# 使用神经网络模拟动力学系统：一个演示项目指南 (PyTorch & LSTM版)

## 1. 引言：用神经网络模拟动力学

### 1.1 概述

本报告旨在引导用户完成一个演示项目，构建一个神经网络模型来学习并模拟一个简单动力学系统（受阻尼驱动的单摆）的行为。重点在于让用户熟悉从物理原理出发、通过数值模拟生成数据，到最终利用机器学习方法进行模型构建、训练和评估的完整工作流程。我们将聚焦于根据系统当前状态和外部输入，预测系统未来状态的应用场景。

### 1.2 为何选择神经网络模拟动力学？

传统上，动力学系统的建模依赖于基于物理定律（如牛顿定律或拉格朗日方程）推导出的数学方程[3]。这种方法虽然能够提供高保真度的模型，但对于高度非线性、存在复杂耦合或物理机制尚不完全清楚的系统（例如真实的液压挖掘机[2]），模型的推导过程可能极其复杂，且数值求解计算量巨大。

近年来，神经网络，特别是循环神经网络（RNN），提供了一种强大的数据驱动建模范式[9]。它们能够直接从观测到的系统行为数据中学习其内在的动力学规律，而无需显式推导复杂的物理方程。本演示项目将结合这两种方法：我们首先利用已知的物理方程生成仿真数据，然后用这些数据训练神经网络，使其“学会”这个物理系统的行为模式。

### 1.3 工作流程预览

我们将遵循以下步骤完成此演示项目：

1. **系统定义：** 选择并描述受阻尼驱动的单摆系统及其运动方程。
2. **数据生成：** 使用数值积分方法（如欧拉法或龙格-库塔法）模拟摆的运动，生成时间序列数据。
3. **数据准备：** 加载并处理仿真数据，构建适用于RNN训练的输入/输出序列。
4. **模型构建：** 选择并构建一个循环神经网络（LSTM）模型。
5. **模型训练：** 使用准备好的数据训练神经网络。
6. **模型评估：** 评估模型性能，并可视化预测结果。
7. **（可选）物理约束：** 简要介绍物理信息神经网络（PINN）的概念。
8. **总结与反思：** 回顾整个流程，并讨论该演示与模拟真实复杂系统（如挖掘机）之间的差距。

### 1.4 所需工具

本项目主要使用以下Python库：

- NumPy: 用于数值计算，特别是数组操作。

- Pandas: 用于数据处理和CSV文件操作。
- Scikit-learn: 用于数据归一化和模型评估。
- **PyTorch**: 用于构建和训练神经网络模型。
- Matplotlib: 用于数据和结果的可视化。

## 2. 步骤 1：定义动力学系统——受阻尼驱动的单摆

### 2.1 系统选择理由

我们选择单摆作为演示对象。这是一个经典的、广为人知的动力学系统，其行为（如振荡、阻尼）既有趣又可以通过相对简单的数学方程描述[14]。这使得我们可以方便地生成高质量的仿真数据，从而将重点放在理解机器学习的工作流程上，而非复杂的物理建模本身。相比之下，真实世界的系统，如液压挖掘机，涉及多体耦合、复杂的液压传动以及与土壤等环境的交互作用，其动力学建模要复杂得多[2]。

### 2.2 系统描述

- **物理设置：**考虑一个质量为 m 的质点，通过一根长度为 L 的无质量刚性杆连接到一个固定的、无摩擦的枢轴上。系统在竖直平面内运动，受到与角速度成正比的阻尼力，并受到一个外部施加的驱动力矩 τ(t)。
- **状态变量：**系统的状态由两个变量描述：
  - 角度 θ (rad): 摆杆与竖直向下方向的夹角。
  - 角速度 θ_dot (rad/s): 角度 θ 对时间的变化率。状态向量可以表示为 x(t) = [θ(t), θ_dot(t)]^T。
- **输入：**系统的输入是外部施加的驱动力矩 τ(t)。

### 2.3 模拟目标

我们的目标是训练一个神经网络模型，该模型能够根据系统在 t 时刻的状态 [θ(t), θ_dot(t)] 和该时刻的输入力矩 τ(t)，预测系统在下一个离散时间步 t+Δt 的状态 [θ(t+Δt), θ_dot(t+Δt)]。

### 2.4 运动方程

根据牛顿第二定律的转动形式（Στ = Iα，其中 I 是转动惯量，α 是角加速度）[21]，或者通过拉格朗日力学，可以推导出单摆的运动方程。考虑重力、阻尼力和外部力矩的作用：

- 重力产生的力矩: -m * g * L * sin(θ)（负号表示恢复力矩）
- 阻尼力矩: -c * θ_dot（假设阻尼力矩与角速度成正比，c 为阻尼系数）
- 外部力矩: τ(t)
- 转动惯量: I = m * L^2
- 角加速度: α = θ_ddot

将这些项代入 Στ = Iα，得到二阶常微分方程 (ODE):

m * L^2 * θ_ddot = -c * θ_dot - m * g * L * sin(θ) + τ(t)
为了方便数值积分和状态空间表示，我们将其改写为标准形式 θ_ddot = f(θ, θ_dot, τ)：
θ_ddot = -(c / (m * L^2)) * θ_dot - (g / L) * sin(θ) + τ(t) / (m * L^2)
这与文献中常见的受阻尼驱动摆的方程形式一致[2]。为了简化表示，我们定义：

- 阻尼系数 β = c / (m * L^2)（注意：这里的定义与某些文献可能不同，例如[20] 中 λ=γ/m，[22] 中 2β=b/m，[2] 中 q 代表阻尼。重要的是物理意义，即阻尼项与角速度成正比）
- 自然频率的平方 ω0^2 = g / L

则方程变为：
θ_ddot = -β * θ_dot - ω0^2 * sin(θ) + (1 / (m * L^2)) * τ(t)
为了进行数值积分，需要将这个二阶ODE转换为一个一阶ODE系统。令状态变量 x1 = θ 和 x2 = θ_dot，则状态向量 x = [x1, x2]^T。该一阶系统为：
dx1/dt = x2
dx2/dt = -β * x2 - ω0^2 * sin(x1) + (1 / (m * L^2)) * τ(t)
这定义了数值积分器所需的状态导数函数 dx/dt = F(t, x, τ)。

### 2.5 参数设定

为了进行仿真，我们需要为系统设定具体的物理参数。这些参数的选择会直接影响系统的动态行为[2]。神经网络模型将从由这些特定参数值生成的数据中学习动力学。这意味着，如果真实系统的参数（如质量、长度、阻尼）发生变化，或者存在不确定性（这在像挖掘机这样的复杂系统中很常见[5]），那么基于旧参数数据训练的模型在新情况下的预测性能可能会下降。这揭示了纯数据驱动模型的一个潜在局限性，相比之下，基于物理的模型通常能更明确地处理参数变化。在后续的"总结与反思"部分（第9节），我们将进一步探讨真实挖掘机系统中参数不确定性的挑战。

以下是本演示项目建议使用的参数值：

**表 2.1:** 单摆仿真参数

| 参数 (符号) | 值 | 单位 | 描述 |
|---|---|---|---|
| 质量 (m) | 1.0 | kg | 摆锤质量 |
| 长度 (L) | 1.0 | m | 摆杆长度 |
| 重力加速度 (g) | 9.81 | $m/s^2$ | 标准重力加速度 |
| 阻尼系数 (c) | 0.5 | N·m·s/rad | 线性阻尼系数 |

| | | | |
|---|---|---|---|
| 时间步长 (Δt) | 0.02 | s | 数值积分步长 |

根据这些值，可以计算出 β = c / (m * L^2) = 0.5 / (1.0 * 1.0^2) = 0.5 s^-1 和 ω0^2 = g / L = 9.81 / 1.0 = 9.81 s^-2。

## 3. 步骤 2：生成仿真数据

### 3.1 数值积分简介

由于单摆的运动方程包含非线性项 sin(θ)，并且可能受到任意变化的驱动力矩 τ(t) 的影响，通常无法得到解析解。因此，我们需要使用数值积分方法，在离散的时间步长 Δt 上逐步近似求解该ODE系统 [23]。

### 3.2 欧拉法 (Euler Method)

- **概念：** 欧拉法是最简单的数值积分方法。它基于函数在当前点的切线来近似下一个点的值 [23]。对于我们的一阶系统 dx/dt = F(t, x, τ)，欧拉法的递推公式为：x(t + Δt) ≈ x(t) + F(t, x(t), τ(t)) * Δt
- **实现：** 可以编写一个简单的Python函数 euler_step(x, t, tau, dt, pendulum_ode) 来执行单步积分。pendulum_ode 函数接收当前状态 x、时间 t 和力矩 tau，返回状态导数 [x2, dx2/dt]。

```python
import numpy as np

def pendulum_ode(t, x, tau, m=1.0, L=1.0, g=9.81, c=0.5):
    theta, theta_dot = x
    beta = c / (m * L**2)
    omega0_sq = g / L
    dtheta_dt = theta_dot
    dtheta_dot_dt = -beta * theta_dot - omega0_sq * np.sin(theta) + tau / (m * L**2)
    return np.array([dtheta_dt, dtheta_dot_dt])

def euler_step(x, t, tau, dt, ode_func, **kwargs):
    dxdt = ode_func(t, x, tau, **kwargs)
    x_next = x + dxdt * dt
    return x_next
```

- **局限性：** 欧拉法虽然简单，但其精度较低（一阶精度），且对于较大的时间步长 Δt 可能不稳定 [5]。为了获得足够的精度，通常需要非常小的 Δt。

### 3.3 四阶龙格-库塔法 (RK4 Method)

- **概念:** RK4是一种更常用且精度更高（四阶精度）的数值积分方法。它通过在每个时间步内计算四个不同位置的斜率（k1, k2, k3, k4），然后取这些斜率的加权平均值来更新状态，从而获得更好的近似效果 [26]。
- **实现:** 同样可以编写一个Python函数 rk4_step(x, t, tau, dt, ode_func, **kwargs) 来实现RK4算法。

```
Python
def rk4_step(x, t, tau, dt, ode_func, **kwargs):
    k1 = ode_func(t, x, tau, **kwargs)
    k2 = ode_func(t + 0.5*dt, x + 0.5*dt*k1, tau, **kwargs) # Assuming tau is constant within dt
    k3 = ode_func(t + 0.5*dt, x + 0.5*dt*k2, tau, **kwargs) # Or use interpolated tau if needed
    k4 = ode_func(t + dt, x + dt*k3, tau, **kwargs)
    x_next = x + (dt/6.0) * (k1 + 2*k2 + 2*k3 + k4)
    return x_next
```

- **使用SciPy库:** scipy.integrate.solve_ivp 是一个功能强大且推荐使用的ODE求解器 [28]。它内置了多种高精度的数值积分方法，包括 'RK45'（一种自适应步长的五阶龙格-库塔法）[30]。使用 solve_ivp 可以简化代码并提高求解的鲁棒性和效率。

```
Python
from scipy.integrate import solve_ivp

# Example usage with solve_ivp
# Define time span for one step
t_span = [t_current, t_current + dt]
# Solve for the next step, passing current state x and current torque tau
# Note: solve_ivp expects f(t, y, *args). We wrap our tau input.
pendulum_params = {'m': 1.0, 'L': 1.0, 'g': 9.81, 'c': 0.5} # Define params
tau_current = 0.0 # Example torque
x_current = np.array([0.1, 0.0]) # Example initial state

sol = solve_ivp(lambda t, x: pendulum_ode(t, x, tau_current, **pendulum_params),
                t_span, x_current, method='RK45', t_eval=[t_current + dt])
x_next = sol.y[:, -1]
```

虽然 solve_ivp 通常使用自适应步长，但可以通过 t_eval 参数指定需要输出解的时间点，从而获得固定时间步长 Δt 的结果，这对于后续构建RNN训练数据通常是必要的 [28]。除非是为了学习数值方法的实现细节，否则推荐使用 solve_ivp。

数值积分方法的选择直接关系到生成数据的质量和效率。欧拉法简单但精度低，可能需要

很小的步长。RK4精度高，允许使用相对较大的步长，但每步计算量更大。solve_ivp 提供了自适应步长和多种方法的选择，通常是平衡精度和效率的最佳选择。无论选择哪种方法，仿真数据的精度是神经网络模型能够达到的最高精度的理论上限。

表 **3.1:** 数值积分器比较

| 方法 | 精度阶数 | 稳定性 | 步长要求 | 实现复杂度 | 推荐用于 **Demo** |
|---|---|---|---|---|---|
| 欧拉法 | 1 | 条件稳定 | 小 | 低 | 仅用于教学 |
| RK4 (固定步长) | 4 | 较好 | 中等 | 中等 | 可选 |
| solve_ivp | 可变 (≥4) | 良好 | 自适应/指定 | 低 (使用库) | 推荐 |

### **3.4** 设计输入力矩序列 **(τ(t))**

为了让神经网络学习到系统在各种情况下的动态响应，我们需要使用多样化的输入力矩 τ(t) 来驱动仿真。如果只使用单一类型的输入（例如，始终为零力矩的自由衰减振荡），模型将无法泛化到其他输入情况。
建议生成以下几种类型的力矩序列：

- 零力矩**:** τ(t) = 0。用于观察系统的固有阻尼振荡。
- 阶跃力矩**:** τ(t) 在某个时刻从0突然变为一个常数值。用于观察系统对突变输入的响应。
- 正弦力矩**:** τ(t) = A * sin(Ω * t)。用于观察系统对周期性输入的响应，特别是频率响应特性。可以尝试不同的幅值 A 和频率 Ω。
- 随机力矩**:** 例如，生成分段常数或经过低通滤波的随机噪声序列。这有助于激发系统更广泛的动态行为。

可以使用NumPy来生成这些力矩序列的时间数组。例如，生成一个包含阶跃、正弦和随机段的组合序列。

### **3.5** 运行仿真

仿真过程通常在一个循环中进行：

1. 初始化**:** 设置初始状态 x0 = [θ0, θ_dot0]（例如，[0.1, 0] 表示从静止位置附近释放）和初始时间 t = 0。
2. 时间迭代**:** 在设定的总仿真时长内，按时间步长 Δt 进行迭代。
3. 计算输入**:** 在每个时间步 t，确定当前的输入力矩 τ(t)。
4. 数值积分**:** 调用选择的积分函数（如 rk4_step 或使用 solve_ivp）计算下一个时间步 t +

Δt 的状态 x(t + Δt)。

5. **存储数据：**将当前时间 t、状态 x(t) = [θ(t), θ_dot(t)] 和输入 τ(t) 存储起来。
6. **更新状态和时间：**将 x(t + Δt) 设为新的当前状态，t + Δt 设为新的当前时间，继续循环。

### 3.6 保存数据

将仿真过程中记录的所有时间戳 t、状态 θ(t)、θ_dot(t) 和输入 τ(t) 组织起来。使用Pandas库创建一个DataFrame，并将数据保存为CSV文件（例如，simulation_data.csv），以便后续处理。

Python

```
import pandas as pd

# Assume results are stored in lists: time_list, theta_list, theta_dot_list, tau_list
# Example placeholder lists:
time_list = np.arange(0, 10, 0.02)
theta_list = np.sin(time_list)
theta_dot_list = np.cos(time_list)
tau_list = np.zeros_like(time_list) # Example zero torque

data = {
    'time': time_list,
    'theta': theta_list,
    'theta_dot': theta_dot_list,
    'tau': tau_list
}
df = pd.DataFrame(data)
df.to_csv('simulation_data.csv', index=False)
```

确保生成足够长且包含多种输入力矩的数据，以覆盖系统可能经历的各种动态行为。

## 4. 步骤 3：准备数据以供RNN训练

### 4.1 加载数据

使用Pandas的 read_csv 函数加载之前保存的 simulation_data.csv 文件。

Python

```
import pandas as pd
df = pd.read_csv('simulation_data.csv')
```

**4.2 为RNN构建数据结构（滑动窗口）**

- **概念:** RNN（如LSTM或GRU）是为处理序列数据而设计的[32]。我们需要将原始的时间序列数据转换为一系列的“输入序列”和对应的“目标输出”。滑动窗口是一种常用的方法：使用过去 N 个时间步的状态和输入作为模型的输入序列，预测下一个时间步 t+1 的状态作为目标输出。
  - 输入序列 (在时间 t): [state(t-N+1), input(t-N+1),..., state(t), input(t)]
  - 目标输出 (在时间 t): state(t+1) 其中 state(t) = [θ(t), θ_dot(t)]，input(t) = [τ(t)]。
- **参数:** 需要定义窗口大小（或序列长度）N。选择 N 需要权衡：较大的 N 能为模型提供更长的历史信息，可能有助于学习长期依赖关系，但同时也会增加计算负担和模型复杂度。对于单摆系统，可以从一个适中的值开始，例如 N = 10 或 20。
- **实现:** 编写Python代码，使用NumPy对加载的DataFrame进行处理。遍历数据，按 N 的长度切片，构建输入数组 X 和输出数组 y。输入 X 的形状应为 (样本数, N, 输入特征数)，其中输入特征数为3（θ, θ_dot, τ）。输出 y 的形状应为 (样本数, 输出特征数)，其中输出特征数为2（θ(t+1), θ_dot(t+1)）。

Python

```
import numpy as np

def create_sequences(data, sequence_length):
    X, y =,
    # Assuming data is a NumPy array with columns [theta, theta_dot, tau]
    # Target will be [theta, theta_dot] at the next step
    for i in range(len(data) - sequence_length):
        # Input sequence: state and input from t-N+1 to t
        input_seq = data[i : i + sequence_length, :] # All features: theta, theta_dot, tau
        # Output target: state at t+1
        target = data[i + sequence_length, 0:2] # Only theta, theta_dot
        X.append(input_seq)
```

```
        y.append(target)
    return np.array(X), np.array(y)

# Example usage:
sequence_length = 10
data_values = df[['theta', 'theta_dot', 'tau']].values
X, y = create_sequences(data_values, sequence_length)

# X.shape will be (num_samples, sequence_length, 3)
# y.shape will be (num_samples, 2)
```

**4.3** 数据归一化

- **原因:** 神经网络的训练过程通常对输入特征的尺度很敏感。如果不同特征（如角度、角速度、力矩）的数值范围差异很大，梯度下降优化过程可能会不稳定，或者数值较大的特征会主导学习过程。将所有特征缩放到相似的范围（如 或 [-1, 1]）可以显著改善训练的稳定性和收敛速度。
- **方法:** 使用 sklearn.preprocessing.MinMaxScaler 是一个简单有效的方法。**关键点:** 必须仅使用训练数据来拟合（fit）scaler，以防止测试集的信息泄露到训练过程中。然后使用这个拟合好的scaler来转换（transform）训练集和测试集。需要分别对输入数据 X 和输出数据 y 进行归一化，并保存好scaler对象，以便在模型评估时将预测结果反归一化回原始尺度。
- **实现:**

```
Python
from sklearn.preprocessing import MinMaxScaler
import joblib # To save the scaler

# Assume X_train, y_train, X_test, y_test are already split (see next step)
# --- Placeholder Split ---
split_ratio = 0.8
split_index = int(len(X) * split_ratio)
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]
# --- End Placeholder Split ---

# Reshape X for scaler (needs 2D input: samples * features)
X_train_reshaped = X_train.reshape(-1, X_train.shape[-1])
X_test_reshaped = X_test.reshape(-1, X_test.shape[-1])
# y is already 2D (samples * features)

# Initialize scalers
```

```
input_scaler = MinMaxScaler(feature_range=(-1, 1))
output_scaler = MinMaxScaler(feature_range=(-1, 1))

# Fit on training data ONLY and transform
X_train_scaled_reshaped = input_scaler.fit_transform(X_train_reshaped)
y_train_scaled = output_scaler.fit_transform(y_train)

# Save the scalers
joblib.dump(input_scaler, 'input_scaler.pkl')
joblib.dump(output_scaler, 'output_scaler.pkl')

# Transform test data using the fitted scalers
X_test_scaled_reshaped = input_scaler.transform(X_test_reshaped)
y_test_scaled = output_scaler.transform(y_test)

# Reshape X back to 3D for RNN input
X_train_scaled = X_train_scaled_reshaped.reshape(X_train.shape)
X_test_scaled = X_test_scaled_reshaped.reshape(X_test.shape)

# Now use X_train_scaled, y_train_scaled, X_test_scaled, y_test_scaled for training/evaluation
```

**4.4** 训练集/测试集划分

- 重要性: 为了客观评估模型在未见过数据上的泛化能力，必须将数据划分为训练集和测试集。模型在训练集上进行训练，在测试集上进行最终评估。
- 时间序列数据的特殊性: 对于时间序列数据，绝对不能随机打乱后划分。这样做会破坏数据的时间连续性，导致模型在训练时“看到未来”的信息，从而得到过于乐观的评估结果。正确的做法是按时间顺序划分：使用数据的前一部分（例如，前80%）作为训练集，后一部分（例如，后20%）作为测试集。这更符合实际应用中用历史数据预测未来的场景。
- 实现: 根据滑动窗口处理后得到的 X 和 y 数组的样本数，计算划分点，然后使用数组切片完成划分。

Python

```
split_ratio = 0.8
split_index = int(len(X) * split_ratio)
```

```
X_train, X_test = X[:split_index], X[split_index:]
y_train, y_test = y[:split_index], y[split_index:]

# Now apply scaling as shown in the previous step
#... (scaling code using X_train, y_train, X_test, y_test)...
```

## 5. 步骤 4：构建循环神经网络模型

### 5.1 选择RNN架构：LSTM vs. GRU

- **RNN基础回顾：** 循环神经网络通过其内部的隐藏状态（一种记忆形式）来处理序列信息，使得过去的输入能够影响当前的输出 [9]。然而，简单的RNN在学习长期依赖关系时会遇到梯度消失或梯度爆炸的问题。
- **LSTM (长短期记忆网络)：** LSTM通过引入精密的门控机制（输入门、遗忘门、输出门）和一个独立的细胞状态（cell state）来克服这些问题，能够有效地捕捉和维持长期依赖关系 [32]。它的结构相对复杂，计算成本也更高 [32]。
- **GRU (门控循环单元)：** GRU是LSTM的一个简化变种，它只有两个门（重置门和更新门），并且没有独立的细胞状态，将隐藏状态同时用于短期和长期记忆 [32]。
- **推荐选择：** 根据你的要求，我们选择使用 **LSTM**。LSTM 在处理复杂序列和捕捉长期依赖方面表现出色，是许多序列建模任务的强大选择 [32]。虽然它比 GRU 更复杂，计算量更大，但其显式的记忆单元（细胞状态）和门控机制使其在理论上更擅长处理需要长时间记忆的任务 [32]。

表 **5.1: LSTM vs. GRU** 比较

| 特性 | **LSTM (长短期记忆网络)** | **GRU (门控循环单元)** |
|---|---|---|
| 门控机制 | 输入门 (Input Gate), 遗忘门 (Forget Gate), 输出门 (Output Gate) [32] | 重置门 (Reset Gate), 更新门 (Update Gate) [32] |
| 记忆单元 | 隐藏状态 + 细胞状态 (Cell State) [32] | 隐藏状态 (Hidden State) [32] |
| 参数量 | 较多 [12] | 较少 [12] |
| 计算成本 | 较高 [12] | 较低 [12] |

| | | |
|---|---|---|
| 性能 | 可能在极长序列上略优 [12] | 通常与LSTM相当 [36] |
| 适用性（本Demo） | 选定 (功能强大，擅长长期依赖) | 可选 (简洁高效) |

**5.2 模型实现 (以 PyTorch 为例)**

我们将使用 PyTorch 来构建 LSTM 模型。

- **框架选择:** PyTorch 以其灵活性和 Pythonic 的接口受到研究社区的欢迎。
- **模型类:** 在 PyTorch 中，通常通过定义一个继承自 torch.nn.Module 的类来构建模型。
- **LSTM层:** 使用 torch.nn.LSTM 层。需要指定 input_size（输入特征数，即3）、hidden_size（LSTM单元数/隐藏状态维度，例如64）和 num_layers（LSTM层数，通常为1或2）。batch_first=True 参数使输入和输出张量的形状为 (batch, seq_len, features)，这通常更直观。
- **全连接层 (Linear):** 在 LSTM 层之后，添加一个或多个全连接层 (torch.nn.Linear)。隐藏的 Linear 层通常使用 ReLU (torch.nn.ReLU) 等非线性激活函数。
- **输出层:** 最后一层是一个 Linear 层，其 out_features 等于输出特征的数量（即2，对应 θ(t+1) 和 θ_dot(t+1)）。这一层不使用激活函数（线性激活），因为我们预测的是连续值。
- **forward 方法:** 定义模型的 forward 方法，描述数据如何通过定义的层进行传播。LSTM 层返回 output（所有时间步的隐藏状态）和 (h_n, c_n)（最后一个时间步的隐藏状态和细胞状态）。我们通常只需要最后一个时间步的输出 output[:, -1, :] 来进行预测。

以下是使用 PyTorch 构建 LSTM 模型的代码示例：

Python

```
import torch
import torch.nn as nn

# Define model parameters
sequence_length = X_train_scaled.shape[1] # e.g., 10
num_input_features = X_train_scaled.shape[2] # e.g., 3
num_output_features = y_train_scaled.shape[1] # e.g., 2
hidden_size = 64 # Example number of LSTM units/hidden size
num_layers = 1 # Example number of LSTM layers
dense_units = 32 # Example number of Dense units
```

```
class PendulumLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size, dense_units):
        super(PendulumLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        # LSTM layer
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        # Optional intermediate dense layer
        self.fc1 = nn.Linear(hidden_size, dense_units)
        self.relu = nn.ReLU()
        # Output layer
        self.fc2 = nn.Linear(dense_units, output_size)

    def forward(self, x):
        # Initialize hidden and cell states
        # h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        # c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
        # Note: If h0, c0 are not provided, they default to zeros.

        # LSTM forward pass
        out, _ = self.lstm(x) # _ contains (h_n, c_n)

        # Decode the hidden state of the last time step
        out = out[:, -1, :] # Get output of the last time step

        # Pass through dense layers
        out = self.fc1(out)
        out = self.relu(out)
        out = self.fc2(out)
        return out

# Instantiate the model
model = PendulumLSTM(num_input_features, hidden_size, num_layers,
num_output_features, dense_units)

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
print(model) # Print model architecture
```

## 6. 步骤 5：训练神经网络

### 6.1 损失函数

- 选择：对于回归问题，均方误差 **(Mean Squared Error, MSE)** 是最常用的损失函数。
- 实现：在 PyTorch 中，使用 torch.nn.MSELoss。

### 6.2 优化器

- 选择：**Adam** 优化器是一种广泛使用且通常效果良好的自适应学习率优化算法。
- 实现：在 PyTorch 中，使用 torch.optim.Adam，并将模型的参数传递给它。

### 6.3 训练过程

- 概念：训练过程与 Keras 类似，但在 PyTorch 中需要手动编写训练循环。在每个周期（epoch）中，遍历训练数据（通常使用 DataLoader 来处理批次和打乱）：
  1. 将模型设置为训练模式 (model.train())。
  2. 获取一个批次的输入 inputs 和目标 targets。
  3. 将数据移动到正确的设备（CPU 或 GPU）。
  4. 清零梯度：optimizer.zero_grad()。这是必须的，因为 PyTorch 默认会累积梯度。
  5. 前向传播：outputs = model(inputs)。
  6. 计算损失：loss = criterion(outputs, targets)。
  7. 反向传播：loss.backward()。计算损失相对于模型参数的梯度。
  8. 优化器步骤：optimizer.step()。根据计算出的梯度更新模型参数。
- 验证：在每个 epoch 结束时，将模型设置为评估模式 (model.eval())，并在验证集上计算损失，不进行梯度计算（使用 with torch.no_grad():）。

Python

```
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader

# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train_scaled, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_scaled, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_scaled, dtype=torch.float32)
```

```
y_test_tensor = torch.tensor(y_test_scaled, dtype=torch.float32)

# Create datasets and dataloaders
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

batch_size = 32 # Example batch size
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True) # Shuffle training data
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False) # No need to shuffle test data

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001) # Example learning rate

# Training loop
num_epochs = 50 # Example number of epochs
train_losses =
val_losses =

for epoch in range(num_epochs):
    model.train() # Set model to training mode
    running_train_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, targets = data
        inputs, targets = inputs.to(device), targets.to(device) # Move data to device

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, targets)

        # Backward pass and optimize
        loss.backward()
        optimizer.step()

        running_train_loss += loss.item()
```

```
    epoch_train_loss = running_train_loss / len(train_loader)
    train_losses.append(epoch_train_loss)

    # Validation loop
    model.eval() # Set model to evaluation mode
    running_val_loss = 0.0
    with torch.no_grad(): # Disable gradient calculation for validation
        for i, data in enumerate(test_loader, 0):
            inputs, targets = data
            inputs, targets = inputs.to(device), targets.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            running_val_loss += loss.item()

    epoch_val_loss = running_val_loss / len(test_loader)
    val_losses.append(epoch_val_loss)

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {epoch_train_loss:.6f}, Val Loss:
{epoch_val_loss:.6f}')

print('Finished Training')
```

**6.4 监控训练过程**

- **可视化：**使用 Matplotlib 绘制 train_losses 和 val_losses 随 epochs 变化的曲线，方法与 Keras 类似。

Python

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.plot(range(1, num_epochs + 1), train_losses, label='Training Loss')
plt.plot(range(1, num_epochs + 1), val_losses, label='Validation Loss')
plt.title('Model Loss During Training')
plt.xlabel('Epoch')
```

```
plt.ylabel('Mean Squared Error')
plt.legend()
plt.grid(True)
plt.show()
```

- 曲线解读: 与 Keras 部分的解读方法相同，关注训练损失和验证损失的变化趋势，判断模型学习情况和是否过拟合。

## 7. 步骤 6：评估模型性能

### 7.1 定量评估

- 测试集损失: 使用测试集 test_loader 在评估模式下计算最终模型的损失。

Python

```
model.eval() # Ensure model is in evaluation mode
test_loss = 0.0
with torch.no_grad():
    for data in test_loader:
        inputs, targets = data
        inputs, targets = inputs.to(device), targets.to(device)
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        test_loss += loss.item()

average_test_loss = test_loss / len(test_loader)
print(f"Average Test MSE: {average_test_loss:.6f}")

# Remember this loss is on the SCALED data.
```

### 7.2 定性评估（多步预测）

- 挑战: 与 Keras 部分相同，需要评估模型在不使用真实值作为输入的迭代预测中的表现。
- 迭代预测过程 **(PyTorch):**
  1. 将模型设置为评估模式 (model.eval())。
  2. 从测试集中选取一个初始序列 current_sequence（需要是 PyTorch 张量并移动到设备）。

3. 在 torch.no_grad() 环境下进行预测：predicted_state_scaled = model(current_sequence)。
4. 将预测结果从 GPU 移回 CPU (.cpu()) 并转换为 NumPy 数组 (.numpy())，然后反归一化。
5. 获取下一个时间步的真实输入力矩 next_tau_original，归一化 next_tau_scaled。
6. 构建下一个时间步的输入特征 next_step_features_scaled（NumPy 数组）。
7. 更新输入序列（NumPy 数组操作），然后转换回 PyTorch 张量并移动到设备，准备下一次预测。
8. 重复步骤 3-7。

- **误差累积：**与 Keras 部分相同，误差会累积。
- **实现：**

Python

```
import joblib
import numpy as np
import torch

# Load scalers
input_scaler = joblib.load('input_scaler.pkl')
output_scaler = joblib.load('output_scaler.pkl')

# Select a starting sequence from the test set (use the scaled NumPy array)
start_index = 0
# Shape (1, seq_len, num_input_features)
initial_sequence_np = X_test_scaled[start_index:start_index+1]

# Convert initial sequence to tensor and move to device
current_sequence_tensor = torch.tensor(initial_sequence_np,
dtype=torch.float32).to(device)

# Define prediction horizon
prediction_horizon = len(X_test) - start_index

predicted_states_scaled_list =

model.eval() # Set model to evaluation mode
```

```
with torch.no_grad(): # Disable gradient calculations
    for i in range(prediction_horizon):
        # Predict the next state (scaled tensor)
        # Output shape is (1, num_output_features)
        next_state_scaled_tensor = model(current_sequence_tensor)

        # Move to CPU, convert to numpy, store
        next_state_scaled_np = next_state_scaled_tensor.cpu().numpy() # Get the 1D array
        predicted_states_scaled_list.append(next_state_scaled_np)

        # Get the known future input torque (original scale)
        next_tau_original = df['tau'].iloc[start_index + sequence_length + i]

        # Scale the next torque
        # Assuming tau is the last feature (index -1)
        # Create a dummy array matching scaler's expected input shape
        dummy_input_for_scaling = np.zeros((1, num_input_features))
        dummy_input_for_scaling[0, -1] = next_tau_original
        next_tau_scaled = input_scaler.transform(dummy_input_for_scaling)[0, -1]

        # Prepare the features for the next time step [theta_pred, theta_dot_pred, tau_next] (all scaled
numpy)
        # next_state_scaled_np has shape (num_output_features,) = (2,)
        next_step_features_scaled_np = np.array([next_state_scaled_np,
next_state_scaled_np[1], next_tau_scaled]) # Shape (3,)

        # Roll the sequence (using numpy for manipulation)
        # current_sequence_tensor shape is (1, seq_len, num_input_features)
        current_sequence_np = current_sequence_tensor.cpu().numpy() # Get (seq_len,
num_input_features)
        next_sequence_np = np.append(current_sequence_np[1:, :],
[next_step_features_scaled_np], axis=0) # Shape (seq_len, num_input_features)

        # Update current_sequence_tensor for the next iteration
        current_sequence_tensor = torch.tensor(np.expand_dims(next_sequence_np,
axis=0), dtype=torch.float32).to(device) # Back to (1, seq_len, num_input_features)

# Inverse transform the predicted states back to original scale
predicted_states_scaled_np_array = np.array(predicted_states_scaled_list) # Shape
(horizon, num_output_features)
```

```
predicted_states_original = output_scaler.inverse_transform(predicted_states_scaled_np_array)

# Get the true states (original scale) for comparison
# Use the original y_test (before scaling)
true_states_original = y_test[start_index : start_index + prediction_horizon]
```

**7.3 可视化比较**

- **绘图:** 使用 Matplotlib 将多步预测得到的 θ 和 θ_dot 轨迹（predicted_states_original）与测试集中对应的真实轨迹（true_states_original）绘制在同一张图上，方法与 Keras 部分相同。
- **解读:** 与 Keras 部分相同，通过视觉比较评估模型的长期预测能力。

Python

```
time_vector = df['time'].iloc[start_index + sequence_length : start_index + sequence_length + prediction_horizon].values

plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.plot(time_vector, true_states_original[:, 0], 'g-', label='True Theta')
plt.plot(time_vector, predicted_states_original[:, 0], 'r--', label='Predicted Theta')
plt.title('Multi-Step Prediction vs True Trajectory (LSTM - PyTorch)')
plt.ylabel('Theta (rad)')
plt.legend()
plt.grid(True)

plt.subplot(2, 1, 2)
plt.plot(time_vector, true_states_original[:, 1], 'g-', label='True Theta_dot')
plt.plot(time_vector, predicted_states_original[:, 1], 'r--', label='Predicted Theta_dot')
plt.ylabel('Theta_dot (rad/s)')
plt.xlabel('Time (s)')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

## 8. 步骤 7：（可选）引入物理约束——物理信息神经网络（PINN）简介

### 8.1 动机

纯粹基于数据的神经网络模型存在一些固有的局限性：它们通常需要大量的训练数据才能学习准确，可能产生违反基本物理定律的预测，并且在训练数据覆盖范围之外的泛化能力可能较差。物理信息神经网络（Physics-Informed Neural Networks, PINNs）旨在通过将已知的物理定律（通常表示为偏微分方程或常微分方程）直接整合到神经网络的训练过程中来克服这些限制 [38]。这种方法有望提高数据利用效率、模型的泛化能力和预测结果的物理一致性。

### 8.2 定义物理残差

回顾我们在步骤1中得到的单摆运动方程：
θ_ddot + β * θ_dot + ω0^2 * sin(θ) - τ(t) / (m * L^2) = 0
这个方程的左侧部分定义了物理残差 R(t, θ(t), θ_dot(t), θ_ddot(t), τ(t))。如果一个函数 θ(t) 精确地描述了单摆的运动，那么将它代入残差表达式，结果应该恒等于零。PINN的目标就是训练一个神经网络 θ_nn(t)，使其输出在满足数据约束的同时，也尽可能地使物理残差接近于零。

### 8.3 利用自动微分（AD）计算导数

PINN的一个关键技术是利用深度学习框架（如TensorFlow, PyTorch）内置的自动微分 (Automatic Differentiation, AD) 功能 40。AD允许我们精确且高效地计算神经网络输出对其输入的导数，而无需手动推导或进行数值差分。
对于单摆问题，如果神经网络 θ_nn(t) 直接预测角度 θ 作为时间 t 的函数，我们可以通过AD计算：

1. 角速度预测: θ_dot_nn = d(θ_nn) / dt
2. 角加速度预测: θ_ddot_nn = d^2(θ_nn) / dt^2 = d(θ_dot_nn) / dt

如果神经网络预测的是状态向量 [θ_nn(t), θ_dot_nn(t)]，则物理残差可以用一阶系统形式表示：

- 残差1: R1 = d(θ_nn)/dt - θ_dot_nn
- 残差2: R2 = d(θ_dot_nn)/dt - (-β * θ_dot_nn - ω0^2 * sin(θ_nn) + τ(t) / (m * L^2))
  在这种情况下，需要用AD计算 d(θ_nn)/dt 和 d(θ_dot_nn)/dt。

### 8.4 构建PINN损失函数

PINN的损失函数通常由两部分组成 40：
Loss_total = Loss_data + λ * Loss_physics

- **Loss_data (数据损失):** 这部分衡量模型预测与可用观测数据（如初始条件、边界条件

或系统在某些时间点的测量值）之间的差异。通常使用均方误差（MSE）。对应 [40] 中的 MSE_y。

- **Loss_physics (物理损失):** 这部分惩罚模型对物理定律的违反程度。它计算物理残差 R 在一系列配置点 **(collocation points)** 上的均方误差。这些配置点是在求解域（例如，时间区间 ``）内选取的点，不需要有对应的真实测量数据 [40]。Loss_physics = mean( R(t_colloc, θ_nn(t_colloc),...)^2 )
- **λ (权重因子):** 这是一个超参数，用于平衡数据损失和物理损失的重要性。λ 的选择对训练效果至关重要，可能需要仔细调整 [40]。

### 8.5 PINN训练

训练过程的目标是最小化总损失 Loss_total。通过最小化 Loss_physics，神经网络被"强制"学习满足控制方程（ODE）的解。

### 8.6 优势与挑战

- 优势:
  - 数据效率: 物理约束提供了额外的"监督"信息，可能减少对大量标记数据的依赖 [40]。
  - 泛化性: 遵循物理规律的模型可能在训练数据未覆盖的区域表现更好。
  - 物理一致性: 预测结果更可能符合物理直觉 [41]。
- 挑战:
  - 训练难度: PINN的训练可能比标准监督学习更困难，容易出现训练停滞、梯度病态等问题 [48]。
  - 超参数敏感: 对网络结构、优化器选择、配置点采样策略以及损失权重 λ 非常敏感 [48]。
  - 约束实现: 将物理约束作为"软约束"加入损失函数是常用方法，但可能无法严格满足约束。实现"硬约束"（即强制网络输出满足某些代数关系）通常更复杂 [38]。

尽管存在挑战，PINN为结合物理知识和机器学习提供了一个有前景的方向。对于本演示，理解将物理方程残差纳入损失函数的基本思想即可。实现完整的PINN需要更深入地使用 PyTorch 的 torch.autograd.grad 来计算必要的导数并构建自定义的训练循环。许多研究致力于改进PINN的训练策略和稳定性 [45]。

## 9. 步骤 8：总结与反思——从演示到真实挖掘机

### 9.1 流程回顾

本报告详细介绍了使用神经网络模拟简单动力学系统（受阻尼驱动的单摆）的完整流程。我们从定义系统及其物理方程开始，通过数值积分生成了仿真数据，接着对数据进行了预处理以适用于RNN训练，然后构建并训练了一个 **LSTM** 模型 (使用 **PyTorch**) 来预测系统的未来状态。我们评估了模型的单步和多步预测性能，并简要介绍了如何通过引入物理信

息神经网络（PINN）的概念来增强模型。

### 9.2 从演示到现实：挖掘机动力学模拟的挑战

虽然单摆演示项目帮助我们掌握了基本的工作流程，但将其应用于模拟真实的液压挖掘机则面临着巨大的复杂性鸿沟。以下是几个关键方面的对比：

- 数据来源与质量：
  - 单摆Demo：我们使用了基于已知精确物理方程生成的、无噪声的仿真数据（步骤2）。
  - 真实挖掘机：数据来源于安装在机器上的各种传感器，如惯性测量单元（IMU）、倾角传感器、电位计或编码器（用于测量关节角度），压力传感器（用于液压系统），GPS/GNSS（用于定位），以及可能的力或应变传感器 [2]。这些传感器数据不可避免地包含噪声、漂移、偏差，需要仔细校准和滤波。多传感器数据融合通常是必要的，以获得对系统状态的可靠估计 [2]。数据采集本身就是一个复杂的过程 [14]，并且挖掘机工作时的强烈振动会对传感器精度产生严重影响 [1]。
- 系统复杂性与建模：
  - 单摆Demo：简单的单刚体系统，运动学关系简单，易于推导运动方程（步骤1）。
  - 真实挖掘机：这是一个复杂的多体系统，包含底盘、回转平台、动臂、斗杆、铲斗等多个部件，通过多个关节连接 [4]。其动力学建模通常需要运用高等动力学方法，如牛顿-欧拉法或拉格朗日法，推导过程繁琐 [23]。部件的柔性（例如动臂的弹性变形）有时也需要考虑，形成刚柔耦合模型 [6]。系统的动力学特性会随挖掘机姿态（各关节角度）的变化而显著改变 [5]。专业的动力学仿真软件，如ADAMS或Simscape Multibody，常被用于挖掘机建模与仿真 [14]。
- 驱动系统：
  - 单摆Demo：仅受一个简单的外部力矩 τ(t) 驱动。
  - 真实挖掘机：主要由复杂的液压系统驱动 [2]。液压系统本身具有高度非线性特性，包括油液的流动特性、阀门（如比例阀）的开关特性、死区、饱和、压力动态响应以及油液的可压缩性等 [2]。建模时需要建立液压缸的力/压力与流量、阀门开度之间的关系，并将液压缸产生的力映射到关节力矩上 [5]。此外，液压泵的功率限制以及多个执行器同时动作时的流量分配问题也增加了复杂性 [8]。
- 外部交互：
  - 单摆Demo：没有与环境的复杂交互（或只有简化的驱动力矩）。
  - 真实挖掘机：挖掘作业的核心在于铲斗与土壤、岩石等挖掘对象的复杂交互作用 [23]。这种交互产生的挖掘阻力是高度变化且难以精确预测的，它强烈依赖于土壤的性质（如粘聚力、内摩擦角、密度等）、挖掘轨迹、铲斗的几何形状和姿态等因素 [63]。准确建模这种交互力是挖掘机动力学模拟和控制中的一个主要挑战，有时土壤参数是未知或随地点变化的 [63]。
- 模型需求与学习：
  - 单摆Demo：一个相对简单的 LSTM 模型就能取得不错的效果。

    - 真实挖掘机: 由于上述复杂性，可能需要更深、更复杂的神经网络架构（例如，包含注意力机制的模型 [63]），或者采用混合建模方法，将物理知识（如PINN [63]）与数据驱动方法相结合。学习的目标也可能更具体，例如专注于预测挖掘力 [63]、评估操作员技能 [67] 或识别特定的作业活动 [69]。考虑到不同挖掘机或不同工况下的差异，迁移学习（Transfer Learning）也可能是一个有价值的研究方向 [72]。

这些对比突显了从简单仿真到真实物理系统应用（即"仿真到现实"的鸿沟）所面临的挑战。仅仅在理想仿真环境中训练的模型（即使是使用ADAMS或Simscape等专业软件 [14] 生成的数据）直接部署到真实硬件上时，性能往往会大幅下降。这是因为模型没有学习到如何处理真实世界的噪声、不确定性、未建模的动力学以及复杂的环境交互。因此，开发能够在真实挖掘机上可靠工作的模型，通常需要结合真实的传感器数据进行训练，并采用能够处理不确定性和噪声的鲁棒建模与控制技术，物理知识的融入（如PINN）也可能在此过程中发挥重要作用。

# 10. 结论

## 10.1 学习总结

本报告通过一个受阻尼驱动单摆的实例，系统性地演示了使用循环神经网络（特别是 **LSTM**）来模拟动力学系统的完整流程，并使用了 **PyTorch** 框架进行实现。用户通过此项目可以实践从系统定义、基于物理方程的数值模拟数据生成、针对时间序列的数据预处理、RNN模型构建与训练，到最终的模型性能评估等关键步骤。同时，报告也简要介绍了物理信息神经网络（PINN）作为一种融合物理知识与机器学习的方法。掌握这一流程为使用数据驱动方法理解和预测更复杂的动态系统行为奠定了基础。

## 10.2 未来方向

基于本项目的基础，用户可以进一步探索以下方向：

- 模型比较: 尝试使用 GRU 代替 LSTM，比较两者的训练过程和预测性能。
- 架构探索: 实验不同的网络结构，如增加 LSTM 层数、改变隐藏单元数量、添加 Dropout 层等，观察其对性能的影响。
- **PINN**实现: 尝试更完整地实现 PINN 方法，包括使用自动微分计算物理残差，并将其纳入损失函数进行训练。
- 更复杂的系统: 将此工作流程应用于稍微复杂一些的系统，例如双摆、二连杆机械臂，或者简单的质量-弹簧-阻尼系统。
- 控制应用: 探索将训练好的动力学模型用于模型预测控制（MPC）或其他控制策略的设计 [40]，或者直接使用强化学习来学习控制策略，并将动力学模型作为环境模拟器或奖励函数的一部分 [67]。

## 10.3 最终思考

虽然简单的演示项目为理解神经网络在动力学建模中的应用提供了宝贵的实践经验，但我

们必须认识到，从模拟走向现实，特别是对于像液压挖掘机这样高度复杂的机电液一体化系统，仍然面临着巨大的挑战。解决这些挑战需要综合运用传感器技术、高级动力学建模、鲁棒控制理论以及更先进的机器学习方法，并需要大量高质量的真实世界数据。本报告所展示的基础流程是迈向这一目标的重要一步。

引用的著作


1. Research Status and Prospect of the Key Technologies for Environment Perception of Intelligent Excavators - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2076-3417/14/23/10919
2. Research Status and Development Trend of Intelligent Excavators, 访问时间为 四月 9, 2025, https://qikan.cmes.org/jxgcxb/EN/10.3901/JME.2020.13.165
3. robot.sia.cn, 访问时间为 四月 9, 2025, https://robot.sia.cn/cn/article/pdf/preview/2396.pdf
4. Modeling and Control of Excavator Dynamics during Digging ... - Scite, 访问时间为 四月 9, 2025, https://scite.ai/reports/modeling-and-control-of-excavator-5NEN2D
5. (PDF) Development of dynamic-mathematical model of hydraulic ..., 访问时间为 四月 9, 2025, https://www.researchgate.net/publication/320261581_Development_of_dynamic-mathematical_model_of_hydraulic_excavator
6. 工程机械臂系统结构动力学特性分析, 访问时间为 四月 9, 2025, https://www.cinn.cn/p/W020220412641521007146.pdf
7. 硕士学位论文, 访问时间为 四月 9, 2025, https://engineering.purdue.edu/~byao/Thesis/%E7%A1%95%E5%A3%AB%E8%AE%BA%E6%96%87-%E9%A5%B6%E9%B9%8F_ZJU16.pdf
8. CONSTRUCTING HYDRAULIC ROBOT MODELS USING MEMORY- BASED LEARNING - CiteSeerX, 访问时间为 四月 9, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4b98eba85be9085d5eaa547501177aa581d283ff
9. Gated Recurrent Units Viewed Through the Lens of ... - Frontiers, 访问时间为 四月 9, 2025, https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2021.678158/full
10. Recurrent neural network - Wikipedia, 访问时间为 四月 9, 2025, https://en.wikipedia.org/wiki/Recurrent_neural_network
11. What is Recurrent Neural Networks (RNN)? - Analytics Vidhya, 访问时间为 四月 9, 2025, https://www.analyticsvidhya.com/blog/2022/03/a-brief-overview-of-recurrent-neural-networks-rnn/
12. RNN vs. LSTM vs. GRU: A Comprehensive Guide to Sequential Data Modeling - Medium, 访问时间为 四月 9, 2025, https://medium.com/@hassaanidrees7/rnn-vs-lstm-vs-gru-a-comprehensive-guide-to-sequential-data-modeling-03aab16647bb
13. Artificial Neural Networks are a New Kind of Dynamical Systems - UW Applied

Mathematics, 访问时间为 四月 9, 2025, https://amath.washington.edu/news/2023/09/05/artificial-neural-networks-are-new-kind-dynamical-systems
14. Road load data acquisition | ARAI, 访问时间为 四月 9, 2025, https://www.araiindia.com/services/department-and-laboratories/vehicle-dynamics
15. Structural Testing of Excavators and Dump Trucks - HBM, 访问时间为 四月 9, 2025, https://www.hbm.com/tw/9341/structural-testing-of-excavators-and-dump-trucks/
16. Multibody dynamic simulation with Simscape: methods and examples - Webthesis, 访问时间为 四月 9, 2025, https://webthesis.biblio.polito.it/17488/1/tesi.pdf
17. Dynamic simulation of the CAD model in SimMechanics with multiple uses - ResearchGate, 访问时间为 四月 9, 2025, https://www.researchgate.net/publication/325882538_Dynamic_simulation_of_the_CAD_model_in_SimMechanics_with_multiple_uses
18. Dynamic modelling of hydraulic excavator motion using Kane's equations - ResearchGate, 访问时间为 四月 9, 2025, https://www.researchgate.net/publication/261920076_Dynamic_modelling_of_hydraulic_excavator_motion_using_Kane's_equations
19. 一种新型正铲液压挖掘机工作机构的研究, 访问时间为 四月 9, 2025, https://qikan.cmes.org/jxgcxb/CN/article/downloadArticleFile.do?attachType=PDF&id=53946
20. Dynamic Modeling and Analysis of Loader Working Mechanism Considering Cooperative Motion with the Vehicle Body - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2075-1702/11/1/9
21. Adams | Hexagon, 访问时间为 四月 9, 2025, https://hexagon.com/products/product-groups/computer-aided-engineering-software/adams
22. Determination of Excavator Tool Position using Absolute Sensors - ResearchGate, 访问时间为 四月 9, 2025, https://www.researchgate.net/publication/351759310_Determination_of_Excavator_Tool_Position_using_Absolute_Sensors
23. Mathematical model of the excavator for the experimental analysis - ResearchGate, 访问时间为 四月 9, 2025, https://www.researchgate.net/figure/Mathematical-model-of-the-excavator-for-the-experimental-analysis_fig3_327061631
24. mathematical modeling of dynamic processes of bucket wheel excavators - ASIM GI, 访问时间为 四月 9, 2025, https://www.asim-gi.org/fileadmin/user_upload_argesim/ARGESIM_Publications_OA/MATHMOD_Publications_OA/MATHMOD_2006_AR30/Sessions/MECH/114_P_Nenad_Zrnic.pdf
25. www.iri.upc.edu, 访问时间为 四月 9, 2025, http://www.iri.upc.edu/files/scidoc/494-Modeling-and-control-of-excavator-dyna

mics-during-digging-operation.pdf
26. Analysis of Dynamic Wear Characteristics of Joint Contact Friction Pair of Excavators Working Device - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2075-4442/12/4/113
27. Online payload estimation for hydraulically actuated manipulators - Sci-Hub, 访问时间为 四月 9, 2025, https://sci-hub.se/downloads/2020-02-02/de/renner2020.pdf
28. Load weight estimation on an excavator in static and dynamic motions - Linköping Electronic Conference Proceedings, 访问时间为 四月 9, 2025, https://ecp.ep.liu.se/index.php/sicfp/article/download/30/29/29
29. ASSISTED DRIVING MIDI-EXCAVATOR FOR AUGMENTED PERFORMANCES AND IMPROVED SAFETY - River Publishers, 访问时间为 四月 9, 2025, https://www.riverpublishers.com/downloadchapter.php?file=RP_9788770042222C75.pdf
30. Analysis of the Position Recognition of the Bucket Tip According to the Motion Measurement Method of Excavator Boom, Stick and Bucket - PMC, 访问时间为 四月 9, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC7284364/
31. Development of Integrative Methodologies for Effective Excavation Progress Monitoring, 访问时间为 四月 9, 2025, https://www.mdpi.com/1424-8220/21/2/364
32. LSTM and GRU Neural Networks as Models of Dynamical ... - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/1424-8220/21/16/5625
33. Deep Learning-based Robot Control using Recurrent Neural Networks (LSTM - Avestia, 访问时间为 四月 9, 2025, https://avestia.com/CDSR2021_Proceedings/files/paper/CDSR_113.pdf
34. VECTOR: Velocity-Enhanced GRU Neural Network for Real-Time 3D UAV Trajectory Prediction - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2504-446X/9/1/8
35. Improved GRU prediction of paper pulp press variables using different pre-processing methods - Estudo Geral, 访问时间为 四月 9, 2025, https://estudogeral.uc.pt/bitstream/10316/114654/1/Improved-GRU-prediction-of-paper-pulp-press-variables-using-different-preprocessing-methodsProduction-and-Manufacturing-Research.pdf
36. Full article: Improved GRU prediction of paper pulp press variables using different pre-processing methods - Taylor & Francis Online, 访问时间为 四月 9, 2025, https://www.tandfonline.com/doi/full/10.1080/21693277.2022.2155263
37. Enhancing the Precision of a Hydraulic Robotic Arm - kth .diva, 访问时间为 四月 9, 2025, https://kth.diva-portal.org/smash/get/diva2:1950567/FULLTEXT01.pdf
38. A Hard-Constraint Wide-Body Physics-Informed Neural Network Model for Solving Multiple Cases in Forward Problems for Partial Differential Equations - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2076-3417/14/1/189
39. Physics-Informed Neural Networks for Solving Contact Problems in Three Dimensions, 访问时间为 四月 9, 2025, https://arxiv.org/html/2412.09022v1
40. arxiv.org, 访问时间为 四月 9, 2025, https://arxiv.org/pdf/2104.02556
41. Understanding Physics-Informed Neural Networks: Techniques, Applications, Trends, and Challenges - MDPI, 访问时间为 四月 9, 2025,

https://www.mdpi.com/2673-2688/5/3/74
42. Scientific Machine Learning through Physics-Informed Neural Networks - arXiv, 访问时间为 四月 9, 2025, http://arxiv.org/pdf/2201.05624
43. Physics-informed neural networks with hard constraints for inverse design - arXiv, 访问时间为 四月 9, 2025, http://arxiv.org/pdf/2102.04626
44. Physics Informed Neural Networks (PINNs) [Physics Informed Machine Learning] - YouTube, 访问时间为 四月 9, 2025, https://www.youtube.com/watch?v=-zrY7P2dVC4
45. [2407.20669] A Tutorial on the Use of Physics-Informed Neural Networks to Compute the Spectrum of Quantum Systems - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/abs/2407.20669
46. A tutorial on the use of physics-informed neural networks to compute the spectrum of quantum systems - arXiv, 访问时间为 四月 9, 2025, http://arxiv.org/pdf/2407.20669
47. Constrained Hamiltonian Systems and Physics-Informed Neural Networks: Hamilton-Dirac Neural Networks - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/html/2401.15485v2
48. [2308.08468] An Expert's Guide to Training Physics-informed Neural Networks - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/abs/2308.08468
49. Learning and discovering multiple solutions using physics-informed neural networks with random initialization and deep ensemble - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/html/2503.06320v1
50. Exploring Physics-Informed Neural Networks: From Fundamentals to Applications in Complex Systems - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/html/2410.00422v1
51. A hands-on introduction to Physics-Informed Neural Networks for solving partial differential equations with benchmark tests taken from astrophysics and plasma physics - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/html/2403.00599v1
52. Physics-Informed Neural Networks and Extensions - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/html/2408.16806v1
53. Dynamic Modeling of a Hydraulic Excavator Stick by Introducing Multi-Case Synthesized Load Spectrum for Bench Fatigue Test - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2075-1702/10/9/741
54. Measure Position/Displacement With LVDT Sensors - Dewesoft, 访问时间为 四月 9, 2025, https://dewesoft.com/blog/measure-position-displacement-with-lvdt-sensors
55. Excavator Design with Simscape - File Exchange - MATLAB Central, 访问时间为 四月 9, 2025, https://www.mathworks.com/matlabcentral/fileexchange/119268-excavator-design-with-simscape
56. Modeling and Simulation for the Excavator in MATLAB Simscape - PID Control - YouTube, 访问时间为 四月 9, 2025, https://www.youtube.com/watch?v=leeysWcXyVk
57. A hydraulic actuator for joint robots with higher torque to weight ratio - ResearchGate, 访问时间为 四月 9, 2025,

https://www.researchgate.net/publication/365224715_A_hydraulic_actuator_for_joint_robots_with_higher_torque_to_weight_ratio?_tp=eyJjb250ZXh0Ijp7InBhZ2UiOiJzY2llbnRpZmljQ29udHJpYnV0aW9ucyIsInByZXZpb3VzUGFnZSI6bnVsbH19
58. Online Gain Switching Algorithm for Joint Position Control of a Hydraulic Humanoid Robot - CMU School of Computer Science, 访问时间为 四月 9, 2025, http://www.cs.cmu.edu/~cga/papers/jyk-hum07.pdf
59. A hydraulic actuator for joint robots with higher torque to weight ratio | Robotica, 访问时间为 四月 9, 2025, https://www.cambridge.org/core/journals/robotica/article/hydraulic-actuator-for-joint-robots-with-higher-torque-to-weight-ratio/22130838F40C27C739CBAFB3FD83CC41
60. Practical Kinematic and Dynamic Calibration Methods for Force-Controlled Humanoid Robots | Disney Research, 访问时间为 四月 9, 2025, https://la.disneyresearch.com/wp-content/uploads/Practical-Kinematic-and-Dynamic-Calibration-Methods-for-Force-Controlled-Humanoid-Robots-Paper.pdf
61. (PDF) Dynamic Simulation of a Hydraulic Excavator to Determine the Joint Reaction Forces of Boom, Stick, Bucket, and Driving Forces of Hydraulic Cylinders - ResearchGate, 访问时间为 四月 9, 2025, https://www.researchgate.net/publication/351803988_Dynamic_Simulation_of_a_Hydraulic_Excavator_to_Determine_the_Joint_Reaction_Forces_of_Boom_Stick_Bucket_and_Driving_Forces_of_Hydraulic_Cylinders
62. Dynamic Simulation of a Hydraulic Excavator to D... — Library of Science - Biblioteka Nauki, 访问时间为 四月 9, 2025, https://bibliotekanauki.pl/articles/318155
63. arxiv.org, 访问时间为 四月 9, 2025, https://arxiv.org/pdf/2309.02575
64. STUDY ON MODELING AND CONTROL OF EXCAVATOR - The International Association for Automation and Robotics in Construction, 访问时间为 四月 9, 2025, https://www.iaarc.org/publications/fulltext/S28-3.pdf
65. Establishment of input and output mathematical model based on vibratory excavating system of excavator | TSI Journals, 访问时间为 四月 9, 2025, https://www.tsijournals.com/articles/establishment-of-input-and-output-mathematical-model-based-on-vibratory-excavating-system-of-excavator.pdf
66. [2301.02731] Attention-LSTM for Multivariate Traffic State Prediction on Rural Roads - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/abs/2301.02731
67. Automatic Evaluation of Excavator Operators using Learned Reward Functions - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/abs/2211.07941
68. (PDF) Automatic Evaluation of Excavator Operators using Learned ..., 访问时间为 四月 9, 2025, https://www.researchgate.net/publication/365414905_Automatic_Evaluation_of_Excavator_Operators_using_Learned_Reward_Functions
69. arXiv:2112.04572v1 [cs.LG] 8 Dec 2021, 访问时间为 四月 9, 2025, https://arxiv.org/pdf/2112.04572
70. Vision-Based Activity Classification of Excavators by Bidirectional LSTM - MDPI, 访问时间为 四月 9, 2025, https://www.mdpi.com/2076-3417/13/1/272
71. Vision-Based Body Pose Estimation of Excavator Using a Transformer-Based

Deep-Learning Model | Journal of Computing in Civil Engineering | Vol 39, No 2 - ASCE Library, 访问时间为 四月 9, 2025, https://ascelibrary.org/doi/abs/10.1061/JCCEE5.CPENG-6079
72. Transfer Learning in Robotics: An Upcoming Breakthrough? A Review of Promises and Challenges - arXiv, 访问时间为 四月 9, 2025, https://arxiv.org/html/2311.18044v2