diferencia de las demás fuentes renovables del *LabDER*, esa se activa cuando se necesita, no depende de las condiciones meteorológicas.

Power rating	**9,7 kW**
Biomass Flow at power rating (Moisture content=8%)	10 kg/h
Syngas LHV	5,500 kJ/m³
η at power rating (Electrical)	19%
Reactor temperature °C	750 – 880 °C

Figuras 13: Planta de gasificación, , sacado de los datasheets

El laboratorio se compone de dos sistemas de almacenamiento de energía. Eso permite almacenar la energía de las fuentes renovables para utilizarla cuando la demanda está superior a la producción.

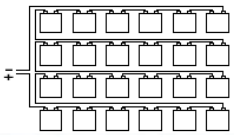
- El primero sistema de almacenamiento se compone de baterías de plomo-acido con un **DoD** de **100%.** Se puede almacenar hasta **10.32kWh** de energía en las 24 baterías. El sistema es conectado a un inversor cargador que permite hacer el vínculo con la red.

Battery model	Sunlight 4 OPzS 200
Number of batteries	24

Capacity of the batteries (DoD=100%):

$24 \cdot 2V \cdot 110\ Ah = \ 5.28\ kWh$ (C1)

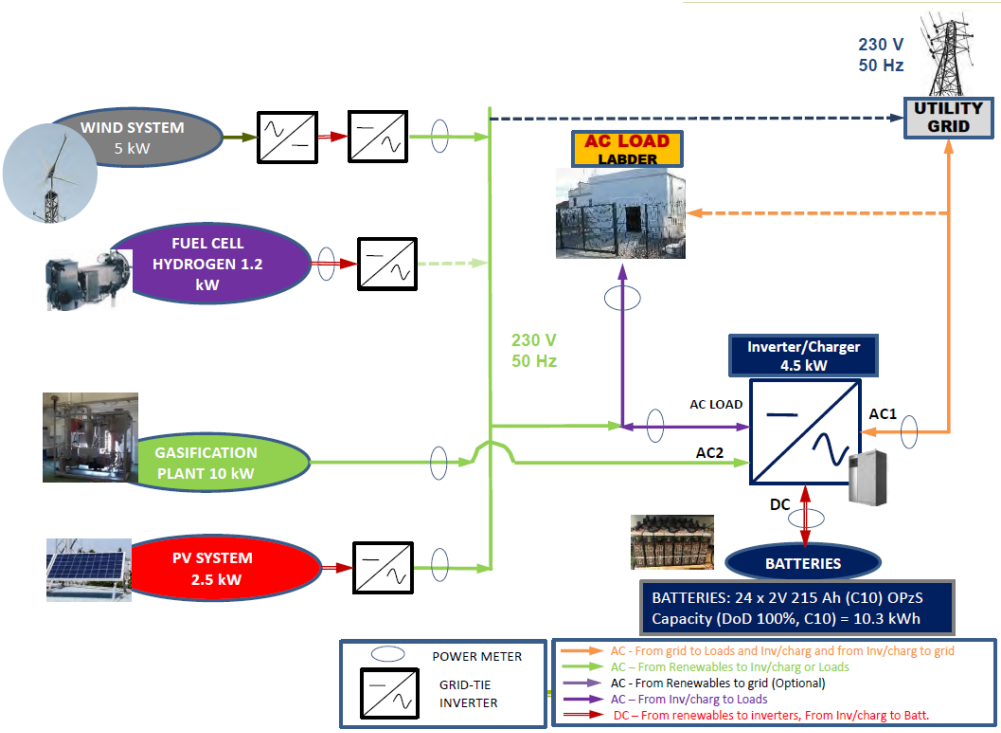
$24 \cdot 2V \cdot 215Ah \ = 10.32\ kWh$ (C10)



Hybrid Inverter/charger model		Schneider Xantrex XW4548
CHARGE MODE		
AC Input	Voltage range	156 - 280V (230V)
	Maximum current	56 A
	Frequency range	40 - 68 Hz (50 Hz)
DC Output	Voltage range	40 - 64 V
	Nominal voltage	50.4 V (48V)
	Maximum current	85A
INVERT MODE		
AC output	Nominal Voltage (50 Hz)	230V
	Maximum current	50A
	Maximum power	4,500 VA
DC Output	Voltage range	42 - 60 V
	Maximum current	120 A

Figuras 14: Baterías e inversor cargador, , sacado de los datasheets

- El segundo sistema de almacenamiento de energía es una planta de hidrógeno. La planta se compone de un electrolizador que permite formar hidrógeno a partir de agua y de energía eléctrica. Después se almacena el hidrógeno en botellas. Se consume después el hidrógeno por una pila de combustible que permite alimentar la Microred con energía eléctrica. Así la capacidad de almacenamiento está limitada solamente por el número de botellas disponibles.



Figuras 15: Esquema LabDER, sacado de http://www.iie.upv.es/labder

La carga del sistema es una carga resistiva programable que permite simular el comportamiento de la demanda eléctrica en varias situaciones.

Con el sistema de control y diagnóstico, se mide la potencia generada por cada fuente, tanto como la demanda eléctrica, la cantidad de energía almacenada y mucho más.

3. Metodología

Con el objetivo de mejor entender el desarrollo del TFM, en esta parte se explica el método global del TFM. El TFM se organiza en 3 grandes partes, la predicción de la producción del sistema solar, la predicción de la producción del sistema eólico y la predicción de la demanda eléctrica. De estas tres partes se predice la energía almacena en las baterías.

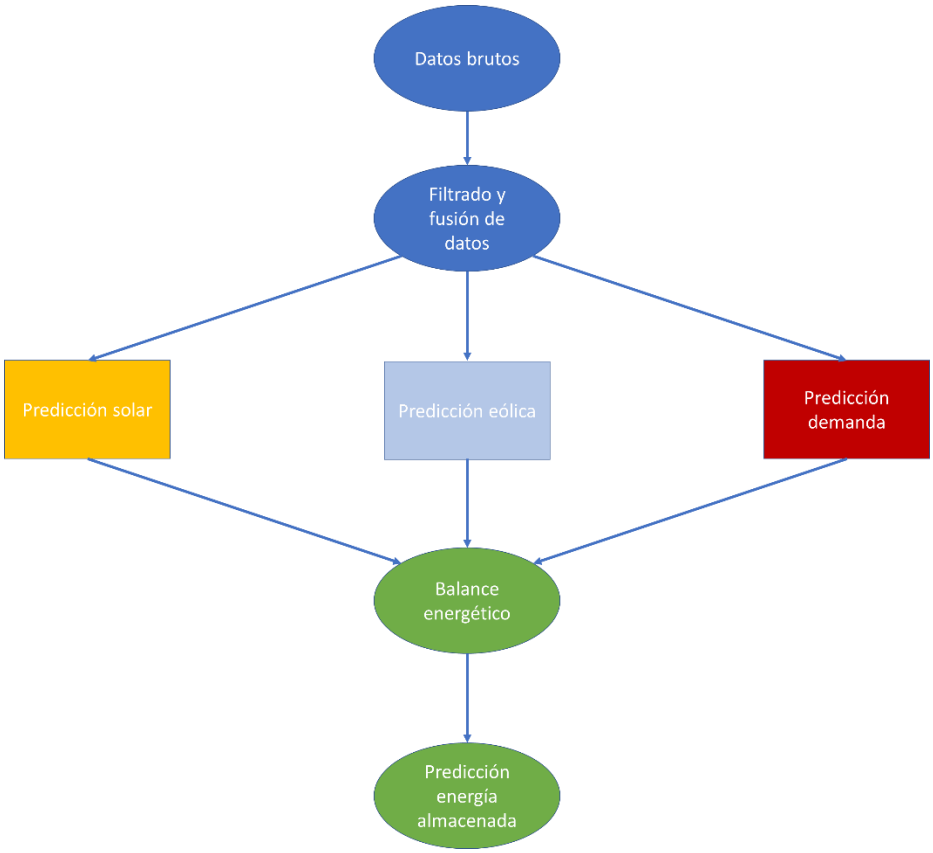


Figura 16: Metodología global

Se explican los métodos de cada parte en la memoria del TFM. Pero si el lector no quiere leer todo o si se necesita aclarar los metodos, se adjuntan imágenes resumen de las tres partes.

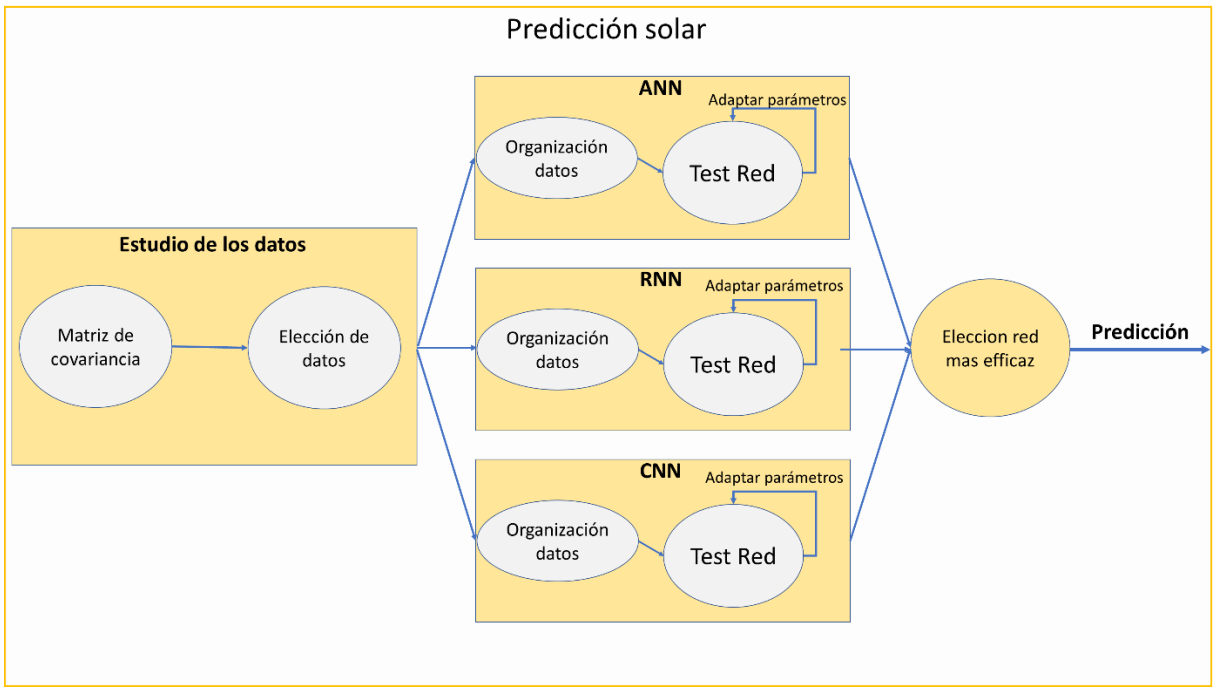


Figura 17: Metodología predicción solar

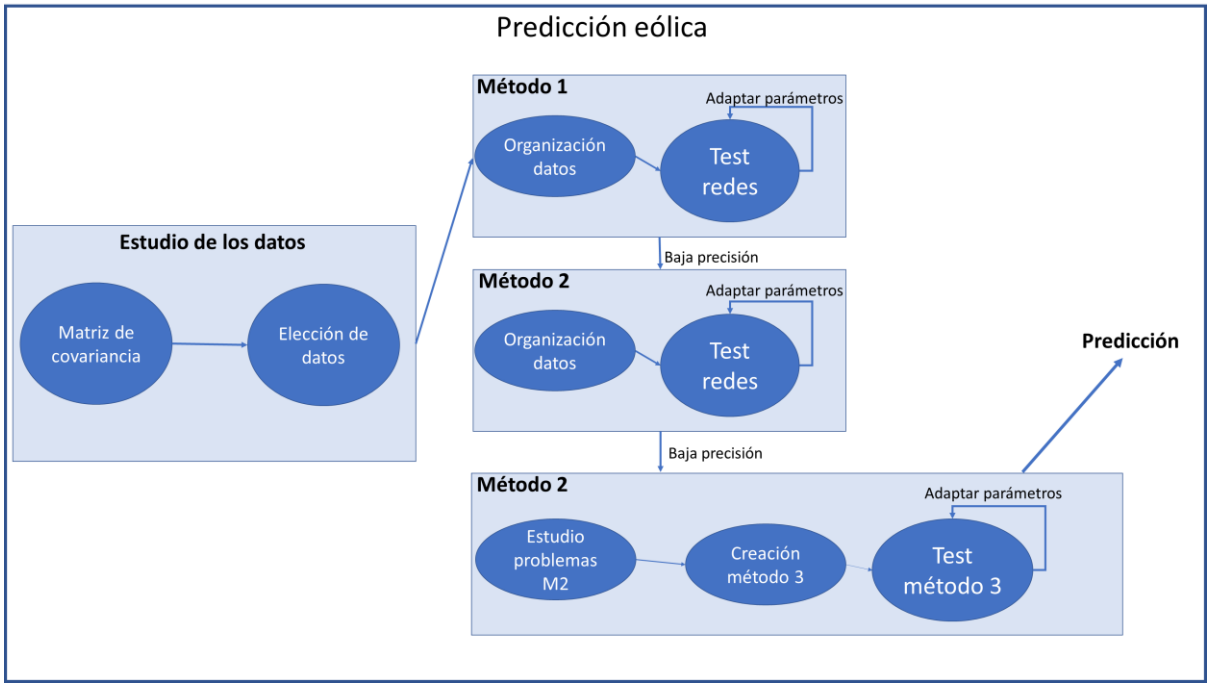


Figura 18: Metodología predicción eólica

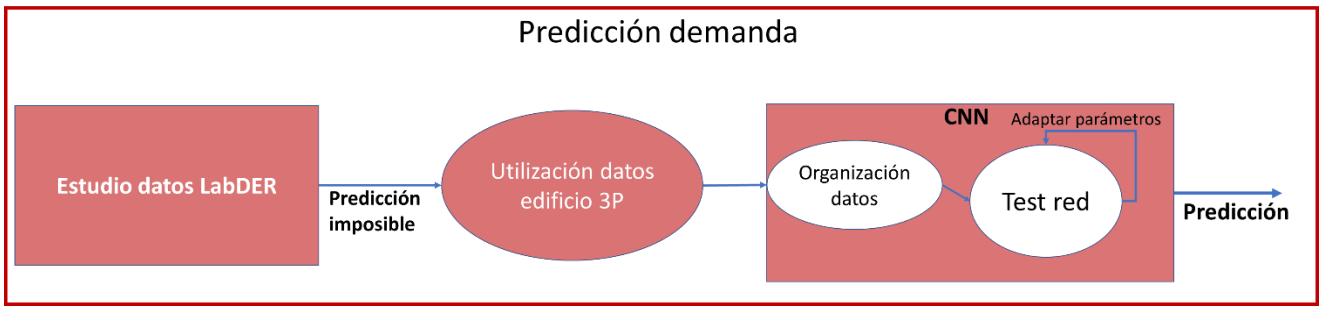


Figura 19: Metodología predicción demanda

4. Datos utilizados

Para poder predecir las potencias generadas por las fuentes renovables se necesita datos de entrenamiento y datos de prueba. Habitualmente, se considera un set de datos que se comparte entre datos de entrenamiento y datos de prueba con una ratio de 70%-30%.

Esto permite evitar los problemas de sobre entrenamiento del modelo que ocurren cuando el modelo informático está adaptado solamente a los datos de entrenamiento. Cuando se le introduce un dato nuevo como un nuevo día, por ejemplo, el modelo está completamente ineficaz.

4.1. Datos del *LabDER*

Se utilizan datos del laboratorio *LabDER* de la UPV.

El *LabDER* tiene un sistema SCADA que saca datos cada segundo y les pone en un archivo CSV. Así hay un montón de datos. Cada semana se crea un nuevo archivo.

Los datos medidos son:

- la potencia producida por los paneles fotovoltaicos
- la potencia producida por la turbina eólica
- la velocidad del viento medido por un anemómetro
- la irradiancia solar
- la potencia consumida por el laboratorio

Breaks	Date	Time	Pact - PM5 - I	Pact - PM9 W	Met Vel_vien	Met -Solar irradicance Avg [R]
OPEN	31/12/2019	01:00:00	10	8,13E-41	1,48	0
	31/12/2019	01:00:01	10	1,1643074	0,79	0
	31/12/2019	01:00:02	10	1,1643074	0,91	0
	31/12/2019	01:00:03	10	8,65E-41	0,91	0
	31/12/2019	01:00:04	10	8,65E-41	0,91	0
	31/12/2019	01:00:05	10	2,27E-41	0,88	0
	31/12/2019	01:00:06	10	2,27E-41	0,88	0
	31/12/2019	01:00:07	10	1,36E-41	0,88	0
	31/12/2019	01:00:08	10	1,36E-41	0,88	0
	31/12/2019	01:00:09	10	2,00E-41	0,94	0
	31/12/2019	01:00:10	10	2,00E-41	0,94	0
	31/12/2019	01:00:11	10	7,74E-41	0,75	0
	31/12/2019	01:00:12	10	7,74E-41	0,75	0
	31/12/2019	01:00:13	10	1,1702915	0,79	0
	31/12/2019	01:00:14	10	1,1702915	0,79	0
	31/12/2019	01:00:15	10	2,3430142	0,79	0
	31/12/2019	01:00:16	10	2,3430142	0,79	0
	31/12/2019	01:00:17	10	1,1651834	0,72	0

Figura 20: Ejemplo de Datos LabDER

Se reorganizan los datos para que sean por mese, y que sea más sencillo de utilización. Además, esto permite ver las incoherencias y los datos que faltan para completarlos.

Hay problemas de datos, a veces los sensores se desconectan o dan valores muy altos como por la potencia eólica.

Además, hay valores negativos que corresponden a la potencia usada por los aparatos de medidas, hay que poner esos valores a 0 tanto como los valores muy grandes que son errores.

Hay que filtrar esos valores falsos. Esto se hace mediante Excel.

$$f(x) = \begin{Bmatrix} x\ si\ x\epsilon[0{:}\,10\,000] \\ 0\ si\ no \end{Bmatrix}$$

Ecuación 8: Filtro de P_{wind}

Donde x es el valor de potencia producida por la turbina.

Se elige el valor de 10 000 como límite según el *datasheet* de la turbina en Figura 21. En efecto la potencia nominal de la turbina Bornay wind 25.3+ es 5 000W. Se elige un umbral de 10 000W para decir que un valor de potencia es falso.

Datos técnicos

Especificaciones técnicas	Wind 13+	Wind 25.2+	Wind 25.3+
Número de hélices	2	2	3
Diámetro	2,86 m	4,05 m	4,05 m
Material	Fibra de vidrio / Fibra de carbono		
Dirección de rotación	En sentido contrario a la agujas del reloj		
Especificaciones eléctricas			
Alternador	Trifásico de imanes permanentes		
Imanes	Neodimio		
Potencia nominal	1500 W	3000 W	5000 W
Voltaje nominal	220 v	220 v	220 v
RPM nominal	600	400	400

Figura 21: Datos técnicos turbina

El SCADA del *LabDER* hace medidas cada segundo para obtener curvas más precisas. En efecto, el viento puede cambiar muy rápidamente y por supuesto la potencia generada por la turbina también.

Se suele utilizar los datos de manera puntual para obtener curvas e informaciones durante un corte periodo de tiempo. Así el tamaño de los archivos no es un problema. Pero en el caso de *Deep Learning*, se necesita un montón de datos para que el aprendizaje funcione bien. Así los datos son muy grandes, por eso se decidió usar datos cada 10 minutos en vez de cada segundo.

Para obtener más precisión, se hace un promedio sobre los 10 minutos siguientes para cada valor que tiene un tiempo acabándose por 0:00. Es decir, cada 10 min. Eso es la manera más sencilla encontrada porque los datos no son continuos. A veces hay huecos de 2 horas o más, lo que hace la automatización muy complicada.

$$SI\big(DROITE(TEXTE(\$TIME;\mathrm{HH{:}MM{:}SS});4) = 0{:}00;MOYENNE.SI(\$VALORES;<>0)\big)$$

Ecuación 9: Promedio 10 min con Excel

Se compara los últimos caracteres del tiempo indicando el formato adecuado, con 0:00. Si es un valor adecuado se hace el promedio sobre los siguientes 10 minutos ignorando los ceros que son valores falsos. Si no es un valor adecuado la formula saca un FALSO.

Date	Time	Pact - PM5 - PV [R]	Pact - PM9 Wind [R]	Met Vel_viento_ms [R]	Met -Solar irradicance Avg [R]						
05/07/2019	09:50:00	710	6,3704906	1,67	403	05/07/2019	09:50:00	736,135593	4,82312537	1,08955707	419,039182
05/07/2019	09:50:01	710	5,2781196	1,48	403	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:02	710	4,9165497	1,48	403	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:03	710	4,9165497	1,48	403	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:04	710	4,0595803	1,36	403	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:05	710	5,5088716	1,61	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:06	710	5,219717	1,58	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:07	710	2,9248743	1,26	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:08	710	2,9248743	1,36	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:09	690	4,0410771	1,51	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:10	690	5,4950385	1,51	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:11	690	5,4950385	1,51	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:13	690	5,4950385	1,45	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:14	690	6,375349	1,51	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:15	690	6,375349	1,51	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:16	710	4,9405766	1,61	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX
05/07/2019	09:50:17	710	4,9405766	1,61	404	FAUX	FAUX	FAUX	FAUX	FAUX	FAUX

Figura 22: Promedio 10 min con Excel

Dado la complejidad de estas fórmulas al nivel informático, el tratamiento puede ser muy largo.

Después de la ejecución de estas fórmulas, se copian y pegan los valores en un nuevo archivo (copiar con enlace estaba muy costoso en tiempo de ejecución). En el nuevo archivo, se reemplazan los valores "Falsos" con células vacías y después se quitan estas células para obtener los datos cada 10 minutos.

Estos pasos fueron muy largos, representaba muchos días de trabajo para encontrar el método explicado antes y muchos otros para ejecutarlo para cada mes de datos. Había otros métodos que podrían funcionar, pero este tenía la ventaja de no bloquear Excel o la computadora.

Después hay que extrapolar para los meses sin datos o con datos falsos.

Falta de datos	Datos con errores
4/03/2019 9:29:50 hasta 08/03/2019 14:31:40	17/09/2019 02:00:01 hasta 15/10/2019
22/03/2019 01:00:00 hasta 08/04/2019 16:45:35	
09/04/2019 11:50:51 hasta 11/04/2019 08:58:41	
23/04/2019 02:00:00 hasta 02/05/2019 12:27:18	
20/05/2019 08:57:36 hasta 31/05/2019 12:03:02	
04/06/2019 13:21:32 hasta 05/06/2019 13:01:47	
20/06/2019 16:03:31 hasta 27/06/2019 10:24:26	
27/06/2019 23:21:46 hasta 01/07/2019 06:59:10	
04/07/2019 09:01:30 hasta 05/07/2019 09:46:30	
09/07/2019 09:12:12 hasta 12/07/2019 10:43:58	

15/07/2019 08:47:25 hasta 23/07/2019 07:34:46	
29/07/2019 13:02:19 hasta 17/09/2019 02:00:01	
15/10/2019 02:00:00 hasta 17/10/2019 11:41:41	
23/10/2019 11:34:30 hasta 25/10/2019 13:44:15	
22/11/2019 01:00:00 hasta 03/12/2019 14:50:34	

Tabla 1: Resumen datos ausentes

Total: 155 días

Representa un 42% de datos que faltan. Si hubiera como un 20%, se podría extrapolar usando los datos que ya están, pero en este caso hay demasiado huecos.

Por eso se va a llenar esos huecos con datos climatológicos encontrado en internet. Hay por ejemplo la velocidad del viendo cada media hora.

4.2. Datos auxiliares

En el sitio web [1], se puede encontrar archivos meteorológicos de 243 ciudades en el mundo. Así se puede obtener los datos de temperatura, humedad, velocidad y dirección del viento para Valencia cada hora o media hora. La estación meteorológica donde provienen estos datos está cerca del aeropuerto de Manises.

Estos datos podrán ayudar de dos formas:

- Completando los datos que faltan del *LabDER*. En efecto se puede utilizar los datos de velocidad de viento del aeropuerto para los momentos donde el anemómetro no funcionaba.
- Utilizándolos como nuevos *inputs* en el trabajo de predicción. En efecto, la temperatura o la presión pueden ayudar la predicción de producción eólica dado que estos parámetros influyen sobre los vientos.

#	Temperature	P at station (airport)	P sea level	relative humidity	Wind direction	mean wind speed	Cloud coverage	horizontal visibility	dewpoint temp
Local time in Valencia (airport)	T	P0	P	U	DD	Ff	c	VV	Td
01/09/2020 23:30	24	755,1	759,7	65	var	1	15	10	17
01/09/2020 23:00	25	755,1	759,7	61	N-E	2	15	10	17
01/09/2020 22:30	25	755,1	759,7	61	E	2	15	10	17
01/09/2020 22:00	25	755,1	759,7	61	E-NE	3	15	10	17
01/09/2020 21:30	25	754,4	759	61	E-SE	3	15	10	17
01/09/2020 21:00	25	754,4	759	65	E	3	15	10	18
01/09/2020 20:30	25	754,4	759	61	E	3	15	10	17
01/09/2020 20:00	26	753,6	758,2	58	E	3	15	10	17
01/09/2020 19:30	26	753,6	758,2	61	E	4	15	10	18
01/09/2020 19:00	26	753,6	758,2	61	E-SE	4	15	10	18
01/09/2020 18:30	26	753,6	758,2	61	E	4	15	10	18
01/09/2020 18:00	27	753,6	758,2	54	E	4	15	10	17
01/09/2020 17:30	26	753,6	758,2	61	E	4	15	10	18
01/09/2020 17:00	26	753,6	758,2	61	S-E	4	15	10	18
01/09/2020 16:30	26	753,6	758,2	61	S-E	5	15	10	18
01/09/2020 16:00	26	754,4	759	58	E-SE	4	15	10	17
01/09/2020 15:30	26	754,4	759	54	S-E	3	0	10	16

Figura 23: Datos meteorológicos

Pero los datos de velocidad de viento en Manises no se pueden utilizar directamente para completar los del *LabDER*. En efecto, pueden dar una idea del viento al instante t en Valencia, pero los vientos

tienen una componente aleatoria importante. Además, los datos son medidos a 12m de altura mientras que el aerogenerador del *LabDER* está a 24m de altura. Hay que adaptar los datos.

Por eso se van a implementar los datos en el software HOMER, se crea un archivo wind.wnd con los datos de viento de Manises cada hora durante el año estudiado.

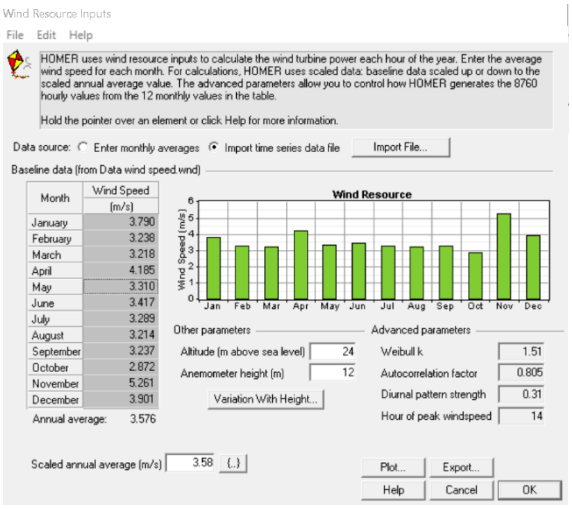


Figura 24: Velocidad de viento con HOMER

Después de haber importado los datos en HOMER se puede cambiar los valores de Altura del anemómetro y de altura del viento para obtener los datos que queremos.

En este caso los datos estaban medidos a 12m de altura en Manises y la turbina del *LabDER* está a 24m de altura.

Se obtienen los datos de irradiancia de Valencia utilizando la herramienta *PVGis*. Pero todavía faltan los datos de producción que no se pueden encontrar online.

Hay que encontrar una relación entre estos datos meteorológicos y los datos de potencia generada por ambos dispositivos: paneles fotovoltaicos y turbina eólica.

Para los paneles hay un factor directo entre la potencia solar llegando al panel y la potencia eléctrica que sale del panel. Es la eficiencia:

$$\eta = \frac{P\ electrica\ saliendo}{P\ solar\ llegando\ al\ panel}$$

Ecuación 10: Eficiencia PV

El valor de la eficiencia está en el *datasheet* del panel. En el caso del *LabDER* los paneles tienen una eficiencia de 15.1%.

Así se puede calcular la potencia generada por los paneles a partir de la potencia solar llegando a los paneles. Solo hay que relacionar la potencia llegando a los paneles a la irradiancia solar.

$$P_{solar\ llegando\ a\ los\ paneles}\ [W] = A[m^2] * Irradiancia[\frac{W}{m^2}]$$

Ecuación 11: Relación P_{solar} e irradiancia

Donde A es el área efectiva de los paneles. En el *LabDER* **A=16.4m²**

Así se puede calcular:

$$P_{electrica} = \eta * A * Irradiancia = 0.151 * 16.4 * Irradiancia = 2.48 * Irradiancia$$

Ecuación 12: $P_{eléctrica}$ a partir de la Irradiancia

Se implementa esta fórmula sencilla en Excel para obtener una aproximación de la potencia producida para los valores ausentes. Hay que insistir sobre el hecho de que solo sea una aproximación, no se considere el impacto del inversor cargador. Además, los valores de irradiancia del HOMER pueden cambiar de la realidad. En efecto, los valores son irradiancias globales de Valencia, en un punto preciso como el *LabDER*, puede existir diferencias según el paso de nubes, por ejemplo.

Así se obtienen los valores de potencia generada para los paneles fotovoltaicos.

En cuanto a la potencia generada por la turbina eólica es un poco más complicado. En efecto, la relación entre la potencia generada y la velocidad del viento no es lineal. Para resolver este problema se dispone de informaciones del constructor.

El *datasheet* da una curva $\mathrm{P_{generada}} = \mathrm{f(velocidad\ del\ viento)}$.

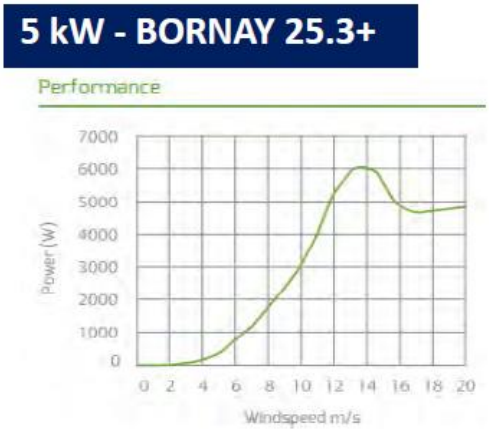


Figura 25: Curva de generación de la turbina

En este caso la solucion elegida es de encontrar una aproximacion de la ecuacion de esa curva. Para lograr esto, se utiliza la herramienta Excel donde se introduce algunos valores de la curva. Se agrega une curva tendencial polinomial para acercarse de la curva real. Se aumenta el grado polynomial para llegar a una curva tendencial bastante cerca de la real. En este caso se elijó un polinomio de grado 5.

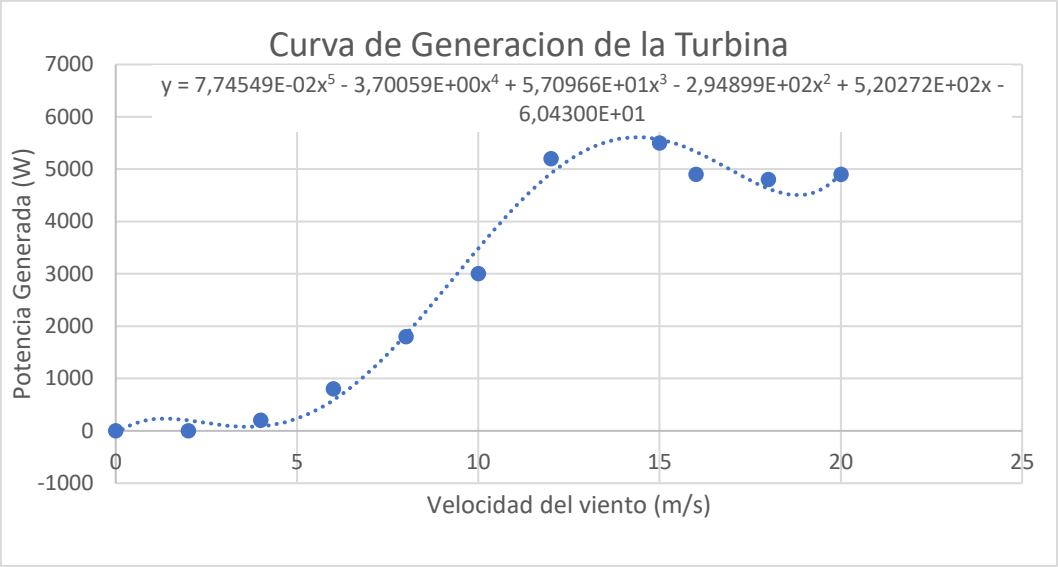


Figura 26: Curva tendencial de generación de la turbina

Se puede obtener la ecuación de la curva tendencial y después utilizarla para calcular la potencia generada por la turbina a partir de los valores de velocidad de viento.

Esto también sigue siendo una aproximación para obtener los valores ausentes del *LabDER* y tener un año completo de datos.

Se utilizan también los datos meteorológicos para completar los datos *LabDER* con otras informaciones. Se seleccionan los valores de temperaturas, presión y de nubosidad.

- La temperatura en grados Celsius. (°C)
- La presión en milímetros de mercurio. (mmHg)
- La nubosidad en porcentaje donde 0% significa un cielo sin nubes y 100% una cubertura total.

Finalmente se dispone de datos durante todo un año de marzo 2019 a febrero 2020. Se puede crear un archivo CSV y empezar a trabajar con *Python* y en particular la herramienta *Pandas* que permite manejar datos de gran tamaño.

4.3. Análisis de datos

Se importan los datos en *Python* para empezar el tratamiento. Con la herramienta *Pandas* se puede manejar muy fácilmente los archivos CSV y los datos numerosos.

Por ejemplo, la importación se hace mediante una sola línea de código:

```
1. import pandas as pd
2. dataProd=pd.read_csv("dataProd.csv",';',index_col=[0,1])
```

Se importa la herramienta y después solo hay que indicar el nombre del archivo y las columnas de referencias para las tablas. En este caso hay que filtrar según la fecha y la hora, así las columnas 0 y 1 servirán para el índice de la tabla.

Pandas forma una tabla con los datos que se puede manejar de forma muy simple. Entre otras cosas, se puede filtrar por un periodo o una hora específico, hacer promedios, trazar curvas.

		PPV	PWind	WindSpeed	SolarIrr
Date	**Time**				
2019-02-27	**16:30:00**	781.534570	1.153769e+02	2.880270	364.632378
	16:40:00	523.745763	1.351749e+02	1.825373	239.346939
	16:50:00	1276.584200	8.868883e+01	4.005000	515.500000
	17:00:00	855.348560	8.868883e+01	4.005000	345.400000
	17:10:00	855.348560	8.868883e+01	4.005000	345.400000
	17:20:00	855.348560	8.868883e+01	4.005000	345.400000
	17:30:00	300.084317	3.312061e-01	4.310793	140.745363
	17:40:00	224.553120	6.161764e-03	1.713699	108.121417
	17:50:00	159.038786	4.131890e-41	3.700523	79.118044
	18:00:00	104.350759	4.080580e-41	2.171653	54.580101
	18:10:00	55.548061	4.080580e-41	1.669865	34.387858
	18:20:00	29.308600	4.080580e-41	1.633153	20.794266
	18:30:00	17.602230	4.080580e-41	1.751164	11.610455
	18:40:00	10.000000	4.080580e-41	1.616870	6.603524
	18:50:00	10.000000	4.080580e-41	1.833103	264.900000

Figura 27: Tabla de datos con Pandas

Primeramente, se traza las curvas durante todo el periodo de estudio. Se hace con la línea de código:

```
1. dataProd.plot.area(subplots=True)
```

El parámetro *subplots* permite que las curvas no se superponen. Cada columna de la tabla tendrá su gráfico.

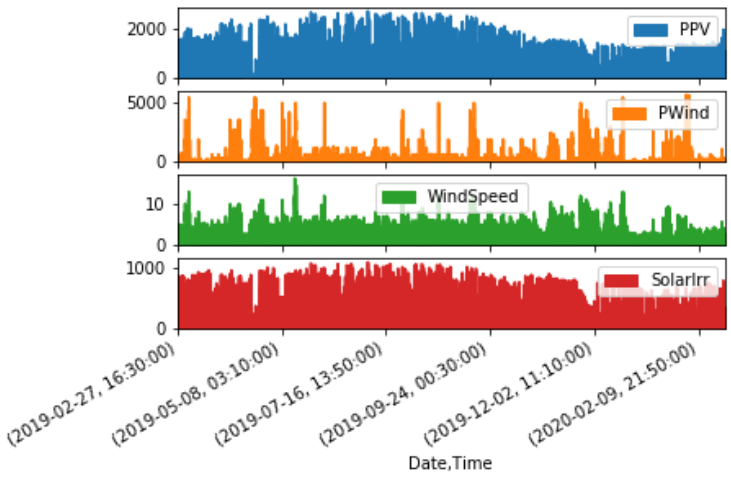


Figura 28: Curva anual de los datos LabDER

La potencia generada por los paneles (**PPV**) está en Watios (**W**) como la potencia eólica (**PWind**). La velocidad del viento (**WindSpeed**) está en **m/s** y la irradiancia solar (**SolarIrr**) en **W/m².**

A primera vista, se puede ver que las curvas **PPV** y **SolarIrr** se parecen mucho. Solo existe un factor entre la potencia producida teóricamente y la irradiancia solar como se estudió en la Ecuación 12.

Para las curvas de potencia eólica y de velocidad de viento la similitud no es tan evidente. Se puede ver algunos picos y valles en común, pero era más significante con los datos solares.

Es interesante ver que no hay producción eólica cuando la velocidad del viento es pequeña, aquí menor que 4-5 m/s. Bajo este umbral, no hay producción.

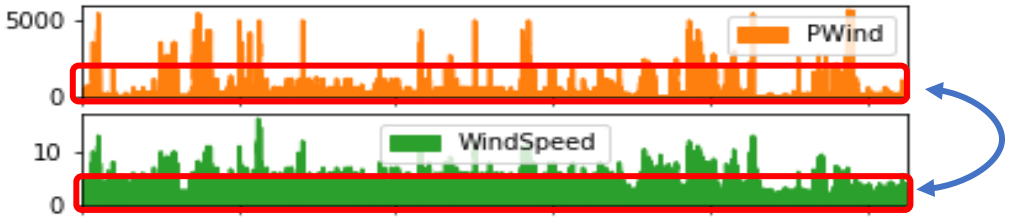


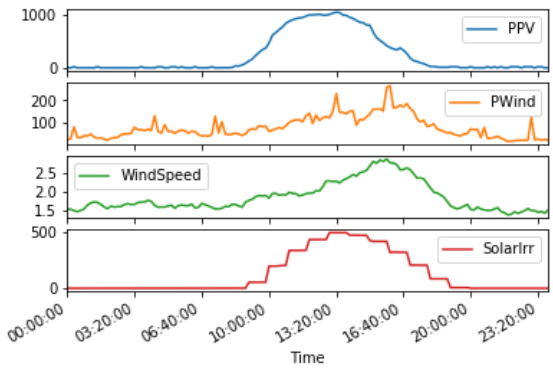
Figura 29: Umbral de la turbina eólica

Se verifica con la curva teórica de la turbina (Figura 25). La similitud entre las dos curvas es menos evidente que en el caso solar porque la relacion no es lineal. Es un análisis a nivel anual y entonces no se ve muy bien los detalles. Usando la potencia de *Pandas* se puede trazar las curvas promedias para cada mes del año. Así se puede observar más detalles y también las tendacias anuales.

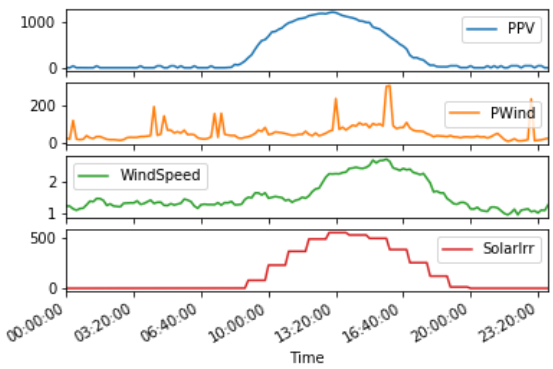
Hacer esto con Excel sería bastante difícil y largo mientras que con *Pandas* se hace en una línea:

```
1. dataProd.loc["2019-04":"2019-
   05"].mean(axis=0,level='Time').plot(subplots=True)
```

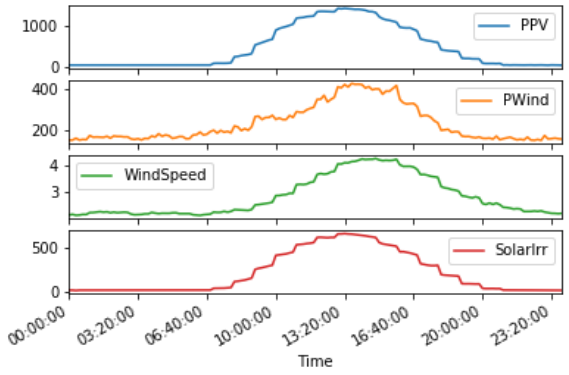
En primer lugar, se elige el periodo, aquí el mes de abril. Después, se indica que se quiere calcular el promedio según el axis 0 o sea el tiempo, el axis 1 siendo las diferentes columnas. Y después se trazan las curvas.



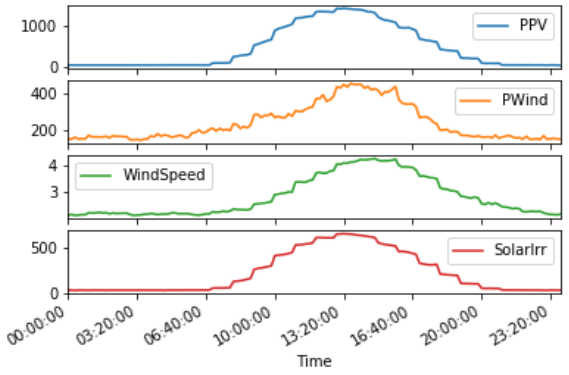
Curva media mensual 1: Enero



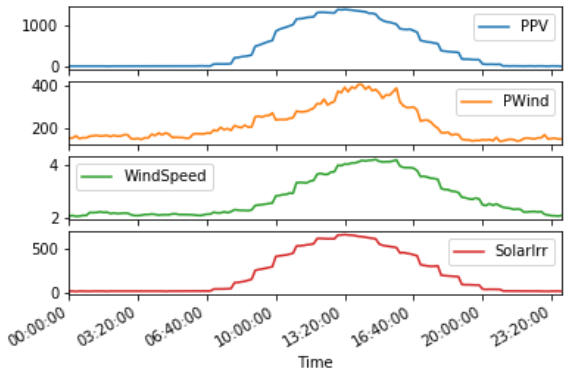
Curva media mensual 2: Febrero



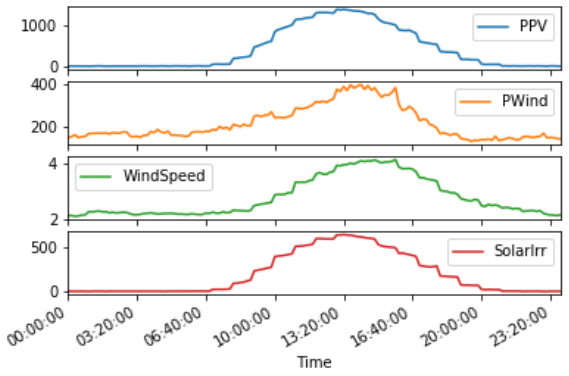
Curva media mensual 3: Marzo



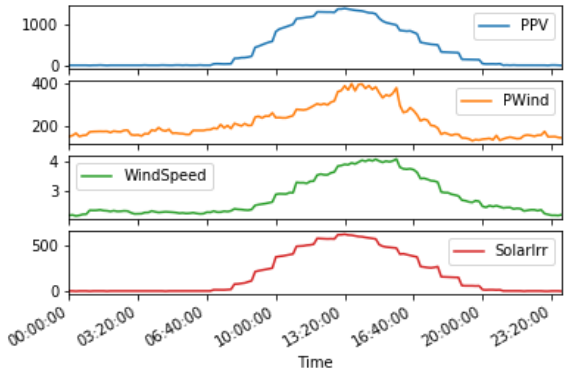
Curva media mensual 4: Abril



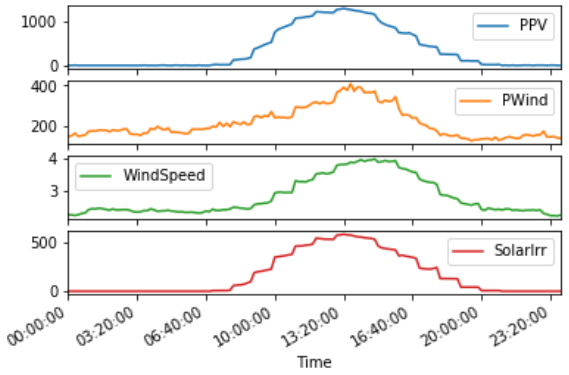
Curva media mensual 5: Mayo



Curva media mensual 6: Junio



Curva media mensual 7: Julio



Curva media mensual 8: Agosto

Curva media mensual 9: Septiembre

Curva media mensual 10: Octubre

Curva media mensual 11: Noviembre

Curva media mensual 12: Diciembre

Figura 30: Curvas medias mensuales

A nivel mensual, se puede observar que las curvas de potencia generada por los paneles (**PPV**) y las curvas de irradiancia (**SolarIrr**) son muy parecidas mientras que las curvas de potencia eólica y de velocidad de viento no lo son. En efecto, las curvas **PPV** et **SolarIrr** tienen una forma similar cada mes con algunas fluctuaciones. Para las curvas **Pwind** y **WindSpeed**, es más complicado. De mayo a septiembre, se puede observar similitudes entre las curvas mientras que para los otros meses las curvas son muy distintas.

Una teoría sobre este fenómeno es que entre de mayo y septiembre, el tiempo es bastante tranquilo, es la temporada caliente. Pero durante los otros meses, la temporada “fría” y hay mayor probabilidad tener episodios climáticos importantes como la gota fría, por ejemplo. Hay más vientos fuertes, rachas y como los datos auxiliares de velocidad de viento no son medidos en el mismo lugar (aeropuerto de Manises) que la turbina se puede tener valores distintos. Por ejemplo, puede ocurrir una racha muy fuerte en la UPV mientras que en Manises el tiempo está tranquilo al mismo tiempo. Sin embargo, solo es una teoría y puede existir diferentes razones a este fenómeno como la no linealidad entre ambas curvas.

Después del estudio al nivel mensual se puede decir que será posible usar los datos de irradiancia para obtener los de potencia producida por los paneles. Pero será más difícil utilizar la velocidad del viento para obtener la potencia generada. Son datos mensuales. Si se quiere obtener informaciones más

generales, hay que utilizar un indicador matemático más preciso. Se va a utilizar la matriz de correlación. La correlación sirve para saber si dos variables tienen una relación entre ellas.

$$r_{XY} = \frac{1}{n}\frac{\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y))}{s_X s_Y}$$

Ecuación 13: Factor de correlación

- Donde X y Y son las variables por que se calculan la correlación.
- E(X) la esperanza de la variable X.
- x_i , y_i los elementos de los vectores X y Y.
- s_X s_Y las varianzas de las variables X y Y.

Cuando el factor r_{XY} es cerca de uno, significa que las variables X y Y están relacionadas. Cuando r_{XY} está cerca de 0 eso significa que las variables son independientes.

En el caso de la predicción de producción de energía, las variables que están relacionadas con las de potencia pueden ser útiles para predecir la producción. Tienen una relación con la potencia producida.

Por eso se va a trazar la matriz de correlación de nuestros datos. Una matriz de correlación funciona como una tabla con dos entradas donde se calcula el factor de correlación para cada combinación. Con la herramienta *Pandas* se hace muy fácilmente. Se utiliza también la biblioteca *seaborn* que permite hacer tablas más visuales con colores.

```
1. sn.heatmap(dataProd.corr(),annot=True,cmap="YlGnBu")
```

Para calcular la matriz de correlación, solo se utiliza el comando *dataProd.corr()*. El resto de los parámetros sirve para elegir los colores y poner los valores dentro de la matriz.

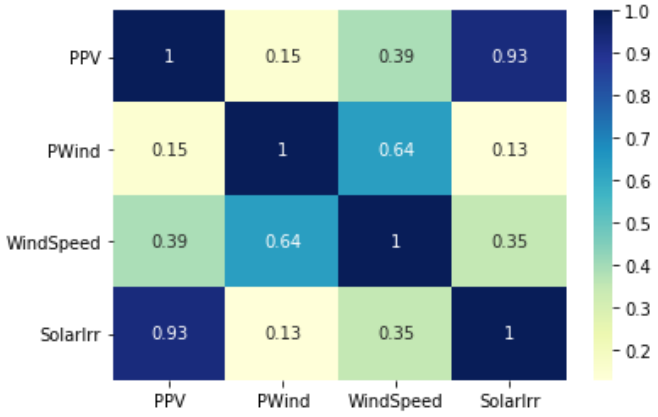


Figura 31: Matriz de correlación

Por definición del factor de correlación, la matriz es simétrica. En efecto r_{XY}=r_{YX}.

Con esa matriz se puede ver que la irradiancia solar está muy relacionada con la potencia producida por los paneles (**PPV**). También se puede observar que la potencia generada por la turbina eólica esta correlacionada a la velocidad del viento por un factor de 0.64, lo que es bastante bien.

Sin embargo, se puede notar que la potencia generada por los paneles solares no está relacionada con la potencia generada por la turbina. Por eso, se puede separar los datos entre los solares (PPV y SolarIrr) y los de viento (Pwind y WindSpeed)

Ahora se va a estudiar los otros parámetros que vienen de los datos auxiliares: presión, temperatura y nubosidad para descubrir si están relacionados con la producción solar o eólica.

Por eso se traza de la misma manera la matriz de correlación añadiendo los datos auxiliares.

```
1. data=pd.merge(dataProd,dataMeteo,how="left", on
   =('Date','Time')).interpolate()
2. sn.heatmap(data.corr(),annot=True,cmap="YlGnBu")
```

La primera línea corresponde a la fusión de los datos de producción (*LabDER*) y de los datos meteorológicos (datos auxiliares). La segunda línea sirve para trazar la siguiente matriz.

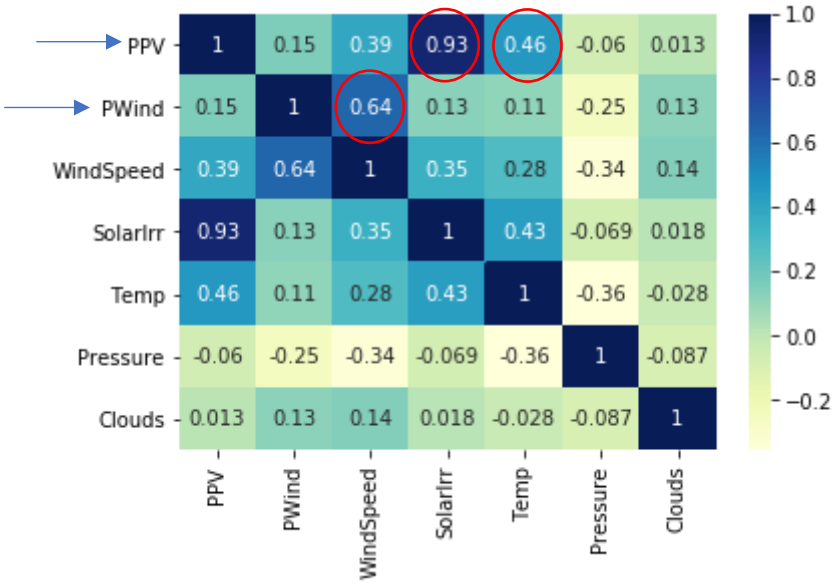


Figura 32: Matriz de correlación con datos meteorológicos

Para analizar esta matriz, hay que mirar los valores estudiados: **PPV** y **Pwind** que son los valores que se van a predecir. Hay que encontrar las variables relacionadas con ellas. Por eso, se seleccionan las líneas 1 y 2 y se observan las variables que tienen un factor de correlación superior a 0.4 (umbral elegido). Así para la potencia generada por los paneles fotovoltaicos, se seleccionan las variables de irradiancia solar y de temperatura. Y para la potencia generada por la turbina eólica, solo se selecciona la velocidad del viento.

Pero en este TFM, el objetivo es de predecir la producción por delante. Por eso hay que observar la correlación temporal de esos variables.

5. Predicción de la producción de la planta fotovoltaica

El objetivo de esta parte es la predicción de la producción de energía por los paneles fotovoltaicos. Es decir que por ejemplo el lunes se quiere estimar la curva de producción solar del martes.

En la parte de presentación de los datos, se elijó la temperatura y la irradiancia solar como parámetros para predecir la producción solar. Pero no habrá acceso a los datos de temperatura y solar irradiancia del futuro. Por ejemplo, el lunes, no se tendrá los datos de irradiancia y temperatura del martes. Sino estimaciones meteorológicas, pero esos son predicciones. Por eso, hay que estudiar la correlación entre la irradiancia y la temperatura del presente con la producción eléctrica del futuro.

5.1. Análisis de datos

Se intentará primeramente predecir la producción un día adelante, es decir la producción de las próximas 24 horas.

Por eso se va a estudiar la siguiente matriz de correlación:

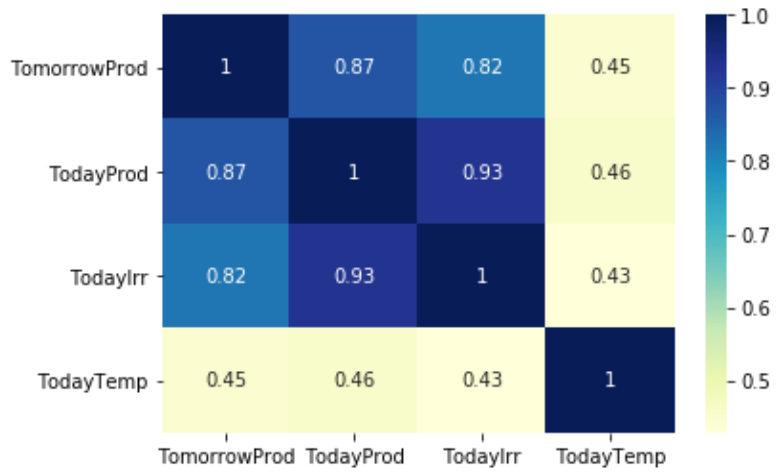


Figura 33: Matriz de correlación predicción solar

Se obtiene esa matriz usando el siguiente código:

```
1. tomorrowP=data['PPV'][6*24:].tolist()
2. todayP=data['PPV'][:-6*24].tolist()
3. todayIrr=data['SolarIrr'][:-6*24].tolist()
4. todayTemp=data['Temp'][:-6*24].tolist()
5.
6.
7. Solar24Corr=pd.DataFrame([tomorrowP,todayP,todayIrr,todayTemp]).trans
   pose()
8. Solar24Corr.columns=['TomorrowProd','TodayProd','TodayIrr','TodayTemp
   ']
9.
10.  sn.heatmap(Solar24Corr.corr(),annot=True,cmap="YlGnBu")
```

Es básicamente desplazar los valores de un día. Como hay 6 medidas por horas, corresponde a un desplazamiento de 6X24 elementos de las listas.

Se puede ver en la Figura 33 que la producción del día D está relacionada con la producción del día D+1 con un 0.87 lo que es mucho. Además, la irradiancia del D está también relacionada con la producción D+1 a un nivel alto. La temperatura también se puede usar para predecir la producción.

Así a primera vista, parece realizable predecir la producción solar un día adelante.

5.2. Tratamiento de los datos

En este TFM, se utiliza *Deep Learning* supervisado, es decir que para el entrenamiento de las redes se necesitan las salidas. Por eso hay que formar el *dataset* de la forma siguiente:

- X=datos de entrada
- y= datos de salida

Cada elemento de **X**, X[k], tiene su salida correspondiente en y[k]. Por el momento los datos son solamente series temporales. En efecto, todos los datos están a continuación cada 10 minutos, hay un dato de producción, de irradiancia, de temperatura.

En entrada **X** son los datos de producción, de irradiancia y temperatura de los días antes y en salida **y** se quiere la producción del día siguiente.

Se decide usar los datos de 48h antes para predecir la producción del siguiente día. Para formar el *dataset* de la forma X,y se va a utilizar una forma de ventana móvil de tamaño 72h que cada vez pone las 48h primeras horas en la entrada **X** y las ultimas 24h en la salida **y**.

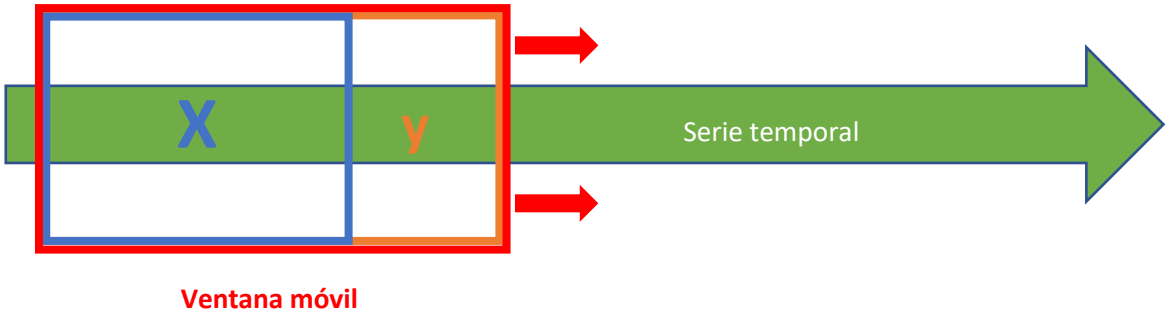


Figura 34: Esquema ventana móvil

Además, permite obtener artificialmente muchos más datos para entrenar las redes. En efecto, hay que definir de cuantos datos avanza la ventana en cada paso. Así se puede obtener una pareja [X,y] cada 10 minutos, cada hora o cada día. Pero hay que tener cuidado porque eso repite muchas veces los mismos datos iniciales y aparece redundancia. Podría ser una causa de sobre entrenamiento, habrá que adaptar el número de *epoch* considerando eso, dado que el número de *epoch* es el número de veces que pasa el *dataset* en la red neuronal para entrenar el modelo.

Después del entrenamiento de modelos sencillos con datos cada 10 minutos, los resultados estaban impresionantes. Demasiados buenos para ser verdaderos. En efecto, como la ventana se desplazaba de 10 minutos cada vez, había datos casi idénticos. Así los datos de prueba contenían datos casi idénticos a algunos del entrenamiento. Para arreglar ese problema de sobre entrenamiento, se decidió mover la ventana de 6 horas cada vez, haciendo un término medio entre el número de datos y la redundancia entre los datos.

Ahora que el principio de la ventana móvil está claro, hay que escribir el código Python para hacerla. Eso se hace mediante un bucle *for*. Se define el parámetro *look_back* que corresponde a la "distancia" donde se quiere mirar, en este caso será 48h X 6 datos/h, y el parámetro *prev* que corresponde a la distancia donde se quiere ver en el futuro, en este caso 24h X 6 datos/h.

El código es el siguiente:

```
1. def split_sequence(data, look_back,prev):
2.     X=[] #initialisacion de las listas
3.     y=[]
4.     for i in range(len(data)):
5.         if i+ look_back+prev>len(data)-1: #llega al final de la serie
6.             break
7.         else:
8.             X.append(data[i:i+look_back])
9.             y.append(data[i+look_back:i+look_back+prev])
10.      return X,y
```

Eso permite crear las parejas [X,y] considerando solo la producción de energía. Hay que añadir los datos de temperatura y de irradiancia de la misma manera. Sin embargo, las parejas [X,y] formadas no se pueden usar así. En efecto, están en el orden de la serie temporal, y para que una red neuronal se entrena correctamente los datos deben ser uniformizados. Por eso hay que mezclar los datos. Se define la función *shuffle* que hará eso.

```
1. def shuffle(X,y):
2.     liste=[]
3.     X2=[]
4.     y2=[]
5.     for k in range(len(X)):
6.             liste.append([X[k],y[k]])
7.     random.shuffle(liste)
8.     for k in liste:
9.         X2.append(k[0])
10.          y2.append(k[1])
11.      return X2,y2
```

Los datos están casi listos para el entrenamiento. Sola falta unas etapas: *la normalización* de los datos y la repartición entre prueba y entrenamiento.

Los datos están en las unidades estándares es decir **W** para la potencia, **W/m²** para la irradiancia y **°C** para la temperatura; y con varios rangos de valores, por ejemplo, la potencia va de 0 hasta 2709W mientras que la irradiancia va de 0 hasta 1094W/m² y la temperatura de -1°C hasta 42°C.

Hay que normalizar por dos principales razones:

- Como los datos tienen rangos de valores diferentes, la red dará más importancia a los datos más grandes. En efecto si un dato esta de 2000 y otro de 2, el peso del primero será más significante cuando estará pasando por la red. Por eso la normalización permite que todos los parámetros tengan la misma importancia.
- Además, las redes neuronales funcionan mejor y convergen más rápidamente cuando los valores inyectados están entre -1 y 1.

La normalización se hace de manera muy simple, de forma lineal con la siguiente formula:

$$\hat{x} = \frac{x - \min(X)}{\max(X) - \min(X)}$$

Ecuación 14: Normalización de los datos

Donde X es el vector de datos, x un elemento de X y $\hat{x}$ el valor normalizado de x. Así los valores de X están entre 0 y 1.

En Python la función es:

```
1. def normalize(datas):
2.   X=[]
3.   max=datas.max()
4.   min=datas.min()
5.   for k in datas:
6.     X.append((k-min)/(max-min))
7.   return X
```

Hay que normalizar cada parámetro (producción, temperatura e irradiancia). Se puede hacer antes de crear los *dataset* o después, eso no va a cambiar los resultados.

La etapa final es la división de los datos entre los datos de entrenamiento y los datos de prueba. Permite evitar el sobre entrenamiento de la red. Se entrena sobre datos y después se hace pruebas sobre los datos de test que nunca ha visto la red. Eso permite asegurarse que la red sea útil para predecir con nuevos datos y no solamente con los que ya conoce. Por eso se dividen los datos con los siguientes porcentajes: 70% serán datos de entrenamiento y 30% datos de prueba.

Aquí se acaba el preprocesado de los datos solares, después queda cambiar un poco la forma de los datos de entrada según la red neuronal usada para el entrenamiento. En efecto un **RNN** no utiliza la misma forma de entrada que un **CNN,** por ejemplo.

5.3. Métodos de Machine Learning (teoría)

En esta parte, se estudiará los diferentes algoritmos utilizados para predecir la producción por los paneles solares. Se comparará la eficiencia de esos algoritmos. Para hacerlo hay que tener indicadores comunes. Cuando se entrena una red neuronal, se define una función de error que hay que minimizar mediante el entrenamiento, también se puede definir otras funciones de error para evaluar el desempeño del algoritmo.

Se vio en la introducción las diferentes funciones de error, como RMSE, MSE o MAPE, por ejemplo. Sin embargo, hay problemas con las funciones de error clásicas en el caso de la predicción de la producción solar. Con las MSE o RMSE, se calcula la diferencia entre el valor real y el valor estimado por la red. Como los valores están entre 0 y 1, la diferencia suele ser pequeña, y poniéndola al cuadrado la reduce aún más. Así la red tiene que minimizar valores de error, ya muy pequeñas, lo que resulta en una convergencia muy lenta. Uno se puede decir que usar el error MAPE va a solucionar ese problema, como es el error comparado con el valor real. Es como un porcentaje de error. Dado que no hay un cuadrado para poner la diferencia a un valor positivo, se puede que los errores por un lado y otro se compensen. Por eso hay que elegir funciones de error personalizadas. Se definen funciones de error

basadas en las RMSE para reglar el problema de MAPE. Y se modifica esa función para obtener valores más interpretables.

$$RMSE_{100} = 100 * \frac{\sqrt{mean((y_{true} - y_{predict})^2)}}{mean(y)}$$

Ecuación 15: Función de error RMSE100

Este error representa el porcentaje de error entre el valor real y el valor calculado, haciendo el cálculo como para el **RMSE**. Así es más fácil interpretar los resultados cuando por ejemplo se dice 34% de error.

Además, se usa otra función de error, también basada en la RMSE.

$$RMSE_W = \text{maxProd} * \sqrt{mean((y_{true} - y_{predict})^2)}$$

Ecuación 16: Función de error RMSEW

Con *maxProd* el valor de producción máximo que se utilizó para normalizar los datos. Así este error representa el error de predicción en Watios. Se puede por ejemplo decir que nuestra predicción tiene un error de 300W. Pero según los datos, habrá días con mucha potencia generada (verano) y días con menos potencia generada (invierno). Así es difícil comparar los errores en Watios. En efecto un error de 500W un día donde la potencia pico es 3000W no es lo mismo que en un día donde la potencia pico es 700W. Por eso no se utilizará esa función para entrenar la red, pero es un indicador interesante de estudio.

En conclusión, para entrenar la red se usa $RMSE_{100}$ y se calcula $RMSE_W$ como indicador.

Se va a comparar 3 tipos de redes diferentes para predecir la producción solar. Un ANN, un CNN y por fin un RNN.

5.3.1. Entrenamiento con un ANN

El primer algoritmo utilizado para predecir la producción solar es un clásico A*rtificial Neural Network*. Como es la forma más básica de algoritmos de *Deep Learning,* permitirá establecer un resultado de referencia.

5.3.1.1. Datos de entrada

Los datos de entrada son directamente los datos creados con la ventana móvil. Es decir:

- En entrada los datos de producción, irradiancia y temperatura de los días D-1 y D-2.
- En salida los datos de producción del día D.

Así, en entrada se obtiene un vector de longitud: $n_{paramétros}$ X n_{horas} X $n_{datos/hora}$= 3 X 48 X 6 = 864.

En salida la longitud del vector es: n_{horas} X $n_{datos/hora}$ = 24 X 6 = 144.

5.3.1.2. Arquitectura

Se elige una red con cuatro capas escondidas. Es decir, cuatro capas más la de entrada y de salida.

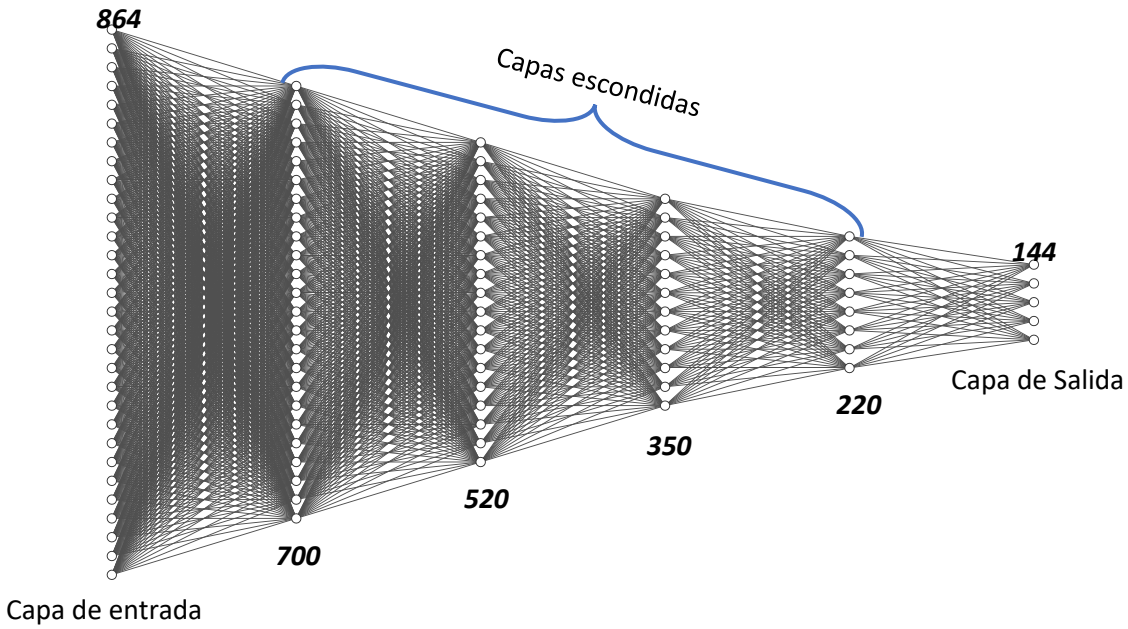


Figura 35: Arquitectura del ANN

La primera capa corresponde a la capa de entrada, no hay función de activación. Los valores en cada neurona corresponden al valor de entrada, o sea los datos de los días D-1 y D-2.

En las capas escondidas y la capa de salida, la función de activación es la sigmoide para que los valores sean entre 0 y 1 como los datos usados. En cada capa, las neuronas son conectados a todos los de la capa anterior y todos de la capa siguiente. Da muchos pesos que calcular y muchos parámetros que adaptar.

Implementar esa red usando *TensorFlow* y la herramienta *Keras* es muy sencillo. Hay que formar el modelo y después añadir cada capa de la red con los parámetros elegidos. Estos parámetros son el número de neuronas en la capa y la función activación para neuronas clásicas. Después, hay que compilar el modelo, es decir crear todas las variables, los pesos y los enlaces entre las capas. Durante esta etapa, hay que elegir la función de error que se tiene que minimizar, y también los indicadores elegidos.

Para acabar hay que entrenar el modelo, indicando:

- Los datos de entrada X_train
- Los datos de salida y_train
- El número de *epoch:* el número de veces se entrenará la red sobre los datos
- El tamaño de los *batch:* es el número de datos de entrada que entra en la red cada vez que se entrena. Un tamaño grande permite al modelo converger rápidamente, pero pierde en precisión. Mientras que con un tamaño pequeño se puede que el modelo no converja.

- También se puede definir datos de validación. Es decir, los datos que no participan al entrenamiento, para los cuales se calcula el error de predicción al final de cada *epoch.* Eso permite ver si el modelo no está en sobre entrenamiento.

Así el código es muy corto y sencillo.

```
#Creación del modelo
modelANN = tf.keras.Sequential()
modelANN.add(tf.keras.layers.Dense(700,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dense(520,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dense(350,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dense(220,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dense(144,activation='sigmoid'))

modelANN.compile(optimizer='adam', loss=mse100 , metrics=[mseW])

#Entrenamiento del modelo
modelANN.fit(X_train, y_train, epochs=200,
verbose=2,callbacks=[tensorboard_callback],batch_size=5,validation_data=(X_
test,y_test))
```

Con *TensorFlow* se puede ver un resumen del modelo con el formato de los datos en salida de cada capa y el número de parámetros calculados. Se hace mediante el siguiente comando:

```
modelANN.summary()
out:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 700)               605500
_________________________________________________________________
dense_1 (Dense)              (None, 520)               364520
_________________________________________________________________
dense_2 (Dense)              (None, 350)               182350
_________________________________________________________________
dense_3 (Dense)              (None, 220)               77220
_________________________________________________________________
dense_4 (Dense)              (None, 144)               31824
=================================================================
Total params: 1,261,414
Trainable params: 1,261,414
Non-trainable params: 0
```

Hay más de un millón de parámetros a entrenar. Puede parecer mucho, pero en el caso de redes neuronales no es tanto.

5.3.1.3. Entrenamiento

La red está formada, ahora hay que entrenarla. Por eso hay que definir los diferentes parámetros mencionados un poco antes.

Hay que elegir el número de *epochs.* Cuanto más se entrena, mejor serán los resultados sobre los datos de entrenamiento, pero puede ocurrir sobre entrenamiento. Además, un gran número de *epochs* significa un gran tiempo de computación. Como la mayoridad de los parámetros en *Deep Learning*, lo

importante es encontrar un término medio. Después de algunas pruebas se eligió un número de *epochs* de 200.

También, hay que elegir el tamaño de los *batch*. En *Deep Learning*, el entrenamiento se puede hacer de dos maneras: de manera estocástica o por *batch*. De manera estocástica, un dato de entrada pasa por la red, se calcula el gradiente del error y se adaptan los pesos para reducir el error. Por *batch*, un número de datos de entrada entran por la red, se calcula el gradiente medio sobre esos datos de entrada y se adaptan los pesos. Este método tiene algunas ventajas como el ahorro de tiempo de computación, en efecto no se calculan los pesos y los gradientes para cada dato. Además, con *batchs* uniformes, la convergencia se alcanza más rápidamente. Pero el método por *batch* pierde en precisión en contra del método estocástico. Una vez más hay que encontrar un término medio para el tamaño de los *batch.* En el caso de la predicción solar los datos están separados de 6 horas por la ventana móvil. Hay predicciones que empiezan a las 00h, 06h, 12h, 18h. Es decir 4 tipos de datos. Para tener *batchs* uniformes, se elige un tamaño de *batch* de 5. Como se mezclan los datos, en la mayoridad de los *batchs* habrá diferentes tipos de datos.

El último parámetro que hay que indicar para el entrenamiento es el *optimizer*. Cuando se adaptan los pesos, se hace mediante el cálculo de gradiente, pero la adaptación de los pesos se puede hacer de diferentes maneras para lograr el mínimo de la función de error. Para mover los pesos de la red hasta ese mínimo, se utiliza los *optimizers*. Existen muchos diferentes y lo que hacen básicamente es adaptar la velocidad de bajada al mínimo de la función de error. Es decir, acelerar cuando no hay variaciones pequeñas, y decelerar cuando los caminos son más estrechos.

En este caso, se utilizó el algoritmo de optimización *Adam* que es el más usado y el más competente para los ANN.

Una vez todos los parámetros elegidos, se inicia la computación. Se obtiene informaciones después de cada *epoch* para seguir el entrenamiento.

```
Epoch 42/200
202/202 - 2s - loss: 53.1324 - mseW: 230.5392 - val_loss: 57.6507 - val_mseW:
244.6189
Epoch 43/200
202/202 - 2s - loss: 52.2452 - mseW: 225.9490 - val_loss: 58.5736 - val_mseW:
242.9533
Epoch 44/200
202/202 - 2s - loss: 52.5406 - mseW: 227.1527 - val_loss: 57.2811 - val_mseW:
240.3341
Epoch 45/200
202/202 - 1s - loss: 52.8854 - mseW: 228.1195 - val_loss: 58.5014 - val_mseW:
245.0080
Epoch 46/200
202/202 - 2s - loss: 52.6351 - mseW: 228.5224 - val_loss: 57.8049 - val_mseW:
243.8023
Epoch 47/200
202/202 - 2s - loss: 51.8300 - mseW: 225.5803 - val_loss: 57.0419 - val_mseW:
240.9104
Epoch 48/200
202/202 - 2s - loss: 53.0658 - mseW: 227.5234 - val_loss: 57.1545 - val_mseW:
240.0945
```

Se puede seguir la progresión con el número de *epoch*, el tiempo de ejecución de cada *epoch,* y los valores de error y de indicador par los datos de entrenamiento (*loss y mseW)* y los datos de validación (*val_loss y val_mseW*).

Estas informaciones se pueden utilizar en tiempo real con la herramienta *Tensorboard* para tener graficas.

5.3.1.4. Problemas encontrados

El mayor problema encontrado con ese algoritmo es el sobre entrenamiento. Se habló mucho de este fenómeno, pero se ve en los resultados del ANN elegido.

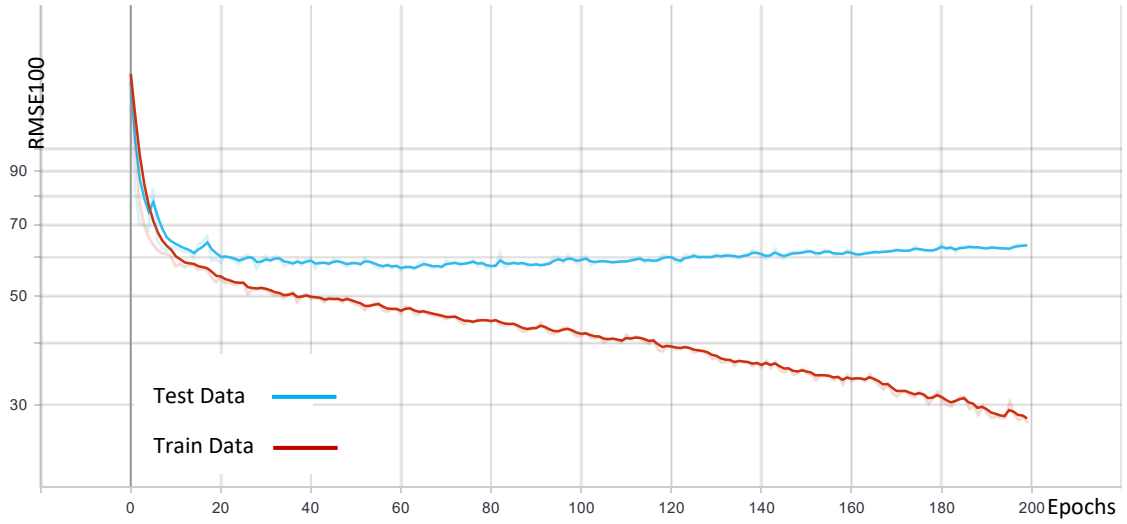


Figura 36: Curva de error con sobre entrenamiento

Se puede observar que, al principio, la curva de *Train,* igual que la curva de *Test* disminuyen con los *epochs.* Pero después del *epoch* 90, la curva de *Test* empieza a crecer mientras que la curva de *Train* sigue disminuyendo. Es decir que la red solamente aprende los datos de entrenamiento a expensa de los datos de prueba y del verdadero aprendizaje. Es un caso de sobre entrenamiento, la red tendrá un rendimiento fenomenal para los datos de entrenamiento, pero un desempeño muy malo para datos desconocidos. Así la red no tiene ninguna utilidad como se quiere predecir datos desconocidos del futuro.

Para paliar este problema, se utiliza el *Dropout.* El *Dropout* permite desconectar algunas conexiones y neuronas de manera aleatoria por cada *Batch.* Así el aprendizaje no puede establecerse en el mismo lugar y el sobre entrenamiento es menos presente. Se define las capas de *Dropout* entre cada capa de neuronas y con un valor de 0.3. Es decir que, para cada capa de neuronas, cada vez habrá 30% de neuronas desconectadas de manera aleatoria.

5.3.1.5. Resultados

Utilizar el *Dropout* permite evitar el sobre entrenamiento. Se va a observar las curvas de error y los indicadores.

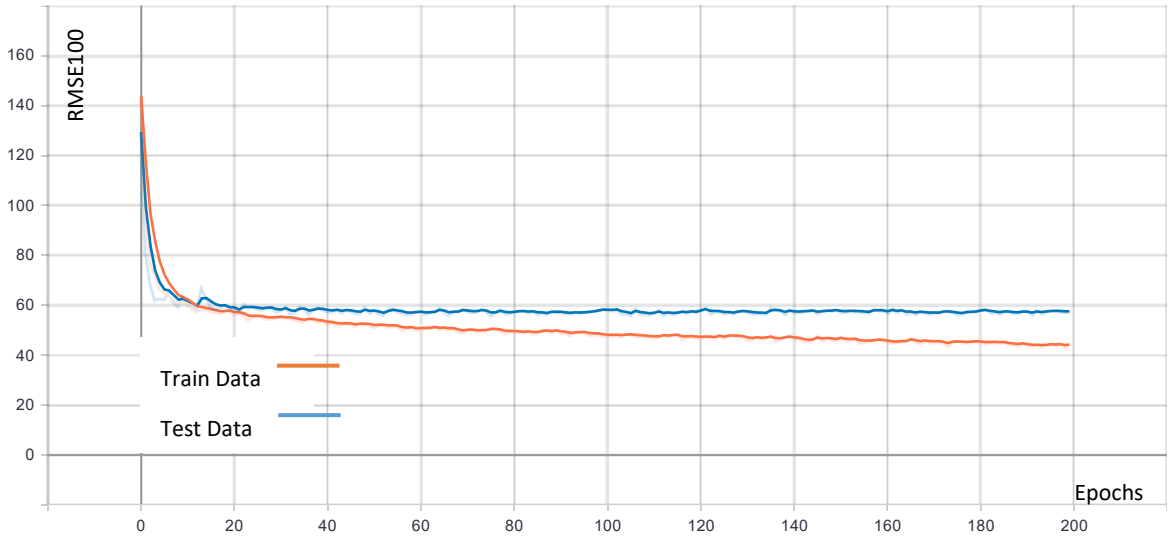


Figura 37: Función de Error ANN (RMSE100)

Comparando con la curva de error sin el *Dropout*, se puede ver que no hay sobre entrenamiento en este caso. Para los datos de test, se obtiene al final del entrenamiento un error de **58%** es decir una precisión de **42%** usando la Función de error RMSE100.

Además, se puede trazar curvas para el indicador elegido antes, la Función de error RMSEW. Aquí también se puede ver que no hay sobre entrenamiento. Por los datos de prueba, el error se estabiliza alrededor de **250W** de error medio.

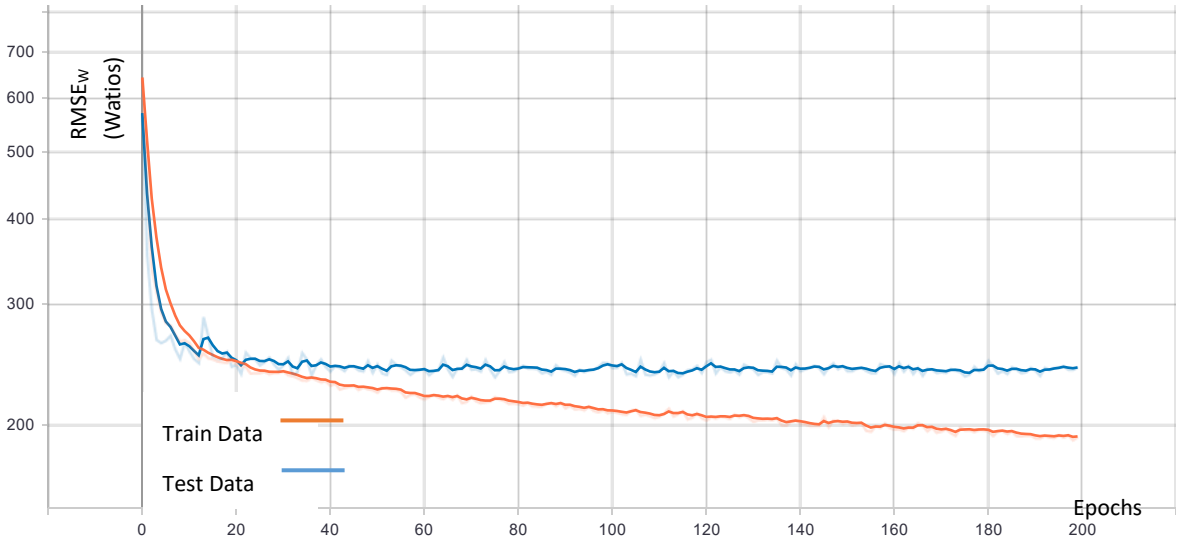


Figura 38: $RMSE_W$ ANN

Eso puede parecer muchísimo, pero hay que observar el comportamiento real de la red usando los datos de prueba. Por eso, se va a seleccionar algunos datos de prueba, desconocidos por la red, y se intentará predecir la producción para esos días.

Para no tener que entrenar cada vez el modelo, se guarda el modelo entrenado y los pesos descargándolo. Eso se hace usando el siguiente comando:

```
modelANN.save('ANN.h5')
```

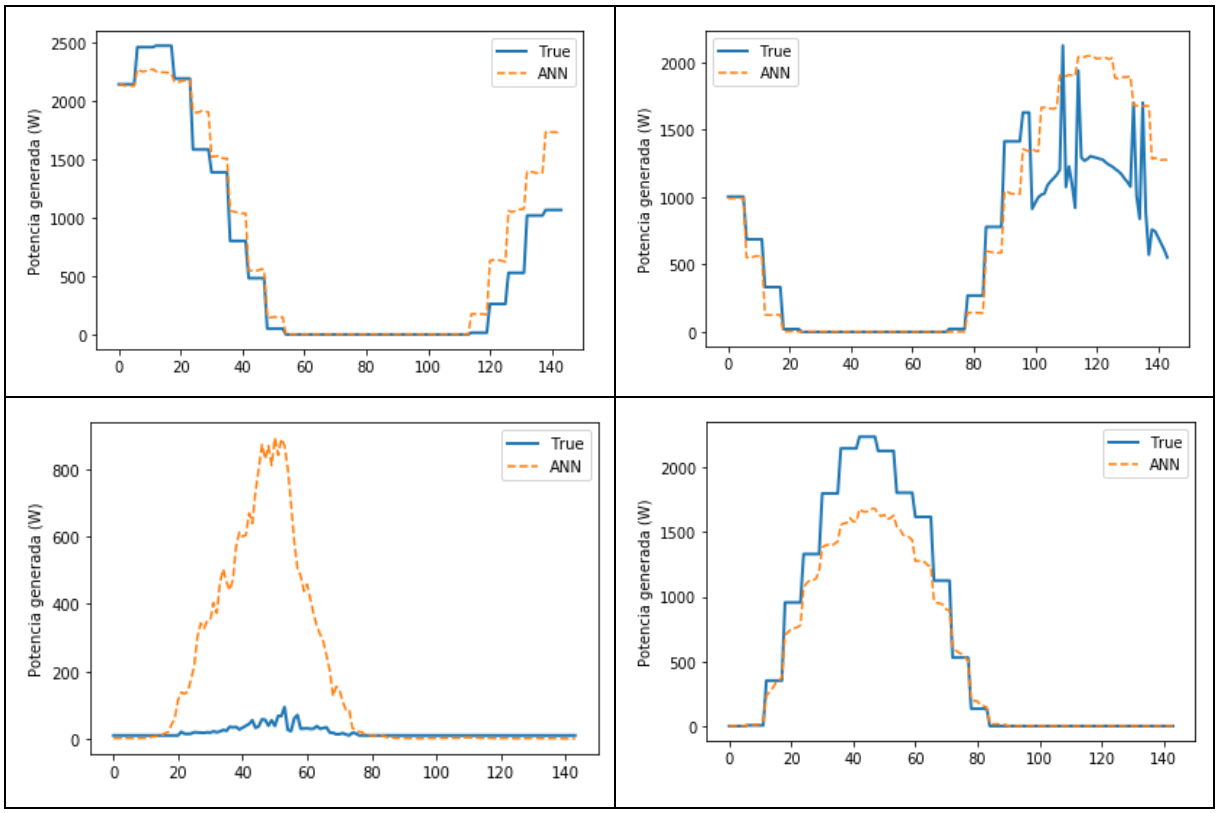
Después se puede cargar el modelo con esa línea de código sin tener que esperar el entrenamiento del modelo.

```
modelANN=tf.keras.models.load_model('modelANN.h5',custom_objects={"mse100":
mse100,"mseW":mseW})
```

Después, se predice el consumo de los datos de prueba. Eso es básicamente dar valores de los días D-1 y D-2 en entrada y calcular el resultado usando los pesos que ahora están fijos. Finalmente se traza dos curvas, la curva de producción real de los datos, y la curva calculada por el modelo.

```
predANN=modelANN.predict(X_test)
n=0
plt.plot(y_test[n]*2709,label='True',linewidth=2) #poner a escala
plt.plot(predANN[n]*2709,'--',label='ANN')
plt.legend()
plt.xlabel("Tiempo (s)", size = 10,)
plt.ylabel("Potencia generada (W)", size = 10)
```

Se cambia el valor de n para ojear a los datos de pruebas. Aquí se va a poner varios ejemplos de predicciones.



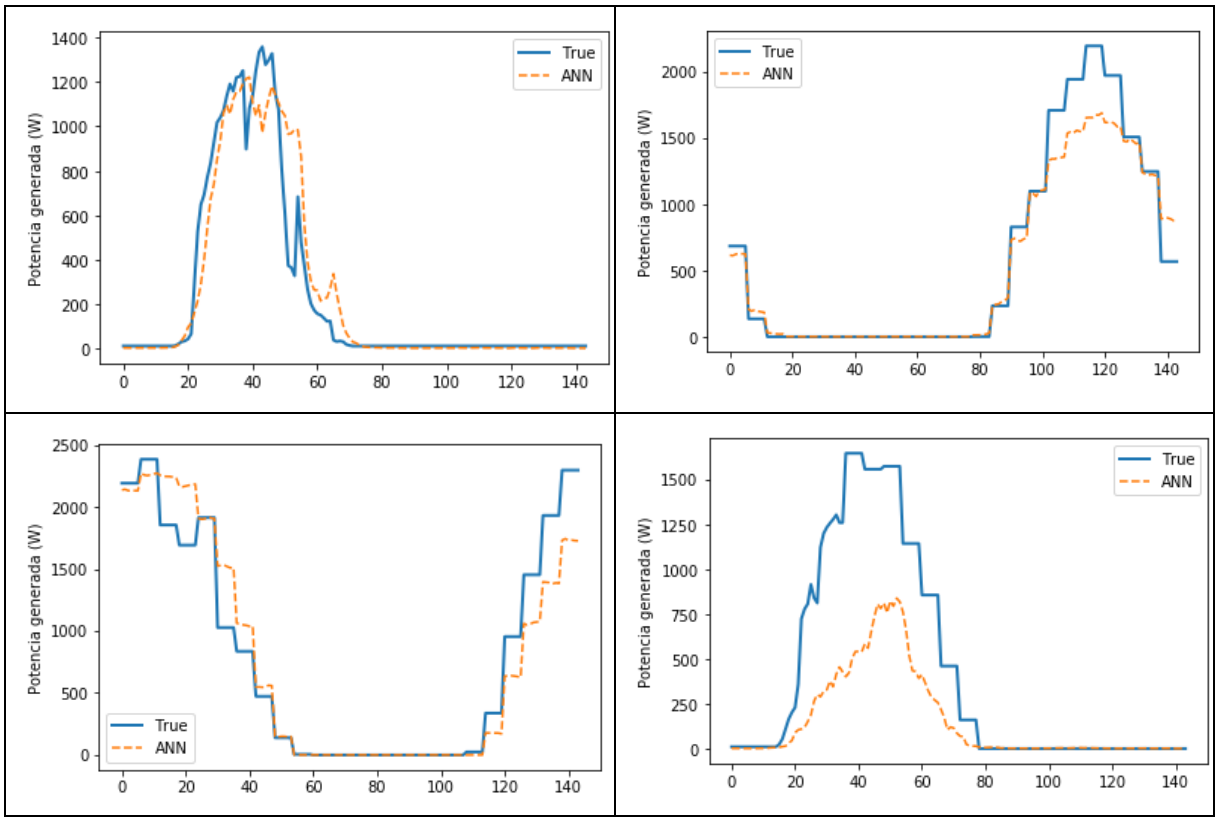


Figura 39: Ejemplos de predicción con el ANN. Eje X en datos con un dato cada 10 minutos

Mientras un error final significante (**58%**), se puede ver que la red permite obtener buenos resultados de manera general. Aunque por algunos días el error es muy grande, lo importante es que una red tan básica como esa permita predecir la tendencia global de producción.

Como se dijo al principio, esa red no será la más eficaz, pero servirá de referencia para comparar las otras.

5.3.2. Entrenamiento con un CNN

En esa parte se implementará y se utilizará una red de convolución para predecir la producción solar del *LabDER*. Esas redes suelen ser usadas para el procesado de imágenes, pero se pueden usar para las series temporales si se reorganizan un poco los datos.

5.3.2.1. Datos de entrada

Los datos de entrada son los mismos que para el ANN, es decir:

- En entrada los datos de producción, irradiancia y temperatura de los días D-1 y D-2.
- En salida los datos de producción del día D.

En el caso del ANN los datos estaban vectores en seguida de 3 X 2 X 24 X 6 = 864. Como se dijo antes, los CNN están hechos para el tratamiento de imágenes. Así hay que darle una imagen en entrada. Para eso se necesita reorganizar los datos.

Para predecir la producción, hay tres características:

- La producción anterior
- La irradiancia anterior
- La temperatura anterior

Una imagen está codificada en tres canales: **Rojo, Verde, Azul**. Se va a codificar los datos en tres canales que son: **Producción, Irradiancia, Temperatura**. De esta manera, se obtendrá una imagen de los datos. Pero así solo son tres vectores de una dimensión y se necesita 3 matrices en 2D. Para cada canal hay 48 horas de datos, se decidió cortar los días en 4. Se obtiene 8 cuarto de días. Con cada cuarto de una longitud de 6 horas. Así cada canal será una matriz de 8 líneas por 36 datos. (6 horas y 6 datos por hora).

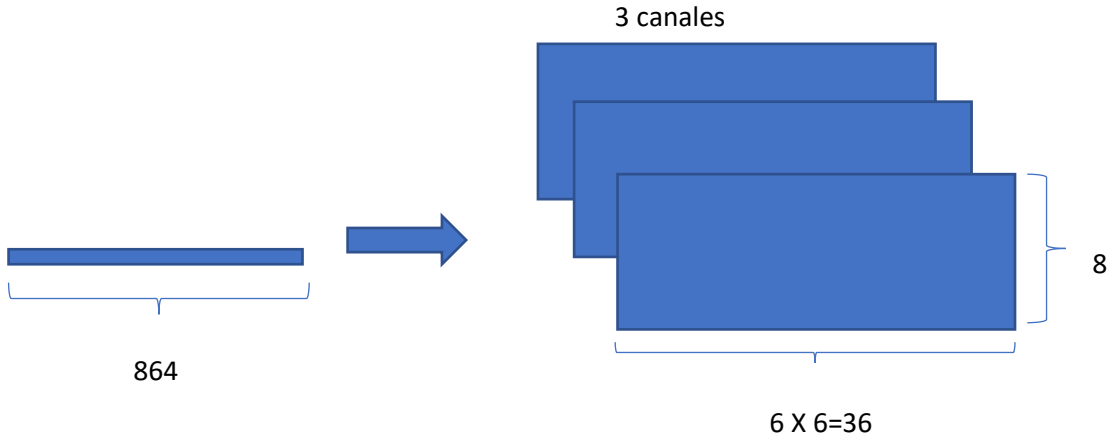


Figura 40: Reorganización de los datos

La reorganización de los datos se hace de manera muy sencilla con el siguiente código *Python*:

```
X_trainCNN=X_train.reshape(len(X_train),3,8,6*6)
X_testCNN=X_test.reshape(len(X_test),3,8,6*6)
```

Se utilizan los datos del ANN ya normalizados, mezclados y divididos entre entrenamiento y prueba para evitar un tiempo de computación adicional.

5.3.2.2. Arquitectura

Una vez los datos formados, hay que elegir la arquitectura de la red. No se hace en un único intento. Hay que probar cosas, echar lo que no funciona y seguir probando. Aumentando el número de filtros, disminuyendo la profundidad de la red, jugando sobre el tamaño de los filtros son elementos que van a influir sobre el rendimiento de la red.

Encontrar una red que funciona con el problema estudiado es un trabajo muy largo. En efecto, hay muchas cosas que probar y para cada ensayo, pueden pasar horas de tiempo de computación. Puede pasar días intentando cosas sin tener buenos resultados y sin avanzar.

Para el CNN la red elegida finalmente es una red con cuatro capas de convolución, una capa de aplanamiento, una de *Dropout* y por fin una de neuronas clásicas. La capa de aplanamiento es una capa muy sencilla. En efecto, en una red de convolución se tratan imágenes, pero en este problema

los datos son series temporales. Hay que hacer una transformación entre las imágenes en 2D y los vectores en 1D. Esta transformación se llama aplanamiento.

En el ANN los parámetros para las capas eran solamente el tamaño de las capas y la función de activación mientras que, en este modelo, hay muchos más. Hay que elegir:

- El número de filtros calculados para cada capa. Cuantos más filtros hay, más compleja es la red.
- El tamaño de los filtros. El tamaño del filtro influye sobre la disminución de las imágenes y el reconocimiento de formas.
- La manera de desplazar los filtros. De la misma manera que la ventana móvil en la parte preprocesamiento, los filtros se mueven sobre los datos.
- La función de activación de la capa.

Para cada capa de convolución se eligió los siguientes parámetros:

Capa 1. Una capa de convolución con **256 filtros**. Los filtros son de **tamaño (2,2)**. Se elige que la ventana móvil se mueve de una unidad cada vez con el parámetro ***padding='same'.*** Así el tamaño de las imágenes no se reducirá de mitad. La función de activación elegida es **ReLu** porque funciona bien con el procesamiento de imágenes.

Capa 2. Es la misma capa que la primera, pero aquí los filtros se mueven de dos en dos sobre la imagen dado que no hay el ***padding='same'***. Así se reduce la dimensión de las imágenes.

Capa 3. Una capa de convolución con **128 filtros**. Los filtros todavía están de **(2,2)**. Se usa el ***padding='same'***. La función de activación es ***ReLu***.

Capa 4. La última capa de convolución es la misma que la tercera, pero sin el ***padding='same'***. También se reduce la dimensión de las imágenes.

Después viene la capa de **aplanamiento**, el ***Dropout*** de **40%,** y por fin, una capa de neuronas interconectados con una **sigmoide** como función de activación.

Se puede observar el esquema de la red en la Figura 41.

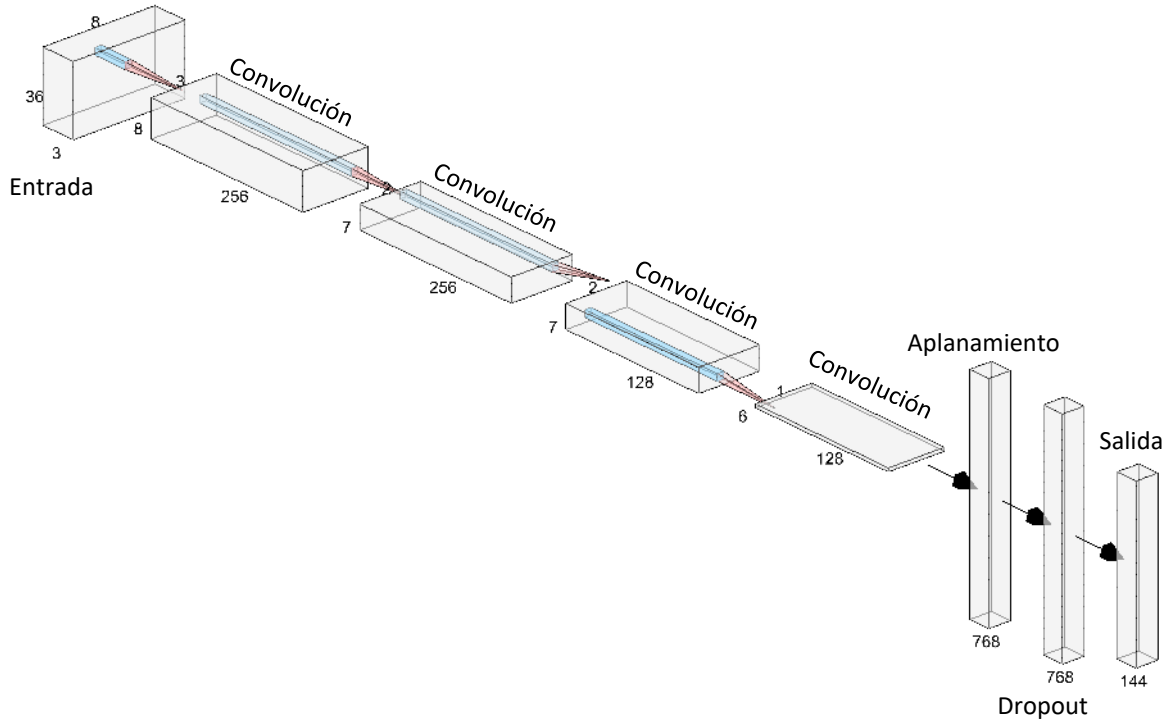


Figura 41: Arquitectura del CNN

Se puede ver cómo cambia la dimensión desde las imágenes de entrada hasta el vector de salida.

Para implementar la red en *Python* y *TensorFlow*, se hace de la misma manera que para el ANN, añadiendo las capas una después de otra. Después se hace la compilación y finalmente el entrenamiento.

```
modelCNN = tf.keras.Sequential()

modelCNN.add(tf.keras.layers.Conv2D(256,(2,2),activation='relu',input_shape
=(3, 8, 36),padding='same'))

modelCNN.add(tf.keras.layers.Conv2D(256,(2,2),activation='relu'))

modelCNN.add(tf.keras.layers.Conv2D(128,(2,2),activation='relu',padding='sa
me'))

modelCNN.add(tf.keras.layers.Conv2D(128,(2,2),activation='relu'))

modelCNN.add(tf.keras.layers.Flatten())
modelCNN.add(tf.keras.layers.Dropout(0.4))
modelCNN.add(tf.keras.layers.Dense(144,activation="sigmoid"))

modelCNN.compile(optimizer='adam', loss=mse100, metrics=[mseW])

#Entrenamiento
modelCNN.fit(X_trainCNN,y_train,epochs=200,verbose=2,batch_size=5,validatio
n_data=(X_testCNN,y_test))
```

De la misma manera que para el ANN, se puede ver el resumen de la red. Eso permitirá comparar el número de parámetros y las dimensiones de los datos después de cada capa.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 3, 8, 256)         37120
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 2, 7, 256)         262400
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 2, 7, 128)         131200
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 1, 6, 128)         65664
_________________________________________________________________
flatten_3 (Flatten)          (None, 768)               0
_________________________________________________________________
dropout_7 (Dropout)          (None, 768)               0
_________________________________________________________________
dense_8 (Dense)              (None, 144)               110736
=================================================================
Total params: 607,120
Trainable params: 607,120
Non-trainable params: 0
```

Hay 607 000 parámetros, casi la mitad de la red ANN. Pero eso no significa que la red sea más sencilla o menos eficaz. Hay que entrenarla para concluir sobre su desempeño.

5.3.2.3. Entrenamiento

Una vez la red creada, hay que entrenar el modelo. Para eso se elige los mismos parámetros que para el ANN. Es decir:

- La Función de error RMSE100.
- La Función de error RMSEW como indicador.
- 200 *epochs.*
- El tamaño de los *batchs* de 5.
- Datos de prueba (*X_test*) como datos de validación.
- El algoritmo *Adam* para optimizar el entrenamiento.

Se inicia el entrenamiento.

```
Epoch 1/200
202/202 - 3s - loss: 110.0691 - mseW: 484.6623 - val_loss: 65.8416 - val_mseW:
275.4400
Epoch 2/200
202/202 - 2s - loss: 66.3885 - mseW: 287.2015 - val_loss: 59.9709 - val_mseW:
249.3497
Epoch 3/200
202/202 - 2s - loss: 61.2111 - mseW: 263.1513 - val_loss: 58.7748 - val_mseW:
248.7656
Epoch 4/200
202/202 - 2s - loss: 57.3530 - mseW: 249.3628 - val_loss: 57.7570 - val_mseW:
241.8669
```

Mientras que hay dos veces menos parámetros que el ANN, el tiempo de computación de cada *epochs* es el mismo, es decir dos segundos. Se debe al hecho de que aparezcan productos de convolución en los cálculos de red. En efecto, en un ANN las operaciones son multiplicaciones y sumas, sin considerar la función de activación mientras que en el CNN se utilizan productos de convolución entre los filtros y los datos. Es un cálculo muy costoso en computación.

5.3.2.4. Problemas encontrados

La implementación de las redes de convolución es mucho más compleja que para las redes clásicas como el ANN. El mayor problema es que no se definen directamente las dimensiones de la salida de cada capa sino los filtros. Es un problema porque según el tamaño de los filtros y el número de ellos, la dimensión de los datos cambia. Así hay que ser muy atento a lo que se calcula y cómo funcionan las diferentes capas.

Los datos de entrada en la configuración con los tres canales no son muy grandes. Las redes de convolución suelen ser utilizadas para imágenes. Aquí la "imagen" es de 288 pixeles lo que es muy poco comparando con fotografías o videos de varios megapíxeles. Así hay que cuidar la dimensión de los datos. En efecto, los productos de convolución, debido al movimiento de los filtros sobre las imágenes, reducen el tamaño de las imágenes. No importa cuando las imágenes son muy grandes y, al contrario, es una buena cosa. Pero no es el caso de los datos usados en este TFM, se puede rápidamente llegar a tamaños demasiados pequeños. Por eso había que jugar con los parámetros de *padding* para conservar un tamaño suficiente.

Además, en los CNN, se suele usar capas de *pooling* para reducir más el tamaño de las imágenes. Es imprescindible cuando se quiere pasar de una imagen de 16 megapíxeles a un vector de 10 bits, en el caso de clasificación, por ejemplo. Pero aquí no es necesario debido al tamaño de nuestros datos.

Lo más difícil era encontrar una red capaz de extraer informaciones de los datos, pero conservando un tamaño suficiente para predecir los siguientes días de producción. Es una vez más encontrar un término medio.

5.3.2.5. Resultados

Con la herramienta *Tensorboard,* se traza la curva de error según los *epochs* y la curva de $RMSE_W$.

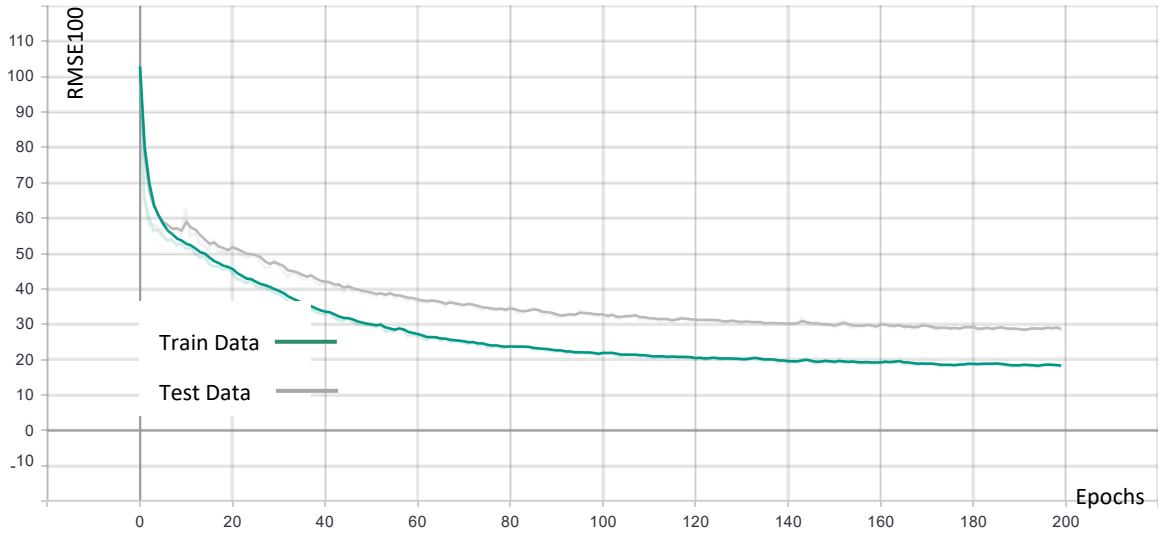


Figura 42: Función de Error CNN (RMSE100)

Con un CNN bastante sencillo, se puede alcanzar un error de 29% sobre los datos de prueba. Es decir, una precisión de **71%** sobre los datos *Test* y más de **80%** sobre los datos de entrenamiento. Es un resultado mucho mejor que el ANN en un tiempo de computación similar.

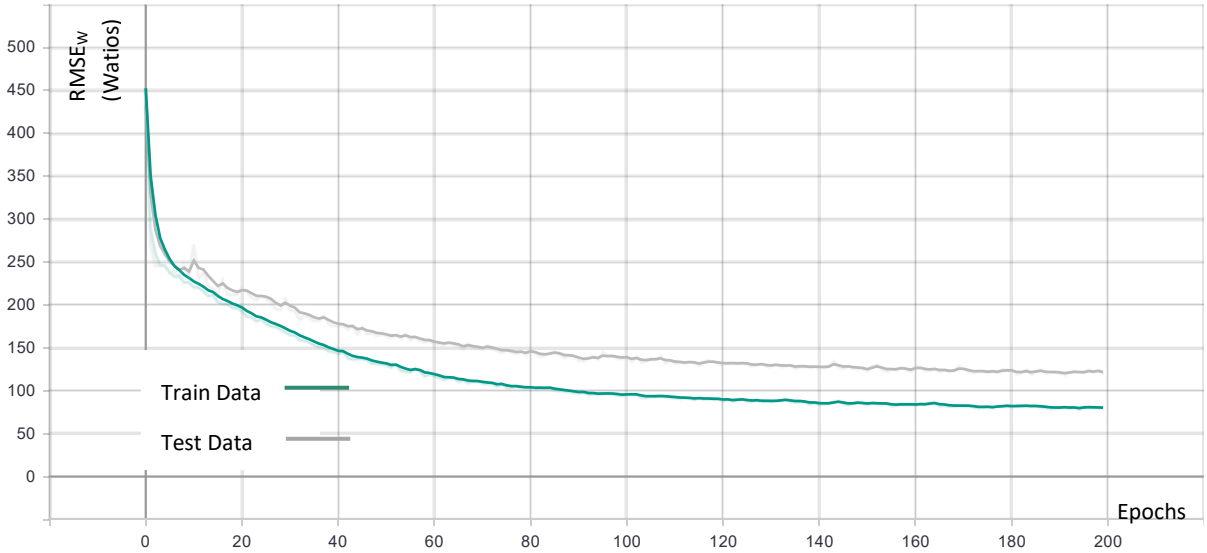


Figura 43: RMSE$_W$ CNN

Con el indicador RMSE$_W$ se puede ver un error medio de **120W** para los datos de prueba, lo que es muy poco comparando con el resultado del ANN.

Las curvas de aprendizajes dejen pensar que el CNN es una muy buena red para este problema. En efecto, en un tiempo de computación bastante rápido, considerando que el computador utilizado no es muy potente, los resultados parecen muy buenos. Pero antes de concluir es necesario observar algunas curvas de producción calculadas por la red y compararlas con las verdaderas curvas de producción. Como para el ANN, se guardan los datos de la red para poder utilizarlos sin tener que entrenar otra vez.

```
modelCNN.save('modelCNN.h5')
```

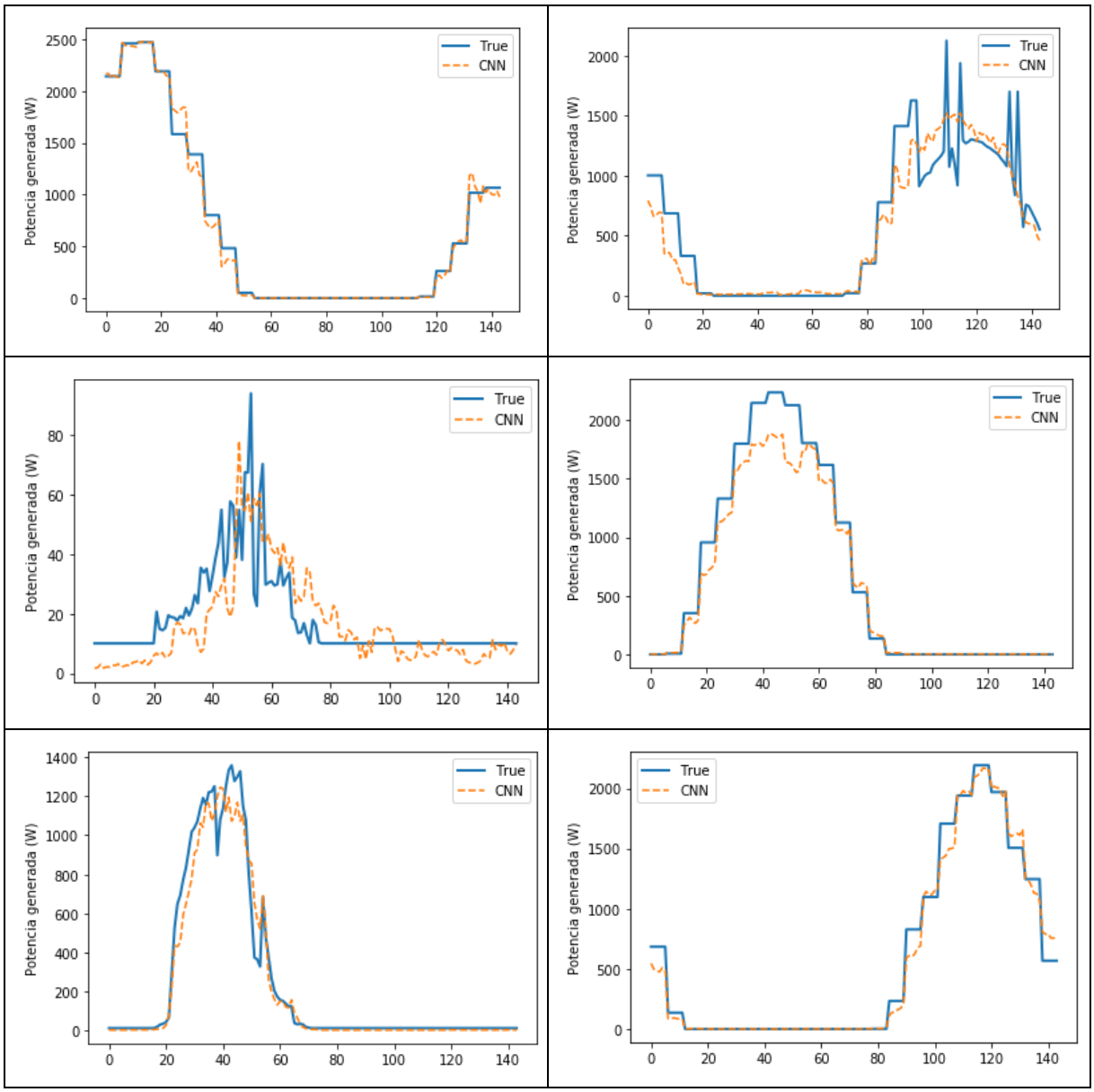
Después se traza algunas curvas de producción de la misma manera que para el ANN, pero utilizando los datos de entrada reorganizados.

```
modelCNN=tf.keras.models.load_model('modelCNN.h5',custom_objects={"mse100":
mse100,"mseW":mseW})

predCNN=modelCNN.predict(X_testCNN)

n=0
plt.plot(y_test[n]*2709,label='True',linewidth=2) #poner a escala
plt.plot(predCNN[n]*2709,'--',label='CNN')
plt.legend()
plt.xlabel("Tiempo (s)", size = 10,)
plt.ylabel("Potencia generada (W)", size = 10)
```

Se utilizan los mismos ejemplos que antes.



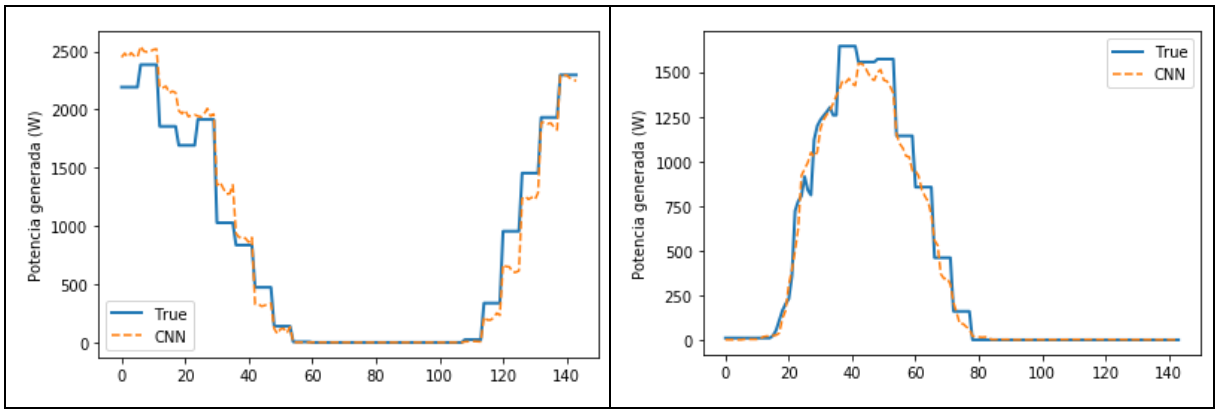


Figura 44: Ejemplos de predicción con el CNN. Eje X en datos con un dato cada 10 minutos

Se puede ver en los ejemplos que el desempeño del CNN es mucho mejor que el del ANN. En efecto, para cada ejemplo, la curva calculada es casi idéntica a la curva real mientras que por algunos ejemplos, el ANN tenía malos resultados para las curvas más raras, aquí el CNN cada vez es muy cerca de la curva real. El CNN podría ser un buen candidato para predecir la producción solar.

5.3.3. Entrenamiento con un RNN

El próximo algoritmo utilizado para predecir la producción solar de energía es un RNN. Las redes recurrentes suelen ser utilizados para trabajar con datos que tienen una dependencia temporal. Un ejemplo de uso es el tratamiento del lenguaje, como la traducción o la predicción de la próxima palabra en una frase. En particular, se utilizan redes recurrentes para escribir automáticamente artículos o libros.

También se usan los RNN en el caso de series temporales. Así esta red podría ser muy eficaz para predecir la producción de energía por los paneles.

5.3.3.1. Datos de entrada

Las redes recurrentes permiten predecir la o las etapas siguientes en una serie temporal. Así se considerará como datos de entrada solamente la producción de los días D-1 y D-2.

Los datos de entrada son vectores de longitud 288 = 2 X 24 X 6.

Como los datos ya están formados con la producción de los días anteriores, la temperatura y la irradiancia, hay que extraer de ellos solamente la producción. En efecto como se ha creado los datos de entrenamiento y prueba y todo está mezclado. Para guardar el mismo orden, es más sencillo extraer lo que se necesita de los datos que ya existen.

Para hacer esto, se escribe una función *OnlyProd* que permite sacar el *Dataset* con solamente los datos de producción de los días D-1 y D-2, quitando la temperatura y la irradiancia.

```
def onlyProd(X):
  X2=[]
  for k in X:
    X2.append(k[:int(len(k)/3)]) #se guarda el primer tercio
  return np.asarray(X2)
```

Después hay que reorganizar los datos de entrada para que sean compatibles con las capas de LSTM. Los datos se reorganizan en columna de manera que entran en las células LSTM. Por eso hay que crear el *Dataset* con la función *OnlyProd* y después cambiar la forma.

```
n_features = 1
n_steps=6*24*2 #prod

X_trainRNN = onlyProd(X_train)
X_testRNN = onlyProd(X_test)
X_trainRNN = X_trainRNN.reshape((X_trainRNN.shape[0], n_steps, n_features))
X_testRNN = X_testRNN.reshape((X_testRNN.shape[0], n_steps, n_features))
```

El número de *features* corresponde a las diferentes informaciones en entrada, en este caso se utiliza solamente la producción, así se pone *n_features=1*. Luego, se cambia la forma con el comando *reshape* cuyas componentes son:

- La primera componente (*X_trainRNN.shape[0]*) corresponde al número de datos.
- La segunda (*n_steps*) es la longitud de los datos de entrada, es decir la producción durante 2 días.
- La última (*n_features)* es el número de parámetros de entrada.

Los datos están formados, se puede crear la red.

5.3.3.2. Arquitectura

Después de muchas pruebas, se elige una red con tres capas de LSTM, una de *Dropout* y una última de neuronas clásicas.

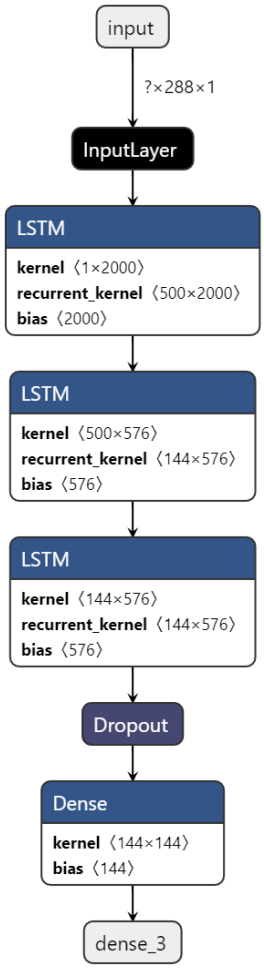


Figura 45: Arquitectura del RNN

Cuando se trata de un RNN, y más precisamente de capas de LSTM (*Long Short Term Memory*), hay que definir el número de bloques LSTM en cada capa. Además, existen diferentes parámetros en las capas LSTM, como el *return_sequences* o el *return_state*.

Los datos entran todos en cada bloque LSTM. Los cálculos se hacen con los datos de entrada y los de la memoria de la celda. Después hay dos posibilidades, el bloque puede sacar solamente el último resultado o puede sacar un vector con todos los datos intermedios. Es conveniente sacar todos los datos cuando se pone capas LSTM en seguida.

Para la primera capa, todos los vectores son los mismos, son los datos de entrada. Y en cada bloque LSTM, hay un número de celdas LSTM igual a la dimensión de vector de entrada. En efecto, cada celda utiliza una componente de este vector y también las memorias de las celdas anteriores.

Así cuando se pone el *return_sequences==True*, se saca todos los resultados de cada celda del bloque, y no solamente el último. Como se puede ver en la Figura 46.

Se define la red presentada en la Figura 45.

- La primera capa se compone de 500 bloques LSTM y el *return_sequences==True* porque la siguiente capa también es una LSTM.

- La segunda capa se compone de 144 bloques LSTM para disminuir hasta la dimensión de salida. El parámetro *return_sequences* también está activado.
- La última capa LSTM también contiene 144 bloques, pero con el *return_sequences* desactivado. En efecto la próxima capa no es una de LSTM.
- La cuarta capa es un *Dropout* de 50% para evitar el sobre entrenamiento.
- Y por fin una capa de neuronas clásicas con 144 neuronas.

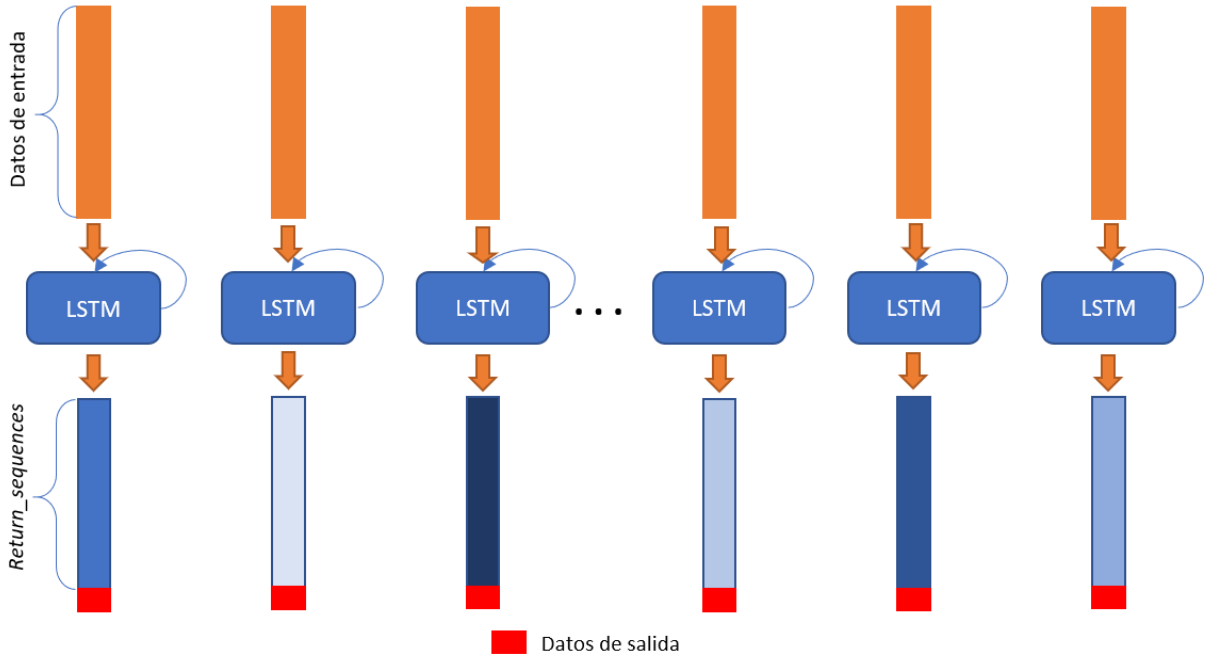


Figura 46: Capa LSTM

Se implementa la red de la misma manera que antes, poniendo cada capa después de otra. Después hay que compilar la red y por fin entrenarla.

```
modelRNN = tf.keras.Sequential()
modelRNN.add(tf.keras.layers.LSTM(500,input_shape=(n_steps,n_features),retu
rn_sequences=True))
modelRNN.add(tf.keras.layers.LSTM(144,return_sequences=True))
modelRNN.add(tf.keras.layers.LSTM(144))
modelRNN.add(tf.keras.layers.Dropout(0.5))
modelRNN.add(tf.keras.layers.Dense(144))

modelRNN.summary()
modelRNN.compile(optimizer='rmsprop', loss=mse100 , metrics=[mseW])

#Entrenamiento
with tf.device('/device:GPU:0'):
  modelRNN.fit(X_trainRNN, y_train, epochs=200,verbose=2,batch_size=5,valid
ation_data=(X_testRNN,y_test))
```

Con la función *summary*, se puede obtener un resumen de la red y el número de parámetros calculados.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 288, 500)          1004000
_________________________________________________________________
lstm_1 (LSTM)                (None, 288, 144)          371520
_________________________________________________________________
lstm_2 (LSTM)                (None, 144)               166464
_________________________________________________________________
dropout (Dropout)            (None, 144)               0
_________________________________________________________________
dense (Dense)                (None, 144)               20880
=================================================================
Total params: 1,562,864
Trainable params: 1,562,864
Non-trainable params: 0
```

Se obtienen 1 562 864 parámetros, es más que para el ANN. Se debe a una red mucho más compleja. Una celda de LSTM tiene más parámetros que una celda de ANN (una neurona).

5.3.3.3. Entrenamiento

La red está formada, hay que seguir con el entrenamiento. Los parámetros son los siguientes:

- La Función de error RMSE100
- La Función de error RMSEW como indicador
- 200 *epochs*
- Un tamaño de los *batchs* de 5.
- Datos de prueba (X_testRNN) como datos de validación
- El algoritmo *RMSprop* para optimizar el entrenamiento

Mientras los dos algoritmos anteriores se ejecutaban en local con la herramienta *Jupyter Notebook*, el RNN debido a la complejidad algorítmica se ejecuta con *Google Colab.*

Eso permite usar un **GPU** y una máquina mucho más potente.

```
Epoch 2/200
202/202 - 12s - loss: 102.6230 - mseW: 452.2551 - val_loss: 96.0820 -
val_mseW: 420.0054
Epoch 3/200
202/202 - 12s - loss: 91.6621 - mseW: 404.4178 - val_loss: 81.6383 - val_mseW:
352.7017
Epoch 4/200
202/202 - 12s - loss: 86.1854 - mseW: 378.2091 - val_loss: 74.9193 - val_mseW:
318.0141
Epoch 5/200
202/202 - 12s - loss: 80.9675 - mseW: 357.8081 - val_loss: 68.6213 - val_mseW:
293.3522
Epoch 6/200
202/202 - 12s - loss: 77.6507 - mseW: 340.9701 - val_loss: 84.3529 - val_mseW:
370.0103
```

Cada *epoch* se trata en 12 segundos mientras en local duraba casi 2 minutos. La herramienta *Google colab* permite ahorrar mucho tiempo. En efecto con mi portátil, el entrenamiento tardaba 200 X 2 min = 400 minutos es decir 6h30 mientras que con *Google Colab* solamente dura 40 minutos.

Además, como el tratamiento se hace en una maquina a distancia, se puede utilizar el portatil para hacer otras cosas, como trabajar sobre otros algoritmos.

5.3.3.4. Problemas encontrados

De las tres redes estudiadas en etsa parte, el RNN fue el más difícil al nivel de implementación. Había que entender cómo funcionan las capas LSTM que son bastante complejas y también crear una red que funcionaba.

El primer problema encontrado fue que las redes RNN formadas no convergían, es decir que el error no bajaba. Quedaba bloqueado a 130% para el RMSE100. A veces el número de *epochs* se aumentó hasta 500 y mismo 1000 para ver si la convergencia ocurría después pero no se producía.

Se intentó varias cosas:

- Reducir el número de capas mientras aumentar el número de bloques LSTM
- Aumentar el número de capas mientras reducir el número de bloques LSTM
- Cambiar las funciones de activación del LSTM por *ReLu, sigmoide, linear, SeLu …*
- Cambiar el número de capas de neuronas clásicas
- Cambiar el formato de los datos de entrada

Todo esto no cambió nada. Se intentó varias combinaciones de todos los parámetros sin tener buenos resultados. A veces, ocurrió lo que se llama una explosión de gradiente, es decir que el error aumentaba a un ritmo fenomenal en vez de reducirse.

La cosa es que una red con varias capas de LSTM es muy difícil de entrenar, puede ser muy inestable. Se puede entrenar dos veces la misma red y una vez el gradiente explota y la otra no.

Después de varios días sin resultado, se eligió cambiar de método intentando un *encoder/decoder RNN.* Es una forma de red que se suele utilizar para la traducción de texto. El texto entra en una red LSTM donde se reduce la dimensión hacia un pequeño vector para extraer el sentido de la frase, es el *encoder*. Después, el vector generado entra en una otra red LSTM donde la dimensión aumenta para convertir el sentido en una frase, pero en un otro idioma, es el *decoder*.

Se intentó hacer esto para convertir los datos de producción dos días antes en datos del día siguiente. Pero en este caso también no hubo buenos resultados.

Después de varios días, volví a intentar con el algoritmo inicial, cambiando la única cosa que no se había cambiado. En efecto, desde el principio se utilizó el algoritmo de optimización *Adam* porque es lo que se suele utilizar y es uno de los más eficiente. Pero en el caso de las redes recurrentes, este algoritmo no es muy estable. Por eso, se cambió por el *RMSprop* que es un algoritmo bastante eficiente y con un funcionamiento más simple que el *Adam*. Además, *RMSprop* tiene mejores resultados en el caso de redes recurrentes.

Hubo resultados a partir del primer entrenamiento. Los resultados no estaban espectaculares pero el error bajaba durante el entrenamiento. A partir de este momento, había que jugar con la arquitectura para encontrar una red eficaz. También había que ser atento a las funciones de activación. En efecto, las funciones *ReLu* hacían explotar el gradiente mientras que las *sigmoides* hacían la convergencia más difícil. Después de muchísimas pruebas, se elijo las funciones de activación *linear*.

El último problema estaba el tiempo de ejecución muy largo, pero se resuelto usando la herramienta *Google Colab*.

5.3.3.5. Resultados

Se trazan las curvas de aprendizaje con la herramienta *Tensorboard*.

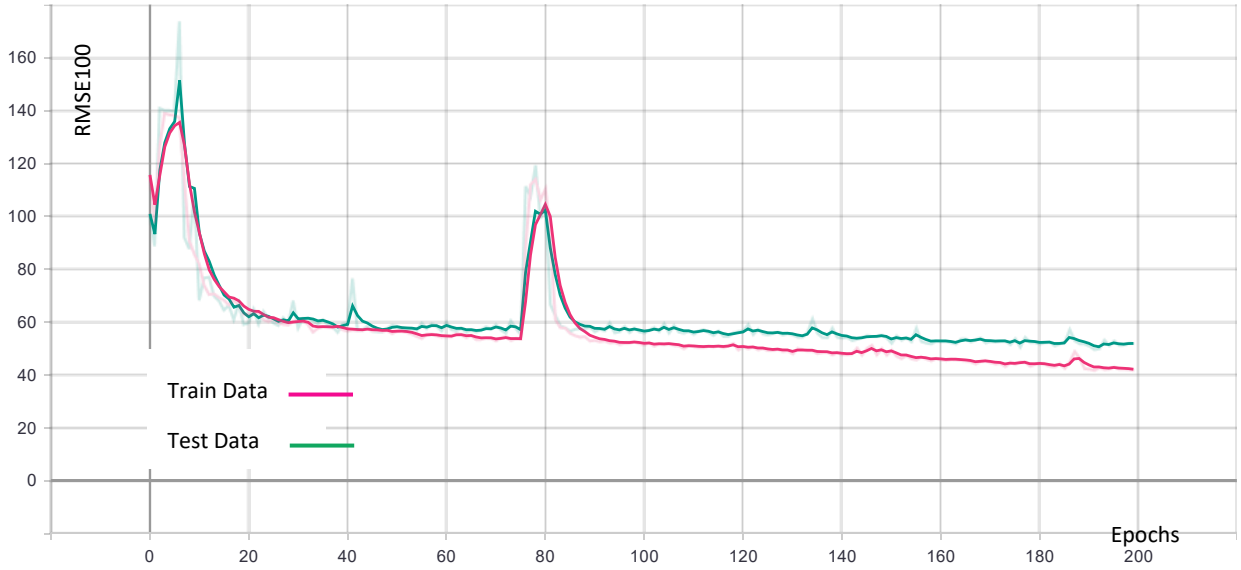


Figura 47: Función de Error RNN (RMSE100)

Se alcanza con el RNN un error sobre los datos de prueba de **51%** es decir una precisión de **49%**. Es mejor que el ANN, pero muy lejano del desempeño del CNN. Además, el entrenamiento es mucho más rápido con el CNN.

Se puede ver que en este caso el entrenamiento no es muy uniforme. En efecto, el error sube y baja, mientras que, para el CNN y el ANN, la tendencia global era el decrecimiento. Eso es debido a la complejidad de la red y que a veces para sacar de mínimos locales, el error tiene que subir.

Una metáfora de este fenómeno puede ser que a veces para bajar en un valle hay que subir una colina.

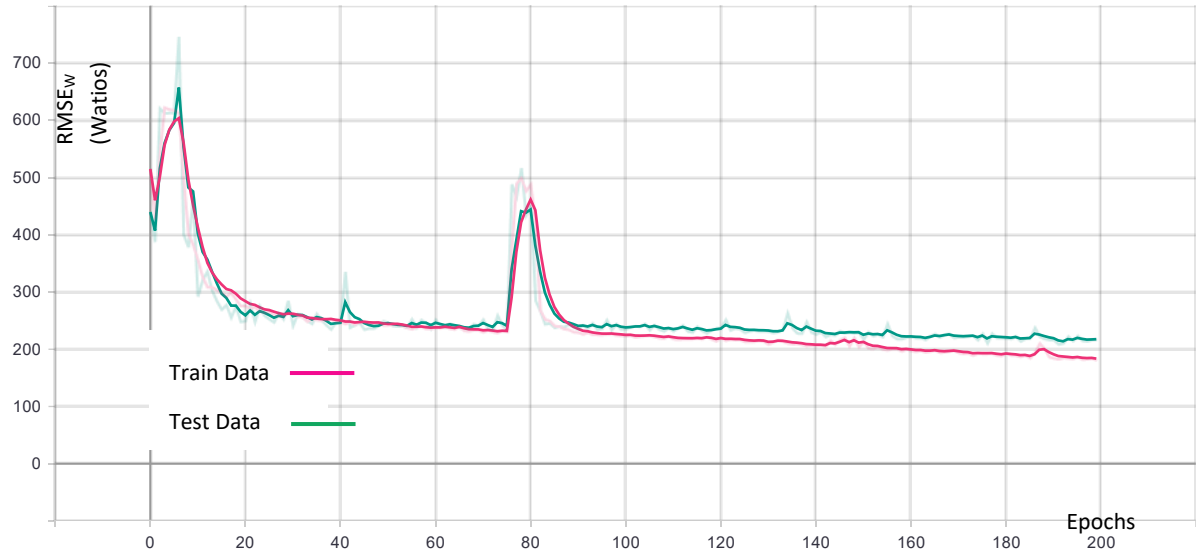
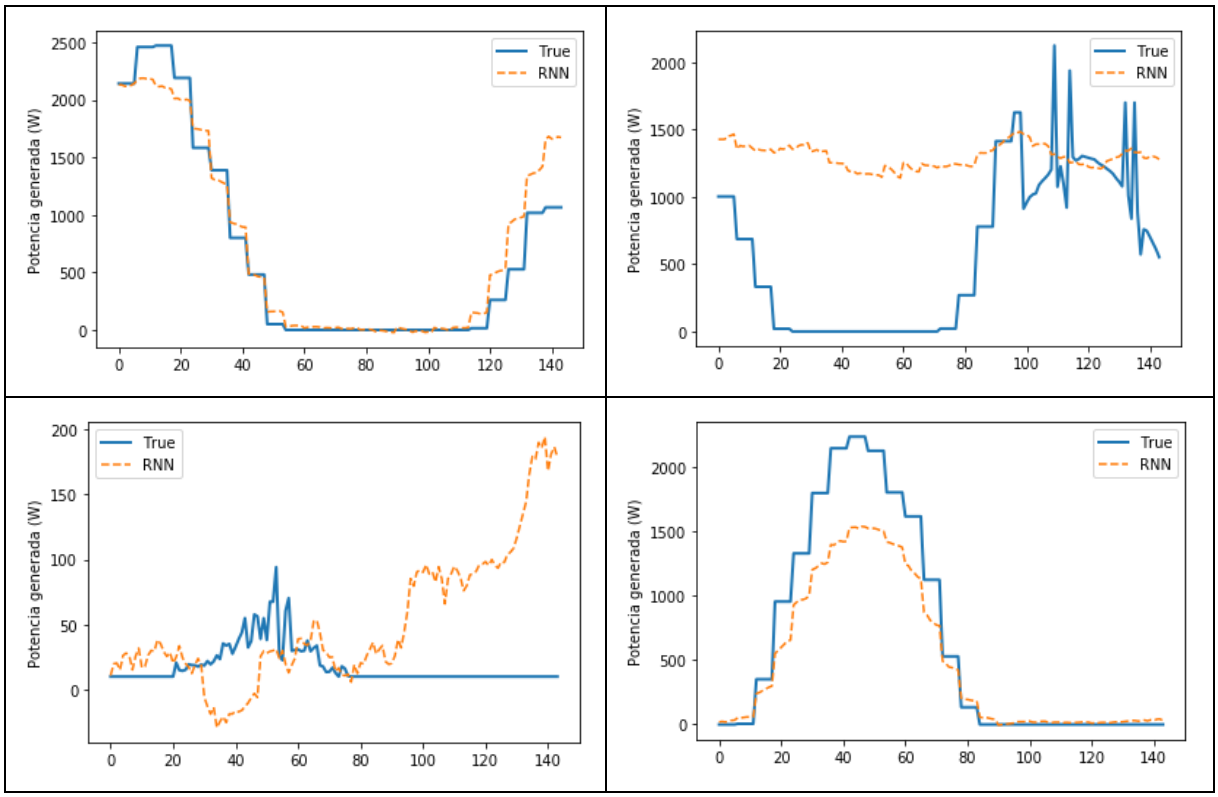


Figura 48: RMSEW RNN

Se puede ver la misma tendencia con el indicador $RMSE_W$. El error de prueba baja hasta **218W.**

Ahora, como antes se va a calcular la predicción sobre algunos datos de prueba con el RNN para ver el comportamiento de la red con más detalles.



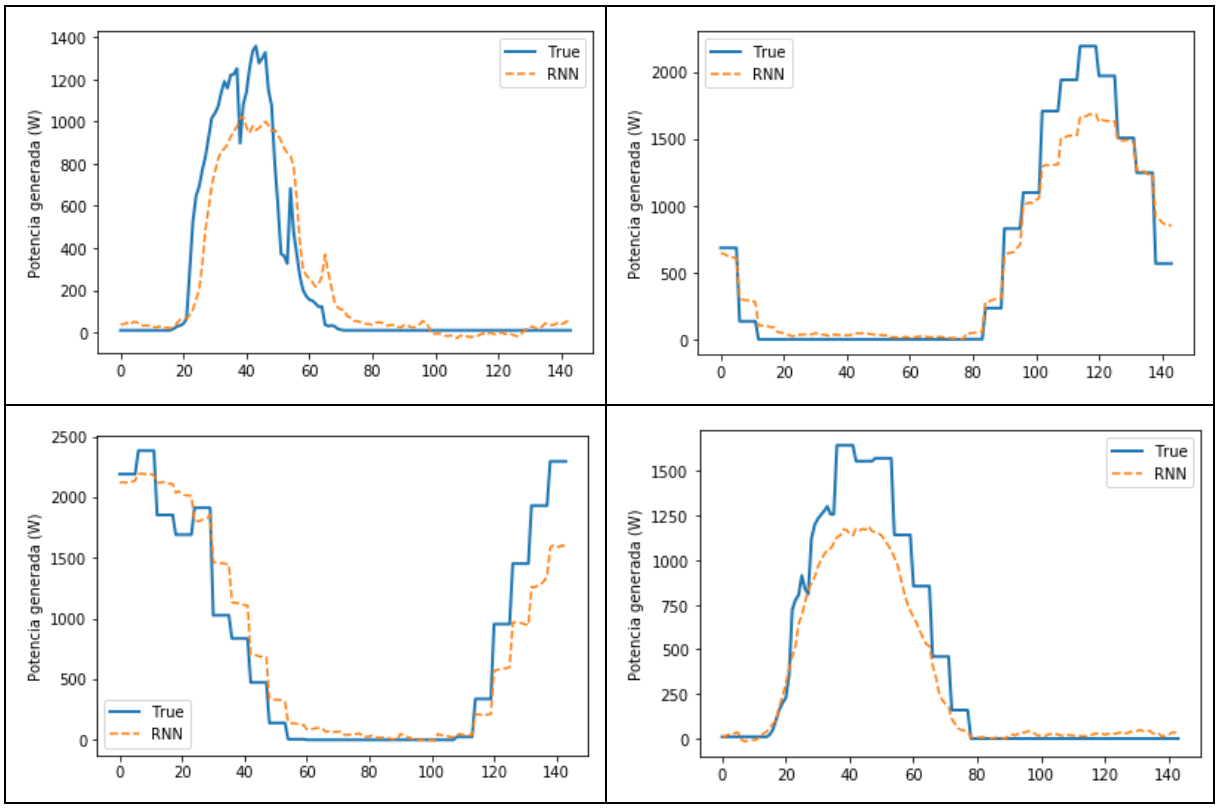


Figura 49: Ejemplos de predicción con el RNN. Eje X en datos con un dato cada 10 minutos

Se puede ver que la predicción está bastante cerca de la realidad en la mayoridad de los ejemplos, aunque en algunos de ellos como el segundo y el tercero, la predicción es muy mala. Son datos raros y la red no puede manejarlos. Pero lo más interesante es ver que los errores que hace la red son diferentes de los errores del ANN.

El RNN tiene un desempeño similar al ANN, pero con una complejidad muy grande y un tiempo de computación muy largo. Por eso no parece interesante para predecir la producción solar comparando con el CNN que tiene una muy buena eficiencia.

5.3.4. Comparación de resultados

En esta parte se hace un resumen de los resultados obtenidos para cada tipo de red.

Red	ANN	CNN	RNN
Datos de entrada	Producción, temperatura, irradiancia D-1 yD-2	Producción, temperatura, irradiancia D-1 y D-2	Producción D-1 yD-2
Forma de los datos	Vector: 864	Imagen: 3 X 8 X 36	Vector: 288
Número de parámetros	1,261,414	607,120	1,562,864

Tiempo entrenamiento para un *Epoch*	2s	3s	12s (con el GPU)
RMSE100 Train	44.6%	18.4%	42.2%
RMSE100 Test	57.7%	28.7%	51.9%
$RMSE_W$ Train	192.4W	80.36W	183.7W
$RMSE_W$ Test	242.5W	119.7W	217.6W

Las curvas más interesantes para comparar los entrenamientos son las curvas de error RMSE100. Se traza solamente las curvas de prueba para tener una mejor visibilidad.

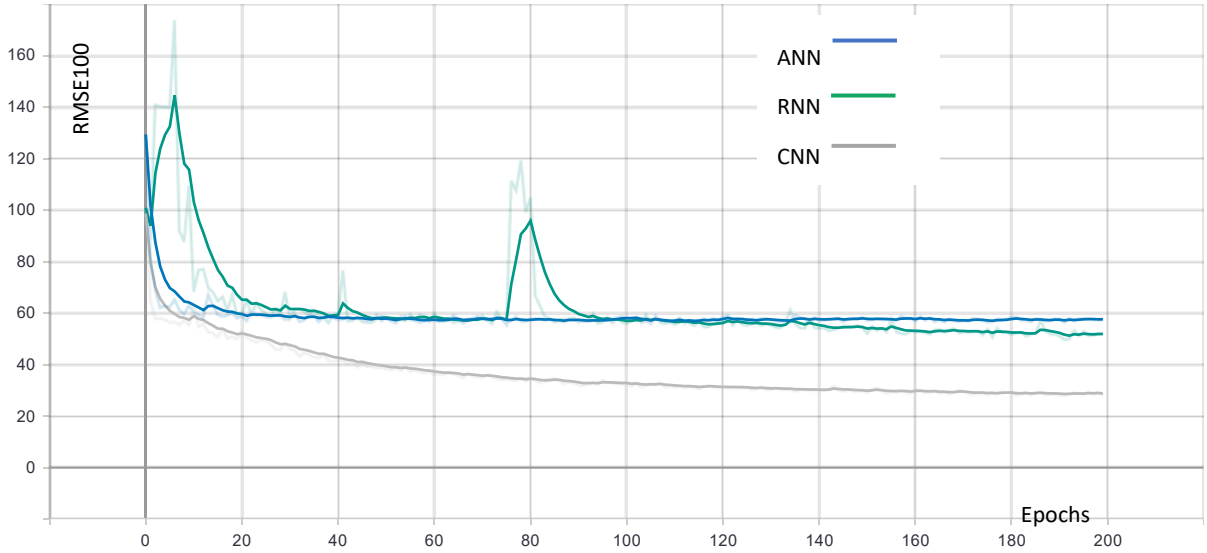
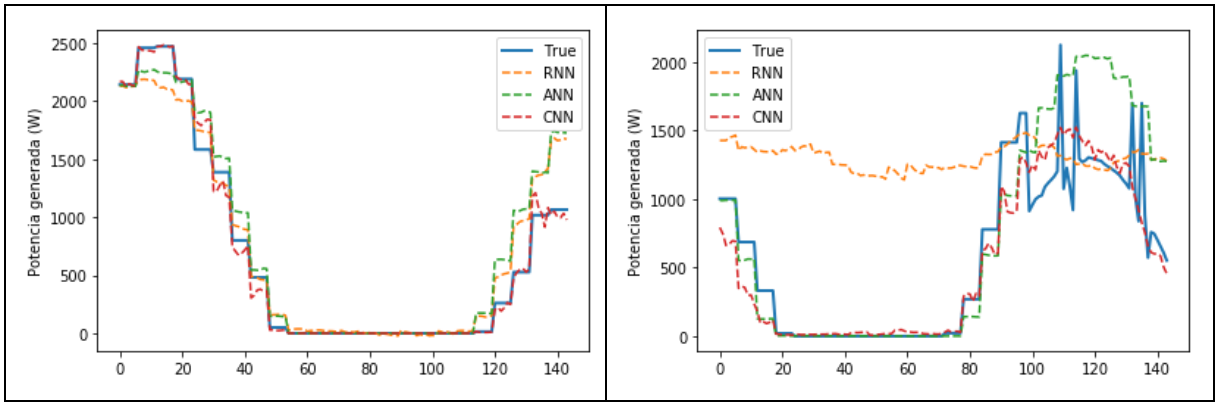


Figura 50:Comparación de las curvas de aprendizaje

Se puede ver que después de solo 10 *epochs* el CNN es más eficaz que los demás y sigue siendo el mejor después del entrenamiento completo.

Ahora se va a comparar las predicciones en algunos ejemplos.



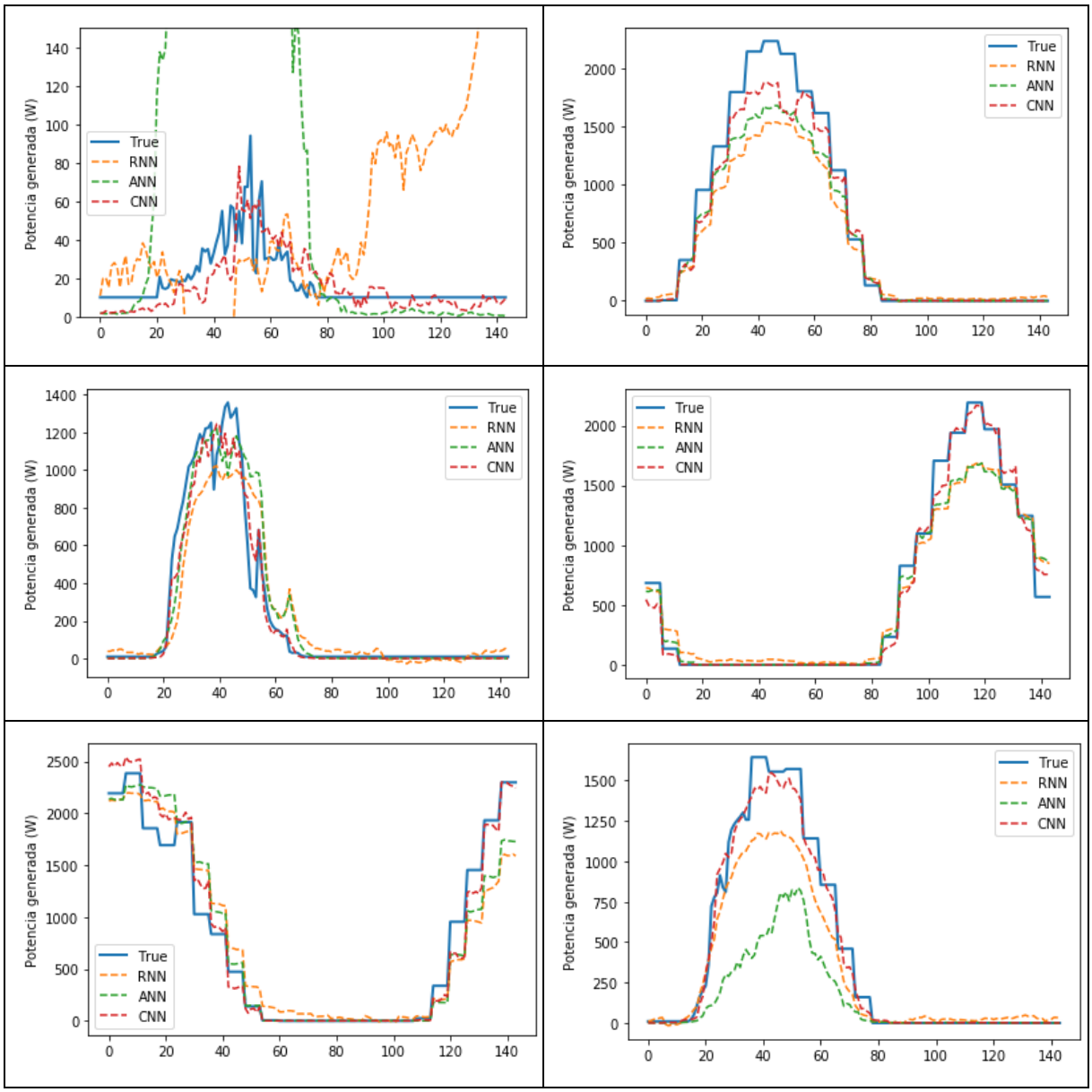


Figura 51: Comparación de predicción sobre ejemplos Test. Eje X en datos con un dato cada 10 minutos

Una vez más, se puede ver que el CNN es mucho más potente que los otros algoritmos. Por eso se elegirá este algoritmo para predecir la producción de energía por los paneles solares de la red *LabDER* con una precisión de **71%**.

6. Predicción de la producción eólica

El objetivo de esta parte es de predecir la energía producida por el sistema eólico del *LabDER*. Es decir, conocer con antelación la producción de la turbina utilizando los datos disponibles.

En la parte de presentación de los datos, se eligió utilizar la producción de los días anteriores y la velocidad de viento para predecir la producción eólica del día siguiente. La mayor diferencia con los datos solares es que los datos de viento son más aleatorios y los cambios son mucho más rápidos.

6.1. Estudio de los datos

Primeramente, se intentó predecir como para la energía solar, es decir un día adelante. Por eso se traza la matriz de correlación de los datos con un desplazamiento de 24h. Es decir, como hay 6 datos por horas: 6 X 24= 144 datos.

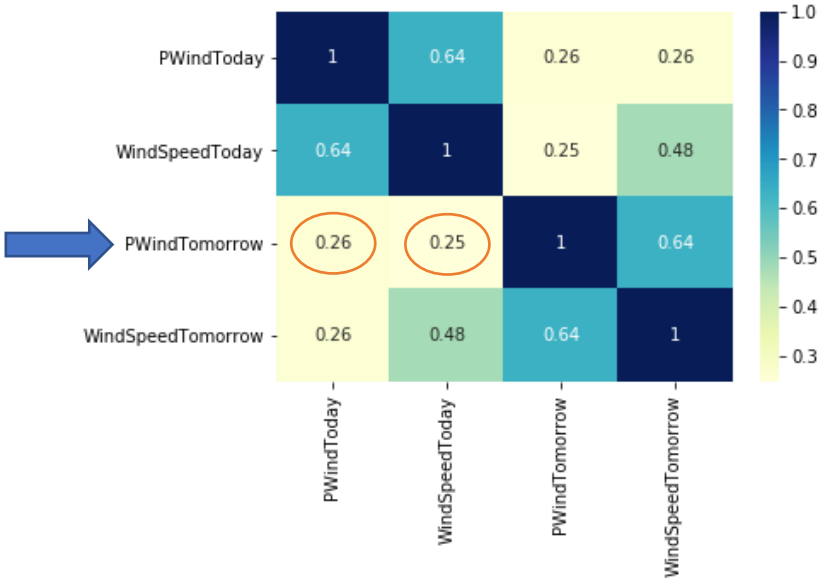


Figura 52: Matriz de correlación de producción eólica 24h

Se puede ver en la Figura 52 que la producción del día siguiente no está relacionada con la producción anterior o la velocidad del viento. En efecto 0.26 y 0.25 no es mucho. Por eso se abandonó la idea de predecir la producción un día adelante. En efecto el viento puede ser muy aleatorio y cambiar de manera muy rápida. Se intenta predecir la producción dos horas adelante, se traza una matriz de correlación.

Figura 53: Matriz de correlación de producción eólica 2h

Se puede ver en la Figura 53 que la producción dos horas en el futuro está bien relacionada a la producción y la velocidad del viento actual, respectivamente, 0.61 y 0.45.

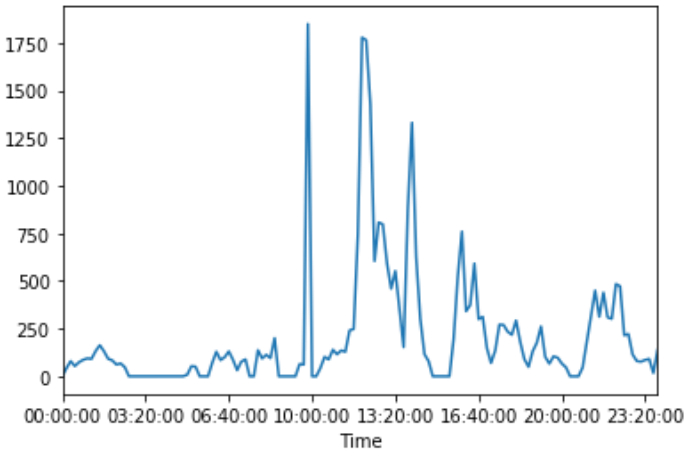


Figura 54: Ejemplo de un día de producción eólica

Por eso tener un dato cada 10 minutos no parece demasiado. Se necesitan más datos. Sin embargo, la parte de organización de los datos y de cálculo de la media cada diez minutos fue muy larga, y se automatizó este proceso.

6.2. Tratamiento de los datos

La herramienta *Pandas* de *Python* es muy potente. Además del estudio de los datos, permite reorganizarlos. Se utilizó esta herramienta para reorganizar los datos iniciales y escribir un script para automatizar las siguientes tareas:

1. Extraer los datos de los diferentes archivos CSV (sacados del *LabDER*).
2. Combinar esos archivos (1 por semana).
3. Extraer los datos meteorológicos.
4. Fusionar los datos meteorológicos y los datos LabDER según la fecha y la hora.
5. Utilizar los datos meteorológicos para completar los huecos de los datos *LabDER*.
6. Elegir la frecuencia de datos necesitada y preparar las muestras.
7. Generar un nuevo archivo CSV listo para trabajar.

Se hizo este script con la posibilidad de elegir la frecuencia deseada y cambiarla de manera muy fácil. Como el código es bastante largo no se describirá en detalle.

Se eligió una frecuencia de muestra de 1 dato por minuto. Es un término medio ente la precisión de los datos y el número de datos sin tener un tiempo de computación monumental.

Después se construyen los datos de la misma manera que para los datos solares. Es decir, se utiliza una ventana móvil para crear:

- los *inputs*: últimas 10 horas de producción
- los *outputs*: las próximas 2 horas de producción

Se seleccionan datos separados de 2 horas para que no se superpongan demasiado. Tener datos cada minuto plantea un problema. En efecto, el *LabDER* mide datos cada segundo, pero los datos meteorológicos usados son datos encontrados online que solamente están medidos cada media hora. Aunque Pandas permite la interpolación de los datos de varias maneras (linear, continua, polinomial),

se pierde la parte aleatoria en caso de hueco grande. En efecto, si falta uno o dos valores no pasa nada, pero si durante varias horas o varios días no hay datos del *LabDER*, los perfiles de viento no serán representativos.

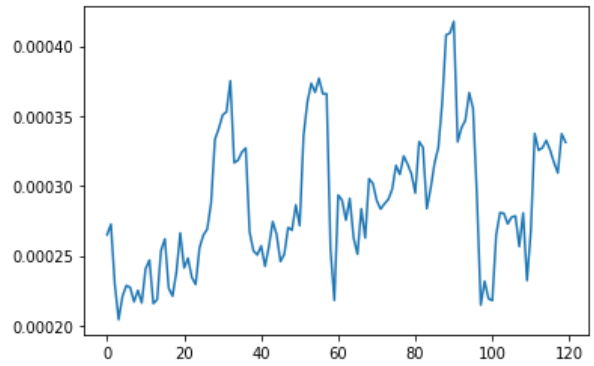


Figura 55: Datos de producción pertinente

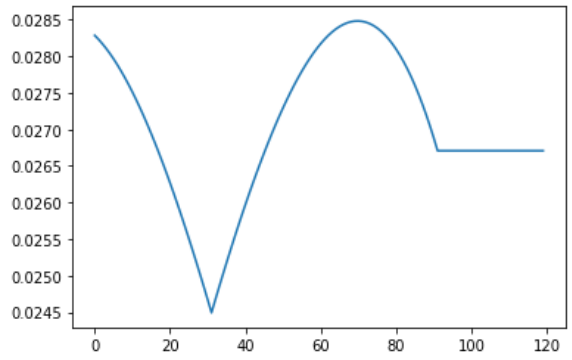


Figura 56: Datos de producción no pertinentes

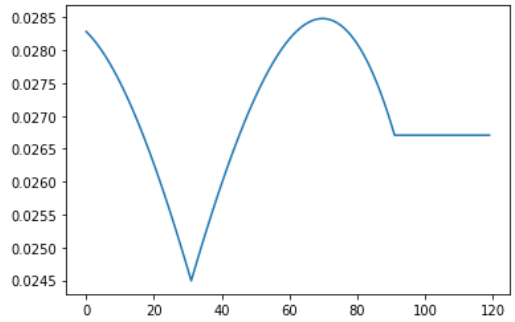
Para los datos solares, no era tan preocupantes porque había datos cada 10 minutos mientras que aquí se nota mucho más la diferencia.

Para que los algoritmos funcionen, hay que quitar los datos no pertinentes. Como hay un montón de datos, eso no se puede hacer de manera manual, hay que encontrar un indicador que caracteriza la suavidad de la curva. La mayor diferencia entre ambas curvas es la dispersión de los valores. Por eso hay que estudiar la diferencia entre un valor y el siguiente. Y como los valores pueden ser muy grandes o pequeños, se divide por la media para normalizar los datos.

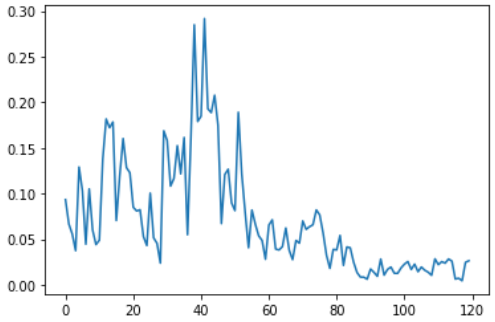
Así se define una función que traduce la suavidad de las curvas.

```
def smooth(X):
  d=0
  for k in range(len(X)-1):
    d+=abs(X[k+1]-X[k])
  return np.mean(d)/np.mean(X)
```

Cuanto más cerca de 0 es la función, más suave es la curva. Después hay que encontrar un criterio para definir que curvas son pertinentes o no. Por eso se trazan las curvas para tener ejemplos.



Smooth= 0.3533117624765939



Smooth= 44.28112052488886

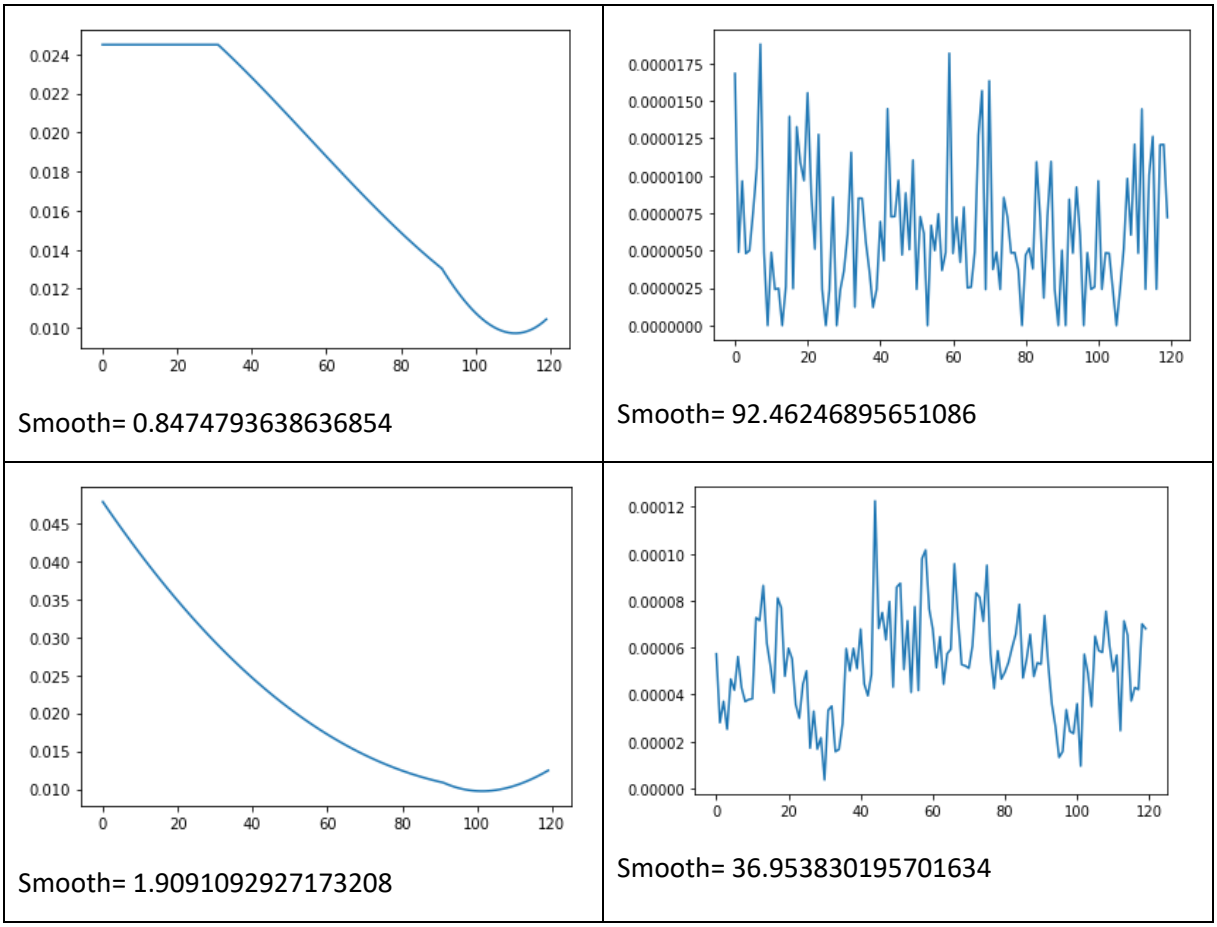


Figura 57: Ejemplos de curvas e indicador de suavidad

Visto algunos ejemplos, se elige un umbral empírico de 15. Es decir que se eliminan todos los datos que no cumplen un indicador superior a 15. De esta manera, se quitan los datos no pertinentes para trabajar solamente con datos reales. El algoritmo deberá predecir la producción utilizando datos reales en el futuro, así es mejor entrenarlo con datos del mismo tipo. Los datos se normalizan para tener valores entre 0 y 1 de la misma manera que en la parte anterior. Y se separan entre datos de entrenamiento y datos de prueba.

Como se puede ver en las últimas ilustraciones, los datos de producción son muy versátiles. Parece muy difícil extraer informaciones directamente de ellas. Además, se intentó aplicar directamente redes neuronales a estos datos y no tuvo gran éxito.

Entonces, se necesita preprocesar los datos. El viento contrariamente al sol tiene una parte aleatoria muy grande. Es muy difícil predecir la próxima borrasca. Se debe a la teoría del caos y al efecto de la mariposa. Hay una infinidad de parámetros que pueden entrar en cuenta para predecir los movimientos del aire. Para el procesamiento de los datos, el objetivo es quitar la parte aleatoria del viento conservando la tendencia global. Por eso, hay que descomponer las señales. Se utilizará el método SSA *Singular Spectrum Analysis.* Las matemáticas detrás este método son bastante complicadas, pero la idea es de descomponer la señal en una suma de componentes.

Para usar la descomposición SSA, se instala una biblioteca *Python* llamada *Pyts.decomposition*. Se necesita definir la señal a descomponer y la ventana de descomposición, es decir el número de componentes deseados.

El código para usar la biblioteca es bastante sencillo.

```
# SSA transformation
window_size = 3
ssa=SingularSpectrumAnalysis(window_size)
Xtrain_ssa = ssa.fit_transform(X_train)
```

Se elige una ventana de 3 componentes en el caso de los datos de viento. Lo que es suficiente para extraer las informaciones. Se puede calcular la descomposición de las señales y trazar esa descomposición comparándola con la señal original.

```
k=1
plt.plot(X_train[k],label='True')
for i in range(len(Xtrain_ssa[k])):
    plt.plot(Xtrain_ssa[k][i],label=str(i))
plt.legend()
```

k corresponde al dato que se quiere trazar.

La señal original tiene la etiqueta *True* y los componentes el número de componente entre 0 y 2.

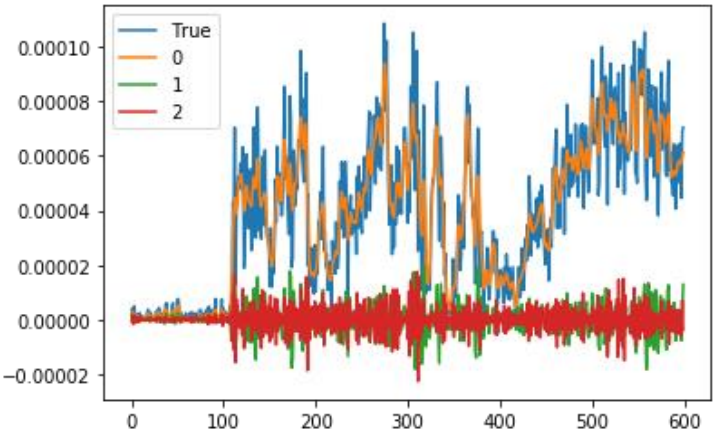


Figura 58: Ejemplo de descomposición SSA sobre la producción de energía eólica

El primer componente, el componente 0, sigue la tendencia global de la señal mientras que las dos otras corresponden a las variaciones aleatoria de los datos. La suma de las señales corresponde a la señal original.

6.3. Métodos de *Machine Learning* (teoría)

En esta parte, se utilizan algoritmos de *Machine Learning* para predecir la producción eólica 2 horas adelante. Fue mucho más difícil de predecir comparando con los datos solares debido a la parte aleatoria del viento y a la falta de datos.

Se conservan las mismas funciones de error en esta parte que la anterior, es decir las funciones RMSE100 y $RMSE_W$. De la misma manera, se comparará la eficiencia de tres tipos de algoritmos: ANN,

RNN, CNN. Pero debido a la complejidad del problema, se utilizó diferentes métodos hasta lograr una eficiencia aceptable.

6.3.1. Método 1: Predecir la curva de producción

La primera idea fue de predecir la curva de producción de la misma manera que para los datos solares. De la misma manera que para el sol, se utilizó tres redes diferentes. Para predecir las dos siguientes horas de producción, se utilizó las últimas 10 horas de producción eólica. Así los datos de salida son vectores de 60 X 2= 120 valores como hay un dato por minuto. Para los datos de entrada se utilizan las 10 horas anteriores, pero se descomponen estos datos con una ventana de 3 componentes (las SSA). Así la longitud de datos de entrada es 60 X 10 X 3= 1 800.

En primer lugar, se intentó entrenar las tres redes: ANN, CNN y RNN, cambiando todos los parámetros, el número de capas, la longitud de las capas, las funciones de error, los *optimizers,* el tamaño de los *batch*. Fue imposible lograr un error inferior a 100% de RMSE100. Es decir que la curva predicha estaba más leja de la curva real que si fuera nula. Se debe al hecho de que la red no logre generar la parte aleatoria de las curvas de salida. Para paliar este problema, se decidió modificar los datos de salida. Como es difícil predecir una curva que varía mucho, se va a descomponer las curvas de salida con el método SSA e intentar predecir el primer componente que corresponde a la tendencia global de la producción.

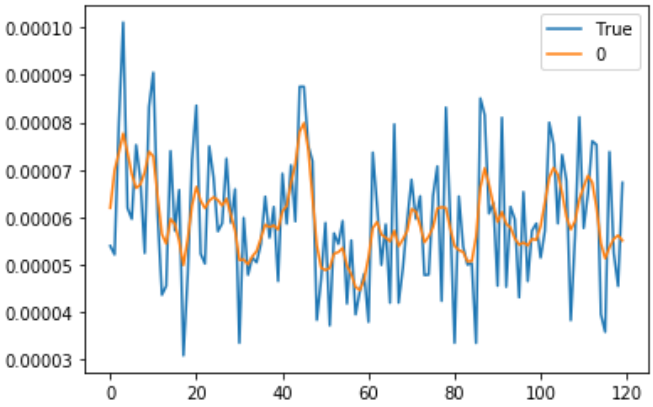


Figura 59: Comparación entre la curva original y la componente SSA principal

Como se puede ver en la Figura 59, la curva de la componente 0 de la descomposición SSA es mucho menos versátil. Así será más fácil de predecir esta curva para las diferentes redes. Pero es una descomposición y entonces se pierde información asumiendo que la curva de producción sea la curva naranja. Hay que cuantificar esta asunción. Para hacer eso se puede usar la función de error RMSE100 entre la curva original y la componente 0 de la descomposición SSA. Para el ejemplo precedente se obtiene un error de **17.8%** es decir que la curva naranja describe la curva original con una precisión de **82.2%**.

Se hace el promedio de todas esas asunciones para obtener un valor global.

```
err=[]
for k in range(len(y_train)):
    err.append(rmse100(y_train[k],ytrain_ssa[k][0]))
for k in range(len(y_test)):
```

```
    err.append(rmse100(y_test[k],ytest_ssa[k][0]))

np.mean(err)
```

Se obtiene un error medio entre la componente SSA principal y la curva real de **48%**, o sea una precisión de **52%**. Es muy poco. Es decir que, si las redes cumplen predecir las curvas aproximadas con una precisión de 100%, lo que nunca será posible, la predicción real será de 52%. Para obtener la precisión global habrá que multiplicar la precisión de la red por la precisión de la aproximación. Significa que no se podrá lograr más de 52% de precisión.

Poniéndose un poco en retraso para pensar permite notar que la diferencia entre las curvas cada minuto no es tan importante sino la energía generada. Es decir, quizás a un instante subvalora la realidad, pero al instante siguiente se sobrestima, y los errores se compensan. Por eso, se va a calcular la energía generada global en 2 horas por las dos curvas y compararlas con la siguiente función. Primero hay que tener una función que calcula la energía generada a partir de una curva. Matemáticamente la energía corresponde a la integral de la potencia.

$$E = \int_0^T P(t)dt$$

Ecuación 17: Expresión matemática de la energía

Donde [0,T] es el periodo de observación y P(t) la potencia instantánea. En el caso este, T=2 horas o como se trata de minutos T=2 X 60=120.

Como en informática no se puede tratar con datos continuos, la integral se transforma en una suma y la energía se obtiene con la siguiente formula.

$$E = \sum_0^T P(k) * T_e = \sum_0^{120} P(k) * \frac{1}{60}$$

Ecuación 18: Formula de la energía en caso discreto

Donde T_e es el periodo de muestro: 1 muestro cada minuto, T_e=1/60 horas.

```
def to_energy(y):
    return(sum(y)/60)
```

Después para comparar las energías de la curva real y de la primera componente de la descomposición SSA se calcula el error relativo.

$$Error\ relativa\ de\ Energia = \frac{|E_{True} - E_{SSA0}|}{E_{True}} * 100$$

Ecuación 19: Error relativa de Energía

```
def errEnergia(y_true, y_SSA):
    abs(to_energy(y_true)-to_energy(y_SSA))
```

De la misma manera que antes, se hace la media de este error sobre todas las salidas. Sale un error relativo de 0.7% lo que es muy poco. Significa una precisión de **99.3%**. Aunque las curvas tienen un error RMSE100 bastante grande, las energías producidas se pueden considerar iguales.

Para seguir se intentará predecir la componente principal de la descomposición SSA en vez de la curva original. Para hacer eso se utiliza tres tipos de red diferentes, como para la predicción solar, un ANN, un CNN, un RNN.

6.3.1.1. Con un ANN

No se va a entrar en detalle sobre cómo se elijo la red y las pruebas realizadas antes de obtener resultados aceptables. La red elegida es bastante sencilla, se compone de solamente 3 capas, la de entrada, una de 1 500 neuronas y una de salida con 120 neuronas. Solo con estas capas, la red ya tiene tendencia a sobreentrenar por eso se puso un *Dropout* de 0.8.

```
modelANN = tf.keras.Sequential()
modelANN.add(tf.keras.layers.Dense(1500,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dropout(0.8))
modelANN.add(tf.keras.layers.Dense(120,activation='sigmoid'))

modelANN.compile(optimizer='adam', loss=mseW , metrics=[mse100])

# fit model
modelANN.fit(X_trainSSA, y_trainSSA, epochs=200, verbose=2,
batch_size=50,validation_data=(X_testSSA,y_testSSA))
```

Se eligió un tamaño de *batch* de 50 elementos y 200 *epochs*. En efecto, los datos son muy diferentes, así un tamaño de *batch* grande es necesario para lograr la convergencia. Pero hay que adaptar este tamaño para obtener un término medio entre rapidez y precisión.

Se obtiene para los datos de prueba un error **$RMSE_W$=87.7W** y **RMSE100=81.6%**. Es bastante alto, pero mucho mejor que sin la descomposición SSA.

6.3.1.2. Con un RNN

La red elegida se compone de dos capas **LSTM** de 50 bloques y una capa de 120 neuronas clásicas. En entrada se utilizan directamente los datos originales y en salida la primera componente de la descomposición.

```
n_features = 1
n_steps=60*10 #prod

X_trainRNN = X_train.reshape((X_train.shape[0], n_steps, n_features))
X_testRNN = X_test.reshape((X_test.shape[0], n_steps, n_features))

# define model
modelRNN = tf.keras.Sequential()
modelRNN.add(tf.keras.layers.LSTM(50,input_shape=(n_steps,n_features),retur
n_sequences=True))
modelRNN.add(tf.keras.layers.LSTM(50,input_shape=(n_steps,n_features)))
modelRNN.add(tf.keras.layers.Dense(120,activation='sigmoid'))

modelRNN.summary()
modelRNN.compile(optimizer='rmsprop', loss=mseW , metrics=[mse100])

modelRNN.fit(X_trainRNN,y_trainSSA,epochs=200,verbose=2,batch_size=50,valid
ation_data=(X_testRNN,y_testSSA))
```

Primero, hay que reorganizar los datos de manera que sean compatibles con las capas **LSTM**. Se utiliza el *optimizer RMSprop* que es mejor sobre las redes recurrentes. Se utiliza **$RMSE_W$** como función de error porque parece ser más efectiva, quizás porque los datos puedan ser muy grandes o pequeños. Así la red se focaliza sobre los grandes errores de manera absoluta en vez de las pequeñas variaciones.

Con esa red se obtiene errores de **$RMSE_W$=81.9W** y **RMSE100=75.9%**. Es alto, pero menos que con el **ANN.**

6.3.1.3. Con un CNN

La red elegida se compone de dos capas de convolución con respectivamente 64 y 32 filtros. En seguida, una capa de *Dropout* de 0.8 y finalmente una capa de 120 neuronas para la salida.

Primero, hay que reorganizar los datos en imágenes compatibles con la red. Aquí se utiliza la descomposición **SSA** de los datos de entrada, cuyas componentes como canales para las imágenes. Así cada imagen se compone de:

- **3 canales** para cada componente de la descomposición SSA
- Cada canal se compone de 20 X 30 valores, lo que corresponde a 30 minutos de datos para cada línea.

Así el **1 800** vector inicial es reorganizado en **3 X 20 X 30.**

```
## Shape X: 3*20*30
X_trainCNN=X_trainSSA.reshape(len(X_trainSSA),3,20,30)
X_testCNN=X_testSSA.reshape(len(X_testSSA),3,20,30)
#keep y flatten

# define model
modelCNN = tf.keras.Sequential()
modelCNN.add(tf.keras.layers.Conv2D(64,(2,2),input_shape=(3,20,30),activati
on='relu'))
modelCNN.add(tf.keras.layers.Conv2D(32,(2,2),input_shape=(3,20,30),activati
on='relu'))
modelCNN.add(tf.keras.layers.Flatten())

modelCNN.add(tf.keras.layers.Dropout(0.8))
modelCNN.add(tf.keras.layers.Dense(120,activation="sigmoid"))
modelCNN.summary()
modelCNN.compile(optimizer='adam', loss=mseW,metrics=[mse100])


modelCNN.fit(X_trainCNN,y_trainSSA,epochs=200,verbose=2,batch_size=30,valid
ation_data=(X_testCNN,y_testSSA))
```

La función de activación de las capas **LSTM** es *ReLu* y el tamaño de los filtros **2 X 2.** El tamaño de los *Batch* se reduce a 30 porque la convergencia no estaba muy precisa con 50.

Esta red logra errores para los datos de prueba de **$RMSE_W$=86.0W** y **RMSE100=80.0%** lo que es menos que con el **RNN,** pero mejor que el **ANN.**

6.3.1.4. Resultados

Los mejores resultados vienen del **RNN** con una precisión de **24.1%**. No es muy alto, pero hay que observar con ejemplos para tener una idea. En efecto, se vi anteriormente que la primera componente de una curva puede tener un error RMSE100 muy grande con la curva real, aunque están muy cercas.

Para tener una mejor idea del trabajo de las redes, se observará algunas predicciones con las verdaderas curvas.

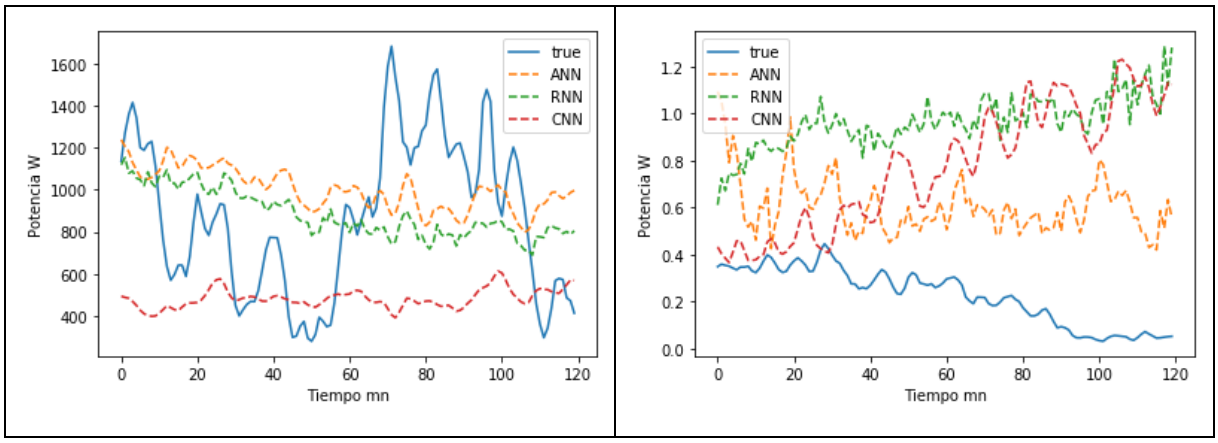


Figura 60: Ejemplo de predicción de curvas de producción eólica

Mirando solamente a la Figura 60, no se puede distinguir una red mejor que las demás. Las tres redes predicen mal la curva de producción. Aunque la magnitud está bien reconocida, las fluctuaciones y las variaciones más grandes no son muy bien predichas por las redes. Quizás se podría obtener mejores resultados, pero se necesitaría muchos más datos, sobre 10 años, por ejemplo, para que la red pudiera aprender todos los casos posibles. Pero el trabajo de un ingeniero es de hacer con lo que es disponible, así se va a intentar otros métodos.

6.3.2. Método 2 predecir energía producida

Como el método 1 no fue muy concluyente, otra idea fue de predecir la cantidad de energía producida durante las dos horas en vez de la curva de producción.

Se utiliza la función *to_energy* anterior para calcular la producción de cada ventana de 2 horas. Con este método, la salida de las redes no es un vector, sino un valor que corresponde a la energía producida por la turbina durante las siguientes dos horas.

Ahora es un problema de regresión mono dimensional. Las funciones de error RMSE100 y $RMSE_W$ siguen funcionando, solo es que el vector de salida tiene una sola componente. Así la función RMSE100 calcula el error de energía producida relativa en porcentaje. La función $RMSE_W$ calcula el error de energía producida en Watio hora (Wh). Para formar los datos, se utiliza el mismo método que para la parte anterior, solamente que al vector de salida se le aplica la función *to_energy.*

```
X,y=split_sequence(normalize(data['PWind']),10*60,2*60) #predict 2h with 10h
Xspeed,yspeed=split_sequence(normalize(data['WindSpeed']),10*60,2*60)

X2=[]
y2=[]


for k in range(len(X)):
    if k%(60*2)==0:   #Condición para desplazamiento de la ventana
        X2.append(X[k])
        y2.append(to_energy(y[k]))


X=X2
y=y2
```

Después se hace lo habitual, es decir filtrar según si los datos están demasiados aleatorios (quitar los falsos datos con la función *smooth*). Luego, se mezclan los datos y se separan en conjunto de entrenamiento y conjunto de prueba. En este caso, se utilizará una vez más los tres tipos de redes **ANN, RNN y CNN**.

6.3.2.1. Con un ANN

Para el **ANN** se utiliza la configuración siguiente:

```
# define model
modelANN = tf.keras.Sequential()
modelANN.add(tf.keras.layers.Dense(1000,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dropout(0.5))
modelANN.add(tf.keras.layers.Dense(200,activation='sigmoid'))
modelANN.add(tf.keras.layers.Dropout(0.5))
modelANN.add(tf.keras.layers.Dense(1,activation='sigmoid'))

modelANN.compile(optimizer='adam', loss=mse100,metrics=[mseW] )

# fit model
modelANN.fit(X_train,y_train,epochs=200,verbose=2,batch_size=10,validation_
data=(X_test,y_test))
```

El modelo se compone de 3 capas de neuronas, más la capa de entrada, con una función de activación sigmoide. Estas capas están separadas por capas de *Dropout* para limitar el *overfitting*. Se puede ver que los datos de entrada no son las descomposiciones SSA sino simplemente las señales normalizadas. Se eligió eso porque los resultados estaban mejor. Se debe probablemente al hecho de que la descomposición sirva para observar la tendencia temporal de la producción mientras que para la energía producida no hace falta.

Se obtiene al final del entrenamiento un error **RMSE100=52.8%** es decir una precisión de **47.2%** sobre los datos de prueba. Aunque no es una precisión fenomenal, es mucho mejor que intentando predecir la curva de producción.

6.3.2.2. Con un RNN

Para el **RNN** se utiliza le siguiente configuración:

```
n_features =1
n_steps=60*10 #prod

X_trainRNN = X_train.reshape((X_train.shape[0], n_steps, n_features))
X_testRNN = X_test.reshape((X_test.shape[0], n_steps, n_features))

# define model
modelRNN = tf.keras.Sequential()
modelRNN.add(tf.keras.layers.LSTM(150,input_shape=(n_steps,n_features),retu
rn_sequences=True))
modelRNN.add(tf.keras.layers.LSTM(150))
modelRNN.add(tf.keras.layers.Dropout(0.15))
modelRNN.add(tf.keras.layers.Dense(1,activation="sigmoid"))

modelRNN.summary()
```

```
modelRNN.compile(optimizer='rmsprop', loss=mseW, metrics=[mse100] )


modelRNN.fit(X_trainRNN,y_train,epochs=200,verbose=2,batch_size=30,validati
on_data=(X_testRNN,y_test))
```

El modelo se compone de dos capas de LSTM de 150 bloques. Luego, hay una capa de *Dropout* para limitar el sobre entrenamiento y finalmente una neurona clásica para la salida. Se utiliza el optimizar *RMSProp* porque el *Adam* no es muy eficaz para las redes recurrentes. Para esa red también se utiliza solamente las señales originales y no la descomposición SSA.

Con esos parámetros, la red logra un error RMSE100 de **45.8%** es decir una precisión de **54.2%** sobre los datos de prueba. Lo que es mejor que el **ANN.**

6.3.2.3. Con un CNN

Para el **CNN** se utiliza la configuración siguiente:

```
## Shape X: 1*20*30
X_trainCNN=X_train.reshape(len(X_train),1,20,30)
X_testCNN=X_test.reshape(len(X_test),1,20,30)

# define model
modelCNN = tf.keras.Sequential()
modelCNN.add(tf.keras.layers.Conv2D(64,(2,2),activation='relu',input_shape=
(1,20, 30),padding='same'))
modelCNN.add(tf.keras.layers.Conv2D(32,(2,2),activation='relu',padding='sam
e'))
modelCNN.add(tf.keras.layers.Flatten())

modelCNN.add(tf.keras.layers.Dropout(0.7))
modelCNN.add(tf.keras.layers.Dense(1,activation="sigmoid"))

modelCNN.summary()
modelCNN.compile(optimizer='adam', loss=mse100,metrics=[mseW])

# fit model
modelCNN.fit(X_trainCNN,y_train,epochs=200,verbose=2,batch_size=10,validati
on_data=(X_testCNN,y_test))
```

Los datos de entrada están reorganizados como imágenes de 20 X 30 en vez de un vector de 600 componentes. El valor de salida sigue siendo la energía producida durante dos horas. No se utiliza la descomposición SSA. La red se compone de dos capas de convolución justo antes de una capa de *Dropout* y por fin una neurona clásica para la salida. Para obtener una convergencia más precisa se reduce el tamaño de *batch* a 10 elementos.

Con esos parámetros, el **CNN** logra un error **RMSE100** de **48.2%,** es decir una precisión de **51.8%** sobre los datos de pruebas.

6.3.2.4. Resultados

La mejor precisión de predicción viene del **RNN** con **54.2%.** Es también mejor que el método 1 pero no es mucho. Se descargan los diferentes pesos de las redes para no tener que entrenar cada vez. Se va a

observar las predicciones hechas para entender más como se hizo el aprendizaje. Para hacer esto, se va a trazar un diagrama de barras comparando las diferentes predicciones con la producción real. También se añadirá el error relativo con la producción real de cada predicción. Permitirá comparar el error relativo del error absoluto.

```
def err(y_true, y_pred):
    return 100*abs(y_true-y_pred)/y_true
```

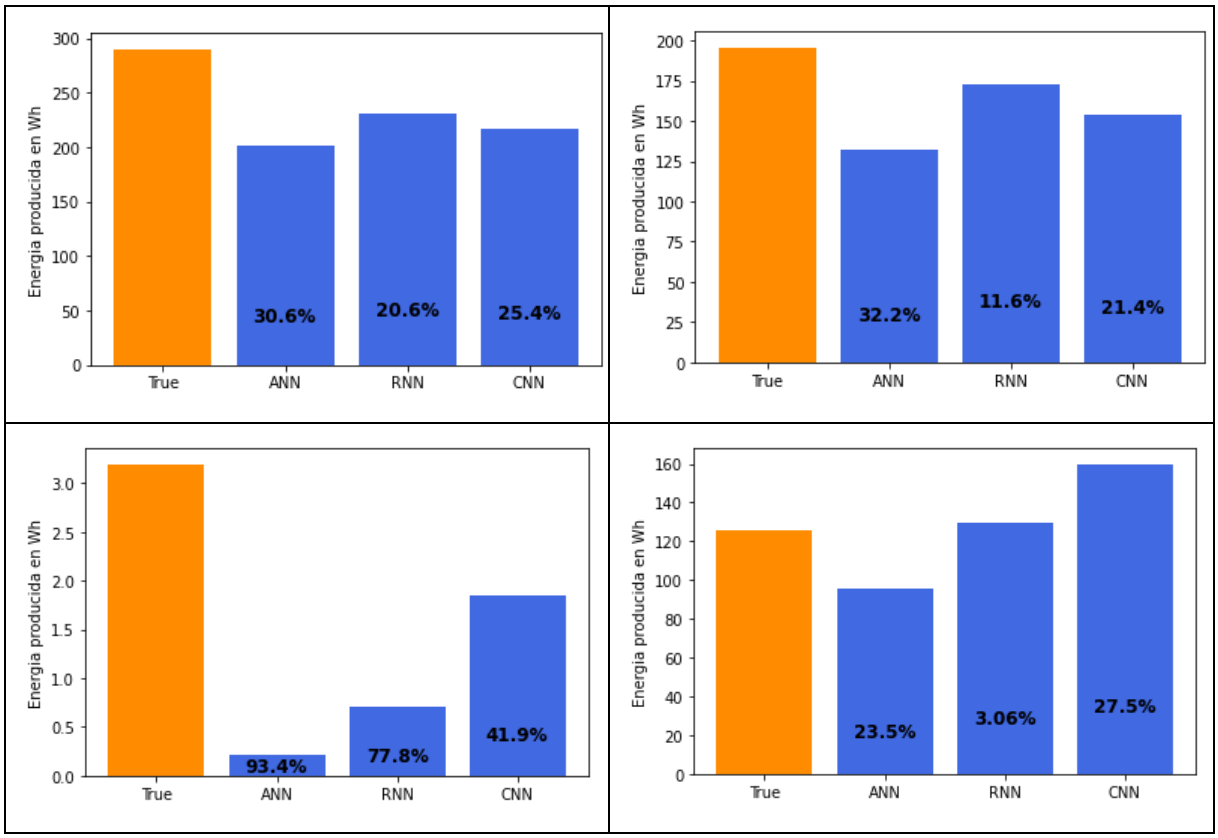
Aquí está el código de la función de error relativo y enseguida el código para trazar los diagramas.

```
 n=0
plt.bar(['True','ANN','RNN','CNN'],[y_test[n]*5594,predANN[n]*5594,predRNN[
n]*5594,predCNN[n]*5594],color=['darkorange','royalblue','royalblue','royal
blue'])

plt.text('ANN',predANN[n]*5594/5,str(err(y_test[n],predANN[n])[0])[:4]+'%',
horizontalalignment='center',fontsize=12,fontweight='bold')
plt.text('RNN',predRNN[n]*5594/5,str(err(y_test[n],predRNN[n])[0])[:4]+'%',
horizontalalignment='center',fontsize=12,fontweight='bold')
plt.text('CNN',predCNN[n]*5594/5,str(err(y_test[n],predCNN[n])[0])[:4]+'%',
horizontalalignment='center',fontsize=12,fontweight='bold')

plt.ylabel('Energia producida en Wh')
```

La *n* permite cambiar el ejemplo de los datos de prueba. La primera parte consiste a trazar el diagrama. La segunda parte permite añadir el error relativo calculado en el diagrama. Y por fin se pone la leyenda del axis y. Como para la predicción solar se va a trazar estos diagramas para algunos ejemplos.



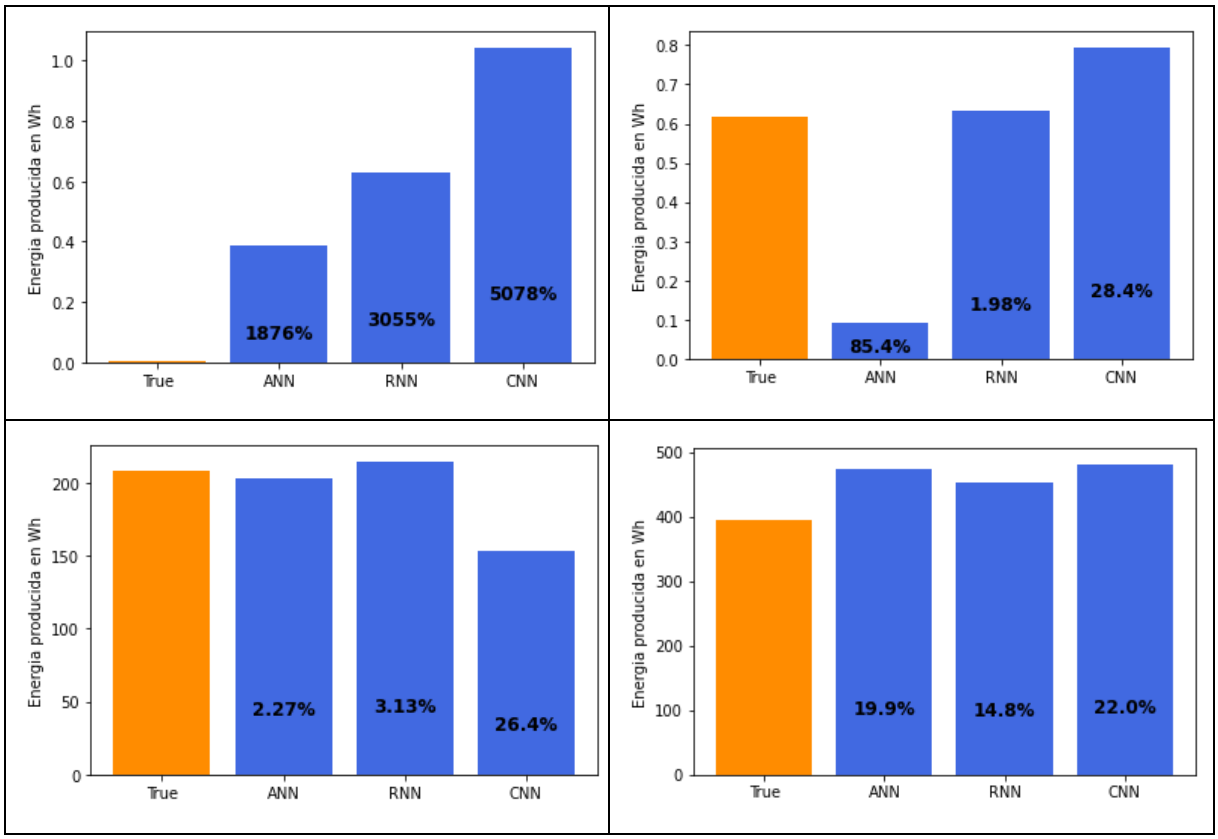


Figura 61: Ejemplos de predicción de energía

Se puede ver que algunas veces la predicción es bastante bien, hasta 3% de error, es decir 97% de precisión. Pero a veces las predicciones son muy malas, como en el ejemplo 5. Pero mirando más atentamente, se puede ver que los grandes errores de predicción ocurren cuando el valor real es muy pequeño, lo que es normal, porque cuando los valores son pequeños, una pequeña diferencia absoluta puede ser un gran porcentaje. Pero si se mira a estos casos, se puede ver que, aunque el error relativo sea muy grande, el error absoluto es del orden de algunos Wh. Lo que es nada, y son estos datos que hacen que el error global sea malo.

Además, cuando la turbina produce 3Wh, 5Wh o mismo 10Wh en 2 horas, es muy poco, es casi como un ruido de la señal. Se podría considerar que no se produce nada. En efecto si la red predice que se va a producir 3Wh en vez 5 o 6 no será muy grave. Además, sobre los grandes valores la predicción está bastante eficaz. Hay que hacer algo para estos periodos de casi no producción de energía. Primero hay que estudiar los datos para ver cómo se reparten los datos. Para hacer eso se va a trazar un histograma, es decir un diagrama de barra comparando el número de datos según la cantidad de energía producida.

Por eso se utiliza el siguiente código:

```
plt.figure(figsize=(20,4))
plt.hist(y_test*5594,bins=50)
plt.xticks(range(0,3700,100))
plt.ylabel('Numero de elementos')
plt.xlabel('Energia producida en Wh')
```

Con eso se obtiene la siguiente figura:

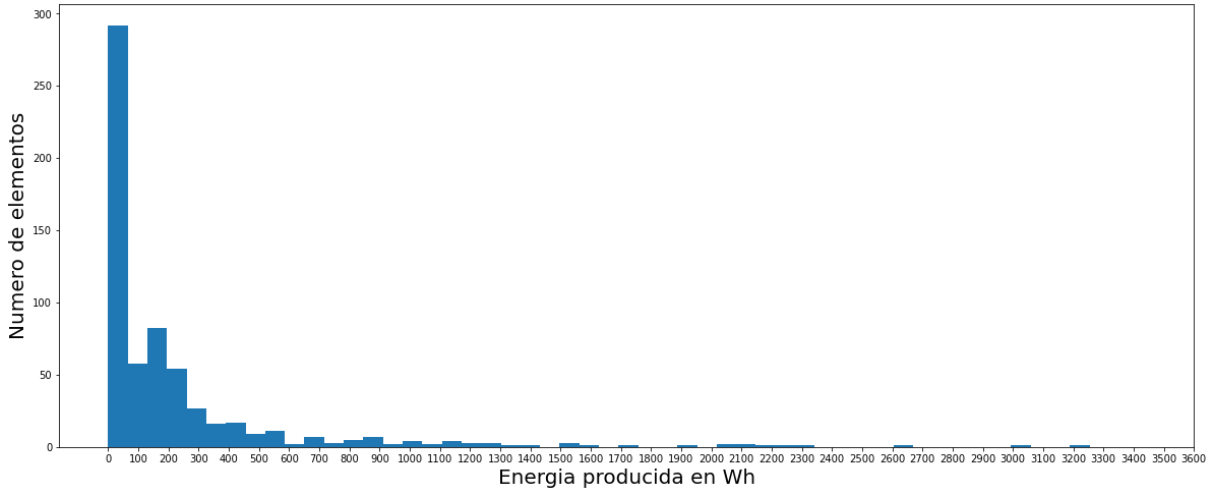


Figura 62: Histograma de la producción de la turbina

Se puede ver que hay un montón de valores entre 0 y 50. Los pequeños valores de producción son un problema por dos razones:

- Primeramente, los errores son muy altos mientras que las predicciones son bastantes buenas. En efecto, cuando el valor que predecir es muy pequeño, pequeñas variaciones implican grandes errores. Así, el error global es más grande de lo que debería ser.
- Además, la red trabaja para adaptarse a pequeñas variaciones, aunque no es necesario. Así los grados de libertad de las redes son gastados en las pequeñas predicciones. Así se necesita una red más compleja para adaptarse a los pequeños y grandes valores.

Para paliar este problema, una solución es de hacer que los datos pequeños no impacten la función de error. La idea es de crear un nuevo indicador que no tenga en cuenta estos pequeños valores, o más precisamente que no tenga en cuenta cuando la predicción está bastante cerca del valor. La idea es de ponderar el cálculo de error. Por eso se considera que cuando la predicción y la producción real son bastante cercas, se utiliza el error absoluto. Cuando no es el caso, se utiliza el error RMSE100.

$$Error\ ponderado = \begin{cases} |Prod - Pred|\ si\ |Prod - Pred| < 5 \\ RMSE100(\ Prod, Pred)\ si\ no \end{cases}$$

Ecuación 20: Función de error ponderada

Implementando esa función en Python da el siguiente código:

```
def errorPond(y_true, y_pred):
  if (K.abs(y_true[0]-y_pred[0])*5594)<5: #5Wh de error se considera que es
un resultado bastante suficiente
    return (2*K.abs(y_true[0]-y_pred[0])*5594)
  else:
    return 100*K.sqrt(K.mean(K.square(y_pred - y_true), axis=-
1))/K.mean(y_true)
```

Con este indicador se obtienen los siguientes errores para cada red.

	ANN	**RNN**	**CNN**

RMSE100	51.2%	45.5%	47.9%
ErrorPond	46.4%	22.3%	33.5%

Tabla 2: Comparación RMSE100 y ErrorPond

Así se puede ver que el impacto de las pequeñas predicciones estaba bastante grande.

Con el **RNN** se puede alcanzar un error de 22.3%, es decir una precisión de 77.7%. Lo que es un muy bueno resultado comparándolo con el método 1. El uso de esta nueva función de error permite paliar el primer problema debido a los datos de casi no producción. Pero todavía queda el segundo problema, las redes trabajan sobre los datos pequeños y así pierden precisión sobre los grandes datos. Además, utilizar la función de error ponderada es una pequeña trampa. En efecto, el error de los datos pequeños, como es bajo, baja el promedio global más que lo que debería. El error de estos datos no debería impactar el error. Pero esto no es posible porque una función de error debe siempre enviar un resultado. Para todas estas razones hay que pensar en otro método.

6.3.3. Método 3 clasificación y regresión

El objetivo de esta parte es que no impacten las pequeñas producciones sobre el aprendizaje de la red.

El tercer método se basa un principio muy sencillo. Para que no impacten las pequeñas producciones en las redes, no deben participar al entrenamiento. Es decir que antes el entrenamiento del método 2, hay que quitar los datos demasiados pequeños. Así antes de las redes neuronales de regresión como el método 2, hay que clasificar los datos pequeños y los datos grandes. Se debe hacer de manera automatizada, utilizando solamente los datos de producción de las 10 horas anteriores. Es un problema de clasificación.

En vez de un problema complicado, se descompone la situación en dos problemas, uno de clasificación y uno de regresión. Hay solamente uno de regresión porque se considera que cuando la producción es muy baja la predicción puede ser 0. Es decir que cuando se produce muy poca energía, se puede considerar que no se produce nada.

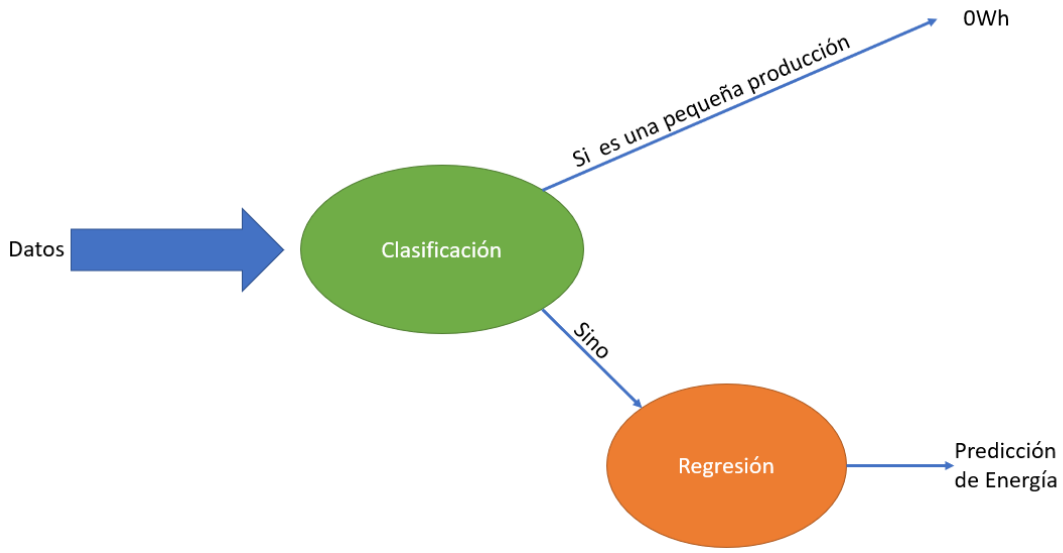


Figura 63: Resumen método 3

Para la parte regresión se utilizará las redes de la parte 2. Pero hay que trabajar sobre la parte clasificación. La parte clasificación consiste en repartir los datos entre dos categorías, la categoría 0 con pequeños valores y la categoría 1 con los datos que van a entrar en la predicción de la producción.

Pero antes de entrenar un modelo o aun pensar en un método, hay que clasificar los datos a mano. Es decir que hay que elegir dónde poner el límite entre pequeñas producciones y las grandes. Es la definición de un umbral. Estudiando la Figura 62, se puede ver que los datos entre 0 y 50Wh son muy representados. Así se podría poner 50Wh como valor limite, aunque puede parecer mucho. En efecto será considerar que cuando se produce 50Wh, se produciría nada.

Hay que volver al aspecto tecnológico de la turbina. La turbina tiene una potencia nominal de 5kW, es decir 5 000W. Si se considera que la turbina funciona a potencia nominal durante la ventana de predicción, dos horas. ¿cuánta energía producirá? La turbina producirá 5 000 X 2= 10000Wh. Así el umbral de 50Wh corresponde a 0.5% de la producción nominal. Por eso se considera que se puede hacer esa asunción.

6.3.3.1. Clasificación

Se dividen los datos entre la categoría 1 y la categoría 0. Para eso, se guarda el preprocesamiento de los métodos 1 y 2, quitando los datos falsos con la función *smooth*. La diferencia se hace después.

En vez de la producción de energía en la lista **y**, se necesita la categoría. Para hacer eso se define un umbral a 50Wh y se cambia los datos de salida. Eso se hace mediante la función asignación

```
def asignación(Y):
  for k in range(len(Y)):
    if Y[k]*5594<50:
      Y[k]=0
    else:
      Y[k]=1
  return Y
```

Así en vez de tener la producción durante dos horas en salida, se obtiene la categoría. Se estudia los datos para obtener la repartición entre las dos categorías.

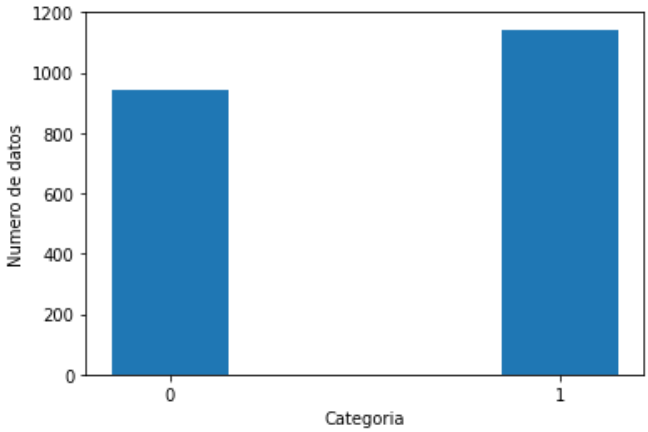


Figura 64: Histograma de las categorías

Se puede ver que las dos categorías están bastante bien equilibradas. Es una buena cosa para que el aprendizaje se haga bien. La clasificación de los datos se puede hacer de varias maneras, con métodos de *Machine Learning* o con métodos de *Deep Learning*: redes neuronales.

Como se trabajó mucho con redes neuronales antes, se decidió trabajar con métodos de M*achine Learning*. Para el *Deep Learning*, la biblioteca de referencia es **TensorFlow**, para el M*achine Learning*, se usa **Scikit-Learn.**

Se estudiará 3 algoritmos diferentes:

- Los **SVM:** El *Support Vector Machine* es un algoritmo de clasificación que se basa sobre el principio siguiente. Se pone todos los datos en un espacio multidimensional y el algoritmo intenta encontrar la frontera entre los datos de diferentes categorías. Se puede elegir la forma de la frontera cambiando el núcleo del **SVM**: *linear, polynomial, Gaussian.*
- El **KNN:** El algoritmo de *K Nearest Neighbours* se basa en la proximidad entre los datos. De la misma manera se pone los datos en un espacio multidimensional. Después, según la proximidad de un dato con los demás, se elige la categoría más cercana.
- El **RandomTreeForest** se basa en algo muy diferente. Se construye una multitud de árboles (un bosque) decisionales según los datos de entrenamiento. Es decir, un árbol con ramas que permiten separar los datos, y al final de las ramas, las hojas corresponden a las categorías. Se puede elegir los parámetros como el número de árboles o como se separan las ramas.

Como los tres algoritmos de basan sobre la biblioteca **Scikit-Learn**, las implementaciones se parecen mucho.

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)

y_pred = neigh.predict(X_test)
```

```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier()
forest.fit(X_train, y_train)

y_pred = forest.predict(X_test)
```

Primero, hay que importar las bibliotecas, después se inicializan los algoritmos y finalmente se entrenan los modelos. Una vez el modelo entrenado, se puede hacer predicciones utilizando los datos de prueba. Ahora, hay que comparar la eficiencia de estos algoritmos para el problema de clasificación de las producciones de energía. Se utiliza una herramienta llamada matriz de confusiones. Esta matriz permite comparar la precisión del modelo, pero también el número de falsos positivos y falsos negativos. Las columnas corresponden a la predicción de clases y las líneas a las verdaderas categorías. Así se puede ver las clases 0 que fueron bien o mal predichas y lo mismo para la clase 1. También se puede obtener la precisión de predicción de cada categoría que corresponde al número de datos bien predichos de una categoría partido el número de datos de esa categoría.

SVM		
	0 pred	**1 pred**
0	271	35
1	51	268
Precisión	0.89	0.84

KNN		
	0 pred	**1 pred**
0	250	56
1	36	283
Precisión	0.82	0.88

RandomTreeForest		
	0 pred	**1 pred**
0	275	31
1	32	287
Precisión	0.90	0.90

Tabla 3: Matrices de confusión de los algoritmos de Machine Learning

De los tres algoritmos el *RandomTreeForest* parece ser el más eficaz con una precisión de **90%** por cada clase. Por eso se elige este algoritmo para la clasificación.

Viendo las cosas con perspectivas, es decir que hay:

- **10%** de las pequeñas producciones que están predichas como grandes y entraran en la red de regresión
- **10%** de las grandes producciones que están predichas como pequeñas y se consideran como una producción de 0Wh

El primer caso no es tan grave, la red podrá predecir que son pequeñas producciones de la misma manera que en el método 2. Pero, el segundo caso es más preocupante. En efecto, cuando grandes producciones son predichas como pequeñas, no hay una segunda oportunidad como en el caso 1, la predicción será 0Wh.

Teniendo en cuanta este problema, hay que minimizar la proporción de datos que entran en el caso 2. Para hacer eso se va a utilizar una pequeña trampa, se va a artificialmente crear más datos de categoría 1. Así el algoritmo predecirá más fácilmente esta categoría. Para hacer esto, se multiplican los datos de la categoría 1. Se multiplican por ocho los valores de categoría 1.

```
X_train2=[]
y_train2=[]
for k in range(len(X_train)):
  if y_train[k]==1:
    X_train2.append(X_train[k])
    X_train2.append(X_train[k])
    y_train2.append(y_train[k])
    y_train2.append(y_train[k])
    X_train2.append(X_train[k])
    X_train2.append(X_train[k])
    y_train2.append(y_train[k])
    y_train2.append(y_train[k])
    X_train2.append(X_train[k])
    X_train2.append(X_train[k])
    y_train2.append(y_train[k])
    y_train2.append(y_train[k])
    X_train2.append(X_train[k])
    X_train2.append(X_train[k])
    y_train2.append(y_train[k])
    y_train2.append(y_train[k])
  else:
    X_train2.append(X_train[k])
    y_train2.append(y_train[k])
```

Se hace mediante este código muy sencillo. Así la categoría 1 es mucho más representada ahora. Crea un sobre entrenamiento del algoritmo de manera que las falsas predicciones de la categoría 0 disminuyen.

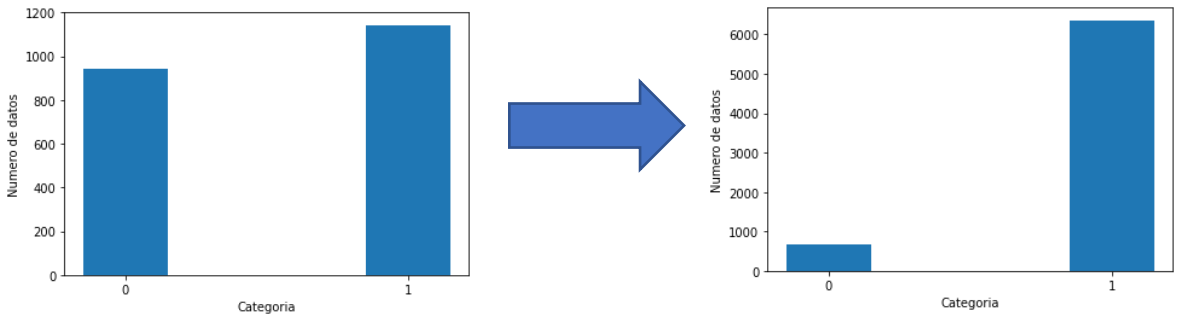


Figura 65: Comparación de histogramas

Este truco combinado con la modificación de los parámetros del *RandomTreeForest* permite lograr una clasificación más adaptada al problema.

Se entrena el algoritmo sobre los datos modificados y se hace la prueba sobre los datos de *y_test*.

```
from sklearn.ensemble import RandomForestClassifier
forest=RandomForestClassifier(n_estimators=50,min_samples_split=4,min_sampl
es_leaf=10)
forest.fit(X_train2, y_train2)
```

```
y_pred = forest.predict(X_test)
```

Se obtiene la matriz de confusión con el truco y los nuevos parámetros del algoritmo.

	0 Pred	**1 Pred**
0	193	85
1	14	334
Precisión	0.8	0.93

Tabla 4: Matriz de confusión de la clasificación

Se puede ver la precisión de la categoría 1 sube. Aunque la precisión de la categoría 0 baja, no es tan importante como la categoría 1. En efecto, el objetivo de la clasificación es eliminar el máximo de pequeñas producciones posibles.

6.3.3.2. Regresión

En esta parte, hay que entrenar la red de regresión para predecir la cantidad de energía de las grandes producciones. El entrenamiento se hará de manera independiente a la clasificación. Se selecciona solamente los datos con una producción superior a 50Wh durante las dos horas de estudio.

Se hace al mismo tiempo que la eliminación de los datos con curvas demasiadas suaves.

```
yf=[]
Xf=[]
for k in range(len(y)-1):
  if smooth(X[k])>15 and (y[k]*5594)>50:
    yf.append(y[k])
    Xf.append(X[k])
X=Xf
y=yf
```

Se elije la red más eficiente del método 2, es decir la **RNN**. En efecto, la parte regresión del método 3 corresponde al método 2 con menos datos. Así se considera que los resultados estarán bastantes similares y que no hace falta intentar otra vez con las 3 redes.

Se cambio un poco los parámetros de la red:

```
n_features =1
n_steps=60*10 #prod
X_trainRNN = X_train.reshape((X_train.shape[0], n_steps, n_features))
X_testRNN = X_test.reshape((X_test.shape[0], n_steps, n_features))

# define model
modelRNN = tf.keras.Sequential()
modelRNN.add(tf.keras.layers.LSTM(50,input_shape=(n_steps,n_features),retur
n_sequences=True))
modelRNN.add(tf.keras.layers.LSTM(50))
modelRNN.add(tf.keras.layers.Dropout(0.5))
modelRNN.add(tf.keras.layers.Dense(1,activation="sigmoid"))
```

```
modelRNN.summary()
modelRNN.compile(optimizer='rmsprop', loss=mseW, metrics=[mse100] )

modelRNN.fit(X_trainRNN,y_train,epochs=200,verbose=2,batch_size=5,validatio
n_data=(X_testRNN,y_test))
```

Se utiliza solamente dos capas **LSTM** de 50 bloques. Es suficiente para obtener buenos resultados.

Se obtiene sobre los datos de prueba un error **RMSE100** de **38.6%**, es decir una precisión de **61.4%**. Es una bastante buena predicción y una verdadera precisión sin trampa comparando con el error ponderado.

Si se considera una repartición 40% de datos de la categoría 0 y 60% de la categoría 1. Se obtiene una precisión teórica de:

$$Precision = 0.4 * P_{clas} + 0.6 * P_{regression}$$

Se obtiene una precisión global teórica de **0.4 X 80 + 0.6 X 61.4= 68.9%**

Pero esta precisión no tome en cuenta la posibilidad que las categorías 0 mal predicha pueden ser bien predichas por la parte de regresión. La precisión real podría ser más alta.

Así se elige el método 3 para predecir la producción eólica.

6.4. Conclusiones sobre la predicción eólica

Se eligió el método 3 para predecir la producción eólica, este método tiene una precisión teórica de **68.9%**. Pero hay que probar este método para obtener su precisión global sobre los datos de prueba. Hay que hacer una prueba como si fuera condiciones reales. Es decir, al principio no se sabe si es una categoría 0 o 1. Para lograr esto, se guardan los modelos de *Machine Learning y Deep Learning*. Primeramente, se predecirá si el dato es de categoría 1 o 0 y según el resultado se predecirá la cantidad de energía producida con la red de regresión.

Para obtener el error global del método, se proceda de la siguiente manera:

- Se predice la categoría del dato, de ahí sale dos casos:
    - Si es de categoría 1: se predice la cantidad de energía producida durante la ventana de dos horas, y se utiliza el error **RMSE100**.
    - Si es de categoría 0, ahí hay dos casos otra vez:
        - Si la producción es realmente de categoría 0 (Producción <50Wh), se considera un error de 0%
        - Si la producción no es realmente de categoría 0, se calcula el error **RMSE100** entre 0Wh y la producción real.
- Cuando todos los errores de los datos de pruebas están obtenidos se hace el promedio para obtener el error global y finalmente se calcula la precisión.

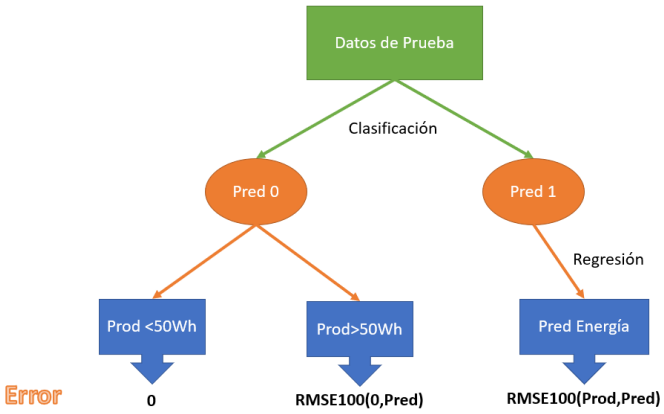


Figura 66: Resumen del cálculo de error global del método 3

Se traduce por el siguiente código:

```
modelRNN=tf.keras.models.load_model('modelRNNmetodo3.h5',custom_objects={"m
se100": mse100,"mseW":mseW})
n_features =1
n_steps=60*10
X_testRNN = X_test.reshape((X_test.shape[0], n_steps, n_features))

ClassPred=forest.predict(X_test)
RegressionPred=modelRNN.predict(X_testRNN)
```

Primero, se cargan los modelos y se predicen para todos los datos de prueba, la categoría y la regresión cuando la categoría predicha es 1.

```
Pred1=[]
True1=[]
Pred2=[]
True2=[]
c0=0
c1=0
c2=0
for k in range(len(ClassPred)):
  if ClassPred[k]==0 and y_testF[k]*5594<=50:
    c0+=1
  elif ClassPred[k]==0 and y_testF[k]*5594>50:
    Pred1.append(0)
    True1.append(y_testF[k])
    c1+=1
  elif ClassPred[k]==1:
    Pred2.append(RegressionPred[k][0])
    True2.append(y_testF[k])
    c2+=1

err2=rmse100(np.array(True2),np.array(Pred2))
err1=rmse100(np.array(True1),np.array(Pred1))

errG=(err1*c1+err2*c2)/(c0+c1+c2)
```

Se pasa sobre los datos de pruebas, cuando la predicción de clase es 0 y está bien predicha, se añade 1 al contador *c0*. La condición siguiente con el contador *c1* corresponde a las categorías predichas 0 que son realmente de categoría 1. Para ellas se pone 0 en valor predichas y se calcula al final el valor

RMSE100. La última condición corresponde a las predicciones de categoría 1, no importa si están falsas o no. En efecto, en todos los casos hay que predecir con el algoritmo de regresión. Al final, se calcula el error de cada condición (para la primera el error se considera 0), y se pondera según los contadores para obtener el error global.

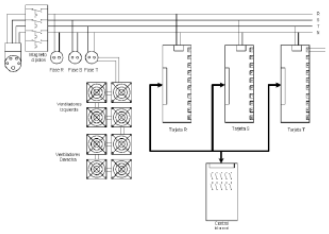
Se obtiene los siguientes resultados:

- El número de categoría 0 bien predichos **c0=243**
- El número de categoría 1 mal predichos **c1=41**
- El número de datos predichos como categoría 1 **c2=342**
- El error sobre los datos de categoría 1 mal predichos: **RMSE100= 100%** lo que parece lógico porque la predicción es 0Wh así se obtiene una diferencia de 100%.
- El error sobre los datos predichos 1: **RMSE100=39.1%** lo que es bastante cerca del error de la red de regresión.

Por fin, se obtiene un error global de **27.9%** es decir una precisión de **72.1%**. Una precisión más alta que la precisión teórica. Eso se debe al hecho que los datos de categoría 0 mal clasificados pueden tener una predicción buena con el algoritmo de regresión.

7. Predicción de la demanda de energía

La Microred del *LabDer* es una red experimental, es decir que no sirve para alimentar ninguna carga. Usualmente, las microrredes pueden alimentar varias cargas diferentes, unas plantas, una casa o un barrio o mismo un pequeño pueblo.



PROGRAMMABLE RESISTIVE LOAD	
Power rating	10 kW
Number of resistor	30
Resistor power rating (At 230 V)	333 W

Figura 67: Carga LabDer

En el laboratorio, se puede distinguir dos tipos de consumo. Un consumo base los días durante los cuales no hay actividad o pruebas. Es el consumo del laboratorio en sí mismo. Y un consumo peculiar cuando hay actividad o pruebas. En este caso, el consumo puede ser muy variable. En efecto, cuando hay prueba, la demanda depende de los experimentos realizados, es impredecible.

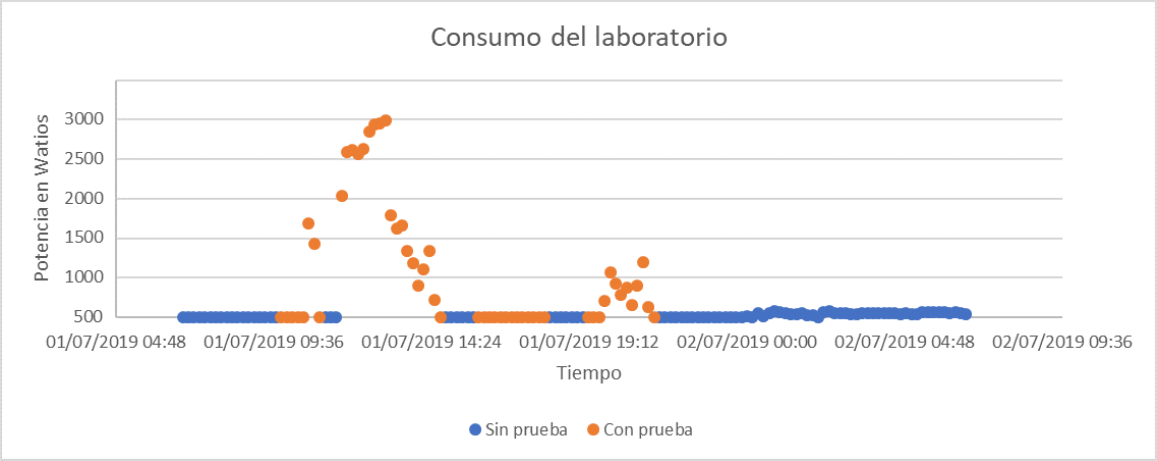


Figura 68: Ejemplo de consumo del laboratorio

Cuando hay pruebas, el consumo es muy alto y aleatorio, mientras que el resto del tiempo el consumo es bastante constante y bajo. Como se dijo antes, no se puede predecir el consumo cuando hay pruebas. Por eso, se estudian los datos de días sin pruebas.

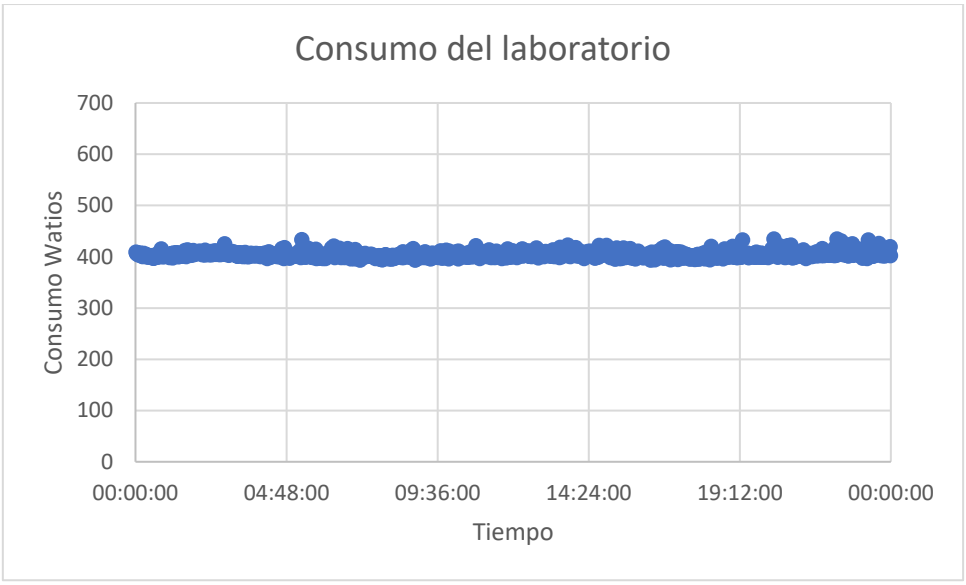


Figura 69: Consumo del laboratorio sin prueba

En la Figura 69, se observa el consumo del laboratorio un día sin prueba. Durante 24 horas, el consumo es constante alrededor de 10W. Corresponde al consumo de base del laboratorio. Así se puede ver que cuando no hay pruebas, el consumo del laboratorio es constante y debido al consumo de los aparatos del laboratorio. En resumen, el consumo del *LabDER* es impredecible si hay pruebas, y si no hay, el consumo es constante. Por eso no vale la pena predecir el consumo del laboratorio.

Considerando eso, se decidió de predecir de manera completamente independiente al *LabDER*, el consumo de un edificio de la UPV. Se obtuvo datos de consumo del edificio 3P cual corresponde a la Escuela Técnica Superior de Ingeniería Agronómica y del Medio Natural.

Figura 70: Edificio 3P

En los datos de consumo del edificio, se utilizan solamente los datos de consumo cuando la UPV está abierta. Se quiere predecir el consumo del edificio 24 horas antes, de la misma manera que la producción del sistema solar. Con el estudio de los datos, se puede ver que le consumo sigue un perfil típico todos los días. Así, parece una buena idea utilizar el mismo método que para la predicción del sistema solar. En efecto, la producción de la planta fotovoltaica sigue también un perfil típico.

Los datos de demanda se miden cada 15 minutos. Así un día de consume corresponde a 24 X 4 = **96 datos.** Para predecir el consumo, los datos de entrada son dos días de consumo anterior. Permitirá a la red de tener una idea sobre la temporada y si es un día de vuelta a clase o no. Los datos se reorganizan en una imagen de dos canales de 4 X 24 datos. Cada canal corresponde a un día de consumo.

Se utiliza una red CNN de la siguiente forma:

- 3 capas de LSTM de tamaños respectivos 256, 256 y 128 bloques LSTM. Con filtros 2 X 2.
- Una capa de *Dropout* de parámetro 0.3
- Una capa de ANN de 96 neuronas

```
X_trainCNN=X_train.reshape(len(X_train),2,4,24)
X_testCNN=X_test.reshape(len(X_test),2,4,24)

# define model
modelCNN = tf.keras.Sequential()
modelCNN.add(tf.keras.layers.Conv2D(256,(2,2),activation='relu',input_shape
=(2, 4, 24),padding='same'))
modelCNN.add(tf.keras.layers.Conv2D(256,(2,2),activation='relu'))
modelCNN.add(tf.keras.layers.Conv2D(128,(2,2),activation='relu',padding='sa
me'))

modelCNN.add(tf.keras.layers.Flatten())
modelCNN.add(tf.keras.layers.Dropout(0.3))
modelCNN.add(tf.keras.layers.Dense(96,activation="relu"))
modelCNN.summary()
modelCNN.compile(optimizer='adam', loss=mse100 )

# fit model
modelCNN.fit(X_trainCNN,y_train,epochs=200,verbose=2,batch_size=5,validatio
n_data=(X_testCNN,y_test))
```

Utilizando esta red, se obtiene un error sobre los datos de prueba de **46.5%**. Corresponde a una precisión de **53.5%.** Esta precisión parece muy baja comparando con las predicciones de la producción solar o eólica. Si se considera la energía consumida se obtiene una precisión de **74%**.

Se traza algunos ejemplos de predicción:

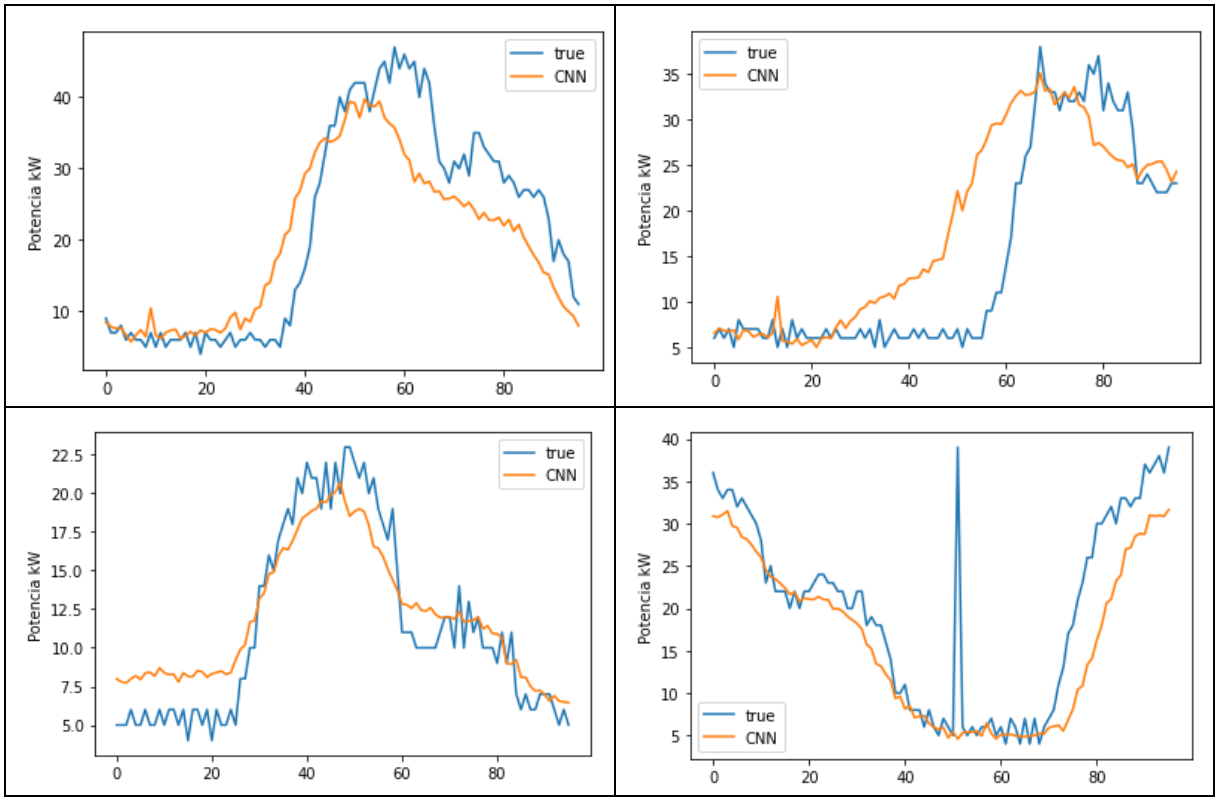


Figura 71: Ejemplos de predicción de demanda

El error de predicción puede ser bastante grande. Se puede explicar de diferentes maneras. La demanda eléctrica del edificio depende de los cursos que hay dentro. Así cuando los horarios de clase cambian, las predicciones son menos precisas. Además, los horarios pueden cambiar cada semestre y la red tendrá que aprender otra vez desde cero para predecir el consumo. También puede ocurrir eventos impredecibles cambiando completamente el consumo. Por ejemplo, cuando la UPV fue cerrada por culpa del COVID 19. Además, errores de medidas pueden ocurrir, como en el último ejemplo, la red neuronal no predice esos cambios brutales.

8. Predicción del estado de carga e implementación

El estado de carga de una batería es la proporción de energía contenida en la batería comparando con su capacidad máxima. Así el estado de carga puede calcular de manera muy sencilla con la siguiente formula:

$$Estado\ de\ carga(\%) = \frac{E_{almacena}}{Capacidad\ de\ la\ bateria} * 100$$

Ecuación 21: Estado de carga

En el caso del *LabDer* se puede almacenar energía de diferentes maneras:

- Se puede almacenar en el sistema de baterías con una capacidad máxima de 10.32 kWh para un tiempo de descarga de 10 horas (C10).
- Se puede también almacenar energía mediante hidrógeno. Se electroliza agua para formar hidrógeno y después se utiliza el hidrógeno para alimentar la pila de combustible. En este caso, no hay una capacidad máxima, sino el número de botellas disponibles. Así no se podrá calcular el estado de carga.

Por estas razones, se predecirá la cantidad de energía en los sistemas de almacenamiento. Además, la repartición entre energía en las baterías y en las botellas de hidrógeno se hace de manera manual y no se puede predecir lo que va a elegir el investigador.

La cantidad de energía que se almacena corresponde al excedente de energía producida comparando con la demanda. Así, para intentar predecir este excedente, hay que estudiar el balance energético.

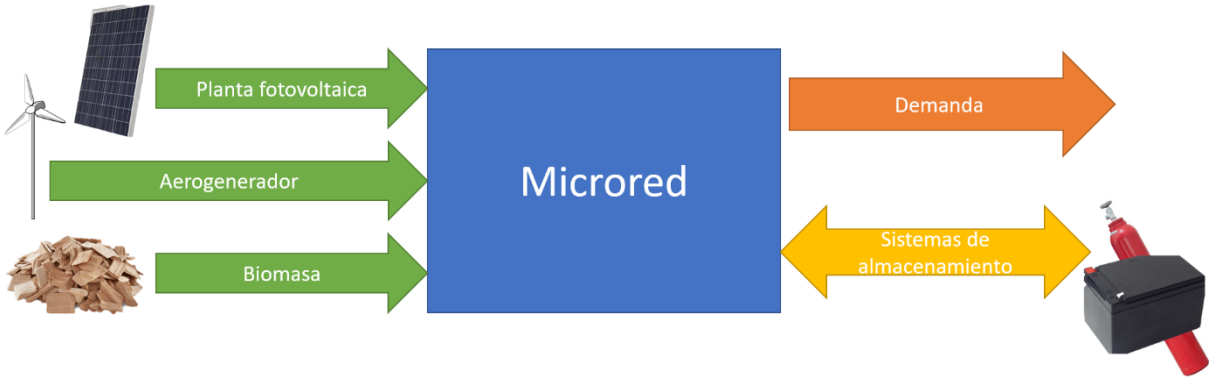


Figura 72: Balance energético de la Microred

En la Figura 72, se puede ver un resumen muy simplificado de la Microred *labDer*. Por un lado, lo que produce energía y por otro lo que consume. El esquema permite ver el flujo de energía a un instante dado. Se va a repasar cada elemento del esquema para ver cómo funciona.

- La planta fotovoltaica genera electricidad a partir del sol, por eso la producción no se puede controlar manualmente. Pero con el estudio anterior con redes neuronales se puede predecir la producción un día adelante con una muy buena precisión de **71%.**
- El aerogenerador genera electricidad usando el viento. No se puede controlar la producción de energía, y no se puede conocer directamente la producción que se vaya a la Microred. Pero en la parte anterior, se hizo una red neuronal para predecir la energía producida 2 horas adelante con una precisión de **72.1%.**
- Las plantas de biomasa se activan cuando se necesita. Hay solamente que introducir pellets en la caldera o encender la planta de gasificación. Por eso no hay que predecir nada sino decir cuándo se va a encender y a cuál potencia. Eso permitirá conocer la cantidad de energía producida por biomasa a cada momento.
- La demanda de energía del *LabDer* se elige manualmente. Así es lo mismo que para las plantas de biomasa, hay solamente que decir cómo se pondrá la carga. En el caso de que sea una verdadera carga, como el ejemplo anterior de la UPV se podría utilizar el método describió antes para obtener las curvas de demanda un día adelante.

- Los sistemas de almacenamiento de energía corresponden al sistema de baterías y el sistema de hidrógeno. Pueden recibir energía de la red como enviar energía. Se decidió reunir estos sistemas porque se elige de manera manual como se reparte la energía entre ambos. El flujo de energía que se va a estos sistemas es lo que se quiere predecir en esta parte.

Como se dijo antes, el esquema corresponde a lo que pasa en cada instante. En cada instante, la potencia que entra en la Microred y la potencia que sale es la misma. Se considera que se puede ignorar las perdidas como la red es bastante pequeña. Así en cada momento, se obtiene la siguiente ecuación:

$$P_{solar} + P_{eolica} + P_{Biomasa} = P_{demanda} + P_{almacenamiento}$$

Ecuación 22: Balance de potencia

De esta ecuación, se puede determinar $P_{almacenamiento}$ que es la única incógnita de la ecuación. $P_{almacenamiento}$ es positiva cuando hay más producción que demanda. En este caso, las baterías se cargan. $P_{almacenamiento}$ es negativa cuando hay menos producción que demanda y las baterías alimentan la red.

Ahora se conoce en cada instante la potencia que va a los sistemas de almacenamiento. Pero lo que se quiere es la cantidad de energía dentro de los sistemas de almacenamiento. Por eso hay que integrar la Ecuación 22. Conociendo la cantidad de energía en las baterías a un instante, se puede obtener la cantidad de energía almacenada los siguientes momentos.

$$E_{almacenada}[k+1] = E_{almacenada}[k] + E_{solar}[\Delta k] + E_{eolica}[\Delta k] + E_{biomasa}[\Delta k] - E_{demanda}[\Delta k]$$

Ecuación 23: Energía almacenada

Resumiendo:

- $E_{almacenada}[k+1]$ es lo que se quiere obtener, la cantidad de energía almacenada a un instante.
- $E_{almacenada}[k]$ la cantidad de energía almacenada al instante anterior. Se mide directamente.
- $E_{solar}[\Delta k]$ la cantidad de energía producida por la planta fotovoltaica. Se predice utilizando las redes neuronales de la parte anterior.
- $E_{eolica}[\Delta k]$ la cantidad de energía producida por el aerogenerador. Se predice utilizando las redes neuronales de la parte anterior.
- $E_{biomasa}[\Delta k]$ la cantidad de energía producida por las plantas de biomasa. Se indica manualmente como se activa la planta cuando se quiere.
- $E_{demanda}[\Delta k]$ la cantidad de energía que se consume por la carga. Si se utiliza la carga manual, se indica manualmente como para la producción por biomasa. Sino se utiliza el método de la UPV con el método anterior.

La limitación de esta ecuación es la producción por el aerogenerador que se puede solamente predecir dos horas adelante. Así se podrá predecir la cantidad de energía en las baterías solamente hasta dos horas en el futuro.

La diferencia con la predicción de producción eólica o de la planta solar fotovoltaica, es que no se puede predecir la cantidad de energía almacenada de manera completamente automatizada. Se necesita una acción humana para entrar los datos de producción de biomasa y de consumo, si no se utiliza el consumo del edificio 3P. Para hacer eso, se puede crear una pequeña herramienta grafica con *Python* para entrar estos valores. Se puede utilizar la herramienta *tkinter*. Esta biblioteca *Python*

permite crear interfaces gráficas y recuperar los datos introducidos a mano para hacer cálculos, por ejemplo. El uso es bastante sencillo. El código es el siguiente:

```
import tkinter as tk

window=tk.Tk()

titre=tk.Label(window,text='Predicción de la cantidad de energía
almacenada',font=("Helvetica", "16",'bold'))
titre.pack(side='top',pady=15,padx=10)

EactualF=tk.Frame(window)
EactualF.pack(pady=10)
Eactual=tk.Label(EactualF,text='Energía almacenada H0 [kWh]')
Eactual.pack(side='left',padx=10)
EactualE=tk.Entry(EactualF)
EactualE.pack(side='right')

BiomF=tk.Frame(window)
BiomF.pack(pady=10)
Biom=tk.Label(BiomF,text='Producción por biomasa H+2 [kWh]')
Biom.pack(side='left',padx=10)
BiomE=tk.Entry(BiomF)
BiomE.pack(side='right')

ConsumF=tk.Frame(window)
ConsumF.pack(pady=10)
Consum=tk.Label(ConsumF,text='Consumo H+2 (si carga manual) [kWh]')
Consum.pack(side='left',padx=10)
ConsumE=tk.Entry(ConsumF)
ConsumE.pack()

CalculoF=tk.Frame(window)
CalculoF.pack(pady=10)
Calculo=tk.Label(CalculoF,text='Energía almacenada H+2 [kWh]')
Calculo.pack(side='left',padx=10)
CalculoB=tk.Button(CalculoF,text='Calcular',font=('bold'))
CalculoB.pack()


window.mainloop()
```

Con este programa, se obtiene la siguiente interfaz.

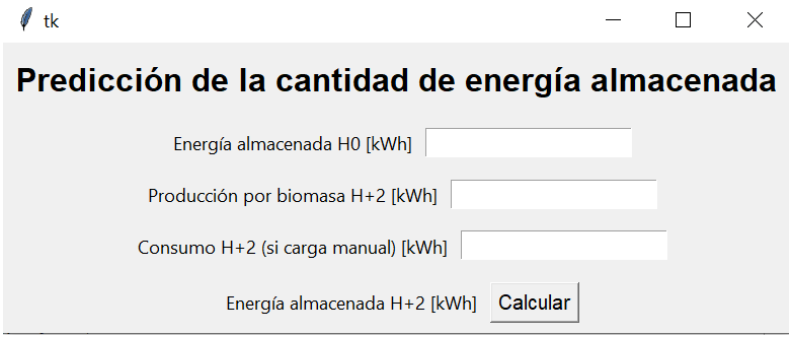


Figura 73: Interfaz de cálculo de la energía almacenada

Se introducen los valores de energía almacenada actualmente, también la previsión de producción con las plantas de biomasas las dos próximas horas. Si se utiliza la carga manual se introduce el consumo de las dos próximas horas. Sino se utilizará la red neuronal como en el ejemplo del edificio 3P.

Después de introducir todos los datos, se pulsa el botón calcular para tener la predicción de energía almacenada dos horas en el futuro. Este botón lanza el cálculo de la Ecuación 23, utilizando los últimos datos del *LabDer* para la predicción de producción con redes neuronales. En efecto, para ser eficaz, esta herramienta gráfica deberá ser implementada directamente en el *LabDer* al sitio donde se guardan los datos. Así, con el script de proceso de datos, se ponen los datos brutos del *LabDer* en el bueno formato para las redes neuronales en tiempo real. Después se predice la producción con estas redes y finalmente se calcula la cantidad de energía almacenada cuando se pulsa el botón calcular.

En realidad, se utilizan las baterías solamente cuando hay pruebas. Así, para comprobar este algoritmo, hay que utilizar datos de pruebas. El problema es no hay muchas pruebas de 72h, que son necesarias para predecir la producción solar.

Se intenta calcular la cantidad de energía que hay que almacenar en las baterías. Por eso se utilizan datos de pruebas. La predicción de la curva solar se hace con una precisión de 70%, se simula una predicción de esta precisión. Lo mismo para la producción eólica, se calcula la energía que hay que almacenar. No obstante, para compararla con la energía cargando las baterías, hay que tener en cuenta el inversor cargador. En efecto el *Smart meter* que mide la potencia yendo a las baterías está después del inversor, mientras que se calcula la potencia yendo al inversor. Para solucionar este problema se multiplica la energía calculada por la eficiencia máxima del inversor: 95%.

Se trazan algunos ejemplos de predicción mediante Excel.

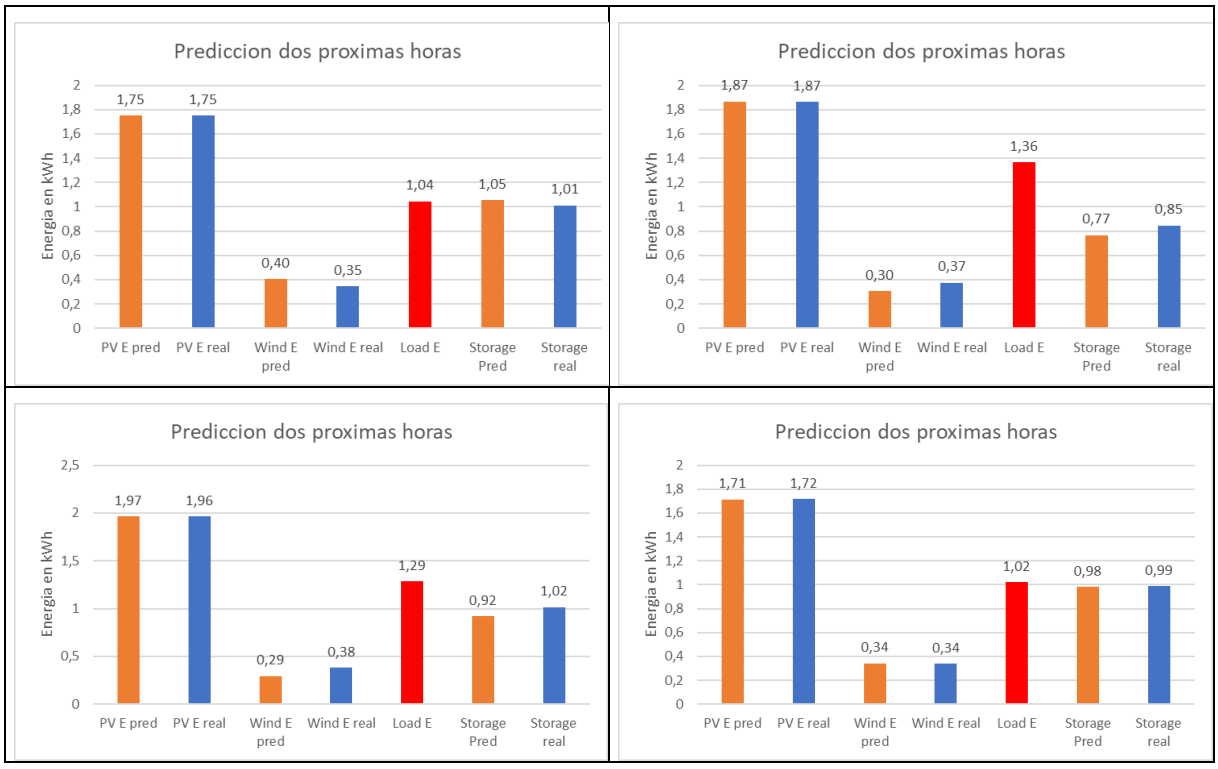


Figura 74: Ejemplos de predicción de energía almacenada

Se puede ver que la predicción de la producción por el sistema solar es muy buena. Mientras que para la predicción de la producción eólica los resultados son un poco menos fiables. Sin embargo, la predicción de la cantidad de energía en las baterías es bastante buena. Se hace un promedio sobre todos los valores disponibles y se obtiene un error medio de **5.7%** que corresponde a una precisión de **94.3%**. Se debe al hecho de que no haya la incertitud sobre la demanda. En efecto, para la demanda se utiliza la demanda medida durante la prueba. En el caso de una demanda como la del edificio 3P, la precisión será más baja.

9. Análisis económico

Las predicciones obtenidas se pueden usar de varias maneras. Son solamente herramientas para gestionar mejor las redes inteligentes. Se estudiará dos escenarios.

El primer escenario corresponde a la automatización completa de una red aislada. Es decir que el uso del generador depende de las predicciones de las redes neuronales. Así para evitar los cortes de energía, se enciende el generador cuando se predice poca energía disponible en el futuro. Se considera un generador diésel. En este escenario se ahorra energía pero se ahorra trabajo. En efecto no se necesita un operador para supervisar la red. Por eso el ahorro corresponde al coste de un operador de red. Se considera un salario a tiempo parcial de 500€/mes.

En el segundo escenario, el objetivo no es ahorrar dinero sino ahorrar energía y disminuir las emisiones de CO2. Se utilizarán las predicciones para implementar respuesta a la demanda. Lo que permite disminuir el uso del generador diésel durante las horas pico y evitar el gasto de energía. La literatura sugiere que la respuesta a la demanda permite ahorrar hasta 20% de energía. En este escenario se va a considerar un ahorro de 15% de energía, principalmente debido al generador diésel. Para el estudio económico se considerará un consumo anual de la red de 100MWh lo que corresponde al orden de magnitud del edificio 3P. Considerando un generador diésel, la energía volumétrica vale 9,63kWh/L y el precio 1,3 €/l. Así el precio de la energía con ese tipo de generador es de: **0,14€/kWh.** Además se podrá ahorrar también sobre el precio de las baterías, porque el sistema necesitará menos capacidad de almacenamiento.

	Escenario 1	**Escenario 2**
Coste de inversión €	20 100€	20 100€
Ahorro de energía kWh	0 kWh	15 000kW
Ahorro por año €	500*12=6000€	2100€
ROI (10 años)	198%	4,5%
PRI (años)	3,35 años	9,6 años
Ahorro de CO_2	0	4 160kg/CO2

Tabla 5: Análisis económico

En el primer escenario, se obtiene un periodo de recuperación de la inversión de 3,35 años y un ROI de 198%. Eso solo con la automatización del uso del generador. En el segundo escenario el objetivo es diferente, es ahorrar energía y disminuir las emisiones. Se puede ver que se pueden utilizar las predicciones para disminuir las emisiones de una red y ahorrar dinero.

Datos:

2,67kgCO2/litros de diésel

9,63kWh/litros de diésel

10. Conclusiones

Los objetivos de este TFM eran de predecir la producción eléctrica de una red inteligente, la Microred *LabDer* de la UPV, y también de predecir el consumo y la cantidad de energía almacenada. Para cumplir estos objetivos, primero, se trabajó sobre la producción de la planta fotovoltaica, después sobre la producción del aerogenerador y finalmente sobre el estado de carga de las baterías y la carga de la Microred. Como es un trabajo de investigación, hay que hacer el máximo para cumplir los objetivos utilizando los recursos disponibles. Por eso, los resultados pueden ser diferentes de los objetivos iniciales. Se va a comparar los resultados obtenidos con los objetivos iniciales.

A propósito de la producción de la planta fotovoltaica, el objetivo inicial era de predecir la curva de producción un día adelante. Utilizando una red de convolución **CNN**, se puede predecir esa curva con una precisión de **71%**. Se hace utilizando como datos de entrada, los datos de producción, de irradiancia y de temperatura de los dos días anteriores.

A propósito de la producción del aerogenerador, con el estudio previo de los datos, se podía decir ya que sería muy complicado predecir la curva de producción un día adelante. Por eso se decidió predecir solamente dos horas en el futuro. Pero, incluso con esta simplificación, los resultados obtenidos fueron muy bajos. Aunque se intentó quitar la parte muy aleatoria con una descomposición espectral, no se pudo lograr más de **24.1%** de precisión, lo que es muy bajo. Finalmente se decidió predecir la energía producida durante las dos próximas horas, asumiendo que los resultados serán más precisos. Trabajando en esta dirección y considerando que a veces la producción estaba tan baja que se podía ignorarla, se obtuvo una precisión de **72.1%** utilizando una red recurrente **RNN**. Para esta parte, se elijo obtener menos información fiable mientras que mucha información poca fiable. Para hacer esto, se utilizaron los datos de producción diez horas antes para predecir las dos próximas horas. Se podría intentar predecir un poco más adelante en el futuro, pero se necesitará más datos sin interrupción. En efecto, debido a la versatilidad del viento, es muy difícil interpolar cuando hay pocos datos.

Sin embargo, para trabajar con muchos más datos, es decir con una frecuencia de datos más alta, se necesita bastante potencia computacional que no tenía durante este trabajo. Con una cantidad mayor de datos, se podría predecir más precisamente y quizás sería posible predecir la curva de producción. En esta parte, tuve que revisar el objetivo a la baja. Pero, finalmente se obtuvo resultados bastante buenos y que pueden ser útil.

Después, se trabajó sobre la demanda y el estado de carga de las baterías. No se pudo utilizar los datos del *LabDer* para predecir el consumo de la misma manera que para la producción. En efecto cuando no hay pruebas, el consumo es constante, alrededor de 10W, y cuando hay pruebas la demanda es impredecible. Por eso se utilizaron los datos de demanda del edificio 3P de manera independiente al *LabDER*. Con estos datos, se obtuvo una precisión de **53.5%.** Este valor puede parecer poco porque es muy difícil predecir esta demanda considerando que cambian los horarios de clases. Con más datos se podría aumentar esta precisión. Además, eso es la precisión sobre la curva de consumo. SI se considera la energía consumida sobre el día, se obtiene una precisión de **74%.**

Para la parte sobre el estado de carga, se prefirió predecir la cantidad de energía almacenada. En efecto, en el *LabDer* se puede almacenar energía en baterías y también en botellas de hidrógeno, la gestión entre ambos sistemas siendo manual. Predecir la cantidad de energía almacenada tiene varias ventajas:

- No hay que considerar la eficiencia de los sistemas de almacenamiento porque solo se calcula lo que hay que almacenar.
- Permite a los operadores elegir como almacenar la energía.
- Se puede calcular con los datos predichos antes. En efecto esta cantidad de energía corresponde a la diferencia entre producción y consumo.

Como esta cantidad depende del consumo, se trata también esta parte de forma teórica. Por culpa de la predicción del aerogenerador, solo se puede predecir esta cantidad de energía dos horas adelante. Se hizo una pequeña interfaz gráfica para introducir los datos que se controlan manualmente. Se utilizan los datos de prueba del *LabDER* para probar el método de cálculo y se obtiene una precisión de **94.3%.**

La principal conclusión de este trabajo es que los algoritmos de *Deep Learning* y *Machine Learning* no son soluciones milagrosas. Es decir que no hay que poner *Deep Learning* por todas partes, hay situaciones donde soluciones más sencillas funcionan mejor, como en la parte de consumo o del estado de carga. Además, cuando se puede utilizar redes neuronales, no hay una solución universal, cada problema tendrá una red más eficaz que no funciona en un otro problema. Para encontrar la mejor red, hay que hacer una multitud de prueba. Por ejemplo, se puede notar que, para predecir la producción del aerogenerador, la **RNN** era la más eficaz mientras que este tipo de red no estaba muy eficaz para predecir la producción fotovoltaica. Lo que es muy interesante es intentar de interpretar ese resultado. ¿Porque un **CNN** es más eficaz que un **RNN** para la producción solar y para la producción del aerogenerador es el contrario?

Se explicó en el estado del arte que los **CNN** se utilizaban para trabajar con imágenes mientras que los **RNN** se utilizaban para los datos que dependen del estado anterior. La producción solar tiene siempre el mismo “aspecto”, una forma de campana alrededor de mediodía. Así se puede interpretar como una imagen que se deforma un poco según las temporadas o las condiciones meteorológicas. Pero sigue tener un aspecto bastante regular. Por eso un **CNN** funciona bien con esos datos.

En el caso de la producción del aerogenerador, los datos son muy diferentes debido a la versatilidad del viento. Así no se pueden considerar como imágenes. Pero, los datos dependen mucho de los estados anteriores. En efecto, cuando hay mucho viento, el siguiente segundo suele hacer todavía un

poco de viento. Por eso, las redes recurrentes funcionan bien para predecir la producción del aerogenerador. Es mi interpretación.

Los modelos y soluciones presentados tienen buenas precisiones, alrededor de **70%**, pero se podría aumentar esta precisión. La eficiencia de un modelo de *Deep Leaning* se basa sobre la calidad de los datos utilizados. Se podría obtener mejores datos de varias maneras:

- Aumentando el número de datos, en este TFM solo se utilizó un año de medidas por varias razones, tiempo de computación, accesibilidad de los datos y más.
- Utilizando datos completos, sin cortes. En efecto en los datos del *LabDer* había huecos que se rellenó con datos auxiliares. Pero las frecuencias de medidas no estaban las mismas. Resultó en curvas en escaleras y valores aproximadas. Así habría que aumentar la fiabilidad del sistema de almacenamiento de datos para evitar los huecos. Además, los datos están medidos solamente cuando está abierta la UPV. Por ejemplo, no hay datos en agosto. Una solución podría ser la automatización de las medidas todo el año para obtener datos en agosto y durante las vacaciones.

Todas estas medidas permitirán aumentar un poco la eficiencia de la predicción de producción solar, pero permitirá un gran aumento de la precisión de la predicción de la producción del aerogenerador. En efecto debido al uso de datos auxiliares se eliminó muchos datos para esta parte.

Pero aun con toda la máxima cantidad de datos posible, los modelos tienen sus límites. En efecto, las redes neuronales no pueden predecir eventos excepcionales. Por ejemplo, en el caso de este TFM, será difícil para las redes de predecir fenómenos meteorológicos puntuales: episodios de gota fría donde no hay sol durante días y hay mucho viento.

Los modelos desarrollados pueden ser implementados en redes aisladas o conectadas a la red eléctrica de suministro local. Con esos métodos se obtienen predicciones bastantes precisas sobre el comportamiento futuro de la red. Se pueden utilizar esas predicciones de varias maneras:

En el caso de redes aisladas, se puede automatizar la gestión de la red y el funcionamiento del generador, que sea de diésel o de biomasa. En efecto, utilizando la predicción de energía almacenada se puede encender automáticamente el generador cuando se predice que en dos horas la energía almacenada será bajo un umbral. Eso permite evitar la gestión manual del generador, y evitar los cortes de energía.

Se puede también utilizar las predicciones para implementar respuesta a la demanda. Cuando se predice una gran cantidad de energía disponible, se puede adelantar obras o tareas que consumen mucha energía. Al contrario, cuando se predice poca energía disponible, se puede desconectar cargas automáticamente, reducir el aire acondicionado o la calefacción para evitar cortes de energía y ahorrar energía. Todo eso se puede programar de manera muy sencilla y evitar que alguien monitoriza manualmente la red 24h/24h.

Los modelos presentados en este TFM son herramientas para obtener informaciones sobre el comportamiento futuro de la red. Se pueden utilizar de diferentes maneras y con diferentes objetivos, ahorrar dinero, disminuir las emisiones de CO2 y otros.

Para concluir, los modelos creados en este TFM permiten tener una muy buena idea del comportamiento de la Microred durante la mayoría del tiempo, pero no hay que esperar un 70% de precisión todos los días. Habrá días con una precisión de 50% o 40% tanto como días con una precisión de 90%. El objetivo de estos modelos es tener una idea de lo que pasará en el futuro para gestionar los flujos de energía. Estas redes no se vanaglorian de conocer el futuro, pero pueden aportar información bastante fiable cuando no se sabía nada antes. En efecto con esas predicciones, se puede ahorrar energía, dinero y emisiones de CO2.

11. Bibliografía

[1] REE, “España cierra 2019 con un 10 % más de potencia instalada de generación renovable.” .

[2] Y. Y. Hong and C. L. P. P. Rioflorido, “A hybrid deep learning-based neural network for 24-h ahead wind power forecasting,” *Appl. Energy*, vol. 250, no. January, pp. 530–539, 2019, doi: 10.1016/j.apenergy.2019.05.044.

[3] H. Zang, L. Cheng, T. Ding, K. W. Cheung, Z. Wei, and G. Sun, “Day-ahead photovoltaic power forecasting approach based on deep convolutional neural networks and meta learning,” *Int. J. Electr. Power Energy Syst.*, vol. 118, no. February 2019, p. 105790, 2020, doi: 10.1016/j.ijepes.2019.105790.

[4] S. Rodrigues Moreno, R. Gomes da Silva, V. Cocco Mariani, and L. dos Santos Coelho, “Multi-step wind speed forecasting based on hybrid multi-stage decomposition model and long short-term memory neural network,” *Energy Convers. Manag.*, vol. 213, no. February, p. 112869, 2020, doi: 10.1016/j.enconman.2020.112869.

[5] M. Gao, J. Li, F. Hong, and D. Long, “Day-ahead power forecasting in a large-scale photovoltaic plant based on weather classification using LSTM,” *Energy*, vol. 187, p. 115838, 2019, doi: 10.1016/j.energy.2019.07.168.

[6] Z. Niu, Z. Yu, W. Tang, Q. Wu, and M. Reformat, “Wind power forecasting using attention-based gated recurrent unit network,” *Energy*, vol. 196, p. 117081, 2020, doi: 10.1016/j.energy.2020.117081.

[7] Z. O. Olaofe, “A 5-day wind speed & power forecasts using a layer recurrent neural network (LRNN),” *Sustain. Energy Technol. Assessments*, vol. 6, pp. 1–24, 2014, doi: 10.1016/j.seta.2013.12.001.

[8] C. M. C. Creayla, F. C. C. Garcia, and E. Q. B. Macabebe, “Next Day Power Forecast Model Using Smart Hybrid Energy Monitoring System and Meteorological Data,” *Procedia Comput. Sci.*, vol. 105, no. December 2016, pp. 256–263, 2017, doi: 10.1016/j.procs.2017.01.219.

II. Presupuesto

El proyecto se compone principalmente de código Python. Se utilizan solamente programas "open source", así al nivel del *software* el coste corresponde a la remuneración del desarrollador. En España el precio horario es alrededor de 25€/hora, considerando que soy un desarrollador Junior. Este TFM corresponde a 30 créditos ECTS es decir 25X30=750 horas.

Considerando la parte *Hardware* es decir el equipo para la implementación de los métodos, una vez los algoritmos entrenados, no se necesita mucha potencia de cálculo. La manera más barata para implementar el sistema será sobre una *Raspberry pi,* una microcomputadora alrededor de 100€ para la más potente.

Se hace un resumen del coste total de proyecto.

Tarea	**Número de horas**	**Precio por hora**	**Precio**
Desarrollo de los algoritmos	750	25 €	18 750 €
Implementación física	50	25€	1250 €
Raspberry pi			100 €
Total			**20 100€**

Tabla 6: Coste del proyecto

Si se quiere implementar los métodos en diferentes laboratorios o redes eléctricas, solo habrá que pagar para la implementación y la computadora.

III. Anexos

Script, automatización del proceso de datos:

```
import pandas as pd
import os
from tqdm import tqdm

def toPWind(WindSpeed):
    return  WindSpeed**5  *7.74549*10**(-2)  -  WindSpeed**4  *3.70059  +
WindSpeed**3 *57.0966 -WindSpeed**2 *2.94899*10**2 + WindSpeed *5.20272*10**2
-60.43
#using excel tendance curve
def toPPV(SolarIrr):
    return 2.48*SolarIrr

def sample(timestep):
    path = os.getcwd()
    if not os.path.exists(path+'/sampled_files'):
        os.mkdir(path+'/sampled_files')

    print('Extracting LabDer data')
    files=os.listdir('Datos_brut')

data=pd.read_csv('Datos_brut/'+files.pop(),';',index_col=[1,2],low_memory=F
alse,decimal=',')
    for file in tqdm(files):

data=data.append(pd.read_csv('Datos_brut/'+file,sep=';',low_memory=False,in
dex_col=[1,2],decimal=','),sort=True)
    del data['Breaks']
    del data['V - PM2 - Load [R]']
    data.columns=['SolarIrr','WindSpeed','PPV','PWind']
    data['PWind']=data['PWind'].apply(lambda x: x if x>0 and x<10000 else
None)
    data=data.drop('17/09/2020',level=0)
    print("LabDer data extracted")

    print("Sampling LabDer")
    data=data.reset_index()
    data['DateTime']=pd.to_datetime(data['Date']+'
'+data['Time'],format='%d/%m/%Y %H:%M:%S')
    data=data.set_index(data['DateTime'])
    del data['Date']
    del data['Time']
    del data['DateTime']
    data=data.resample(timestep).mean()


    print("Extracting Auxiliary data")

homer=pd.read_csv('Homer.csv',sep=';',low_memory=False,index_col=[0,1],deci
mal='.')
    homer=homer.reset_index()
    homer['DateTime']=pd.to_datetime(homer['Date']+'                    '+
homer['Time'],format='%d/%m/%Y %H:%M')
    homer=homer.set_index(homer['DateTime'])
```

```
    del homer['Date']
    del homer['Time']
    del homer['DateTime']

    print("Sampling Auxiliary data")
    homer=homer.resample(timestep).interpolate() #better approximation

    print("Merging data")
    dataG=pd.merge(data,homer,how="left", on =('DateTime'))

    dataG['SolarIrr']=dataG['SolarIrr'].apply(lambda x: None if x==0 else x)
    dataG['WindSpeed']=dataG['WindSpeed'].apply(lambda x: None if x==0 else
x)
    dataG['PPV']=dataG['PPV'].apply(lambda x: None if x==0 else x)
    dataG['PWind']=dataG['PWind'].apply(lambda x: None if x==0 else x)

    dataG['SolarIrr']=dataG.SolarIrr.fillna(dataG.Irr*1000)#conversion en W
    dataG['WindSpeed']=dataG.WindSpeed.fillna(dataG.Wind)
    dataG['PPV']=dataG.PPV.fillna(toPPV(dataG.SolarIrr))
    dataG['PWind']=dataG.PWind.fillna(toPWind(dataG.WindSpeed))
    dataG['PWind']=dataG['PWind'].apply(lambda x: x if x>0 and x<10000 else
0)

    del dataG['Irr']
    del dataG['Wind']

    print("Exporting Data sampled")
    dataG.to_csv(path+'/sampled_files/'+timestep+'.csv')
    print("Data exported")
```