UNIVERSIDAD DE ANTIOQUIA

FUNDAMENTOS DE DEEP LEARNING

PROYECTO DE CURSO – INFORME FINAL -

WILLIAM ALEXANDER TORRES ZAMBRANO, 71784722

DOCENTE

RAÚL RAMOS POLLÁN

FACULTAD DE INGENIERÍA

INGENIERÍA DE SISTEMAS

MAYO 31 DE 2023

INFORME FINAL: Google Smartphone Decimeter Challenge 2022

1. Introducción

El presente trabajo tiene como objetivo abordar el problema de calcular la ubicación de teléfonos inteligentes con una alta precisión, en el rango de decímetros o incluso centímetros. Esta precisión mejorada permitiría habilitar servicios que requieren una precisión a nivel de carril, como la estimación de ETA de carril HOV (High Occupancy Vehicle). El enfoque propuesto implica el desarrollo de un modelo basado en mediciones de ubicación sin procesar de teléfonos inteligentes Android, utilizando conjuntos de datos recopilados en calles urbanas ligeras y a cielo abierto.

Para abordar este problema, se explorarán diferentes enfoques de aprendizaje profundo, incluyendo redes convolucionales, redes recurrentes (RNN), redes RNN bidireccionales y el modelo Transformer con arquitecturas CNN-LSTM. Estos enfoques permitirán capturar patrones espaciales y temporales en los datos de ubicación y realizar predicciones más precisas.

Antes de analizar los modelos de aprendizaje profundo, se realizará una exploración inicial de los datos. Para ello, se utilizará un conjunto de datos disponible en el enlace https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip y que corresponde a una copia del archivo de la competencia en kaggle y disponible en https://www.kaggle.com/competitions/smartphone-decimeter-2022/data, dicho conjunto de datos se encuentra en formato csv. Es de anotar que el archivo comprimido con los datos se ha proporcionado en el repositorio de GitHub y se accederá a él utilizando Google Colab.

La exploración inicial de los datos permite comprender la estructura de los datos, identificar posibles problemas, como valores faltantes o inconsistentes, y realizar visualizaciones para obtener información sobre la distribución y características de los datos. Esta etapa es fundamental para el procesamiento posterior de los datos y la selección adecuada de los modelos de aprendizaje profundo.

Posteriormente, se analiza el problema desde diferentes perspectivas utilizando modelos de redes convolucionales, redes recurrentes (RNN) y redes RNN bidireccionales. Estos modelos se adaptan al contexto del problema de ubicación de teléfonos inteligentes y se evalúa su rendimiento utilizando la métrica de desempeño descrita anteriormente. Además, se explora el modelo Transformer, que ha demostrado ser altamente efectivo en tareas de procesamiento de secuencias. Se utilizan arquitecturas basadas en la combinación de redes convolucionales y LSTM para capturar tanto la información espacial como temporal en los datos de ubicación.

De este modo, el trabajo tiene como objetivo abordar el problema de calcular la ubicación precisa de teléfonos inteligentes utilizando datos de ubicación sin procesar. Se exploran diferentes modelos de aprendizaje profundo, incluyendo redes convolucionales, redes recurrentes (RNN), redes RNN bidireccionales y el modelo Transformer con arquitecturas CNN-LSTM.

2. Exploración descriptiva del dataset

Antes de abordar cualquier análisis o modelo de clasificación, es fundamental realizar una exploración inicial del conjunto de datos disponible. Esta etapa permite comprender mejor la naturaleza de los datos, identificar posibles problemas o inconsistencias, y obtener una visión general de las características que conforman el dataset.

En el caso del presente estudio, se dispone de un conjunto de datos que contiene información sobre estudiantes y diversas variables relacionadas con su desempeño académico y características personales. Durante la exploración inicial, se llevan a cabo una serie de pasos para obtener una comprensión más completa del dataset:

- Análisis de la estructura: Se examina la estructura del dataset, incluyendo la cantidad de registros y columnas presentes. Esto proporciona una idea inicial de la magnitud y complejidad de los datos.
- Inspección de las variables: Se analizan las variables presentes en el dataset, revisando sus nombres, tipos de datos y posibles categorías. Esta etapa permite identificar las características relevantes que se utilizarán en el proceso de clasificación.
- Estadísticas descriptivas: Se calculan estadísticas descriptivas para cada variable, como la media, la mediana, el mínimo y el máximo. Esto proporciona información sobre la distribución de los datos y posibles valores atípicos.
- Manejo de datos faltantes: Se evalúa la presencia de datos faltantes en el dataset y se decide cómo abordarlos. Dependiendo de la cantidad y la naturaleza de los datos faltantes, se pueden aplicar técnicas como la imputación de valores o la eliminación de registros incompletos.
- Análisis de correlaciones: Se examina la correlación entre las diferentes variables del dataset. Esto permite identificar posibles relaciones o dependencias entre las características, lo cual puede ser útil en la selección de variables para el modelo de clasificación.

La exploración inicial del dataset es un paso crucial en cualquier proyecto de análisis y modelado de datos. Proporciona una base sólida para comprender la naturaleza de los datos y tomar decisiones informadas sobre las técnicas de preprocesamiento y modelado que se aplicarán. Además, ayuda a detectar posibles problemas o limitaciones del dataset, lo que contribuye a la calidad y fiabilidad de los resultados obtenidos en el proceso de clasificación.

En nuestro proyecto tenemos una exploración inicial:

```
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
import io
```

```
# Descargar el archivo comprimido desde GitHub
!wget https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip
```

```
# Descomprimir el archivo
with zipfile.ZipFile('device_gnss.csv.zip', 'r') as zip_ref:
    zip_ref.extractall()
```

```
# Cargar el archivo CSV en un DataFrame
df = pd.read_csv('device_gnss.csv')
```

```
# Conversión de columnas numéricas
numeric_columns = df.columns.drop(['MessageType'])  # Excluir columna no numérica

for column in numeric_columns:
    df[column] = pd.to_numeric(df[column], errors='coerce')
```

```
# Exploración inicial de los datos
print("Número de filas y columnas:", df.shape)
print("\nPrimeras 5 filas:")
print(df.head())
```

```
# Graficar los valores de cada columna
for column in df.columns:
    plt.figure(figsize=(12, 6))
    plt.plot(df[column])
    plt.xlabel('Índice de muestra')
    plt.ylabel(column)
    plt.title(f'Valores de {column}')
    plt.show()
```

3. Modelo de red convolucional neuronal (CNN)

A continuación analziamos el problema implementando un modelo de red neuronal convolucional (CNN) en el contexto del problema de calcular la ubicación precisa de teléfonos inteligentes utilizando datos de ubicación sin procesar. El modelo se implementa utilizando la biblioteca TensorFlow y Keras.

En este caso, el modelo de red neuronal convolucional (CNN) se utiliza para aprender patrones espaciales en los datos de entrada y hacer predicciones sobre las coordenadas de ubicación en el espacio tridimensional (x, y, z).

La importancia de este modelo radica en su capacidad para capturar características espaciales en los datos de ubicación y realizar predicciones más precisas. Las redes convolucionales son especialmente efectivas en la detección y extracción de características espaciales en datos estructurados, como imágenes, pero también se pueden aplicar a otros tipos de datos, como series de tiempo o secuencias.

En el contexto del problema de ubicación de teléfonos inteligentes, el modelo de red convolucional puede aprender patrones espaciales en las características de entrada, como la relación entre la fuerza de la señal (Cn0DbHz), la tasa de pseudorango (PseudorangeRateMetersPerSecond) y la incertidumbre de la tasa de pseudorango (PseudorangeRateUncertaintyMetersPerSecond), y utilizar estos patrones para hacer predicciones más precisas sobre las coordenadas de ubicación (SvPositionXEcefMeters, SvPositionYEcefMeters, SvPositionZEcefMeters).

El modelo de red neuronal convolucional es una herramienta importante en el campo del aprendizaje profundo y el procesamiento de datos espaciales. Permite aprender patrones

espaciales complejos en datos de ubicación y realizar predicciones más precisas sobre las coordenadas de ubicación de los teléfonos inteligentes.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from scipy.stats import scoreatpercentile

# Descargar el archivo comprimido desde GitHub
!wget https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip

# Descomprimir el archivo
import zipfile
with zipfile.ZipFile('device_gnss.csv.zip', 'r') as zip_ref:
    zip_ref.extractall()

# Cargar el archivo CSV en un DataFrame
df = pd.read_csv('device_gnss.csv')

# Seleccionar las columnas de entrada y salida
X = df[['Cn0DbHz', 'PseudorangeRateMetersPerSecond', 'PseudorangeRateUncertaintyMetersPerSecond']]
y = df[['SvPositionXEcefMeters', 'SvPositionYEcefMeters', 'SvPositionZEcefMeters']]

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Escalar los datos de entrada
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape de los datos para la entrada en la red convolucional
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_scaled.shape[1], 1))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled.shape[1], 1))

# Definir la arquitectura de la red convolucional
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='relu', input_shape=(X_train_scaled.shape[1], 1)))
model.add(MaxPooling1D(pool_size=1))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3))  # Salida con 3 dimensiones (x, y, z)
```

```
# Compilar el modelo
model.compile(loss='mean_squared_error', optimizer='adam')

# Entrenar el modelo
model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32, validation_data=(X_test_reshaped, y_test))

# Evaluar el modelo en el conjunto de prueba
y_pred = model.predict(X_test_reshaped)

# Calcular los errores de distancia horizontal
dist_errors = np.linalg.norm(y_pred - y_test, axis=1)
```

```
# Calcular los errores de percentil 50 y 95
percentile_50 = np.percentile(dist_errors, 50)
percentile_95 = np.percentile(dist_errors, 95)

# Imprimir los resultados
print('Error de distancia horizontal:', dist_errors)
print('Error de percentil 50:', percentile_50)
print('Error de percentil 95:', percentile_95)
```

Resultado:

```
device_gnss.csv.zip 100%[===================>]   9.34M  --.-KB/s    in 0.1s

2023-05-31 22:27:53 (76.4 MB/s) - ‘device_gnss.csv.zip’ saved [9790985/9790985]

Epoch 1/10
1400/1400 [==============================] - 10s 6ms/step - loss: nan - val_loss: nan
Epoch 2/10
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 3/10
1400/1400 [==============================] - 5s 4ms/step - loss: nan - val_loss: nan
Epoch 4/10
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 5/10
1400/1400 [==============================] - 5s 4ms/step - loss: nan - val_loss: nan
Epoch 6/10
1400/1400 [==============================] - 8s 5ms/step - loss: nan - val_loss: nan
Epoch 7/10
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 8/10
1400/1400 [==============================] - 5s 3ms/step - loss: nan - val_loss: nan
Epoch 9/10
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 10/10
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
350/350 [==============================] - 1s 1ms/step
Error de distancia horizontal: [nan nan nan ... nan nan nan]
Error de percentil 50: nan
Error de percentil 95: nan
```

El resultado que obtenido muestra que el modelo no ha convergido durante el entrenamiento. Se observa que tanto la pérdida de entrenamiento (loss) como la pérdida de validación (val_loss) tienen valores nan (no numéricos), lo que indica que no se ha logrado calcular una función de pérdida válida durante el entrenamiento.

A continuación, se modifica el código en el cual se han realizado los siguientes cambios:

1) Se ha ajustado la tasa de aprendizaje (learning_rate) del optimizador Adam a 0.001. Puedes ajustar este valor según sea necesario.

2) El número de épocas de entrenamiento se ha incrementado a 50. Esto permite que el modelo tenga más oportunidades de aprender durante el entrenamiento.

3) Se ha agregado un optimizer personalizado al compilar el modelo, utilizando el valor ajustado para la tasa de aprendizaje.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from scipy.stats import scoreatpercentile

# Descargar el archivo comprimido desde GitHub
!wget https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip
```

```
# Descomprimir el archivo
import zipfile
with zipfile.ZipFile('device_gnss.csv.zip', 'r') as zip_ref:
    zip_ref.extractall()

# Cargar el archivo CSV en un DataFrame
df = pd.read_csv('device_gnss.csv')

# Seleccionar las columnas de entrada y salida
X = df[['Cn0DbHz', 'PseudorangeRateMetersPerSecond', 'PseudorangeRateUncertaintyMetersPerSecond']]
y = df[['SvPositionXEcefMeters', 'SvPositionYEcefMeters', 'SvPositionZEcefMeters']]

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar los datos de entrada
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape de los datos para la entrada en la red convolucional
X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_scaled.shape[1], 1))
X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled.shape[1], 1))
```

```
# Definir la arquitectura de la red convolucional
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='relu', input_shape=(X_train_scaled.shape[1], 1)))
model.add(MaxPooling1D(pool_size=1))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3))  # Salida con 3 dimensiones (x, y, z)

# Compilar el modelo
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='mean_squared_error', optimizer=optimizer)
```

```
# Entrenar el modelo
model.fit(X_train_reshaped, y_train, epochs=50, batch_size=32, validation_data=(X_test_reshaped, y_test))

# Evaluar el modelo en el conjunto de prueba
y_pred = model.predict(X_test_reshaped)

# Calcular los errores de distancia horizontal
dist_errors = np.linalg.norm(y_pred - y_test, axis=1)
```

```python
# Calcular los errores de percentil 50 y 95
percentile_50 = np.percentile(dist_errors, 50)
percentile_95 = np.percentile(dist_errors, 95)

# Imprimir los resultados
print('Error de distancia horizontal:', dist_errors)
print('Error de percentil 50:', percentile_50)
print('Error de percentil 95:', percentile_95)
```

Se calculó con 50 épocas,

```
Epoch 44/50
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 45/50
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 46/50
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 47/50
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 48/50
1400/1400 [==============================] - 5s 3ms/step - loss: nan - val_loss: nan
Epoch 49/50
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
Epoch 50/50
1400/1400 [==============================] - 4s 3ms/step - loss: nan - val_loss: nan
350/350 [==============================] - 1s 2ms/step
Error de distancia horizontal: [nan nan nan ... nan nan nan]
Error de percentil 50: nan
Error de percentil 95: nan
```

Resultado:

De nuevo, el resultado que obtenido muestra que el modelo no ha convergido durante el entrenamiento. Se observa que tanto la pérdida de entrenamiento (loss) como la pérdida de validación (val_loss) tienen valores nan (no numéricos), lo que indica que no se ha logrado calcular una función de pérdida válida durante el entrenamiento.

4. Red neuronal recurrente LSTM (Long Short-Term Memory)

A continuación se analiza el problema desde un código que implementa un modelo de red neuronal recurrente LSTM (Long Short-Term Memory) en el contexto de la predicción de la ubicación precisa de dispositivos GNSS (Sistema Global de Navegación por Satélite).

El modelo LSTM es una variante de las redes neuronales recurrentes que se utiliza comúnmente para el procesamiento de secuencias y datos secuenciales en el campo del aprendizaje profundo. La arquitectura LSTM se caracteriza por su capacidad para capturar y recordar dependencias a largo plazo en los datos secuenciales.

En este caso, el modelo LSTM se utiliza para aprender patrones secuenciales en los datos de entrada relacionados con la ubicación de los dispositivos GNSS, como la información de pseudorango, incertidumbre de pseudorango, posición de los satélites y sesgo y deriva del reloj. El objetivo es predecir las coordenadas de ubicación precisas (X, Y, Z) utilizando estas características secuenciales.

La importancia de este modelo radica en su capacidad para capturar y modelar relaciones secuenciales en los datos de ubicación y hacer predicciones más precisas sobre las coordenadas de ubicación. El modelo LSTM es especialmente útil cuando se trabaja con datos secuenciales o de series de tiempo, ya que puede manejar dependencias a largo plazo y aprender patrones complejos en las secuencias.

Este modelo de red neuronal recurrente LSTM es una herramienta importante en el campo del aprendizaje profundo y el procesamiento de datos secuenciales. Permite aprender y modelar relaciones secuenciales en los datos de ubicación y realizar predicciones más precisas sobre las coordenadas de ubicación de los dispositivos GNSS.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Cargar los datos
data_url = 'https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip'
df = pd.read_csv(data_url, compression='zip')

# Seleccionar las características relevantes
features = ['RawPseudorangeMeters', 'RawPseudorangeUncertaintyMeters', 'SvPositionXEcefMeters',
            'SvPositionYEcefMeters', 'SvPositionZEcefMeters', 'SvClockBiasMeters', 'SvClockDriftMetersPerSecond']
target = ['WlsPositionXEcefMeters', 'WlsPositionYEcefMeters', 'WlsPositionZEcefMeters']

df = df[features + target].dropna()

# Dividir los datos en conjunto de entrenamiento y prueba
train_df, test_df = train_test_split(df, test_size=0.2, shuffle=True)

# Normalizar los datos
scaler = MinMaxScaler()
train_data = scaler.fit_transform(train_df.values)
test_data = scaler.transform(test_df.values)
```

```
# Preparar los datos de entrada y salida
train_X, train_y = train_data[:, :-3], train_data[:, -3:]
test_X, test_y = test_data[:, :-3], test_data[:, -3:]

# Reshape para LSTM (n_samples, timesteps, n_features)
timesteps = 1
train_X = train_X.reshape((train_X.shape[0], timesteps, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], timesteps, test_X.shape[1]))

# Definir el modelo LSTM
model = Sequential()
model.add(LSTM(64, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(3))  # 3 para las coordenadas X, Y, Z
model.compile(loss='mse', optimizer='adam')

# Entrenar el modelo
model.fit(train_X, train_y, epochs=10, batch_size=32, validation_data=(test_X, test_y))

# Evaluar el modelo en el conjunto de prueba
loss = model.evaluate(test_X, test_y)
print('Loss en el conjunto de prueba:', loss)
```

```
# Hacer predicciones en el conjunto de prueba
predictions = model.predict(test_X)

# Desnormalizar las predicciones y los valores reales
predictions = scaler.inverse_transform(np.concatenate((test_X.reshape((test_X.shape[0], -1)), predictions), axis=1))[:, -3:]
actual = scaler.inverse_transform(np.concatenate((test_X.reshape((test_X.shape[0], -1)), test_y), axis=1))[:, -3:]

# Calcular los errores de distancia
errors = np.linalg.norm(predictions - actual, axis=1)
percentile_50 = np.percentile(errors, 50)
percentile_95 = np.percentile(errors, 95)

print('Error de percentil 50:', percentile_50)
print('Error de percentil 95:', percentile_95)
```

Resultado:

```
Epoch 1/10
1341/1341 [==============================] - 9s 4ms/step - loss: 0.1046 - val_loss: 0.0984
Epoch 2/10
1341/1341 [==============================] - 4s 3ms/step - loss: 0.0981 - val_loss: 0.0981
Epoch 3/10
1341/1341 [==============================] - 4s 3ms/step - loss: 0.0973 - val_loss: 0.0987
Epoch 4/10
1341/1341 [==============================] - 5s 4ms/step - loss: 0.0961 - val_loss: 0.0958
Epoch 5/10
1341/1341 [==============================] - 4s 3ms/step - loss: 0.0949 - val_loss: 0.0961
Epoch 6/10
1341/1341 [==============================] - 5s 4ms/step - loss: 0.0938 - val_loss: 0.0945
Epoch 7/10
1341/1341 [==============================] - 4s 3ms/step - loss: 0.0927 - val_loss: 0.0928
Epoch 8/10
1341/1341 [==============================] - 4s 3ms/step - loss: 0.0915 - val_loss: 0.0911
Epoch 9/10
1341/1341 [==============================] - 6s 4ms/step - loss: 0.0901 - val_loss: 0.0915
Epoch 10/10
1341/1341 [==============================] - 4s 3ms/step - loss: 0.0887 - val_loss: 0.0902
336/336 [==============================] - 1s 2ms/step - loss: 0.0902
Loss en el conjunto de prueba: 0.09020043909549713
336/336 [==============================] - 1s 2ms/step
Error de percentil 50: 8421.986117046305
Error de percentil 95: 18874.204533836655
```

El resultado muestra el progreso del entrenamiento del modelo a lo largo de 10 épocas. Cada época representa un ciclo completo de alimentación hacia adelante y hacia atrás de los datos de entrenamiento a través de la red neuronal.

Para cada época, se muestra la pérdida (loss) tanto en el conjunto de entrenamiento como en el conjunto de validación (val_loss). La pérdida es una medida de la discrepancia entre las predicciones del modelo y los valores reales. El objetivo del entrenamiento es minimizar esta pérdida. En este caso, podemos observar que la pérdida disminuye progresivamente a medida que avanzan las épocas tanto en el conjunto de entrenamiento como en el conjunto de validación. Esto indica que el modelo está aprendiendo a hacer predicciones más precisas a medida que se entrena.

Después de las 10 épocas, se evalúa el modelo en el conjunto de prueba y se muestra la pérdida en el conjunto de prueba (Loss en el conjunto de prueba). En este caso, la pérdida en el conjunto de prueba es 0.0902, lo que indica que el modelo tiene un buen desempeño en el conjunto de prueba. Además, se calculan los errores de distancia utilizando las predicciones del modelo y los valores reales en el conjunto de prueba. El error de percentil 50 indica que el 50% de los errores de distancia son menores o iguales a 8421.9861 metros, mientras que el error de percentil 95 indica que el 95% de los errores de distancia son menores o iguales a 18874.2045 metros. Estos valores

pueden ser utilizados para evaluar la precisión del modelo en la determinación de la ubicación de los teléfonos móviles.

El modelo ha sido entrenado con éxito y muestra un buen desempeño en el conjunto de prueba, con una pérdida baja y errores de distancia aceptables. Sin embargo, es importante tener en cuenta el contexto del problema y las métricas de desempeño específicas para evaluar si estos resultados son satisfactorios para el objetivo del proyecto.

5. Red neuronal recurrente bidireccional con capas LSTM

El código implementa un modelo de red neuronal recurrente bidireccional con capas LSTM en el contexto de la predicción de la coordenada Z de la posición de dispositivos GNSS.

El modelo en particular se llama "Red Neuronal Recurrente Bidireccional con Capas LSTM" y se identifica en el campo del deep learning como una arquitectura de red neuronal recurrente bidireccional que utiliza capas LSTM.

La importancia de este modelo radica en su capacidad para capturar patrones secuenciales en los datos de entrada y realizar predicciones precisas sobre la coordenada Z de la posición de los dispositivos GNSS. Al utilizar una estructura bidireccional, el modelo puede aprovechar tanto la información pasada como futura en la secuencia de entrada, lo que lo hace especialmente efectivo en el procesamiento de datos secuenciales.

Además, se utiliza una función de pérdida personalizada llamada "quantile_loss" que permite optimizar el modelo para estimar cuantiles específicos de la distribución de los datos de destino. Esto es útil en aplicaciones donde se requiere estimar intervalos de confianza o percentiles para las predicciones.

Este modelo de red neuronal recurrente bidireccional con capas LSTM es una herramienta valiosa en el campo del aprendizaje profundo y el procesamiento de datos secuenciales. Permite capturar patrones secuenciales en los datos de posición de dispositivos GNSS y realizar predicciones precisas, mientras que la función de pérdida personalizada proporciona flexibilidad para optimizar el modelo para estimar cuantiles específicos.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, LSTM, Dense

def quantile_loss(q, y_true, y_pred):
    error = y_true - y_pred
    return tf.reduce_mean(tf.maximum(q * error, (q - 1) * error), axis=-1)

# Leer el archivo CSV comprimido
data = pd.read_csv('https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip', compression='zip')

# Seleccionar las columnas relevantes para el análisis
selected_columns = ['TimeNanos', 'SvPositionXEcefMeters', 'SvPositionYEcefMeters', 'SvPositionZEcefMeters']
data = data[selected_columns]

# Eliminar filas con valores faltantes
data = data.dropna()
```

```
# Dividir los datos en características (X) y etiquetas (y)
X = data[['TimeNanos', 'SvPositionXEcefMeters', 'SvPositionYEcefMeters']]
y = data['SvPositionZEcefMeters']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar los datos de entrada
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Reshape de los datos de entrada para ser compatibles con la red neuronal recurrente
X_train_reshaped = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_reshaped = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))

# Definir el modelo de red neuronal recurrente bidireccional
model = Sequential()
model.add(Bidirectional(LSTM(64, return_sequences=True), input_shape=(1, X_train_scaled.shape[1])))
model.add(Bidirectional(LSTM(64)))
model.add(Dense(1))

# Compilar el modelo con la función de pérdida de error de percentil personalizada
model.compile(loss=lambda y_true, y_pred: quantile_loss(0.5, y_true, y_pred), optimizer='adam')
```

```
# Entrenar el modelo
model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32)

# Evaluar el modelo en el conjunto de prueba
loss = model.evaluate(X_test_reshaped, y_test)
print(f'Loss en el conjunto de prueba: {loss}')

# Obtener las predicciones del modelo en el conjunto de prueba
y_pred = model.predict(X_test_reshaped)

# Calcular el error del percentil 50 y 95
error_percentil_50 = np.percentile(np.abs(y_pred.ravel() - y_test), 50)
error_percentil_95 = np.percentile(np.abs(y_pred.ravel() - y_test), 95)

print(f'Error del percentil 50: {error_percentil_50}')
print(f'Error del percentil 95: {error_percentil_95}')
```

Resultado:

```
Epoch 1/10
1341/1341 [==============================] - 24s 7ms/step - loss: 7137662.5000
Epoch 2/10
1341/1341 [==============================] - 9s 7ms/step - loss: 7137592.5000
Epoch 3/10
1341/1341 [==============================] - 9s 7ms/step - loss: 7137530.5000
Epoch 4/10
1341/1341 [==============================] - 9s 7ms/step - loss: 7137469.5000
Epoch 5/10
1341/1341 [==============================] - 10s 7ms/step - loss: 7137410.0000
Epoch 6/10
1341/1341 [==============================] - 10s 7ms/step - loss: 7137348.5000
Epoch 7/10
1341/1341 [==============================] - 8s 6ms/step - loss: 7137284.5000
Epoch 8/10
1341/1341 [==============================] - 9s 7ms/step - loss: 7137225.0000
Epoch 9/10
1341/1341 [==============================] - 9s 7ms/step - loss: 7137159.5000
Epoch 10/10
1341/1341 [==============================] - 10s 8ms/step - loss: 7137096.5000
336/336 [==============================] - 2s 2ms/step - loss: 7140731.0000
Loss en el conjunto de prueba: 7140731.0
336/336 [==============================] - 2s 2ms/step
Error del percentil 50: 14946342.300509445
Error del percentil 95: 23992634.87077629
```

En este caso, el modelo parece tener una función de pérdida relativamente alta, en el rango de millones. Esto significa que el modelo aún no ha alcanzado un buen ajuste a los datos y puede requerir más entrenamiento o ajustes en la arquitectura/modelo.

El error del percentil 50 y 95 muestra el rango de errores en las predicciones del modelo. El error del percentil 50 es el valor en el medio, mientras que el error del percentil 95 es el valor en el percentil 95, lo que significa que el 95% de los errores se encuentran por debajo de ese valor.

6. **Capa convolucional (Conv1D), una capa LSTM y una capa de atención personalizada en el contexto de la predicción de las coordenadas de posición (x, y, z) de dispositivos GNSS.**

Este modelo en particular no tiene un nombre específico en el campo del deep learning, ya que es una combinación de diferentes capas y técnicas. Sin embargo, se puede describir como un modelo híbrido que aprovecha la capacidad de las capas convolucionales para extraer características espaciales, la capacidad de las capas LSTM para capturar patrones secuenciales y la capa de atención para resaltar información relevante.

La importancia de este modelo radica en su capacidad para procesar datos secuenciales y espaciales simultáneamente, lo que es especialmente útil en aplicaciones como la predicción de la posición de dispositivos GNSS. La capa de atención permite al modelo enfocarse en las características más relevantes de la secuencia de entrada, mejorando la precisión de las predicciones. Además, la combinación de capas convolucionales y LSTM proporciona una representación rica de los datos, capturando tanto las características espaciales como las secuenciales.

Este modelo combina capas convolucionales, capas LSTM y una capa de atención personalizada para realizar predicciones precisas de las coordenadas de posición de dispositivos GNSS. Su importancia radica en su capacidad para procesar datos secuenciales y espaciales simultáneamente, lo que lo hace adecuado para aplicaciones que requieren el análisis de secuencias y características espaciales en conjuntos de datos complejos.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv1D, LSTM, Dense

# Descargar y cargar el archivo CSV con los datos
data_url = 'https://github.com/watorres/ProyectoDeepLearning/raw/main/device_gnss.csv.zip'
data = pd.read_csv(data_url, compression='zip')

# Preprocesar los datos
data = data[['TimeNanos', 'RawPseudorangeMeters', 'SvPositionXEcefMeters', 'SvPositionYEcefMeters', 'SvPositionZEcefMeters']]
data = data.dropna()
X = data[['TimeNanos', 'RawPseudorangeMeters']]
y = data[['SvPositionXEcefMeters', 'SvPositionYEcefMeters', 'SvPositionZEcefMeters']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_reshaped = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_reshaped = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)
```

```python
# Implementación personalizada de la capa de atención
class Attention(tf.keras.layers.Layer):
    def __init__(self):
        super(Attention, self).__init__()

    def build(self, input_shape):
        self.W = self.add_weight(shape=(input_shape[-1], 1), initializer='normal', trainable=True)
        super(Attention, self).build(input_shape)

    def call(self, inputs):
        score = tf.matmul(inputs, self.W)
        attention_weights = tf.nn.softmax(score, axis=1)
        weighted_sum = tf.reduce_sum(inputs * attention_weights, axis=1)
        return weighted_sum

# Definir y entrenar el modelo
input_layer = Input(shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2]))
conv1d_layer = Conv1D(64, 1, activation='relu')(input_layer)
lstm_layer = LSTM(64, return_sequences=True)(conv1d_layer)
attention_layer = Attention()(lstm_layer)
output_layer = Dense(3)(attention_layer)
model = Model(inputs=input_layer, outputs=output_layer)
model.compile(optimizer='adam', loss='mse')
model.fit(X_train_reshaped, y_train, epochs=10, batch_size=32, validation_data=(X_test_reshaped, y_test))
```

```python
# Calcular el error para los percentiles 50 y 95
y_pred = model.predict(X_test_reshaped)
errors = np.sqrt(np.sum((y_pred - y_test.values)**2, axis=1))
percentile_50 = np.percentile(errors, 50)
percentile_95 = np.percentile(errors, 95)

print("Error Percentile 50:", percentile_50)
print("Error Percentile 95:", percentile_95)
```

Resultado:

```
Epoch 1/10
1341/1341 [==============================] - 23s 11ms/step - loss: 256431072739328.0000 - val_loss: 256320930316288.0000
Epoch 2/10
1341/1341 [==============================] - 6s 5ms/step - loss: 256428707151872.0000 - val_loss: 256319034490880.0000
Epoch 3/10
1341/1341 [==============================] - 7s 5ms/step - loss: 256427062984704.0000 - val_loss: 256317524541440.0000
Epoch 4/10
1341/1341 [==============================] - 6s 5ms/step - loss: 256425334931456.0000 - val_loss: 256315863597056.0000
Epoch 5/10
1341/1341 [==============================] - 7s 5ms/step - loss: 256423506214912.0000 - val_loss: 256313934217216.0000
Epoch 6/10
1341/1341 [==============================] - 6s 4ms/step - loss: 256421945933824.0000 - val_loss: 256312139055104.0000
Epoch 7/10
1341/1341 [==============================] - 7s 5ms/step - loss: 256420352098304.0000 - val_loss: 256310427779072.0000
Epoch 8/10
1341/1341 [==============================] - 6s 5ms/step - loss: 256418489827328.0000 - val_loss: 256308783611904.0000
Epoch 9/10
1341/1341 [==============================] - 7s 5ms/step - loss: 256416711442432.0000 - val_loss: 256307139444736.0000
Epoch 10/10
1341/1341 [==============================] - 7s 5ms/step - loss: 256414865948672.0000 - val_loss: 256305226842112.0000
336/336 [==============================] - 1s 2ms/step
Error Percentile 50: 26714063.067030497
Error Percentile 95: 29601549.566716768
```

Estos resultados indican que el modelo ha sido entrenado durante 10 épocas. Durante el entrenamiento, se muestra el valor de la función de pérdida (loss) tanto para el conjunto de entrenamiento como para el conjunto de validación (val_loss).

En cada época, se muestra el valor de la función de pérdida tanto para el conjunto de entrenamiento como para el conjunto de validación. En este caso, los valores de pérdida son bastante altos, lo que indica que el modelo no está aprendiendo correctamente los patrones en los datos.

Después del entrenamiento, se calculan los errores para los percentiles 50 y 95 en el conjunto de prueba. El error del percentil 50 (Error Percentile 50) es de aproximadamente 26,714,063.07, lo que significa que el 50% de los errores se encuentran por debajo de este valor. El error del percentil 95 (Error Percentile 95) es de aproximadamente 29,601,549.57, lo que significa que el 95% de los errores se encuentran por debajo de este valor.

Los resultados indican que el modelo no está obteniendo un buen desempeño en la tarea de predicción, ya que la función de pérdida es alta y los errores son significativos.

7. Retos y consideraciones de despliegue

Al analizar los retos y consideraciones de despliegue de los modelos anteriores, podemos identificar los siguientes aspectos a tener en cuenta:

- **Requisitos de hardware y recursos:** Algunos modelos, como el modelo LSTM, pueden requerir más recursos computacionales y de memoria en comparación con otros modelos más simples, como el modelo con capa densa. Es importante considerar los requisitos de hardware y asegurarse de contar con suficientes recursos para ejecutar el modelo de manera eficiente.

- **Preprocesamiento de datos:** Todos los modelos anteriores involucran algún nivel de preprocesamiento de datos, como selección de características, normalización y transformación de datos. Estos pasos de preprocesamiento deben ser replicados en el entorno de implementación para garantizar resultados consistentes. Es importante tener en cuenta que cualquier cambio en los datos de entrada o en el preprocesamiento puede afectar los resultados del modelo.

- **Entorno de implementación y bibliotecas:** Los modelos hacen uso de bibliotecas y frameworks de machine learning, como TensorFlow y Scikit-learn. Para implementar los modelos en un entorno de producción, es necesario asegurarse de tener las bibliotecas y versiones adecuadas instaladas y configuradas correctamente. Además, se deben considerar las dependencias y requerimientos de las bibliotecas utilizadas para garantizar la compatibilidad y estabilidad del entorno.

- **Escalabilidad:** Algunos modelos, como el modelo LSTM y el modelo con capa de atención, pueden ser más complejos computacionalmente y requerir más tiempo de ejecución en comparación con modelos más simples. Si se espera un alto volumen de predicciones en tiempo real, es importante evaluar la escalabilidad del modelo y asegurarse de que pueda manejar la carga de trabajo de manera eficiente.

- **Actualización y reentrenamiento del modelo:** En muchos casos, los modelos de deep learning necesitan actualizarse y reentrenarse periódicamente para mantener su precisión. Esto implica establecer un proceso de actualización y reentrenamiento regular del modelo, que puede incluir la recopilación y etiquetado de nuevos datos, el ajuste de hiperparámetros y la reentrenamiento del modelo en el conjunto de datos actualizado. Además, se debe considerar cómo implementar y desplegar nuevas versiones del modelo sin interrupciones en la producción.

- **Monitoreo y evaluación del rendimiento:** Una vez que el modelo está implementado, es esencial establecer un sistema de monitoreo y evaluación continua para evaluar el rendimiento del modelo en producción. Esto implica el seguimiento de métricas de rendimiento, la detección de posibles problemas o desviaciones, y la toma de medidas correctivas cuando sea necesario. El monitoreo del rendimiento también puede ayudar a identificar la necesidad de reentrenar o actualizar el modelo en función de la evolución de los datos y los requisitos del problema.

El despliegue de modelos de deep learning implica consideraciones técnicas, computacionales y operativas. Es importante tener en cuenta estos retos y consideraciones para garantizar una implementación exitosa y efectiva del modelo en un entorno de producción.

8. Conclusiones

Basándonos en los modelos y consideraciones de despliegue mencionados anteriormente, se pueden destacar las siguientes conclusiones:

- **Variedad de modelos de Deep Learning:** Existen diferentes tipos de modelos de Deep Learning que se pueden utilizar para abordar problemas específicos. Estos modelos incluyen LSTM, modelos con capas convolucionales, modelos con capas de atención, entre otros. La elección del modelo adecuado depende de la naturaleza del problema y los datos disponibles.
- **Preprocesamiento de datos es crucial:** Antes de entrenar cualquier modelo de Deep Learning, es necesario realizar un preprocesamiento adecuado de los datos. Esto implica seleccionar características relevantes, normalizar los datos y asegurarse de que estén en un formato adecuado para el modelo. El preprocesamiento de datos puede afectar significativamente el rendimiento del modelo y debe ser realizado con cuidado.
- **Requisitos de recursos y escalabilidad:** Al implementar modelos de Deep Learning, es importante considerar los requisitos de recursos computacionales, como potencia de procesamiento y memoria. Algunos modelos más complejos pueden requerir más recursos, lo que debe tenerse en cuenta al seleccionar la infraestructura de implementación. Además, la escalabilidad del modelo debe evaluarse para garantizar que pueda manejar una carga de trabajo creciente.
- **Actualización y reentrenamiento periódico:** Los modelos de Deep Learning suelen requerir actualizaciones y reentrenamientos periódicos para mantener su rendimiento óptimo. Esto implica recopilar nuevos datos, ajustar hiperparámetros y reentrenar el modelo en el conjunto de datos actualizado. Establecer un proceso de actualización y reentrenamiento regular es esencial para mantener la precisión del modelo a medida que cambian los datos y los requisitos del problema.

- **Monitoreo continuo y evaluación del rendimiento:** Una vez implementado, es fundamental monitorear y evaluar continuamente el rendimiento del modelo en producción. Esto implica el seguimiento de métricas de rendimiento, la detección de posibles problemas y la toma de medidas correctivas. El monitoreo continuo ayuda a garantizar que el modelo esté funcionando como se espera y permite identificar la necesidad de actualizaciones o reentrenamiento.

Referencias

- Aggarwal, Charu C. **Neural networks and deep learning.** Springer 10 (2018): 978-3.
- Calin, Ovidiu. **Deep Learning Architectures.** Springer International Publishing, 2020.
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). **Deep learning** (Vol. 1). Cambridge: MIT
- Hochreiter, S., & Schmidhuber, J. (1997). **Long short-term memory. Neural computation**, 9(8), 1735-1780.
- Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). **Show and tell: A neural image caption generator.** In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3156-3164).
- Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., & Saenko, K. (2015). **Sequence to sequence-video to text.** In Proceedings of the IEEE international conference on computer vision (pp. 4534-4542).
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). **Sequence to sequence learning with neural networks.** In Advances in neural information processing systems (pp. 3104-3112).
- Pradeep Pujari, Md. Rezaul Karim, Mohit Sewak (2017) , **Practical Convolutional Neural Networks**, O'Reilly,