Aproximación de la Ecuación de Movimiento de un Péndulo Simple utilizando Redes Neuronales

Juan José González Giraldo

Mayo 2024

Abstract

En este proyecto, exploramos cómo las redes neuronales pueden utilizarse para aproximar la solución de la ecuación de movimiento de un péndulo simple. El péndulo simple es un sistema físico idealizado que consiste en una masa puntual suspendida de un hilo inextensible de longitud L. Diseñamos una red neuronal que predice el ángulo θ del péndulo en función del tiempo t, la longitud L y la gravedad g.

1 Introducción

La ecuación diferencial que describe el movimiento de un péndulo simple es:

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{L}\sin(\theta) = 0 \tag{1}$$

donde:

- θ es el ángulo de desplazamiento del péndulo desde la vertical,
- t es el tiempo,
- g es la aceleración debido a la gravedad,
- L es la longitud del péndulo.

2 Objetivo

Diseñar una red neuronal para aproximar la solución de la ecuación de movimiento del péndulo simple. Específicamente, la red deberá predecir el ángulo θ del péndulo en función del tiempo t, la longitud L y la gravedad g.

3 Recopilación de Datos

La ecuación diferencial que describe el movimiento de un péndulo simple se puede solucionar como una ecuación diferencial ordinaria. Esto se expresa en la función pendulum:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp


# Definir la función que representa el sistema de ecuaciones diferenciales
def pendulum(t, y, L=1, g=9.81):
    theta, omega = y  # theta es el ángulo, omega es la velocidad angular
    dtheta_dt = omega
    domega_dt = -(g / L) * np.sin(theta)
    return np.array([dtheta_dt, domega_dt])
```

A partir del sistema de ecuaciones diferenciales que dan soluciones a la ecuación diferencial del péndulo simple se generó un conjunto de datos con las siguientes constantes:

- Rango de tiempo = [0, 10]
- $g = 9.81$
- $L = 1$

El conjunto de datos generado consiste en pares de entrada-salida, donde la entrada es el tiempo t, y la salida es el ángulo θ correspondiente en ese momento.

4 Preprocesamiento de Datos

Utilizamos normalización MinMax para transformar los datos de entrada y salida a un rango entre [0, 1]. Esto ayuda a mejorar la convergencia y estabilidad del entrenamiento. Hay que tener en cuenta que el rango de los datos de entrada al scaler determinará el rango en el que el modelo funcionará ya que valores superiores o inferiores causan valores menores que 0 y mayores que 1.

```
from sklearn.preprocessing import MinMaxScaler

# Normalizar los datos
scaler_X = MinMaxScaler()
X_normalized = scaler_X.fit_transform(X.reshape(-1, 1))

scaler_y = MinMaxScaler()
y_normalized = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()
```

Usamos 2 scalers para diferenciar los valores de entrada y de salida.

5 Diseño de la Red Neuronal

La red neuronal diseñada para este proyecto es una red feedforward (FFNN) secuencial con la siguiente arquitectura:

- Capa de entrada: 1 neurona (para t).
- 8 capas ocultas, cada una con 64 neuronas y función de activación ReLU.
- Capa de salida: 1 neurona (para θ) con función de activación lineal.
- Optimizador: Adam
- Función de pérdida: MSE

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import RootMeanSquaredError

model = Sequential()
model.add(Input(shape=(1,)))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(64, activation="relu"))
```

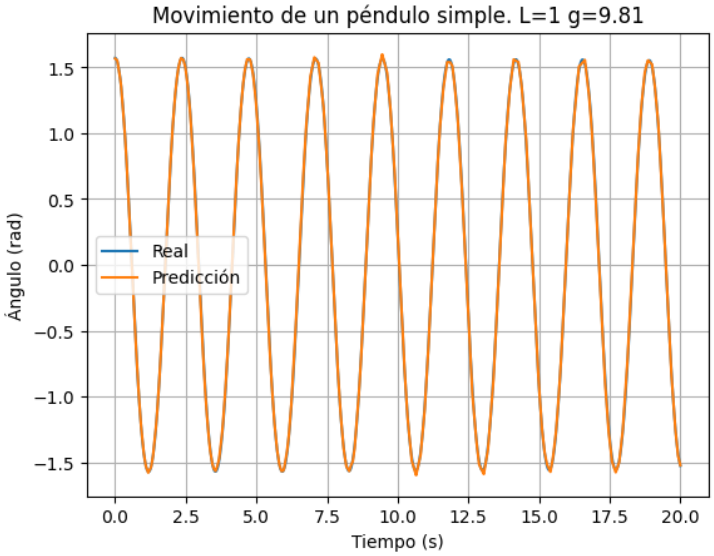
```
model.add(Dense(1))

model.compile(optimizer=Adam(), loss="mse", metrics=[RootMeanSquaredError()])
model.summary()
```

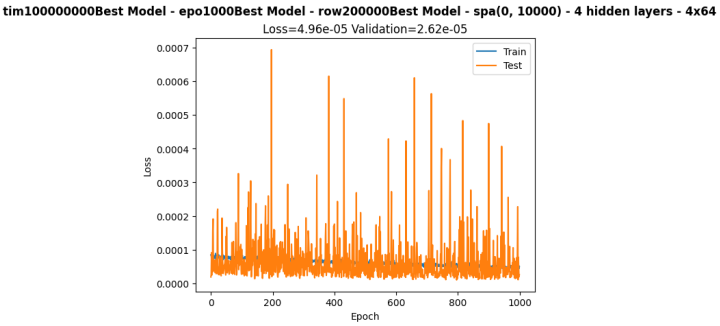
6 Resultados

El modelo final muestra una buena capacidad de generalización y precisión en las predicciones del ángulo θ.

- MSE = $\{1.21e-4\}$
- RMSE = $\{6.67e-5\}$



(a) Reales vs predicciones en el péndulo

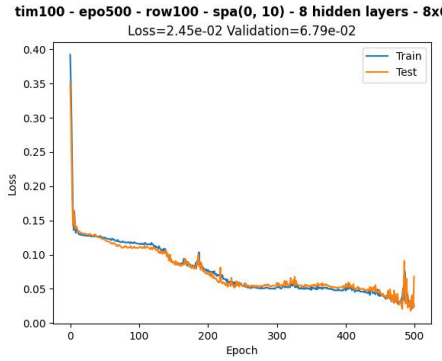


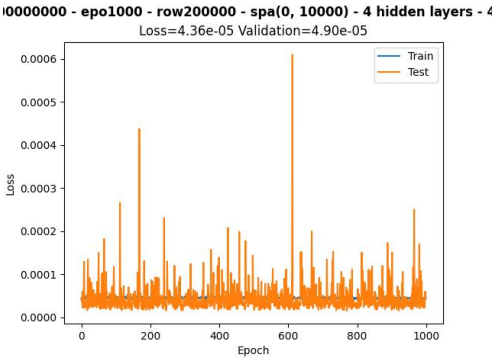
(b) Reducción del error

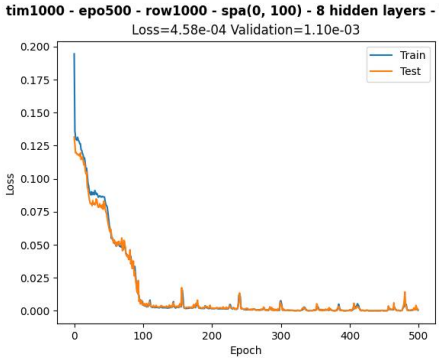
Figure 1: Rendimiento del modelo

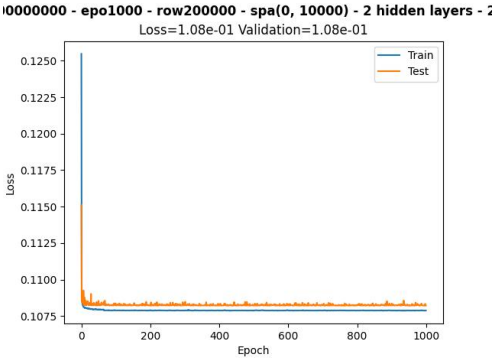
7 Discusión

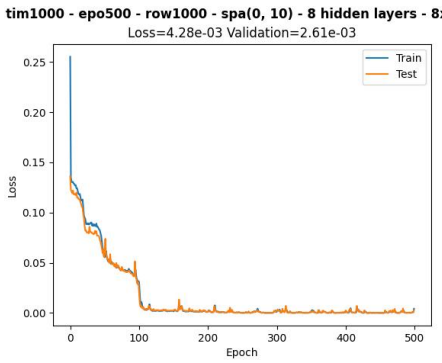
Se entrenaron múltiples modelos variando los hiperparámetros como la cantidad de epochs, la cantidad de capas y la cantidad de neuronas en cada capa. La decisión del mejor modelo se tomó a partir del valor mas bajo de error en validación.

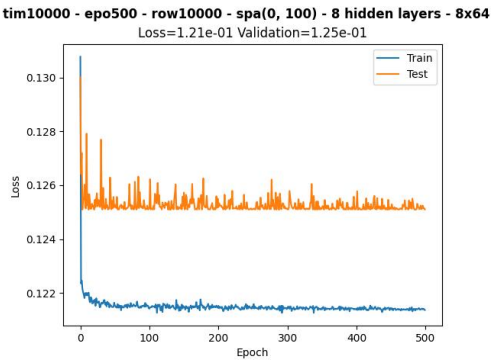

tim100 - epo500 - row100 - spa(0, 10) - 8 hidden layers - 8x64
Loss=2.45e-02 Validation=6.79e-02
Train
Test
Loss
Epoch

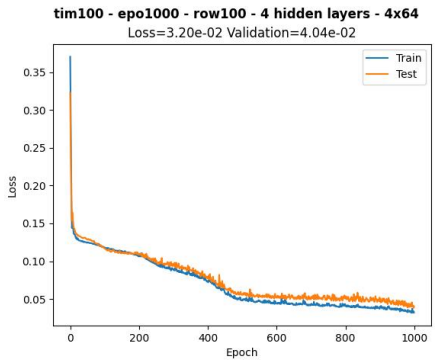

0000000 - epo1000 - row200000 - spa(0, 10000) - 4 hidden layers - 4
Loss=4.36e-05 Validation=4.90e-05
Train
Test
Loss
Epoch

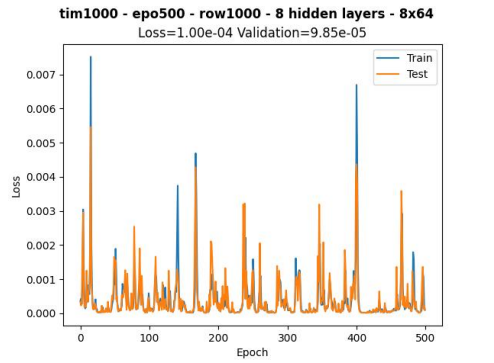

tim1000 - epo500 - row1000 - spa(0, 100) - 8 hidden layers - 8x64
Loss=4.58e-04 Validation=1.10e-03
Train
Test
Loss
Epoch

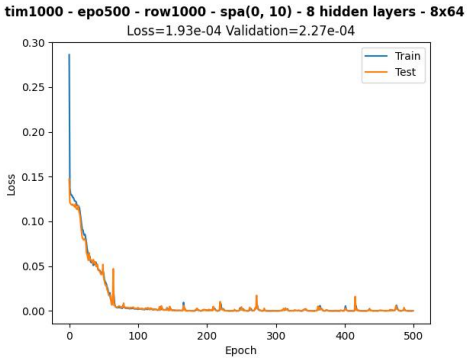

0000000 - epo1000 - row200000 - spa(0, 10000) - 2 hidden layers - 2
Loss=1.08e-01 Validation=1.08e-01
Train
Test
Loss
Epoch

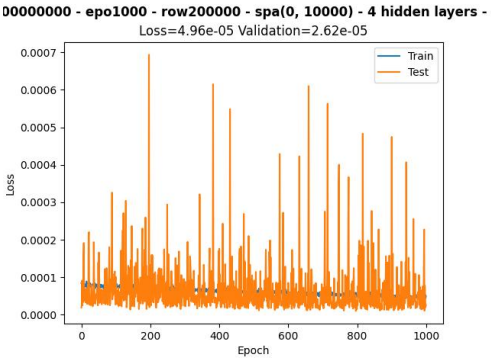

tim1000 - epo500 - row1000 - spa(0, 10) - 8 hidden layers - 8x64
Loss=4.28e-03 Validation=2.61e-03
Train
Test
Loss
Epoch

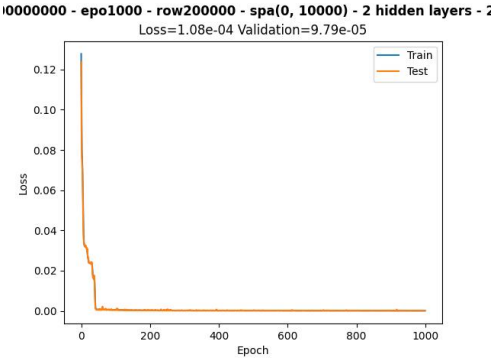

tim10000 - epo500 - row10000 - spa(0, 100) - 8 hidden layers - 8x64
Loss=1.21e-01 Validation=1.25e-01
Train
Test
Loss
Epoch

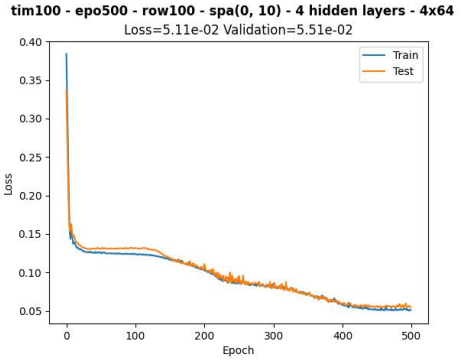

tim100 - epo1000 - row100 - 4 hidden layers - 4x64
Loss=3.20e-02 Validation=4.04e-02
Train
Test
Loss
Epoch

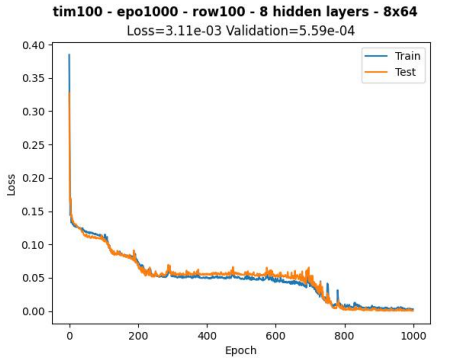

tim1000 - epo500 - row1000 - 8 hidden layers - 8x64
Loss=1.00e-04 Validation=9.85e-05
Train
Test
Loss
Epoch

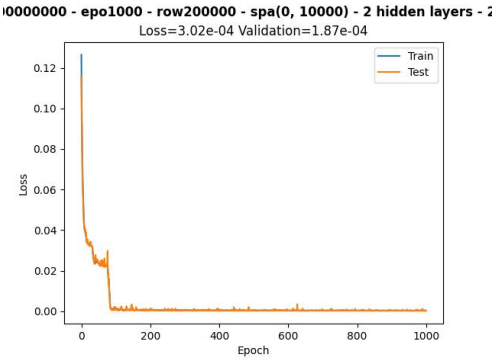

tim1000 - epo500 - row1000 - spa(0, 10) - 8 hidden layers - 8x64
Loss=1.93e-04 Validation=2.27e-04
Train
Test
Loss
Epoch

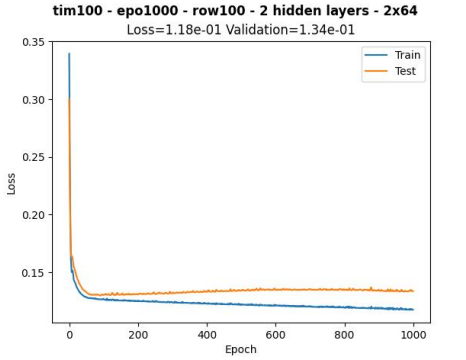

00000000 - epo1000 - row200000 - spa(0, 10000) - 4 hidden layers -
Loss=4.96e-05 Validation=2.62e-05
Train
Test
Loss
Epoch

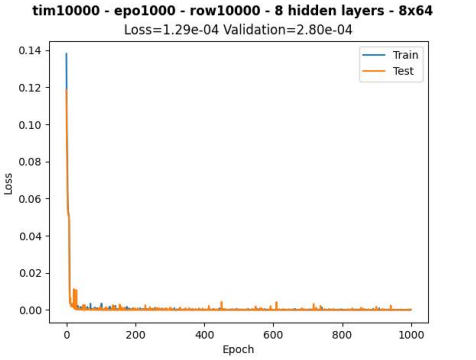

00000000 - epo1000 - row200000 - spa(0, 10000) - 2 hidden layers - 2
Loss=1.08e-04 Validation=9.79e-05
Train
Test
Loss
Epoch


tim100 - epo500 - row100 - spa(0, 10) - 4 hidden layers - 4x64
Loss=5.11e-02 Validation=5.51e-02
Train
Test
Loss
Epoch


tim100 - epo1000 - row100 - 8 hidden layers - 8x64
Loss=3.11e-03 Validation=5.59e-04
Train
Test
Loss
Epoch

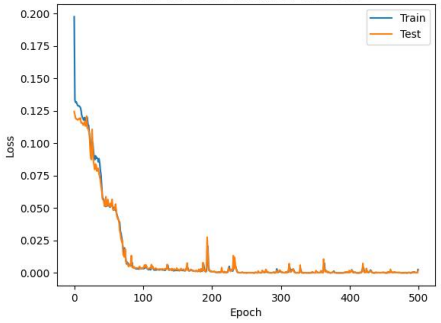

00000000 - epo1000 - row200000 - spa(0, 10000) - 2 hidden layers - 2
Loss=3.02e-04 Validation=1.87e-04
Train
Test
Loss
Epoch


tim100 - epo1000 - row100 - 2 hidden layers - 2x64
Loss=1.18e-01 Validation=1.34e-01
Train
Test
Loss
Epoch


tim10000 - epo1000 - row10000 - 8 hidden layers - 8x64
Loss=1.29e-04 Validation=2.80e-04
Train
Test
Loss
Epoch

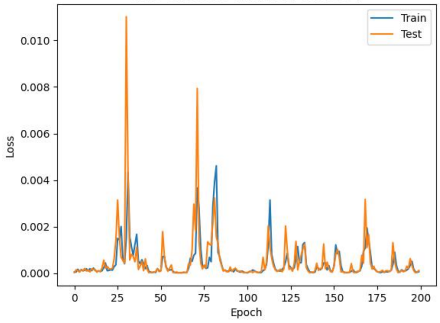
tim1000 - epo500 - row1000 - spa(0, 10) - 8 hidden layers - 8x64

Loss=2.66e-03 Validation=1.69e-03



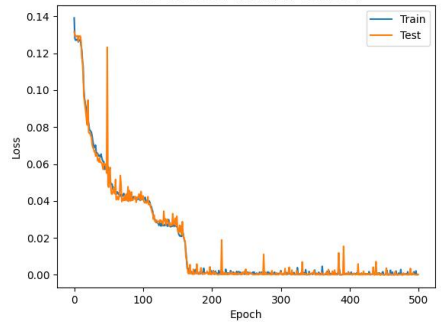
tim1000 - epo200 - row1000 - 8 hidden layers - 8x64

Loss=7.26e-05 Validation=1.16e-04



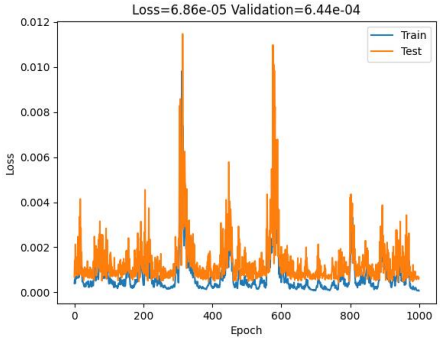
tim10000 - epo500 - row10000 - spa(0, 20) - 8 hidden layers - 8x64

Loss=1.85e-04 Validation=1.41e-04



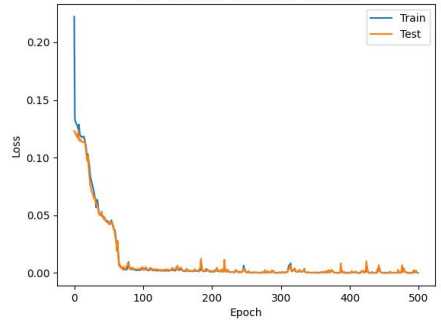
tim100 - epo1000 - row100 - 4 hidden layers - 4x64

Loss=6.86e-05 Validation=6.44e-04



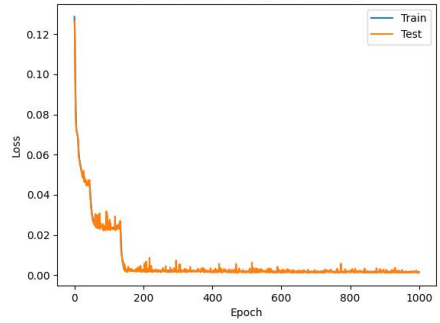
tim1000 - epo500 - row1000 - 8 hidden layers - 8x64

Loss=1.21e-04 Validation=6.67e-05



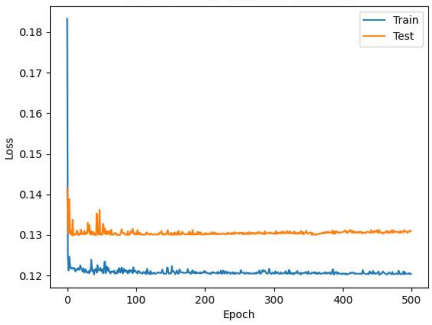
00000000 - epo1000 - row200000 - spa(0, 10000) - 2 hidden layers -

Loss=1.69e-03 Validation=1.20e-03



tim1000 - epo500 - row1000 - spa(0, 100) - 8 hidden layers - 8x64

Loss=1.20e-01 Validation=1.31e-01



8 Conclusión

Concluimos que las redes neuronales pueden ser una herramienta eficaz para aproximar la solución de la ecuación de movimiento de un péndulo simple. Sin embargo, es importante considerar la complejidad del modelo y la calidad de los datos de entrada para obtener resultados precisos.