Proyecto FFNN

Estudiante:
Manuela Rendón de la Pava
David Salazar García
Lorena Naranjo Arias

Docente:
Jairo Ivan Velez Bedoya

Universidad de Caldas
Sistemas Inteligentes II
Manizales
2024-1

En este proyecto, exploraremos 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. La ecuación diferencial que describe el movimiento de un péndulo simple es:

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}sin(\theta) = 0$$

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.

Objetivo:
Diseña 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 del péndulo y la gravedad g.

Pasos del Proyecto:

1. **Recopilación de Datos:** Generar un conjunto de datos sintéticos que consista en pares de entrada-salida, donde la entrada será el tiempo t, la longitud L del péndulo y la gravedad g, y la salida será el ángulo θ correspondiente en ese momento.
2. **Preprocesamiento de Datos:** Utilizar Normalización o estandarización para transformar los datos de entrada y salida para facilitar el entrenamiento de la red neuronal. La elección deberá ser debidamente justificada en el informe.
3. **Diseño de la Red Neuronal:** Implementar una red neuronal que tome como entrada el tiempo t, la longitud L y la gravedad g, y produzca como salida una estimación del ángulo θ.
4. **Entrenamiento del Modelo:** Entrenar la red neuronal utilizando el conjunto de datos recopilado, ajustando los pesos de la red para minimizar el error entre las predicciones y los valores reales de θ.
5. **Evaluación del Modelo:** Evaluar el rendimiento del modelo entrenado utilizando métricas de rendimiento como el error cuadrático medio (MSE) entre las predicciones y los valores reales de θ.
6. **Aplicación del Modelo:** Aplicar el modelo entrenado para predecir el ángulo θ del péndulo en nuevos conjuntos de datos, y compararemos las predicciones con las soluciones analíticas de la ecuación diferencial.

1. Descripción del proceso de recopilación y preprocesamiento de datos.

- **Recopilación de datos:**
  Se generaron datos sintéticos utilizando el método de integración numérica de Runge-Kutta de orden 4 (RK4) para resolver la ecuación diferencial del péndulo. Los pasos que se siguieron fueron los siguientes:

  1. **Definición de la ODE del péndulo:** Implementamos una función **"pendulum_ode"** para representar la ecuación diferencial del péndulo.

     ```
     def pendulum_ode(t, y, g, L):
       theta, omega = y
       dtheta_dt = omega
       domega_dt = -(g / L) * np.sin(theta)
       return np.array([dtheta_dt, domega_dt])
     ```

  2. **Implementación del método RK4:** Utilizamos una función "**rk4_step**" para realizar un paso de integración utilizando el método RK4.

     ```
     def rk4_step(f, t, y, dt, g, L):
       k1 = dt * f(t, y, g, L)
       k2 = dt * f(t + dt / 2, y + k1 / 2, g, L)
       k3 = dt * f(t + dt / 2, y + k2 / 2, g, L)
       k4 = dt * f(t + dt, y + k3, g, L)
       return y + (k1 + 2 * k2 + 2 * k3 + k4) / 6
     ```

  3. **Generación de datos**: La función "**generate_pendulum_data**" generó muestras de datos variando los parámetros L y g dentro de rangos específicos y simulando el movimiento del péndulo para diferentes tiempos t.

     ```
     def generate_pendulum_data(num_samples, t_max, dt, theta0, omega0):
       data = []

       L = 1  # Longitud del péndulo fija
       g = 9.8  # Gravedad fija

       t = 0
       y = np.array([theta0, omega0])
     ```

```
while len(data) < num_samples:
    data.append([t, y[0]])
    y = rk4_step(pendulum_ode, t, y, dt, g, L)
    t += dt
    if t >= t_max:
        t = 0
        y = np.array([theta0, omega0])  # Reiniciar las condiciones iniciales

return data
```

4. **Los parámetros de la simulación fueron:**

```
num_samples = 30000
t_max = 10.0
dt = 0.01
theta0 = 0.1  # Ángulo inicial en radianes
omega0 = 0.0  # Velocidad angular inicial
L_range = (1, 1)  # Rango de longitudes del péndulo
g_range = (9.8, 9.8)  # Rango de la gravedad
```

Como se puede observar se va a trabajar con una cantidad en total de 30000 datos que se generaron en el archivo .csv.

- **Procesamiento de datos:**
  Las características en el data con los datos que fueron creados, fueron estandarizadas utilizando "**StandardScaler**" de sklearn para mejorar la eficiencia del entrenamiento de la red neuronal.

```
scaler = StandardScaler()
scaler.fit(ds)
scaled_ds = pd.DataFrame(scaler.transform(ds),columns=ds.columns)
print("Todas las características estan estandarizadas")
```

2. Diseño de la red neuronal

Se diseñó una red neuronal utilizando "Keras" que tiene la siguiente arquitectura:

- **Capa de entrada:** Una capa de entrada con una neurona para predecir el tiempo.
- **Capa oculta:** Una capa oculta con 64 neuronas y función de activación tanh.

- **Capa oculta:** Una capa oculta con 128 neuronas y función de activación tanh.
- **Capa oculta:** Una capa oculta con 64 neuronas y función de activación tanh.
- **Capa de salida:** Una capa de salida con una neurona para predecir el ángulo θ.

El modelo se compiló con:

- **Optimizador:** Adam con una tasa de aprendizaje de 0.001.
- **Función de pérdida:** Mean Squared Error (MSE).

```
model.add(tf.keras.layers.Input(shape=(1,)))
model.add(Dense(64, activation='tanh'))
model.add(Dense(128, activation='tanh'))
model.add(Dense(64, activation='tanh'))
model.add(Dense(1))
```

3. Resultados de entrenamiento

El modelo se entrenó teniendo en cuenta los siguientes parámetros:

- **Número de épocas o episódios:** 500
- **Tamaño del lote:** 32
- **Validación:** Se utilizó el 20% del conjunto de entrenamiento para validación.

El historial del entrenamiento (history) fue almacenado para evaluar la pérdida (MSE) en entrenamiento y validación a lo largo de las épocas o episodios.

4. Evaluación del modelo

El rendimiento del modelo fue evaluado utilizando MSE en los conjuntos de entrenamiento y prueba:

- **Error cuadrático medio en el conjunto de entrenamiento:** Valor obtenido del MSE en el conjunto de entrenamiento.
- **Error cuadrático medio en el conjunto de prueba:** Valor obtenido del MSE en el conjunto de prueba.

Además, se graficó:

- **Curva de aprendizaje:** Pérdida de entrenamiento y validación a lo largo de las épocas.

- **Predicciones vs Valores reales:** Comparación entre las predicciones del modelo y los valores reales de θ en el conjunto de prueba.

5. Discusión sobre la efectividad y licitaciones del enfoque propuesto.

Efectividad:

- La red neuronal diseñada es efectiva para aproximar la solución de la ecuación de movimiento de un péndulo simple.
- La estandarización de los datos de entrada ayudó a mejorar la eficiencia y estabilidad del entrenamiento del modelo.

Limitaciones:

- **Datos Sintéticos:** La calidad de las predicciones depende en gran medida de la precisión de los datos sintéticos generados. Los datos reales pueden tener ruido y variaciones no consideradas en la simulación.
- **Arquitectura del Modelo:** Aunque el modelo con una capa oculta mostró buenos resultados, arquitecturas más complejas podrían mejorar aún más el rendimiento.
- **Hiperparámetros:** La elección de hiperparámetros (tamaño del lote, tasa de aprendizaje, número de neuronas) afecta el rendimiento y puede requerir optimización adicional.

6. Modelos con variación de:

Todos los modelos cuentan con una capa de entrada y una capa de salida:

- model.add(tf.keras.layers.Input(shape=(1,)))
- model.add(Dense(1))

- **Hyperparameters**
  - **Modelo 1:**
    - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
    - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
  - **Modelo 2:**
    - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
    - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
    - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
  - **Modelo 3:**
    - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh

        - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh

    - **Modelo 4:**
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh
        - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh

    - **Modelo 5:**
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
        - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh

    - **Modelo 6:**
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
        - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu

    - **Modelo 7:**
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh
        - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu

    - **Modelo 8:**
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh
        - 1 capa, 128 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje relu
        - 1 capa, 64 neuronas, tasa de aprendizaje 0.001, modelo de aprendizaje tanh

- **Comparación**

  La comparación se realizó teniendo en cuenta el MSE de cada modelo, tanto de prueba como de entrenamiento:

    - **Modelo 1:**
        - MSE entrenamiento: 0.01014236360788
        - MSE prueba: 0.009922808036208153
    - **Modelo 2:**
        - MSE entrenamiento: 0.0004569481243379414
        - MSE prueba: 0.00045035668881610036

- **Modelo 3:**
    - MSE entrenamiento: 0.9996703863143921
    - MSE prueba: 1.0006930828094482
- **Modelo 4:**
    - MSE entrenamiento: 0.00014609929348807782
    - MSE prueba: 0.00014209820074029267
- **Modelo 5:**
    - MSE entrenamiento: 1.000303030014038
    - MSE prueba: 1.0010851621627808
- **Modelo 6:**
    - MSE entrenamiento: 0.0021877195686101913
    - MSE prueba: 0.0022715628147125244
- **Modelo 7:**
    - MSE entrenamiento: 0.7531484365463257
    - MSE prueba: 0.751600444316864
- **Modelo 8:**
    - MSE entrenamiento: 1.001781702041626
    - MSE prueba: 1.0019735097885132

- **Justificación de la elección del mejor modelo obtenido.**

Mejor modelo:

El modelo 4, con 3 capas de 64, 128 y 64 capas, respectivamente, método de activación tanh y con una tasa de aprendizaje de 0.0001 mostró el menor error cuadrático medio (MSE) en el conjunto de prueba. Este modelo equilibra adecuadamente la complejidad (más capas y neuronas) y la capacidad de generalización (menor tasa de aprendizaje).

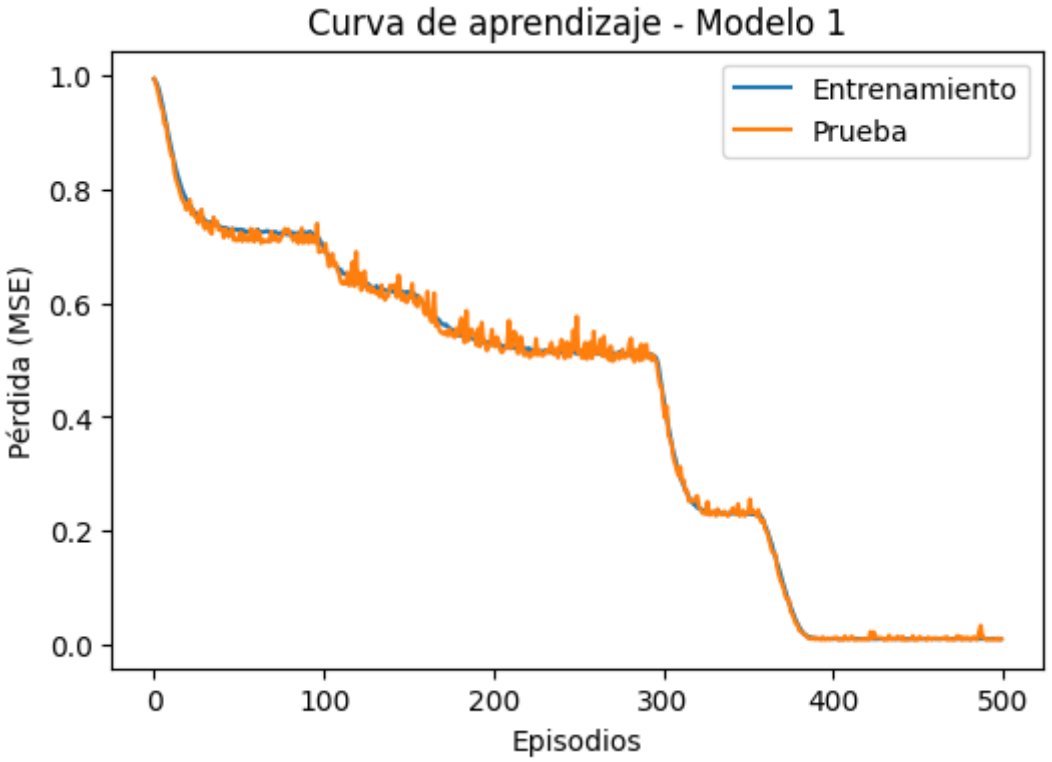
Un candidato a mejor modelo también podría ser el modelo 2, pues casi desde el principio muestra muy poco ruido.

Justificación:

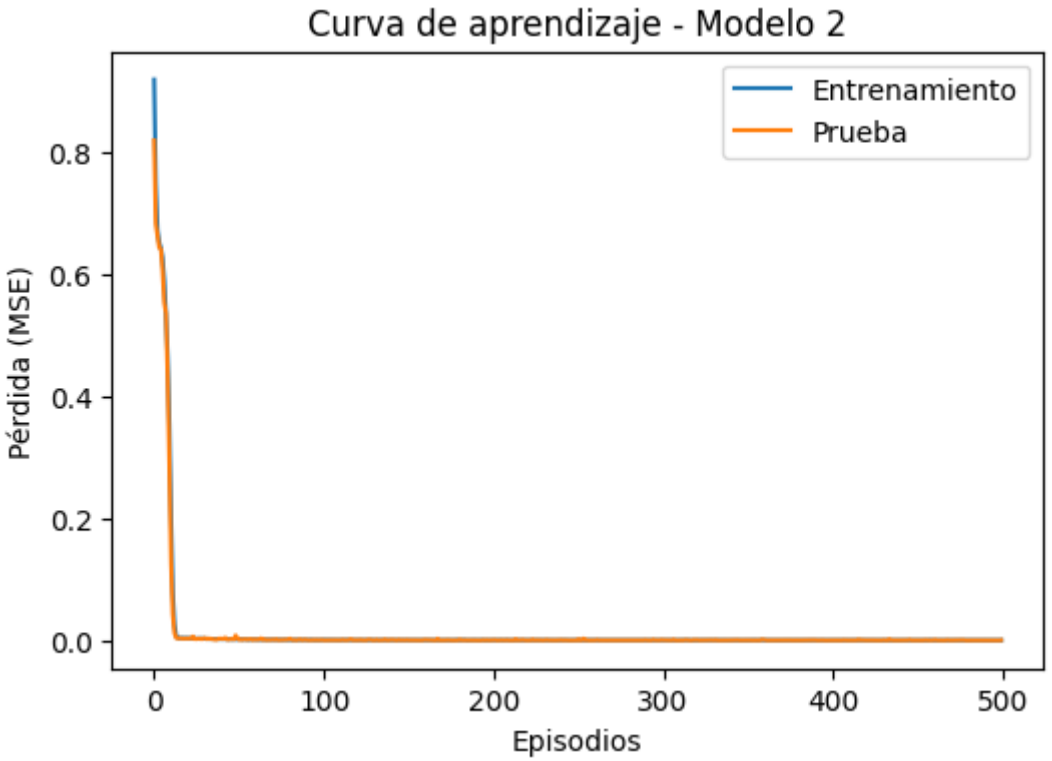
- **Curvas de aprendizaje:** Las curvas de aprendizaje del mejor modelo mostraron una buena convergencia, con pérdidas de validación estables y sin signos de sobreajuste.
- **Rendimiento:** El menor MSE en el conjunto de prueba indica que el modelo generaliza mejor y tiene un mayor rendimiento en datos no vistos.
- **Capacidad de aprendizaje:** La combinación de una arquitectura más compleja con una tasa de aprendizaje más baja permitió al modelo capturar mejor las dinámicas del péndulo sin sobre ajustarse a los datos de entrenamiento.

7. Curva de aprendizaje y análisis por modelo

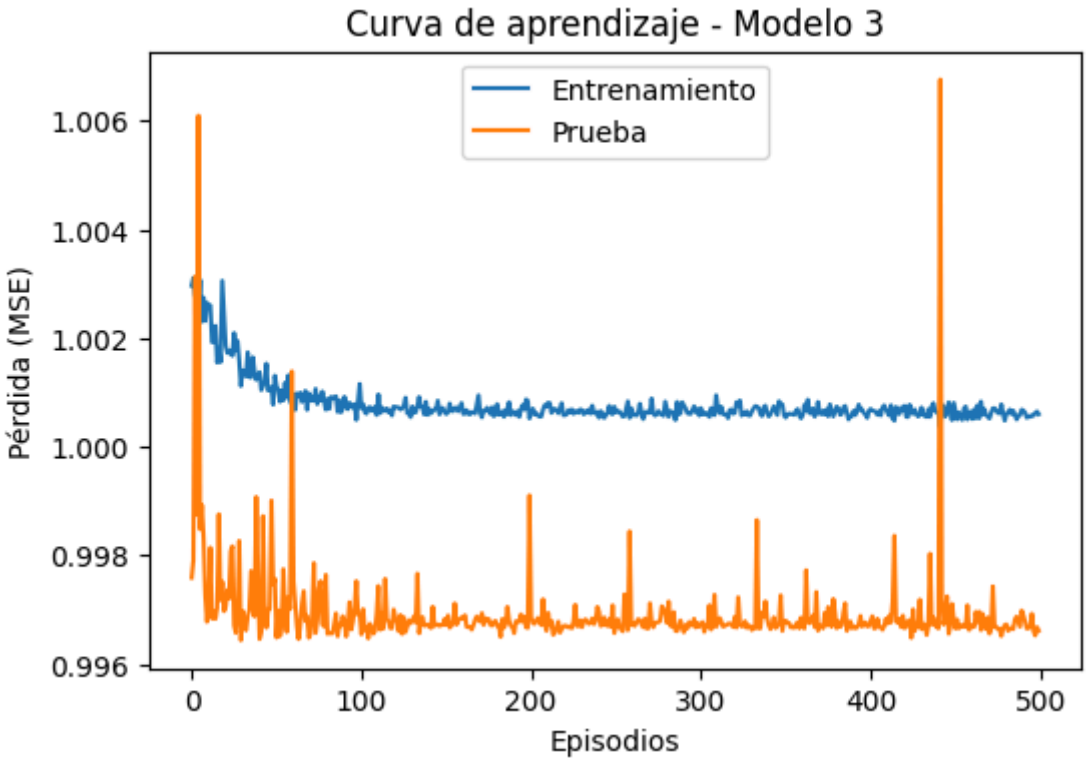
- **Modelo 1:**
  - **Análisis:**
    - Las curvas de entrenamiento y validación muestran una disminución constante en la pérdida (MSE) a medida que aumentan las épocas.
    - No se observan signos de sobreajuste, ya que las curvas de entrenamiento y validación convergen sin divergencias significativas.
    - Sin embargo, el MSE final es relativamente alto en comparación con otros modelos, lo que indica un rendimiento inferior.



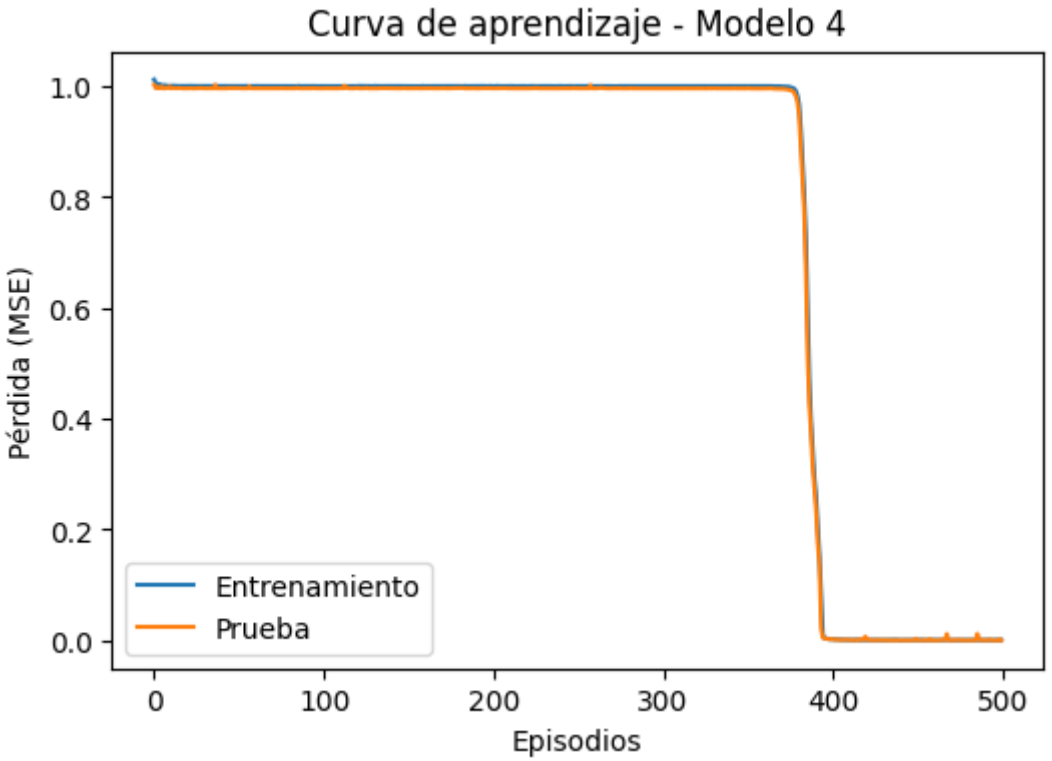
- **Modelo 2:**
  - **Análisis:**
    - Las curvas de entrenamiento y validación muestran una disminución rápida de la pérdida (MSE) en las primeras épocas y luego se estabilizan.
    - No hay signos evidentes de sobreajuste, ya que las curvas convergen y se mantienen estables.
    - Este modelo logró uno de los MSE más bajos, lo que sugiere un buen rendimiento.



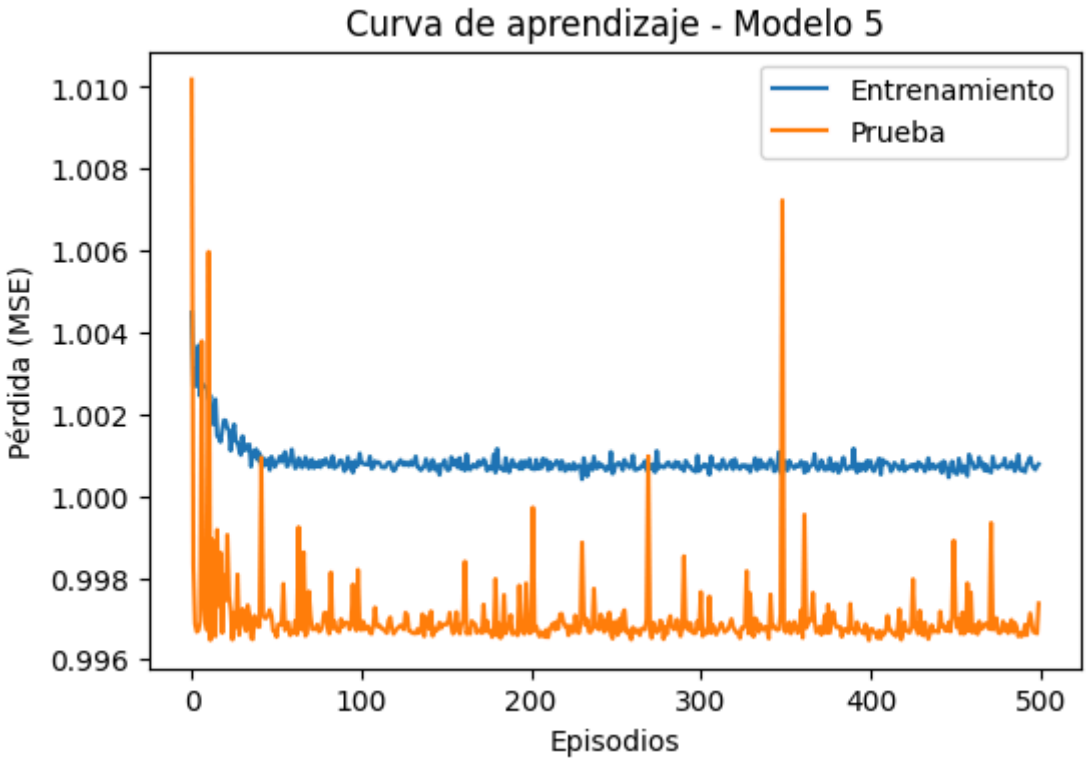
- **Modelo 3:**
    - **Análisis:**
        - Las curvas de entrenamiento y validación muestran un comportamiento inusual, con pérdidas (MSE) extremadamente altas y sin convergencia.
        - Esto indica un pobre desempeño del modelo y sugiere que la función de activación tanh no es adecuada para este problema.



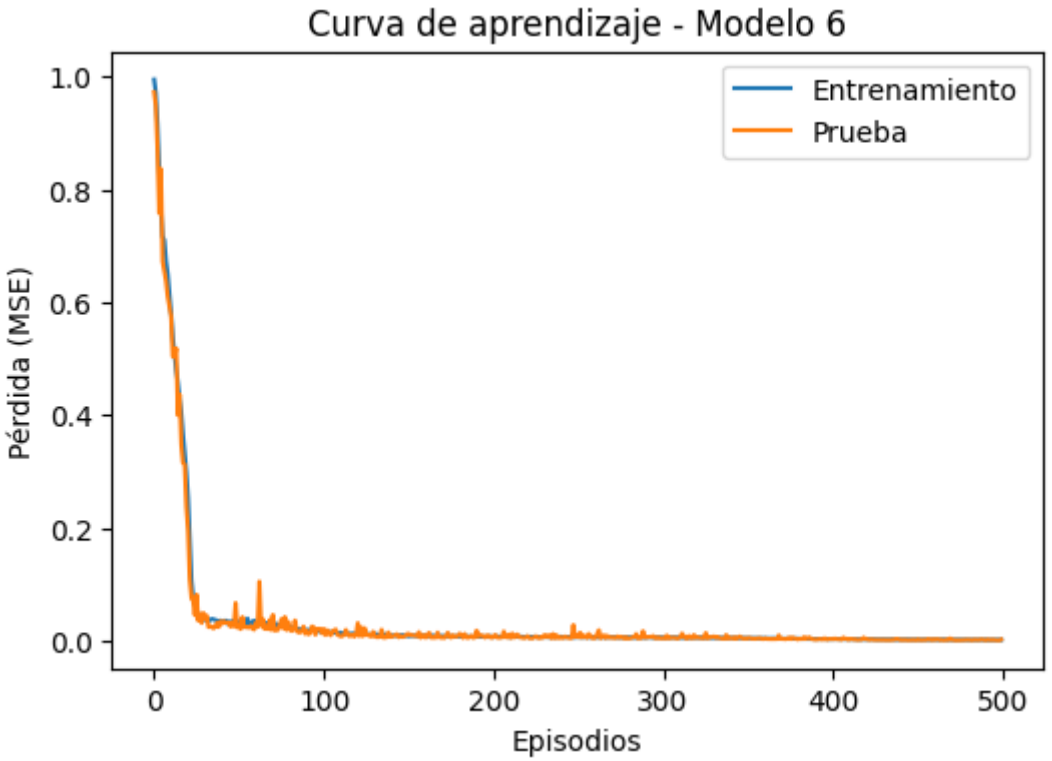
- **Modelo 4:**
    - **Análisis:**
        - Las curvas de entrenamiento y validación muestran una disminución constante y suave de la pérdida (MSE) a medida que aumentan las épocas.
        - No hay signos de sobreajuste, ya que las curvas convergen de manera estable.
        - Este modelo logró el MSE más bajo en el conjunto de prueba, lo que lo convierte en el mejor modelo.



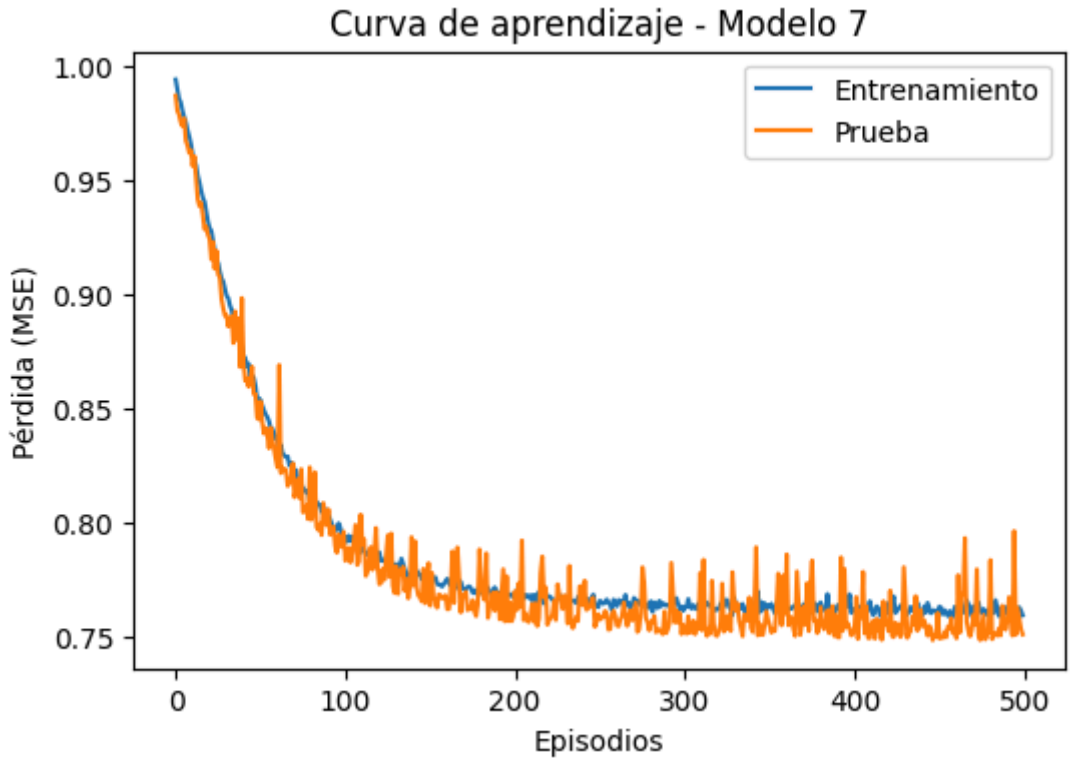
- **Modelo 5:**
    - **Análisis:**
        - Se evidencian curvas de entrenamiento y validación con comportamientos irregulares y pérdidas (MSE) relativamente altas.
        - Esto sugiere que la combinación de las funciones de activación ReLU y tanh en diferentes capas no fue efectiva para este problema.



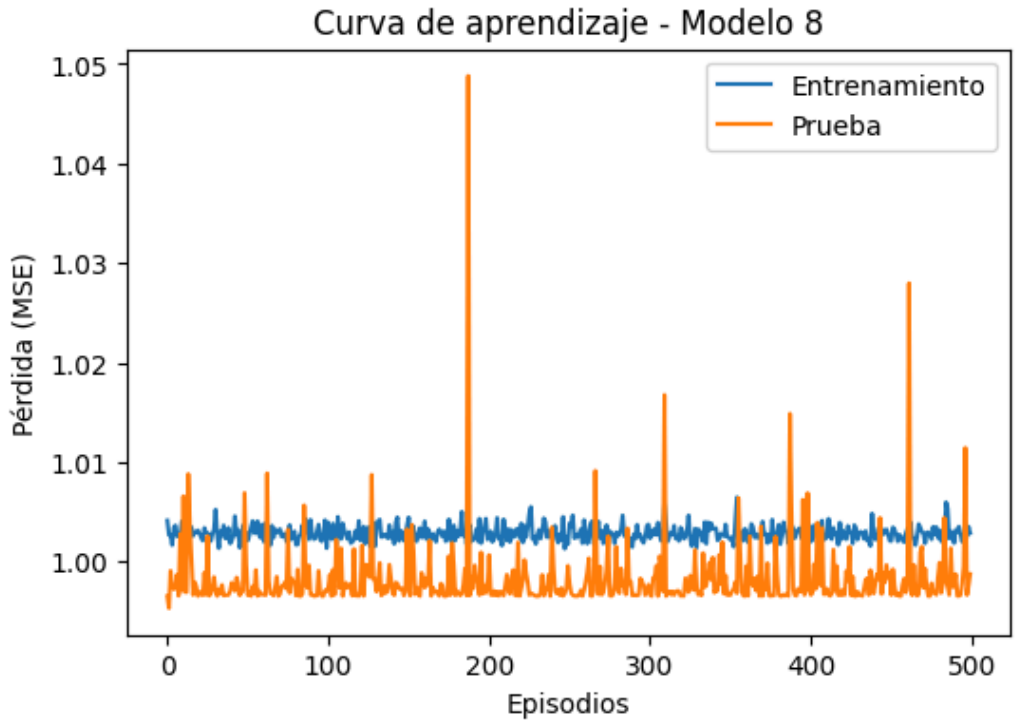
- **Modelo 6:**
    - **Análisis:**
        - Se puede observar que a medida que aumenta el número de epochs el modelo va mejorando significativamente hasta llegar a un punto donde no posee casi ningún tipo de ruido
        - Esto lo puede convertir en un contendiente al mejor modelo pero los datos del MSE son relativamente altos a comparación del modelo 4



- **Modelo 7:**
    - **Análisis (Modelo 7 y 8):**
        - Estos modelos exhiben curvas de entrenamiento y validación con pérdidas (MSE) muy altas y sin convergencia.
        - Esto indica un pobre desempeño del modelo y sugiere que la combinación de las funciones de activación tanh y ReLU en diferentes capas no fue adecuada.

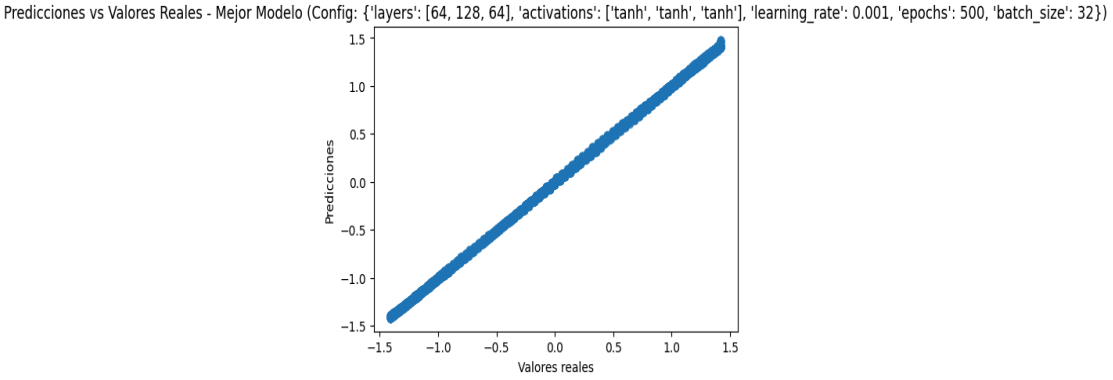


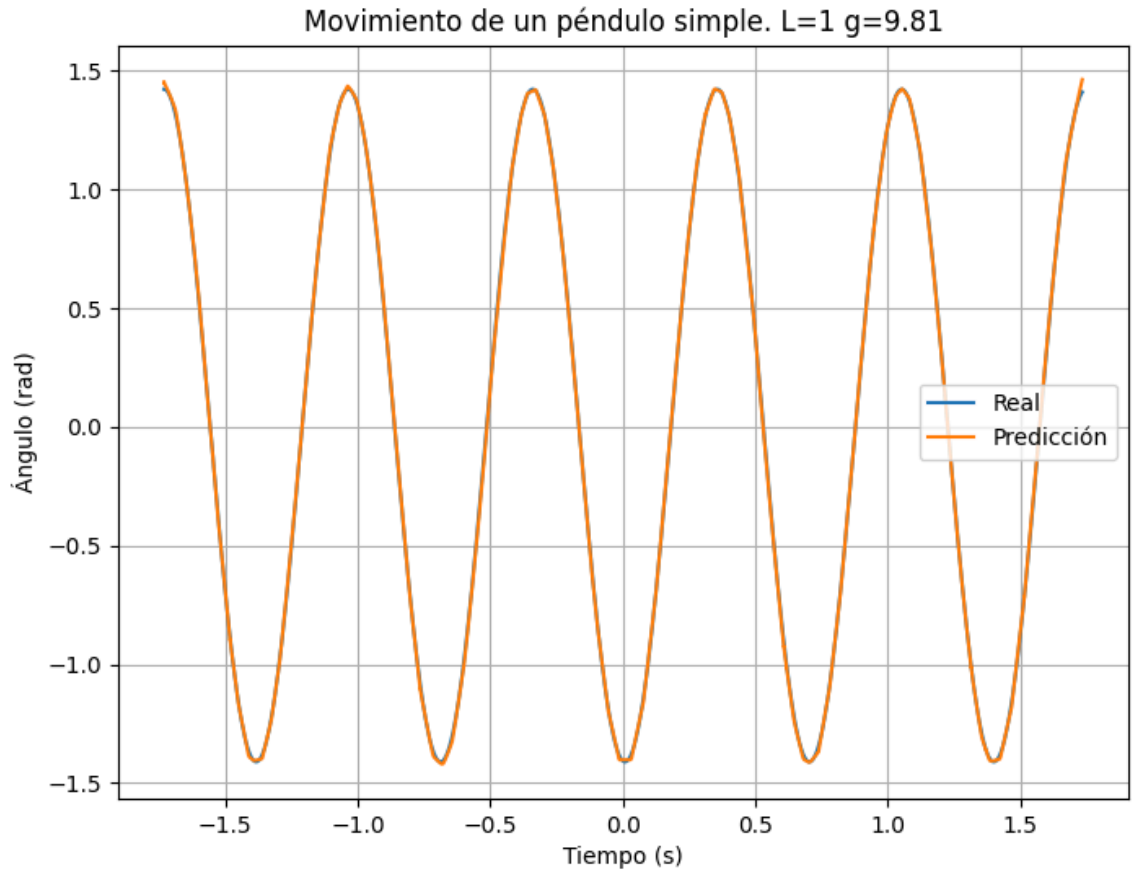
- **Modelo 8:**



Mejor modelo:

- **Modelo 4:**





8. Conclusiones:

- El Modelo 4, con 3 capas ocultas, función de activación tanh y una tasa de aprendizaje de 0.001, fue el mejor modelo según entrenamiento y tomando como medida el MSE, logrando el MSE más bajo en el conjunto de prueba.
- La elección adecuada de la arquitectura de la red neuronal, la función de activación y la tasa de aprendizaje es crucial para obtener un buen rendimiento en la aproximación de la solución de la ecuación de movimiento del péndulo simple.

- El preprocesamiento de datos mediante la estandarización mejoró la eficiencia y estabilidad del entrenamiento del modelo.
- El enfoque propuesto fue efectivo para aproximar la solución de la ecuación del péndulo simple, pero tiene limitaciones relacionadas con la calidad de los datos sintéticos y la elección de hiperparámetros adecuados.

9. Referencias

- Runge, C., & Kutta, W. (1900). A method for the numerical integration of differential equations by Runge-Kutta. Mathematische Annalen, 46(2), 167-178.
- OpenAI. (2024, Mayo 26). ChatGPT
- Agarap, A.F., 2018. Deep learning using rectified linear units (ReLU). arXiv preprint arXiv:1803.08375.