

Registro de mi trabajo en el OHMC

Prueba de script para obtener datos de la api de EPEC

Lo primero que hice fue probar el script que se me había enviado por mail para ir viendo qué información podía obtener de la API de EPEC.

El script me permitía obtener datos de la demanda eléctrica registrada por distintas estaciones transformadoras de la provincia de Córdoba en fechas específicas que se podían especificar al momento de hacer la solicitud a la API

A través de ese script pude obtener la siguiente información:

- los códigos que tienen en EPEC para identificar a las distintas estaciones transformadoras (por ejemplo, *DEM_AGRACI13*)
- los nombres de las estaciones transformadoras (por ejemplo, Mendiolaza 132)
- la demanda eléctrica en MW registrada para dichas estaciones transformadoras en las fechas solicitadas.

Estudio del código del repositorio *estimador-demanda-electrica*

En esta etapa me dediqué a leer todo el código que estaba en el repositorio *estimador-demanda-electrica*. Primero realicé una lectura general del código y después empecé a indagar en más profundidad en una segunda lectura para poder entender cómo funcionaba cada parte del mismo.

Descripción general del repositorio

En el repositorio se encontraban los siguientes archivos de código python:

- *estimar_demanda.py*: en este archivo se encuentra el script principal. Desde aquí se especifican la estación que se quiere consultar, la fecha y el modo (horario o diario) y se llama a los otros scripts que se encuentran en el archivo *api_resources.py*. Este script nos permite obtener los datos predichos de demanda para las 24 horas próximas al día solicitado.
- *api_resources.py*: en este código se encuentran varias funciones que se llaman desde *estimar_demanda.py*, las cuales se utilizan para obtener diferentes datos de la API del OHMC y de la API de EPEC. Por ejemplo, los datos de temperatura, humedad, viento, códigos de identificación de estaciones, demanda eléctrica registrada, datos de actividad económica, etc.
- *redes.py*: este es el código que se utilizó para entrenar las redes neuronales que se utilizan para realizar las predicciones de demanda eléctrica. Se encuentran funciones para configurar las redes, para entrenarlas y para adecuar los datos que necesitan las redes para ser entrenadas.

Prueba del código del repositorio *estimador-demanda-electrica*

En esta etapa estuve tratando de hacer funcionar el código que ya había estudiado previamente.

Dificultades

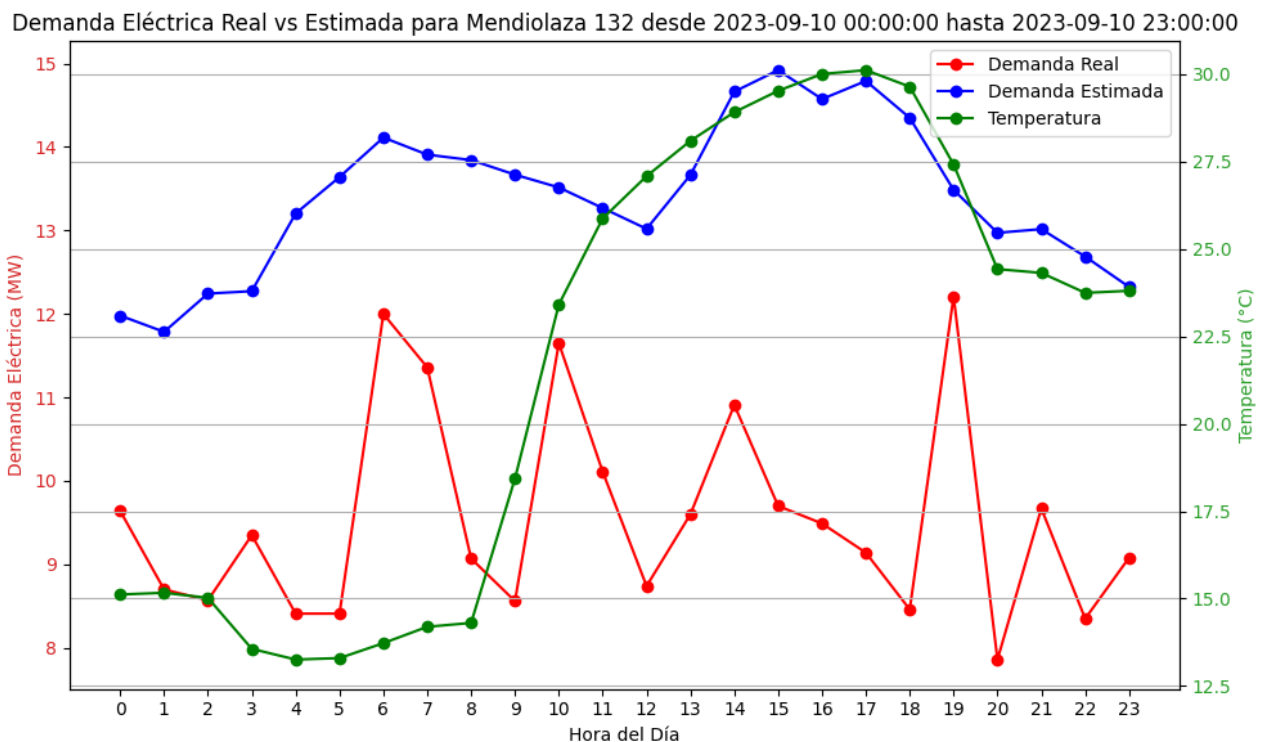
En ese proceso me encontré con dificultades diversas:

- **Problemas para obtener datos desde las APIs:** problemas de autenticación y problemas con el formato que estaba usando para pedir los datos.
- **Problemas de librerías que tuve que instalar:** algunas no las tenía instaladas en mi máquina y otras sí las tenía instaladas pero no eran compatibles las versiones de las mismas que yo tenía instaladas con las que se usaron a la hora de entrenar las redes neuronales utilizadas.
- **Problemas con los modelos ya entrenados:** hay modelos que no están terminados de algunas partes de Córdoba entonces si quería consultar la demanda predicha para esos lugares, el código no encontraba los modelos (porque todavía no existen).

Una vez superadas esas dificultades, el código me proporcionó las demandas predichas para las 24 horas siguientes a la fecha que le ingresé.

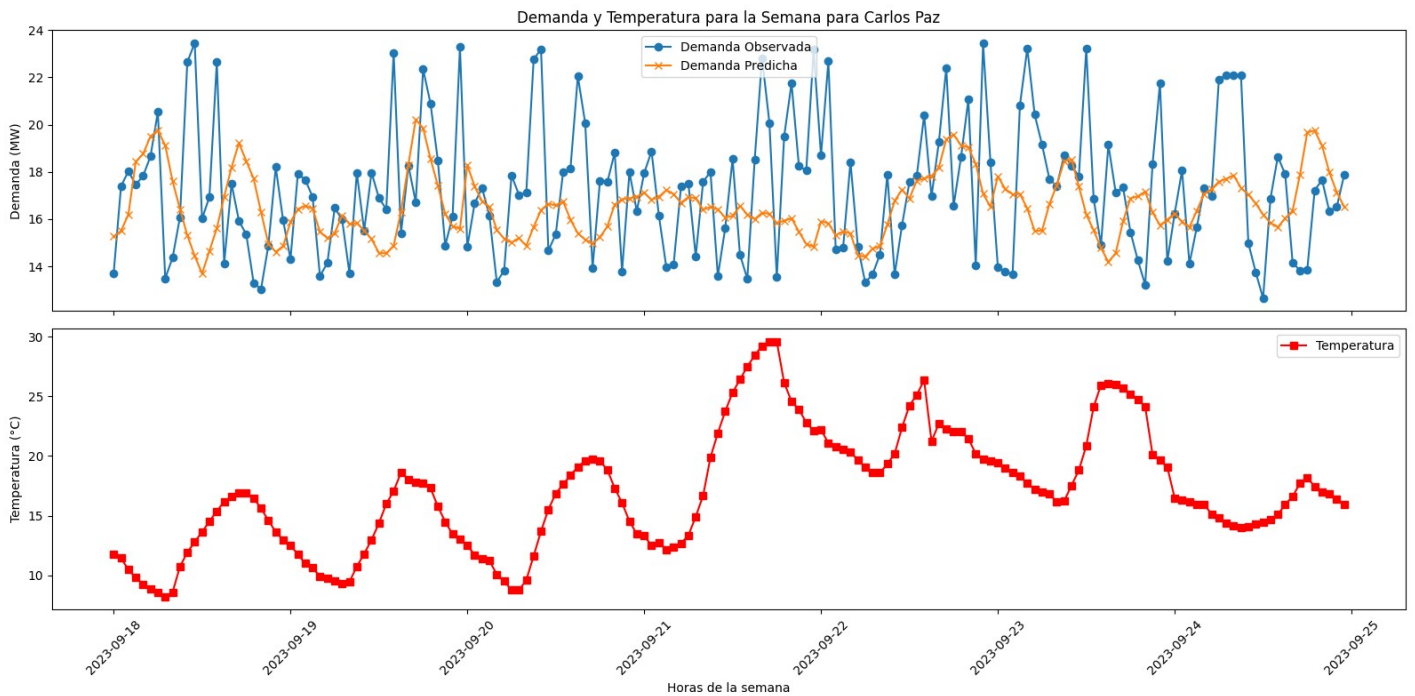
Gráficos de los datos obtenidos

Para poder observar más claramente los datos devueltos por el programa, hice gráficos en los que se pueden observar la demanda predicha para un día en el pasado, la demanda real registrada para ese día en el pasado y la temperatura registrada para ese mismo día. A continuación, se muestra un ejemplo de dichos gráficos:



Después de la obtención de estos gráficos en una franja temporal de un día, me puse a trabajar en la obtención de gráficos en una franja temporal de una semana de diferentes lugares de Córdoba.

A continuación, un ejemplo de los gráficos semanales:



Diccionario de existencia de modelos

Para ir haciendo los gráficos de diferentes lugares de Córdoba, necesitaba saber cuáles de esos lugares de Córdoba tenían modelos y cuáles no, para no tener que andar probando uno por uno cada *uid* de las distintas estaciones transformadoras. Con tal objetivo, escribí un pequeño script que me arma un diccionario con la información de cuáles estaciones transformadoras tienen modelos existentes. A continuación, parte de ese diccionario a modo de ejemplo:

UID de Estación: 900000000000000000110

Nombre de Estación: Morteros 1

Existencia de Modelo: False

UID de Estación: 900000000000000000021

Nombre de Estación: Río Cuarto

Existencia de Modelo: False

UID de Estación: 900000000000000000027

Nombre de Estación: Afisa

Existencia de Modelo: True

Obtención de información de los modelos

Desarrollé un script con el objetivo de obtener información de los archivos pickle en los que se guardaron los modelos de redes neuronales. A modo de ejemplo, se muestran las salidas obtenidas correspondientes a los modelos para Carlos Paz

Pprint

Con el método *pprint* obtuve lo siguiente:

```
2023-09-20 11:54:19.026895: I tensorflow/core/common_runtime/process_util.cc:111
{'Abril': [0.025868303662983448,
  <keras.engine.sequential.Sequential object at 0x7f0576df61d0>],
'Agoosto': [0.029682758855455252,
  <keras.engine.sequential.Sequential object at 0x7f05737edc50>],
'Diario': [0.07406870549734153,
  <keras.engine.sequential.Sequential object at 0x7f056e89fb10>],
'Diciembre': [0.037733017782124574,
  <keras.engine.sequential.Sequential object at 0x7f05c80dbe90>],
'Enero': [0.04320535982115719,
  <keras.engine.sequential.Sequential object at 0x7f05b8bb1d10>],
'Febrero': [0.03910042150194731,
  <keras.engine.sequential.Sequential object at 0x7f05c80cd9d0>],
'Julio': [0.03124751698793618,
  <keras.engine.sequential.Sequential object at 0x7f0573cdbc10>],
'Junio': [0.03221242327306712,
  <keras.engine.sequential.Sequential object at 0x7f0574396f90>],
'Marzo': [0.031698227410497784,
  <keras.engine.sequential.Sequential object at 0x7f05b831f5d0>],
'Mayo': [0.02587990423883395,
  <keras.engine.sequential.Sequential object at 0x7f0574881510>],
'Noviembre': [0.03565262622552105,
  <keras.engine.sequential.Sequential object at 0x7f05d37a0190>],
'Octubre': [0.028912195431566232,
  <keras.engine.sequential.Sequential object at 0x7f0572f5bfd0>],
'Setiembre': [0.031723650688766855,
  <keras.engine.sequential.Sequential object at 0x7f057344a950>],
'scalerD': MinMaxScaler(),
'scalerH': MinMaxScaler()]
```

En la impresión anterior podemos ver que el archivo pickle consta de

- un modelo para cada mes,
- un modelo diario,
- dos escaladores (objetos utilizados comúnmente para escalar los datos antes de alimentarlos a un modelo de red neuronal): uno para el modelo diario y otro para el modelo horario
- un coeficiente correspondiente a cada modelo

Summary

```
Mes: Diciembre  
Model: "sequential_254"
```

Layer (type)	Output Shape	Param #
lstm_507 (LSTM)	(None, 1, 200)	336000
lstm_508 (LSTM)	(None, 120)	154080
dense_507 (Dense)	(None, 60)	7260
dense_508 (Dense)	(None, 24)	1464
Total params: 498,804		
Trainable params: 498,804		
Non-trainable params: 0		

En la impresión anterior podemos ver el summary del modelo correspondiente al mes de Diciembre.

En el mismo podemos visualizar:

- las **capas** que componen el modelo: en este caso 4, dos Long Short-Term Memory y dos densas
- la **cantidad de parámetros (pesos)** de cada capa
- **nombre** único generado automáticamente por Keras (lstm_507)
- **Output Shape:** Indica la forma de salida de cada capa. Por ejemplo, "(None, 1, 200)" significa que la capa produce una salida con una forma de (lotes, secuencia, características).

Información de capas y matrices de pesos

Se muestra la impresión para una capa lstm:

Layer Name: lstm_836

Input Shape: (None, 1, 60)

Output Shape: (None, 40)

Number of Parameters: 16160

Weights for Layer 'lstm_836':

Weight Matrix 1:

```
[[ 0.0517061 -0.10271774  0.16142586 ...  0.09403803  0.01161215
  0.16019535]
 [-0.13012934 -0.0985771  0.04730595 ... -0.02721054 -0.05456793
  0.15195489]
 [ 0.01761824  0.17416228 -0.13023885 ... -0.10804179  0.26557225
 -0.17864378]
 ...
 [ 0.08174995  0.21200669  0.1506116 ... -0.111077 -0.01633638
 -0.02115928]
 [ 0.06193711 -0.13980553  0.0150413 ... -0.15299164  0.04684298
  0.07469747]
 [ 0.04632233  0.03879151 -0.20065151 ...  0.12939979  0.03612133
  0.05273721]]
```

Weight Matrix 2:

```
[[ 0.02870736 -0.00449466  0.13970482 ...  0.0354326 -0.06820517
  0.06457805]
 [ 0.06753191 -0.05849891 -0.07864323 ...  0.02569957 -0.12245804
 -0.09819219]
 [ 0.06594665 -0.02871433  0.0152472 ... -0.06596284  0.05832894
 -0.0111298 ]
 ...
 [ 0.08318278 -0.03959909 -0.10605905 ...  0.04722703 -0.06498064
 -0.08457663]
 [-0.02443106  0.09345431  0.02517675 ... -0.14176087  0.03420343
  0.02170275]
 [ 0.10816718  0.1744341 -0.12845294 ...  0.06533765  0.03120348
 -0.10497423]]
```

Weight Matrix 3:

```
[ 0.02831241  0.05552637  0.0061329  0.09568036  0.13174543  0.02227883
  0.04785749  0.03526322  0.04455858  0.00887446  0.05124194  0.02560772
  0.01753489  0.04029208  0.02702462  0.02774538  0.03080031  0.03288876
  0.02179749  0.04341364  0.05251166  0.08549421  0.00866922  0.02911646
 -0.01507983  0.01931487  0.02438706  0.05795122  0.20559278  0.03313088
  0.00604366  0.02255473 -0.03779866  0.01730093  0.01654872  0.03281313
  0.05277842  0.01128651  0.04725223  0.01393472  1. 1.
  1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1.
  1. 1. 1. 1. 1. 1.
  1. 1. 0.0267358 0.02171625 -0.01966031 -0.01090038
  0.02767416  0.01750599 -0.01879091  0.02164968 -0.02340522 -0.01898091
  0.03259889 -0.02707404 -0.02048539  0.03450866  0.02594724  0.02611421
  0.02507129  0.01205178 -0.01400451 -0.02516558  0.00429585 -0.0288572
 -0.02084251 -0.02458972 -0.01127432  0.02624235 -0.02699085  0.01425057
 -0.01100512 -0.02778159 -0.0119422  0.02040534 -0.02126742 -0.02617841
  0.02721947 -0.02327848 -0.02559346 -0.00542438  0.02822355 -0.02282634
  0.02289294  0.04702287  0.00408652  0.09494297  0.12821496  0.02290001
  0.05387903  0.03532497  0.047156  0.01007266  0.05116784  0.02268287]
```


Lectura de informes

Leí los informes de avances que se fueron entregando a EPEC por parte del grupo que venía trabajando en el proyecto de estimación de la demanda eléctrica anteriormente, con el objetivo de estar al tanto de lo que se vino trabajando hasta ahora.

A continuación, voy a mencionar brevemente partes de los informes que me parecieron relevantes.

Rellenos

El código existente realiza relleno de datos en 3 casos distintos:

1. el dato falta
2. el dato es igual a varios datos anteriores
3. el dato es muy distinto del anterior y del que le sigue (outliers)

Se utilizan dos técnicas para rellenar datos:

1. **promedio** de los datos anterior y posterior al dato a rellenar: en el caso de que falte un dato
2. **método de polinomios spline cúbico**: en el caso de que falten dos datos seguidos. Para realizar este cálculo son necesarios tres datos anteriores y posteriores a los datos a rellenar.

Se estableció como pauta que no se pueden rellenar 3 datos o más porque el relleno introduce un error significativo.

Arquitectura y Topología de Redes Neuronales Artificiales a escala horaria

Se optimizó una arquitectura de Red Neuronal de tipo feedforward constituida por:

- Una capa de entrada con 14 neuronas (1 por cada variable explicativa)
- Dos capas ocultas de 20 neuronas cada una
- Una capa de salida de 1 neurona (Demanda)
- Full connection
- Función de activación: Sigmoide
- Optimización de parámetros: Back propagation momentum
- División de Patrones: Entrenamiento: 70%, Validación: 20%, Test: 10%
- Presentación de patrones: Aleatorios
- Control de Overfitting
- Inicialización de pesos: Aleatorios

Arquitectura y topología de Redes Neuronales a escala diaria

Se optimizó una arquitectura de Red Neuronal de tipo feedforward constituida por:

- Una capa de entrada de 13 Neuronas

- Una capa oculta de 13 Neuronas
- Una capa de Salida de una Neurona (Demanda)
- Full connection
- Función de activación: Sigmoide
- Optimización de parámetros: Back propagation momentum
- División de Patrones: Entrenamiento: 70%, Validación: 20%, Test: 10%
- Presentación de patrones: Aleatorios
- Control de Overfitting
- Inicialización de pesos: Aleatorios

Justificación

La elección de esta arquitectura se corresponde con la intención de captar no solo la correlación a lo largo de las horas de cada día, sino también la correlación con la misma hora del día anterior. La primera en este caso se obtiene a través de las capas ocultas LSTM y la segunda se obtiene a través de la capa oculta densa.

Este esquema, fué el que mejor desempeño tuvo en la optimización realizada en la primera etapa con el entrenamiento de algunas ET de la Zona A.

Para el caso Horario se determinó la conveniencia de entrenar una red para cada mes del año así se evitaba las implicancias de ciclos en las series de tiempo que involucrarían muchos más parámetros

Inputs

Los inputs de la RNA fueron:

- el pronóstico de WRF de la temperatura de los dos días anteriores más la actual (48 inputs),
- la demanda histórica de dos días anteriores más la actual (48 inputs),
- el pronóstico de WRF de temperatura, humedad, presión atmosférica, dirección e intensidad del viento de las 24 horas del día al que se le estimará la demanda (120 inputs),
- el día de la semana,
- el estimador mensual de actividad económica brindado por el INDEC (un input cada uno).

En total se utilizaron 218 inputs. Este esquema también surgió de la investigación realizada en la primera etapa y presentada en el Congreso Latinoamericano de Sociedades de Estadísticas (CLATSE) en Montevideo, Uruguay durante octubre de 2021.

Datos Meteorológicos

La elección de pronósticos de WRF como datos meteorológicos se justifica con el hecho de que no se encontraron diferencias significativas en la comparación de poblaciones de errores al entrenar las redes con datos del SMN, del OHMC o datos de WRF para los modelos de paso horario.

Este hecho facilita la obtención de datos dado que en el interior de la Provincia, las estaciones meteorológicas no siempre se corresponden con ET, mientras que WRF estima datos para cualquier ubicación de la provincia.

Herramienta estadística

Conforme a la investigación realizada se utilizó RNA tal como lo sugieren publicaciones científicas del año 2020. (Hammad, Mahmoud A.; Jereb, Borut; Rosi, Bojan y Dragan, Dejan “Methods and Models for Electric Load Forecasting: A Comprehensive Review” como ejemplo).

Modelos con período diario

Los modelos para predicciones diarias se realizaron para todo el año sin distinción de mes. Es decir que sólo hay un modelo por cada Sea. Esto no es recomendable, pero la cantidad de datos históricos requerida para esta estimación no estaba disponible.

Medición de errores

Se utilizó el Error Cuadrático Medio de Predicción (ECMP), que es una métrica estadística ampliamente utilizada en el campo del pronóstico y la predicción.

Esta es la fórmula para calcular el ECMP:

$$ECMP = \sum (\text{Valor_real} - \text{Valor_pronosticado})^2 / N$$

Donde:

- Σ representa la suma de los errores al cuadrado para las 24 predicciones.
- Valor_real es el valor real de la demanda de energía en una hora determinada.
- Valor_pronosticado es el valor pronosticado por el modelo para la misma hora.
- N es el número total de pronósticos (en este caso, la cantidad de pronósticos que se hayan registrado para este cálculo, como ejemplo, 24 horas de 30 días de 2 meses: 1440).

Investigación

Durante la lectura de los informes y durante el estudio de las redes neuronales existentes, me fui encontrando con términos con los que no estaba familiarizado, por lo que me puse a investigarlos. Dejo en esta sección parte de esa investigación.

Forma de salida o entrada (lotes, secuencia, características)

La notación "Input Shape: (None, 1, 60)" en el contexto de una red neuronal se refiere a la forma o dimensiones de los datos de entrada que se esperan en una capa de la red neuronal.

- **Lotes (Batch):** Este valor representa la cantidad de ejemplos de datos que se procesan en paralelo en cada paso de tiempo.

- **Secuencia:** se refiere a la longitud de la secuencia o serie temporal que se está procesando en cada paso. En el caso de datos de series temporales diarias, cada paso de tiempo representa un día y esta dimensión podría ser igual a 7.
- **Características (Features):** representa la cantidad de características o variables que se están utilizando en cada paso de tiempo. Por ejemplo, si estamos prediciendo el clima y utilizamos temperatura y humedad como características de entrada, esta dimensión sería igual a 2

Un valor de "None" significa un valor que no está fijo y puede variar dependiendo de cómo se configure el modelo.

Long Short-Term Memory

Una capa *LSTM* es una unidad de procesamiento que es especialmente eficaz para trabajar con datos secuenciales gracias a su capacidad para capturar relaciones temporales a largo plazo.

Capas densas (completamente conectadas)

Las capas densas son capas fundamentales en una red neuronal artificial que realizan operaciones de ponderación y suma de entradas, seguidas de funciones de activación. Estas capas permiten que la red aprenda representaciones complejas y no lineales de los datos, lo que las hace esenciales en una variedad de aplicaciones de aprendizaje automático y redes neuronales.

Las capas densas tienen las siguientes características:

- **Conexiones Densas:** cada neurona (o nodo) está conectada a todas las neuronas de la capa anterior y a todas las neuronas de la capa siguiente.
- **Operaciones de Peso y Sesgo:** Cada conexión entre neuronas tiene un peso asociado. Durante el proceso de entrenamiento, estos pesos se ajustan mediante algoritmos de optimización para aprender patrones en los datos. Además de los pesos, cada neurona tiene un sesgo (bias) que se suma a la salida ponderada de las conexiones. Esto permite a la red aprender representaciones no lineales de los datos.
- **Funciones de Activación:** Esta función introduce no linealidad en la red y permite que aprenda relaciones y patrones más complejos en los datos. Ejemplos de funciones de activación: función *sigmoide*, función *ReLU* y la función *tangente hiperbólica*.
- **Última Capa en una Red:** En muchas arquitecturas de redes neuronales, la capa densa final es la capa de salida que produce las predicciones o clasificaciones finales.

Red Neuronal de tipo feedforward

En este tipo de red la información fluye en una dirección, de la capa de entrada a la capa de salida, sin bucles o conexiones retroalimentadas.

Características:

- **Capas:** consta de al menos tres capas de neuronas: la capa de entrada, una o más capas ocultas y la capa de salida. La capa de entrada recibe las características o datos de entrada,

las capas ocultas realizan transformaciones y cálculos intermedios, y la capa de salida produce las salidas finales.

- **Conexiones:** se producen solo entre capas consecutivas. Las neuronas en una capa no están conectadas directamente entre sí ni retroalimentadas a capas anteriores.
- **Aprendizaje:** Son entrenadas utilizando algoritmos de aprendizaje supervisado, como el descenso de gradiente, para ajustar los pesos y sesgos de las conexiones de manera que minimicen el error entre las salidas predichas y las salidas reales.

Back propagation momentum

Backpropagation momentum es una técnica que combina la retropropagación (para ajustar pesos en una red neuronal) con el concepto de momento (para acelerar la convergencia) en el proceso de optimización de parámetros durante el entrenamiento de una red neuronal.

- **Backpropagation (retropropagación):** Es un algoritmo de entrenamiento comúnmente utilizado en redes neuronales para ajustar los pesos y los sesgos de la red en función del error calculado entre las salidas predichas y las salidas reales. La retropropagación calcula este error y propaga gradualmente las correcciones hacia atrás a través de la red, ajustando los pesos en todas las capas.
- **Momentum (momento):** El "momento" es un concepto utilizado en la optimización del descenso de gradiente para ayudar a acelerar la convergencia del algoritmo de entrenamiento. Agregar momento implica tener en cuenta el gradiente de los pasos anteriores para influir en la dirección y magnitud de los pasos futuros. Esto puede ayudar a superar los mínimos locales y acelerar la convergencia hacia un mínimo global en el proceso de entrenamiento.

Lectura del paper “Methods and Models for Electric Load Forecasting: A Comprehensive Review”

Durante la lectura de los informes encontré que se usaba este paper como justificación de la elección de redes neuronales para realizar la predicción de la demanda eléctrica así que decidí leerlo.

En este paper se realiza una descripción de varios métodos para realizar predicciones de demanda eléctrica según el tipo de predicción:

- **Long-term load forecasting:** de 1 a 20 años en el futuro
- **Medium-term load forecasting:** de 1 semana a 1 año de predicción
- **Short-term load forecasting:** de 1 hora a 1 semana
- **Ultra/very short-term load forecasting:** algunos minutos a 1 hora de predicción

El caso nuestro es el de *Short-term load forecasting*.

Algunas citas que justifican el uso de redes neuronales son:

- *“As for short-term predictions, machine learning or artificial intelligence-based models such as Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Fuzzy logic are favored”*
- *“The ANNs have been successfully applied in many different areas, especially for forecasting and classification purposes. ANNs models have been used and studied intensively as a tool to be used for electric load forecasting and gained huge popularity in the last few decades”*
- *“The study also analyzed some patterns in the use of these models. Some of them are more appropriate and preferred for electric load forecasts such as regression analysis based models and artificial neural networks (ANN), which are the most utilized models in electricity predictions. In this scope, the artificial neural networks (ANN) models are mainly employed for short-term predictions where electricity and power consumption patterns are more complicated.”*

Consulta a GPT

Una vez leído el paper fui a consultarle a gpt sobre lo que había leído. A continuación, fragmentos del chat:

- yo:

“que es lo mejor para predecir demanda electrica de las proximas 24 horas?”

- gpt:

“ En mi opinión, el uso de algoritmos de aprendizaje automático, como las redes neuronales recurrentes (RNN) o modelos de series temporales, ha demostrado ser efectivo para predecir la demanda eléctrica a corto plazo. Estos modelos pueden tener en cuenta patrones temporales y variaciones en los datos para hacer predicciones precisas. También es importante considerar fuentes de datos en tiempo real, como pronósticos meteorológicos, ya que el clima puede influir significativamente en la demanda eléctrica.”

- yo:

“las redes neuronales feed forward son utiles?”

- gpt:

“Sí, las redes neuronales feedforward (también conocidas como redes neuronales multicapa) son útiles en una variedad de aplicaciones de aprendizaje automático, incluyendo la predicción de demanda eléctrica y muchas otras tareas.

Sin embargo, es importante ajustar la arquitectura de la red, el número de capas y neuronas, y los hiperparámetros adecuados para el problema específico que estás abordando. En algunas situaciones, otras arquitecturas de redes neuronales, como las redes recurrentes o las redes convolucionales, pueden ser más apropiadas.”

- yo:

“pero que es mejor, las feed forward o las recurrentes?”

- gpt:

“La elección entre redes neuronales feedforward y redes neuronales recurrentes depende en gran medida de la naturaleza del problema que estás tratando de resolver. Ambas tienen sus ventajas y desventajas, y la "mejor" opción varía según el contexto. Aquí hay algunas consideraciones clave:

- *Redes Neuronales Feedforward (FNN): Son adecuadas para tareas donde no es necesario considerar la secuencia o el orden de los datos, como el reconocimiento de imágenes o el procesamiento de datos tabulares. Tienden a ser más simples de entrenar y menos propensas a problemas de convergencia. No tienen memoria a largo plazo, lo que puede ser una desventaja en problemas secuenciales que dependen de eventos pasados a largo plazo.*
- *Redes Neuronales Recurrentes (RNN): Son ideales para tareas donde la secuencia de datos es crucial, como el procesamiento de texto, traducción automática o predicción de series temporales, como la demanda eléctrica. Tienen memoria a largo plazo y pueden mantener información sobre eventos pasados a lo largo de la secuencia. Pueden ser más complejas de entrenar debido a problemas como el desvanecimiento del gradiente, pero existen variantes como LSTM y GRU que abordan estos problemas.*

En resumen, si estás trabajando en un problema secuencial, como la predicción de la demanda eléctrica a lo largo del tiempo, las redes neuronales recurrentes, especialmente las LSTM o GRU, a menudo son la elección preferida debido a su capacidad para modelar dependencias temporales. Sin embargo, la elección específica depende de la naturaleza precisa de tu tarea y de la cantidad de datos disponibles para el entrenamiento. En algunos casos, también se pueden explorar arquitecturas híbridas que combinan lo mejor de ambos mundos.”

A partir de esta última respuesta obtenida, me di cuenta de que las redes que estamos utilizando no son de tipo *feedforward* como decía en el informe, sino que son redes neuronales recurrentes (me di cuenta por la presencia de capas lstm en los modelos observados anteriormente).

Documentación de funciones

Durante mi trabajo en la obtención de gráficos de demanda predicha vs observada, fui generando funciones para obtener los datos necesarios para hacer esos gráficos:

- Modelos existentes,
- datos meteorológicos para una semana,
- datos de demanda observada para una semana,
- correr el script de predicción de demanda para un día 7 veces (para obtener la demanda predicha para una semana),
- etc.

Documenté esas funciones que generé siguiendo el formato utilizado para las funciones ya existentes.

Encapsulación de funciones de print_pickles

Metí adentro de funciones parte del script que usé para obtener información sobre los modelos de redes neuronales existentes para que el script quedase más ordenado y para que se pudiesen separar las partes del mismo que sirven para obtener distintos datos sobre los modelos.

App para la obtención de datos de apis

Se me asignó la tarea de desarrollar una aplicación que permita obtener los datos de las apis de EPEC y del OHMC de una manera más sencilla o práctica.

Decidí dividir el trabajo en 3 partes:

1. Desarrollo de aplicación para obtener datos de la api de EPEC
2. Desarrollo de aplicación para obtener datos de la api del OHMC
3. Fusión de las dos apps en una

Desarrollo de aplicación para obtener datos de la api de EPEC

Desarrollé una aplicación en Python que retorna los datos de demanda registrados por EPEC para las distintas estaciones transformadoras que tienen funcionando.

La app recibe los siguientes argumentos por línea de comandos:

- **grupo:** la ET que se quiere consultar
- **desde:** la fecha de inicio de consulta de datos de demanda
- **hasta:** la fecha de fin de consulta de datos de demanda

En un principio la app retornaba los datos tal cual los devolvía la api:

- en formato JSON y
- desordenados

Ejemplo de salida:

```
{  
  "grupo": "DEM_MCRISTNVA132",  
  "descripcion": "DEMANDA MONTE CRISTO NUEVA 132 KV",  
  "anio": "2023",  
  "mes": "10",  
  "dia": "24",  
  "hora": "24",  
  "minuto": "00",  
  "demanda": 44.98  
},
```

```
{
  "grupo": "DEM_MENDIO132",
  "descripcion": "DEMANDA ET. MENDIOLAZA 132 KV",
  "anio": "2023",
  "mes": "10",
  "dia": "24",
  "hora": "20",
  "minuto": "00",
  "demanda": 12.73
}
```

Pude observar que este formato no es muy cómodo para trabajar con los datos, y además pude observar que **la api de epec no devuelve los datos ordenados cronológicamente**.

Por esta razón, me puse a trabajar en que la app devolviera los datos en una forma más conveniente para trabajar con ellos y de forma ordenada cronológicamente.

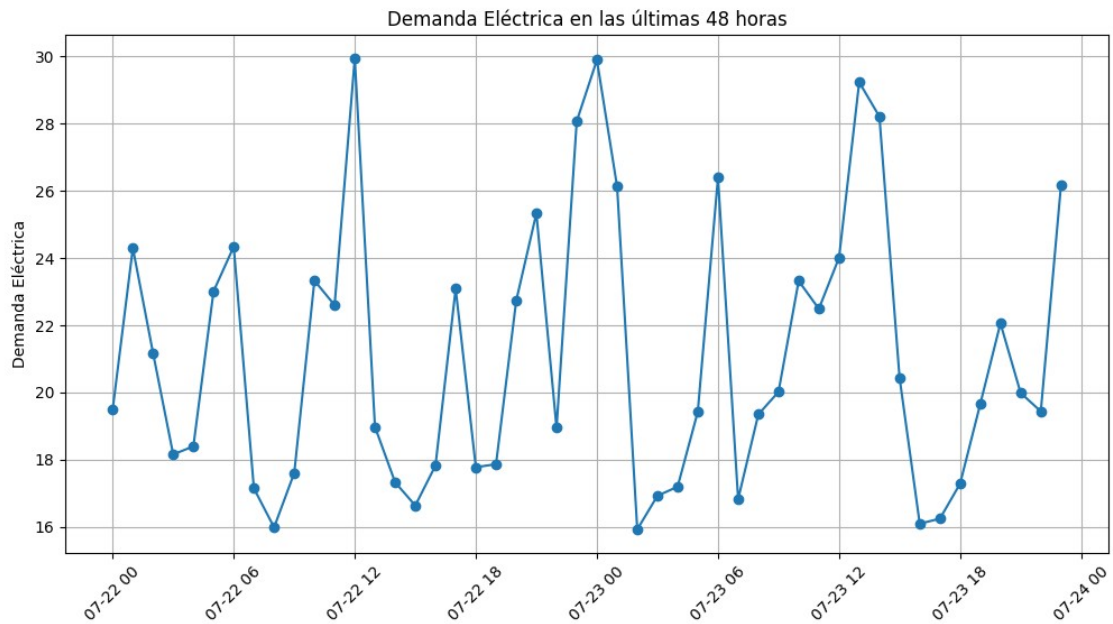
Con esos cambios, ahora la app espera un argumento más por línea de comandos llamado **format**. Este argumento le especifica a la función si tiene que imprimir los datos en el formato por defecto o en el formato ordenado.

Ejemplo de cómo imprime la app los datos formateados:

Grupo	Fecha y hora	Demanda
-----	-----	-----
AGRACIN132	03/06/2023 10:00	8.4
AGRACIN132	03/06/2023 11:00	8.28
BAR_CRD_CBOSQUE_13_1	03/06/2023 10:00	0
BAR_CRD_CBOSQUE_13_1	03/06/2023 11:00	0
BAR_CRD_CBOSQUE_13_2	03/06/2023 10:00	6e-05
BAR_CRD_CBOSQUE_13_2	03/06/2023 11:00	6e-05
BAR_CRD_DELICIA_13_1	03/06/2023 10:00	0.00209
BAR_CRD_DELICIA_13_1	03/06/2023 11:00	0.00195
BAR_CRD_DELICIA_13_2	03/06/2023 10:00	0.00061
BAR_CRD_DELICIA_13_2	03/06/2023 11:00	0.00057
BAR_CRD_MANANTI_13_1	03/06/2023 10:00	0.00045
BAR_CRD_MANANTI_13_1	03/06/2023 11:00	0.00044
BAR_CRD_MANANTI_13_2	03/06/2023 10:00	3.72292
BAR_CRD_MANANTI_13_2	03/06/2023 11:00	3.84313
DEM_25MAY013	03/06/2023 10:00	4.31139
DEM_25MAY013	03/06/2023 11:00	4.4624
DEM_ACABRAL132	03/06/2023 10:00	158.353
DEM_ACABRAL132	03/06/2023 11:00	160.125

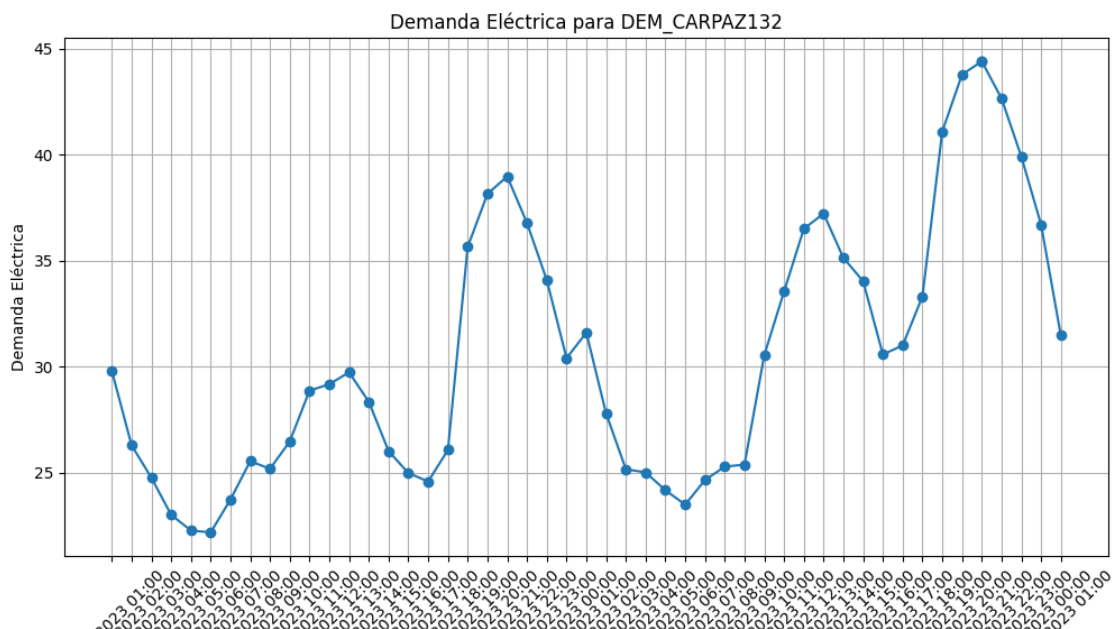
Gráfico de datos de demanda observada

Antes de ordenar los datos que obtenía de la api de epec (porque no sabía que estaban desordenados), estaba obteniendo gráficos como este:



Analizando estos gráficos, pude observar que no tenían mucho sentido con lo que se esperaría ver referido a los consumos energéticos de las personas.

Ahora, con los datos ya ordenados obtengo gráficos como estos:



los cuales tienen más sentido