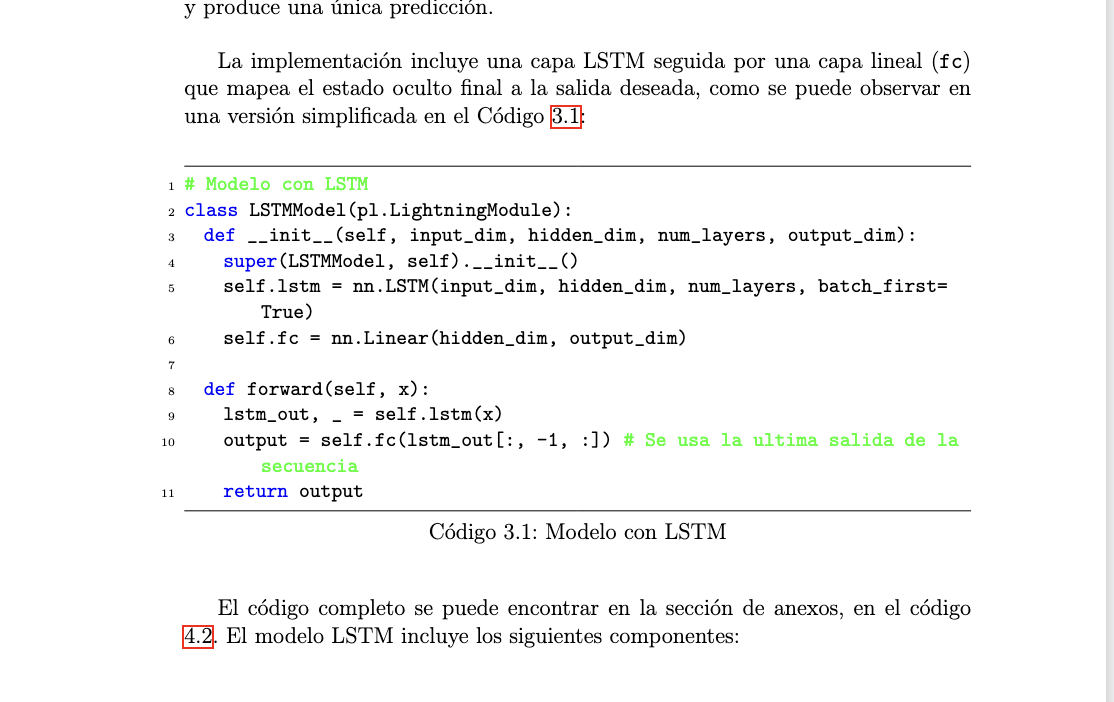
Para construir un **intervalo de confianza** en base a las predicciones de tu modelo y al MSE obtenido, primero es necesario entender cómo se calculan estos intervalos. Un intervalo de confianza para las predicciones refleja el rango en el cual esperamos que caigan los valores reales en función de las predicciones del modelo, dado un nivel de confianza (por ejemplo, 95%).

**Teoría del Intervalo de Confianza:**

* Para una predicción y^iy^​i​, el intervalo de confianza (IC) a un nivel de confianza αα generalmente sigue esta fórmula:IC=y^i±Zα/2⋅MSEnIC=y^​i​±Zα/2​⋅nMSE​​ Donde:
  + y^iy^​i​ es el valor predicho.
  + Zα/2Zα/2​ es el valor crítico de una distribución normal estándar (por ejemplo, 1.96 para un intervalo del 95%).
  + MSEMSE es el error cuadrático medio obtenido en el conjunto de validación.
  + nn es el número de observaciones o la ventana sobre la cual calculas los intervalos.

Luego de imagen 3.1



El modelo LSTM incluye los siguientes componentes:

\begin{itemize}

\item \textbf{Capa LSTM:} Definida como \texttt{self.lstm}, donde \texttt{input\\_dim} corresponde al número de características en cada paso de tiempo (en este caso, es 1), \texttt{hidden\\_dim} define el tamaño de las celdas de memoria ocultas, y \texttt{num\\_layers} indica la cantidad de capas LSTM utilizadas. Esta capa permite que la secuencia de datos fluya a través del tiempo y pueda aprender dependencias temporales a largo plazo.

\item \textbf{Capa Lineal:} \texttt{self.fc} toma como entrada el estado oculto final de la secuencia (última salida) y lo transforma en la salida deseada. En este caso, \texttt{output\\_dim} corresponde a una única predicción.

\item \textbf{Método \texttt{forward()}:} Define cómo el modelo procesa los datos de entrada. Primero, se aplica la capa LSTM sobre la secuencia de entrada y, luego, la salida oculta final (\texttt{lstm\\_out[:, -1, :]}) se pasa a la capa lineal (\texttt{fc}) para generar la predicción. Esto permite que el modelo tome en cuenta toda la información acumulada durante la secuencia para realizar una predicción precisa.

\end{itemize}

Para encontrar el tamaño de ventana adecuado que permita capturar la periodicidad de las señales de demanda energética de manera experimental, se evaluaron diferentes tamaños de ventana en un rango de 1 a 48, que corresponden a horas. Este proceso se realizó mediante una función (\texttt{evaluate\\_window\\_size}) que crea un \texttt{DataModule} con los datos de entrenamiento, validación y prueba. La implementación de esta función se puede ver en el Código \ref{lst:evaluate\_window}:

\vspace{0.5 cm}

\begin{lstlisting}[language=Python, caption=Evaluar el tamaño de la ventana, label={lst:evaluate\_window}]

# Evaluar el tamaño de la ventana

def evaluate\_window\_size(train\_data, val\_data, test\_data, window\_size):

data\_module = DemandDataModule(train\_data, val\_data, test\_data, window\_size)

input\_dim = 1

hidden\_dim = 64

num\_layers = 2

output\_dim = 1

model = LSTMModel(input\_dim=input\_dim, hidden\_dim=hidden\_dim, num\_layers=num\_layers, output\_dim=output\_dim)

# Definir el entrenador

trainer = pl.Trainer(max\_epochs=10, logger=False, enable\_checkpointing=False)

# Entrenar el modelo

trainer.fit(model, data\_module)

# Acceder a las métricas después del entrenamiento

train\_loss = trainer.callback\_metrics.get('train\_loss')

val\_result = trainer.validate(model, data\_module)

return train\_loss.item() if train\_loss else None, val\_result[0]['val\_loss'], model, data\_module

\end{lstlisting}

El usar capas recurrentes para procesar la secuencia de datos significa procesar información de manera secuencial para retener dependencias a largo plazo. Este proceso puede limitar la cantidad de capas debido a problemas de "desvanecimiento del gradiente", pues, a pesar de que al aumentar el número de capas se tiene una mayor capacidad para aprender dependencias temporales complejas \cite{staudemeyer2019}, agregar muchas puede traducirse en problemas de overfitting (sobreajuste) y altos costos computacionales. Es por esto que, para mantener un equilibrio entre capacidad de modelado y eficiencia, se ha optado por 2 capas.

El parámetro hidden$\\_$dim = 64 hace referencia al tamaño de las celdas de memoria internas que permiten aprender las dependencias temporales. Este valor fue considerado como un tamaño suficientemente representativo para la complejidad del problema.

A screenshot of a computer program

Description automatically generated

El modelo TCN incluye los siguientes componentes:

\begin{itemize}

\item \textbf{Capas Convolucionales con Dilatación:} Definidas en un bucle que crea tantas capas como \texttt{num\\_layers} especificados. Cada capa aplica convoluciones 1D con un \texttt{kernel\\_size} determinado y un \texttt{dilation\\_size} que se incrementa exponencialmente (\texttt{2 \*\* i}). Esto permite a TCN capturar dependencias a largo plazo sin requerir un procesamiento secuencial.

\item \textbf{Activación y Dropout:} Después de cada convolución se aplica una función de activación ReLU seguida de una capa de Dropout para evitar el sobreajuste. Esto se realiza mediante \texttt{nn.ReLU()} y \texttt{nn.Dropout(dropout)}.

\item \textbf{Capa Lineal:} Similar al modelo LSTM, la capa lineal \texttt{self.fc} toma la salida del último paso de la secuencia para generar la predicción.

\item \textbf{Método \texttt{forward()}:} Dado que la convolución 1D en PyTorch espera entradas con la forma (batch\\_size, channels, sequence\\_length), es necesario transponer la entrada antes de aplicar la TCN y después del procesamiento convolucional (\texttt{x = x.transpose(1, 2)} y \texttt{tcn\\_out = tcn\\_out.transpose(1, 2)}). La salida luego pasa por la capa lineal para obtener la predicción final.

\end{itemize}

Similar a la implementación del modelo LSTM, se procedió a evaluar diferentes tamaños de ventana con el modelo TCN, como se puede observar en el Código \ref{lst:evaluate\_tcn\_window}.

\vspace{0.5cm}

\begin{lstlisting}[language=Python, caption=Evaluar el tamaño de la ventana, label={lst:evaluate\_tcn\_window}]

# Evaluar el tamano de la ventana

def evaluate\_window\_size(train\_data, val\_data, test\_data, window\_size):

data\_module = DemandDataModule(train\_data, val\_data, test\_data, window\_size)

input\_dim = 1 # Cada valor de entrada es un valor por paso de tiempo

hidden\_dim = 64 # Numero de canales ocultos de la TCN

num\_layers = 3 # Numero de capas convolucionales

output\_dim = 1 # Queremos predecir un solo valor (proxima demanda)

# Instancia el modelo TCN

model = TCNModel(input\_dim=input\_dim, hidden\_dim=hidden\_dim, output\_dim=output\_dim, num\_layers=num\_layers)

\end{lstlisting}

En TCN, el aumento de capas ayuda a capturar dependencias de largo alcance y, gracias a su capacidad para realizar operaciones en paralelo, agregar más capas resulta más eficiente que en el caso de LSTM \cite{ankle2020, oshea2015}. Por eso se usa un num$\\_$layers = 3, aumentando su capacidad de representar características espaciales y temporales sin los problemas comunes de las RNN como el desvanecimiento del gradiente.

El modelo TCN fue diseñado para capturar dependencias temporales, tanto a corto como a largo plazo, en una secuencia de datos. Una de las principales características del modelo TCN es el uso de \textbf{convoluciones dilatadas}, que permiten que la red pueda tener una ventana de percepción más amplia con el aumento de capas, sin necesidad de aumentar el número de parámetros excesivamente \cite{oshea2015}.

\begin{itemize}

\item \textbf{Capas Convolucionales con Dilatación:}

Las capas convolucionales de las TCN están diseñadas para aplicar convoluciones sobre los datos, lo que implica realizar operaciones que permiten extraer características específicas de la secuencia \cite{ankle2020}. En el modelo TCN, se utilizan convoluciones \textbf{1D}, ya que se trabaja con datos unidimensionales (como las series temporales).

En el Código \ref{lst:tcn\_model}, se puede observar que las capas convolucionales se definen dentro de un bucle. Este bucle se ejecuta \texttt{num\\_layers} veces, con el objetivo de construir múltiples capas para capturar características más complejas de la secuencia. Aquí es importante entender la noción de \textbf{dilatación}:

\begin{itemize}

\item La \textbf{dilatación} se refiere a la separación entre los valores sobre los que se aplica la convolución. Por ejemplo, una dilatación de 1 significa que cada valor es adyacente al siguiente, mientras que una dilatación de 2 significa que se omite un valor entre cada aplicación. En el código, el valor de dilatación se establece como \texttt{2 \*\* i}, donde \texttt{i} es el número de la capa, lo cual hace que la dilatación aumente exponencialmente (1, 2, 4, etc.). Esto permite que la red capture relaciones a largo plazo de manera más eficiente.

\item La \textbf{convolución dilatada} permite que la TCN tenga una ventana de percepción mayor sin necesidad de aumentar mucho la cantidad de parámetros o el tamaño del kernel. Es decir, el modelo puede ver una mayor porción de la secuencia a medida que aumenta el número de capas y la dilatación, lo cual resulta fundamental para capturar patrones de larga duración en las series temporales \cite{ankle2020}.

\end{itemize}

En cada capa, se especifica un \texttt{padding\\_size} que garantiza que la salida tenga el mismo tamaño que la entrada, lo cual es útil para que las diferentes capas convolucionales se puedan encadenar sin problemas.

\item \textbf{Activación y Dropout:}

Después de cada operación de convolución, se aplica una \textbf{función de activación ReLU} (\texttt{nn.ReLU()}) para introducir no linealidades en el modelo. Esto es importante para que el modelo pueda aprender relaciones complejas y no lineales en los datos.

Luego, se aplica una \textbf{capa de Dropout} (\texttt{nn.Dropout(dropout)}) con el fin de evitar el sobreajuste del modelo. El Dropout desactiva aleatoriamente algunas de las unidades durante el entrenamiento, lo cual obliga al modelo a ser más robusto y a no depender demasiado de características específicas. El valor de \texttt{dropout} se suele establecer en un valor bajo (en este caso, 0.2) para evitar la desactivación excesiva de unidades.

\item \textbf{Capa Lineal:}

Similar al modelo LSTM, la \textbf{capa lineal} (\texttt{self.fc}) toma la salida del último paso de la secuencia para generar la predicción. Esta capa es la responsable de convertir las características extraídas por las capas convolucionales en un valor de salida que corresponda a la predicción deseada, como la demanda energética para el próximo período.

\item \textbf{Método \texttt{forward()}:}

La función \texttt{forward()} define cómo se procesan los datos en el modelo. Es importante notar que, dado que la convolución 1D en PyTorch espera que los datos de entrada tengan la forma \texttt{(batch\\_size, channels, sequence\\_length)}, es necesario transponer la entrada (\texttt{x = x.transpose(1, 2)}) antes de aplicar la TCN y, luego de aplicar las capas convolucionales, se vuelve a transponer la salida (\texttt{tcn\\_out = tcn\\_out.transpose(1, 2)}).

Finalmente, la salida se pasa por la capa lineal para obtener la predicción final de la serie temporal.

\end{itemize}

A screenshot of a computer program

Description automatically generated

El modelo híbrido incluye los siguientes componentes:

\begin{itemize}

\item \textbf{Capas Convolucionales con Dilatación (TCN):} Definidas mediante \texttt{nn.Conv1d}, estas capas se encargan de capturar patrones temporales mediante el uso de convoluciones con dilatación exponencial (\texttt{2 \*\* i}). Esta dilatación permite que el modelo observe relaciones a mayor distancia temporal sin necesidad de aumentar el número de parámetros. Las capas TCN están dispuestas de manera secuencial (\texttt{nn.Sequential}) y cada capa está seguida por una función de activación ReLU y un Dropout para evitar el sobreajuste.

\item \textbf{Capa LSTM:} La salida de la TCN se alimenta a una capa LSTM (\texttt{self.lstm}) que se encarga de aprender dependencias secuenciales en las características extraídas por la TCN. La LSTM se configura con un \texttt{hidden\\_size} de 64, que permite manejar información a nivel secuencial.

\item \textbf{Capa Lineal (Fully Connected):} Finalmente, la capa de salida (\texttt{self.fc}) convierte la última salida de la LSTM en la predicción requerida. En este caso, se predice un solo valor que corresponde a la demanda de energía futura.

\item \textbf{Método \texttt{forward()}:} El método \texttt{forward()} define cómo los datos fluyen a través del modelo. Primero, los datos pasan por la TCN, luego la salida se transfiere a la LSTM, y finalmente la última salida de la secuencia de la LSTM se convierte en la predicción mediante la capa lineal.

\end{itemize}

El modelo híbrido se diseñó con la intención de combinar la capacidad de la TCN de capturar patrones a largo plazo mediante convoluciones con la capacidad de la LSTM de manejar dependencias secuenciales complejas. Esto permite al modelo captar de manera más robusta las variaciones en la demanda de energía.

La función \texttt{evaluate\\_window\\_size} se utiliza para probar diferentes tamaños de ventana con el objetivo de identificar el que mejor captura la periodicidad de la demanda de energía. Esta función crea un módulo de datos (\texttt{DemandDataModule}) con los conjuntos de datos de entrenamiento, validación y prueba. La implementación de esta función se puede observar en el Código \ref{lst:evaluate\_hybrid\_window}:

\vspace{0.5cm}

\begin{lstlisting}[language=Python, caption=Evaluar el tamaño de la ventana en el modelo híbrido, label={lst:evaluate\_hybrid\_window}]

# Evaluar el tamaño de la ventana

def evaluate\_window\_size(train\_data, val\_data, test\_data, window\_size):

data\_module = DemandDataModule(train\_data, val\_data, test\_data, window\_size)

input\_dim = 1 # Cada valor de entrada es un valor por paso de tiempo

tcn\_hidden\_dim = 64 # Número de canales ocultos de la TCN

lstm\_hidden\_dim = 64 # Dimensión oculta de la LSTM

num\_tcn\_layers = 3 # Número de capas convolucionales

num\_lstm\_layers = 1 # Número de capas de la LSTM

output\_dim = 1 # Queremos predecir un solo valor (próxima demanda)

# Instancia el modelo híbrido

model = HybridModel(input\_dim=input\_dim, tcn\_hidden\_dim=tcn\_hidden\_dim, lstm\_hidden\_dim=lstm\_hidden\_dim, output\_dim=output\_dim, num\_tcn\_layers=num\_tcn\_layers)

# Definir el entrenador

trainer = pl.Trainer(max\_epochs=10, logger=False, enable\_checkpointing=False)

# Entrenar el modelo

trainer.fit(model, data\_module)

# Acceder a las métricas después del entrenamiento

train\_loss = trainer.callback\_metrics.get('train\_loss')

val\_result = trainer.validate(model, data\_module)

return train\_loss.item() if train\_loss else None, val\_result[0]['val\_loss'], model, data\_module

\end{lstlisting}

En la función anterior se establecen ciertos valores específicos para la configuración del modelo:

\begin{itemize}

\item \textbf{\texttt{input\\_dim} = 1:} Cada valor de entrada es un único valor de la serie temporal (es decir, la demanda energética en un instante de tiempo).

\item \textbf{\texttt{tcn\\_hidden\\_dim} = 64:} Esta es la dimensión oculta de las capas convolucionales (TCN). Se seleccionó un valor de 64 como un compromiso entre capacidad de representación y eficiencia computacional.

\item \textbf{\texttt{lstm\\_hidden\\_dim} = 64:} Similar al \texttt{tcn\\_hidden\\_dim}, se seleccionó 64 como tamaño del vector de estado oculto de la LSTM para garantizar la consistencia en las capacidades de aprendizaje.

\item \textbf{\texttt{num\\_tcn\\_layers} = 3:} Este valor indica el número de capas convolucionales que se aplican secuencialmente. Se eligieron 3 capas para permitir una mayor profundidad y capacidad de aprendizaje.

\item \textbf{\texttt{num\\_lstm\\_layers} = 1:} Se utilizó una única capa LSTM debido a la naturaleza secuencial de los datos. Esto es suficiente para capturar la relación temporal restante después del procesamiento inicial por las capas convolucionales.

\item \textbf{\texttt{output\\_dim} = 1:} El modelo predice un único valor que representa la demanda energética en el siguiente instante de tiempo.

\end{itemize}

\noindent

La función \texttt{evaluate\\_window\\_size} permite entrenar el modelo con diferentes tamaños de ventana y registrar las métricas de entrenamiento y validación para cada uno, lo cual es crucial para determinar el tamaño de ventana más efectivo para capturar patrones cíclicos en los datos de demanda energética.

Los transformers permiten enfocar el análisis en aquellas secciones de la secuencia que tienen mayor influencia sobre el comportamiento futuro, lo cual es particularmente útil para identificar anomalías.

Un intervalo de confianza para las predicciones refleja el rango en el cual esperamos que caigan los valores reales en función de las predicciones del modelo, dado un nivel de confianza (por ejemplo, 95%).

